

Richard Gomez

Le petit Python

Aide-mémoire pour Python 3

2^e édition



Références sciences

Le petit Python

Aide-mémoire pour Python 3

2^e édition

Richard Gomez



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 9782340-112926

Dépôt légal : mai 2026

©Ellipses Édition Marketing S.A.
8/10 rue la Quintinie 75015 Paris



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 1^{er} 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 Pepe Belinchón Cantarero (mi bisabuelo), Martina Martínez Roldán (mi bisabuela), Pilar Belinchón Martínez (mi abuela), Pedro Sáez Martínez (mi abuelo), Brígida Sáez Belinchón (mi madre), Juliana Sáez Belinchón (mi tía), Manolo Sáez Belinchón (mi tío), Pedrito Sáez Belinchón (mi tío), Federico Gómez Rosell (mi abuelo), Antonio Gómez Minaya (mi padre), David Gomez (mon frère), Didier Calas, Farid Belkalem, Ivan Revel, Ludovic Jacquet, les Reeds, Jean-Jacques Chervin, Stéphane Jammes, Mahfoud Chebboub, Stéphane Baseilhac, Miguel Angel Muñoz Fuentes, Alexandre Trijoulet, Ysé Commenges, Yanis Riguet et Estelle Chaumont.

À la mémoire de Rufino Jimenez Rosell, Pepa Belinchón Martínez, Bautista Villanueva Sáez, Cédric Darolles et Nicolas Guilloux.

À la mémoire de Leonardo Torres Quevedo (1852-1936).

Prologue

Prologue de la présente édition

Suite au succès de la première édition [2], Paul de Laboulaye, des éditions Ellipses, m’a proposé de rééditer *Le petit Python*. J’ai accepté et profité de cette occasion pour intégrer les nouveautés du langage (comme `match-case`, par exemple) et étoffer certaines parties, notamment celles consacrées aux bibliothèques, à l’encodage du texte et aux modules `math`, `cmath`, `fractions`, `decimal`, `random`, `os`, `os.path` et `turtle`. Nous avons également ajouté un chapitre consacré au module `statistics`.

Prologue corrigé de la première édition

Python est un langage de programmation (*langage de script*) permettant de pratiquer la programmation *impérative* (écriture d’une séquence d’instructions), la programmation *fonctionnelle* (résolution de problèmes par la création de fonctions) et la programmation *orientée objet* (définition d’objets que l’on fait interagir entre eux).

Ce langage a été créé en 1989 par Guido van Rossum et publié en 1991. Python est distribué sous une licence libre. On peut le télécharger gratuitement sur le site officiel (<http://www.python.org>) et l’installer sur n’importe quel ordinateur (quel que soit le système, y compris les tablettes et les calculatrices).

Python est le deuxième langage le plus utilisé au monde (classement REDMONK 2025). Le big data et l’intelligence artificielle l’ont massivement adopté. Il est largement employé par les géants du web comme Google, YouTube, Dropbox ou Instagram, et il alimente des technologies telle que le framework Django. Il a également été adopté par le milieu universitaire, les étudiants, la plupart des scientifiques – en particulier les mathématiciens – et par de nombreux systèmes éducatifs. Aujourd’hui, les lycéens français écrivent, dès la classe de seconde, des programmes en Python pour résoudre des problèmes en mathématiques et en physique.

Les qualités de Python sont nombreuses et séduisantes :

- une syntaxe élégante, lisible et intuitive ;
- une facilité d’apprentissage ;
- une évolution constante ;
- une certaine puissance ;
- et des milliers de bibliothèques couvrant tous les usages.

Face à une telle richesse et à de si nombreuses possibilités, le présent ouvrage propose un aide-mémoire permettant de retrouver rapidement et efficacement n'importe quelle instruction, objet ou élément de syntaxe du langage. Ce petit volume permet de maîtriser Python de manière avancée, sans avoir à faire de longues recherches sur Internet.

Nous présentons clairement et rigoureusement les notions fondamentales : les objets, les types, l'aliasing, les espaces de noms, la mutation, la résolution d'un identifiant, les modules, l'encodage des nombres, les conditions logiques, les boucles, les exceptions et leur traitement, les gestionnaires de contexte, les fonctions, les décorateurs, les booléens, les chaînes de caractères (encodage, formatage, etc.), les octets, les listes, les ensembles, les dictionnaires, les fichiers, les itérateurs, les générateurs, ainsi que les normes ASCII et UNICODE. Nous abordons également, en passant, l'auto-test, les docstrings, les doctests et les fonctions lambda. Nous détaillons les modules `math`, `cmath`, `fractions`, `decimal`, `random`, `statistics`, `os`, `os.path` et `turtle`. Enfin, nous consacrons un chapitre à l'introspection, c'est-à-dire à l'exploration du langage lui-même.

L'intérêt de cet ouvrage réside dans sa clarté et la profusion d'exemples : chaque notion est illustrée par un ou plusieurs cas concrets.

Le lecteur n'est pas obligé de lire les chapitres dans l'ordre, il peut se rendre directement à l'endroit où se trouve l'information recherchée. La table des matières est détaillée et conçue dans ce but (711 entrées). À la fin de l'ouvrage, on trouve un index pratique et riche de 752 entrées. Lorsqu'une notion apparaît dans plusieurs chapitres, des renvois clairs indiquent la section où poursuivre son exploration.

Le présent ouvrage repose sur Python version 3.13.15. Le lecteur pourra consulter les nouveautés (à venir) sur le site officiel, à l'adresse <https://docs.python.org/3/whatsnew/index.html>.

Dans ce texte, nous employons quelques abus d'écriture. Par exemple, pour indiquer que `x` est un objet de type `bytes`, nous écrivons parfois : `x` est un `bytes`. Le `s` peut surprendre alors que nous sommes au singulier, mais nous considérons le mot `bytes` comme un nom propre, c'est-à-dire le nom d'une classe d'objets. D'ailleurs, `x` est probablement composé non pas d'un seul byte, mais de plusieurs. On retrouve le même usage avec des phrases comme : `x` et `y` sont des `str`, pour signifier que `x` et `y` sont des objets de type `str`. Ici, `str` reste au singulier, car il s'agit du nom propre d'une classe. Un autre choix d'écriture consiste à couper une ligne de code lorsque celle-ci s'avère trop longue pour le format de ce livre. Néanmoins, la césure est faite de manière à ce que la lecture reste facile et naturelle. Autre point important : lorsqu'une référence renvoie à la section 6.5, par exemple, cela signifie la section 5 du chapitre 6. La référence 18.16.1, quant à elle, désigne la sous-section 1 de la section 16 du chapitre 18.

Le présent document est le fruit d'une longue compilation d'informations récoltées et mûries au fil des années, des besoins, et de divers projets de programmation. Je tiens à remercier chaleureusement mon ami et collègue Jocelyn Etienne, avec lequel j'ai pu échanger de nombreuses idées sur la programmation et sur Python en particulier, par le biais de conversations, de messages électroniques (parfois de SMS) et de défis passionnants lancés mutuellement. Je le remercie également pour toutes les remarques

qu'il a bien voulu formuler en lisant ces pages ainsi que pour les multiples erreurs et coquilles qu'il a relevées. Je remercie aussi mon ami et collègue Stéphane Lévy, avec lequel j'ai pu également échanger sur la programmation. Stéphane a lui aussi participé activement à la recherche des erreurs et des coquilles. Je tiens à citer ChatGPT avec lequel j'ai testé certaines idées de programmation. Je remercie Charles-Elie Liebart (informaticien) et Philippe Joyez (expert en TeXmacs) pour m'avoir aidé à utiliser correctement l'éditeur de texte ayant servi à la première édition de ce document. Pour la deuxième édition, je remercie Joris van der Hoeven (créateur de TeXmacs, [21]) et Grégoire Lecerf (expert TeXmacs) pour le temps qu'ils m'ont consacré. Sans eux, ce document n'existerait probablement pas sous cette forme. Je remercie ma compagne Estelle Chaumont pour son soutien et la patience dont elle a fait preuve au cours des mois ayant précédé la publication de ce livre. Je remercie également mes parents, Antoine et Brigitte, pour leur soutien. Enfin, je voudrais remercier Paul de Laboulaye, Corinne Baud et les éditions Ellipses pour avoir eu la gentillesse de s'intéresser à ce document et pour avoir accepté de le publier.

