Jean-Paul Roy

Python

Apprentissage actif

pour l'étudiant et le futur enseignant



ellipses

Références sciences

Python
Apprentissage actif

pour l'étudiant et le futur enseignant

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 9782340-036642
©Ellipses Édition Marketing S.A., 2020
32, rue Bargue 75740 Paris cedex 15



Le Code de la propriété intellectuelle n'autorisant, aux termes de l'article L. 122-5.2° et 3°a), 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 » (art. L. 122-4).
Cette représentation ou reproduction, par quelque procédé que ce soit constituerait 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

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. Mais nous rencontrerons aussi Jython qui est basé sur Java, ainsi que le compilateur Cython pour les programmeurs C.

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)`
`  (print x)`
`  (print y))`
	`if x > 0:`
	`  print(x)`
	`else:`
	`  print(y)`
		`if (x > 0) {`
		`  System.out.println(x);`
		`} else {`
		`  System.out.println(y);`
		`}`

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 elle contient le langage Python mais 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.

Après quelques hésitations, il m'a paru intéressant de présenter aussi le système *Processing*, destiné aux artistes numériques et organisé autour du langage Java. Depuis peu, il existe un mode Python à l'intérieur de *Processing*, ce dernier prenant alors le nom de **Processing.py**. Nous aurons l'occasion d'y faire bien entendu du graphisme, mais aussi d'entr'ouvrir la porte de la musique électronique (avec la librairie *Beads* de Java). Nous verrons à cette occasion une autre implémentation de Python qui se nomme Jython et qui permet l'accès en Python 2 à des modules programmés en Java. Hélas, à ce jour nous serons en effet quelques rares fois obligés d'utiliser du Python dans son ancienne version 2 et devrons nous adapter. Je n'ai pas voulu masquer la réalité des faits, qui sont têtus car il peut s'avérer nécessaire d'utiliser d'autres langages voire d'autres versions de Python, cela fait partie de la vie du programmeur. Nous pénétrerons aussi dans les grosses bibliothèques mathématiques `numpy` et `sympy` essentielles au scientifique travaillant en Python. Nous n'y passerons pas un temps démesuré car il existe des ouvrages spécialisés pour un travail plus 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.

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`

Qui dit programmation dit algorithmes, mais ce livre traite de la pratique du codage. Ce n'est ni un cours ni un ouvrage d'algorithmique qui explorerait toutes les structures de données. Il vous faudrait pour ces dernières par exemple l'indispensable [COR] [7] qui fait à juste titre abstraction du langage de programmation, ou des ouvrages comme [DUR] s'appuyant explicitement sur Python.

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

La seconde partie consiste en une sélection de petits exercices ou problèmes 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. Si vous résolvez vite tous les problèmes en tapant sur le clavier avec le seul petit doigt de la main gauche, ce livre vous aura été sans doute inutile.

Étudiant en Faculté des Sciences, en Ecole d'Ingénieurs, en IUT, peut-être futur enseignant des lycées préparant le CAPES d'Informatique ou déjà chargé de l'option informatique et utilisant sans doute Python, 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 simplement un amateur de programmation, sachez que le contenu utilise un peu de 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] ou [LAN], ou vraiment axés sur les mathématiques : [ZIM] ou [BAR] avec le logiciel SAGE souplement basé sur Python.

La bibliographie contient aussi des références en anglais lorsque je n'ai pas trouvé de livres équivalents en français. J'en suis désolé mais à ma décharge, la pratique de l'anglais est devenue indispensable dans toute activité scientifique.

Remerciements

Cet ouvrage est en majeure partie issu de cours donnés en Licence au Département Informatique de l'UFR Sciences de l'Université Nice Sophia-Antipolis. Je remercie les nombreux enseignants et moniteurs ayant collaboré à ces enseignements et qui m'ont fait des retours instructifs. Merci aussi à celles et ceux qui ont relu certaines parties de ce livre. S'il reste des erreurs, il va de soi que j'en suis l'unique responsable.

Table des matières

I L'essentiel du langage **1**

1 Environnement de programmation **3**
1.1 Installation de la distribution Anaconda . . . 3
1.2 L'éditeur IDLE et sa console . . . 4
1.3 Le gestionnaire de paquets `conda` . . . 5
1.4 L'interface graphique : Anaconda Navigator . . . 9
1.5 Un autre IDE pour Anaconda : Spyder . . . 10
1.5.1 L'éditeur de Spyder . . . 11
1.5.2 La console de Spyder . . . 11
1.6 Le projet Jupyter . . . 12
1.6.1 Les *notebooks* de Jupyter . . . 13
1.6.2 Jupyter Lab . . . 15
1.7 Conclusion . . . 15

2 Variables, nombres et fonctions **17**
2.1 Qu'est-ce qu'une variable ? . . . 17
2.2 Qu'est-ce qu'un type ? . . . 18
2.3 Le type `int` des nombres entiers . . . 20
2.4 Les nombres flottants . . . 20
2.5 Les nombres complexes . . . 22
2.6 Qu'est-ce qu'un module en Python ? . . . 22
2.7 Donc pas de nombres rationnels en Python ? . . . 24
2.8 L'égalité est-elle fiable sur les nombres flottants ? . . . 24
2.9 L'infini existe-t-il en programmation ? . . . 25
2.10 Qu'est-ce que la priorité d'un opérateur ? . . . 25
2.11 Comment construire couples, triplets, *etc.* ? . . . 26
2.12 Quelle différence entre expression et instruction ? . . . 27
2.13 Les expressions booléennes . . . 28

2.14 Les instructions conditionnelles . . . 30
2.15 Comment tirer un nombre au hasard ? . . . 32
2.16 Fonction, paramètres et variables locales . . . 32
2.17 Qu'est-ce qu'une variable globale ? . . . 34
2.18 Les différentes formes de paramètres . . . 36
2.19 Les arguments optionnels de la fonction `print` . . . 37
2.20 Les lambda-expressions . . . 38
2.21 Comment documenter une fonction ? . . . 40
2.22 Quelle est la différence entre fonction et méthode ? . . . 41
2.23 Qu'est-ce qu'une fonction récursive ? . . . 41
2.23.1 Suivre une fonction à la trace . . . 43
2.23.2 Le rôle de la pile de récursion . . . 44
2.24 La boucle `while` . . . 47
2.25 La boucle `for` . . . 49
2.26 Comment interrompre brutalement une boucle ? . . . 51
2.27 Le calcul d'une somme en compréhension . . . 51
2.28 La complexité d'un calcul . . . 52

