Python Setup and Usage

Version 3.14.0a5

Guido van Rossum and the Python development team

février 17, 2025

Python Software Foundation
Email : docs@python.org

Table des matières

1 Ligne de commande et environnement **3**
1.1 Ligne de commande . . . 3
1.1.1 Options de l'interface . . . 3
1.1.2 Options génériques . . . 5
1.1.3 Options diverses . . . 6
1.1.4 Controlling color . . . 10
1.1.5 Options à ne pas utiliser . . . 10
1.2 Variables d'environnement . . . 10
1.2.1 Variables en mode débogage . . . 17

2 Utilisation de Python sur les plateformes Unix **19**
2.1 Récupérer et installer la dernière version de Python . . . 19
2.1.1 Sur Linux . . . 19
2.1.2 Sur FreeBSD et OpenBSD . . . 20
2.2 Compiler Python . . . 20
2.3 Fichiers et chemins liés à Python . . . 20
2.4 Divers . . . 21
2.5 Version personnalisée d'OpenSSL . . . 21

3 Configurer Python **23**
3.1 Build Requirements . . . 23
3.2 Generated files . . . 23
3.2.1 configure script . . . 24
3.3 Options de configuration . . . 24
3.3.1 Options générales . . . 24
3.3.2 C compiler options . . . 27
3.3.3 Linker options . . . 27
3.3.4 Options for third-party dependencies . . . 27
3.3.5 Options de WebAssembly . . . 28
3.3.6 Options d'installation . . . 29
3.3.7 Options de performance . . . 29
3.3.8 Compilation de Python en mode débogage . . . 31
3.3.9 Debug options . . . 31
3.3.10 Linker options . . . 32
3.3.11 Libraries options . . . 32
3.3.12 Security Options . . . 33
3.3.13 macOS Options . . . 34
3.3.14 iOS Options . . . 35
3.3.15 Cross Compiling Options . . . 35
3.4 Python Build System . . . 36
3.4.1 Main files of the build system . . . 36

3.4.2 Main build steps . . . 36
3.4.3 Main Makefile targets . . . 36
3.4.4 C extensions . . . 37
3.5 Compiler and linker flags . . . 38
3.5.1 Preprocessor flags . . . 38
3.5.2 Compiler flags . . . 38
3.5.3 Linker flags . . . 40

4 Utilisation de Python sur Windows **43**
4.1 L'installateur complet . . . 43
4.1.1 Étapes d'installation . . . 43
4.1.2 Suppression de la limitation `MAX_PATH` . . . 44
4.1.3 Installation sans l'interface utilisateur . . . 45
4.1.4 Installation sans téléchargement . . . 47
4.1.5 Modification d'une installation . . . 47
4.1.6 Installing Free-threaded Binaries . . . 48
4.2 Le paquet Microsoft Store . . . 48
4.2.1 Problèmes connus . . . 49
4.3 Les paquets *nuget.org* . . . 50
4.3.1 Free-threaded packages . . . 50
4.4 Le paquet intégrable . . . 51
4.4.1 Application Python . . . 51
4.4.2 Embarquer Python . . . 51
4.5 Paquets alternatifs . . . 52
4.6 Configurer Python . . . 52
4.6.1 Digression : définition des variables d'environnement . . . 52
4.6.2 Trouver l'exécutable Python . . . 53
4.7 Mode UTF-8 . . . 53
4.8 Lanceur Python pour Windows . . . 54
4.8.1 Pour commencer . . . 54
4.8.2 Lignes Shebang . . . 56
4.8.3 Arguments dans les lignes *shebang* . . . 57
4.8.4 Personnalisation . . . 57
4.8.5 Diagnostics . . . 58
4.8.6 Exécution à vide . . . 58
4.8.7 Installation à la demande . . . 58
4.8.8 Codes de retour . . . 58
4.9 Recherche de modules . . . 59
4.10 Modules supplémentaires . . . 60
4.10.1 PyWin32 . . . 60
4.10.2 cx_Freeze . . . 60
4.11 Compiler Python sous Windows . . . 60
4.12 Autres plateformes . . . 61

5 Using Python on macOS **63**
5.1 Using Python for macOS from `python.org` . . . 63
5.1.1 Étapes d'installation . . . 63
5.1.2 Comment exécuter un script Python . . . 71
5.2 Alternative Distributions . . . 71
5.3 Installation de paquets Python additionnels . . . 71
5.4 GUI Programming . . . 71
5.5 Sujets avancés . . . 72
5.5.1 Installing Free-threaded Binaries . . . 72
5.5.2 Installing using the command line . . . 73
5.5.3 Distributing Python Applications . . . 75
5.5.4 App Store Compliance . . . 75
5.6 Autres ressources . . . 75

6 Using Python on Android **77**
6.1 Adding Python to an Android app . . . 77

7 Using Python on iOS **79**
7.1 Python at runtime on iOS . . . 79
7.1.1 iOS version compatibility . . . 79
7.1.2 Platform identification . . . 79
7.1.3 Standard library availability . . . 80
7.1.4 Binary extension modules . . . 80
7.1.5 Compiler stub binaries . . . 80
7.2 Installing Python on iOS . . . 81
7.2.1 Tools for building iOS apps . . . 81
7.2.2 Adding Python to an iOS project . . . 81
7.2.3 Testing a Python package . . . 83
7.3 App Store Compliance . . . 84

8 Éditeurs et IDEs **85**
8.1 IDLE --- Python editor and shell . . . 85
8.2 Other Editors and IDEs . . . 85

A Glossaire **87**

B About this documentation **105**
B.1 Contributors to the Python documentation . . . 105

C Histoire et licence **107**
C.1 Histoire du logiciel . . . 107
C.2 Conditions générales pour accéder à, ou utiliser, Python . . . 108
C.2.1 PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2 . . . 108
C.2.2 LICENCE D'UTILISATION BEOPEN.COM POUR PYTHON 2.0 . . . 109
C.2.3 LICENCE D'UTILISATION CNRI POUR PYTHON 1.6.1 . . . 109
C.2.4 LICENCE D'UTILISATION CWI POUR PYTHON 0.9.0 à 1.2 . . . 111
C.2.5 ZERO-CLAUSE BSD LICENSE FOR CODE IN THE PYTHON DOCUMENTATION . 111
C.3 Licences et remerciements pour les logiciels tiers . . . 111
C.3.1 Mersenne twister . . . 111
C.3.2 Interfaces de connexion (*sockets*) . . . 112
C.3.3 Interfaces de connexion asynchrones . . . 113
C.3.4 Gestion de témoin (*cookie*) . . . 113
C.3.5 Traçage d'exécution . . . 114
C.3.6 Les fonctions UUencode et UUdecode . . . 114
C.3.7 Appel de procédures distantes en XML (*RPC*, pour *Remote Procedure Call*) . . . 115
C.3.8 test_epoll . . . 115
C.3.9 Select kqueue . . . 116
C.3.10 SipHash24 . . . 116
C.3.11 *strtod* et *dtoa* . . . 117
C.3.12 OpenSSL . . . 117
C.3.13 expat . . . 120
C.3.14 libffi . . . 121
C.3.15 zlib . . . 121
C.3.16 cfuhash . . . 122
C.3.17 libmpdec . . . 123
C.3.18 Ensemble de tests C14N du W3C . . . 123
C.3.19 mimalloc . . . 124
C.3.20 *asyncio* . . . 124
C.3.21 Global Unbounded Sequences (GUS) . . . 125

D Copyright **127**

Index **129**

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.

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 [-bBdEhiIOPqRsSuvVWx?] [-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 :

- When called with standard input connected to a tty device, it prompts for commands and executes them until an EOF (an end-of-file character, you can produce that with `Ctrl-D` on UNIX or `Ctrl-Z, Enter` on Windows) is read. For more on interactive mode, see tut-interac.
- 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).

Lève un évènement d'audit `cpython.run_command` avec comme argument `command`.

-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`.

`-I` option can be used to run the script in isolated mode where `sys.path` contains neither the current directory nor the user's site-packages directory. All `PYTHON*` environment variables are ignored, too.

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 -m timeit -s "setup here" "benchmarked code here"
python -m timeit -h # for details
```

Lève un évènement d'audit `cpython.run_command` avec comme argument `module-name`.

> **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`.

Lève un évènement d'audit `cpython.run_stdin` sans argument.

<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__`.

`-I` option can be used to run the script in isolated mode where `sys.path` contains neither the script's directory nor the user's site-packages directory. All `PYTHON*` environment variables are ignored, too.

Lève un évènement d'audit `cpython.run_file` avec comme argument `filename`.

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 et des variables d'environnement correspondantes, puis quitte.

--help-env

Affiche une brève description des variables d'environnement spécifiques à Python et quitte.

Ajouté dans la version 3.11.

--help-xoptions

Affiche une description spécifique à l'implémentation des options `-X` et quitte.

Ajouté dans la version 3.11.

--help-all

Affiche toutes les informations d'utilisation et quitte.

Ajouté dans la version 3.11.

-V

--version

Affiche la version de Python et termine. Par exemple :

```
Python 3.8.0b2+
```

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

```
Python 3.8.0b2+ (3.8:0c076caaa8, Apr 20 2019, 21:55:00)
[GCC 6.2.0 20161005]
```

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

1.1.3 Options diverses

-b

Issue a warning when converting `bytes` or `bytearray` to `str` without specifying encoding or comparing `bytes` or `bytearray` with `str` or `bytes` with `int`. Issue an error when the option is given twice (`-bb`).

Modifié dans la version 3.5 : Affects also comparisons of `bytes` with `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*.

--check-hash-based-pycs `default|always|never`

Contrôle la façon dont sont validés les fichiers `.pyc` avec empreinte (voir pyc-invalidation). Quand la valeur est `default`, les caches de fichiers de code intermédiaire sont validés en fonction de leur sémantique par défaut. Quand la valeur est `always`, tous les fichiers `.pyc`, qu'ils soient vérifiés ou non, sont validés par rapport à leurs fichiers sources correspondants. Quand la valeur est `never`, les fichiers `.pyc` ne sont pas validés par rapport à leurs fichiers sources correspondants.

La sémantique des fichiers `.pyc` générés en fonction de l'horodatage n'est pas affectée par cette option.

-d

Turn on parser debugging output (for expert only). See also the *PYTHONDEBUG* environment variable.

This option requires a *debug build of Python*, otherwise it's ignored.

-E

Ignore all `PYTHON*` environment variables, e.g. *PYTHONPATH* and *PYTHONHOME*, that might be set.

Voir aussi les options *-P* et *-I* (mode isolé).

-i

Enter interactive mode after execution.

Using the *-i* option will enter interactive mode in any of the following circumstances :
— When a script is passed as first argument
— When the *-c* option is used
— When the *-m* option is used
Interactive mode will start even when `sys.stdin` does not appear to be a terminal. The *PYTHONSTARTUP* file is not read.

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 les options *-E*, *-P* et *-s*.

In isolated mode `sys.path` contains neither the script's directory nor the user's site-packages directory. All `PYTHON*` environment variables are ignored, too. Further restrictions may be imposed to prevent the user from injecting malicious code.

Ajouté 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**.

-P

Ne pas ajouter de chemin (qui serait alors prioritaire) potentiellement non sûr à `sys.path` :
— `python -m module` en ligne de commande : ne pas ajouter le répertoire de travail courant.
— `python script.py` en ligne de commande : ne pas ajouter le répertoire du script. Si c'est un lien symbolique, résoudre les liens symboliques.
— `python -c code` et `python` (REPL) en lignes de commande : ne pas ajouter de chaîne vide, ce qui voudrait dire le répertoire de travail courant.
Voir aussi la variable d'environnement `PYTHONSAFEPATH` ainsi que les options `-E` et `-I` (mode isolé).

Ajouté dans la version 3.11.

-q

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

Ajouté dans la version 3.2.

-R

Active l'imprévisibilité du hachage. Cette option ne produit un effet que si la variable d'environnement `PYTHONHASHSEED` est mise à `0`, puisque l'imprévisibilité du hachage est activée par défaut.

On previous versions of Python, this option turns on hash randomization, so that the `__hash__()` values of str and bytes objects are "salted" with an unpredictable random value. Although they remain constant within an individual Python process, they are not predictable between repeated invocations of Python.

Hash randomization is intended to provide protection against a denial-of-service caused by carefully chosen inputs that exploit the worst case performance of a dict construction, $O(n^2)$ complexity. See http://ocert.org/advisories/ocert-2011-003.html for details.

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

Ajouté dans la version 3.2.3.

Modifié dans la version 3.7 : Cette option n'est plus ignorée.

-s

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

See also `PYTHONNOUSERSITE`.

> **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

Force les flux de sortie et d'erreur standards à ne pas utiliser de tampon. Cette option n'a pas d'effet sur le flux d'entrée standard.

Voir aussi *PYTHONUNBUFFERED*.

Modifié dans la version 3.7 : La couche texte des flux de sortie et d'erreur standards n'utilise maintenant plus de tampon.

-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.

Modifié dans la version 3.10 : Le module `site` affiche les emplacements de recherche de modules spécifiques à l'installation ainsi que les fichiers `.pth` qui sont lus.

Voir aussi *PYTHONVERBOSE*.

-W `arg`

Contrôle des avertissements. Le mécanisme d'avertissement de Python, par défaut, affiche les messages d'avertissement sur `sys.stderr`.

Les configurations les plus simples forcent l'application de l'action à tous les avertissements émis par un processus (même ceux qui auraient été ignorés par défaut) :