Ce livre a été composé avec $\text{\TeX}_{\text{MACS}}$ 2.1.4 – éditeur de documents scientifiques, logiciel libre (GNU).



Table des matières

Prologue	5
Prologue de la présente édition	5
Prologue corrigé de la première édition	5
1 Lancer Python et éditer un programme	27
1.1 Télécharger Python	27
1.2 IDLE	27
1.3 Fenêtre <code>Shell</code>	27
1.4 Versions	28
1.5 Éditer un script	28
1.6 Recommandations	28
1.7 Commentaires	29
1.8 Shebang et encodage	29
1.9 Blocs et indentations	30
1.10 Affichage dans le <code>Shell</code>	31
1.11 Utiliser un terminal	32
1.12 Fonction <code>help</code>	33
2 Mots réservés et objets internes	35
2.1 Mots réservés	35
2.2 Fonctions internes	35
2.3 Module <code>builtins</code>	37
2.4 Méthodes internes	38
2.5 Méthodes spéciales	39
2.6 Autres objets internes	40
2.7 Introspection	41
3 Types offerts par Python	43
3.1 Objets	43
3.2 Types usuels	43
3.3 Classifications des types	47
3.4 Tester le type d'un objet	50
3.5 Typage dynamique	50
4 Affectations, identificateurs et aliasing	53
4.1 Introduction	53
4.2 Fonction <code>id</code> et opérateur <code>is</code>	54

4.3	Noms, identificateurs et références	54
4.4	Effacer un nom : <code>del</code>	55
4.5	Aliasing	56
4.6	Affectations	57
4.7	Décompression	59
4.8	Opérateur <code>splat</code>	61
4.9	Indentificateurs autorisés par Python	62
4.10	Espaces de noms	63
5	Opérateurs	67
5.1	Arité	67
5.2	Chaînage et sens de regroupement	68
5.3	Fonction <code>reduce</code>	69
5.4	Ordre de priorité	70
6	Modules et bibliothèques	73
6.1	Notion de module et de bibliothèque	73
6.2	Importer un module ou son contenu	74
6.2.1	Instruction <code>import</code>	74
6.2.2	Instruction <code>from-import</code>	75
6.3	Bibliothèques standards	75
6.4	Autres bibliothèques (PyPI)	76
6.5	Gestionnaire de paquets <code>pip</code>	76
6.6	Importer un module personnel	77
6.7	Chemin d'un module	78
6.8	Fabriquer une bibliothèque (<i>packages</i>)	79
6.8.1	Namespace <code>packages</code>	79
6.8.2	Regular <code>packages</code>	80
6.8.3	Objets exposés	80
6.8.4	Astérisque magique et variable <code>__all__</code>	81
6.8.5	Arborescence	82
6.8.6	Résolution d'un nom	82
7	Affichage <code>print</code>, <code>str</code> et <code>repr</code>	83
7.1	Introduction	83
7.2	Paramètre <code>sep</code>	83
7.3	Paramètre <code>end</code>	83
7.4	Paramètre <code>file</code>	84
7.5	Paramètre <code>flush</code>	85
7.6	Fonction <code>str</code> versus <code>repr</code>	86
7.7	Module Pretty Print (<code>pprint</code>)	86
8	Saisie au clavier : <code>input</code>	87
8.1	Fonctionnement	87
8.2	Convertir la saisie	88

8.3 Flux des saisies au clavier	88
9 Nombres et mathématiques	89
9.1 Entiers (int)	89
9.1.1 Introduction	89
9.1.2 Attributs-données	89
9.1.3 Méthodes pour le duck typing	90
9.1.4 Méthodes de représentation binaire	90
9.1.5 Remarque sur les booléens	91
9.2 Décimaux (float)	92
9.2.1 Introduction	92
9.2.2 Attributs pour le duck typing	92
9.2.3 Méthodes de représentation fractionnaire	92
9.3 Complexes (complex)	94
9.4 Bases 2, 8 et 16	94
9.4.1 Écriture binaire	94
9.4.2 Écriture octale	95
9.4.3 Écriture hexadécimale	95
9.4.4 Rappel des préfixes	96
9.5 Opérations élémentaires	96
9.6 Opérations bit à bit	97
9.7 Comparaisons	100
9.7.1 Égalité	100
9.7.2 Relation d'ordre	100
9.7.3 Identité	101
9.8 Fonctions mathématiques	101
9.8.1 Opérations usuelles	101
9.8.1.1 Valeur absolue : abs	101
9.8.1.2 Division euclidienne : divmod	101
9.8.1.3 Fonction pow	101
9.8.2 Troncature et arrondi	102
9.8.2.1 Troncature : int	102
9.8.2.2 Arrondir : round	103
9.8.3 Sommation et autres	104
9.8.3.1 Sommation : sum	104
9.8.3.2 Fonctions min et max	104
9.8.4 Binaire et logique	105
9.8.4.1 Fonction all	105
9.8.4.2 Fonction any	105
9.8.4.3 Fonctions bin , oct et hex	106
9.8.5 Typage	106
9.8.5.1 Fonction int	106
9.8.5.2 Fonction float	106
9.8.5.3 Fonction complex	107
9.8.6 Autres	107
9.8.6.1 Fonction eval	107

9.8.6.2 Fonction range	108
9.9 Problème de l'encodage des décimaux	108
9.10 Objet NAN et nombres infinis	109
9.11 Annexe : écriture binaire	111
9.11.1 Complément à deux	111
9.11.2 Boutisme	111
9.12 Annexe : division euclidienne	112
10 Structure if-elif-else	115
10.1 Instruction if simple	115
10.2 Instruction if-else	115
10.3 Instruction if-elif-else	116
10.4 Expression ternaire « x if y else z »	116
11 Structure match-case	119
11.1 Introduction	119
11.2 Motifs typés	119
11.3 Motifs typés avec attributs	120
11.4 Motifs typés avec attributs de position	121
11.5 Motifs disjonctifs	122
11.6 Motifs avec condition	122
11.7 Motifs séquences	123
11.8 Motifs séquences avec item disjonctif	124
11.9 Motifs dictionnaires	124
11.10 Effets de bord	125
12 Boucles avec while	127
12.1 Instruction while simple	127
12.2 Instruction continue	127
12.3 Instruction break	128
12.4 Instruction while-break-else	128
13 Boucles avec for	131
13.1 Instruction for-in	131
13.2 Instruction continue	131
13.3 Instruction break	132
13.4 Instruction for-break-else	132
13.5 Fonction enumerate	133
13.6 Fonction zip	133
13.7 Mécanisme derrière une itération	134
14 Gestion des erreurs avec try-except-else	135
14.1 Échec et exception	135
14.2 Structure try-except-else-finally	136
14.3 Instruction except	138

14.4	Instruction finally	139
14.5	Types d'exceptions	140
14.6	Lever une exception avec raise	142
14.7	Traceback	143
14.8	Lever une exception avec raise-from	144
14.9	Tester une assertion avec assert	146
14.10	Cas des fichiers	147
14.11	Paquets d'exceptions	147
14.12	Gestionnaire global d'exceptions	148
14.13	Affichages	149
14.14	Exemple amusant	150
15	Gestionnaires de contexte : with	151
15.1	Définition	151
15.2	Exemple orienté objet	151
15.3	Faire passer des données au gestionnaire	152
15.4	Instruction with-as	153
15.5	Fonction contextmanager du module contextlib	154
15.6	Décorateur gestionnaire de contexte	155
15.7	Question de style : with versus try	157
16	Fonctions	159
16.1	Définir et appeler une fonction	159
16.2	Paramètres	161
16.3	Arguments	162
16.4	Valeurs par défaut et effets de bord	163
16.5	Variables locales	164
16.6	Variables non locales (libres)	165
16.7	Décompression *	167
16.8	Décompression **	168
16.9	Documentation (docstring)	169
16.10	Fonction testmod du module doctest	170
16.11	Annotations	171
16.12	Instruction lambda	172
16.13	Objets appelables	173
17	Décorateurs	175
17.1	Définition	175
17.2	Opérateur @	175
17.3	Signature de la décorée	177
17.4	Préserver la docstring	177
17.5	Fonction wraps du module functools	178
17.6	Décorateurs avec paramètres	179
17.7	Exemples classiques	180
17.7.1	Journal de bord attaché à une fonction : logging	180

