Python Setup and Usage

Version 3.6.11

Guido van Rossum
and the Python development team

juin 29, 2020

Python Software Foundation
Email : docs@python.org

Table des matières

Cette partie de la documentation est consacrée aux informations générales au sujet de l'installation de l'environnement Python sur différentes plateformes, l'invocation de l'interpréteur et des choses qui facilitent le travail avec Python.

CHAPITRE 1

Ligne de commande et environnement

L'interpréteur CPython analyse la ligne de commande et l'environnement à la recherche de différents paramètres.

CPython implementation detail : Le format des lignes de commande utilisé par d'autres implémentations peut s'avérer différent. Voir implementations pour plus d'informations.

1.1 Ligne de commande

Quand vous invoquez Python, vous pouvez spécifier n'importe laquelle de ces options :

```
python [-bBdEhiIOqsSuvVWx?] [-c command | -m module-name | script | - ] [args]
```

Le cas d'utilisation le plus courant est, bien entendu, la simple invocation d'un script :

```
python myscript.py
```

1.1.1 Options de l'interface

L'interface de l'interpréteur ressemble à celle du shell UNIX mais fournit quelques méthodes d'invocation supplémentaires :

- Quand l'interpréteur est appelé avec l'entrée standard connectée à un périphérique tty, il lit les lignes de commande et les exécute jusqu'à ce qu'un caractère EOF (caractère fin de fichier, que vous pouvez produire avec `Ctrl-D` sous UNIX ou `Ctrl-Z, Enter` sous Windows) soit lu.
- Quand l'interpréteur est appelé avec un argument correspondant à un nom de fichier ou avec un fichier comme entrée standard, il lit et exécute le script contenu dans ce fichier.
- Quand l'interpréteur est appelé avec un argument correspondant à un répertoire, il lit et exécute un script d'un certain nom dans ce répertoire.
- Quand l'interpréteur est appelé avec l'option `-c commande`, il exécute la ou les instructions Python données comme *commande*. Ici *commande* peut contenir plusieurs instructions séparées par des fins de ligne. Les blancs en début de ligne ne sont pas ignorés dans les instructions Python !

— Quand l'interpréteur est appelé avec l'option `-m nom-de-module`, le module donné est recherché dans le chemin des modules Python et est exécuté en tant que script.

En mode non-interactif, toute l'entrée est analysée avant d'être exécutée.

Une option d'interface termine la liste des options consommées par l'interpréteur ; tous les arguments atterrissent dans `sys.argv` — notez que le premier élément, à l'indice zéro (`sys.argv[0]`), est une chaîne de caractères indiquant la source du programme.

`-c` `<command>`

Exécute le code Python dans *command*. *command* peut être une ou plusieurs instructions, séparées par des fins de ligne, dont les espaces en début de ligne sont significatives, comme dans le code d'un module.

Si cette option est donnée, le premier élément de `sys.argv` est `"-c"` et le répertoire courant est ajouté au début de `sys.path` (permettant aux modules de ce répertoire d'être importés comme des modules de premier niveau).

`-m` `<module-name>`

Parcourt `sys.path` à la recherche du module donné et exécute son contenu en tant que module `__main__`.

L'argument étant un nom de *module*, vous ne devez pas fournir d'extension de fichier (`.py`). Le nom du module doit être un nom de module Python valide, absolu, mais l'implémentation peut ne pas l'imposer (par exemple, l'utilisation d'un trait d'union peut être autorisée).

Les noms de paquets sont aussi autorisés (ainsi que les paquets-espace de nommage, *namespace packages* en anglais). Quand un nom de paquet est donné à la place d'un simple module, l'interpréteur exécute `<pkg>.__main__` comme module principal. Ce comportement est délibérément identique au traitement d'un dossier ou d'un fichier zip donné en argument à l'interpréteur comme script.

Note : Cette option ne peut pas être utilisée avec les modules natifs et les modules d'extension écrits en C, étant donné qu'ils ne possèdent pas de fichiers modules en Python. Cependant, elle peut toujours être utilisée pour les modules pré-compilés, même si le fichier source original n'est pas disponible.

Si cette option est donnée, le premier élément de `sys.argv` est le chemin complet d'accès au fichier du module (pendant que le fichier est recherché, le premier élément est mis à `"-m"`). Comme avec l'option `-c`, le dossier courant est ajouté au début de `sys.path`.

De nombreux modules de la bibliothèque standard contiennent du code qui est invoqué quand ils sont exécutés comme scripts. Un exemple est le module `timeit` :

```
python -mtimeit -s 'setup here' 'benchmarked code here'
python -mtimeit -h # for details
```

Voir aussi :

`runpy.run_module()` Fonctionnalité équivalente directement disponible en code Python

PEP 338 – Exécuter des modules en tant que scripts

Modifié dans la version 3.1 : Fournir le nom d'un paquet pour exécuter un sous-module `__main__`.

Modifié dans la version 3.4 : les paquets-espaces de nommage sont aussi gérés

`-`

Lit les commandes depuis l'entrée standard (`sys.stdin`). Si l'entrée standard est un terminal, l'option `-i` est activée implicitement.

Si cette option est donnée, le premier élément de `sys.argv` est `"-"` et le dossier courant est ajouté au début de `sys.path`.

`<script>`

Exécute le code Python contenu dans *script*, qui doit être un chemin d'accès (absolu ou relatif) à un fichier, faisant référence à un fichier Python, à un répertoire contenant un fichier `__main__.py` ou à un fichier zip contenant un fichier `__main__.py`.

Si cette option est donnée, le premier élément de `sys.argv` est le nom du script tel que donné sur la ligne de commande.

Si le nom du script se réfère directement à un fichier Python, le répertoire contenant ce fichier est ajouté au début de `sys.path` et le fichier est exécuté en tant que module `__main__`.

Si le nom du script fait référence à un dossier ou à un fichier zip, le nom du script est ajouté au début de `sys.path` et le fichier `__main__.py` à cet endroit est exécuté en tant que module `__main__`.

Voir aussi :

`runpy.run_path()` Fonctionnalité équivalente directement disponible en code Python

Si aucune option d'interface n'est donnée, l'option *`-i`* est activée implicitement, `sys.argv[0]` est une chaine vide (`""`) et le dossier courant est ajouté au début de `sys.path`. Aussi, la complétion par tabulation et l'édition de l'historique sont automatiquement activés, s'ils sont disponibles sur votre système (voir rlcompleter-config).

Voir aussi :

tut-invoking

Modifié dans la version 3.4 : Activation automatique de la complétion par tabulation et édition de l'historique.

1.1.2 Options génériques

`-?`
`-h`
`--help`
Affiche une brève description de toutes les options de la ligne de commande.

`-V`
`--version`
Affiche la version de Python et termine. Par exemple :

```
Python 3.6.0b2+
```

Lorsque l'option est doublée, affiche davantage d'informations sur la manière dont Python a été compilé, comme :

```
Python 3.6.0b2+ (3.6:84a3c5003510+, Oct 26 2016, 02:33:55)
[GCC 6.2.0 20161005]
```

Nouveau dans la version 3.6 : L'option `-VV`.

1.1.3 Options diverses

`-b`
Affiche un avertissement (*warning* en anglais) lors d'une comparaison d'un objet de type `bytes` ou `bytearray` avec un objet de type `str` ou un objet de type `bytes` avec un objet de type `int`. Lève une erreur si cette option est doublée (`-bb`).

Modifié dans la version 3.5 : Concerne les comparaisons de `bytes` avec `int`.

`-B`
S'il est donné, Python ne tente pas d'écrire de fichier `.pyc` ou `.pyo` à l'importation des modules sources. Voir aussi *`PYTHONDONTWRITEBYTECODE`*.

`-d`
Active la sortie de l'analyseur en mode débogage (pour les experts uniquement, en fonction des options de compilation). Voir aussi *`PYTHONDEBUG`*.

`-E`
Ignore toutes les variables d'environnement `PYTHON*` qui pourraient être définies. Par exemple, *`PYTHONPATH`* et *`PYTHONHOME`*.

-i

Quand un script est passé comme premier argument ou que l'option *-c* est utilisée, entre en mode interactif après avoir exécuté le script ou la commande, même lorsque `sys.stdin` ne semble pas être un terminal. Le fichier *PYTHONSTARTUP* n'est pas lu.

Cela peut être utile pour examiner les variables globales ou une trace de la pile lorsque le script lève une exception. Voir aussi *PYTHONINSPECT*.

-I

Lance Python en mode isolé. Cela implique aussi *-E* et *-s*. En mode isolé, `sys.path` ne contient ni le répertoire du script ni le répertoire *site-packages* de l'utilisateur. Toutes les variables d'environnement `PYTHON*` sont aussi ignorées. Davantage de restrictions peuvent être imposées pour éviter que l'utilisateur n'injecte du code malicieux.

Nouveau dans la version 3.4.

-O

Enlève les instructions *assert* et tout le code qui dépend de la valeur de `__debug__`. Ajoute `.opt-1` au nom de fichier du code intermédiaire (*bytecode*), avant l'extension `.pyc` (voir la **PEP 488**). Voir aussi *PYTHONOPTIMIZE*.

Modifié dans la version 3.5 : Modifie les noms de fichiers `.pyc` suivant la **PEP 488**.

-OO

Agit comme *-O* et ignore aussi les *docstrings*. Ajoute `.opt-2` au nom de fichier du code intermédiaire (*bytecode*), avant l'extension `.pyc` (voir la **PEP 488**).

Modifié dans la version 3.5 : Modifie les noms de fichiers `.pyc` suivant la **PEP 488**.

-q

N'affiche pas le copyright et la version, même en mode interactif.

Nouveau dans la version 3.2.

-R

Conservé pour compatibilité ascendante. Sur Python 3.3 ou supérieur, la randomisation des empreintes (*hash* en anglais) est activée par défaut.

Sur les versions précédentes de Python, cette option activait la randomisation des empreintes de manière à ce que les `__hash__()` de chaînes, bytes et `datetime` soient « salés » avec une valeur aléatoire non prévisible. Bien que ce sel soit constant durant le déroulement d'un processus Python, il n'est pas prévisible pour des invocations répétées de code Python.

L'imprévisibilité des empreintes a pour objectif de se protéger contre les dénis de service qui utiliseraient des valeurs d'entrée judicieusement choisies afin de forcer la construction des dictionnaires dans le pire cas, c'est-à-dire avec une complexité en $O(n^2)$. Voir http://www.ocert.org/advisories/ocert-2011-003.html pour obtenir les détails.

PYTHONHASHSEED vous permet de définir vous-même la valeur du sel pour l'algorithme de calcul des empreintes.

Nouveau dans la version 3.2.3.

-s

N'ajoute pas le répertoire utilisateur `site-packages` à `sys.path`.

Voir aussi :

PEP 370 – Répertoire site-packages propre à l'utilisateur

-S

Désactive l'importation du module `site` et les modifications locales de `sys.path` qu'il implique. Désactive aussi ces manipulations si `site` est importé explicitement plus tard (appelez `site.main()` si vous voulez les déclencher).

-u

Désactive les mémoires tampons de la couche binaire des flux de la sortie standard (*stdout*) et de la sortie d'erreurs (*stderr*) (ils restent accessibles *via* leur attribut `buffer`). La couche d'entrée-sortie est mise en tampon ligne par ligne lors de l'écriture sur la console, ou par blocs si elle est redirigée sur un fichier non-interactif.

Voir aussi *PYTHONUNBUFFERED*.

-v

Affiche un message chaque fois qu'un module est initialisé, montrant l'emplacement (nom du fichier ou module natif) à partir duquel il est chargé. Lorsque l'option est doublée (`-vv`), affiche un message pour chaque fichier vérifié lors de la recherche du module. Fournit aussi des informations sur le nettoyage des modules à la fin. Voir aussi *PYTHONVERBOSE*.

-W `arg`

Contrôle des avertissements. Le mécanisme d'avertissement de Python, par défaut, affiche les messages d'avertissement sur `sys.stderr`. Un message d'avertissement type est de la forme suivante :

```
file:line: category: message
```

Par défaut, chaque avertissement est affiché une seule fois pour chaque ligne de source où il se trouve. Cette option définit à quelle fréquence afficher ces avertissements.

L'option *-W* peut être répétée ; lorsqu'un avertissement correspond à plus d'une option, l'action associée à la dernière correspondance est effectuée. Les options *-W* invalides sont ignorées (cependant, un message d'avertissement est affiché sur les options invalides au moment où le premier avertissement est généré).

Les avertissements peuvent aussi être contrôlés dans le programme Python en utilisant le module `warnings`.

La forme la plus simple de l'argument est l'une des chaînes d'actions suivantes (ou une abréviation unique) :

ignore Ignore tous les avertissements.

default Demande explicitement le comportement par défaut (affiche chaque avertissement une fois par ligne de code source).

all Affiche un avertissement à chaque fois qu'il se produit (ce qui peut générer beaucoup de messages si l'avertissement est déclenché à plusieurs reprises, comme à l'intérieur d'une boucle).

module Affiche chaque avertissement uniquement la première fois qu'il apparaît dans chaque module.

once Affiche chaque avertissement uniquement la première fois qu'il apparaît dans le programme.

error Déclenche une exception au lieu d'afficher un message d'avertissement.

La forme complète de l'argument est :

```
action:message:category:module:line
```

Ici, *action* est tel qu'expliqué ci-dessus, mais s'applique uniquement aux messages qui correspondent aux champs restants. Les champs vides correspondent à toutes les valeurs ; les champs vides de fin peuvent être omis. Le champ *message* correspond au début du message d'avertissement affiché, cette expression est insensible à la casse. Le champ *category* correspond à la catégorie d'avertissement. Ce nom doit être un nom complet de classe ; La règle s'applique à tous les messages d'alertes construits avec une classe qui hérite de celle spécifiée. Le nom de la classe complète doit être donnée. Le champ *module* correspond au nom (pleinement qualifié) du module, cette correspondance est sensible à la casse. Le champ *line* correspond au numéro de ligne, où zéro correspond à n'importe quel numéro de ligne et correspond donc à l'option par défaut.

Voir aussi :

`warnings` – le module qui gère les avertissements.

PEP 230 – Gestion des alertes

PYTHONWARNINGS

-x

Saute la première ligne du code source, autorisant ainsi les directives de type `#!cmd` non conformes au standard Unix. L'objectif est de proposer une astuce spécifique pour le DOS.

-X

Réservée pour les options spécifiques aux différentes implémentations. CPython reconnaît actuellement les valeurs suivantes :

— `-X faulthandler` pour activer `faulthandler` ;

— `-X showrefcount` pour afficher le compteur des références et le nombre de blocs mémoire utilisés lorsque le programme se termine ou après chaque entrée de l'interpréteur interactif. Ceci ne fonctionne que sur les versions compilées en mode débogage.

— `-X tracemalloc` pour lancer le suivi des allocations mémoire par Python en utilisant le module `tracemalloc`. Par défaut, seul l'appel (la *frame* en anglais) le plus récent est stocké dans la trace de la pile d'appels. Utilisez `-X tracemalloc=NFRAME` pour lancer le suivi avec une limite des traces à *NFRAME* appels. Voir `tracemalloc.start()` pour plus d'informations.

— `-X showalloccount` pour afficher à la fin de l'exécution du programme le total des objets alloués pour chaque type. Ceci ne fonctionne que si Python a été compilé avec l'option `COUNT_ALLOCS`.

Il est aussi possible de passer des valeurs arbitraires et de les récupérer par le dictionnaire `sys._xoptions`.