3 Traitement du texte : les bases 55
3.1 Les caractères Unicode . . . 55
3.2 Les textes sont des chaînes de caractères . . . 56
3.3 Comment programmer avec des chaînes ? . . . 59
3.4 Les chaînes formatées . . . 60
3.5 Une chaîne peut-elle contenir du code Python ? . . . 61

4 Dessiner 65
4.1 Le graphisme de la tortue . . . 65
4.1.1 La fenêtre graphique . . . 65
4.1.2 Le graphisme cartésien . . . 66
4.1.3 Le graphisme polaire . . . 69
4.1.4 Une courbe fractale . . . 70
4.2 Le système *Processing.py* . . . 72
4.3 Stéganographie avec le module *Pillow* . . . 75

5 Les collections d'objets 79
5.1 Collections et séquences . . . 79
5.2 Les listes sont des séquences mutables . . . 79
5.3 Les constructions fonctionnelles de listes . . . 81
5.4 Déstructuration d'un objet itérable . . . 83
5.5 Suppression d'objets avec `del` . . . 84
5.6 Listes et partage de mémoire . . . 84

5.7 Les ensembles 88
5.8 Les dictionnaires 90
5.9 Gestion des erreurs : les exceptions 93

6 Les entrées-sorties 97
6.1 Le système de fichiers et le module `os` 97
6.2 Ecriture dans un fichier texte sur disque (en sortie) 99
6.3 Lecture d'un fichier texte sur disque (en entrée) 100
6.3.1 La construction `with` 102
6.3.2 L'utilisation de `split` 103
6.4 Des fichiers en mémoire centrale 103
6.5 Exécution d'un script Python au terminal 104

7 Les objets et les classes 107
7.1 Définition d'une nouvelle classe d'objets 107
7.2 Les attributs de classe 112
7.3 Utilisation des méthodes spéciales 114
7.4 Boucle `for` et méthodes spéciales : les itérateurs 114
7.5 Les générateurs 116
7.6 Les objets *soft* 117
7.7 La hiérarchie des classes 118
7.8 Les classes abstraites 121
7.9 Qu'est-ce qu'un décorateur ? 122

8 Traitement avancé du texte 125
8.1 Expressions régulières : le module `re` 125
8.1.1 Présence d'une sous-chaîne dans une chaîne 126
8.1.2 Les méta-caractères d'un motif 127
8.1.3 Remplacements dans une chaîne 130
8.2 Analyse lexicale avec l'outil `lex` 131
8.2.1 Installation avec `pip` 131
8.2.2 Programmation d'un analyseur lexical 132
8.3 Analyse syntaxique avec l'outil `yacc` 134
8.4 Recherche de texte dans une page Web : `requests` 138

9 Les interfaces graphiques (GUI) 139
9.1 Le module `tkinter` 139
9.1.1 La programmation dirigée par les évènements 139
9.1.2 La fenêtre du programme 140
9.1.3 Dessiner avec les classes `Canvas` et `Button` 141
9.1.4 L'interface graphique comme sous-classe de Tk 144

9.1.5 Les options du gestionnaire `pack` . . . 145
9.1.6 Gestion des évènements souris et clavier . . . 145
9.1.7 Variables de contrôle . . . 148
9.1.8 Compléments sur les évènements souris . . . 150
9.1.9 Animation en style MVC avec `tkinter` . . . 152
9.1.10 La tortue dans une interface graphique . . . 155
9.2 Programmer une animation avec `pyglet` . . . 156

10 De Java vers Python **159**
10.1 Le typage : fort ou dynamique ? . . . 159
10.2 Les boucles . . . 161
10.3 Les listes . . . 162
10.4 Les classes d'objets . . . 163

11 EXERCICES **167**

Nombres, fonctions, boucles
11.1 Valeur d'une expression . . . 167
11.2 Quotient et reste. Écriture binaire d'un entier . . . 167
11.3 Écriture binaire d'un nombre flottant . . . 167
11.4 Avec ou sans résultat ? . . . 168
11.5 Heures – minutes – secondes . . . 169
11.6 Décomposition en sous-fonctions . . . 169
11.7 Une fonction, comment ça marche ? . . . 169
11.8 Addition relativiste des vitesses . . . 170
11.9 La formule de Stirling . . . 170
11.10 Payez vos impôts ! . . . 170
11.11 Une variable locale dans une lambda ? . . . 171
11.12 Tirage au hasard de nombres entiers . . . 171
11.13 Nombres fractionnaires (rationnels) . . . 172
11.14 Solutions complexes d'une équation de degré 2 . . . 172
11.15 Nombres complexes et géométrie plane . . . 172
11.16 La division euclidienne dans les entiers . . . 173
11.17 Travail arithmétique sur les chiffres d'un entier . . . 173
11.18 Boucles `for` emboîtées . . . 174
11.19 Calcul de x^n par dichotomie . . . 174
11.20 Tester si un nombre entier est premier (naïf) . . . 174
11.21 Nombres premiers entre eux. Le PGCD . . . 175
11.22 Calcul de a^b modulo n dans les grands entiers . . . 176
11.23 Tester si un grand nombre est premier (Fermat) . . . 176
11.24 Calcul des nombres de Fibonacci . . . 177
11.25 Vérification qu'un dé est correctement truqué . . . 178

11.26 Dérivation numérique approchée 179
11.27 Approximation numérique d'un sinus 180
11.28 Valeur approchée d'une intégrale simple 180
11.29 Coût du (temps de) calcul de n ! 182
11.30 Rectification d'un arc paramétré 183
11.31 Valeur approchée d'une intégrale double 184
11.32 Centre de gravité d'une figure plane 185
11.33 Solution approchée d'une équation (dichotomie) 185
11.34 Solution approchée d'une équation (Newton) 186
11.35 Régime d'un système dépendant du temps 186
11.36 Cryptographie à clé publique (RSA/1) 187

Chaînes de caractères

11.37 Affichage d'une table de multiplication 188
11.38 Construction de l'alphabet minuscule 188
11.39 Les plus longs mots d'une phrase 188
11.40 Travail textuel sur les chiffres d'un entier 189
11.41 Récurrence sur des chaînes 189
11.42 Le codage secret selon César 190
11.43 Les palindromes 190
11.44 Le nombre de Champernowne 191
11.45 Vérification d'un code bancaire IBAN 191
11.46 Les règles de réécriture 192
11.47 L'addition binaire avec retenue 192
11.48 Analyse lexicale *à la main* 193
11.49 HTML et Unicode 194

