Jean Paul Roy

# Python

## Apprentissage actif

pour l'étudiant et le futur enseignant

2e édition



ellipses

Références sciences

# Python
# Apprentissage actif

## pour l'étudiant et le futur enseignant

2e édition

Jean Paul Roy

ellipses

Collection Références sciences

dirigée par Paul de Laboulaye
paul.delaboulaye@editions-ellipses.fr

*Retrouvez tous les livres de la collection et des extraits sur www.editions-ellipses.fr*

ISBN 9782340-111332
Dépôt légal : janvier 2026
©Ellipses Édition Marketing S.A.
8/10 rue la Quintinie 75015 Paris

DANGER LE PHOTOCOPILLAGE TUE LE LIVRE

Le Code de la propriété intellectuelle et artistique n'autorisant, aux termes des alinéas 2 et 3 de l'article L. 122-5, d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite » (alinéa 1er de l'article L. 122-4).
Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles L. 335-2 et suivants du Code de la propriété intellectuelle.

www.editions-ellipses.fr

*À ma mère.*



*Saint-Alban*

Collection privée

# Introduction

## Le langage Python

Les programmeurs ont coutume de dire que les langages de programmation sont *universels*. Chaque langage contient les constructions linguistiques suffisantes pour exprimer n'importe quel calcul, et si les professionnels acceptent de passer par les fourches caudines de langages industriels complexes, efficaces et sécurisés, le choix d'un *petit langage* suffisamment performant à enseigner aux grands débutants s'est posé dès le début. En 1964 naquit le langage **BASIC**[1], comme simplification de son grand aîné **FORTRAN**[2] (1955). BASIC a été le plus populaire en son temps, accompagnant la naissance de la micro-informatique et des premiers ordinateurs personnels. Le langage interactif **LISP**[3] (1958) fut au cœur de l'apparition de l'Intelligence Artificielle, et sa simplification **Logo** fut aussi utilisée à l'école dès la fin des années 1960 dans le cadre de recherches sur le développement cognitif de l'enfant, puis de l'enseignement des bases de la programmation. La maturation des langages industriels continuait d'être mise à profit par des chercheurs souhaitant disposer de langages de script permettant de piloter de gros systèmes logiciels, mais aussi par des universitaires soucieux de langages à vocation pédagogique pour remplacer ce BASIC un peu spartiate.

Le langage ABC développé à l'institut CWI[4] d'Amsterdam visait ces deux objectifs. C'est Guido van Rossum, chercheur entré au CWI en 1983 et influencé par les idées et les types de données présents dans ABC, qui définit le langage **Python** en 1989, un petit langage bientôt appelé à jouer dans la cour des grands.

L'ambition de son créateur (qui se définissait lui-même comme *dictateur bénévole à vie* jusqu'à ce qu'il se retire en 2018 du comité décisionnel de la *Python Software Foundation*) consistait à créer un langage interactif à objets le plus simple possible, basé sur le langage C[5]. Il s'appuyait entre autres sur une phrase

1. **B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode, par John Kemeny et Thomas Kurtz.
2. **FOR**mula **TRAN**slator, par John Backus.
3. **LIS**t **P**rocessing, par John McCarthy.
4. **C**entrum **W**iskunde & **I**nformatica, où *Wiskunde* signifie *Mathématiques* en néerlandais.
5. L'implémentation de Python par la maison-mère `www.python.org` se nomme CPython, c'est elle que nous utiliserons majoritairement dans ce livre.

d'Albert Einstein : *Les choses doivent être aussi simples que possible, mais pas plus simples.* Le but fut assez remarquablement atteint : la syntaxe est légère, toute valeur est un *objet* (les types sont les classes) que l'on peut ranger dans une variable, passer en argument à une fonction, *etc.*

Certains ont noté une ressemblance avec le langage LISP, mais Guido van Rossum n'a jamais été un grand partisan de la programmation fonctionnelle, bien qu'ayant doté Python des *lambda-fonctions* de LISP (dans une version très limitée) et de primitives comme `map` ou `filter`, voire `reduce`[6]. Ce que van Rossum n'a d'ailleurs jamais voulu reprendre de LISP, et que l'on peut regretter, c'est l'élimination automatique de la récurrence terminale, pour des raisons liées à la mise au point. De son point de vue, la programmation par récurrence doit être évitée. C'est une opinion qui n'engage que lui, mais de fait la programmation en Python utilise majoritairement des itérations et des constructions fonctionnelles comme les *compréhensions de listes*. Les parenthèses de LISP ou les accolades de C/Java, qui structurent les phrases d'un programme, sont remplacées par une *indentation* (distance à la marge) *obligatoire*. Python reste assez solitaire dans cette voie qui s'avère pourtant un bon atout pédagogique d'après mon expérience d'enseignant. Le langage ABC, ou Occam au même moment, tous deux oubliés, en faisaient autant.

| LISP (1958) | Python (1989) | Java (1995) |
|---|---|---|
| `(if (> x 0)`<br>`  (print x)`<br>`  (print y))` | `if x > 0:`<br>`  print(x)`<br>`else:`<br>`  print(y)` | `if (x > 0) {`<br>`  System.out.println(x);`<br>`} else {`<br>`  System.out.println(y);`<br>`}` |

Comme LISP et contrairement à C ou Java, le langage Python est *dynamiquement typé* : les variables sont simplement des noms qui font référence à des objets en mémoire. Comme les autres langages dynamiques, la vérification du typage se fait à l'exécution et non lors d'une phase de compilation.

Les programmes sont organisés en *modules*, permettant de développer ou faire appel à des fonctionnalités distinctes dans des fichiers séparés. Une large bibliothèque de modules est fournie en standard et nous aurons l'occasion d'en explorer plusieurs, ainsi que certains en provenance de sources extérieures.

J'ai choisi la distribution *Anaconda*, car outre le langage Python, elle contient aussi un grand nombre de logiciels scientifiques parfois pénibles à installer individuellement à cause de leurs dépendances. Les systèmes Anaconda et Jupyter mériteraient un livre à eux seuls, j'en dirai le minimum pour les rendre vite opérationnels, en privilégiant le fond sur la forme.

6. L'algorithme qui fit la gloire de Google se nomme *MapReduce* en l'honneur des fonctions `map` et `reduce` mises en avant par LISP.

Dans cette nouvelle édition, nous pénétrerons davantage dans les grosses bibliothèques `NumPy`, `SciPy` et `SymPy` essentielles au scientifique utilisant Python, mais sans y passer un temps démesuré : il existe des ouvrages spécialisés pour un travail approfondi. J'ai été guidé par un souci constant de vous *mettre le pied à l'étrier* et de vous laisser lors des exercices consulter toutes les ressources du Web, notamment les documentations en ligne et les IA (Intelligences Artificielles).

Il m'a paru intéressant de présenter aussi le système *Processing*, destiné à l'éducation et aux artistes numériques mais organisé autour du langage Java. La première édition de ce livre travaillait directement dans *Processing* avec un mode Python, mais ceci n'est plus possible à ce jour avec la version 4 de Processing. J'ai donc opté pour le module `py5` dont le but est de proposer un avatar de Processing intégré dans l'écosystème Python tout en profitant de manière totalement transparente des librairies Java nécessaires.

Pour la même raison, j'ai modifié la section sur l'audio en la recentrant sur Python, tout en profitant du nouveau chapitre sur NumPy. Elle est moins ambitieuse que la précédente mais présente la structure numérique d'une note de musique et peut servir de base à de nombreux projets et approfondissements.

Enfin, le secteur des réseaux de neurones a été élagué et réduit à l'essentiel permettant de pénétrer dans le *machine learning* (apprentissage automatique). Ce sera une occasion de pouvoir analyser un code complexe généré par une IA. D'ailleurs, à certains endroits, une rubrique *Interrogez votre IA* proposera au lecteur des *prompts*[7] à envoyer – s'il le souhaite – à une IA pour avoir des compléments, négocier un programme ou corriger une erreur. Les IA *font* aussi des erreurs, même si elles font dorénavant partie de l'entourage du programmeur.

## Guide de lecture

La première partie est là pour éclaircir le langage lui-même, en détaillant ses constructions linguistiques et ses tournures idiomatiques les plus fréquentes, sans vouloir faire un manuel de référence complet que l'on peut trouver par ailleurs sur `https://docs.python.org/fr/3/`. Ayant observé que les étudiants lisaient de moins en moins les livres de A à Z, chapitre après chapitre (moi aussi je l'avoue), j'ai décidé de structurer le texte comme une FAQ sur Internet (*Frequently Asked Questions*) en essayant d'insister sur les points qui cristallisaient la plus forte demande d'éclaircissements tout en gardant une progression suffisamment linéaire, sans m'y plier complètement, pour le lecteur préférant un apprentissage plus traditionnel. Vous téléchargerez les solutions en Python sur la page Web de ce livre aux Editions Ellipses (`www.editions-ellipses.fr`) ainsi que sur :

`http://jean.paul.roy.free.fr`

7. *Prompt* : un texte court pour engager une demande ou un dialogue.