```
-Wdefault  # Warn once per call location
-Werror    # Convert to exceptions
-Walways   # Warn every time
-Wall      # Same as -Walways
-Wmodule   # Warn once per calling module
-Wonce     # Warn once per Python process
-Wignore   # Never warn
```

Les noms des actions peuvent être abrégés à votre convenance, l'interpréteur fait la résolution vers le nom adéquat. Par exemple, `-Wi` équivaut à `-Wignore`.

La forme développée de l'argument de `-W` est :

```
action:message:category:module:lineno
```

Les champs *message*, *category*, *module*, *line* appliquent des filtres de sélection. Ils sont facultatifs : on peut les laisser vides, ou en omettre certains à la fin. Par exemple, `-W ignore::DeprecationWarning` permet d'ignorer tous les avertissements de type `DeprecationWarning`.

Le champ *action* déjà décrit s'applique donc uniquement aux avertissements qui sont passés à travers les filtres.

Un filtre *message* restreint l'action aux avertissements dont le texte complet correspond à *message* (la comparaison ne prend pas en compte la casse).

Un filtre *category* restreint l'action aux avertissements dont la classe est celle nommée *category*, ou bien une classe fille de la classe nommée *category*. Un exemple pour *category* est `DeprecationWarning`.

The *module* field matches the (fully qualified) module name ; this match is case-sensitive.

Enfin, un filtre *lineno* restreint l'action aux avertissements qui proviennent d'un numéro de ligne donné. La valeur zéro revient à appliquer le filtre sur toutes les lignes, c.-à-d. l'équivalent de ne pas mettre de numéro.

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 en utilisant la variable d'environnement *PYTHONWARNINGS* et depuis un programme Python en utilisant le module `warnings`. Par exemple, la fonction `warnings.filterwarnings()` peut être utilisée pour filtrer les messages d'avertissement en utilisant une expression rationnelle.

Voir warning-filter et describing-warning-filters pour plus de détails.

-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` to enable `faulthandler`. See also *PYTHONFAULTHANDLER*.

Ajouté dans la version 3.3.

— `-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*.

Ajouté dans la version 3.4.

— `-X tracemalloc` to start tracing Python memory allocations using the `tracemalloc` module. By default, only the most recent frame is stored in a traceback of a trace. Use `-X tracemalloc=NFRAME` to start tracing with a traceback limit of *NFRAME* frames. See `tracemalloc.start()` and *PYTHONTRACEMALLOC* for more information.

Ajouté dans la version 3.4.

— `-X int_max_str_digits` configure la limitation de la longueur de conversion des chaînes entières. Voir aussi *PYTHONINTMAXSTRDIGITS*.

Ajouté dans la version 3.11.

— `-X importtime` affiche le temps mis pour chaque importation. Le temps total par module est affiché (y compris le temps pris par les importations imbriquées) ainsi que le temps propre du module (c.-à-d. sans les importations imbriquées). Notez que l'affichage peut être perturbé dans une application avec de multiples fils d'exécution. L'utilisation classique est `python3 -X importtime -c 'import asyncio'`. Voir aussi *PYTHONPROFILEIMPORTTIME*.

Ajouté dans la version 3.7.

— `-X dev` : enable Python Development Mode, introducing additional runtime checks that are too expensive to be enabled by default. See also *PYTHONDEVMODE*.

Ajouté dans la version 3.7.

— `-X utf8` enables the Python UTF-8 Mode. `-X utf8=0` explicitly disables Python UTF-8 Mode (even when it would otherwise activate automatically). See also *PYTHONUTF8*.

Ajouté dans la version 3.7.

— `-X pycache_prefix=PATH` place l'arborescence des fichiers `.pyc` à partir du chemin donné au lieu de l'arborescence du code source. Voir aussi *PYTHONPYCACHEPREFIX*.

Ajouté dans la version 3.8.

— `-X warn_default_encoding` pour émettre un `EncodingWarning` lorsqu'un fichier est ouvert avec l'encodage par défaut des paramètres régionaux. Voir aussi *PYTHONWARNDEFAULTENCODING*.

Ajouté dans la version 3.10.

— `-X no_debug_ranges` désactive l'inclusion des tableaux qui font la correspondance avec des informations extérieures (ligne de fin, numéros des colonnes de début et de fin) pour toutes les instructions du code. C'est utile quand vous souhaitez diminuer la taille du code objet et des fichiers pyc ou alors quand vous voulez supprimer des informations de position quand l'interpréteur affiche les traces d'appels. Regardez aussi *PYTHONNODEBUGRANGES*.

Ajouté dans la version 3.11.

— `-X frozen_modules` determines whether or not frozen modules are ignored by the import machinery. A value of `on` means they get imported and `off` means they are ignored. The default is `on` if this is an installed Python (the normal case). If it's under development (running from the source tree) then the default is `off`. Note that the `importlib_bootstrap` and `importlib_bootstrap_external` frozen modules are always used, even if this flag is set to `off`. See also *PYTHON_FROZEN_MODULES*.

Ajouté dans la version 3.11.

— `-X perf` enables support for the Linux `perf` profiler. When this option is provided, the `perf` profiler will be able to report Python calls. This option is only available on some platforms and will do nothing if is not supported on the current system. The default value is "off". See also *PYTHONPERFSUPPORT* and perf_profiling.

Ajouté dans la version 3.12.

— `-X perf_jit` enables support for the Linux `perf` profiler with DWARF support. When this option is provided, the `perf` profiler will be able to report Python calls using DWARF information. This option is only available on some platforms and will do nothing if is not supported on the current system. The default value is "off". See also *PYTHON_PERF_JIT_SUPPORT* and perf_profiling.

Ajouté dans la version 3.13.

— `-X cpu_count=n` overrides `os.cpu_count()`, `os.process_cpu_count()`, and `multiprocessing.cpu_count()`. *n* must be greater than or equal to 1. This option may be useful for users who need to limit CPU resources of a container system. See also *PYTHON_CPU_COUNT*. If *n* is `default`, nothing is overridden.

Ajouté dans la version 3.13.

— `-X presite=package.module` specifies a module that should be imported before the `site` module is executed and before the `__main__` module exists. Therefore, the imported module isn't `__main__`. This can be used to execute code early during Python initialization. Python needs to be *built in debug mode* for this option to exist. See also *PYTHON_PRESITE*.

Ajouté dans la version 3.13.

— `-X gil=0,1` forces the GIL to be disabled or enabled, respectively. Setting to `0` is only available in builds configured with *--disable-gil*. See also *PYTHON_GIL* and whatsnew313-free-threaded-cpython.

Ajouté dans la version 3.13.

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

Ajouté dans la version 3.2.

Modifié dans la version 3.9 : Removed the `-X showalloccount` option.

Modifié dans la version 3.10 : Removed the `-X oldparser` option.

1.1.4 Controlling color

The Python interpreter is configured by default to use colors to highlight output in certain situations such as when displaying tracebacks. This behavior can be controlled by setting different environment variables.

Setting the environment variable `TERM` to `dumb` will disable color.

If the `FORCE_COLOR` environment variable is set, then color will be enabled regardless of the value of TERM. This is useful on CI systems which aren't terminals but can still display ANSI escape sequences.

If the `NO_COLOR` environment variable is set, Python will disable all color in the output. This takes precedence over `FORCE_COLOR`.

All these environment variables are used also by other tools to control color output. To control the color output only in the Python interpreter, the *PYTHON_COLORS* environment variable can be used. This variable takes precedence over `NO_COLOR`, which in turn takes precedence over `FORCE_COLOR`.

1.1.5 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.

PYTHONSAFEPATH

Si elle est définie à une chaîne non vide, ne pas ajouter un chemin potentiellement non sûr à sys.path : voir l'option *-P* pour les détails.

Ajouté dans la version 3.11.

PYTHONPLATLIBDIR

Si elle est définie à une chaîne non vide, elle remplace la valeur de sys.platlibdir.

Ajouté dans la version 3.9.

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.

Lève un évènement d'audit cpython.run_startup avec le nom du fichier en argument lorsqu'il est lancé au démarrage.

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.

PYTHONBREAKPOINT

Si elle est définie, elle fait référence à un appelable en utilisant la notation des chemins délimités par des points. Le module contenant l'appelable est importé et l'appelable est alors lancé par l'implémentation par défaut de sys.breakpointhook(), elle-même étant appelée par la fonction native breakpoint(). Si elle n'est pas définie ou définie par une chaîne vide, elle vaut la même chose que pdb.set_trace. La définir à la chaîne "0" entraine que l'implémentation par défaut de sys.breakpointhook() ne fait rien et s'interrompt immédiatement.

Ajouté dans la version 3.7.

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.

This environment variable requires a *debug build of Python*, otherwise it's ignored.

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.

Lève un évènement d'audit `cpython.run_stdin` sans argument.

Modifié dans la version 3.12.5 : (also 3.11.10, 3.10.15, 3.9.20, and 3.8.20) Emits audit events.

Modifié dans la version 3.13 : Uses PyREPL if possible, in which case *PYTHONSTARTUP* is also executed. Emits audit events.

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 macOS.

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*.

PYTHONPYCACHEPREFIX

Si elle est définie, Python n'écrit pas ses fichiers `.pyc` dans `__pycache__` mais dans une arborescence miroir à ce chemin. C'est l'équivalent de l'option *-X* `pycache_prefix=PATH`.

Ajouté dans la version 3.8.

PYTHONHASHSEED

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

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.

Ajouté dans la version 3.2.3.

PYTHONINTMAXSTRDIGITS

Si un entier est assigné à cette variable, elle est utilisée pour configurer la limite globale de l'interpréteur limitation de la longueur de conversion des chaînes de caractères entiers.

Ajouté dans la version 3.11.

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

Defines the `user base directory`, which is used to compute the path of the `user site-packages directory` and installation paths for `python -m pip 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 macOS.

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. Dans ce cas, les filtres spécifiés en derniers prennent le pas sur ceux qui les précèdent dans la liste.

Les configurations les plus simples forcent l'application de l'action à tous les avertissements émis par un processus (même ceux qui auraient été ignorés par défaut) :

