



Programmer avec Lexis+

Le guide complet du débutant : toutes les fonctionnalités du langage, expliquées pas à pas, avec des exemples complets et commentés, exécutés pour de vrai.

16 chapitres

87 exemples exécutés

740 instructions et fonctions

15 domaines

Démarrer · Les bases · Décider & répéter · Procédures · Tables & objets · Texte & calcul · Erreurs · Bases de données · Fichiers & formats · Système · Réseau · Multitâche · Interface graphique · Pages web · Intelligence artificielle

wdlexis+ · wdlexisrt

Développez naturellement vos applications métier

© 2026 Jean-Marc QUÉRÉ

sonaliwan.fr

Bienvenue dans Lexis+

Ce guide s'adresse à celles et ceux qui n'ont **jamais programmé** comme à ceux qui découvrent Lexis+ après un autre langage. Il présente **toutes** les fonctionnalités du langage et de ses paquets, dans l'ordre où on en a besoin : d'abord les bases (afficher, calculer, décider, répéter), puis les données (tables, bases, fichiers), enfin les interfaces, le réseau, le parallélisme et l'intelligence artificielle.

Chaque notion est accompagnée d'un **exemple complet et commenté**, tiré d'une situation réelle de gestion : facture, stock, clients, planning… Les programmes ont été **réellement exécutés** par l'interpréteur Lexis+ pendant la fabrication de ce guide, et ce que vous voyez sous « Ce qu'affiche le programme » est leur sortie exacte. Seuls les exemples qui demandent un écran (fenêtres), un serveur distant ou un grand modèle de langage sont présentés avec un aperçu ou une sortie type ; leur syntaxe a, elle aussi, été vérifiée.

Comment lire ce guide

- Les **onglets de couleur** sur le bord droit des pages indiquent le domaine ; leur logo reprend l'illustration du domaine.
- Les encadrés vous guident : **À retenir** (l'essentiel), **Pour bien comprendre** (une notion expliquée simplement), **Astuce**, **Attention** (un piège fréquent), **Syntaxe** (la forme exacte d'une instruction) et **À vous de jouer** (un exercice).
- Dans le code, les **mots-clés**, les chaînes, les nombres, les **paquets**, les **@références** et les *commentaires* ont chacun leur couleur, comme dans l'atelier.
- Tapez les exemples vous-même : c'est en écrivant du code qu'on apprend à programmer.

Les annexes

Annexe A : la référence complète, domaine par domaine. **Annexe B** : les aides-mémoires, qui résument sur une page toute la syntaxe et toutes les instructions d'un domaine, avec leurs enchaînements. **Annexe C** : un glossaire. **Annexe D** : le poster, une carte heuristique de toutes les instructions. Les aides-mémoires et le poster existent aussi en documents séparés, à imprimer et à garder sous la main.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Les quinze domaines de Lexis+

Chaque domaine a sa couleur, son illustration et son onglet. Les nombres indiquent combien d'instructions, de fonctions et de méthodes il regroupe.



Démarrer

Écrire, exécuter, déboguer et lancer un programme.

21 entrées



Les bases

Instructions, commentaires, valeurs, opérateurs et variables.

32 entrées



Décider & répéter

Choisir un chemin, répéter un traitement.

16 entrées



Procédures

Nommer un traitement pour le réutiliser partout.

13 entrées



Tables & objets

Listes, fiches, objets et métatables.

33 entrées



Texte & calcul

La boîte à outils toujours disponible : texte, nombres, dates, sécurité.

55 entrées



Erreurs

Signaler, intercepter et traiter les erreurs.

12 entrées



Bases de données

DuckDB intégré (SQL et zones dBase), SQLite, Firebird.

60 entrées



Fichiers & formats

Lire et écrire fichiers, dossiers, CSV, JSON et binaire.

54 entrées



Système

Dates, environnement, commandes, arguments.

22 entrées



Réseau

HTTP, TCP, UDP, FTP, SSH et SFTP.

41 entrées



Multitâche

Tâches coopératives et fils d'exécution.

22 entrées



Interface graphique

Fenêtres wdgui : contrôles, événements, dialogues.

154 entrées



Pages web

Pages HTML / CSS / JavaScript reliées à Lexis+.

31 entrées



Intelligence artificielle

Modèles de langage, GGUF, dessin, audio, tenseurs, entraînement.

174 entrées

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Sommaire

1. Vos premiers pas, en français 7

1.1 Programmer, qu'est-ce que c'est ? 7
1.2 Installer et lancer l'atelier 8
1.3 Votre premier programme 9
1.4 Dialoguer avec l'utilisateur 9
1.5 Organiser un projet 10
1.6 L'atelier wdlexis+ en détail 11
1.7 Le runtime wdlexisrt 13

2. Les bases du langage 14

2.1 Lignes, instructions et commentaires 14
2.2 Afficher des résultats : ? et ?? 15
2.3 Les valeurs et leurs types 16
2.4 Convertir d'un type à l'autre 17
2.5 Les opérateurs 18
2.6 Les variables 20

3. Décider et répéter 23

3.1 Choisir avec si ... sinon ... finsi 23
3.2 Choisir parmi de nombreux cas : faire cas 25
3.3 Choisir une valeur : sisinon 26
3.4 Répéter un nombre connu de fois : pour 27
3.5 Répéter tant qu'une condition est vraie : tantque 28
3.6 Parcourir une liste ou une fiche : pour chaque 29
3.7 Sortir d'une boucle ou passer au tour suivant 31

4. Procédures et fonctions 33

4.1 Pourquoi des procédures ? 33
4.2 Définir et appeler une procédure 34
4.3 Renvoyer un résultat : retourner 35
4.4 La portée des variables 37
4.5 Les autres façons d'appeler 38
4.6 La récursivité 39
4.7 Organiser ses procédures 40

5. Tables, listes et objets 42

5.1 Une table, deux usages 42
5.2 Les listes 43
5.3 Les fiches 44
5.4 Des données réelles : listes de fiches 46
5.5 Partage, copie et JSON 46
5.6 Des objets avec les métatables 47

6. Texte, nombres, dates et sécurité 51

6.1 Mettre du texte en forme 51
6.2 Découper et extraire 52
6.3 Chercher, remplacer, tester 52
6.4 Caractères, HTML et URL 53
6.5 Calculer avec les nombres 53
6.6 Dates et heures 54
6.7 Tester une valeur 55
6.8 Sécurité : empreintes, mots de passe, identifiants 55

7. Prévoir et traiter les erreurs 57

7.1 Quand quelque chose tourne mal 57
7.2 Intercepter une erreur : essayer ... capturer 58
7.3 Signaler une erreur : l'instruction erreur 59
7.4 Relancer et journaliser 61
7.5 Une stratégie pour vos programmes 62

8. Les bases de données 64

8.1 Une base de données, c'est quoi ? 64
8.2 Créer des tables et ajouter des données 65
8.3 Interroger la base 66
8.4 Modifier, supprimer et regrouper en transaction 67
8.5 Les zones : travailler enregistrement par enregistrement 68
8.6 Se déplacer et chercher dans une zone 69
8.7 Modifier les données dans une zone 70
8.8 Plusieurs zones en même temps 71
8.9 SQLite 73
8.10 Firebird 74

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

9. Fichiers et formats de données 77

9.1 Les paquets d'entrées / sorties 77
9.2 Lire et écrire un fichier d'un seul coup 78
9.3 Lire au fil de l'eau avec es 79
9.4 Dossiers et gestion des fichiers 81
9.5 Le format CSV 82
9.6 Le format JSON 84
9.7 Les données binaires 85

10. Dialoguer avec le système 88

10.1 Le temps en secondes 88
10.2 Mettre une date en forme 89
10.3 Mesurer une durée et faire une pause 90
10.4 Environnement et commandes du système 90
10.5 Arguments et code de retour : un vrai outil en ligne de commande 92

11. Communiquer sur le réseau 94

11.1 Quelques notions de réseau 94
11.2 Le web avec le paquet http 95
11.3 TCP : clients et serveurs maison 97
11.4 UDP : des messages sans connexion 98
11.5 FTP 99
11.6 SSH et SFTP 100

12. Faire plusieurs choses à la fois 102

12.1 Tâches ou fils ? 102
12.2 Les tâches coopératives 103
12.3 Les canaux : faire communiquer des tâches 105
12.4 Les fils d'exécution 108

13. Des fenêtres et des boutons 112

13.1 La programmation par événements 112
13.2 La fenêtre 114
13.3 Les contrôles de base 114
13.4 Les événements 117
13.5 Organiser la fenêtre : conteneurs et mise en page 118
13.6 Listes, tableaux et arbres 120
13.7 La grille de saisie 121
13.8 Menus, graphiques et contrôles de planification 124
13.9 Les boîtes de dialogue 126

14. Des pages web dans vos programmes 128

14.1 Le principe 128
14.2 Ouvrir une fenêtre web 129
14.3 Lexis+ appelle JavaScript 131
14.4 JavaScript appelle Lexis+ 132
14.5 Événements, console et fermeture 134
14.6 Une application complète 135

15. L'intelligence artificielle 138

15.1 Panorama 139
15.2 Charger un modèle et dialoguer 139
15.3 Régler les réponses 141
15.4 Recevoir la réponse au fil de l'eau 142
15.5 Des réponses structurées : JSON 143
15.6 Pièces jointes et formats de réponse 144
15.7 La mémoire d'une conversation 145
15.8 Sous le capot : tokens, logits et vecteurs 146
15.9 Recherche documentaire et agents 147
15.10 Les fichiers GGUF et la quantification 148
15.11 Dessiner : le paquet dessin 150
15.12 Faire parler : le paquet audio 151
15.13 Apprendre : le paquet tenseur 152
15.14 Créer et spécialiser un modèle : le paquet entrainement 154

16. Aller plus loin 157

16.1 Écrire un paquet en Nim 157
16.2 Du poste au serveur wdGestion V 159
16.3 Organiser une application réelle 159
16.4 Bonnes pratiques 160
16.5 Pour aller plus loin 160

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Annexe A · Référence complète 161

demarrer 162
bases 163
controle 165
procedures 166
tables 167
calcul 169
erreurs 171
donnees 172
fichiers 175
systeme 178
reseau 179
multitache 181
interface 182
web 189
ia 191

Annexe B · Aides-mémoires 199

lexis 200
demarrer 201
bases 202
controle 203
procedures 204
tables 205
calcul 206
erreurs 207
donnees 208
fichiers 209
systeme 210
reseau 211
multitache 212
interface 213
web 215
ia 216

Annexe C · Glossaire 218

Annexe D · Le poster 222

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

CHAPITRE 1 · DÉMARRER

Vos premiers pas, en français

Qu'est-ce qu'un programme, comment l'écrire, le lancer et le corriger avec l'atelier wdlexis+. À la fin de ce chapitre, vous aurez écrit, exécuté et suivi pas à pas votre premier programme Lexis+.



1.1 Programmer, qu'est-ce que c'est ?
1.2 Installer et lancer l'atelier
1.3 Votre premier programme
1.4 Dialoguer avec l'utilisateur
1.5 Organiser un projet
1.6 L'atelier wdlexis+ en détail
1.7 Le runtime wdlexisrt

Dans ce chapitre, vous apprendrez à…

- comprendre ce qu'est un programme et un interpréteur
- écrire, exécuter et enregistrer un programme
- organiser un projet en plusieurs fichiers
- lancer l'atelier wdlexis+ et le runtime wdlexisrt
- dialoguer avec l'utilisateur (afficher, saisir)
- suivre un programme pas à pas avec le débogueur

1.1 Programmer, qu'est-ce que c'est ?

Un **programme** est une suite d'**instructions** que l'ordinateur exécute l'une après l'autre, du haut vers le bas, comme une recette de cuisine. Chaque instruction est un ordre précis : « affiche ce texte », « additionne ces deux nombres », « recommence trois fois », « enregistre cette fiche dans la base ».

Ces instructions s'écrivent dans un **langage de programmation**. Lexis+ a une particularité qui le rend idéal pour débuter : ses mots-clés sont **en français**. On écrit `si`, `sinon`, `pour`, `tantque`, `retourner`… et non `if`, `else` ou `while`. Le programme se lit presque comme une phrase.

L'ordinateur ne comprend pas directement le texte d'un programme. Un logiciel, l'**interpréteur**, le lit et l'exécute instruction par instruction. Pour Lexis+, l'interpréteur est intégré à l'**atelier wdlexis+** (l'éditeur dans lequel vous écrivez vos programmes) et au **runtime wdlexisrt** (qui exécute un programme sans l'éditeur).

© 2026 Jean-Marc QUÉRÉ,sonaliwan.fr · SIRET13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Le vocabulaire de base

- **Code source** : le texte du programme, tel que vous l'écrivez. En Lexis+, il est rangé dans des fichiers `.prg`.
- **Exécuter** (ou *lancer*) : demander à l'interpréteur de réaliser les instructions.
- **Console** : la zone de texte où le programme affiche ses résultats et où l'on tape les réponses.
- **Bogue** (*bug*) : une erreur dans le programme. **Déboguer**, c'est la trouver et la corriger.

Lexis+ est le langage du serveur **wdGestion V** et des applications de la gamme (LexiPilot, CaSchd…). Il sait manipuler des bases de données, des fichiers, le réseau, des fenêtres, des pages web et même des modèles d'intelligence artificielle. Mais tout commence par une ligne.

1.2 Installer et lancer l'atelier

L'atelier et le runtime sont deux programmes autonomes. Il suffit de les copier dans un dossier :

Programme	Rôle	Comment le lancer
`wdlexis+`	l'**atelier** : éditeur, console, débogueur	double-clic, ou `./wdlexis+` dans un terminal
`wdlexisrt`	le **runtime** : exécute un programme sans éditeur	`wdlexisrt principal.prg`

Certaines fonctions s'appuient sur des **bibliothèques** chargées seulement si le programme s'en sert. On les place dans un dossier `winOS/`, `macOS/` ou `linOS/` (selon votre système) à côté de l'exécutable :

Bibliothèque	Nécessaire pour
DuckDB (`duckdb.dll`, `libduckdb.dylib`, `libduckdb.so`)	la base de données intégrée (chapitre 8)
SDL3 et SDL3_ttf	les fenêtres de l'interface graphique (chapitre 13)
webview (`webview.dll`, `.dylib`, `.so`)	les pages web dans une fenêtre (chapitre 14)
client Firebird, SQLite, OpenSSL	les bases Firebird et SQLite, HTTPS

Astuce

Sans ces bibliothèques, l'atelier et le runtime démarrent quand même : seules les instructions qui en ont besoin signalent leur absence par un message clair. Vous pouvez donc commencer tout de suite.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

1.3 Votre premier programme

Lancez l'atelier, puis choisissez *Fichier › Nouveau projet…* et acceptez le dossier proposé. L'atelier crée un dossier de projet contenant un fichier `principal.prg`. Remplacez son contenu par ces lignes, puis appuyez sur **Ctrl+R** (ou cliquez sur ▶ **Exécuter**).

EXEMPLE Bonjour le monde — principal.prg

```
&& Mon tout premier programme Lexis+
? "Bonjour le monde !"
? "Je sais compter :", 1 + 2 + 3
? "Aujourd'hui, nous sommes le", date()
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Bonjour le monde !
Je sais compter : 6
Aujourd'hui, nous sommes le 2026-10-16
```

Décortiquons ces quatre lignes :

1. La ligne qui commence par `&&` est un **commentaire** : l'interpréteur l'ignore. Les commentaires servent à expliquer le programme aux humains qui le lisent (y compris vous, dans six mois).
2. Le point d'interrogation `?` est l'instruction **afficher**. Elle écrit dans la console la valeur qui la suit, puis passe à la ligne.
3. On peut afficher plusieurs valeurs en les séparant par des virgules : elles sont écrites à la suite, séparées par un espace. `1 + 2 + 3` est un **calcul** : c'est son résultat, 6, qui s'affiche.
4. `date()` est une **fonction intégrée** : elle donne la date du jour. Les parenthèses indiquent qu'on *appelle* la fonction.

À retenir

Un programme s'exécute **ligne par ligne, de haut en bas**. Chaque ligne contient une instruction. Les textes s'écrivent entre guillemets `"..."` ; les nombres et les calculs s'écrivent tels quels.

1.4 Dialoguer avec l'utilisateur

Un programme devient vraiment utile quand il réagit à ce que l'on tape. La fonction `saisir()` affiche une question dans la console, attend que l'utilisateur tape une réponse et valide avec **Entrée**, puis renvoie le texte tapé. On range cette réponse dans une **variable**, une boîte qui porte un nom.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Un programme qui pose des questions — principal.prg

```
&& Demande le prénom et l'année de naissance, puis calcule l'âge
locale prenom = saisir("Votre prénom ? ")
locale annee = nombre(saisir("Votre année de naissance ? "))

si annee = nul                          && la réponse n'est pas un nombre
  ? "Ce n'est pas une année valide."
sinon
  locale age = 2026 - annee
  ? "Bonjour", prenom + " !"
  ? "En 2026, vous fêtez vos", age, "ans."
finsi
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Votre prénom ? Ada
Votre année de naissance ? 1815
Bonjour Ada !
En 2026, vous fêtez vos 211 ans.
```

- `locale prenom = ...` crée une variable nommée `prenom` et y range la valeur.
- `saisir()` renvoie toujours du **texte**. `nombre(...)` le convertit en nombre ; si c'est impossible (l'utilisateur a tapé « bientôt »), il renvoie la valeur spéciale `nul`.
- `si ... sinon ... finsi` choisit entre deux chemins (chapitre 3).
- `"Bonjour", prenom + " !"` : la virgule ajoute un espace, le `+` colle deux textes sans espace.

Dans l'atelier, la question apparaît dans la console et vous tapez la réponse dans la **ligne de saisie** en bas de la console. Dans le runtime, c'est le terminal qui sert de console.

À vous de jouer

Modifiez le programme pour demander aussi la ville de l'utilisateur, puis affichez « Bonjour Ada, de Londres ! ».

1.5 Organiser un projet

Un **projet** Lexis+ est un **dossier**. Il contient :

- un fichier principal, `principal.prg` : ses lignes **en tête** (avant la première procédure) forment le **programme principal**, celui qui s'exécute au lancement ;
- autant d'autres fichiers `.prg` que vous voulez, contenant des **procédures** et **fonctions** (des morceaux de programme réutilisables, chapitre 4). Toutes sont chargées au lancement et s'appellent par leur nom depuis n'importe quel fichier ;
- éventuellement vos données : base `donnees.duckdb`, fichiers CSV, pages HTML…

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

FICHIER Un second fichier du projet — outils.prg

```
&& outils.prg : des fonctions utilisables partout dans le projet
fonction TTC(ht)
retourner arrondi(ht * 1.2, 2)

fonction EUROS(montant)
retourner chaine(montant, 1, 2) + " €"
```

EXEMPLE Le programme principal utilise outils.prg — principal.prg

```
&& principal.prg
locale prix = 49.90
? "Prix HT  :", EUROS(prix)
? "Prix TTC :", EUROS(TTC(prix))   && TTC est définie dans outils.prg
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Prix HT  : 49.90 €
Prix TTC : 59.88 €
```

Si le fichier principal ne contient **que** des procédures, l'interpréteur exécute la procédure nommée `PRINCIPAL`. Et l'instruction `finprocedure` (ou `finfonction`) marque explicitement la fin d'une procédure : les lignes qui suivent appartiennent de nouveau au programme principal.

EXEMPLE Procédure PRINCIPAL et finprocedure — principal.prg

```
procedure PRINCIPAL
  ? "Démarrage par la procédure PRINCIPAL"
  SALUER("Ada")
finprocedure

procedure SALUER(qui)
  ? "Bonjour", qui
finprocedure
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Démarrage par la procédure PRINCIPAL
Bonjour Ada
```

À retenir

Un projet = un dossier. Le programme principal est écrit **en tête** de `principal.prg` ; les procédures peuvent être dans n'importe quel fichier `.prg` du dossier et de ses sous-dossiers.

1.6 L'atelier wdlexis+ en détail

L'atelier réunit tout ce qu'il faut pour écrire et mettre au point un programme :

- l'**éditeur**, avec coloration des mots-clés, complétion (**Ctrl+Espace**) et **analyse en continu** : les fautes de syntaxe, les appels de procédures inconnues et les variables globales implicites sont

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

soulignés pendant que vous tapez (onglet **Problèmes**) ;
- la **console**, où le programme affiche ses résultats, avec sa ligne de saisie ;
- le panneau **Structure** (procédures et variables globales du fichier) et le panneau **Référence** (toutes les fonctions et tous les paquets : un double-clic insère l'appel) ;
- un **diagramme de structure** dans la marge, qui dessine les procédures, les `si`, les boucles et les blocs `essayer` ;
- le **débogueur**, pour exécuter le programme pas à pas.

Touche	Action
Ctrl+R	exécuter (sans débogueur, plus rapide)
F5	déboguer (s'arrête aux points d'arrêt) / continuer
F9 ou clic dans la marge	poser ou retirer un point d'arrêt
F11	pas à pas : exécute une instruction, en entrant dans les procédures
F10	pas principal : exécute une instruction sans entrer dans les procédures
Maj+F11	termine la procédure en cours
Maj+F5	arrête le programme
Ctrl+Espace	complétion
Ctrl+B	commente / décommente les lignes (`&&`)
F12 ou Ctrl+clic	aller à la définition d'une procédure

Suivre un programme pas à pas

Le débogueur est votre meilleur allié pour comprendre ce que fait un programme. Essayez avec celui-ci :

EXEMPLE Un programme à suivre au débogueur — `principal.prg`

```
1  locale total = 0
2  pour i = 1 a 4
3    total = total + CARRE(i)      && posez un point d'arrêt ici (F9)
4  suivant
5  ? "Somme des carrés de 1 à 4 :", total
6
7  fonction CARRE(n)
8  retourner n * n
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Somme des carrés de 1 à 4 : 30
```

1. Cliquez dans la marge à gauche de la ligne 3 : un point rouge apparaît.
2. Appuyez sur **F5** : le programme s'arrête **avant** d'exécuter cette ligne, qui est surlignée.
3. L'onglet **Variables** montre `total` et `i`. La **pile d'appels** montre où vous êtes.
4. **F11** entre dans `CARRE` : vous voyez `n` valoir 1. **Maj+F11** en ressort.
5. **F5** continue jusqu'au prochain arrêt : `i` vaut maintenant 2, `total` vaut 1.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Astuce

Quand un programme ne fait pas ce que vous attendez, ne devinez pas : posez un point d'arrêt et regardez les variables. C'est presque toujours plus rapide.

Les documents ouverts sont exécutés **tels qu'ils sont à l'écran**, même s'ils ne sont pas enregistrés. En cas d'erreur, l'atelier affiche le message et la pile d'appels, et place le curseur sur la ligne fautive.

1.7 Le runtime wdlexisrt

Le runtime exécute un programme sans l'atelier : c'est lui que vous utiliserez pour lancer une application terminée, une tâche planifiée ou un script.

TERMINAL — Les commandes du runtime — terminal

```
wdlexisrt programme.prg [arguments...]   # exécute (avec les autres .prg du dossier)
wdlexisrt --verifier programme.prg       # vérifie la syntaxe sans exécuter
wdlexisrt -e "? 1 + 1"                   # exécute une ligne de code
wdlexisrt --paquets                      # liste les paquets disponibles
```

Les mots placés après le nom du programme sont ses **arguments**. Le programme les retrouve dans la liste `systeme.args` (`systeme.args[0]` est le chemin du programme lui-même). `systeme.quitter(code)` termine le programme en renvoyant un **code de retour** au système (0 signifie « tout s'est bien passé »).

EXEMPLE — Lire les arguments de la ligne de commande — principal.prg

```
&& Lancement : wdlexisrt principal.prg facture.csv 2026
? "Programme :", fichier.nom(systeme.args[0])
? "Nombre d'arguments :", #systeme.args
pour i = 1 a #systeme.args
  ? "  argument", i, "=", systeme.args[i]
suivant
si #systeme.args < 2
  ? "Usage : principal.prg fichier annee"
  systeme.quitter(1)                  && code 1 : erreur d'utilisation
finsi
? "Traitement de", systeme.args[1], "pour l'année", systeme.args[2]
```

▶ CE QU'AFFICHE LE PROGRAMME

```
Programme : principal.prg
Nombre d'arguments : 2
  argument 1 = facture.csv
  argument 2 = 2026
Traitement de facture.csv pour l'année 2026
```

À retenir

- `?` affiche, `saisir()` demande, `&&` commente.
- **Ctrl+R** exécute, **F5** débogue, **F9** pose un point d'arrêt.
- Un projet est un dossier ; `principal.prg` en est le point d'entrée ; `wdlexisrt` l'exécute sans l'atelier.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

CHAPITRE 2 · LES BASES

Les bases du langage

Les briques avec lesquelles tout programme est construit — instructions, commentaires, valeurs, types, opérateurs et variables. Prenez le temps de bien les comprendre : tout le reste repose sur elles.



2.1 Lignes, instructions et commentaires

2.2 Afficher des résultats : ? et ??

2.3 Les valeurs et leurs types

2.4 Convertir d'un type à l'autre

2.5 Les opérateurs

2.6 Les variables

Dans ce chapitre, vous apprendrez à…

- ✔ écrire des instructions et des commentaires lisibles
- ✔ connaître les types de valeurs (nombre, chaîne, logique, nul…)
- ✔ calculer et comparer avec les opérateurs
- ✔ afficher des résultats avec ? et ??
- ✔ convertir un texte en nombre et inversement
- ✔ ranger des valeurs dans des variables locales et globales

2.1 Lignes, instructions et commentaires

Un programme Lexis+ s'écrit **une instruction par ligne**. La fin de la ligne marque la fin de l'instruction : pas besoin de point-virgule comme dans d'autres langages.

Quand une instruction est trop longue, on la **continue** sur la ligne suivante en terminant la ligne par un point-virgule `;` (héritage de dBase). À l'intérieur de parenthèses `( )`, de crochets `[ ]` ou d'accolades `{ }`, les retours à la ligne sont libres.

Trois façons d'écrire un **commentaire**, c'est-à-dire un texte ignoré par l'interpréteur :

Écriture	Portée
`&& texte`	de `&&` jusqu'à la fin de la ligne
`// texte`	de `//` jusqu'à la fin de la ligne
`* texte`	toute la ligne, si `*` est le premier caractère de l'instruction

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Instructions, continuation et commentaires — principal.prg

```
* Calcul d'une facture — exemple de mise en formeCalcul d'une facture — exemple de mise en
forme
&& Les commentaires expliquent le POURQUOI, le code dit le COMMENT.
locale quantite = 12
locale prix_unitaire = 4.5        // prix en euros
locale total = quantite * ;
               prix_unitaire       && suite de la ligne précédente
locale lignes = {"stylo",         && dans des accolades,
                 "cahier",        && les retours à la ligne sont libres
                 "gomme"}
? "Total :", total, "€ pour", #lignes, "articles"
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Total : 54 € pour 3 articles
```

La casse et les accents

Les **mots-clés** du langage (`si`, `pour`, `procedure`…) s'écrivent en minuscules. Les **noms** que vous choisissez (variables, procédures, champs) ne tiennent **pas compte de la casse** : `total`, `Total` et `TOTAL` désignent la même variable. Les lettres accentuées sont permises dans les noms (`numéro`, `quantité`), et le mot-clé `à` peut s'écrire `a` (`pour i = 1 a 10`).

À retenir

Une instruction par ligne ; `;` en fin de ligne pour continuer ; `&&`, `//` ou `*` pour commenter. Les noms ignorent la casse, les mots-clés s'écrivent en minuscules.

2.2 Afficher des résultats : ? et ??

L'instruction `?` affiche une ou plusieurs valeurs **séparées par un espace**, puis passe à la ligne. `??` fait de même **sans** passer à la ligne : l'affichage suivant continue sur la même ligne, collé au précédent (aucun espace n'est ajouté entre deux affichages). Un `?` seul passe simplement à la ligne.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Construire un affichage en plusieurs fois — principal.prg

```
? "Rapport des ventes"
? "------------------"
?? "Lundi : "                    && pas de retour à la ligne (notez l'espace final)
?? 120, "ventes"                 && …on continue sur la même ligne
?                                && un ? seul termine la ligne
? "Mardi :", 95, "ventes"
? "Total :", 120 + 95
? 1, "deux", vrai, nul, 4.5      && nul s'affiche comme un vide
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Rapport des ventes
------------------
Lundi : 120 ventes
Mardi : 95 ventes
Total : 215
1 deux vrai  4.5
```

Astuce

Pour aligner des colonnes, utilisez `chaine(nombre, largeur, decimales)` : `chaine(3.5, 8, 2)` donne `" 3.50"`, un texte de 8 caractères aligné à droite (chapitre 6). Pour écrire sans aucun espace ajouté, il existe aussi `es.ecrire(...)` (chapitre 9).

2.3 Les valeurs et leurs types

Toute donnée manipulée par un programme est une **valeur**, et chaque valeur a un **type**. La fonction `type(v)` donne le type d'une valeur sous forme de texte.

Type	Exemples	Ce que c'est
`nul`	`nul`	l'absence de valeur ; s'affiche vide
`logique`	`vrai`, `faux` (ou `.v.`, `.f.`)	le résultat d'une question oui / non
`nombre`	`42`, `-3`, `3.14`, `1e3`	entiers et décimaux (le point est le séparateur décimal)
`chaine`	`"texte"`, `'texte'`, `[[plusieurs lignes]]`	du texte
`table`	`{1, 2, 3}`, `{nom = "Ada"}`	une liste ou une fiche (chapitre 5)
`procedure`	`@CARRE`	une référence à une procédure (chapitre 4)
`objet`	fenêtre, fichier ouvert, connexion…	des objets fournis par les paquets

Les chaînes de caractères

Une **chaîne** (du texte) s'écrit entre guillemets doubles `"..."` ou simples `'...'`. Dans une chaîne, la barre oblique inverse `\` introduit un caractère spécial : `\n` (retour à la ligne), `\t` (tabulation), `\"`

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

(guillemet), `\\` (la barre elle-même). Pour un texte sur plusieurs lignes, on l'entoure de doubles crochets `[[ ... ]]` : tout ce qui est entre les crochets est gardé tel quel.

EXEMPLE Les types de valeurs en pratique — principal.prg

```
locale age = 36
locale nom = "Ada Lovelace"
locale majeur = age >= 18                && une comparaison donne un logique
locale adresse = nul                     && pas encore connue
locale citation = 'Elle a dit : "Bonjour"'
locale lettre = [[Madame, Monsieur,
Veuillez trouver ci-joint votre facture.
Cordialement.]]

? nom, "a", age, "ans ; majeure :", majeur
? "Type de age      :", type(age)
? "Type de nom      :", type(nom)
? "Type de majeur   :", type(majeur)
? "Type de adresse  :", type(adresse)
? "Type de {1, 2}   :", type({1, 2})
? citation
? "Colonne 1\tColonne 2"
? lettre
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Ada Lovelace a 36 ans ; majeure : vrai
Type de age      : nombre
Type de nom      : chaine
Type de majeur   : logique
Type de adresse  : nul
Type de {1, 2}   : table
Elle a dit : "Bonjour"
Colonne 1       Colonne 2
Madame, Monsieur,
Veuillez trouver ci-joint votre facture.
Cordialement.
```

nul, faux, 0 : trois choses différentes

`nul` veut dire « il n'y a pas de valeur », `faux` veut dire « la réponse est non », et `0` est un nombre comme un autre. Dans une condition (`si`), **seuls `nul` et `faux` sont faux** : le nombre `0` et la chaîne vide `""` sont considérés comme **vrais**. Pour tester « vide ou zéro », utilisez la fonction `vide(v)`.

2.4 Convertir d'un type à l'autre

Les valeurs saisies par l'utilisateur, lues dans un fichier ou reçues du réseau sont du **texte**. Pour calculer, il faut les convertir en nombres ; pour afficher joliment, on convertit les nombres en texte.

Fonction	Rôle	Exemple → résultat
`nombre(s)`	texte → nombre, `nul` **si impossible**	`nombre("42")` → 42 ; `nombre("abc")` → `nul`

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Fonction	Rôle	Exemple → résultat
`valeur(s)`	texte → nombre, **0 si impossible**, virgule acceptée	`valeur("12,5")` → 12.5
`chaine(v)`	valeur → texte	`chaine(42)` → `"42"`
`chaine(n, l, d)`	nombre → texte de largeur `l` avec `d` décimales	`chaine(3.14159, 8, 2)` → `" 3.14"`
`texte(v)`	valeur → texte (utilise `__tostring` des objets)	`texte(vrai)` → `"vrai"`

Le calcul est **tolérant** : une chaîne qui contient un nombre est acceptée par les opérateurs de calcul, et la virgule décimale française est comprise. `"12,5" * 2` vaut 25.

EXEMPLE Contrôler et convertir une saisie — principal.prg

```
&& On demande trois montants et on vérifie chacun
pour i = 1 a 3
  locale brut = saisir("Montant n°" + i + " ? ")
  locale n = nombre(brut)
  si n = nul
    ? "  « " + brut + " » n'est pas un nombre ; valeur() donnerait", valeur(brut)
  sinon
    ? "  OK :", n, "→ avec deux décimales :", chaine(n, 1, 2)
  finsi
suivant
```

▶ CE QU'AFFICHE LE PROGRAMME

```
Montant n°1 ? 12,5
  OK : 12.5 → avec deux décimales : 12.50
Montant n°2 ? abc
  « abc » n'est pas un nombre ; valeur() donnerait 0
Montant n°3 ?   42
  OK : 42 → avec deux décimales : 42.00
```

Attention

`nombre()` renvoie `nul` quand la conversion échoue : c'est ce qui permet de **détecter** une saisie invalide. `valeur()` renvoie 0, ce qui est pratique pour additionner des colonnes, mais cache les erreurs.

2.5 Les opérateurs

Les **opérateurs** combinent des valeurs pour en calculer de nouvelles.

Opérateurs	Sens	Exemple → résultat
`+ - * /`	addition, soustraction, multiplication, division	`10 / 4` → 2.5
`%`	reste de la division (du signe du diviseur)	`7 % 3` → 1 ; `-7 % 3` → 2

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Opérateurs	Sens	Exemple → résultat
`^` ou `**`	puissance	`2 ^ 10` → 1024
`+` avec du texte, `..`	concaténation (coller des textes)	`"a" + 1` → `"a1"`
`=` ou `==`	égal à	`3 = 3.0` → vrai
`<>` `!=` `#`	différent de	`2 <> 3` → vrai
`< <= > >=`	comparaison de nombres ou de textes	`"abc" < "abd"` → vrai
`$`	inclusion d'un texte dans un autre	`"jour" $ "Bonjour"` → vrai
`et` `ou` `non`	logique (aussi `.et.` `.ou.` `.non.`)	`non faux` → vrai
`#` devant une valeur	longueur d'un texte ou d'une liste	`#"héllo"` → 5

L'ordre des opérations

Comme en mathématiques, la puissance passe avant la multiplication et la division, qui passent avant l'addition et la soustraction. Les parenthèses imposent un ordre. La puissance se calcule de droite à gauche (`2 ^ 3 ^ 2` = `2 ^ 9` = 512) et s'applique avant le signe moins (`-2 ^ 2` = -4).

et, ou : plus malins qu'il n'y paraît

`et` et `ou` ne renvoient pas seulement `vrai` ou `faux` : ils renvoient **l'une des deux valeurs**. `a ou b` renvoie `a` si `a` est « vrai » (ni `nul` ni `faux`), sinon `b`. C'est très pratique pour donner une **valeur par défaut** : `nom ou "inconnu"`. De plus, la seconde valeur n'est calculée que si c'est nécessaire.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Calculatrice de remise avec les opérateurs — principal.prg

```
locale prix = 80
locale quantite = 7
locale client_fidele = vrai
locale code_promo = nul

locale brut = prix * quantite
locale remise = 0
si quantite >= 5 et client_fidele      && les deux conditions
  remise = 10
finsi
locale net = brut - brut * remise / 100

? "Brut :", brut, "€ ; remise :", remise, "% ; net :", net, "€"
? "Lots de 3 :", entier(quantite / 3), "et il reste", quantite % 3, "article(s)"
? "Code promo :", code_promo ou "aucun"              && valeur par défaut
? "Commande importante :", net > 400
? "Le mot « pro » dans « promotion » :", "pro" $ "promotion"
? "Priorités :", 2 + 3 * 4, "/", (2 + 3) * 4, "/", 2 ^ 3 ^ 2
? "Mélange texte et nombre :", "Total " + net, "/", 2 + "3", "/", "2" * "3"
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Brut : 560 € ; remise : 10 % ; net : 504 €
Lots de 3 : 2 et il reste 1 article(s)
Code promo : aucun
Commande importante : vrai
Le mot « pro » dans « promotion » : vrai
Priorités : 14 / 20 / 512
Mélange texte et nombre : Total 504 / 23 / 6
```

Le piège du « + » avec du texte

Dès qu'un des deux côtés du `+` est une **chaîne**, les deux valeurs sont **collées** : `2 + "3"` donne `"23"`, pas 5. Les autres opérateurs (`- * /`) convertissent les chaînes en nombres : `"2" * "3"` donne 6. En cas de doute, convertissez avec `nombre()`.

Comparer des textes

La comparaison de chaînes **tient compte de la casse** : `"a" = "A"` est faux. Pour comparer sans tenir compte des majuscules, comparez `majuscule(a) = majuscule(b)`.

2.6 Les variables

Une **variable** est une boîte qui porte un nom et contient une valeur. On y range une valeur avec `=` (l'**affectation**), et on peut la changer à tout moment.

- `locale x = valeur` crée une variable **locale** : elle n'existe que dans la procédure (ou le programme principal) où elle est déclarée. On peut en déclarer plusieurs : `locale a = 1, b, c = 3`.
- Un nom **non déclaré** est une variable **globale** : elle est partagée par tout le programme et rangée dans la table spéciale `SESSION`. `globale x` le dit explicitement.
- `x += 5` et `x -= 5` ajoutent ou retirent une quantité.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

- `stocker valeur dans a, b, c` range la même valeur dans plusieurs variables.
- `liberer x` remet la variable à `nul`.

EXEMPLE Un compteur de visites avec variables locales et globales — principal.prg

```
visites = 0                          && globale : partagée par tout le programme
globale site = "boutique.fr"         && globale déclarée explicitement

ENREGISTRER_VISITE("Ada")
ENREGISTRER_VISITE("Bob")
ENREGISTRER_VISITE("Chloé")
? "Visites sur", site, ":", visites
? "Lue dans SESSION :", SESSION.VISITES

stocker 0 dans lundi, mardi, mercredi
mardi += 12
? "Lundi", lundi, "/ mardi", mardi, "/ mercredi", mercredi
liberer mardi
? "Après liberer, mardi vaut", type(mardi)

procedure ENREGISTRER_VISITE(qui)
  locale message = "Bienvenue " + qui     && locale : n'existe que dans la procédure
  visites += 1                            && modifie la globale
  ? message, "(visite n°" + visites + ")"
retourner
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Bienvenue Ada (visite n°1)
Bienvenue Bob (visite n°2)
Bienvenue Chloé (visite n°3)
Visites sur boutique.fr : 3
Lue dans SESSION : 3
Lundi 0 / mardi 12 / mercredi 0
Après liberer, mardi vaut nul
```

Le piège des globales implicites

Si, dans une procédure, vous écrivez `total = 0` sans `locale`, vous créez (ou écrasez !) une variable **globale** `total` partagée par tout le programme. C'est la source d'erreur la plus fréquente. Prenez l'habitude de déclarer avec `locale` toutes les variables d'une procédure ; l'atelier vous signale les oublis (*Affichage › Signaler les variables globales implicites*).

Bien nommer ses variables

Un bon nom dit ce que contient la variable : `prix_ht`, `nb_clients`, `date_livraison` plutôt que `x`, `n2` ou `truc`. Un nom commence par une lettre (accentuée ou non) ou `_`, puis contient des lettres, des chiffres et `_`. Il ne peut pas être un mot-clé (`si`, `pour`, `table`…).

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Programme complet : ticket de caisse — principal.prg

```
&& Saisit des articles jusqu'à une ligne vide, puis imprime un ticket
locale lignes = 0
locale total = 0
? "Saisissez les articles (Entrée sur un nom vide pour terminer)"
tantque vrai
  locale article = saisir("Article ? ")
  si article = nul ou rogner(article) = ""
    sortir
  finsi
  locale prix = valeur(saisir("Prix unitaire ? "))
  locale qte = valeur(saisir("Quantité ? "))
  lignes += 1
  total += prix * qte
  ? "  →", qte, "x", article, "=", chaine(prix * qte, 1, 2), "€"
fintantque
? repliquer("=", 32)
? "Articles :", lignes, "  Total TTC :", chaine(total, 1, 2), "€"
? "dont TVA 5,5 % :", chaine(total - total / 1.055, 1, 2), "€"
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Saisissez les articles (Entrée sur un nom vide pour terminer)
Article ? Baguette
Prix unitaire ? 1,10
Quantité ? 3
  → 3 x Baguette = 3.30 €
Article ? Croissant
Prix unitaire ? 0,95
Quantité ? 4
  → 4 x Croissant = 3.80 €
Article ?
================================
Articles : 2   Total TTC : 7.10 €
dont TVA 5,5 % : 0.37 €
```

À retenir

- Types : `nul`, `logique`, `nombre`, `chaine`, `table`, `procedure`, `objet` — `type(v)` les nomme.
- `nombre()` convertit (ou renvoie `nul`), `chaine()` met en forme.
- `+` colle dès qu'il y a du texte ; `ou` donne une valeur par défaut.
- `locale` dans les procédures, toujours ; le reste est global et visible dans `SESSION`.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

CHAPITRE 3 · DÉCIDER & RÉPÉTER

Décider et répéter

Un programme qui ne sait que descendre ligne après ligne ne va pas loin. Les conditions lui permettent de choisir un chemin, les boucles de répéter un travail autant de fois que nécessaire.



3.1 Choisir avec si … sinon … finsi

3.2 Choisir parmi de nombreux cas : faire cas

3.3 Choisir une valeur : sisinon

3.4 Répéter un nombre connu de fois : pour

3.5 Répéter tant qu'une condition est vraie : tantque

3.6 Parcourir une liste ou une fiche : pour chaque

3.7 Sortir d'une boucle ou passer au tour suivant

Dans ce chapitre, vous apprendrez à…

- ✔ choisir entre plusieurs chemins avec si / sinonsi / sinon
- ✔ traiter de nombreux cas avec faire cas
- ✔ choisir une valeur avec sisinon
- ✔ répéter avec pour et tantque
- ✔ parcourir les éléments d'une liste ou d'une fiche
- ✔ sortir d'une boucle ou passer au tour suivant

3.1 Choisir avec si … sinon … finsi

L'instruction `si` exécute un bloc d'instructions **seulement si** une condition est vraie. Le bloc s'arrête au mot `finsi`. On peut ajouter un bloc `sinon`, exécuté quand la condition est fausse, et autant de `sinonsi` (ou `sinon si`) que nécessaire pour tester d'autres conditions **dans l'ordre** : dès qu'une condition est vraie, son bloc est exécuté et les suivantes sont ignorées.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Syntaxe

```
si condition
  instructions si la condition est vraie
sinonsi autre_condition        (facultatif, autant que nécessaire)
  instructions
sinon                          (facultatif)
  instructions si aucune condition n'est vraie
finsi
```

Une **condition** est toute expression dont le résultat est vrai ou faux : une comparaison (`age >= 18`), une combinaison (`a > 0 et b > 0`), un appel de fonction (`fichier.existe(chemin)`)… Rappelez-vous : seuls `nul` et `faux` comptent comme faux.

EXEMPLE Mention d'un examen — principal.prg

```
1  locale note = valeur(saisir("Note sur 20 ? "))
2
3  si note < 0 ou note > 20
4    ? "Note impossible !"
5  sinonsi note >= 16
6    ? "Mention très bien"
7  sinonsi note >= 14
8    ? "Mention bien"
9  sinonsi note >= 12
10   ? "Mention assez bien"
11 sinonsi note >= 10
12   ? "Admis sans mention"
13 sinon
14   ? "Ajourné : il manque", 10 - note, "point(s)"
15 finsi
```

CE QU'AFFICHE LE PROGRAMME

```
Note sur 20 ? 14,5
Mention bien
```

Indentez !

Décalez de deux espaces les instructions à l'intérieur d'un bloc. L'interpréteur ne l'exige pas, mais le programme devient bien plus lisible, et l'on voit tout de suite quel `finsi` ferme quel `si`.

Des conditions imbriquées

Un bloc peut contenir lui-même d'autres `si`. Voici un calcul de frais de port qui dépend du pays **et** du poids :

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Frais de port selon le pays et le poids — principal.prg

```
FRAIS("FR", 0.5, 30)                        && le programme principal, en tête
FRAIS("FR", 3, 45)
FRAIS("BE", 2, 60)
FRAIS("DE", 5, 150)

procedure FRAIS(pays, poids, montant)
  locale frais
  si montant >= 100
    frais = 0                               && livraison offerte
  sinonsi pays = "FR"
    si poids <= 1
      frais = 4.90
    sinon
      frais = 4.90 + (poids - 1) * 1.5
    finsi
  sinon
    frais = 12 + poids * 2                  && international
  finsi
  ? pays, poids, "kg,", montant, "€ →", chaine(frais, 1, 2), "€ de port"
retourner
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
FR 0.5 kg, 30 € → 4.90 € de port
FR 3 kg, 45 € → 7.90 € de port
BE 2 kg, 60 € → 16.00 € de port
DE 5 kg, 150 € → 0.00 € de port
```

⚠ **Le programme principal d'abord**

Tout ce qui suit une ligne `procedure` (ou `fonction`) appartient à cette procédure, jusqu'à la procédure suivante ou jusqu'à `finprocedure`. Écrivez donc le programme principal **en tête** du fichier, et les procédures **après**.

3.2 Choisir parmi de nombreux cas : faire cas

Quand on enchaîne beaucoup de tests, `faire cas … fincas` est plus lisible. Chaque `cas` est suivi d'une condition ; le premier `cas` vrai est exécuté, et `autrement` (facultatif) traite tous les autres.

{ } **Syntaxe**

```
faire cas
cas condition_1
  instructions
cas condition_2
  instructions
autrement
  instructions
fincas
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Un petit menu en mode texte — principal.prg

```
tantque vrai
  ? "1. Dire bonjour   2. Afficher la date   3. Lancer un dé   q. Quitter"
  locale choix = saisir("Votre choix ? ")
  faire cas
  cas choix = "1"
    ? "Bonjour !"
  cas choix = "2"
    ? "Nous sommes le", date(), "et il est", heure()
  cas choix = "3"
    ? "Le dé indique", aleatoire(6)                  && un entier de 1 à 6
  cas choix = "q" ou choix = nul
    ? "Au revoir."
    sortir
  autrement
    ? "Choix inconnu :", choix
  fincas
fintantque
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
1. Dire bonjour   2. Afficher la date   3. Lancer un dé   q. Quitter
Votre choix ? 2
Nous sommes le 2026-10-16 et il est 20:45:51
1. Dire bonjour   2. Afficher la date   3. Lancer un dé   q. Quitter
Votre choix ? 9
Choix inconnu : 9
1. Dire bonjour   2. Afficher la date   3. Lancer un dé   q. Quitter
Votre choix ? 3
Le dé indique 3
1. Dire bonjour   2. Afficher la date   3. Lancer un dé   q. Quitter
Votre choix ? q
Au revoir.
```

3.3 Choisir une valeur : sisinon

La fonction `sisinon(condition, valeur_si_vrai, valeur_si_faux)` renvoie l'une de deux valeurs. Elle s'utilise **à l'intérieur** d'une expression, là où `si … finsi` serait trop lourd. Seule la valeur choisie est calculée.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Accords et valeurs conditionnelles — principal.prg

```
pour n = 0 a 3
  ? n, sisinon(n > 1, "articles", "article"), ;
    "—", sisinon(n = 0, "panier vide", "à payer")
suivant
locale stock = 2
? "Stock :", sisinon(stock < 5, "à réapprovisionner", "suffisant")
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
0 article — panier vide
1 article — à payer
2 articles — à payer
3 articles — à payer
Stock : à réapprovisionner
```

3.4 Répéter un nombre connu de fois : pour

La boucle `pour` répète un bloc en faisant varier une **variable compteur** d'une valeur de départ à une valeur d'arrivée. Le bloc se termine par `suivant` (ou `finpour`). Avec `pas`, le compteur avance d'un autre pas que 1, éventuellement négatif. Le mot `a` peut aussi s'écrire `à` ou `jusqua`.

Syntaxe

```
pour compteur = depart a arrivee [pas increment]
  instructions
suivant
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Table de multiplication et compte à rebours — principal.prg

```
locale n = 7
? "Table de", n
pour i = 1 a 10
  ? chaine(i, 2), "x", n, "=", chaine(i * n, 3)
suivant

?? "Décollage : "
pour s = 10 a 0 pas -2
  ?? s, ""
suivant
? "partez !"

locale somme = 0
pour i = 1 a 100
  somme += i
suivant
? "1 + 2 + … + 100 =", somme
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Table de 7
 1 x 7 =   7
 2 x 7 =  14
 3 x 7 =  21
 4 x 7 =  28
 5 x 7 =  35
 6 x 7 =  42
 7 x 7 =  49
 8 x 7 =  56
 9 x 7 =  63
10 x 7 =  70
Décollage : 10 8 6 4 2 0 partez !
1 + 2 + … + 100 = 5050
```

La variable compteur est **locale** à la boucle. Les bornes sont calculées une seule fois, au début de la boucle.

3.5 Répéter tant qu'une condition est vraie : tantque

La boucle `tantque` répète son bloc **tant que** la condition est vraie ; elle se termine par `fintantque`. La condition est vérifiée **avant** chaque tour : si elle est fausse dès le départ, le bloc n'est jamais exécuté. On écrit aussi `faire tantque`.

On utilise `tantque` quand on ne sait pas à l'avance combien de tours il faudra : attendre une saisie correcte, lire un fichier jusqu'au bout, calculer jusqu'à atteindre un objectif.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Combien d'années pour doubler son épargne ? — principal.prg

```
locale capital = 1000
locale taux = 3.5                    && intérêts annuels en %
locale annees = 0
tantque capital < 2000
  capital = capital * (1 + taux / 100)
  annees += 1
fintantque
? "À", taux, "% par an, le capital double en", annees, "ans."
? "Capital final :", chaine(capital, 1, 2), "€"
```

▶ CE QU'AFFICHE LE PROGRAMME

```
À 3.5 % par an, le capital double en 21 ans.
Capital final : 2059.43 €
```

La boucle infinie

Si la condition d'un `tantque` reste toujours vraie, le programme ne s'arrête jamais. Vérifiez que quelque chose, dans le bloc, finit par rendre la condition fausse (ici `capital` augmente). En cas d'accident, le bouton **Arrêter** de l'atelier (Maj+F5) interrompt le programme.

Valider une saisie

Un usage classique : redemander tant que la réponse n'est pas valide.

EXEMPLE Redemander jusqu'à obtenir une quantité valide — principal.prg

```
locale qte = nul
tantque qte = nul ou qte <= 0
  qte = nombre(saisir("Quantité (nombre positif) ? "))
  si qte = nul ou qte <= 0
    ? "  Saisie invalide, recommencez."
  finsi
fintantque
? "Quantité retenue :", qte
```

▶ CE QU'AFFICHE LE PROGRAMME

```
Quantité (nombre positif) ? beaucoup
  Saisie invalide, recommencez.
Quantité (nombre positif) ? -3
  Saisie invalide, recommencez.
Quantité (nombre positif) ? 12
Quantité retenue : 12
```

3.6 Parcourir une liste ou une fiche : pour chaque

`pour chaque v dans liste` exécute le bloc une fois **pour chaque élément** de la liste, en rangeant l'élément dans `v`. Avec deux variables, `pour chaque k, v dans fiche`, on obtient à chaque tour une **clef** et sa **valeur** (les tables sont présentées au chapitre 5).

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Parcourir des listes et des fiches — principal.prg

```
locale fruits = {"pomme", "poire", "kiwi"}
pour chaque f dans fruits
  ? "J'aime les", f + "s"
suivant

locale stock = {pommes = 12, poires = 0, kiwis = 30}
pour chaque nom, qte dans stock
  ? nom, ":", qte, sisinon(qte = 0, "← rupture !", "")
suivant

locale total = 0
pour chaque qte dans valeurs(stock)       && seulement les valeurs
  total += qte
suivant
? "Total en stock :", total
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
J'aime les pommes
J'aime les poires
J'aime les kiwis
POMMES : 12
POIRES : 0 ← rupture !
KIWIS : 30
Total en stock : 42
```

Astuce

Dans `pour chaque k, v dans fiche`, les clefs écrites comme des noms (`pommes = 12`) sont rangées **en majuscules** (`POMMES`), car Lexis+ ne tient pas compte de la casse des noms. Elles s'affichent donc en majuscules.

Les itérateurs

`pour chaque` accepte aussi un **itérateur** : une procédure appelée à chaque tour, qui renvoie l'élément suivant, puis `nul` quand c'est fini. Beaucoup de paquets en fournissent : `es.lignes(chemin)` lit un fichier ligne par ligne, `db:lignes(sql)` parcourt les résultats d'une requête. Vous pouvez en écrire vous-même :

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Un itérateur maison : les jours ouvrés d'une semaine — principal.prg

```
jours = {"lundi", "mardi", "mercredi", "jeudi", "vendredi", "samedi", "dimanche"}
i = 0                                  && globales : partagées avec la fonction

pour chaque j dans @PROCHAIN_JOUR_OUVRE
  ? "Réunion le", j
suivant

fonction PROCHAIN_JOUR_OUVRE()
  i += 1
  si i > 5
    retourner nul                      && nul : l'itération s'arrête
  finsi
retourner jours[i]
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Réunion le lundi
Réunion le mardi
Réunion le mercredi
Réunion le jeudi
Réunion le vendredi
```

L'itérateur garde son état (le numéro du jour) dans la variable globale `i`. Pour des itérateurs plus élaborés, on range cet état dans une table ou un objet (chapitre 5).

3.7 Sortir d'une boucle ou passer au tour suivant

Deux instructions modifient le déroulement d'une boucle (`pour`, `pour chaque` ou `tantque`) :

- `sortir` quitte immédiatement la boucle ; le programme continue après `suivant` / `fintantque` ;
- `boucler` abandonne le tour en cours et passe directement au tour suivant.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Chercher le premier client en retard de paiement — principal.prg

```
locale clients = {{nom = "Durand", retard = 0}, {nom = "Martin", retard = 0}, ;
                  {nom = "Petit", retard = 45}, {nom = "Roux", retard = 12}}
locale trouve = nul
pour chaque c dans clients
  si c.retard = 0
    boucler                          && client à jour : on passe au suivant
  finsi
  ? "Vérification de", c.nom, "…"
  si c.retard > 30
    trouve = c
    sortir                           && inutile de chercher plus loin
  finsi
suivant
si trouve <> nul
  ? "Premier client à relancer :", trouve.nom, "(" + trouve.retard + " jours)"
finsi
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Vérification de Petit …
Premier client à relancer : Petit (45 jours)
```

À retenir

- `si / sinonsi / sinon / finsi` : choisir un chemin ; `faire cas / cas / autrement / fincas` : choisir parmi beaucoup ; `sisinon()` : choisir une valeur.
- `pour i = a a b [pas p] … suivant` : nombre de tours connu ; `tantque … fintantque` : tant qu'une condition est vraie.
- `pour chaque v dans liste` et `pour chaque k, v dans fiche` : parcourir ; un itérateur renvoie `nul` pour s'arrêter.
- `sortir` quitte la boucle, `boucler` passe au tour suivant.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

CHAPITRE 4 · PROCÉDURES

Procédures et fonctions

Une procédure est un morceau de programme qui porte un nom. On l'écrit une fois, on l'utilise partout. C'est l'outil essentiel pour découper un problème en petits morceaux simples, éviter les répétitions et rendre un programme lisible.



4.1 Pourquoi des procédures ?
4.2 Définir et appeler une procédure
4.3 Renvoyer un résultat : retourner
4.4 La portée des variables
4.5 Les autres façons d'appeler
4.6 La récursivité
4.7 Organiser ses procédures

Dans ce chapitre, vous apprendrez à...

- ✓ écrire une procédure et lui passer des paramètres
- ✓ comprendre la portée des variables locales
- ✓ manipuler des références de procédures (@NOM)
- ✓ renvoyer un résultat avec retourner
- ✓ appeler une procédure de plusieurs façons
- ✓ écrire une fonction récursive

4.1 Pourquoi des procédures ?

Imaginez un programme de facturation qui calcule la TVA à dix endroits différents. Le jour où le taux change, il faut corriger dix lignes... et on en oublie toujours une. Avec une **procédure** `TVA(montant)`, le calcul est écrit **une seule fois** : on le corrige à un seul endroit.