Qui dit programmation dit algorithmes, mais ce livre traite de la pratique du codage. Ce n'est ni un cours complet ni un ouvrage d'algorithmique qui explorerait toutes les structures de données. Il vous faudrait pour ces dernières par exemple l'indispensable [COR] [8] qui fait à juste titre abstraction du langage de programmation, ou des ouvrages comme [DUR] s'appuyant explicitement sur Python.

La seconde partie consiste en une sélection d'exercices de programmation. En plus de la partie algorithmique de base, essentielle, j'ai dû faire des choix et en ai profité pour vous emmener dans des zones moins enseignées comme le graphisme, l'analyse syntaxique, l'intelligence artificielle ou la musique électronique. Je ne saurai trop insister sur l'**apprentissage actif** du lecteur. *Hands on !* comme disent les anglo-saxons, mettez les mains sur le clavier. Mais rappelez-vous : *le plus simple possible, mais pas plus simple*. Ce sera donc un peu délicat par endroits. Lisez et programmez à votre rythme, consultez les références et aides en ligne, cherchez des compléments sur le Web, travaillez en groupe. L'important consiste à passer des obstacles, à avoir le sentiment que l'on a gravi une marche de plus : il vous faut sentir la progression.

Étudiant en Faculté des Sciences, en IUT, en Ecole d'Ingénieurs, peut-être futur enseignant des lycées préparant un concours ou bien déjà en poste et utilisant Python dans son enseignement, j'espère que ce livre vous plaira et que la partie *Exercices* vous donnera envie d'aller plus loin. J'ai humblement essayé de faire le livre que j'aurai voulu tenir entre les mains en étant étudiant.

Et si vous êtes codeur ou simplement amateur de programmation, sachez que le contenu utilise des mathématiques sans en faire un point de mire. Sa lecture ne requiert jamais plus qu'un niveau de classe terminale scientifique ou exceptionnellement de première année post-Bac. Pour le lecteur à la recherche de sensations plus fortes, il existe des ouvrages de programmation scientifique comme [KAR] et [LAN], ou vraiment axés sur les mathématiques : [JOH], [KIU], ou encore [ZIM], [ANA] avec le logiciel SAGE largement basé sur Python.

La bibliographie contient aussi des références en anglais lorsque je n'ai pas trouvé de livres équivalents en français.

## Remerciements

Cet ouvrage est en grande partie issu de cours donnés en Licence au campus Valrose de l'Université Côte d'Azur à Nice. Merci aux enseignants, moniteurs et étudiants ayant participé à cet enseignement et qui m'ont fait des retours instructifs. Cette édition a été revue et corrigée en partie grâce aux mails de lecteurs que je remercie, ainsi qu'aux assistants IA maintenant disponibles. S'il reste des erreurs, il va de soi que j'en suis l'unique responsable.

8. Cette notation [COR] renvoie au livre de Cormen & al. cité en bibliographie.

# Table des matières

**I L'essentiel du langage** **1**

**1 L'environnement Python** **3**
1.1 Installation de la distribution Python standard avec IDLE . . . 3
1.2 Installation de la distribution Anaconda . . . . . 5
1.2.1 Le gestionnaire de paquets : `conda` . . . . . 6
1.2.2 IDLE est aussi intégré dans Anaconda ! . . . . . 9
1.2.3 L'interface graphique : Anaconda Navigator . . . . . 10
1.2.4 L'éditeur Spyder . . . . . 10
1.2.5 La console Qt : juste le goût de IPython pur . . . . . 11
1.2.6 Le projet Jupyter . . . . . 12
1.3 L'éditeur professionnel : Visual Studio . . . . . 16
1.4 Conclusion. Quel éditeur choisir ? . . . . . 18

**2 Variables, nombres et fonctions** **21**
2.1 Qu'est-ce qu'une variable ? . . . . . 21
2.2 Qu'est-ce qu'un type ? . . . . . 22
2.3 Le type `int` des nombres entiers . . . . . 24
2.4 Les nombres flottants . . . . . 24
2.5 Les nombres complexes . . . . . 26
2.6 Qu'est-ce qu'un module en Python ? . . . . . 27
2.7 Donc pas de nombres rationnels en Python ? . . . . . 28
2.8 L'égalité est-elle fiable sur les nombres flottants ? . . . . . 28
2.9 L'infini existe-t-il en programmation ? . . . . . 29
2.10 Qu'est-ce que la priorité d'un opérateur ? . . . . . 29
2.11 Comment construire couples, triplets, *etc.* ? . . . . . 30
2.12 Quelle différence entre expression et instruction ? . . . . . 31
2.13 Les expressions booléennes . . . . . 32
2.14 Les instructions conditionnelles . . . . . 34
2.15 Comment tirer un nombre au hasard ? . . . . . 36

2.16 Fonctions, paramètres et variables locales . . . . . 36
2.16.1 Le typage optionnel . . . . . 38
2.17 Qu'est-ce qu'une variable globale ? . . . . . 39
2.18 Les différentes formes de paramètres . . . . . 40
2.19 Les arguments optionnels de la fonction `print` . . . . . 42
2.20 Les lambda-expressions . . . . . 42
2.21 Comment documenter une fonction ? . . . . . 44
2.22 La différence entre fonction et méthode . . . . . 45
2.23 Qu'est-ce qu'une fonction récursive ? . . . . . 46
2.23.1 Suivre une fonction à la trace . . . . . 48
2.23.2 Le rôle de la pile de récursion . . . . . 50
2.23.3 La récurrence *terminale* . . . . . 50
2.24 La boucle `while` . . . . . 53
2.25 La boucle `for` . . . . . 55
2.26 Comment interrompre brutalement une boucle ? . . . . . 57
2.27 Le calcul d'une somme en compréhension . . . . . 58
2.28 La complexité d'un calcul . . . . . 58

**3 Traitement du texte : les bases 61**
3.1 Les caractères Unicode . . . . . 61
3.2 Les textes sont des chaînes de caractères . . . . . 62
3.3 Comment programmer avec des chaînes ? . . . . . 65
3.4 Les chaînes formatées . . . . . 66
3.5 Une chaîne peut-elle contenir du code Python ? . . . . . 68

**4 Dessiner 71**
4.1 Le graphisme de la tortue . . . . . 71
4.1.1 L'état d'une tortue . . . . . 72
4.1.2 La fenêtre graphique . . . . . 72
4.1.3 Le graphisme cartésien . . . . . 73
4.1.4 Le graphisme polaire . . . . . 76
4.1.5 Une courbe fractale . . . . . 76
4.2 Le système *py5* : un *Processing* en Python . . . . . 78
4.2.1 Installation de py5 . . . . . 79
4.2.2 Fonctionnement de py5 . . . . . 80
4.2.3 Application à la chute des corps . . . . . 81
4.3 Stéganographie avec le module *Pillow* . . . . . 82

**5 Les collections d'objets 87**
5.1 Collections et séquences . . . . . 87
5.2 Les chaînes et les tuples sont des séquences non mutables . . . . 87

5.3 Les listes sont des séquences mutables . . . 88
5.4 Les constructions fonctionnelles de listes . . . 90
5.5 Déstructuration d'un objet itérable . . . 92
5.6 Suppression d'objets avec `del` . . . 93
5.7 Listes et partage de mémoire . . . 94
5.8 Les ensembles . . . 97
5.9 Les dictionnaires . . . 99
5.10 Les annotations de type . . . 102
5.11 Gestion des erreurs : les exceptions . . . 103

**6 Les entrées-sorties (i/o) 109**
6.1 Le système de fichiers et le module `os` . . . 109
6.2 Ecriture dans un fichier texte sur disque (sortie) . . . 111
6.3 Lecture d'un fichier texte sur disque (entrée) . . . 112
6.3.1 La construction `with` . . . 114
6.3.2 Comment utiliser `split` ? . . . 115
6.4 Des fichiers en mémoire centrale . . . 115
6.5 Exécution d'un script Python au terminal . . . 117

**7 Les objets et les classes 119**
7.1 Définition d'une nouvelle classe d'objets . . . 119
7.2 Les attributs de classe . . . 124
7.3 Utilisation des méthodes spéciales . . . 126
7.4 Boucle `for` et méthodes spéciales : les itérateurs . . . 126
7.5 Les générateurs . . . 128
7.6 Les objets *soft* . . . 129
7.7 La hiérarchie des classes . . . 130
7.8 Les classes abstraites . . . 133
7.9 Qu'est-ce qu'un décorateur ? . . . 134

**8 Traitement avancé du texte 137**
8.1 Expressions régulières : le module `re` . . . 137
8.1.1 Présence d'une sous-chaîne dans une chaîne . . . 138
8.1.2 Les méta-caractères d'un motif . . . 139
8.1.3 Remplacements dans une chaîne . . . 141
8.2 Analyse lexicale avec l'outil `lex` . . . 143
8.3 Analyse syntaxique avec l'outil `yacc` . . . 146
8.4 Recherche de texte dans une page Web : `requests` . . . 150