```
PYTHONWARNINGS=default  # Warn once per call location
PYTHONWARNINGS=error    # Convert to exceptions
PYTHONWARNINGS=always   # Warn every time
PYTHONWARNINGS=all      # Same as PYTHONWARNINGS=always
PYTHONWARNINGS=module   # Warn once per calling module
PYTHONWARNINGS=once     # Warn once per Python process
PYTHONWARNINGS=ignore   # Never warn
```

Voir warning-filter et describing-warning-filters pour plus de détails.

PYTHONFAULTHANDLER

If this environment variable is set to a non-empty string, `faulthandler.enable()` is called at startup : install a handler for `SIGSEGV`, `SIGFPE`, `SIGABRT`, `SIGBUS` and `SIGILL` signals to dump the Python traceback. This is equivalent to `-X faulthandler` option.

Ajouté dans la version 3.3.

PYTHONTRACEMALLOC

If this environment variable is set to a non-empty string, start tracing Python memory allocations using the `tracemalloc` module. The value of the variable is the maximum number of frames stored in a traceback of a trace. For example, `PYTHONTRACEMALLOC=1` stores only the most recent frame. See the `tracemalloc.start()` function for more information. This is equivalent to setting the `-X tracemalloc` option.

Ajouté dans la version 3.4.

PYTHONPROFILEIMPORTTIME

If this environment variable is set to a non-empty string, Python will show how long each import takes. This

is equivalent to setting the `-X importtime` option.

Ajouté dans la version 3.7.

PYTHONASYNCIODEBUG

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

Ajouté 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 :
— `default` : utilise les allocateurs de mémoire par défaut.
— `malloc` : use the `malloc()` function of the C library for all domains (`PYMEM_DOMAIN_RAW`, `PYMEM_DOMAIN_MEM`, `PYMEM_DOMAIN_OBJ`).
— `pymalloc` : use the pymalloc allocator for `PYMEM_DOMAIN_MEM` and `PYMEM_DOMAIN_OBJ` domains and use the `malloc()` function for the `PYMEM_DOMAIN_RAW` domain.
— `mimalloc` : use the mimalloc allocator for `PYMEM_DOMAIN_MEM` and `PYMEM_DOMAIN_OBJ` domains and use the `malloc()` function for the `PYMEM_DOMAIN_RAW` domain.
Chacun de ces modes possède un pendant qui ajoute en plus des points d'entrée de débogage :
— `debug` : installe des points d'entrée de débogage au-dessus des allocateurs de mémoire par défaut.
— `malloc_debug` : identique à `malloc` mais installe aussi des points d'entrée de débogage.
— `pymalloc_debug` pour `pymalloc`.
— `mimalloc_debug` : same as `mimalloc` but also install debug hooks.
Ajouté dans la version 3.6.

Modifié dans la version 3.7 : L'allocateur `default` a été ajouté.

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 utilisé pour interagir avec le système de fichiers et le gestionnaire d'erreurs associé* reviennent à leurs valeurs par défaut pré-3.6, respectivement `'mbcs'` et `'replace'`. Sinon, elles prennent les nouvelles valeurs par défaut `"UTF-8"` et `"surrogatepass"`.

This may also be enabled at runtime with `sys._enablelegacywindowsfsencoding()`.

Availability : Windows.

Ajouté 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.

Availability : Windows.

Ajouté dans la version 3.6.

PYTHONCOERCECLOCALE

Si elle est définie à la valeur `0`, l'application en ligne de commande principale Python ne prend pas en compte les configurations régionales C et POSIX à base ASCII, mais bascule sur une alternative basée sur l'UTF-8 qui doit posséder plus de possibilités.

Si cette variable n'est pas définie (ou est définie à une valeur autre que `0`), que la variable d'environnement de configuration régionale `LC_ALL` est également non définie et que la configuration régionale indiquée dans la catégorie `LC_TYPE` est soit la région par défaut `C`, soit la région `POSIX` qui demande explicitement de l'ASCII, alors l'interpréteur en ligne de commande Python essaie de configurer les paramètres régionaux pour la catégorie `LC_TYPE` dans l'ordre suivant avant de charger l'exécutable de l'interpréteur :
— `C.UTF-8`
— `C.utf8`
— `UTF-8`
Si la définition d'une des configurations régionales fonctionne, la variable d'environnement `LC_TYPE` est aussi définie ainsi dans l'environnement du processus avant que l'exécutable Python ne soit initialisé. Ceci assure que, en plus d'être vue par l'interpréteur lui-même et tous les autres composants prenant en charge la configuration régionale dans le même processus (telle que la bibliothèque GNU `readline`), la configuration mise à jour est aussi valable pour les sous-processus (indépendamment du fait qu'ils utilisent un interpréteur Python ou non) ainsi que pour les opérations qui font appel à l'environnement (qui utiliseraient sinon la configuration régionale C courante, tel que c'est le cas pour la propre fonction Python `locale.getdefaultlocale()`).

La configuration d'une de ces variables régionales (soit explicitement, soit *via* le mécanisme décrit ci-dessus) active automatiquement `surrogateescape` pour la gestion des erreurs de `sys.stdin` et `sys.stdout` (`sys.stderr` continue d'utiliser `backslashreplace` comme pour toute autre configuration régionale). Ce comportement relatif à la gestion des flux standards peut être surchargé, comme d'habitude, par *`PYTHONIOENCODING`*.

À fin de débogage, définir `PYTHONCOERCECLOCALE=warn` indique à Python d'émettre les messages d'avertissement sur `stderr` si la configuration régionale est activée, ou si une configuration régionale qui *aurait* activé est toujours active quand l'interpréteur Python est initialisé.

Notez également que même si la contrainte sur la configuration régionale est désactivée, ou si elle ne trouve pas une configuration satisfaisante et échoue, *`PYTHONUTF8`* s'active par défaut avec une configuration régionale par défaut à base ASCII. Ces fonctionnalités doivent être désactivées pour forcer l'interpréteur à utiliser `ASCII` en lieu et place de `UTF-8` pour les interfaces avec le système.

Availability : Unix.

Ajouté dans la version 3.7 : Voir la **PEP 538** pour plus d'informations.

PYTHONDEVMODE

If this environment variable is set to a non-empty string, enable Python Development Mode, introducing additional runtime checks that are too expensive to be enabled by default. This is equivalent to setting the *`-X`* `dev` option.

Ajouté dans la version 3.7.

PYTHONUTF8

La valeur `1` active le mode UTF-8.

La valeur `0` désactive le mode UTF-8.

Définir une valeur autre (non vide) entraine une erreur pendant l'initialisation de l'interpréteur.

Ajouté dans la version 3.7.

PYTHONWARNDEFAULTENCODING

Si elle est définie à une chaîne non vide, un avertissement `EncodingWarning` est émis lorsque l'encodage par défaut de la configuration régionale est utilisé.

Voir io-encoding-warning pour plus de détails.

Ajouté dans la version 3.10.

PYTHONNODEBUGRANGES

Si cette variable est définie, cela désactive l'inclusion des tableaux qui font la correspondance avec des informations extérieures (ligne de fin, numéros des colonnes de début et de fin) pour toutes les instructions du code. C'est utile quand vous souhaitez diminuer la taille du code objet et des fichiers pyc ou alors quand vous voulez supprimer des informations de position quand l'interpréteur affiche les traces d'appels.

Ajouté dans la version 3.11.

PYTHONPERFSUPPORT

If this variable is set to a nonzero value, it enables support for the Linux `perf` profiler so Python calls can be detected by it.

If set to `0`, disable Linux `perf` profiler support.

See also the *`-X perf`* command-line option and perf_profiling.

Ajouté dans la version 3.12.

PYTHON_PERF_JIT_SUPPORT

If this variable is set to a nonzero value, it enables support for the Linux `perf` profiler so Python calls can be detected by it using DWARF information.

If set to `0`, disable Linux `perf` profiler support.

See also the *`-X perf_jit`* command-line option and perf_profiling.

Ajouté dans la version 3.13.

PYTHON_CPU_COUNT

If this variable is set to a positive integer, it overrides the return values of `os.cpu_count()` and `os.process_cpu_count()`.

See also the *`-X cpu_count`* command-line option.

Ajouté dans la version 3.13.

PYTHON_FROZEN_MODULES

If this variable is set to `on` or `off`, it determines whether or not frozen modules are ignored by the import machinery. A value of `on` means they get imported and `off` means they are ignored. The default is `on` for non-debug builds (the normal case) and `off` for debug builds. Note that the `importlib_bootstrap` and `importlib_bootstrap_external` frozen modules are always used, even if this flag is set to `off`.

See also the *`-X frozen_modules`* command-line option.

Ajouté dans la version 3.13.

PYTHON_COLORS

If this variable is set to `1`, the interpreter will colorize various kinds of output. Setting it to `0` deactivates this behavior. See also *Controlling color*.

Ajouté dans la version 3.13.

PYTHON_BASIC_REPL

If this variable is set to any value, the interpreter will not attempt to load the Python-based *REPL* that requires `curses` and `readline`, and will instead use the traditional parser-based *REPL*.

Ajouté dans la version 3.13.

PYTHON_HISTORY

This environment variable can be used to set the location of a `.python_history` file (by default, it is `.python_history` in the user's home directory).

Ajouté dans la version 3.13.

PYTHON_GIL

If this variable is set to `1`, the global interpreter lock (GIL) will be forced on. Setting it to `0` forces the GIL off (needs Python configured with the *`--disable-gil`* build option).

See also the *`-X gil`* command-line option, which takes precedence over this variable, and whatsnew313-free-threaded-cpython.

Ajouté dans la version 3.13.

1.2.1 Variables en mode débogage

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.

Needs Python configured with the `--with-trace-refs` build option.

PYTHONDUMPREFSFILE

If set, Python will dump objects and reference counts still alive after shutting down the interpreter into a file under the path given as the value to this environment variable.

Needs Python configured with the `--with-trace-refs` build option.

Ajouté dans la version 3.11.

PYTHON_PRESITE

If this variable is set to a module, that module will be imported early in the interpreter lifecycle, before the `site` module is executed, and before the `__main__` module is created. Therefore, the imported module is not treated as `__main__`.

This can be used to execute code early during Python initialization.

To import a submodule, use `package.module` as the value, like in an import statement.

See also the `-X presite` command-line option, which takes precedence over this variable.

Needs Python configured with the `--with-pydebug` build option.

Ajouté dans la version 3.13.

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 préinstallé sur la plupart des distributions Linux, et est disponible en paquet sur toutes les autres. Cependant, il y a certaines fonctionnalités que vous pourriez vouloir utiliser qui ne sont pas disponibles dans le paquet de votre distribution. Vous pouvez compiler la dernière version de Python depuis les sources.

Dans le cas où Python n'est pas préinstallé et n'est pas disponible dans les dépôts, vous pouvez 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/fr/package-maintainers/Packaging_Tutorial_2_GNU_Hello/
pour les utilisateurs de Fedora

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

Installing IDLE

In some cases, IDLE might not be included in your Python installation.

— For Debian and Ubuntu users :

```
sudo apt update
sudo apt install idle
```

— For Fedora, RHEL, and CentOS users :

```
sudo dnf install python3-idle
```

— For SUSE and OpenSUSE users :

```
sudo zypper in python3-idle
```

— For Alpine Linux users :

```
sudo apk add python3-idle
```

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
```

2.2 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.)

La compilation s'effectue avec les commandes habituelles :

```
./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* et *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 Version personnalisée d'OpenSSL

1. Pour compiler avec votre propre configuration d'OpenSSL et le magasin des certificats de confiance du système, commencez par trouver le sous-dossier de `/etc` qui contient le fichier ou le lien symbolique `openssl.cnf`. Sur la plupart des distributions, c'est soit `/etc/ssl`, soit `/etc/pki/tls`. Ce dossier doit également contenir un fichier `cert.pem` ou un sous-dossier `certs` (ou les deux).

```
$ find /etc/ -name openssl.cnf -printf "%h\n"
/etc/ssl
```

2. Téléchargez et compilez OpenSSL. Installez-le, en prenant bien garde de le faire avec la cible `install_sw` et non pas `install`. En effet, `install` réécrirait le fichier `openssl.cnf`.

```
$ curl -O https://www.openssl.org/source/openssl-VERSION.tar.gz
$ tar xzf openssl-VERSION
$ pushd openssl-VERSION
$ ./config \
    --prefix=/usr/local/custom-openssl \
    --libdir=lib \
    --openssldir=/etc/ssl
$ make -j1 depend
$ make -j8
$ make install_sw
$ popd
```

3. Ensuite, compilez Python avec votre version d'OpenSSL personnalisée (voir les options `--with-openssl` et `--with-openssl-rpath` du script `configure`).

```
$ pushd python-3.x.x
$ ./configure -C \
    --with-openssl=/usr/local/custom-openssl \
    --with-openssl-rpath=auto \
    --prefix=/usr/local/python-3.x.x
$ make -j8
$ make altinstall
```

Note

Les versions d'OpenSSL dans une même série (incrémentation du troisième chiffre) conservent la compatibilité de l'interface binaire. Pour mettre à jour OpenSSL, il n'y a pas besoin de recompiler Python, il suffit de remplacer l'installation d'OpenSSL avec la nouvelle version.

CHAPITRE 3

Configurer Python

3.1 Build Requirements

Features and minimum versions required to build CPython :

- A C11 compiler. Optional C11 features are not required.
- On Windows, Microsoft Visual Studio 2017 or later is required.
- Support for IEEE 754 floating-point numbers and floating-point Not-a-Number (NaN).
- Support for threads.
- OpenSSL 1.1.1 is the minimum version and OpenSSL 3.0.9 is the recommended minimum version for the `ssl` and `hashlib` extension modules.
- SQLite 3.15.2 for the `sqlite3` extension module.
- Tcl/Tk 8.5.12 for the `tkinter` module.
- Autoconf 2.72 and aclocal 1.16.5 are required to regenerate the `configure` script.

Modifié dans la version 3.1 : Tcl/Tk version 8.3.1 is now required.

Modifié dans la version 3.5 : On Windows, Visual Studio 2015 or later is now required. Tcl/Tk version 8.4 is now required.

Modifié dans la version 3.6 : Selected C99 features are now required, like `<stdint.h>` and `static inline` functions.

Modifié dans la version 3.7 : Thread support and OpenSSL 1.0.2 are now required.

Modifié dans la version 3.10 : OpenSSL 1.1.1 is now required. Require SQLite 3.7.15.

Modifié dans la version 3.11 : C11 compiler, IEEE 754 and NaN support are now required. On Windows, Visual Studio 2017 or later is required. Tcl/Tk version 8.5.12 is now required for the `tkinter` module.

Modifié dans la version 3.13 : Autoconf 2.71, aclocal 1.16.5 and SQLite 3.15.2 are now required.

Modifié dans la version 3.14 : Autoconf 2.72 is now required.

See also **PEP 7** ”Style Guide for C Code” and **PEP 11** ”CPython platform support”.

3.2 Generated files

To reduce build dependencies, Python source code contains multiple generated files. Commands to regenerate all generated files :

```
make regen-all
make regen-stdlib-module-names
make regen-limited-abi
make regen-configure
```

The `Makefile.pre.in` file documents generated files, their inputs, and tools used to regenerate them. Search for `regen-*` make targets.

3.2.1 configure script

The `make regen-configure` command regenerates the `aclocal.m4` file and the `configure` script using the `Tools/build/regen-configure.sh` shell script which uses an Ubuntu container to get the same tools versions and have a reproducible output.

The container is optional, the following command can be run locally :

```
autoreconf -ivf -Werror
```

The generated files can change depending on the exact `autoconf-archive`, `aclocal` and `pkg-config` versions.

3.3 Options de configuration

List all `configure` script options using :