Les procédures servent aussi à **découper** un problème. Plutôt qu'un long programme de 300 lignes, on écrit `LIRE_COMMANDES()`, `CALCULER_TOTAUX()`, `IMPRIMER_FACTURES()`... Chaque morceau est court, se comprend seul, et se teste séparément.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Procédure ou fonction ?

Dans Lexis+, les mots `procedure` et `fonction` sont **équivalents**. Par convention, on dit « fonction » quand elle **renvoie un résultat** (`TTC(100)` vaut 120) et « procédure » quand elle **agit** (affiche, enregistre…) sans résultat particulier. Les noms s'écrivent souvent en MAJUSCULES pour les distinguer des variables, mais ce n'est pas obligatoire.

4.2 Définir et appeler une procédure

Une procédure commence par `procedure NOM(parametres)` (ou `fonction`). Son corps va jusqu'à `retourner`, jusqu'à la procédure suivante, ou jusqu'à `finprocedure` / `finfonction`. On l'**appelle** en écrivant son nom suivi de parenthèses.

Syntaxe

```
procedure NOM(param1, param2, ...)
  instructions
retourner [valeur]

NOM(valeur1, valeur2)                  appel
x = NOM(valeur1, valeur2)              appel en récupérant le résultat
faire NOM avec valeur1, valeur2        appel « à la dBase »
```

Les **paramètres** sont des variables locales qui reçoivent les valeurs passées à l'appel (les **arguments**), dans l'ordre. Un argument manquant vaut `nul` : c'est ainsi qu'on fait des paramètres **facultatifs**.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Une procédure d'affichage réutilisable — principal.prg

```
TITRE("Ventes du mois")
LIGNE("Stylos", 120, 1.5)
LIGNE("Cahiers", 48, 3.2)
LIGNE("Agrafeuses", 5, 14.9)
TITRE("Fin du rapport", "-")

procedure TITRE(texte, trait)
  trait = trait ou "="                  && paramètre facultatif : "=" par défaut
  ? repliquer(trait, 40)
  ? texte
  ? repliquer(trait, 40)
retourner

procedure LIGNE(article, quantite, prix)
  ? gauche(article + espace(14), 14), chaine(quantite, 5), "x", ;
    chaine(prix, 6, 2), "=", chaine(quantite * prix, 8, 2), "€"
retourner
```

CE QU'AFFICHE LE PROGRAMME

```
========================================
Ventes du mois
========================================
Stylos           120 x   1.50 =   180.00 €
Cahiers           48 x   3.20 =   153.60 €
Agrafeuses         5 x  14.90 =    74.50 €
----------------------------------------
Fin du rapport
----------------------------------------
```

4.3 Renvoyer un résultat : retourner

`retourner valeur` termine la procédure et **renvoie** la valeur à l'endroit de l'appel, où elle peut être affichée, rangée dans une variable ou utilisée dans un calcul. Une procédure peut contenir plusieurs `retourner` : le premier exécuté termine la procédure. Sans valeur, `retourner` renvoie `nul`.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Des fonctions de calcul commercial — principal.prg

```
locale ht = 250
? "HT :", ht, "→ TTC :", TTC(ht)
? "TTC à 5,5 % :", TTC(ht, 5.5)
? "Remise sur 1 200 € :", REMISE(1200), "€"
? "Remise sur 300 € :", REMISE(300), "€"
? "Prix psychologique de 20 € :", PSYCHO(20)

fonction TTC(montant, taux)
  taux = taux ou 20                    && 20 % si on ne précise pas
retourner arrondi(montant * (1 + taux / 100), 2)

fonction REMISE(montant)
  si montant >= 1000
    retourner montant * 0.10            && on sort ici pour les grosses commandes
  finsi
  si montant >= 500
    retourner montant * 0.05
  finsi
retourner 0

fonction PSYCHO(prix)
retourner plancher(prix) - 0.01
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
HT : 250 → TTC : 300
TTC à 5,5 % : 263.75
Remise sur 1 200 € : 120 €
Remise sur 300 € : 0 €
Prix psychologique de 20 € : 19.99
```

Renvoyer plusieurs valeurs

Une fonction renvoie **une** valeur. Pour en renvoyer plusieurs, on renvoie une **table** (chapitre 5) :

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Une fonction qui renvoie une fiche — principal.prg

```
locale r = DIVISION(17, 5)
? "17 = 5 x", r.quotient, "+", r.reste

locale s = STATS({12, 8, 17, 9, 14})
? "min", s.min, "/ max", s.max, "/ moyenne", s.moyenne

fonction DIVISION(a, b)
retourner {quotient = entier(a / b), reste = a % b}

fonction STATS(notes)
  locale total = 0, mini = notes[1], maxi = notes[1]
  pour chaque n dans notes
    total += n
    mini = min(mini, n)
    maxi = max(maxi, n)
  suivant
retourner {min = mini, max = maxi, moyenne = arrondi(total / #notes, 2)}
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
17 = 5 x 3 + 2
min 8 / max 17 / moyenne 12
```

4.4 La portée des variables

Les paramètres et les variables déclarées avec `locale` n'existent **que pendant l'exécution** de la procédure. Deux procédures peuvent avoir chacune une variable `total` sans qu'elles se mélangent. Une variable non déclarée est **globale** (chapitre 2) : toutes les procédures la partagent.

EXEMPLE Locales indépendantes, globale partagée — principal.prg

```
compteur_appels = 0                  && globale

? "A donne", A()
? "B donne", B()
? "A donne", A()
? "Appels :", compteur_appels

fonction A()
  locale total = 10                  && le total de A…
  compteur_appels += 1
retourner total

fonction B()
  locale total = 99                  && …n'a rien à voir avec celui de B
  compteur_appels += 1
retourner total
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
A donne 10
B donne 99
A donne 10
Appels : 3
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Les tables sont partagées

Quand on passe une **table** en argument, la procédure reçoit **la même table**, pas une copie : si elle la modifie, l'appelant voit la modification. C'est voulu (et économique), mais il faut le savoir. Pour travailler sur une copie, utilisez `copier(t)`.

4.5 Les autres façons d'appeler

faire ... avec et parametres (style dBase)

Pour les habitués de dBase et de wdGestion, Lexis+ accepte `faire NOM avec a, b` pour appeler une procédure, et `parametres a, b` pour lire les arguments en tête de la procédure.

EXEMPLE Le style dBase : faire ... avec et parametres — principal.prg

```
faire BIENVENUE avec "Ada", "Brest"
faire BIENVENUE avec "Bob"

procedure BIENVENUE
  parametres qui, ville
  ? "Bienvenue", qui + ",", "de", ville ou "quelque part"
retourner
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Bienvenue Ada, de Brest
Bienvenue Bob, de quelque part
```

Les références de procédures : @NOM

`@NOM` désigne la procédure elle-même, **sans l'appeler**. On obtient une valeur de type `procedure` qu'on peut ranger dans une variable, mettre dans une table ou passer à une autre procédure. C'est ainsi que l'on fournit une procédure de **comparaison** à `trier`, un **gestionnaire d'événement** à un bouton, une **tâche** à lancer...

- `appeler(@NOM, a, b)` appelle la procédure référencée ;
- une variable qui contient une référence s'appelle comme une procédure : `f = @NOM` puis `f(a, b)`.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Choisir un calcul à l'exécution — principal.prg

```
locale operations = {plus = @PLUS, fois = @FOIS, puissance = @PUISSANCE}
pour chaque nom, op dans operations
  ? nom, ":", appeler(op, 3, 4)
suivant

locale f = @FOIS
? "f(6, 7) =", f(6, 7), "— type :", type(f)
? "Somme des carrés :", CUMULER({1, 2, 3, 4}, @CARRE)
? "Somme des cubes  :", CUMULER({1, 2, 3, 4}, @CUBE)

fonction PLUS(a, b)
retourner a + b
fonction FOIS(a, b)
retourner a * b
fonction PUISSANCE(a, b)
retourner a ^ b
fonction CARRE(x)
retourner x * x
fonction CUBE(x)
retourner x * x * x

fonction CUMULER(liste, transformation)
  locale s = 0
  pour chaque x dans liste
    s += transformation(x)              && appelle la procédure reçue
  suivant
retourner s
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
PLUS : 7
FOIS : 12
PUISSANCE : 81
f(6, 7) = 42 — type : procedure
Somme des carrés : 30
Somme des cubes  : 100
```

À retenir

`NOM` appelle la procédure ; `@NOM` la **désigne** sans l'appeler. C'est cette référence qu'on passe à `trier`, aux boutons (`btn.surclic = @CLIC`), aux tâches (`tache.lancer(@TRAVAIL)`)…

4.6 La récursivité

Une procédure peut s'appeler elle-même : c'est la **récursivité**. Elle convient aux problèmes qui se décomposent en versions plus petites d'eux-mêmes : factorielle, parcours d'arborescence (dossiers et sous-dossiers, catégories et sous-catégories)… Il faut toujours un **cas d'arrêt**, sinon les appels ne s'arrêtent jamais. Lexis+ limite la profondeur à **200** appels imbriqués et signale une erreur au-delà.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Afficher une arborescence de catégories — principal.prg

```
locale catalogue = {nom = "Catalogue", enfants = {
  {nom = "Papeterie", enfants = {{nom = "Stylos"}, {nom = "Cahiers"}}},
  {nom = "Mobilier", enfants = {{nom = "Bureaux"},
                                {nom = "Sièges", enfants = {{nom = "Ergonomiques"}}}}}}
AFFICHER(catalogue, 0)
? "Nombre de catégories :", COMPTER(catalogue)
? "10! =", FACTORIELLE(10)

procedure AFFICHER(noeud, niveau)
  ? espace(niveau * 2) + "• " + noeud.nom
  pour chaque enfant dans noeud.enfants ou {}
    AFFICHER(enfant, niveau + 1)        && la procédure s'appelle elle-même
  suivant
retourner

fonction COMPTER(noeud)
  locale n = 1
  pour chaque enfant dans noeud.enfants ou {}
    n += COMPTER(enfant)
  suivant
retourner n

fonction FACTORIELLE(n)
  si n <= 1
    retourner 1                         && cas d'arrêt
  finsi
retourner n * FACTORIELLE(n - 1)
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
• Catalogue
  • Papeterie
    • Stylos
    • Cahiers
  • Mobilier
    • Bureaux
    • Sièges
      • Ergonomiques
Nombre de catégories : 8
10! = 3628800
```

4.7 Organiser ses procédures

Quelques règles pour des programmes qui restent clairs en grandissant :

- **Une procédure, une tâche.** Si vous avez du mal à la nommer, c'est qu'elle en fait trop.
- **Des noms parlants** : un verbe pour les procédures (`ENVOYER_FACTURE`), un nom pour les fonctions qui calculent (`TTC`, `AGE`).
- **Peu de globales.** Passez les données en paramètres et récupérez les résultats avec `retourner` : la procédure devient indépendante et facile à tester.
- **Regroupez par fichiers** : `factures.prg`, `clients.prg`, `outils.prg`… Toutes les procédures du dossier sont disponibles partout.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Programme complet : bulletin de notes découpé en fonctions — principal.prg

```
locale eleves = {{nom = "Ada", notes = {15, 17, 19}},
                 {nom = "Bob", notes = {8, 11, 9}},
                 {nom = "Chloé", notes = {12, 14, 10}}}
pour chaque e dans eleves
  IMPRIMER_BULLETIN(e)
suivant

procedure IMPRIMER_BULLETIN(eleve)
  locale m = MOYENNE(eleve.notes)
  ? gauche(eleve.nom + espace(8), 8), gauche(joindre(eleve.notes, " ") + espace(9), 9), ;
    "→", chaine(m, 5, 2), " ", MENTION(m)
retourner

fonction MOYENNE(liste)
  locale s = 0
  pour chaque n dans liste
    s += n
  suivant
retourner s / #liste

fonction MENTION(m)
  faire cas
  cas m >= 16
    retourner "Très bien"
  cas m >= 14
    retourner "Bien"
  cas m >= 10
    retourner "Admis"
  fincas
retourner "Ajourné"
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Ada      15 17 19  → 17.00   Très bien
Bob      8 11 9    →  9.33   Ajourné
Chloé    12 14 10  → 12.00   Admis
```

À retenir

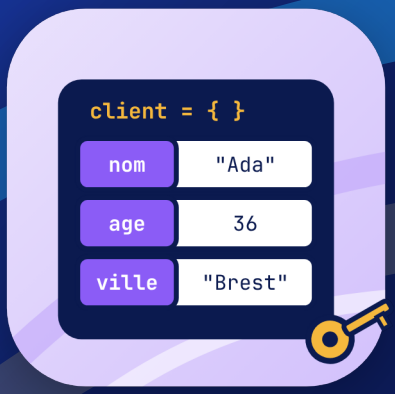
- `procedure NOM(a, b) … retourner [valeur]` ; `procedure` = `fonction`.
- Argument manquant = `nul` ; `x = x ou defaut` pour un paramètre facultatif.
- `locale` : propre à l'appel ; les tables passées en argument sont partagées.
- `@NOM` = référence ; `appeler(@NOM, …)` ou `ref(…)` pour l'appeler.
- Récursivité : toujours un cas d'arrêt ; 200 niveaux au plus.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

CHAPITRE 5 · TABLES & OBJETS

Tables, listes et objets

La table est la structure de données à tout faire de Lexis+ — à la fois liste numérotée et fiche à rubriques. Avec les métatables, elle devient un véritable objet doté de méthodes et d'opérateurs.



5.1 Une table, deux usages
5.2 Les listes
5.3 Les fiches
5.4 Des données réelles : listes de fiches
5.5 Partage, copie et JSON
5.6 Des objets avec les métatables

Dans ce chapitre, vous apprendrez à…

- ✔ créer des listes et les modifier (ajouter, insérer, retirer, trier)
- ✔ créer des fiches et accéder à leurs rubriques
- ✔ combiner listes et fiches pour représenter des données réelles
- ✔ copier, comparer et parcourir des tables
- ✔ convertir une table en JSON et inversement
- ✔ créer des objets avec méthodes et métatables

5.1 Une table, deux usages

Une **table** est un conteneur qui range plusieurs valeurs sous une seule variable. Elle s'écrit entre accolades `{ }` et sert de deux façons :

- comme **liste** : des valeurs numérotées à partir de **1** — `{"pomme", "poire", "kiwi"}` ;
- comme **fiche** : des valeurs rangées sous des **clefs** (des noms) — `{nom = "Durand", ville = "Brest"}`.

Une même table peut d'ailleurs mélanger les deux. On accède à un élément avec des crochets — `fruits[2]`, `client["Ville"]` — ou, pour une clef qui est un nom, avec un point : `client.nom`.

Une image pour s'en souvenir

Une **liste** est une file de casiers numérotés 1, 2, 3… Une **fiche** est un classeur dont chaque intercalaire porte une étiquette (« nom », « ville », « téléphone »). Une liste de fiches, c'est… un fichier de clients !

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

5.2 Les listes

EXEMPLE Gérer une liste de courses — principal.prg

```
locale courses = {"pain", "lait", "œufs"}
? "Au départ :", joindre(courses, ", "), "(" + #courses + " articles)"
? "Premier article :", courses[1], "— dernier :", courses[#courses]

inserer "beurre" dans courses                 && ajoute à la fin
inserer "café" dans courses position 1        && insère en tête
? "Après ajouts :", joindre(courses, ", ")

retirer de courses position 3 dans enleve     && retire le 3e élément
? "Retiré :", enleve, "→", joindre(courses, ", ")

courses[2] = "pain complet"                   && remplace un élément
trier(courses)                                && ordre alphabétique
? "Triée :", joindre(courses, ", ")
? "Contient du lait ?", contient(courses, "lait")
? "Élément 10 :", type(courses[10])           && hors de la liste : nul
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Au départ : pain, lait, œufs (3 articles)
Premier article : pain — dernier : œufs
Après ajouts : café, pain, lait, œufs, beurre
Retiré : lait → café, pain, œufs, beurre
Triée : beurre, café, pain complet, œufs
Contient du lait ? faux
Élément 10 : nul
```

Pour…	Instruction ou fonction
connaître la longueur	`#liste` ou `taille(liste)`
ajouter à la fin	`inserer v dans liste` ou `inserer(liste, v)`
insérer à la position p	`inserer v dans liste position p` ou `inserer(liste, v, p)`
retirer l'élément p	`retirer de liste position p dans x` ou `x = retirer(liste, p)`
retirer le dernier	`x = retirer(liste)`
trier	`trier(liste)` ou `trier(liste, @COMPARER)`
chercher une valeur	`contient(liste, v)`
assembler en texte	`joindre(liste, separateur)` ; l'inverse : `decouper(texte, separateur)`

Trier selon ses propres règles

`trier(liste, @COMPARER)` utilise une fonction de **comparaison** : elle reçoit deux éléments `a` et `b` et renvoie `vrai` si `a` doit être placé **avant** `b`. On peut ainsi trier par ordre décroissant, par longueur, ou des fiches selon une rubrique.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Classement des vendeurs — principal.prg

```
locale vendeurs = {{nom = "Martin", ca = 48200}, {nom = "Durand", ca = 61750},
                   {nom = "Petit", ca = 39900}, {nom = "Roux", ca = 61750}}
trier(vendeurs, @MEILLEUR_D_ABORD)
pour i = 1 a #vendeurs
  ? i, gauche(vendeurs[i].nom + espace(8), 8), chaine(vendeurs[i].ca, 8), "€"
suivant

fonction MEILLEUR_D_ABORD(a, b)
  si a.ca <> b.ca
    retourner a.ca > b.ca              && le plus gros chiffre d'affaires d'abord
  finsi
retourner a.nom < b.nom                && à égalité : ordre alphabétique
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
1 Durand      61750 €
2 Roux        61750 €
3 Martin      48200 €
4 Petit       39900 €
```

5.3 Les fiches

Une fiche range des valeurs sous des **clefs**. Les clefs écrites comme des noms (`nom = "Ada"`, `client.ville`) sont, comme tous les noms Lexis+, **insensibles à la casse** : elles sont rangées en majuscules (`NOM`). Une clef écrite entre crochets avec des guillemets (`client["Ville"]`) est gardée **exactement** telle quelle.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Une fiche client — principal.prg

```
nouvelle table client                          && équivaut à : client = {}
client.nom = "Durand"
client.prenom = "Élise"
client.ville = "Brest"
inserer "06 12 34 56 78" dans client clef "TELEPHONE"

? client.prenom, client.NOM, "habite", client.Ville       && casse indifférente
? "A un téléphone ?", aclef(client, "TELEPHONE")
? "A un courriel ?", aclef(client, "COURRIEL")
? "Clefs :", joindre(clefs(client), ", ")
? "Nombre de rubriques :", nbelements(client)

pour chaque rubrique, contenu dans client
  ? "  " + rubrique, "=", contenu
suivant
client.ville = nul                             && supprime la rubrique
? "Après suppression :", joindre(clefs(client), ", ")
supprimer table client                         && la variable redevient nul
? "client vaut maintenant :", type(client)
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Élise Durand habite Brest
A un téléphone ? vrai
A un courriel ? faux
Clefs : NOM, PRENOM, VILLE, TELEPHONE
Nombre de rubriques : 4
  NOM = Durand
  PRENOM = Élise
  VILLE = Brest
  TELEPHONE = 06 12 34 56 78
Après suppression : NOM, PRENOM, TELEPHONE
client vaut maintenant : nul
```

⚠ **Point ou crochets ?**

`client.ville`, `client.VILLE` et `client["VILLE"]` désignent la même rubrique. Mais `client["Ville"]` en désigne une **autre** : entre crochets et guillemets, la casse compte. Dans un programme, préférez toujours la notation pointée.

Pour…	Instruction ou fonction
créer une table vide	`t = {}` ou `nouvelle table t`
lire / écrire une rubrique	`t.cle` ou `t["CLE"]` ; `t.cle = valeur`
ajouter sous une clef	`inserer v dans t clef "CLE"`
supprimer une rubrique	`t.cle = nul`
tester l'existence	`aclef(t, "CLE")`
lister clefs, valeurs	`clefs(t)`, `valeurs(t)`
compter tout	`nbelements(t)` (liste **et** rubriques)

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Pour…	Instruction ou fonction
supprimer la table	`supprimer table t`

5.4 Des données réelles : listes de fiches

En combinant les deux, on représente n'importe quelle donnée : une commande avec ses lignes, un catalogue de produits, un planning… On écrit `commande.lignes[2].quantite` pour descendre dans la structure.

EXEMPLE Une commande et ses lignes — principal.prg

```
locale commande = {
  numero = 1042, client = "Durand SA", date = "2026-10-16",
  lignes = {
    {article = "Ramette A4", quantite = 10, prix = 4.20},
    {article = "Stylo bleu", quantite = 50, prix = 0.35},
    {article = "Classeur", quantite = 6, prix = 2.90}
  }
}

? "Commande n°" + commande.numero, "de", commande.client, "du", commande.date
locale total = 0
pour chaque l dans commande.lignes
  locale montant = l.quantite * l.prix
  total += montant
  ? "  " + gauche(l.article + espace(12), 12), chaine(l.quantite, 4), "x", ;
    chaine(l.prix, 6, 2), "=", chaine(montant, 8, 2)
suivant
? "  Total HT :", chaine(total, 1, 2), "€ pour", #commande.lignes, "lignes"
? "  Article de la 2e ligne :", commande.lignes[2].article
```

▶ CE QU'AFFICHE LE PROGRAMME

```
Commande n°1042 de Durand SA du 2026-10-16
  Ramette A4     10 x   4.20 =    42.00
  Stylo bleu     50 x   0.35 =    17.50
  Classeur        6 x   2.90 =    17.40
  Total HT : 76.90 € pour 3 lignes
  Article de la 2e ligne : Stylo bleu
```

5.5 Partage, copie et JSON

Une table n'est **jamais copiée** automatiquement : `b = a` donne un second nom à **la même** table. `copier(t)` fait une copie de premier niveau (les sous-tables restent partagées).

La fonction `json(v)` transforme une valeur en texte **JSON**, le format d'échange universel du web ; `dejson(texte)` fait l'inverse (les clefs deviennent des noms Lexis+, en majuscules). C'est aussi le moyen le plus simple d'**afficher** le contenu d'une table pour la vérifier.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Partager, copier, sérialiser — principal.prg

```
locale a = {1, 2, 3}
locale b = a                    && même table
locale c = copier(a)            && copie indépendante
b[1] = 100
c[2] = 200
? "a =", json(a), " b =", json(b), " c =", json(c)

locale produit = {ref = "ST-01", libelle = "Stylo", prix = 1.5, tags = {"bureau",
"écriture"}}
locale texte_json = json(produit)
? texte_json
locale relu = dejson('{"ref": "CA-02", "libelle": "Cahier", "stock": 12}')
? relu.ref, relu.libelle, relu.stock, "— clefs :", joindre(clefs(relu), ",")
```

▶ CE QU'AFFICHE LE PROGRAMME

```
a = [100,2,3]  b = [100,2,3]  c = [1,200,3]
{"REF":"ST-01","LIBELLE":"Stylo","PRIX":1.5,"TAGS":["bureau","écriture"]}
CA-02 Cahier 12 — clefs : REF,LIBELLE,STOCK
```

5.6 Des objets avec les métatables

Un **objet** regroupe des **données** (ses rubriques) et des **actions** qui s'appliquent à elles (ses **méthodes**). Un compte bancaire a un titulaire et un solde (données) et sait déposer et retirer (méthodes).

En Lexis+, un objet est une table, et ses méthodes sont rangées dans une autre table, sa **métatable**. On appelle une méthode avec deux-points : `compte:deposer(50)` appelle la procédure en lui passant l'objet lui-même comme **premier argument** (on l'appelle souvent `soi`).

La recette :

1. Créer la table « modèle » (la **classe**) : `Compte = {}` et `Compte.__index = Compte` (« ce qui manque dans un objet se cherche dans Compte ») ;
2. Y ranger les méthodes : `Compte:procedure COMPTE_DEPOSER comme deposer` ;
3. Créer chaque objet avec `fixermetatable({…données…}, Compte)`.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Un compte bancaire en objet — principal.prg

```
Compte = {}
Compte.__index = Compte
Compte:procedure COMPTE_DEPOSER comme deposer
Compte:procedure COMPTE_RETIRER comme retirer
Compte:procedure COMPTE_TEXTE comme __tostring     && affichage de l'objet

locale c1 = NOUVEAU_COMPTE("Ada")
locale c2 = NOUVEAU_COMPTE("Bob")
c1:deposer(150)
c2:deposer(40)
c1:retirer(30)
c2:retirer(100)                    && refusé : solde insuffisant
? c1
? c2

fonction NOUVEAU_COMPTE(titulaire)
retourner fixermetatable({titulaire = titulaire, solde = 0, operations = 0}, Compte)

procedure COMPTE_DEPOSER(soi, montant)
  soi.solde += montant
  soi.operations += 1
retourner

procedure COMPTE_RETIRER(soi, montant)
  si montant > soi.solde
    ? "Retrait de", montant, "€ refusé pour", soi.titulaire
    retourner
  finsi
  soi.solde -= montant
  soi.operations += 1
retourner

fonction COMPTE_TEXTE(soi)
retourner soi.titulaire + " : " + chaine(soi.solde, 1, 2) + " € (" + soi.operations +
sisinon(soi.operations > 1, " opérations)", " opération)")
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Retrait de 100 € refusé pour Bob
Ada : 120.00 € (2 opérations)
Bob : 40.00 € (1 opération)
```

Les métaméthodes : donner des super-pouvoirs aux tables

Certaines clefs de la métatable, qui commencent par deux soulignés, changent le comportement des opérateurs et des accès :

Métaméthode	Appelée quand…
`__index`	on lit une clef absente (table ou procédure `(t, clef)`)
`__newindex`	on écrit une clef absente — procédure `(t, clef, valeur)`
`__call`	on appelle la table comme une procédure

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Métaméthode	Appelée quand…
`__tostring`	on affiche la table ou on la convertit avec `texte()`
`__len`	on calcule `#t`
`__eq`, `__lt`, `__le`	on compare avec `=`, `<`, `<=`
`__add`, `__sub`, `__mul`, `__div`, `__mod`, `__pow`, `__unm`	on calcule `+ - * / % ^` ou `-t`
`__concat`	on concatène avec `..`

`obtenirmetatable(t)` renvoie la métatable d'une table ; `lirebrut(t, k)` et `ecrirebrut(t, k, v)` lisent et écrivent **sans** déclencher les métaméthodes.

EXEMPLE Des montants en devises avec opérateurs — principal.prg

```
Montant = {}
Montant.__index = Montant
Montant:procedure M_ADD comme __add
Montant:procedure M_MUL comme __mul
Montant:procedure M_EQ comme __eq
Montant:procedure M_LT comme __lt
Montant:procedure M_TXT comme __tostring

locale prix = EUR(19.99)
locale port = EUR(4.90)
locale total = prix * 3 + port                  && opérateurs redéfinis
? "Total :", total
? "Prix < port ?", prix < port, "— égaux ?", EUR(5) = EUR(5)

locale defauts = fixermetatable({}, {__index = @VALEUR_PAR_DEFAUT})
? "Couleur :", defauts.couleur, "/ taille :", defauts.taille

fonction EUR(v)
retourner fixermetatable({v = arrondi(v, 2)}, Montant)
fonction M_ADD(a, b)
retourner EUR(a.v + b.v)
fonction M_MUL(a, n)
retourner EUR(a.v * n)
fonction M_EQ(a, b)
retourner a.v = b.v
fonction M_LT(a, b)
retourner a.v < b.v
fonction M_TXT(a)
retourner chaine(a.v, 1, 2) + " €"
fonction VALEUR_PAR_DEFAUT(t, clef)
retourner "(" + minuscule(clef) + " non précisée)"
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Total : 64.87 €
Prix < port ? faux — égaux ? vrai
Couleur : (couleur non précisée) / taille : (taille non précisée)
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

À retenir

- `{a, b, c}` : liste (indices à partir de 1) ; `{cle = v}` : fiche ; `#t` : longueur de la liste.
- `inserer … dans`, `retirer de … position`, `trier`, `joindre`, `clefs`, `valeurs`, `aclef`, `copier`.
- Les tables sont partagées, pas copiées ; `json()` / `dejson()` pour les échanger ou les afficher.
- Objet = table + métatable : `Classe.__index = Classe`, `Classe:procedure P comme methode`, `fixermetatable(t, Classe)`, appel `obj:methode()`.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

CHAPITRE 6 · TEXTE & CALCUL

Texte, nombres, dates et sécurité

Lexis+ fournit une boîte à outils de fonctions intégrées, toujours disponibles, pour transformer du texte, calculer, manipuler des dates et protéger des mots de passe. Ce chapitre les passe toutes en revue, avec des exemples tirés de la gestion courante.

6.1 Mettre du texte en forme · 6.2 Découper et extraire · 6.3 Chercher, remplacer, tester
6.4 Caractères, HTML et URL · 6.5 Calculer avec les nombres · 6.6 Dates et heures · 6.7 Tester une valeur
6.8 Sécurité : empreintes, mots de passe, identifiants



Dans ce chapitre, vous apprendrez à…

- mettre en forme et découper du texte
- arrondir, calculer, tirer au hasard
- tester le type et le contenu d'une valeur
- chercher, remplacer, tester le contenu d'un texte
- obtenir la date et l'heure
- hacher et vérifier un mot de passe, créer des identifiants

6.1 Mettre du texte en forme

Fonction	Rôle	Exemple → résultat
`majuscule(s)`	en majuscules (accents compris)	`majuscule("été")` → `"ÉTÉ"`
`minuscule(s)`	en minuscules	`minuscule("ÉCOLE")` → `"école"`
`rogner(s)`	retire les espaces aux deux bouts	`rogner(" a b ")` → `"a b"`
`rognerg(s)` , `rognerd(s)`	à gauche seulement, à droite seulement	`rognerd("a ")` → `"a"`
`espace(n)`	n espaces	`espace(3)` → `" "`
`repliquer(s, n)`	s répété n fois	`repliquer("-", 5)` → `"-----"`
`chaine(n, l, d)`	nombre en texte, largeur l, d décimales	`chaine(3.5, 6, 2)` → `" 3.50"`
`longueur(s)`	nombre de caractères (aussi `#s`)	`longueur("héllo")` → 5

EXEMPLE Imprimer une facture alignée — principal.prg

```
locale lignes = {{"Ramette A4", 10, 4.2}, {"Stylo bille bleu", 50, 0.35}, {"Classeur", 6, 2.9}}
locale total = 0
? repliquer("=", 44)
? CENTRER("FACTURE N° 2026-0142", 44)
? repliquer("=", 44)
pour chaque l dans lignes
  locale montant = l[2] * l[3]
  total += montant
  ? COLONNE(majuscule(l[1]), 20), chaine(l[2], 4), chaine(l[3], 8, 2), chaine(montant, 10, 2)
suivant
? repliquer("-", 44)
? COLONNE("TOTAL HT", 32), chaine(total, 12, 2)
? COLONNE("TVA 20 %", 32), chaine(total * 0.2, 12, 2)
? COLONNE("TOTAL TTC", 32), chaine(total * 1.2, 12, 2)

fonction COLONNE(texte, largeur)       && texte calé à gauche sur une largeur fixe
retourner gauche(texte + espace(largeur), largeur)

fonction CENTRER(texte, largeur)
retourner espace(entier((largeur - longueur(texte)) / 2)) + texte
```

▶ CE QU'AFFICHE LE PROGRAMME

```
============================================
            FACTURE N° 2026-0142
============================================
RAMETTE A4             10     4.20      42.00
STYLO BILLE BLEU       50     0.35      17.50
CLASSEUR                6     2.90      17.40
--------------------------------------------
TOTAL HT                                 76.90
TVA 20 %                                 15.38
TOTAL TTC                                92.28
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

6.2 Découper et extraire

Fonction	Rôle	Exemple → résultat
`gauche(s, n)`	les n premiers caractères	`gauche("Bonjour", 3)` → `"Bon"`
`droite(s, n)`	les n derniers	`droite("Bonjour", 4)` → `"jour"`
`souschaine(s, debut, [n])`	n caractères à partir de debut (1 = premier)	`souschaine("Bonjour", 4)` → `"jour"`
`position(sous, s)`	position de sous dans s, 0 si absent	`position("jour", "Bonjour")` → 4
`decouper(s, [sep])`	liste des morceaux (sep `","` par défaut)	`decouper("a;b", ";")` → `{"a", "b"}`
`joindre(liste, [sep])`	l'inverse : assemble une liste	`joindre({1, 2}, "-")` → `"1-2"`

EXEMPLE Analyser une ligne de fichier et un numéro de pièce — principal.prg

```
locale ligne = "2026-10-16;DUR-0042;Durand SA;1 250,80"
locale champs = decouper(ligne, ";")
? "Date      :", champs[1]
? "Année     :", gauche(champs[1], 4), "/ mois :", souschaine(champs[1], 6, 2)
? "Client    :", champs[3], "(code", champs[2] + ")"
? "Préfixe   :", gauche(champs[2], position("-", champs[2]) - 1)
? "Numéro    :", nombre(droite(champs[2], 4))
locale montant = valeur(remplacertexte(champs[4], " ", ""))
? "Montant   :", montant, "→ TTC", arrondi(montant * 1.2, 2)
```

▶ CE QU'AFFICHE LE PROGRAMME

```
Date      : 2026-10-16
Année     : 2026 / mois : 10
Client    : Durand SA (code DUR-0042)
Préfixe   : DUR
Numéro    : 42
Montant   : 1250.8 → TTC 1500.96
```

6.3 Chercher, remplacer, tester

Fonction	Rôle
`contient(s, sous)`	vrai si s contient sous ; avec une table : vrai si la valeur y figure
`commencepar(s, debut)`	vrai si s commence par debut
`finitpar(s, fin)`	vrai si s finit par fin
`remplacertexte(s, a, b)`	remplace toutes les occurrences de a par b
`"sous" $ s`	opérateur d'inclusion (comme `contient`)

EXEMPLE Trier des fichiers et nettoyer des libellés — principal.prg

```
locale fichiers = {"ventes_2025.csv", "photo.png", "ventes_2026.csv", "notes.txt", "VENTES_mars.CSV"}
pour chaque f dans fichiers
  locale m = minuscule(f)
  si commencepar(m, "ventes") et finitpar(m, ".csv")
    ? "à importer :", f
  sinonsi contient(m, ".png")
    ? "image ignorée :", f
  finsi
suivant
locale libelle = "  Stylo   BILLE --  bleu  "
locale propre = rogner(libelle)
tantque contient(propre, "  ")
  propre = remplacertexte(propre, "  ", " ")        && espaces doublés
fintantque
? "[" + remplacertexte(propre, " -- ", " ") + "]"
```

▶ CE QU'AFFICHE LE PROGRAMME

```
à importer : ventes_2025.csv
image ignorée : photo.png
à importer : ventes_2026.csv
à importer : VENTES_mars.CSV
[Stylo BILLE bleu]
```

Astuce

Pour réduire des espaces multiples à un seul, on remplace « deux espaces » par « un espace » **tant qu'il en reste** : un seul remplacement ne suffit pas pour trois espaces à la suite.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

6.4 Caractères, HTML et URL

Fonction	Rôle	Exemple → résultat
`car(code, ...)`	caractères à partir de codes Unicode	`car(65, 233, 8364)` → `"Aé€"`
`code(s, [i])`	code Unicode du i-ème caractère	`code("A")` → 65
`html(s)`	échappe `< > & " '` pour l'insérer dans une page web	`html("a<b")` → `"a<b"`
`url(s)`	encode un texte pour une adresse web	`url("é à")` → `"%C3%A9%20%C3%A0"`

EXEMPLE Fabriquer un lien et une page HTML sûrs — principal.prg

```
locale produit = "Câble <USB-C> & adaptateur"
locale recherche = "https://exemple.fr/recherche?q=" + url(produit)
? recherche
? "<a href=\"" + html(recherche) + "\">" + html(produit) + "</a>"
? "Saut de ligne :", code(car(10)), "— tabulation :", code(car(9))
? "Alphabet :", ALPHABET()

fonction ALPHABET()
  locale s = ""
  pour c = code("A") a code("Z")
    s = s + car(c)
  suivant
retourner s
```

▶ CE QU'AFFICHE LE PROGRAMME

```
https://exemple.fr/recherche?q=C%C3%A2ble%20%3CUSB-C%3E%20%26%20adaptateur
<a href="https://exemple.fr/recherche?q=C%C3%A2ble%20%3CUSB-C%3E%20%26%20adaptateur">Câble &lt;USB-C&gt; &amp; adaptateur</a>
Saut de ligne : 10 — tabulation : 9
Alphabet : ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Sécurité

Tout texte venant d'un utilisateur doit passer par `html()` avant d'être inséré dans une page web, sinon un petit malin peut y glisser du code JavaScript. De même, utilisez `url()` pour fabriquer une adresse.

6.5 Calculer avec les nombres

Fonction	Rôle	Exemple → résultat
`abs(x)`	valeur absolue	`abs(-3)` → 3
`entier(x)`	partie entière (vers zéro)	`entier(-3.7)` → -3
`arrondi(x, [d])`	arrondi à d décimales (0 par défaut)	`arrondi(2.345, 2)` → 2.35
`plancher(x)`, `plafond(x)`	entier inférieur, supérieur	`plancher(-1.5)` → -2 ; `plafond(1.2)` → 2
`racine(x)`	racine carrée	`racine(16)` → 4
`mod(a, b)`	reste (du signe du diviseur), comme `a % b`	`mod(-7, 3)` → 2
`max(a, b, ...)`, `min(a, b, ...)`	plus grande / plus petite valeur (nombres ou textes)	`max(3, 9, 1)` → 9
`aleatoire([n])`	réel entre 0 et 1, ou entier entre 1 et n	`aleatoire(6)` → 1 à 6
`sin cos tan asin acos atan(y, [x])`	trigonométrie (radians)	`atan(1, 1)` → 0.785…
`exp(x)`, `log(x, [base])`	exponentielle, logarithme (népérien par défaut)	`log(100, 10)` → 2
`pi()`	π	3.14159…

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Mensualités d'un prêt et tirage au sort — principal.prg

```
locale capital = 15000
locale taux_annuel = 4.2
locale mois = 48
locale t = taux_annuel / 100 / 12
locale mensualite = capital * t / (1 - (1 + t) ^ (-mois))
? "Mensualité :", arrondi(mensualite, 2), "€"
? "Coût total du crédit :", arrondi(mensualite * mois - capital, 2), "€"
? "Nombre de chiffres de", capital, ":", plancher(log(capital, 10)) + 1
? "Lots de 12 :", plafond(1000 / 12), "cartons pour 1000 bouteilles"

&& Tirage d'un gagnant parmi des participants (sans remise)
locale inscrits = {"Ada", "Bob", "Chloé", "David", "Emma"}
locale gagnant = retirer(inscrits, aleatoire(#inscrits))
? "Gagnant :", gagnant, "— il reste", #inscrits, "participants"

&& Distance entre deux villes (formule de haversine)
? "Paris → Brest :", arrondi(DISTANCE(48.8566, 2.3522, 48.3904, -4.4861)), "km"

fonction DISTANCE(lat1, lon1, lat2, lon2)
  locale r = 6371, rad = pi() / 180
  locale dlat = (lat2 - lat1) * rad, dlon = (lon2 - lon1) * rad
  locale a = sin(dlat / 2) ^ 2 + cos(lat1 * rad) * cos(lat2 * rad) * sin(dlon / 2) ^ 2
retourner 2 * r * atan(racine(a), racine(1 - a))
```

▶ CE QU'AFFICHE LE PROGRAMME

```
Mensualité : 340.03 €
Coût total du crédit : 1321.44 €
Nombre de chiffres de 15000 : 5
Lots de 12 : 84 cartons pour 1000 bouteilles
Gagnant : Chloé — il reste 4 participants
Paris → Brest : 505 km
```

Les nombres décimaux

Comme dans la plupart des langages, `0.1 + 0.2` ne donne pas exactement `0.3` mais `0.30000000000000004` : les ordinateurs calculent en binaire. Pour les montants, **arrondissez** au moment d'afficher ou d'enregistrer : `arrondi(x, 2)` ou `chaine(x, 1, 2)`.

6.6 Dates et heures

Fonction	Rôle	Exemple de résultat
`date()`	date du jour, AAAA-MM-JJ	`"2026-10-16"`
`heure()`	heure, HH:MM:SS	`"14:05:09"`
`maintenant()`	date et heure	`"2026-10-16 14:05:09"`
`horodatage()`	secondes depuis le 1er janvier 1970 (avec décimales)	`1792152309.42`

Les dates au format `AAAA-MM-JJ` ont un avantage : leur **ordre alphabétique est l'ordre chronologique**. On les compare donc directement avec `<` et `>`. Pour les calculs plus poussés (jours de la semaine, écarts, mises en forme comme « vendredi 16 octobre »), le paquet `systeme` fournit `systeme.temps()`, `systeme.date()` et `systeme.ecart()` (chapitre 10).

EXEMPLE Échéances et durée d'un traitement — principal.prg

```
locale debut = horodatage()
locale echeances = {"2026-09-30", "2026-10-15", "2026-11-30", "2026-12-31"}
locale aujourd_hui = "2026-10-16"
pour chaque e dans echeances
  si e < aujourd_hui
    ? e, ": échue"
  sinon
    ? e, ": à venir"
  finsi
suivant
? "Année :", gauche(date(), 4), "— trimestre :", plafond(nombre(souschaine(aujourd_hui, 6, 2)) / 3)
locale s = 0
pour i = 1 a 200000
  s += i
suivant
? "Calcul terminé en", arrondi((horodatage() - debut) * 1000), "millisecondes"
```

▶ CE QU'AFFICHE LE PROGRAMME

```
2026-09-30 : échue
2026-10-15 : échue
2026-11-30 : à venir
2026-12-31 : à venir
Année : 2026 — trimestre : 4
Calcul terminé en 33 millisecondes
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

6.7 Tester une valeur

Fonction	Rôle
`type(v)`	`"nul"`, `"logique"`, `"nombre"`, `"chaine"`, `"table"`, `"procedure"`, `"objet"` ou `"tache"`
`estnul(v)`	vrai si v vaut `nul`
`vide(v)`	vrai pour `nul`, `faux`, 0, une chaîne blanche ou une table sans élément
`sisinon(c, a, b)`	a si c est vraie, sinon b (seule la branche choisie est calculée)

EXEMPLE Valider un formulaire — principal.prg

```
VALIDER({nom = "Durand", email = "durand@exemple.fr", age = 41})
VALIDER({nom = "  ", email = "pas-un-email", age = "quarante"})

procedure VALIDER(f)
  locale erreurs = {}
  si vide(f.nom)
    inserer "le nom est obligatoire" dans erreurs
  finsi
  si non contient(f.email ou "", "@")
    inserer "l'adresse e-mail est invalide" dans erreurs
  finsi
  si type(f.age) <> "nombre"
    inserer "l'âge doit être un nombre (reçu : " + type(f.age) + ")" dans erreurs
  finsi
  si vide(erreurs)
    ? "Formulaire de", f.nom, ": valide"
  sinon
    ? "Formulaire refusé :", joindre(erreurs, " ; ")
  finsi
retourner
```

▶ CE QU'AFFICHE LE PROGRAMME

```
Formulaire de Durand : valide
Formulaire refusé : le nom est obligatoire ; l'adresse e-mail est invalide ; l'âge doit être un nombre (reçu : chaine)
```

6.8 Sécurité : empreintes, mots de passe, identifiants

Fonction	Rôle
`hachage(s)`	empreinte SHA-256 en hexadécimal (pour vérifier qu'un contenu n'a pas changé ; **pas** pour les mots de passe)
`hachermdp(mdp)`	empreinte PBKDF2-SHA256 **salée** d'un mot de passe : c'est elle qu'on enregistre
`verifiermdp(mdp, empreinte)`	vérifie un mot de passe saisi contre l'empreinte (en temps constant)
`mdpaleatoire([n])`	mot de passe aléatoire (16 caractères par défaut)
`uuid()`	identifiant unique aléatoire, du type `"3f2a…-…"`

Pourquoi « hacher » un mot de passe ?

On ne stocke **jamais** un mot de passe en clair : si la base est volée, tous les comptes le sont. On stocke une **empreinte** calculée par `hachermdp()`, impossible à « inverser ». Pour vérifier une connexion, `verifiermdp()` recalcule l'empreinte du mot de passe tapé et la compare. Chaque empreinte contient un « sel » aléatoire : deux utilisateurs ayant le même mot de passe ont des empreintes différentes.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Création de compte et connexion — principal.prg

```
&& À la création du compte : on enregistre l'empreinte, jamais le mot de passe
locale comptes = {}
locale provisoire = "Lovelace-1815!"
comptes.ADA = {id = uuid(), empreinte = hachermdp(provisoire)}
? "Compte créé, identifiant de", longueur(comptes.ADA.id), "caractères"
? "Empreinte enregistrée :", gauche(comptes.ADA.empreinte, 22) + "…"

&& À la connexion : deux tentatives
pour essai = 1 a 2
  locale qui = majuscule(saisir("Identifiant ? "))
  locale mdp = saisir("Mot de passe ? ")
  si aclef(comptes, qui) et verifiermdp(mdp, comptes[qui].empreinte)
    ? "  Connexion réussie. Bienvenue !"
    sortir
  sinon
    ? "  Identifiant ou mot de passe incorrect."
  finsi
suivant
? "Nouveau mot de passe proposé : (" + longueur(mdpaleatoire(20)) + " caractères aléatoires)"
? "Empreinte SHA-256 de « abc » :", gauche(hachage("abc"), 16) + "…"
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Compte créé, identifiant de 36 caractères
Empreinte enregistrée : pbkdf2-sha256$120000$a…
Identifiant ? ada
Mot de passe ? motdepasse-faux
  Identifiant ou mot de passe incorrect.
Identifiant ? ada
Mot de passe ? Lovelace-1815!
  Connexion réussie. Bienvenue !
Nouveau mot de passe proposé : (20 caractères aléatoires)
Empreinte SHA-256 de « abc » : ba7816bf8f01cfea…
```

À retenir

- Texte : `majuscule minuscule rogner gauche droite souschaine position decouper joindre remplacertexte contient commencepar finitpar chaine r`
- Nombres : `abs entier arrondi plancher plafond racine mod max min aleatoire` et la trigonométrie.
- Dates : `date heure maintenant horodatage` (et le paquet `systeme` pour aller plus loin).
- Tests : `type estnul vide sisinon` ; sécurité : `hachermdp verifiermdp mdpaleatoire uuid hachage`.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

CHAPITRE 7 · ERREURS

Prévoir et traiter les erreurs

Un fichier absent, une division par zéro, une saisie farfelue, un serveur qui ne répond pas… Un programme solide prévoit l'imprévu. Lexis+ signale chaque problème par une erreur, que l'on peut intercepter pour réagir proprement.



7.1 Quand quelque chose tourne mal

7.2 Intercepter une erreur : essayer … capturer

7.3 Signaler une erreur : l'instruction erreur

7.4 Relancer et journaliser

7.5 Une stratégie pour vos programmes

Dans ce chapitre, vous apprendrez à…

- lire et comprendre un message d'erreur
- signaler soi-même une erreur avec erreur
- relancer une erreur et la journaliser
- intercepter une erreur avec essayer / capturer
- transmettre des informations détaillées dans une erreur
- adopter une stratégie de traitement des erreurs

7.1 Quand quelque chose tourne mal

Lorsqu'une instruction ne peut pas s'exécuter — diviser par zéro, lire un fichier qui n'existe pas, appeler une procédure inconnue —, Lexis+ **lève une erreur**. Si personne ne l'intercepte, le programme **s'arrête** et affiche un message qui indique **le fichier, la ligne et la cause** :

```
principal.prg:12: division par zéro
```

Dans l'atelier, le curseur est placé sur la ligne fautive, et la **pile d'appels** montre par quelles procédures le programme est passé pour y arriver. Lisez toujours le message en entier : il dit presque toujours exactement ce qui ne va pas.

Les messages les plus fréquents :

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Message	Cause probable
`division par zéro`	un diviseur vaut 0 (souvent un total ou un nombre d'éléments vide)
`indexation d'une valeur nulle (clef « x »)`	on lit `t.x` alors que `t` vaut `nul` (table jamais créée, faute de frappe)
`procédure ou fonction inconnue : nom`	faute de frappe, ou fichier `.prg` absent du dossier du projet
`opération « * » : nombre attendu, chaine "abc" reçu`	un calcul sur un texte qui n'est pas un nombre
`argument n°2 invalide pour « gauche » (nombre attendu…)`	mauvais type d'argument passé à une fonction
`fichier.txt : lecture impossible`	fichier absent, mal nommé, ou droits insuffisants
`profondeur d'appel maximale atteinte`	récursivité sans cas d'arrêt

7.2 Intercepter une erreur : essayer … capturer

Le bloc `essayer` exécute des instructions « sous surveillance ». Si une erreur se produit, l'exécution saute immédiatement au bloc `capturer`, qui reçoit le **message** de l'erreur dans une variable ; le programme continue ensuite normalement après `finessayer`. Si tout se passe bien, le bloc `capturer` est ignoré.

Syntaxe

```
essayer
  instructions qui peuvent échouer
capturer e                    (la variable e reçoit l'erreur ; elle est facultative)
  instructions de secours
finessayer
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Calculer une moyenne sans planter — principal.prg

```
pour chaque liste dans {{12, 15, 9}, {}, {10, "abc", 14}}
  locale m = MOYENNE_SURE(liste)
  ? "Moyenne de", json(liste), ":", m ou "non calculable"
suivant
? "Le programme continue normalement."

fonction MOYENNE_SURE(notes)
  essayer
    locale total = 0
    pour chaque n dans notes
      total += n * 1                 && échoue si n n'est pas un nombre
    suivant
    retourner arrondi(total / #notes, 2)   && échoue si la liste est vide
  capturer e
    ? "  (calcul impossible :", e + ")"
    retourner nul
  finessayer
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Moyenne de [12,15,9] : 12
  (calcul impossible : division par zéro)
Moyenne de {} : non calculable
  (calcul impossible : opération « * » : nombre attendu, chaine "abc" reçu)
Moyenne de [10,"abc",14] : non calculable
Le programme continue normalement.
```

L'erreur est interceptée **même si elle se produit dans une procédure appelée** depuis le bloc `essayer`, aussi profondément que ce soit : elle « remonte » la pile d'appels jusqu'au premier `essayer` rencontré.

7.3 Signaler une erreur : l'instruction erreur

Vos propres procédures doivent, elles aussi, signaler les situations anormales. L'instruction `erreur valeur` lève une erreur, exactement comme le font les fonctions intégrées. C'est bien plus sûr que de renvoyer une valeur spéciale que l'appelant risque d'oublier de tester.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Débiter un stock en refusant les opérations impossibles — principal.prg

```
stock = {stylos = 120, cahiers = 8}

COMMANDER("stylos", 20)
COMMANDER("cahiers", 15)
COMMANDER("gommes", 3)
COMMANDER("stylos", -5)
? "Stock final :", json(stock)

procedure COMMANDER(article, quantite)
  essayer
    DEBITER(article, quantite)
    ? "OK :", quantite, article, "— reste", stock[majuscule(article)]
  capturer e
    ? "REFUS :", e
  finessayer
retourner

procedure DEBITER(article, quantite)
  locale cle = majuscule(article)
  si quantite <= 0
    erreur "quantité invalide (" + quantite + ")"
  finsi
  si non aclef(stock, cle)
    erreur "article inconnu : " + article
  finsi
  si stock[cle] < quantite
    erreur "stock insuffisant pour " + article + " : " + stock[cle] + " disponible(s)"
  finsi
  stock[cle] -= quantite
retourner
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
OK : 20 stylos — reste 100
REFUS : stock insuffisant pour cahiers : 8 disponible(s)
REFUS : article inconnu : gommes
REFUS : quantité invalide (-5)
Stock final : {"STYLOS":100,"CAHIERS":8}
```

Des erreurs qui transportent des informations

La valeur passée à `erreur` n'est pas forcément un texte : ce peut être une **table**. Le bloc `capturer` la reçoit telle quelle, ce qui permet de transmettre un code, un message, la donnée fautive…

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Erreurs structurées avec un code — principal.prg

```
pour chaque ref dans {"A-100", "B-7", "Z-999"}
  essayer
    locale p = CHERCHER_PRODUIT(ref)
    ? ref, "→", p.libelle, p.prix, "€"
  capturer e
    si type(e) = "table"
      ? ref, "→ erreur", e.code + " :", e.message
    sinon
      ? ref, "→ erreur imprévue :", e
    finsi
  finessayer
suivant

fonction CHERCHER_PRODUIT(ref)
  locale catalogue = {{ref = "A-100", libelle = "Agenda", prix = 12.5},
                      {ref = "B-7", libelle = "Bloc-notes", prix = 3.2}}
  pour chaque p dans catalogue
    si p.ref = ref
      retourner p
    finsi
  suivant
  erreur {code = 404, message = "produit introuvable", ref = ref}
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
A-100 → Agenda 12.5 €
B-7 → Bloc-notes 3.2 €
Z-999 → erreur 404 : produit introuvable
```

7.4 Relancer et journaliser

Un bloc `capturer` peut lui-même lever une erreur, par exemple pour **ajouter du contexte** avant de la transmettre plus haut. Les blocs `essayer` peuvent s'imbriquer : l'erreur relancée est interceptée par le bloc englobant.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Importer des lignes en journalisant les rejets — principal.prg

```
locale lignes = {"Durand;1200", "Martin;abc", "Petit;450", ";300", "Roux;980"}
locale journal = {}
locale importees = 0
pour i = 1 a #lignes
  essayer
    IMPORTER(lignes[i])
    importees += 1
  capturer e
    inserer "ligne " + i + " : " + e dans journal      && on note et on continue
  finessayer
suivant
? importees, "ligne(s) importée(s),", #journal, "rejet(s)"
pour chaque j dans journal
  ? "  -", j
suivant

procedure IMPORTER(ligne)
  locale c = decouper(ligne, ";")
  essayer
    si vide(c[1])
      erreur "nom manquant"
    finsi
    si nombre(c[2]) = nul
      erreur "montant « " + c[2] + " » non numérique"
    finsi
  capturer e
    erreur "rejet de « " + ligne + " » (" + e + ")"   && relance avec du contexte
  finessayer
retourner
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
3 ligne(s) importée(s), 2 rejet(s)
  - ligne 2 : rejet de « Martin;abc » (montant « abc » non numérique)
  - ligne 4 : rejet de « ;300 » (nom manquant)
```

7.5 Une stratégie pour vos programmes

- **Interceptez au bon niveau.** Une procédure de bas niveau (lire un fichier, calculer) **signale** l'erreur ; c'est la procédure qui sait quoi faire (prévenir l'utilisateur, passer à la ligne suivante, annuler une transaction) qui l'**intercepte**.
- **Ne masquez pas les erreurs.** Un `capturer` vide qui ne fait rien cache les problèmes : au minimum, affichez ou journalisez le message.
- **Validez tôt.** Vérifiez les saisies dès leur arrivée (`nombre()`, `vide()`, `type()`) : les messages seront plus clairs.
- **Pour les données**, combinez erreurs et **transactions** (chapitre 8) : en cas d'erreur, la base revient à son état d'avant.
- **Pour arrêter le programme** volontairement, utilisez `systeme.quitter(code)` ; une erreur non interceptée l'arrête aussi, avec un code de retour 1.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Programme complet : convertisseur robuste — principal.prg

```
taux = {EUR = 1, USD = 1.08, GBP = 0.86, CHF = 0.95}
tantque vrai
  locale saisie = saisir("Montant (q pour quitter) ? ")
  si saisie = nul ou saisie = "q"
    sortir
  finsi
  essayer
    locale r = CONVERTIR(saisie, saisir("De (EUR, USD…) ? "), saisir("Vers ? "))
    ? "  =", chaine(r, 1, 2)
  capturer e
    ? "  Impossible :", e
  finessayer
fintantque
? "Fin du convertisseur."

fonction CONVERTIR(montant, de, vers)
  locale m = nombre(montant)
  si m = nul
    erreur "« " + montant + " » n'est pas un montant"
  finsi
  de = majuscule(de ou "")
  vers = majuscule(vers ou "")
  si non aclef(taux, de) ou non aclef(taux, vers)
    erreur "devise inconnue (" + de + " → " + vers + ")"
  finsi
retourner m / taux[de] * taux[vers]
```

▶ CE QU'AFFICHE LE PROGRAMME

```
Montant (q pour quitter) ? 100
De (EUR, USD…) ? EUR
Vers ? USD
  = 108.00
Montant (q pour quitter) ? abc
De (EUR, USD…) ? EUR
Vers ? USD
  Impossible : « abc » n'est pas un montant
Montant (q pour quitter) ? 50
De (EUR, USD…) ? EUR
Vers ? XYZ
  Impossible : devise inconnue (EUR → XYZ)
Montant (q pour quitter) ? q
Fin du convertisseur.
```

À retenir

- Une erreur non interceptée arrête le programme et indique fichier, ligne et cause.
- `essayer … capturer e … finessayer` intercepte, même depuis des procédures appelées.
- `erreur "message"` ou `erreur {code = …, message = …}` signale une situation anormale.
- On peut relancer une erreur dans `capturer` pour ajouter du contexte.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

CHAPITRE 8 · BASES DE DONNÉES

Les bases de données

Clients, factures, stocks, rendez-vous : une application de gestion vit de ses données. Lexis+ embarque la base DuckDB, que l'on interroge en SQL ou enregistrement par enregistrement à la manière de dBase, et sait aussi dialoguer avec SQLite et Firebird.



