**Glossaire**

>>>

L'invite de commande utilisée par défaut dans l'interpréteur interactif. On la voit souvent dans des exemples de code qui peuvent être exécutés interactivement dans l'interpréteur.

...

Peut faire référence à :

* L'invite de commande utilisée par défaut dans l'interpréteur interactif lorsqu'on entre un bloc de code indenté, dans des délimiteurs fonctionnant par paires (parenthèses, crochets, accolades, triple guillemets), ou après un avoir spécifié un décorateur.
* La constante [Ellipsis](https://docs.python.org/fr/3.9/library/constants.html#Ellipsis).

*2to3*

Outil qui essaie de convertir du code pour Python 2.x en code pour Python 3.x en gérant la plupart des incompatibilités qui peuvent être détectées en analysant la source et parcourant son arbre syntaxique.

*2to3* est disponible dans la bibliothèque standard sous le nom de [lib2to3](https://docs.python.org/fr/3.9/library/2to3.html#module-lib2to3) ; un point d’entrée indépendant est fourni via Tools/scripts/2to3. Cf. [2to3 — Traduction automatique de code en Python 2 vers Python 3](https://docs.python.org/fr/3.9/library/2to3.html#to3-reference).

classe de base abstraite

Les classes de base abstraites (ABC, suivant l'abréviation anglaise *Abstract Base Class*) complètent le [duck-typing](https://docs.python.org/fr/3.9/glossary.html#term-duck-typing) en fournissant un moyen de définir des interfaces pour les cas où d'autres techniques comme [hasattr()](https://docs.python.org/fr/3.9/library/functions.html#hasattr) seraient inélégantes ou subtilement fausses (par exemple avec les [méthodes magiques](https://docs.python.org/fr/3.9/reference/datamodel.html#special-lookup)). Les ABC introduisent des sous-classes virtuelles qui n'héritent pas d'une classe mais qui sont quand même reconnues par [isinstance()](https://docs.python.org/fr/3.9/library/functions.html#isinstance) ou [issubclass()](https://docs.python.org/fr/3.9/library/functions.html#issubclass) (voir la documentation du module [abc](https://docs.python.org/fr/3.9/library/abc.html#module-abc)). Python contient de nombreuses ABC pour les structures de données (dans le module [collections.abc](https://docs.python.org/fr/3.9/library/collections.abc.html#module-collections.abc)), les nombres (dans le module [numbers](https://docs.python.org/fr/3.9/library/numbers.html#module-numbers)), les flux (dans le module [io](https://docs.python.org/fr/3.9/library/io.html#module-io)) et les chercheurs-chargeurs du système d'importation (dans le module [importlib.abc](https://docs.python.org/fr/3.9/library/importlib.html#module-importlib.abc)). Vous pouvez créer vos propres ABC avec le module [abc](https://docs.python.org/fr/3.9/library/abc.html#module-abc).

annotation

Étiquette associée à une variable, un attribut de classe, un paramètre de fonction ou une valeur de retour. Elle est utilisée par convention comme [type hint](https://docs.python.org/fr/3.9/glossary.html#term-type-hint).

Les annotations de variables locales ne sont pas accessibles au moment de l'exécution, mais les annotations de variables globales, d'attributs de classe et de fonctions sont stockées dans l'attribut spécial \_\_annotations\_\_ des modules, classes et fonctions, respectivement.

Voir [variable annotation](https://docs.python.org/fr/3.9/glossary.html#term-variable-annotation), [function annotation](https://docs.python.org/fr/3.9/glossary.html#term-function-annotation), [**PEP 484**](https://www.python.org/dev/peps/pep-0484) et [**PEP 526**](https://www.python.org/dev/peps/pep-0526), qui décrivent cette fonctionnalité.

argument

Valeur, donnée à une [fonction](https://docs.python.org/fr/3.9/glossary.html#term-function) ou à une [méthode](https://docs.python.org/fr/3.9/glossary.html#term-method) lors de son appel. Il existe deux types d'arguments :

* *argument nommé* : un argument précédé d'un identifiant (comme name=) ou un dictionnaire précédé de \*\*, lors d'un appel de fonction. Par exemple, 3 et 5 sont tous les deux des arguments nommés dans l'appel à [complex()](https://docs.python.org/fr/3.9/library/functions.html#complex) ici :
* complex(real=3, imag=5)
* complex(\*\*{'real': 3, 'imag': 5})
* *argument positionnel* : un argument qui n'est pas nommé. Les arguments positionnels apparaissent au début de la liste des arguments, ou donnés sous forme d'un [itérable](https://docs.python.org/fr/3.9/glossary.html#term-iterable) précédé par \*. Par exemple, 3 et 5 sont tous les deux des arguments positionnels dans les appels suivants :
* complex(3, 5)
* complex(\*(3, 5))

Les arguments se retrouvent dans le corps de la fonction appelée parmi les variables locales. Voir la section [Appels](https://docs.python.org/fr/3.9/reference/expressions.html#calls) à propos des règles dictant cette affectation. Syntaxiquement, toute expression est acceptée comme argument, et c'est la valeur résultante de l'expression qui sera affectée à la variable locale.

Voir aussi [parameter](https://docs.python.org/fr/3.9/glossary.html#term-parameter) dans le glossaire, la question [Différence entre argument et paramètre](https://docs.python.org/fr/3.9/faq/programming.html#faq-argument-vs-parameter) de la FAQ et la [**PEP 362**](https://www.python.org/dev/peps/pep-0362).

gestionnaire de contexte asynchrone

(*asynchronous context manager* en anglais) Objet contrôlant l'environnement à l'intérieur d'une instruction [with](https://docs.python.org/fr/3.9/reference/compound_stmts.html#with) en définissant les méthodes [\_\_aenter\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__aenter__) et [\_\_aexit\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__aexit__). A été Introduit par la [**PEP 492**](https://www.python.org/dev/peps/pep-0492).

générateur asynchrone

Fonction qui renvoie un [asynchronous generator iterator](https://docs.python.org/fr/3.9/glossary.html#term-asynchronous-generator-iterator). Cela ressemble à une coroutine définie par [async def](https://docs.python.org/fr/3.9/reference/compound_stmts.html#async-def), sauf qu'elle contient une ou des expressions [yield](https://docs.python.org/fr/3.9/reference/simple_stmts.html#yield) produisant ainsi uns série de valeurs utilisables dans une boucle [async for](https://docs.python.org/fr/3.9/reference/compound_stmts.html#async-for).

Générateur asynchrone fait généralement référence à une fonction, mais peut faire référence à un *itérateur de générateur asynchrone* dans certains contextes. Dans les cas où le sens voulu n'est pas clair, utiliser l'ensemble des termes lève l’ambiguïté.

Un générateur asynchrone peut contenir des expressions [await](https://docs.python.org/fr/3.9/reference/expressions.html#await) ainsi que des instructions [async for](https://docs.python.org/fr/3.9/reference/compound_stmts.html#async-for), et [async with](https://docs.python.org/fr/3.9/reference/compound_stmts.html#async-with).

itérateur de générateur asynchrone

Objet créé par une fonction [asynchronous generator](https://docs.python.org/fr/3.9/glossary.html#term-asynchronous-generator).

C'est un [asynchronous iterator](https://docs.python.org/fr/3.9/glossary.html#term-asynchronous-iterator) qui, lorsqu'il est appelé via la méthode [\_\_anext\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__anext__) renvoie un objet *awaitable* qui exécute le corps de la fonction du générateur asynchrone jusqu'au prochain [yield](https://docs.python.org/fr/3.9/reference/simple_stmts.html#yield).

Chaque [yield](https://docs.python.org/fr/3.9/reference/simple_stmts.html#yield) suspend temporairement l'exécution, en gardant en mémoire l'endroit et l'état de l'exécution (ce qui inclut les variables locales et les *try* en cours). Lorsque l'exécution de l'itérateur de générateur asynchrone reprend avec un nouvel *awaitable* renvoyé par [\_\_anext\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__anext__), elle repart de là où elle s'était arrêtée. Voir la [**PEP 492**](https://www.python.org/dev/peps/pep-0492) et la [**PEP 525**](https://www.python.org/dev/peps/pep-0525).

itérable asynchrone

Objet qui peut être utilisé dans une instruction [async for](https://docs.python.org/fr/3.9/reference/compound_stmts.html#async-for). Sa méthode [\_\_aiter\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__aiter__) doit renvoyer un [asynchronous iterator](https://docs.python.org/fr/3.9/glossary.html#term-asynchronous-iterator). A été introduit par la [**PEP 492**](https://www.python.org/dev/peps/pep-0492).

itérateur asynchrone

Objet qui implémente les méthodes [\_\_aiter\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__aiter__) et [\_\_anext\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__anext__). \_\_anext\_\_ doit renvoyer un objet [awaitable](https://docs.python.org/fr/3.9/glossary.html#term-awaitable). Tant que la méthode [\_\_anext\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__anext__) produit des objets *awaitable*, le [async for](https://docs.python.org/fr/3.9/reference/compound_stmts.html#async-for) appelant les consomme. L'itérateur asynchrone lève une exception [StopAsyncIteration](https://docs.python.org/fr/3.9/library/exceptions.html#StopAsyncIteration) pour signifier la fin de l'itération. A été introduit par la [**PEP 492**](https://www.python.org/dev/peps/pep-0492).

attribut

Valeur associée à un objet et désignée par son nom via une notation utilisant des points. Par exemple, si un objet *o* possède un attribut *a*, il sera référencé par *o.a*.