```
./configure --help
```

Voir aussi le fichier `Misc/SpecialBuilds.txt` dans la distribution des sources Python.

3.3.1 Options générales

--enable-loadable-sqlite-extensions

Support loadable extensions in the `_sqlite` extension module (default is no) of the `sqlite3` module.

Voir la méthode `sqlite3.Connection.enable_load_extension()` du module `sqlite3`.

Ajouté dans la version 3.6.

--disable-ipv6

Désactive la prise en charge d'IPv6 (activé par défaut si géré), voir le module `socket`.

--enable-big-digits=`[15|30]`

Définit la taille en bits des chiffres Python `int` : 15 ou 30 bits.

La valeur par défaut est `30` bits.

Définit `PYLONG_BITS_IN_DIGIT` à `15` ou `30`.

Voir `sys.int_info.bits_per_digit`.

--with-suffix=`SUFFIX`

Définit à *SUFFIX* le suffixe de l'exécutable Python.

Le suffixe par défaut est `.exe` sur Windows et macOS (exécutable `python.exe`), `.js` pour Emscripten (`node`), `.html` pour Emscripten (navigateur), `.wasm` pour WASI, et une chaîne vide sur les autres plateformes (exécutable `python`).

Modifié dans la version 3.11 : les suffixes par défaut pour la plateforme WASM sont maintenant `.js`, `.html` et `.wasm`.

--with-tzpath=`<list of absolute paths separated by pathsep>`

Select the default time zone search path for `zoneinfo.TZPATH`. See the Compile-time configuration of the `zoneinfo` module.

Par défaut : `/usr/share/zoneinfo:/usr/lib/zoneinfo:/usr/share/lib/zoneinfo:/etc/zoneinfo`.

Voir le séparateur de chemins `os.pathsep`.

Ajouté dans la version 3.9.

--without-decimal-contextvar

Compile le module d'extension `_decimal` en utilisant un contexte local au fil d'exécution plutôt qu'un contexte local de coroutine (défaut). Voir le module `decimal`.

See `decimal.HAVE_CONTEXTVAR` and the `contextvars` module.

Ajouté dans la version 3.9.

--with-dbmliborder=<list of backend names>

Change l'ordre de détection des implémentations de base de données pour le module `dbm`.

Une valeur valide est une chaîne de noms d'implémentations séparés par des deux-points (`:`) :
— `ndbm`;
— `gdbm`;
— `bdb`.

--without-c-locale-coercion

Désactive le forçage des paramètres régionaux C pour un paramètre régional basé sur UTF-8 (activé par défaut).

Ne pas définir la macro `PY_COERCE_C_LOCALE`.

Voir *`PYTHONCOERCECLOCALE`* et la **PEP 538**.

--with-platlibdir=DIRNAME

Nom du dossier de bibliothèques Python (par défaut : `lib`).

Fedora et SuSE utilisent `lib64` sur les systèmes 64-bit.

Voir `sys.platlibdir`.

Ajouté dans la version 3.9.

--with-wheel-pkg-dir=PATH

Dossier de paquets *wheel* utilisé par le module `ensurepip` (par défaut : aucun).

Some Linux distribution packaging policies recommend against bundling dependencies. For example, Fedora installs wheel packages in the `/usr/share/python-wheels/` directory and don't install the `ensurepip._bundled` package.

Ajouté dans la version 3.10.

--with-pkg-config=[check|yes|no]

Est-ce que *configure* doit utiliser **pkg-config** pour détecter les dépendances de construction.
— `check` (par défaut) : **pkg-config** est optionnel
— `yes` : **pkg-config** est obligatoire
— `no` : *configure* n'utilise pas **pkg-config** même s'il est présent
Ajouté dans la version 3.11.

--enable-pystats

Turn on internal Python performance statistics gathering.

By default, statistics gathering is off. Use `python3 -X pystats` command or set `PYTHONSTATS=1` environment variable to turn on statistics gathering at Python startup.

At Python exit, dump statistics if statistics gathering was on and not cleared.

Effects :
— Add *`-X pystats`* command line option.
— Add `PYTHONSTATS` environment variable.
— Define the `Py_STATS` macro.
— Add functions to the `sys` module :

  - `sys._stats_on()` : Turns on statistics gathering.
  - `sys._stats_off()` : Turns off statistics gathering.
  - `sys._stats_clear()` : Clears the statistics.
  - `sys._stats_dump()` : Dump statistics to file, and clears the statistics.

The statistics will be dumped to a arbitrary (probably unique) file in `/tmp/py_stats/` (Unix) or `C:\temp\py_stats\` (Windows). If that directory does not exist, results will be printed on stderr.

Utilisez `Tools/scripts/summarize_stats.py` pour lire les statistiques.

Statistics :

- Opcode :
  - Specialization : success, failure, hit, deferred, miss, deopt, failures ;
  - Execution count ;
  - Pair count.
- Call :
  - Inlined Python calls ;
  - PyEval calls ;
  - Frames pushed ;
  - Frame object created ;
  - Eval calls : vector, generator, legacy, function VECTORCALL, build class, slot, function "ex", API, method.
- Object :
  - incref and decref ;
  - interpreter incref and decref ;
  - allocations : all, 512 bytes, 4 kiB, big ;
  - free ;
  - to/from free lists ;
  - dictionary materialized/dematerialized ;
  - type cache ;
  - optimization attempts ;
  - optimization traces created/executed ;
  - uops executed.
- Garbage collector :
  - Garbage collections ;
  - Objects visited ;
  - Objects collected.

Ajouté dans la version 3.11.

`--disable-gil`

Enables **experimental** support for running Python without the *global interpreter lock* (GIL) : free threading build.

Defines the `Py_GIL_DISABLED` macro and adds `"t"` to `sys.abiflags`.

See whatsnew313-free-threaded-cpython for more detail.

Ajouté dans la version 3.13.

`--enable-experimental-jit``=[no|yes|yes-off|interpreter]`

Indicate how to integrate the JIT compiler.

- `no` - build the interpreter without the JIT.
- `yes` - build the interpreter with the JIT.
- `yes-off` - build the interpreter with the JIT but disable it by default.
- `interpreter` - build the interpreter without the JIT, but with the tier 2 enabled interpreter.

By convention, `--enable-experimental-jit` is a shorthand for `--enable-experimental-jit=yes`.

> **Note**
>
> When building CPython with JIT enabled, ensure that your system has Python 3.11 or later installed.

Ajouté dans la version 3.13.

PKG_CONFIG

Path to `pkg-config` utility.

PKG_CONFIG_LIBDIR

PKG_CONFIG_PATH

`pkg-config` options.

3.3.2 C compiler options

CC

C compiler command.

CFLAGS

C compiler flags.

CPP

C preprocessor command.

CPPFLAGS

C preprocessor flags, e.g. `-I`*`include_dir`*.

3.3.3 Linker options

LDFLAGS

Linker flags, e.g. `-L`*`library_directory`*.

LIBS

Libraries to pass to the linker, e.g. `-l`*`library`*.

MACHDEP

Name for machine-dependent library files.

3.3.4 Options for third-party dependencies

Ajouté dans la version 3.11.

BZIP2_CFLAGS

BZIP2_LIBS

C compiler and linker flags to link Python to `libbz2`, used by `bz2` module, overriding `pkg-config`.

CURSES_CFLAGS

CURSES_LIBS

C compiler and linker flags for `libncurses` or `libncursesw`, used by `curses` module, overriding `pkg-config`.

GDBM_CFLAGS

GDBM_LIBS

C compiler and linker flags for `gdbm`.

LIBB2_CFLAGS

LIBB2_LIBS

C compiler and linker flags for `libb2` (BLAKE2), used by `hashlib` module, overriding `pkg-config`.

LIBEDIT_CFLAGS

LIBEDIT_LIBS

C compiler and linker flags for `libedit`, used by `readline` module, overriding `pkg-config`.

LIBFFI_CFLAGS

LIBFFI_LIBS

C compiler and linker flags for `libffi`, used by `ctypes` module, overriding `pkg-config`.

LIBMPDEC_CFLAGS

LIBMPDEC_LIBS

C compiler and linker flags for `libmpdec`, used by `decimal` module, overriding `pkg-config`.

> **Note**
>
> These environment variables have no effect unless `--with-system-libmpdec` is specified.

LIBLZMA_CFLAGS

LIBLZMA_LIBS

C compiler and linker flags for `liblzma`, used by `lzma` module, overriding `pkg-config`.

LIBREADLINE_CFLAGS

LIBREADLINE_LIBS

C compiler and linker flags for `libreadline`, used by `readline` module, overriding `pkg-config`.

LIBSQLITE3_CFLAGS

LIBSQLITE3_LIBS

C compiler and linker flags for `libsqlite3`, used by `sqlite3` module, overriding `pkg-config`.

LIBUUID_CFLAGS

LIBUUID_LIBS

C compiler and linker flags for `libuuid`, used by `uuid` module, overriding `pkg-config`.

PANEL_CFLAGS

PANEL_LIBS

C compiler and linker flags for PANEL, overriding `pkg-config`.

C compiler and linker flags for `libpanel` or `libpanelw`, used by `curses.panel` module, overriding `pkg-config`.

TCLTK_CFLAGS

TCLTK_LIBS

C compiler and linker flags for TCLTK, overriding `pkg-config`.

ZLIB_CFLAGS

ZLIB_LIBS

C compiler and linker flags for `libzlib`, used by `gzip` module, overriding `pkg-config`.

3.3.5 Options de WebAssembly

--enable-wasm-dynamic-linking

Active la gestion d'édition de liens dynamique pour WASM

L'édition de liens dynamique autorise `dlopen`. La taille du fichier exécutable augmente en raison de l'élimination moindre du code mort et des fonctionnalités supplémentaires.

Ajouté dans la version 3.11.

--enable-wasm-pthreads

Active la gestion des *pthreads* pour WASM.

Ajouté dans la version 3.11.

3.3.6 Options d'installation

--prefix=PREFIX

Install architecture-independent files in PREFIX. On Unix, it defaults to `/usr/local`.

This value can be retrieved at runtime using `sys.prefix`.

As an example, one can use `--prefix="$HOME/.local/"` to install a Python in its home directory.

--exec-prefix=EPREFIX

Install architecture-dependent files in EPREFIX, defaults to *--prefix*.

This value can be retrieved at runtime using `sys.exec_prefix`.

--disable-test-modules

Don't build nor install test modules, like the `test` package or the `_testcapi` extension module (built and installed by default).

Ajouté dans la version 3.10.

--with-ensurepip=[upgrade|install|no]

Sélectionne la commande `ensurepip` exécutée à l'installation de Python :
— `upgrade` (défaut) : exécute la commande `python -m ensurepip --altinstall --upgrade`;
— `install` : exécute la commande `python -m ensurepip --altinstall`;
— `no` : n'exécute pas *ensurepip*.
Ajouté dans la version 3.6.

3.3.7 Options de performance

Configuring Python using `--enable-optimizations --with-lto` (PGO + LTO) is recommended for best performance. The experimental `--enable-bolt` flag can also be used to improve performance.

--enable-optimizations

Active l'optimisation guidée par profilage (*Profile Guided Optimization*, PGO) en utilisant *PROFILE_TASK* (désactivé par défaut).

L'utilisation de PGO avec le compilateur C Clang nécessite `llvm-profdata`. Sur macOS, GCC est juste un alias vers Clang et partage donc cette contrainte.

Désactive également l'interposition sémantique dans *libpython* si GCC est utilisé avec `--enable-shared` : ajoute l'option `-fno-semantic-interposition` à la compilation et à l'édition des liens.

> **Note**
>
> During the build, you may encounter compiler warnings about profile data not being available for some source files. These warnings are harmless, as only a subset of the code is exercised during profile data acquisition. To disable these warnings on Clang, manually suppress them by adding `-Wno-profile-instr-unprofiled` to *CFLAGS*.

Ajouté dans la version 3.6.

Modifié dans la version 3.10 : Utilisation de `-fno-semantic-interposition` avec GCC.

PROFILE_TASK

Variable d'environnement utilisée dans le `Makefile` : arguments Python utilisés pour la tâche de préparation de la PGO.

Par défaut : `-m test --pgo --timeout=$(TESTTIMEOUT)`.

Ajouté dans la version 3.8.

Modifié dans la version 3.13 : Task failure is no longer ignored silently.

--with-lto=`[full|thin|no|yes]`

Active l'optimisation à l'édition des liens (*Link Time Optimization*, LTO) à la compilation (désactivé par défaut).

Le compilateur C Clang nécessite `llvm-ar` pour la LTO (`ar` sur macOS), ainsi qu'un éditeur de liens implémentant la LTO (`ld.gold` ou `lld`).

Ajouté dans la version 3.6.

Ajouté dans la version 3.11 : Pour avoir la fonctionnalité ThinLTO, utilisez `--with-lto=thin` avec Clang.

Modifié dans la version 3.12 : Use ThinLTO as the default optimization policy on Clang if the compiler accepts the flag.

--enable-bolt

Enable usage of the BOLT post-link binary optimizer (disabled by default).

BOLT is part of the LLVM project but is not always included in their binary distributions. This flag requires that `llvm-bolt` and `merge-fdata` are available.

BOLT is still a fairly new project so this flag should be considered experimental for now. Because this tool operates on machine code its success is dependent on a combination of the build environment + the other optimization configure args + the CPU architecture, and not all combinations are supported. BOLT versions before LLVM 16 are known to crash BOLT under some scenarios. Use of LLVM 16 or newer for BOLT optimization is strongly encouraged.

The `BOLT_INSTRUMENT_FLAGS` and `BOLT_APPLY_FLAGS` **configure** variables can be defined to override the default set of arguments for **llvm-bolt** to instrument and apply BOLT data to binaries, respectively.

Ajouté dans la version 3.12.

BOLT_APPLY_FLAGS

Arguments to `llvm-bolt` when creating a BOLT optimized binary.

Ajouté dans la version 3.12.

BOLT_INSTRUMENT_FLAGS

Arguments to `llvm-bolt` when instrumenting binaries.

Ajouté dans la version 3.12.

--with-computed-gotos

Autorise les *gotos* calculés dans les boucles (activé par défaut pour les compilateurs qui le gèrent).

--with-tail-call-interp

Enable interpreters using tail calls in CPython. If enabled, enabling PGO (*`--enable-optimizations`*) is highly recommended. This option specifically requires a C compiler with proper tail call support, and the preserve_none calling convention. For example, Clang 19 and newer supports this feature.

Ajouté dans la version 3.14.

--without-mimalloc

Disable the fast mimalloc allocator (enabled by default).

Voir aussi la variable d'environnement *`PYTHONMALLOC`*.

--without-pymalloc

Désactive l'allocateur spécialisé de mémoire de Python pymalloc (activé par défaut).

Voir aussi la variable d'environnement *PYTHONMALLOC*.

--without-doc-strings

Désactive les chaînes de documentation statiques pour réduire l'empreinte mémoire (activé par défaut). Les chaînes définies dans Python ne sont pas affectées.

Ne définit pas la macro `WITH_DOC_STRINGS`.

Voir la macro `PyDoc_STRVAR()`.

--enable-profiling

Active le profilage du code C avec `gprof` (désactivé par défaut).

--with-strict-overflow

Add `-fstrict-overflow` to the C compiler flags (by default we add `-fno-strict-overflow` instead).

3.3.8 Compilation de Python en mode débogage

Une compilation de Python en mode débogage se fait avec l'option de configuration *--with-pydebug*.

Effets du mode débogage :

- Affichage de tous les avertissements par défaut : le filtre par défaut des avertissements est vide dans le module `warnings`.
- Ajout de `d` à `sys.abiflags`.
- Add `sys.gettotalrefcount()` function.
- Ajout de l'option de ligne de commande *-X showrefcount*.
- Add *-d* command line option and *PYTHONDEBUG* environment variable to debug the parser.
- Prise en charge de la variable `__ltrace__` : si la variable est définie, active le traçage de bas niveau dans la boucle d'évaluation du code intermédiaire.
- Installation des debug hooks on memory allocators pour détecter les débordements de mémoire tampon et autres erreurs de mémoire.
- Définition des macros `Py_DEBUG` et `Py_REF_DEBUG`.
- Ajout de vérifications à l'exécution : codes entourés de `#ifdef Py_DEBUG` et `#endif`. Active les assertions `assert(...)` et `_PyObject_ASSERT(...)` : ne définit pas la macro `NDEBUG` (voir aussi l'option de configuration *--with-assertions*). Les principales vérifications à l'exécution :
  - Ajout des contrôles d'intégrité sur les arguments de fonction.
  - Les objets `unicode` et `int` sont créés avec un motif particulier à l'initialisation de leur mémoire afin de détecter l'usage d'objets non initialisés.
  - S'assurer que les fonctions qui peuvent effacer ou remplacer l'exception courante ne sont pas appelées avec une exception levée.
  - S'assurer que les fonctions qui désallouent ne changent pas l'exception en cours.
  - Le ramasse-miettes (la fonction `gc.collect()`) effectue quelques tests basiques relatifs à la cohérence des objets.
  - The `Py_SAFE_DOWNCAST()` macro checks for integer underflow and overflow when downcasting from wide types to narrow types.

Voir aussi le mode de développement Python et l'option de configuration *--with-trace-refs*.

Modifié dans la version 3.8 : Release builds and debug builds are now ABI compatible : defining the `Py_DEBUG` macro no longer implies the `Py_TRACE_REFS` macro (see the *--with-trace-refs* option).

3.3.9 Debug options

--with-pydebug

Build Python in debug mode : define the `Py_DEBUG` macro (disabled by default).

--with-trace-refs

Enable tracing references for debugging purpose (disabled by default).

Effects :

— Define the `Py_TRACE_REFS` macro.
— Add `sys.getobjects()` function.
— Add *`PYTHONDUMPREFS`* environment variable.

The *`PYTHONDUMPREFS`* environment variable can be used to dump objects and reference counts still alive at Python exit.

Statically allocated objects are not traced.

Ajouté dans la version 3.8.

Modifié dans la version 3.13 : This build is now ABI compatible with release build and *debug build*.

`--with-assertions`

Build with C assertions enabled (default is no) : `assert(...);` and `_PyObject_ASSERT(...);`.

If set, the `NDEBUG` macro is not defined in the *`OPT`* compiler variable.

See also the *`--with-pydebug`* option (*debug build*) which also enables assertions.

Ajouté dans la version 3.6.

`--with-valgrind`

Enable Valgrind support (default is no).

`--with-dtrace`

Enable DTrace support (default is no).

See Instrumenting CPython with DTrace and SystemTap.

Ajouté dans la version 3.6.

`--with-address-sanitizer`

Enable AddressSanitizer memory error detector, `asan` (default is no).

Ajouté dans la version 3.6.

`--with-memory-sanitizer`

Enable MemorySanitizer allocation error detector, `msan` (default is no).

Ajouté dans la version 3.6.

`--with-undefined-behavior-sanitizer`

Enable UndefinedBehaviorSanitizer undefined behaviour detector, `ubsan` (default is no).

Ajouté dans la version 3.6.

`--with-thread-sanitizer`

Enable ThreadSanitizer data race detector, `tsan` (default is no).

Ajouté dans la version 3.13.

3.3.10 Linker options

`--enable-shared`

Enable building a shared Python library : `libpython` (default is no).

`--without-static-libpython`

Do not build `libpythonMAJOR.MINOR.a` and do not install `python.o` (built and enabled by default).

Ajouté dans la version 3.10.

3.3.11 Libraries options

`--with-libs``='lib1 ...'`

Link against additional libraries (default is no).

--with-system-expat

Build the `pyexpat` module using an installed `expat` library (default is no).

--with-system-libmpdec

Build the `_decimal` extension module using an installed `mpdecimal` library, see the `decimal` module (default is yes).

Ajouté dans la version 3.3.

Modifié dans la version 3.13 : Default to using the installed `mpdecimal` library.

Deprecated since version 3.13, will be removed in version 3.15 : A copy of the `mpdecimal` library sources will no longer be distributed with Python 3.15.

> **Voir aussi**
>
> *LIBMPDEC_CFLAGS* and *LIBMPDEC_LIBS*.

--with-readline=`readline|editline`

Designate a backend library for the `readline` module.
— readline : Use readline as the backend.
— editline : Use editline as the backend.
Ajouté dans la version 3.10.

--without-readline

Don't build the `readline` module (built by default).

Don't define the `HAVE_LIBREADLINE` macro.

Ajouté dans la version 3.10.

--with-libm=`STRING`

Override `libm` math library to *STRING* (default is system-dependent).

--with-libc=`STRING`

Override `libc` C library to *STRING* (default is system-dependent).

--with-openssl=`DIR`

Root of the OpenSSL directory.

Ajouté dans la version 3.7.

--with-openssl-rpath=`[no|auto|DIR]`

Set runtime library directory (rpath) for OpenSSL libraries :
— `no` (default) : don't set rpath ;
— `auto` : auto-detect rpath from *--with-openssl* and `pkg-config` ;
— *DIR* : set an explicit rpath.
Ajouté dans la version 3.10.

3.3.12 Security Options

--with-hash-algorithm=`[fnv|siphash13|siphash24]`

Select hash algorithm for use in `Python/pyhash.c` :
— `siphash13` (default) ;
— `siphash24` ;
— `fnv`.
Ajouté dans la version 3.4.

Ajouté dans la version 3.11 : `siphash13` is added and it is the new default.

--with-builtin-hashlib-hashes=`md5,sha1,sha256,sha512,sha3,blake2`

Built-in hash modules :

— `md5`;
— `sha1`;
— `sha256`;
— `sha512`;
— `sha3` (with shake);
— `blake2`.

Ajouté dans la version 3.9.

--with-ssl-default-suites=[python|openssl|STRING]

Override the OpenSSL default cipher suites string :
— `python` (default) : use Python's preferred selection;
— `openssl` : leave OpenSSL's defaults untouched;
— *STRING* : use a custom string

See the `ssl` module.

Ajouté dans la version 3.7.

Modifié dans la version 3.10 : The settings `python` and *STRING* also set TLS 1.2 as minimum protocol version.

--disable-safety

Disable compiler options that are recommended by OpenSSF for security reasons with no performance overhead. If this option is not enabled, CPython will be built based on safety compiler options with no slow down. When this option is enabled, CPython will not be built with the compiler options listed below.

The following compiler options are disabled with `--disable-safety` :
— -fstack-protector-strong : Enable run-time checks for stack-based buffer overflows.
— -Wtrampolines : Enable warnings about trampolines that require executable stacks.

Ajouté dans la version 3.14.

--enable-slower-safety

Enable compiler options that are recommended by OpenSSF for security reasons which require overhead. If this option is not enabled, CPython will not be built based on safety compiler options which performance impact. When this option is enabled, CPython will be built with the compiler options listed below.

The following compiler options are enabled with `--enable-slower-safety` :
— -D_FORTIFY_SOURCE=3 : Fortify sources with compile- and run-time checks for unsafe libc usage and buffer overflows.

Ajouté dans la version 3.14.

3.3.13 macOS Options

See Mac/README.rst.

--enable-universalsdk

--enable-universalsdk=SDKDIR

Create a universal binary build. *SDKDIR* specifies which macOS SDK should be used to perform the build (default is no).

--enable-framework

--enable-framework=INSTALLDIR

Create a Python.framework rather than a traditional Unix install. Optional *INSTALLDIR* specifies the installation path (default is no).

--with-universal-archs=ARCH

Specify the kind of universal binary that should be created. This option is only valid when *`--enable-universalsdk`* is set.

Options :
— `universal2`;
— `32-bit`;
— `64-bit`;

— `3-way`;
— `intel`;
— `intel-32`;
— `intel-64`;
— `all`.

--with-framework-name=FRAMEWORK

Specify the name for the python framework on macOS only valid when *--enable-framework* is set (default : `Python`).

--with-app-store-compliance

--with-app-store-compliance=PATCH-FILE

The Python standard library contains strings that are known to trigger automated inspection tool errors when submitted for distribution by the macOS and iOS App Stores. If enabled, this option will apply the list of patches that are known to correct app store compliance. A custom patch file can also be specified. This option is disabled by default.

Ajouté dans la version 3.13.

3.3.14 iOS Options

See iOS/README.rst.

--enable-framework=INSTALLDIR

Create a Python.framework. Unlike macOS, the *INSTALLDIR* argument specifying the installation path is mandatory.

--with-framework-name=FRAMEWORK

Specify the name for the framework (default : `Python`).

3.3.15 Cross Compiling Options

Cross compiling, also known as cross building, can be used to build Python for another CPU architecture or platform. Cross compiling requires a Python interpreter for the build platform. The version of the build Python must match the version of the cross compiled host Python.

--build=BUILD

configure for building on BUILD, usually guessed by **`config.guess`**.

--host=HOST

cross-compile to build programs to run on HOST (target platform)

--with-build-python=path/to/python

path to build `python` binary for cross compiling

Ajouté dans la version 3.11.

CONFIG_SITE=file

An environment variable that points to a file with configure overrides.

Example *config.site* file :