17.7.2	Compteur attaché à une fonction	180
17.7.3	Chronomètre attaché à une fonction : benchmark	181
17.7.4	Cache attaché à une fonction : mémoïsation	181
17.7.5	Annexe : amélioration du cache avec <code>bind</code>	182
18	Booléens : type <code>bool</code>	185
18.1	Introduction	185
18.2	Opérateurs de comparaison	185
18.3	Valeur booléenne d'un objet	186
18.4	Valeur numérique d'un booléen	188
18.5	Opérations logiques	188
18.5.1	Négation	188
18.5.2	Conjonction	189
18.5.3	Disjonction	189
18.5.4	Disjonction exclusive	189
18.5.5	Règles de priorité	189
18.6	Short-cut	190
18.7	Introspection	192
19	Chaînes de caractères : type <code>str</code>	193
19.1	Introduction	193
19.2	Délimiteurs	193
19.3	Caractères et UNICODE	194
19.4	Caractères blancs	198
19.5	Codes d'échappement	198
19.6	Représentation formelle	199
19.6.1	Cas d'un caractère	200
19.6.2	Cas d'une chaîne de plusieurs caractères	200
19.6.3	Bilan	201
19.6.4	Fonction <code>ascii</code>	202
19.7	Triples guillemets ou triples quotes	202
19.8	Chaînes brutes (<i>raw strings</i>)	202
19.9	Chaîne retournée par <code>input</code>	203
19.10	Fonction <code>str</code>	203
19.11	Indexation	204
19.12	Slicing	204
19.13	Itération	205
19.14	Test <code>in</code>	205
19.15	Ordre	206
19.16	Conversion avec <code>list</code> , <code>tuple</code> , <code>set</code> et <code>frozenset</code>	206
19.17	Interprétation du contenu	206
19.17.1	Fonction <code>int</code>	207
19.17.2	Fonction <code>float</code>	207
19.17.3	Fonction <code>complex</code>	207
19.17.4	Fonction <code>eval</code>	207
19.18	Concaténation	207

19.19	Méthode format	208
19.20	Chaînes formatées : préfixe f	208
19.21	Introspection	209
19.22	Méthodes de str	209
19.22.1	Méthode count	209
19.22.2	Méthode find	210
19.22.3	Méthode rfind	210
19.22.4	Méthode index	210
19.22.5	Méthode rindex	210
19.22.6	Méthode split	211
19.22.7	Méthode rsplit	211
19.22.8	Méthode splitlines	211
19.22.9	Méthode partition	212
19.22.10	Méthode rpartition	212
19.22.11	Méthode join	212
19.22.12	Méthode upper	213
19.22.13	Méthode isupper	213
19.22.14	Méthode lower	213
19.22.15	Méthode islower	213
19.22.16	Méthode casefold	213
19.22.17	Méthode title	214
19.22.18	Méthode istitle	214
19.22.19	Méthode capitalize	215
19.22.20	Méthode swapcase	215
19.22.21	Méthode isalnum	215
19.22.22	Méthode isalpha	215
19.22.23	Méthode isdecimal	215
19.22.24	Méthode isdigit	216
19.22.25	Méthode isnumeric	216
19.22.26	Méthode isspace	216
19.22.27	Méthode isprintable	216
19.22.28	Méthode isascii	217
19.22.29	Méthode isidentifier	217
19.22.30	Méthode startswith	217
19.22.31	Méthode endswith	218
19.22.32	Méthode strip	218
19.22.33	Méthode lstrip	218
19.22.34	Méthode rstrip	219
19.22.35	Méthode removeprefix	219
19.22.36	Méthode removesuffix	219
19.22.37	Méthode center	219
19.22.38	Méthode ljust	220
19.22.39	Méthode rjust	220
19.22.40	Méthode expandtabs	220
19.22.41	Méthode zfill	221
19.22.42	Méthode format	221

19.22.43	Méthode <code>format_map</code>	222
19.22.44	Méthode <code>replace</code>	222
19.22.45	Méthode <code>maketrans</code>	222
19.22.46	Méthode <code>translate</code>	223
19.22.47	Méthode <code>encode</code>	223
19.23	Pour aller plus loin	224
20	Formater les chaînes avec <code>format</code>	225
20.1	Introduction	225
20.2	Syntaxe d'une balise	225
20.3	Coder une référence	226
20.3.1	Code <i>nom</i>	226
20.3.2	Code <i>attribut</i> , <i>clé</i> ou <i>indice</i>	227
20.4	Coder une <i>conversion</i>	228
20.5	Coder un <i>formatage</i>	229
20.5.1	Codes <i>fill</i> , <i>align</i> et <i>width</i>	229
20.5.2	Code <i>sign</i>	230
20.5.3	Code <i>type</i>	230
20.5.4	Code <i>precision</i>	233
20.5.5	Code z	235
20.5.6	Code #	236
20.5.7	Code 0	237
20.5.8	Code <i>grouping</i>	237
20.6	Application : afficher un tableau	238
20.7	Formater les dates	238
20.8	Échappement	241
20.9	Formatage dynamique	241
20.10	Méthode <code>format_map</code>	242
20.11	Fonction <code>format</code>	243
20.12	Ancien formatage	243
21	Chaînes d'octets : types <code>bytes</code> et <code>bytearray</code>	245
21.1	Bits, bytes et octets	245
21.2	Types <code>bytes</code> et <code>bytearray</code>	245
21.3	Écrire une chaîne d'octets	246
21.3.1	Un octet	246
21.3.2	Plusieurs octets	247
21.4	Délimiteurs d'une chaîne d'octets	247
21.5	Affichages des chaînes d'octets	247
21.6	Indexage et itération	248
21.7	Slicing	248
21.8	Fonction <code>bytes</code>	249
21.9	Test <code>in</code>	250
21.10	Ordre	250
21.11	Conversion avec <code>list</code> , <code>tuple</code> , <code>set</code> et <code>frozenset</code>	250
21.12	Concaténation	251

21.13	Écrire un tableau d'octets	251
21.14	Opérations avec tableaux d'octets	251
21.15	Opérations avec chaînes et tableaux d'octets	252
21.16	Fichiers	253
21.17	Introspection	253
21.18	Méthodes	255
21.18.1	Méthodes spécifiques	255
21.18.1.1	Méthode <code>decode</code>	255
21.18.1.2	Méthode <code>hex</code>	255
21.18.1.3	Méthode <code>fromhex</code>	256
21.18.2	Méthodes de <code>bytes</code>	256
21.18.3	Méthodes de <code>bytearray</code>	256
22	Plages d'entiers : type <code>range</code>	257
22.1	Introduction	257
22.2	Indexation	258
22.3	Slicing	258
22.4	Itération	259
22.5	Test <code>in</code>	259
22.6	Conversion avec <code>list</code> , <code>tuple</code> , <code>set</code> et <code>frozenset</code>	260
22.7	Conversion avec <code>bytes</code> et <code>bytearray</code>	260
22.8	Introspection	260
23	Listes : type <code>list</code>	263
23.1	Introduction	263
23.2	Fonction <code>list</code>	263
23.3	Indexation	264
23.4	Slicing	265
23.5	Itération	266
23.6	Test <code>in</code>	267
23.7	Ordre	267
23.8	Conversion avec <code>tuple</code> , <code>set</code> et <code>frozenset</code>	268
23.9	Conversion avec <code>bytes</code> et <code>bytearray</code>	268
23.10	Concaténation	269
23.11	Gare à l'aliasing !	269
23.12	Compréhension	270
23.13	Produit cartésien	271
23.14	Aplatir	271
23.15	Introspection	272
23.16	Méthodes	272
23.16.1	Méthode <code>index</code>	272
23.16.2	Méthode <code>count</code>	273
23.16.3	Méthode <code>remove</code>	273
23.16.4	Méthode <code>pop</code>	273
23.16.5	Méthode <code>clear</code>	274
23.16.6	Méthode <code>append</code>	274