8.1 Une base de données, c'est quoi ?

8.2 Créer des tables et ajouter des données

8.3 Interroger la base

8.4 Modifier, supprimer et regrouper en transaction

8.5 Les zones : travailler enregistrement par enregistrement

8.6 Se déplacer et chercher dans une zone

8.7 Modifier les données dans une zone

8.8 Plusieurs zones en même temps

8.9 SQLite

8.10 Firebird

Dans ce chapitre, vous apprendrez à…

- ✔ comprendre tables, lignes, colonnes et le langage SQL
- ✔ ajouter, lire, modifier et supprimer des données en SQL
- ✔ parcourir et modifier une table dans une zone (style dBase)
- ✔ ouvrir une base et créer des tables
- ✔ utiliser des paramètres pour des requêtes sûres
- ✔ utiliser SQLite et Firebird

8.1 Une base de données, c'est quoi ?

Une **base de données** range des informations dans des **tables**, comme un classeur range des feuilles de calcul. Chaque table a des **colonnes** (les rubriques : code, nom, ville…) et des **lignes** (les enregistrements : un client par ligne). Contrairement à de simples fichiers, une base sait retrouver très vite une information parmi des millions, garantir la cohérence des données et supporter plusieurs utilisateurs.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

On parle à une base en **SQL**, un langage spécialisé (et en anglais) : `create table` crée une table, `insert` ajoute des lignes, `select` les lit, `update` les modifie, `delete` les supprime.

Lexis+ intègre **DuckDB**, une base rapide qui tient dans un seul fichier. Par défaut, c'est le fichier `donnees.duckdb` du dossier du programme, créé automatiquement à la première requête : vous n'avez rien à installer ni à configurer (à part la bibliothèque DuckDB, chapitre 1).

Fonction	Rôle
`ouvrirbase(chemin)`	ouvre (ou crée) une base ; `":memoire:"` crée une base temporaire en mémoire
`fermerbase()`	ferme la base et toutes les zones
`base()`	chemin de la base ouverte (`""` si aucune)

Astuce

Pour vos essais, `ouvrirbase(":memoire:")` crée une base qui disparaît à la fin du programme : idéal pour expérimenter sans rien abîmer. Tous les exemples de ce chapitre l'utilisent.

8.2 Créer des tables et ajouter des données

L'instruction `sql "…"` exécute une commande SQL. Les valeurs variables ne doivent **jamais** être collées dans le texte de la requête : on écrit des **points d'interrogation** `?` et on fournit les valeurs après le mot `avec`, dans l'ordre.

Syntaxe

```
sql "commande SQL"
sql "commande avec ? et ?" avec valeur1, valeur2
sql "select …" [avec valeurs] dans variable        (le résultat va dans une liste)
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Créer le fichier des clients — principal.prg

```
ouvrirbase(":memoire:")
sql "create table clients (code varchar primary key, nom varchar, ville varchar, ca
double)"

&& Une ligne à la fois, avec des paramètres
sql "insert into clients values (?, ?, ?, ?)" avec "C001", "Durand", "Lyon", 12500
sql "insert into clients values (?, ?, ?, ?)" avec "C002", "Martin", "Brest", 8300

&& Plusieurs lignes depuis une liste Lexis+
locale nouveaux = {{"C003", "Petit", "Lyon", 4500}, {"C004", "Roux", "Nantes", 15200},
                   {"C005", "L'Hôte", "Brest", 990}}
pour chaque n dans nouveaux
  sql "insert into clients values (?, ?, ?, ?)" avec n[1], n[2], n[3], n[4]
suivant
? "Clients enregistrés :", valeursql("select count(*) from clients")
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Clients enregistrés : 5
```

⚠ **Toujours des paramètres « ? »**

N'écrivez jamais `sql "insert into clients values ('" + nom + "')"` : le nom `L'Hôte` (avec une apostrophe) casserait la requête, et un utilisateur malveillant pourrait y glisser ses propres commandes SQL (« injection SQL »). Avec `?` et `avec`, Lexis+ transmet les valeurs **séparément**, en toute sécurité.

8.3 Interroger la base

Une requête `select` renvoie des **lignes**. Avec `dans variable`, Lexis+ range le résultat dans une **liste de fiches** : une fiche par ligne, une rubrique par colonne. Les noms de colonnes sont des noms Lexis+, donc insensibles à la casse : `l.nom`, `l.NOM`.

Fonction	Rôle
`sql "select …" avec … dans liste`	liste des lignes (fiches)
`requete(sql, valeurs...)`	la même chose, sous forme de fonction
`valeursql(sql, valeurs...)`	la première valeur de la première ligne, ou `nul`

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Statistiques des ventes par ville — principal.prg

```
ouvrirbase(":memoire:")
sql "create table clients (code varchar, nom varchar, ville varchar, ca double)"
sql "insert into clients values ('C1','Durand','Lyon',12500),
('C2','Martin','Brest',8300), ('C3','Petit','Lyon',4500), ('C4','Roux','Nantes',15200),
('C5','Lhote','Brest',990)"

? "Clients de plus de 5 000 € :"
sql "select nom, ville, ca from clients where ca > ? order by ca desc" avec 5000 dans gros
pour chaque c dans gros
  ? "  " + gauche(c.nom + espace(8), 8), gauche(c.ville + espace(7), 7), chaine(c.ca, 8,
2)
suivant

? "Par ville :"
pour chaque v dans requete("select ville, count(*) as nb, sum(ca) as total from clients
group by ville order by total desc")
  ? "  " + gauche(v.ville + espace(7), 7), v.nb, "client(s),", chaine(v.total, 9, 2), "€"
suivant

? "CA total :", valeursql("select sum(ca) from clients"), "€"
? "Meilleur client :", valeursql("select nom from clients order by ca desc limit 1")
? "Client C9 :", valeursql("select nom from clients where code = ?", "C9") ou "(inconnu)"
```

▶ CE QU'AFFICHE LE PROGRAMME

```
Clients de plus de 5 000 € :
  Roux     Nantes  15200.00
  Durand   Lyon    12500.00
  Martin   Brest    8300.00
Par ville :
  Lyon    2 client(s),  17000.00 €
  Nantes  1 client(s),  15200.00 €
  Brest   2 client(s),   9290.00 €
CA total : 41490 €
Meilleur client : Roux
Client C9 : (inconnu)
```

8.4 Modifier, supprimer et regrouper en transaction

`update` modifie des lignes existantes, `delete` en supprime. N'oubliez pas la clause `where`, sinon **toutes** les lignes sont concernées !

Une **transaction** regroupe plusieurs commandes qui doivent réussir **ensemble** ou pas du tout : un virement débite un compte **et** en crédite un autre. On l'ouvre par `begin transaction`, on la valide par `commit`, ou on l'annule par `rollback`. Combinée à `essayer`, elle garantit qu'une erreur ne laisse jamais la base à moitié modifiée.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Un virement entre deux comptes — principal.prg

```
ouvrirbase(":memoire:")
sql "create table comptes (id integer primary key, titulaire varchar, solde double)"
sql "insert into comptes values (1, 'Ada', 500), (2, 'Bob', 100)"

VIRER(1, 2, 200)
VIRER(2, 1, 1000)                && refusé : solde insuffisant
pour chaque c dans requete("select * from comptes order by id")
  ? c.titulaire, ":", c.solde, "€"
suivant
sql "delete from comptes where solde < ?" avec 0
sql "update comptes set titulaire = upper(titulaire)"
? "Titulaires :", joindre(LISTE_TITULAIRES(), ", ")

procedure VIRER(de, vers, montant)
  sql "begin transaction"
  essayer
    sql "update comptes set solde = solde - ? where id = ?" avec montant, de
    sql "update comptes set solde = solde + ? where id = ?" avec montant, vers
    si valeursql("select solde from comptes where id = ?", de) < 0
      erreur "solde insuffisant"
    finsi
    sql "commit"
    ? "Virement de", montant, "€ effectué"
  capturer e
    sql "rollback"               && la base revient à son état d'avant
    ? "Virement de", montant, "€ annulé :", e
  finessayer
retourner

fonction LISTE_TITULAIRES()
  locale l = {}
  pour chaque c dans requete("select titulaire from comptes order by id")
    inserer c.titulaire dans l
  suivant
retourner l
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Virement de 200 € effectué
Virement de 1000 € annulé : solde insuffisant
Ada : 300 €
Bob : 300 €
Titulaires : ADA, BOB
```

8.5 Les zones : travailler enregistrement par enregistrement

Pour les habitués de dBase, Clipper ou wdGestion, Lexis+ propose une seconde façon de travailler : ouvrir une table dans une **zone** de travail, s'y **déplacer** d'enregistrement en enregistrement, et lire les champs **directement par leur nom**, comme des variables.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Syntaxe

```
utiliser table [alias a] [ordre "colonnes"] [filtre "condition SQL"]
parcourir [pour condition]
  instructions (les champs se lisent par leur nom)
finparcourir
fermer [alias | tout]
```

EXEMPLE Parcourir les clients de Lyon — principal.prg

```
1  ouvrirbase(":memoire:")
2  sql "create table clients (code varchar primary key, nom varchar, ville varchar, ca
   double)"
3  sql "insert into clients values ('C1','Durand','Lyon',12500),
   ('C2','Martin','Brest',8300), ('C3','Petit','Lyon',4500), ('C4','Roux','Nantes',15200)"
4
5  utiliser clients ordre "nom" alias cli          && ouvre la zone, triée par nom
6  ? "La zone", alias(), "contient", nbenr(), "enregistrements"
7  parcourir pour ville = "Lyon"                    && seulement les Lyonnais
8    ? "  n°" + numenr(), code, nom, ca             && champs lus par leur nom
9  finparcourir
10 fermer cli
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
La zone CLI contient 4 enregistrements
  n°1 C1 Durand 12500
  n°3 C3 Petit 4500
```

8.6 Se déplacer et chercher dans une zone

Instruction / fonction	Rôle (équivalent dBase)
`aller haut`, `aller bas`, `aller n`	premier, dernier, n-ième enregistrement (`GO TOP / BOTTOM / n`)
`sauter [n]`	avancer de n enregistrements, 1 par défaut ; n négatif : reculer (`SKIP`)
`chercher valeur`	cherche sur la première colonne de l'ordre (`SEEK`)
`localiser pour condition`	premier enregistrement qui vérifie la condition (`LOCATE`)
`continuer`	enregistrement suivant qui vérifie la même condition (`CONTINUE`)
`trouve()`	vrai si le dernier `chercher` / `localiser` / `continuer` a réussi (`FOUND()`)
`fdf()`, `ddf()`	fin, début de fichier dépassés (`EOF()`, `BOF()`)
`numenr()`, `nbenr()`	numéro de l'enregistrement courant, nombre d'enregistrements (`RECNO()`, `RECCOUNT()`)

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Recherches dans la zone — principal.prg

```
ouvrirbase(":memoire:")
sql "create table clients (code varchar, nom varchar, ville varchar, ca double)"
sql "insert into clients values ('C1','Durand','Lyon',12500),
('C2','Martin','Brest',8300), ('C3','Petit','Lyon',4500), ('C4','Roux','Nantes',15200)"
utiliser clients ordre "nom"

aller bas
? "Dernier (ordre alphabétique) :", nom, "— enregistrement", numenr(), "sur", nbenr()
aller haut
? "Premier :", nom
sauter 2
? "Deux plus loin :", nom

chercher "Petit"                          && recherche sur la colonne de l'ordre (nom)
? "Petit trouvé ?", trouve(), "— ville :", ville
chercher "Zorro"
? "Zorro trouvé ?", trouve()

localiser pour ville = "Lyon"
tantque trouve()
  ? "Lyonnais :", nom, ca
  continuer                               && le suivant qui vérifie la condition
fintantque

aller haut
tantque non fdf()                         && parcours « à l'ancienne »
  ?? nom + " "
  sauter
fintantque
?
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Dernier (ordre alphabétique) : Roux — enregistrement 4 sur 4
Premier : Durand
Deux plus loin : Petit
Petit trouvé ? vrai — ville : Lyon
Zorro trouvé ? faux
Lyonnais : Durand 12500
Lyonnais : Petit 4500
Durand Martin Petit Roux
```

8.7 Modifier les données dans une zone

Instruction	Rôle (équivalent dBase)
`ajouter avec champ par valeur, …`	ajoute un enregistrement (`APPEND` + `REPLACE`) ; `ajouter vide` ajoute un enregistrement vide
`remplacer champ par valeur, …`	modifie l'enregistrement courant (`REPLACE`)
`effacer`	supprime l'enregistrement courant (`DELETE`)

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Instruction	Rôle (équivalent dBase)
`compter [pour condition] dans variable`	compte les enregistrements (`COUNT`)
`rafraichir`	relit la zone (après des commandes `sql`, par exemple)

Toutes ces modifications sont **immédiatement** enregistrées dans la base.

EXEMPLE Mise à jour des tarifs — principal.prg

```
ouvrirbase(":memoire:")
sql "create table produits (ref varchar, libelle varchar, prix double, stock integer)"
sql "insert into produits values ('A1','Agenda',12.5,40), ('B2','Bloc',3.2,0),
('C3','Classeur',2.9,120)"
utiliser produits ordre "ref"

parcourir                                   && +5 % sur tous les prix
  remplacer prix par arrondi(prix * 1.05, 2)
finparcourir
chercher "B2"
si trouve()
  effacer                                   && produit abandonné
finsi
ajouter avec ref par "D4", libelle par "Dateur", prix par 8.9, stock par 15
compter pour stock > 20 dans bien_fournis

rafraichir
parcourir
  ? ref, gauche(libelle + espace(9), 9), chaine(prix, 6, 2), "€", stock, "en stock"
finparcourir
? bien_fournis, "produit(s) avec plus de 20 unités ;", nbenr(), "produits en tout"
```

▶ CE QU'AFFICHE LE PROGRAMME

```
A1 Agenda     13.13 € 40 en stock
C3 Classeur    3.05 € 120 en stock
D4 Dateur      8.90 € 15 en stock
2 produit(s) avec plus de 20 unités ; 3 produits en tout
```

8.8 Plusieurs zones en même temps

On peut ouvrir plusieurs tables, chacune dans sa zone, sous un **alias**. Une seule zone est **courante** (celle où agissent `aller`, `parcourir`, les noms de champs…) : `selectionner alias` en change. Pour lire un champ d'une autre zone, on écrit `alias->champ`. Et si une variable porte le même nom qu'un champ, `m->nom` désigne explicitement la **variable** (m comme « mémoire »).

Écriture	Rôle
`selectionner alias`	change de zone courante (`SELECT`)
`alias->champ`	champ d'une autre zone
`m->variable`	la variable, et non le champ du même nom

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Écriture	Rôle
`champ("nom", [alias])`	valeur d'un champ dont le nom est dans une variable (`FIELD()`)
`enregistrement([alias])`	l'enregistrement courant sous forme de fiche
`alias()`	alias de la zone courante
`fermer alias`, `fermer tout`	ferme une zone, toutes les zones

EXEMPLE Factures et clients : deux zones liées — principal.prg

```
ouvrirbase(":memoire:")
sql "create table clients (code varchar, nom varchar, remise double)"
sql "insert into clients values ('C1','Durand',5), ('C2','Martin',0), ('C3','Petit',10)"
sql "create table factures (num integer, client varchar, montant double)"
sql "insert into factures values (101,'C2',300), (102,'C1',1200), (103,'C3',450),
(104,'C1',80)"

utiliser clients ordre "code" alias cli
utiliser factures ordre "num" alias fac          && fac devient la zone courante
remise = "variable homonyme"
parcourir
  selectionner cli
  chercher fac->client                           && positionne cli sur le bon client
  selectionner fac
  locale net = montant * (1 - cli->remise / 100)
  ? "Facture", num, gauche(cli->nom + espace(7), 7), chaine(montant, 8, 2), ;
    "→ net", chaine(net, 8, 2), "(remise", cli->remise, "%)"
finparcourir
aller haut                                       && après parcourir, on est en fin de
fichier
? "Fiche courante :", json(enregistrement())
? "Champ par son nom :", champ("montant"), "/ m->remise :", m->remise
fermer tout
```

▶ CE QU'AFFICHE LE PROGRAMME

```
Facture 101 Martin    300.00 → net   300.00 (remise 0 %)
Facture 102 Durand   1200.00 → net  1140.00 (remise 5 %)
Facture 103 Petit     450.00 → net   405.00 (remise 10 %)
Facture 104 Durand     80.00 → net    76.00 (remise 5 %)
Fiche courante : {"NUM":101,"CLIENT":"C2","MONTANT":300}
Champ par son nom : 300 / m->remise : variable homonyme
```

SQL ou zones ?

Les deux approches travaillent sur **la même base** et se combinent librement. Le SQL est imbattable pour les traitements de masse (totaux, regroupements, mises à jour de milliers de lignes en une commande) ; les zones sont naturelles pour un traitement ligne à ligne ou pour reprendre du code dBase existant.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

8.9 SQLite

Le paquet `sqlite` ouvre des bases **SQLite** (le format de base le plus répandu au monde, utilisé par les téléphones, les navigateurs…). On obtient un **objet** base, dont les méthodes s'appellent avec `:`.

Appel	Rôle
`sqlite.ouvrir(chemin)`	ouvre ou crée une base (`":memoire:"` en mémoire) ; renvoie un objet `db`
`sqlite.version()`	version de la bibliothèque SQLite
`db:executer(sql, …)`	exécute une commande ; renvoie le nombre de lignes modifiées
`db:requete(sql, …)`	liste des lignes (fiches, colonnes en majuscules)
`db:premier(sql, …)`	première ligne ou `nul`
`db:lignes(sql, …)`	itérateur : `pour chaque r dans db:lignes(sql) … suivant`
`db:dernierid()`	identifiant de la dernière ligne insérée
`db:modifications()`	lignes modifiées par la dernière commande
`db:transaction(@PROC)`	appelle `PROC(db)` dans une transaction, annulée en cas d'erreur
`db:fermer()`	ferme la base

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Un carnet de contacts SQLite — principal.prg

```
db = sqlite.ouvrir(":memoire:")
db:executer("create table contacts (id integer primary key, nom text, tel text)")
db:executer("insert into contacts (nom, tel) values (?, ?)", "Durand", "02 98 00 00 01")
? "Identifiant attribué :", db:dernierid()

db:transaction(@IMPORTER)                    && tout ou rien
pour chaque c dans db:lignes("select * from contacts order by nom")
  ? c.id, c.nom, c.tel
suivant
? "Premier :", db:premier("select nom from contacts order by id").nom
essayer
  db:transaction(@IMPORT_RATE)
capturer e
  ? "Import annulé :", e, "— contacts :", db:premier("select count(*) as n from
contacts").n
finessayer
db:fermer()

procedure IMPORTER(d)
  d:executer("insert into contacts (nom, tel) values (?, ?)", "Martin", "02 98 00 00 02")
  d:executer("insert into contacts (nom, tel) values (?, ?)", "Ada", "06 00 00 00 03")
retourner

procedure IMPORT_RATE(d)
  d:executer("insert into contacts (nom, tel) values ('Fantôme', '')")
  erreur "fichier d'import corrompu"         && la ligne Fantôme est annulée
retourner
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Identifiant attribué : 1
3 Ada 06 00 00 00 03
1 Durand 02 98 00 00 01
2 Martin 02 98 00 00 02
Premier : Durand
Import annulé : fichier d'import corrompu — contacts : 3
```

8.10 Firebird

Le paquet `firebird` se connecte à un **serveur Firebird** (ou à un fichier local) avec les mêmes méthodes que SQLite, plus quelques possibilités propres aux bases d'entreprise : séquences, transactions explicites avec niveau d'isolation, description des tables. La bibliothèque cliente Firebird (`fbclient`) est chargée à l'exécution, seulement si le programme s'en sert.

Appel	Rôle
`firebird.connecter(base, utilisateur, mdp, [options])`	connexion : `"serveur:/chemin/base.fdb"`, `"serveur/3050:alias"` ou fichier ; ou une table `{base=, hote=, port=, utilisateur=, motdepasse=, role=, jeu="UTF8", isolation=, attente=}`
`firebird.creer(base, utilisateur, mdp, [{taillepage=, jeu=}])`	crée une base puis s'y connecte

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Appel	Rôle
`firebird.version()`, `firebird.disponible()`	version de la bibliothèque, présence du client
`db:executer`, `db:requete`, `db:premier`, `db:lignes`, `db:modifications`	comme SQLite (accepte `insert … returning` et `execute procedure`)
`db:valeur(sql, …)`	première colonne de la première ligne
`db:sequence(nom, [pas])`	valeur suivante d'une séquence (générateur)
`db:transaction(@PROC, [options])`	transaction validée, ou annulée en cas d'erreur
`db:debut([options])`, `db:valider()`, `db:annuler()`, `db:entransaction()`	transaction explicite
`db:tables()`, `db:colonnes(table)`, `db:serveur()`	description de la base et du serveur
`db:fermer()`	ferme (une transaction non validée est annulée)

EXEMPLE Facturation sur un serveur Firebird — principal.prg

```
db = firebird.connecter({hote = "srv-gestion", base = "/data/gestion.fdb",
                         utilisateur = "COMPTA", motdepasse = systeme.env("FB_MDP")})
locale n = db:sequence("GEN_FACTURE")                    && numéro unique
db:transaction(@CREER_FACTURE)
? "nouvelle facture n°", n
pour chaque c dans db:lignes("select code, nom, solde from client where solde > ? order by code", 0)
  ? " ", c.code, gauche(c.nom + espace(12), 12), c.solde   && NUMERIC -> nombre
suivant
? db:valeur("select count(*) from client where solde > 0"), "client(s) débiteurs"
? "tables :", joindre(db:tables(), ", ")
db:fermer()

procedure CREER_FACTURE(d)
  d:executer("insert into facture(num, client, date_f) values (?, ?, ?)", n, "C001", date())
  d:executer("insert into ligne(facture, article, qte, pu) values (?, ?, ?, ?)", n, "A1", 3, 12.5)
retourner
```

▶ CE QUE L'ON OBTIENT

```
nouvelle facture n° 2087
  C001 Durand        1250.8
  C017 Le Goff        980
2 client(s) débiteurs
tables : CLIENT, FACTURE, LIGNE
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

À retenir

- `ouvrirbase()` ; `sql "…" avec … dans liste` ; `requete()` ; `valeursql()` — toujours des paramètres `?`.
- Transactions : `begin transaction` / `commit` / `rollback` avec `essayer`.
- Zones : `utiliser … parcourir … finparcourir`, `aller`, `sauter`, `chercher` / `trouve()`, `localiser` / `continuer`, `ajouter`, `remplacer`, `effacer`, `compter`, `selectionner`, `alias->champ`, `fermer`.
- `sqlite.ouvrir()` et `firebird.connecter()` donnent un objet `db` : `executer`, `requete`, `premier`, `lignes`, `transaction`, `fermer`.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

CHAPITRE 9 · FICHIERS & FORMATS

Fichiers et formats de données

Lire un relevé bancaire en CSV, produire un export JSON pour un site web, archiver des journaux, décoder un fichier binaire… Les paquets es, fichier, csv, json et binaire couvrent tous les échanges de données avec le disque.



9.1 Les paquets d'entrées / sorties

9.2 Lire et écrire un fichier d'un seul coup

9.3 Lire au fil de l'eau avec es

9.4 Dossiers et gestion des fichiers

9.5 Le format CSV

9.6 Le format JSON

9.7 Les données binaires

Dans ce chapitre, vous apprendrez à…

- lire et écrire un fichier texte en une ligne
- gérer dossiers, copies, déplacements et suppressions
- lire et produire du JSON pour le web
- traiter un gros fichier ligne par ligne
- lire et produire du CSV pour les tableurs
- coder et décoder des données binaires

9.1 Les paquets d'entrées / sorties

Un **paquet** est un ensemble de fonctions regroupées sous un nom : `fichier.lire(...)`, `csv.analyser(...)`. Les paquets `es` (entrées / sorties) et `systeme` sont toujours disponibles ; les autres se chargent automatiquement à la première utilisation de leur nom (ou avec `p = paquet("nom")`).

Paquet	Pour…
`fichier`	lire / écrire un fichier d'un coup, gérer dossiers et chemins
`es`	lire / écrire au fil de l'eau (gros fichiers, console, flux ouverts)
`csv`	les tableaux au format CSV (Excel, LibreOffice, exports bancaires…)
`json`	le format d'échange du web et des API

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Paquet	Pour…
`binaire`	les fichiers et protocoles binaires (octets, entiers, flottants)

Chemin absolu, chemin relatif

Un **chemin** désigne un fichier : `factures/2026/janvier.csv`. Un chemin **relatif** (qui ne commence pas par `/` ou par une lettre de lecteur `C:\`) part du **dossier courant**, qui est en général le dossier du programme. Pour construire un chemin sans vous soucier des `/` ou `\`, utilisez `fichier.joindre("factures", "2026", "janvier.csv")`.

9.2 Lire et écrire un fichier d'un seul coup

Pour les fichiers de taille raisonnable (jusqu'à quelques dizaines de mégaoctets), le paquet `fichier` lit ou écrit **tout le contenu en une instruction**.

Appel	Rôle
`fichier.lire(chemin)`	tout le contenu (texte ou binaire)
`fichier.ecrire(chemin, donnees)`	remplace le contenu (crée le fichier s'il n'existe pas)
`fichier.ajouter(chemin, donnees)`	ajoute à la fin
`fichier.lignes(chemin)`	liste des lignes
`fichier.ecrirelignes(chemin, lignes)`	écrit une liste de lignes
`fichier.lireoctets(chemin)`, `fichier.ecrireoctets(chemin, d)`	variantes binaires

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Un journal d'activité — principal.prg

```
locale journal = "activite.log"
NOTER(journal, "démarrage du traitement")
NOTER(journal, "42 factures générées")
NOTER(journal, "envoi terminé")

? "Contenu du journal :"
? fichier.lire(journal)
locale lignes = fichier.lignes(journal)
? "Le journal compte", #lignes, "lignes ; la dernière :", lignes[#lignes]

&& Une liste de lignes devient un fichier, et inversement
fichier.ecrirelignes("courses.txt", {"pain", "lait", "café"})
? "courses.txt :", joindre(fichier.lignes("courses.txt"), " / ")

procedure NOTER(chemin, message)
  fichier.ajouter(chemin, "2026-10-16 09:00 " + message + car(10))   && car(10) = fin de ligne
retourner
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Contenu du journal :
2026-10-16 09:00 démarrage du traitement
2026-10-16 09:00 42 factures générées
2026-10-16 09:00 envoi terminé

Le journal compte 3 lignes ; la dernière : 2026-10-16 09:00 envoi terminé
courses.txt : pain / lait / café
```

9.3 Lire au fil de l'eau avec es

Pour un gros fichier, ou pour traiter les lignes **au fur et à mesure**, on utilise le paquet `es`. `es.lignes(chemin)` est un **itérateur** : la boucle `pour chaque` reçoit les lignes une à une, sans jamais charger tout le fichier en mémoire.

On peut aussi **ouvrir** un fichier pour obtenir un objet sur lequel lire et écrire à son rythme :

Appel	Rôle
`es.lignes([chemin])`	itérateur sur les lignes d'un fichier (ou de l'entrée standard)
`es.ouvrir(chemin, [mode])`	ouvre un fichier : `"l"` lecture (défaut), `"e"` écriture, `"a"` ajout, `"l+"`, `"e+"`
`f:lire([format])`	`"l"` une ligne, `"L"` une ligne avec sa fin, `"n"` un nombre, `"t"` tout, ou un nombre d'octets ; `nul` en fin de fichier
`f:ecrire(...)`	écrit des valeurs
`f:lignes()`	itérateur sur les lignes restantes
`f:position([origine], [decalage])`	se déplace (`"debut"`, `"courant"`, `"fin"`) et renvoie la position

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Appel	Rôle
`f:vider()`, `f:fermer()`	vide les tampons, ferme le fichier
`es.type(f)`	`"fichier"`, `"fichier fermé"` ou `nul`
`es.ecrire(...)`, `es.lire([format])`	écrire sur la console (sans espace ni fin de ligne) ; lire au clavier
`es.sortie`, `es.erreur`, `es.entree`	les flux standard de la console

EXEMPLE Analyser un journal de serveur ligne par ligne — principal.prg

```
&& On fabrique d'abord un fichier de démonstration
locale f = es.ouvrir("acces.log", "e")
pour i = 1 a 12
  f:ecrire("10.0.0." + (i % 4), " ", sisinon(i % 5 = 0, "500", "200"), " /page", i % 3,
car(10))
suivant
f:fermer()

&& Puis on le relit ligne par ligne
locale par_statut = {}, par_ip = {}, n = 0
pour chaque ligne dans es.lignes("acces.log")
  locale c = decouper(ligne, " ")
  par_statut[c[2]] = (par_statut[c[2]] ou 0) + 1
  par_ip[c[1]] = (par_ip[c[1]] ou 0) + 1
  n += 1
suivant
? n, "requêtes analysées"
pour chaque statut, nb dans par_statut
  ? "  statut", statut, ":", nb
suivant
? "  adresses distinctes :", nbelements(par_ip)

&& Lecture « à la main » : première ligne, puis le reste
locale g = es.ouvrir("acces.log")
? "Première ligne :", g:lire()
? "Taille du reste :", longueur(g:lire("t")), "caractères"
g:fermer()
es.ecrire("Écriture brute ", 1, 2, 3, car(10))     && sans espaces ajoutés
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
12 requêtes analysées
  statut 200 : 10
  statut 500 : 2
  adresses distinctes : 4
Première ligne : 10.0.0.1 200 /page1
Taille du reste : 220 caractères
Écriture brute 123
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

9.4 Dossiers et gestion des fichiers

Appel	Rôle
`fichier.existe(c)`, `fichier.estfichier(c)`, `fichier.estdossier(c)`	tests d'existence
`fichier.taille(c)`	taille en octets
`fichier.infos(c)`	`{taille, genre, modifie, cree}` ou `nul` (dates en secondes Unix)
`fichier.liste([dossier], [recursif])`	noms contenus (les dossiers finissent par `/`)
`fichier.creerdossier(c)`	crée un dossier et ses parents
`fichier.copier(src, dest)`, `fichier.deplacer(src, dest)`	copie (dossiers récursivement), déplace ou renomme
`fichier.supprimer(c)`	supprime un fichier ou un dossier (récursif !)
`fichier.joindre(a, b, …)`, `fichier.nom(c)`, `fichier.dossier(c)`, `fichier.extension(c)`, `fichier.absolu(c)`	manipulation de chemins
`fichier.courant()`, `fichier.changer(d)`	dossier courant
`fichier.dossiertemp()`, `fichier.dossierperso()`	dossier temporaire, dossier personnel

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Archiver les exports par année — principal.prg

```
&& Préparation : quelques fichiers d'export
fichier.creerdossier("exports")
pour chaque n dans {"ventes_2025-11.csv", "ventes_2025-12.csv", "ventes_2026-01.csv",
"notes.txt"}
  fichier.ecrire(fichier.joindre("exports", n), "données de " + n)
suivant

&& Rangement : exports/ventes_AAAA-MM.csv  →  archives/AAAA/ventes_AAAA-MM.csv
locale ranges = 0
pour chaque nom dans fichier.liste("exports")
  si fichier.extension(nom) <> ".csv"
    boucler
  finsi
  locale annee = souschaine(nom, 8, 4)
  locale dest = fichier.joindre("archives", annee)
  fichier.creerdossier(dest)
  fichier.deplacer(fichier.joindre("exports", nom), fichier.joindre(dest, nom))
  ranges += 1
suivant
? ranges, "fichier(s) archivé(s)"
pour chaque d dans fichier.liste("archives")
  ? " ", d, "→", joindre(fichier.liste(fichier.joindre("archives", d)), ", ")
suivant
? "Restent dans exports :", joindre(fichier.liste("exports"), ", ")
locale info = fichier.infos("archives/2026/ventes_2026-01.csv")
? "Taille du dernier :", info.taille, "octets (" + info.genre + ")"
fichier.supprimer("archives")                          && nettoyage
? "Archives supprimées :", non fichier.existe("archives")
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
3 fichier(s) archivé(s)
  2025/ → ventes_2025-11.csv, ventes_2025-12.csv
  2026/ → ventes_2026-01.csv
Restent dans exports : notes.txt
Taille du dernier : 30 octets (fichier)
Archives supprimées : vrai
```

Attention

`fichier.supprimer()` efface **définitivement**, sans corbeille, et vide les dossiers **récursivement**. Vérifiez deux fois le chemin, surtout s'il est calculé.

9.5 Le format CSV

Le **CSV** (*valeurs séparées par des virgules*) est le format de tableau le plus simple : une ligne de texte par ligne du tableau, les colonnes séparées par une virgule (ou un point-virgule dans les exports français). Excel, LibreOffice, les banques et les logiciels comptables savent tous l'ouvrir et le produire.

Appel	Rôle
`csv.analyser(texte, [options])`	liste des lignes d'un texte CSV

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Appel	Rôle
`csv.lire(chemin, [options])`	idem, depuis un fichier
`csv.encoder(lignes, [options])`	texte CSV
`csv.ecrire(chemin, lignes, [options])`	écrit un fichier CSV

Options de lecture : `sep` (séparateur, `","` par défaut), `entete` (`vrai` : la première ligne donne les noms de colonnes, et chaque ligne devient une **fiche** ; la liste des noms est dans `lignes.entete`), `convertir` (`vrai` : les nombres sont convertis). Options d'écriture : `sep`, `entete` (liste des colonnes, ou `vrai` pour les déduire).

FICHIER Le relevé bancaire exporté par la banque releve.csv

```
date;libelle;montant
2026-10-01;SALAIRE OCTOBRE;2450.00
2026-10-02;LOYER;-780.00
2026-10-05;SUPERMARCHE;-96.40
2026-10-09;ELECTRICITE;-64.12
2026-10-12;SUPERMARCHE;-58.75
2026-10-15;REMBOURSEMENT SECU;23.10
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Analyser le relevé et produire un bilan CSV — principal.prg

```
locale lignes = csv.lire("releve.csv", {sep = ";", entete = vrai, convertir = vrai})
? #lignes, "opérations ; colonnes :", joindre(lignes.entete, ", ")

locale credits = 0, debits = 0, par_libelle = {}
pour chaque op dans lignes
  si op.montant > 0
    credits += op.montant
  sinon
    debits += op.montant
  finsi
  par_libelle[op.libelle] = (par_libelle[op.libelle] ou 0) + op.montant
suivant
? "Crédits :", chaine(credits, 1, 2), " Débits :", chaine(debits, 1, 2), ;
  " Solde :", chaine(credits + debits, 1, 2)

&& Un bilan par libellé, au format CSV
locale bilan = {}
pour chaque libelle, total dans par_libelle
  inserer {libelle = libelle, total = arrondi(total, 2)} dans bilan
suivant
csv.ecrire("bilan.csv", bilan, {sep = ";", entete = {"libelle", "total"}})
? fichier.lire("bilan.csv")
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
6 opérations ; colonnes : date, libelle, montant
Crédits : 2473.10  Débits : -999.27  Solde : 1473.83
libelle;total
SALAIRE OCTOBRE;2450
LOYER;-780
SUPERMARCHE;-155.15
ELECTRICITE;-64.12
REMBOURSEMENT SECU;23.1
```

9.6 Le format JSON

Le **JSON** est le format d'échange des services web, des API et de nombreux fichiers de configuration. Il ressemble beaucoup aux tables Lexis+ : listes `[1, 2]` et objets `{"nom": "Ada"}`.

Appel	Rôle
`json.encoder(valeur, [joli])`	texte JSON (`joli = vrai` : indenté, lisible)
`json.decoder(texte, [exact])`	valeur Lexis+ ; clefs en majuscules, sauf `exact = vrai`
`json.lire(chemin, [exact])`, `json.ecrire(chemin, valeur, [joli])`	depuis / vers un fichier (indenté par défaut)
`json.nul`	représente `null` dans les données à encoder
`json(v)`, `dejson(s)`	fonctions intégrées équivalentes, toujours disponibles

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Fichier de configuration et export pour un site web — principal.prg

```
&& Une configuration enregistrée puis relue
locale config = {societe = "Papeterie Durand", tva = 20, devises = {"EUR", "CHF"},
                 smtp = {serveur = "smtp.exemple.fr", port = 587}}
json.ecrire("config.json", config)
locale relue = json.lire("config.json")
? "Société :", relue.societe, "— port SMTP :", relue.smtp.port, "— devises :",
joindre(relue.devises, "/")

&& Un catalogue exporté pour un site web (null explicite ; les noms sortent en majuscules)
locale catalogue = {{ref = "A1", prix = 12.5, promo = json.nul},
                    {ref = "B2", prix = 3.2, promo = 2.9}}
? json.encoder(catalogue)

&& Données reçues d'une API, en gardant la casse exacte des clefs
locale recu = json.decoder('{"orderId": 1042, "items": [{"sku": "A1", "qty": 3}]}', vrai)
? "Commande", recu["orderId"], ":", recu["items"][1]["qty"], "x", recu["items"][1]["sku"]
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Société : Papeterie Durand — port SMTP : 587 — devises : EUR/CHF
[{"REF":"A1","PRIX":12.5,"PROMO":null},{"REF":"B2","PRIX":3.2,"PROMO":2.9}]
Commande 1042 : 3 x A1
```

Astuce

Sans `exact`, `json.decoder` met les clefs en majuscules : `recu.orderid` fonctionne (casse indifférente). Avec `exact = vrai`, il faut écrire la clef exacte entre crochets : `recu["orderId"]`. Choisissez `exact` quand vous devez renvoyer les données telles quelles à un autre programme.

9.7 Les données binaires

Certains fichiers (images, formats propriétaires) et protocoles réseau ne contiennent pas du texte mais des **octets**. Le paquet `binaire` code et décode des valeurs selon un **format** : une suite de lettres qui décrit chaque champ.

Code de format	Signification
`<` / `>`	ordre des octets : petit-boutiste (PC) / gros-boutiste (réseau)
`b` `B`	entier 8 bits signé / non signé
`h` `H`	entier 16 bits
`i` `I` (`iN`)	entier 32 bits (ou N octets)
`l` `L` `j` `J`	entier 64 bits
`f` `d`	flottant 32 / 64 bits
`z`	chaîne terminée par un octet nul
`s`	chaîne précédée de sa longueur

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Code de format	Signification
`cN`	chaîne de N octets exactement
`x`	un octet nul (bourrage)

Appel	Rôle
`binaire.coder(format, valeurs...)`	octets
`binaire.decoder(format, donnees, [position])`	liste des valeurs, avec le champ `suite` (position suivante)
`binaire.taille(format)`	taille d'un format fixe
`binaire.hex(donnees)`, `binaire.dehex(hex)`	affichage hexadécimal et inverse

EXEMPLE Un enregistrement binaire de capteur — principal.prg

```
&& Format : identifiant 16 bits, horodatage 32 bits, température flottante, nom terminé par 0
locale format = "<HIfz"
locale paquet = binaire.coder(format, 513, 1792152000, 21.5, "salon")
? "Octets :", binaire.hex(paquet), "(" + NB_OCTETS(paquet) + " octets)"
? "Partie fixe :", binaire.taille("<HIf"), "octets"

locale v = binaire.decoder(format, paquet)
? "Capteur", v[1], "— horodatage", v[2], "— température", v[3], "°C — pièce", v[4]
? "Position après lecture :", v.suite

&& Plusieurs enregistrements à la suite dans un même fichier
fichier.ecrireoctets("mesures.bin", paquet + binaire.coder(format, 514, 1792152060, 19.25, "cuisine"))
locale donnees = fichier.lireoctets("mesures.bin")
locale pos = 1
tantque pos <= NB_OCTETS(donnees)
  locale m = binaire.decoder(format, donnees, pos)
  ? "  capteur", m[1], m[4], m[3], "°C"
  pos = m.suite
fintantque

fonction NB_OCTETS(d)                  && longueur() compte des caractères, pas des octets
retourner longueur(binaire.hex(d)) / 2
```

▶ CE QU'AFFICHE LE PROGRAMME

```
Octets : 0102c011d26a0000ac4173616c6f6e00 (16 octets)
Partie fixe : 10 octets
Capteur 513 — horodatage 1792152000 — température 21.5 °C — pièce salon
Position après lecture : 17
  capteur 513 salon 21.5 °C
  capteur 514 cuisine 19.25 °C
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Caractères et octets

`longueur()` et `#` comptent des **caractères** : un « é » compte pour 1, alors qu'il occupe 2 octets. Sur des données binaires, comptez les octets avec `longueur(binaire.hex(d)) / 2` (ou `fichier.taille()` pour un fichier).

À retenir

- `fichier.lire / ecrire / ajouter / lignes` pour les fichiers d'un bloc ; `es.lignes(chemin)` pour les gros fichiers.
- `fichier.joindre` pour construire les chemins ; `liste`, `copier`, `deplacer`, `supprimer`, `infos`, `creerdossier`.
- `csv.lire(chemin, {sep = ";", entete = vrai, convertir = vrai})` ; `csv.ecrire`.
- `json.lire / ecrire / encoder / decoder` ; `exact = vrai` pour garder la casse.
- `binaire.coder / decoder` avec un format (`<HIfz` …).

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

CHAPITRE 10 · SYSTÈME

Dialoguer avec le système

Dates mises en forme, variables d'environnement, commandes du système d'exploitation, pauses, arguments et code de retour — le paquet systeme, toujours disponible, relie vos programmes à la machine qui les exécute.



10.1 Le temps en secondes
10.2 Mettre une date en forme
10.3 Mesurer une durée et faire une pause
10.4 Environnement et commandes du système
10.5 Arguments et code de retour : un vrai outil en ligne de commande

Dans ce chapitre, vous apprendrez à…

- ✔ obtenir et mettre en forme dates et heures
- ✔ lire et définir des variables d'environnement
- ✔ mesurer un temps d'exécution, faire une pause
- ✔ calculer des durées et des écarts entre dates
- ✔ exécuter une commande et récupérer sa sortie
- ✔ terminer un programme avec un code de retour

10.1 Le temps en secondes

Pour l'ordinateur, un instant est un **nombre de secondes** écoulées depuis le 1er janvier 1970 à minuit (temps « Unix »). Ce format est idéal pour calculer : la différence de deux instants est une durée en secondes.

Appel	Rôle
`systeme.temps()`	l'instant présent, en secondes
`systeme.temps({annee=, mois=, jour=, heure=, minute=, seconde=})`	l'instant d'une date (heure par défaut : 12 h)
`systeme.ecart(t2, t1)`	t2 − t1 en secondes
`systeme.horloge()`	secondes écoulées depuis le lancement (horloge monotone, pour mesurer des durées)

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

10.2 Mettre une date en forme

`systeme.date([format], [instant])` écrit une date en texte selon un **format** où chaque code `%x` est remplacé par un élément de la date. Sans instant, c'est maintenant ; sans format, c'est le format complet (`%c`).

Code	Donne	Code	Donne
`%Y`	année (2026)	`%y`	année sur 2 chiffres (26)
`%m`	mois (10)	`%B` / `%b`	nom du mois (octobre / oct)
`%d`	jour (16)	`%A` / `%a`	jour de la semaine (vendredi / ven)
`%H` `%M` `%S`	heure, minute, seconde	`%j`	jour de l'année (289)
`%F`	`%Y-%m-%d`	`%T` / `%X`	`%H:%M:%S`
`%x`	`%d/%m/%Y`	`%w`	jour de la semaine (0 = dimanche)
`%c`	date et heure complètes	`%Z` / `%%`	fuseau / le caractère %

Deux formats spéciaux : `"*t"` renvoie une **fiche** `{annee, mois, jour, heure, minute, seconde, joursemaine, jourannee, ete}` ; un `!` en tête du format donne l'heure **UTC** (temps universel) au lieu de l'heure locale.

EXEMPLE Dates en toutes lettres et compte à rebours — principal.prg

```
? "Aujourd'hui :", systeme.date("%A %d %B %Y")
? "Horodatage de fichier :", systeme.date("%Y%m%d_%H%M%S") + ".csv"
? "Jour de l'année :", systeme.date("%j"), "— semaine commencée le", systeme.date("%a")

locale noel = systeme.temps({annee = 2026, mois = 12, jour = 25})
? "Noël tombe un", systeme.date("%A", noel)
locale jours = arrondi(systeme.ecart(noel, systeme.temps()) / 86400)
? "Noël est dans", jours, "jours"

locale t = systeme.date("*t")
? "Mois", t.mois, "— trimestre", plafond(t.mois / 3), "— heure d'été :", t.ete
? "Échéance à 30 jours :", systeme.date("%x", systeme.temps() + 30 * 86400)
? "Époque Unix (UTC) :", systeme.date("!%F %T", 0)
```

CE QU'AFFICHE LE PROGRAMME

```
Aujourd'hui : vendredi 16 octobre 2026
Horodatage de fichier : 20261016_205824.csv
Jour de l'année : 289 — semaine commencée le ven
Noël tombe un vendredi
Noël est dans 70 jours
Mois 10 — trimestre 4 — heure d'été : vrai
Échéance à 30 jours : 15/11/2026
Époque Unix (UTC) : 1970-01-01 00:00:00
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Astuce

Une journée compte 86 400 secondes, mais pas toujours : le jour du passage à l'heure d'hiver en compte 90 000. Arrondissez les nombres de jours (`arrondi(ecart / 86400)`) ou fixez l'heure à midi, comme le fait `systeme.temps({…})` par défaut.

10.3 Mesurer une durée et faire une pause

`systeme.horloge()` mesure le temps écoulé de façon fiable (elle n'est pas affectée par un changement d'heure de l'ordinateur). `systeme.attendre(secondes)` fait une pause ; pendant ce temps, les fenêtres du programme restent réactives et le bouton **Arrêter** fonctionne.

EXEMPLE Chronométrer deux méthodes de calcul — principal.prg

```
locale debut = systeme.horloge()
locale s = 0
pour i = 1 a 300000
  s += i
suivant
locale duree_boucle = systeme.horloge() - debut

debut = systeme.horloge()
locale s2 = 300000 * 300001 / 2              && la formule de Gauss
locale duree_formule = systeme.horloge() - debut

? "Boucle  :", s, "en", arrondi(duree_boucle * 1000, 1), "ms"
? "Formule :", s2, "en", arrondi(duree_formule * 1000, 3), "ms — plus rapide :",
duree_formule < duree_boucle
? "Pause d'un quart de seconde…"
debut = systeme.horloge()
systeme.attendre(0.25)
? "… attendu", arrondi(systeme.horloge() - debut, 2), "s"
```

▶ CE QU'AFFICHE LE PROGRAMME

```
Boucle  : 45000150000 en 40.3 ms
Formule : 45000150000 en 0.001 ms — plus rapide : vrai
Pause d'un quart de seconde…
… attendu 0.25 s
```

10.4 Environnement et commandes du système

Les **variables d'environnement** sont des réglages transmis par le système à chaque programme : dossier personnel (`HOME` ou `USERPROFILE`), chemins de recherche, ou vos propres réglages (mot de passe de base de données, mode test…), pratiques pour ne pas écrire de secret dans le code.

Appel	Rôle
`systeme.env(nom)`	valeur d'une variable d'environnement, `nul` si absente
`systeme.fixerenv(nom, valeur)`	définit une variable (pour ce programme et ceux qu'il lance)

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Appel	Rôle
`systeme.plateforme()`	`"linux"`, `"macosx"`, `"windows"`…
`systeme.executer(commande)`	exécute une commande du système ; renvoie son **code de retour** (0 = succès)
`systeme.capturer(commande)`	exécute et renvoie `{sortie, code}` : le texte affiché par la commande
`systeme.nomtemporaire()`	un nom de fichier temporaire unique
`systeme.renommer(a, b)`, `systeme.supprimer(c)`	renommer, supprimer un fichier ou un dossier
`systeme.dossierprogramme()`	dossier du programme principal
`systeme.args`	arguments du programme (`systeme.args[0]` : son chemin)
`systeme.quitter([code])`	termine le programme avec un code de retour

EXEMPLE Rapport sur la machine et commande externe — principal.prg

```
? "Système :", systeme.plateforme()
locale mode = systeme.env("APP_MODE") ou "production"
? "Mode :", mode
systeme.fixerenv("APP_MODE", "test")
? "Mode après fixerenv :", systeme.env("APP_MODE")

&& Une commande qui réussit et une qui échoue
locale r = systeme.capturer("echo Bonjour depuis le système")
? "Sortie :", rogner(r.sortie), "— code :", r.code
locale code = systeme.executer("ls /dossier/inexistant 2>/dev/null")
? "Code de retour d'une commande en échec :", code

&& Un fichier temporaire, renommé puis supprimé
locale tmp = systeme.nomtemporaire()
fichier.ecrire(tmp, "brouillon")
systeme.renommer(tmp, tmp + ".bak")
? "Renommé :", fichier.existe(tmp + ".bak")
systeme.supprimer(tmp + ".bak")
? "Supprimé :", non fichier.existe(tmp + ".bak")
```

▶ CE QU'AFFICHE LE PROGRAMME

```
Système : linux
Mode : production
Mode après fixerenv : test
Sortie : Bonjour depuis le système — code : 0
Code de retour d'une commande en échec : 2
Renommé : vrai
Supprimé : vrai
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

> **Attention**
>
> Les commandes du système dépendent de la plateforme : `ls` sous Linux et macOS, `dir` sous Windows. Testez `systeme.plateforme()` si votre programme doit fonctionner partout, et ne passez jamais à `executer` un texte saisi par un utilisateur sans le contrôler.

10.5 Arguments et code de retour : un vrai outil en ligne de commande

Un programme lancé par `wdlexisrt` (depuis un terminal, une tâche planifiée, un autre programme) reçoit ses **arguments** dans `systeme.args` et signale sa réussite ou son échec par son **code de retour** (`systeme.quitter(code)` ; 0 = succès).

EXEMPLE Outil de purge des vieux fichiers — principal.prg

```
&& Usage : wdlexisrt principal.prg dossier jours [--simulation]
si #systeme.args < 2
  ? "Usage : purge dossier jours [--simulation]"
  systeme.quitter(2)
finsi
locale dossier = systeme.args[1]
locale jours = nombre(systeme.args[2])
locale simulation = systeme.args[3] = "--simulation"

&& Préparation de la démonstration : trois journaux d'âges différents
fichier.creerdossier(dossier)
pour chaque n dans {"a.log", "b.log", "c.log"}
  fichier.ecrire(fichier.joindre(dossier, n), "…")
suivant

locale limite = systeme.temps() - jours * 86400
locale n = 0
pour chaque nom dans fichier.liste(dossier)
  locale chemin = fichier.joindre(dossier, nom)
  locale age = (systeme.temps() - fichier.infos(chemin).modifie) / 86400
  ? sisinon(simulation, "[simulation]", "[suppression]"), nom, "— âge :", arrondi(age),
"jour(s)"
  si fichier.infos(chemin).modifie < limite et non simulation
    fichier.supprimer(chemin)
    n += 1
  finsi
suivant
? n, "fichier(s) supprimé(s) — fin à", systeme.date("%T")
systeme.quitter(0)
```

CE QU'AFFICHE LE PROGRAMME

```
[simulation] a.log — âge : 0 jour(s)
[simulation] b.log — âge : 0 jour(s)
[simulation] c.log — âge : 0 jour(s)
0 fichier(s) supprimé(s) — fin à 20:58:24
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

À retenir

- `systeme.temps()`, `systeme.temps({…})`, `systeme.ecart()`, `systeme.date(format, t)` ; `"*t"` pour une fiche ; `!` pour l'UTC.
- `systeme.horloge()` pour chronométrer ; `systeme.attendre(s)` pour patienter.
- `systeme.env()`, `systeme.fixerenv()`, `systeme.executer()`, `systeme.capturer()`.
- `systeme.args`, `systeme.quitter(code)`, `systeme.dossierprogramme()`, `systeme.plateforme()`.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

CHAPITRE 11 · RÉSEAU

Communiquer sur le réseau

Interroger une API web, télécharger un fichier, déposer un export sur un serveur FTP, piloter un serveur distant en SSH ou écrire votre propre serveur : les paquets réseau ouvrent vos programmes sur le monde.



11.1 Quelques notions de réseau

11.2 Le web avec le paquet http

11.3 TCP : clients et serveurs maison

11.4 UDP : des messages sans connexion

11.5 FTP

11.6 SSH et SFTP

Dans ce chapitre, vous apprendrez à...

- ✔ comprendre adresses, ports, clients et serveurs
- ✔ télécharger un fichier
- ✔ transférer des fichiers en FTP et SFTP
- ✔ envoyer des requêtes HTTP et HTTPS, lire des API JSON
- ✔ écrire un client et un serveur TCP, échanger en UDP
- ✔ exécuter des commandes à distance en SSH

11.1 Quelques notions de réseau

Deux programmes qui communiquent jouent deux rôles : le **serveur** attend qu'on le contacte, le **client** le contacte. Pour joindre un serveur, il faut son **adresse** (un nom comme `www.exemple.fr`, traduit en adresse IP comme `93.184.215.14`, ou `127.0.0.1` pour la machine elle-même) et un **port**, un numéro qui désigne le service : 80 pour le web, 443 pour le web sécurisé, 21 pour FTP, 22 pour SSH...

Paquet	Pour...
`http`	le web : pages, API REST, téléchargements (HTTP et HTTPS)
`reseau`	les connexions TCP et UDP « brutes » : protocoles maison, serveurs
`ftp`	déposer et récupérer des fichiers sur un serveur FTP

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Paquet	Pour…
`ssh`, `sftp`	commandes et transferts sécurisés (client OpenSSH installé)

Les exemples de ce chapitre

Pour que chaque exemple fonctionne partout, même sans connexion Internet, plusieurs d'entre eux démarrent eux-mêmes un petit **serveur local** dans un **fil d'exécution** (`fil.demarrer`, chapitre 12) puis le contactent à l'adresse `127.0.0.1`. Dans vos programmes, remplacez simplement cette adresse par celle du vrai serveur.

11.2 Le web avec le paquet http

Appel	Rôle
`http.lire(url, [entetes])`	requête GET ; renvoie le **corps** de la réponse
`http.envoyer(url, corps, [entetes])`	requête POST ; renvoie le corps de la réponse
`http.requete{methode=, url=, entetes=, corps=, delai=}`	requête quelconque (GET, POST, PUT, DELETE…) ; renvoie `{statut, raison, corps, entetes}`
`http.telecharger(url, chemin)`	enregistre la réponse dans un fichier ; renvoie le statut
`http.encoder(texte ou table)`	encode pour une URL (`{a = 1, b = 2}` → `"a=1&b=2"`)
`http.decoder(texte)`	décode un texte d'URL

Les adresses `https://` fonctionnent de la même façon (TLS, avec OpenSSL). Les **en-têtes** de la réponse sont rangés en minuscules : `r.entetes["content-type"]`. Le **statut** dit si tout s'est bien passé : 200 (OK), 201 (créé), 404 (introuvable), 500 (erreur du serveur)…

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Interroger une API JSON (serveur de démonstration local) — principal.prg

```
&& 1. Un mini serveur web de démonstration, dans un fil d'exécution
locale serveur = fil.demarrer([[
  locale s = reseau.ecouter(8810, "127.0.0.1")
  fil.envoyer("pret")
  pour i = 1 a 2
    locale c = s:accepter()
    locale requete = c:recevoir()                    && « GET /api/meteo?ville=Brest HTTP/1.1 »
    locale l = requete
    tantque l <> nul et rogner(l) <> ""               && on ignore les en-têtes
      l = c:recevoir()
    fintantque
    locale ville = decouper(decouper(requete, "=")[2], " ")[1]
    locale corps = json({ville = http.decoder(ville), temperature = 14 + i, ciel = "nuageux"})
    c:envoyer("HTTP/1.1 200 OK\r\nContent-Type: application/json\r\n" +
              "Content-Length: " + longueur(corps) + "\r\nConnection: close\r\n\r\n" + corps)
    c:fermer()
  suivant
]])
serveur:recevoir()                                    && attend que le serveur soit prêt

&& 2. Le client : ce que vous écrirez avec une vraie API
locale url = "http://127.0.0.1:8810/api/meteo?ville=" + http.encoder("Brest")
locale reponse = http.requete{url = url, entetes = {Accept = "application/json"}, delai = 5}
? "Statut :", reponse.statut, "— type :", reponse.entetes["content-type"]
locale meteo = dejson(reponse.corps)
? "Météo à", meteo.ville, ":", meteo.temperature, "°C,", meteo.ciel

locale corps = http.lire("http://127.0.0.1:8810/api/meteo?ville=" + http.encoder("Saint-Malo"))
? "Réponse brute :", corps
serveur:joindre()
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Statut : 200 — type : application/json
Météo à Brest : 15 °C, nuageux
Réponse brute : {"VILLE":"Saint-Malo","TEMPERATURE":16,"CIEL":"nuageux"}
```

Envoyer des données : POST, PUT, DELETE

Pour créer ou modifier une ressource sur une API, on envoie un corps (souvent du JSON) avec la méthode adaptée :

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE **Créer une commande sur une API REST** principal.prg

```
locale jeton = systeme.env("API_JETON")              && le secret n'est pas dans le code
locale entetes = {["Content-Type"] = "application/json", Authorization = "Bearer " + jeton}
locale commande = {client = "C001", lignes = {{ref = "A1", qte = 3}}}

locale r = http.requete{methode = "POST", url = "https://api.exemple.fr/commandes",
                        entetes = entetes, corps = json(commande), delai = 10}
? "Statut :", r.statut
si r.statut <> 201
  erreur "création refusée : " + r.corps
finsi
locale cree = dejson(r.corps)
? "Commande créée : n°", cree.id

r = http.requete{methode = "DELETE", url = "https://api.exemple.fr/commandes/" + cree.id, entetes = entetes}
? "Supprimée :", r.statut
http.telecharger("https://api.exemple.fr/factures/" + cree.id + ".pdf", "facture.pdf")
```

▶ **CE QUE L'ON OBTIENT**

```
Statut : 201
Commande créée : n° 5512
Supprimée : 204
```

11.3 TCP : clients et serveurs maison

Le paquet `reseau` ouvre des **sockets** : des tuyaux de communication entre deux programmes. En **TCP**, la connexion est établie une fois, puis les deux côtés s'échangent des octets de façon fiable et ordonnée.