*awaitable*

Objet pouvant être utilisé dans une expression [await](https://docs.python.org/fr/3.9/reference/expressions.html#await). Ce peut être une [coroutine](https://docs.python.org/fr/3.9/glossary.html#term-coroutine) ou un objet avec une méthode [\_\_await\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__await__). Voir aussi la [**PEP 492**](https://www.python.org/dev/peps/pep-0492).

*BDFL*

Dictateur bienveillant à vie (*Benevolent Dictator For Life* en anglais). Pseudonyme de [Guido van Rossum](https://gvanrossum.github.io/), le créateur de Python.

fichier binaire

Un [file object](https://docs.python.org/fr/3.9/glossary.html#term-file-object) capable de lire et d'écrire des [bytes-like objects](https://docs.python.org/fr/3.9/glossary.html#term-bytes-like-object). Des fichiers binaires sont, par exemple, les fichiers ouverts en mode binaire ('rb', 'wb', ou 'rb+'), sys.stdin.buffer, sys.stdout.buffer, les instances de [io.BytesIO](https://docs.python.org/fr/3.9/library/io.html#io.BytesIO) ou de [gzip.GzipFile](https://docs.python.org/fr/3.9/library/gzip.html#gzip.GzipFile).

Consultez [fichier texte](https://docs.python.org/fr/3.9/glossary.html#term-text-file), un objet fichier capable de lire et d'écrire des objets [str](https://docs.python.org/fr/3.9/library/stdtypes.html#str).

objet octet-compatible

Un objet gérant les [Protocole tampon](https://docs.python.org/fr/3.9/c-api/buffer.html#bufferobjects) et pouvant exporter un tampon (*buffer* en anglais) C-[contiguous](https://docs.python.org/fr/3.9/glossary.html#term-contiguous). Cela inclut les objets [bytes](https://docs.python.org/fr/3.9/library/stdtypes.html#bytes), [bytearray](https://docs.python.org/fr/3.9/library/stdtypes.html#bytearray) et [array.array](https://docs.python.org/fr/3.9/library/array.html#array.array), ainsi que beaucoup d'objets [memoryview](https://docs.python.org/fr/3.9/library/stdtypes.html#memoryview). Les objets bytes-compatibles peuvent être utilisés pour diverses opérations sur des données binaires, comme la compression, la sauvegarde dans un fichier binaire ou l'envoi sur le réseau.

Certaines opérations nécessitent de travailler sur des données binaires variables. La documentation parle de ceux-ci comme des *read-write bytes-like objects*. Par exemple, [bytearray](https://docs.python.org/fr/3.9/library/stdtypes.html#bytearray) ou une [memoryview](https://docs.python.org/fr/3.9/library/stdtypes.html#memoryview) d'un [bytearray](https://docs.python.org/fr/3.9/library/stdtypes.html#bytearray) en font partie. D'autres opérations nécessitent de travailler sur des données binaires stockées dans des objets immuables (*"read-only bytes-like objects"*), par exemples [bytes](https://docs.python.org/fr/3.9/library/stdtypes.html#bytes) ou [memoryview](https://docs.python.org/fr/3.9/library/stdtypes.html#memoryview) d'un objet byte.

code intermédiaire (*bytecode*)

Le code source, en Python, est compilé en un code intermédiaire (*bytecode* en anglais), la représentation interne à CPython d'un programme Python. Le code intermédiaire est mis en cache dans un fichier .pyc de manière à ce qu'une seconde exécution soit plus rapide (la compilation en code intermédiaire a déjà été faite). On dit que ce *langage intermédiaire* est exécuté sur une [virtual machine](https://docs.python.org/fr/3.9/glossary.html#term-virtual-machine) qui exécute des instructions machine pour chaque instruction du code intermédiaire. Notez que le code intermédiaire n'a pas vocation à fonctionner sur différentes machines virtuelles Python ou à être stable entre différentes versions de Python.

La documentation du [module dis](https://docs.python.org/fr/3.9/library/dis.html#bytecodes) fournit une liste des instructions du code intermédiaire.

fonction de rappel

Une sous-fonction passée en argument pour être exécutée plus tard.

classe

Modèle pour créer des objets définis par l'utilisateur. Une définition de classe (*class*) contient normalement des définitions de méthodes qui agissent sur les instances de la classe.

variable de classe

Une variable définie dans une classe et destinée à être modifiée uniquement au niveau de la classe (c'est-à-dire, pas dans une instance de la classe).

coercition

Conversion implicite d'une instance d'un type vers un autre lors d'une opération dont les deux opérandes doivent être de même type. Par exemple int(3.15) convertit explicitement le nombre à virgule flottante en nombre entier 3. Mais dans l'opération 3 + 4.5, les deux opérandes sont d'un type différent (un entier et un nombre à virgule flottante), alors qu'ils doivent avoir le même type pour être additionnés (sinon une exception [TypeError](https://docs.python.org/fr/3.9/library/exceptions.html#TypeError) serait levée). Sans coercition, tous les opérandes, même de types compatibles, devraient être convertis (on parle aussi de *cast*) explicitement par le développeur, par exemple : float(3) + 4.5 au lieu du simple 3 + 4.5.

nombre complexe

Extension des nombres réels familiers, dans laquelle tous les nombres sont exprimés sous la forme d'une somme d'une partie réelle et d'une partie imaginaire. Les nombres imaginaires sont les nombres réels multipliés par l'unité imaginaire (la racine carrée de -1, souvent écrite i en mathématiques ou j par les ingénieurs). Python comprend nativement les nombres complexes, écrits avec cette dernière notation : la partie imaginaire est écrite avec un suffixe j, exemple, 3+1j. Pour utiliser les équivalents complexes de [math](https://docs.python.org/fr/3.9/library/math.html#module-math), utilisez [cmath](https://docs.python.org/fr/3.9/library/cmath.html#module-cmath). Les nombres complexes sont un concept assez avancé en mathématiques. Si vous ne connaissez pas ce concept, vous pouvez tranquillement les ignorer.

gestionnaire de contexte

Objet contrôlant l'environnement à l'intérieur d'un bloc [with](https://docs.python.org/fr/3.9/reference/compound_stmts.html#with) en définissant les méthodes [\_\_enter\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__enter__) et [\_\_exit\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__exit__). Consultez la [**PEP 343**](https://www.python.org/dev/peps/pep-0343).

variable de contexte

Une variable qui peut avoir des valeurs différentes en fonction de son contexte. Cela est similaire au stockage par fil d’exécution (*Thread Local Storage* en anglais) dans lequel chaque fil d’exécution peut avoir une valeur différente pour une variable. Toutefois, avec les variables de contexte, il peut y avoir plusieurs contextes dans un fil d’exécution et l’utilisation principale pour les variables de contexte est de garder une trace des variables dans les tâches asynchrones concourantes. Voir [contextvars](https://docs.python.org/fr/3.9/library/contextvars.html#module-contextvars).

contigu

Un tampon (*buffer* en anglais) est considéré comme contigu s’il est soit *C-contigu* soit *Fortran-contigu*. Les tampons de dimension zéro sont C-contigus et Fortran-contigus. Pour un tableau à une dimension, ses éléments doivent être placés en mémoire l’un à côté de l’autre, dans l’ordre croissant de leur indice, en commençant à zéro. Pour qu’un tableau multidimensionnel soit C-contigu, le dernier indice doit être celui qui varie le plus rapidement lors du parcours de ses éléments dans l’ordre de leur adresse mémoire. À l'inverse, dans les tableaux Fortran-contigu, c’est le premier indice qui doit varier le plus rapidement.

coroutine

Les coroutines sont une forme généralisée des fonctions. On entre dans une fonction en un point et on en sort en un autre point. On peut entrer, sortir et reprendre l'exécution d'une coroutine en plusieurs points. Elles peuvent être implémentées en utilisant l'instruction [async def](https://docs.python.org/fr/3.9/reference/compound_stmts.html#async-def). Voir aussi la [**PEP 492**](https://www.python.org/dev/peps/pep-0492).

fonction coroutine

Fonction qui renvoie un objet [coroutine](https://docs.python.org/fr/3.9/glossary.html#term-coroutine). Une fonction coroutine peut être définie par l'instruction [async def](https://docs.python.org/fr/3.9/reference/compound_stmts.html#async-def) et peut contenir les mots clés [await](https://docs.python.org/fr/3.9/reference/expressions.html#await), [async for](https://docs.python.org/fr/3.9/reference/compound_stmts.html#async-for) ainsi que [async with](https://docs.python.org/fr/3.9/reference/compound_stmts.html#async-with). A été introduit par la [**PEP 492**](https://www.python.org/dev/peps/pep-0492).

CPython

L'implémentation canonique du langage de programmation Python, tel que distribué sur [python.org](https://www.python.org). Le terme "CPython" est utilisé dans certains contextes lorsqu'il est nécessaire de distinguer cette implémentation des autres comme *Jython* ou *IronPython*.

décorateur

Fonction dont la valeur de retour est une autre fonction. Un décorateur est habituellement utilisé pour transformer une fonction via la syntaxe @wrapper, dont les exemples typiques sont : [classmethod()](https://docs.python.org/fr/3.9/library/functions.html#classmethod) et [staticmethod()](https://docs.python.org/fr/3.9/library/functions.html#staticmethod).

La syntaxe des décorateurs est simplement du sucre syntaxique, les définitions des deux fonctions suivantes sont sémantiquement équivalentes :

def f(...):

...

f = staticmethod(f)

@staticmethod

def f(...):

...