Graphisme de la tortue, Processing

11.50 Graphismes polaire et cartésien 194
11.51 Tracé d'un cercle avec centre et rayon 195
11.52 Mangez des oursins ! 195
11.53 Un traceur minimal de courbes paramétrées 195
11.54 Dessin d'une courbe en coordonnées polaires 196
11.55 Le jeu du chaos 197
11.56 Une courbe fractale du dragon 197
11.57 La tortue, le hasard et les flaques 198
11.58 Génération d'une fractale par un L-système 199
11.59 Graphisme avec *Processing.py* 200

Les séquences

11.60 Tirage d'un élément aléatoire dans une séquence 202
11.61 Courbe d'un système dépendant du temps 203

11.62 Inversion des mots d'une phrase . . . 203
11.63 Recherche séquentielle dans une liste non triée . . . 203
11.64 Comptage d'éléments dans une collection . . . 204
11.65 Extraction d'éléments dans une séquence . . . 204
11.66 Recherche dichotomique dans une liste triée . . . 205
11.67 Compactage d'une liste . . . 205
11.68 Le drapeau hollandais de Dijkstra . . . 206
11.69 Les fonctionnelles `map` et `reduce` . . . 207
11.70 Vérification des éléments d'une collection . . . 208
11.71 Attention aux mutations dans les listes ! . . . 208
11.72 Le triangle de Pascal . . . 209
11.73 Tri stupide d'une liste . . . 209
11.74 Tri d'une liste sur place, par sélection . . . 211
11.75 Chronométrage d'un algorithme sur une longue liste aléatoire . . . 211
11.76 Tri par insertion : récurrence et itération . . . 212
11.77 Tri d'une liste avec copie, par fusion . . . 213
11.78 Tri rapide . . . 214
11.79 Trier sur des critères exotiques . . . 215
11.80 Crible d'Ératosthène pour les nombres premiers . . . 215
11.81 Cryptographie à clé publique (RSA/2) . . . 216
11.82 Unicode et les bizarreries typographiques . . . 217
11.83 Opérations sur des matrices . . . 218
11.84 Construction d'un carré magique d'ordre impair . . . 219
11.85 Message caché dans une image . . . 220
11.86 Automates cellulaires 1-D . . . 220
11.87 Les piles de données . . . 222
11.88 Retour sur la pile de récursion . . . 223
11.89 Polynômes creux . . . 225
11.90 Polynômes de Legendre . . . 227
11.91 Polynôme d'Interpolation de Lagrange . . . 228
11.92 Racines simples d'un polynôme (1) . . . 229
11.93 Racines simples d'un polynôme (2) . . . 231

Itérateurs, ensembles et dictionnaires

11.94 Fermeture-éclair : l'itérateur `zip` . . . 232
11.95 Générateur des éléments distincts d'une liste . . . 233
11.96 Réunion et intersection de listes . . . 233
11.97 Produit cartésien d'ensembles . . . 234
11.98 Combinaisons dans un ensemble . . . 234
11.99 Dictionnaires et fréquences . . . 235
11.100 Mémorisation : remplacer le temps par l'espace . . . 236

11.101 Automate cellulaire 2-D : le Jeu de la Vie 237
11.102 Arbres binaires d'expression 239
11.103 La machine à pile VRISC1 242
11.104 La machine VRISC2 : les sauts à étiquette 243
11.105 Parcours itératif d'un arbre binaire 245
11.106 Introduction à la simplification symbolique 246
11.107 L'algorithme de dérivation symbolique 248
11.108 Calcul numérique avec `numpy` 250
11.109 Ajustement numérique d'une courbe avec `scipy` 253

Expressions régulières

11.110 Quelques expressions régulières 255
11.111 Mais tout n'est pas régulier ! 256
11.112 Fin de l'analyseur lexical du § 8.2 avec *lex* 257
11.113 Fin de l'analyseur syntaxique du § 8.3 avec *yacc* 258
11.114 Une calculette interactive avec *lex & yacc* 259
11.114.1 L'analyseur lexical (*lexer*) 260
11.114.2 L'analyseur syntaxique (*parser*) 261
11.114.3 L'évaluateur 262
11.115 Retour sur les *cpbb* à dos de *yacc* 264

Fichiers, Web

11.116 Écriture dans un fichier texte sur disque 264
11.117 Lecture d'un fichier texte sur disque 265
11.118 Lecture d'un fichier au format CSV 265
11.119 Transformation d'un fichier texte sur disque 266
11.120 Construction d'un graphique Excel en Python 266
11.121 Trouver une heure locale sur Internet 268
11.122 Échange de données au format JSON 269
11.123 Sérialisation des données : `json` ou `pickle` ? 270
11.124 Lancer un programme à heure donnée : `cron` 271
11.125 Dissection d'une page `HTML` avec `Beautiful Soup` 272

Objets

11.126 Classe des matrices $n \times p$ sans `numpy` 276
11.127 Naissance d'une tortue avec *Processing.py* 277
11.128 Incursion dans le graphisme 3D 281
11.129 Ajustement par programmation génétique 283
11.130 Introduction au moteur physique `pymunk` 286
11.131 Moteur physique appliqué : pendule de Newton 292
11.132 Décoration de la récurrence terminale 294
11.133 Neurones : le perceptron 295

11.134 Neurones : la rétro-propagation (théorie) 298
11.135 Neurones : la rétro-propagation (pratique) 301
11.136 Programmation audio (théorie) 304
11.137 Programmation audio avec `Beads` (pratique) 308

Bibliographie **313**

Index **317**

Première partie

L'essentiel du langage

1 Environnement de programmation

1.1 Installation de la distribution Anaconda

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, sous Spyder, sous Jupyter, voire Emacs, Eclipse, PyCharm, XCode ou Visual Studio, que sais-je ? Il était illusoire de les passer tous en revue. Je n'ai retenu que trois éditeurs de programmes : **IDLE**, **Spyder** et **Jupyter**, tous immédiatement disponibles sous Anaconda. Nous verrons comment exécuter avec eux un programme.

La maison-mère `www.python.org` livre le langage Python avec un petit éditeur intégré IDLE, fournissant le minimum requis pour programmer en Python. Mais 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** contenant le langage Python (version au moins 3.7), de très nombreuses bibliothèques scientifiques (`numpy`, *etc.*), une application *Navigator*, et le gestionnaire `conda` pour les mises à jour.
`https://www.anaconda.com/distribution/`

— soit une version réduite **Miniconda** ne contenant que Python et le gestionnaire `conda` pour installer les paquets qui vous intéressent.
`https://docs.conda.io/en/latest/miniconda.html`

Téléchargez et installez la version complète *Anaconda* si vous avez suffisamment d'espace disque, cela vous simplifiera la vie.

