PROGRAMMATION PYTHON AVANCÉE

Chez le même éditeur

Python 3
2e édition
Bob Cordeau, Laurent Pointal
304 pages
Dunod, 2025

Python pour le data scientist
3e édition
Emmanuel Jakobowicz
336 pages
Dunod, 2024

Python précis et concis
5e édition
Mark Lutz
272 pages
Dunod, 2019

XAVIER OLIVE



PROGRAMMATION PYTHON AVANCÉE

GUIDE POUR UNE PRATIQUE ÉLÉGANTE ET EFFICACE

DUNOD



Le code source des exemples avancés présentés au cours des chapitres ainsi que des ressources complémentaires sont librement accessibles en ligne sur le site de l'auteur : https://www.xoolive.org/python/

© Dunod, 2025
11 rue Paul Bert, 92240 Malakoff
www.dunod.com
ISBN 978-2-10-088187-1



Préface à la première édition

Lorsque Xavier Olive m'a contacté pour me proposer de relire ce livre que vous tenez actuellement entre les mains, ma première réaction a été « Oh non, encore un livre sur Python ! » Un certain nombre de livres sur Python trônent déjà sur les étagères de mon bureau, même s'il est vrai que la plupart sont en anglais. Mais lors de ce premier échange par mail, Xavier a glissé subrepticement le fait qu'il avait pour but d'*écrire le livre qu'il aurait voulu lire*. Cela a eu pour effet immédiat d'attiser ma curiosité et j'ai donc accepté de relire cet ouvrage sans trop savoir où je mettais les pieds. Après avoir relu les quelque 350 pages qui composent ce livre, je réalise que c'était une très bonne décision de ma part tant le livre est agréable à lire, tant sur la forme que sur le fond.

Sur la forme d'abord, car Xavier a véritablement soigné le style du livre qui tranche avec un certain nombre d'ouvrages que j'ai pu lire par le passé. Ayant une solide expérience quant à la typographie et à la mise en page, je suis admiratif du soin et du souci du détail qui ont été apportés à l'ouvrage. Trop souvent les auteurs négligent cet aspect en se disant que seul le fond importe pour un ouvrage technique alors que la forme peut véritablement jouer un rôle essentiel pour la compréhension de concepts parfois ardus. La relecture et les échanges avec Xavier sur ces aspects ont été d'autant plus agréables qu'il en a une parfaite maîtrise.

Mais la prouesse du livre se trouve bien évidemment sur le fond. Ayant moi-même une assez grande expérience de Python, notamment sur ses versants scientifiques, je reste admiratif de cet ouvrage qui est à la fois bien écrit, bien structuré, bien documenté et surtout extrêmement pédagogique dans son approche. Lors de ma relecture du premier chapitre, je me suis d'abord fait la réflexion qu'il allait un peu vite en besogne en présentant les bases du langage Python, avant de me rappeler qu'il s'agissait d'un ouvrage avancé venant compléter celui de Bob Cordeau et Laurent Pointal qui s'adresse, lui, aux débutants. Or, présenter les bases du langage Python à des utilisateurs avancés est un vrai numéro d'équilibriste. Mais je crois que Xavier a su justement trouver le bon équilibre en choisissant méticuleusement les aspects peut-être moins connus du langage et en les illustrant à l'aide d'exemples pertinents (et pour certains passionnants au niveau théorique, comme les L-systèmes).

Le livre est structuré autour de cinq grandes parties (Les bases du langage Python, L'écosystème Python, Écrire un Python naturel et efficace, Python, couteau suisse du quotidien, Développer un projet en Python) et de trois interludes (Calcul du rayon de la Terre, Reconstruire une carte d'Europe, La démodulation de signaux FM) qui viennent agréablement aérer la technicité de certains chapitres. Étant avant tout un livre pour des utilisateurs avancés, il est

évident que Xavier ne pouvait éviter d'être technique sur certains des aspects les plus avancés de Python. J'avoue ne pas être un grand fan des dernières possibilités offertes par le langage Python car je crois que cela alourdit inutilement le langage mais Xavier a su malgré tout me convaincre de l'utilité de la majorité d'entre elles, notamment lorsque l'on se retrouve à gérer de gros projets collaboratifs.