Modifié dans la version 3.2 : L'option *-X* a été ajoutée.

Nouveau dans la version 3.3 : L'option `-X faulthandler`.

Nouveau dans la version 3.4 : Les options `-X showrefcount` et `-X tracemalloc`.

Nouveau dans la version 3.6 : L'option `-X showalloccount`.

1.1.4 Options à ne pas utiliser

-J

Utilisation réservée à Jython.

1.2 Variables d'environnement

Les variables d'environnement suivantes modifient le comportement de Python. Elles sont analysées avant les options de la ligne de commande, autres que *-E* ou *-I*. Il est d'usage que les options de la ligne de commande prennent le pas sur les variables d'environnement en cas de conflit.

PYTHONHOME

Modifie l'emplacement des bibliothèques standards de Python. Par défaut, les bibliothèques sont recherchées dans *préfixe*`/lib/python`*version* et *préfixe_exec*`/lib/python`*version* où *préfixe* et *préfixe_exec* sont des répertoires qui dépendent de l'installation (leur valeur par défaut étant `/usr/local`).

Quand *PYTHONHOME* est défini à un simple répertoire, sa valeur remplace à la fois *préfixe* et *préfixe_exec*. Pour spécifier des valeurs différentes à ces variables, définissez *PYTHONHOME* à *préfixe*`:`*préfixe_exec*.

PYTHONPATH

Augmente le chemin de recherche par défaut des fichiers de modules. Le format est le même que pour `PATH` du shell : un ou plusieurs chemins de répertoires séparés par `os.pathsep` (par exemple, le caractère deux-points sous Unix et point-virgule sous Windows). Les répertoires qui n'existent pas sont ignorés silencieusement.

En plus des répertoires normaux, des entrées individuelles de *PYTHONPATH* peuvent faire référence à des fichiers zip contenant des modules en pur Python (soit sous forme de code source, soit sous forme compilée). Les modules d'extensions ne peuvent pas être importés à partir de fichiers zip.

Le chemin de recherche par défaut dépend de l'installation mais commence généralement par *préfixe*`/lib/python`*version* (voir *PYTHONHOME* ci-dessus). Il est *toujours* ajouté à *PYTHONPATH*.

Comme indiqué ci-dessus dans *Options de l'interface*, un répertoire supplémentaire est inséré dans le chemin de recherche devant *PYTHONPATH*. Le chemin de recherche peut être manipulé depuis un programme Python avec la variable `sys.path`.

PYTHONSTARTUP

S'il s'agit d'un nom de fichier accessible en lecture, les commandes Python de ce fichier sont exécutées avant que la première invite ne soit affichée en mode interactif. Le fichier est exécuté dans le même espace de nommage que les commandes interactives, de manière à ce que les objets définis ou importés puissent être utilisés sans qualificatif dans la session interactive. Vous pouvez aussi changer les invites `sys.ps1` et `sys.ps2` ainsi que le point d'entrée (*hook* en anglais) `sys.__interactivehook__` dans ce fichier.

PYTHONOPTIMIZE

Si elle est définie à une chaîne non vide, c'est équivalent à spécifier l'option *-O*. Si elle est définie à un entier, c'est équivalent à spécifier l'option *-O* plusieurs fois.

PYTHONDEBUG
Si elle est définie à une chaîne non vide, c'est équivalent à spécifier l'option *-d*. Si elle est définie à un entier, c'est équivalent à spécifier l'option *-d* plusieurs fois.

PYTHONINSPECT
Si elle est définie à une chaîne non vide, c'est équivalent à spécifier l'option *-i*.

Cette variable peut aussi être modifiée par du code Python en utilisant `os.environ` pour forcer le mode introspectif à la fin du programme.

PYTHONUNBUFFERED
Si elle est définie à une chaîne non vide, c'est équivalent à spécifier l'option *-u*.

PYTHONVERBOSE
Si elle est définie à une chaîne non vide, c'est équivalent à spécifier l'option *-v*. Si elle est définie à un entier, c'est équivalent à spécifier l'option *-v* plusieurs fois.

PYTHONCASEOK
Si elle est définie, Python ignore la casse dans les instructions `import`. Ceci ne fonctionne que sous Windows et OS X.

PYTHONDONTWRITEBYTECODE
Si elle est définie et n'est pas une chaîne vide, Python n'écrit pas de fichier `.pyc` à l'importation des modules sources. C'est équivalent à spécifier l'option *-B*.

PYTHONHASHSEED
Si cette variable n'est pas définie ou définie à `random`, une valeur aléatoire est utilisée pour saler les empreintes des objets chaines, bytes et `datetime`.

Si *PYTHONHASHSEED* est définie à une valeur entière, elle est utilisée comme valeur de salage pour générer les empreintes des types utilisant l'imprévisibilité du hachage.

L'objectif est d'avoir des empreintes reproductibles, pour des tests de l'interpréteur lui-même ou pour qu'un groupe de processus Python puisse partager des empreintes.

Le nombre entier doit être écrit en base 10 et être compris entre 0 et 4 294 967 295. Spécifier la valeur 0 désactive l'imprévisibilité des empreintes.

Nouveau dans la version 3.2.3.

PYTHONIOENCODING
Si la variable est définie sous la forme `nom_encodage:gestionnaire_erreur` avant le lancement de l'interpréteur, cela prend le pas sur l'encodage utilisé pour l'entrée standard, la sortie standard ou la sortie d'erreur. `nom_encodage` et `:gestionnaire_erreur` sont facultatifs tous les deux et possèdent la même signification que dans `str.encode()`.

Pour la sortie d'erreur, la partie `:gestionnaire_erreur` est ignorée : le gestionnaire est toujours `'backslashreplace'`.

Modifié dans la version 3.4 : La partie `nom_encodage` est maintenant optionnelle.

Modifié dans la version 3.6 : Sous Windows, l'encodage spécifié par cette variable est ignoré pour le tampon des consoles interactives à moins que *PYTHONLEGACYWINDOWSSTDIO* ne soit aussi spécifié. Les fichiers et tubes (*pipes* en anglais) redirigés vers les flux standards ne sont pas concernés.

PYTHONNOUSERSITE
Si elle est définie, Python n'ajoute pas le répertoire `site-packages propre à l'utilisateur` à `sys.path`.

Voir aussi :

PEP 370 – Répertoire site-packages propre à l'utilisateur

PYTHONUSERBASE
Définit le répertoire `base utilisateur`. Celui-ci est utilisé pour déterminer le chemin du `répertoire site-packages propre à l'utilisateur` et des schémas d'installation de Distutils pour `python setup.py install --user`.

Voir aussi :

PEP 370 – Répertoire site-packages propre à l'utilisateur

PYTHONEXECUTABLE

Si cette variable d'environnement est définie, `sys.argv[0]` est définie à cette valeur au lieu de la valeur fournie par l'exécutable. Ne fonctionne que sur Mac OS X.

PYTHONWARNINGS

C'est équivalent à spécifier l'option *-W*. Si la valeur est une chaîne séparée par des virgules, c'est équivalent à spécifier l'option *-W* plusieurs fois.

PYTHONFAULTHANDLER

Si elle est définie à une chaîne non vide, `faulthandler.enable()` est appelée au démarrage : ceci installe un gestionnaire pour les signaux `SIGSEGV`, `SIGFPE`, `SIGABRT`, `SIGBUS` et `SIGILL` afin de générer une trace de la pile d'appels. C'est équivalent à spécifier l'option *-X* `faulthandler`.

Nouveau dans la version 3.3.

PYTHONTRACEMALLOC

Si elle est définie à une chaîne non vide, lance le suivi des allocations mémoire par Python en utilisant le module `tracemalloc`. La valeur de la variable définit le nombre maximum d'appels (les *frames* en anglais) stockés dans la trace de pile d'appels. Par exemple, `PYTHONTRACEMALLOC=1` ne stocke que l'appel le plus récent. Voir `tracemalloc.start()` pour davantage d'informations.

Nouveau dans la version 3.4.

PYTHONASYNCIODEBUG

Si elle est définie à une chaîne non vide, active le mode debogage du module `asyncio`.

Nouveau dans la version 3.4.

PYTHONMALLOC

Définit l'allocateur mémoire de Python ou installe des points d'entrée (*hooks* en anglais) de débogage.

Définit la famille d'allocateurs mémoire utilisés par Python :

— `malloc` : utilise la fonction `malloc()` de la bibliothèque C standard pour tous les domaines (`PYMEM_DOMAIN_RAW`, `PYMEM_DOMAIN_MEM`, `PYMEM_DOMAIN_OBJ`).

— `pymalloc` : utilise l'allocateur pymalloc pour les domaines `PYMEM_DOMAIN_MEM` ainsi que `PYMEM_DOMAIN_OBJ` et utilise la fonction `malloc()` pour le domaine `PYMEM_DOMAIN_RAW`.

Installe des points d'entrée de débogage :

— `debug` : installe des fonctions automatiques de débogage au-dessus de l'allocateur mémoire par défaut.

— `malloc_debug` : identique à `malloc` mais installe aussi des fonctions automatiques de débogage.

— `pymalloc_debug` : identique à `pymalloc` mais installe aussi des fonctions automatiques de débogage.

Quand Python est compilé en mode *release*, la valeur par défaut est `pymalloc`. Quand il est compilé en mode débogage, la valeur par défaut est `pymalloc_debug` et les fonctions automatiques de débogage sont utilisées automatiquement.

Si Python est configuré sans le support de `pymalloc`, `pymalloc` et `pymalloc_debug` ne sont pas disponibles. Les valeurs par défaut sont `malloc` en mode *release* et `malloc_debug` en mode débogage.

Reportez-vous à la fonction `PyMem_SetupDebugHooks()` pour ce qui concerne les fonctions automatiques de débogage de l'allocateur mémoire de Python.

Nouveau dans la version 3.6.

PYTHONMALLOCSTATS

Si elle est définie à une chaîne non vide, Python affiche des statistiques relatives à l'allocateur mémoire pymalloc chaque fois qu'un objet est créé par ce gestionnaire, ainsi qu'à la fin de l'exécution du programme.

Cette variable est ignorée si la variable d'environnement *PYTHONMALLOC* est utilisée pour forcer l'allocateur `malloc()` de la bibliothèque C standard ou si Python est configuré sans le support de `pymalloc`.

Modifié dans la version 3.6 : Cette variable peut maintenant être utilisée avec Python compilé en mode *release*. Elle n'a pas d'effet si elle est définie à une chaine vide.

PYTHONLEGACYWINDOWSFSENCODING
Si elle est définie et n'est pas une chaîne vide, l'encodage par défaut respectivement du système de fichiers et des erreurs reviennent à leur valeur pré-3.6, respectivement `'mbcs'` et `'replace'`. Sinon, les nouvelles valeurs par défaut `"UTF-8"` et `"surrogatepass"` sont utilisées.

Vous pouvez aussi activer ceci à l'exécution avec `sys._enablelegacywindowsfsencoding()`.

Disponibilité : Windows

Nouveau dans la version 3.6 : Voir la **PEP 529** pour plus d'informations.

PYTHONLEGACYWINDOWSSTDIO
Si elle est définie et n'est pas une chaîne vide, n'utilise pas les lecteur et écrivain de la nouvelle console. Cela signifie que les caractères Unicode sont encodés avec l'encodage de la console active plutôt qu'en UTF-8.

Cette variable est ignorée si les flux standards sont redirigés (vers des fichiers ou des tubes) plutôt que pointant vers des mémoires tampons de console.

Disponibilité : Windows

Nouveau dans la version 3.6.

1.2.1 Variables en mode débogage

Définir ces variables n'a d'effet que si Python a été compilé en mode débogage, c'est-à-dire que l'option de compilation `--with-pydebug` a été spécifiée.

PYTHONTHREADDEBUG
Si elle est définie, Python affiche des informations de débogage relatives aux différents fils d'exécution.

PYTHONDUMPREFS
Si elle est définie, Python affiche (de manière brute) les objets et les compteurs de références toujours existant après la fermeture de l'interpréteur.

CHAPITRE 2

Utilisation de Python sur les plateformes Unix

2.1 Récupérer et installer la dernière version de Python

2.1.1 Sur Linux

Python est pré-installé sur la plupart des distributions Linux, et est disponible en paquet sur toutes les autres. Cependant, il y a certaines fonctionnalités que vous voudrez utiliser qui ne seront pas disponibles sur le paquet de votre distribution. Vous pouvez facilement compiler la dernière version de Python depuis les sources.

Dans le cas où Python n'est pas pré-installé et n'est pas dans les dépôts non plus, vous pouvez facilement faire les paquets pour votre propre distribution. Jetez un œil à ces liens :

Voir aussi :

https://www.debian.org/doc/manuals/maint-guide/first.fr.html pour les utilisateurs de Debian

https://en.opensuse.org/Portal:Packaging pour les utilisateurs d'OpenSuse

https://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/ch-creating-rpms.html pour les utilisateurs de Fedora

http://www.slackbook.org/html/package-management-making-packages.html pour les utilisateurs de Slackware

2.1.2 Sur FreeBSD et OpenBSD

— Utilisateurs de FreeBSD, pour installer le paquet, utilisez :

```
pkg install python3
```

— Utilisateurs d'OpenBSD, pour installer le paquet, utilisez :

```
pkg_add -r python

pkg_add ftp://ftp.openbsd.org/pub/OpenBSD/4.2/packages/<insert your architecture
↪here>/python-<version>.tgz
```

Par exemple les utilisateurs d'`i386` récupèrent la version 2.5.1 de Python en faisant :

```
pkg_add ftp://ftp.openbsd.org/pub/OpenBSD/4.2/packages/i386/python-2.5.1p2.tgz
```

2.1.3 Sur OpenSolaris

Vous pouvez récupérer Python depuis OpenCSW. Différentes versions de Python sont disponibles et peuvent être installées. Exemple : `pkgutil -i python27`.

2.2 Compiler Python

Si vous voulez compiler CPython vous-même, la première chose à faire est de récupérer le code source. Vous pouvez télécharger la dernière version ou faire un clone. (Si vous voulez contribuer à des correctifs, il vous faut un clone.)

Le processus de compilation est le suivant :

```
./configure
make
make install
```

Les options de configuration et mises en garde pour certaines plateformes Unix spécifiques sont largement documentées dans le fichier README.rst à la racine du dossier des sources Python.

Avertissement : `make install` peut écraser ou cacher le binaire `python3`. `make altinstall` est donc recommandé à la place de `make install` puisqu'il installe seulement *`exec_prefix`*`/bin/python`*`version`*.

2.3 Fichiers et chemins liés à Python

Ceux-ci sont sujets à des différences en fonction des conventions locales d'installation ; `prefix` (`${prefix}`) et `exec_prefix` (`${exec_prefix}`) sont dépendants de l'installation et doivent être interprétés comme pour des logiciels GNU ; ils peuvent être égaux.

Par exemple, sur la plupart des systèmes Linux, leur valeur par défaut est `/usr`.

Fichier/dossier	Signification
`exec_prefix``/bin/python3`	Emplacement recommandé de l'interpréteur.
`prefix``/lib/python`*`version`*, *`exec_prefix`*`/lib/python`*`version`*	Emplacements recommandés des dossiers contenant les modules standards.
`prefix``/include/python`*`version`*, *`exec_prefix`*`/include/python`*`version`*	Emplacements recommandés des dossiers contenant les entêtes nécessaires au développement d'extensions Python et à l'intégration de l'interpréteur.