Appel	Rôle
`reseau.connecter(hote, port, [{tls=, delai=}])`	client : se connecte à un serveur (TLS possible)
`reseau.ecouter(port, [hote])`	serveur : socket d'écoute
`s:accepter()`	serveur : attend un client et renvoie son socket
`s:envoyer(donnees)`	envoie ; renvoie le nombre d'octets envoyés
`s:recevoir([mode])`	reçoit : `"l"` une ligne (défaut), `"t"` tout jusqu'à la fermeture, ou n octets ; `nul` si fermé
`s:delai(secondes)`	délai maximal des réceptions
`s:distant()`	`{adresse, port}` de l'autre côté
`s:fermer()`	ferme
`reseau.resoudre(nom)`, `reseau.nomhote()`	adresse IP d'un nom ; nom de cette machine

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Un service de calcul en réseau (protocole texte maison) — principal.prg

```
&& Le serveur : reçoit « opération a b », répond le résultat
locale serveur = fil.demarrer([[
  locale ecoute = reseau.ecouter(8820, "127.0.0.1")
  fil.envoyer("pret")
  locale client = ecoute:accepter()
  tantque vrai
    locale ligne = client:recevoir()
    si ligne = nul ou ligne = "FIN"
      sortir
    finsi
    locale m = decouper(ligne, " ")
    locale a = nombre(m[2]), b = nombre(m[3])
    faire cas
    cas m[1] = "ADD"
      client:envoyer("OK " + (a + b) + "\n")
    cas m[1] = "MUL"
      client:envoyer("OK " + (a * b) + "\n")
    autrement
      client:envoyer("ERR commande inconnue\n")
    fincas
  fintantque
  client:fermer()
  ecoute:fermer()
]])
? "Serveur :", serveur:recevoir()

&& Le client
locale c = reseau.connecter("127.0.0.1", 8820, {delai = 5})
? "Connecté à", json(c:distant())
pour chaque question dans {"ADD 2 3", "MUL 6 7", "DIV 1 0"}
  c:envoyer(question + "\n")
  ? question, "→", c:recevoir()
suivant
c:envoyer("FIN\n")
c:fermer()
serveur:joindre()
? "Nom de cette machine :", type(reseau.nomhote()), "— localhost =",
reseau.resoudre("localhost")
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Serveur : pret
Connecté à {"ADRESSE":"127.0.0.1","PORT":8820}
ADD 2 3 → OK 5
MUL 6 7 → OK 42
DIV 1 0 → ERR commande inconnue
Nom de cette machine : chaine — localhost = 127.0.0.1
```

11.4 UDP : des messages sans connexion

En **UDP**, il n'y a pas de connexion : on envoie des **messages** isolés (datagrammes) à une adresse et un port, sans garantie de livraison. C'est le protocole des découvertes réseau, des journaux système (syslog), des capteurs…

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Appel	Rôle
`reseau.udp([port], [hote])`	socket UDP, éventuellement lié à un port pour recevoir
`s:envoyervers(donnees, port, hote)`	envoie un message
`s:recevoirde([taille])`	reçoit un message : `{donnees, adresse, port}`

EXEMPLE Un capteur qui envoie ses mesures en UDP — principal.prg

```
locale recepteur = reseau.udp(8830, "127.0.0.1")       && le « tableau de bord »
recepteur:delai(2)
locale capteur = reseau.udp()                          && le capteur n'écoute pas
pour chaque t dans {19.5, 19.8, 20.1}
  capteur:envoyervers(json({piece = "salon", temperature = t}), 8830, "127.0.0.1")
suivant
pour i = 1 a 3
  locale m = recepteur:recevoirde()
  locale d = dejson(m.donnees)
  ? "Reçu de", m.adresse + ":", d.piece, d.temperature, "°C"
suivant
capteur:fermer()
recepteur:fermer()
```

▶ CE QU'AFFICHE LE PROGRAMME

```
Reçu de 127.0.0.1: salon 19.5 °C
Reçu de 127.0.0.1: salon 19.8 °C
Reçu de 127.0.0.1: salon 20.1 °C
```

11.5 FTP

Le paquet `ftp` se connecte à un serveur FTP (en mode passif) pour lister, déposer et récupérer des fichiers.

Appel	Rôle
`ftp.connecter{hote=, port=21, utilisateur="anonymous", motdepasse=""}`	ouvre une session
`c:liste([chemin], [detail])`	noms des fichiers (`detail = vrai` : format détaillé)
`c:lire(distant, [local])`	télécharge : renvoie les données, ou les écrit dans le fichier local
`c:envoyer(local_ou_donnees, [distant], [sontdesdonnees])`	dépose un fichier local, ou des données
`c:dossier(chemin)`, `c:courant()`	change de dossier distant, dossier courant
`c:creerdossier(c)`, `c:supprimerdossier(c)`, `c:supprimer(c)`, `c:renommer(a, b)`	gestion distante
`c:fermer()`	ferme la session

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE **Déposer l'export du jour chez un partenaire** principal.prg

```
locale c = ftp.connecter{hote = "ftp.partenaire.fr", utilisateur = "papeterie",
                         motdepasse = systeme.env("FTP_MDP")}
c:dossier("/depot")
? "Dossier distant :", c:courant()
locale nom = "ventes_" + systeme.date("%Y%m%d") + ".csv"
fichier.ecrire(nom, "ref;qte\nA1;12\nB2;4\n")
c:envoyer(nom, nom)                                  && fichier local → distant
? "Déposé :", nom
? "Sur le serveur :", joindre(c:liste(), ", ")
? "Accusé reçu :", c:lire("accuse.txt")               && contenu récupéré en mémoire
c:fermer()
```

▶ **CE QUE L'ON OBTIENT**

```
Dossier distant : /depot
Déposé : ventes_20261016.csv
Sur le serveur : ventes_20261015.csv, ventes_20261016.csv
Accusé reçu : OK 2026-10-16
```

11.6 SSH et SFTP

Le paquet `ssh` pilote un serveur distant de façon sécurisée, en s'appuyant sur le client **OpenSSH** installé sur la machine. L'authentification se fait de préférence par **clef** (le mot de passe demande l'outil `sshpass`). `sftp.connecter` ouvre la même session, pour les transferts.

Appel	Rôle
`ssh.connecter{hote=, utilisateur=, port=22, clef=, motdepasse=, options={}}`	ouvre une session (vérifie la connexion)
`ssh.executer{hote=, utilisateur=, commande=, …}`	exécute une commande sans garder de session : `{sortie, code}`
`session:executer(commande)`	exécute une commande distante : `{sortie, code}`
`session:lire(distant, [local])`, `session:envoyer(local, [distant])`	transferts (scp), fichiers ou dossiers
`session:liste([c])`, `session:creerdossier(c)`, `session:supprimerdossier(c)`, `session:supprimer(c)`, `session:renommer(a, b)`	gestion distante (sftp)
`sftp.connecter{…}`	même session, orientée transferts

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Sauvegarde nocturne d'un serveur — principal.prg

```
locale s = ssh.connecter{hote = "srv-gestion", utilisateur = "admin", clef =
"~/.ssh/id_ed25519"}
locale libre = s:executer("df -h /data | tail -1 | awk '{print $4}'")
? "Espace libre :", rogner(libre.sortie)

locale archive = "sauvegarde_" + systeme.date("%Y%m%d") + ".tar.gz"
locale r = s:executer("tar czf /tmp/" + archive + " /data/gestion")
? "Sauvegarde créée (code", r.code + ")"
si r.code <> 0
  erreur "échec de la sauvegarde : " + r.sortie
finsi
s:lire("/tmp/" + archive, fichier.joindre("sauvegardes", archive))
? "Rapatriée :", archive
s:supprimer("/tmp/" + archive)
? "Sauvegardes distantes :", #s:liste("/data/sauvegardes")
```

▶ **CE QUE L'ON OBTIENT**

```
Espace libre : 41G
Sauvegarde créée (code 0)
Rapatriée : sauvegarde_20261016.tar.gz
Sauvegardes distantes : 7
```

Ne jamais écrire un mot de passe dans le code

Lisez les secrets dans des variables d'environnement (`systeme.env("FTP_MDP")`), dans un fichier de configuration protégé, ou utilisez une clef SSH. Un programme se copie, s'envoie, se publie… ses mots de passe avec lui.

À retenir

- `http.lire`, `http.envoyer`, `http.requete{…}` (statut, corps, en-têtes en minuscules), `http.telecharger`, `http.encoder`.
- TCP : `reseau.connecter` / `reseau.ecouter` + `accepter`, `envoyer`, `recevoir`, `fermer` ; UDP : `reseau.udp`, `envoyervers`, `recevoirde`.
- `ftp.connecter{…}` puis `liste`, `lire`, `envoyer`… ; `ssh.connecter{…}` puis `executer`, `lire`, `envoyer`, `liste`…

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

CHAPITRE 12 · MULTITÂCHE

Faire plusieurs choses à la fois

Surveiller trois serveurs en même temps, calculer sur tous les cœurs du processeur, faire patienter une tâche pendant qu'une autre avance… Lexis+ propose deux outils complémentaires, les tâches coopératives et les fils d'exécution.



12.1 Tâches ou fils ?
12.2 Les tâches coopératives
12.3 Les canaux : faire communiquer des tâches
12.4 Les fils d'exécution

Dans ce chapitre, vous apprendrez à…

- distinguer tâche coopérative et fil d'exécution
- faire communiquer des tâches par des canaux
- répartir un calcul sur plusieurs fils
- lancer des tâches et attendre leurs résultats
- lancer un fil d'exécution et lui envoyer des messages
- arrêter proprement tâches et fils

12.1 Tâches ou fils ?

Un programme exécute normalement **une instruction à la fois**. Pour mener plusieurs activités de front, Lexis+ offre deux mécanismes :

	Tâches (`tache`)	Fils d'exécution (`fil`)
Principe	des procédures qui s'exécutent **à tour de rôle** dans le même programme	de vrais fils du système, qui s'exécutent **en parallèle** sur plusieurs cœurs
Variables	partagées avec le programme	**séparées** : chaque fil a son propre interpréteur
Communication	variables globales, canaux	**messages** (`envoyer` / `recevoir`)
Idéal pour	organiser des activités qui attendent souvent (minuteries, files de travail, simulation)	les **calculs lourds**, les attentes bloquantes (réseau, commandes longues)

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Une image : la cuisine

Les **tâches**, c'est un cuisinier seul qui surveille trois casseroles : il remue l'une, passe à la suivante, revient… Rien ne se fait vraiment en même temps, mais rien n'est oublié. Les **fils**, ce sont trois cuisiniers : chacun travaille vraiment en parallèle, mais ils ne partagent pas leur plan de travail ; ils se passent les plats par un passe-plat (les messages).

12.2 Les tâches coopératives

`tache.lancer(@PROC, args…)` crée une tâche qui exécutera `PROC(args…)`. Elle ne démarre pas tout de suite : les tâches tournent quand le programme appelle `tache.executer()` (qui les fait tourner jusqu'à ce qu'elles soient toutes terminées) ou `tache.attendre()`. Une tâche **rend la main** aux autres quand elle appelle `tache.ceder()`, `tache.attendre(secondes)` ou attend un message dans un canal.

Appel	Rôle
`tache.lancer(@proc, ...)`	crée une tâche ; renvoie un objet tâche `t`
`tache.executer()`	fait tourner les tâches jusqu'à ce qu'elles soient toutes terminées
`tache.attendre(s)`	dans une tâche : la suspend s secondes ; ailleurs : attend en faisant tourner les tâches
`tache.ceder()`	rend la main aux autres tâches
`tache.nombre()`, `tache.maintenant()`	nombre de tâches en cours ; heure en secondes
`t:attendre()`	attend la fin de la tâche et renvoie son résultat
`t:etat()`	`"suspendue"`, `"en cours"`, `"terminee"` ou `"erreur"`
`t:annuler()`	arrête une tâche suspendue

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Trois minuteurs de cuisine — principal.prg

```
tache.lancer(@MINUTEUR, "pâtes", 0.30)
tache.lancer(@MINUTEUR, "œufs", 0.18)
tache.lancer(@MINUTEUR, "sauce", 0.42)
? tache.nombre(), "minuteurs lancés"
debut = tache.maintenant()
tache.executer()                          && fait tourner les tâches jusqu'au bout
? "Tout est prêt."

procedure MINUTEUR(plat, duree)
  tache.attendre(duree)                   && rend la main pendant l'attente
  ? "  Ding !", plat, ": prêt après", arrondi(tache.maintenant() - debut, 1), "s"
retourner
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
3 minuteurs lancés
  Ding ! œufs : prêt après 0.2 s
  Ding ! pâtes : prêt après 0.3 s
  Ding ! sauce : prêt après 0.4 s
Tout est prêt.
```

Les trois minuteurs attendent **en même temps** : le programme dure environ 0,4 seconde, et non 0,9.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Récupérer un résultat, gérer une erreur

EXEMPLE Résultats et états des tâches — principal.prg

```
locale t1 = tache.lancer(@SOMME_CARRES, 100)
locale t2 = tache.lancer(@SOMME_CARRES, 1000)
locale t3 = tache.lancer(@DIVISER, 1, 0)
? "État avant :", t1:etat()
tache.executer()
? "Résultats :", t1:attendre(), "et", t2:attendre()
? "États après :", t1:etat(), "/", t3:etat()

locale longue = tache.lancer(@ATTENTE_LONGUE)
tache.attendre(0.05)                    && laisse démarrer la tâche
? "Tâche longue :", longue:etat()
longue:annuler()
? "Après annulation :", longue:etat()

fonction SOMME_CARRES(n)
  locale s = 0
  pour i = 1 a n
    s += i * i
    si i % 100 = 0
      tache.ceder()                     && laisse travailler les autres
    finsi
  suivant
retourner s

fonction DIVISER(a, b)
retourner a / b

procedure ATTENTE_LONGUE()
  tache.attendre(60)
retourner
```

CE QU'AFFICHE LE PROGRAMME

```
État avant : suspendue
[erreur] tâche : principal.prg:26: division par zéro
Résultats : 338350 et 333833500
États après : terminee / erreur
Tâche longue : suspendue
Après annulation : terminee
```

Une erreur dans une tâche ne fait pas planter le programme : elle est affichée, et la tâche passe à l'état `"erreur"`.

12.3 Les canaux : faire communiquer des tâches

Un **canal** est une file d'attente : une tâche y **envoie** des valeurs, une autre les **reçoit** dans l'ordre. Si le canal est vide, `recevoir` attend (en rendant la main) ; si le canal a une capacité limitée et qu'il est plein, `envoyer` attend. C'est le schéma classique **producteur / consommateur**.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Appel	Rôle
`tache.canal([capacite])`	crée un canal (capacité 0 = illimitée)
`c:envoyer(valeur)`	envoie (attend si le canal est plein)
`c:recevoir([delai])`	reçoit (attend si vide) ; `nul` après le délai
`c:nombre()`	nombre de valeurs en attente

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Une chaîne de préparation de commandes — principal.prg

```
a_preparer = tache.canal(2)              && au plus 2 commandes en attente
a_expedier = tache.canal()

tache.lancer(@RECEPTION, {"C-101", "C-102", "C-103", "C-104"})
tache.lancer(@PREPARATEUR, "Ada")
tache.lancer(@EXPEDITION)
tache.executer()
? "Journée terminée."

procedure RECEPTION(commandes)
  pour chaque c dans commandes
    a_preparer:envoyer(c)                && attend si 2 commandes sont déjà en attente
    ? "Reçue    :", c, "(en attente :", a_preparer:nombre() + ")"
  suivant
  a_preparer:envoyer("FIN")
retourner

procedure PREPARATEUR(nom)
  tantque vrai
    locale c = a_preparer:recevoir()
    si c = "FIN"
      sortir
    finsi
    tache.attendre(0.02)                 && le temps de préparer
    ? "Préparée :", c, "par", nom
    a_expedier:envoyer(c)
  fintantque
  a_expedier:envoyer("FIN")
retourner

procedure EXPEDITION()
  locale n = 0
  tantque vrai
    locale c = a_expedier:recevoir()
    si c = "FIN"
      sortir
    finsi
    n += 1
    ? "Expédiée :", c
  fintantque
  ? n, "colis expédiés"
retourner
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Reçue    : C-101 (en attente : 1)
Reçue    : C-102 (en attente : 2)
Reçue    : C-103 (en attente : 2)
Préparée : C-101 par Ada
Expédiée : C-101
Reçue    : C-104 (en attente : 2)
Préparée : C-102 par Ada
Expédiée : C-102
Préparée : C-103 par Ada
Expédiée : C-103
Préparée : C-104 par Ada
Expédiée : C-104
4 colis expédiés
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

```
Journée terminée.
```

> **Attention**
>
> Les tâches se partagent un seul « cuisinier » : une attente **bloquante** (une requête réseau longue, `systeme.attendre`, une commande système) bloque **toutes** les tâches. Pour ce genre d'attente, utilisez un **fil**.

12.4 Les fils d'exécution

`fil.demarrer(code)` lance un **vrai fil** du système, avec son **propre interpréteur** : il exécute le code Lexis+ donné (une chaîne, souvent écrite entre `[[ ]]`) ou un fichier `.prg`, **en parallèle** du programme principal. Comme il ne partage aucune variable, on communique par **messages** : tables, chaînes, nombres et logiques sont copiés d'un côté à l'autre.

Appel	Côté	Rôle
`fil.demarrer(code_ou_fichier, ...)`	parent	lance un fil ; renvoie l'objet fil `f`
`f:envoyer(valeur)`	parent	envoie une valeur au fil
`f:recevoir([delai])`	parent	reçoit une valeur du fil ; `nul` après le délai
`f:actif()`	parent	vrai tant que le fil s'exécute
`f:joindre()`	parent	attend la fin du fil ; son erreur éventuelle est relevée
`f:arreter()`	parent	demande l'arrêt du fil
`fil.recevoir([delai])`	fil	reçoit une valeur du parent
`fil.envoyer(valeur)`	fil	envoie une valeur au parent
`fil.estenfant()`	les deux	vrai dans un fil lancé par `fil.demarrer`

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Un calcul en arrière-plan pendant que le programme continue — principal.prg

```
locale f = fil.demarrer([[
  locale n = fil.recevoir()                && attend sa consigne
  locale s = 0
  pour i = 1 a n
    s += racine(i)
  suivant
  fil.envoyer({n = n, resultat = arrondi(s, 2), enfant = fil.estenfant()})
]])
f:envoyer(2000000)
? "Calcul lancé ; le programme principal continue…"
locale autre = 0
pour i = 1 a 1000
  autre += i
suivant
? "… pendant ce temps, 1 + … + 1000 =", autre
locale r = f:recevoir()                    && attend le message du fil
? "Somme des racines jusqu'à", r.n, ":", r.resultat, "(calculée dans un fil :", r.enfant +
")"
f:joindre()
? "Fil terminé, encore actif ?", f:actif()
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Calcul lancé ; le programme principal continue…
… pendant ce temps, 1 + … + 1000 = 500500
Somme des racines jusqu'à 2000000 : 1885618790.06 (calculée dans un fil : vrai)
Fil terminé, encore actif ? faux
```

Répartir un calcul sur plusieurs cœurs

Pour un programme plus important, le code du fil s'écrit dans un fichier `.prg` placé dans un **sous-dossier** du projet et qui ne contient **que des procédures**, dont `PRINCIPAL` : le projet principal le charge sans l'exécuter, et le fil exécute sa procédure `PRINCIPAL`.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

FICHIER Le programme de chaque fil — fils/premiers.prg

```
&& fils/premiers.prg : compte les nombres premiers d'un intervalle
procedure PRINCIPAL
  locale tranche = fil.recevoir()           && {de, a}
  locale n = 0
  pour x = tranche.de a tranche.a
    si EST_PREMIER(x)
      n += 1
    finsi
  suivant
  fil.envoyer({de = tranche.de, a = tranche.a, premiers = n})
retourner

fonction EST_PREMIER(x)
  si x < 2
    retourner faux
  finsi
  pour d = 2 a entier(racine(x))
    si x % d = 0
      retourner faux
    finsi
  suivant
retourner vrai
```

EXEMPLE Quatre fils pour compter les nombres premiers — principal.prg

```
locale limite = 200000, nb_fils = 4
locale fils = {}
locale debut = systeme.horloge()
pour k = 1 a nb_fils
  locale f = fil.demarrer("fils/premiers.prg")
  f:envoyer({de = (k - 1) * limite / nb_fils + 1, a = k * limite / nb_fils})
  inserer f dans fils
suivant
locale total = 0
pour chaque f dans fils
  locale r = f:recevoir()
  ? "  de", chaine(r.de, 7), "à", chaine(r.a, 7), ":", r.premiers, "nombres premiers"
  total += r.premiers
  f:joindre()
suivant
? "Total :", total, "nombres premiers jusqu'à", limite
? "Durée :", arrondi(systeme.horloge() - debut, 2), "s (les fils ont travaillé en
parallèle)"
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
   de       1 à   50000 : 5133 nombres premiers
   de   50001 à  100000 : 4459 nombres premiers
   de  100001 à  150000 : 4256 nombres premiers
   de  150001 à  200000 : 4136 nombres premiers
Total : 17984 nombres premiers jusqu'à 200000
Durée : 0.68 s (les fils ont travaillé en parallèle)
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Arrêter un fil, récupérer son erreur

EXEMPLE Surveillance, arrêt et erreurs — principal.prg

```
locale veilleur = fil.demarrer([[
  tantque vrai
    systeme.attendre(0.05)                  && un travail sans fin
  fintantque
]])
? "Veilleur actif :", veilleur:actif()
? "Message reçu dans les 0,1 s :", type(veilleur:recevoir(0.1))
veilleur:arreter()                          && demande d'arrêt
veilleur:joindre()
? "Après arrêt, actif :", veilleur:actif()

locale fragile = fil.demarrer([[ erreur "fichier de configuration absent" ]])
essayer
  fragile:joindre()                         && l'erreur du fil remonte ici
capturer e
  ? "Le fil a échoué :", e
finessayer
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Veilleur actif : vrai
Message reçu dans les 0,1 s : nul
Après arrêt, actif : faux
Le fil a échoué : fil : fil:1: fichier de configuration absent
```

À retenir

- **Tâches** : `tache.lancer(@P, …)`, `tache.executer()`, `tache.attendre(s)`, `tache.ceder()`, `t:attendre()`, `t:etat()`, `t:annuler()`, canaux `tache.canal()` + `envoyer` / `recevoir`.
- **Fils** : `fil.demarrer(code ou "dossier/fichier.prg")`, `f:envoyer` / `f:recevoir`, côté fil `fil.recevoir` / `fil.envoyer`, `f:joindre()`, `f:arreter()`, `f:actif()`.
- Tâches = variables partagées, à tour de rôle ; fils = parallélisme réel, variables séparées, messages.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

CHAPITRE 13 · INTERFACE GRAPHIQUE

Des fenêtres et des boutons

Le paquet interface donne à vos programmes de vraies fenêtres — boutons, champs de saisie, listes, grilles, graphiques, calendriers, menus — construites avec la bibliothèque wdgui. Le programme ne déroule plus une suite d'instructions : il réagit aux actions de l'utilisateur.



13.1 La programmation par événements
13.2 La fenêtre
13.3 Les contrôles de base
13.4 Les événements
13.5 Organiser la fenêtre : conteneurs et mise en page
13.6 Listes, tableaux et arbres
13.7 La grille de saisie
13.8 Menus, graphiques et contrôles de planification
13.9 Les boîtes de dialogue

Dans ce chapitre, vous apprendrez à…

- ✔ créer une fenêtre et y placer des contrôles
- ✔ lire et modifier les propriétés des contrôles
- ✔ afficher des données dans des listes, tableaux et grilles
- ✔ réagir aux clics et aux saisies avec des gestionnaires
- ✔ organiser une fenêtre avec conteneurs, cadres et onglets
- ✔ utiliser menus, graphiques et boîtes de dialogue

13.1 La programmation par événements

Jusqu'ici, nos programmes se déroulaient du haut vers le bas, puis s'arrêtaient. Un programme à fenêtres fonctionne autrement : il **construit** la fenêtre, puis **attend**. Chaque action de l'utilisateur (un clic, une frappe, un choix dans une liste) déclenche un **événement**, auquel le programme répond en exécutant une procédure, le **gestionnaire** de cet événement.

1. On charge le paquet : `ui = paquet("interface")` (l'usage est de le ranger dans `ui`).
2. On crée une **fenêtre** : `fen = ui.fenetre{titre = "…", largeur = …, hauteur = …}`.
3. On y **ajoute** des contrôles : `btn = fen:ajouter(ui.bouton("OK"))`.
4. On associe des **gestionnaires** : `btn.surclic = @VALIDER` (une référence de procédure).

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

5. On lance la boucle d'événements : `ui.executer()`, qui rend la main quand toutes les fenêtres sont fermées.

EXEMPLE Un compteur de clics — principal.prg

```
ui = paquet("interface")

fen = ui.fenetre{titre = "Compteur", largeur = 360, hauteur = 180}
lbl = fen:ajouter(ui.etiquette("Vous n'avez pas encore cliqué", "centre"))
btn = fen:ajouter(ui.bouton("Cliquez-moi", {defaut = vrai}))
raz = fen:ajouter(ui.bouton("Remettre à zéro"))
nb = 0                                  && globale : partagée avec les gestionnaires

btn.surclic = @COMPTER                  && référence de procédure (@)
raz.surclic = @REMETTRE
ui.executer()                           && attend la fermeture de la fenêtre

procedure COMPTER(ev)
  nb += 1
  lbl.libelle = "Clics : " + nb + sisinon(nb >= 10, " — bravo !", "")
retourner

procedure REMETTRE(ev)
  nb = 0
  lbl.libelle = "Compteur remis à zéro"
retourner
```



Aperçu schématique de la fenêtre après trois clics

Les variables des gestionnaires sont globales

Les gestionnaires sont des procédures appelées par l'interface, pas par votre programme : ils ne voient que les variables **globales**. C'est pourquoi `fen`, `lbl`, `btn` et `nb` sont déclarées sans `locale` dans le programme principal.

Astuce

Dans l'atelier, `ui.executer()` attend que vous fermiez les fenêtres. Avec le runtime, si le programme se termine sans l'appeler, la boucle est lancée automatiquement. `ui.tracer(vrai)` affiche dans la console chaque événement reçu : très utile pour comprendre ce qui se passe.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

13.2 La fenêtre

`ui.fenetre{…}` crée une fenêtre. Ses options :

Option	Rôle
`titre`	texte de la barre de titre
`largeur`, `hauteur`	taille en pixels
`disposition`	façon de ranger les contrôles : `vertical` (défaut), `horizontal`, `grille`, `flux`, `bordure`, `pile`, `absolue`
`theme`	apparence : `windows`, `macos`, `linux`, `natif`
`sombre`	`vrai` pour le thème sombre

Méthodes : `fen:ajouter(controle, …)`, `fen:fermer()`, `fen:fixertheme(nom, [sombre])`. Fonctions du paquet : `ui.executer()`, `ui.quitter()` (ferme toutes les fenêtres), `ui.fenetres()` (nombre de fenêtres ouvertes), `ui.tracer([actif])`.

13.3 Les contrôles de base

Création	Contrôle	Propriété principale
`ui.etiquette(texte, [alignement])`	texte fixe (`gauche`, `centre`, `droite`)	`libelle`
`ui.bouton(libelle, [{defaut=, annuler=}])`	bouton (défaut : Entrée ; annuler : Échap)	`libelle`
`ui.saisie(texte)` ou `ui.saisie{texte=, indication=, multiligne=}`	champ de saisie	`valeur`
`ui.cases(option, …)`	cases à cocher (valeur = masque de bits)	`valeur`, `:coche(i)`
`ui.radio(option, …)`	boutons radio	`valeur` : numéro choisi (1, 2…)
`ui.combo(element, …)`	liste déroulante	`valeur` : numéro choisi ; texte : `:element(i)`
`ui.liste([multi])`	liste	`:selection()` : numéro choisi
`ui.curseur()`, `ui.plage()`, `ui.ascenseur()`	potentiomètre, plage, ascenseur	`valeur`, `min`, `max`
`ui.compteur()`	nombre avec flèches	`valeur`, `min`, `max`
`ui.jauge()`	barre de progression	`valeur`

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Création	Contrôle	Propriété principale
`ui.notation()`	étoiles	`valeur`
`ui.image(chemin)`, `ui.codebarre(texte)`	image, code-barres	—

Les propriétés

Chaque contrôle est un **objet** dont les propriétés se lisent et s'écrivent avec le point :

Propriété	Rôle
`valeur` (ou `texte`), `num`	contenu sous forme de texte, de nombre (pour une liste, un combo, des boutons radio ou des onglets : le **numéro** de l'élément choisi, à partir de 1 ; pour un calendrier : `"AAAAMMJJ"`)
`libelle` (ou `titre`)	texte affiché d'une étiquette, d'un bouton, d'un cadre
`visible`, `actif`, `etat`	visibilité ; activation ; `actif`, `grise`, `lecture`
`infobulle`	texte affiché au survol
`nom`	nom libre, pour vous y retrouver
`x`, `y`, `largeur`, `hauteur`	position et taille
`couleur`, `fond`, `bordure`, `accent`, `rayon`	couleurs (`"#RRGGBB"`) et arrondi
`coche`	état d'une case
`min`, `max`	bornes d'un curseur, d'un compteur, d'une jauge
`ancrage`, `poids`, `largeurfixe`, `hauteurfixe`, `etirer`, `marge`, `espacement`, `colonnes`	mise en page (voir plus loin)
`famille`, `genre`	famille du contrôle (`bouton`, `saisie`…), genre d'un graphe

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Un formulaire de commande — principal.prg

```
ui = paquet("interface")
fen = ui.fenetre{titre = "Nouvelle commande", largeur = 420, hauteur = 360}

fen:ajouter(ui.etiquette("Client"))
client = fen:ajouter(ui.saisie{indication = "Nom du client"})
fen:ajouter(ui.etiquette("Produit"))
produit = fen:ajouter(ui.combo("Ramette A4", "Stylos (boîte de 50)", "Classeurs (lot de 10)"))
fen:ajouter(ui.etiquette("Quantité"))
qte = fen:ajouter(ui.compteur())
qte.min = 1
qte.max = 99
qte.valeur = 1
livraison = fen:ajouter(ui.radio("Retrait en magasin", "Livraison standard", "Express"))
options = fen:ajouter(ui.cases("Emballage cadeau", "Facture par e-mail"))
total = fen:ajouter(ui.etiquette("", "droite"))
ok = fen:ajouter(ui.bouton("Commander", {defaut = vrai}))

prix = {4.20, 17.50, 28.90}
ports = {0, 5.90, 12.50}
produit.surchangement = @RECALCULER
qte.surchangement = @RECALCULER
livraison.surchangement = @RECALCULER
ok.surclic = @COMMANDER
RECALCULER(nul)
ui.executer()

procedure RECALCULER(ev)
  locale p = prix[produit.num] ou 0              && num : numéro choisi (à partir de 1)
  locale port = ports[livraison.num] ou 0
  total.libelle = "Total : " + chaine(p * qte.num + port, 1, 2) + " €"
retourner

procedure COMMANDER(ev)
  si vide(client.valeur)
    ui.alerte("Indiquez le nom du client.", "attention")
    client:focus()
    retourner
  finsi
  si ui.confirmer("Valider la commande de " + client.valeur + " ?")
    ui.alerte("Commande enregistrée. " + total.libelle, "information", "Merci")
    fen:fermer()
  finsi
retourner
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr



Aperçu schématique du formulaire

13.4 Les événements

On associe un gestionnaire à un événement en affectant une référence de procédure à la propriété `sur<événement>` du contrôle. Le gestionnaire reçoit un paramètre `ev`, une fiche qui décrit l'événement.

Propriété	Événement
`surclic`, `surdoubleclic`, `surclicdroit`, `surclicmilieu`, `surdoubleclicdroit`	clics de souris
`surboutonbas`, `surboutonhaut`	bouton de souris enfoncé / relâché
`surentree`, `sursortie`, `surdeplacement`, `surmolette`	survol, déplacement, molette
`surtouchebas`, `surtouchehaut`, `sursaisie`	clavier : touche, texte tapé
`surfocus`, `surfinfocus`	le contrôle reçoit / perd le focus
`surchangement`, `surselection`, `survalidation`	contenu modifié, élément choisi, validation
`surdeploiement`, `surrepli`	nœud d'arbre déplié / replié
`surouverture`, `surfermeture`, `surredimension`, `suractivation`, `surdesactivation`	fenêtre
`surevenement`	**tous** les événements du contrôle

Les champs de `ev` : `genre` (nom de l'événement), `id` (contrôle visé), `courant` (contrôle dont le gestionnaire s'exécute), `x`, `y`, `dx`, `dy`, `touche`, `modificateurs`, `index`, `colonne`, `bouton` (`gauche`, `droit`, `milieu`), `texte`, `chemin` (menus et arbres), `repetition`. Un événement **remonte** du contrôle vers ses conteneurs puis la fenêtre ; `ev:arreter()` arrête cette remontée.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Un convertisseur qui réagit à la frappe — principal.prg

```
ui = paquet("interface")
fen = ui.fenetre{titre = "Convertisseur", largeur = 360, hauteur = 170}
euros = fen:ajouter(ui.saisie{indication = "Montant en euros"})
devise = fen:ajouter(ui.combo("USD", "GBP", "CHF", "JPY"))
resultat = fen:ajouter(ui.etiquette("", "centre"))
taux = {USD = 1.08, GBP = 0.86, CHF = 0.95, JPY = 162.4}

euros.sursaisie = @CONVERTIR          && à chaque caractère tapé
devise.surselection = @CONVERTIR
fen.surfermeture = @AU_REVOIR
ui.executer()

procedure CONVERTIR(ev)
  locale m = nombre(euros.valeur)
  si m = nul
    resultat.libelle = "Tapez un montant"
    resultat.couleur = "#B42318"
    retourner
  finsi
  resultat.couleur = "#0B1A4F"
  locale code = devise:element(devise.num)        && texte de l'élément choisi
  resultat.libelle = chaine(m * taux[code], 1, 2) + " " + code
retourner

procedure AU_REVOIR(ev)
  ? "Fenêtre fermée par l'utilisateur"
retourner
```

13.5 Organiser la fenêtre : conteneurs et mise en page

Les contrôles d'une fenêtre se rangent selon sa **disposition** (vertical par défaut : les uns sous les autres). Pour des mises en page plus riches, on imbrique des **conteneurs**, chacun avec sa propre disposition, et l'on règle la place de chaque contrôle.

Conteneur	Rôle
`ui.conteneur([disposition], [colonnes])`	conteneur général (`vertical`, `horizontal`, `grille`, `flux`, `bordure`, `pile`, `absolue`)
`ui.colonne()`, `ui.rangee()`	conteneur vertical, horizontal
`ui.superconteneur([disposition])`	conteneur sans marge
`ui.cadre(titre, [disposition])`	cadre de groupe avec titre
`ui.onglets()` puis `o:ajouterpage(titre, [disposition])`	onglets ; chaque page est un conteneur
`ui.separateur{vertical=, position=}`	deux zones séparées par une barre déplaçable
`ui.panneau{disposition=, defilement=, colonnes=}`	zone défilante (`toujours`, `automatique`, `masque`) ; `:defilera(x, y)`, `:defilerde(dx, dy)`, `:rendrevisible(ctl)`

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Propriété de mise en page	Rôle
`ancrage`	dans une disposition `bordure` : `haut`, `bas`, `gauche`, `droite`, `centre`
`poids`	part de l'espace libre attribuée au contrôle
`largeurfixe`, `hauteurfixe`	taille imposée
`etirer`	le contrôle s'étire pour remplir la place
`marge`, `espacement`	marge intérieure, espace entre contrôles (conteneurs)
`colonnes`	nombre de colonnes d'une disposition `grille`

EXEMPLE Fiche client avec cadres, rangées et onglets — principal.prg

```
ui = paquet("interface")
fen = ui.fenetre{titre = "Fiche client", largeur = 520, hauteur = 400, disposition = "bordure"}

entete = fen:ajouter(ui.rangee())
entete.ancrage = "haut"
entete:ajouter(ui.etiquette("Client n° C001 — Durand SA"))

onglets = fen:ajouter(ui.onglets())
onglets.ancrage = "centre"
p1 = onglets:ajouterpage("Coordonnées", "grille")
p1.colonnes = 2
p1:ajouter(ui.etiquette("Adresse"), ui.saisie("12 rue des Lilas"))
p1:ajouter(ui.etiquette("Ville"), ui.saisie("Brest"))
p1:ajouter(ui.etiquette("Téléphone"), ui.saisie("02 98 00 00 01"))
p2 = onglets:ajouterpage("Commercial")
cadre = p2:ajouter(ui.cadre("Conditions"))
cadre:ajouter(ui.cases("Client prioritaire", "Paiement à 60 jours"))
note = p2:ajouter(ui.notation())
note.valeur = 4

boutons = fen:ajouter(ui.rangee())
boutons.ancrage = "bas"
boutons:ajouter(ui.bouton("Annuler", {annuler = vrai}), ui.bouton("Enregistrer", {defaut = vrai}))
ui.executer()
```



© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

13.6 Listes, tableaux et arbres

Contrôle	Méthodes
`ui.liste([multi])`, `ui.combo(…)`	`:ajouter(element, …)`, `:inserer(i, e)`, `:supprimer(i)`, `:vider()`, `:selectionner(i)`, `:deselectionner(i)`, `:selection()`, `:nombre()`, `:element(i)`, `:modifier(i, texte)`, `:chercher(texte)`
`ui.tableau()`	`:ajoutercolonne(titre, [largeur], [alignement])`, `:ajouterligne(cellule, …)`, `:insererligne(…)`, `:supprimer(i)`, `:vider()`, `:cellule(l, c)`, `:fixercellule(l, c, texte)`, `:ligne(i)`, `:selection()`, `:selectionner(i)`, `:nombre()`, `:trier(colonne, [croissant])`
`ui.arbre()`	`:ajouter(chemin…)`, `:supprimer(chemin…)`, `:vider()`, `:deplier(chemin…)`, `:replier(chemin…)`, `:estdeplie(chemin…)`, `:selectionner(chemin…)`, `:nombre()`

> **La numérotation commence à 1**
>
> Comme dans tout Lexis+, les éléments des listes, des tableaux, des grilles, des onglets, des séries et catégories de graphes sont numérotés **à partir de 1**.

Un **chemin** d'arbre s'écrit sous forme de plusieurs arguments (`"Clients", "Brest"`), d'une liste, ou d'un texte `"Clients/Brest"`. Le séparateur interne est `ui.sep`.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Explorateur de clients : arbre et tableau liés — principal.prg

```
ui = paquet("interface")
ouvrirbase(":memoire:")
sql "create table clients (nom varchar, ville varchar, ca double)"
sql "insert into clients values ('Durand','Brest',12500), ('Martin','Brest',8300),
('Petit','Lyon',4500), ('Roux','Lyon',15200)"

fen = ui.fenetre{titre = "Clients par ville", largeur = 600, hauteur = 320, disposition =
"horizontal"}
arbre = fen:ajouter(ui.arbre())
arbre.largeurfixe = 180
tab = fen:ajouter(ui.tableau())
tab.etirer = vrai
tab:ajoutercolonne("Nom", 160)
tab:ajoutercolonne("Chiffre d'affaires", 140, "droite")

pour chaque v dans requete("select distinct ville from clients order by ville")
  arbre:ajouter("Villes", v.ville)
suivant
arbre:deplier("Villes")
arbre.surselection = @AFFICHER_VILLE
ui.executer()

procedure AFFICHER_VILLE(ev)
  si ev.chemin = nul ou #ev.chemin < 2
    retourner
  finsi
  tab:vider()
  pour chaque c dans requete("select nom, ca from clients where ville = ? order by nom",
ev.chemin[2])
    tab:ajouterligne(c.nom, chaine(c.ca, 1, 2) + " €")
  suivant
retourner
```

13.7 La grille de saisie

La **grille** est le contrôle le plus riche : un véritable petit tableur, avec colonnes typées, saisie dans les cellules, tris, pieds de colonne (totaux), mises en forme conditionnelles et imports / exports.

Méthode	Rôle
`ui.grille{multi=, remplissage=}`	crée la grille (`remplissage` : `programme`, `manuel`, `automatique`)
`g:ajoutercolonne{nom=, titre=, genre=, largeur=, choix=, decimales=}`	colonne (`genre` : `texte`, `nombre`, `date`, `image`, `bouton`, `combo`, `case`)
`g:ajouterligne(…)`, `g:insererligne(i, …)`, `g:modifierligne(i, …)`, `g:supprimerligne(i)`, `g:vider()`	lignes
`g:cellule(l, col)`, `g:fixercellule(l, col, v)`, `g:ligne(l)`, `g:modifiercellule(l, col)`	cellules (colonne par nom ou par numéro)
`g:nombre()`, `g:nombrecolonnes()`, `g:colonne(col)`	dimensions, objet colonne

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Méthode	Rôle
`g:selection([i])`, `g:nombreselection()`, `g:selectionner(l)`, `g:deselectionner(l)`, `g:toutselectionner()`	sélection
`g:trier(col, [croissant])`, `g:chercher(col, valeur, [exact], [debut])`, `g:trouver(texte)`	tri et recherche
`g:fixerpied(col, genre)`, `g:valeurpied(col, [genre])`	pied : `somme`, `moyenne`, `min`, `max`, `nombre`
`g:ajouterformat(col, operateur, valeur, {fond=, texte=, ligneentiere=})`, `g:effacerformats()`	mise en forme conditionnelle (`=`, `<>`, `<`, `<=`, `>`, `>=`, `contient`, `vide`, `nonvide`)
`g:fixerrupture([col])`	regroupement visuel sur une colonne
`g:fixermodifiable([b])`, `g:fixermodecolonne(col, mode)`, `g:fixermodeligne(l, mode)`, `g:fixerremplissage(mode)`	saisie (`lecture`, `modifiable`, `herite`)
`g:fixerlibellebouton(col, texte)`	libellé des boutons d'une colonne `bouton`
`g:verscsv([sep], [entete])`, `g:decsv(texte, [ajout])`, `g:chargercsv(chemin, [sep])`, `g:enregistrercsv(chemin, [sep])`	CSV
`g:versjson([joli])`, `g:chargerjson(chemin)`, `g:enregistrerjson(chemin)`	JSON

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Saisie de notes de frais — principal.prg

```
ui = paquet("interface")
fen = ui.fenetre{titre = "Notes de frais", largeur = 640, hauteur = 380, disposition = "bordure"}
menu = fen:ajouter(ui.barremenu())
menu:ajouter("Fichier", "Exporter en CSV…")
menu:ajouter("Fichier", "Quitter")
menu:ajouter("Lignes", "Ajouter une ligne")

g = fen:ajouter(ui.grille{remplissage = "manuel"})
g:ajoutercolonne{nom = "date", titre = "Date", genre = "date", largeur = 100}
g:ajoutercolonne{nom = "nature", titre = "Nature", genre = "combo", largeur = 150,
                 choix = {"Repas", "Transport", "Hôtel", "Fournitures"}}
g:ajoutercolonne{nom = "montant", titre = "Montant", genre = "nombre", largeur = 100, decimales = 2}
g:ajoutercolonne{nom = "justif", titre = "Justificatif", genre = "case", largeur = 90}
g:fixermodifiable(vrai)
g:fixerpied("montant", "somme")
g:ajouterformat("montant", ">", "100", {fond = "#FFF4E5", ligneentiere = vrai})
g:ajouterformat("justif", "=", "0", {texte = "#B42318"})
g:ajouterligne("2026-10-02", "Transport", "86.40", "1")
g:ajouterligne("2026-10-03", "Hôtel", "128.00", "0")
g:ajouterligne("2026-10-03", "Repas", "24.90", "1")
menu.surselection = @MENU
ui.executer()

procedure MENU(ev)
  locale choix = ev.chemin[#ev.chemin]
  faire cas
  cas choix = "Ajouter une ligne"
    g:ajouterligne(date(), "Repas", "0", "0")
    g:modifiercellule(g:nombre(), "montant")       && place le curseur de saisie
  cas choix = "Exporter en CSV…"
    locale p = ui.enregistrerfichier{nom = "frais.csv", filtre = "CSV|*.csv"}
    si p <> nul
      g:enregistrercsv(p, ";")
      ui.alerte("Export terminé : " + g:nombre() + " lignes, total " + g:valeurpied("montant"))
    finsi
  cas choix = "Quitter"
    fen:fermer()
  fincas
retourner
```

Notes de frais

Fichier Lignes

Date	Nature	Montant	Justificatif
2026-10-02	Transport	86.40	☑
2026-10-03	Hôtel	128.00	☐
2026-10-03	Repas	24.90	☑
	Total	**239.30**	

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

La ligne de plus de 100 € est surlignée ; le pied de colonne fait la somme

13.8 Menus, graphiques et contrôles de planification

Contrôle	Création et méthodes
barre de menus	`ui.barremenu()` ; `m:ajouter("Fichier", "Ouvrir")` ; le choix arrive dans `surselection` (`ev.chemin`)
menu contextuel	`c:menucontextuel(element, …)` sur n'importe quel contrôle ; le choix arrive dans `surselection`
graphe	`ui.graphe([genre], [titre])` (`colonnes`, `lignes`, `aires`, `secteurs`) ; `:ajouterdonnee(serie, categorie, valeur)`, `:libellecategorie(i, t)`, `:libelleserie(i, t)`, `:fixergenre(g)`
calendrier	`ui.calendrier(["AAAAMMJJ"])` ; `:afficher(annee, mois)`
kanban	`ui.kanban()` ; `:ajouterliste(titre)` → n° de liste ; `:ajouter(liste, texte)`
mosaïque (treemap)	`ui.mosaique()` ; `:ajouter(libelle, valeur)`
Gantt	`ui.gantt()` ; `:ajoutertache(nom, debut, duree, [avancement])` → n° ; `:fixeravancement(i, v)`
visionneuse PDF	`ui.pdf()` ; `:ouvrir(chemin, [mdp])`, `:nouveau([l], [h])`, `:enregistrer([chemin])`, `:fermer()`, `:nombrepages()`, `:page()`, `:allera(n)`, `:zoom(z)`, `:ajoutertexte(page, x, y, texte, [taille])`, `:importerpages(chemin, [plage], [position])`

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Tableau de bord des ventes — principal.prg

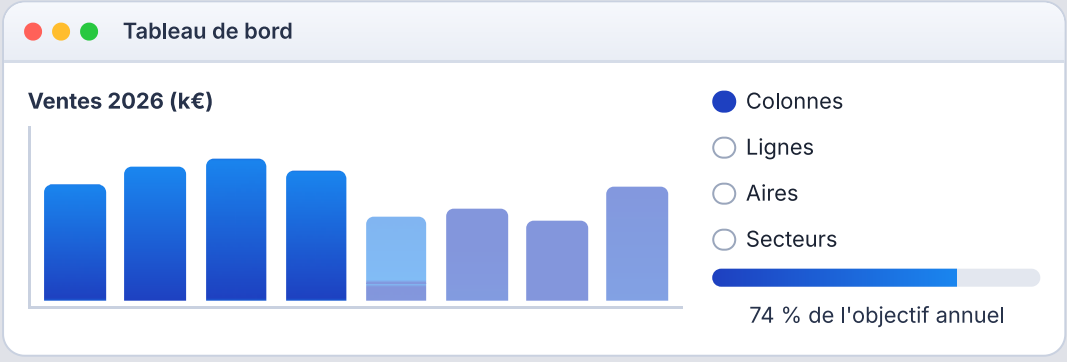
```
ui = paquet("interface")
fen = ui.fenetre{titre = "Tableau de bord", largeur = 720, hauteur = 420, disposition = "horizontal"}

graphe = fen:ajouter(ui.graphe("colonnes", "Ventes 2026 (k€)"))
graphe.poids = 2
mois = {"Jan", "Fév", "Mar", "Avr"}
series = {"Magasin", "En ligne"}
ventes = {{42, 48, 51, 47}, {30, 33, 29, 41}}
pour s = 1 a 2                                  && séries et catégories numérotées à partir de 1
  graphe:libelleserie(s, series[s])
  pour m = 1 a 4
    graphe:ajouterdonnee(s, m, ventes[s][m])
  suivant
suivant
pour m = 1 a 4
  graphe:libellecategorie(m, mois[m])
suivant

droite = fen:ajouter(ui.colonne())
choix = droite:ajouter(ui.radio("Colonnes", "Lignes", "Aires", "Secteurs"))
objectif = droite:ajouter(ui.jauge())
objectif.valeur = 74
droite:ajouter(ui.etiquette("74 % de l'objectif annuel", "centre"))
projet = droite:ajouter(ui.gantt())
projet:ajoutertache("Inventaire", 0, 3, 100)
projet:ajoutertache("Soldes", 2, 5, 40)
choix.surchangement = @CHANGER_GENRE
graphe:menucontextuel("Exporter", "Imprimer")
graphe.surselection = @MENU_GRAPHE
ui.executer()

procedure CHANGER_GENRE(ev)
  locale genres = {"colonnes", "lignes", "aires", "secteurs"}
  graphe:fixergenre(genres[choix.num])
retourner

procedure MENU_GRAPHE(ev)
  ui.alerte("Vous avez choisi : " + ev.texte)
retourner
```



Graphe à deux séries, choix du genre, jauge d'objectif

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

13.9 Les boîtes de dialogue

Les dialogues standard s'ouvrent par une simple fonction ; le programme attend la réponse de l'utilisateur.

Fonction	Rôle et résultat
`ui.alerte(message, [icone], [titre])`	message (`information` , `question` , `attention` , `stop` , `aucune`)
`ui.confirmer(message, [icone])`	OK / Annuler → `vrai` / `faux`
`ui.demander(message, [defaut])`	demande un texte
`ui.ouvrirfichier{titre=, filtre=, dossier=}`	choix d'un fichier à ouvrir → chemin, ou `nul` si annulé
`ui.enregistrerfichier{nom=, filtre=}`	choix d'un fichier à enregistrer → chemin ou `nul`
`ui.couleur(["#RRGGBB"])`	sélecteur de couleur → `"#RRGGBB"` ou `nul`
`ui.police([famille], [taille])`	sélecteur de police → `{famille, taille, gras, italique}` ou `nul`

Un **filtre** de fichiers s'écrit `"Libellé|*.ext"`, par exemple `"Tableurs|*.csv;*.xlsx"`.

EXEMPLE Ouvrir un CSV et l'afficher dans une grille — principal.prg

```
ui = paquet("interface")
chemin = ui.ouvrirfichier{titre = "Choisir un fichier CSV", filtre = "Fichiers CSV|*.csv"}
si chemin = nul
  ui.alerte("Aucun fichier choisi.", "information")
  systeme.quitter(0)
finsi
lignes = csv.lire(chemin, {sep = ";", entete = vrai})
fen = ui.fenetre{titre = fichier.nom(chemin) + " — " + #lignes + " lignes", largeur = 760, hauteur = 480}
g = fen:ajouter(ui.grille())
pour chaque nom dans lignes.entete
  g:ajoutercolonne{nom = nom, titre = nom}
suivant
pour chaque l dans lignes
  locale cellules = {}
  pour chaque nom dans lignes.entete
    inserer chaine(l[majuscule(nom)] ou "") dans cellules
  suivant
  g:ajouterligne(cellules)
suivant
si ui.confirmer("Activer le thème sombre ?", "question")
  fen:fixertheme("natif", vrai)
finsi
ui.executer()
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

À retenir

- `ui = paquet("interface")`, `fen = ui.fenetre{…}`, `fen:ajouter(ui.controle(…))`, `ui.executer()`.
- Gestionnaires : `controle.surclic = @PROC`, procédure `PROC(ev)` ; variables partagées **globales**.
- Propriétés : `valeur`, `num`, `libelle`, `visible`, `etat`, `couleur`… ; mise en page : conteneurs, `ancrage`, `poids`, `etirer`.
- Données : `liste`, `tableau`, `arbre`, `grille` (colonnes typées, pieds, formats, CSV / JSON).
- Dialogues : `alerte`, `confirmer`, `demander`, `ouvrirfichier`, `enregistrerfichier`, `couleur`, `police`.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

CHAPITRE 14 · PAGES WEB

Des pages web dans vos programmes

Le paquet webview affiche une page HTML, CSS et JavaScript dans une fenêtre native, avec le moteur web du système, et la relie à votre programme dans les deux sens. Vous dessinez l'interface comme un site web ; Lexis+ garde les données et les règles.



14.1 Le principe
14.2 Ouvrir une fenêtre web
14.3 Lexis+ appelle JavaScript
14.4 JavaScript appelle Lexis+
14.5 Événements, console et fermeture
14.6 Une application complète

Dans ce chapitre, vous apprendrez à…

- ouvrir une fenêtre web avec du HTML, un fichier ou une URL
- appeler une procédure Lexis+ depuis la page
- réagir au chargement de la page et à la fermeture
- appeler une fonction JavaScript depuis Lexis+ et recevoir son résultat
- échanger des données (listes, fiches) entre les deux mondes
- construire une petite application complète

14.1 Le principe

Une fenêtre `webview` contient un **navigateur** sans barre d'adresse : WebView2 sous Windows, WKWebView sous macOS, WebKitGTK sous Linux. Vous y affichez une page écrite en **HTML** (la structure), **CSS** (la présentation) et **JavaScript** (le comportement), exactement comme un site web — avec toute la richesse visuelle que cela permet. Le paquet construit un **pont** entre la page et le programme :

Sens	Côté Lexis+	Côté JavaScript
Lexis+ → JS	`f:appeler("fonction", arg…, [@rappel])`	`function fonction(arg…) { … }`

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Sens	Côté Lexis+	Côté JavaScript
Lexis+ → JS	`f:evaluer("code", [@rappel])`	le code est exécuté dans la page
Lexis+ → JS	`f:emettre("nom", donnees)`	`lexis.sur("nom", d => …)`
JS → Lexis+	`f:lier("nomjs", @PROC)`	`await nomjs(a, b)` renvoie le retour de `PROC(a, b)`
JS → Lexis+	`f:autoriser("PROC", …)`	`await lexis.appeler("PROC", a, b)`
JS → Lexis+	`f.surmessage = @PROC`	`lexis.envoyer(donnees)`
JS → Lexis+	console de l'atelier	`console.log(…)`, `lexis.journal(…)`, erreurs JavaScript

Les valeurs traversent le pont en **JSON** : nombres, textes, logiques et `nul` sont conservés ; une liste devient un tableau, une fiche un objet. Les noms Lexis+ (`NOM`) deviennent `nom` en JavaScript, une table vide devient `[]`, et les clefs JavaScript deviennent des noms Lexis+ (`d.texte`).

HTML, CSS, JavaScript en deux mots

Le **HTML** décrit le contenu : `<h1>Titre</h1>`, `<button id="go">OK</button>`. Le **CSS** l'habille : couleurs, polices, marges. Le **JavaScript** le fait vivre : `document.getElementById("go").onclick = …`. Une instruction JavaScript précédée de `await` attend le résultat d'une opération asynchrone, comme un appel à Lexis+.