**9 Les interfaces graphiques (GUI) 151**
9.1 Le module `tkinter` . . . 151
9.1.1 La programmation dirigée par les évènements . . . 151
9.1.2 La fenêtre du programme . . . 152
9.1.3 Dessiner avec les classes `Canvas` et `Button` . . . 153
9.1.4 L'interface graphique comme sous-classe de Tk . . . 156
9.1.5 Les options du gestionnaire `pack` . . . 157
9.1.6 Gestion des évènements souris et clavier . . . 157
9.1.7 Variables de contrôle . . . 160
9.1.8 Compléments sur les évènements souris . . . 162
9.1.9 Animation en style MVC avec `tkinter` . . . 164
9.1.10 La tortue dans une interface graphique `tkinter` . . . 167
9.2 Programmer une animation avec `pyglet` . . . 168

**10 Introduction à SciPy pour la science des données 171**
10.1 Les tableaux numériques de Numpy . . . 171
10.2 Les graphiques avec Gnuplot . . . 175
10.3 Les graphiques avec Matplotlib . . . 177
10.4 Un petit tour de SymPy . . . 180

**II Exercices de programmation 185**

**11 EXERCICES 187**

**Nombres, fonctions, boucles**
11.1 Valeur d'une expression . . . 187
11.2 Avec ou sans résultat ? . . . 187
11.3 Quotient et reste. Écriture binaire d'un entier . . . 188
11.4 Heures – minutes – secondes . . . 188
11.5 Écriture binaire d'un nombre flottant . . . 188
11.6 Décomposition en sous-fonctions . . . 189
11.7 Une fonction, comment ça marche ? . . . 189
11.8 Addition relativiste des vitesses . . . 190
11.9 La formule de Stirling . . . 190
11.10 Payez vos impôts ! . . . 190
11.11 Une variable locale dans une lambda ? . . . 191
11.12 Tirage au hasard de nombres entiers . . . 191
11.13 Nombres fractionnaires (rationnels) . . . 192
11.14 Solutions complexes d'une équation de degré 2 . . . 192
11.15 Nombres complexes et géométrie plane . . . 192
11.16 La division euclidienne dans les entiers . . . 193

11.17 Travail arithmétique sur les chiffres d'un entier . . . . . 193
11.18 Boucles `for` emboîtées . . . . . 194
11.19 Calcul de $x^n$ par dichotomie . . . . . 194
11.20 Tester si un nombre entier est premier (naïf) . . . . . 194
11.21 Nombres premiers entre eux. Le PGCD . . . . . 195
11.22 Calcul de $a^b$ modulo n dans les grands entiers . . . . . 196
11.23 Tester si un grand nombre est premier (Fermat) . . . . . 196
11.24 Calcul des nombres de Fibonacci . . . . . 197
11.25 Vérification qu'un dé est correctement truqué . . . . . 198
11.26 Dérivation numérique approchée . . . . . 199
11.27 Approximation numérique d'un sinus . . . . . 200
11.28 Valeur approchée d'une intégrale simple . . . . . 200
11.29 Coût du (temps de) calcul de n ! . . . . . 202
11.30 Rectification d'un arc paramétré . . . . . 203
11.31 Valeur approchée d'une intégrale double . . . . . 204
11.32 Centre de gravité d'une figure plane . . . . . 205
11.33 Solution approchée d'une équation (dichotomie) . . . . . 205
11.34 Solution approchée d'une équation (Newton) . . . . . 206
11.35 Régime d'un système dépendant du temps . . . . . 206
11.36 Cryptographie à clé publique (RSA/1) . . . . . 207

**Chaînes de caractères**

11.37 Affichage d'une table de multiplication . . . . . 208
11.38 Construction de l'alphabet minuscule . . . . . 208
11.39 Les plus longs mots d'une phrase . . . . . 208
11.40 Travail textuel sur les chiffres d'un entier . . . . . 209
11.41 Récurrence sur des chaînes . . . . . 209
11.42 Le codage secret selon César . . . . . 210
11.43 Les palindromes . . . . . 210
11.44 Le nombre de Champernowne . . . . . 211
11.45 Vérification d'un code bancaire IBAN . . . . . 211
11.46 Les règles de réécriture . . . . . 212
11.47 L'addition binaire avec retenue . . . . . 212
11.48 Analyse lexicale *à la main* . . . . . 213
11.49 HTML et Unicode . . . . . 214

**Graphisme de la tortue, Processing**

11.50 Graphismes polaire et cartésien . . . . . 214
11.51 Tracé d'un cercle avec centre et rayon . . . . . 215
11.52 Mangez des oursins ! . . . . . 215
11.53 Un traceur minimal de courbes paramétrées . . . . . 215
11.54 Dessin d'une courbe en coordonnées polaires . . . . . 216

11.55 Le jeu du chaos . . . 217
11.56 Une courbe fractale du dragon . . . 217
11.57 La tortue, le hasard et les flaques . . . 218
11.58 Génération d'une fractale par un L-système . . . 219
11.59 Animation d'une collision . . . 220

**Les séquences**

11.60 Tirage d'un élément aléatoire dans une séquence . . . 221
11.61 Courbe d'un système dépendant du temps . . . 222
11.62 Inversion des mots d'une phrase . . . 222
11.63 Recherche séquentielle dans une liste non triée . . . 223
11.64 Comptage d'éléments dans une collection . . . 223
11.65 Extraction d'éléments dans une séquence . . . 224
11.66 Recherche dichotomique dans une liste triée . . . 224
11.67 Compactage d'une liste . . . 225
11.68 Le drapeau hollandais de Dijkstra . . . 225
11.69 Les fonctionnelles `map` et `reduce` . . . 226
11.70 Vérification des éléments d'une collection . . . 227
11.71 Attention aux mutations dans les listes ! . . . 228
11.72 Le triangle de Pascal . . . 228
11.73 Tri stupide d'une liste . . . 229
11.74 Tri d'une liste sur place, par sélection . . . 230
11.75 Chronométrage d'un algorithme sur une longue liste aléatoire . 231
11.76 Tri par insertion : récurrence et itération . . . 231
11.77 Tri d'une liste avec copie, par fusion . . . 232
11.78 Tri rapide . . . 233
11.79 Trier sur des critères exotiques . . . 234
11.80 Crible d'Ératosthène pour les nombres premiers . . . 235
11.81 Cryptographie à clé publique (RSA/2) . . . 235
11.82 Unicode et les bizarreries typographiques . . . 236
11.83 Opérations sur des matrices . . . 238
11.84 Un carré magique d'ordre impair . . . 239
11.85 Message caché dans une image . . . 240
11.86 Automates cellulaires 1-D . . . 240
11.87 Les piles de données . . . 242
11.88 Retour sur la pile de récursion . . . 243
11.89 Polynômes creux . . . 245
11.90 Polynômes de Legendre . . . 248
11.91 Polynôme d'Interpolation de Lagrange . . . 248
11.92 Racines simples d'un polynôme (1) . . . 250

11.93 Racines simples d'un polynôme (2) . . . . . 251

**Itérateurs, ensembles et dictionnaires**

11.94 Fermeture-éclair : l'itérateur `zip` . . . . . 252
11.95 Générateur des éléments distincts d'une liste . . . . . 253
11.96 Réunion et intersection de listes . . . . . 253
11.97 Produit cartésien d'ensembles . . . . . 254
11.98 Combinaisons dans un ensemble . . . . . 255
11.99 Dictionnaires et fréquences . . . . . 256
11.100 Mémorisation : remplacer le temps par l'espace . . . . . 256
11.101 Automate cellulaire 2-D : le Jeu de la Vie . . . . . 257
11.102 Arbres binaires d'expression . . . . . 259
11.103 La machine à pile VRISC1 . . . . . 263
11.104 La machine VRISC2 : les sauts à étiquette . . . . . 264
11.105 Parcours itératif d'un arbre binaire . . . . . 266
11.106 Introduction à la simplification symbolique . . . . . 267
11.107 L'algorithme de dérivation symbolique . . . . . 268
11.108 Calcul numérique avec `numpy` . . . . . 270
11.109 Ajustement numérique d'une courbe : SciPy . . . . . 272

**Expressions régulières**

11.110 Quelques expressions régulières . . . . . 274
11.111 Mais tout n'est pas régulier ! . . . . . 274
11.112 Fin de l'analyseur lexical du § 8.2 avec *lex* . . . . . 276
11.113 Fin de l'analyseur syntaxique du § 8.3 . . . . . 276
11.114 Une calculette interactive avec *lex & yacc* . . . . . 277
11.114.1 L'analyseur lexical (*lexer*) . . . . . 279
11.114.2 L'analyseur syntaxique (*parser*) . . . . . 279
11.114.3 L'évaluateur . . . . . 281
11.115 Retour sur les *cpbb* à dos de *yacc* . . . . . 282