Je suis donc à la fois très honoré et très heureux d'écrire aujourd'hui cette préface pour un livre qui, je le crois, deviendra un classique. Évidemment, ce n'est que la première édition et, au vu de la rapidité d'évolution du langage Python, je crois aussi que Xavier s'est engagé, sans peut-être le savoir, à écrire une nouvelle édition tous les deux ou trois ans. C'est tout le mal que je lui souhaite. D'ailleurs, à l'heure où j'écris ces lignes (février 2021), le PEP 636 concernant l'identification structurelle de motifs (*Structural Pattern Matching* en anglais) vient d'être accepté, validant ainsi encore un peu plus la 10e règle de Greenspun [1].

Nicolas P. Rougier

Docteur en informatique et chercheur
à l'Inria en neurosciences computationnelles

Février 2021

1. https://en.wikipedia.org/wiki/Greenspun's_tenth_rule



Table des matières

Préface à la première édition v

Avant-propos ix

Introduction à la deuxième édition 1

I Les bases du langage Python 3

1 Types et arithmétique de base 5

2 La bibliothèque Python standard 23

3 La gestion des fichiers 37

4 Structures de données avancées 49

Interlude : Calcul du rayon de la Terre 61

II L'écosystème Python 67

5 Le calcul numérique avec NumPy 69

6 Produire des graphiques avec Matplotlib 83

7 La boîte à outils scientifiques SciPy 99

8 L'environnement interactif Jupyter 107

Interlude : Reconstruire une carte d'Europe 113

9 L'analyse de données avec Pandas 119

10 La visualisation interactive avec Altair et ipyleaflet 137

11 L'analyse de tableaux multidimensionnels avec Xarray 155

III Écrire un Python naturel et efficace 163

12 La programmation fonctionnelle 165

13 Décorateurs de fonctions et fermetures 181

14 Itérateurs, générateurs et coroutines 195

15 La programmation orientée objet 211

16 Interfaces et protocoles 235

17 L'ABC de la métaprogrammation 251

18 La programmation concurrente 269

19 La programmation asynchrone 279

Interlude : La démodulation de signaux FM 287

IV Python, couteau suisse du quotidien 297

20 Comment manipuler des formats de fichiers courants ? 299

21 Comment interroger et construire des services web ? 309

22 Comment écrire un outil graphique ou en ligne de commande ? 321

23 Comment exécuter du code Python dans un navigateur web ? 333

V Développer un projet en Python 341

24 Manipuler des environnements Python 343

25 Publier une bibliothèque Python 349

26 Mettre en place un environnement de tests 357

27 Annotations et typage statique 367

28 Optimiser du code Python 381

Pour aller plus loin 393

Index 397



Avant-propos

Python est un langage généraliste et multi-plateforme, développé suivant un modèle open source depuis le début des années 1990 : sa première version vient de fêter ses 30 ans ! C'est un langage interprété, populaire pour sa facilité d'utilisation, pour sa polyvalence, et pour les apports de sa communauté. Python est apprécié parmi les scientifiques et les ingénieurs d'horizons divers.

Dans certaines spécialités, d'autres langages peuvent être plus rapides, plus sûrs ou plus complets, mais Python se démarque par sa polyvalence, par la concision et la lisibilité de sa syntaxe, par la facilité avec laquelle on peut écrire un prototype en quelques heures, construire une chaîne de traitements à partir de briques logicielles écrites par d'autres, parfois dans d'autres langages. Enfin, il reste une solution de choix pour outiller des tâches informatiques simples de notre quotidien.

À qui s'adresse ce livre ?

Ce livre s'adresse à un public qui a déjà une bonne expérience de la programmation, que celle-ci soit avec Python ou non. L'ouvrage propose différentes grilles de lecture, avec un contenu théorique de base et des chapitres complémentaires, adaptés à une deuxième lecture. Ceux-ci seront une opportunité de mettre en pratique les concepts et les outils sur des exemples engageants. L'objectif est de présenter au lecteur un ouvrage qui rappelle les concepts-clés pour une utilisation idiomatique du langage et qui les illustre dans des cadres d'utilisation variés.