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

1.2 L'éditeur IDLE et sa console

Installation faite, un nouveau répertoire `anaconda3` (ou `miniconda3`) vient d'être créé à la racine de votre compte utilisateur, et son répertoire `bin` contient entre autres un IDE nommé **IDLE** tout juste installé avec une version 3.x du Python de la maison-mère. Vous pouvez lancer IDLE au terminal de votre machine (le terminal Unix pour Linux et MacOS, et le PowerShell pour Windows [2]) avec la commande `idle3`. Je précise `idle3` car sur certaines machines vous pourriez avoir un Python 2 déjà installé. Placez au besoin un raccourci vers IDLE sur le bureau.

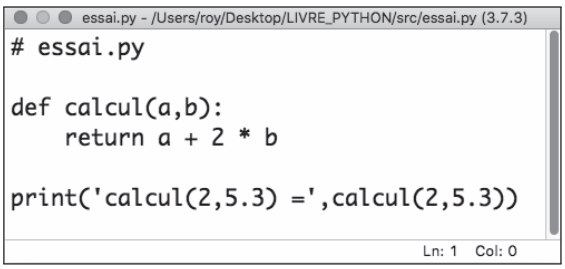
Cette application IDLE ouverte, vous êtes en présence d'une fenêtre **console** ou *shell*, une sorte de calculette Python interactive.

```
1  Python 3.7.3 (Mar 27 2019)
2  >>> 2 + 3 * 4
3  14
4  >>> from math import sqrt                    # la racine carrée
5  >>> sqrt(2) ** 2                            # un calcul approché !
6  2.0000000000000004
```

Une aide en ligne sommaire est disponible à la console avec la fonction `help`.

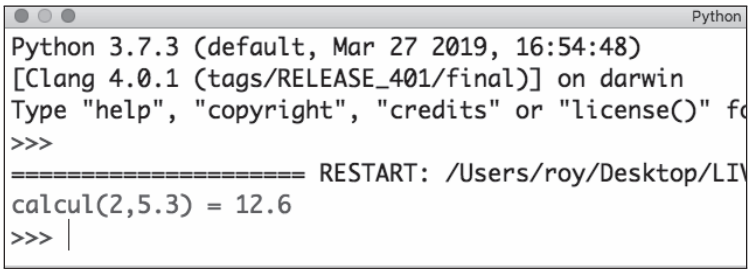
```
7  >>> help(sqrt)                               # aide en ligne
8  Help on built-in function sqrt in module math:
9  sqrt(x, /)
10     Return the square root of x.
```

En général, vous rédigerez un programme avec l'**éditeur de texte** de IDLE, 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 $a + 2b$.



2. Sous Windows, vous pouvez aussi utiliser *Anaconda Prompt* que vous trouverez dans le menu *Démarrer*.

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 sous le nom choisi `essai.py`. Ce contenu sera ensuite analysé sommairement pour détecter d'éventuelles fautes d'indentation, la mémoire sera remise à neuf (*Restart*) et enfin le contenu sera évalué de haut en bas. Son évaluation produira au passage des affichages de résultats à la console :



Les tests du programme se font donc plutôt dans l'éditeur qu'à la console, de manière à ce qu'ils soient exécutés automatiquement sans avoir à les retaper à chaque modification du programme. Nous verrons plus tard au § 6.5 comment rendre un programme exécutable au terminal en-dehors d'un IDE.

Cet environnement intégré minimal est rudimentaire mais plus que suffisant pour une première initiation à Python (par exemple pour tous les programmes de ce livre). Il ne convient pas à un développement avancé ou professionnel.

1.3 Le gestionnaire de paquets `conda`

Vous serez amenés un jour ou l'autre à devoir utiliser une bibliothèque logicielle extérieure écrite en Python, on parle de **paquet** (en anglais *package*). Pour l'installer sous Anaconda, nous utiliserons au terminal un **gestionnaire de paquets et d'environnements** (*environment and package manager*) nommé `conda`[3]. Il existe un autre gestionnaire de paquets `pip` et d'environnements `virtualenv`, mais nous veillerons à nous en tenir à `conda` pour éviter les soucis ! Pour savoir si `conda` est bien installé, demandons son numéro de version. Il est recommandé de tenir `conda` à jour (*update*). Nous passons de la version 4.5.12 à la version 4.6.8.

```
11 (base) $ conda --version
12 conda 4.5.12
13 (base) $ conda update conda
14 ...conda: 4.5.12-py37_0 --> 4.6.8-py37_0...
```

3. 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.

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 267).

```
(base) $ conda list
# packages in environment at ~/anaconda3:
# Name                    Version                   Build  Channel
_ipyw_jlab_nb_ext_conf    0.1.0                     py37_0
alabaster                 0.7.12                    py37_0
anaconda                  2018.12                   py37_0
anaconda-client           1.7.2                     py37_0
anaconda-navigator        1.9.6                     py37_0
...
matplotlib                3.0.2                     py37h54f8f79_0
numpy                     1.15.4                    py37hacdab7b_0
...
```

Vous pouvez travailler dans l'environnement `(base)` pour du Python pur par exemple, mais si vous commencer à importer des paquets [4] extérieurs, il se peut que tel paquet A 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 (nous le ferons plus tard) que nous souhaitions programmer une simulation graphique avec la bibliothèque `Pyglet` et utiliser éventuellement 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 ces deux paquets (qui ne font pas partie des 267 paquets initiaux de notre Anaconda).

```
(base) $ conda create --name simul
... environment location: /Users/roy/anaconda3/envs/simul
```

Vous pouvez abréger `--name` en `-n`. Les environnements se trouvent dans votre 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`.

```
(base) $ conda activate simul
(simul) $ conda list
(simul) $
```

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

Maintenant que nous sommes dans `simul`, installons le logiciel `Pyglet`. Je trouve sur le Web une page `https://pypi.org/project/pyglet/` qui me propose de l'installer avec le gestionnaire `pip` peu apprécié d'Anaconda. Résistez à la tentation. Nous ferons notre possible pour l'installer avec `conda install` et seulement en désespoir de cause aurons-nous recours à `pip` (qui fait partie des paquets d'Anaconda [5]). La commande usuelle d'installation d'un paquet dans `simul` est :

```
(simul) $ conda install pyglet
The following packages are not available from current channels:
  - pyglet
```