2.4 Divers

Pour utiliser facilement des scripts Python sur Unix, vous devez les rendre exécutables, par exemple avec

```
$ chmod +x script
```

et mettre un *shebang* approprié en haut du script. Un bon choix est généralement

```
#!/usr/bin/env python3
```

qui cherche l'interpréteur Python dans le `PATH` complet. Cependant, certains systèmes Unix peuvent ne pas avoir la commande **`env`**, donc vous devrez littéralement écrire `/usr/bin/python3` en tant que chemin d'interpréteur.

Pour utiliser des commandes *shell* dans vos scripts Python, regardez le module `subprocess`.

2.5 Éditeurs et IDEs

There are a number of IDEs that support Python programming language. Many editors and IDEs provide syntax highlighting, debugging tools, and **PEP 8** checks.

Merci d'aller sur Python Editors et Integrated Development Environments pour une liste exhaustive.

CHAPITRE 3

Utilisation de Python sur Windows

Ce document a pour but de donner une vue d'ensemble des comportements spécifiques à Windows dont vous devriez être au courant si vous utilisez Python sur Microsoft Windows.

3.1 Installer Python

Contrairement à la plupart des systèmes Unix, Windows n'inclut pas d'installation de Python par défaut. Pour rendre Python disponible, l'équipe CPython a compilé des installateurs (paquets MSI) pour chaque release depuis plusieurs années. Ces installateurs sont principalement destinés à une installation par utilisateur de Python, avec l'interpréteur et la bibliothèque standard utilisés par un seul utilisateur. Cet installateur peut aussi installer Python pour tous les utilisateurs sur une seule machine et un fichier ZIP séparé est disponible pour intégrer Python dans d'autres applications.

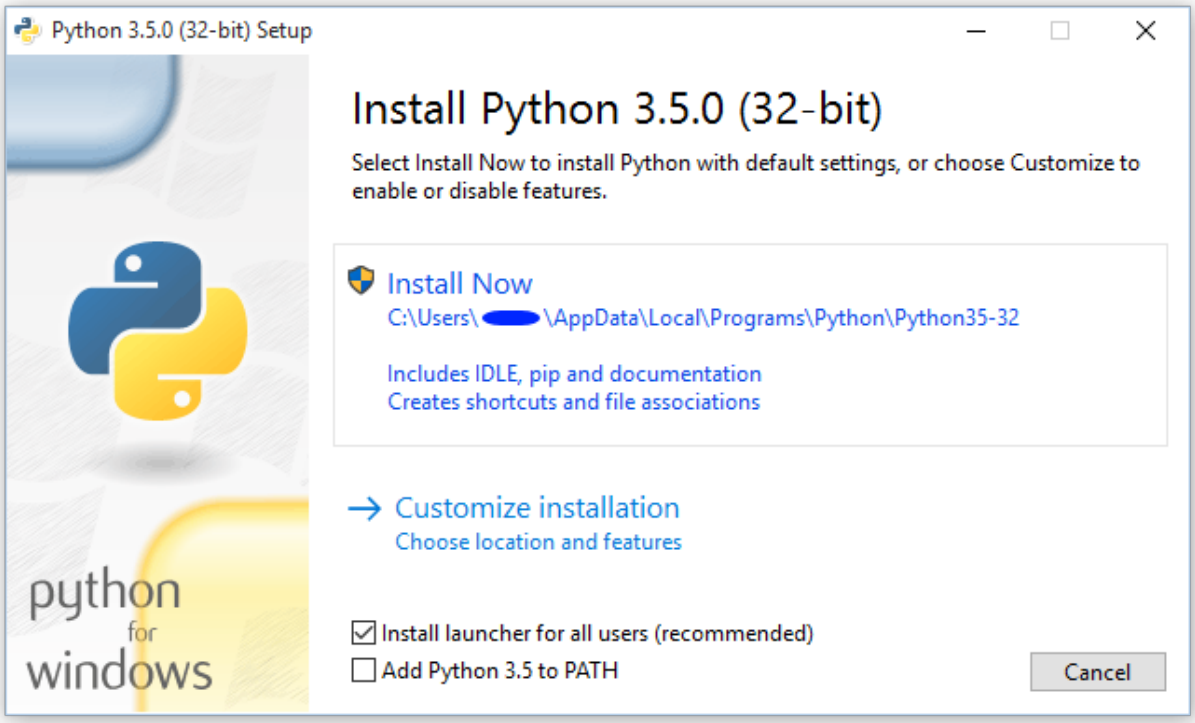
3.1.1 Versions supportées

Comme spécifié dans la **PEP 11**, une *release* Python ne gère qu'une plateforme Windows alors que Microsoft considère la plateforme sous support étendu. Ce qui veut dire que Python 3.6 supporte Windows Vista et plus. Si vous avez besoin de compatibilité Windows XP, vous devez utiliser Python 3.4.

3.1.2 Étapes d'installation

Quatre installateurs Python 3.6 sont disponibles au téléchargement — deux de chaque pour les versions 32-bit et 64-bit de l'interpréteur. L'**installateur web** est léger, et téléchargera automatiquement les composants nécessaires. L'**installateur hors-ligne** inclut les composants nécessaires pour une installation par défaut et n'a besoin d'une connexion internet que pour des fonctionnalités optionnelles. Voir *Installation sans téléchargement* pour d'autres moyens d'éviter des téléchargements durant l'installation.

Après avoir lancé l'installateur, deux options s'affichent :



Si vous sélectionnez « Installer Maintenant » (`Install Now`) :

- Vous n'aurez *pas* besoin d'avoir les droits d'administrateur (sauf si une mise à jour de la bibliothèque d'exécution C est nécessaire ou si vous installez le *Lanceur Python pour Windows* pour tous les utilisateurs)
- Python sera installé dans votre répertoire utilisateur
- Le *Lanceur Python pour Windows* sera installé suivant l'option en bas de la première page
- La bibliothèque standard, la suite de tests, le lanceur et *pip* seront installés
- Si l'option est cochée, le dossier d'installation sera ajouté à votre `PATH`
- Les raccourcis ne seront visibles que pour l'utilisateur actuel

Sélectionner « Personnaliser l'installation » (`Customize installation`) vous permettra de sélectionner les fonctionnalités à installer, le chemin d'installation et d'autres options ou des options post-installation. Pour installer des binaires ou symboles de débogage, vous devrez utiliser cette option.

Pour effectuer une installation pour tous les utilisateurs, vous devez sélectionner « Personnaliser l'installation ». Dans ce cas :

- Vous pouvez avoir à donner une approbation ou des identifiants administrateur
- Python sera installé dans le dossier *Program Files*
- Le *Lanceur Python pour Windows* sera installé dans le dossier *Windows*
- Des fonctionnalités optionnelles peuvent être sélectionnées durant l'installation
- La bibliothèque standard peut être pré-compilée en code intermédiaire (*bytecode* en anglais)
- Si sélectionné, le chemin d'installation sera ajouté au `PATH` système
- Les raccourcis sont disponibles pour tous les utilisateurs

3.1.3 Suppression de la limitation *MAX_PATH*

Historiquement les chemins sous Windows étaient limités 260 caractères. Cela impliquait que les chemins plus longs n'étaient pas résolus, et seraient une cause d'erreurs.

Dans les dernières versions de Windows, cette limitation peut être étendue à approximativement 32.000 caractères. Votre administrateur devra activer la stratégie de groupe « **Enable Win32 long paths** » ou mettre la valeur du registre `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem@LongPathsEnabled` à `1`.

Ceci permet à la fonction `open()`, le module `os` et la plupart des autres fonctionnalités utilisant des chemins d'accepter et de renvoyer des chemins plus longs que 260 caractères quand vous utilisez des chaînes. (L'utilisation de chemins sous forme de *bytes* obsolète sur Windows, et cette fonctionnalité n'est pas disponible quand vous utilisez des *bytes*.)

Après avoir changé l'option si-dessus, aucune configuration supplémentaire n'est requise.

Modifié dans la version 3.6 : Gestion des chemins longs.

3.1.4 Installation sans l'interface utilisateur

Toutes les options disponibles dans l'installateur graphique peuvent aussi être spécifiées dans l'invite de commande, permettant à des installateurs scriptés de répliquer une installation sur plusieurs machines sans interaction humaine. Ces options peuvent aussi être ajoutées sans enlever l'interface graphique pour changer les valeurs par défauts.

Pour complétement cacher l'interface de l'installateur et installer Python silencieusement, passez l'option `/quiet`. Pour sauter les interactions utilisateur mais afficher la progression et les erreurs, passez l'option `/passive`. L'option `/uninstall` peut être passée pour immédiatement démarrer la suppression de Python – Aucune confirmation ne sera demandée.

Toutes les autres options sont passées sous la forme `name=value`, ou `value` est normalement soit `0` pour désactiver une fonctionnalité, soit `1` pour activer une fonctionnalité, soit un chemin. Ci-dessous la liste complète des options.

Nom	Description	Valeur par défaut
InstallAllUsers	Effectue une installation pour tous les utilisateurs.	0
TargetDir	Le dossier d'installation	Sélection basée sur InstallAllUsers
DefaultAllUsersTargetDir	Le dossier d'installation par défaut pour les installations pour tous les utilisateurs	`%ProgramFiles%\Python X.Y` ou `%ProgramFiles(x86)%\Python X.Y`
DefaultJustForMeTargetDir	Le dossier d'installation par défaut pour des installations juste pour soi	`%LocalAppData%\Programs\PythonXY` or `%LocalAppData%\Programs\PythonXY-32`
DefaultCustomTargetDir	Le dossier d'installation personnalisé par défaut affiché par l'interface utilisateur	(vide)
AssociateFiles	Crée les associations de fichiers si le lanceur est aussi installé.	1
CompileAll	Compile tous les fichiers `.py` en `.pyc`.	0
PrependPath	Add install and Scripts directories to `PATH` and `.PY` to `PATHEXT`	0
Shortcuts	Crée des raccourcis pour l'interpréteur, la documentation et IDLE si installé.	1
Include_doc	Installe le manuel Python	1
Include_debug	Installe les binaires de débogage	0
Include_dev	Installe les fichiers d'en-tête et les bibliothèques développeur	1
Include_exe	Installe `python.exe` et les fichiers connexes	1
Include_launcher	Installe le *Lanceur Python pour Windows*.	1
InstallLauncherAllUsers	Installe le *Lanceur Python pour Windows* pour tous les utilisateurs.	1
Include_lib	Installe la bibliothèque standard et les modules d'extension	1
Include_pip	Installe `pip` et `setuptools`	1
Include_symbols	Installe les symboles de débogage (`*.pdb`)	0
Include_tcltk	Installe Tcl/Tk et IDLE	1
Include_test	Installe la suite de tests de la bibliothèque standard	1
Include_tools	Installe les scripts utilitaires	1
LauncherOnly	Installe seulement le lanceur. Ceci écrasera la plupart des autres options.	0
SimpleInstall	Désactive la plupart de l'interface d'installation	0
SimpleInstallDescription	Un message personnalisé à afficher quand l'interface d'installation simplifiée est utilisée.	(vide)

Par exemple, pour installer silencieusement Python sur tout le système, vous pourriez utilisez la commande suivante (depuis une invite de commande administrateur) :

```
python-3.6.0.exe /quiet InstallAllUsers=1 PrependPath=1 Include_test=0
```

Pour permettre à l'utilisateur d'installer facilement une copie de Python sans la suite de tests, vous pouvez proposer un raccourci avec la commande suivante. Cela affichera une page initiale simplifiée et interdira la personnalisation :

```
python-3.6.0.exe InstallAllUsers=0 Include_launcher=0 Include_test=0
    SimpleInstall=1 SimpleInstallDescription="Just for me, no test suite."
```

(Notez qu'omettre le lanceur omet aussi les associations de fichiers, et n'est recommandé que pour les installations par utilisateur quand il y a aussi une installation complète sur le système qui a inclus de lanceur.)

The options listed above can also be provided in a file named `unattend.xml` alongside the executable. This file specifies a list of options and values. When a value is provided as an attribute, it will be converted to a number if possible. Values provided as element text are always left as strings. This example file sets the same options as the previous example :

```
<Options>
    <Option Name="InstallAllUsers" Value="no" />
    <Option Name="Include_launcher" Value="0" />
    <Option Name="Include_test" Value="no" />
    <Option Name="SimpleInstall" Value="yes" />
    <Option Name="SimpleInstallDescription">Just for me, no test suite</Option>
</Options>
```

3.1.5 Installation sans téléchargement

Comme certaines fonctionnalités de Python ne sont pas incluses dans l'installateur initial, la sélection de certaines de ces fonctionnalités peut demander une connexion Internet. Pour éviter ce besoin, tous les composants nécessaires peuvent être téléchargés à la demande pour créer un installateur complet qui ne demandera plus de connexion Internet indépendamment des options sélectionnées. Notez que ce téléchargement peut être plus gros que nécessaire, mais lorsqu'un un grand nombre d'installations doivent être faites, il est très utile d'avoir une copie locale.

Execute the following command from Command Prompt to download all possible required files. Remember to substitute `python-3.6.0.exe` for the actual name of your installer, and to create layouts in their own directories to avoid collisions between files with the same name.

```
python-3.6.0.exe /layout [optional target directory]
```

Vous pouvez aussi spécifier l'option `/quiet` pour masquer la progression.

3.1.6 Modification d'une installation

Une fois Python installé, vous pouvez ajouter ou supprimer des fonctionnalités depuis l'outil Windows *Programs and Features* (Programmes et Fonctionnalités). Sélectionnez la ligne *Python* et choisissez « Uninstall/Change » (Désinstaller/Modifier) pour ouvrir l'installateur en mode maintenance.

« Modify » vous permet d'ajouter ou d'enlever des fonctionnalités en modifiant les cases à cocher (les cases inchangées n'installeront ou ne supprimeront rien). Certaines options ne peuvent pas être modifiées dans ce mode, comme le dossier d'installation. Pour modifier ces options, vous devrez ré-installer Python entièrement.

« Repair » vérifiera tous les fichiers qui doivent être installés avec les paramètres actuels le sont, et remplacera ceux qui ont étés supprimés ou modifiés.

« Uninstall » désinstallera Python entièrement, à l'exception du *Lanceur Python pour Windows* qui à sa propre ligne dans *Programs and Features*.

3.1.7 Autres plateformes

Avec le développement continu de Python, certaines plateformes qui étaient auparavant prises en charge ne sont plus prises en charge (en raison du manque d'utilisateurs ou de développeurs). Voir **PEP 11** pour plus de détails sur toutes les plateformes non prises en charge.

— Windows CE est toujours pris en charge.

— L'installateur Cygwin offre d'installer l'interpréteur Python (cf. Cygwin package source, Maintainer releases)

Voir Python pour Windows pour des instructions détaillées sur les plateformes avec installateurs pré-compilés.

Voir aussi :

Python on XP « 7 Minutes to « Hello World ! » » by Richard Dooling, 2006

Installing on Windows in « Dive into Python : Python from novice to pro » by Mark Pilgrim, 2004, ISBN 1-59059-356-1

For Windows users in « Installing Python » in « A Byte of Python » by Swaroop C H, 2003

3.2 Paquets alternatifs

À part la distribution standard CPython, il y a des paquets modifiés incluant des fonctionnalités additionnelles. La liste qui suit est une liste de versions populaires et de leurs fonctionnalités principales :

ActivePython Installeur avec une compatibilité multi-plateforme, de la documentation, et *PyWin32*

Anaconda Des modules scientifiques populaires (comme *numpy*, *scipy* et *pandas*) et le gestionnaire de paquets `conda`.

Canopy Un « environnement d'analyse complet Python » avec des éditeurs et autres outils de développement.