Quoique moins fréquemment utilisé, le même concept existe pour les classes. Consultez la documentation [définitions de fonctions](https://docs.python.org/fr/3.9/reference/compound_stmts.html#function) et [définitions de classes](https://docs.python.org/fr/3.9/reference/compound_stmts.html#class) pour en savoir plus sur les décorateurs.

descripteur

N'importe quel objet définissant les méthodes [\_\_get\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__get__), [\_\_set\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__set__), ou [\_\_delete\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__delete__). Lorsque l'attribut d'une classe est un descripteur, son comportement spécial est déclenché lors de la recherche des attributs. Normalement, lorsque vous écrivez *a.b* pour obtenir, affecter ou effacer un attribut, Python recherche l'objet nommé *b* dans le dictionnaire de la classe de *a*. Mais si *b* est un descripteur, c'est la méthode de ce descripteur qui est alors appelée. Comprendre les descripteurs est requis pour avoir une compréhension approfondie de Python, ils sont la base de nombre de ses caractéristiques notamment les fonctions, méthodes, propriétés, méthodes de classes, méthodes statiques et les références aux classes parentes.

For more information about descriptors' methods, see [Implémentation de descripteurs](https://docs.python.org/fr/3.9/reference/datamodel.html#descriptors) or the [Descriptor How To Guide](https://docs.python.org/fr/3.9/howto/descriptor.html#descriptorhowto).

dictionnaire

Structure de donnée associant des clés à des valeurs. Les clés peuvent être n'importe quel objet possédant les méthodes [\_\_hash\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__hash__) et [\_\_eq\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__eq__). En Perl, les dictionnaires sont appelés "*hash*".

dictionary comprehension

A compact way to process all or part of the elements in an iterable and return a dictionary with the results. results = {n: n \*\* 2 for n in range(10)} generates a dictionary containing key n mapped to value n \*\* 2. See [Agencements des listes, ensembles et dictionnaires](https://docs.python.org/fr/3.9/reference/expressions.html#comprehensions).

vue de dictionnaire

Objets retournés par les méthodes [dict.keys()](https://docs.python.org/fr/3.9/library/stdtypes.html#dict.keys), [dict.values()](https://docs.python.org/fr/3.9/library/stdtypes.html#dict.values) et [dict.items()](https://docs.python.org/fr/3.9/library/stdtypes.html#dict.items). Ils fournissent des vues dynamiques des entrées du dictionnaire, ce qui signifie que lorsque le dictionnaire change, la vue change. Pour transformer une vue en vraie liste, utilisez list(dictview). Voir [Les vues de dictionnaires](https://docs.python.org/fr/3.9/library/stdtypes.html#dict-views).

*docstring* (chaîne de documentation)

Première chaîne littérale qui apparaît dans l'expression d'une classe, fonction, ou module. Bien qu'ignorée à l'exécution, elle est reconnue par le compilateur et placée dans l'attribut \_\_doc\_\_ de la classe, de la fonction ou du module. Comme cette chaîne est disponible par introspection, c'est l'endroit idéal pour documenter l'objet.

*duck-typing*

Style de programmation qui ne prend pas en compte le type d'un objet pour déterminer s'il respecte une interface, mais qui appelle simplement la méthode ou l'attribut (*Si ça a un bec et que ça cancane, ça doit être un canard*, *duck* signifie canard en anglais). En se concentrant sur les interfaces plutôt que les types, du code bien construit améliore sa flexibilité en autorisant des substitutions polymorphiques. Le *duck-typing* évite de vérifier les types via [type()](https://docs.python.org/fr/3.9/library/functions.html#type) ou [isinstance()](https://docs.python.org/fr/3.9/library/functions.html#isinstance), Notez cependant que le *duck-typing* peut travailler de pair avec les [classes de base abstraites](https://docs.python.org/fr/3.9/glossary.html#term-abstract-base-class). À la place, le *duck-typing* utilise plutôt [hasattr()](https://docs.python.org/fr/3.9/library/functions.html#hasattr) ou la programmation [EAFP](https://docs.python.org/fr/3.9/glossary.html#term-eafp).

EAFP

Il est plus simple de demander pardon que demander la permission (*Easier to Ask for Forgiveness than Permission* en anglais). Ce style de développement Python fait l'hypothèse que le code est valide et traite les exceptions si cette hypothèse s'avère fausse. Ce style, propre et efficace, est caractérisé par la présence de beaucoup de mots clés [try](https://docs.python.org/fr/3.9/reference/compound_stmts.html#try) et [except](https://docs.python.org/fr/3.9/reference/compound_stmts.html#except). Cette technique de programmation contraste avec le style [LBYL](https://docs.python.org/fr/3.9/glossary.html#term-lbyl) utilisé couramment dans les langages tels que C.

expression

Suite logique de termes et chiffres conformes à la syntaxe Python dont l'évaluation fournit une valeur. En d'autres termes, une expression est une suite d'éléments tels que des noms, opérateurs, littéraux, accès d'attributs, méthodes ou fonctions qui aboutissent à une valeur. Contrairement à beaucoup d'autres langages, les différentes constructions du langage ne sont pas toutes des expressions. On trouve également des [instructions](https://docs.python.org/fr/3.9/glossary.html#term-statement) qui ne peuvent pas être utilisées comme expressions, tel que [while](https://docs.python.org/fr/3.9/reference/compound_stmts.html#while). Les affectations sont également des instructions et non des expressions.

module d'extension

Module écrit en C ou C++, utilisant l'API C de Python pour interagir avec Python et le code de l'utilisateur.

f-string

Chaîne littérale préfixée de 'f' ou 'F'. Les "f-strings" sont un raccourci pour [formatted string literals](https://docs.python.org/fr/3.9/reference/lexical_analysis.html#f-strings). Voir la [**PEP 498**](https://www.python.org/dev/peps/pep-0498).

objet fichier

Objet exposant une ressource via une API orientée fichier (avec les méthodes read() ou write()). En fonction de la manière dont il a été créé, un objet fichier peut interfacer l'accès à un fichier sur le disque ou à un autre type de stockage ou de communication (typiquement l'entrée standard, la sortie standard, un tampon en mémoire, un connecteur réseau…). Les objets fichiers sont aussi appelés *file-like-objects* ou *streams*.

Il existe en réalité trois catégories de fichiers objets : les [fichiers binaires](https://docs.python.org/fr/3.9/glossary.html#term-binary-file) bruts, les [fichiers binaires](https://docs.python.org/fr/3.9/glossary.html#term-binary-file) avec tampon (*buffer*) et les [fichiers textes](https://docs.python.org/fr/3.9/glossary.html#term-text-file). Leurs interfaces sont définies dans le module [io](https://docs.python.org/fr/3.9/library/io.html#module-io). Le moyen le plus simple et direct de créer un objet fichier est d'utiliser la fonction [open()](https://docs.python.org/fr/3.9/library/functions.html#open).

objet fichier-compatible

Synonyme de [objet fichier](https://docs.python.org/fr/3.9/glossary.html#term-file-object).

chercheur

Objet qui essaie de trouver un [chargeur](https://docs.python.org/fr/3.9/glossary.html#term-loader) pour le module en cours d'importation.

Depuis Python 3.3, il existe deux types de chercheurs : les [chercheurs dans les méta-chemins](https://docs.python.org/fr/3.9/glossary.html#term-meta-path-finder) à utiliser avec [sys.meta\_path](https://docs.python.org/fr/3.9/library/sys.html#sys.meta_path) ; les [chercheurs d'entrée dans path](https://docs.python.org/fr/3.9/glossary.html#term-path-entry-finder) à utiliser avec [sys.path\_hooks](https://docs.python.org/fr/3.9/library/sys.html#sys.path_hooks).

Voir les [**PEP 302**](https://www.python.org/dev/peps/pep-0302), [**PEP 420**](https://www.python.org/dev/peps/pep-0420) et [**PEP 451**](https://www.python.org/dev/peps/pep-0451) pour plus de détails.

division entière

Division mathématique arrondissant à l'entier inférieur. L'opérateur de la division entière est //. Par exemple l'expression 11 // 4 vaut 2, contrairement à 11 / 4 qui vaut 2.75. Notez que (-11) // 4 vaut -3 car l'arrondi se fait à l'entier inférieur. Voir la [**PEP 328**](https://www.python.org/dev/peps/pep-0328).

fonction

Suite d'instructions qui renvoie une valeur à son appelant. On peut lui passer des [arguments](https://docs.python.org/fr/3.9/glossary.html#term-argument) qui pourront être utilisés dans le corps de la fonction. Voir aussi [paramètre](https://docs.python.org/fr/3.9/glossary.html#term-parameter), [méthode](https://docs.python.org/fr/3.9/glossary.html#term-method) et [Définition de fonctions](https://docs.python.org/fr/3.9/reference/compound_stmts.html#function).

annotation de fonction

[annotation](https://docs.python.org/fr/3.9/glossary.html#term-annotation) d'un paramètre de fonction ou valeur de retour.

Les annotations de fonctions sont généralement utilisées pour des [indications de types](https://docs.python.org/fr/3.9/glossary.html#term-type-hint) : par exemple, cette fonction devrait prendre deux arguments [int](https://docs.python.org/fr/3.9/library/functions.html#int) et devrait également avoir une valeur de retour de type [int](https://docs.python.org/fr/3.9/library/functions.html#int) :

def sum\_two\_numbers(a: int, b: int) -> int:

return a + b

L'annotation syntaxique de la fonction est expliquée dans la section [Définition de fonctions](https://docs.python.org/fr/3.9/reference/compound_stmts.html#function).

Voir [variable annotation](https://docs.python.org/fr/3.9/glossary.html#term-variable-annotation) et [**PEP 484**](https://www.python.org/dev/peps/pep-0484), qui décrivent cette fonctionnalité.