23.16.7	Méthode extend	274
23.16.8	Méthode insert	275
23.16.9	Méthode sort	275
23.16.10	Méthode reverse	276
23.16.11	Méthode copy	276
24	Uplets : type tuple	277
24.1	Introduction	277
24.2	Fonction tuple	278
24.3	Indexation	278
24.4	Slicing	279
24.5	Itérer un uplet	279
24.6	Test in	279
24.7	Ordre	280
24.8	Conversion avec list , set et frozenset	280
24.9	Conversion avec bytes et bytearray	280
24.10	Concaténation	280
24.11	Mutations	281
24.12	Pas de compréhension	281
24.13	Introspection	281
24.14	Méthodes	282
24.14.1	Méthode index	282
24.14.2	Méthode count	282
24.15	Fonction namedtuple	282
25	Ensembles : types set et frozenset	285
25.1	Introduction	285
25.2	Fonctions set et frozenset	286
25.3	Itération	286
25.4	Test in	287
25.5	Ordre	287
25.6	Conversion avec list et tuple	288
25.7	Conversion avec bytes et bytearray	288
25.8	Opérations	288
25.9	Compréhension	289
25.10	Tables de hachages	290
25.11	Piocher au hasard	291
25.12	Introspection	291
25.13	Méthodes communes à set et frozenset	292
25.13.1	Méthode union	292
25.13.2	Méthode intersection	292
25.13.3	Méthode difference	292
25.13.4	Méthode symmetric_difference	293
25.13.5	Méthode issubset	293
25.13.6	Méthode isuperset	293
25.13.7	Méthode isdisjoint	293

25.13.8	Méthode <code>copy</code>	293
25.14	Méthodes propres à <code>set</code>	294
25.14.1	Méthode <code>add</code>	294
25.14.2	Méthode <code>remove</code>	294
25.14.3	Méthode <code>discard</code>	295
25.14.4	Méthode <code>pop</code>	295
25.14.5	Méthode <code>clear</code>	295
25.14.6	Méthode <code>update</code>	296
25.14.7	Méthode <code>intersection_update</code>	296
25.14.8	Méthode <code>difference_update</code>	296
25.14.9	Méthode <code>symmetric_difference_update</code>	296
26	Dictionnaires : type <code>dict</code>	297
26.1	Introduction	297
26.2	Fonction <code>dict</code>	298
26.3	Clés, valeurs et items	299
26.4	Accès par clé	299
26.5	Itération	300
26.6	Ordre des items	301
26.7	Test <code>in</code>	301
26.8	Conversion avec <code>list</code> , <code>tuple</code> , <code>set</code> , <code>frozenset</code>	301
26.9	Conversion avec <code>bytes</code> et <code>bytearray</code>	302
26.10	Union	302
26.11	Compréhension	302
26.12	Programmer avec un dictionnaire	303
26.13	Introspection	303
26.14	Méthodes d'un dictionnaire	304
26.14.1	Méthode <code>fromkeys</code>	304
26.14.2	Méthode <code>setdefault</code>	304
26.14.3	Méthode <code>update</code>	304
26.14.4	Méthode <code>keys</code>	305
26.14.5	Méthode <code>items</code>	305
26.14.6	Méthode <code>values</code>	305
26.14.7	Méthode <code>get</code>	306
26.14.8	Méthode <code>pop</code>	306
26.14.9	Méthode <code>popitem</code>	306
26.14.10	Méthode <code>clear</code>	307
26.14.11	Méthode <code>copy</code>	307
27	Fichiers	309
27.1	Introduction	309
27.2	Chemin, nom et extension	310
27.3	Dossier courant	311
27.4	Instancier un objet de type <i>fichier</i>	312
27.5	Modes	312
27.6	Types d'accès	313

27.6.1	Lecture	313
27.6.2	Écriture	313
27.6.3	Création-écriture	314
27.6.4	Ajout	314
27.6.5	Mixte	314
27.7	Fin de ligne, lignes et curseur	314
27.8	Attributs-données	315
27.9	Itération	316
27.10	Méthodes communes à tous les fichiers	317
27.10.1	Méthode <code>readable</code>	317
27.10.2	Méthode <code>read</code>	317
27.10.3	Méthode <code>readline</code>	317
27.10.4	Méthode <code>readlines</code>	318
27.10.5	Méthode <code>writable</code>	318
27.10.6	Méthode <code>write</code>	318
27.10.7	Méthode <code>writelines</code>	319
27.10.8	Méthode <code>close</code>	319
27.10.9	Méthode <code>seekable</code>	319
27.10.10	Méthode <code>tell</code>	320
27.10.11	Méthode <code>seek</code>	320
27.10.12	Méthode <code>isatty</code>	320
27.10.13	Autres méthodes	321
27.11	Méthode propre aux fichiers <i>texte</i>	321
27.12	Méthodes propres aux fichiers <i>binaires</i>	321
27.13	Pseudo-fichiers	321
27.13.1	Objet de type <i>fichier classique</i>	322
27.13.2	Flux standard	322
27.13.3	Objet de type <i>fichier tampon</i>	324
27.14	Paramètre <code>file</code> de <code>print</code>	324
27.15	Exemple	325
27.16	Module <code>pickle</code>	326
28	Itérateurs	329
28.1	Introduction	329
28.2	Fonction <code>next</code>	329
28.3	Fonction <code>iter</code>	330
28.4	Fonction <code>reversed</code>	332
28.5	Fonction <code>enumerate</code>	333
28.6	Fonction <code>zip</code>	334
28.7	Fonction <code>map</code>	335
28.8	Fonction <code>filter</code>	336
28.9	Module <code>itertools</code>	337
28.9.1	Fonction <code>chain</code>	337
28.9.2	Fonction <code>islice</code>	337
29	Générateurs	339

29.1	Fonction génératrice	339
29.2	Générateur	339
29.3	Instruction <code>yield from</code>	342
29.4	Instruction <code>return</code>	343
29.5	Méthodes d'un générateur	344
29.5.1	Méthode <code>close</code>	344
29.5.2	Méthode <code>throw</code>	346
29.5.3	Méthode <code>send</code>	346
29.6	Attributs d'un générateur	348
29.7	Expressions génératrices	348
29.8	Notion de paresse	349
30	Encodage et décodage	351
30.1	Introduction	351
30.1.1	Standard	351
30.1.2	Norme	351
30.1.3	Standard versus norme	352
30.2	Méthodes pour encoder et décoder	352
30.3	Paramètre <code>errors</code>	353
30.4	Fonction <code>register_error</code>	355
30.4.1	Encodage	355
30.4.2	Décodage	356
30.5	Souplesse du paramètre <code>encoding</code>	357
30.6	Norme ASCII	358
30.7	Norme LATIN-1	359
30.8	Norme UTF-8	360
30.9	Fichiers textes	361
30.10	Encodage d'un script Python	362
31	Introspection	363
31.1	Tester le type	363
31.2	Manipuler les attributs d'un objet	364
31.2.1	Fonction <code>hasattr</code>	364
31.2.2	Fonction <code>delattr</code>	365
31.2.3	Fonction <code>setattr</code>	365
31.2.4	Fonction <code>getattr</code>	366
31.3	Lister les attributs : <code>dir</code>	366
31.4	Explorer les classes	366
31.4.1	Classes et instances	366
31.4.2	Fonction <code>isinstance</code>	367
31.4.3	Fonction <code>issubclass</code>	367
31.4.4	Attribut <code>__bases__</code> et méthode <code>mro</code>	367
31.5	Fonction <code>callable</code>	368
31.6	Fonction <code>help</code> et attribut <code>__doc__</code>	368
31.7	Attribut <code>__name__</code>	369
31.8	Module <code>builtins</code>	370