Si vous débutez en programmation et souhaitez apprendre Python, ce livre sera difficile à suivre. Les structures de données sont reprises en détail, mais la syntaxe du langage et les fondements de la programmation ne sont pas traités. L'ouvrage *Python 3*, de Bob Cordeau et Laurent Pointal aux éditions Dunod, est plus adapté pour s'initier au langage, apprendre des notions élémentaires (boucles, valeurs, expressions, variables, etc.) et découvrir la syntaxe.

Comment est construit ce livre ?

Le contenu ne se limite pas aux seuls aspects proposés par le langage avec ses apports les plus récents mais expose une approche de l'écosystème Python dans son ensemble, avec une

présentation des principales bibliothèques tierces développées par la communauté, devenues aujourd'hui incontournables. Nous abordons aussi les pratiques recommandées de gestion de projet logiciel en Python.

Ce livre s'appuie sur les bases de l'algorithmique et de la programmation, il présente comment des concepts génériques de programmation non spécifiques au langage sont déclinés en Python. Il aide à appréhender le vocabulaire et les mots-clés propres au langage pour rechercher en ligne de manière autonome et efficace les réponses aux problématiques fréquemment rencontrées lors de l'écriture de code.

Les exemples d'application présentés dans cet ouvrage s'appuient sur des rudiments de culture générale relatifs à des domaines variés tels le calcul numérique, le traitement du signal ou l'intelligence artificielle pour les illustrer et les mettre en évidence de manière naturelle et élégante à l'aide du langage Python.

Ce livre se décompose en cinq grandes parties :

— **Les bases du langage Python.** Cette partie reprend les bases du langage en se concentrant sur les structures de données, avec leurs atouts et leurs limitations. De nombreuses structures avancées sont fournies par le langage ; elles permettent de s'attaquer efficacement à des problèmes difficiles.

— **L'écosystème Python.** Python ne se limite pas à ses fonctionnalités et à ses bibliothèques intégrées déjà bien fournies. C'est également une communauté : certaines bibliothèques écrites par des développeurs indépendants et des laboratoires scientifiques sont devenues incontournables.

— **Écrire un Python naturel et efficace.** Un bon programme Python n'est pas seulement un programme qui fonctionne. C'est un code qui suit les conventions de la communauté et qui utilise le langage comme il a été pensé. Cette partie présente comment exploiter les caractéristiques du langage pour écrire un code qui est clair, concis et facile à maintenir.

— **Python, couteau suisse du quotidien.** Python est un langage adapté pour outiller des tâches du quotidien. Cette partie guide le lecteur pour une utilisation du langage orientée vers la manipulation des fichiers standard (images, CSV, Excel, XML, PDF, JSON et plus) et l'interaction avec des services web ouverts. La production d'outils graphiques et en ligne de commande est également abordée.

— **Développer un projet en Python.** Le développement d'un projet Python qui prend de l'ampleur se prépare et se sécurise à l'aide d'un certain nombre de pratiques standard : intégration continue, environnements virtuels, suivi de la performance et de la non-régression. Cette partie reprend les différents aspects de la gestion logicielle et présente des outils standards, couramment utilisés dans la plupart des projets logiciels.

Les exemples de code

Tous les chapitres de cet ouvrage contiennent du code source. Les commandes à entrer dans un terminal (bash, zsh, etc.), un PowerShell Windows ou une invite de commande Anaconda sont préfixées par `$` :

```
$ uv pip install .
```

Les exemples courts démarrent par >>> pour refléter l'invite de l'interpréteur Python. Les retours sont alors affichés à la ligne.

```
>>> import math
>>> math.pi
3.141592653589793
```

Les exemples plus longs ne sont pas exécutables tels quels.

```
import numpy as np  # ①

...
```

Tout fichier ou toute partie d'un fichier Python est imprimé dans le livre et annoté à l'aide de balises numérotées ①, ②, etc. Les fichiers complets sont disponibles sur le site https://www.xoolive.org/python/. Tous les exemples ont été testés pour Python 3.12.