\_\_future\_\_

Pseudo-module que les développeurs peuvent utiliser pour activer de nouvelles fonctionnalités du langage qui ne sont pas compatibles avec l'interpréteur utilisé.

En important le module [\_\_future\_\_](https://docs.python.org/fr/3.9/library/__future__.html#module-__future__) et en affichant ses variables, vous pouvez voir à quel moment une nouvelle fonctionnalité a été rajoutée dans le langage et quand elle devient le comportement par défaut :

>>>

>>> import \_\_future\_\_

>>> \_\_future\_\_.division

\_Feature((2, 2, 0, 'alpha', 2), (3, 0, 0, 'alpha', 0), 8192)

ramasse-miettes

(*garbage collection* en anglais) Mécanisme permettant de libérer de la mémoire lorsqu'elle n'est plus utilisée. Python utilise un ramasse-miettes par comptage de référence et un ramasse-miettes cyclique capable de détecter et casser les références circulaires. Le ramasse-miettes peut être contrôlé en utilisant le module [gc](https://docs.python.org/fr/3.9/library/gc.html#module-gc).

générateur

Fonction qui renvoie un [itérateur de générateur](https://docs.python.org/fr/3.9/glossary.html#term-generator-iterator). Cela ressemble à une fonction normale, en dehors du fait qu'elle contient une ou des expressions [yield](https://docs.python.org/fr/3.9/reference/simple_stmts.html#yield) produisant une série de valeurs utilisable dans une boucle *for* ou récupérées une à une via la fonction [next()](https://docs.python.org/fr/3.9/library/functions.html#next).

Fait généralement référence à une fonction générateur mais peut faire référence à un *itérateur de générateur* dans certains contextes. Dans les cas où le sens voulu n'est pas clair, utiliser les termes complets lève l’ambiguïté.

itérateur de générateur

Objet créé par une fonction [générateur](https://docs.python.org/fr/3.9/glossary.html#term-generator).

Chaque [yield](https://docs.python.org/fr/3.9/reference/simple_stmts.html#yield) suspend temporairement l'exécution, en se rappelant l'endroit et l'état de l'exécution (y compris les variables locales et les *try* en cours). Lorsque l'itérateur de générateur reprend, il repart là où il en était (contrairement à une fonction qui prendrait un nouveau départ à chaque invocation).

expression génératrice

Expression qui donne un itérateur. Elle ressemble à une expression normale, suivie d'une clause for définissant une variable de boucle, un intervalle et une clause if optionnelle. Toute cette expression génère des valeurs pour la fonction qui l'entoure :

>>>

>>> sum(i\*i for i in range(10)) # sum of squares 0, 1, 4, ... 81

285

fonction générique

Fonction composée de plusieurs fonctions implémentant les mêmes opérations pour différents types. L'implémentation à utiliser est déterminée lors de l'appel par l'algorithme de répartition.

Voir aussi [single dispatch](https://docs.python.org/fr/3.9/glossary.html#term-single-dispatch), le décorateur [functools.singledispatch()](https://docs.python.org/fr/3.9/library/functools.html#functools.singledispatch) et la [**PEP 443**](https://www.python.org/dev/peps/pep-0443).

generic type

A [type](https://docs.python.org/fr/3.9/glossary.html#term-type) that can be parameterized; typically a container like [list](https://docs.python.org/fr/3.9/library/stdtypes.html#list). Used for [type hints](https://docs.python.org/fr/3.9/glossary.html#term-type-hint) and [annotations](https://docs.python.org/fr/3.9/glossary.html#term-annotation).

See [**PEP 483**](https://www.python.org/dev/peps/pep-0483) for more details, and [typing](https://docs.python.org/fr/3.9/library/typing.html#module-typing) or [generic alias type](https://docs.python.org/fr/3.9/library/stdtypes.html#types-genericalias) for its uses.

GIL

Voir [global interpreter lock](https://docs.python.org/fr/3.9/glossary.html#term-global-interpreter-lock).

verrou global de l'interpréteur

(*global interpreter lock* en anglais) Mécanisme utilisé par l'interpréteur [CPython](https://docs.python.org/fr/3.9/glossary.html#term-cpython) pour s'assurer qu'un seul fil d'exécution (*thread* en anglais) n'exécute le [bytecode](https://docs.python.org/fr/3.9/glossary.html#term-bytecode) à la fois. Cela simplifie l'implémentation de CPython en rendant le modèle objet (incluant des parties critiques comme la classe native [dict](https://docs.python.org/fr/3.9/library/stdtypes.html#dict)) implicitement protégé contre les accès concourants. Verrouiller l'interpréteur entier rend plus facile l'implémentation de multiples fils d'exécution (*multi-thread* en anglais), au détriment malheureusement de beaucoup du parallélisme possible sur les machines ayant plusieurs processeurs.

Cependant, certains modules d'extension, standards ou non, sont conçus de manière à libérer le GIL lorsqu'ils effectuent des tâches lourdes tel que la compression ou le hachage. De la même manière, le GIL est toujours libéré lors des entrées / sorties.

Les tentatives précédentes d'implémenter un interpréteur Python avec une granularité de verrouillage plus fine ont toutes échouées, à cause de leurs mauvaises performances dans le cas d'un processeur unique. Il est admis que corriger ce problème de performance induit mènerait à une implémentation beaucoup plus compliquée et donc plus coûteuse à maintenir.

*pyc* utilisant le hachage

Un fichier de cache de code intermédiaire (*bytecode* en anglais) qui utilise le hachage plutôt que l'heure de dernière modification du fichier source correspondant pour déterminer sa validité. Voir [Invalidation de bytecode mis en cache](https://docs.python.org/fr/3.9/reference/import.html#pyc-invalidation).

hachable

Un objet est *hachable* s'il a une empreinte (*hash*) qui ne change jamais (il doit donc implémenter une méthode [\_\_hash\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__hash__)) et s'il peut être comparé à d'autres objets (avec la méthode [\_\_eq\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__eq__)). Les objets hachables dont la comparaison par \_\_eq\_\_ est vraie doivent avoir la même empreinte.

La hachabilité permet à un objet d'être utilisé comme clé de dictionnaire ou en tant que membre d'un ensemble (type *set*), car ces structures de données utilisent ce *hash*.

La plupart des types immuables natifs de Python sont hachables, mais les conteneurs muables (comme les listes ou les dictionnaires) ne le sont pas ; les conteneurs immuables (comme les n-uplets ou les ensembles figés) ne sont hachables que si leurs éléments sont hachables. Les instances de classes définies par les utilisateurs sont hachables par défaut. Elles sont toutes considérées différentes (sauf avec elles-mêmes) et leur valeur de hachage est calculée à partir de leur [id()](https://docs.python.org/fr/3.9/library/functions.html#id).

IDLE

Environnement de développement intégré pour Python. IDLE est un éditeur basique et un interpréteur livré avec la distribution standard de Python.

immuable

Objet dont la valeur ne change pas. Les nombres, les chaînes et les n-uplets sont immuables. Ils ne peuvent être modifiés. Un nouvel objet doit être créé si une valeur différente doit être stockée. Ils jouent un rôle important quand une valeur de *hash* constante est requise, typiquement en clé de dictionnaire.

chemin des importations

Liste de [entrées](https://docs.python.org/fr/3.9/glossary.html#term-path-entry) dans lesquelles le [chercheur basé sur les chemins](https://docs.python.org/fr/3.9/glossary.html#term-path-based-finder) cherche les modules à importer. Typiquement, lors d'une importation, cette liste vient de [sys.path](https://docs.python.org/fr/3.9/library/sys.html#sys.path) ; pour les sous-paquets, elle peut aussi venir de l'attribut \_\_path\_\_ du paquet parent.

importer

Processus rendant le code Python d'un module disponible dans un autre.

importateur

Objet qui trouve et charge un module, en même temps un [chercheur](https://docs.python.org/fr/3.9/glossary.html#term-finder) et un [chargeur](https://docs.python.org/fr/3.9/glossary.html#term-loader).

interactif

Python a un interpréteur interactif, ce qui signifie que vous pouvez écrire des expressions et des instructions à l'invite de l'interpréteur. L'interpréteur Python va les exécuter immédiatement et vous en présenter le résultat. Démarrez juste python (probablement depuis le menu principal de votre ordinateur). C'est un moyen puissant pour tester de nouvelles idées ou étudier de nouveaux modules (souvenez-vous de help(x)).

interprété

Python est un langage interprété, en opposition aux langages compilés, bien que la frontière soit floue en raison de la présence d'un compilateur en code intermédiaire. Cela signifie que les fichiers sources peuvent être exécutés directement, sans avoir à compiler un fichier exécutable intermédiaire. Les langages interprétés ont généralement un cycle de développement / débogage plus court que les langages compilés. Cependant, ils s'exécutent généralement plus lentement. Voir aussi [interactif](https://docs.python.org/fr/3.9/glossary.html#term-interactive).

arrêt de l'interpréteur

Lorsqu'on lui demande de s'arrêter, l'interpréteur Python entre dans une phase spéciale où il libère graduellement les ressources allouées, comme les modules ou quelques structures de données internes. Il fait aussi quelques appels au [ramasse-miettes](https://docs.python.org/fr/3.9/glossary.html#term-garbage-collection). Cela peut déclencher l'exécution de code dans des destructeurs ou des fonctions de rappels de *weakrefs*. Le code exécuté lors de l'arrêt peut rencontrer des exceptions puisque les ressources auxquelles il fait appel sont susceptibles de ne plus fonctionner, (typiquement les modules des bibliothèques ou le mécanisme de *warning*).