14.2 Ouvrir une fenêtre web

Appel	Rôle
`webview.fenetre{titre=, largeur=, hauteur=, taille=, fichier=, url=, html=, debogage=, console=}`	crée la fenêtre ; `taille` : `normale`, `min`, `max`, `fixe` ; `debogage = vrai` : outils de développement ; `console = faux` : ne pas renvoyer `console.log` vers Lexis+
`f:html(code)`, `f:ouvrir(chemin)`, `f:naviguer(url)`	affiche du HTML, un fichier local (relatif au programme), une URL
`f:initialiser(js)`	script exécuté avant chaque page
`f:taille(l, h, [genre])`, `f.titre`	taille et titre
`f:executer()`	affiche la fenêtre et traite ses événements jusqu'à sa fermeture
`f:fermer()`	ferme la fenêtre
`webview.disponible()`, `webview.erreur()`, `webview.version()`, `webview.bibliotheque()`	diagnostic de la bibliothèque

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Appel	Rôle
`webview.url(chemin)`, `webview.js(valeur)`	URL `file://` d'un fichier ; littéral JavaScript d'une valeur

EXEMPLE Bonjour webview : un bouton JavaScript qui appelle Lexis+ — principal.prg

```
f = webview.fenetre{titre = "Bonjour webview", largeur = 520, hauteur = 340, html = PAGE()}
f:lier("saluer", @SALUER)          && crée la fonction JavaScript window.saluer(nom)
f:executer()                       && rend la main quand la fenêtre est fermée
? "fenêtre fermée"

fonction SALUER(nom)
  si nom = nul ou nom = ""
    nom = "inconnu"
  finsi
retourner "Bonjour " + nom + " ! (réponse de Lexis+ à " + heure() + ")"

fonction PAGE()
retourner [[<!doctype html>
<html lang="fr"><head><meta charset="utf-8">
<style>
  body { font-family: system-ui; display: grid; place-items: center; height: 100vh; margin: 0; background: #f4f1ea; }
  main { background: #fff; padding: 28px; border-radius: 12px; box-shadow: 0 6px 24px rgba(0,0,0,.08); }
  button { background: #2f5d8a; color: #fff; border: 0; border-radius: 6px; padding: 8px 14px; }
</style></head>
<body><main>
  <h1>Lexis+ &amp;amp; webview</h1>
  <input id="nom" placeholder="Votre prénom"> <button id="go">Saluer</button>
  <p id="reponse"></p>
</main>
<script>
  document.getElementById('go').onclick = async () => {
    const nom = document.getElementById('nom').value;
    // saluer() appelle la procédure Lexis+ SALUER et attend sa réponse
    document.getElementById('reponse').textContent = await saluer(nom);
  };
</script></body></html>]]
```

Où placer la bibliothèque

`webview.dll` (Windows), `webview.dylib` (macOS) ou `webview.so` (Linux, avec WebKitGTK 4.1 : paquet `libwebkit2gtk-4.1-0`) se place dans `winOS/`, `macOS/` ou `linOS/` à côté de l'exécutable ou du programme, ou dans la variable `WDLEXIS_WEBVIEW`. Sous **macOS**, la fenêtre doit appartenir au fil principal : lancez ces programmes avec le runtime `wdlexisrt`.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

14.3 Lexis+ appelle JavaScript

La page doit être **chargée** avant qu'on l'appelle : on place donc les appels dans la procédure `f.surpret`, appelée quand la page est prête (les appels faits plus tôt sont mis en attente automatiquement). La réponse de JavaScript arrive de façon **asynchrone** : on ne l'attend pas, on la reçoit dans une procédure de **rappel** `RAPPEL(valeur, erreur)` — `erreur` vaut `nul` en cas de succès.

EXEMPLE Remplir la page depuis Lexis+ et récupérer un calcul — principal.prg

```
produits = {{nom = "Stylo", prix = 1.2, stock = 40}, {nom = "Cahier", prix = 3.5, stock = 12},
            {nom = "Agrafeuse", prix = 14.9, stock = 3}}
f = webview.fenetre{titre = "Lexis+ -> JavaScript", largeur = 760, hauteur = 560, fichier = "page.htm"}
f.surpret = @PRET                  && la page est chargée : on peut l'appeler
f.surmessage = @MESSAGE            && la page demande quelque chose (lexis.envoyer)
f:executer()

procedure PRET(ev)
  f:appeler("afficher", "Bonjour ! Ce message vient de Lexis+.")      && sans résultat
  pour chaque p dans produits
    f:appeler("ajouterLigne", p)                  && une fiche devient un objet JS
  suivant
  f:appeler("calculer", produits, @TOTAL)         && avec résultat (fonction async)
  f:evaluer("navigator.userAgent", @RESULTAT)     && évaluer une expression
  f:appeler("app.theme", "clair")                 && chemin pointé vers un objet
  f:emettre("horloge", {heure = heure(), date = date()})
retourner

procedure TOTAL(valeur, erreur)
  si erreur <> nul
    ? "calculer() a échoué :", erreur
    retourner
  finsi
  f.titre = "Stock : " + chaine(valeur.total, 1, 2) + " €"
retourner

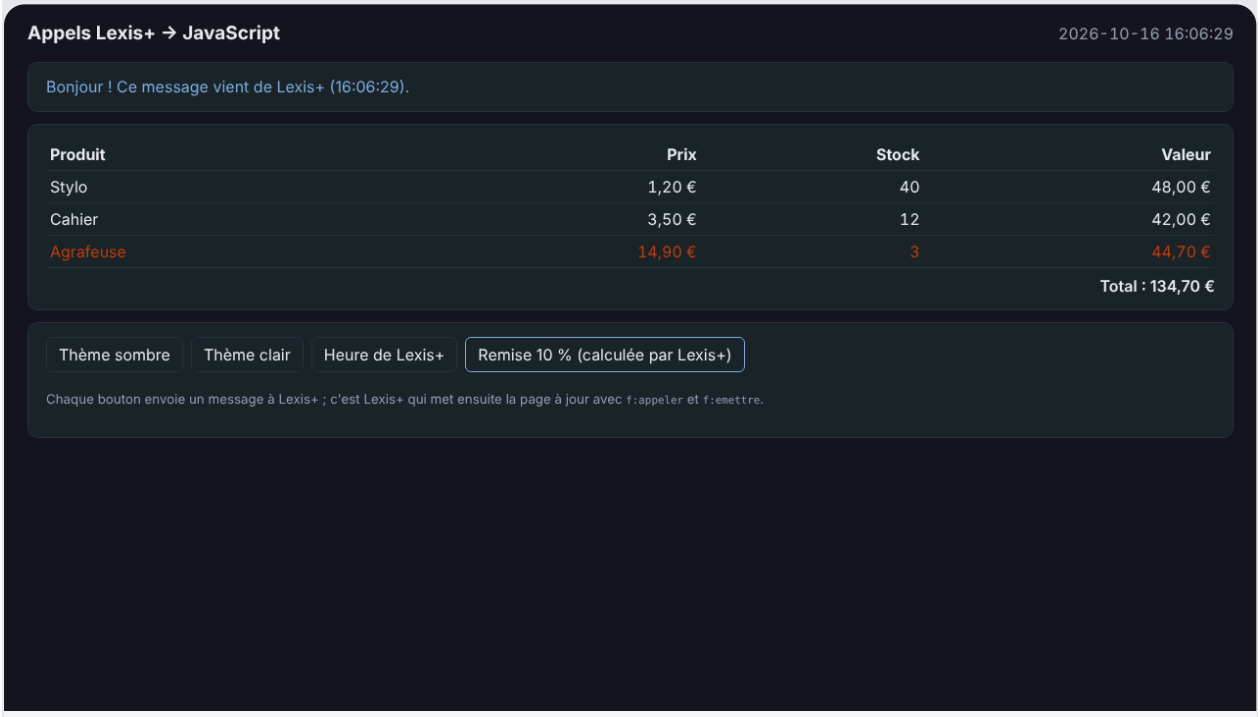
procedure RESULTAT(valeur, erreur)
  ? "moteur web :", valeur ou erreur
retourner

procedure MESSAGE(d)
  si d.action = "theme"
    f:appeler("app.theme", d.valeur)
  finsi
retourner
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

PAGE **Les fonctions JavaScript appelées par Lexis+** page.htm (extrait)

```
<script>
  function afficher(message) {                          "c">// f:appeler("afficher", texte)
    document.getElementById('message').textContent = message;
  }
  function ajouterLigne(p) {                            "c">// p = {nom, prix, stock}
    const tr = document.createElement('tr');
    tr.innerHTML = `<td>${p.nom}</td><td>${p.prix}</td><td>${p.stock}</td>`;
    document.getElementById('produits').appendChild(tr);
  }
  async function calculer(produits) {                   "c">// le résultat repart vers Lexis+
    let total = 0;
    for (const p of produits) total += p.prix * p.stock;
    return { total, articles: produits.length };
  }
  window.app = { theme(nom) { document.body.classList.toggle('sombre', nom === 'sombre');
} };
  lexis.sur('horloge', d => document.getElementById('horloge').textContent = d.heure);
  "c">// un bouton de la page demande à Lexis+ de changer le thème :
  "c">//   <button onclick="lexis.envoyer({action: 'theme', valeur:
'sombre'})">Sombre</button>
</script>
```



La page de l'exemple 02_lexis_vers_js après les appels de Lexis+ (thème sombre demandé par la page)

14.4 JavaScript appelle Lexis+

Deux moyens d'exposer vos procédures à la page :

- `f:lier("nomjs", @PROC)` crée une **fonction JavaScript globale** `nomjs(…)` ; `f:lier("nomjs")` seul lie la procédure du même nom ;

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

- `f:autoriser("PROC", …)` permet d'appeler les procédures nommées par `lexis.appeler("PROC", …)` (`"*"` les autorise toutes — à éviter si la page affiche du contenu venu d'Internet).

Dans les deux cas, l'appel renvoie une **Promise** : `await` attend le résultat. Si la procédure lève une erreur (`erreur "…"`), la Promise est **rejetée** : on l'intercepte avec `try … catch`.

EXEMPLE Calculs de TVA et enregistrement de factures appelés par la page — principal.prg

```
taux = {normal = 20, intermediaire = 10, reduit = 5.5}
factures = {}
f = webview.fenetre{titre = "JavaScript -> Lexis+", largeur = 720, hauteur = 600, fichier = "page.htm"}
f:lier("calculerTTC", @CALCULER_TTC)      && window.calculerTTC(ht, genre)
f:lier("taux")                            && window.taux() -> procédure TAUX
f:autoriser("ENREGISTRER", "HISTORIQUE") && lexis.appeler("ENREGISTRER", fiche)
f:executer()

fonction TAUX()
retourner taux

fonction CALCULER_TTC(ht, genre)
  locale t = taux[majuscule(genre)]
  si t = nul
    erreur "taux inconnu : " + genre          && rejette la Promise JavaScript
  finsi
  locale tva = arrondi(ht * t / 100, 2)
retourner {ht = ht, taux = t, tva = tva, ttc = arrondi(ht + tva, 2)}

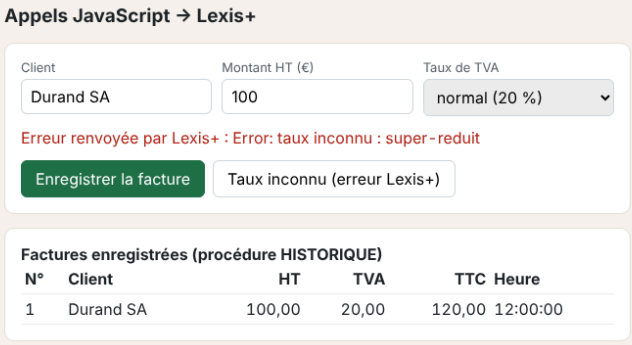
fonction ENREGISTRER(fiche)
  si vide(fiche.client)
    retourner {ok = faux, message = "Le nom du client est obligatoire."}
  finsi
  locale c = CALCULER_TTC(fiche.ht, fiche.genre)
  c.client = fiche.client
  c.numero = #factures + 1
  inserer c dans factures
retourner {ok = vrai, numero = c.numero, message = "Facture n°" + c.numero + " enregistrée."}

fonction HISTORIQUE()
retourner factures
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

PAGE Le côté JavaScript — page.htm (extrait)

```
<script>
  async function recalculer() {
    try {
      const r = await calculerTTC(Number(ht.value), genre.value);   "c">// procédure liée
      calcul.textContent = `${r.ht} € HT + ${r.taux} % = ${r.ttc} € TTC`;
    } catch (e) {
      calcul.textContent = 'Lexis+ : ' + e;                          "c">// erreur "taux
inconnu"
    }
  }
  formulaire.onsubmit = async ev => {
    ev.preventDefault();
    const r = await lexis.appeler('ENREGISTRER', { client: client.value, ht:
Number(ht.value), genre: genre.value });
    calcul.textContent = r.message;
    const liste = await lexis.appeler('HISTORIQUE');                 "c">// liste de fiches
    console.log(liste.length + ' facture(s)');                       "c">// → console Lexis+
  };
</script>
```



L'exemple 03_js_vers_lexis : les montants sont calculés et les factures enregistrées par Lexis+

14.5 Événements, console et fermeture

Propriété / méthode	Rôle
`f.surpret = @P`	page chargée : `P(ev)`, `ev.url`
`f.surmessage = @P`	message de `lexis.envoyer(donnees)` : `P(donnees)`
`f.surfermeture = @P`	après la fermeture de la fenêtre

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Propriété / méthode	Rôle
`f.prete`, `f.ouverte`	page chargée ; fenêtre encore ouverte
`f:delier(nomjs)`	retire une liaison
`lexis.journal(…)`, `console.log(…)`	écrivent dans la console Lexis+ (`[js] …`), erreurs JavaScript comprises (`[js:erreur] …`)
`lexis.pret` (JS)	vrai une fois la page chargée

Déboguer la page

`webview.fenetre{debogage = vrai}` active les **outils de développement** du navigateur (clic droit › *Inspecter*) : vous inspectez le HTML, testez du JavaScript, suivez les échanges. Et toutes les erreurs JavaScript s'affichent dans la console de l'atelier.

14.6 Une application complète

L'exemple `04_application` réunit tout : la page est l'interface, Lexis+ garde les données (dans `taches.json`), applique les règles (texte obligatoire, 120 caractères au plus…) et prévient la page à chaque changement avec `f:emettre`.

EXEMPLE **Liste de tâches : le programme Lexis+** — principal.prg

```
chemin = fichier.joindre(systeme.dossierprogramme(), "taches.json")
taches = sisinon(fichier.existe(chemin), json.lire(chemin), {})
prochain = 1
pour chaque t dans taches
  prochain = max(prochain, t.id + 1)
suivant

f = webview.fenetre{titre = "Tâches - Lexis+", largeur = 560, hauteur = 640, fichier = "page.htm"}
f:autoriser("AJOUTER", "BASCULER", "SUPPRIMER", "STATISTIQUES")
f.surpret = @PRET
f.surfermeture = @ENREGISTRER
f:executer()

procedure PRET(ev)
  f:emettre("taches", taches)                  && la page affiche la liste
retourner

procedure PREVENIR()
  ENREGISTRER()
  f:emettre("taches", taches)                  && mise à jour de la page
retourner

procedure ENREGISTRER()
  json.ecrire(chemin, taches)
retourner

fonction AJOUTER(texte, priorite)
  texte = rogner(chaine(texte))
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

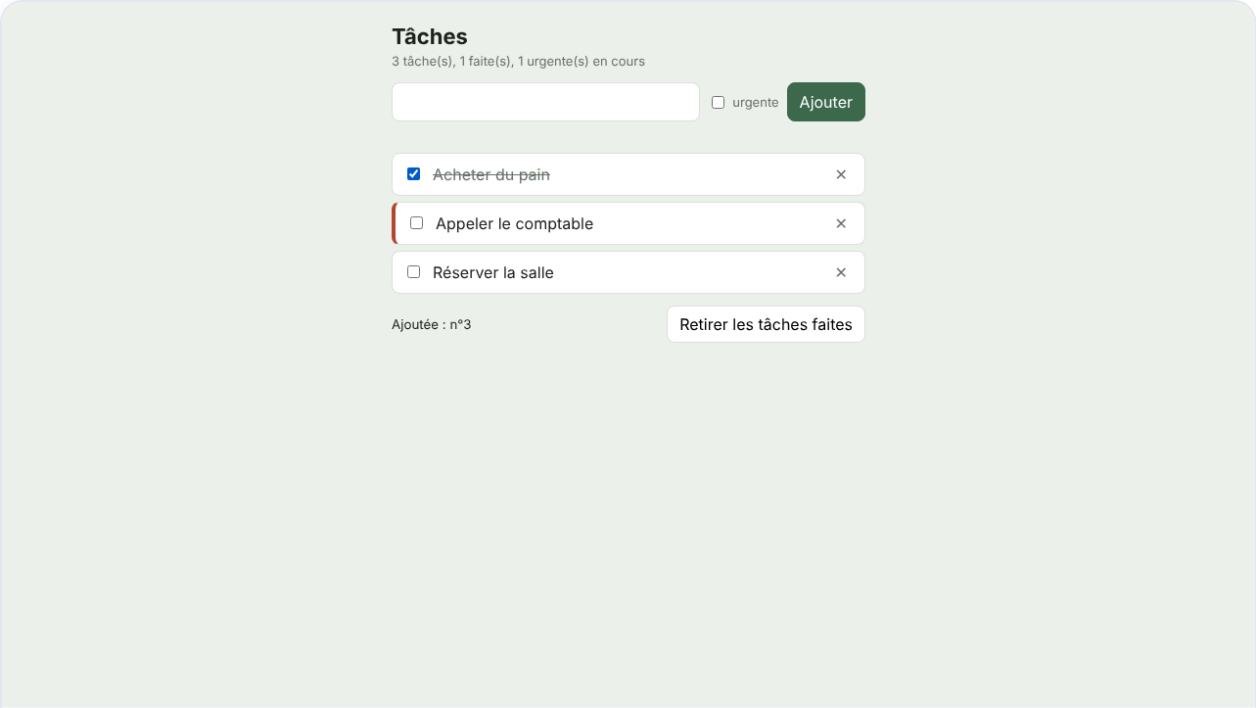
```
  si longueur(texte) = 0
    erreur "Le texte de la tâche est vide."
  finsi
  locale t = {id = prochain, texte = texte, faite = faux,
              priorite = sisinon(priorite = "haute", "haute", "normale"), creee =
maintenant()}
  prochain += 1
  inserer t dans taches
  PREVENIR()
retourner t

fonction BASCULER(id)
  pour chaque t dans taches
    si t.id = id
      t.faite = non t.faite
      PREVENIR()
      retourner t.faite
    finsi
  suivant
  erreur "tâche introuvable : " + id

fonction SUPPRIMER(id)
  pour i = 1 a #taches
    si taches[i].id = id
      retirer(taches, i)
      PREVENIR()
      retourner vrai
    finsi
  suivant
retourner faux

fonction STATISTIQUES()
  locale s = {total = #taches, faites = 0}
  pour chaque t dans taches
    si t.faite
      s.faites += 1
    finsi
  suivant
retourner s
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr



L'exemple 04_application : interface en HTML, données et règles en Lexis+

À retenir

- `f = webview.fenetre{fichier = "page.htm"}`, `f.surpret = @PRET`, `f:executer()`.
- Lexis+ → JS : `f:appeler("fn", …, @RAPPEL)`, `f:evaluer("code", @RAPPEL)`, `f:emettre("nom", d)` ; rappel `(valeur, erreur)`.
- JS → Lexis+ : `f:lier("nomjs", @P)` → `await nomjs()` ; `f:autoriser("P")` → `await lexis.appeler("P")` ; `lexis.envoyer(d)` → `f.surmessage`.
- Erreur Lexis+ = Promise rejetée ; `console.log` = console Lexis+ ; `debogage = vrai` pour inspecter.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

CHAPITRE 15 · INTELLIGENCE ARTIFICIELLE

L'intelligence artificielle

Six paquets, écrits entièrement en Nim et intégrés à Lexis+, donnent accès aux grands modèles de langage, au dessin, à la synthèse vocale et aux bases de l'apprentissage automatique. Tout tourne sur votre processeur, sans service en ligne et sans bibliothèque externe.



15.1 Panorama
15.2 Charger un modèle et dialoguer
15.3 Régler les réponses
15.4 Recevoir la réponse au fil de l'eau
15.5 Des réponses structurées : JSON
15.6 Pièces jointes et formats de réponse
15.7 La mémoire d'une conversation
15.8 Sous le capot : tokens, logits et vecteurs
15.9 Recherche documentaire et agents
15.10 Les fichiers GGUF et la quantification
15.11 Dessiner : le paquet dessin
15.12 Faire parler : le paquet audio
15.13 Apprendre : le paquet tenseur
15.14 Créer et spécialiser un modèle : le paquet entrainement

Dans ce chapitre, vous apprendrez à…

- charger un modèle de langage et dialoguer avec lui
- obtenir des réponses structurées en JSON, joindre des documents
- inspecter et quantifier un fichier GGUF
- régler la créativité et la longueur des réponses
- comprendre tokens, logits et vecteurs
- dessiner, faire parler, entraîner un réseau et créer un modèle

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

15.1 Panorama

Un **modèle de langage** (LLM) est un très grand réseau de neurones entraîné à prédire la suite d'un texte. À force de lire des milliards de phrases, il a appris la langue, des connaissances générales et une forme de raisonnement : il sait répondre à une question, résumer, traduire, extraire des informations, écrire du code…

Lexis+ lit les modèles au format **GGUF**, celui de llama.cpp, Ollama et LM Studio : Llama 3, Mistral, Qwen, Gemma, Phi… Vous téléchargez un fichier (par exemple `Llama-3.2-1B-Instruct-Q4_K_M.gguf`, environ 800 Mo), et c'est tout : le calcul se fait **sur votre machine**, vos données ne la quittent pas.

Paquet	Rôle
`llm`	charger un modèle, conversations, pièces jointes, réponses texte / JSON / image / audio / fichier, tokens, logits, vecteurs
`gguf`	inspecter un fichier GGUF, quantifier un modèle, fusionner un adaptateur LoRA
`dessin`	images BMP / PPM / PNG, rendu SVG, dessin vectoriel sur une toile
`audio`	synthèse vocale (français, anglais), mélodies, fichiers WAV
`tenseur`	tenseurs à différentiation automatique, optimiseurs : les bases de l'apprentissage
`entrainement`	tokeniseur, création d'un modèle, pré-entraînement, dialogue, LoRA, export GGUF

Les exemples de ce chapitre

Les exemples de dialogue utilisent le fichier `mini-assistant-q8.gguf`, un **tout petit modèle** (≈ 1 million de paramètres) que Lexis+ crée lui-même en quelques minutes à la fin de ce chapitre (section 15.14). Il ne sait parler que de capitales, de fruits, de jours et d'additions, mais il s'utilise **exactement** comme un vrai modèle : remplacez simplement son nom par celui d'un Llama ou d'un Mistral pour obtenir des réponses de qualité. Les sorties affichées ont été réellement produites.

Performances

Compilez l'atelier et le runtime avec `-d:release` (10 à 30 fois plus rapide) ; pour un usage sur votre seule machine, `--passC:-march=native` active en plus les instructions SIMD du processeur. `llm.fils(0)` utilise tous les cœurs. Pendant un calcul, l'interpréteur est libéré : les fenêtres restent réactives et le bouton **Arrêter** interrompt la génération.

15.2 Charger un modèle et dialoguer

Appel	Rôle
`llm.charger(chemin, [{fils=0, verbeux=faux}])`	charge un modèle GGUF (fichier « mappé » : quasi instantané)

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Appel	Rôle
`m:conversation({systeme=, contexte=4096, maxtokens=512, echantillonnage=, gabarit="auto", langue="fr", arrets=})`	nouvelle conversation
`c:demander(question, [options ou @flux])`	pose une question ; renvoie une **réponse**
`m:decrire()`	résumé du modèle ; `m.tokeniseur` : son tokeniseur
`m:fermer()`	libère le modèle

La **réponse** est une fiche qui s'affiche comme son texte : `texte`, `brut`, `fichiers`, `json`, `tokensprompt`, `tokensgeneres`, `arret` (raison de l'arrêt), `secondes`, `vitesse` (tokens par seconde).

EXEMPLE Première conversation — principal.prg

```
locale modele = llm.charger("mini-assistant-q8.gguf")
locale conv = modele:conversation({echantillonnage = llm.glouton(), maxtokens = 60,
contexte = 128})

pour chaque question dans {"Bonjour !", "Quelle est la capitale de la Grèce ?", "Combien
font 3 plus 4 ?"}
  locale r = conv:demander(question)
  ? "Vous :", question
  ? "IA   :", r                                  && la réponse s'affiche comme son texte
  conv:reinitialiser()                            && chaque question repart de zéro
suivant

locale r = conv:demander("De quelle couleur est une fraise ?")
? "Réponse :", r.texte
? "Tokens : question", r.tokensprompt, "/ réponse", r.tokensgeneres, "— arrêt :", r.arret
? "Vitesse :", entier(r.vitesse), "tokens/s — durée :", arrondi(r.secondes, 3), "s"
```

▶ CE QU'AFFICHE LE PROGRAMME

```
Vous : Bonjour !
IA   : Bonjour ! Comment puis-je t'aider ?
Vous : Quelle est la capitale de la Grèce ?
IA   : La capitale de la Grèce est Athènes.
Vous : Combien font 3 plus 4 ?
IA   : 3 plus 4 font 7.
Réponse : Une fraise est rouge.
Tokens : question 22 / réponse 5 — arrêt : fin de texte
Vitesse : 2208 tokens/s — durée : 0.002 s
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Le même programme avec un vrai modèle — principal.prg

```
locale modele = llm.charger(systeme.env("NIMLLM_MODEL") ou "Llama-3.2-1B-Instruct-Q4_K_M.gguf")
locale conv = modele:conversation({systeme = "Tu es un assistant comptable concis.", maxtokens = 200})
? "Vous : Explique en une phrase ce qu'est une facture d'acompte."
? "IA   :", conv:demander("Explique en une phrase ce qu'est une facture d'acompte.")
```

▶ **CE QUE L'ON OBTIENT**

```
Vous : Explique en une phrase ce qu'est une facture d'acompte.
IA   : Une facture d'acompte est une facture émise avant la livraison ou la fin d'une prestation, pour demander au client le paiement d'une partie du prix total, qui sera déduite de la facture finale.
```

15.3 Régler les réponses

Un modèle choisit chaque mot (chaque **token**) parmi les plus probables. L'**échantillonnage** règle ce choix : `llm.glouton()` prend toujours le plus probable (réponses déterministes) ; une **température** plus haute rend les réponses plus variées, plus « créatives ».

Réglage	Rôle
`llm.echantillonnage({…})`, `llm.glouton({…})`	réglages par défaut, décodage glouton
`temperature`	0 = toujours le plus probable ; 0,7 = équilibré ; > 1 = fantaisiste
`topk`, `topp`, `minp`	ne garder que les k meilleurs, le noyau de probabilité p, ceux au-dessus de minp × le meilleur
`repetition`, `fenetre`, `presence`, `frequence`	pénaliser les répétitions
`graine`	rend un tirage reproductible
`biais`	`{[id] = -100}` interdit un token, `+5` le favorise

Propriétés de la conversation : `c.systeme` (consigne générale), `c.maxtokens`, `c.arrets` (textes qui arrêtent la réponse), `c.echantillonnage`, `c.langue`, `c.gabarit`, `c.essaisjson`, `c.sansreflexion`, `c.tailleimage`, `c.voix` ; en lecture : `historique`, `messages`, `tokens`, `contexte`, `memoire`.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Glouton, température et arrêts — principal.prg

```
locale modele = llm.charger("mini-assistant-q8.gguf")
locale question = "Quelle est la capitale de la France ?"

locale c1 = modele:conversation({echantillonnage = llm.glouton(), maxtokens = 40, contexte = 128})
? "Glouton        :", c1:demander(question)

pour chaque t dans {0.5, 1.5}
  locale c = modele:conversation({echantillonnage = {temperature = t, graine = 42}, maxtokens = 40, contexte = 128})
  ? "Température", t, ":", c:demander(question)
suivant

locale c3 = modele:conversation({echantillonnage = llm.glouton(), contexte = 128})
c3.maxtokens = 5                              && réponse tronquée à 5 tokens
locale r = c3:demander("Qui es-tu ?")
? "Tronquée :", r.texte, "— arrêt :", r.arret
```

CE QU'AFFICHE LE PROGRAMME

```
Glouton        : La capitale de la France est Paris.
Température 0.5 : La capitale de la France est Paris.
Température 1.5 : La capitale de la France est Paris.
Tronquée : Je suis — arrêt : limite de tokens
```

15.4 Recevoir la réponse au fil de l'eau

Avec un gros modèle, une réponse peut prendre plusieurs secondes. En passant une procédure de **flux** (`@AFFICHER`), on reçoit le texte **morceau par morceau**, comme dans les assistants en ligne. Si la procédure renvoie `faux`, la génération s'arrête.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE **Affichage progressif et interruption** principal.prg

```
locale modele = llm.charger("mini-assistant-q8.gguf")
locale conv = modele:conversation({echantillonnage = llm.glouton(), maxtokens = 60,
contexte = 128})
morceaux = 0
es.ecrire("IA : ")
locale r = conv:demander("Quel jour vient après jeudi ?", @AFFICHER)
? ""
? "(" + morceaux + " morceaux reçus)"

conv:reinitialiser()
es.ecrire("IA (interrompue) : ")
r = conv:demander("Quelle est la capitale de la Suède ?", @TROIS_MORCEAUX)
? ""
? "Raison de l'arrêt :", r.arret

fonction AFFICHER(morceau)
  es.ecrire(morceau)
  morceaux += 1
retourner vrai                                  && vrai : continuer

fonction TROIS_MORCEAUX(morceau)
  es.ecrire(morceau)
  morceaux += 1
retourner morceaux < 3 + 6                      && faux : arrêter
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
IA : Après jeudi vient vendredi.
(5 morceaux reçus)
IA (interrompue) : La capitale de la
Raison de l'arrêt : interrompu
```

15.5 Des réponses structurées : JSON

Pour qu'un programme exploite une réponse, mieux vaut des **données** qu'un texte libre. `c:demanderjson(question, schema)` demande une réponse JSON conforme à un **schéma** et la convertit en table Lexis+ (clefs en majuscules). L'option `format = "json"` de `demander` fait de même (résultat dans `r.json`). Le modèle réessaie automatiquement (`c.essaisjson`) si sa réponse n'est pas un JSON valide. `llm.extrairejson(texte)` retrouve un objet JSON dans n'importe quel texte.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Extraire les informations d'un courriel client — principal.prg

```
locale modele = llm.charger("Llama-3.2-1B-Instruct-Q4_K_M.gguf")
locale conv = modele:conversation({echantillonnage = {temperature = 0.2}, maxtokens =
300})
locale courriel = [[Bonjour, je m'appelle Julie Martin, j'ai commandé le 12 mars
une bouilloire (réf. BX-220) qui ne chauffe plus. Pouvez-vous me rembourser ?
Mon téléphone : 06 12 34 56 78.]]
locale fiche = conv:demanderjson("Extrais les informations de ce message client :\n" +
courriel,
  [[{"nom": str, "date_commande": str, "produit": str, "reference": str,
     "demande": "remboursement|échange|information", "telephone": str}]])
? json.encoder(fiche, vrai)
? "Fiche :", fiche.nom, "/", fiche.produit, "/", fiche.reference
```

▶ **CE QUE L'ON OBTIENT**

```
{
  "NOM": "Julie Martin",
  "DATE_COMMANDE": "12 mars",
  "PRODUIT": "bouilloire",
  "REFERENCE": "BX-220",
  "DEMANDE": "remboursement",
  "TELEPHONE": "06 12 34 56 78"
}
Fiche : Julie Martin / bouilloire / BX-220
```

15.6 Pièces jointes et formats de réponse

On peut joindre des **documents** à une question : texte, CSV, JSON, PDF, images PNG / BMP, sons WAV… Les modèles multimodaux comprennent les images ; pour les autres, le contenu textuel est inséré dans le message.

Appel	Rôle
`llm.joindre(chemin)`	pièce jointe lue sur le disque
`llm.joindretexte(nom, contenu)`, `llm.joindredonnees(nom, octets)`	pièce jointe construite en mémoire
`p:texte([max])`	texte inséré dans le message ; propriétés `nom`, `type`, `mime`, `taille`, `largeur`, `hauteur`, `image`
`c:demander(q, {pieces = {…}})`	question avec pièces jointes
`c:demander(q, {format = "markdown" / "json" / "image" / "audio" / "fichier", sortie = chemin})`	réponse mise en forme ; `image` : le modèle dessine en SVG, rendu en image ; `audio` : réponse lue à voix haute ; `fichier` : réponse enregistrée

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Analyser un fichier de ventes et produire un graphique — principal.prg

```
locale modele = llm.charger("Qwen2.5-3B-Instruct-Q4_K_M.gguf")
locale c = modele:conversation({maxtokens = 400})
locale r = c:demander("Quel produit rapporte le plus ? Réponds en une phrase.",
                      {pieces = {llm.joindre("ventes.csv")}})
? r
r = c:demander("Dessine un histogramme de la marge par produit.",
               {format = "image", sortie = "ventes.png"})
? "Image enregistrée :", r.fichiers[1], "(" + c.tailleimage + ")"
```

▶ **CE QUE L'ON OBTIENT**

```
Le produit le plus rentable est le Classeur (1 240 € de marge), suivi de l'Agenda.
Image enregistrée : ventes.png (640 x 400)
```

15.7 La mémoire d'une conversation

Une conversation **se souvient** de ses échanges : chaque question est envoyée au modèle avec tout l'historique, dans la limite de son **contexte** (en tokens). On peut l'enrichir, la corriger et la sauvegarder.

Appel	Rôle
`c:ajouter(role, texte)`	ajoute un message (`systeme` , `utilisateur` , `assistant`) : exemples, résultats d'outils
`c:generer([options ou @flux])`	réponse de l'assistant à l'historique actuel
`c:regenerer([@flux])` , `c:prolonger([maxtokens, @flux])`	nouveau tirage de la dernière réponse ; la prolonger
`c:annuler()` , `c:reinitialiser()`	retire le dernier échange ; oublie tout (sauf la consigne système)
`c:enregistrer(chemin)` , `c:charger(chemin)`	sauvegarde / reprise (JSON)
`c:etat()` , `c:prompt([q])` , `c:tokensprompt()`	état du contexte ; texte exact envoyé au modèle ; ses tokens

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Historique, sauvegarde et reprise d'une conversation — principal.prg

```
locale modele = llm.charger("mini-assistant-q8.gguf")
locale c = modele:conversation({echantillonnage = llm.glouton(), maxtokens = 40, contexte = 128})
c:demander("Bonjour !")
c:demander("Quelle est la capitale de la Suède ?")
? "Messages :", #c.historique
? "État :", c:etat()
pour chaque m dans c.historique
  ? "  [" + m.role + "]", m.contenu
suivant
c:enregistrer("conversation.json")

locale reprise = modele:conversation({echantillonnage = llm.glouton(), maxtokens = 40, contexte = 128})
? "Messages rechargés :", reprise:charger("conversation.json")
c:annuler()
? "Après annuler :", #c.historique, "messages"
? "Gabarit :", c.gabarit
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Messages : 4
État : contexte : 65/128 tokens, 4 messages, vitesse moyenne 9103.7 tok/s
  [user] Bonjour !
  [assistant] Bonjour ! Comment puis-je t'aider ?
  [user] Quelle est la capitale de la Suède ?
  [assistant] La capitale de la Suède est Stockholm.
Messages rechargés : 4
Après annuler : 2 messages
Gabarit : chatml
```

15.8 Sous le capot : tokens, logits et vecteurs

Un modèle ne lit pas des lettres mais des **tokens** : des morceaux de mots numérotés. Pour chaque position, il calcule un score (**logit**) pour chacun des tokens possibles ; les meilleurs scores donnent les suites les plus probables. Il sait aussi transformer un texte en **vecteur** (*embedding*) : deux textes de sens proche ont des vecteurs proches, ce qui permet la recherche par le sens.

Appel	Rôle
`t = m.tokeniseur` ; `t:encoder(texte, [bos], [speciaux])`, `t:decoder(ids)`, `t:morceau(id)`, `t:id(texte)`, `t:estspecial(id)`	tokens ; propriétés `taille`, `bos`, `eos`, `fins`, `gabarit`…
`x = m:contexte([taille])` ; `x:evaluer(ids)`, `x:completer(texte)`, `x:generer(ids)`, `x:reinitialiser()`, `x:tronquer(n)`	contexte brut (cache KV), sans gabarit de conversation
`l:meilleurs([n])`, `l:valeur(id)`, `l:argmax()`	logits : `{{id=, proba=}…}`
`s = llm.echantillonneur([reglages])` ; `s:tirer(logits)`, `s:accepter(id)`	échantillonneur manuel
`m:vecteur(texte)`, `llm.similarite(v1, v2)`	vecteur d'un texte, similarité cosinus (de −1 à 1)

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Appel	Rôle
`m:perplexite(texte)`	à quel point un texte « surprend » le modèle (bas = attendu)
`llm.prompt(gabarit, messages)`, `llm.gabarits()`, `llm.tokeniseur(chemin)`, `llm.predecouper(texte)`	outils de bas niveau

EXEMPLE Tokens, probabilités et similarité — principal.prg

```
locale modele = llm.charger("mini-assistant-q8.gguf")
locale tok = modele.tokeniseur
locale ids = tok:encoder("La capitale de la France est")
? "Tokens :", joindre(ids, " "), "→", #ids, "tokens sur", tok.taille
pour chaque id dans ids
  ?? "[" + tok:morceau(id) + "]"
suivant
?

locale ctx = modele:contexte(128)
locale logits = ctx:evaluer(tok:encoder("La capitale de la France est", vrai))
? "Suites les plus probables :"
pour chaque t dans logits:meilleurs(3)
  ? "  " + chaine(t.proba * 100, 5, 1) + " %", json(tok:morceau(t.id))
suivant

locale v1 = modele:vecteur("La capitale de la France est Paris.")
locale v2 = modele:vecteur("La capitale de la Suède est Stockholm.")
locale v3 = modele:vecteur("Une banane est jaune.")
? "Similarité capitale/capitale :", arrondi(llm.similarite(v1, v2), 3)
? "Similarité capitale/fruit    :", arrondi(llm.similarite(v1, v3), 3)
? "Perplexité phrase connue  :", arrondi(modele:perplexite("Après lundi vient mardi."), 2)
? "Perplexité phrase inconnue:", arrondi(modele:perplexite("Le moteur tourne vite."), 2)
```

▶ CE QU'AFFICHE LE PROGRAMME

```
Tokens : 295 288 282 275 463 279 → 6 tokens sur 512
[La][ capitale][ de][ la][ France][ est]
Suites les plus probables :
   87.9 % " P"
    1.4 % "La"
    1.0 % " France"
Similarité capitale/capitale : 0.86
Similarité capitale/fruit    : 0.809
Perplexité phrase connue  : 333.94
Perplexité phrase inconnue: 1284.91
```

15.9 Recherche documentaire et agents

Deux usages professionnels combinent ces briques :

- le **RAG** (*génération augmentée par recherche*) : on découpe ses documents en passages, on calcule leurs vecteurs, et pour chaque question on joint au modèle les passages les plus proches — il répond **à partir de vos documents** ;

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

- les **agents** : le modèle répond par un JSON qui désigne un **outil** (une procédure Lexis+ : calcul, requête SQL, météo…), le programme l'exécute et lui renvoie le résultat avec `c:ajouter("utilisateur", …)` puis `c:generer()`.

EXEMPLE Répondre à partir de documents internes (RAG) — principal.prg

```
modele = llm.charger("Llama-3.2-3B-Instruct-Q4_K_M.gguf")
index = {}
pour chaque nom dans fichier.liste("documents")
  locale texte = fichier.lire(fichier.joindre("documents", nom))
  inserer {source = nom, texte = texte, vecteur = modele:vecteur(texte)} dans index
suivant
locale conv = modele:conversation({systeme = "Réponds uniquement à partir des documents fournis. " +
  "Si l'information n'y figure pas, dis-le. Cite la source entre crochets.", maxtokens = 200})
pour chaque question dans {"Combien de jours de télétravail sont permis ?", "Quel est le salaire minimum ?"}
  locale v = modele:vecteur(question)
  trier(index, @PLUS_PROCHE)                    && trie par similarité avec la question
  locale pieces = {llm.joindretexte(index[1].source, index[1].texte),
                   llm.joindretexte(index[2].source, index[2].texte)}
  conv:reinitialiser()
  ? "Q :", question
  ? "R :", conv:demander(question, {pieces = pieces})
suivant

fonction PLUS_PROCHE(a, b)
retourner llm.similarite(a.vecteur, v) > llm.similarite(b.vecteur, v)
```

CE QUE L'ON OBTIENT

```
Q : Combien de jours de télétravail sont permis ?
   (passages : teletravail.txt, conges.txt)
R : Le télétravail est autorisé deux jours par semaine, le mardi et le jeudi
[teletravail.txt].

Q : Quel est le salaire minimum ?
R : Cette information ne figure pas dans les documents fournis.
```

15.10 Les fichiers GGUF et la quantification

Un fichier GGUF contient les **poids** du modèle (des millions ou milliards de nombres) et des **métadonnées** (architecture, tokeniseur, gabarit de conversation…). La **quantification** réduit la précision des poids (32 bits → 8, 5 ou 4 bits) : le fichier devient 4 à 8 fois plus petit et plus rapide, avec une perte de qualité souvent minime.

Appel	Rôle
`g = gguf.ouvrir(chemin)` ; `g:decrire([n])`, `g:cles()`, `g:existe(cle)`, `g:valeur(cle)`, `g:texte / entier / reel(cle, [defaut])`	métadonnées ; propriétés `version`, `chemin`, `alignement`, `architecture`
`g:tenseurs()`, `g:tenseur(nom, [max])`, `g:fermer()`	liste et valeurs des tenseurs

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Appel	Rôle
`gguf.quantifiermodele(entree, sortie, [type], [garder_sortie])`	re-quantifie un modèle (`f32` , `f16` , `bf16` , `q8_0` , `q6_k` , `q5_k` , `q4_k` , `q4_0` ...)
`gguf.fusionnerlora(base, adaptateur, sortie, [type])`	fusionne un adaptateur LoRA dans un modèle autonome
`gguf.types()` , `gguf.bitsparpoids(type)` , `gguf.quantifier(type, valeurs)`	types disponibles, bits par poids, essai de quantification

EXEMPLE Inspecter et quantifier un modèle — principal.prg

```
locale g = gguf.ouvrir("mini-assistant-f32.gguf")
? "Architecture :", g.architecture, "— version GGUF :", g.version
? "Nom :", g:texte("general.name"), "— couches :", g:entier("llama.block_count"), ;
  "— contexte :", g:entier("llama.context_length")
locale t = g:tenseurs()
? #t, "tenseurs ; le premier :", t[1].nom, "forme", json(t[1].dims), t[1].type
g:fermer()

? "Types :", joindre(gguf.types(), " ")
pour chaque type dans {"f16", "q8_0", "q4_k"}
  ? " ", type, ":", gguf.bitsparpoids(type), "bits par poids"
suivant
locale valeurs = {}
pour i = 1 a 64                                  && les blocs de quantification font 32 valeurs
  inserer sin(i / 5) dans valeurs
suivant
locale essai = gguf.quantifier("q4_0", valeurs)
? "Erreur moyenne en q4_0 :", arrondi(essai.erreur, 4), "— octets :", essai.octets

gguf.quantifiermodele("mini-assistant-f32.gguf", "mini-q4.gguf", "q4_k")
? "Taille f32 :", entier(fichier.taille("mini-assistant-f32.gguf") / 1024), "Ko → q4_k :", ;
  entier(fichier.taille("mini-q4.gguf") / 1024), "Ko"
```

▶ CE QU'AFFICHE LE PROGRAMME

```
Architecture : llama — version GGUF : 3
Nom : mini-assistant — couches : 4 — contexte : 128
38 tenseurs ; le premier : token_embd.weight forme [128,512] f32
Types : f32 f16 q4_0 q4_1 q5_0 q5_1 q8_0 q4_k q5_k q6_k bf16
  f16 : 16 bits par poids
  q8_0 : 8.5 bits par poids
  q4_k : 4.5 bits par poids
Erreur moyenne en q4_0 : 0.1235 — octets : 36
Taille f32 : 3154 Ko → q4_k : 1526 Ko
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

15.11 Dessiner : le paquet dessin

Appel	Rôle
`dessin.svg(code, [l, h, fond])`	rastérise un dessin SVG → image
`dessin.toile(l, h, [fond, lissage])`	surface de dessin anticrénelée ; `t:terminer()` → image
`t:rectangle`, `t:cercle`, `t:ellipse`, `t:polygone`, `t:ligne`, `t:polyligne`	formes
`dessin.image(l, h, [fond])`, `dessin.lire(chemin)`	image unie ; lecture BMP / PPM / PNG
`img:enregistrer(chemin)` (`.bmp`, `.ppm`), `img:pixel(x, y)`, `img:fixerpixel(x, y, c)`, `img:decrire([l])`	image ; propriétés `largeur`, `hauteur`
`dessin.couleur("#RRGGBB" ou r, g, b, [a])`, `dessin.taillesvg(code)`, `dessin.extrairesvg(texte)`	outils

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

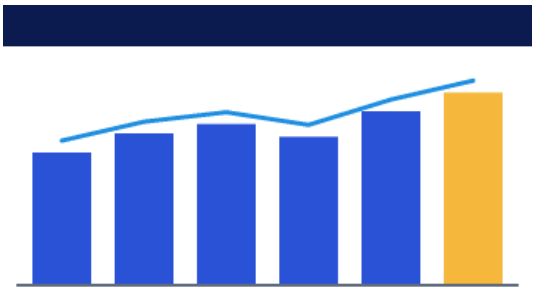
EXEMPLE Un graphique de ventes dessiné par programme — principal.prg

```
locale ventes = {42, 48, 51, 47, 55, 61}
locale t = dessin.toile(360, 200, "#FFFFFF")
t:rectangle(0, 0, 360, 28, "#0B1A4F")                    && bandeau
locale maxi = 70
pour i = 1 a #ventes
  locale h = ventes[i] / maxi * 150
  locale x = 20 + (i - 1) * 56
  t:rectangle(x, 190 - h, 40, h, sisinon(ventes[i] = 61, "#F5B83D", "#2A52D6"))
suivant
locale points = {}
pour i = 1 a #ventes
  inserer {40 + (i - 1) * 56, 190 - ventes[i] / maxi * 150 - 8} dans points
suivant
t:polyligne(points, 3, "#1E8FE0")
t:ligne(10, 190, 350, 190, 2, "#5F6B7F")
t:terminer():enregistrer("ventes.bmp")

&& Un logo à partir d'un code SVG
locale logo = dessin.svg([[<svg xmlns="http://www.w3.org/2000/svg" width="160"
height="90">
  <rect width="160" height="90" rx="18" fill="#0B1A4F"/>
  <circle cx="45" cy="45" r="22" fill="#F5B83D"/>
  <rect x="80" y="30" width="56" height="10" rx="5" fill="#FFFFFF"/>
  <rect x="80" y="50" width="38" height="10" rx="5" fill="#1E8FE0"/></svg>]])
logo:enregistrer("logo.bmp")
? "Logo :", logo.largeur, "x", logo.hauteur, "— pixel central :", json(logo:pixel(45, 45))
```

▶ **CE QU'AFFICHE LE PROGRAMME**

```
Logo : 160 x 90 — pixel central : {"R":245,"G":184,"B":61}
```



ventes.bmp



logo.bmp

15.12 Faire parler : le paquet audio

Appel	Rôle
`audio.parler(texte, [{langue="fr", hauteur=115, vitesse=1, frequence=16000}])`	synthèse vocale (fr, en) → son
`audio.melodie("C4 E4 G4:2 R", [bpm])`	mélodie (note + octave [: temps], `R` = silence)

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Appel	Rôle
`audio.nouveau([freq])`, `audio.lire(chemin)`, `audio.infos(chemin)`	son vide, lecture d'un WAV, en-tête
`s:silence(sec)`, `s:ajouter(autre)`, `s:normaliser([crete])`, `s:copier()`, `s:enregistrer(chemin)`	montage ; propriétés `duree`, `frequence`, `echantillons`
`audio.phonemes(texte, [langue])`, `audio.enlettres(n, [langue])`, `audio.frequencenote(note)`	outils

EXEMPLE Annonce vocale d'un numéro de commande — principal.prg

```
locale numero = 1984
locale annonce = audio.melodie("C5:0.5 E5:0.5 G5", 160)              && carillon
annonce:silence(0.3)
annonce:ajouter(audio.parler("Commande numéro " + audio.enlettres(numero) + ", prête au comptoir."))
annonce:normaliser(0.9)
annonce:enregistrer("annonce.wav")
locale i = audio.infos("annonce.wav")
? "Fichier :", i.frequence, "Hz,", i.canaux, "canal,", i.bits, "bits,", arrondi(i.duree, 1), "s"
? "En lettres :", audio.enlettres(numero), "/", audio.enlettres(numero, "en")
? "Phonèmes de « prête » :", joindre(audio.phonemes("prête"), " ")
? "La du diapason :", audio.frequencenote("A4"), "Hz"
```

▶ CE QU'AFFICHE LE PROGRAMME

```
Fichier : 22050 Hz, 1 canal, 16 bits, 5.5 s
En lettres : mille neuf cent quatre-vingt-quatre / one thousand nine hundred eighty four
Phonèmes de « prête » : p R E t |
La du diapason : 440 Hz
```

15.13 Apprendre : le paquet tenseur

Un **tenseur** est un tableau de nombres (vecteur, matrice…) qui **se souvient des calculs** qui l'ont produit. Après un calcul, `retropropager()` calcule automatiquement le **gradient** : dans quel sens modifier chaque paramètre pour réduire l'erreur. Un **optimiseur** applique ces corrections, étape après étape : c'est l'**apprentissage**.