```
# config.site-aarch64
ac_cv_buggy_getaddrinfo=no
ac_cv_file__dev_ptmx=yes
ac_cv_file__dev_ptc=no
```

HOSTRUNNER

Program to run CPython for the host platform for cross-compilation.

Ajouté dans la version 3.11.

Cross compiling example :

```
CONFIG_SITE=config.site-aarch64 ../configure \
    --build=x86_64-pc-linux-gnu \
    --host=aarch64-unknown-linux-gnu \
    --with-build-python=../x86_64/python
```

3.4 Python Build System

3.4.1 Main files of the build system

- `configure.ac` => `configure`;
- `Makefile.pre.in` => `Makefile` (created by `configure`);
- `pyconfig.h` (created by `configure`);
- `Modules/Setup` : C extensions built by the Makefile using `Module/makesetup` shell script;

3.4.2 Main build steps

- C files (`.c`) are built as object files (`.o`).
- A static `libpython` library (`.a`) is created from objects files.
- `python.o` and the static `libpython` library are linked into the final `python` program.
- C extensions are built by the Makefile (see `Modules/Setup`).

3.4.3 Main Makefile targets

make

For the most part, when rebuilding after editing some code or refreshing your checkout from upstream, all you need to do is execute `make`, which (per Make's semantics) builds the default target, the first one defined in the Makefile. By tradition (including in the CPython project) this is usually the `all` target. The `configure` script expands an `autoconf` variable, `@DEF_MAKE_ALL_RULE@` to describe precisely which targets `make all` will build. The three choices are :

- `profile-opt` (configured with `--enable-optimizations`)
- `build_wasm` (chosen if the host platform matches `wasm32-wasi*` or `wasm32-emscripten`)
- `build_all` (configured without explicitly using either of the others)

Depending on the most recent source file changes, Make will rebuild any targets (object files and executables) deemed out-of-date, including running `configure` again if necessary. Source/target dependencies are many and maintained manually however, so Make sometimes doesn't have all the information necessary to correctly detect all targets which need to be rebuilt. Depending on which targets aren't rebuilt, you might experience a number of problems. If you have build or test problems which you can't otherwise explain, `make clean && make` should work around most dependency problems, at the expense of longer build times.

make platform

Build the `python` program, but don't build the standard library extension modules. This generates a file named `platform` which contains a single line describing the details of the build platform, e.g., `macosx-14.3-arm64-3.12` or `linux-x86_64-3.13`.

make profile-opt

Build Python using profile-guided optimization (PGO). You can use the configure *`--enable-optimizations`* option to make this the default target of the `make` command (`make all` or just `make`).

make clean

Remove built files.

make distclean

In addition to the work done by `make clean`, remove files created by the configure script. `configure` will have to be run before building again. [1]

make install

Build the `all` target and install Python.

make test

Build the `all` target and run the Python test suite with the `--fast-ci` option without GUI tests. Variables :

- `TESTOPTS` : additional regrtest command-line options.
- `TESTPYTHONOPTS` : additional Python command-line options.
- `TESTTIMEOUT` : timeout in seconds (default : 10 minutes).

make ci

This is similar to `make test`, but uses the `-ugui` to also run GUI tests.

Ajouté dans la version 3.14.

make buildbottest

This is similar to `make test`, but uses the `--slow-ci` option and default timeout of 20 minutes, instead of `--fast-ci` option.

make regen-all

Regenerate (almost) all generated files. These include (but are not limited to) bytecode cases, and parser generator file. `make regen-stdlib-module-names` and `autoconf` must be run separately for the remaining *generated files*.

3.4.4 C extensions

Some C extensions are built as built-in modules, like the `sys` module. They are built with the `Py_BUILD_CORE_BUILTIN` macro defined. Built-in modules have no `__file__` attribute :