La principale raison d'arrêt de l'interpréteur est que le module \_\_main\_\_ ou le script en cours d'exécution a terminé de s'exécuter.

itérable

Objet capable de renvoyer ses éléments un à un. Par exemple, tous les types séquence (comme [list](https://docs.python.org/fr/3.9/library/stdtypes.html#list), [str](https://docs.python.org/fr/3.9/library/stdtypes.html#str), et [tuple](https://docs.python.org/fr/3.9/library/stdtypes.html#tuple)), quelques autres types comme [dict](https://docs.python.org/fr/3.9/library/stdtypes.html#dict), [objets fichiers](https://docs.python.org/fr/3.9/glossary.html#term-file-object) ou tout objet d'une classe ayant une méthode [\_\_iter\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__iter__) ou [\_\_getitem\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__getitem__) qui implémente la sémantique d'une [Sequence](https://docs.python.org/fr/3.9/glossary.html#term-sequence).

Les itérables peuvent être utilisés dans des boucles [for](https://docs.python.org/fr/3.9/reference/compound_stmts.html#for) et à beaucoup d'autres endroits où une séquence est requise ([zip()](https://docs.python.org/fr/3.9/library/functions.html#zip), [map()](https://docs.python.org/fr/3.9/library/functions.html#map)…). Lorsqu'un itérable est passé comme argument à la fonction native [iter()](https://docs.python.org/fr/3.9/library/functions.html#iter), celle-ci fournit en retour un itérateur sur cet itérable. Cet itérateur n'est valable que pour une seule passe sur le jeu de valeurs. Lors de l'utilisation d'itérables, il n'est habituellement pas nécessaire d'appeler [iter()](https://docs.python.org/fr/3.9/library/functions.html#iter) ou de s'occuper soi-même des objets itérateurs. L'instruction for le fait automatiquement pour vous, créant une variable temporaire anonyme pour garder l'itérateur durant la boucle. Voir aussi [itérateur](https://docs.python.org/fr/3.9/glossary.html#term-iterator), [séquence](https://docs.python.org/fr/3.9/glossary.html#term-sequence) et [générateur](https://docs.python.org/fr/3.9/glossary.html#term-generator).

itérateur

Objet représentant un flux de donnée. Des appels successifs à la méthode [\_\_next\_\_()](https://docs.python.org/fr/3.9/library/stdtypes.html#iterator.__next__) de l'itérateur (ou le passer à la fonction native [next()](https://docs.python.org/fr/3.9/library/functions.html#next)) donne successivement les objets du flux. Lorsque plus aucune donnée n'est disponible, une exception [StopIteration](https://docs.python.org/fr/3.9/library/exceptions.html#StopIteration) est levée. À ce point, l'itérateur est épuisé et tous les appels suivants à sa méthode \_\_next\_\_() lèveront encore une exception [StopIteration](https://docs.python.org/fr/3.9/library/exceptions.html#StopIteration). Les itérateurs doivent avoir une méthode [\_\_iter\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__iter__) qui renvoie l'objet itérateur lui-même, de façon à ce que chaque itérateur soit aussi itérable et puisse être utilisé dans la plupart des endroits où d'autres itérables sont attendus. Une exception notable est un code qui tente plusieurs itérations complètes. Un objet conteneur, (tel que [list](https://docs.python.org/fr/3.9/library/stdtypes.html#list)) produit un nouvel itérateur neuf à chaque fois qu'il est passé à la fonction [iter()](https://docs.python.org/fr/3.9/library/functions.html#iter) ou s'il est utilisé dans une boucle [for](https://docs.python.org/fr/3.9/reference/compound_stmts.html#for). Faire ceci sur un itérateur donnerait simplement le même objet itérateur épuisé utilisé dans son itération précédente, le faisant ressembler à un conteneur vide.

Vous trouverez davantage d'informations dans [Les types itérateurs](https://docs.python.org/fr/3.9/library/stdtypes.html#typeiter).

fonction clé

Une fonction clé est un objet appelable qui renvoie une valeur à fins de tri ou de classement. Par exemple, la fonction [locale.strxfrm()](https://docs.python.org/fr/3.9/library/locale.html#locale.strxfrm) est utilisée pour générer une clé de classement prenant en compte les conventions de classement spécifiques aux paramètres régionaux courants.

Plusieurs outils dans Python acceptent des fonctions clés pour déterminer comment les éléments sont classés ou groupés. On peut citer les fonctions [min()](https://docs.python.org/fr/3.9/library/functions.html#min), [max()](https://docs.python.org/fr/3.9/library/functions.html#max), [sorted()](https://docs.python.org/fr/3.9/library/functions.html#sorted), [list.sort()](https://docs.python.org/fr/3.9/library/stdtypes.html#list.sort), [heapq.merge()](https://docs.python.org/fr/3.9/library/heapq.html#heapq.merge), [heapq.nsmallest()](https://docs.python.org/fr/3.9/library/heapq.html#heapq.nsmallest), [heapq.nlargest()](https://docs.python.org/fr/3.9/library/heapq.html#heapq.nlargest) et [itertools.groupby()](https://docs.python.org/fr/3.9/library/itertools.html#itertools.groupby).

Il existe plusieurs moyens de créer une fonction clé. Par exemple, la méthode [str.lower()](https://docs.python.org/fr/3.9/library/stdtypes.html#str.lower) peut servir de fonction clé pour effectuer des recherches insensibles à la casse. Aussi, il est possible de créer des fonctions clés avec des expressions [lambda](https://docs.python.org/fr/3.9/reference/expressions.html#lambda), comme lambda r: (r[0], r[2]). Vous noterez que le module [operator](https://docs.python.org/fr/3.9/library/operator.html#module-operator) propose des constructeurs de fonctions clefs : [attrgetter()](https://docs.python.org/fr/3.9/library/operator.html#operator.attrgetter), [itemgetter()](https://docs.python.org/fr/3.9/library/operator.html#operator.itemgetter) et [methodcaller()](https://docs.python.org/fr/3.9/library/operator.html#operator.methodcaller). Voir [Comment Trier](https://docs.python.org/fr/3.9/howto/sorting.html#sortinghowto) pour des exemples de création et d'utilisation de fonctions clefs.

argument nommé

Voir [argument](https://docs.python.org/fr/3.9/glossary.html#term-argument).

lambda

Fonction anonyme sous la forme d'une [expression](https://docs.python.org/fr/3.9/glossary.html#term-expression) et ne contenant qu'une seule expression, exécutée lorsque la fonction est appelée. La syntaxe pour créer des fonctions lambda est : lambda [parameters]: expression

LBYL

Regarde avant de sauter, (*Look before you leap* en anglais). Ce style de programmation consiste à vérifier des conditions avant d'effectuer des appels ou des accès. Ce style contraste avec le style [EAFP](https://docs.python.org/fr/3.9/glossary.html#term-eafp) et se caractérise par la présence de beaucoup d'instructions [if](https://docs.python.org/fr/3.9/reference/compound_stmts.html#if).

Dans un environnement avec plusieurs fils d'exécution (*multi-threaded* en anglais), le style *LBYL* peut engendrer un séquencement critique (*race condition* en anglais) entre le "regarde" et le "sauter". Par exemple, le code if key in mapping: return mapping[key] peut échouer si un autre fil d'exécution supprime la clé *key* du *mapping* après le test mais avant l'accès. Ce problème peut être résolu avec des verrous (*locks*) ou avec l'approche EAFP.

*list*

Un type natif de [sequence](https://docs.python.org/fr/3.9/glossary.html#term-sequence) dans Python. En dépit de son nom, une list ressemble plus à un tableau (*array* dans la plupart des langages) qu'à une liste chaînée puisque les accès se font en O(1).

liste en compréhension (ou liste en intention)

Écriture concise pour manipuler tout ou partie des éléments d'une séquence et renvoyer une liste contenant les résultats. result = ['{:#04x}'.format(x) for x in range(256) if x % 2 == 0] génère la liste composée des nombres pairs de 0 à 255 écrits sous formes de chaînes de caractères et en hexadécimal (0x…). La clause [if](https://docs.python.org/fr/3.9/reference/compound_stmts.html#if) est optionnelle. Si elle est omise, tous les éléments du range(256) seront utilisés.

chargeur

Objet qui charge un module. Il doit définir une méthode nommée load\_module(). Un chargeur est typiquement donné par un [chercheur](https://docs.python.org/fr/3.9/glossary.html#term-finder). Voir la [**PEP 302**](https://www.python.org/dev/peps/pep-0302) pour plus de détails et importlib.ABC.Loader pour sa [classe de base abstraite](https://docs.python.org/fr/3.9/glossary.html#term-abstract-base-class).

méthode magique

Un synonyme informel de [special method](https://docs.python.org/fr/3.9/glossary.html#term-special-method).

tableau de correspondances

(*mapping* en anglais) Conteneur permettant de rechercher des éléments à partir de clés et implémentant les méthodes spécifiées dans les [classes de base abstraites](https://docs.python.org/fr/3.9/library/collections.abc.html#collections-abstract-base-classes) [collections.abc.Mapping](https://docs.python.org/fr/3.9/library/collections.abc.html#collections.abc.Mapping) ou [collections.abc.MutableMapping](https://docs.python.org/fr/3.9/library/collections.abc.html#collections.abc.MutableMapping). Les classes suivantes sont des exemples de tableaux de correspondances : [dict](https://docs.python.org/fr/3.9/library/stdtypes.html#dict), [collections.defaultdict](https://docs.python.org/fr/3.9/library/collections.html#collections.defaultdict), [collections.OrderedDict](https://docs.python.org/fr/3.9/library/collections.html#collections.OrderedDict) et [collections.Counter](https://docs.python.org/fr/3.9/library/collections.html#collections.Counter).

chercheur dans les méta-chemins

Un [chercheur](https://docs.python.org/fr/3.9/glossary.html#term-finder) renvoyé par une recherche dans [sys.meta\_path](https://docs.python.org/fr/3.9/library/sys.html#sys.meta_path). Les chercheurs dans les méta-chemins ressemblent, mais sont différents des [chercheurs d'entrée dans path](https://docs.python.org/fr/3.9/glossary.html#term-path-entry-finder).