La première partie du livre sur les bases du langage ne nécessite pas de bibliothèques particulières : n'importe quel interpréteur Python fera l'affaire.

Dans les parties suivantes, il est recommandé de cloner le dépôt du livre depuis le lien https://github.com/xoolive/python ①, d'installer l'outil pixi (☞ p. 348, § **24.6**), puis de créer un environnement dédié. Lors de la première exécution de la commande pixi ②, l'environnement Python et ses dépendances seront installées.

```
$ git clone https://github.com/xoolive/python # ①
$ pixi run python  # ②
Python 3.12.5 | packaged by conda-forge | (main, Aug  8 2024, 18:32:50) [Clang 16.0.6 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Conventions utilisées dans l'ouvrage

Les conventions suivantes sont utilisées au long de l'ouvrage :

— le texte *en italique* retranscrit les termes anglais équivalents au vocabulaire utilisé en français, par exemple « bibliothèque (*library* en anglais) » ;

— le texte à `chasse fixe` retranscrit notamment des noms de variables, de modules et des scripts Python.

Attention !

Cet encart met en valeur les erreurs et pièges courants dans lesquels le programmeur, même averti, peut tomber.

Bonnes pratiques

Cet encart met en valeur les bonnes pratiques, couramment admises, qui améliorent la qualité et la lisibilité du code.

Cas d'application. Cet encart met en valeur un exemple appliqué au milieu d'un chapitre théorique.

En quelques mots...

Cet encart conclut un chapitre en rappelant les éléments essentiels à retenir pour comprendre les chapitres suivants.

Page web du livre

Le code source et les illustrations du livre, qui ont été générées avec Python, sont disponibles sur la page web du livre, avec un erratum qui recense les coquilles relevées après l'impression de l'ouvrage : `https://www.xoolive.org/python/`. Il est possible depuis le site de poser des questions, d'ouvrir des discussions et de proposer des corrections aux éventuelles coquilles qui se seraient glissées dans ces lignes.

Remerciements

Ce projet n'aurait pas vu le jour si Bob Cordeau ne m'avait pas présenté Jean-Luc Blanc des éditions Dunod, juste après la soumission du manuscrit de la deuxième édition de son ouvrage *Python 3. Apprendre à programmer dans l'écosystème Python*. Je tiens à les remercier tous les deux, notamment M. Blanc pour la confiance qu'il m'a accordée pour la définition de ce nouveau projet et pour la souplesse avec laquelle il m'a accompagné.

La rédaction d'un tel ouvrage est un moment intense, un marathon dans lequel sont embarqués malgré eux épouse, enfants, proches et collègues. Tous ont participé d'une manière ou d'une autre à ce travail, en faisant face à une indisponibilité qu'ils n'avaient pas choisie, en apportant leur soutien, leur point de vue critique après relectures, leur réponse pendant les moments de doute et leurs encouragements. Cet ouvrage leur est dédié.

Je remercie notamment Luis Basora et Thomas Dubot, fidèles soutiens et relecteurs de la première heure, ainsi que Judicaël Bedouet pour sa gentillesse et ses métacommentaires avisés. Junzi Sun et Enrico Spinielli ont également contribué à cet ouvrage sans vraiment le savoir, le fruit de nos riches échanges transparaît dans de nombreux exemples.

Nicolas Rougier, auteur de nombreuses références Python de qualité, a répondu à mon invitation, a accepté de relire l'intégralité de l'ouvrage puis m'a fait l'honneur de le préfacer. Brice Martin, des éditions Dunod, a également été d'une grande aide pour la préparation du manuscrit. Leurs commentaires ont contribué à améliorer la lisibilité de l'ensemble.

Il faut enfin saluer tous les étudiants qui, par leurs progrès, leurs doutes, leurs interrogations et reformulations, ont contribué à améliorer les ressources pédagogiques préliminaires sur lesquelles sont construites les fondations de ce livre.



Introduction à la deuxième édition

L'apprentissage d'un langage de programmation est un processus progressif. Comme pour une langue vivante, il est tout à fait possible, et même recommandé, de pratiquer un langage même sans en connaître toutes les subtilités. C'est par la pratique qu'on découvre de nouvelles problématiques, des solutions pour y répondre ; puis vient la réflexion pour généraliser ces solutions à des classes générales de problèmes. Un langage vivant est capable de faire vivre de telles réflexions pour proposer des améliorations de celui-ci.

Ce rôle est joué en Python par les PEP, les *Python Enhancement Proposals*, ces discussions levées par la communauté avec des propositions d'améliorations du langage. Certaines améliorations sont controversées, mais les décisions sont toujours prises après échanges d'arguments. Ces améliorations font évoluer le langage et les pratiques mois après mois. Le Python de 1989 n'est pas le même que le Python de 2003, qui n'est pas le même que le Python de 2020.