**Fichiers, Web**

11.116 Écriture dans un fichier texte sur disque . . . . . 283
11.117 Lecture d'un fichier texte sur disque . . . . . 283
11.118 Transformation d'un fichier texte sur disque . . . . . 284
11.119 Lecture d'un fichier CSV : en Python pur . . . . . 284
11.120 Échange de données au format JSON . . . . . 285
11.121 Lecture d'un fichier CSV : avec l'IA et `pandas` . . . . . 286
11.122 Sérialisation des données : `json` ou `pickle` ? . . . . . 288
11.123 Lancer un programme à heure donnée : `cron` . . . . . 289

**Objets et projets avancés**

11.124 Classe des matrices $n \times p$ sans `numpy` . . . . . 290

11.125 Une poursuite dans la classe des tortues en `py5` . . . . . 292
11.126 Incursion dans le graphisme 3D avec `tkinter` . . . . . 295
11.127 Ajustement par programmation génétique . . . . . 297
11.128 Introduction au moteur physique `pymunk` . . . . . 301
11.129 Décoration de la récurrence terminale . . . . . 308
11.130 Des neurones pour l'IA : le perceptron . . . . . 309
11.130.1 Un neurone formel simplifié . . . . . 309
11.130.2 L'algorithme d'apprentissage d'un perceptron . . . . . 311
11.130.3 Les réseaux de perceptrons à couches cachées . . . . . 312
11.131 La programmation audionumérique . . . . . 315
11.131.1 Le son numérique . . . . . 315
11.131.2 Construire une note de musique en python . . . . . 316
11.131.3 Exercices autour de la construction d'une note . . . . . 317
11.131.4 Introduction minimaliste au solfège pour informaticien 321

**Bibliographie** **325**

**Index** **329**

# Première partie

# L'essentiel du langage

# 1 L'environnement Python

*J'ai essayé de ne pas trop détailler l'**environnement intégré de programmation** ou IDE*[1]*, vous trouverez sur le Web beaucoup de tutoriaux à cet effet. Suivant la formation que vous suivrez, vous serez sous IDLE, Spyder, Jupyter, Emacs, Eclipse, PyCharm, XCode ou Visual Studio, que sais-je ? Il est impossible de les passer tous en revue. Pour les débutants,* **IDLE** *(par les créateurs de Python) et* **Jupyter** *(plus évolué, avec la plateforme Anaconda) sont de bons choix, avec* **Visual Studio Code** *pour les plus hardis. Nous verrons comment rédiger et exécuter avec eux un programme.*

## 1.1 Installation de la distribution Python standard avec IDLE

La maison-mère **www.python.org** distribue le langage Python avec un petit éditeur intégré **IDLE**, fournissant le minimum requis pour programmer en Python. Il s'agit sans doute de la solution la plus simple au niveau du lycée en pré-BAC. Ultérieurement, il faudra disposer d'un environnement élargi pour la sience des données, tel celui d'Anaconda qui reste utilisable par des débutants.

La distribution standard se trouve sur **www.python.org/downloads** qui vous proposera la version adaptée à votre machine. Une fois l'installation terminée, fabriquez un raccourci vers le programme IDLE, sur le bureau ou dans un menu de démarrage. C'est ce programme qui va lancer Python dans une console avec un caractère d'invite (*prompt*) `>>>`. Par exemple, aux couleurs près :

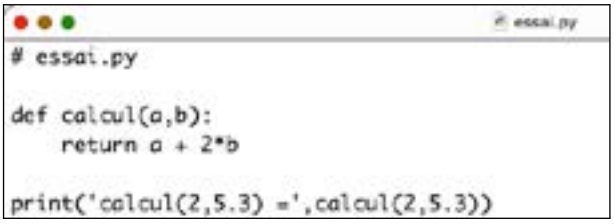
```
1 Python 3.14.0, Oct 7 2025, on darwin          # ← darwin = Apple
2 >>> 2 + 3 * 4
3 14
4 >>> from math import sqrt                     # la racine carrée
5 >>> sqrt(2) ** 2                             # un calcul approché :-)
6 2.0000000000000004
```

1. IDE = *Integrated Development Environment*, comprenant les outils essentiels pour programmer dont en premier lieu un éditeur de texte et une console d'exécution.

La documentation de Python est un fichier `.html` situé dans le répertoire de l'application Python, à côté de IDLE. Une aide en ligne sommaire est disponible à la console avec la fonction `help`.

```
>>> help(sqrt)                                          # aide en ligne
Help on built-in function sqrt in module math:
sqrt(x, /)
    Return the square root of x.
```

Le caractère dièse `#` débute un *commentaire* sur la ligne, ignoré par Python. On travaille assez peu avec cette ligne de commande (*shell* ou *toplevel*) assez spartiate, plutôt pour de petits tests de programmes situés dans une fenêtre d'édition, que vous obtiendrez soit par le menu *File/New File...*, soit en ouvrant un fichier Python d'extension `.py` par *File/Open...*. Ouvrons une nouvelle fenêtre d'édition et dans cette fenêtre définissons une fonction `calcul` à deux paramètres `a` et `b`, fonction dont le résultat sera simplement le nombre $a + 2b$.




Si maintenant vous appuyez sur la touche F5 de votre ordinateur, ou si vous invoquez le menu *Run/Run Module*, il vous sera demandé de sauvegarder le contenu de l'éditeur sur disque, ici sous le nom `essai.py`. Ce contenu sera ensuite analysé sommairement pour détecter d'éventuelles fautes de syntaxe, la mémoire sera remise à neuf (*Restart Shell*) et enfin le contenu sera évalué de haut en bas. Son évaluation produira au passage des affichages de résultats à la console toplevel.

```
========= RESTART: /Users/roy/Documents/prog-python/essai.py ======
calcul(2,5.3) = 12.6
```

Les **tests** du programme se font plutôt **dans l'éditeur** qu'au toplevel, de manière à ce qu'ils soient exécutés automatiquement sans avoir à les retaper à chaque modification du programme. Nous verrons au § 6.5 comment rédiger un *script*, petit programme exécutable au terminal en-dehors d'IDLE.

Sur Mac, le répertoire `"/Applications/Python 3.x/"` contient les logiciels pour l'utilisateur, dont IDLE. Mais les entrailles de la bête se situent à un niveau où seul le spécialiste ira mettre le nez, par exemple (sur Mac) :

`/Library/Frameworks/Python.framework/Versions/Current` .

dans le dossier `bin` duquel on trouve les commandes au Terminal `python3`, `idle3` et `pip3`. Cet environnement intégré minimal IDLE est rudimentaire mais plus que suffisant pour une première initiation à Python. Il est trop spartiate pour un développement avancé ou professionnel.

## 1.2 Installation de la distribution Anaconda

Pour les scientifiques, nous conseillons plutôt la **distribution Anaconda**.



**ANACONDA**

La société *Anaconda* (spécialisée dans l'analyse des données : *data science*) propose en effet une version gratuite mais très complète d'un environnement pour programmer en Python. Vous avez deux choix possibles pour l'installer :

- soit la version complète **Anaconda** (> 4 Go) contenant Python, de très nombreuses bibliothèques scientifiques (`numpy`, *etc.*), une interface graphique *Navigator*, et le gestionnaire `conda` pour les mises à jour.
- soit une version **Miniconda** (< 1 Go) contenant Python et quelques bibliothèques seulement, avec `conda` pour installer les autres.

  `https://docs.anaconda.com/distro-or-miniconda/`

Si vous avez suffisamment d'espace disque, téléchargez et installez *Anaconda Distribution*, cela vous simplifiera la vie. Prenez éventuellement un compte sur la plateforme *Anaconda Cloud*, vous pourrez profiter de nombreux contenus. Utilisez un mot de passe fort.

Vérifions si l'installation s'est bien passée. Je suis sur Mac, vous voudrez donc adapter à votre système d'exploitation Windows ou Linux. Ouvrons l'application *Terminal* située sur Mac dans le dossier `Applications/Utilitaires/`. Sur Windows, lancez l'application *Anaconda Prompt* qui vient d'être installée dans le menu `Démarrer`. Vous devez voir `(base)` au début de la ligne de commande. Je suppose hardiment que tout va bien : vous êtes dans l'environnement `base` d'Anaconda. Fermez le Terminal et poussez un soupir de soulagement.

Sur mon Mac, le Python installé par Anaconda vit dans le répertoire... euh, lequel ? Essayons au Terminal :

```
11 (base) $ which python3        # en Unix au Terminal sur Mac et Linux,
12 /Users/roy/opt/anaconda3/bin/python3        #  "where" sous Windows...
```

Donc les commandes `python3` et `idle3` concernent bien le Python d'Anaconda, celui de la maison-mère étant, lui, lancé à la souris par l'icône de l'application IDLE dans `"/Applications/Python 3.x"`. Donc deux Python co-existent paisiblement... ou pas. Il faut savoir avec lequel on travaille.