Voir [importlib.abc.MetaPathFinder](https://docs.python.org/fr/3.9/library/importlib.html#importlib.abc.MetaPathFinder) pour les méthodes que les chercheurs dans les méta-chemins doivent implémenter.

métaclasse

Classe d'une classe. Les définitions de classe créent un nom pour la classe, un dictionnaire de classe et une liste de classes parentes. La métaclasse a pour rôle de réunir ces trois paramètres pour construire la classe. La plupart des langages orientés objet fournissent une implémentation par défaut. La particularité de Python est la possibilité de créer des métaclasses personnalisées. La plupart des utilisateurs n'auront jamais besoin de cet outil, mais lorsque le besoin survient, les métaclasses offrent des solutions élégantes et puissantes. Elles sont utilisées pour journaliser les accès à des propriétés, rendre sûrs les environnements *multi-threads*, suivre la création d'objets, implémenter des singletons et bien d'autres tâches.

Plus d'informations sont disponibles dans : [Méta-classes](https://docs.python.org/fr/3.9/reference/datamodel.html#metaclasses).

méthode

Fonction définie à l'intérieur d'une classe. Lorsqu'elle est appelée comme un attribut d'une instance de cette classe, la méthode reçoit l'instance en premier [argument](https://docs.python.org/fr/3.9/glossary.html#term-argument) (qui, par convention, est habituellement nommé self). Voir [function](https://docs.python.org/fr/3.9/glossary.html#term-function) et [nested scope](https://docs.python.org/fr/3.9/glossary.html#term-nested-scope).

ordre de résolution des méthodes

L'ordre de résolution des méthodes (*MRO* pour *Method Resolution Order* en anglais) est, lors de la recherche d'un attribut dans les classes parentes, la façon dont l'interpréteur Python classe ces classes parentes. Voir [The Python 2.3 Method Resolution Order](https://www.python.org/download/releases/2.3/mro/) pour plus de détails sur l'algorithme utilisé par l'interpréteur Python depuis la version 2.3.

module

Objet utilisé pour organiser une portion unitaire de code en Python. Les modules ont un espace de nommage et peuvent contenir n'importe quels objets Python. Charger des modules est appelé [importer](https://docs.python.org/fr/3.9/glossary.html#term-importing).

Voir aussi [paquet](https://docs.python.org/fr/3.9/glossary.html#term-package).

spécificateur de module

Espace de nommage contenant les informations, relatives à l'importation, utilisées pour charger un module. C'est une instance de la classe [importlib.machinery.ModuleSpec](https://docs.python.org/fr/3.9/library/importlib.html#importlib.machinery.ModuleSpec).

MRO

Voir [ordre de résolution des méthodes](https://docs.python.org/fr/3.9/glossary.html#term-method-resolution-order).

muable

Un objet muable peut changer de valeur tout en gardant le même [id()](https://docs.python.org/fr/3.9/library/functions.html#id). Voir aussi [immuable](https://docs.python.org/fr/3.9/glossary.html#term-immutable).

n-uplet nommé

Le terme "n-uplet nommé" s'applique à tous les types ou classes qui héritent de la classe tuple et dont les éléments indexables sont aussi accessibles en utilisant des attributs nommés. Les types et classes peuvent avoir aussi d'autres caractéristiques.

Plusieurs types natifs sont appelés n-uplets, y compris les valeurs retournées par [time.localtime()](https://docs.python.org/fr/3.9/library/time.html#time.localtime) et [os.stat()](https://docs.python.org/fr/3.9/library/os.html#os.stat). Un autre exemple est [sys.float\_info](https://docs.python.org/fr/3.9/library/sys.html#sys.float_info) :

>>>

>>> sys.float\_info[1] # indexed access

1024

>>> sys.float\_info.max\_exp # named field access

1024

>>> isinstance(sys.float\_info, tuple) # kind of tuple

True

Certains *n-uplets nommés* sont des types natifs (comme les exemples ci-dessus). Sinon, un *n-uplet nommé* peut être créé à partir d'une définition de classe habituelle qui hérite de [tuple](https://docs.python.org/fr/3.9/library/stdtypes.html#tuple) et qui définit les champs nommés. Une telle classe peut être écrite à la main ou être créée avec la fonction [collections.namedtuple()](https://docs.python.org/fr/3.9/library/collections.html#collections.namedtuple). Cette dernière méthode ajoute des méthodes supplémentaires qui ne seront pas trouvées dans celles écrites à la main ni dans les n-uplets nommés natifs.

espace de nommage

L'endroit où une variable est stockée. Les espaces de nommage sont implémentés avec des dictionnaires. Il existe des espaces de nommage globaux, natifs ou imbriqués dans les objets (dans les méthodes). Les espaces de nommage favorisent la modularité car ils permettent d'éviter les conflits de noms. Par exemple, les fonctions [builtins.open](https://docs.python.org/fr/3.9/library/functions.html#open) et [os.open()](https://docs.python.org/fr/3.9/library/os.html#os.open) sont différenciées par leurs espaces de nom. Les espaces de nommage aident aussi à la lisibilité et la maintenabilité en rendant clair quel module implémente une fonction. Par exemple, écrire [random.seed()](https://docs.python.org/fr/3.9/library/random.html#random.seed) ou [itertools.islice()](https://docs.python.org/fr/3.9/library/itertools.html#itertools.islice) affiche clairement que ces fonctions sont implémentées respectivement dans les modules [random](https://docs.python.org/fr/3.9/library/random.html#module-random) et [itertools](https://docs.python.org/fr/3.9/library/itertools.html#module-itertools).

paquet-espace de nommage

Un [paquet](https://docs.python.org/fr/3.9/glossary.html#term-package) tel que défini dans la [**PEP 421**](https://www.python.org/dev/peps/pep-0421) qui ne sert qu'à contenir des sous-paquets. Les paquets-espace de nommage peuvent n'avoir aucune représentation physique et, plus spécifiquement, ne sont pas comme un [paquet classique](https://docs.python.org/fr/3.9/glossary.html#term-regular-package) puisqu'ils n'ont pas de fichier \_\_init\_\_.py.

Voir aussi [module](https://docs.python.org/fr/3.9/glossary.html#term-module).

portée imbriquée

Possibilité de faire référence à une variable déclarée dans une définition englobante. Typiquement, une fonction définie à l'intérieur d'une autre fonction a accès aux variables de cette dernière. Souvenez-vous cependant que cela ne fonctionne que pour accéder à des variables, pas pour les assigner. Les variables locales sont lues et assignées dans l'espace de nommage le plus proche. Tout comme les variables globales qui sont stockés dans l'espace de nommage global, le mot clef [nonlocal](https://docs.python.org/fr/3.9/reference/simple_stmts.html#nonlocal) permet d'écrire dans l'espace de nommage dans lequel est déclarée la variable.

nouvelle classe

Ancien nom pour l'implémentation actuelle des classes, pour tous les objets. Dans les anciennes versions de Python, seules les nouvelles classes pouvaient utiliser les nouvelles fonctionnalités telles que [\_\_slots\_\_](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__slots__), les descripteurs, les propriétés, [\_\_getattribute\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__getattribute__), les méthodes de classe et les méthodes statiques.

objet

N'importe quelle donnée comportant des états (sous forme d'attributs ou d'une valeur) et un comportement (des méthodes). C'est aussi (object) l'ancêtre commun à absolument toutes les [nouvelles classes](https://docs.python.org/fr/3.9/glossary.html#term-new-style-class).

paquet

[module](https://docs.python.org/fr/3.9/glossary.html#term-module) Python qui peut contenir des sous-modules ou des sous-paquets. Techniquement, un paquet est un module qui possède un attribut \_\_path\_\_.

Voir aussi [paquet classique](https://docs.python.org/fr/3.9/glossary.html#term-regular-package) et [namespace package](https://docs.python.org/fr/3.9/glossary.html#term-namespace-package).

paramètre

Entité nommée dans la définition d'une [fonction](https://docs.python.org/fr/3.9/glossary.html#term-function) (ou méthode), décrivant un [argument](https://docs.python.org/fr/3.9/glossary.html#term-argument) (ou dans certains cas des arguments) que la fonction accepte. Il existe cinq sortes de paramètres :

* *positional-or-keyword* : l'argument peut être passé soit par sa [position](https://docs.python.org/fr/3.9/glossary.html#term-argument), soit en tant que [argument nommé](https://docs.python.org/fr/3.9/glossary.html#term-argument). C'est le type de paramètre par défaut. Par exemple, *foo* et *bar* dans l'exemple suivant :
* def func(foo, bar=None): ...
* *positional-only* : définit un argument qui ne peut être fourni que par position. Les paramètres *positional-only* peuvent être définis en insérant un caractère "/" dans la liste de paramètres de la définition de fonction après eux. Par exemple : *posonly1* et *posonly2* dans le code suivant :
* def func(posonly1, posonly2, /, positional\_or\_keyword): ...
* *keyword-only* : l'argument ne peut être fourni que nommé. Les paramètres *keyword-only* peuvent être définis en utilisant un seul paramètre *var-positional*, ou en ajoutant une étoile (\*) seule dans la liste des paramètres avant eux. Par exemple, *kw\_only1* et *kw\_only2* dans le code suivant :
* def func(arg, \*, kw\_only1, kw\_only2): ...
* *var-positional* : une séquence d'arguments positionnels peut être fournie (en plus de tous les arguments positionnels déjà acceptés par d'autres paramètres). Un tel paramètre peut être défini en préfixant son nom par une \*. Par exemple *args* ci-après :
* def func(\*args, \*\*kwargs): ...
* *var-keyword* : une quantité arbitraire d'arguments peut être passée, chacun étant nommé (en plus de tous les arguments nommés déjà acceptés par d'autres paramètres). Un tel paramètre est défini en préfixant le nom du paramètre par \*\*. Par exemple, *kwargs* ci-dessus.