Ces pages ont pour vocation d'explorer cet environnement changeant du langage, de son écosystème, construit autour des structures de données efficaces de la bibliothèque NumPy (☞ p. 69, § **5**) et des technologies du web vers lesquelles la communauté se tourne progressivement (☞ p. 107, § **8** ; ☞ p. 137, § **10**), ainsi que des fondements théoriques sur lesquels il s'est construit. Elles guident le lecteur dans le monde des bibliothèques tierces de référence conçues pour faciliter la réalisation de tâches élémentaires : télécharger des données (☞ p. 309, § **21**), extraire des informations d'un document (☞ p. 299, § **20**), les structurer, les transformer ou les visualiser, le tout dans l'objectif d'y apporter une valeur ajoutée.

Le livre se termine sur la conception d'un projet Python (☞ p. 349, § **25**). Les outils évoluent de jour en jour, il n'a jamais été aussi facile qu'en 2020 de partager du code tout en s'assurant qu'il pourra être exécuté sur une nouvelle plateforme, sur un nouveau système d'exploitation, en embarquant les bibliothèques nécessaires à son bon fonctionnement.

La première version de cet ouvrage était basée sur Python 3.9, qui avait déjà vu l'introduction de plusieurs innovations majeures :

— les mots-clés pour la programmation asynchrone,
— l'opérateur de multiplication matricielle,
— les annotations de types (PEP 526),
— les chaînes de caractères formatées, ou *f-strings*, (PEP 498, ☞ p. 9, § **1.3**)),
— le séparateur de milliers avec le souligné dans les entiers,
— les dataclasses (PEP 557 , ☞ p. 50, § **4.2**),

— le *walrus operator* := du PEP 572 au terme duquel le créateur du langage Guido van Rossum a quitté son poste de *Benevolent Dictator For Life* (BDFL) ;
— les types standards génériques (PEP 585, ☞ p. 367, § **27**).

Cet ouvrage doit évoluer avec l'écosystème Python. Le manuscrit de cette nouvelle édition a été finalisé juste avant la sortie de Python 3.13. Les principaux ajouts de cette deuxième édition incluent :

— La mise à jour vers NumPy 2.0 (☞ p. 80, § **5.7**) ;
— La mise à jour vers Pandas 2.0 (☞ p. 133, § **9.6**), ainsi qu'un aperçu sur la bibliothèque Polars (☞ p. 133, § **9.7**) ;
— Une présentation de la bibliothèque Xarray (☞ p. 155, § **11**) ;
— Le *structural pattern matching* (☞ p. 166, § **12.1**), un principe qui vient de la programmation fonctionnelle (☞ p. 165, § **12**).
— Les nouveautés concernant la gestion de la concurrence (☞ p. 269, § **18**).
— Un chapitre dédié à la programmation asynchrone (☞ p. 279, § **19**), et l'introduction de nouvelles bibliothèques Python adaptées à ce paradigme :
  — Requests est remplacé par HTTPX (☞ p. 309, § **21.1**) ;
  — Flask est remplacé par FastAPI (☞ p. 312, § **21.2**) ;
  — la gestion des bases de données (y compris DuckDB ☞ p. 317, § **21.3.1**) est présentée avec SQLAlchemy (☞ p. 317, § **21.3.2**) ;
  — Dans le terminal on présente désormais les bibliothèques Rich (☞ p. 321, § **22.1**) et Textual (☞ p. 325, § **22.3**).
— La possibilité d'exécuter Python directement dans un navigateur web (☞ p. 333, § **23**) ;
— La gestion d'un projet Python mise à jour avec les outils Ruff (☞ p. 349, § **25**) et uv (☞ p. 345, § **24.4**) ;
— Les dernières évolutions de la syntaxe de typage statique (☞ p. 367, § **27**) ;
— L'intégration de Rust dans le développement Python (☞ p. 389, § **28.3**) ;
— Quelques remarques sur les bonnes pratiques de code en fin d'ouvrage.

Python reste un outil, pas une fin en soi. Comme le dit A. Maslow, *I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail*, « J'imagine qu'il est tentant, si le seul outil dont vous disposez est un marteau, de tout considérer comme un clou. »

Ces lignes visitent parfois des contrées atypiques, des manières de programmer qui peuvent sembler déroutantes. L'objectif de cet ouvrage est d'apporter des éléments qui permettent de clarifier des éléments de syntaxe et de conception couramment utilisés dans des grosses bases logicielles. Attention cependant par la suite à bien distinguer « bonne pratique » d'« usage inutilement complexe ».

Keep it simple.

I

Les bases du langage Python

1

Types et arithmétique de base

L'objectif de ce chapitre est de redécouvrir les types de base de Python : nombres (entiers, flottants, complexes), chaînes de caractères, structures conteneurs (tuples, listes, ensembles, dictionnaires), fonctions et exceptions. Nous nous concentrons ici sur les particularités du langage et les singularités de chacune des structures présentées. La documentation officielle[1] reste la référence pour une description exhaustive des possibilités des structures conteneurs de base.

1.1. Les entiers

Les entiers (type `int` en Python) sont munis des quatre opérations arithmétiques habituelles : l'addition `+`, la soustraction `-` et la multiplication `*`. On fait une distinction entre la division flottante `/` et la division entière `//` :