Appel	Rôle
`tenseur.depuis(valeurs, [forme], [gradient])`, `tenseur.plein`, `tenseur.zeros`, `tenseur.scalaire`, `tenseur.aleatoire`, `tenseur.parametre(forme, [ecart])`	créer
`tenseur.lineaire(x, w, [b])`, `tenseur.produit(a, b)`, `tenseur.echelle(t, s)`, `+ - *`	calculer
`tenseur.relu`, `silu`, `sigmoide`, `tanh`, `gelu`, `softmax`, `concatener`, `decrochage`	fonctions de réseau de neurones

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Appel	Rôle
`tenseur.somme`, `moyenne`, `mse(pred, cible)`, `entropiecroisee(logits, classes)`	réductions et pertes
`t:retropropager()`, `t:gradient()`, `t:zerograd()`, `t:valeur([i])`, `t:donnees()`, `t:forme()`, `t:taille()`, `t:detacher()`, `t:remodeler(f)`	gradients et lecture
`tenseur.sgd(params, {lr, moment})`, `tenseur.adamw(params, {lr, …})` ; `o:zerograd()`, `o:etape()`, `o:ecreter(n)`, `o:enregistrer / charger`	optimiseurs
`tenseur.lrcosinus(...)`, `tenseur.normegradient(p)`, `tenseur.sansgradient(@p)`, `tenseur.verifiergradient(@p, t)`, `tenseur.operation(...)`, `tenseur.graine(n)`, `tenseur.hasard([max])`	outils

EXEMPLE Retrouver la règle d'un tarif à partir d'exemples — principal.prg

```
&& On observe des prix de livraison (kg → €) et on cherche la règle prix = a × kg + b
tenseur.graine(7)
locale kilos = {1, 2, 3, 5, 8, 10, 12, 15}
locale prix = {}
pour chaque k dans kilos
  inserer 1.5 * k + 4.9 + tenseur.hasard("gauss") * 0.1 dans prix   && règle cachée + bruit
suivant
locale x = tenseur.depuis(kilos, {8, 1})
locale y = tenseur.depuis(prix, {8, 1})
locale a = tenseur.parametre({1, 1}, 0.1, "a")
locale b = tenseur.plein({1}, 0, vrai, "b")
locale opt = tenseur.adamw({a, b}, {lr = 0.1, decroissance = 0})
pour etape = 1 a 400
  opt:zerograd()
  locale perte = tenseur.mse(tenseur.lineaire(x, a, b), y)
  perte:retropropager()                          && calcule les gradients
  opt:etape()                                    && corrige a et b
  si etape % 100 = 0
    ? "étape", etape, "erreur", chaine(perte:valeur(), 1, 4), " a =", chaine(a:valeur(), 1, 3), ;
      " b =", chaine(b:valeur(), 1, 3)
  finsi
suivant
? "Règle apprise : prix ≈", chaine(a:valeur(), 1, 2), "× kg +", chaine(b:valeur(), 1, 2), "€"
? "Prédiction pour 20 kg :", chaine(a:valeur() * 20 + b:valeur(), 1, 2), "€"
```

CE QU'AFFICHE LE PROGRAMME

```
étape 100 erreur 0.3153  a = 1.596  b = 4.038
étape 200 erreur 0.0075  a = 1.496  b = 4.984
étape 300 erreur 0.0075  a = 1.498  b = 4.978
étape 400 erreur 0.0083  a = 1.496  b = 4.975
Règle apprise : prix ≈ 1.50 × kg + 4.98 €
Prédiction pour 20 kg : 34.89 €
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

15.14 Créer et spécialiser un modèle : le paquet entrainement

Le paquet `entrainement` reproduit, en miniature, la fabrication des grands modèles : apprendre un **tokeniseur**, définir l'**architecture** (Transformer de type Llama), **pré-entraîner** sur du texte, **ajuster au dialogue**, puis **exporter** un vrai fichier GGUF utilisable par `llm.charger`, llama.cpp ou Ollama. Il sait aussi **spécialiser** un modèle existant avec un adaptateur **LoRA** (seule une petite partie des poids est entraînée).

Appel	Rôle
`entrainement.tokeniseur(textes, [{taille, speciaux, frequencemin, mode}])`, `entrainement.tokeniseuroctets()`	apprendre un tokeniseur BPE
`entrainement.nouveau(tok, {dimension, couches, tetes, tetes_kv, contexte, nom, graine})`	nouveau modèle
`entrainement.charger(chemin, [{mode="lora", rang, alpha, cibles}])`	modèle existant à spécialiser
`entrainement.textes(tok, texte)`, `entrainement.fichiers(tok, chemins)`	données de pré-entraînement
`entrainement.dialogues(tok, gabarit, dialogues)`, `entrainement.jsonl(chemin, tok)`	données de dialogue (seules les réponses sont apprises)
`entrainement.gabarit(tok)`, `entrainement.configuration()`	gabarit adapté, réglages par défaut
`m:entrainer(donnees, {etapes, lot, sequence, lr, echauffement, evaluation, validation, rappel…})`	entraîne ; renvoie l'optimiseur
`m:evaluer(d)`, `m:generer(amorce)`, `m:parametres()`, `m:nombreparametres()`, `m:optimiseur()`	suivi
`m:enregistrer(chemin, [type])`, `m:enregistrerlora(chemin)`, `m:sauvegarder(opt, prefixe)`	export GGUF, adaptateur, point de reprise
`d:separer([fraction])`, `d:taille()`, `d:pluslong()`	données : validation, taille, plus long dialogue

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

EXEMPLE Créer le mini-assistant de ce chapitre (extrait de l'exemple 17) — principal.prg

```
llm.fils(0)
dialogues = {}
pour chaque pc dans {{"France", "Paris"}, {"Grèce", "Athènes"}, {"Suède", "Stockholm"}}
  QA("Quelle est la capitale de la " + pc[1] + " ?", "La capitale de la " + pc[1] + " est " + pc[2] + ".")
suivant
&& … (capitales, fruits, jours, additions : 160 dialogues en tout)
locale tok = entrainement.tokeniseur(echantillon, {taille = 512})
locale modele = entrainement.nouveau(tok, {dimension = 128, couches = 4, tetes = 4, tetes_kv = 2,
                                           contexte = 128, nom = "mini-assistant", graine = 7})
? "Paramètres :", modele:nombreparametres()
modele:entrainer(entrainement.textes(tok, repliquer(texte, 4)), {etapes = 150, lot = 8, sequence = 64, lr = 3e-3})
locale sft = entrainement.dialogues(tok, "chatml", dialogues)
locale parts = sft:separer(0.05)
modele:entrainer(parts[1], {etapes = 900, lot = 8, sequence = sft:pluslong(), lr = 2e-3, validation = parts[2]})
modele:enregistrer("mini-assistant-f32.gguf")
gguf.quantifiermodele("mini-assistant-f32.gguf", "mini-assistant-q8.gguf", "q8_0")

procedure QA(q, r)
  inserer {question = q, reponse = r} dans dialogues
retourner
```

▶ **CE QU'A AFFICHÉ L'EXEMPLE COMPLET (EXÉCUTION RÉELLE : 159 SECONDES SUR 2 CŒURS)**

```
Tokeniseur : 512 tokens
Paramètres : 803968

== Pré-entraînement ==
étape     1 | perte 6.2052 | lr 2.00e-04 | ||g|| 4.65 | 1695 tok/s
étape   150 | perte 0.0423 | lr 1.00e-04 | ||g|| 0.17 | 2511 tok/s

== Ajustement au dialogue (149 exemples) ==
étape     1 | perte 8.6491 | lr 1.33e-04 | ||g|| 40.52 | 3493 tok/s
étape   150 | perte 0.5528 | lr 1.89e-03 | ||g|| 1.42 | 3322 tok/s
étape   300 | perte 0.7358 | val 0.9596 | lr 1.56e-03 | ||g|| 1.09 | 3112 tok/s
étape   600 | perte 0.4846 | val 1.0544 | lr 5.93e-04 | ||g|| 0.70 | 3257 tok/s
étape   900 | perte 0.4851 | val 0.9845 | lr 1.00e-04 | ||g|| 1.31 | 3448 tok/s
Durée totale : 159 s

Modèle exporté : mini-assistant-f32.gguf et mini-assistant-q8.gguf
Vous : Bonjour !
IA   : Bonjour ! Comment puis-je t'aider ?
Vous : Quelle est la capitale de la Grèce ?
IA   : La capitale de la Grèce est Athènes.
Vous : Combien font 3 plus 4 ?
IA   : 3 plus 4 font 7.
Vous : Quel jour vient après jeudi ?
IA   : Après jeudi vient vendredi.
Vous : De quelle couleur est une fraise ?
IA   : Une fraise est rouge.
Vous : Qui es-tu ?
IA   : Je suis un tout petit modèle de langage écrit en Nim.
```


© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

À retenir

- `m = llm.charger("modele.gguf")`, `c = m:conversation({…})`, `c:demander(q)` → réponse (`texte`, `tokensgeneres`, `vitesse`…).
- Réglages : `llm.glouton()`, `{temperature, topk, topp, graine}`, `c.maxtokens`, `c.arrets`, `c.systeme` ; flux : `c:demander(q, @AFFICHER)`.
- `demanderjson(q, schema)`, pièces jointes `{pieces = {llm.joindre(…)}}`, formats `json`, `image`, `audio`, `fichier`.
- Tokens, logits, `m:vecteur`, `llm.similarite` (RAG) ; `gguf.ouvrir`, `gguf.quantifiermodele`.
- `dessin.toile / svg`, `audio.parler / melodie`, `tenseur` + optimiseurs, `entrainement` → modèle GGUF.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

L+ CHAPITRE 16 · LEXIS+

Aller plus loin

Écrire vos propres paquets en Nim, porter vos programmes vers le serveur wdGestion V, et adopter les bonnes habitudes qui font les applications durables.



16.1 Écrire un paquet en Nim
16.2 Du poste au serveur wdGestion V
16.3 Organiser une application réelle
16.4 Bonnes pratiques
16.5 Pour aller plus loin

Dans ce chapitre, vous apprendrez à…

- ✔ comprendre comment un paquet s'intègre à Lexis+
- ✔ écrire un paquet simple en Nim
- ✔ connaître les différences entre poste et serveur
- ✔ organiser une application de taille réelle
- ✔ adopter de bonnes pratiques de programmation
- ✔ savoir où trouver de l'aide

16.1 Écrire un paquet en Nim

Lexis+ est écrit en **Nim**, un langage compilé rapide et lisible. Tous les paquets que vous avez utilisés (`fichier`, `http`, `llm`…) sont de petits modules Nim qui s'**enregistrent** auprès de l'interpréteur. Vous pouvez faire de même pour exposer à Lexis+ une bibliothèque de votre entreprise, un calcul spécialisé ou un matériel.

Un paquet est un module placé dans `src/paquets/`, qui :

1. importe l'API des paquets : `import ../lexis/api` ;
2. définit une procédure d'ouverture qui construit une table de fonctions (`newLib`, `m.fn "nom", proc …`) ;
3. s'enregistre avec `registerPackage PackageDef(name: …, open: …, docs: …)` ;
4. est ajouté à `paquets/tous.nim`.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

NIM paquets/pkg_tva.nim : un paquet de calculs de TVA pkg_tva.nim

```nim
import ../lexis/api

proc ouvrirTva(L: Interp): Value =
  let m = newLib()
  m.fn "ttc", proc (L: Interp, a: seq[Value]): seq[Value] =
    let ht = L.checkNum(a, 1, "ttc")                  # argument 1 obligatoire
    let taux = L.optNum(a, 2, "ttc", 20.0)            # argument 2 facultatif
    ret numV(ht * (1 + taux / 100))
  m.fn "taux", proc (L: Interp, a: seq[Value]): seq[Value] =
    ret record({"normal": numV(20), "intermediaire": numV(10), "reduit": numV(5.5)})
  tableV(m)

registerPackage PackageDef(name: "tva", open: ouvrirTva,
  description: "Calculs de TVA",
  docs: docs("tva", [
    "tva.ttc(ht, [taux])", "Montant TTC (taux 20 % par défaut).",
    "tva.taux()", "Les taux en vigueur : {normal, intermediaire, reduit}."]))
```

Après recompilation de l'atelier et du runtime, `tva.ttc(100)` fonctionne dans tout programme ; la complétion, l'aide et la coloration de l'atelier connaissent le paquet, grâce aux `docs` enregistrées.

L'API des paquets (`lexis/api.nim`) fournit l'essentiel :

Outil	Rôle
`checkStr`, `checkInt`, `checkNum`, `checkTable`, `checkFunc`, `checkAny`	vérifier un argument (message d'erreur clair sinon)
`optStr`, `optInt`, `optNum`, `optBool`, `optField`	arguments et options facultatifs
`strV`, `numV`, `intV`, `boolV`, `tableV`, `record`, `toV`, `nilV`	fabriquer des valeurs Lexis+
`ret(...)`, `fail(message)`	renvoyer des valeurs ; signaler un échec (erreur interceptable par `essayer`)
`wrap(L, objet, nom, methodes)`, `checkUser`	objets natifs avec méthodes (`obj:methode()`)
`L.blocking:`	libérer l'interpréteur pendant une attente (réseau, calcul long)
`L.call(f, args)`, `L.callable(v)`	rappeler une procédure Lexis+ (gestionnaires, rappels)
`global: true`	paquet installé comme variable globale au démarrage (comme `es` et `systeme`)

Bibliothèques chargées à l'exécution

Si votre paquet s'appuie sur une bibliothèque C (`.dll`, `.so`, `.dylib`), chargez-la **à l'exécution** avec `std/dynlib`, comme le font DuckDB, Firebird et webview : l'atelier démarre alors même sans la bibliothèque, et seul le programme qui s'en sert signale son absence.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

16.2 Du poste au serveur wdGestion V

Lexis+ est d'abord le langage du serveur **wdGestion V**, qui exécute des programmes `.prg` en réponse aux requêtes des navigateurs. Les programmes écrits sur le poste s'y exécutent **tels quels**, à quelques différences près :

Sur le serveur wdGestion V	Sur le poste (wdlexis+)
fichiers `.prg` du dossier `PROG`, appelés par le navigateur	fichiers `.prg` du projet ; programme principal en tête de `principal.prg`
`SESSION` : la mémoire du visiteur	`SESSION` : les variables globales du programme
pages web : `envoyer`, `inclure`, `rediriger`, `statut`, `entete`, `deconnecter`, `REQUETE`	non disponibles (erreur de syntaxe explicite)
base configurée dans le JSON du serveur	`donnees.duckdb` du dossier du programme, ou `ouvrirbase()`
limites d'instructions et de durée par requête	pas de limite (bouton **Arrêter**)
—	paquets (interface, fichiers, réseau, IA…), `saisir()`, appels `f{...}`, itérateurs

À retenir

Écrivez la **logique métier** (calculs, règles, accès aux données) dans des procédures indépendantes de l'affichage : elles fonctionneront à l'identique dans un programme console, une fenêtre, une page webview… et sur le serveur.

16.3 Organiser une application réelle

```
mon_application/
  principal.prg          démarrage : ouvre la base, la fenêtre, lance la boucle
  config.json            réglages (lus avec json.lire)
  donnees.duckdb         la base
  metier/
    clients.prg          procédures CLIENT_CREER, CLIENT_CHERCHER…
    factures.prg         FACTURE_CALCULER, FACTURE_ENREGISTRER…
  ecrans/
    ecran_clients.prg    construction de la fenêtre clients et gestionnaires
  outils/
    format.prg           EUROS(), DATE_FR(), COLONNE()…
  fils/
    import.prg           procédure PRINCIPAL d'un fil d'import (fil.demarrer)
  tests/
    tests_factures.prg   procédure TEST_FACTURES() appelée à la demande
```

- Préfixez les procédures par leur domaine (`CLIENT_`, `FACTURE_`) : on s'y retrouve dans le panneau **Structure** et on évite les doublons de noms.
- Gardez le programme principal **court** : il assemble les morceaux.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

- Prévoyez des **procédures de test** qui vérifient vos calculs avec des valeurs connues ; lancez-les après chaque modification.

16.4 Bonnes pratiques

Pratique	Pourquoi
`locale` pour toutes les variables de procédure	évite les collisions de globales, la première source de bogues
des noms parlants	le code se lit comme une explication
paramètres `?` en SQL, jamais de concaténation	sécurité (injection SQL) et apostrophes
`hachermdp` / `verifiermdp` pour les mots de passe	ne jamais stocker un mot de passe en clair
secrets dans l'environnement (`systeme.env`)	un programme copié ne divulgue rien
`essayer … capturer` aux bons endroits, transactions	des erreurs maîtrisées, des données cohérentes
`html()` et `url()` pour tout texte venu de l'utilisateur	évite l'injection de code dans les pages
des procédures courtes, une tâche chacune	lisibles, testables, réutilisables
des commentaires qui disent **pourquoi**	le code dit déjà comment
le débogueur plutôt que des suppositions	on voit ce qui se passe vraiment

16.5 Pour aller plus loin

- La documentation de référence `DOCUMENTATION.md` livrée avec Lexis+, et le panneau **Référence** de l'atelier (double-clic pour insérer un appel).
- Les exemples livrés : `exemples/bonjour`, `carnet`, `gestion`, `donnees`, `reseau`, `taches`, `objet`, `firebird`, `webview/…` et les vingt exemples d'IA `exemples/llm/01_bonjour` à `20_reprise_entrainement`.
- Les aides-mémoires et le poster en annexe de ce guide, également disponibles en documents séparés.
- Contact : metalab (at) sonaliwan.fr — sonaliwan.fr.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

ANNEXE A

Référence complète

Toutes les instructions, fonctions, méthodes et propriétés, domaine par domaine, avec leur description intégrale.



© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr



Référence · Démarrer

Écrire, exécuter, déboguer et lancer un programme. — 21 entrées.

Afficher et saisir

Instruction, fonction ou méthode	Rôle
`? v1, v2, …`	affiche, séparé par des espaces, puis passe à la ligne
`?? v1, v2, …`	affiche sans passer à la ligne
`? (seul)`	passe à la ligne
`saisir([invite])`	Lit une ligne saisie (console de l'atelier ou terminal) ; nul en fin d'entrée.
`paquet(nom)`	Charge un paquet : interface, fichier, json, csv, http, reseau, tache…

Projet

Instruction, fonction ou méthode	Rôle
`principal.prg`	fichier principal : son en-tête est le programme principal
`autres .prg`	procédures disponibles dans tout le projet (sous-dossiers compris)
`procedure PRINCIPAL`	exécutée si le fichier principal ne contient que des procédures
`finprocedure / finfonction`	fin explicite d'une procédure : la suite revient au programme principal

Atelier wdlexis+

Instruction, fonction ou méthode	Rôle
`Ctrl+R`	exécuter
`F5`	déboguer / continuer
`F9`	point d'arrêt
`F11 · F10 · Maj+F11`	pas à pas · pas principal · sortir
`Maj+F5`	arrêter
`Ctrl+Espace`	complétion
`Ctrl+B`	commenter
`F12`	aller à la définition

Runtime wdlexisrt

Instruction, fonction ou méthode	Rôle
`wdlexisrt prog.prg [args]`	exécute
`--verifier prog.prg`	vérifie la syntaxe
`-e "code"`	exécute une ligne
`--paquets`	liste les paquets

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr



Référence · Les bases

Instructions, commentaires, valeurs, opérateurs et variables. — 32 entrées.

Lignes et commentaires

Instruction, fonction ou méthode	Rôle
`une instruction par ligne`	fin de ligne = fin d'instruction
`… ;`	continue sur la ligne suivante
`&& · // · *`	commentaires
`casse`	noms insensibles à la casse ; mots-clés en minuscules
`à = a`	accent facultatif

Valeurs

Instruction, fonction ou méthode	Rôle
`nul`	absence de valeur
`vrai faux .v. .f.`	logiques
`42 3.14 1e3`	nombres
`"…" '…' [[…]]`	chaînes (\n \t \" \\)
`{…}`	table
`@NOM`	référence de procédure

Opérateurs

Instruction, fonction ou méthode	Rôle
`+ - * / % ^ **`	calcul
`+ (avec texte) ..`	concaténation
`= ==`	égal
`<> != #`	différent
`< <= > >=`	comparaison
`$`	inclusion de texte
`et ou non`	logique (renvoient une des valeurs)
`#v`	longueur

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Variables

Instruction, fonction ou méthode	Rôle
`locale x [= v], y`	variable locale
`globale x`	globale explicite
`x = v`	affectation (nom non déclaré = globale)
`x += v · x -= v`	ajouter, retirer
`stocker v dans a, b`	même valeur partout
`liberer x`	remet à nul
`SESSION`	table des variables globales
`m->x`	la variable x (et non un champ)

Types et conversions

Instruction, fonction ou méthode	Rôle
`type(v)`	"nul", "logique", "nombre", "chaine", "table", "procedure", "objet" ou "tache".
`chaine(v, [largeur], [decimales])`	Texte d'une valeur ; avec largeur : nombre aligné à droite (STR).
`texte(v)`	Texte d'une valeur (utilise __tostring).
`valeur(s)`	Nombre d'un texte (virgule décimale admise), 0 si impossible (VAL).
`nombre(s)`	Nombre d'un texte, nul si impossible.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr



Référence · Décider & répéter

Choisir un chemin, répéter un traitement. — 16 entrées.

Conditions

Instruction, fonction ou méthode	Rôle
`si cond … finsi`	exécute si la condition est vraie
`sinonsi cond`	autre condition (ou « sinon si »)
`sinon`	aucune condition vraie
`faire cas … fincas`	choix multiple
`cas cond`	une branche
`autrement`	tous les autres cas
`sisinon(condition, si_vrai, si_faux)`	Valeur selon la condition ; seule la branche choisie est calculée (IIF).
`vide(v)`	Vrai pour nul, faux, 0, une chaîne blanche ou une table sans élément (EMPTY).
`estnul(v)`	Vrai si v vaut nul.

Boucles

Instruction, fonction ou méthode	Rôle
`pour i = a a b [pas p] … suivant`	compteur (a / à / jusqua ; finpour)
`pour chaque v dans liste … suivant`	chaque valeur
`pour chaque k, v dans fiche`	chaque clef et valeur
`pour chaque v dans @ITERATEUR`	procédure appelée jusqu'à nul
`tantque cond … fintantque`	tant que (ou faire tantque)
`sortir`	quitte la boucle
`boucler`	passe au tour suivant

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr



Référence · Procédures

Nommer un traitement pour le réutiliser partout. — 13 entrées.

Définir

Instruction, fonction ou méthode	Rôle
`procedure NOM(a, b)`	début d'une procédure
`fonction NOM(a, b)`	synonyme
`parametres a, b`	lire les arguments (style dBase)
`retourner [valeur]`	fin et résultat (aussi « retour »)
`finprocedure · finfonction`	fin explicite
`argument manquant`	vaut nul : x = x ou défaut

Appeler

Instruction, fonction ou méthode	Rôle
`NOM(a, b)`	appel
`x = NOM(a, b)`	appel avec résultat
`faire NOM avec a, b`	appel style dBase
`@NOM`	référence (sans appeler)
`ref(a, b)`	appel d'une référence rangée dans une variable
`appeler(@proc, ...)`	Appelle une procédure référencée.
`récursivité`	une procédure s'appelle elle-même (200 niveaux au plus)

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr



Référence · Tables & objets

Listes, fiches, objets et métatables. — 33 entrées.

Créer et accéder

Instruction, fonction ou méthode	Rôle
`{a, b, c}`	liste (indices à partir de 1)
`{cle = v}`	fiche
`t[i] · t.cle · t["CLE"]`	accès
`nouvelle table t`	t = {}
`supprimer table t`	t = nul
`#t`	longueur de la liste

Modifier

Instruction, fonction ou méthode	Rôle
`inserer v dans t [position p]`	ajoute
`inserer v dans t clef "K"`	ajoute sous une clef
`retirer de t position p dans x`	enlève
`inserer(t, v, [position])`	Ajoute v (à la fin ou à la position) ; renvoie t.
`retirer(t, [position])`	Enlève et renvoie un élément (le dernier par défaut).
`trier(t, [@comparer])`	Trie la liste sur place ; renvoie t.
`copier(t)`	Copie de premier niveau.

Interroger

Instruction, fonction ou méthode	Rôle
`taille(t)`	Nombre d'éléments de la partie liste (#t).
`nbelements(t)`	Nombre total d'éléments.
`clefs(t)`	Liste des clefs.
`valeurs(t)`	Liste des valeurs.
`aclef(t, k)`	Vrai si la clef existe.
`contient(s, sous)`	Vrai si s contient sous ; avec une table : vrai si la valeur y figure.
`joindre(liste, [separateur])`	Texte des éléments d'une liste.
`decouper(s, [separateur])`	Liste des morceaux (séparateur « , » par défaut).

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Instruction, fonction ou méthode	Rôle
`json(v)`	Texte JSON d'une valeur.
`dejson(s)`	Valeur d'un texte JSON (clefs d'objets en majuscules).

Objets et métatables

Instruction, fonction ou méthode	Rôle
`Classe.__index = Classe`	méthodes partagées
`Classe:procedure P comme m`	range une méthode
`obj:m(…)`	appel de méthode (obj en premier argument)
`fixermetatable t, mt`	forme instruction
`fixermetatable(t, mt)`	Donne une métatable ; renvoie t.
`obtenirmetatable(t)`	La métatable ou nul.
`lirebrut(t, k)`	Lecture sans métaméthode.
`ecrirebrut(t, k, v)`	Écriture sans métaméthode ; renvoie t.
`__index __newindex __call __tostring __len`	accès, appel, texte, longueur
`__eq __lt __le __add __sub __mul __div __mod __pow __unm __concat`	opérateurs

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr



Référence · Texte & calcul

La boîte à outils toujours disponible : texte, nombres, dates, sécurité. — 55 entrées.

Texte

Instruction, fonction ou méthode	Rôle
`majuscule(s)`	Texte en majuscules (accents compris).
`minuscule(s)`	Texte en minuscules.
`rogner(s)`	Retire les espaces aux deux bouts (ALLTRIM).
`rognerg(s)`	Retire les espaces à gauche (LTRIM).
`rognerd(s)`	Retire les espaces à droite (RTRIM).
`longueur(s)`	Nombre de caractères (ou d'éléments d'une liste).
`souschaine(s, debut, [n])`	Extrait n caractères à partir de debut (SUBSTR) ; positions à partir de 1.
`gauche(s, n)`	Les n premiers caractères.
`droite(s, n)`	Les n derniers caractères.
`position(sous, s)`	Position de sous dans s, 0 si absente (AT).
`espace(n)`	n espaces (SPACE).
`repliquer(s, n)`	s répété n fois (REPLICATE).
`remplacertexte(s, a, b)`	Remplace toutes les occurrences de a par b (STRTRAN).
`contient(s, sous)`	Vrai si s contient sous ; avec une table : vrai si la valeur y figure.
`commencepar(s, debut)`	Vrai si s commence par debut.
`finitpar(s, fin)`	Vrai si s finit par fin.
`decouper(s, [separateur])`	Liste des morceaux (séparateur « , » par défaut).
`joindre(liste, [separateur])`	Texte des éléments d'une liste.
`chaine(v, [largeur], [decimales])`	Texte d'une valeur ; avec largeur : nombre aligné à droite (STR).
`html(s)`	Échappe < > & " ' pour le HTML.
`url(s)`	Encode un texte pour une URL.
`car(code, ...)`	Caractères à partir de leurs codes Unicode.
`code(s, [i])`	Code Unicode du i-ème caractère.

Nombres

Instruction, fonction ou méthode	Rôle
`abs(x)`	Valeur absolue.
`entier(x)`	Partie entière, vers zéro (INT).
`arrondi(x, [decimales])`	Arrondi (0 décimale par défaut).
`plancher(x)`	Entier inférieur.
`plafond(x)`	Entier supérieur.
`racine(x)`	Racine carrée.
`mod(a, b)`	Reste de la division, du signe du diviseur.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Instruction, fonction ou méthode	Rôle
`max(a, b, ...)`	Plus grande valeur (nombres ou chaînes).
`min(a, b, ...)`	Plus petite valeur.
`aleatoire([n])`	Réel entre 0 et 1, ou entier entre 1 et n.
`sin(x)`	Sinus (radians).
`cos(x)`	Cosinus.
`tan(x)`	Tangente.
`asin(x)`	Arc sinus.
`acos(x)`	Arc cosinus.
`atan(y, [x])`	Arc tangente (atan2 avec deux arguments).
`exp(x)`	Exponentielle.
`log(x, [base])`	Logarithme (népérien par défaut).
`pi()`	π.

Dates

Instruction, fonction ou méthode	Rôle
`date()`	Date du jour, AAAA-MM-JJ.
`heure()`	Heure, HH:MM:SS.
`maintenant()`	Date et heure.
`horodatage()`	Secondes depuis le 1er janvier 1970 (avec décimales).

Tests

Instruction, fonction ou méthode	Rôle
`type(v)`	"nul", "logique", "nombre", "chaine", "table", "procedure", "objet" ou "tache".
`estnul(v)`	Vrai si v vaut nul.
`vide(v)`	Vrai pour nul, faux, 0, une chaîne blanche ou une table sans élément (EMPTY).
`sisinon(condition, si_vrai, si_faux)`	Valeur selon la condition ; seule la branche choisie est calculée (IIF).

Sécurité

Instruction, fonction ou méthode	Rôle
`hachage(s)`	Empreinte SHA-256 en hexadécimal (pas pour les mots de passe).
`hachermdp(mdp)`	Empreinte PBKDF2-SHA256 salée d'un mot de passe.
`verifiermdp(mdp, empreinte)`	Vérifie un mot de passe (temps constant).
`mdpaleatoire([n])`	Mot de passe aléatoire (16 caractères par défaut).
`uuid()`	Identifiant unique aléatoire.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr



Référence · Erreurs

Signaler, intercepter et traiter les erreurs. — 12 entrées.

Intercepter

Instruction, fonction ou méthode	Rôle
`essayer`	début du bloc surveillé
`capturer [e]`	e reçoit l'erreur (texte ou table)
`finessayer`	fin du bloc

Signaler

Instruction, fonction ou méthode	Rôle
`erreur "message"`	lève une erreur
`erreur {code = …, message = …}`	erreur structurée
`relancer`	erreur ... dans capturer
`systeme.quitter(code)`	arrêt volontaire

Messages fréquents

Instruction, fonction ou méthode	Rôle
`division par zéro`	diviseur nul
`indexation d'une valeur nulle`	t vaut nul
`procédure ou fonction inconnue`	faute de frappe, fichier absent
`nombre attendu, chaine reçu`	calcul sur un texte
`profondeur d'appel maximale`	récursivité sans fin

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr



Référence · Bases de données

DuckDB intégré (SQL et zones dBase), SQLite, Firebird. — 60 entrées.

Base DuckDB

Instruction, fonction ou méthode	Rôle
`ouvrirbase(chemin)`	Ouvre une base DuckDB (":memoire:" pour une base en mémoire) ; par défaut donnees.duckdb du dossier du programme.
`fermerbase()`	Ferme la base et les zones.
`base()`	Chemin de la base ouverte ("" si aucune).
`requete(sql, ...)`	Liste des lignes d'une requête DuckDB (paramètres « ? »).
`valeursql(sql, ...)`	Première valeur de la première ligne, ou nul.
`sql "…" [avec v, …] [dans liste]`	exécute du SQL (paramètres ?)
`begin transaction · commit · rollback`	transactions (via sql)

Zones (dBase)

Instruction, fonction ou méthode	Rôle
`utiliser t [alias a] [ordre "…"] [filtre "…"]`	ouvre une zone (USE)
`selectionner a`	zone courante (SELECT)
`fermer [a | tout]`	ferme (CLOSE)
`aller haut | bas | n`	se positionner (GO)
`sauter [n]`	avancer (SKIP)
`chercher v`	recherche sur l'ordre (SEEK)
`localiser pour cond`	LOCATE
`continuer`	suivant (CONTINUE)
`ajouter [vide | avec c par v, …]`	APPEND
`remplacer c par v, …`	REPLACE
`effacer`	DELETE
`parcourir [pour cond] … finparcourir`	SCAN
`compter [pour cond] dans x`	COUNT
`rafraichir`	relit la zone
`alias->champ · m->var`	champ d'une zone · variable
`fdf([alias])`	Fin de fichier de la zone (EOF).
`ddf([alias])`	Début de fichier (BOF).
`numenr([alias])`	Numéro de l'enregistrement courant (RECNO).
`nbenr([alias])`	Nombre d'enregistrements (RECCOUNT).
`trouve([alias])`	Résultat du dernier chercher / localiser (FOUND).
`alias()`	Alias de la zone courante.
`champ(nom, [alias])`	Valeur d'un champ par son nom (FIELD).

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Instruction, fonction ou méthode	Rôle
`enregistrement([alias])`	L'enregistrement courant sous forme de table.

SQLite

Instruction, fonction ou méthode	Rôle
`sqlite.ouvrir(chemin)`	Ouvre (ou crée) une base ; ":memoire:" pour une base en mémoire.
`sqlite.version()`	Version de la bibliothèque SQLite.
`db:executer(sql, ...)`	Exécute une instruction avec paramètres « ? » ; renvoie le nombre de lignes modifiées. Sans paramètre, accepte un script d'instructions séparées par « ; » (hors blocs PSQL).
`db:requete(sql, ...)`	Liste des lignes ; chaque ligne est une table colonne (majuscules) → valeur. Accepte INSERT ... RETURNING et EXECUTE PROCEDURE.
`db:premier(sql, ...)`	Première ligne ou nul.
`db:lignes(sql, ...)`	Itérateur sur les lignes : pour chaque r dans db:lignes(sql) ... suivant.
`db:dernierid()`	Identifiant de la dernière ligne insérée.
`db:modifications()`	Lignes modifiées par le dernier executer().
`db:transaction(@proc)`	Appelle proc(db) dans une transaction (annulée en cas d'erreur).
`db:fermer()`	Ferme la connexion (une transaction explicite non validée est annulée).

Firebird

Instruction, fonction ou méthode	Rôle
`firebird.connecter(base, utilisateur, motdepasse[, options])`	Connexion : "serveur:/chemin/base.fdb", "serveur/3050:alias" ou fichier local. Accepte aussi une table {base=, hote=, port=, utilisateur=, motdepasse=, role=, jeu="UTF8", isolation=, attente=}.
`firebird.creer(base, utilisateur, motdepasse[, {taillepage=8192, jeu="UTF8"}])`	Crée une base puis s'y connecte.
`firebird.version()`	Version de la bibliothèque cliente.
`firebird.disponible()`	Vrai si la bibliothèque cliente est trouvée.
`db:executer(sql, ...)`	Exécute une instruction avec paramètres « ? » ; renvoie le nombre de lignes modifiées. Sans paramètre, accepte un script d'instructions séparées par « ; » (hors blocs PSQL).
`db:requete(sql, ...)`	Liste des lignes ; chaque ligne est une table colonne (majuscules) → valeur. Accepte INSERT ... RETURNING et EXECUTE PROCEDURE.
`db:premier(sql, ...)`	Première ligne ou nul.
`db:valeur(sql, ...)`	Première colonne de la première ligne ou nul.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Instruction, fonction ou méthode	Rôle
`db:lignes(sql, ...)`	Itérateur sur les lignes : pour chaque r dans db:lignes(sql) ... suivant.
`db:modifications()`	Lignes modifiées par le dernier executer().
`db:sequence(nom[, pas=1])`	Valeur suivante d'une séquence (générateur).
`db:transaction(@proc[, options])`	Appelle proc(db) dans une transaction (validée, ou annulée en cas d'erreur).
`db:debut([{isolation="lecturevalidee"|"instantane"|"serialisable", lecture=faux, attente=vrai|faux|secondes}])`	Commence une transaction explicite.
`db:valider()`	Valide la transaction explicite.
`db:annuler()`	Annule la transaction explicite.
`db:entransaction()`	Vrai si une transaction explicite est en cours.
`db:tables()`	Tables de l'utilisateur.
`db:colonnes(table)`	Colonnes d'une table : {nom=, type=, longueur=, echelle=, obligatoire=}.
`db:serveur()`	Version du serveur.
`db:fermer()`	Ferme la connexion (une transaction explicite non validée est annulée).

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr



Référence · Fichiers & formats

Lire et écrire fichiers, dossiers, CSV, JSON et binaire. — 54 entrées.

fichier

Instruction, fonction ou méthode	Rôle
`fichier.lire(chemin)`	Contenu complet d'un fichier (texte ou binaire).
`fichier.lireoctets(chemin)`	Contenu binaire (les chaînes sont des suites d'octets).
`fichier.ecrire(chemin, donnees)`	Remplace le contenu d'un fichier.
`fichier.ecrireoctets(chemin, donnees)`	Écrit des octets.
`fichier.ajouter(chemin, donnees)`	Ajoute à la fin d'un fichier.
`fichier.lignes(chemin)`	Liste des lignes d'un fichier.
`fichier.ecrirelignes(chemin, lignes)`	Écrit une liste de lignes.
`fichier.existe(chemin)`	Vrai si le fichier ou le dossier existe.
`fichier.estfichier(chemin)`	Vrai pour un fichier.
`fichier.estdossier(chemin)`	Vrai pour un dossier.
`fichier.taille(chemin)`	Taille en octets.
`fichier.liste([dossier], [recursif])`	Noms contenus dans un dossier (les dossiers finissent par /).
`fichier.creerdossier(chemin)`	Crée un dossier (et ses parents).
`fichier.supprimer(chemin)`	Supprime un fichier ou un dossier (récursif).
`fichier.copier(source, destination)`	Copie (dossiers récursivement).
`fichier.deplacer(source, destination)`	Déplace ou renomme.
`fichier.infos(chemin)`	Table {taille, genre ("fichier" ou "dossier"), modifie, cree} ou nul.
`fichier.courant()`	Dossier courant.
`fichier.changer(dossier)`	Change de dossier courant.
`fichier.joindre(a, b, ...)`	Assemble des chemins.
`fichier.nom(chemin)`	Nom du fichier.
`fichier.dossier(chemin)`	Dossier parent.
`fichier.extension(chemin)`	Extension.
`fichier.absolu(chemin)`	Chemin absolu.
`fichier.dossiertemp()`	Dossier temporaire.
`fichier.dossierperso()`	Dossier personnel.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

es

Instruction, fonction ou méthode	Rôle
`es.ecrire(...)`	Écrit les valeurs (sans séparateur ni fin de ligne) sur la sortie : console de l'atelier ou terminal.
`es.lire([format])`	Lit l'entrée : "l" une ligne (défaut), "n" un nombre, "L" une ligne avec sa fin ; nul en fin d'entrée.
`es.lignes([chemin])`	Itérateur sur les lignes d'un fichier (ou de l'entrée) : pour chaque l dans es.lignes(chemin).
`es.ouvrir(chemin, [mode])`	Ouvre un fichier : "l" lecture (défaut), "e" écriture, "a" ajout, "l+", "e+".
`es.type(v)`	"fichier", "fichier fermé" ou nul.
`es.sortie`	Sortie standard.
`es.erreur`	Sortie d'erreur.
`es.entree`	Entrée standard.
`f:lire([format])`	Lit : "l", "L", "n", "t" (tout) ou un nombre d'octets ; nul en fin de fichier.
`f:ecrire(...)`	Écrit des valeurs ; renvoie le fichier.
`f:lignes()`	Itérateur sur les lignes restantes.
`f:position([origine], [decalage])`	Déplace la position ("debut", "courant", "fin") et la renvoie.
`f:vider()`	Vide les tampons.
`f:fermer()`	Ferme le fichier.

csv

Instruction, fonction ou méthode	Rôle
`csv.analyser(texte, [options])`	Liste des lignes d'un texte CSV. options : sep (","), entete (faux : lignes = listes ; vrai : lignes = tables par nom de colonne, et lignes.entete donne les noms), convertir (vrai : nombres convertis).
`csv.lire(chemin, [options])`	Lit un fichier CSV (mêmes options).
`csv.encoder(lignes, [options])`	Texte CSV. options : sep, entete (liste des colonnes, ou vrai pour les déduire).
`csv.ecrire(chemin, lignes, [options])`	Écrit un fichier CSV.

json

Instruction, fonction ou méthode	Rôle
`json.encoder(valeur, [joli])`	Texte JSON ; une liste 1..n devient un tableau, sinon un objet (joli = vrai : indenté).

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Instruction, fonction ou méthode	Rôle
`json.decoder(texte, [exact])`	Valeur Lexis+ ; les clefs d'objets sont mises en majuscules sauf si exact = vrai.
`json.lire(chemin, [exact])`	Lit et décode un fichier JSON.
`json.ecrire(chemin, valeur, [joli])`	Encode et écrit un fichier JSON (indenté par défaut).
`json.nul`	Valeur représentant null dans les données à encoder.

binaire

Instruction, fonction ou méthode	Rôle
`binaire.coder(format, ...)`	Code des valeurs en binaire. Format : < petit-boutiste, > gros-boutiste, b/B 8 bits, h/H 16, i/I 32 (iN : N octets), l/L/j/J 64, f flottant 32, d flottant 64, z chaîne terminée par zéro, s chaîne préfixée par sa longueur, cN chaîne de N octets, x octet nul.
`binaire.decoder(format, donnees, [position])`	Décode ; renvoie la liste des valeurs, avec le champ suite (position suivante).
`binaire.taille(format)`	Taille d'un format fixe.
`binaire.hex(donnees)`	Représentation hexadécimale.
`binaire.dehex(hex)`	Octets à partir de l'hexadécimal.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr



Référence · Système

Dates, environnement, commandes, arguments. — 22 entrées.

systeme

Instruction, fonction ou méthode	Rôle
`systeme.temps([date])`	Heure actuelle (secondes Unix), ou date d'une table {annee, mois, jour, heure, minute, seconde}.
`systeme.horloge()`	Temps écoulé depuis le lancement, en secondes (horloge monotone).
`systeme.date([format], [t])`	Date formatée : %Y %m %d %H %M %S %A (jour) %B (mois) %F %T %x… ; "*t" renvoie une table ; "!" en tête pour l'heure UTC.
`systeme.ecart(t2, t1)`	Différence en secondes.
`systeme.env(nom)`	Variable d'environnement (nul si absente).
`systeme.fixerenv(nom, valeur)`	Définit une variable d'environnement.
`systeme.executer(commande)`	Exécute une commande système ; renvoie son code de retour.
`systeme.capturer(commande)`	Exécute une commande ; renvoie {sortie, code}.
`systeme.supprimer(chemin)`	Supprime un fichier ou un dossier.
`systeme.renommer(ancien, nouveau)`	Renomme un fichier.
`systeme.nomtemporaire()`	Nom de fichier temporaire.
`systeme.quitter([code])`	Termine le programme.
`systeme.attendre(secondes)`	Pause (les fenêtres restent réactives).
`systeme.plateforme()`	"linux", "macosx", "windows"…
`systeme.dossierprogramme()`	Dossier du programme principal.
`systeme.args`	Arguments du programme ; systeme.args[0] est le chemin du programme.

Formats de systeme.date

Instruction, fonction ou méthode	Rôle
`%Y %m %d`	année, mois, jour
`%H %M %S`	heure, minute, seconde
`%A %a %B %b`	jour, mois (en lettres)
`%F %T %x %X %c`	formats composés
`%j %w %y %Z %%`	jour de l'année, de la semaine…
`"*t" "!…"`	fiche de date ; heure UTC

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Référence · Réseau

HTTP, TCP, UDP, FTP, SSH et SFTP. — 41 entrées.

http

Instruction, fonction ou méthode	Rôle
`http.lire(url, [entetes])`	Requête GET ; renvoie le corps de la réponse.
`http.envoyer(url, corps, [entetes])`	Requête POST ; renvoie le corps de la réponse.
`http.requete{methode=, url=, entetes=, corps=, delai=}`	Requête quelconque ; renvoie {statut, raison, corps, entetes} (en-têtes en minuscules : r.entetes["content-type"]).
`http.telecharger(url, chemin)`	Télécharge dans un fichier ; renvoie le statut.
`http.encoder(texte | table)`	Encodage d'URL (une table devient a=1&b=2).
`http.decoder(texte)`	Décodage d'URL.

reseau

Instruction, fonction ou méthode	Rôle
`reseau.connecter(hote, port, [options])`	Connexion TCP. options : tls (vrai pour TLS, -d:ssl), delai (secondes).
`reseau.ecouter(port, [hote])`	Socket serveur TCP.
`reseau.udp([port], [hote])`	Socket UDP, éventuellement lié à un port.
`reseau.resoudre(hote)`	Adresse IP d'un nom.
`reseau.nomhote()`	Nom de la machine.
`s:envoyer(donnees)`	Envoie des octets ; renvoie le nombre envoyé.
`s:recevoir([mode])`	Reçoit : "l" une ligne (défaut), "t" tout jusqu'à la fermeture, ou un nombre d'octets ; nul si la connexion est fermée.
`s:delai(secondes)`	Délai maximal des réceptions (nul = aucun).
`s:accepter()`	Attend un client (socket serveur) ; renvoie son socket.
`s:envoyervers(donnees, port, hote)`	Envoi UDP.
`s:recevoirde([taille])`	Réception UDP : {donnees, adresse, port}.
`s:distant()`	{adresse, port} de l'autre extrémité.
`s:fermer()`	Ferme le socket.

ftp

Instruction, fonction ou méthode	Rôle
`ftp.connecter{hote=, port=21, utilisateur="anonymous", motdepasse=""}`	Ouvre une session FTP (mode passif).
`c:liste([chemin], [detail])`	Noms des fichiers (detail = vrai : format détaillé).
`c:lire(distant, [local])`	Télécharge : renvoie les données, ou les écrit dans le fichier local.
`c:envoyer(local_ou_donnees, [distant], [sontdesdonnees])`	Envoie un fichier local (ou des données).
`c:dossier(chemin)`	Change de dossier distant.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Instruction, fonction ou méthode	Rôle
`c:courant()`	Dossier distant courant.
`c:creerdossier(chemin)`	Crée un dossier.
`c:supprimerdossier(chemin)`	Supprime un dossier.
`c:supprimer(chemin)`	Supprime un fichier.
`c:renommer(ancien, nouveau)`	Renomme.
`c:fermer()`	Ferme la session.

ssh

Instruction, fonction ou méthode	Rôle
`ssh.connecter{hote=, utilisateur=, port=22, clef=, motdepasse=, options={}}`	Ouvre une session (vérifie la connexion). Authentification par clef ; motdepasse demande sshpass.
`ssh.executer{hote=, utilisateur=, commande=, ...}`	Exécute une commande sans garder de session : {sortie, code}.
`session:executer(commande)`	Exécute une commande distante : {sortie, code}.
`session:lire(distant, [local])`	Télécharge un fichier ou dossier (scp).
`session:envoyer(local, [distant])`	Envoie un fichier ou dossier (scp).
`session:liste([chemin])`	Noms d'un dossier distant (sftp).
`session:creerdossier(chemin)`	Crée un dossier distant (sftp).
`session:supprimerdossier(chemin)`	Supprime un dossier distant (sftp).
`session:supprimer(chemin)`	Supprime un fichier distant (sftp).
`session:renommer(ancien, nouveau)`	Renomme (sftp).

sftp

Instruction, fonction ou méthode	Rôle
`sftp.connecter{hote=, utilisateur=, port=22, clef=, motdepasse=}`	Ouvre une session de transfert (identique à ssh.connecter).

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr



Référence · Multitâche

Tâches coopératives et fils d'exécution. — 22 entrées.

tache

Instruction, fonction ou méthode	Rôle
`tache.lancer(@proc, ...)`	Crée une tâche qui exécutera proc(...) ; elle démarre au prochain tache.executer(), tache.attendre() ou attente.
`tache.executer()`	Fait tourner les tâches jusqu'à ce qu'elles soient toutes terminées.
`tache.attendre(secondes)`	Dans une tâche : la suspend ; ailleurs : attend en faisant tourner les tâches.
`tache.ceder()`	Rend la main aux autres tâches.
`tache.canal([capacite])`	Canal de communication (capacité 0 = illimitée).
`tache.nombre()`	Nombre de tâches en cours.
`tache.maintenant()`	Heure en secondes (avec décimales).
`t:attendre()`	Attend la fin d'une tâche et renvoie son résultat.
`t:etat()`	"suspendue", "en cours", "terminee" ou "erreur".
`t:annuler()`	Arrête une tâche suspendue.
`c:envoyer(valeur)`	Envoie une valeur (attend si le canal est plein).
`c:recevoir([delai])`	Reçoit une valeur (attend si vide) ; nul après le délai.
`c:nombre()`	Nombre de valeurs en attente.

fil

Instruction, fonction ou méthode	Rôle
`fil.demarrer(code_ou_fichier, ...)`	Lance un fil qui exécute du code Lexis+ (chaîne) ou un fichier .prg, avec ses propres variables ; renvoie le fil.
`fil.envoyer(valeur)`	Dans le fil : envoie une valeur au programme qui l'a lancé.
`fil.recevoir([delai])`	Dans le fil : reçoit une valeur (nul après le délai).
`fil.estenfant()`	Vrai dans un fil lancé par fil.demarrer.
`f:envoyer(valeur)`	Envoie une valeur au fil (tables, chaînes, nombres, logiques).
`f:recevoir([delai])`	Reçoit une valeur du fil ; nul après le délai.
`f:actif()`	Vrai tant que le fil s'exécute.
`f:joindre()`	Attend la fin du fil ; son erreur éventuelle est relevée.
`f:arreter()`	Demande l'arrêt du fil.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr



Référence · Interface graphique

Fenêtres wdgui : contrôles, événements, dialogues. — 154 entrées.

Fenêtre et boucle

Instruction, fonction ou méthode	Rôle
`interface.fenetre{titre=, largeur=, hauteur=, disposition=, theme=, sombre=}`	Crée une fenêtre
`interface.fenetres()`	Nombre de fenêtres ouvertes
`interface.executer()`	Attend la fermeture des fenêtres (atelier) ou lance la boucle de wdgui...
`interface.quitter()`	Ferme toutes les fenêtres du programme
`interface.fenetres()`	Nombre de fenêtres ouvertes
`interface.tracer([actif])`	Affiche dans la console chaque événement reçu (diagnostic)
`interface.separateur{vertical=, position=}`	Séparateur (deux enfants)
`interface.sep`	Séparateur des chemins d'arbre et de menu
`fen:fermer()`	Ferme la fenêtre
`fen:fixertheme(nom, [sombre])`	Change le thème
`c:ajouter(enfant, ...)`	Ajoute des contrôles à un conteneur ou une fenêtre
`c:focus()`	Donne le focus
`c:menucontextuel(element, ...)`	Menu contextuel
`c:redessiner()`	Redessine le contrôle

Contrôles

Instruction, fonction ou méthode	Rôle
`interface.etiquette(texte, [alignement])`	Libellé (alignement : gauche, centre, droite)
`interface.bouton(libelle, [{defaut=, annuler=}])`	Bouton
`interface.saisie(texte | {texte=, indication=, multiligne=})`	Champ de saisie
`interface.cases(option, ...)`	Cases à cocher
`interface.radio(option, ...)`	Boutons radio
`interface.liste([multi])`	Liste

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Instruction, fonction ou méthode	Rôle
`interface.combo(element, ...)`	Liste déroulante
`interface.tableau()`	Tableau simple
`interface.arbre()`	Arbre
`interface.grille{multi=, remplissage="programme|manuel|automatique"}`	Grille de saisie
`interface.jauge()`	Jauge
`interface.curseur()`	Potentiomètre
`interface.plage()`	Potentiomètre de plage
`interface.ascenseur()`	Ascenseur
`interface.compteur()`	Compteur à flèches
`interface.notation()`	Notation
`interface.image(chemin)`	Image
`interface.codebarre(texte)`	Code-barres
`interface.graphe([genre], [titre])`	Graphe : colonnes, lignes, aires, secteurs
`interface.calendrier(["AAAAMMJJ"])`	Calendrier
`interface.kanban()`	Kanban
`interface.mosaique()`	Mosaïque proportionnelle (treemap)
`interface.gantt()`	Gantt
`interface.barremenu()`	Barre de menus
`interface.onglets()`	Onglets
`interface.separateur{vertical=, position=}`	Séparateur (deux enfants)
`interface.conteneur([disposition], [colonnes])`	Conteneur
`interface.colonne()`	Conteneur vertical
`interface.rangee()`	Conteneur horizontal
`interface.superconteneur([disposition])`	Conteneur sans marge
`interface.cadre(titre, [disposition])`	Cadre de groupe
`interface.panneau{disposition=, defilement=, colonnes=}`	Conteneur défilant (defilement : toujours, automatique, masque)
`interface.pdf()`	Visionneuse PDF (PDFium)

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Dialogues

Instruction, fonction ou méthode	Rôle
`interface.alerte(message, [icone], [titre])`	Message (icone : information, question, attention, stop, aucune)
`interface.confirmer(message, [icone])`	OK / Annuler
`interface.demander(message, [defaut])`	Saisie d'un texte
`interface.ouvrirfichier{titre=, filtre=, dossier=}`	Choix d'un fichier à ouvrir
`interface.enregistrerfichier{nom=, filtre=}`	Choix d'un fichier à enregistrer
`interface.couleur(["#RRGGBB"])`	Sélecteur de couleur
`interface.police([famille], [taille])`	Sélecteur de police : {famille, taille, gras, italique}

Listes, tableaux, arbres

Instruction, fonction ou méthode	Rôle
`liste:ajouter(element, ...)`	Ajoute (listes et combos)
`liste:inserer(i, element)`	Insère
`liste:supprimer(i)`	Supprime
`liste:vider()`	Vide
`liste:selectionner(i)`	Sélectionne
`liste:selection()`	Index sélectionné
`liste:nombre()`	Nombre d'éléments
`liste:element(i)`	Texte d'un élément
`liste:modifier(i, texte)`	Modifie
`liste:chercher(texte)`	Recherche
`liste:deselectionner(i)`	Désélectionne
`tableau:ajoutercolonne(titre, [largeur], [alignement])`	Ajoute une colonne
`tableau:ajouterligne(cellule, ...)`	Ajoute une ligne
`tableau:cellule(ligne, colonne)`	Texte d'une cellule
`tableau:fixercellule(ligne, colonne, texte)`	Modifie une cellule
`tableau:selection()`	Ligne sélectionnée
`tableau:trier(colonne, [croissant])`	Trie

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Instruction, fonction ou méthode	Rôle
`tableau:insererligne(i, cellule, ...)`	Insère une ligne
`tableau:ligne(i)`	Cellules d'une ligne
`tableau:nombre()`	Nombre de lignes
`tableau:selectionner(i)`	Sélectionne une ligne
`tableau:supprimer(i)`	Supprime une ligne
`tableau:vider()`	Vide le tableau
`arbre:ajouter(chemin...)`	Ajoute un nœud ("A", "B" ou "A/B")
`arbre:deplier(chemin...)`	Déplie
`arbre:replier(chemin...)`	Replie
`arbre:supprimer(chemin...)`	Supprime
`arbre:vider()`	Vide
`arbre:estdeplie(chemin...)`	Vrai si le nœud est déplié
`arbre:selectionner(chemin...)`	Sélectionne un nœud
`arbre:nombre()`	Nombre de nœuds
`onglets:ajouterpage(titre, [disposition])`	Ajoute une page
`barremenu:ajouter(chemin...)`	Ajoute un élément de menu ("Fichier", "Ouvrir")
`cases:coche([i])`	État d'une case
`cases:cocher(i, [b])`	Coche une case
`cases:ajouteroption(texte)`	Ajoute une case
`radio:ajouteroption(texte)`	Ajoute un bouton radio

Grille

Instruction, fonction ou méthode	Rôle
`grille:ajoutercolonne{nom=, titre=, genre=, largeur=, choix=, decimales=}`	Ajoute une colonne (genre : texte, nombre, date, image, bouton, combo, case)
`grille:ajouterligne(cellule, ...)`	Ajoute une ligne
`grille:cellule(ligne, colonne)`	Valeur d'une cellule (colonne = nom ou index)
`grille:fixercellule(ligne, colonne, valeur)`	Modifie une cellule
`grille:nombre()`	Nombre de lignes
`grille:trier(colonne, [croissant])`	Trie

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Instruction, fonction ou méthode	Rôle
`grille:chercher(colonne, valeur, [exact], [debut])`	Cherche une ligne
`grille:fixerpied(colonne, genre)`	Pied de colonne : somme, moyenne, min, max, nombre
`grille:ajouterformat(colonne, operateur, valeur, {fond=, texte=, ligneentiere=})`	Mise en forme conditionnelle
`grille:fixermodifiable([b])`	Mode saisie
`grille:verscsv([sep])`	Export CSV
`grille:chargercsv(chemin, [sep])`	Import CSV
`grille:versjson()`	Export JSON
`grille:chargerjson(chemin)`	Import JSON
`grille:insererligne(i, ...)`	Insère une ligne
`grille:modifierligne(i, ...)`	Remplace une ligne
`grille:supprimerligne(i)`	Supprime une ligne
`grille:vider()`	Vide la grille
`grille:ligne(i)`	Cellules d'une ligne
`grille:nombrecolonnes()`	Nombre de colonnes
`grille:colonne(col)`	Objet colonne
`grille:selection([i])`	Ligne sélectionnée (i-ème)
`grille:nombreselection()`	Nombre de lignes sélectionnées
`grille:selectionner(l)`	Sélectionne une ligne
`grille:deselectionner(l)`	Désélectionne
`grille:toutselectionner()`	Sélectionne tout
`grille:trouver(texte)`	Cherche partout : {ligne, colonne}
`grille:valeurpied(col, [genre])`	Valeur d'un pied de colonne
`grille:effacerformats()`	Retire les mises en forme
`grille:fixerrupture([col])`	Regroupe sur une colonne
`grille:fixermodecolonne(col, mode)`	Saisie d'une colonne (lecture, modifiable)
`grille:fixermodeligne(l, mode)`	Saisie d'une ligne
`grille:fixerremplissage(mode)`	programme, manuel, automatique
`grille:fixerlibellebouton(col, texte)`	Libellé des boutons d'une colonne
`grille:modifiercellule(l, col)`	Place la saisie dans une cellule

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Instruction, fonction ou méthode	Rôle
`grille:decsv(texte, [ajout])`	Remplit depuis un texte CSV
`grille:enregistrercsv(chemin, [sep])`	Écrit un CSV
`grille:enregistrerjson(chemin)`	Écrit un JSON

Graphes, planning, PDF

Instruction, fonction ou méthode	Rôle
`graphe:ajouterdonnee(serie, categorie, valeur)`	Ajoute une donnée
`graphe:libellecategorie(i, texte)`	Libellé de catégorie
`graphe:libelleserie(i, texte)`	Libellé de série
`graphe:fixergenre(genre)`	Genre de graphe
`calendrier:afficher(annee, mois)`	Affiche un mois
`kanban:ajouterliste(titre)`	Ajoute une liste ; renvoie son numéro
`kanban:ajouter(liste, texte)`	Ajoute une carte
`mosaique:ajouter(libelle, valeur)`	Ajoute une tuile proportionnelle
`gantt:ajoutertache(nom, debut, duree, [avancement])`	Ajoute une tâche ; renvoie son numéro
`gantt:fixeravancement(i, valeur)`	Avancement d'une tâche (%)
`panneau:defilera(x, y)`	Fait défiler
`panneau:defilerde(dx, dy)`	Fait défiler d'un décalage
`panneau:rendrevisible(controle)`	Amène un contrôle à l'écran
`pdf:ouvrir(chemin)`	Ouvre un PDF
`pdf:enregistrer([chemin])`	Enregistre
`pdf:nouveau([largeur], [hauteur])`	Nouveau document vide
`pdf:fermer()`	Ferme le document
`pdf:nombrepages()`	Nombre de pages
`pdf:page()`	Page courante
`pdf:allera(n)`	Va à la page n
`pdf:zoom(z)`	Zoom
`pdf:ajoutertexte(page, x, y, texte, [taille])`	Écrit sur une page
`pdf:importerpages(chemin, [plage], [position])`	Insère des pages d'un autre PDF

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Propriétés

Instruction, fonction ou méthode	Rôle
`valeur texte num libelle titre visible etat actif infobulle nom x y largeur hauteur couleur fond bordure accent rayon coche min max ancrage poids largeurfixe hauteurfixe etirer marge espacement colonnes famille genre`	ctl.propriete en lecture et en écriture

Événements

Instruction, fonction ou méthode	Rôle
`surclic surdoubleclic surclicdroit surclicmilieu surdoubleclicdroit surboutonbas surboutonhaut surentree sursortie surdeplacement surmolette surtouchebas surtouchehaut sursaisie surfocus surfinfocus surchangement surselection surdeploiement surrepli survalidation surouverture surfermeture surredimension suractivation surdesactivation surevenement`	ctl.surXXX = @PROC ; PROC(ev) ; ev:arreter()

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr



Référence · Pages web

Pages HTML / CSS / JavaScript reliées à Lexis+. — 31 entrées.

webview

Instruction, fonction ou méthode	Rôle
`webview.fenetre{titre=, largeur=, hauteur=, taille=, fichier=, url=, html=, debogage=, console=}`	Crée une fenêtre web. taille : normale, min, max, fixe ; fichier : page locale (relative au programme) ; debogage = vrai : outils de développement ; console = faux : ne pas renvoyer console.log vers la console Lexis+.
`webview.disponible()`	Vrai si la bibliothèque webview est chargée (sinon voir webview.erreur()).
`webview.erreur()`	Raison de l'échec du chargement de la bibliothèque (nul si aucune).
`webview.version()`	Version de la bibliothèque webview.
`webview.bibliotheque()`	Chemin de la bibliothèque chargée.
`webview.url(chemin)`	URL file:// d'un fichier local (relatif au dossier du programme).
`webview.js(valeur)`	Littéral JavaScript (JSON) d'une valeur Lexis+ ; les clefs deviennent minuscules, une table vide devient [].
`fen:html(code)`	Affiche du HTML.
`fen:naviguer(url)`	Charge une URL (https://, file://, data:...).
`fen:ouvrir(chemin)`	Charge un fichier HTML local (relatif au dossier du programme).
`fen:initialiser(js)`	Ajoute un script exécuté avant chaque page.
`fen:evaluer(code, [@rappel])`	Lexis+ → JavaScript : évalue du code ; rappel(valeur, erreur) reçoit le résultat.
`fen:appeler(fonction, arg..., [@rappel])`	Lexis+ → JavaScript : appelle window.fonction(arg...) (chemin "app.maj" accepté) ; rappel(valeur, erreur).
`fen:emettre(nom, [donnees])`	Lexis+ → JavaScript : déclenche les écouteurs lexis.sur(nom, fn).
`fen:lier(nomjs, [@procedure])`	JavaScript → Lexis+ : crée window.nomjs(...) (Promise) qui appelle la procédure (par défaut celle du même nom).
`fen:delier(nomjs)`	Retire une liaison.
`fen:autoriser(nom, ...)`	JavaScript → Lexis+ : procédures appelables par lexis.appeler(nom, ...) ; "*" les autorise toutes.
`fen:taille(largeur, hauteur, [genre])`	Taille de la fenêtre (genre : normale, min, max, fixe).
`fen:executer()`	Affiche la fenêtre et traite ses événements jusqu'à sa fermeture.
`fen:fermer()`	Ferme la fenêtre

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Côté JavaScript

Instruction, fonction ou méthode	Rôle
`nomjs(…)`	fonction liée par f:lier : Promise
`lexis.appeler(nom, …)`	procédure autorisée : Promise
`lexis.envoyer(d)`	→ f.surmessage
`lexis.journal(…)`	→ console Lexis+
`lexis.sur(nom, fn) · lexis.retirer`	écouteurs de f:emettre
`lexis.pret`	page chargée

Propriétés

Instruction, fonction ou méthode	Rôle
`f.titre`	titre (lecture / écriture)
`f.surpret`	page chargée : P(ev), ev.url
`f.surmessage`	lexis.envoyer(d) : P(d)
`f.surfermeture`	après la fermeture
`f.prete · f.ouverte`	états

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr



Référence · Intelligence artificielle

Modèles de langage, GGUF, dessin, audio, tenseurs, entraînement. — 174 entrées.

llm

Instruction, fonction ou méthode	Rôle
`llm.charger(chemin, {fils=0, verbeux=faux})`	Charge un modèle GGUF (fichier « mappé » : quasi instantané) ; renvoie un modèle.
`llm.version()`	Version de la bibliothèque nimllm.
`llm.fils([n])`	Nombre de fils de calcul (0 = tous les cœurs) ; renvoie le nombre utilisé.
`llm.echantillonnage([table])`	Réglages d'échantillonnage par défaut (complétés par la table) : temperature, topk, topp, minp, repetition, fenetre, presence, frequence, graine, biais.
`llm.glouton([table])`	Réglages du décodage glouton (déterministe).
`llm.echantillonneur([reglages])`	Échantillonneur manuel : s:tirer(logits), s:accepter(id).
`llm.joindre(chemin)`	Pièce jointe lue sur le disque (texte, CSV, JSON, PDF, PNG/BMP, WAV...).
`llm.joindretexte(nom, contenu)`	Pièce jointe construite en mémoire.
`llm.joindredonnees(nom, octets)`	Pièce jointe à partir d'octets (le type est déduit du nom et du contenu).
`llm.extrairejson(texte)`	Premier objet JSON trouvé dans un texte (table) ou nul.
`llm.sansbalises(texte)`	Retire les balises ``` d'un bloc de code.
`llm.similarite(v1, v2)`	Similarité cosinus de deux vecteurs (m:vecteur).
`llm.prompt(gabarit, messages[, suite=vrai])`	Texte d'un dialogue {{role=, contenu=}...} mis en forme (llama3, chatml, mistral, llama2, gemma, phi3).
`llm.gabarits()`	Liste des gabarits de conversation.
`llm.tokeniseur(chemin)`	Tokeniseur seul lu dans un fichier GGUF.
`llm.predecouper(texte[, mode])`	Pré-découpage avant BPE (llama3, qwen2, gpt2).
`m:conversation({systeme=, contexte=4096, maxtokens=512, echantillonnage=, gabarit="auto", langue="fr", arrets=})`	Nouvelle conversation (le texte seul est accepté comme prompt système).
`m:contexte([taille])`	Contexte brut (cache KV) : evaluer, completer, generer.
`m:decrire()`	Résumé du modèle.
`m:vecteur(texte)`	Vecteur (embedding) d'un texte.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Instruction, fonction ou méthode	Rôle
`m:perplexite(texte)`	Perplexité d'un texte (plus elle est basse, plus le texte est attendu).
`m:appliquerlora(chemin)`	Applique un adaptateur LoRA (.lora.gguf) à la volée.
`m:retirerlora()`	Retire l'adaptateur LoRA.
`m:fermer()`	Libère le modèle.
`m.tokeniseur`	Tokeniseur du modèle (propriétés : nom, architecture, couches, dimension, vocabulaire, contexte_max, parametres, gabarit).
`c:demander(question[, options | @flux])`	Réponse (fiche texte, brut, fichiers, json, tokensprompt, tokensgeneres, arret, secondes, vitesse ; s'affiche comme son texte). Options : pieces, format (texte, markdown, json, image, audio, fichier), sortie, schema, maxtokens, flux=@proc(morceau) qui renvoie faux pour interrompre.
`c:demanderjson(question[, schema])`	Réponse JSON convertie en table (clefs en majuscules).
`c:regenerer([@flux])`	Nouveau tirage de la dernière réponse.
`c:prolonger([maxtokens, @flux])`	Prolonge la dernière réponse.
`c:generer([options | @flux])`	Réponse de l'assistant à l'historique actuel (après c:ajouter).
`c:ajouter(role, texte)`	Ajoute un message (systeme, utilisateur, assistant) : exemples « few-shot », résultats d'outils.
`c:annuler()`	Retire le dernier échange.
`c:reinitialiser()`	Oublie l'historique (garde le prompt système).
`c:enregistrer(chemin)`	Enregistre la conversation (JSON).
`c:charger(chemin)`	Recharge une conversation ; renvoie le nombre de messages.
`c:etat()`	État du contexte (tokens, messages, vitesse).
`c:prompt([question])`	Texte exact envoyé au modèle.
`c:tokensprompt()`	Tokens de l'historique mis en forme.
`c.systeme`	Propriétés : systeme, maxtokens, arrets, echantillonnage, tailleimage, voix, essaisjson, langue, sansreflexion, gabarit ; lecture seule : historique, messages, tokens, contexte, memoire.
`x:evaluer(ids)`	Ajoute des tokens au contexte et renvoie les logits du dernier.
`x:completer(texte[, options | @flux])`	Complétion brute (sans gabarit de conversation).
`x:generer(ids[, options | @flux])`	Génère une suite à partir de tokens.
`x:reinitialiser()`	Vide le contexte.
`x:tronquer(n)`	Ne garde que les n premiers tokens.
`l:meilleurs([n])`	Les n tokens les plus probables : {{id=, proba=}...}.
`l:valeur(id)`	Logit d'un token.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Instruction, fonction ou méthode	Rôle
`l:argmax()`	Token le plus probable.
`s:tirer(logits)`	Tire un token selon les réglages.
`s:accepter(id)`	Mémorise un token (pénalités de répétition).
`t:encoder(texte[, bos=faux, speciaux=vrai])`	Liste des identifiants de tokens.
`t:decoder(ids[, speciaux=faux])`	Texte d'une liste de tokens.
`t:morceau(id[, special=faux])`	Texte d'un token.
`t:id(texte)`	Identifiant d'un token (nul s'il n'existe pas).
`t:estspecial(id)`	Vrai pour un token spécial.
`t:enregistrer(chemin)`	Enregistre le tokeniseur dans un fichier GGUF.
`t.taille`	Propriétés : taille, bos, eos, pad, inconnu, fins, fusions, type, gabarit, ajoutebos.
`p:texte([max])`	Texte inséré dans le message (propriétés : nom, type, mime, contenu, taille, largeur, hauteur, image).

gguf

Instruction, fonction ou méthode	Rôle
`gguf.ouvrir(chemin)`	Ouvre un fichier GGUF (lecture seule, mappé en mémoire).
`gguf.quantifiermodele(entree, sortie[, type="q4_k", garder_sortie=vrai])`	Re-quantifie un modèle (f32, f16, bf16, q8_0, q6_k, q5_k, q5_0, q5_1, q4_k, q4_0, q4_1…).
`gguf.fusionnerlora(base, adaptateur, sortie[, type="q8_0"])`	Fusionne un adaptateur LoRA dans un nouveau modèle autonome.
`gguf.types()`	Types de poids utilisables pour la quantification.
`gguf.bitsparpoids(type)`	Nombre moyen de bits par poids d'un type.
`gguf.quantifier(type, valeurs)`	Quantifie puis relit une liste : {octets=, valeurs=, erreur=}.
`g:decrire([max_tenseurs])`	Résumé du fichier.
`g:existe(cle)`	Vrai si la métadonnée existe.
`g:valeur(cle)`	Valeur d'une métadonnée (nombre, texte, logique ou liste).
`g:texte(cle[, defaut])`	Métadonnée texte.
`g:entier(cle[, defaut])`	Métadonnée entière.
`g:reel(cle[, defaut])`	Métadonnée réelle.
`g:cles()`	Liste des clefs de métadonnées.
`g:tenseurs()`	Liste des tenseurs : {nom=, type=, dims=, elements=, octets=}.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Instruction, fonction ou méthode	Rôle
`g:tenseur(nom[, max])`	Valeurs d'un tenseur (déquantifiées).
`g:fermer()`	Ferme le fichier (propriétés : version, chemin, alignement, architecture).

dessin

Instruction, fonction ou méthode	Rôle
`dessin.svg(code[, largeur, hauteur, fond])`	Rastérise un dessin SVG ; renvoie une image.
`dessin.image(largeur, hauteur[, fond])`	Image unie.
`dessin.toile(largeur, hauteur[, fond, lissage=3])`	Surface de dessin anticrénelée.
`dessin.lire(chemin)`	Lit une image BMP, PPM ou PNG.
`dessin.extrairesvg(texte)`	Code SVG contenu dans un texte (réponse d'un modèle) ou nul.
`dessin.taillesvg(code)`	Dimensions déclarées d'un SVG : {largeur=, hauteur=}.
`dessin.couleur(texte | r, g, b[, a])`	Couleur sous forme de table {r=, g=, b=, a=}.
`t:rectangle(x, y, largeur, hauteur, couleur)`	Rectangle plein.
`t:cercle(cx, cy, rayon, couleur)`	Disque.
`t:ellipse(cx, cy, rx, ry, couleur)`	Ellipse pleine.
`t:polygone(points, couleur)`	Polygone plein ; points : {{x, y}, ...}.
`t:ligne(x0, y0, x1, y1, epaisseur, couleur)`	Segment.
`t:polyligne(points, epaisseur, couleur[, fermee])`	Ligne brisée.
`t:terminer()`	Termine le dessin et renvoie l'image.
`img:enregistrer(chemin)`	Enregistre selon l'extension (.bmp ou .ppm).
`img:enregistrerbmp(chemin)`	Enregistre en BMP.
`img:enregistrerppm(chemin)`	Enregistre en PPM.
`img:pixel(x, y)`	Couleur d'un pixel : {r=, g=, b=}.
`img:fixerpixel(x, y, couleur)`	Colore un pixel.
`img:decrire([largeur])`	Description textuelle (couleurs, aperçu en caractères) ; propriétés largeur, hauteur.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

audio

Instruction, fonction ou méthode	Rôle
`audio.parler(texte[, {langue="fr", hauteur=115, vitesse=1, frequence=16000}])`	Synthèse vocale (fr ou en) ; renvoie un son.
`audio.melodie(notation[, bpm=120])`	Mélodie : « C4 E4 G4:2 R » (note+octave[:temps], R = silence).
`audio.nouveau([frequence])`	Son vide.
`audio.lire(chemin)`	Lit un fichier WAV PCM 16 bits.
`audio.infos(chemin)`	En-tête d'un WAV : {frequence=, canaux=, bits=, duree=}.
`audio.phonemes(texte[, langue])`	Transcription phonétique utilisée par la synthèse.
`audio.enlettres(n[, langue])`	Nombre écrit en toutes lettres.
`audio.frequencenote(note)`	Fréquence d'une note (« A4 » = 440).
`s:enregistrer(chemin)`	Écrit un fichier WAV (16 bits mono).
`s:silence(secondes)`	Ajoute un silence.
`s:ajouter(autre)`	Ajoute un autre son à la suite (rééchantillonné si besoin).
`s:normaliser([crete=0.9])`	Ajuste le volume.
`s:copier()`	Copie du son (propriétés : duree, frequence, echantillons).

tenseur

Instruction, fonction ou méthode	Rôle
`tenseur.depuis(valeurs[, forme, gradient=faux, nom])`	Tenseur à partir d'une liste ; forme {lignes, colonnes}.
`tenseur.plein(forme[, valeur=0, gradient=faux, nom])`	Tenseur rempli d'une valeur.
`tenseur.zeros(forme[, gradient=faux, nom])`	Tenseur nul.
`tenseur.scalaire(v[, gradient=faux])`	Tenseur d'une seule valeur.
`tenseur.aleatoire(forme[, ecart=1, gradient=vrai, nom])`	Valeurs gaussiennes N(0, écart²).
`tenseur.parametre(forme[, ecart=0.1, nom])`	Paramètre entraînable initialisé au hasard.
`tenseur.graine(n)`	Fixe la graine du générateur aléatoire du paquet.
`tenseur.hasard([max | "gauss"])`	Nombre aléatoire du générateur du paquet.
`tenseur.lineaire(x, w[, b])`	$x \cdot w^T + b$ (x : {n, entrées}, w : {sorties, entrées}).
`tenseur.produit(a, b)`	Produit matriciel {n, k} · {k, m}.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Instruction, fonction ou méthode	Rôle
`tenseur.echelle(t, s)`	Multiplie par un nombre (aussi : t * s).
`tenseur.somme(t)`	Somme des éléments.
`tenseur.moyenne(t)`	Moyenne des éléments.
`tenseur.relu(t)`	max(0, x).
`tenseur.silu(t)`	x · sigmoïde(x).
`tenseur.sigmoide(t)`	1 / (1 + e^-x).
`tenseur.tanh(t)`	Tangente hyperbolique.
`tenseur.gelu(t)`	GELU.
`tenseur.softmax(t)`	Softmax par ligne.
`tenseur.mse(prediction, cible)`	Erreur quadratique moyenne.
`tenseur.entropiecroisee(logits, classes)`	Entropie croisée (classes : liste d'entiers à partir de 0 ; -1 = ignorée).
`tenseur.concatener(a, b)`	Concaténation des colonnes.
`tenseur.decrochage(t, p)`	Dropout (entraînement).
`tenseur.sansgradient(@proc, ...)`	Appelle proc sans construire de graphe (inférence).
`tenseur.verifiergradient(@proc, t[, eps, echantillons])`	Écart relatif entre gradient calculé et différence finie.
`tenseur.operation(parents, valeurs, forme, @retro)`	Opération personnalisée ; retro(resultat, parents) appelle p:ajoutergradient().
`tenseur.gradientrequis(t, ...)`	Vrai si l'un des tenseurs demande un gradient.
`tenseur.adamw(parametres[, {lr=3e-4, beta1=0.9, beta2=0.95, decroissance=0.1}])`	Optimiseur AdamW.
`tenseur.sgd(parametres[, {lr=0.01, moment=0.9, decroissance=0}])`	Descente de gradient avec moment.
`tenseur.lrcosinus(etape, echauffement, total, lrmax[, lrmin])`	Taux d'apprentissage (échauffement + cosinus).
`tenseur.normegradient(parametres)`	Norme globale des gradients.
`t:retropropager()`	Calcule les gradients de tout le graphe (t scalaire).
`t:valeur([i=1])`	Valeur d'un élément.
`t:donnees()`	Liste des valeurs.
`t:gradient()`	Liste du gradient (nul s'il n'a pas été calculé).
`t:forme()`	Forme (liste des dimensions).
`t:taille()`	Nombre d'éléments.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Instruction, fonction ou méthode	Rôle
`t:zerograd()`	Remet le gradient à zéro.
`t:detacher()`	Copie sans gradient.
`t:remodeler(forme)`	Même données, autre forme.
`t:ajoutergradient(valeurs)`	Ajoute au gradient (opérations personnalisées) ; propriétés : nom, gradientrequis.
`o:zerograd()`	Remet à zéro les gradients des paramètres.
`o:etape()`	Met à jour les paramètres.
`o:ecreter(norme_max)`	Limite la norme des gradients ; renvoie la norme d'origine.
`o:enregistrer(chemin)`	Enregistre l'état (moments) de l'optimiseur.
`o:charger(chemin)`	Recharge l'état ; propriétés : pas, lr (modifiable), type.

entrainement

Instruction, fonction ou méthode	Rôle
`entrainement.tokeniseur(textes[, {taille=2048, speciaux=, frequencemin=2, mode="llama3"}])`	Apprend un tokeniseur BPE niveau octet (compatible llama.cpp).
`entrainement.tokeniseuroctets([speciaux])`	Tokeniseur minimal (256 octets + spéciaux).
`entrainement.nouveau(tok, {dimension=256, couches=4, tetes=4, tetes_kv=, cache=, contexte=256, nom=, graine=42})`	Nouveau modèle Transformer (architecture Llama).
`entrainement.charger(chemin[, {mode="lora", rang=8, alpha=16, cibles={"q","k","v","o"}}])`	Charge un GGUF à spécialiser : LoRA (poids gelés) ou complet.
`entrainement.textes(tok, texte[, {graine=, bos=vrai}])`	Données de pré-entraînement (texte continu).
`entrainement.fichiers(tok, chemins)`	Données de pré-entraînement lues dans des fichiers.
`entrainement.dialogues(tok, gabarit, dialogues[, {systeme=}])`	Données de dialogue (seules les réponses sont apprises).
`entrainement.jsonl(chemin, tok[, gabarit="auto", systeme])`	Dialogues lus dans un fichier JSONL.
`entrainement.gabarit(tok)`	Gabarit de conversation adapté au tokeniseur.
`entrainement.configuration()`	Réglages d'entraînement par défaut.
`m:entrainer(donnees[, {etapes, lot, sequence, accumulation, lr, lrmin, echauffement, decroissance, ecretage, evaluation, lotsevaluation, journal, sauvegarde, chemin, echantillon, amorce, graine, validation, optimiseur, rappel}])`	Entraîne le modèle ; renvoie l'optimiseur (pour reprendre). rappel=@proc(journal) reçoit {etape, perte, validation, lr, norme, vitesse, duree}.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Instruction, fonction ou méthode	Rôle
`m:parametres()`	Liste des tenseurs entraînables.
`m:nombreparametres([tous=faux])`	Nombre de paramètres.
`m:optimiseur([{lr=, decroissance=}])`	AdamW sur les paramètres du modèle.
`m:enregistrer(chemin[, type="f32"])`	Exporte un vrai fichier GGUF (llm.charger, llama.cpp, Ollama).
`m:enregistrerlora(chemin)`	Enregistre l'adaptateur LoRA.
`m:sauvegarder(optimiseur, prefixe)`	Point de sauvegarde (modèle ou adaptateur + optimiseur).
`m:evaluer(donnees[, lots=4, lot=8, sequence=64])`	Perte moyenne (exp(perte) = perplexité).
`m:generer(amorce[, {maxtokens=50, temperature=0.8}])`	Génère un texte (surveillance de l'entraînement) ; propriétés : tokeniseur, mode, config, gabarit.
`d:separer([fraction=0.1])`	Sépare en {entrainement, validation}.
`d:taille()`	Nombre de tokens (texte) ou de dialogues.
`d:pluslong()`	Longueur du plus long dialogue (tokens).

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

ANNEXE B

Aides-mémoires

Toute la syntaxe et toutes les instructions de chaque domaine sur une page, avec leurs enchaînements et les renvois entre domaines.



© 2026 Jean-Marc QUÉRÉ,sonaliwan.fr · SIRET13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Toute la syntaxe en une page

Les blocs du langage, leurs mots d'ouverture, de suite et de fermeture, la priorité des opérateurs et les types.

100 mots-clés et opérateurs

Lignes, valeurs, variables