### 1.2.1 Le gestionnaire de paquets : `conda`

Nous serons amenés un jour ou l'autre à devoir utiliser une bibliothèque logicielle extérieure écrite en Python – on parle de **paquet** (*package*). Pour l'installer sous Anaconda, nous utiliserons au terminal un **gestionnaire de paquets et d'environnements** (*environment and package manager*) nommé `conda` [2]. Il existe un autre gestionnaire de paquets `pip` et d'environnements `venv`, mais nous veillerons à nous en tenir à `conda` pour éviter les soucis [3] ! Pour savoir si `conda` est bien installé, demandons son numéro de version. Il est recommandé de tenir `conda` à jour (`conda update conda`).

```
(base) $ conda update --all                    # mise à jour globale !
Executing transaction: done
(base) $ conda --version
24.11.0
(base) $ python3 --version
3.12.8                          # un poil en retard sur la maison-mère
```

Devant le prompt `$` de mon Terminal est apparu `(base)` qui me dit dans quel **environnement** de `conda` je me trouve. Je suis dans l'environnement de base, avec un grand nombre de paquets logiciels que je peux immédiatement utiliser. Pour savoir lesquels, je demande leur liste avec `conda list` (il y en a *beaucoup* !). En voici seulement quelques-uns :

```
(base) $ conda list
packages in environment at /opt/anaconda3:
# Name                    Version
anaconda-navigator        2.6.3
conda                     24.11.0
ipython                   8.30.0
jupyter                   1.1.1
matplotlib                3.9.3
notebook                  7.3.1
numpy                     1.26.4
```

Ne vous fiez pas aux numéros de version, ils évoluent au fil des mois. Vous pouvez en théorie travailler dans l'environnement `(base)` pour du Python pur, mais si vous commencer à importer des paquets [4] extérieurs, il se peut que tel paquet A

2. Documentation sur `https://conda.io/docs/`. L'application *Anaconda Navigator* offre une interface graphique pour `conda`, mais il est souvent plus rapide et sain de travailler au terminal.

3. Voir : `https://www.anaconda.com/blog/using-pip-in-a-conda-environment`.

4. Ces paquets peuvent être des modules ou des répertoires de modules et de paquets.

utilise un autre paquet P en version 7.2 alors qu'un paquet B demande le paquet P en version 6.5, vous voyez le problème qui surgit. C'est encore pire s'ils exigent des versions de Python différentes !

Imaginons que nous souhaitions programmer une simulation avec le *moteur de calcul physique* `Pymunk`. Il est sain de créer avec `conda create` un nouvel environnement – par exemple de nom `simul` – et d'y installer ce paquet (qui ne fait pas partie des paquets initiaux de notre distribution Anaconda).

```
29 (base) $ conda create --name simul                    # --name ou -n
30 ...environment location: /Users/roy/opt/anaconda3/envs/simul
```

Les environnements se trouvent donc dans le répertoire `anaconda3/envs/`, vérifiez-le en ouvrant ce répertoire ou mieux avec la commande `conda info --envs` où `--envs` peut s'abréger en `-e`, vous verrez que l'environnement courant est précédé d'une étoile `*`. Plaçons-nous dans l'environnement `simul` encore vide avec la commande `conda activate`.

```
31 (base) $ conda activate simul
32 (simul) $ conda list                                  # simul est vide !
33 (simul) $
```

Maintenant que nous sommes dans l'environnement `simul`, installons le moteur physique `Pymunk` dont la page Web est `https://www.pymunk.org`. Celle-ci me propose d'installer le paquet `pymunk` version 7.0.1 avec l'un des gestionnaires `pip` ou `conda`. Étant sous Anaconda, j'opte pour un `conda install` et seulement en désespoir de cause aurais-je eu recours à `pip` (qui fait partie des paquets d'Anaconda[5]). On me dit même d'utiliser le **canal** `conda-forge`. J'utilise ci-dessous la manière traditionnelle avec `--channel` au lieu de simplement `-c` :

```
34 (simul) $ conda install --channel conda-forge pymunk
35 ...
36 Executing transaction: done
```

Pour installer un paquet, Anaconda utilise en effet des **canaux de recherche** (*channels*) et de manière générale, si aucun d'eux ne trouve ce que je cherche, j'essaye le canal `conda-forge` qui contient de nombreux paquets scientifiques. Ceci dit, avec un navigateur Web, j'aurais pu mettre la main sur la page du canal `anaconda.org/conda-forge` et y trouver `pymunk`, pour obtenir une requête `conda install conda-forge::pymunk` avec une troisième syntaxe équivalente.

5. Les gestionnaires `pip` et `conda` sont parfois en conflit, surtout si l'on ne s'assure pas qu'on utilise les bons, au cas où l'on conserve plusieurs distributions de Python !

```
(simul) $ conda list
...
pymunk          7.0.1           py312hea69d52_0         conda-forge
python          3.12.8          hc22306f_1_cpython      conda-forge
...
```

Vérifions au Terminal que tout fonctionne bien en essayant d'importer `pymunk` au sein de l'environnement `simul`.

```
(simul) $ python3
>>> import pymunk
>>> pymunk.version                          # ok, l'importation est correcte
'7.0.1'
>>> quit()
(simul) $
```

Ce dépôt `conda-forge` me semble plus qu'intéressant, je l'ajoute donc en tête (`--add`) ou en queue (`--append`) à la liste des canaux qui seront essayés dans l'ordre lors d'une nouvelle installation.

```
(simul) $ conda config --add channels conda-forge   # ajouter en tête
(simul) $ conda config --get channels               # puis vérifier
--add channels 'defaults'      # lowest priority
--add channels 'conda-forge'   # highest priority
```

Une autre manière de le vérifier consiste à demander `conda config --show` montrant la configuration de `conda`. Les canaux de `defaults` sont ceux hébergés par la société Anaconda, tandis que `conda-forge` est une communauté libre sur *GitHub*. Nous n'aurons pas besoin d'autres canaux dans ce livre.

Vous pouvez **supprimer** un paquet `paq` dans l'environnement courant avec la commande `conda remove paq`, ou même l'environnement `simul` tout entier à partir de l'environnement `base` avec `conda env remove --name simul`. Pour remonter dans `base`, il suffit de désactiver l'environnement courant.

```
(simul) $ conda deactivate          # retour à l'environnement de base
(base) $ python3
>>> import pymunk
ModuleNotFoundError: No module named 'pymunk'      # pas dans base !
>>> quit()
(base) $
```

Oui, il faut mémoriser quelques commandes au terminal, mais on s'y fait vite. Au besoin, demandez à Google la dernière version de la « **conda cheat sheet** » avec votre navigateur et la vie pétillera ☺. Ou pas, si vous êtes allergique au Terminal, auquel cas vous voudrez passer par *Anaconda Navigator* (page suivante).

**Remarque** — S'il vous arrive de perdre le prompt `(base)` au Terminal, vérifiez que `conda` est bien accessible dans le PATH, et ré-initialisez-le avec `conda init`.

### 1.2.2 IDLE est aussi intégré dans Anaconda !

L'éditeur et toplevel IDLE (§ 1.1), conçu par la maison-mère `www.python.org`, a été intégré à Anaconda pour faire du travail simple, ce qui montre qu'il n'est pas utile de télécharger la distribution de la maison-mère pour profiter de IDLE qui est accessible en ligne de commande sur Anaconda.

```
58 (base) $ which idle3          #  où est l'exécutable IDLE maintenant ?
59 /Users/roy/opt/anaconda3/bin/idle3                #  dans Anaconda !
60 (base) $ idle3&               #  j'ouvre une fenêtre IDLE
```

L'esperluette `&` à la fin du mot `idle3` est une convention Unix pour signifier que l'exécution de IDLE doit se faire *en tâche de fond*, permettant d'entrer d'autres commandes au Terminal qui ne sera pas bloqué jusqu'à la sortie de IDLE.