```
>>> 7 / 3  # division flottante
2.3333333333333335
```

```
>>> 7 // 3  # division entière
2
```

On peut manipuler les entiers à partir de leur représentation binaire (préfixe `0b`), octale (préfixe `0o`), décimale ou hexadécimale (préfixe `0x`), mais la représentation par défaut est en base 10. Des opérateurs renvoient une représentation en base 2, 8 ou 16 sous forme de chaîne de caractères. À l'inverse, on peut lire un entier dans n'importe quelle base à partir d'une chaîne de caractères : il suffit de la préciser en paramètre.

```
>>> 0b1111111
127
>>> 0o177
127
>>> 0x7f
127
```

```
>>> bin(127)
'0b1111111'
>>> oct(127)
'0o177'
>>> hex(127)
'0x7f'
```

```
>>> int("0b1111111", 2)
127
>>> int("0o177", 8)
127
>>> int("0x7f", 16)
127
```

Suivant le modulo (le reste de la division entière) à calculer, il peut être plus efficace de faire appel à une opération bit à bit qui s'opère sur la représentation binaire des entiers : par exemple, l'entier 3 s'écrit 11 en binaire, l'opération `& 3`, en bit à bit, filtre les deux derniers bits

1. https://docs.python.org/fr/3/tutorial/datastructures.html

de la représentation de chaque entier et correspond au modulo 4 (100 en binaire). L'opérateur modulo « % » implique de calculer une division flottante, une partie entière, une multiplication et une soustraction (4 opérations), alors que le & bit à bit ne nécessite qu'une seule opération. Cette astuce est valide pour tous les modulos par une puissance de 2.

```
>>> 7 % 4  # modulo
3
```

```
>>> 7 & 3  # opération "et bit-à-bit"
3
```

De la même manière, les opérations de décalage de bit (*bitshift*) vers la gauche << ou vers la droite >> peuvent être plus efficaces que des calculs de puissance de 2. Le passage à la puissance implique un grand nombre de multiplications alors que le décalage de bit est une opération unitaire du point de vue du processeur.