32 Module <code>math</code>	373
32.1 Introspection	373
32.2 Constantes	373
32.3 Calcul	374
32.4 Théorie des nombres	375
32.5 Arithmétique	375
32.6 Géométrie	376
32.7 Analyse combinatoire	377
32.8 Fonctions classiques en analyse réelle	378
32.9 Fonctions trigonométriques	380
32.10 Fonctions hyperboliques	381
32.11 Fonctions de test	381
32.12 Fonctions informatiques	382
32.13 Annexe : division euclidienne	383
33 Module <code>cmath</code>	387
33.1 Introspection	387
33.2 Constantes	387
33.3 Fonctions élémentaires	388
33.4 Fonctions classiques en analyse complexe	389
33.5 Fonctions de test	391
34 Module <code>fractions</code>	393
34.1 Introduction	393
34.2 Instancier une fraction	393
34.3 Représentations d'une fraction	394
34.4 Attributs d'une fraction	394
34.5 Méthodes d'une fraction	395
34.5.1 Méthode <code>as_integer_ratio</code>	395
34.5.2 Méthode <code>is_integer</code>	395
34.5.3 Méthode <code>limit_denominator</code>	395
34.5.4 Méthode <code>from_decimal</code>	395
34.5.5 Méthode <code>from_float</code>	396
34.5.6 Méthode <code>conjugate</code>	396
34.6 Compatibilité avec les opérations	396
34.7 Compatibilité avec les fonctions mathématiques	397
34.8 Compatibilité avec <code>math</code>	397
34.9 Compatibilité avec <code>cmath</code>	397
35 Module <code>decimal</code>	399
35.1 Introduction	399
35.2 Instancier un décimal	399
35.3 Représentation d'un décimal	400
35.4 Forme normalisée : <code>DecimalTuple</code>	400
35.5 Contextes	401

35.5.1	Fonction <code>getcontext</code>	401
35.5.2	Attribut <code>prec</code>	402
35.5.3	Attribut <code>rounding</code>	403
35.5.4	Attribut <code>Emin</code>	403
35.5.5	Attribut <code>Emax</code>	404
35.5.6	Attribut <code>capitals</code>	404
35.5.7	Attribut <code>flags</code>	405
35.5.8	Attribut <code>traps</code>	405
35.5.9	Attribut <code>clamp</code>	406
35.5.10	Fonctions pour gérer le contexte	406
35.5.11	Contextes enregistrés	407
35.5.12	Méthodes d'un contexte	407
35.6	Modes d'arrondi (<i>rounding</i>)	408
35.7	Méthodes d'un décimal	409
35.8	Exceptions décimales	410
35.9	Compatibilité avec les opérations	411
35.10	Compatibilité avec les fonctions mathématiques	411
35.11	Compatibilité avec <code>math</code>	412
35.12	Compatibilité avec <code>cmath</code>	412
36	Module <code>random</code>	413
36.1	Introduction	413
36.2	Variables aléatoires discrètes	413
36.2.1	Fonction <code>getrandbits</code>	413
36.2.2	Fonction <code>randint</code>	413
36.2.3	Fonction <code>randrange</code>	414
36.2.4	Fonction <code>binomialvariate</code>	414
36.3	Variables aléatoires continues	414
36.3.1	Fonction <code>random</code>	414
36.3.2	Fonction <code>uniform</code>	414
36.3.3	Fonction <code>triangular</code>	415
36.3.4	Fonction <code>expovariate</code>	415
36.3.5	Fonction <code>gauss</code>	415
36.3.6	Fonction <code>normalvariate</code>	416
36.3.7	Fonction <code>betavariate</code>	416
36.3.8	Fonction <code>paretovariate</code>	416
36.3.9	Fonction <code>lognormvariate</code>	417
36.3.10	Fonction <code>gammavariate</code>	417
36.3.11	Fonction <code>weibullvariate</code>	417
36.3.12	Fonction <code>vonmisesvariate</code>	418
36.4	Octets aléatoires	418
36.5	Échantillonnage et tirages	418
36.5.1	Tirage unique	418
36.5.2	Tirages ordonnés avec remise	419
36.5.3	Échantillonnage : tirages ordonnés sans remise	419
36.6	Mélanger les items	420

36.7	État du générateur aléatoire	420
36.8	Classe <code>Random</code>	422
36.9	Classe <code>SystemRandom</code>	423
36.10	Constantes internes	423
36.11	Visualisation	423
37	Module <code>statistics</code>	425
37.1	Fonctions	425
37.2	Classe <code>NormalDist</code>	426
37.3	Autres objets	426
38	Modules <code>os</code>, <code>os.path</code> et <code>glob</code>	427
38.1	Introduction	427
38.2	Ouvrir, lire, écrire et fermer	429
38.3	Dossier courant	430
38.4	Exploration	430
38.5	Création	432
38.6	Édition	433
38.7	Destruction	433
38.8	Variables d'environnement	434
38.9	Commandes et OS	435
38.10	Concaténation de noms	435
38.11	Analyse d'un nom	436
38.12	Décomposition d'un nom	436
38.13	Reformulation d'un nom	437
38.14	Module <code>glob</code>	439
38.14.1	Fonction <code>glob</code>	439
38.14.2	Syntaxe d'un motif	439
38.14.3	Autres fonctions	440
39	Module <code>turtle</code>	441
39.1	Coordonnées (modes)	441
39.2	Angles	442
39.2.1	Unités d'angle	442
39.2.2	Angle de la tortue	443
39.3	Fonctions de base pour dessiner	443
39.3.1	Lever et baisser le stylo	443
39.3.2	Avancer et reculer	444
39.3.3	Pivoter	444
39.3.4	Remplir une surface	445
39.4	Cercles	445
39.5	Points	446
39.6	Empreintes	446
39.7	Texte	447
39.8	Déplacements absolus	447

39.9	Enregistrer un polygone	448
39.10	Couleurs	448
39.10.1	Mode couleur 1	449
39.10.2	Mode couleur 255	449
39.10.3	Syntaxe Tkinter	449
39.10.4	Couleur du stylo	449
39.10.5	Couleur de remplissage	450
39.10.6	Stylo et remplissage	450
39.10.7	Fond du canevas	451
39.11	Effacer, remettre à zéro	451
39.12	Aspect	452
39.12.1	Épaisseur du stylo	452
39.12.2	Titre et taille de la fenêtre Turtle	453
39.12.3	Apparence de la tortue	453
39.13	Vitesse	456
39.14	Programmation événementielle	458
39.14.1	Saisie au clavier	458
39.14.2	Évènements sur le canevas	458
39.14.3	Évènements sur la tortue	459
39.14.4	Temporiser	459
39.15	Éléments d'une fenêtre Turtle	460
39.15.1	Fenêtre Turtle elle-même	460
39.15.2	Canevas	460
39.15.3	Tortues	461
39.16	Classes	461
39.16.1	Classe TurtleScreen	461
39.16.2	Classe Turtle	462
39.16.3	Classe Shape	462
39.16.4	Classe Vec2D	463
	Bibliographie	465
	Index	467

Chapitre 1

Lancer Python et éditer un programme

1.1 Télécharger Python

On télécharge Python sur le site officiel de la Python Software Foundation, www.python.org (c'est gratuit). Le téléchargement et l'installation ne durent pas plus de 5 minutes.

1.2 IDLE

Nous conseillons aux débutants de travailler avec IDLE PYTHON (*Integrated DeveLopment Environment*). Il s'agit de l'IDE (*integrated development environment*) incluse dans le Python téléchargé sur www.python.org. Elle est simple et efficace.

1.3 Fenêtre **Shell**

Quand on lance IDLE PYTHON, la fenêtre Python **Shell** s'ouvre :

```
Python 3.13.5 (v3.13.5:6cb20a219a8, Jun 11 2025, 12:23:45) [Clang
16.0.0 (clang-1600.0.26.6)] on darwin
Enter "help" below or click "Help" above for more information.
>>> |
```

Derrière l'invite de commande >>> (prompt) se trouve le curseur | qui attend nos instructions : nous nous trouvons dans un environnement interactif. Dans les versions précédentes, le texte d'accueil nous invitait à taper `help`, `copyright`, `credits` ou `license()`. Le lecteur testera lui-même ces suggestions. Il testera également un petit calcul en ligne. Nous tapons « `salut` » et nous validons avec la touche ENTRER :

```
>>> salut
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
```

```
salut
NameError: name 'salut' is not defined
```

Python répond par un message d'erreur : pour lui, **salut** ne veut rien dire.