WinPython Distribution spécifique à Windows avec des paquets scientifiques pré-compilés et des outils pour construire des paquets.

Notez que ces paquets peuvent ne pas inclure la dernière version de Python ou d'autres bibliothèques, et ne sont pas maintenus ni supportés par les *core devs* Python.

3.3 Configurer Python

Pour exécuter Python confortablement à partir d'une invite de commandes, vous pouvez envisager de modifier certaines variables d'environnement par défaut dans Windows. Bien que l'installateur offre une option pour configurer les variables `PATH` et `PATHEXT` pour vous, ce n'est fiable que pour une seule installation à l'échelle du système. Si vous utilisez régulièrement plusieurs versions de Python, pensez à utiliser le *Lanceur Python pour Windows*.

3.3.1 Digression : Définition des variables d'environnement

Windows permet de configurer les variables d'environnement de façon permanente au niveau de l'utilisateur et du système, ou temporairement dans une invite de commandes.

Pour définir temporairement les variables d'environnement, ouvrez l'invite de commandes et utilisez la commande **`set`** :

```
C:\>set PATH=C:\Program Files\Python 3.6;%PATH%
C:\>set PYTHONPATH=%PYTHONPATH%;C:\My_python_lib
C:\>python
```

Ces modifications s'appliqueront à toutes les autres commandes exécutées dans cette console et seront héritées par toutes les applications démarrées à partir de cette console.

Un nom de variable entre des signes pour cent sera remplacé par sa valeur, vous permettant d'ajouter votre nouvelle valeur au début ou à la fin. Ajouter **`python.exe`** au début de `PATH` est un moyen courant de s'assurer que la version correcte de Python est lancée.

Pour modifier définitivement les variables d'environnement par défaut, recherchez « modifier les variables d'environnement », via le menu démarrer, ou ouvrez « Propriétés Système », *Paramètres Système Avancés* et cliquez sur le bouton *Variables d'Environnement*. Dans cette boîte de dialogue, vous pouvez ajouter ou modifier des variables utilisateurs et systèmes. Pour modifier les variables systèmes, vous avez besoin d'un accès non restreint à votre ordinateur (c'est-à-dire aux droits d'administrateur).

Note : Windows va concaténer les variables utilisateurs *après* les variables systèmes, ce qui peut provoquer des résultats inattendus lors de la modification de `PATH`.

La variable *`PYTHONPATH`* est utilisée par toutes les versions de Python 2 et Python 3, vous ne devez donc pas configurer cette variable de façon permanente à moins qu'elle n'ajoute que du code compatible avec toutes les versions de Python installées.

Voir aussi :

https://support.microsoft.com/kb/100843 Variables d'environnement dans Windows NT

https://technet.microsoft.com/en-us/library/cc754250.aspx La commande SET, pour modifier temporairement les variables d'environnement

https://technet.microsoft.com/en-us/library/cc755104.aspx La commande SETX, pour modifier de façon permanente les variables d'environnement

https://support.microsoft.com/kb/310519 Comment gérer les variables d'environnement sous Windows XP

https://www.chem.gla.ac.uk/~louis/software/faq/q1.html Définir les variables d'environnement, *Louis J. Farrugia*

3.3.2 Trouver l'exécutable Python

Modifié dans la version 3.5.

En plus d'utiliser l'entrée du menu Démarrer automatiquement créée pour l'interpréteur Python, vous souhaiterez peut-être démarrer Python depuis l'invite de commandes. L'installateur a une option pour installer cela pour vous.

Sur la première page de l'installateur, une option intitulée « Ajouter Python au *PATH* » peut être sélectionnée pour que l'installateur ajoute l'emplacement d'installation dans le `PATH`. L'emplacement du dossier `Scripts\` y est également ajouté. Cela vous permet de taper **`Python`** pour exécuter l'interpréteur, et **`pip`** pour l'installateur de paquets. Ainsi, vous pouvez également exécuter vos scripts avec les options de ligne de commande, voir la documentation *Ligne de commande*.

Si vous n'activez pas cette option au moment de l'installation, vous pouvez toujours ré-exécuter l'installateur, sélectionner Modifier et l'activer. Vous pouvez également modifier manuellement le `PATH` à l'aide des instructions de *Digression : Définition des variables d'environnement*. Vous devez définir votre variable d'environnement `PATH` pour inclure le répertoire de votre installation Python, délimité par un point-virgule des autres entrées. Une variable d'exemple pourrait ressembler à ceci (en supposant que les deux premières entrées existaient déjà) :

```
C:\WINDOWS\system32;C:\WINDOWS;C:\Program Files\Python 3.6
```

3.4 Lanceur Python pour Windows

Nouveau dans la version 3.3.

Le lanceur Python pour Windows est un utilitaire qui facilite la recherche et l'exécution de différentes versions de Python. Il permet aux scripts (ou à la ligne de commande) d'indiquer une préférence pour une version Python spécifique, cherchera et exécutera cette version.

Contrairement à la variable `PATH`, le lanceur sélectionne correctement la version la plus appropriée de Python. Il préfère les installations par utilisateur sur celles du système, et les trie par version plutôt que d'utiliser la version la plus récente installée.

3.4.1 Pour commencer

Depuis la ligne de commande

Modifié dans la version 3.6.

Les installations systèmes de Python 3.3 et ultérieur mettent le lanceur dans votre `PATH`. Le lanceur est compatible avec toutes les versions disponibles de Python, peu importe lesquelles sont installées. Pour vérifier que le lanceur est disponible, exécutez la commande suivante dans l'invite de commandes :

```
py
```

Vous devriez voir se lancer la dernière version de Python installée — il peut être quitté normalement, et tous les arguments de ligne de commande supplémentaires spécifiés seront envoyés directement à Python.

Si plusieurs versions de Python sont installées (par exemple, 2.7 et 3.6), vous aurez remarqué que Python 3.6 se lance – pour lancer Python 2.7, essayez la commande :

```
py -2.7
```

Si vous voulez que la dernière version de Python 2.x que vous avez installé, essayez la commande :

```
py -2
```

Remarquez que la dernière version de Python 2.x démarre.

Si vous voyez l'erreur suivante, le lanceur n'est pas installé :

```
'py' is not recognized as an internal or external command,
operable program or batch file.
```

Les installations par utilisateur de Python n'ajoutent pas le lanceur à `PATH` sauf si l'option a été sélectionnée lors de l'installation.

Environnements virtuels

Nouveau dans la version 3.5.

Si le lanceur est exécuté sans version de Python explicite et qu'un environnement virtuel (créé avec le module de la bibliothèque standard `venv` ou l'outil externe `virtualenv`) est actif, le lanceur exécute l'interpréteur de l'environnement virtuel plutôt que l'interpréteur global. Pour exécuter l'interpréteur global, désactivez l'environnement virtuel ou spécifiez explicitement la version Python globale.

À partir d'un script

Créons un script Python de test, créez un fichier appelé `hello.py` avec le contenu suivant

```
#! python
import sys
sys.stdout.write("hello from Python %s\n" % (sys.version,))
```

À partir du répertoire dans lequel se trouve `hello.py`, exécutez la commande :

```
py hello.py
```

Vous devriez voir le numéro de version du Python 2.x le plus récemment installé. Maintenant, essayez de changer la première ligne en :

```
#! python3
```

La commande doit maintenant afficher les dernières informations de Python 3.x. Comme pour les exemples de ligne de commande ci-dessus, vous pouvez spécifier un qualificateur de version plus explicite. En supposant que vous avez installé Python 2.6, essayez de changer la première ligne à `#! python2.6` et vous devriez trouver les informations de version 2.6 imprimées.

Notez que, contrairement à l'utilisation interactive, un « python » nu utilisera la dernière version de Python 2.x que vous avez installé. C'est pour la compatibilité ascendante et pour la compatibilité avec UNIX, où la commande `python` fait généralement référence à Python 2.

À partir d'associations de fichiers

Le lanceur aurait dû être associé à des fichiers Python (des fichiers comme `.py`, `.pyw`, `.pyc`) lorsqu'il a été installé. Cela signifie que lorsque vous double-cliquez sur l'un de ces fichiers à partir de l'Explorateur Windows, le lanceur sera utilisé, et donc vous pouvez utiliser les mêmes installations décrites ci-dessus pour que le script spécifie la version qui doit être utilisée.

L'avantage principal de ceci est qu'un seul lanceur peut prendre en charge plusieurs versions de Python en même temps en fonction du contenu de la première ligne.

3.4.2 Lignes Shebang

Si la première ligne d'un fichier de script commence par `#!`, elle est connue sous le nom de ligne « *shebang* ». Linux et d'autres systèmes basés sur Unix ont une prise en charge native de ces lignes et les *shebangs* sont couramment utilisés sur ces systèmes pour indiquer comment un script doit être exécuté. Ce lanceur permet aux mêmes installations d'être utilisés avec des scripts Python sur Windows et les exemples ci-dessus démontrent leur utilisation.

Pour permettre aux *shebang* dans les scripts Python d'être portables entre UNIX et Windows, ce lanceur prend en charge un certain nombre de commandes « virtuelles » pour spécifier l'interpréteur à utiliser. Les commandes virtuelles prises en charge sont :

- `/usr/bin/env python`
- `/usr/bin/python`
- `/usr/local/bin/python`
- `python`

Par exemple, si la première ligne de votre script commence par

```
#! /usr/bin/python
```

Le Python par défaut sera trouvé et utilisé. Comme de nombreux scripts Python écrits pour fonctionner sur UNIX auront déjà cette ligne, ils devraient fonctionner avec le lanceur sans modification. Si vous écrivez un nouveau script sur Windows et que vous pensez qu'il sera utile sur UNIX, vous devez utiliser l'une des lignes *shebang* commençant par `/usr`.

Any of the above virtual commands can be suffixed with an explicit version (either just the major version, or the major and minor version) - for example `/usr/bin/python2.7` - which will cause that specific version to be located and used.

La forme `/usr/bin/env` de ligne *shebang* possède une autre propriété spéciale. Avant de rechercher les interpréteurs Python installés, cette forme recherche d'abord l'exécutable dans le `PATH`. Cela correspond au comportement du programme Unix `env`, qui effectue une recherche dans `PATH`.

3.4.3 Arguments dans les lignes *shebang*

Les lignes *shebang* peuvent également spécifier des options supplémentaires à passer à l'interpréteur Python. Par exemple, si vous avez une ligne *shebang* :

```
#! /usr/bin/python -v
```

Alors, Python sera démarré avec l'option `-v`

3.4.4 Personnalisation

Personnalisation via des fichiers INI

Two .ini files will be searched by the launcher - `py.ini` in the current user's « application data » directory (i.e. the directory returned by calling the Windows function `SHGetFolderPath` with `CSIDL_LOCAL_APPDATA`) and `py.ini` in the same directory as the launcher. The same .ini files are used for both the "console" version of the launcher (i.e. py.exe) and for the "windows" version (i.e. pyw.exe).

Customization specified in the « application directory » will have precedence over the one next to the executable, so a user, who may not have write access to the .ini file next to the launcher, can override commands in that global .ini file.

Personnalisation des versions Python par défaut

In some cases, a version qualifier can be included in a command to dictate which version of Python will be used by the command. A version qualifier starts with a major version number and can optionally be followed by a period (".") and a minor version specifier. If the minor qualifier is specified, it may optionally be followed by « -32 » to indicate the 32-bit implementation of that version be used.

Par exemple, une ligne *shebang* valant `#!python` n'a pas de qualificateur de version, tandis que `#!python3` a un qualificateur de version qui ne spécifie qu'une version majeure.

If no version qualifiers are found in a command, the environment variable `PY_PYTHON` can be set to specify the default version qualifier - the default value is « 2 ». Note this value could specify just a major version (e.g. « 2 ») or a major.minor qualifier (e.g. « 2.6 »), or even major.minor-32.

Si aucun qualificateur de version mineure n'est trouvé, la variable d'environnement `PY_PYTHON{major}` (où `{major}` est le qualificateur de version principale actuelle tel que déterminé ci-dessus) peut être définie pour spécifier la version complète. Si aucune option de ce type n'est trouvée, le lanceur énumérera les versions de Python installées et utilisera la dernière version mineure trouvée pour la version principale, qui est probablement la plus récemment installée dans cette famille.

On 64-bit Windows with both 32-bit and 64-bit implementations of the same (major.minor) Python version installed, the 64-bit version will always be preferred. This will be true for both 32-bit and 64-bit implementations of the launcher - a 32-bit launcher will prefer to execute a 64-bit Python installation of the specified version if available. This is so the behavior of the launcher can be predicted knowing only what versions are installed on the PC and without regard to the order in which they were installed (i.e., without knowing whether a 32 or 64-bit version of Python and corresponding launcher was installed last). As noted above, an optional « -32 » suffix can be used on a version specifier to change this behaviour.

Exemples :

— Si aucune option pertinente n'est définie, les commandes `python` et `python2` utiliseront la dernière version de Python 2.x installée et la commande `python3` utilisera le dernier Python 3.x installé.

— Les commandes `python3.1` et `python2.7` ne consultent aucune option du tout car les versions sont entièrement spécifiées.

— Si `PY_PYTHON=3`, les commandes `python` et `python3` utiliseront la dernière version de Python 3 installée.

— Si `PY_PYTHON=3.1-32`, la commande `python` utilisera l'implémentation 32-bits de 3.1 alors que la commande `python3` utilisera le dernier Python installé (*PY_PYTHON* n'a pas été considéré du tout comme une version majeure a été spécifiée.)

— Si `PY_PYTHON=3` et `PY_PYTHON3=3.1`, les commandes `python` et `python3` utiliseront spécifiquement 3.1

En plus des variables d'environnement, les mêmes paramètres peuvent être configurés dans le `.INI` utilisé par le lanceur. La section dans le fichier INI est appelée `[defaults]` et le nom de la clé sera le même que les variables d'environnement sans le préfixe `PY_` principal (et notez que les noms de clés dans le fichier **INI** sont insensibles à la case.) Le contenu d'une variable d'environnement remplacera les éléments spécifiés dans le fichier *INI*.

Par exemple :

— Le paramètre `PY_PYTHON=3.1` équivaut au fichier **INI** contenant :

```
[defaults]
python=3.1
```

— Le paramètre `PY_PYTHON=3` et `PY_PYTHON3=3.1` équivaut au fichier *INI* contenant :

```
[defaults]
python=3
python3=3.1
```

3.4.5 Diagnostics

Si une variable d'environnement PYLAUNCH_DEBUG est définie (à n'importe quelle valeur), le lanceur affichera des informations de diagnostic sur *stderr* (c'est-à-dire sur la console). Bien que ces informations parviennent à être en même temps nombreuses et concises, elles devraient vous permettre de voir quelles versions de Python ont été trouvées, pourquoi une version particulière a été choisie et la ligne de commande exacte utilisée pour exécuter le Python cible.

3.5 Recherche de modules

Python stocke généralement sa bibliothèque (et donc votre dossier site-packages) dans le répertoire d'installation. Donc, si vous aviez installé Python dans C:\Python\, la bibliothèque par défaut résiderait dans C:\Python\Lib\ et les modules tiers devraient être stockés dans C:\Python\Lib\site-packages\.

To completely override sys.path, create a ._pth file with the same name as the DLL (python36._pth) or the executable (python._pth) and specify one line for each path to add to sys.path. The file based on the DLL name overrides the one based on the executable, which allows paths to be restricted for any program loading the runtime if desired.