```
>>> import sys
>>> sys
<module 'sys' (built-in)>
>>> sys.__file__
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module 'sys' has no attribute '__file__'
```

Other C extensions are built as dynamic libraries, like the `_asyncio` module. They are built with the `Py_BUILD_CORE_MODULE` macro defined. Example on Linux x86-64 :

```
>>> import _asyncio
>>> _asyncio
<module '_asyncio' from '/usr/lib64/python3.9/lib-dynload/_asyncio.cpython-39-x86_
↪64-linux-gnu.so'>
>>> _asyncio.__file__
'/usr/lib64/python3.9/lib-dynload/_asyncio.cpython-39-x86_64-linux-gnu.so'
```

1. `git clean -fdx` is an even more extreme way to "clean" your checkout. It removes all files not known to Git. When bug hunting using `git bisect`, this is recommended between probes to guarantee a completely clean build. **Use with care**, as it will delete all files not checked into Git, including your new, uncommitted work.

`Modules/Setup` is used to generate Makefile targets to build C extensions. At the beginning of the files, C extensions are built as built-in modules. Extensions defined after the `*shared*` marker are built as dynamic libraries.

The `PyAPI_FUNC()`, `PyAPI_DATA()` and `PyMODINIT_FUNC` macros of `Include/exports.h` are defined differently depending if the `Py_BUILD_CORE_MODULE` macro is defined :

- Use `Py_EXPORTED_SYMBOL` if the `Py_BUILD_CORE_MODULE` is defined
- Use `Py_IMPORTED_SYMBOL` otherwise.

If the `Py_BUILD_CORE_BUILTIN` macro is used by mistake on a C extension built as a shared library, its `PyInit_xxx()` function is not exported, causing an `ImportError` on import.

3.5 Compiler and linker flags

Options set by the `./configure` script and environment variables and used by `Makefile`.

3.5.1 Preprocessor flags

CONFIGURE_CPPFLAGS

Value of *CPPFLAGS* variable passed to the `./configure` script.

Ajouté dans la version 3.6.

CPPFLAGS

(Objective) C/C++ preprocessor flags, e.g. `-Iinclude_dir` if you have headers in a nonstandard directory *include_dir*.

Both *CPPFLAGS* and *LDFLAGS* need to contain the shell's value to be able to build extension modules using the directories specified in the environment variables.

BASECPPFLAGS

Ajouté dans la version 3.4.

PY_CPPFLAGS

Extra preprocessor flags added for building the interpreter object files.

Default : `$(BASECPPFLAGS) -I. -I$(srcdir)/Include $(CONFIGURE_CPPFLAGS) $(CPPFLAGS)`.

Ajouté dans la version 3.2.

3.5.2 Compiler flags

CC

C compiler command.

Example : `gcc -pthread`.

CXX

C++ compiler command.

Example : `g++ -pthread`.

CFLAGS

C compiler flags.

CFLAGS_NODIST

CFLAGS_NODIST is used for building the interpreter and stdlib C extensions. Use it when a compiler flag should *not* be part of *CFLAGS* once Python is installed (gh-65320).

In particular, *CFLAGS* should not contain :

- the compiler flag `-I` (for setting the search path for include files). The `-I` flags are processed from left to right, and any flags in *CFLAGS* would take precedence over user- and package-supplied `-I` flags.
- hardening flags such as `-Werror` because distributions cannot control whether packages installed by users conform to such heightened standards.

Ajouté dans la version 3.5.

COMPILEALL_OPTS

Options passed to the `compileall` command line when building PYC files in `make install`. Default : `-j0`.

Ajouté dans la version 3.12.

EXTRA_CFLAGS

Extra C compiler flags.

CONFIGURE_CFLAGS

Value of *CFLAGS* variable passed to the `./configure` script.

Ajouté dans la version 3.2.

CONFIGURE_CFLAGS_NODIST

Value of *CFLAGS_NODIST* variable passed to the `./configure` script.

Ajouté dans la version 3.5.

BASECFLAGS

Base compiler flags.

OPT

Optimization flags.

CFLAGS_ALIASING

Strict or non-strict aliasing flags used to compile `Python/dtoa.c`.

Ajouté dans la version 3.7.

CCSHARED

Compiler flags used to build a shared library.

For example, `-fPIC` is used on Linux and on BSD.

CFLAGSFORSHARED

Extra C flags added for building the interpreter object files.

Default : `$(CCSHARED)` when *--enable-shared* is used, or an empty string otherwise.

PY_CFLAGS

Default : `$(BASECFLAGS) $(OPT) $(CONFIGURE_CFLAGS) $(CFLAGS) $(EXTRA_CFLAGS)`.

PY_CFLAGS_NODIST

Default : `$(CONFIGURE_CFLAGS_NODIST) $(CFLAGS_NODIST) -I$(srcdir)/Include/internal`.

Ajouté dans la version 3.5.

PY_STDMODULE_CFLAGS

C flags used for building the interpreter object files.

Default : `$(PY_CFLAGS) $(PY_CFLAGS_NODIST) $(PY_CPPFLAGS) $(CFLAGSFORSHARED)`.

Ajouté dans la version 3.7.

PY_CORE_CFLAGS

Default : `$(PY_STDMODULE_CFLAGS) -DPy_BUILD_CORE`.

Ajouté dans la version 3.2.

PY_BUILTIN_MODULE_CFLAGS

Compiler flags to build a standard library extension module as a built-in module, like the `posix` module.

Default : `$(PY_STDMODULE_CFLAGS) -DPy_BUILD_CORE_BUILTIN`.

Ajouté dans la version 3.8.

PURIFY

Purify command. Purify is a memory debugger program.

Default : empty string (not used).

3.5.3 Linker flags

LINKCC

Linker command used to build programs like `python` and `_testembed`.

Default : `$(PURIFY) $(CC)`.

CONFIGURE_LDFLAGS

Value of *LDFLAGS* variable passed to the `./configure` script.

Avoid assigning *CFLAGS*, *LDFLAGS*, etc. so users can use them on the command line to append to these values without stomping the pre-set values.

Ajouté dans la version 3.2.

LDFLAGS_NODIST

LDFLAGS_NODIST is used in the same manner as *CFLAGS_NODIST*. Use it when a linker flag should *not* be part of *LDFLAGS* once Python is installed (gh-65320).

In particular, *LDFLAGS* should not contain :

— the compiler flag `-L` (for setting the search path for libraries). The `-L` flags are processed from left to right, and any flags in *LDFLAGS* would take precedence over user- and package-supplied `-L` flags.

CONFIGURE_LDFLAGS_NODIST

Value of *LDFLAGS_NODIST* variable passed to the `./configure` script.

Ajouté dans la version 3.8.

LDFLAGS

Linker flags, e.g. `-Llib_dir` if you have libraries in a nonstandard directory *lib_dir*.

Both *CPPFLAGS* and *LDFLAGS* need to contain the shell's value to be able to build extension modules using the directories specified in the environment variables.

LIBS

Linker flags to pass libraries to the linker when linking the Python executable.

Example : `-lrt`.

LDSHARED

Command to build a shared library.

Default : `@LDSHARED@ $(PY_LDFLAGS)`.

BLDSHARED

Command to build `libpython` shared library.

Default : `@BLDSHARED@ $(PY_CORE_LDFLAGS)`.

PY_LDFLAGS

Default : `$(CONFIGURE_LDFLAGS) $(LDFLAGS)`.

PY_LDFLAGS_NODIST

Default : `$(CONFIGURE_LDFLAGS_NODIST) $(LDFLAGS_NODIST)`.

Ajouté dans la version 3.8.

PY_CORE_LDFLAGS

Linker flags used for building the interpreter object files.

Ajouté dans la version 3.8.

Notes

CHAPITRE 4

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.

Unlike most Unix systems and services, Windows does not include a system supported installation of Python. To make Python available, the CPython team has compiled Windows installers with every release for many years. These installers are primarily intended to add a per-user installation of Python, with the core interpreter and library being used by a single user. The installer is also able to install for all users of a single machine, and a separate ZIP file is available for application-local distributions.

As specified in **PEP 11**, a Python release only supports a Windows platform while Microsoft considers the platform under extended support. This means that Python 3.14 supports Windows 10 and newer. If you require Windows 7 support, please install Python 3.8. If you require Windows 8.1 support, please install Python 3.12.

Il existe un certain nombre d'installateurs différents disponibles pour Windows, chacun avec certains avantages et inconvénients.

L'installateur complet contient tous les composants et est la meilleure option pour les développeurs utilisant Python pour tout type de projet.

Le paquet Microsoft Store est une simple installation de Python qui convient à l'exécution de scripts, de paquets et à l'utilisation d'IDLE ou d'autres environnements de développement. Il nécessite Windows 10 ou supérieur, mais peut être installé en toute sécurité sans endommager d'autres programmes. Il fournit également de nombreuses commandes pratiques pour lancer Python et ses outils.

Les paquets nuget.org sont des installations légères destinées aux systèmes d'intégration continue. Ils peuvent être utilisés pour créer des packages Python ou exécuter des scripts, mais ne sont pas modifiables et n'ont pas d'outils d'interface utilisateur.

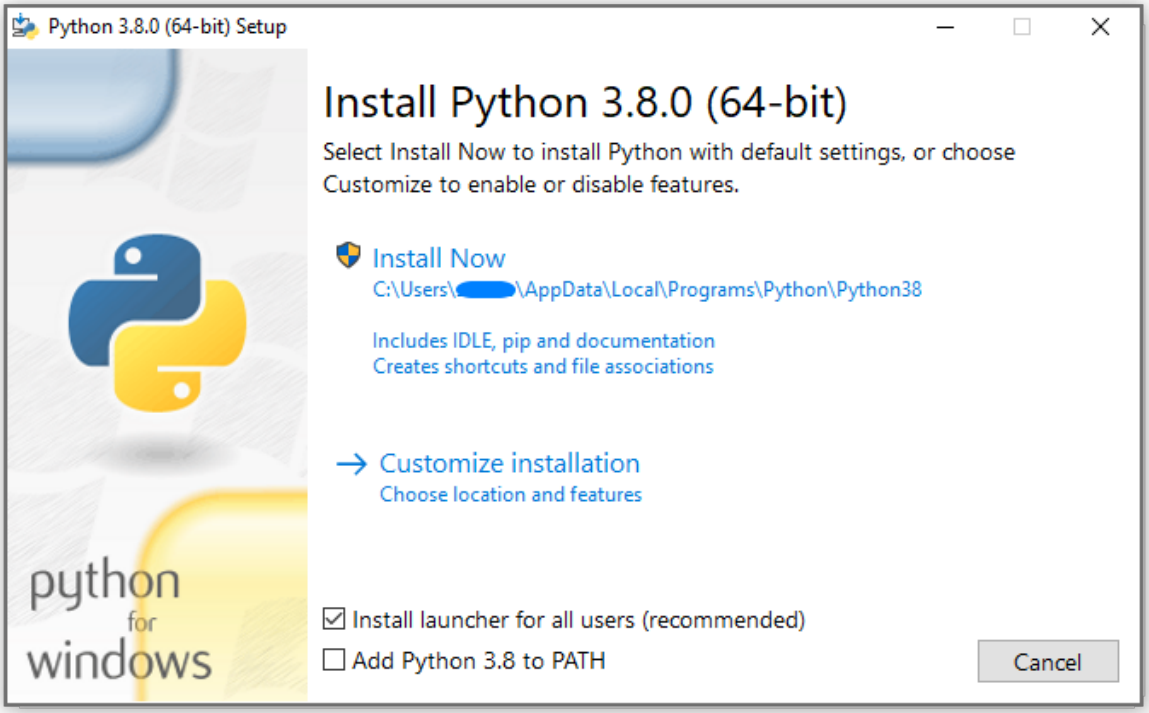
Le paquet intégrable est un paquet minimal de Python approprié pour l'intégration dans une plus grande application.

4.1 L'installateur complet

4.1.1 Étapes d'installation

Quatre installateurs Python 3.14 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

4.1.2 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 de `LongPathsEnabled` à `1` dans de registre à `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem`.

Ceci permet à la fonction `open()`, au module `os` et à la plupart des autres fonctionnalités utilisant des chemins d'accepter et de renvoyer des chemins plus longs que 260 caractères.

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

Modifié dans la version 3.6 : Gestion des chemins longs.

4.1.3 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.

The following options (found by executing the installer with `/?`) can be passed into the installer :

Nom	Description
/passive	to display progress without requiring user interaction
/quiet	to install/uninstall without displaying any UI
/simple	to prevent user customization
/uninstall	to remove Python (without confirmation)
/layout [directory]	to pre-download all components
/log [filename]	to specify log files location

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\Python\PythonXY` ou `%LocalAppData%\Programs\Python\PythonXY-32` ou `%LocalAppData%\Programs\Python\PythonXY-64`
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	Ajoute les dossiers `install` et `Scripts` à `PATH` et assigne `.PY` à `PATHEXT`	0
AppendPath	Ajoute les dossiers `install` et `Scripts` à `PATH` et assigne `.PY` à `PATHEXT`	0
Shortcuts	Crée des raccourcis pour l'interpréteur, la documentation et IDLE si installé.	1
Include_doc	Installe le manuel Python	1
Include_deb	Installe les binaires de débogage	0
Include_dev	Installe les en-têtes et les bibliothèques de développement. L'omission de cette étape peut conduire à une installation inutilisable.	1
Include_exe	Installer `python.exe` et les fichiers associés. L'omission de cette étape peut conduire à une installation inutilisable.	1
Include_laun	Installe le *Lanceur Python pour Windows*.	1
InstallLauncherAllUsers	Installe le lanceur pour tous les utilisateurs. Nécessite que `Include_launcher` soit mis à `1`	1
Include_lib	Installe la bibliothèque standard et les modules d'extension. L'omission de cette étape peut conduire à une installation inutilisable.	1
Include_pip	Installe `pip` et `setuptools`	1
Include_sym	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_tool	Installe les scripts utilitaires	1
LauncherOnly	Installe seulement le lanceur. Ceci écrasera la plupart des autres options.	0

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