1.4 Versions

Pour connaître la version de Python utilisée par notre machine, on peut consulter la variable `version` du module `sys` :

```
>>> from sys import version
>>> version
'3.13.5 (v3.13.5:6cb20a219a8, Jun 11 2025, 12:23:45) [Clang 16.0.0
(clang-1600.0.26.6)]'
```

1.5 Éditer un script

Pour écrire un programme (un script) avec IDLE PYTHON :

Menu : File / New File `⌘N`

Une fenêtre d'édition de texte « **untitled** » s'ouvre dans laquelle on peut écrire un programme (un code, un script). On travaille dans cette fenêtre d'édition comme dans n'importe quel éditeur de texte (édition, sauvergarde, etc.)

Pour exécuter un programme depuis la fenêtre d'édition :

Menu : Run / Run Module `F5`

ou plus simplement la touche `F5`.

Les affichages et les saisies (entrées, *inputs*) se font dans la fenêtre **Python Shell**.

Pour ouvrir un ancien script depuis la fenêtre **Python Shell** :

Menu : File / Open... `⌘O`

ou :

Menu : File / Recent Files >

1.6 Recommandations

On recommande d'écrire des lignes de code de 79 caractères maximum (recommandation PEP 8 [1]). Si une ligne est trop longue, on peut la casser avec un `\`. Python comprend qu'il ne s'agit pas d'un saut de ligne :

```
print("Les chansons de Reeds sont accessibles sur \
https://www.youtube.com/@davidreeds-s2x")
```

Pour que cela fonctionne, il ne faut pas d'espace après le \. Cette méthode n'est pas recommandée (PEP 8 [1]), on lui préfère celle-ci :

```
print("Les chansons de Reeds sont accessibles sur "
      "https://www.youtube.com/@davidreeds-s2x")
```

(l'indentation ci-dessus n'est pas obligatoire mais recommandée).

On peut également casser une ligne entre deux items dans une séquence (liste, uplet, dictionnaire, etc.) :

```
conway = [1, 11, 21, 1211, 111221, 312211, 13112221,
          1113213211, 31131211131221, 13211311123113112211]
```

On peut écrire plusieurs instructions sur la même ligne en les séparant avec un point-virgule, mais c'est déconseillé (PEP 8) car cela nuit à la lisibilité du script :

```
a = input(); x = 2 * a; print(x)
```

1.7 Commentaires

Il est conseillé d'écrire des commentaires dans nos programmes. On utilise pour cela le symbole # :

```
âge = input("Ton âge : ")      # Saisie de l'âge
âge = float(âge)              # conversion en nombre
année = 2025 - âge            # Calcul de l'année de naissance
print(année)                  # affichage de l'année
```

1.8 Shebang et encodage

Typiquement, l'en-tête d'un programme ressemble à ceci :

```
#!/usr/bin/env python3
#-*- coding: utf-8 -*-
"""
Ici, je tape la docstring (documentation) de mon programme.
La documentation peut être aussi longue que l'on veut.
```

```
La docstring est recommandée.  
"""
```

La première ligne s'appelle *shebang* ou *hashbang*. Elle n'est pas obligatoire. Elle indique à un système d'exploitation UNIX, LINUX ou MAC OS qu'il faut appeler Python 3 pour interpréter le script, ce qui permet l'appel direct du programme en ligne de commande (depuis un terminal) sans avoir à écrire `python` devant. Elle est inutile si c'est un système WINDOWS qui exécute le script.

La deuxième ligne indique que le script est encodé selon la norme UTF-8 (voir chapitre 30). Elle n'est pas obligatoire. Cette ligne est de moins en moins utile car Python suppose par défaut que le script est encodé avec UTF-8 et la plupart des éditeurs utilisent cette norme. Si un jour on utilisait un éditeur fonctionnant avec une autre norme, on l'indiquerait dans l'entête.

1.9 Blocs et indentations

Le langage Python ne délimite pas les blocs d'instructions avec des balises `début/fin`, `begin/end` ou `{/}`. Les blocs sont tout simplement définis grâce à l'indentation. Python a choisi l'indentation pour améliorer la lisibilité des scripts.

La ligne qui précède un bloc d'instructions se termine toujours par un deux-points (`:`) comme ci-dessous :

```
def f(x):  
    if x == 0: return 0      #  
    else: return 1 / x      #  
n = 2  
if n > 0:  
    print("Ceci")  
    print("Cela")  
else:  
    print("Blabla")  
for x in ["Tony", "Brigi", "David", "Richard"]:  
    n = n + 1  
    print("Étape :", n)  
while n < 12:  
    n = n + 1  
    print(f(n))
```

Lorsqu'un bloc ne contient qu'une ligne, on peut l'écrire juste après le deux-points (sans aller à la ligne). C'est ce qu'on voit aux lignes marquées par un `#` dans l'exemple ci-dessus.

Cette série d'exemples n'est pas exhaustive : on aurait pu ajouter des `try`, `class`, `with`, `match`, etc.

On utilise l'instruction `pass` pour créer un bloc vide (un bloc qui ne fait rien) :

```
def f(x):
    pass          # à compléter plus tard
if x > 0:
    print("x est supérieur à 0")
else:
    pass          # à compléter plus tard
```

Attention. Dans un programme en langage Python, l'indentation (tabulation) fait partie du langage. Le non respect des règles d'indentation produit des messages d'erreur.

L'indentation est gérée automatiquement par IDLE (ou n'importe quel IDE servant à écrire du Python). Chaque indentation mesure 4 caractères (cadratin). On peut les générer avec la touche TAB. Le lecteur essaiera d'écrire le code ci-dessous :

```
"""
Étude d'une suite géométrique
"""
q = 0.3      # raison
u = 545      # premier terme u(0)
e = 0.01     # précision
n = 0
while abs(u) >= e:
    u = u * q
    n = n + 1
print("u(n) <", e, "à partir de n =", n)
```

1.10 Affichage dans le **Shell**

Quand on travaille sur le `shell` (en direct avec l'interpréteur), on n'a pas besoin de la fonction `print`. Au lieu d'écrire :

```
>>> print(2 + 3)
5
```

on se contente de :


```
>>> 2 + 3
5
```

Dans le `shell`, l'identifiant `_` (underscore) pointe vers la dernière donnée retournée :

```
>>> 2 * 3
6
>>> _ + 1
7
```

1.11 Utiliser un terminal

On peut lancer Python depuis un terminal. Sous Windows, on fait comme ceci :

```
C:\> python
```

(on peut aussi utiliser la commande `py`.) Sous MAC OS, c'est plutôt :

```
richard$ python3
```

car `python` tout court lançait Python 2 qui était installé par défaut jusqu'à récemment.

On se retrouve alors avec le même affichage que celui de la fenêtre `Shell` montré à la section 3. On dit que l'on se trouve dans l'environnement interactif de Python. Pour sortir de cet environnement, on appelle `exit` :

```
>>> exit()
```

On peut demander la version de Python utilisée depuis le terminal :

```
richard$ python3 --version
Python 3.13.5
```

On peut, depuis le terminal, exécuter une ligne de code Python sans entrer dans son environnement. Il suffit pour cela d'utiliser l'option `-c` :

```
richard$ python3 -c "print('Hello, World') ; print('Reeds')"
```

On peut faire en sorte qu'un programme Python se comporte comme une fonction depuis le terminal. Par exemple le programme `addition.py` ci-dessous :

```
from sys import argv
```

```
def somme(x, y): return x + y
x = float(argv[1])
y = float(argv[2])
print(somme(x, y))
```

est appelé de cette manière depuis le terminal :

```
richard$ python3 addition.py 23.65 100.1
```

On notera que

- tout argument de la commande `python3` est reçu sous forme de chaîne ;
- `argv[0]` reçoit le premier argument, c'est-à-dire `"addition.py"`.