Lorsque le fichier existe, toutes les variables de registre et d'environnement sont ignorées, le mode isolé est activé et site n'est pas importé, sauf si une ligne du fichier spécifie import site. Les chemins et les lignes vides commençant par # sont ignorés. Chaque chemin d'accès peut être absolu ou relatif à l'emplacement du fichier. Les importations autres que vers *site* ne sont pas autorisées, ni n'importe quelle instruction Python.

Note that .pth files (without leading underscore) will be processed normally by the site module.

Lorsque aucun fichier ._pth n'est trouvé, voilà comment sys.path est construit sur Windows :

- Une entrée vide est ajoutée au début, qui correspond au répertoire courant.
- Si la variable d'environnement *PYTHONPATH* existe, comme décrit dans *Variables d'environnement*, ses entrées sont ajoutées ensuite. Notez que sur Windows, les chemins d'accès de cette variable doivent être séparés par des points-virgules, pour les distinguer des deux points utilisés dans les identificateurs de lecteur (C:\ etc.).
- Des « chemins d'accès d'application » supplémentaires peuvent être ajoutés dans le registre en tant que sous-clés de \SOFTWARE\Python\PythonCore{version}\PythonPath sous les ruches HKEY_CURRENT_USER et HKEY_LOCAL_MACHINE. Les sous-clés qui ont des chaînes de chemin délimitées par des points-virgules comme valeur par défaut entraînent l'ajout de chaque chemin d'accès à sys.path. (Notez que tous les installateurs connus utilisent seulement HKLM, donc HKCU est généralement vide.)
- Si la variable d'environnement *PYTHONHOME* est définie, elle est supposée comme « Python Home ». Sinon, le chemin de l'exécutable principal de Python est utilisé pour chercher un « fichier de repère » (soit Lib\os.py ou pythonXY.zip) pour déduire le « Python Home ». Si un « Python Home » est trouvé, les sous-répertoires correspondants ajoutés à sys.path (Lib, plat-win, etc) sont basés sur ce dossier. Sinon, le chemin d'accès Python principal est construit à partir du PythonPath stocké dans le registre.
- Si le « Python Home » ne peut pas être trouvé, *PYTHONPATH* n'est pas spécifié dans l'environnement et aucune entrée de registre ne peut être trouvée, un chemin par défaut avec des entrées relatives est utilisé (par exemple .\Lib;.\plat-win, etc.).

Si un fichier pyvenv.cfg se trouve à côté de l'exécutable principal ou dans le répertoire un niveau au-dessus de l'exécutable, les variantes suivantes s'appliquent :

- Si home est un chemin absolu et *PYTHONHOME* n'est pas défini, ce chemin d'accès est utilisé au lieu du chemin d'accès à l'exécutable principal lors de la déduction de l'emplacement du *home*.

Le résultat final de tout ceci est :

- Lors de l'exécution de python.exe, ou tout autre .exe dans le répertoire principal de Python (soit une version installée, soit directement à partir du répertoire PCbuild), le chemin principal est déduit et les chemins d'accès principaux dans le Registre sont ignorés. D'autres « chemins d'application » dans le registre sont toujours lus.
- Lorsque Python est hébergé dans un autre fichier .exe (répertoire différent, intégré via COM, etc.), le « Python Home » ne sera pas déduit, de sorte que le chemin d'accès principal du registre est utilisé. D'autres « chemins d'application » dans le registre sont toujours lus.

— Si Python ne peut pas trouver son « home » et il n'y a pas de valeur de registre (*.exe* gelé, une installation très étrange) vous obtenez un chemin d'accès avec certains chemins par défaut, mais relatif.

Pour ceux qui veulent intégrer Python dans leur application ou leur distribution, les conseils suivants empêcheront les conflits avec d'autres installations :

— Incluez un fichier `._pth` à côté de votre exécutable contenant les répertoires à inclure. Ceci ignorera les chemins répertoriés dans le registre et les variables d'environnement, et ignorera également `site` à moins que `import site` soit listé.
— If you are loading `python3.dll` or `python36.dll` in your own executable, explicitly call `Py_SetPath()` or (at least) `Py_SetProgramName()` before `Py_Initialize()`.
— Effacer et/ou écraser *`PYTHONPATH`* et configurez *`PYTHONHOME`* avant de lancer le `python.exe` de votre application.
— Si vous ne pouvez pas utiliser les suggestions précédentes (par exemple, vous êtes une distribution qui permet aux gens d'exécuter `python.exe` directement), assurez-vous que le point de repère `Lib\os.py` existe dans votre répertoire d'installation. (Notez qu'il ne sera pas détecté à l'intérieur d'un fichier ZIP, mais un fichier ZIP correctement nommé sera détecté à la place.)

These will ensure that the files in a system-wide installation will not take precedence over the copy of the standard library bundled with your application. Otherwise, your users may experience problems using your application. Note that the first suggestion is the best, as the others may still be susceptible to non-standard paths in the registry and user site-packages.

Modifié dans la version 3.6 :
— Ajout de la gestion des `._pth` et suppression de l'option `applocal` de `pyvenv.cfg`.
— Ajout de `pythonXX.zip` comme point de repère potentiel lorsqu'il est directement adjacent à l'exécutable.

Obsolète depuis la version 3.6 : Les modules spécifiés dans le registre sous `Modules` (pas `PythonPath`) peuvent être importés par `importlib.machinery.WindowsRegistryFinder`. Ce Finder est activé sur Windows dans 3.6.0 et plus récent, mais il pourrait être nécessaire de l'ajouter explicitement à `sys.meta_path` à l'avenir.

3.6 Modules supplémentaires

Même si Python a l'ambition d'être portable parmi toutes les plates-formes, il existe des fonctionnalités propres à Windows. Certains modules, à la fois dans la bibliothèque standard et externe, et des exemples existent pour utiliser ces fonctionnalités.

Les modules standard de Windows sont documentés dans mswin-specific-services.

3.6.1 PyWin32

Le module PyWin32 de Mark Hammond est une collection de modules pour un support avancé spécifique à Windows. Cela inclut les services pour :

— Component Object Model (COM)
— Appels à l'API Win32
— Registre
— Journal d'événement
— Microsoft Foundation Classes (MFC) interfaces utilisateur

PythonWin est un exemple d'application MFC livrée avec PyWin32. Il s'agit d'un IDE embarqué avec débogueur intégré.

Voir aussi :

Win32 How Do I…? par Tim Golden

Python and COM par David et Paul Boddie

3.6.2 cx_Freeze

cx_Freeze is a `distutils` extension (see extending-distutils) which wraps Python scripts into executable Windows programs (`*.exe` files). When you have done this, you can distribute your application without requiring your users to install Python.

3.6.3 WConio

Depuis la couche avancée de gestion de terminal de Python, `curses`, est limité aux systèmes de type UNIX, il existe une bibliothèque exclusive à Windows : *Windows Console I/O for Python*.

WConio est un *wrapper* pour les fichiers Turbo-C `CONIO.H`, utilisé pour créer des interfaces texte utilisateur.

3.7 Compiler Python sous Windows

Si vous voulez compiler CPython vous-même, la première chose à faire est obtenir la source. Vous pouvez télécharger soit la source de la dernière version ou tout simplement prendre un checkout.

L'arborescence source contient une solution de compilation et des fichiers projet pour Microsoft Visual Studio 2015, qui est le compilateur utilisé pour générer les versions officielles de Python. Ces fichiers se trouvent dans le répertoire `PCbuild`.

Consultez `PC/readme.txt` pour des informations générales sur le processus de construction.

Pour les modules d'extension, consultez building-on-windows.

Voir aussi :

Python + Windows + distutils + SWIG + gcc MinGW ou « *Creating Python extensions in C/C++ with SWIG and compiling them with MinGW gcc under Windows* » ou « *Installing Python extension with distutils and without Microsoft Visual C++* » par Sébastien Sauvage, 2003

MingW – Python extensions par Trent Apted et al, 2007

3.8 Embedded Distribution

Nouveau dans la version 3.5.

La distribution embarquée est un fichier ZIP contenant un environnement Python minimal. Il est destiné à agir dans le cadre d'une autre application, plutôt que d'être directement accessible par les utilisateurs finaux.

When extracted, the embedded distribution is (almost) fully isolated from the user's system, including environment variables, system registry settings, and installed packages. The standard library is included as pre-compiled and optimized `.pyc` files in a ZIP, and `python3.dll`, `python36.dll`, `python.exe` and `pythonw.exe` are all provided. Tcl/tk (including all dependants, such as Idle), pip and the Python documentation are not included.

Note : La distribution intégrée n'inclut pas le Microsoft C Runtime et il est de la responsabilité de l'installateur d'application de le fournir. Le *runtime* peut avoir déjà été installé sur le système d'un utilisateur précédemment ou automatiquement via Windows Update, et peut être détecté en trouvant `ucrtbase.dll` dans le répertoire système.

Les paquets tiers doivent être installés par le programme d'installation de l'application parallèlement à la distribution embarquée. L'utilisation de pip pour gérer les dépendances comme pour une installation Python régulière n'est pas prise en charge avec cette distribution, mais il reste possible d'inclure et d'utiliser pip pour les mises à jour automatiques. En

général, les paquets tiers doivent être traités dans le cadre de l'application (*vendoring*) afin que le développeur puisse assurer la compatibilité avec les versions plus récentes avant de fournir des mises à jour aux utilisateurs.

Les deux cas d'utilisation recommandés pour cette distribution sont décrits ci-dessous.

3.8.1 Application Python

Une application écrite en Python n'exige pas nécessairement que les utilisateurs soient au courant de ce fait. La distribution embarquée peut être utilisée dans ce cas pour inclure une version privée de Python dans un package d'installation. Selon la façon dont il devrait être transparent (ou inversement, à quel point il doit paraître professionnel), il y a deux options.

L'utilisation d'un exécutable spécialisé en tant que lanceur nécessite de la programmation, mais fournit l'expérience la plus transparente pour les utilisateurs. Avec un lanceur personnalisé, il n'y a pas d'indications évidentes que le programme s'exécute sur Python : les icônes peuvent être personnalisées, les informations de la société et de la version peuvent être spécifiées, et les associations de fichiers se comportent correctement. Dans la plupart des cas, un lanceur personnalisé devrait simplement pouvoir appeler `Py_Main` avec une ligne de commande codée en dur.

L'approche la plus simple consiste à fournir un fichier batch ou un raccourci généré qui appelle directement le `python.exe` ou `pythonw.exe` avec les arguments de ligne de commande requis. Dans ce cas, l'application semble être Python et non son nom réel, et les utilisateurs peuvent avoir du mal à le distinguer des autres processus Python en cours d'exécution ou des associations de fichiers.

Avec cette dernière approche, les packages doivent être installés en tant que répertoires à côté de l'exécutable Python pour s'assurer qu'ils soient visibles par Python. Avec le lanceur spécialisé, les paquets peuvent être installés dans d'autres emplacements car il y a une possibilité de spécifier le chemin de recherche avant de lancer l'application.

3.8.2 Embarquer Python

Les applications écrites en code natif nécessitent souvent une certaine forme de langage de *scripting*, et la distribution Python intégrée peut être utilisée à cette fin. En général, la majorité de l'application est dans le code natif, qui soit invoque `python.exe` soit utilise directement `python3.dll`. Dans les deux cas, l'extraction de la distribution intégrée dans un sous-répertoire de l'installation de l'application est suffisante pour fournir un interpréteur Python chargeable.

Comme pour l'utilisation de l'application, les paquets peuvent être installés sur n'importe quel emplacement, car il est possible de spécifier des chemins de recherche avant d'initialiser l'interpréteur. Sinon, il n'y a pas de différences fondamentales entre l'utilisation de la distribution embarquée et une installation classique.

3.9 Autres ressources

Voir aussi :

Python Programming On Win32 « Help for Windows Programmers » de Mark Hammond et Andy Robinson, O'Reilly Media, 2000, ISBN 1-56592-621-8

A Python for Windows Tutorial par Amanda Birmingham, 2004

PEP 397 - Python launcher for Windows The proposal for the launcher to be included in the Python distribution.

CHAPITRE 4

Utilisation de Python sur un Macintosh

Auteur Bob Savage <bobsavage@mac.com>

Python sur un Macintosh exécutant Mac OS X est en principe très similaire à Python sur n'importe quelle autre plateforme Unix, mais il y a un certain nombre de fonctionnalités additionnelle telle que l'IDE et le gestionnaire de paquets qui méritent d'être soulignées.

4.1 Obtenir et installer MacPython

Mac OS X 10.8 contient déjà Python 2.7 pré-installé par Apple. Si vous le souhaitez, vous êtes invités à installer la version la plus récente de Python 3 à partir du site de Python (https://www.python.org). Une version « binaire universelle » de Python, qui s'exécute nativement sur les nouveaux processeurs Intel de Mac et les processeurs PPC, CPUs hérités de Mac, y est disponible.

Vous obtiendrez un certain nombre de choses après installation :

— Un dossier `MacPython 3.6` dans votre dossier `Applications`. Dedans vous trouverez **IDLE**, l'environnement de développement qui fait partie des distributions Python officielles ; **PythonLauncher**, qui gère le lancement de scripts Python depuis le `Finder` ; et l'outil **« Build Applet »**, qui permets d'empaqueter des scripts Python en tant qu'applications à part entière sur votre système.

— Un **framework** `/Library/Frameworks/Python.framework`, qui inclut l'exécutable Python et ses bibliothèques. L'installateur ajoute ce chemin à votre **PATH**. Pour désinstaller MacPython, vous pouvez simplement supprimer ces trois choses. Un lien symbolique vers l'exécutable Python est placé dans `/usr/local/bin/`.

Le **build** Python fourni par Apple est installé dans `/System/Library/Frameworks/Python.framework` et `/usr/bin/python`. Vous ne devriez jamais les modifier ou supprimer, car ils sont contrôlés et utilisés par Apple (ou d'autres logiciels). Rappelez vous que si vous choisissez d'installer un Python plus récent depuis python.org, vous aurez deux installations de Python différentes et fonctionnelles sur votre ordinateur, il est donc important que vos chemins et utilisations soit cohérentes avec ce que vous voulez faire.

IDLE inclut un menu d'aide qui vous permet d'accéder à la documentation Python. Si vous êtes entièrement novice, vous devriez commencer par lire le tutoriel d'introduction dans ce document.

Si vous êtes familier avec Python sur d'autres plateformes Unix, vous devriez lire la section sur comment exécuter des scripts Python depuis un shell Unix.

4.1.1 Comment exécuter un script Python

Le meilleur moyen de démarrer avec Python sur Max OS X est d'utiliser l'environnement de développement intégré **IDLE**, voir la section *L'IDE* et utilisez le menu d'aide (**Help**) quand l'`IDE` est lancé.

If you want to run Python scripts from the Terminal window command line or from the Finder you first need an editor to create your script. Mac OS X comes with a number of standard Unix command line editors, **`vim`** and **`emacs`** among them. If you want a more Mac-like editor, **`BBEdit`** or **`TextWrangler`** from Bare Bones Software (see http://www.barebones.com/products/bbedit/index.html) are good choices, as is **`TextMate`** (see https://macromates.com/). Other editors include **`Gvim`** (http://macvim.org) and **`Aquamacs`** (http://aquamacs.org/).

Pour exécuter votre script depuis la fenêtre Terminal, vous devez vous assurer que `/usr/local/bin` est dans le chemin de recherche de votre shell (**PATH**).