```
&& commentaire   // commentaire   * ligne de commentaire
locale total = 0, nom = "Ada"      && locale
compteur += 1                       && globale implicite
x = a + ;                           && « ; » : suite à la ligne
    b
? "Total :", total   ?? "sans fin de ligne"
```

Conditions

```
si cond
  …
sinonsi cond      (0..n)
sinon             (0..1)
finsi
faire cas / cas cond / autrement / fincas
```

Boucles

```
pour i = 1 a 10 [pas 2]   …  suivant
pour chaque v dans liste   …  suivant
pour chaque k, v dans fiche …  suivant
tantque cond        …  fintantque
sortir   boucler     dans toute boucle
```

Procédures

```
procedure NOM(a, b)     = fonction
  parametres x, y       (style dBase)
  locale r = a * b
retourner r               ou finprocedure
NOM(1, 2)   faire NOM avec 1, 2   @NOM
```

Tables et objets

```
t = {1, 2, nom = "Ada"}   t[1]  t.nom
inserer v dans t [position p | clef "K"]
retirer de t position p dans x
nouvelle table t   supprimer table t
C = {}  C.__index = C
C:procedure P comme m   obj:m()
```

Erreurs

```
essayer
  …
capturer e
  ? "erreur :", e
finessayer
erreur "message"   erreur {code = 404}
```

SQL

```
sql "insert into t values (?, ?)" avec a, b
sql "select * from t where x > ?" avec 5 dans l
requete(sql, …)   valeursql(sql, …)
```

Zones (dBase)

```
utiliser clients alias c ordre "nom" filtre "…"
parcourir pour ville = "Brest" … finparcourir
aller haut|bas|n  sauter n  chercher v
localiser pour cond   continuer
ajouter avec c par v  remplacer c par v  effacer
compter pour cond dans n  selectionner a  fermer tout
```

Événements

```
ui = paquet("interface")
fen = ui.fenetre{titre = "…"}
b = fen:ajouter(ui.bouton("OK"))
b.surclic = @CLIC
ui.executer()
procedure CLIC(ev) … retourner
```

Priorité des opérateurs

De la plus forte…	Opérateurs
1	`^ **` (de droite à gauche)
2	`- non #` (unaires)
3	`* / %`
4	`+ - ..`
5	`= == <> != # < <= > >= $`
6	`et`
7 …à la plus faible	`ou`

Types — type(v)

`nul`	absence de valeur (faux dans un si)
`logique`	vrai, faux
`nombre`	42, 3.14 — 0 est vrai !
`chaine`	"…" — "" est vrai !
`table`	liste et / ou fiche
`procedure`	@NOM
`objet · tache`	fournis par les paquets

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr



Aide-mémoire · Démarrer

Écrire, exécuter, déboguer et lancer un programme.

Voir aussi : Les bases · Système · Procédures

21 instructions et fonctions

ENCHAÎNEMENTS

`écrire principal.prg` → `Ctrl+R` → `console` — `F9 point d'arrêt` → `F5` → `F11 / F10` → `variables`

`wdlexisrt prog.prg a b` → `systeme.args` → `systeme.quitter(code)`

Modèle type

```
&& principal.prg
? "Bonjour", saisir("Nom ? ")
pour i = 1 a 3
  ? i, CARRE(i)
suivant

fonction CARRE(n)
retourner n * n
```

Afficher et saisir

`? v1, v2, …`
affiche, séparé par des espaces, puis passe à la ligne

`?? v1, v2, …`
affiche sans passer à la ligne

`? (seul)`
passe à la ligne

`saisir([invite])`
Lit une ligne saisie (console de l'atelier ou terminal)

`paquet(nom)`
Charge un paquet : interface, fichier, json, csv, http, reseau, tache

Projet

`principal.prg`
fichier principal : son en-tête est le programme principal

`autres .prg`
procédures disponibles dans tout le projet (sous-dossiers compris)

`procedure PRINCIPAL`
exécutée si le fichier principal ne contient que des procédures

`finprocedure / finfonction`
fin explicite d'une procédure : la suite revient au programme principal

Atelier wdlexis+

`Ctrl+R`
exécuter

`F5`
déboguer / continuer

`F9`
point d'arrêt

`F11 · F10 · Maj+F11`
pas à pas · pas principal · sortir

`Maj+F5`
arrêter

`Ctrl+Espace`
complétion

`Ctrl+B`
commenter

`F12`
aller à la définition

Runtime wdlexisrt

`wdlexisrt prog.prg [args]`
exécute

`--verifier prog.prg`
vérifie la syntaxe

`-e "code"`
exécute une ligne

`--paquets`
liste les paquets

Pièges à éviter

⚠ Le programme principal s'écrit en tête : tout ce qui suit une procédure lui appartient.

⚠ Un seul fichier peut contenir des instructions en tête : principal.prg.

⚠ Ctrl+R exécute le document tel qu'il est à l'écran, même non enregistré.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Aide-mémoire · Les bases

Instructions, commentaires, valeurs, opérateurs et variables.

Voir aussi : Décider & répéter · Texte & calcul · Tables & objets

32 instructions et fonctions

ENCHAÎNEMENTS `"12,5"` → `nombre()` → `12.5` → `chaine(n, l, d)` → `" 12.50"` — `locale x` → `x = …` → `x += …` → `liberer x`

Modèle type

```
locale prix = nombre(saisir("Prix ? "))
si prix = nul
  ? "Ce n'est pas un nombre"
sinon
  ? "TTC :", chaine(prix * 1.2, 1, 2)
finsi
? nom ou "inconnu"      && valeur par défaut
```

Lignes et commentaires

une instruction par ligne
fin de ligne = fin d'instruction

… ;
continue sur la ligne suivante

&& · // · *
commentaires

casse
noms insensibles à la casse ; mots-clés en minuscules

à = a
accent facultatif

Valeurs

nul
absence de valeur

vrai faux .v. .f.
logiques

42 3.14 1e3
nombres

"…" '…' [[…]]
chaînes (\n \t \" \\)

{…}
table

@NOM
référence de procédure

Opérateurs

+ - * / % ^ **
calcul

+ (avec texte) ..
concaténation

= ==
égal

<> != #
différent

< <= > >=
comparaison

$
inclusion de texte

et ou non
logique (renvoient une des valeurs)

#v
longueur

Variables

locale x [= v], y
variable locale

globale x
globale explicite

x = v
affectation (nom non déclaré = globale)

x += v · x -= v
ajouter, retirer

stocker v dans a, b
même valeur partout

liberer x
remet à nul

SESSION
table des variables globales

m->x
la variable x (et non un champ)

Types et conversions

type(v)
"nul", "logique", "nombre", "chaine", "table", "procedure", "objet" ou "tache"

chaine(v, [largeur], [decimales])
Texte d'une valeur

texte(v)
Texte d'une valeur (utilise __tostring)

valeur(s)
Nombre d'un texte (virgule décimale admise), 0 si impossible (VAL)

nombre(s)
Nombre d'un texte, nul si impossible

Pièges à éviter

- ⚠ 2 + "3" donne "23" : le + colle dès qu'il y a du texte.
- ⚠ "a" = "A" est faux : les textes respectent la casse.
- ⚠ 0 et "" sont VRAIS dans un si ; seuls nul et faux sont faux.
- ⚠ -2 ^ 2 vaut -4 ; 2 ^ 3 ^ 2 vaut 512.
- ⚠ Sans locale, une variable de procédure est globale.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr



Aide-mémoire · Décider & répéter

Choisir un chemin, répéter un traitement.

Voir aussi : Les bases · Tables & objets · Erreurs

16 instructions et fonctions

ENCHAÎNEMENTS

`si` → `sinonsi *` → `sinon ?` → `finsi`

`faire cas` → `cas *` → `autrement ?` → `fincas`

`pour / pour chaque` → `boucler · sortir` → `suivant`

`tantque` → `boucler · sortir` → `fintantque`

Modèle type

```
pour chaque c dans clients
  si c.retard = 0
    boucler
  sinonsi c.retard > 30
    ? "À relancer :", c.nom
  finsi
suivant
tantque qte = nul ou qte <= 0
  qte = nombre(saisir("Qté ? "))
fintantque
```

Conditions

`si cond … finsi`
exécute si la condition est vraie

`sinonsi cond`
autre condition (ou « sinon si »)

`sinon`
aucune condition vraie

`faire cas … fincas`
choix multiple

`cas cond`
une branche

`autrement`
tous les autres cas

`sisinon(condition, si_vrai, si_faux)`
Valeur selon la condition

`vide(v)`
Vrai pour nul, faux, 0, une chaîne blanche ou une table sans élément (EMPTY)

`estnul(v)`
Vrai si v vaut nul

Boucles

`pour i = a a b [pas p] … suivant`
compteur (a / à / jusqua ; finpour)

`pour chaque v dans liste … suivant`
chaque valeur

`pour chaque k, v dans fiche`
chaque clef et valeur

`pour chaque v dans @ITERATEUR`
procédure appelée jusqu'à nul

`tantque cond … fintantque`
tant que (ou faire tantque)

`sortir`
quitte la boucle

`boucler`
passe au tour suivant

Pièges à éviter

⚠ pour se ferme par suivant, tantque par fintantque, si par finsi.

⚠ Les bornes d'un pour sont calculées une seule fois.

⚠ Un tantque dont la condition reste vraie ne s'arrête jamais (Maj+F5).

⚠ Un itérateur renvoie nul pour arrêter la boucle.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr



Aide-mémoire · Procédures

Nommer un traitement pour le réutiliser partout.

Voir aussi : Tables & objets · Multitâche · Interface graphique

13 instructions et fonctions

ENCHAÎNEMENTS `procedure NOM(a)` → `parametres ?` → `retourner v` → `finprocedure ?` — `@NOM` → `appeler(@NOM, …) · ref(…)` → `trier · surclic · tache.lancer`

Modèle type

```
? TTC(100), TTC(100, 5.5)

fonction TTC(ht, taux)
  taux = taux ou 20
retourner arrondi(ht * (1 + taux /
100), 2)

trier(liste, @PLUS_GRAND)
fonction PLUS_GRAND(a, b)
retourner a > b
```

Définir

`procedure NOM(a, b)`
début d'une procédure

`fonction NOM(a, b)`
synonyme

`parametres a, b`
lire les arguments (style dBase)

`retourner [valeur]`
fin et résultat (aussi « retour »)

`finprocedure · finfonction`
fin explicite

`argument manquant`
vaut nul : x = x ou défaut

Appeler

`NOM(a, b)`
appel

`x = NOM(a, b)`
appel avec résultat

`faire NOM avec a, b`
appel style dBase

`@NOM`
référence (sans appeler)

`ref(a, b)`
appel d'une référence rangée dans une variable

`appeler(@proc, ...)`
Appelle une procédure référencée

`récursivité`
une procédure s'appelle elle-même (200 niveaux au plus)

Pièges à éviter

⚠ NOM appelle, @NOM désigne sans appeler.

⚠ Les tables passées en argument sont partagées (copier() pour une copie).

⚠ Les gestionnaires d'événements ne voient que les globales.

⚠ Toujours un cas d'arrêt dans une fonction récursive (200 niveaux).

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr



Aide-mémoire · Tables & objets

Listes, fiches, objets et métatables.

Voir aussi : Procédures · Fichiers & formats · Texte & calcul

33
instructions et fonctions

ENCHAÎNEMENTS `{ }` → `inserer · retirer` → `trier` → `joindre · json` — `Classe = {}` → `__index` → `:procedure … comme` → `fixermetatable` → `obj:methode()`

Modèle type

```
locale l = {"b", "a"}
inserer "c" dans l
trier(l)
? joindre(l, ", "), #l
locale f = {nom = "Ada", age = 36}
f.ville = "Brest"
pour chaque k, v dans f
  ? k, v
suivant
```

Créer et accéder

`{a, b, c}`
liste (indices à partir de 1)

`{cle = v}`
fiche

`t[i] · t.cle · t["CLE"]`
accès

`nouvelle table t`
t = {}

`supprimer table t`
t = nul

`#t`
longueur de la liste

Modifier

`inserer v dans t [position p]`
ajoute

`inserer v dans t clef "K"`
ajoute sous une clef

`retirer de t position p dans x`
enlève

`inserer(t, v, [position])`
Ajoute v (à la fin ou à la position)

`retirer(t, [position])`
Enlève et renvoie un élément (le dernier par défaut)

`trier(t, [@comparer])`
Trie la liste sur place

`copier(t)`
Copie de premier niveau

Interroger

`taille(t)`
Nombre d'éléments de la partie liste (#t)

`nbelements(t)`
Nombre total d'éléments

`clefs(t)`
Liste des clefs

`valeurs(t)`
Liste des valeurs

`aclef(t, k)`
Vrai si la clef existe

`contient(s, sous)`
Vrai si s contient sous

`joindre(liste, [separateur])`
Texte des éléments d'une liste

`decouper(s, [separateur])`
Liste des morceaux (séparateur « , » par défaut)

`json(v)`
Texte JSON d'une valeur

`dejson(s)`
Valeur d'un texte JSON (clefs d'objets en majuscules)

Objets et métatables

`Classe.__index = Classe`
méthodes partagées

`Classe:procedure P comme m`
range une méthode

`obj:m(…)`
appel de méthode (obj en premier argument)

`fixermetatable t, mt`
forme instruction

`fixermetatable(t, mt)`
Donne une métatable

`obtenirmetatable(t)`
La métatable ou nul

`lirebrut(t, k)`
Lecture sans métaméthode

`ecrirebrut(t, k, v)`
Écriture sans métaméthode

`__index __newindex __call __tostring __len`
accès, appel, texte, longueur

`__eq __lt __le __add __sub __mul __div __mod __pow __unm __concat`
opérateurs

Pièges à éviter

- ⚠ Les indices commencent à 1.
- ⚠ b = a ne copie pas : b et a sont la même table.
- ⚠ f.nom = f.NOM = f["NOM"] mais f["nom"] est une autre clef.
- ⚠ Une table vide s'encode {} en JSON.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Aide-mémoire · Texte & calcul

La boîte à outils toujours disponible : texte, nombres, dates, sécurité.

Voir aussi : Les bases · Système · Tables & objets

55 instructions et fonctions

ENCHAÎNEMENTS `texte` → `decouper()` → `liste` → `joindre()` → `texte` | `hachermdp(mdp)` → `empreinte en base` → `verifiermdp(saisie, empreinte)`

Modèle type

```
locale l = "2026-10-16;DUR-0042;1250,80"
locale c = decouper(l, ";")
? gauche(c[1], 4), droite(c[2], 4)
? valeur(c[3]) * 1.2
? chaine(3.14159, 8, 2), arrondi(2.345, 2)
? hachermdp("secret") <> nul
```

Texte

majuscule(s)
Texte en majuscules (accents compris)

minuscule(s)
Texte en minuscules

rogner(s)
Retire les espaces aux deux bouts (ALLTRIM)

rognerg(s)
Retire les espaces à gauche (LTRIM)

rognerd(s)
Retire les espaces à droite (RTRIM)

longueur(s)
Nombre de caractères (ou d'éléments d'une liste)

souschaine(s, debut, [n])
Extrait n caractères à partir de debut (SUBSTR)

gauche(s, n)
Les n premiers caractères

droite(s, n)
Les n derniers caractères

position(sous, s)
Position de sous dans s, 0 si absente (AT)

espace(n)
n espaces (SPACE)

repliquer(s, n)
s répété n fois (REPLICATE)

remplacertexte(s, a, b)
Remplace toutes les occurrences de a par b (STRTRAN)

contient(s, sous)
Vrai si s contient sous

commencepar(s, debut)
Vrai si s commence par debut

finitpar(s, fin)
Vrai si s finit par fin

decouper(s, [separateur])
Liste des morceaux (séparateur « , » par défaut)

joindre(liste, [separateur])
Texte des éléments d'une liste

chaine(v, [largeur], [decimales])
Texte d'une valeur

html(s)
Échappe < > & " ' pour le HTML

url(s)
Encode un texte pour une URL

car(code, ...)
Caractères à partir de leurs codes Unicode

code(s, [i])
Code Unicode du i-ème caractère

Nombres

abs(x)
Valeur absolue

entier(x)
Partie entière, vers zéro (INT)

arrondi(x, [decimales])
Arrondi (0 décimale par défaut)

plancher(x)
Entier inférieur

plafond(x)
Entier supérieur

racine(x)
Racine carrée

mod(a, b)
Reste de la division, du signe du diviseur

max(a, b, ...)
Plus grande valeur (nombres ou chaînes)

min(a, b, ...)
Plus petite valeur

aleatoire([n])
Réel entre 0 et 1, ou entier entre 1 et n

sin(x)
Sinus (radians)

cos(x)
Cosinus

tan(x)
Tangente

asin(x)
Arc sinus

acos(x)
Arc cosinus

atan(y, [x])
Arc tangente (atan2 avec deux arguments)

exp(x)
Exponentielle

log(x, [base])
Logarithme (népérien par défaut)

pi()
π

Dates

date()
Date du jour, AAAA-MM-JJ

heure()
Heure, HH:MM:SS

maintenant()
Date et heure

horodatage()
Secondes depuis le 1er janvier 1970 (avec décimales)

Tests

type(v)
"nul", "logique", "nombre", "chaine", "table", "procedure", "objet" ou "tache"

estnul(v)
Vrai si v vaut nul

vide(v)
Vrai pour nul, faux, 0, une chaîne blanche ou une table sans élément (EMPTY)

sisinon(condition, si_vrai, si_faux)
Valeur selon la condition

Sécurité

hachage(s)
Empreinte SHA-256 en hexadécimal (pas pour les mots de passe)

hachermdp(mdp)
Empreinte PBKDF2-SHA256 salée d'un mot de passe

verifiermdp(mdp, empreinte)
Vérifie un mot de passe (temps constant)

mdpaleatoire([n])
Mot de passe aléatoire (16 caractères par défaut)

uuid()
Identifiant unique aléatoire

Pièges à éviter

⚠ 0.1 + 0.2 ne vaut pas exactement 0.3 : arrondissez les montants.

⚠ longueur() compte des caractères, pas des octets.

⚠ hachage() n'est pas fait pour les mots de passe : hachermdp().

⚠ html() et url() pour tout texte venu de l'utilisateur.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Aide-mémoire · Erreurs

Signaler, intercepter et traiter les erreurs.

Voir aussi : Bases de données · Procédures · Décider & répéter

12 instructions et fonctions

ENCHAÎNEMENTS

`essayer` → `instructions` → `capturer e` → `finessayer`

`erreur …` → `remonte la pile` → `premier capturer`

`begin transaction` → `essayer` → `commit / rollback`

Modèle type

```
essayer
  DEBITER(article, qte)
capturer e
  ? "Refus :", e
finessayer

procedure DEBITER(a, q)
  si q <= 0
    erreur "quantité invalide"
  finsi
retourner
```

Intercepter

`essayer`
début du bloc surveillé

`capturer [e]`
e reçoit l'erreur (texte ou table)

`finessayer`
fin du bloc

Signaler

`erreur "message"`
lève une erreur

`erreur {code = …, message = …}`
erreur structurée

`relancer`
erreur ... dans capturer

`systeme.quitter(code)`
arrêt volontaire

Messages fréquents

`division par zéro`
diviseur nul

`indexation d'une valeur nulle`
t vaut nul

`procédure ou fonction inconnue`
faute de frappe, fichier absent

`nombre attendu, chaine reçu`
calcul sur un texte

`profondeur d'appel maximale`
récursivité sans fin

Pièges à éviter

⚠ Un capturer vide masque les problèmes : affichez ou journalisez.

⚠ L'erreur remonte jusqu'au premier essayer, même loin dans les appels.

⚠ Une erreur non interceptée arrête le programme (code 1).

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Aide-mémoire · Bases de données

DuckDB intégré (SQL et zones dBase), SQLite, Firebird.

Voir aussi : Erreurs · Fichiers & formats · Tables & objets

60 instructions et fonctions

ENCHAÎNEMENTS

`ouvrirbase()` → `sql "…" avec … dans l` → `pour chaque r dans l`

`utiliser` → `parcourir [pour]` → `finparcourir` → `fermer`

`chercher · localiser` → `trouve()` → `continuer`

`sqlite.ouvrir · firebird.connecter` → `db:executer · db:requete` → `db:transaction(@P)` → `db:fermer()`

Modèle type

```
ouvrirbase(":memoire:")
sql "create table c(code varchar,
nom varchar)"
sql "insert into c values (?, ?)"
avec "C1", "Ada"
pour chaque r dans requete("select
* from c")
  ? r.code, r.nom
suivant
utiliser c ordre "nom"
parcourir
  ? numenr(), nom
finparcourir
```

Base DuckDB

`ouvrirbase(chemin)`
Ouvre une base DuckDB (":memoire:" pour une base en mémoire)

`fermerbase()`
Ferme la base et les zones

`base()`
Chemin de la base ouverte ("" si aucune)

`requete(sql, ...)`
Liste des lignes d'une requête DuckDB (paramètres « ? »)

`valeursql(sql, ...)`
Première valeur de la première ligne, ou nul

`sql "…" [avec v, …] [dans liste]`
exécute du SQL (paramètres ?)

`begin transaction · commit · rollback`
transactions (via sql)

Zones (dBase)

`utiliser t [alias a] [ordre "…"] [filtre "…"]`
ouvre une zone (USE)

`selectionner a`
zone courante (SELECT)

`fermer [a | tout]`
ferme (CLOSE)

`aller haut | bas | n`
se positionner (GO)

`sauter [n]`
avancer (SKIP)

`chercher v`
recherche sur l'ordre (SEEK)

`localiser pour cond`
LOCATE

`continuer`
suivant (CONTINUE)

`ajouter [vide | avec c par v, …]`
APPEND

`remplacer c par v, …`
REPLACE

`effacer`
DELETE

`parcourir [pour cond] … finparcourir`
SCAN

`compter [pour cond] dans x`
COUNT

`rafraichir`
relit la zone

`alias->champ · m->var`
champ d'une zone · variable

`fdf([alias])`
Fin de fichier de la zone (EOF)

`ddf([alias])`
Début de fichier (BOF)

`numenr([alias])`
Numéro de l'enregistrement courant (RECNO)

`nbenr([alias])`
Nombre d'enregistrements (RECCOUNT)

`trouve([alias])`
Résultat du dernier chercher / localiser (FOUND)

`alias()`
Alias de la zone courante

`champ(nom, [alias])`
Valeur d'un champ par son nom (FIELD)

`enregistrement([alias])`
L'enregistrement courant sous forme de table

SQLite

`sqlite.ouvrir(chemin)`
Ouvre (ou crée) une base

`sqlite.version()`
Version de la bibliothèque SQLite

`db:executer(sql, ...)`
Exécute une instruction avec paramètres « ? »

`db:requete(sql, ...)`
Liste des lignes

`db:premier(sql, ...)`
Première ligne ou nul

`db:lignes(sql, ...)`
Itérateur sur les lignes : pour chaque r dans db:lignes(sql)

`db:dernierid()`
Identifiant de la dernière ligne insérée

`db:modifications()`
Lignes modifiées par la dernière instruction

`db:transaction(@proc)`
Appelle proc(db) dans une transaction (annulée en cas d'erreur)

`db:fermer()`
Ferme la base

Firebird

`firebird.connecter(base, utilisateur, motdepasse[, options])`
Connexion : "serveur:/chemin/base.fdb", "serveur/3050:alias" ou fichier local

`firebird.creer(base, utilisateur, motdepasse[, {taillepage=8192, jeu="UTF8"}])`
Crée une base puis s'y connecte

`firebird.version()`
Version de la bibliothèque cliente

`firebird.disponible()`
Vrai si la bibliothèque cliente est trouvée

`db:executer(sql, ...)`
Exécute une instruction avec paramètres « ? »

`db:requete(sql, ...)`
Liste des lignes

`db:premier(sql, ...)`
Première ligne ou nul

`db:valeur(sql, ...)`
Première colonne de la première ligne ou nul

`db:lignes(sql, ...)`
Itérateur sur les lignes : pour chaque r dans db:lignes(sql)

`db:modifications()`
Lignes modifiées par le dernier executer()

`db:sequence(nom[, pas=1])`
Valeur suivante d'une séquence (générateur)

`db:transaction(@proc[, options])`
Appelle proc(db) dans une transaction (validée, ou annulée en cas d'erreur)

`db:debut([{isolation="lecturevalidee lecture=faux, attente=vrai|faux|secondes}])`
Commence une transaction explicite

`db:valider()`
Valide la transaction explicite

`db:annuler()`
Annule la transaction explicite

`db:entransaction()`
Vrai si une transaction explicite est en cours

`db:tables()`
Tables de l'utilisateur

`db:colonnes(table)`
Colonnes d'une table : {nom=, type=, longueur=, echelle=, obligatoire=}

`db:serveur()`
Version du serveur

`db:fermer()`
Ferme la connexion (une transaction explicite non validée est annulée)

Pièges à éviter

⚠ Jamais de valeurs collées dans le SQL : paramètres ? et avec.

⚠ update / delete sans where modifient TOUTES les lignes.

⚠ Après parcourir, la zone est en fin de fichier : aller haut.

⚠ Les colonnes se lisent comme des noms : r.nom = r.NOM.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Aide-mémoire · Fichiers & formats

Lire et écrire fichiers, dossiers, CSV, JSON et binaire.

Voir aussi : Bases de données · Tables & objets · Système

54 instructions et fonctions

ENCHAÎNEMENTS

`es.ouvrir(c, mode)` → `f:lire · f:ecrire` → `f:fermer()`

`csv.lire` → `liste de fiches` → `csv.ecrire`

`json.lire` → `table` → `json.ecrire`

`binaire.coder(fmt, …)` → `octets` → `binaire.decoder(fmt, d)`

Modèle type

```
pour chaque l dans
es.lignes("journal.log")
  ? l
suivant
locale t =
csv.lire("ventes.csv", {sep =
";", entete = vrai, convertir =
vrai})
json.ecrire("ventes.json", t)
fichier.ajouter("journal.log",
date() + car(10))
```

fichier

fichier.lire(chemin)
Contenu complet d'un fichier (texte ou binaire)

fichier.lireoctets(chemin)
Contenu binaire (les chaînes sont des suites d'octets)

fichier.ecrire(chemin, donnees)
Remplace le contenu d'un fichier

fichier.ecrireoctets(chemin, donnees)
Écrit des octets

fichier.ajouter(chemin, donnees)
Ajoute à la fin d'un fichier

fichier.lignes(chemin)
Liste des lignes d'un fichier

fichier.ecrirelignes(chemin, lignes)
Écrit une liste de lignes

fichier.existe(chemin)
Vrai si le fichier ou le dossier existe

fichier.estfichier(chemin)
Vrai pour un fichier

fichier.estdossier(chemin)
Vrai pour un dossier

fichier.taille(chemin)
Taille en octets

fichier.liste([dossier], [recursif])
Noms contenus dans un dossier (les dossiers finissent par /)

fichier.creerdossier(chemin)
Crée un dossier (et ses parents)

fichier.supprimer(chemin)
Supprime un fichier ou un dossier (récursif)

fichier.copier(source, destination)
Copie (dossiers récursivement)

fichier.deplacer(source, destination)
Déplace ou renomme

fichier.infos(chemin)
Table {taille, genre ("fichier" ou "dossier"), modifie, cree} ou nul

fichier.courant()
Dossier courant

fichier.changer(dossier)
Change de dossier courant

fichier.joindre(a, b, ...)
Assemble des chemins

fichier.nom(chemin)
Nom du fichier

fichier.dossier(chemin)
Dossier parent

fichier.extension(chemin)
Extension

fichier.absolu(chemin)
Chemin absolu

fichier.dossiertemp()
Dossier temporaire

fichier.dossierperso()
Dossier personnel

es

es.ecrire(...)
Écrit les valeurs (sans séparateur ni fin de ligne) sur la sortie : console...

es.lire([format])
Lit l'entrée : "l" une ligne (défaut), "n" un nombre, "L" une ligne avec sa...

es.lignes([chemin])
Itérateur sur les lignes d'un fichier (ou de l'entrée) : pour chaque l dans...

es.ouvrir(chemin, [mode])
Ouvre un fichier : "l" lecture (défaut), "e" écriture, "a" ajout, "l+", "e+"

es.type(v)
"fichier", "fichier fermé" ou nul

es.sortie
Sortie standard

es.erreur
Sortie d'erreur

es.entree
Entrée standard

f:lire([format])
Lit : "l", "L", "n", "t" (tout) ou un nombre d'octets

f:ecrire(...)
Écrit des valeurs

f:lignes()
Itérateur sur les lignes restantes

f:position([origine], [decalage])
Déplace la position ("debut", "courant", "fin") et la renvoie

f:vider()
Vide les tampons

f:fermer()
Ferme le fichier

csv

csv.analyser(texte, [options])
Liste des lignes d'un texte CSV

csv.lire(chemin, [options])
Lit un fichier CSV (mêmes options)

csv.encoder(lignes, [options])
Texte CSV

csv.ecrire(chemin, lignes, [options])
Écrit un fichier CSV

json

json.encoder(valeur, [joli])
Texte JSON

json.decoder(texte, [exact])
Valeur Lexis+

json.lire(chemin, [exact])
Lit et décode un fichier JSON

json.ecrire(chemin, valeur, [joli])
Encode et écrit un fichier JSON (indenté par défaut)

json.nul
Valeur représentant null dans les données à encoder

binaire

binaire.coder(format, ...)
Code des valeurs en binaire

binaire.decoder(format, donnees, [position])
Décode

binaire.taille(format)
Taille d'un format fixe

binaire.hex(donnees)
Représentation hexadécimale

binaire.dehex(hex)
Octets à partir de l'hexadécimal

Pièges à éviter

⚠ fichier.supprimer() est définitif et récursif.

⚠ fichier.joindre() pour construire les chemins.

⚠ json.decoder met les clefs en majuscules, sauf exact = vrai.

⚠ es.lignes() pour les gros fichiers (pas de chargement complet).

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Aide-mémoire · Système

Dates, environnement, commandes, arguments.

Voir aussi : Texte & calcul · Fichiers & formats · Démarrer

22 instructions et fonctions

ENCHAÎNEMENTS

`systeme.temps({…})` → `systeme.ecart()` → `/ 86400 = jours`

`systeme.capturer(cmd)` → `{sortie, code}`

`systeme.args` → `traitement` → `systeme.quitter(code)`

Modèle type

```
? systeme.date("%A %d %B %Y")
locale n = systeme.temps({annee = 2026, mois
= 12, jour = 25})
? arrondi(systeme.ecart(n, systeme.temps())
/ 86400), "jours"
locale r = systeme.capturer("ls")
si #systeme.args < 1
  systeme.quitter(2)
finsi
```

systeme

`systeme.temps([date])`
Heure actuelle (secondes Unix), ou date d'une table {annee, mois, jour,...

`systeme.horloge()`
Temps écoulé depuis le lancement, en secondes (horloge monotone)

`systeme.date([format], [t])`
Date formatée : %Y %m %d %H %M %S %A (jour) %B (mois) %F %T %x

`systeme.ecart(t2, t1)`
Différence en secondes

`systeme.env(nom)`
Variable d'environnement (nul si absente)

`systeme.fixerenv(nom, valeur)`
Définit une variable d'environnement

`systeme.executer(commande)`
Exécute une commande système

`systeme.capturer(commande)`
Exécute une commande

`systeme.supprimer(chemin)`
Supprime un fichier ou un dossier

`systeme.renommer(ancien, nouveau)`
Renomme un fichier

`systeme.nomtemporaire()`
Nom de fichier temporaire

`systeme.quitter([code])`
Termine le programme

`systeme.attendre(secondes)`
Pause (les fenêtres restent réactives)

`systeme.plateforme()`
"linux", "macosx", "windows"

`systeme.dossierprogramme()`
Dossier du programme principal

`systeme.args`
Arguments du programme

Formats de systeme.date

`%Y %m %d`
année, mois, jour

`%H %M %S`
heure, minute, seconde

`%A %a %B %b`
jour, mois (en lettres)

`%F %T %x %X %c`
formats composés

`%j %w %y %Z %%`
jour de l'année, de la semaine...

`"*t" "!…"`
fiche de date ; heure UTC

Pièges à éviter

⚠ Une journée ne fait pas toujours 86 400 s (changement d'heure) : arrondissez.

⚠ Les commandes du système diffèrent entre Linux, macOS et Windows.

⚠ Les secrets se lisent dans l'environnement : systeme.env().

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Aide-mémoire · Réseau

HTTP, TCP, UDP, FTP, SSH et SFTP.

Voir aussi : Multitâche · Fichiers & formats · Système

41 instructions et fonctions

ENCHAÎNEMENTS

`http.requete{…}` → `{statut, corps, entetes}` → `dejson(corps)`

`reseau.ecouter(port)` → `s:accepter()` → `c:recevoir · c:envoyer` → `c:fermer()`

`reseau.connecter(h, p)` → `s:envoyer · s:recevoir` → `s:fermer()`

`ssh.connecter{…}` → `s:executer(cmd)` → `{sortie, code}`

Modèle type

```
locale r = http.requete{url =
"https://api.exemple.fr/x", delai = 10}
si r.statut = 200
  locale d = dejson(r.corps)
finsi
locale s = reseau.connecter("srv", 7000)
s:envoyer("PING\n")
? s:recevoir()
s:fermer()
```

http

`http.lire(url, [entetes])`
Requête GET

`http.envoyer(url, corps, [entetes])`
Requête POST

`http.requete{methode=, url=, entetes=, corps=, delai=}`
Requête quelconque

`http.telecharger(url, chemin)`
Télécharge dans un fichier

`http.encoder(texte | table)`
Encodage d'URL (une table devient a=1&b=2)

`http.decoder(texte)`
Décodage d'URL

reseau

`reseau.connecter(hote, port, [options])`
Connexion TCP

`reseau.ecouter(port, [hote])`
Socket serveur TCP

`reseau.udp([port], [hote])`
Socket UDP, éventuellement lié à un port

`reseau.resoudre(hote)`
Adresse IP d'un nom

`reseau.nomhote()`
Nom de la machine

s (socket)

`s:envoyer(donnees)`
Envoie des octets

`s:recevoir([mode])`
Reçoit : "l" une ligne (défaut), "t" tout jusqu'à la fermeture, ou un nombre…

`s:delai(secondes)`
Délai maximal des réceptions (nul = aucun)

`s:accepter()`
Attend un client (socket serveur)

`s:envoyervers(donnees, port, hote)`
Envoi UDP

`s:recevoirde([taille])`
Réception UDP : {donnees, adresse, port}

`s:distant()`
{adresse, port} de l'autre extrémité

`s:fermer()`
Ferme le socket

ftp

`ftp.connecter{hote=, port=21, utilisateur="anonymous", motdepasse=""}`
Ouvre une session FTP (mode passif)

`c:liste([chemin], [detail])`
Noms des fichiers (detail = vrai : format détaillé)

`c:lire(distant, [local])`
Télécharge : renvoie les données, ou les écrit dans le fichier local

`c:envoyer(local_ou_donnees, [distant], [sontdesdonnees])`
Envoie un fichier local (ou des données)

`c:dossier(chemin)`
Change de dossier distant

`c:courant()`
Dossier distant courant

`c:creerdossier(chemin)`
Crée un dossier

`c:supprimerdossier(chemin)`
Supprime un dossier

`c:supprimer(chemin)`
Supprime un fichier

`c:renommer(ancien, nouveau)`
Renomme

`c:fermer()`
Ferme la session

ssh

`ssh.connecter{hote=, utilisateur=, port=22, clef=, motdepasse=, options={}}`
Ouvre une session (vérifie la connexion)

`ssh.executer{hote=, utilisateur=, commande=, ...}`
Exécute une commande sans garder de session : {sortie, code}

`session:executer(commande)`
Exécute une commande distante : {sortie, code}

`session:lire(distant, [local])`
Télécharge un fichier ou dossier (scp)

`session:envoyer(local, [distant])`
Envoie un fichier ou dossier (scp)

`session:liste([chemin])`
Noms d'un dossier distant (sftp)

`session:creerdossier(chemin)`
Crée un dossier distant (sftp)

`session:supprimerdossier(chemin)`
Supprime un dossier distant (sftp)

`session:supprimer(chemin)`
Supprime un fichier distant (sftp)

`session:renommer(ancien, nouveau)`
Renomme (sftp)

sftp

`sftp.connecter{hote=, utilisateur=, port=22, clef=, motdepasse=}`
Ouvre une session de transfert (identique à ssh.connecter)

Pièges à éviter

⚠ Les en-têtes de réponse sont en minuscules : r.entetes["content-type"].

⚠ Fixez un délai (delai = …) pour ne pas attendre indéfiniment.

⚠ Jamais de mot de passe dans le code : systeme.env() ou clef SSH.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Aide-mémoire · Multitâche

Tâches coopératives et fils d'exécution.

Voir aussi : Réseau · Procédures · Intelligence artificielle

22 instructions et fonctions

ENCHAÎNEMENTS

`tache.lancer(@P)` → `tache.executer()` → `t:attendre()`

`tache.canal()` → `c:envoyer(v)` → `c:recevoir()`

`fil.demarrer(code)` → `f:envoyer ⇄ fil.recevoir` → `fil.envoyer ⇄ f:recevoir` → `f:joindre()`

Modèle type

```
tache.lancer(@MINUTEUR, "pâtes", 0.3)
tache.lancer(@MINUTEUR, "œufs", 0.2)
tache.executer()
locale f = fil.demarrer([[
fil.envoyer(fil.recevoir() * 2) ]])
f:envoyer(21)
? f:recevoir()
f:joindre()
```

tache

`tache.lancer(@proc, ...)`
Crée une tâche qui exécutera proc(...)

`tache.executer()`
Fait tourner les tâches jusqu'à ce qu'elles soient toutes terminées

`tache.attendre(secondes)`
Dans une tâche : la suspend

`tache.ceder()`
Rend la main aux autres tâches

`tache.canal([capacite])`
Canal de communication (capacité 0 = illimitée)

`tache.nombre()`
Nombre de tâches en cours

`tache.maintenant()`
Heure en secondes (avec décimales)

`t:attendre()`
Attend la fin d'une tâche et renvoie son résultat

`t:etat()`
"suspendue", "en cours", "terminee" ou "erreur"

`t:annuler()`
Arrête une tâche suspendue

`c:envoyer(valeur)`
Envoie une valeur (attend si le canal est plein)

`c:recevoir([delai])`
Reçoit une valeur (attend si vide)

`c:nombre()`
Nombre de valeurs en attente

fil

`fil.demarrer(code_ou_fichier, ...)`
Lance un fil qui exécute du code Lexis+ (chaîne) ou un fichier .prg, avec...

`fil.envoyer(valeur)`
Dans le fil : envoie une valeur au programme qui l'a lancé

`fil.recevoir([delai])`
Dans le fil : reçoit une valeur (nul après le délai)

`fil.estenfant()`
Vrai dans un fil lancé par fil.demarrer

`f:envoyer(valeur)`
Envoie une valeur au fil (tables, chaînes, nombres, logiques)

`f:recevoir([delai])`
Reçoit une valeur du fil

`f:actif()`
Vrai tant que le fil s'exécute

`f:joindre()`
Attend la fin du fil

`f:arreter()`
Demande l'arrêt du fil

Pièges à éviter

⚠ Une attente bloquante (réseau, systeme.attendre) bloque toutes les tâches : utilisez un fil.

⚠ Un fil ne partage aucune variable : uniquement des messages.

⚠ Fichier de fil : sous-dossier, uniquement des procédures dont PRINCIPAL.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Aide-mémoire · Interface graphique

Fenêtres wdgui : contrôles, événements, dialogues.

Voir aussi : Pages web · Procédures · Bases de données

154 instructions et fonctions

ENCHAÎNEMENTS

`ui = paquet("interface")` → `ui.fenetre{…}` → `fen:ajouter(ui.bouton(…))` → `btn.surclic = @P` → `ui.executer()`

`PROC(ev)` → `ev.genre · ev.x · ev.texte · ev.chemin` → `ev:arreter()`

Modèle type