Oups. Anaconda utilise des **canaux de recherche** (*channels*) par défaut, et aucun d'eux ne trouve `pyglet`. Dans ce cas, retour sur le Web en demandant « install pyglet conda » au moteur de recherche, et en croisant les doigts. Je trouve vite la page `https://anaconda.org/conda-forge/pyglet` qui m'invite à utiliser le canal `conda-forge`. Dont acte. À nous de le préciser à `conda install`.

```
(simul) $ conda install --channel conda-forge pyglet
...
Executing transaction: done
(simul) $ conda list
...
pyglet                1.3.2              py37_1000     conda-forge
python                3.7.3              h0d93f26_0    conda-forge
...
```

La dernière commande montre que `simul` contient maintenant 18 paquets (dont les **dépendances**), avec un Pyglet 1.3.2 et même un Python 3.7.3 puisqu'il n'y en avait pas, notre Python étant jusqu'à présent dans l'environnement `base` seulement. Vérifions que tout fonctionne bien (`import` ne doit pas provoquer d'erreur).

```
(simul) $ python3
Python 3.7.3 | packaged by conda-forge | (default Mar 27 2019, 15:43:19)
>>> import pyglet
>>> quit()
(simul) $
```

Je vous laisse vous entraîner à installer le paquet `pymunk` dans `simul`. Nous aurons besoin de `pyglet` et `pymunk` dans certains exercices.

5. Les gestionnaires `pip` et `conda` sont parfois en conflit, à cause par exemple du paquet `qt` assez utilisé dans le monde scientifique.

Ce dépôt `conda-forge` semble intéressant, ajoutons-le en queue (*append*) à la liste des canaux utilisés lors d'une installation, et vérifions la liste des canaux.

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

Une autre manière de le vérifier consiste à lire le contenu du fichier caché `.condarc` contenant des options de configuration de `conda`. Les canaux de `defaults` sont ceux hébergés par la société Anaconda, et `conda-forge` est une communauté libre sur GitHub. Nous n'aurons pas besoin d'autres canaux dans ce livre.

```
(simul) $ cat ~/.condarc       # à la racine de mon répertoire utilisateur
channels:
  - defaults
  - conda-forge
```

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

Enfin, il faut savoir **mettre à jour** vos paquets avec `conda update`. Les paquets dépendants seront calculés et aussi mis à jour dans l'environnement courant. Par exemple, ma distribution Anaconda arrive avec un Python 3.7.1 dans `base` alors que Pyglet m'a installé une version 3.7.3 dans `simul`. Je tente une mise à jour en sortant de `simul` pour revenir dans `base` avec `conda deactivate`. Notez ci-dessous que la mise à jour est *mineure*, refusée par `update` et donc à installer en plus. Sur une mise à jour vers la version 3.8, `update` aurait joué son rôle.

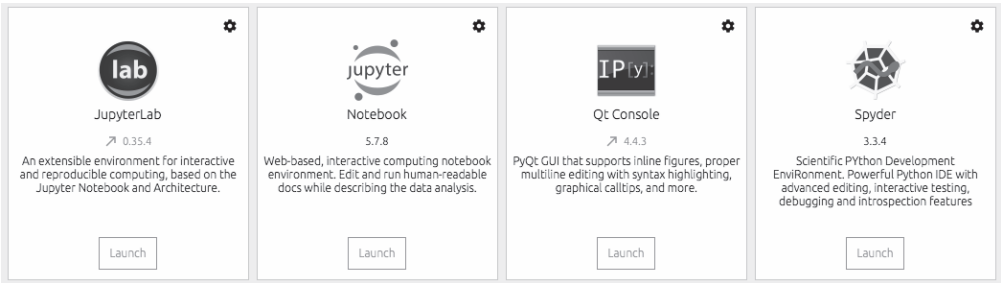
```
(simul) $ conda deactivate            # retour à l'environnement de base
(base) $ conda update python
All requested packages already installed.
(base) $ conda install python==3.7.3             # je force la version !
...
Executing transaction: done
(base) $ python3
Python 3.7.3 (default, Mar 27 2019, 16:54:48)
>>> import pyglet
ModuleNotFoundError: No module named 'pyglet'   # pas dans base !
>>> quit()
(base) $
```

Oui, il faut mémoriser quelques commandes au terminal, mais on s'y fait vite. Ou pas, et dans ce cas mieux vaut lancer l'application *Anaconda Navigator*. Pour vous convaincre de fonctionner plutôt au terminal, un conseil : téléchargez la dernière version de la « **conda cheat sheet** » avec votre navigateur et la vie pétillera ☺.

1.4 L'interface graphique : Anaconda Navigator

Le répertoire `anaconda3` contient l'application **Anaconda Navigator**. Sous Windows, vous la trouverez aussi dans le menu *Démarrer*. Si vous avez téléchargé *Miniconda*, vous pouvez utiliser `conda` pour installer ou mettre à jour le paquet `anaconda-navigator`. L'application *Navigator* peut aussi être lancée au terminal avec la commande `anaconda-navigator`.

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



Parmi eux se trouvent **Jupyter Notebook** et **Spyder** qui vous serviront à rédiger les programmes si vous n'avez pas opté pour IDLE. Nous en parlerons plus bas. Disons juste que 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 les deux environnements créés au § 1.3 : `base` et `simul` avec au-dessous quatre boutons : pour créer un nouvel environnement, ou en cloner, importer [6] 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 précédente pour bien saisir le sens des manipulations, et nous vous renvoyons aux tutoriels en ligne :

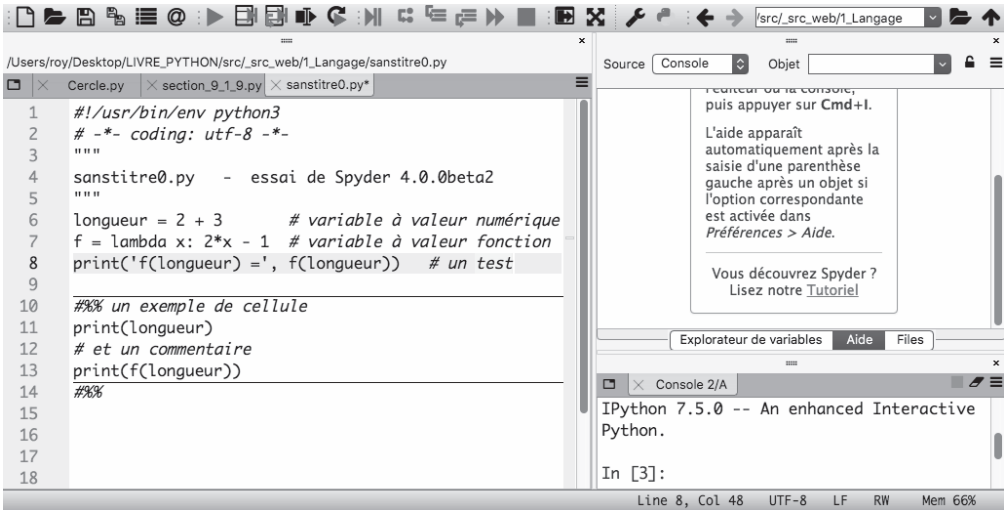
`https://docs.anaconda.com/anaconda/navigator/tutorials/`

notamment aux chapitres *Managing environments*, *packages* et *channels*. Vous trouverez aussi des vidéos sur YouTube.