Pour exécuter votre script depuis le Finder vous avez deux options :

- Glissez-le vers **`PythonLauncher`**
- Sélectionnez **`PythonLauncher`** en tant qu'application par défaut pour ouvrir votre script (ou n'importe quel script `.py`) depuis la fenêtre **info** de votre Finder puis double-cliquez votre script. **`PythonLauncher`** a des préférences variées pour contrôler comment votre script est exécuté. Glisser des options permets de les changer pour une invocation, ou utilisez le menu Préférences pour changer les choses globalement.

4.1.2 Lancer des scripts avec une interface graphique

Avec les anciennes versions de Python, il y a une bizarrerie Max OS X dont vous devez être au courant : les programmes qui communiquent avec le gestionnaires de fenêtre **Aqua** (en d'autres termes, tout ce qui a une interface graphique) doivent être exécutés de façon spécifique. Utilisez **`pythonw`** au lieu de **`python`** pour exécuter ce genre de scripts.

Avec Python 3.6, vous pouvez utilisez **`python`** ou **`pythonw`**.

4.1.3 Configuration

Python sur OS X respecte tous les standards Unix pour les variables d'environnement comme *`PYTHONPATH`*, mais définir ces variables pour des programmes exécutés depuis le Finder n'est pas standard car le Finder ne lit pas votre `.profile` ou `.cshrc` au démarrage. Vous devez créer un fichier `~/.MacOSX/environment.plist`. Voir le document technique d'Apple QA1067 pour plus de détails.

Pour plus d'informations sur l'installation de paquets Python dans **MacPython**, voir la section *Installation de paquets Python additionnels*.

4.2 L'*IDE*

MacPython ships with the standard IDLE development environment. A good introduction to using IDLE can be found at https://hkn.eecs.berkeley.edu/~dyoo/python/idle_intro/index.html.

4.3 Installation de paquets Python additionnels

Il y a plusieurs méthodes pour installer des paquets Python supplémentaires :

— Les paquets peuvent être installés en utilisant **distutils** (`python setup.py install`).
— Beaucoup de paquets peuvent aussi être installés via l'extension **`setuptools`** ou **`pip`**, voir https://pip.pypa.io/.

4.4 Programmation d'interface graphique sur le Mac

Il y a plusieurs options pour construire des applications avec interface graphique sur le Mac avec Python.

PyObjC est un **binding** Python vers le **framework** Objective-C/Cocoa d'Apple, qui est la base de la plupart des développements modernes sur Mac. Des informations sur PyObjC sont disponible à https://pythonhosted.org/pyobjc/.

La boîte à outils standard de Python pour des interfaces graphique est `tkinter`, basé sur la boite a outils multi-plateformes **Tk** (https://www.tcl.tk). Une version native **Aqua** de **Tk** est empaquetée avec OS X par Apple, et la dernière version peut être téléchargée et installée depuis https://www.activestate.com ; elle peut aussi être construite depuis les sources.

wxPython is another popular cross-platform GUI toolkit that runs natively on Mac OS X. Packages and documentation are available from http://www.wxpython.org.

PyQt est une boîte à outils multi-plateformes pour interfaces graphique populaire qui tourne nativement sur Mac OS X. Plus d'informations disponible sur https://riverbankcomputing.com/software/pyqt/intro.

4.5 Distribuer des Applications Python sur le Mac

L'outil « Build Applet » qui est placé dans le dossier MacPython 3.6 est suffisant pour empaqueter des petits scripts Python sur votre propre machine et pour les exécuter en tant qu'application Mac standard. Cependant, cet outil n'est pas assez robuste pour distribuer des applications Python à d'autres utilisateurs.

L'outil standard pour déployer des applications Python sur le Mac est **`py2app`**. Plus d'information sur l'installation et l'utilisation de **py2app** sur http://undefined.org/python/#py2app.

4.6 Autres ressources

La liste de diffusion courriel **MacPython** est une excellente ressource support pour les utilisateurs et développeurs Python sur Mac :

https://www.python.org/community/sigs/current/pythonmac-sig/

Une autre ressource utile est le wiki **MacPython** :

https://wiki.python.org/moin/MacPython

ANNEXE A

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.

. . . 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 décorateur.

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`; un point d'entrée indépendant est fourni via `Tools/scripts/2to3`. Cf. 2to3-reference.

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

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*, *function annotation*, **PEP 484** et **PEP 526**, qui décrivent cette fonctionnalité.

argument Valeur, donnée à une *fonction* ou à une *méthode* 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()` 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* 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 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* dans le glossaire, la question Différence entre argument et paramètre de la FAQ et la **PEP 362**.

gestionnaire de contexte asynchrone (*asynchronous context manager* en anglais) Objet contrôlant l'environnement à l'intérieur d'une instruction `with` en définissant les méthodes `__aenter__()` et `__aexit__()`. A été Introduit par la **PEP 492**.

générateur asynchrone Fonction qui renvoie un *asynchronous generator iterator*. Cela ressemble à une coroutine définie par `async def`, sauf qu'elle contient une ou des expressions `yield` produisant ainsi uns série de valeurs utilisables dans une boucle `async for`.

Usually refers to an asynchronous generator function, but may refer to an *asynchronous generator iterator* in some contexts. In cases where the intended meaning isn't clear, using the full terms avoids ambiguity.

Un générateur asynchrone peut contenir des expressions `await` ainsi que des instructions `async for`, et `async with`.

itérateur de générateur asynchrone Objet créé par une fonction *asynchronous generator*.

This is an *asynchronous iterator* which when called using the `__anext__()` method returns an awaitable object which will execute the body of the asynchronous generator function until the next `yield` expression.

Chaque `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__()`, elle repart de là où elle s'était arrêtée. Voir la **PEP 492** et la **PEP 525**.

itérable asynchrone Objet qui peut être utilisé dans une instruction `async for`. Sa méthode `__aiter__()` doit renvoyer un *asynchronous iterator*. A été introduit par la **PEP 492**.

itérateur asynchrone An object that implements the `__aiter__()` and `__anext__()` methods. `__anext__` must return an *awaitable* object. `async for` resolves the awaitables returned by an asynchronous iterator's `__anext__()` method until it raises a `StopAsyncIteration` exception. Introduced by **PEP 492**.

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`. Ce peut être une *coroutine* ou un objet avec une méthode `__await__()`. Voir aussi la **PEP 492**.

BDFL Dictateur bienveillant à vie (*Benevolent Dictator For Life* en anglais). Pseudonyme de Guido van Rossum, le créateur de Python.

fichier binaire Un *file object* capable de lire et d'écrire des *bytes-like objects*. 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` ou de `gzip.GzipFile`.

Consultez *fichier texte*, un objet fichier capable de lire et d'écrire des objets `str`.

objet octet-compatible Un objet gérant les bufferobjects et pouvant exporter un tampon (*buffer* en anglais) C-*contiguous*. Cela inclut les objets `bytes`, `bytearray` et `array.array`, ainsi que beaucoup d'objets `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` ou une `memoryview` d'un `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` ou `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* 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 fournit une liste des instructions du code intermédiaire.

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, alors qu'elles doivent avoir le même type pour être additionnées (sinon une exception `TypeError` serait levée). Sans coercition, toutes les opérandes, même de types compatibles, devraient être converties (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`, utilisez `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` en définissant les méthodes `__enter__()` et `__exit__()`. Consultez la **PEP 343**.

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 généralisées 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`. Voir aussi la **PEP 492**.

fonction coroutine Fonction qui renvoie un objet *coroutine*. Une fonction coroutine peut être définie par l'instruction `async def` et peut contenir les mots clés `await`, `async for` ainsi que `async with`. A été introduit par la **PEP 492**.

CPython L'implémentation canonique du langage de programmation Python, tel que distribué sur 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()` et `staticmethod()`.

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

```
def f(...):
    ...
f = staticmethod(f)
```

(suite sur la page suivante)

(suite de la page précédente)

```
@staticmethod
def f(...):
    ...
```

Quoique moins fréquemment utilisé, le même concept existe pour les classes. Consultez la documentation définitions de fonctions et définitions de classes pour en savoir plus sur les décorateurs.

descripteur N'importe quel objet définissant les méthodes `__get__()`, `__set__()`, ou `__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.

Pour plus d'informations sur les méthodes des descripteurs, consultez descriptors.

dictionnaire Structure de donnée associant des clés à des valeurs. Les clés peuvent être n'importe quel objet possédant les méthodes `__hash__()` et `__eq__()`. En Perl, les dictionnaires sont appelés « *hash* ».

vue de dictionnaire Objets retournés par les méthodes `dict.keys()`, `dict.values()` et `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 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()` ou `isinstance()`, Notez cependant que le *duck-typing* peut travailler de pair avec les *classes de base abstraites*. À la place, le *duck-typing* utilise plutôt `hasattr()` ou la programmation *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` et `except`. Cette technique de programmation contraste avec le style *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* qui ne peuvent pas être utilisées comme expressions, tel que `if`. 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. Voir la **PEP 498**.

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* bruts, les *fichiers binaires* avec tampon (*buffer*) et les *fichiers textes*. Leurs interfaces sont définies dans le module `io`. Le moyen le plus simple et direct de créer un objet fichier est d'utiliser la fonction `open()`.

objet fichier-compatible Synonyme de *objet fichier*.

chercheur Objet qui essaie de trouver un *chargeur* pour le module en cours d'importation.

Depuis Python 3.3, il existe deux types de chercheurs : les *chercheurs dans les méta-chemins* à utiliser avec `sys.meta_path` ; les *chercheurs d'entrée dans path* à utiliser avec `sys.path_hooks`.

Voir les **PEP 302**, **PEP 420** et **PEP 451** 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**.

fonction Suite d'instructions qui renvoie une valeur à son appelant. On peut lui passer des *arguments* qui pourront être utilisés dans le corps de la fonction. Voir aussi *paramètre*, *méthode* et function.

annotation de fonction *annotation* d'un paramètre de fonction ou valeur de retour.

Function annotations are usually used for *type hints* : for example, this function is expected to take two `int` arguments and is also expected to have an `int` return value :

```
def sum_two_numbers(a: int, b: int) -> int:
   return a + b
```

L'annotation syntaxique de la fonction est expliquée dans la section function.

Voir *variable annotation* et **PEP 484**, 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__` 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`.

générateur Fonction qui renvoie un *itérateur de générateur*. Cela ressemble à une fonction normale, en dehors du fait qu'elle contient une ou des expressions `yield` produisant une série de valeurs utilisable dans une boucle *for* ou récupérées une à une via la fonction `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'ambigüité.

itérateur de générateur Objet créé par une fonction *générateur*.

Chaque `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 expression `for` définissant une variable de boucle, un intervalle et une expression `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*, le décorateur `functools.singledispatch()` et la **PEP 443**.

GIL Voir *global interpreter lock*.

verrou global de l'interpréteur (*global interpreter lock* en anglais) Mécanisme utilisé par l'interpréteur *CPython* pour s'assurer qu'un seul fil d'exécution (*thread* en anglais) n'exécute le *bytecode* à la fois. Cela simplifie l'implémentation de CPython en rendant le modèle objet (incluant des parties critiques comme la classe native `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.

hachable Un objet est *hachable* s'il a une empreinte (*hash*) qui ne change jamais (il doit donc implémenter une méthode `__hash__()`) et s'il peut être comparé à d'autres objets (avec la méthode `__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*.

Tous les types immuables natifs de Python sont hachables, mais les conteneurs muables (comme les listes ou les dictionnaires) ne le sont pas. Toutes 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()`.

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* dans lesquelles le *chercheur basé sur les chemins* cherche les modules à importer. Typiquement, lors d'une importation, cette liste vient de `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* et un *chargeur*.

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*.

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*. 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`, `str`, et `tuple`), quelques autres types comme `dict`, *objets fichiers* ou tout objet d'une classe ayant une méthode `__iter__()` ou `__getitem__()` qui implémente la sémantique d'une *Sequence*.

Les itérables peuvent être utilisés dans des boucles `for` et à beaucoup d'autres endroits où une séquence est requise (`zip()`, `map()`...). Lorsqu'un itérable est passé comme argument à la fonction native `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()` 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*, *séquence* et *générateur*.

itérateur Objet représentant un flux de donnée. Des appels successifs à la méthode `__next__()` de l'itérateur (ou le passer à la fonction native `next()`) donne successivement les objets du flux. Lorsque plus aucune donnée n'est disponible, une exception `StopIteration` est levée. À ce point, l'itérateur est épuisé et tous les appels suivants à sa méthode `__next__()` lèveront encore une exception `StopIteration`. Les itérateurs doivent avoir une méthode `__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`) produit un nouvel itérateur neuf à chaque fois qu'il est passé à la fonction `iter()` ou s'il est utilisé dans une boucle `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 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()` 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()`, `max()`, `sorted()`, `list.sort()`, `heapq.merge()`, `heapq.nsmallest()`, `heapq.nlargest()` et `itertools.groupby()`.

Il existe plusieurs moyens de créer une fonction clé. Par exemple, la méthode `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`, comme `lambda r: (r[0], r[2])`. Vous noterez que le module `operator` propose des constructeurs de fonctions clefs : `attrgetter()`, `itemgetter()` et `methodcaller()`. Voir Comment Trier pour des exemples de création et d'utilisation de fonctions clefs.

argument nommé Voir *argument*.

lambda Fonction anonyme sous la forme d'une *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* et se caractérise par la présence de beaucoup d'instructions `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* 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 intension) 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` 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*. Voir la **PEP 302** pour plus de détails et `importlib.ABC.Loader` pour sa *classe de base abstraite*.

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 `collections.abc.Mapping`

ou `collections.abc.MutableMapping`. Les classes suivantes sont des exemples de tableaux de correspondances : `dict`, `collections.defaultdict`, `collections.OrderedDict` et `collections.Counter`.

chercheur dans les méta-chemins Un *chercheur* renvoyé par une recherche dans `sys.meta_path`. Les chercheurs dans les méta-chemins ressemblent, mais sont différents des *chercheurs d'entrée dans path*.

Voir `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 : 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* (qui, par convention, est habituellement nommé `self`). Voir *function* et *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 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*.

Voir aussi *paquet*.

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`.

MRO Voir *ordre de résolution des méthodes*.

muable Un objet muable peut changer de valeur tout en gardant le même `id()`. Voir aussi *immuable*.

n-uplet nommé (*named-tuple* en anglais) Classe qui, comme un *n-uplet* (*tuple* en anglais), a ses éléments accessibles par leur indice. Et en plus, les éléments sont accessibles par leur nom. Par exemple, `time.localtime()` donne un objet ressemblant à un *n-uplet*, dont *year* est accessible par son indice : `t[0]` ou par son nom : `t.tm_year`).

Un *n-uplet nommé* peut être un type natif tel que `time.struct_time` ou il peut être construit comme une simple classe. Un *n-uplet nommé* complet peut aussi être créé via la fonction `collections.namedtuple()`. Cette dernière approche fournit automatiquement des fonctionnalités supplémentaires, tel qu'une représentation lisible comme `Employee(name='jones', title='programmer')`.

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` et `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()` ou `itertools.islice()` affiche clairement que ces fonctions sont implémentées respectivement dans les modules `random` et `itertools`.

paquet-espace de nommage Un *paquet* tel que défini dans la **PEP 421** 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* puisqu'ils n'ont pas de fichier `__init__.py`.

Voir aussi *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` 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__`, les descripteurs, les propriétés, `__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*.

paquet *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* et *namespace package*.

paramètre Entité nommée dans la définition d'une *fonction* (ou méthode), décrivant un *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*, soit en tant que *argument nommé*. C'est le type de paramètre par défaut. Par exemple, *foo* et *bar* dans l'exemple suivant :

```
def func(foo, bar=None): ...
```

— *positional-only* : l'argument ne peut être donné que par sa position. Python n'a pas de syntaxe pour déclarer de tels paramètres, cependant des fonctions natives, comme `abs()`, en utilisent.