```
ui = paquet("interface")
fen = ui.fenetre{titre = "Démo",
largeur = 360, hauteur = 160}
lbl = fen:ajouter(ui.etiquette("",
"centre"))
btn = fen:ajouter(ui.bouton("OK",
{defaut = vrai}))
btn.surclic = @CLIC
ui.executer()
procedure CLIC(ev)
  lbl.libelle = "Cliqué à " +
heure()
retourner
```

Fenêtre et boucle

`interface.fenetre{titre=, largeur=, hauteur=, disposition=, theme=, sombre=}`
Crée une fenêtre

`interface.fenetres()`
Nombre de fenêtres ouvertes

`interface.executer()`
Attend la fermeture des fenêtres (atelier) ou lance la boucle de wdgui…

`interface.quitter()`
Ferme toutes les fenêtres du programme

`interface.fenetres()`
Nombre de fenêtres ouvertes

`interface.tracer([actif])`
Affiche dans la console chaque événement reçu (diagnostic)

`interface.separateur{vertical=, position=}`
Séparateur (deux enfants)

`interface.sep`
Séparateur des chemins d'arbre et de menu

`fen:fermer()`
Ferme la fenêtre

`fen:fixertheme(nom, [sombre])`
Change le thème

`c:ajouter(enfant, ...)`
Ajoute des contrôles à un conteneur ou une fenêtre

`c:focus()`
Donne le focus

`c:menucontextuel(element, ...)`
Menu contextuel

`c:redessiner()`
Redessine le contrôle

Contrôles

`interface.etiquette(texte, [alignement])`
Libellé (alignement : gauche, centre, droite)

`interface.bouton(libelle, [{defaut=, annuler=}])`
Bouton

`interface.saisie(texte | {texte=, indication=, multiligne=})`
Champ de saisie

`interface.cases(option, ...)`
Cases à cocher

`interface.radio(option, ...)`
Boutons radio

`interface.liste([multi])`
Liste

`interface.combo(element, ...)`
Liste déroulante

`interface.tableau()`
Tableau simple

`interface.arbre()`
Arbre

`interface.grille{multi=, remplissage="programme|manuel|automa`
Grille de saisie

`interface.jauge()`
Jauge

`interface.curseur()`
Potentiomètre

`interface.plage()`
Potentiomètre de plage

`interface.ascenseur()`
Ascenseur

`interface.compteur()`
Compteur à flèches

`interface.notation()`
Notation

`interface.image(chemin)`
Image

`interface.codebarre(texte)`
Code-barres

`interface.graphe([genre], [titre])`
Graphe : colonnes, lignes, aires, secteurs

`interface.calendrier(["AAAAMMJJ"])`
Calendrier

`interface.kanban()`
Kanban

`interface.mosaique()`
Mosaïque proportionnelle (treemap)

`interface.gantt()`
Gantt

`interface.barremenu()`
Barre de menus

`interface.onglets()`
Onglets

`interface.separateur{vertical=, position=}`
Séparateur (deux enfants)

`interface.conteneur([disposition], [colonnes])`
Conteneur

`interface.colonne()`
Conteneur vertical

`interface.rangee()`
Conteneur horizontal

`interface.superconteneur([dispositic`
Conteneur sans marge

`interface.cadre(titre, [disposition])`
Cadre de groupe

`interface.panneau{disposition=, defilement=, colonnes=}`
Conteneur défilant (defilement : toujours, automatique, masque)

`interface.pdf()`
Visionneuse PDF (PDFium)

Dialogues

`interface.alerte(message, [icone], [titre])`
Message (icone : information, question, attention, stop, aucune)

`interface.confirmer(message, [icone])`
OK / Annuler

`interface.demander(message, [defaut])`
Saisie d'un texte

`interface.ouvrirfichier{titre=, filtre=, dossier=}`
Choix d'un fichier à ouvrir

`interface.enregistrerfichier{nom=, filtre=}`
Choix d'un fichier à enregistrer

`interface.couleur(["#RRGGBB"])`
Sélecteur de couleur

`interface.police([famille], [taille])`
Sélecteur de police : {famille, taille, gras, italique}

Listes, tableaux, arbres

`liste:ajouter(element, ...)`
Ajoute (listes et combos)

`liste:inserer(i, element)`
Insère

`liste:supprimer(i)`
Supprime

`liste:vider()`
Vide

`liste:selectionner(i)`
Sélectionne

`liste:selection()`
Index sélectionné

`liste:nombre()`
Nombre d'éléments

`liste:element(i)`
Texte d'un élément

`liste:modifier(i, texte)`
Modifie

`liste:chercher(texte)`
Recherche

`liste:deselectionner(i)`
Désélectionne

`tableau:ajoutercolonne(titre, [largeur], [alignement])`
Ajoute une colonne

`tableau:ajouterligne(cellule, ...)`
Ajoute une ligne

`tableau:cellule(ligne, colonne)`
Texte d'une cellule

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

tableau:fixercellule(ligne, colonne, texte)
Modifie une cellule

tableau:selection()
Ligne sélectionnée

tableau:trier(colonne, [croissant])
Trie

tableau:insererligne(i, cellule, ...)
Insère une ligne

tableau:ligne(i)
Cellules d'une ligne

tableau:nombre()
Nombre de lignes

tableau:selectionner(i)
Sélectionne une ligne

tableau:supprimer(i)
Supprime une ligne

tableau:vider()
Vide le tableau

arbre:ajouter(chemin...)
Ajoute un nœud ("A", "B" ou "A/B")

arbre:deplier(chemin...)
Déplie

arbre:replier(chemin...)
Replie

arbre:supprimer(chemin...)
Supprime

arbre:vider()
Vide

arbre:estdeplie(chemin...)
Vrai si le nœud est déplié

arbre:selectionner(chemin...)
Sélectionne un nœud

arbre:nombre()
Nombre de nœuds

onglets:ajouterpage(titre, [disposition])
Ajoute une page

barremenu:ajouter(chemin...)
Ajoute un élément de menu ("Fichier", "Ouvrir")

cases:coche([i])
État d'une case

cases:cocher(i, [b])
Coche une case

cases:ajouteroption(texte)
Ajoute une case

radio:ajouteroption(texte)
Ajoute un bouton radio

Grille

grille:ajoutercolonne{nom=, titre=, genre=, largeur=, choix=, decimales=}
Ajoute une colonne (genre : texte, nombre, date, image, bouton, combo, case)

grille:ajouterligne(cellule, ...)
Ajoute une ligne

grille:cellule(ligne, colonne)
Valeur d'une cellule (colonne = nom ou index)

grille:fixercellule(ligne, colonne, valeur)
Modifie une cellule

grille:nombre()
Nombre de lignes

grille:trier(colonne, [croissant])
Trie

grille:chercher(colonne, valeur, [exact], [debut])
Cherche une ligne

grille:fixerpied(colonne, genre)
Pied de colonne : somme, moyenne, min, max, nombre

grille:ajouterformat(colonne, operateur, valeur, {fond=, texte=, ligneentiere=})
Mise en forme conditionnelle

grille:fixermodifiable([b])
Mode saisie

grille:verscsv([sep])
Export CSV

grille:chargercsv(chemin, [sep])
Import CSV

grille:versjson()
Export JSON

grille:chargerjson(chemin)
Import JSON

grille:insererligne(i, ...)
Insère une ligne

grille:modifierligne(i, ...)
Remplace une ligne

grille:supprimerligne(i)
Supprime une ligne

grille:vider()
Vide la grille

grille:ligne(i)
Cellules d'une ligne

grille:nombrecolonnes()
Nombre de colonnes

grille:colonne(col)
Objet colonne

grille:selection([i])
Ligne sélectionnée (i-ème)

grille:nombreselection()
Nombre de lignes sélectionnées

grille:selectionner(l)
Sélectionne une ligne

grille:deselectionner(l)
Désélectionne

grille:toutselectionner()
Sélectionne tout

grille:trouver(texte)
Cherche partout : {ligne, colonne}

grille:valeurpied(col, [genre])
Valeur d'un pied de colonne

grille:effacerformats()
Retire les mises en forme

grille:fixerrupture([col])
Regroupe sur une colonne

grille:fixermodecolonne(col, mode)
Saisie d'une colonne (lecture, modifiable)

grille:fixermodeligne(l, mode)
Saisie d'une ligne

grille:fixerremplissage(mode)
programme, manuel, automatique

grille:fixerlibellebouton(col, texte)
Libellé des boutons d'une colonne

grille:modifiercellule(l, col)
Place la saisie dans une cellule

grille:decsv(texte, [ajout])
Remplit depuis un texte CSV

grille:enregistrercsv(chemin, [sep])
Écrit un CSV

grille:enregistrerjson(chemin)
Écrit un JSON

Graphes, planning, PDF

graphe:ajouterdonnee(serie, categorie, valeur)
Ajoute une donnée

graphe:libellecategorie(i, texte)
Libellé de catégorie

graphe:libelleserie(i, texte)
Libellé de série

graphe:fixergenre(genre)
Genre de graphe

calendrier:afficher(annee, mois)
Affiche un mois

kanban:ajouterliste(titre)
Ajoute une liste ; renvoie son numéro

kanban:ajouter(liste, texte)
Ajoute une carte

mosaique:ajouter(libelle, valeur)
Ajoute une tuile proportionnelle

gantt:ajoutertache(nom, debut, duree, [avancement])
Ajoute une tâche ; renvoie son numéro

gantt:fixeravancement(i, valeur)
Avancement d'une tâche (%)

panneau:defilera(x, y)
Fait défiler

panneau:defilerde(dx, dy)
Fait défiler d'un décalage

panneau:rendrevisible(controle)
Amène un contrôle à l'écran

pdf:ouvrir(chemin)
Ouvre un PDF

pdf:enregistrer([chemin])
Enregistre

pdf:nouveau([largeur], [hauteur])
Nouveau document vide

pdf:fermer()
Ferme le document

pdf:nombrepages()
Nombre de pages

pdf:page()
Page courante

pdf:allera(n)
Va à la page n

pdf:zoom(z)
Zoom

pdf:ajoutertexte(page, x, y, texte, [taille])
Écrit sur une page

pdf:importerpages(chemin, [plage], [position])
Insère des pages d'un autre PDF

Propriétés

valeur texte num libelle titre visible etat actif infobulle nom x y largeur hauteur couleur fond bordure accent rayon coche min max ancrage poids largeurfixe hauteurfixe etirer marge espacement colonnes famille genre
ctl.propriete en lecture et en écriture

Événements

surclic surdoubleclic surclicdroit surclicmilieu surdoubleclicdroit surboutonbas surboutonhaut surentree sursortie surdeplacement surmolette surtouchebas surtouchehaut sursaisie surfocus surfinfocus surchangement surselection surdeploiement surrepli survalidation surouverture surfermeture surredimension suractivation surdesactivation surevenement
ctl.surXXX = @PROC ; PROC(ev) ; ev:arreter()

Pièges à éviter

⚠ Listes, combos, radios, onglets : valeur = numéro choisi (à partir de 1).

⚠ Les gestionnaires ne voient que les globales.

⚠ ui.tracer(vrai) affiche chaque événement reçu.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Aide-mémoire · Pages web

Pages HTML / CSS / JavaScript reliées à Lexis+.

Voir aussi : Interface graphique · Fichiers & formats · Tables & objets

31 instructions et fonctions

ENCHAÎNEMENTS

`webview.fenetre{fichier}` → `f:lier · f:autoriser` → `f.surpret = @PRET` → `f:executer()`

`f:appeler("fn", …, @R)` → `fn() en JS` → `R(valeur, erreur)` — `await nomjs(a)` → `PROC(a)` → `retourner v`

Modèle type

```
f = webview.fenetre{titre = "Démo", fichier = "page.htm"}
f:lier("total", @TOTAL)          && JS : await total(2, 3)
f.surpret = @PRET
f:executer()
procedure PRET(ev)
  f:appeler("afficher", "Bonjour", @VU)
retourner
```

webview

`webview.fenetre{titre=, largeur=, hauteur=, taille=, fichier=, url=, html=, debogage=, console=}`
Crée une fenêtre web

`webview.disponible()`
Vrai si la bibliothèque webview est chargée (sinon voir webview.erreur())

`webview.erreur()`
Raison de l'échec du chargement de la bibliothèque (nul si aucune)

`webview.version()`
Version de la bibliothèque webview

`webview.bibliotheque()`
Chemin de la bibliothèque chargée

`webview.url(chemin)`
URL file:// d'un fichier local (relatif au dossier du programme)

`webview.js(valeur)`
Littéral JavaScript (JSON) d'une valeur Lexis+

`fen:html(code)`
Affiche du HTML

`fen:naviguer(url)`
Charge une URL (https://, file://, data:...)

`fen:ouvrir(chemin)`
Charge un fichier HTML local (relatif au dossier du programme)

`fen:initialiser(js)`
Ajoute un script exécuté avant chaque page

`fen:evaluer(code, [@rappel])`
Lexis+ → JavaScript : évalue du code

`fen:appeler(fonction, arg..., [@rappel])`
Lexis+ → JavaScript : appelle window.fonction(arg...) (chemin "app.maj"...

`fen:emettre(nom, [donnees])`
Lexis+ → JavaScript : déclenche les écouteurs lexis.sur(nom, fn)

`fen:lier(nomjs, [@procedure])`
JavaScript → Lexis+ : crée window.nomjs(...) (Promise) qui appelle la...

`fen:delier(nomjs)`
Retire une liaison

`fen:autoriser(nom, ...)`
JavaScript → Lexis+ : procédures appelables par lexis.appeler(nom, ...)

`fen:taille(largeur, hauteur, [genre])`
Taille de la fenêtre (genre : normale, min, max, fixe)

`fen:executer()`
Affiche la fenêtre et traite ses événements jusqu'à sa fermeture

`fen:fermer()`
Ferme la fenêtre (termine fen:executer())

Côté JavaScript

`nomjs(…)`
fonction liée par f:lier : Promise

`lexis.appeler(nom, …)`
procédure autorisée : Promise

`lexis.envoyer(d)`
→ f.surmessage

`lexis.journal(…)`
→ console Lexis+

`lexis.sur(nom, fn) · lexis.retirer`
écouteurs de f:emettre

`lexis.pret`
page chargée

Propriétés

`f.titre`
titre (lecture / écriture)

`f.surpret`
page chargée : P(ev), ev.url

`f.surmessage`
lexis.envoyer(d) : P(d)

`f.surfermeture`
après la fermeture

`f.prete · f.ouverte`
états

Pièges à éviter

- ⚠ Appelez la page dans surpret (les appels antérieurs attendent).
- ⚠ La réponse JS arrive dans un rappel (valeur, erreur).
- ⚠ macOS : lancez avec wdlexisrt (fil principal).
- ⚠ Évitez autoriser("*") si la page affiche du contenu externe.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Aide-mémoire · Intelligence artificielle

Modèles de langage, GGUF, dessin, audio, tenseurs, entraînement.

Voir aussi : Multitâche · Fichiers & formats · Pages web

174 instructions et fonctions

ENCHAÎNEMENTS

`llm.charger(chemin)` → `m:conversation{…}` → `c:demander(q)` → `réponse`

`entrainement.tokeniseur` → `entrainement.nouveau` → `m:entrainer` → `m:enregistrer` → `llm.charger`

`tenseur.parametre` → `perte:retropropager()` → `opt:etape()`

Modèle type

```
m = llm.charger("modele.gguf")
c = m:conversation({systeme = "Sois
concis.", maxtokens = 200})
? c:demander("Résume ce texte.",
{pieces = {llm.joindre("doc.txt")}})
f = c:demanderjson("Trois villes ?",
[[{"villes": [str]}]])
? joindre(f.villes, ", ")
```

llm

`llm.charger(chemin, {fils=0, verbeux=faux})`
Charge un modèle GGUF (fichier « mappé » : quasi instantané)

`llm.version()`
Version de la bibliothèque nimllm

`llm.fils([n])`
Nombre de fils de calcul (0 = tous les cœurs)

`llm.echantillonnage([table])`
Réglages d'échantillonnage par défaut (complétés par la table) :...

`llm.glouton([table])`
Réglages du décodage glouton (déterministe)

`llm.echantillonneur([reglages])`
Échantillonneur manuel : s:tirer(logits), s:accepter(id)

`llm.joindre(chemin)`
Pièce jointe lue sur le disque (texte, CSV, JSON, PDF, PNG/BMP, WAV...)

`llm.joindretexte(nom, contenu)`
Pièce jointe construite en mémoire

`llm.joindredonnees(nom, octets)`
Pièce jointe à partir d'octets (le type est déduit du nom et du contenu)

`llm.extrairejson(texte)`
Premier objet JSON trouvé dans un texte (table) ou nul

`llm.sansbalises(texte)`
Retire les balises ``` d'un bloc de code

`llm.similarite(v1, v2)`
Similarité cosinus de deux vecteurs (m:vecteur)

`llm.prompt(gabarit, messages[, suite=vrai])`
Texte d'un dialogue {{role=, contenu=}...} mis en forme (llama3, chatml,...

`llm.gabarits()`
Liste des gabarits de conversation

`llm.tokeniseur(chemin)`
Tokeniseur seul lu dans un fichier GGUF

`llm.predecouper(texte[, mode])`
Pré-découpage avant BPE (llama3, qwen2, gpt2)

`m:conversation({systeme=, contexte=4096, maxtokens=512, echantillonnage=, gabarit="auto", langue="fr", arrets=})`
Nouvelle conversation (le texte seul est accepté comme prompt système)

`m:contexte([taille])`
Contexte brut (cache KV) : evaluer, completer, generer

`m:decrire()`
Résumé du modèle

`m:vecteur(texte)`
Vecteur (embedding) d'un texte

`m:perplexite(texte)`
Perplexité d'un texte (plus elle est basse, plus le texte est attendu)

`m:appliquerlora(chemin)`
Applique un adaptateur LoRA (.lora.gguf) à la volée

`m:retirerlora()`
Retire l'adaptateur LoRA

`m:fermer()`
Libère le modèle

`m.tokeniseur`
Tokeniseur du modèle (propriétés : nom, architecture, couches, dimension,...

`c:demander(question[, options | @flux])`
Réponse (fiche texte, brut, fichiers, json, tokensprompt, tokensgeneres,...

`c:demanderjson(question[, schema])`
Réponse JSON convertie en table (clefs en majuscules)

`c:regenerer([@flux])`
Nouveau tirage de la dernière réponse

`c:prolonger([maxtokens, @flux])`
Prolonge la dernière réponse

`c:generer([options | @flux])`
Réponse de l'assistant à l'historique actuel (après c:ajouter)

`c:ajouter(role, texte)`
Ajoute un message (systeme, utilisateur, assistant) : exemples « few-shot »,...

`c:annuler()`
Retire le dernier échange

`c:reinitialiser()`
Oublie l'historique (garde le prompt système)

`c:enregistrer(chemin)`
Enregistre la conversation (JSON)

`c:charger(chemin)`
Recharge une conversation

`c:etat()`
État du contexte (tokens, messages, vitesse)

`c:prompt([question])`
Texte exact envoyé au modèle

`c:tokensprompt()`
Tokens de l'historique mis en forme

`c.systeme`
Propriété : systeme, maxtokens, arrets, echantillonnage, tailleimage, voix,...

`x:evaluer(ids)`
Ajoute des tokens au contexte et renvoie les logits du dernier

`x:completer(texte[, options | @flux])`
Complétion brute (sans gabarit de conversation)

`x:generer(ids[, options | @flux])`
Génère une suite à partir de tokens

`x:reinitialiser()`
Vide le contexte

`x:tronquer(n)`
Ne garde que les n premiers tokens

`l:meilleurs([n])`
Les n tokens les plus probables : {{id=, proba=}...}

`l:valeur(id)`
Logit d'un token

`l:argmax()`
Token le plus probable

`s:tirer(logits)`
Tire un token selon les réglages

`s:accepter(id)`
Mémorise un token (pénalités de répétition)

`t:encoder(texte[, bos=faux, speciaux=vrai])`
Liste des identifiants de tokens

`t:decoder(ids[, speciaux=faux])`
Texte d'une liste de tokens

`t:morceau(id[, special=faux])`
Texte d'un token

`t:id(texte)`
Identifiant d'un token (nul s'il n'existe pas)

`t:estspecial(id)`
Vrai pour un token spécial

`t:enregistrer(chemin)`
Enregistre le tokeniseur dans un fichier GGUF

`t.taille`
Propriétés : taille, bos, eos, pad, inconnu, fins, fusions, type, gabarit,...

`p:texte([max])`
Texte inséré dans le message (propriétés : nom, type, mime, contenu, taille,...

gguf

`gguf.ouvrir(chemin)`
Ouvre un fichier GGUF (lecture seule, mappé en mémoire)

`gguf.quantifiermodele(entree, sortie[, type="q4_k", garder_sortie=vrai])`
Re-quantifie un modèle (f32, f16, bf16, q8_0, q6_k, q5_k, q5_0, q5_1, q4_k,...

`gguf.fusionnerlora(base, adaptateur, sortie[, type="q8_0"])`
Fusionne un adaptateur LoRA dans un nouveau modèle autonome

`gguf.types()`
Types de poids utilisables pour la quantification

`gguf.bitsparpoids(type)`
Nombre moyen de bits par poids d'un type

`gguf.quantifier(type, valeurs)`
Quantifie puis relit une liste : {octets=, valeurs=, erreur=}

`g:decrire([max_tenseurs])`
Résumé du fichier

`g:existe(cle)`
Vrai si la métadonnée existe

`g:valeur(cle)`
Valeur d'une métadonnée (nombre, texte, logique ou liste)

`g:texte(cle[, defaut])`
Métadonnée texte

`g:entier(cle[, defaut])`
Métadonnée entière

`g:reel(cle[, defaut])`
Métadonnée réelle

`g:cles()`
Liste des clefs de métadonnées

`g:tenseurs()`
Liste des tenseurs : {nom=, type=, dims=, elements=, octets=}

`g:tenseur(nom[, max])`
Valeurs d'un tenseur (déquantifiées)

`g:fermer()`
Ferme le fichier (propriétés : version, chemin, alignement, architecture)

dessin

`dessin.svg(code[, largeur, hauteur, fond])`
Rastérise un dessin SVG

`dessin.image(largeur, hauteur[, fond])`
Image unie

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

dessin.toile(largeur, hauteur[, fond, lissage=3])
Surface de dessin anticrénelée

dessin.lire(chemin)
Lit une image BMP, PPM ou PNG

dessin.extrairesvg(texte)
Code SVG contenu dans un texte (réponse d'un modèle) ou nul

dessin.taillesvg(code)
Dimensions déclarées d'un SVG : {largeur=, hauteur=}

dessin.couleur(texte | r, g, b[, a])
Couleur sous forme de table {r=, g=, b=, a=}

t:rectangle(x, y, largeur, hauteur, couleur)
Rectangle plein

t:cercle(cx, cy, rayon, couleur)
Disque

t:ellipse(cx, cy, rx, ry, couleur)
Ellipse pleine

t:polygone(points, couleur)
Polygone plein

t:ligne(x0, y0, x1, y1, epaisseur, couleur)
Segment

t:polyligne(points, epaisseur, couleur[, fermee])
Ligne brisée

t:terminer()
Termine le dessin et renvoie l'image

img:enregistrer(chemin)
Enregistre selon l'extension (.bmp ou .ppm)

img:enregistrerbmp(chemin)
Enregistre en BMP

img:enregistrerppm(chemin)
Enregistre en PPM

img:pixel(x, y)
Couleur d'un pixel : {r=, g=, b=}

img:fixerpixel(x, y, couleur)
Colore un pixel

img:decrire([largeur])
Description textuelle (couleurs, aperçu en caractères)

audio

audio.parler(texte[, {langue="fr", hauteur=115, vitesse=1, frequence=16000}])
Synthèse vocale (fr ou en)

audio.melodie(notation[, bpm=120])
Mélodie : « C4 E4 G4:2 R » (note+octave[:temps], R = silence)

audio.nouveau([frequence])
Son vide

audio.lire(chemin)
Lit un fichier WAV PCM 16 bits

audio.infos(chemin)
En-tête d'un WAV : {frequence=, canaux=, bits=, duree=}

audio.phonemes(texte[, langue])
Transcription phonétique utilisée par la synthèse

audio.enlettres(n[, langue])
Nombre écrit en toutes lettres

audio.frequencenote(note)
Fréquence d'une note (« A4 » = 440)

s:enregistrer(chemin)
Écrit un fichier WAV (16 bits mono)

s:silence(secondes)
Ajoute un silence

s:ajouter(autre)
Ajoute un autre son à la suite (rééchantillonné si besoin)

s:normaliser([crete=0.9])
Ajuste le volume

s:copier()
Copie du son (propriétés : duree, frequence, echantillons)

tenseur

tenseur.depuis(valeurs[, forme, gradient=faux, nom])
Tenseur à partir d'une liste

tenseur.plein(forme[, valeur=0, gradient=faux, nom])
Tenseur rempli d'une valeur

tenseur.zeros(forme[, gradient=faux, nom])
Tenseur nul

tenseur.scalaire(v[, gradient=faux])
Tenseur d'une seule valeur

tenseur.aleatoire(forme[, ecart=1, gradient=vrai, nom])
Valeurs gaussiennes N(0, écart²)

tenseur.parametre(forme[, ecart=0.1, nom])
Paramètre entraînable initialisé au hasard

tenseur.graine(n)
Fixe la graine du générateur aléatoire du paquet

tenseur.hasard([max | "gauss"])
Nombre aléatoire du générateur du paquet

tenseur.lineaire(x, w[, b])
x · wᵀ + b (x : {n, entrées}, w : {sorties, entrées})

tenseur.produit(a, b)
Produit matriciel {n, k} · {k, m}

tenseur.echelle(t, s)
Multiplie par un nombre (aussi : t * s)

tenseur.somme(t)
Somme des éléments

tenseur.moyenne(t)
Moyenne des éléments

tenseur.relu(t)
max(0, x)

tenseur.silu(t)
x · sigmoïde(x)

tenseur.sigmoide(t)
1 / (1 + e^-x)

tenseur.tanh(t)
Tangente hyperbolique

tenseur.gelu(t)
GELU

tenseur.softmax(t)
Softmax par ligne

tenseur.mse(prediction, cible)
Erreur quadratique moyenne

tenseur.entropiecroisee(logits, classes)
Entropie croisée (classes : liste d'entiers à partir de 0

tenseur.concatener(a, b)
Concaténation des colonnes

tenseur.decrochage(t, p)
Dropout (entraînement)

tenseur.sansgradient(@proc, ...)
Appelle proc sans construire de graphe (inférence)

tenseur.verifiergradient(@proc, t[, eps, echantillons])
Écart relatif entre gradient calculé et différence finie

tenseur.operation(parents, valeurs, forme, @retro)
Opération personnalisée

tenseur.gradientrequis(t, ...)
Vrai si l'un des tenseurs demande un gradient

tenseur.adamw(parametres[, {lr=3e-4, beta1=0.9, beta2=0.95, decroissance=0.1}])
Optimiseur AdamW

tenseur.sgd(parametres[, {lr=0.01, moment=0.9, decroissance=0}])
Descente de gradient avec moment

tenseur.lrcosinus(etape, echauffement, total, lrmax[, lrmin])
Taux d'apprentissage (échauffement + cosinus)

tenseur.normegradient(parametres)
Norme globale des gradients

t:retropropager()
Calcule les gradients de tout le graphe (t scalaire)

t:valeur([i=1])
Valeur d'un élément

t:donnees()
Liste des valeurs

t:gradient()
Liste du gradient (nul s'il n'a pas été calculé)

t:forme()
Forme (liste des dimensions)

t:taille()
Nombre d'éléments

t:zerograd()
Remet le gradient à zéro

t:detacher()
Copie sans gradient

t:remodeler(forme)
Même données, autre forme

t:ajoutergradient(valeurs)
Ajoute au gradient (opérations personnalisées)

o:zerograd()
Remet à zéro les gradients des paramètres

o:etape()
Met à jour les paramètres

o:ecreter(norme_max)
Limite la norme des gradients

o:enregistrer(chemin)
Enregistre l'état (moments) de l'optimiseur

o:charger(chemin)
Recharge l'état

entrainement

entrainement.tokeniseur(textes[, {taille=2048, speciaux=, frequencemin=2, mode="llama3"}])
Apprend un tokeniseur BPE niveau octet (compatible llama.cpp)

entrainement.tokeniseuroctets([speciaux])
Tokeniseur minimal (256 octets + spéciaux)

entrainement.nouveau(tok, {dimension=256, couches=4, tetes=4, tetes_kv=, cache=, contexte=256, nom=, graine=42})
Nouveau modèle Transformer (architecture Llama)

entrainement.charger(chemin[, {mode="lora", rang=8, alpha=16, cibles={"q","k","v","o"}}])
Charge un GGUF à spécialiser : LoRA (poids gelés) ou complet

entrainement.textes(tok, texte[, {graine=, bos=vrai}])
Données de pré-entraînement (texte continu)

entrainement.fichiers(tok, chemins)
Données de pré-entraînement lues dans des fichiers

entrainement.dialogues(tok, gabarit, dialogues[, {systeme=}])
Données de dialogue (seules les réponses sont apprises)

entrainement.jsonl(chemin, tok[, gabarit="auto", systeme])
Dialogues lus dans un fichier JSONL

entrainement.gabarit(tok)
Gabarit de conversation adapté au tokeniseur

entrainement.configuration()
Réglages d'entraînement par défaut

m:entrainer(donnees[, {etapes, lot, sequence, accumulation, lr, lrmin, echauffement, decroissance, ecretage, evaluation, lotsevaluation, journal, sauvegarde, chemin, echantillon, amorce, graine, validation, optimiseur, rappel}])
Entraîne le modèle

m:parametres()
Liste des tenseurs entraînables

m:nombreparametres([tous=faux])
Nombre de paramètres

m:optimiseur([{lr=, decroissance=}])
AdamW sur les paramètres du modèle

m:enregistrer(chemin[, type="f32"])
Exporte un vrai fichier GGUF (llm.charger, llama.cpp, Ollama)

m:enregistrerlora(chemin)
Enregistre l'adaptateur LoRA

m:sauvegarder(optimiseur, prefixe)
Point de sauvegarde (modèle ou adaptateur + optimiseur)

m:evaluer(donnees[, lots=4, lot=8, sequence=64])
Perte moyenne (exp(perte) = perplexité)

m:generer(amorce[, {maxtokens=50, temperature=0.8}])
Génère un texte (surveillance de l'entraînement)

d:separer([fraction=0.1])
Sépare en {entrainement, validation}

d:taille()
Nombre de tokens (texte) ou de dialogues

d:pluslong()
Longueur du plus long dialogue (tokens)

Pièges à éviter

⚠ Compilez avec -d:release : 10 à 30 fois plus rapide.

⚠ c:reinitialiser() pour repartir d'une conversation vide.

⚠ Le contexte (en tokens) limite la mémoire de la conversation.

⚠ Les blocs de quantification font 32 valeurs.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

ANNEXE C

Glossaire

Les mots du programmeur, expliqués simplement.



© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Annexe C · Glossaire

Les mots du programmeur, expliqués simplement.

Affectation
Ranger une valeur dans une variable : x = 5.

Argument
Valeur passée à une procédure lors de son appel : dans TTC(100), 100 est l'argument.

Asynchrone
Se dit d'un résultat qui arrivera plus tard ; on le reçoit dans un rappel (webview, JavaScript).

Atelier
wdlexis+, le logiciel dans lequel on écrit, exécute et débogue les programmes.

Booléen / logique
Valeur vrai ou faux.

Boucle
Bloc d'instructions répété : pour, pour chaque, tantque.

Canal
File d'attente qui permet à des tâches d'échanger des valeurs.

Chaîne
Valeur de type texte, écrite entre guillemets.

Clef
Nom sous lequel une valeur est rangée dans une fiche (table).

Commentaire
Texte ignoré par l'interpréteur, destiné aux lecteurs (&&, //, *).

Concaténation
Mise bout à bout de textes : "a" + "b" donne "ab".

Condition
Expression vraie ou fausse qui décide du chemin suivi (si, tantque).

Console
Zone où le programme affiche ses résultats et où l'on tape les réponses.

CSV
Format de tableau en texte : une ligne par ligne, des colonnes séparées par , ou ;.

Débogueur
Outil qui exécute un programme pas à pas en montrant les variables.

DuckDB
Base de données intégrée à Lexis+, interrogée en SQL ou par zones.

Échantillonnage
Façon dont un modèle de langage choisit chaque token (température...).

Erreur
Problème qui interrompt l'exécution ; on l'intercepte avec essayer / capturer.

Événement
Action de l'utilisateur (clic, frappe...) à laquelle un programme à fenêtres réagit.

Expression
Combinaison de valeurs et d'opérateurs qui donne un résultat : 2 * prix + 1.

Fiche
Table dont les valeurs sont rangées sous des clefs : {nom = "Ada"}.

Fil d'exécution
Programme qui s'exécute en parallèle, avec son propre interpréteur (paquet fil).

Fonction / procédure
Morceau de programme nommé, réutilisable ; il peut renvoyer un résultat.

GGUF
Format de fichier des modèles de langage (llama.cpp, Ollama, Lexis+).

Gestionnaire
Procédure appelée quand un événement se produit (surclic = @CLIC).

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Globale (variable)
Variable partagée par tout le programme, rangée dans SESSION.

Gradient
Direction dans laquelle modifier un paramètre pour réduire l'erreur (apprentissage).

HTML / CSS / JavaScript
Les trois langages d'une page web : contenu, présentation, comportement.

Indice
Numéro d'un élément dans une liste, à partir de 1.

Instruction
Ordre élémentaire donné à l'ordinateur ; une par ligne en Lexis+.

Interpréteur
Logiciel qui lit et exécute un programme Lexis+.

Itérateur
Procédure qui fournit les éléments d'une boucle pour chaque, puis nul.

JSON
Format d'échange de données du web : objets {...} et tableaux [...].

Liste
Table dont les valeurs sont numérotées : {"a", "b"}.

Locale (variable)
Variable qui n'existe que dans la procédure où elle est déclarée.

Métatable
Table qui donne des méthodes et des opérateurs à une autre table (objets).

Méthode
Procédure attachée à un objet, appelée avec : (obj:methode()).

Modèle de langage
Réseau de neurones qui prédit la suite d'un texte et sait dialoguer.

nul
Valeur spéciale qui signifie « pas de valeur ».

Objet
Données et actions regroupées : une fenêtre, un fichier ouvert, une connexion...

Opérateur
Symbole de calcul ou de comparaison : + - * / = < et ou...

Paquet
Ensemble de fonctions regroupées sous un nom : fichier, http, llm...

Paramètre
Variable d'une procédure qui reçoit un argument.

Port
Numéro qui désigne un service réseau sur une machine (80, 443...).

Programme principal
Les instructions écrites en tête de principal.prg.

Quantification
Réduction de la précision des poids d'un modèle pour l'alléger.

Rappel
Procédure passée en argument et appelée plus tard avec un résultat.

Récursivité
Procédure qui s'appelle elle-même.

Référence (@NOM)
Désignation d'une procédure sans l'appeler.

Requête
Question posée à une base de données, en SQL.

Runtime
wdlexisrt, qui exécute un programme sans l'atelier.

SQL
Langage des bases de données : select, insert, update, delete.

Table
Conteneur de valeurs, liste et / ou fiche, écrit entre accolades.

Tâche
Procédure qui s'exécute à tour de rôle avec d'autres (paquet tache).

Token
Morceau de mot manipulé par un modèle de langage.

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Transaction
Groupe de modifications d'une base validées ensemble ou annulées ensemble.

Type
Nature d'une valeur : nul, logique, nombre, chaine, table, procedure, objet.

Variable
Boîte nommée qui contient une valeur.

Zone
Table ouverte pour un travail enregistrement par enregistrement (utiliser).

© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

ANNEXE D

Le poster

La carte heuristique de toutes les instructions de Lexis+, par grappes de domaines (format A0, à agrandir à l'écran ou à imprimer).



© 2026 Jean-Marc QUÉRÉ, sonaliwan.fr · SIRET 13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr

Lexis+ en une carte

Toutes les instructions, fonctions, méthodes et propriétés du langage, rangées par grappes de domaines. Le noyau est le langage ; chaque branche mène à un domaine.

862
mots, fonctions et méthodes

Démarrer
Écrire, exécuter, déboguer et lancer un programme. **22**

- **AFFICHER ET SAISIR**: `?` `??` `saisir()` `paquet()`
- **PROJET**: `principal.prg` `fichiers .prg` `PRINCIPAL` `finprocedure`
- **ATELIER WDLEXIS+**: `Ctrl+R` `F5` `F9` `F10` `F11` `Maj+F11` `Maj+F5` `Ctrl+Espace` `Ctrl+B` `F12`
- **RUNTIME WDLEXISRT**: `wdlexisrt` `--verifier` `-e` `--paquets`

Les bases
Instructions, commentaires, valeurs, opérateurs et variables. **55**

- **LIGNES ET COMMENTAIRES**: `&&` `//` `*` `;` `casse libre`
- **VALEURS**: `nul` `vrai` `faux` `.v.` `.f.` `nombre` `"chaine"` `[[…]]` `{table}` `@NOM`
- **OPÉRATEURS**: `+` `-` `*` `/` `%` `^` `**` `..` `=` `==` `<>` `!=` `#` `<` `<=` `>` `>=` `$` `et` `ou` `non` `.et.` `.ou.` `.non.` `#t`
- **VARIABLES**: `locale` `local` `globale` `=` `+=` `-=` `stocker … dans` `liberer` `SESSION` `m~>`
- **TYPES ET CONVERSIONS**: `type()` `chaine()` `texte()` `valeur()` `nombre()`

Décider & répéter
Choisir un chemin, répéter un traitement. **23**

- **CONDITIONS**: `si` `sinonsi` `sinon si` `sinon` `finsi` `faire cas` `cas` `autrement` `fincas` `sisinon()` `vide()` `estnul()`
- **BOUCLES**: `pour … a` `jusqua` `pas` `suivant` `finpour` `pour chaque … dans` `tantque` `faire tantque` `fintantque` `sortir` `boucler`

Procédures
Nommer un traitement pour le réutiliser partout. **13**

- **DÉFINIR**: `procedure` `fonction` `parametres` `retourner` `retour` `finprocedure` `finfonction`
- **APPELER**: `NOM()` `faire … avec` `@NOM` `ref()` `récursivité` `appeler()`

Tables & objets
Listes, fiches, objets et métatables. **47**

- **CRÉER ET ACCÉDER**: `{ }` `t[i]` `t.cle` `t["CLE"]` `nouvelle table` `supprimer table` `#t`
- **MODIFIER**: `inserer … dans` `position` `clef` `retirer de … position … dans` `inserer()` `retirer()` `trier()` `copier()`
- **INTERROGER**: `taille()` `nbelements()` `clefs()` `valeurs()` `aclef()` `contient()` `joindre()` `decouper()` `json()` `dejson()`
- **OBJETS ET MÉTATABLES**: `:procedure … comme` `obj:methode()` `__index` `__newindex` `__call` `__tostring` `__len` `__eq` `__lt` `__le` `__add` `__sub` `__mul` `__div` `__mod` `__pow` `__unm` `__concat` `fixermetatable()` `obtenirmetatable()` `librebrut()` `ecrirebrut()`

Texte & calcul
La boîte à outils toujours disponible : texte, nombres, dates, sécurité. **55**

- **TEXTE**: `majuscule()` `minuscule()` `rogner()` `rognerg()` `rognerd()` `longueur()` `souschaine()` `gauche()` `droite()` `position()` `espace()` `repliquer()` `remplacertexte()` `contient()` `commencepar()` `finitpar()` `decouper()` `joindre()` `chaine()` `html()` `url()` `car()` `code()`
- **NOMBRES**: `abs()` `entier()` `arrondi()` `plancher()` `plafond()` `racine()` `mod()` `max()` `min()` `aleatoire()` `sin()` `cos()` `tan()` `asin()` `acos()` `atan()` `exp()` `log()` `pi()`
- **DATES**: `date()` `heure()` `maintenant()` `horodatage()`
- **TESTS**: `type()` `estnul()` `vide()` `sisinon()`
- **SÉCURITÉ**: `hachage()` `hachermdp()` `verifiermdp()` `mdpaleatoire()` `uuid()`

Bases de données
DuckDB intégré (SQL et zones dBase), SQLite, Firebird. **70**

- **BASE DUCKDB**: `ouvrirbase()` `fermerbase()` `base()` `requete()` `valeursql()` `sql "…" [avec v, …] [dans liste]` `begin transaction · commit · rollback`
- **ZONES (DBASE)**: `utiliser` `alias` `ordre` `filtre` `selectionner` `fermer` `tout` `aller haut` `aller bas` `aller n` `sauter` `chercher` `localiser pour` `continuer` `ajouter` `vide` `avec … par` `remplacer … par` `effacer` `parcourir` `finparcourir` `compter … dans` `rafraichir` `alias->champ` `m->` `fdf()` `ddf()` `numenr()` `nbenr()` `trouve()` `alias()` `champ()` `enregistrement()`
- **SQLITE**: `sqlite.ouvrir()` `sqlite.version()` `db:executer()` `db:requete()` `db:premier()` `db:lignes()` `db:dernierid()` `db:modifications()` `db:transaction()` `db:fermer()`
- **FIREBIRD**: `firebird.connecter()` `firebird.creer()` `firebird.version()` `firebird.disponible()` `db:executer()` `db:requete()` `db:premier()` `db:valeur()` `db:lignes()` `db:modifications()` `db:sequence()` `db:transaction()` `db:debut()` `db:valider()` `db:annuler()` `db:entransaction()` `db:tables()` `db:colonnes()` `db:serveur()` `db:fermer()`

Système
Dates, environnement, commandes, arguments. **37**

- **SYSTEME**: `systeme.temps()` `systeme.horloge()` `systeme.date()` `systeme.ecart()` `systeme.env()` `systeme.fixerenv()` `systeme.executer()` `systeme.capturer()` `systeme.supprimer()` `systeme.renommer()` `systeme.nomtemporaire()` `systeme.quitter()` `systeme.attendre()` `systeme.plateforme()` `systeme.dossierprogramme()` `systeme.args`
- **FORMATS DE SYSTEME.DATE**: `%Y` `%m` `%d` `%H` `%M` `%S` `%A` `%a` `%B` `%b` `%F` `%T` `%x` `%X` `%c` `%j` `%w` `%y` `%Z` `"*t"` `"!"`

Erreurs
Signaler, intercepter et traiter les erreurs. **6**

- **INTERCEPTER**: `essayer` `capturer` `finessayer`
- **SIGNALER**: `erreur` `erreur {code=…}` `relancer`

Fichiers & formats
Lire et écrire fichiers, dossiers, CSV, JSON et binaire. **54**

- **FICHIER**: `fichier.lire()` `fichier.lireoctets()` `fichier.ecrire()` `fichier.ecrireoctets()` `fichier.ajouter()` `fichier.lignes()` `fichier.ecrirelignes()` `fichier.existe()` `fichier.estfichier()` `fichier.estdossier()` `fichier.taille()` `fichier.liste()` `fichier.creerdossier()` `fichier.supprimer()` `fichier.copier()` `fichier.deplacer()` `fichier.infos()` `fichier.courant()` `fichier.changer()` `fichier.joindre()` `fichier.nom()` `fichier.dossier()` `fichier.extension()` `fichier.absolu()` `fichier.dossiertemp()` `fichier.dossierperso()`
- **ES**: `es.ecrire()` `es.lire()` `es.lignes()` `es.ouvrir()` `es.type()` `es.sortie` `es.erreur` `es.entree` `f:lire()` `f:ecrire()` `f:lignes()` `f:position()` `f:vider()` `f:fermer()`
- **CSV**: `csv.analyser()` `csv.lire()` `csv.encoder()` `csv.ecrire()`
- **JSON**: `json.encoder()` `json.decoder()` `json.lire()` `json.ecrire()` `json.nul`
- **BINAIRE**: `binaire.coder()` `binaire.decoder()` `binaire.taille()` `binaire.hex()` `binaire.dehex()`

Réseau
HTTP, TCP, UDP, FTP, SSH et SFTP. **41**

- **HTTP**: `http.lire()` `http.envoyer()` `http.requete{}` `http.telecharger()` `http.encoder()` `http.decoder()`
- **RESEAU**: `reseau.connecter()` `reseau.ecouter()` `reseau.udp()` `reseau.resoudre()` `reseau.nomhote()` `s:envoyer()` `s:recevoir()` `s:delai()` `s:accepter()` `s:envoyerversl()` `s:recevoirde()` `s:distant()` `s:fermer()`
- **FTP**: `ftp.connecter{}` `c:liste()` `c:lire()` `c:envoyer()` `c:dossier()` `c:courant()` `c:creerdossier()` `c:supprimerdossier()` `c:supprimer()` `c:renommer()` `c:fermer()`
- **SSH**: `ssh.connecter{}` `ssh.executer{}` `session:executer()` `session:lire()` `session:envoyer()` `session:liste()` `session:creerdossier()` `session:supprimerdossier()` `session:supprimer()` `session:renommer()`
- **SFTP**: `sftp.connecter{}`

Multitâche
Tâches coopératives et fils d'exécution. **22**

- **TACHE**: `tache.lancer()` `tache.executer()` `tache.attendre()` `tache.ceder()` `tache.canal()` `tache.nombre()` `tache.maintenant()` `t:attendre()` `t:etat()` `t:annuler()` `c:envoyer()` `c:recevoir()` `c:nombre()`
- **FIL**: `fil.demarrer()` `fil.envoyer()` `fil.recevoir()` `fil.estenfant()` `f:envoyer()` `f:recevoir()` `f:actif()` `f:joindre()` `f:arreter()`

Pages web
Pages HTML / CSS / JavaScript reliées à Lexis+. **33**

- **WEBVIEW**: `webview.fenetre{}` `webview.disponible()` `webview.erreur()` `webview.version()` `webview.bibliotheque()` `webview.url()` `webview.js()` `fen:html()` `fen:naviguer()` `fen:ouvrir()` `fen:initialiser()` `fen:evaluer()` `fen:appeler()` `fen:emettre()` `fen:lier()` `fen:delier()` `fen:autoriser()` `fen:taille()` `fen:executer()` `fen:fermer()`
- **CÔTÉ JAVASCRIPT**: `nomjs()` `lexis.appeler` `lexis.envoyer` `lexis.journal` `lexis.sur` `lexis.retirer` `lexis.pret`
- **PROPRIÉTÉS**: `f.titre` `f.surpret` `f.surmessage` `f.surfermeture` `f.prete` `f.ouverte`

Interface graphique
Fenêtres wdgui : contrôles, événements, dialogues. **210**

- **FENÊTRE ET BOUCLE**: `ui.fenetre{}` `ui.fenetres()` `ui.executer()` `ui.quitter()` `ui.tracer()` `ui.separateur{}` `ui.sep` `fen:fermer()` `fen:fixertheme()` `c:ajouter()` `c:focus()` `c:menucontextuel()` `c:redessiner()`
- **CONTRÔLES**: `ui.etiquette()` `ui.bouton()` `ui.saisie()` `ui.cases()` `ui.radio()` `ui.liste()` `ui.combo()` `ui.tableau()` `ui.arbre()` `ui.grille{}` `ui.jauge()` `ui.curseur()` `ui.plage()` `ui.ascenseur()` `ui.compteur()` `ui.notation()` `ui.image()` `ui.codebarre()` `ui.graphe()` `ui.calendrier()` `ui.kanban()` `ui.mosaique()` `ui.gantt()` `ui.barremenu()` `ui.onglets{}` `ui.separateur{}` `ui.conteneur()` `ui.colonne()` `ui.rangee()` `ui.superconteneur()` `ui.cadre()` `ui.panneau{}` `ui.pdf()`
- **DIALOGUES**: `ui.alerte()` `ui.confirmer()` `ui.demander()` `ui.ouvrirfichier{}` `ui.enregistrerfichier{}` `ui.couleur()` `ui.police()`
- **LISTES, TABLEAUX, ARBRES**: `liste:ajouter()` `liste:inserer()` `liste:supprimer()` `liste:vider()` `liste:selectionner()` `liste:selection()` `liste:nombre()` `liste:element()` `liste:modifier()` `liste:chercher()` `liste:deselectionner()` `tableau:ajoutercolonne()` `tableau:ajouterligne()` `tableau:cellule()` `tableau:fixercellule()` `tableau:selection()` `tableau:trier()` `tableau:insererligne()` `tableau:ligne()` `tableau:nombre()` `tableau:selectionner()` `tableau:supprimer()` `tableau:vider()` `arbre:ajouter()` `arbre:deplier()` `arbre:replier()` `arbre:supprimer()` `arbre:vider()` `arbre:estdeplie()` `arbre:selectionner()` `arbre:nombre()` `onglets:ajouterpage()` `barremenu:ajouter()` `cases:coche()` `cases:cocher()` `cases:ajouteroption()` `radio:ajouteroption()`
- **GRILLE**: `grille:ajoutercolonne{}` `grille:ajouterligne()` `grille:cellule()` `grille:fixercellule()` `grille:nombre()` `grille:trier()` `grille:chercher()` `grille:fixerpied()` `grille:ajouterformat()` `grille:fixermodifiable()` `grille:verscsv()` `grille:chargercsv()` `grille:versjson()` `grille:chargerjson()` `grille:insererligne()` `grille:modifierligne()` `grille:supprimerligne()` `grille:vider()` `grille:ligne()` `grille:nombrecolonnes()` `grille:colonne()` `grille:selection()` `grille:nombreselection()` `grille:selectionner()` `grille:deselectionner()` `grille:toutselectionner()` `grille:trouver()` `grille:valeurpied()` `grille:effacerformats()` `grille:fixerrupture()` `grille:fixermodecolonne()` `grille:fixermodeligne()` `grille:fixerremplissage()` `grille:fixerlibellebouton()` `grille:modifiercellule()` `grille:decsv()` `grille:enregistrercsv()` `grille:enregistrerjson()`
- **GRAPHES, PLANNING, PDF**: `graphe:ajouterdonnee()` `graphe:libellecategorie()` `graphe:libelleserie()` `graphe:fixergenre()` `calendrier:afficher()` `kanban:ajouterliste()` `kanban:ajouter()` `mosaique:ajouter()` `gantt:ajoutertache()` `gantt:fixeravancement()` `panneau:deflera()` `panneau:deflerde()` `panneau:rendrevisible()` `pdf:ouvrir()` `pdf:enregistrer()` `pdf:nouveau()` `pdf:fermer()` `pdf:nombrepages()` `pdf:page()` `pdf:allera()` `pdf:zoom()` `pdf:ajoutertexte()` `pdf:importerpages()`
- **PROPRIÉTÉS**: `valeur` `texte` `num` `libelle` `titre` `visible` `etat` `actif` `infobulle` `nom` `x` `y` `largeur` `hauteur` `couleur` `fond` `bordure` `accent` `rayon` `coche` `min` `max` `ancrage` `poids` `largeurfixe` `hauteurfixe` `etirer` `marge` `espacement` `colonnes` `famille` `genre`
- **ÉVÉNEMENTS**: `surclic` `surdoubleclic` `surclicdroit` `surclicmilieu` `surdoubleclicdroit` `surboutonbas` `surboutonhaut` `surentree` `sursortie` `surdeplacement` `surmolette` `surtouchebas` `surtouchehaut` `sursaisie` `surfocus` `surfinfocus` `surchangement` `surselection` `surdeploiement` `surrepli` `survalidation` `surouverture` `surfermeture` `surredimension` `suractivation` `surdesactivation` `surevenement`

Intelligence artificielle
Modèles de langage, GGUF, dessin, audio, tenseurs, entraînement. **174**

- **LLM**: `llm.charger()` `llm.version()` `llm.fils()` `llm.echantillonnage()` `llm.glouton()` `llm.echantillonneur()` `llm.joindre()` `llm.joindretexte()` `llm.joindredonnees()` `llm.extrairejson()` `llm.sansbalises()` `llm.similarite()` `llm.prompt()` `llm.gabarits()` `llm.tokeniseur()` `llm.predecouper()` `m:conversation()` `m:contexte()` `m:decrire()` `m:vecteur()` `m:perplexite()` `m:appliquerlora()` `m:retirerlora()` `m:fermer()` `m.tokeniseur` `c:demander()` `c:demanderjson()` `c:regenerer()` `c:prolonger()` `c:generer()` `c:ajouter()` `c:annuler()` `c:reinitialiser()` `c:enregistrer()` `c:charger()` `c:etat()` `c:prompt()` `c:tokensprompt()` `c.systeme` `x:evaluer()` `x:completer()` `x:generer()` `x:reinitialiser()` `x:tronquer()` `l:meilleurs()` `l:valeur()` `l:argmax()` `s:tirer()` `s:accepter()` `t:encoder()` `t:decoder()` `t:morceau()` `t:id()` `t:estspecial()` `t:enregistrer()` `t.taille` `p:texte()`
- **GGUF**: `gguf.ouvrir()` `gguf.quantifiermodele()` `gguf.fusionnerlora()` `gguf.types()` `gguf.bitsparpoids()` `gguf.quantifier()` `g:decrire()` `g:existe()` `g:valeur()` `g:texte()` `g:entier()` `g:reel()` `g:cles()` `g:tenseurs()` `g:tenseur()` `g:fermer()`
- **DESSIN**: `dessin.svg()` `dessin.image()` `dessin.toile()` `dessin.lire()` `dessin.extrairesvg()` `dessin.taillesvg()` `dessin.couleur()` `t:rectangle()` `t:cercle()` `t:ellipse()` `t:polygone()` `t:ligne()` `t:polyligne()` `t:terminer()` `img:enregistrer()` `img:enregistrerbmp()` `img:enregistrerppm()` `img:pixel()` `img:fixerpixel()` `img:decrire()`
- **AUDIO**: `audio.parler()` `audio.melodie()` `audio.nouveau()` `audio.lire()` `audio.infos()` `audio.phonemes()` `audio.enlettres()` `audio.frequencenote()` `s:enregistrer()` `s:silence()` `s:ajouter()` `s:normaliser()` `s:copier()`
- **TENSEUR**: `tenseur.depuis()` `tenseur.plein()` `tenseur.zeros()` `tenseur.scalaire()` `tenseur.aleatoire()` `tenseur.parametre()` `tenseur.graine()` `tenseur.hasard()` `tenseur.lineaire()` `tenseur.produit()` `tenseur.echelle()` `tenseur.somme()` `tenseur.moyenne()` `tenseur.relu()` `tenseur.silu()` `tenseur.sigmoide()` `tenseur.tanh()` `tenseur.gelu()` `tenseur.softmax()` `tenseur.mse()` `tenseur.entropiecroisee()` `tenseur.concatener()` `tenseur.decrochage()` `tenseur.sansgradient()` `tenseur.verifiergradient()` `tenseur.operation()` `tenseur.gradientrequis()` `tenseur.adamw()` `tenseur.sgd()` `tenseur.lrcosinus()` `tenseur.normegradient()` `t:retropropager()` `t:valeur()` `t:donnees()` `t:gradient()` `t:forme()` `t:taille()` `t:zerograd()` `t:detacher()` `t:remodeler()` `t:ajoutergradient()` `o:zerograd()` `o:etape()` `o:ecrireter()` `o:enregistrer()` `o:charger()`
- **ENTRAINEMENT**: `entrainement.tokeniseur()` `entrainement.tokeniseuroctets()` `entrainement.nouveau()` `entrainement.charger()` `entrainement.textes()` `entrainement.fichiers()` `entrainement.dialogues()` `entrainement.jsonl()` `entrainement.gabarit()` `entrainement.configuration()` `m:entrainer()` `m:parametres()` `m:nombreparametres()` `m:optimiseur()` `m:enregistrer()` `m:enregistrerlora()` `m:sauvegarder()` `m:evaluer()` `m:generer()` `d:separer()` `d:taille()` `d:pluslong()`



© 2026 Jean-Marc QUERE, sonaliwan.fr · SIRET13033319800013 · Lexis+ et wdlexis+ sont distribués sous licence CC BY-NC-SA 4.0 · usage commercial : autorisation de l'auteur · metalab (at) sonaliwan.fr