Attention donc à **ne pas vous mélanger les pinceaux** si vous gardez la distribution de la maison-mère en même temps qu'Anaconda ! Il y aura deux exécutables `idle3` dans la machine, par exemple chez moi :

```
/Library/Frameworks/Python.framework/Versions/Current/bin/idle3 (*)
/Users/roy/opt/anaconda3/bin/idle3                              (**)
```

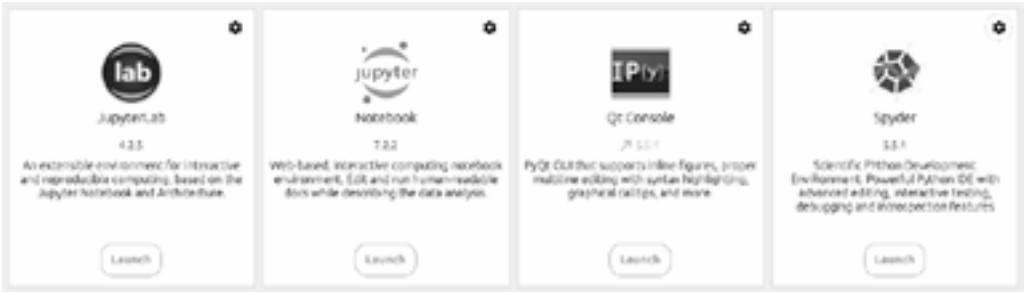
La différence inestimable entre les deux est que le premier (*) ne bénéfiera **pas** des paquets installés dans Anaconda, alors que le second (**) oui. Si je demande `import numpy` dans le premier, j'aurai une erreur *No module named 'numpy'* et je devrai alors l'installer avec `pip3` (celui de (*) !), alors que le second considère que `numpy` est déjà présent dans Anaconda – comme on le vérifie avec `conda list`. Il est possible de gérer des environnements virtuels dans (*) avec `venv` et `pip`, je n'en parlerai pas, vous pouvez demander à une IA de vous montrer les différences entre `conda` et `venv`. N'hésitez pas à utiliser une IA en mode gratuit : *ChatGPT*, *Claude*, *DeepSeek*, *Mistral*, *etc.*

Il peut être intéressant de réserver un environnement `idle` pour faire dans IDLE (d'Anaconda) des programmes ne nécessitant que les paquets initiaux d'Anaconda. Travailler directement dans `base` est possible mais assez déconseillé.

### 1.2.3 L'interface graphique : Anaconda Navigator

Ananconda propose une application **Anaconda Navigator** que vous mettrez dans vos raccourcis. Si vous avez installé *Miniconda*, utilisez `conda` pour installer le paquet `anaconda-navigator`.

Une fois Navigator lancé, sa fenêtre s'ouvre, dans laquelle vous trouverez différents logiciels (*JupyterLab*, *Notebook*, *Qt Console*, *Spyder*, *VS Code etc.*).




Parmi eux se trouvent **Jupyter Notebook** et **Spyder** qui vous serviront à rédiger des programmes si vous n'avez pas opté pour IDLE.

Vous pouvez gérer les environnements avec Navigator, plutôt qu'en ligne de commande avec `conda`. En cliquant sur le bouton *Environments* sur la gauche de la fenêtre du Navigator, vous allez voir apparaître nos trois environnements courants : `base`, `simul` et `idle` avec au-dessous cinq boutons : pour créer un nouvel environnement, dupliquer, importer[6], sauvegarder ou supprimer un autre. Sur la droite, vous visualisez les paquets de chaque environnement, avec la possibilité d'en rajouter ou d'en supprimer. Vous avez même le bouton *Channels* pour gérer les canaux de recherche.

Mieux vaut avoir lu la section 1.2.1 précédente pour bien saisir le sens des manipulations, et nous vous renvoyons aux tutoriels en ligne :

`https://docs.anaconda.com/navigator/tutorials/`

notamment aux chapitres *Managing environments*, *packages* et *channels*. Vous trouverez aussi des vidéos sur YouTube. Je n'utilise pas trop *Navigator*.

### 1.2.4 L'éditeur Spyder

Outre la gestion des environnements et des canaux, Navigator permet de lancer des logiciels, dont certains sont de véritables usines à traitement des données, complexes à utiliser sans un livre spécialisé sous la main. La tendance actuelle

6. Vous pouvez télécharger ou transmettre à autrui une description d'environnement sous la forme d'un fichier `.yml`, demandez à votre IA de vous parler des fichiers YAML. L'analogue pour `pip` est le fichier `requirements.txt`.

est à l'Intelligence Artificielle, qui a pénétré aussi le monde des codeurs, et il existe déjà un *Anaconda AI Navigator* collaboratif permettant d'installer les outils d'IA.

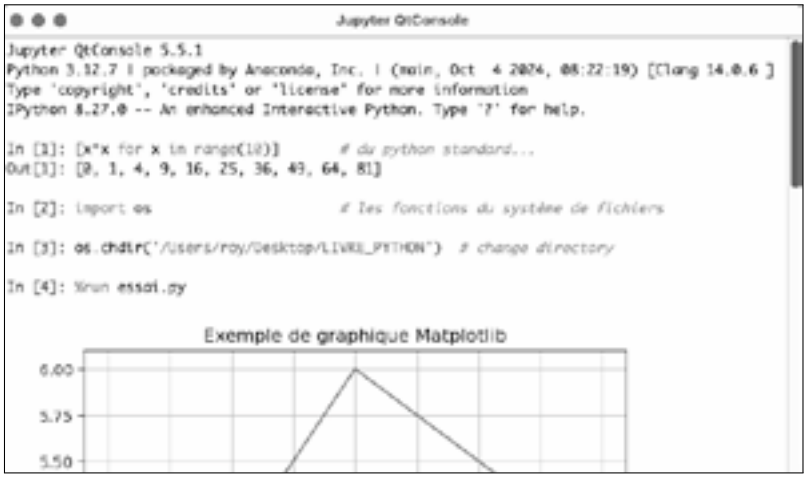
Parmi tous ces logiciels, l'éditeur **Spyder** a des aspects plaisants, il est maintenu par une équipe indépendante dévouée et ambitieuse, et peut se lancer *via* Navigator : c'est un paquet d'Anaconda présent dans l'environnement `base`.

**Remarque** — Sur `https://www.spyder-ide.org`, il est possible de télécharger Spyder comme **application autonome**. De nombreux paquets pour l'analyse des données sont déjà intégrés (`numpy`, `matplotlib`, `pandas`, *etc.*).

En ouvrant la fenêtre de Spyder, vous verrez sa console toplevel (à droite en bas) basée sur **IPython**. Ce toplevel est plus complexe que celui d'IDLE mais avec un peu de pratique, il s'avère productif. Il permet de voir les graphiques produits soit dans des fenêtres séparées, soit directement au toplevel.

Spyder est la conjonction d'une console IPython, d'un éditeur de programmes pouvant utiliser des cellules (et destiné à évoluer à terme vers le concept de **notebook** comme dans Jupyter), et d'une vision sur le système de fichiers. Cet éditeur gratuit vaut néanmoins le détour même s'il a des soucis par moments, en général résolubles *via* un forum dédié et réactif. Vous trouverez de l'aide sur le site de Spyder et sur YouTube, avec des tutoriaux et des vidéos de prise en main.

### 1.2.5 La console Qt : juste le goût de IPython pur




Pour avoir une idée du fonctionnement de IPython, lançons avec Navigator la **Console Qt** : `https://qtconsole.readthedocs.io`. On peut aussi au Terminal entrer la commande : `jupyter qtconsole`. Une simple fenêtre toplevel surgit, de nom *Jupyter QtConsole*, sans éditeur.

Vous ne pouvez qu'entrer des demandes de calcul ou d'affichage de graphiques, ainsi qu'exécuter un fichier avec la **commande magique**[7] `%run`. Vous produisez et modifiez ce fichier avec un éditeur de programmes de votre choix (Spyder, Visual Studio, PyCharm, etc.), et vous exécutez dans la Qt-Console avec `%run`.

Le site plus général de IPython est `https://ipython.org`. La *console IPython* de Spyder est une variante de **QtConsole**. Il s'agit d'un lieu interactif où l'on peut avec la commande magique `load_ext` charger des extensions (nous utiliserons `load_ext cython`), échanger des données entre plusieurs langages dont majoritairement Python, et qui sera utilisé simplement comme une calculette évoluée ou bien pour visualiser les affichages et graphiques d'un programme lors de son exécution. Dans Spyder, la pression de l'icône *Exécuter* (**F5**) dans l'éditeur déclenche à la console IPython un appel à la commande magique `%runfile`.

Les *notebooks* de Jupyter sont aussi basés sur IPython, mais apportent des fonctionnalités supplémentaires : multi-langages (Python, R, Julia, Javascript, Matlab, Fortran...), édition en TEX, conversion vers PDF ou HTML, *etc.*).

Voir [ROSS], [VAU], [VAN] pour la prise en main de IPython et Jupyter en science des données. Ainsi que les fabuleuses ressources du Web et des IA.

### 1.2.6 Le projet Jupyter

Les travaux autour de IPython, lancé en 2001, s'élargiront pour aboutir vers 2014 au **projet Jupyter** `https://jupyter.org` du nom de trois langages de programmation scientifique : **Ju**lia, **Pyt**hon et **R**. Il a aussi été influencé par les *notebooks* de Mathematica, créé vers 1988 par Steven Wolfram. L'ambition de Mathematica, reprise par Jupyter, était de créer un environnement de production de documents électroniques contenant à la fois du texte bien mis en forme et du code exécutable produisant des résultats qui s'intègrent dynamiquement au texte.