— *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* dans le glossaire, la question sur la différence entre les arguments et les paramètres dans la FAQ, la classe `inspect.Parameter`, la section function et la **PEP 362**.

entrée de chemin Emplacement dans le *chemin des importations* (*import path* en anglais, d'où le *path*) que le *chercheur basé sur les chemins* consulte pour trouver des modules à importer.

chercheur de chemins *chercheur* renvoyé par un appelable sur un `sys.path_hooks` (c'est-à-dire un *point d'entrée pour la recherche dans path*) qui sait où trouver des modules lorsqu'on lui donne une *entrée de path*.

Voir `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* s'il sait où trouver des modules pour une *entrée dans path* donnée.

chercheur basé sur les chemins L'un des *chercheurs dans les méta-chemins* par défaut qui cherche des modules dans un *chemin des importations*.

objet simili-chemin Objet représentant un chemin du système de fichiers. Un objet simili-chemin est un objet `str` ou un objet `bytes` représentant un chemin ou un objet implémentant le protocole `os.PathLike`. Un objet qui accepte le protocole `os.PathLike` peut être converti en un chemin `str` ou `bytes` du système de fichiers en appelant la fonction `os.fspath()`. `os.fsdecode()` et `os.fsencode()` peuvent être utilisées, respectivement, pour garantir un résultat de type `str` ou `bytes` à la place. A été Introduit par la **PEP 519**.

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**.

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**.

argument positionnel Voir *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** pour plus de détails.

paquet provisoire Voir *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`. 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**. 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*. Le module `sys` définit une fonction `getrefcount()` que les développeurs peuvent utiliser pour obtenir le nombre de références à un objet donné.

paquet classique *paquet* traditionnel, tel qu'un dossier contenant un fichier `__init__.py`.

Voir aussi *paquet-espace de nommage*.

`__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* qui offre un accès efficace à ses éléments par un indice sous forme de nombre entier via la méthode spéciale `__getitem__()` et qui définit une méthode `__len__()` donnant sa taille. Voici quelques séquences natives : `list`, `str`, `tuple`, et `bytes`. Notez que `dict` possède aussi une méthode `__getitem__()` et une méthode `__len__()`, mais il est considéré comme un *mapping* plutôt qu'une séquence, car ses accès se font par une clé arbitraire *immuable* plutôt qu'un nombre entier.

La classe abstraite de base `collections.abc.Sequence` définit une interface plus riche qui va au-delà des simples `__getitem__()` et `__len__()`, en ajoutant `count()`, `index()`, `__contains__()` et `__reversed__()`. Les types qui implémentent cette interface étendue peuvent s'enregistrer explicitement en utilisant `register()`.

distribution simple Forme de distribution, comme les *fonction génériques*, 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*. 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` 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 specialnames.

instruction Une instruction (*statement* en anglais) est un composant d'un « bloc » de code. Une instruction est soit une *expression*, soit une ou plusieurs constructions basées sur un mot-clé, comme `if`, `while` ou `for`.

struct sequence A tuple with named elements. Struct sequences expose an interface similar to *named tuple* in that elements can be accessed either by index or as an attribute. However, they do not have any of the named tuple methods like `_make()` or `_asdict()`. Examples of struct sequences include `sys.float_info` and the return value of `os.stat()`.

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

fichier texte *file object* capable de lire et d'écrire des objets `str`. Souvent, un fichier texte (*text file* en anglais) accède en fait à un flux de donnée en octets et gère l'*text encoding* automatiquement. Des exemples de fichiers textes sont les fichiers ouverts en mode texte (`'r'` ou `'w'`), `sys.stdin`, `sys.stdout` et les instances de `io.StringIO`.

Voir aussi *binary file* pour un objet fichier capable de lire et d'écrire *bytes-like objects*.

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__` 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*. Par exemple :

```
from typing import List, Tuple

def remove_gray_shades(
        colors: List[Tuple[int, int, int]]) -> List[Tuple[int, int, int]]:
    pass
```

pourrait être rendu plus lisible comme ceci :

```
from typing import List, Tuple

Color = Tuple[int, int, int]

def remove_gray_shades(colors: List[Color]) -> List[Color]:
    pass
```

Voir `typing` et **PEP 484**, qui décrivent cette fonctionnalité.

indication de type Le *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()`.

Voir `typing` et **PEP 484**, 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** et la **PEP 3116**, ainsi que la fonction `bytes.splitlines()` pour d'autres usages.

annotation de variable *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* : par exemple, cette variable devrait prendre des valeurs de type `int` :

```
count: int = 0
```

La syntaxe d'annotation de la variable est expliquée dans la section annassign.

Reportez-vous à *function annotation*, à la **PEP 484** et à la **PEP 526** 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`.

machine virtuelle Ordinateur défini entièrement par du logiciel. La machine virtuelle (*virtual machine*) de Python exécute le *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.

ANNEXE B

À propos de ces documents

Ces documents sont générés à partir de sources en reStructuredText par Sphinx, un analyseur de documents spécialement conçu pour la documentation Python.

Le développement de la documentation et de ses outils est entièrement basé sur le volontariat, tout comme Python. Si vous voulez contribuer, allez voir la page reporting-bugs qui contient des informations pour vous y aider. Les nouveaux volontaires sont toujours les bienvenus !

Merci beaucoup à :

— Fred L. Drake, Jr., créateur des outils originaux de la documentation Python et rédacteur de la plupart de son contenu ;
— le projet Docutils pour avoir créé *reStructuredText* et la suite d'outils *Docutils* ;
— Fredrik Lundh pour son projet Alternative Python Reference, dont Sphinx a pris beaucoup de bonnes idées.

B.1 Contributeurs de la documentation Python

De nombreuses personnes ont contribué au langage Python, à sa bibliothèque standard et à sa documentation. Consultez Misc/ACKS dans les sources de la distribution Python pour avoir une liste partielle des contributeurs.

Ce n'est que grâce aux suggestions et contributions de la communauté Python que Python a une documentation si merveilleuse — Merci !

ANNEXE C

Histoire et licence

C.1 Histoire du logiciel

Python a été créé au début des années 1990 par Guido van Rossum, au Stichting Mathematisch Centrum (CWI, voir https://www.cwi.nl/) au Pays-Bas en tant que successeur d'un langage appelé ABC. Guido est l'auteur principal de Python, bien qu'il inclut de nombreuses contributions de la part d'autres personnes.

En 1995, Guido continua son travail sur Python au Corporation for National Research Initiatives (CNRI, voir https://www.cnri.reston.va.us/) de Reston, en Viriginie, d'où il diffusa plusieurs versions du logiciel.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. In October of the same year, the PythonLabs team moved to Digital Creations (now Zope Corporation ; see https://www.zope.org/). In 2001, the Python Software Foundation (PSF, see https://www.python.org/psf/) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation is a sponsoring member of the PSF.

Toutes les versions de Python sont Open Source (voir https://www.opensource.org/ pour la définition d'Open Source). Historiquement, la plupart, mais pas toutes, des versions de Python ont également été compatible avec la GPL, le tableau ci-dessous résume les différentes versions.

Version	Dérivé de	Année	Propriétaire	Compatible avec la GPL ?
0.9.0 à 1.2	n/a	1991-1995	CWI	oui
1.3 à 1.5.2	1.2	1995-1999	CNRI	oui
1.6	1.5.2	2000	CNRI	non
2.0	1.6	2000	BeOpen.com	non
1.6.1	1.6	2001	CNRI	non
2.1	2.0+1.6.1	2001	PSF	non
2.0.1	2.0+1.6.1	2001	PSF	oui
2.1.1	2.1+2.0.1	2001	PSF	oui
2.1.2	2.1.1	2002	PSF	oui
2.1.3	2.1.2	2002	PSF	oui
2.2 et ultérieure	2.1.1	2001-maintenant	PSF	oui

Note : Compatible GPL ne signifie pas que nous distribuons Python sous licence GPL. Toutes les licences Python, excepté la licence GPL, vous permettent la distribution d'une version modifiée sans rendre open source ces changements. La licence « compatible GPL » rend possible la diffusion de Python avec un autre logiciel qui est lui, diffusé sous la licence GPL ; les licences « non-compatibles GPL » ne le peuvent pas.

Merci aux nombreux bénévoles qui ont travaillé sous la direction de Guido pour rendre ces versions possibles.

C.2 Conditions générales pour accéder à, ou utiliser, Python

C.2.1 PSF LICENSE AGREEMENT FOR PYTHON 3.6.11

```
1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"),␣
↪and
   the Individual or Organization ("Licensee") accessing and otherwise using␣
↪Python
   3.6.11 software in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, PSF hereby
   grants Licensee a nonexclusive, royalty-free, world-wide license to␣
↪reproduce,
   analyze, test, perform and/or display publicly, prepare derivative works,
   distribute, and otherwise use Python 3.6.11 alone or in any derivative
   version, provided, however, that PSF's License Agreement and PSF's notice␣
↪of
   copyright, i.e., "Copyright © 2001-2020 Python Software Foundation; All␣
↪Rights
   Reserved" are retained in Python 3.6.11 alone or in any derivative version
   prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or
   incorporates Python 3.6.11 or any part thereof, and wants to make the
   derivative work available to others as provided herein, then Licensee␣
↪hereby
   agrees to include in any such work a brief summary of the changes made to␣
↪Python
   3.6.11.

4. PSF is making Python 3.6.11 available to Licensee on an "AS IS" basis.
   PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED.  BY WAY OF
   EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION␣
↪OR
   WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT␣
↪THE
   USE OF PYTHON 3.6.11 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 3.6.11
   FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT␣
↪OF
   MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 3.6.11, OR ANY␣
↪DERIVATIVE
```

```
   THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach␣
↪of
   its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any␣
↪relationship
   of agency, partnership, or joint venture between PSF and Licensee.  This␣
↪License
   Agreement does not grant permission to use PSF trademarks or trade name in␣
↪a
   trademark sense to endorse or promote products or services of Licensee, or␣
↪any
   third party.

8. By copying, installing or otherwise using Python 3.6.11, Licensee agrees
   to be bound by the terms and conditions of this License Agreement.
```

C.2.2 LICENCE D'UTILISATION BEOPEN.COM POUR PYTHON 2.0

LICENCE D'UTILISATION LIBRE BEOPEN PYTHON VERSION 1

```
1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at
   160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization
   ("Licensee") accessing and otherwise using this software in source or binary
   form and its associated documentation ("the Software").

2. Subject to the terms and conditions of this BeOpen Python License Agreement,
   BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license
   to reproduce, analyze, test, perform and/or display publicly, prepare derivative
   works, distribute, and otherwise use the Software alone or in any derivative
   version, provided, however, that the BeOpen Python License is retained in the
   Software, alone or in any derivative version prepared by Licensee.

3. BeOpen is making the Software available to Licensee on an "AS IS" basis.
   BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED.  BY WAY OF
   EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR
   WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE
   USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR
   ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING,
   MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF
   ADVISED OF THE POSSIBILITY THEREOF.

5. This License Agreement will automatically terminate upon a material breach of
   its terms and conditions.

6. This License Agreement shall be governed by and interpreted in all respects
   by the law of the State of California, excluding conflict of law provisions.
   Nothing in this License Agreement shall be deemed to create any relationship of
   agency, partnership, or joint venture between BeOpen and Licensee.  This License
   Agreement does not grant permission to use BeOpen trademarks or trade names in a
   trademark sense to endorse or promote products or services of Licensee, or any
```

(suite sur la page suivante)

(suite de la page précédente)

```
   third party.  As an exception, the "BeOpen Python" logos available at
   http://www.pythonlabs.com/logos.html may be used according to the permissions
   granted on that web page.

7. By copying, installing or otherwise using the software, Licensee agrees to be
   bound by the terms and conditions of this License Agreement.
```

C.2.3 LICENCE D'UTILISATION CNRI POUR PYTHON 1.6.1

```
1. This LICENSE AGREEMENT is between the Corporation for National Research
   Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191
   ("CNRI"), and the Individual or Organization ("Licensee") accessing and
   otherwise using Python 1.6.1 software in source or binary form and its
   associated documentation.

2. Subject to the terms and conditions of this License Agreement, CNRI hereby
   grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce,
   analyze, test, perform and/or display publicly, prepare derivative works,
   distribute, and otherwise use Python 1.6.1 alone or in any derivative version,
   provided, however, that CNRI's License Agreement and CNRI's notice of copyright,
   i.e., "Copyright © 1995-2001 Corporation for National Research Initiatives; All
   Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version
   prepared by Licensee.  Alternately, in lieu of CNRI's License Agreement,
   Licensee may substitute the following text (omitting the quotes): "Python 1.6.1
   is made available subject to the terms and conditions in CNRI's License
   Agreement.  This Agreement together with Python 1.6.1 may be located on the
   Internet using the following unique, persistent identifier (known as a handle):
   1895.22/1013.  This Agreement may also be obtained from a proxy server on the
   Internet using the following URL: http://hdl.handle.net/1895.22/1013."

3. In the event Licensee prepares a derivative work that is based on or
   incorporates Python 1.6.1 or any part thereof, and wants to make the derivative
   work available to others as provided herein, then Licensee hereby agrees to
   include in any such work a brief summary of the changes made to Python 1.6.1.

4. CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis.  CNRI
   MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED.  BY WAY OF EXAMPLE,
   BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY
   OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF
   PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR
   ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF
   MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE
   THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of
   its terms and conditions.

7. This License Agreement shall be governed by the federal intellectual property
   law of the United States, including without limitation the federal copyright
   law, and, to the extent such U.S. federal law does not apply, by the law of the
   Commonwealth of Virginia, excluding Virginia's conflict of law provisions.
   Notwithstanding the foregoing, with regard to derivative works based on Python
```

(suite sur la page suivante)

(suite de la page précédente)

```
   1.6.1 that incorporate non-separable material that was previously distributed
   under the GNU General Public License (GPL), the law of the Commonwealth of
   Virginia shall govern this License Agreement only as to issues arising under or
   with respect to Paragraphs 4, 5, and 7 of this License Agreement.  Nothing in
   this License Agreement shall be deemed to create any relationship of agency,
   partnership, or joint venture between CNRI and Licensee.  This License Agreement
   does not grant permission to use CNRI trademarks or trade name in a trademark
   sense to endorse or promote products or services of Licensee, or any third
   party.

8. By clicking on the "ACCEPT" button where indicated, or by copying, installing
   or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and
   conditions of this License Agreement.
```

C.2.4 LICENCE D'UTILISATION CWI POUR PYTHON 0.9.0 à 1.2

```
Copyright © 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The
Netherlands.  All rights reserved.

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted, provided that
the above copyright notice appear in all copies and that both that copyright
notice and this permission notice appear in supporting documentation, and that
the name of Stichting Mathematisch Centrum or CWI not be used in advertising or
publicity pertaining to distribution of the software without specific, written
prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO
EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT
OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE,
DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS
SOFTWARE.
```

C.3 Licences et remerciements pour les logiciels tiers

Cette section est une liste incomplète mais grandissante de licences et remerciements pour les logiciels tiers incorporés dans la distribution de Python.