6. Vous pouvez transmettre à autrui une description d'environnement sous la forme d'un fichier `.yml` pour qu'il soit dans la même situation que vous.

1.5 Un autre IDE pour Anaconda : Spyder

Plus sophistiqué que IDLE, il est plus délicat à régler mais une fois stabilisé, il est assez agréable. Avec Navigator, vous pouvez à la fois le lancer (bouton *launch*) et le mettre à jour grâce à l'icône en forme de roue dentée. Mais vous pouvez aussi le lancer au terminal avec la commande `spyder`, ci-dessous en version 4.0.0β2. Une grande fenêtre s'ouvre, contenant trois sous-fenêtres.



• À gauche, l'**éditeur de texte** proprement dit, pour rédiger nos programmes.
• À droite en haut, une fenêtre d'**aide** avec trois onglets à sa base : *Explorateur de variables*, *Aide* et *Explorateur de fichiers*.
• À droite en bas, une **console** (parfois nommée *toplevel*), ici dans sa version **IPython** pour *Interactive Python*, reconnaissable à ses `In:` et `Out:`. Analogue au *Shell* de IDLE, elle permet de voir les affichages (textuels ou parfois graphiques) d'un programme. Nous sommes dans une console IPython 7.5.0.

Remarque — Si Spyder est en anglais, ouvrez le menu *Tools/Preferences* puis *Advanced Settings*, et choisissez la langue : Français.

Chaque sous-fenêtre est détachable : en cliquant sur le menu porté par l'icône *Options* ≡ sur la droite de sa barre de titre, vous pourrez la détacher (*undock*) de la fenêtre mère de Spyder et en faire une fenêtre à part entière. Vous pourrez inversement la ré-intégrer dans la fenêtre mère et la placer où vous voudrez. Au cas où vous auriez perdu l'une de ces fenêtres (!), allez dans le menu **Affichage/Volets** pour l'afficher à nouveau. Si votre écran n'est pas de grande taille, conservez seulement l'éditeur et la console. En déplaçant la barre qui les sépare, vous pourrez modifier leur taille respective. Conseil à deux sous : achetez un *petit* ordinateur portable avec un bel écran séparé.

1.5.1 L'éditeur de Spyder

Dans le menu **Aide**, vous trouverez un tutoriel [7], ainsi qu'un rappel des raccourcis clavier (en anglais *shortcuts*). Pour un travail minimal, il vous suffit de savoir ouvrir et enregistrer un programme (menu **Fichier**), et surtout exécuter un programme (avec le menu **Exécution**, ou son équivalent clavier **F5**, ou encore à la souris avec le bouton vert ▶ dans la barre horizontale d'icônes).



Si vous passez la souris sur chaque icône, vous aurez une info-bulle sur son rôle. Explorez-les, attendez un peu d'avoir les boucles pour activer le *débogueur* et sa série d'icônes bleues (ou par le menu **Déboguer**) qui vous permettra de mettre au point (corriger) un programme en suivant pas à pas son exécution ou en y plaçant un point d'arrêt temporaire (*breakpoint*).

Fixez votre **répertoire de travail** (*working directory*) avec l'icône en haut à droite. Ce répertoire s'affichera sur la gauche de cette icône. Le menu **Affichage**/**Volets** vous offre un explorateur de fichier (*file explorer*) associé au répertoire de travail.

L'info-bulle de l'icône à droite de l'icône ▶ indique *Exécuter la cellule courante*. Une « **cellule** » [8] (en anglais *cell*) permet de regrouper plusieurs lignes, du code et des commentaires, pour former un bloc exécutable seul. Une cellule débute par `#%%` et termine par le même `#%%` seul sur une ligne. Placez le curseur dans une ligne quelconque de la cellule et celle-ci sera exécutée à la console par . Les **notebooks** de *Jupyter* (§ 1.6) sont une extension du concept de cellule de Spyder et permettent de rédiger des documents scientifiques exécutables.

Enfin, l'icône **Options** en forme de roue dentée située à droite juste en-dessous de la barre de titre d'une fenêtre permet quelques actions fréquentes. Par exemple, dans l'éditeur, aller à un numéro de ligne ou séparer la fenêtre en deux horizontalement ou verticalement, ce qui est souvent fort pratique. Explorez toutes ces options. Le temps passé dans le tutoriel intégré de Spyder (menu *Aide*) ou dans ceux de YouTube ne sera certainement pas du temps perdu !

1.5.2 La console de Spyder

La **console**, contrairement au *shell* de IDLE, est basée sur un **IPython** simplifié [9]. Il s'agit d'un lieu interactif où l'on parle plusieurs langages mais très majoritairement Python, et qui sera utilisé simplement comme une calculette évoluée, ou

7. Plus d'informations sur `https://www.spyder-ide.org` accessible par pression de **F1**.

8. On retrouve le concept de cellule popularisé par *Mathematica* de Wolfram Research.

9. Le site de IPython est `http://ipython.org`, mais la console de Spyder est analogue à l'application indépendante **QtConsole** d'Anaconda. Voir [ROSS] ou [VAN] pour l'utilisation avancée de IPython et Jupyter en science des données.

bien pour visualiser les affichages d'un programme lors de son exécution. Celle-ci s'obtient par pression de l'icône *Exécuter* (**F5**) dans l'éditeur, qui déclenche à la console un appel à la commande `runfile` de Spyder (`%run` en IPython).