La naissance de Jupyter visait à faciliter la rédaction de ces *carnets* électroniques ou **notebooks** contenant à la fois du texte enrichi (avec *Markdown* et LATEX), des graphiques résultats de calculs, et du code – Python ou autre – exécutable[8]. Un *notebook* Python – carnet[9] de notes – est un fichier d'extension `.ipynb`[10] organisé en **cellules** plus puissantes que celles de l'éditeur Spyder. Vous pourrez ainsi rédiger un texte scientifique (un exposé de maths, de physique ou

7. Les *commandes magiques* de IPython, préfixées par `%` pour une ligne et `%%` pour une cellule, permettent d'effectuer des actions. On y trouve `%run`, `%time`, `%timeit`, `%load`, `%load_ext`, `%gui`, `%matplotlib`, `%pwd`, `%ls`, `%%bash`, `%%latex`, `%%markdown`.... Demandez `%lsmagic` au toplevel.

8. Même si Python est le principal, une centaine d'autres langages sont disponibles. Vous les trouverez en cherchant « Jupyter kernels » dans votre moteur de recherche ou avec une IA.

9. Une application sur iPad se nomme *Carnets*.

10. Un fichier `.ipynb` est sauvegardé au format JSON et non comme un fichier usuel `.py`. Il faut au moins un *Notebook Viewer* pour le consulter et Jupyter pour le modifier.

de biologie par exemple) contenant des calculs interactifs. Pédagogiquement, cela permet de joindre la *rédaction* – trop souvent délaissée – au *codage*.

La force de Jupyter est de **mélanger l'éditeur de texte et la console**[11] pour former ce que l'on nommait déjà en LISP au début des années 1980 un *éditeur-toplevel*, initié par le celèbre éditeur Emacs.

### Les *notebooks* (carnets) de Jupyter

*Un carnet (notebook) contient les* ***entrées et sorties*** *d'une session interactive ainsi que du* ***texte narratif*** *qui accompagne le code mais qui n'est pas destiné à être exécuté. La* ***sortie enrichie*** *générée par l'exécution du code, y compris avec du HTML, des images, vidéos et graphiques, est intégrée dans le carnet, ce qui en fait un document complet et autonome issu d'un calcul.*
*(jupyter-notebook.readthedocs.io)*

Une documentation des *carnets* se trouve en anglais sur `(*)` très complet, et en français sur `(**)` pour l'étudiant en sciences physiques :

`https://jupyter-notebook.readthedocs.io` `(*)`
`https://pyspc.readthedocs.io` `(**)`

Commençons par installer dans votre environnement un petit paquet trouvé dans `https://pypi.org/org/jupyter`, et permettant de **franciser** Jupyter.

```
61 (base) $ conda install -c conda-forge jupyterlab-language-pack-fr-FR
```

Ceci fait, vous disposez de deux manières pour ouvrir **Jupyter Notebook** :

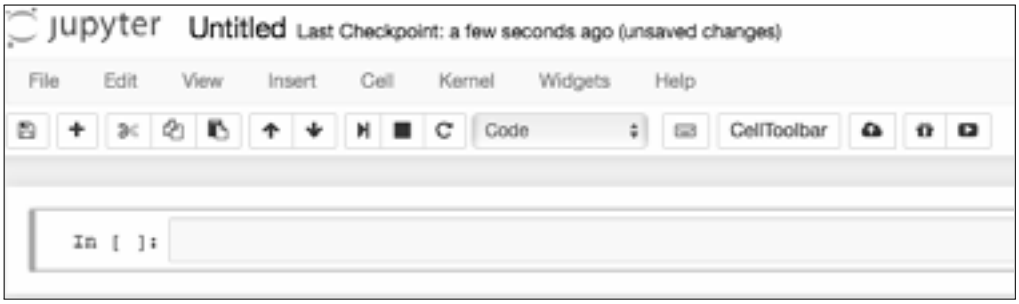
- Soit vous le lancez au Terminal dans l'environnement courant avec la commande `jupyter notebook` et au besoin en installant le paquet `jupyter`.
- Soit avec *Anaconda Navigator* : cliquez sur *Environments* pour choisir votre environnement (`base` ou bien par exemple notre `simul`). Ensuite revenez dans *Home* et cliquez sur *Launch* dans le cadre *Jupyter Notebook*, ou bien dans *Install* s'il n'est pas déjà installé dans cet environnement. Un terminal surgit qui va lui-même lancer une commande ouvrant un serveur Web dans votre navigateur à l'adresse `http://localhost:8888`. Les navigateurs conseillés à ce jour sont Chrome, Edge, Firefox et Safari.

Bref, dans les deux cas, vous vous retrouvez dans l'explorateur de fichiers de *Jupyter* qui affiche votre répertoire, dans lequel vous allez naviguer pour ouvrir un carnet `.ipynb` pour édition (il y a de nombreux exemples sur les sites web de [ROSS] et [VAN]). Attention, ce n'est pas un fichier textuel simple `xxx.py` en

11. Vous pouvez l'essayer sur `https://jupyter.org/try` avec Python, R, C++, Scheme, *etc.*

Python pur, mais un fichier structuré au format JSON que nous rencontrerons plus tard dans ce livre. Peu importe.

Nous allons créer un nouveau *carnet* avec le menu **Nouveau** situé en haut à droite, optez pour *Python 3 (ipykernel)*. Vous arrivez alors dans la fenêtre d'un carnet de nom *Untitled* en haut à gauche, sur lequel vous cliquez pour le changer en `essai` (l'extension `.ipynb` sera ajoutée automatiquement).



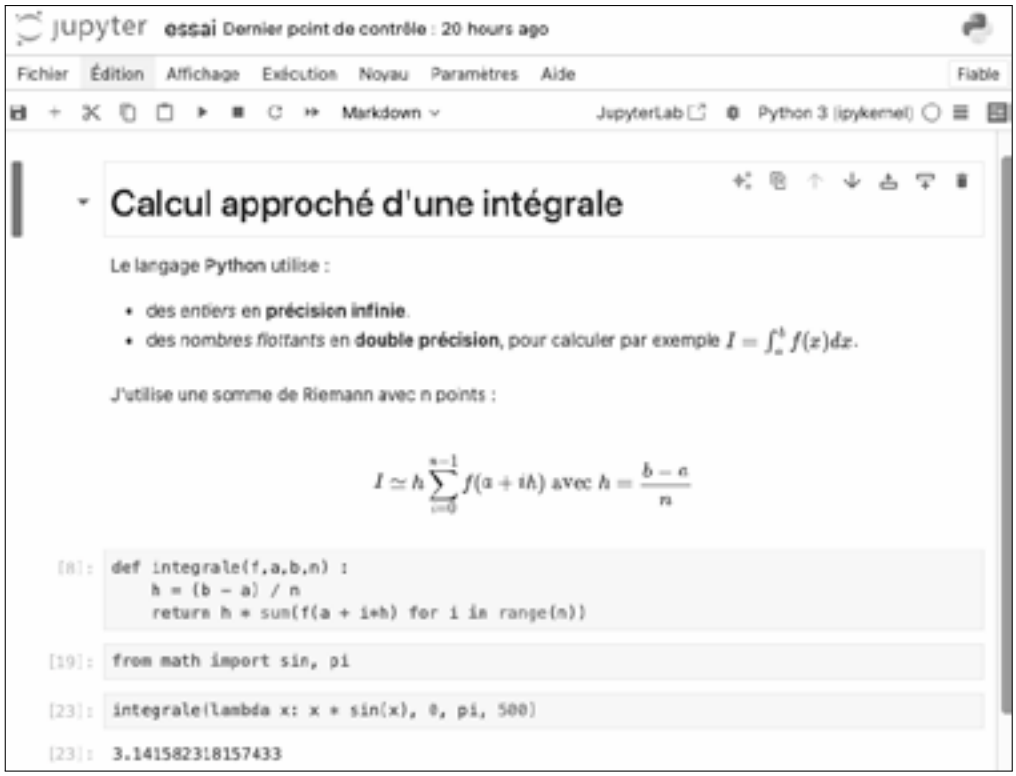

La première et unique cellule ci-dessus est en mode **Code** comme indiqué dans un menu déroulant : elle est censée contenir du code Python. Utilisons ce menu déroulant pour la mettre en mode **Markdown**, une sorte de HTML caché destiné à produire un **texte enrichi** avec balises et formules mathématiques en **TEX** (j'ai bien entendu installé le logiciel TEX sur ma machine, chez moi *TeXShop*). Le menu *Aide* contient une entrée pour apprendre *Markdown*[12]. Voici ce que j'entre dans les quatre premières cellules qui sont dans ce même mode ; j'utilise un '`------`' virtuel pour indiquer la séparation des cellules.

```
# Calcul approché d'une intégrale
-------
Le langage [**Python**](http://www.python.org) utilise :
-------
* des *entiers* en **précision infinie**.
* des *nombres flottants* en **double précision**, pour calculer
par exemple $I = \int_{a}^{b} f(x) dx$.
-------
J'utilise une somme de Riemann avec n points :
-------
$$I\simeq h\sum_{i=0}^{n-1}f(a+ih)\;{\rm avec}\;h=\frac{b-a}{n}$$
```

Les **cellules** qui suivent sont en mode **Code**, exécutables par le noyau (*kernel*) Python 3. Elles contiennent une définition de fonction, une importation et un

12. *Markdown* a été choisi pour contrebalancer le M de HTML qui signifie *Markup*. Pour les commandes les plus utiles, consultez `https://commonmark.org/help`.

calcul. Au final, un *notebook* exécutable a été produit, exportable en PDF, HTML, LaTeX, *etc.* : outil précieux pour étudiants, enseignants et ingénieurs. Les menus *Édition*, *Affichage*, *Exécution*, *etc.* permettent de manipuler le style et l'évaluation individuelle ou collective des cellules, ou procéder à divers réglages.