```
>>> 2 ** 8  # puissance
256
```

```
>>> 1 << 8  # décalage de bits
256
```

Les entiers en Python ont une amplitude illimitée. C'est un atout de taille par rapport à d'autres langages de programmation classiques (basés sur le langage C) qui s'appuient sur les spécificités du processeur et du système d'exploitation pour représenter les entiers sur un nombre défini d'octets, sans qu'aucun standard ne permette d'inférer quoi que ce soit. En pratique, Python propose les opérations classiques sur les longs entiers en mobilisant autant d'espace mémoire que nécessaire pour les représenter.[2]

Il n'y a pas de standard pour la taille ou la représentation des entiers dans les langages habituels, qui peuvent suivant les architectures occuper 32, 64 ou 128 bits. Les nombres de l'exemple suivant ne pourraient pas être manipulés par le langage C par exemple :

```
>>> 2 ** 128  # le plus grand entier non signé sur 128 bits + 1
340282366920938463463374607431768211456
>>> 123 ** 24  # cet entier s'écrit sur 167 bits (pas de représentation en C)
1437880104467752488482378752031633364946536 2343841
>>> 123 ** 24 * 134 ** 45  # l'opération est très rapide!
754116773913301291674484428969148011550171822575090416487017684057230784745 92
380051352543980177649477418579845319891270034417439450350881010089984
```

1.2. Les flottants

Les flottants (type `float`) sont notés en Python avec un point (`1.` sans décimale, `2.0` avec une décimale explicite, `.3` sans partie entière ou `3.14` pour des flottants classiques) ou en utilisant la notation scientifique (`3e8` par exemple). Les flottants sont représentés en mémoire suivant le standard IEEE 754, avec un bit de signe, 11 bits d'information pour les exposants et 52 bits d'information pour la mantisse (pour les flottants 64 bits). Les opérations sur les flottants sont soumis aux mêmes effets qu'avec les autres langages de programmation.

Le standard définit quelques flottants particuliers : l'infini, appelé `inf`, (aucun flottant n'est supérieur à l'infini, pas même l'infini) et le résultat d'une opération invalide, appelé `nan` pour *not a number* (pas un nombre). Aucun nombre n'est égal à `nan`, pas même `nan` : pour tester si un nombre vaut la valeur `nan`, il est courant de tester s'il est égal à lui-même.

2. La version 2.3 de Python a introduit pour la première fois la multiplication de Karatsuba pour les grands entiers. D'autres méthodes rapides basées sur la transformée de Fourier rapide (FFT, pour *Fast Fourier Transform*), de l'algorithme de Schönhage-Strassen (SSA) ou celui de De, Kurur, Saha and Saptharishi (DKSS) sont accessibles pour la multiplication des nombres décimaux (☞ p. 27, § 2.2).

```
>>> float('inf') > float('inf')
False
>>> float('inf') == float('inf')
True
>>> float('inf') - float('inf')
nan
```

```
>>> float('nan') == float('nan')
False
>>> a = float('nan')
>>> if a != a:  # math.isnan(a)
...     print("a is NaN")
a is NaN
```

✧ Bonnes pratiques

Ne pas faire de test d'égalité entre deux flottants : préférer un test d'inégalité sur leur différence :

```
>>> 0.1 + 0.2 == 0.3
False
```

```
>>> abs(0.3 - (0.1 + 0.2)) < 1e-12
True
```

L'artefact ci-dessus s'explique par le fait que le résultat de l'addition `0.1 + 0.2` sur les flottants n'a pas la même représentation binaire que la valeur `0.3`. Il est possible d'accéder en Python à la représentation binaire (ou hexadécimale) des flottants *via* le module `float` : `0.1`, `0.2` et `0.3` sont représentés sous forme de série hexadécimale infinie et le résultat de l'addition est alors soumis à une erreur d'arrondi qui biaise le test d'égalité.