Les paramètres peuvent spécifier des arguments obligatoires ou optionnels, ainsi que des valeurs par défaut pour les arguments optionnels.

Voir aussi [argument](https://docs.python.org/fr/3.9/glossary.html#term-argument) dans le glossaire, la question sur [la différence entre les arguments et les paramètres](https://docs.python.org/fr/3.9/faq/programming.html#faq-argument-vs-parameter) dans la FAQ, la classe [inspect.Parameter](https://docs.python.org/fr/3.9/library/inspect.html#inspect.Parameter), la section [Définition de fonctions](https://docs.python.org/fr/3.9/reference/compound_stmts.html#function) et la [**PEP 362**](https://www.python.org/dev/peps/pep-0362).

entrée de chemin

Emplacement dans le [chemin des importations](https://docs.python.org/fr/3.9/glossary.html#term-import-path) (*import path* en anglais, d'où le *path*) que le [chercheur basé sur les chemins](https://docs.python.org/fr/3.9/glossary.html#term-path-based-finder) consulte pour trouver des modules à importer.

chercheur de chemins

[chercheur](https://docs.python.org/fr/3.9/glossary.html#term-finder) renvoyé par un appelable sur un [sys.path\_hooks](https://docs.python.org/fr/3.9/library/sys.html#sys.path_hooks) (c'est-à-dire un [point d'entrée pour la recherche dans path](https://docs.python.org/fr/3.9/glossary.html#term-path-entry-hook)) qui sait où trouver des modules lorsqu'on lui donne une [entrée de path](https://docs.python.org/fr/3.9/glossary.html#term-path-entry).

Voir [importlib.abc.PathEntryFinder](https://docs.python.org/fr/3.9/library/importlib.html#importlib.abc.PathEntryFinder) pour les méthodes qu'un chercheur d'entrée dans *path* doit implémenter.

point d'entrée pour la recherche dans *path*

Appelable dans la liste sys.path\_hook qui donne un [chercheur d'entrée dans path](https://docs.python.org/fr/3.9/glossary.html#term-path-entry-finder) s'il sait où trouver des modules pour une [entrée dans path](https://docs.python.org/fr/3.9/glossary.html#term-path-entry) donnée.

chercheur basé sur les chemins

L'un des [chercheurs dans les méta-chemins](https://docs.python.org/fr/3.9/glossary.html#term-meta-path-finder) par défaut qui cherche des modules dans un [chemin des importations](https://docs.python.org/fr/3.9/glossary.html#term-import-path).

objet simili-chemin

Objet représentant un chemin du système de fichiers. Un objet simili-chemin est un objet [str](https://docs.python.org/fr/3.9/library/stdtypes.html#str) ou un objet [bytes](https://docs.python.org/fr/3.9/library/stdtypes.html#bytes) représentant un chemin ou un objet implémentant le protocole [os.PathLike](https://docs.python.org/fr/3.9/library/os.html#os.PathLike). Un objet qui accepte le protocole [os.PathLike](https://docs.python.org/fr/3.9/library/os.html#os.PathLike) peut être converti en un chemin [str](https://docs.python.org/fr/3.9/library/stdtypes.html#str) ou [bytes](https://docs.python.org/fr/3.9/library/stdtypes.html#bytes) du système de fichiers en appelant la fonction [os.fspath()](https://docs.python.org/fr/3.9/library/os.html#os.fspath). [os.fsdecode()](https://docs.python.org/fr/3.9/library/os.html#os.fsdecode) et [os.fsencode()](https://docs.python.org/fr/3.9/library/os.html#os.fsencode) peuvent être utilisées, respectivement, pour garantir un résultat de type [str](https://docs.python.org/fr/3.9/library/stdtypes.html#str) ou [bytes](https://docs.python.org/fr/3.9/library/stdtypes.html#bytes) à la place. A été Introduit par la [**PEP 519**](https://www.python.org/dev/peps/pep-0519).

PEP

*Python Enhancement Proposal* (Proposition d'amélioration Python). Un PEP est un document de conception fournissant des informations à la communauté Python ou décrivant une nouvelle fonctionnalité pour Python, ses processus ou son environnement. Les PEP doivent fournir une spécification technique concise et une justification des fonctionnalités proposées.

Les PEPs sont censés être les principaux mécanismes pour proposer de nouvelles fonctionnalités majeures, pour recueillir les commentaires de la communauté sur une question et pour documenter les décisions de conception qui sont intégrées en Python. L’auteur du PEP est responsable de l’établissement d’un consensus au sein de la communauté et de documenter les opinions contradictoires.

Voir [**PEP 1**](https://www.python.org/dev/peps/pep-0001).

portion

Jeu de fichiers dans un seul dossier (pouvant être stocké sous forme de fichier zip) qui contribue à l'espace de nommage d'un paquet, tel que défini dans la [**PEP 420**](https://www.python.org/dev/peps/pep-0420).

argument positionnel

Voir [argument](https://docs.python.org/fr/3.9/glossary.html#term-argument).

API provisoire

Une API provisoire est une API qui n'offre aucune garantie de rétrocompatibilité (la bibliothèque standard exige la rétrocompatibilité). Bien que des changements majeurs d'une telle interface ne soient pas attendus, tant qu'elle est étiquetée provisoire, des changements cassant la rétrocompatibilité (y compris sa suppression complète) peuvent survenir si les développeurs principaux le jugent nécessaire. Ces modifications ne surviendront que si de sérieux problèmes sont découverts et qu'ils n'avaient pas été identifiés avant l'ajout de l'API.

Même pour les API provisoires, les changements cassant la rétrocompatibilité sont considérés comme des "solutions de dernier recours". Tout ce qui est possible sera fait pour tenter de résoudre les problèmes en conservant la rétrocompatibilité.

Ce processus permet à la bibliothèque standard de continuer à évoluer avec le temps, sans se bloquer longtemps sur des erreurs d'architecture. Voir la [**PEP 411**](https://www.python.org/dev/peps/pep-0411) pour plus de détails.

paquet provisoire

Voir [provisional API](https://docs.python.org/fr/3.9/glossary.html#term-provisional-api).

Python 3000

Surnom donné à la série des Python 3.x (très vieux surnom donné à l'époque où Python 3 représentait un futur lointain). Aussi abrégé *Py3k*.

*Pythonique*

Idée, ou bout de code, qui colle aux idiomes de Python plutôt qu'aux concepts communs rencontrés dans d'autres langages. Par exemple, il est idiomatique en Python de parcourir les éléments d'un itérable en utilisant [for](https://docs.python.org/fr/3.9/reference/compound_stmts.html#for). Beaucoup d'autres langages n'ont pas cette possibilité, donc les gens qui ne sont pas habitués à Python utilisent parfois un compteur numérique à la place :

for i in range(len(food)):

print(food[i])

Plutôt qu'utiliser la méthode, plus propre et élégante, donc *Pythonique* :

for piece in food:

print(piece)

nom qualifié

Nom, comprenant des points, montrant le "chemin" de l'espace de nommage global d'un module vers une classe, fonction ou méthode définie dans ce module, tel que défini dans la [**PEP 3155**](https://www.python.org/dev/peps/pep-3155). Pour les fonctions et classes de premier niveau, le nom qualifié est le même que le nom de l'objet :

>>>

>>> class C:

... class D:

... def meth(self):

... pass

...

>>> C.\_\_qualname\_\_

'C'

>>> C.D.\_\_qualname\_\_

'C.D'

>>> C.D.meth.\_\_qualname\_\_

'C.D.meth'

Lorsqu'il est utilisé pour nommer des modules, le *nom qualifié complet* (*fully qualified name - FQN* en anglais) signifie le chemin complet (séparé par des points) vers le module, incluant tous les paquets parents. Par exemple : email.mime.text :

>>>

>>> import email.mime.text

>>> email.mime.text.\_\_name\_\_

'email.mime.text'

nombre de références

Nombre de références à un objet. Lorsque le nombre de références à un objet descend à zéro, l'objet est désalloué. Le comptage de référence n'est généralement pas visible dans le code Python, mais c'est un élément clé de l'implémentation [CPython](https://docs.python.org/fr/3.9/glossary.html#term-cpython). Le module [sys](https://docs.python.org/fr/3.9/library/sys.html#module-sys) définit une fonction [getrefcount()](https://docs.python.org/fr/3.9/library/sys.html#sys.getrefcount) que les développeurs peuvent utiliser pour obtenir le nombre de références à un objet donné.

paquet classique

[paquet](https://docs.python.org/fr/3.9/glossary.html#term-package) traditionnel, tel qu'un dossier contenant un fichier \_\_init\_\_.py.

Voir aussi [paquet-espace de nommage](https://docs.python.org/fr/3.9/glossary.html#term-namespace-package).