C.3.1 Mersenne twister

Le module `_random` inclut du code construit à partir d'un téléchargement depuis http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html. Voici mot pour mot les commentaires du code original :

```
A C-program for MT19937, with initialization improved 2002/1/26.
Coded by Takuji Nishimura and Makoto Matsumoto.

Before using, initialize the state by using init_genrand(seed)
or init_by_array(init_key, key_length).
```

(suite sur la page suivante)

(suite de la page précédente)

```
Copyright (C) 1997 - 2002, Makoto Matsumoto and Takuji Nishimura,
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

 1. Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.

 2. Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

 3. The names of its contributors may not be used to endorse or promote
    products derived from this software without specific prior written
    permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


Any feedback is very welcome.
http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
email: m-mat @ math.sci.hiroshima-u.ac.jp (remove space)
```

C.3.2 Interfaces de connexion (*sockets*)

Le module `socket` utilise les fonctions `getaddrinfo()` et `getnameinfo()` codées dans des fichiers source séparés et provenant du projet WIDE : http://www.wide.ad.jp/.

```
Copyright (C) 1995, 1996, 1997, and 1998 WIDE Project.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. Neither the name of the project nor the names of its contributors
   may be used to endorse or promote products derived from this software
   without specific prior written permission.
```

(suite sur la page suivante)

(suite de la page précédente)

```
THIS SOFTWARE IS PROVIDED BY THE PROJECT AND CONTRIBUTORS ``AS IS'' AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.  IN NO EVENT SHALL THE PROJECT OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
```

C.3.3 Virgule flottante et contrôle d'exception

Le code source pour le module `fpectl` inclut la note suivante :

```
  ---------------------------------------------------------------------
 /                      Copyright (c) 1996.                            \
|          The Regents of the University of California.                 |
|                        All rights reserved.                           |
|                                                                       |
|   Permission to use, copy, modify, and distribute this software for   |
|   any purpose without fee is hereby granted, provided that this en-   |
|   tire notice is included in all copies of any software which is or   |
|   includes  a  copy  or  modification  of  this software and in all   |
|   copies of the supporting documentation for such software.           |
|                                                                       |
|   This  work was produced at the University of California, Lawrence   |
|   Livermore National Laboratory under  contract  no.  W-7405-ENG-48   |
|   between  the  U.S.  Department  of  Energy and The Regents of the   |
|   University of California for the operation of UC LLNL.              |
|                                                                       |
|                              DISCLAIMER                               |
|                                                                       |
|   This  software was prepared as an account of work sponsored by an   |
|   agency of the United States Government. Neither the United States   |
|   Government  nor the University of California nor any of their em-   |
|   ployees, makes any warranty, express or implied, or  assumes  any   |
|   liability  or  responsibility  for the accuracy, completeness, or   |
|   usefulness of any information,  apparatus,  product,  or  process   |
|   disclosed,   or  represents  that  its  use  would  not  infringe   |
|   privately-owned rights. Reference herein to any specific  commer-   |
|   cial  products,  process,  or  service  by trade name, trademark,   |
|   manufacturer, or otherwise, does not  necessarily  constitute  or   |
|   imply  its endorsement, recommendation, or favoring by the United   |
|   States Government or the University of California. The views  and   |
|   opinions  of authors expressed herein do not necessarily state or   |
|   reflect those of the United States Government or  the  University   |
|   of  California,  and shall not be used for advertising or product   |
 \  endorsement purposes.                                              /
  ---------------------------------------------------------------------
```

C.3.4 Interfaces de connexion asynchrones

Les modules `asynchat` et `asyncore` contiennent la note suivante :

```
Copyright 1996 by Sam Rushing

                        All Rights Reserved

Permission to use, copy, modify, and distribute this software and
its documentation for any purpose and without fee is hereby
granted, provided that the above copyright notice appear in all
copies and that both that copyright notice and this permission
notice appear in supporting documentation, and that the name of Sam
Rushing not be used in advertising or publicity pertaining to
distribution of the software without specific, written prior
permission.

SAM RUSHING DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN
NO EVENT SHALL SAM RUSHING BE LIABLE FOR ANY SPECIAL, INDIRECT OR
CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS
OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
```

C.3.5 Gestion de témoin (*cookie*)

Le module `http.cookies` contient la note suivante :

```
Copyright 2000 by Timothy O'Malley <timo@alum.mit.edu>

               All Rights Reserved

Permission to use, copy, modify, and distribute this software
and its documentation for any purpose and without fee is hereby
granted, provided that the above copyright notice appear in all
copies and that both that copyright notice and this permission
notice appear in supporting documentation, and that the name of
Timothy O'Malley  not be used in advertising or publicity
pertaining to distribution of the software without specific, written
prior permission.

Timothy O'Malley DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS, IN NO EVENT SHALL Timothy O'Malley BE LIABLE FOR
ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
```

C.3.6 Traçage d'exécution

Le module `trace` contient la note suivante :

```
portions copyright 2001, Autonomous Zones Industries, Inc., all rights...
err...  reserved and offered to the public under the terms of the
Python 2.2 license.
Author: Zooko O'Whielacronx
http://zooko.com/
mailto:zooko@zooko.com

Copyright 2000, Mojam Media, Inc., all rights reserved.
Author: Skip Montanaro

Copyright 1999, Bioreason, Inc., all rights reserved.
Author: Andrew Dalke

Copyright 1995-1997, Automatrix, Inc., all rights reserved.
Author: Skip Montanaro

Copyright 1991-1995, Stichting Mathematisch Centrum, all rights reserved.


Permission to use, copy, modify, and distribute this Python software and
its associated documentation for any purpose without fee is hereby
granted, provided that the above copyright notice appears in all copies,
and that both that copyright notice and this permission notice appear in
supporting documentation, and that the name of neither Automatrix,
Bioreason or Mojam Media be used in advertising or publicity pertaining to
distribution of the software without specific, written prior permission.
```

C.3.7 Les fonctions UUencode et UUdecode

Le module `uu` contient la note suivante :

```
Copyright 1994 by Lance Ellinghouse
Cathedral City, California Republic, United States of America.
                       All Rights Reserved
Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in
supporting documentation, and that the name of Lance Ellinghouse
not be used in advertising or publicity pertaining to distribution
of the software without specific, written prior permission.
LANCE ELLINGHOUSE DISCLAIMS ALL WARRANTIES WITH REGARD TO
THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS, IN NO EVENT SHALL LANCE ELLINGHOUSE CENTRUM BE LIABLE
FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT
OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Modified by Jack Jansen, CWI, July 1995:
- Use binascii module to do the actual line-by-line conversion
  between ascii and binary. This results in a 1000-fold speedup. The C
```

(suite sur la page suivante)

(suite de la page précédente)

```
  version is still 5 times faster, though.
- Arguments more compliant with Python standard
```

C.3.8 Appel de procédures distantes en XML (*RPC*, pour *Remote Procedure Call*)

Le module `xmlrpc.client` contient la note suivante :

```
    The XML-RPC client interface is

Copyright (c) 1999-2002 by Secret Labs AB
Copyright (c) 1999-2002 by Fredrik Lundh

By obtaining, using, and/or copying this software and/or its
associated documentation, you agree that you have read, understood,
and will comply with the following terms and conditions:

Permission to use, copy, modify, and distribute this software and
its associated documentation for any purpose and without fee is
hereby granted, provided that the above copyright notice appears in
all copies, and that both that copyright notice and this permission
notice appear in supporting documentation, and that the name of
Secret Labs AB or the author not be used in advertising or publicity
pertaining to distribution of the software without specific, written
prior permission.

SECRET LABS AB AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD
TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANT-
ABILITY AND FITNESS.  IN NO EVENT SHALL SECRET LABS AB OR THE AUTHOR
BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY
DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE
OF THIS SOFTWARE.
```

C.3.9 test_epoll

Le module `test_epoll` contient la note suivante :

```
Copyright (c) 2001-2006 Twisted Matrix Laboratories.

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
```

(suite sur la page suivante)

(suite de la page précédente)

```
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

C.3.10 Select kqueue

Le module `select` contient la note suivante pour l'interface *kqueue* :

```
Copyright (c) 2000 Doug White, 2006 James Knight, 2007 Christian Heimes
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
```

C.3.11 SipHash24

Le fichier `Python/pyhash.c` contiens une implémentation par Marek Majkowski de l'algorithme *SipHash24* de Dan Bernstein. Il contient la note suivante :

```
<MIT License>
Copyright (c) 2013  Marek Majkowski <marek@popcount.org>

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.
</MIT License>
```

(suite sur la page suivante)

(suite de la page précédente)

```
Original location:
   https://github.com/majek/csiphash/

Solution inspired by code from:
   Samuel Neves (supercop/crypto_auth/siphash24/little)
   djb (supercop/crypto_auth/siphash24/little2)
   Jean-Philippe Aumasson (https://131002.net/siphash/siphash24.c)
```

C.3.12 *strtod* et *dtoa*

Le fichier `Python/dtoa.c`, qui fournit les fonctions `dtoa` et `strtod` pour la conversion de *doubles* C vers et depuis les chaînes, est tiré d'un fichier du même nom par David M. Gay, actuellement disponible sur http://www.netlib.org/fp/. Le fichier original, tel que récupéré le 16 mars 2009, contient la licence suivante :

```
/****************************************************************
 *
 * The author of this software is David M. Gay.
 *
 * Copyright (c) 1991, 2000, 2001 by Lucent Technologies.
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose without fee is hereby granted, provided that this entire notice
 * is included in all copies of any software which is or includes a copy
 * or modification of this software and in all copies of the supporting
 * documentation for such software.
 *
 * THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED
 * WARRANTY.  IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY
 * REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY
 * OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.
 *
 ***************************************************************/
```

C.3.13 OpenSSL

Les modules `hashlib`, `posix`, `ssl`, et `crypt` utilisent la bibliothèque OpenSSL pour améliorer les performances, si elle est disponible via le système d'exploitation. Aussi les outils d'installation sur Windows et Mac OS X peuvent inclure une copie des bibliothèques d'OpenSSL, donc on colle une copie de la licence d'OpenSSL ici :

```
 LICENSE ISSUES
 ==============

 The OpenSSL toolkit stays under a dual license, i.e. both the conditions of
 the OpenSSL License and the original SSLeay license apply to the toolkit.
 See below for the actual license texts. Actually both licenses are BSD-style
 Open Source licenses. In case of any license issues related to OpenSSL
 please contact openssl-core@openssl.org.

 OpenSSL License
 ---------------

   /* ====================================================================
    * Copyright (c) 1998-2008 The OpenSSL Project.  All rights reserved.
```

(suite sur la page suivante)

(suite de la page précédente)

```
    *
    * Redistribution and use in source and binary forms, with or without
    * modification, are permitted provided that the following conditions
    * are met:
    *
    * 1. Redistributions of source code must retain the above copyright
    *    notice, this list of conditions and the following disclaimer.
    *
    * 2. Redistributions in binary form must reproduce the above copyright
    *    notice, this list of conditions and the following disclaimer in
    *    the documentation and/or other materials provided with the
    *    distribution.
    *
    * 3. All advertising materials mentioning features or use of this
    *    software must display the following acknowledgment:
    *    "This product includes software developed by the OpenSSL Project
    *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
    *
    * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
    *    endorse or promote products derived from this software without
    *    prior written permission. For written permission, please contact
    *    openssl-core@openssl.org.
    *
    * 5. Products derived from this software may not be called "OpenSSL"
    *    nor may "OpenSSL" appear in their names without prior written
    *    permission of the OpenSSL Project.
    *
    * 6. Redistributions of any form whatsoever must retain the following
    *    acknowledgment:
    *    "This product includes software developed by the OpenSSL Project
    *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
    *
    * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
    * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
    * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
    * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
    * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
    * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
    * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
    * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
    * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
    * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
    * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
    * OF THE POSSIBILITY OF SUCH DAMAGE.
    * ====================================================================
    *
    * This product includes cryptographic software written by Eric Young
    * (eay@cryptsoft.com).  This product includes software written by Tim
    * Hudson (tjh@cryptsoft.com).
    *
    */

Original SSLeay License
-----------------------

   /* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
    * All rights reserved.
```

(suite sur la page suivante)

(suite de la page précédente)

```
 *
 * This package is an SSL implementation written
 * by Eric Young (eay@cryptsoft.com).
 * The implementation was written so as to conform with Netscapes SSL.
 *
 * This library is free for commercial and non-commercial use as long as
 * the following conditions are aheared to.  The following conditions
 * apply to all code found in this distribution, be it the RC4, RSA,
 * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
 * included with this distribution is covered by the same copyright terms
 * except that the holder is Tim Hudson (tjh@cryptsoft.com).
 *
 * Copyright remains Eric Young's, and as such any Copyright notices in
 * the code are not to be removed.
 * If this package is used in a product, Eric Young should be given attribution
 * as the author of the parts of the library used.
 * This can be in the form of a textual message at program startup or
 * in documentation (online or textual) provided with the package.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *    "This product includes cryptographic software written by
 *     Eric Young (eay@cryptsoft.com)"
 *    The word 'cryptographic' can be left out if the rouines from the library
 *    being used are not cryptographic related :-).
 * 4. If you include any Windows specific code (or a derivative thereof) from
 *    the apps directory (application code) you must include an acknowledgement:
 *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
 *
 * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed.  i.e. this code cannot simply be
 * copied and put under another distribution licence
 * [including the GNU Public Licence.]
 */
```

C.3.14 expat

Le module `pyexpat` est compilé avec une copie des sources d'*expat*, sauf si la compilation est configurée avec `--with-system-expat` :

```
Copyright (c) 1998, 1999, 2000 Thai Open Source Software Center Ltd
                               and Clark Cooper

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

C.3.15 libffi

Le module `_ctypes` est compilé en utilisant une copie des sources de la *libffi*, sauf si la compilation est configurée avec `--with-system-libffi` :

```
Copyright (c) 1996-2008  Red Hat, Inc and others.

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
``Software''), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED ``AS IS'', WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT.  IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.
```

C.3.16 zlib

Le module `zlib` est compilé en utilisant une copie du code source de *zlib* si la version de *zlib* trouvée sur le système est trop vieille pour être utilisée :

```
Copyright (C) 1995-2011 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied
warranty.  In no event will the authors be held liable for any damages
arising from the use of this software.

Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute it
freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not
   claim that you wrote the original software. If you use this software
   in a product, an acknowledgment in the product documentation would be
   appreciated but is not required.

2. Altered source versions must be plainly marked as such, and must not be
   misrepresented as being the original software.

3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly        Mark Adler
jloup@gzip.org          madler@alumni.caltech.edu
```

C.3.17 cfuhash

L'implémentation des dictionnaires, utilisée par le module `tracemalloc` est basée sur le projet *cfuhash* :

```
Copyright (c) 2005 Don Owens
All rights reserved.

This code is released under the BSD license:

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

  * Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.

  * Redistributions in binary form must reproduce the above
    copyright notice, this list of conditions and the following
    disclaimer in the documentation and/or other materials provided
    with the distribution.

  * Neither the name of the author nor the names of its
    contributors may be used to endorse or promote products derived
    from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
```

(suite sur la page suivante)

(suite de la page précédente)

```
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.
```

C.3.18 libmpdec

Le module `_decimal` est construit en incluant une copie de la bibliothèque *libmpdec*, sauf si elle est compilée avec `--with-system-libmpdec`:

```
Copyright (c) 2008-2016 Stefan Krah. All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
```

ANNEXE D

Copyright

Python et cette documentation sont :

Voir *Histoire et licence* pour des informations complètes concernant la licence et les permissions.

Index

Non alphabétique

A

B

C

D

P

Q

R

S

T

U

V

W

X