```
python-3.9.0.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-3.9.0.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.)

Les options listées ci-dessus peuvent aussi être passées dans un fichier nommé `unattend.xml` à côté de l'exécutable. Ce fichier spécifie une liste d'options et de valeurs. Quand une valeur est donnée en tant qu'attribut, elle sera convertie en nombre si possible. Les valeurs données en élément texte sont toujours laissées en tant que chaînes de caractères. Ce fichier d'exemple propose les mêmes options que l'exemple précédent :

```
<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>
```

4.1.4 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 agencement (*layout*) 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.

Exécutez la commande suivante depuis l'invite de commande pour télécharger tous les fichiers requis possibles. Rappelez-vous de remplacer `python-3.9.0.exe` par le nom réel de votre installateur et de créer des agencements avec leurs propres dossiers pour éviter les conflits entre fichiers du même nom.

```
python-3.9.0.exe /layout [optional target directory]
```

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

4.1.5 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.

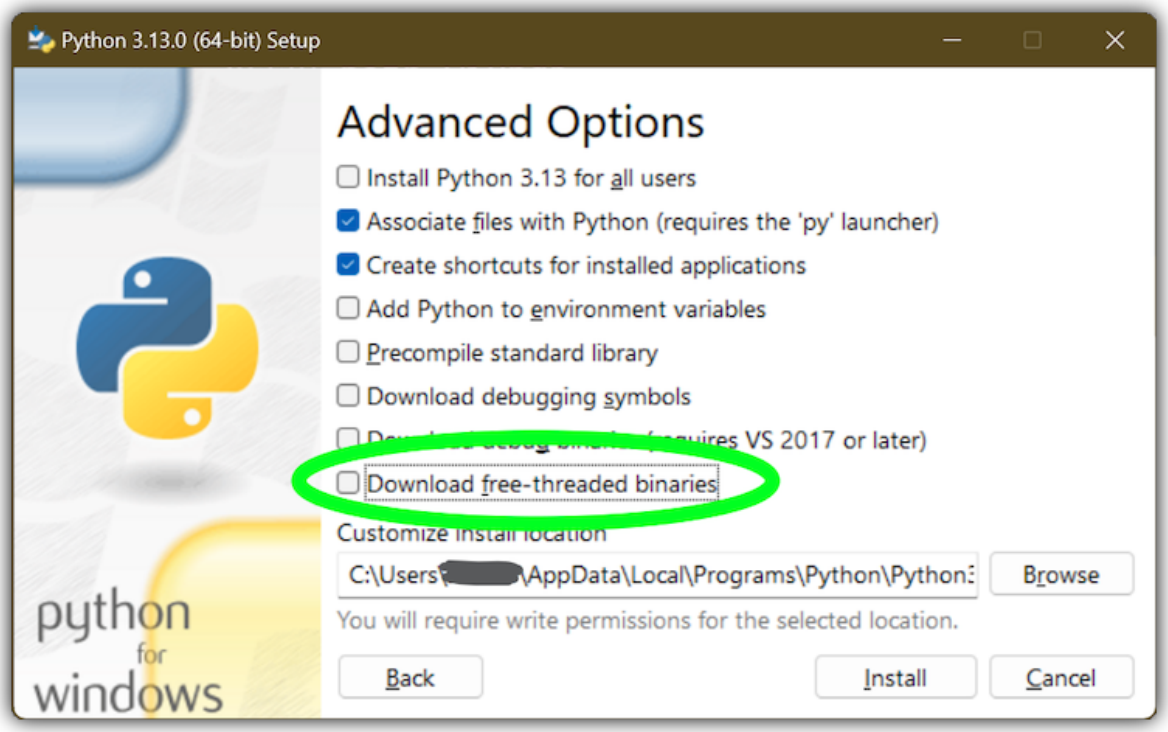
« Uninstall » désinstallera Python entièrement, à l'exception du *Lanceur Python pour Windows* qui à sa propre ligne dans *Programs and Features*.

4.1.6 Installing Free-threaded Binaries

Ajouté dans la version 3.13 : (Experimental)

> **Note**
>
> Everything described in this section is considered experimental, and should be expected to change in future releases.

To install pre-built binaries with free-threading enabled (see **PEP 703**), you should select "Customize installation". The second page of options includes the "Download free-threaded binaries" checkbox.



Selecting this option will download and install additional binaries to the same location as the main Python install. The main executable is called `python3.13t.exe`, and other binaries either receive a `t` suffix or a full ABI suffix. Python source files and bundled third-party dependencies are shared with the main install.

The free-threaded version is registered as a regular Python install with the tag `3.13t` (with a `-32` or `-arm64` suffix as normal for those platforms). This allows tools to discover it, and for the *Lanceur Python pour Windows* to support `py.exe -3.13t`. Note that the launcher will interpret `py.exe -3` (or a `python3` shebang) as "the latest 3.x install", which will prefer the free-threaded binaries over the regular ones, while `py.exe -3.13` will not. If you use the short style of option, you may prefer to not install the free-threaded binaries at this time.

To specify the install option at the command line, use `Include_freethreaded=1`. See *Installation sans téléchargement* for instructions on pre-emptively downloading the additional binaries for offline install. The options to include debug symbols and binaries also apply to the free-threaded builds.

Free-threaded binaries are also available *on nuget.org*.

4.2 Le paquet Microsoft Store

Ajouté dans la version 3.7.2.

Le paquet *Microsoft Store* est un interpréteur Python facilement installable qui est destiné principalement à une utilisation interactive, par exemple, par des étudiants.

Pour installer le paquet, assurez-vous d'avoir les dernières mises à jour de Windows 10 et de chercher dans l'application *Microsoft Store* "Python 3.14". Assurez-vous que l'application que vous sélectionnez est publiée par la *Python Software Foundation* et installez-la.

> **Avertissement**
>
> Python sera toujours disponible gratuitement sur le Microsoft Store. Si vous êtes invité à payer, vous n'avez pas sélectionné le bon paquet.

Après l'installation, Python peut être lancé depuis le menu Démarrer. Il est aussi disponible à partir de n'importe quelle invite de commande ou session PowerShell en entrant `python`. De plus, pip et IDLE peuvent être utilisés en entrant `pip` ou `idle` . IDLE peut également être trouvé dans le menu Démarrer.

Les trois commandes sont également disponibles avec les suffixes de numéro de version, par exemple, `python3.exe` et `python3.x.exe` ainsi que `python.exe` (ou `3.x` est la version spécifique que vous souhaitez lancer, tel que 3.14). Depuis le « menu Démarrer », ouvrez « Alias d'exécution d'application » (ou « *Manage App Execution Aliases* » en anglais) pour choisir quelle version de Python est associée à quelle commande. Il est recommandé de s'assurer que `pip` et `idle` utilisent la même version de Python que la commande `python`.

Les environnements virtuels peuvent être créés avec `python -m venv` et activés et utilisés normalement.

Si vous avez installé une autre version de Python et l'avez ajoutée à votre variable d'environnement `PATH`, elle sera disponible en tant que `python.exe` plutôt que celle de la boutique Microsoft. Pour accéder à la nouvelle installation, utilisez `python3.exe` ou `python3.x.exe`.

Le lanceur `py.exe` va détecter cette installation Python, mais préfèrera les versions de l'installateur traditionnel.

Pour supprimer Python, depuis les paramètres Windows ouvrez *Apps and Features* (Applications et Fonctionnalités), ou bien trouvez Python dans le menu Démarrer et cliquez avec le bouton droit pour sélectionner désinstaller. La désinstallation supprimera tous les paquets que vous avez installés directement dans cette installation Python, mais ne supprimera aucun environnement virtuel

4.2.1 Problèmes connus

Redirection des données locales, du registre et des chemins temporaires

En raison de restrictions sur les applications Microsoft Store, les scripts Python peuvent ne pas avoir un accès en écriture complet aux emplacements partagés tels que `TEMP` et le registre. Au lieu de cela, il écrira sur une copie privée. Si vos scripts doivent modifier les emplacements partagés, vous devrez installer le programme d'installation complet.

At runtime, Python will use a private copy of well-known Windows folders and the registry. For example, if the environment variable `%APPDATA%` is `c:\Users\<user>\AppData\`, then when writing to `C:\Users\<user>\AppData\Local` will write to `C:\Users\<user>\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.8_qbz5n2kfra8p0\LocalCache\Local\`.

When reading files, Windows will return the file from the private folder, or if that does not exist, the real Windows directory. For example reading `C:\Windows\System32` returns the contents of `C:\Windows\System32` plus the contents of `C:\Program Files\WindowsApps\package_name\VFS\SystemX86`.

Vous pouvez trouver le chemin réel de n'importe quel fichier existant en utilisant `os.path.realpath()` :

```
>>> import os
>>> test_file = 'C:\\Users\\example\\AppData\\Local\\test.txt'
>>> os.path.realpath(test_file)
'C:\\Users\\example\\AppData\\Local\\Packages\\PythonSoftwareFoundation.Python.3.8_
↪qbz5n2kfra8p0\\LocalCache\\Local\\test.txt'
```

Lors de l'écriture dans le registre Windows, les comportements suivants existent :

— Reading from `HKLM\\Software` is allowed and results are merged with the `registry.dat` file in the package.

— Writing to `HKLM\\Software` is not allowed if the corresponding key/value exists, i.e. modifying existing keys.
— Writing to `HKLM\\Software` is allowed as long as a corresponding key/value does not exist in the package and the user has the correct access permissions.

For more detail on the technical basis for these limitations, please consult Microsoft's documentation on packaged full-trust apps, currently available at docs.microsoft.com/en-us/windows/msix/desktop/desktop-to-uwp-behind-the-scenes

4.3 Les paquets *nuget.org*

Ajouté dans la version 3.5.2.

Le paquet *nuget.org* est un environnement Python de taille réduite destiné à être utilisé sur des systèmes d'intégration et de génération continus qui n'ont pas Python d'installé. Alors que *nuget* est "le gestionnaire de package pour .NET", il fonctionne également parfaitement bien pour les packages contenant des outils de *build-time*.

Visitez nuget.org pour les informations les plus à jour sur l'utilisation de *nuget*. Ce qui suit est un résumé suffisant pour les développeurs Python.

L'outil de ligne de commande `nuget.exe` peut être téléchargé directement à partir de `https://aka.ms/nugetclidl`, par exemple, à l'aide de *curl* ou de PowerShell. Avec l'outil, la dernière version de Python pour les machines 64 bits ou 32 bits est installée à l'aide de :

```
nuget.exe install python -ExcludeVersion -OutputDirectory .
nuget.exe install pythonx86 -ExcludeVersion -OutputDirectory .
```

Pour sélectionner une version particulière, ajoutez un `-Version 3.x.y`. Le répertoire d'installation être modifié (de `.`), et le paquet sera installé dans un sous-répertoire. Par défaut, le sous-répertoire est nommé comme le paquet, et sans l'option `-ExcludeVersion`, ce nom inclura la version spécifique installée. À l'intérieur du sous-répertoire se trouve un répertoire `tools` qui contient l'installation Python :

```
# Without -ExcludeVersion
> .\python.3.5.2\tools\python.exe -V
Python 3.5.2

# With -ExcludeVersion
> .\python\tools\python.exe -V
Python 3.5.2
```

En général, les paquets *nuget* ne peuvent pas êtres mis à jour et les versions plus récentes doivent être installées côte à côte et référencées à l'aide du chemin d'accès complet. Vous pouvez également supprimer le répertoire du paquet manuellement et l'installer à nouveau. De nombreux systèmes CI le feront automatiquement s'ils ne conservent pas les fichiers entre les *builds*.