Si on souhaite rester dans l'environnement de Python après l'exécution d'un programme, on utilise l'option `-i` :

```
richard$ python3 -i test.py
```

Si on fait du Python depuis le terminal, on peut modifier l'invite et les espaces d'indentation de l'environnement interactif. Normalement, on a ceci :

```
>>> for k in [0, 1, 2]:
...     print(k)
```

mais après modification :

```
>>> import sys
>>> sys.ps1, sys.ps2 = "Richard> ", " "
```

on obtient ceci :

```
Richard> for k in [0, 1, 2]:
          print(k)
```

1.12 Fonction `help`

Un appel `help(<objet>)` affiche une documentation sur l'objet passé en argument ou le type de cet objet, selon les cas. Un appel `help()` sans argument ouvre un environnement d'aide permettant de taper n'importe quel objet ou mot réservé. On invite le lecteur à utiliser cette fonction le plus possible :

```
>>> help(sum)
(...)
```


Chapitre 2

Mots réservés et objets internes

2.1 Mots réservés

Python possède 35 mots réservés :

```
>>> from keyword import kwlist
>>> kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await',
'break', 'class', 'continue', 'def', 'del', 'elif', 'else',
'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in',
'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return',
'try', 'while', 'with', 'yield']
```

On ne peut pas nommer un objet (une donnée) avec un mot réservé :

```
>>> for = 2300
SyntaxError: invalid syntax
```

2.2 Fonctions internes

Python possède 71 fonctions internes (*built-in functions*) :

abs	bin	callable	delattr
aiter	bool	chr	dict
all	breakpoint	classmethod	dir
anext	bytearray	compile	divmod
any	bytes	complex	
ascii			
enumerate	filter	getattr	hasattr
eval	float	globals	hash
exec	format		help

frozenset			hex
id	len	map	next
input	list	max	
int	locals	memoryview	
isinstance		min	
issubclass			
iter			
object	pow	range	set
oct	print	repr	setattr
open	property	reversed	slice
ord		round	sorted
			staticmethod
			str
			sum
			super
tuple	vars	zip	__import__
type			

Pour appeler une fonction, on utilise des parenthèses à l'intérieur desquelles on insère éventuellement des arguments. Appelons, par exemple, la fonction `help` sans argument :

```
>>> help()
Welcome to Python 3.13's help utility!
If this is your first time using Python, you should
definitely check out the tutorial on the Internet at
https://docs.python.org/3.13/tutorial/ (...)
help>
```

L'invite `help>` attend notre question. Le lecteur tapera `for` et ENTRER pour obtenir de l'aide sur les *boucles for*, par exemple. Pour quitter l'environnement *help*, on tape `quit` et ENTRER :

```
help> quit
>>>
```

Appelons, par exemple, la fonction `abs` avec un argument :

```
>>> abs(-56)
56
```

Notes. 1) La plupart des fonctions internes sont des objets de type *builtin function* or *method*, mais il y a des exceptions : certaines d'entre elles sont des objets de type *type*. C'est le cas, par exemple, de *bool*, *bytearray*, *bytes*, etc. Cela ne change rien au fait que dans la pratique, ces objets sont utilisés comme des fonctions. Notons que *help* est de type *Helper*.

2) Les objets *quit*, *exit*, *copyright*, *license* et *credits* ne sont pas catalogués comme fonctions internes du langage bien qu'ils soient utilisés comme des fonctions. On notera d'ailleurs, que ces objets ne sont même pas de type *function*. Le lecteur affichera *type(exit)*, par exemple. Ces objets sont importés automatiquement depuis le module *site* dès le lancement de Python. Ils ont vocation à être appelés depuis le *Shell*. Le lecteur les testera lui-même. Curieusement, on peut appeler *copyright* et *credits* sans utiliser de parenthèses.

2.3 Module **builtins**

Le programme ci-dessous importe le module *builtins* pour en extraire les noms des fonctions internes présentes dans Python :

```
import builtins
def appellable(chaine):
    attribut = getattr(builtins, chaine)
    return callable(attribut)
def sans_underscore(chaine): return chaine[0] != "_"
def sans_capitales(chaine): return chaine.islower()
NOMS = dir(builtins)
fonctions_internes = [k for k in NOMS if appellable(k) and
                      sans_underscore(k) and sans_capitales(k)]
```

Ce programme se base sur une astuce : grosso modo, les fonctions internes de Python sont les attributs de *builtins* dont le nom ne commence pas par un underscore et ne contient que des lettres minuscules.

On lance ce programme et on affiche la récolte :

```
>>> fonctions_internes
['abs', 'aiter', 'all', 'anext', 'any', 'ascii', 'bin', 'bool',
'breakpoint', 'bytearray', 'bytes', 'callable', 'chr', 'classme-
thod', 'compile', 'complex', 'copyright', 'credits', 'delattr',
'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec', 'exit',
'filter', 'float', 'format', 'frozenset', 'getattr', 'globals',
'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isins-
tance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals',
'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct',
'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'repr',
'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'static-
method', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
```

On retrouve toutes les fonctions internes (sauf `__import__`) ainsi que les objets `copyright`, `credits`, `exit`, `license` et `quit` ! Pour un programme plus élaboré, on consultera le chapitre 31 et [3].

Il aurait été difficile de trier les attributs de `builtins` à la main, sachant qu'il y en a 160 :

```
>>> len(dir(builtins))
160
```

Note. En vérité, il n'y a pas besoin d'importer `builtins`. Le nom `__builtins__` pointe déjà vers ce module. On recommande néanmoins d'utiliser `builtins` pour faire de l'introspection.

```
>>> builtins is __builtins__
True
```

2.4 Méthodes internes

Python est un *langage orienté objet* : toutes les données qu'il traite sont des *objets*. Chaque objet possède des *attributs*. Parmi les attributs, il y a des *méthodes*. On consultera éventuellement le chapitre 1 de [3] pour se familiariser avec ces notions.

Pour lister les attributs d'un objet `x`, on appelle `dir(x)`. Pour connaître les attributs d'une chaîne de caractères, par exemple, on peut regarder `dir("blabla")`, ou directement `dir(str)`, sachant que `str` est le type des chaînes de caractères.

```
>>> attributs = dir(str)
>>> len(attributs)
81
```

La plupart des attributs d'un objet sont des méthodes appelées *méthodes internes* car elles sont définies en interne par Python. Parmi elles, il y a des *méthodes spéciales* (leurs noms commencent et finissent par deux underscores) et des *méthodes ordinaires* (leurs noms sont écrits en lettres minuscules).

Quand on débute, on n'utilise que les méthodes internes ordinaires. Les méthodes spéciales sont utilisées par l'interpréteur en interne ou par nous-mêmes quand on fait de la programmation orientée objet (surcharge).

L'appel d'une méthode suit une syntaxe précise : pour appeler la méthode `maméthode` de l'objet `monobjet` en faisant passer des arguments `x`, `y` et `z`, par exemple, on écrit :

```
monobjet.maméthode(x, y, z)
```

Exemple :

```
"Alexandre Trijoulet".count("r")
2
```

Nous donnons ci-dessous le nombre de méthodes internes (spéciales ou ordinaires) pour chaque type usuel :

Type	Méthodes spéciales	Méthodes ordinaires
<code>bool</code>	61	7
<code>int</code>	61	7
<code>float</code>	50	5
<code>complex</code>	38	1
<code>str</code>	32	47
<code>range</code>	28	2
<code>bytes</code>	34	42
<code>bytearray</code>	37	50
<code>list</code>	34	11
<code>tuple</code>	31	2
<code>set</code>	37	17
<code>frozenset</code>	34	8
<code>dict</code>	32	11
<i>function</i>	24	0
<i>fichier binaire</i>	32	20
<i>fichier texte</i>	31	17

Toutes ces méthodes, à l'instar des fonctions internes, enrichissent considérablement le langage. Nous laissons au lecteur le soin d'écrire un programme lui permettant de recalculer ce tableau. Il notera que certaines méthodes apparaissent dans plusieurs types différents.

2.5 Méthodes spéciales

Tous les objets possèdent des méthodes spéciales. Le nom d'une méthode spéciale commence et termine toujours par deux underscores.

Derrière chaque opération se cache une méthode spéciale et c'est cette dernière qui est appelée en coulisse pour effectuer l'opération demandée.

Nous parlons ici de toutes sortes d'opérations : construction d'objet, initialisation d'objet, destruction, dérivation d'une classe, calcul de l'espace des noms, représentations formelle et informelle, formatage, comparaison, opération d'arité 1, opération d'arité 2, calcul de la valeur booléenne d'un objet, accès à la valeur associée à une clé ou un indice, test d'appartenance, calcul de longueur, inversion, itération, hachage, appel, gestion de contexte, accès à un attribut, accès à un descripteur, introspection, etc.