\_\_slots\_\_

Déclaration dans une classe qui économise de la mémoire en pré-allouant de l'espace pour les attributs des instances et qui élimine le dictionnaire (des attributs) des instances. Bien que populaire, cette technique est difficile à maîtriser et devrait être réservée à de rares cas où un grand nombre d'instances dans une application devient un sujet critique pour la mémoire.

séquence

[itérable](https://docs.python.org/fr/3.9/glossary.html#term-iterable) qui offre un accès efficace à ses éléments par un indice sous forme de nombre entier via la méthode spéciale [\_\_getitem\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__getitem__) et qui définit une méthode [\_\_len\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__len__) donnant sa taille. Voici quelques séquences natives : [list](https://docs.python.org/fr/3.9/library/stdtypes.html#list), [str](https://docs.python.org/fr/3.9/library/stdtypes.html#str), [tuple](https://docs.python.org/fr/3.9/library/stdtypes.html#tuple), et [bytes](https://docs.python.org/fr/3.9/library/stdtypes.html#bytes). Notez que [dict](https://docs.python.org/fr/3.9/library/stdtypes.html#dict) possède aussi une méthode [\_\_getitem\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__getitem__) et une méthode [\_\_len\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__len__), mais il est considéré comme un *mapping* plutôt qu'une séquence, car ses accès se font par une clé arbitraire [immuable](https://docs.python.org/fr/3.9/glossary.html#term-immutable) plutôt qu'un nombre entier.

The [collections.abc.Sequence](https://docs.python.org/fr/3.9/library/collections.abc.html#collections.abc.Sequence) abstract base class defines a much richer interface that goes beyond just [\_\_getitem\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__getitem__) and [\_\_len\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__len__), adding count(), index(), [\_\_contains\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__contains__), and [\_\_reversed\_\_()](https://docs.python.org/fr/3.9/reference/datamodel.html#object.__reversed__). Types that implement this expanded interface can be registered explicitly using [register()](https://docs.python.org/fr/3.9/library/abc.html#abc.ABCMeta.register).

set comprehension

A compact way to process all or part of the elements in an iterable and return a set with the results. results = {c for c in 'abracadabra' if c not in 'abc'} generates the set of strings {'r', 'd'}. See [Agencements des listes, ensembles et dictionnaires](https://docs.python.org/fr/3.9/reference/expressions.html#comprehensions).

distribution simple

Forme de distribution, comme les [fonction génériques](https://docs.python.org/fr/3.9/glossary.html#term-generic-function), où l'implémentation est choisie en fonction du type d'un seul argument.

tranche

(*slice* en anglais), un objet contenant habituellement une portion de [séquence](https://docs.python.org/fr/3.9/glossary.html#term-sequence). Une tranche est créée en utilisant la notation [] avec des : entre les nombres lorsque plusieurs sont fournis, comme dans variable\_name[1:3:5]. Cette notation utilise des objets [slice](https://docs.python.org/fr/3.9/library/functions.html#slice) en interne.

méthode spéciale

(*special method* en anglais) Méthode appelée implicitement par Python pour exécuter une opération sur un type, comme une addition. De telles méthodes ont des noms commençant et terminant par des doubles tirets bas. Les méthodes spéciales sont documentées dans [Méthodes spéciales](https://docs.python.org/fr/3.9/reference/datamodel.html#specialnames).

instruction

Une instruction (*statement* en anglais) est un composant d'un "bloc" de code. Une instruction est soit une [expression](https://docs.python.org/fr/3.9/glossary.html#term-expression), soit une ou plusieurs constructions basées sur un mot-clé, comme [if](https://docs.python.org/fr/3.9/reference/compound_stmts.html#if), [while](https://docs.python.org/fr/3.9/reference/compound_stmts.html#while) ou [for](https://docs.python.org/fr/3.9/reference/compound_stmts.html#for).

encodage de texte

Codec (codeur-décodeur) qui convertit des chaînes de caractères Unicode en octets (classe *bytes*).

fichier texte

[file object](https://docs.python.org/fr/3.9/glossary.html#term-file-object) capable de lire et d'écrire des objets [str](https://docs.python.org/fr/3.9/library/stdtypes.html#str). Souvent, un fichier texte (*text file* en anglais) accède en fait à un flux de donnée en octets et gère l'[text encoding](https://docs.python.org/fr/3.9/glossary.html#term-text-encoding) automatiquement. Des exemples de fichiers textes sont les fichiers ouverts en mode texte ('r' ou 'w'), [sys.stdin](https://docs.python.org/fr/3.9/library/sys.html#sys.stdin), [sys.stdout](https://docs.python.org/fr/3.9/library/sys.html#sys.stdout) et les instances de [io.StringIO](https://docs.python.org/fr/3.9/library/io.html#io.StringIO).

Voir aussi [binary file](https://docs.python.org/fr/3.9/glossary.html#term-binary-file) pour un objet fichier capable de lire et d'écrire [bytes-like objects](https://docs.python.org/fr/3.9/glossary.html#term-bytes-like-object).

chaîne entre triple guillemets

Chaîne qui est délimitée par trois guillemets simples (') ou trois guillemets doubles ("). Bien qu'elle ne fournisse aucune fonctionnalité qui ne soit pas disponible avec une chaîne entre guillemets, elle est utile pour de nombreuses raisons. Elle vous autorise à insérer des guillemets simples et doubles dans une chaîne sans avoir à les protéger et elle peut s'étendre sur plusieurs lignes sans avoir à terminer chaque ligne par un \. Elle est ainsi particulièrement utile pour les chaînes de documentation (*docstrings*).

type

Le type d'un objet Python détermine quel genre d'objet c'est. Tous les objets ont un type. Le type d'un objet peut être obtenu via son attribut [\_\_class\_\_](https://docs.python.org/fr/3.9/library/stdtypes.html#instance.__class__) ou via type(obj).

alias de type

Synonyme d'un type, créé en affectant le type à un identifiant.

Les alias de types sont utiles pour simplifier les [indications de types](https://docs.python.org/fr/3.9/glossary.html#term-type-hint). Par exemple :

def remove\_gray\_shades(

colors: list[tuple[int, int, int]]) -> list[tuple[int, int, int]]:

pass

pourrait être rendu plus lisible comme ceci :

Color = tuple[int, int, int]

def remove\_gray\_shades(colors: list[Color]) -> list[Color]:

pass

Voir [typing](https://docs.python.org/fr/3.9/library/typing.html#module-typing) et [**PEP 484**](https://www.python.org/dev/peps/pep-0484), qui décrivent cette fonctionnalité.

indication de type

Le [annotation](https://docs.python.org/fr/3.9/glossary.html#term-annotation) qui spécifie le type attendu pour une variable, un attribut de classe, un paramètre de fonction ou une valeur de retour.

Les indications de type sont facultatives et ne sont pas indispensables à l'interpréteur Python, mais elles sont utiles aux outils d'analyse de type statique et aident les IDE à compléter et à réusiner (*code refactoring* en anglais) le code.

Les indicateurs de type de variables globales, d'attributs de classe et de fonctions, mais pas de variables locales, peuvent être consultés en utilisant [typing.get\_type\_hints()](https://docs.python.org/fr/3.9/library/typing.html#typing.get_type_hints).

Voir [typing](https://docs.python.org/fr/3.9/library/typing.html#module-typing) et [**PEP 484**](https://www.python.org/dev/peps/pep-0484), qui décrivent cette fonctionnalité.

retours à la ligne universels

Une manière d'interpréter des flux de texte dans lesquels sont reconnues toutes les fins de ligne suivantes : la convention Unix '\n', la convention Windows '\r\n' et l'ancienne convention Macintosh '\r'. Voir la [**PEP 278**](https://www.python.org/dev/peps/pep-0278) et la [**PEP 3116**](https://www.python.org/dev/peps/pep-3116), ainsi que la fonction [bytes.splitlines()](https://docs.python.org/fr/3.9/library/stdtypes.html#bytes.splitlines) pour d'autres usages.

annotation de variable

[annotation](https://docs.python.org/fr/3.9/glossary.html#term-annotation) d'une variable ou d'un attribut de classe.

Lorsque vous annotez une variable ou un attribut de classe, l'affectation est facultative :

class C:

field: 'annotation'

Les annotations de variables sont généralement utilisées pour des [indications de types](https://docs.python.org/fr/3.9/glossary.html#term-type-hint) : par exemple, cette variable devrait prendre des valeurs de type [int](https://docs.python.org/fr/3.9/library/functions.html#int) :

count: int = 0

La syntaxe d'annotation de la variable est expliquée dans la section [Les assignations annotées](https://docs.python.org/fr/3.9/reference/simple_stmts.html#annassign).

Reportez-vous à [function annotation](https://docs.python.org/fr/3.9/glossary.html#term-function-annotation), à la [**PEP 484**](https://www.python.org/dev/peps/pep-0484) et à la [**PEP 526**](https://www.python.org/dev/peps/pep-0526) qui décrivent cette fonctionnalité.

environnement virtuel

Environnement d'exécution isolé (en mode coopératif) qui permet aux utilisateurs de Python et aux applications d'installer et de mettre à jour des paquets sans interférer avec d'autres applications Python fonctionnant sur le même système.

Voir aussi [venv](https://docs.python.org/fr/3.9/library/venv.html#module-venv).

machine virtuelle

Ordinateur défini entièrement par du logiciel. La machine virtuelle (*virtual machine*) de Python exécute le [bytecode](https://docs.python.org/fr/3.9/glossary.html#term-bytecode) produit par le compilateur de *bytecode*.

Le zen de Python

Liste de principes et de préceptes utiles pour comprendre et utiliser le langage. Cette liste peut être obtenue en tapant "import this" dans une invite Python interactive.