```
>>> 0.1 + 0.2
0.30000000000000004
>>> float.hex(0.1)  # 0.0999999...
'0x1.999999999999ap-4'
>>> float.hex(0.2)  # 0.1999999...
'0x1.999999999999ap-3'
```

```
>>> float.hex(0.1 + 0.2)
'0x1.3333333333334p-2'
# arrondi supérieur: finit par un 4
>>> float.hex(0.3)
'0x1.3333333333333p-2'
# arrondi inférieur: finit par un 3
```

Python permet par ailleurs d'afficher une chaîne de caractères (☞ p. 8, § **1.3**) représentant la valeur des flottants manipulés avec une précision suffisante pour voir l'effet de ces artefacts :

```
>>> print("{0:.32f}".format(0.1))
0.10000000000000000555111512312578
>>> print("{0:.32f}".format(0.2))
0.20000000000000001110223024625157
```

```
>>> print("{0:.32f}".format(0.3))
0.29999999999999998889776975374843
```

Pour pallier ces problèmes potentiels, les modules `fractions` et `decimal` (☞ p. 27, § **2.2**) en Python permettent de manipuler des nombres décimaux avec une précision potentiellement infinie sans subir les effets d'arrondi sur la représentation des nombres flottants. En revanche, ces modules coupent la portabilité avec les autres langages de programmation.

Enfin l'arithmétique des nombres complexes (type `complex`) est accessible : la partie imaginaire d'un complexe s'exprime à l'aide du suffixe `j` :

```
>>> 1j
1j
>>> 1j * 1j
(-1+0j)
>>> cmath.sqrt(-1)
1j
```

```
>>> a = 3 + 4j
>>> a.real, a.imag
(3.0, 4.0)
>>> a * a.conjugate()
(25+0j)
```

Attention !

Ne pas confondre le symbole `j`, qui peut représenter une variable, du suffixe `j` qui s'adjoint à la fin d'une valeur numérique entière ou flottante :

```
>>> j
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'j' is not defined
```

Historique. La question du choix de la lettre `j`, couramment utilisée en électronique, pour dénoter la partie imaginaire des complexes à la place de la lettre `i` a fait l'objet de discussions sur la plateforme de suivi `https://bugs.python.org/issue10562` : Guido van Rossum ferme la discussion : `i` et `I` ressemblent trop à des chiffres.

1.3. Les chaînes de caractères

En Python, le type `str` représente une suite de caractères Unicode. Tous les caractères (ceux utilisés dans la plupart des langues connues, y compris les accentués) peuvent être concaténés dans une chaîne de caractères valide. Seul le caractère *antislash* \ (*backslash* en anglais) doit être doublé car il donne une signification spéciale à certaines séquences de caractères. Le préfixe `r""` (pour *raw*) désactive l'interprétation de l'*antislash*.

On peut utiliser indifféremment les guillemets simples ou doubles pour délimiter une chaîne de caractères. Les triples guillemets délimitent une chaîne de caractères multi-lignes ; ils sont couramment utilisés pour documenter les fonctions (☞ p. 16, § **1.8**).

```
>>> "Bonjour les amis"
'Bonjour les amis'
>>> "Bonjour les amis\n"
'Bonjour les amis\n'
>>> r"Bonjour les amis\n"
'Bonjour les amis\\n'
```

```
>>> print("Bonjour les amis\n")
Bonjour les amis

>>> print(r"Bonjour les amis\n")
Bonjour les amis\n
```

Chaque chaîne de caractères a une longueur. On peut concaténer deux chaînes de caractères (opérateur `+`), les répéter (opérateur `*`) et les indexer (opérateur `[]`). L'indexation peut être faite à l'aide d'un entier positif (le premier élément est indexé `[0]`), négatif (« en partant de la fin ») ou à l'aide d'un intervalle (type `slice` ☞ p. 13, § **1.5**) : la syntaxe `[2:4]` signifie « de 2 à 4 (exclus) » ; `[-4:]` signifie « les quatre derniers » ; `[:]` reprend l'intégralité du conteneur.

```
>>> "bon" + 'jour'
'bonjour'
>>> a = """Bonjour
... à tous"""
>>> a
'Bonjour\nà tous'
>>> (a + ' ') * 2
'bonjour bonjour '
```

```
>>> a[0]
'B'
>>> a[2:4]
'nj'
>>> a[-4:]
'tous'
>>> a[:]
'Bonjour\nà tous'
```

De nombreuses méthodes permettent de tester (`.isupper()`, `.startswith()`, etc.) ou de modifier une chaîne de caractères (`.lower()`, `.split()`, `.replace()`) :