Quelques mots sur le codage en ***Markdown***. En ligne 62, la balise `#` signifie `<h1>` en HTML, elle indique un **titre** sur une seule ligne. En ligne 64, un **lien** en gras (balise `**`) est situé entre crochets, l'URL étant placée à la suite entre parenthèses, c'est l'analogue du `<a href="..."><b>...</b></a>` de HTML. Une ligne vide permet de changer de paragraphe en restant dans la même cellule, c'est pourquoi il faut valider une cellule entière par *Maj-Entrée*. Enfin, la balise solitaire `*` en début de ligne indique une puce de liste non ordonnée.

Les *carnets* de Jupyter sont sophistiqués. On peut y afficher des graphiques, intégrer des cellules audio ou vidéo, et les **diffuser** grâce au **Notebook Viewer**, consultez `https://nbviewer.org/faq`. L'ensemble reste quand même assez complexe, notamment au niveau de la sécurité dans le cadre d'un travail partagé ou dans le *cloud*. Sans laisser la forme prendre le pas sur le fond, qui est pour ce livre l'apprentissage de la programmation en Python, j'ai quand même

rédigé certaines solutions à la fois sous la forme de fichiers d'extension `.py` exécutables en principe par tous, et de fichiers `.ipynb` pour être ouverts dans Jupyter ou Visual Studio Code sous la forme de *notebooks*. À vous de faire votre choix, et d'en changer le moment venu.

Une introduction à Jupyter se trouve au chap. 18 du cours pour les biologistes décrit dans [FUC] et à un niveau plus avancé sur la page de Marc Buffat à l'Université Lyon-1 qui utilise `nbgrader` permettant à l'enseignant de concevoir des *carnets* Jupyter et préparer leur évaluation avec notation automatique.




**Jupyter Lab**

Le projet Jupyter est issu de IPython, avec la volonté de se détacher de Python et d'abstraire le concept de *notebook* qui est devenu indépendant du langage de programmation. Les pédagogues intéressés par le travail collaboratif avec Jupyter jetteront un œil sur le livre **Teaching and Learning with Jupyter** [JUP] qui explore des *tactiques* d'enseignement avec les carnets. Vous pourrez enfin suivre les nouveautés sur Jupyter dans les conférences annuelles des spécialistes disponibles sur la chaîne *JupyterCon* de YouTube.

Le niveau suivant **Jupyter Lab** `https://jupyterlab.readthedocs.io` devient un véritable environnement interactif permettant d'utiliser conjointement des *carnets*, des éditeurs de texte, des Terminaux et des outils de visualisation pour ces fameuses *data* qui sont le point de mire de l'informatique d'aujourd'hui.

Afin de ne pas forcer le trait sur les IDE, je vous laisserai devenir acteur de cette aventure en explorant les ressources du Web, notamment celles des universités voire lycées qui utilisent Jupyter depuis plusieurs années. En cas de difficultés liées à l'environnement Jupyter, dirigez-vous vers `discourse.jupyter.org`.

## 1.3 L'éditeur professionnel : Visual Studio

**Visual Studio** (VS) est un IDE développé par Microsoft pour Windows ([VIS]) mais on trouve aussi sur `visualstudio.microsoft.com/fr/` une version allégée **Visual Studio Code** pour Windows, Mac et Linux. Il se trouve que **VS Code** se trouve dans les applications proposées par *Anaconda Navigator*. Je vous propose

un petit tour plus que minimal, il existe de nombreux tutoriels et vidéos de prise en main pour cet éditeur très complet. Lançons Navigator[13] et cherchons *VS Code* parmi les applications (installées ou à installer). Cliquons sur **Launch**.




Une fenêtre s'ouvre, je presse Maj-Cmd-P (sur Mac) puis j'écris *installer la commande* `code` *dans PATH* afin de pouvoir lancer VS Code à partir du Terminal.

**Edition ou exécution d'un fichier Python** `xxx.py` situé dans un dossier de programmes Python. Dans le menu *Fichier*, je clique sur *Ouvrir...* et je cherche mon fichier pour l'ouvrir. Il est alors chargé dans l'éditeur avec des lignes numérotées mais il faut **installer un interpréteur Python**, comme on le voit en bas à droite. Je peux l'exécuter dans l'environnement `(base)` par exemple. Je clique sur *Sélectionner un interpréteur* et j'opte pour *Python 3.12.9 ('base')*. Oui, chaque environnement a son propre Python, même le Python de la maison-mère *Python 3.13.0 64-bit* aurait convenu aussi. On peut lire maintenant en bas à droite *Python 3.12.9 ('base' : conda)*. J'exécute sans débogage avec le menu *Exécuter* ou Ctl-F5, et je vois les affichages dans le Terminal en bas.

> **Remarque** — L'éditeur connaît les cellules IPython, comme Jupyter. Si j'avais chargé `xxx.ipynb`, j'aurai eu droit à une présentation en cellules comme avec Jupyter, chacune pouvant être exécutée isolément avec son propre résultat juste en-dessous. Il existe un module `ipynb2py` pour obtenir un fichier Python à plat, non cellulaire, à partir d'un *notebook*.

**Installation de Copilot**. Microsoft (à-travers GitHub) livre une version de Copilot, une IA d'aide à la programmation. Dans menu *Affichage* de VS, cliquez sur *Extensions* (ou sur la 4ème icône verticale sur le bord gauche, ou Cmd-Maj-X).

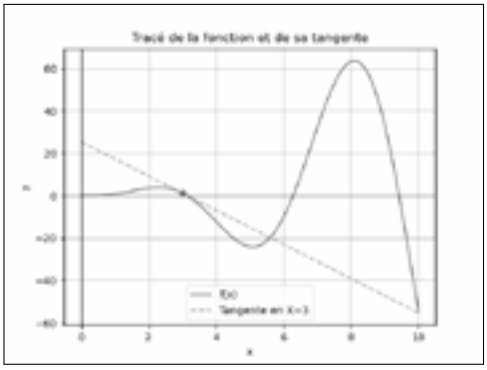
13. Si vous n'êtes pas sous Anaconda, téléchargez et lancez l'application *Visual Studio Code*.

Tapez `GitHub Copilot` puis cliquez sur *Installer*. Vous devrez avoir un compte GitHub, même si vous ne l'utilisez pas et il vous faudra peut-être accepter un accès sécurisé à ce compte GitHub, vous entrez dans un domaine professionnel.

Une fois installé, ouvrez un nouveau fichier. Au lieu d'entrer du code dans l'éditeur, tapez Cmd-I. Vous êtes alors invité à demander à Copilot de préparer votre travail. Exemple :

> **Demander à Copilot** : *Prépare-moi un code pour tracer une fonction f(x) sur un intervalle [a,b]. Tu traceras la tangente à la courbe de f au point d'abscisse X. Je définirai moi-même f,a,b,X. Le repère est orthonormé.*

Un programme Python utilisant `numpy` et `matplotlib` s'affiche alors dans l'éditeur, que je complète en choisissant la fonction $f(x) = x^2 \sin(x)$, $X = 3$ et $[a, b] = [0, 10]$. Je l'accepte puis j'exécute le fichier, et voilà. Il a oublié (heureusement) que le repère était orthonormé. C'est de la triche ? Oui, mais il faut accepter que nous sommes en 2025 et que l'IA monte, monte... Profitons-en mais restons vigilant. Parfois les codes sont un peu faux et ce peut être la galère.




Dans le menu *Affichage*, vous trouverez *Conversation* qui est une sorte de ChatGPT adapté à la programmation, ainsi que *Editions Copilot* pour lui demander de travailler sur votre code : le comprendre, l'analyser, le transformer.

## 1.4 Conclusion. Quel éditeur choisir ?

• Si vous êtes **débutant** (lycée, université, futur enseignant), je vous conseille de démarrer très simplement en installant la distribution CPython de la maison-mère (`www.python.org`) avec son éditeur IDLE et son toplevel `>>>`. Elle est solide, vous n'aurez sans doute pas besoin d'environnements et s'il vous faut installer des modules, vous le ferez avec `pip` (ou mieux : `pip3`).

• Si vous êtes en période **intermédiaire**, disons que vous avez déjà un peu touché à la programmation, vous savez ce qu'est un Terminal et sa ligne de commande,