En plus de Python, la console reconnaît aussi des **commandes magiques** placées en début de ligne et préfixées par un `%`, comme `%timeit` pour chronométrer un calcul. Parmi ces commandes magiques, on trouve aussi une partie du vocabulaire Unix pour accéder au système de fichiers, à l'instar de `%ls` qui affiche la liste des fichiers du répertoire courant, ou bien `%cat` pour afficher le contenu d'un fichier, `%cd` pour changer de répertoire courant, *etc.*

La commande `%clear` permet d'effacer l'écran de la console, et `%reset` de supprimer toutes vos variables Python définies à la console, ce qui peut se faire plus sûrement avec le menu **Options** ≡ de la fenêtre console, en *redémarrant le noyau* (analogue du *Restart Shell* de IDLE). Pour avoir de l'aide en ligne sur la commande magique `%timeit`, demandez `%timeit?` à la console. Idem avec `print?` pour une fonction Python, mais cette fois la fenêtre d'aide sera plus agréable : placez le curseur dans le mot `print` et tapez Ctrl-I ou Cmd-I.

La liste des commandes magiques s'obtient avec `%magic`, ou `%lsmagic` en version courte. Il est à noter que la console n'est pas en mode auto-magique au lancement, le préfixe `%` est obligatoire pour les commandes magiques. Pour autoriser la commande `cd` à la console au lieu de `%cd`, demandez `%automagic ON`.

Vous pouvez sauvegarder l'**historique** de la console au format texte ou HTML avec un clic droit dans la console. Vous pouvez faire référence à l'entrée `In[14]` avec `_i14` ou l'éditer pour la reprendre avec `%rep 14`. Les trois dernières entrées sont accessibles avec `_i`, `_ii` et `_iii`. Les entrées (resp. sorties) sont maintenues dans une liste `In` (resp. un dictionnaire `Out`).

```
In[1] : x = 2
In[2] : x + 1
Out[2]: 3
In[3] : 2 * x
Out[3]: 4
```

```
In[4] : (_i1, _i2, In[1], In[2])
Out[4]: ('x = 2', 'x + 1', 'x = 2', 'x + 1')
In[5] : Out[3] + 1       # Out[1] n'existe pas
Out[5]: 5
```

1.6 Le projet Jupyter

Le projet **Jupyter** (`jupyter.org`) est un successeur de IPython visant à faciliter la rédaction de documents électroniques ou *notebooks* contenant à la fois du texte bien mis en forme (avec *Markdown* et LaTeX) et du code Python exécutable [10]. Un

10. 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.

notebook Python (carnet de notes) est un fichier d'extension `.ipynb`[11] organisé en **cellules** plus puissantes que celles de l'éditeur Spyder. Vous pourrez ainsi rédiger un véritable texte scientifique contenant des calculs interactifs.

La force de Jupyter est de **mélanger un éditeur de texte et une console**[12]. Cette idée de travailler uniquement dans un éditeur surpuissant – et nous sommes loin d'avoir épuisé cette idée – a été popularisée dès les années 1970 par l'éditeur de texte extensible **Emacs**, et par sa dernière version **GNU TEXmacs**[13] qui est programmable en Scheme.

1.6.1 Les *notebooks* de Jupyter

La documentation se trouve sur :

`https://jupyter-notebook.readthedocs.io/en/latest/`

Vous disposez de deux manières pour travailler avec les *notebooks* de Jupyter.

— dans la fenêtre d'*Anaconda Navigator*, cliquez sur *Environments* et choisissez votre environnement (`base` ou bien par exemple notre `simul` si vous souhaitez importer Pyglet ou Pymunk). Ensuite revenez dans *Home* et cliquez sur *Launch* dans le cadre *Jupyter Notebook*. Un terminal s'ouvre qui va lui-même lancer un serveur Web dans votre navigateur à l'adresse `http://localhost:8888`. Les navigateurs conseillés à ce jour sont Firefox, Chrome et Safari.

— ou bien entrez directement au terminal la commande `jupyter notebook`.

Dans les deux cas, vous vous retrouvez dans l'explorateur de fichiers de *Jupyter*, qui affiche vos fichiers et dossiers. Descendez dans votre répertoire de travail usuel.

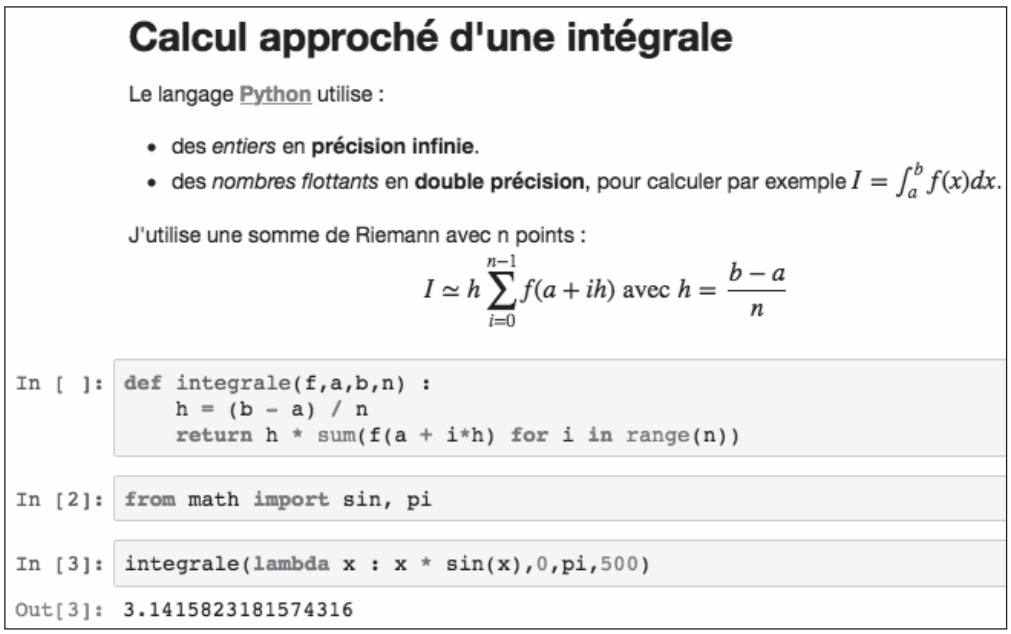
Nous allons créer un *notebook* avec le menu **Nouveau** situé en haut à droite, optez pour *Notebook/Python 3*. Vous arrivez alors dans une fenêtre de *notebook* de nom *Untitled* en haut à gauche. Il est temps pour vous de suivre cette **Visite de l'Interface Utilisateur** dans le menu **Aide**, jusqu'au bout si possible. Changez le nom du fichier en `essai` (l'extension `.ipynb` sera ajoutée automatiquement).

La première et unique cellule est en mode **Code** comme indiqué dans un menu déroulant juste au-dessus : elle est censée contenir du code Python. Utilisons ce menu déroulant pour la mettre en mode **Markdown**[14] : texte enrichi avec balises et formules mathématiques en TEX. Voici ce que j'entre dans cette première cellule qui est en mode *Markdown* (j'ai installé le logiciel TEX sur ma machine).