Les méthodes spéciales n'intéressent que les programmeurs désirant les surcharger dans des classes, elles ne concernent donc que la programmation orientée objet. Toutes ces notions sont étudiées en détail dans [3]. Nous nous contenterons ici de trois exemples. Quand l'interpréteur de Python tombe sur « `x + y` », il exécute en interne ceci :

```
x.__add__(y)
```

Quand l'interpréteur tombe sur « `x < y` », il exécute ceci :

```
x.__lt__(y)
```

Quand l'interpréteur tombe sur « `a = x[y]` », il exécute ceci :

```
a = x.__getitem__(y)
```

Notons que les méthodes spéciales n'ont rien à voir avec les méthodes fournies par le module `operator`. En effet, même si ce dernier fournit les méthodes spéciales sous formes de fonctions, on ne mélangera pas les usages. On utilise une fonction de `operator` dans une fonction de haut niveau, comme par exemple ci-dessous :

```
from operator import itemgetter
famille = [("Tony", 80), ("David", 52), ("Brigida", 77),
           ("Richard", 54)]
famille_ordonnée = sorted(famille, key=itemgetter(1))
```

Quand on envoie un objet `x` à la fonction `itemgetter(1)`, celle-ci retourne `x[1]`. Le programme ci-dessus crée une liste `famille_ordonnée` en classant les items de la liste `famille` par âge croissant, c'est-à-dire en ordonnant selon la valeur de `item[1]` pour chaque `item`. Dans ce contexte, on utilise une fonction de `operator` et non pas une méthode spéciale. L'exemple ci-dessous montre comment calculer le produit des items d'une liste avec la fonction `mul` de `operator` :

```
from operator import mul
from functools import reduce
liste = [2, 10, 5, 3]
produit = reduce(mul, liste)          # 2 * 10 * 5 * 3
```

Il existe de nombreuses fonctions de haut niveau : `map`, `filter`, `sorted`, `reduce`, etc.

2.6 Autres objets internes

Il y a d'autres objets internes comme `Ellipsis`, `NotImplemented`, `None`, etc. qui sont décrits en détail dans [3].

2.7 Introspection

Prenons une fonction interne quelconque, regardons comment Python la représente dans un affichage et examinons son type précis :

```
>>> abs
<built-in function abs>
>>> type(abs)
<class 'builtin_function_or_method'>
```

On constate que `abs` est de type *builtin function or method*. Ce n'est pas le cas de la fonction interne `float` :

```
>>> float
<class 'float'>
>>> type(float)
<class 'type'>
```

Examinons de la même manière une méthode interne ordinaire :

```
>>> "https://www.youtube.com/@davidreeds-s2x".index
<built-in method index of str object at 0x7fa94aea4330>
>>> type("https://www.youtube.com/@davidreeds-s2x".index)
<class 'builtin_function_or_method'>
```

Nous ne sommes pas étonnés de voir qu'une méthode interne ordinaire est de type *builtin function or method*. Examinons une méthode spéciale :

```
>>> x = 6
>>> x.__add__
<method-wrapper '__add__' of int object at 0x101173b60>
>>> type(x.__add__)
<class 'method-wrapper'>
>>> int.__add__
<slot wrapper '__add__' of 'int' objects>
>>> type(int.__add__)
<class 'wrapper_descriptor'>
```

On constate que `add` est un *wrapper descriptor* de `int` alors que la méthode `add` liée à un entier est un *method wrapper*.

Chapitre 3

Types offerts par Python

3.1 Objets

Chez Python, tout est objet. Toutes les données manipulées sont des objets. Les nombres, les chaînes, les listes, les fonctions et les exceptions (erreurs) sont des objets. Même le type d'un objet est un objet !

Tout objet possède une valeur, un type, des attributs, une identité et zéro ou plusieurs noms. Voici un exemple :

```
>>> liste = [1, 2, 3]
>>> id(liste)
140309299644288
>>> u = (0, 1, liste)
```

Nous venons de créer l'objet `<140309299644288>` et :

- la valeur de `<140309299644288>` est `[1, 2, 3]` ;
- le type de `<140309299644288>` est `list` ;
- les attributs de `<140309299644288>` sont `append`, `clear`, `copy`, etc.
- les identificateurs (noms) de `<140309299644288>` sont `liste` et `u[2]`.

Tout objet, quel que soit son niveau d'abstraction, peut être affiché grâce à la fonction `print`.

On notera que `print(x)` affiche la chaîne de caractère retournée par `str(x)`, tandis que « `x` suivi de ENTRER » dans le `Shell`, affiche la chaîne retournée par `repr(x)`, voir sections 1.10, 19.6 et 19.10 :

```
>>> liste
[1, 2, 3]
```

3.2 Types usuels

1. Objet de type `NoneType` (le rien)

```
>>> None
>>> print(None)
None
>>> type(None)
<class 'NoneType'>
```

2. Objets de type bool (booléens)

```
>>> True
True
>>> 2 > 5
False
>>> type(True)
<class 'bool'>
```

3. Objets de type int (entiers)

```
>>> -56
-56
>>> type(-56)
<class 'int'>
```

4. Objets de type float (nombres décimaux)

```
>>> 5.63
5.63
>>> type(5.63)
<class 'float'>
```

5. Objets de type complex (nombres complexes)

```
>>> 5 + 3j
(5+3j)
>>> type(5 + 3j)
<class 'complex'>
```

6. Objets de type str (chaînes de caractères)

```
>>> "Je suis content"
'Je suis content'
>>> print("Je suis content")
Je suis content
>>> type("Je suis content")
<class 'str'>
```

7. Objets de type `bytes` (chaînes d'octets)

```
>>> b"AaAaAAaBB"  
b'AaAaAAaBB'  
>>> type(b"AaAaAAaBB")  
<class 'bytes'>
```

8. Objets de type `bytearray` (tableau d'octets)

```
>>> bytearray(b"AaAaAAaBB")  
bytearray(b'AaAaAAaBB')  
>>> type(bytearray(b"AaAaAAaBB"))  
<class 'bytearray'>
```

9. Objets de type `range` (plages d'entiers)

```
>>> range(5, 10, 2)  
range(5, 10, 2)  
>>> type(range(5, 10, 2))  
<class 'range'>
```

10. Objets de type `tuple` (uplets)

```
>>> (4, 0, 10, "a")  
(4, 0, 10, 'a')  
>>> type((4, 0, 10, "a"))  
<class 'tuple'>
```

10. Objets de type `list` (listes)

```
>>> [4, 0, 10, "a"]  
[4, 0, 10, 'a']  
>>> type([4, 0, 10, 'a'])  
<class 'list'>
```

11. Objets de type `set` (ensembles)

```
>>> {4, 0, 10, "a"}  
{0, 10, 4, 'a'}  
>>> type({4, 0, 10, "a"})  
<class 'set'>
```

12. Objets de type `frozenset` (ensembles)

```
>>> frozenset({"Pedro", "Pilar"})
frozenset({'Pedro', 'Pilar'})
>>> type(frozenset({"Pedro", "Pilar"}))
<class 'frozenset'>
```

13. Objets de type `dict` (dictionnaires)

```
>>> {"Ivan": 74, "David": 73, "Richard": 71}
{'Ivan': 74, 'David': 73, 'Richard': 71}
>>> type({"Ivan": 74, "David": 73, "Richard": 71})
<class 'dict'>
```

14. Objets de type *function* (fonctions)

```
>>> def f(x, y): return x**2 + y**2
>>> f
<function f at 0x7f9c4dfc94c0>
>>> type(f)
<class 'function'>
>>> lambda x, y: x**2 + y**2
<function <lambda> at 0x7f9c4df350d0>
>>> type(lambda x, y: x**2 + y**2)
<class 'function'>
```

15. Objets de type *fichier texte*

```
>>> fichier = open("riri", "w")
>>> fichier.close()
>>> fichier
<_io.TextIOWrapper name='riri' mode='w' encoding='UTF-8'>
>>> type(fichier)
<class '_io.TextIOWrapper'>
```

16. Objets de type *fichier binaire*

```
>>> fichier = open("riri", "rb")
>>> fichier.close()
>>> fichier
<_io.BufferedReader name='riri'>
>>> type(fichier)
<class '_io.BufferedReader'>          # fichier bin-lecture
>>> fichier = open("riri", "wb")
>>> fichier.close()
```