11. 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.
12. Vous pouvez l'essayer sur `https://jupyter.org/try` avec Python, R, C++, Scheme *etc.*.
13. Voir `www.texmacs.org`.
14. Pour la syntaxe de Markdown, consultez en français le site :
`https://michelf.ca/projets/php-markdown/syntaxe/`
ou en anglais sur Github : `https://guides.github.com/features/mastering-markdown/`.

```
11 # Calcul approché d'une intégrale
12 Le langage [**Python**](http://www.python.org) utilise :
13
14 * des *entiers* en **précision infinie**.
15
16 * des *nombres flottants* en **double précision**, pour calculer
17 par exemple $I = \int_{a}^{b} f(x) dx$.
18
19 J'utilise une somme de Riemann avec n points :
20 $$I\simeq h\sum_{i=0}^{n-1}f(a+ih)\ ;{\rm avec}\ ;h=\frac{b-a}{n}$$
```

Les cellules suivantes sont en mode **Code**, exécutables. Elles contiennent du code Python : une définition de fonction, une importation et un test. Au final, un *notebook* exécutable a été produit.



Calcul approché d'une intégrale

Le langage Python 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 :

$$I \simeq h \sum_{i=0}^{n-1} f(a+ih) \text{ avec } h = \frac{b-a}{n}$$

In [ ]:
```
def integrale(f,a,b,n) :
    h = (b - a) / n
    return h * sum(f(a + i*h) for i in range(n))
```

In [2]:
```
from math import sin, pi
```

In [3]:
```
integrale(lambda x : x * sin(x),0,pi,500)
```

Out[3]: 3.1415823181574316

Quelques mots sur le codage en *Markdown*. En ligne 11, la balise `#` signifie `<h1>` en HTML, elle indique un **titre** sur une seule ligne. En ligne 12, 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 (qui est un *Markup*).

Une ligne vide permet de changer de paragraphe. La balise solitaire `*` en début de ligne indique une puce de liste non ordonnée. Enfin, il est possible d'inclure des formules mathématiques rédigées en langage TEX comme en lignes 17 et 20,

elles seront parfaitement rendues à l'écran et traduites en MathML pour une exportation du *notebook* en HTML [15]. Remarquez le lien `Python` souligné.

Les *notebooks* de Jupyter sont extrêmement sophistiqués. On peut y afficher des graphiques, leur intégrer des cellules audio ou vidéo, et diffuser ces *notebooks* grâce au **Notebook Viewer** [16]. 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` pour IDLE et Spyder, et de fichiers `.ipynb` pour être ouverts dans Jupyter sous la forme de *notebooks*.

1.6.2 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. Un excellent exposé pour le travail collaboratif avec Jupyter, notamment dans les lycées et universités (il est rédigé par des programmeurs et des enseignants) est donné dans le livre **Teaching and Learning with Jupyter** [JUP] que vous enregistrerez dans vos signets car il regorge d'informations débordant du simple enseignement de la programmation. Vous pourrez enfin suivre les nouveautés sur Jupyter dans les conférences annuelles des spécialistes disponibles sur la chaîne *JupyterCon* de YouTube.

Poussant encore plus loin cette idée, le projet successeur **Jupyter Lab** devient un véritable environnement interactif permettant d'utiliser conjointement des *notebooks*, des éditeurs de texte et des outils de visualisation pour ces fameuses *data* qui deviennent 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.

`https://jupyterlab.readthedocs.io/en/stable/`

1.7 Conclusion

Après ce tour d'horizon, installez la distribution Anaconda et évitez d'avoir plusieurs distributions concurrentes. Vous avez ensuite le choix entre plusieurs IDE [17]. Le classique IDLE est très vite pris en main, Spyder est plus sophistiqué avec sa

15. Ceci suppose bien entendu que vous avez installé le logiciel TEX sur votre machine, destiné à produire des documents scientifiques au format pdf. Ce livre est entièrement rédigé avec TEX.

16. Voir `http://nbviewer.jupyter.org`. Tout ceci semble – à juste titre – très inspiré des *notebooks* de Mathematica et du *Computable Document Format* (CDF) de Wolfram Research.

17. Vous pouvez lire ce livre avec tout autre IDE pour Python que vous maîtrisez. Le gros avantage d'Anaconda est le nombre de bibliothèques installées et la souplesse de `conda`.

console IPython, et Jupyter brille dans le firmament au prix d'un certain apprentissage. Quant à Jupyter Lab, attendez d'avoir les idées claires, à moins que vous ne soyez dans un laboratoire administré, d'enseignement ou de recherche.

Si vous êtes un grand débutant, je vous conseille de démarrer très simplement avec IDLE en restant dans l'environnement de base tant que vous n'importez pas de modules extérieurs. Vous voudrez peut-être migrer ensuite vers Spyder ou directement vers Jupyter une fois que vous aurez acquis une certaine aisance.

N'hésitez pas à visionner des vidéos en cherchant « Installation Anaconda, Python, Jupyter, Spyder » sur YouTube. Elles seront le plus souvent en anglais, mais un anglais technique et facile à suivre. Parmi elles, une vidéo sur Spyder du projet *Mathscope* de l'APMEP est en français, obtenue en cherchant : « Programmer avec Python - Présentation de Spyder ».

Si vous souhaitez quelques arguments pour choisir Jupyter, jetez un œil à quelques *notebooks* publics sur :

`https://github.com/jupyter/jupyter/wiki`

Je vais dans ce livre oublier Jupyter pour des raisons typographiques et non scientifiques. Quant à IDLE et Spyder, je vais couper la poire en deux. Avec Spyder, je déplore en tant qu'auteur la place perdue avec les `'In␣[14]:␣'`, les `'Out[14]:␣'` et les lignes vides de séparation, qui gaspillent un espace précieux sur le papier [18]. Je conserve donc le *prompt* `>>>` de IDLE, et remplacerai `Out[i]` par le symbole ▶. En effet, il est parfois désagréable avec IDLE de ne pouvoir distinguer le simple **effet** de l'évaluation d'une expression et la sortie de sa **valeur** affichée par la console.

```
>>> print(2 + 1)
3                       # effet
>>> 2 + 1
▶ 3                     # résultat
```

```
In[1] : print(2 +1)
3                       # effet
In[2] : 2 + 1
Out[2]: 3               # résultat
```

18. Bien qu'il soit possible de personnaliser les *prompts* comme vous le lirez dans le chapitre *Configuration and Customization* de la documentation de IPython.