À côté du répertoire `tools` est un répertoire `build\native`. Il contient un fichier de propriétés MSBuild `python.props` qui peut être utilisé dans un projet C++ pour référencer l'installation de Python. L'inclusion des paramètres utilisera automatiquement les en-têtes et les bibliothèques d'importation dans votre *build*.

The package information pages on nuget.org are www.nuget.org/packages/python for the 64-bit version, www.nuget.org/packages/pythonx86 for the 32-bit version, and www.nuget.org/packages/pythonarm64 for the ARM64 version

4.3.1 Free-threaded packages

Ajouté dans la version 3.13 : (Experimental)

> **Note**
>
> Everything described in this section is considered experimental, and should be expected to change in future releases.

Packages containing free-threaded binaries are named python-freethreaded for the 64-bit version, pythonx86-freethreaded for the 32-bit version, and pythonarm64-freethreaded for the ARM64 version. These packages contain both the `python3.13t.exe` and `python.exe` entry points, both of which run free threaded.

4.4 Le paquet intégrable

Ajouté 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`, `python37.dll`, `python.exe` and `pythonw.exe` are all provided. Tcl/tk (including all dependents, such as Idle), pip and the Python documentation are not included.

> **Note**
>
> The embedded distribution does not include the Microsoft C Runtime and it is the responsibility of the application installer to provide this. The runtime may have already been installed on a user's system previously or automatically via Windows Update, and can be detected by finding `ucrtbase.dll` in the system directory.

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.

4.4.1 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.

4.4.2 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.

4.5 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
: Installateur avec une compatibilité multiplateforme, de la documentation, et *PyWin32*

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

Enthought Deployment Manager
: « L'environnement Python et gestionnaire de paquets de nouvelle génération ».

  Auparavant, Enthought fournissait Canopy, mais celui-ci est arrivé en fin de vie en 2016.

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.

4.6 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*.

4.6.1 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 3.9;%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, 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://learn.microsoft.com/windows/win32/procthread/environment-variables
Aperçu des variables d'environnement dans Windows

https://learn.microsoft.com/windows-server/administration/windows-commands/set_1
La commande `set`, pour modifier temporairement les variables d'environnement

https://learn.microsoft.com/windows-server/administration/windows-commands/setx
La commande `setx`, pour modifier de façon permanente les variables d'environnement

4.6.2 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 3.9
```

4.7 Mode UTF-8

Ajouté dans la version 3.7.

Windows utilise toujours les anciens codages pour l'encodage du système (la page de code ANSI). Python l'utilise pour le codage par défaut des fichiers texte (par exemple `locale.getencoding()`).

Cela peut poser des problèmes car l'UTF-8 est largement utilisé sur Internet et sur la plupart des systèmes Unix, y compris le WSL (*Windows Subsystem for Linux*).

Vous pouvez utiliser le mode UTF-8 pour changer le codage de texte par défaut en UTF-8. Vous pouvez activer le mode UTF-8 via l'option de ligne de commande `-X utf8` ou la variable d'environnement `PYTHONUTF8=1`. Voir `PYTHONUTF8` pour activer le mode UTF-8, et *Digression : définition des variables d'environnement* pour savoir comment modifier les variables d'environnement.

Lorsque le mode UTF-8 est activé, vous pouvez toujours utiliser l'encodage système (la page de code ANSI) via le codec « *mbcs* ».

Notez que l'ajout de `PYTHONUTF8=1` aux variables d'environnement par défaut affectera toutes les applications Python 3.7+ sur votre système. Si vous avez des applications Python 3.7+ qui dépendent de l'encodage du système existant, il est recommandé de définir la variable d'environnement temporairement ou d'utiliser l'option de ligne de commande `-X utf8`.

Note

même lorsque le mode UTF-8 est désactivé, Python utilise UTF-8 par défaut sur Windows pour :
- Les E/S de la console, y compris les E/S standards (voir **PEP 528** pour plus de détails).
- L'*encodage du système de fichiers* (voir **PEP 529** pour plus de détails).

4.8 Lanceur Python pour Windows

Ajouté 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.

Le lanceur a été initialement spécifié dans **PEP 397**.

4.8.1 Pour commencer

Depuis la ligne de commande

Modifié dans la version 3.6.

Les installations systèmes de Python 3.3 et ultérieur placent 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, 3.7 et 3.14), vous aurez remarqué que Python 3.14 se lance -- pour lancer Python 3.7, essayez la commande :

```
py -3.7
```

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

```
py -2
```

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.
```

La commande :

```
py --list
```

affiche la ou les versions actuellement installées de Python.

The `-x.y` argument is the short form of the `-V:Company/Tag` argument, which allows selecting a specific Python runtime, including those that may have come from somewhere other than python.org. Any runtime registered by following **PEP 514** will be discoverable. The `--list` command lists all available runtimes using the `-V:` format.

When using the `-V:` argument, specifying the Company will limit selection to runtimes from that provider, while specifying only the Tag will select from all providers. Note that omitting the slash implies a tag :

```
# Select any '3.*' tagged runtime
py -V:3

# Select any 'PythonCore' released runtime
py -V:PythonCore/

# Select PythonCore's latest Python 3 runtime
py -V:PythonCore/3
```

The short form of the argument (`-3`) only ever selects from core Python releases, and not other distributions. However, the longer form (`-V:3`) will select from any.

The Company is matched on the full string, case-insensitive. The Tag is matched on either the full string, or a prefix, provided the next character is a dot or a hyphen. This allows `-V:3.1` to match `3.1-32`, but not `3.10`. Tags are sorted using numerical ordering (`3.10` is newer than `3.1`), but are compared using text (`-V:3.01` does not match `3.1`).

Environnements virtuels

Ajouté 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 3.7, essayez de changer la première ligne en `#! python3.7` et vous devriez trouver les informations de version 3.7 affichées.

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.

4.8.2 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`
- `/usr/bin/python`
- `/usr/local/bin/python`
- `python`

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

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

The default Python or an active virtual environment will be located and used. As many Python scripts written to work on Unix will already have this line, you should find these scripts can be used by the launcher without modification. If you are writing a new script on Windows which you hope will be useful on Unix, you should use one of the shebang lines starting with `/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). Furthermore the 32-bit version can be requested by adding "-32" after the minor version. I.e. `/usr/bin/python3.7-32` will request usage of the 32-bit Python 3.7. If a virtual environment is active, the version will be ignored and the environment will be used.

Ajouté dans la version 3.7 : Depuis la version 3.7 du lanceur Python, il est possible de demander une version *64-bit* en utilisant le suffixe **-64**. De plus il est possible de spécifier une version majeure et une architecture sans version mineure (par exemple `/usr/bin/python3-64`).

Modifié dans la version 3.11 : The "-64" suffix is deprecated, and now implies "any architecture that is not provably i386/32-bit". To request a specific environment, use the new `-V:TAG` argument with the complete tag.

Modifié dans la version 3.13 : Virtual commands referencing `python` now prefer an active virtual environment rather than searching `PATH`. This handles cases where the shebang specifies `/usr/bin/env python3` but `python3.exe` is not present in the active environment.

La forme `/usr/bin/env` de la ligne shebang possède une autre propriété spéciale. Avant de rechercher les interpréteurs Python installés, cette forme recherchera dans `PATH` un exécutable Python correspondant au nom fourni en premier argument. Cela correspond au comportement du programme Unix `env`, qui effectue une recherche dans `PATH`. Si un exécutable correspondant au premier argument de la commande `env` ne peut être trouvé, mais que l'argument commence par `python`, il sera traité comme décrit pour les autres commandes virtuelles. La variable d'environnement `PYLAUNCHER_NO_SEARCH_PATH` peut être définie (à n'importe quelle valeur) pour ignorer cette recherche dans `PATH`.

Shebang lines that do not match any of these patterns are looked up in the `[commands]` section of the launcher's *.INI file*. This may be used to handle certain commands in a way that makes sense for your system. The name of the command must be a single argument (no spaces in the shebang executable), and the value substituted is the full path to the executable (additional arguments specified in the .INI will be quoted as part of the filename).

```
[commands]
/bin/xpython=C:\Program Files\XPython\python.exe
```

Any commands not found in the .INI file are treated as **Windows** executable paths that are absolute or relative to the directory containing the script file. This is a convenience for Windows-only scripts, such as those generated by an installer, since the behavior is not compatible with Unix-style shells. These paths may be quoted, and may include multiple arguments, after which the path to the script and any additional arguments will be appended.

4.8.3 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`

4.8.4 Personnalisation

Personnalisation via des fichiers INI

Deux fichiers `.ini` seront recherchés par le lanceur - `py.ini` dans le répertoire de données de l'application de l'utilisateur courant (`%LOCALAPPDATA%` ou `$env:LocalAppData`) et `py.ini` dans le même répertoire que le lanceur. Les mêmes fichiers `.ini` sont utilisés à la fois pour la version *console* du lanceur (i.e. `py.exe`) et pour la version *windows* (i.e. `pyw.exe`).

La personnalisation spécifiée dans le « répertoire de l'application » aura la priorité sur celle à côté de l'exécutable, de sorte qu'un utilisateur, qui peut ne pas avoir accès en écriture au fichier `.ini` à côté du lanceur, peut substituer des commandes dans ce fichier `.ini` global)

Personnalisation des versions Python par défaut

Dans certains cas, un qualificateur de version peut être inclus dans une commande pour dicter quelle version de Python sera utilisée par la commande. Un qualificateur de version commence par un numéro de version majeure et peut éventuellement être suivi d'un point (`.`) et d'un spécificateur de version secondaire. De plus il est possible de préciser si une implémentation 32 ou 64 bit doit être demandée en ajoutant **-32** ou **-64**.

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.

Si aucun qualificateur de version n'est trouvé dans une commande, la variable d'environnement `PY_PYTHON` peut être définie pour spécifier le qualificateur de version par défaut. Si non définie, la valeur par défaut est `3`. La variable peut spécifier n'importe quelle valeur qui peut être passée dans la ligne de commande telle que `3`, `3.7`, `3.7-32` ou `3.7-64`. (Notez que l'option **-64** est seulement disponible avec le lanceur inclus avec Python 3.7 ou plus récent.)

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.

Sur Windows 64-bits avec les implémentations 32-bits et 64-bits de la même version Python (`major.minor`) installée, la version 64-bit est toujours préférée. Cela est vrai pour les implémentations 32-bits et 64-bits du lanceur – un lanceur 32-bits préfère exécuter une installation Python de 64-bits de la version spécifiée si elle est disponible. Le comportement du lanceur est donc prévisible en sachant seulement quelles versions sont installées sur le PC et sans tenir compte de l'ordre dans lequel ils ont été installés (c.-à-d. sans savoir si une version 32 ou 64-bit de Python et le lanceur correspondant a été installé en dernier). Comme indiqué ci-dessus, un suffixe optionnel `-32` ou `-64` peut être utilisé sur un spécificateur de version pour modifier ce comportement.

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.
— La commande `python3.7` ne consulte 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=3.7-32`, la commande `python` utilisera l'implémentation 32-bits de 3.7 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=3.7`, les commandes `python` et `python3` utiliseront spécifiquement 3.7

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=3.7` équivaut au fichier **INI** contenant :

```
[defaults]
python=3.7
```

— Le paramètre `PY_PYTHON=3` et `PY_PYTHON3=3.7` équivaut au fichier *INI* contenant :

```
[defaults]
python=3
python3=3.7
```

4.8.5 Diagnostics

Si une variable d'environnement `PYLAUNCHER_DEBUG` est définie (à n'importe quelle valeur), le lanceur affiche 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.

4.8.6 Exécution à vide

Si une variable d'environnement `PYLAUNCHER_DRYRUN` est définie (à n'importe quelle valeur), le lanceur affiche la commande qu'il aurait exécutée, mais ne lance pas réellement Python. Cela peut être utile pour les outils qui souhaitent utiliser le lanceur pour détecter puis lancer Python directement. Notez que la commande écrite sur la sortie standard est toujours codée en UTF-8 et peut ne pas s'afficher correctement dans la console.

4.8.7 Installation à la demande

Si une variable d'environnement `PYLAUNCHER_ALLOW_INSTALL` est définie (à n'importe quelle valeur) et que la version de Python demandée n'est pas installée mais est disponible sur le Microsoft Store, le lanceur tente de l'installer. Cela peut nécessiter une intervention de l'utilisateur et il faudra peut-être exécuter à nouveau la commande.

Une variable supplémentaire `PYLAUNCHER_ALWAYS_INSTALL` oblige le lanceur à toujours essayer d'installer Python, même s'il est détecté. Ceci est principalement destiné aux tests (et doit être utilisé avec `PYLAUNCHER_DRYRUN`).

4.8.8 Codes de retour

Les codes de retour suivants peuvent être renvoyés par le lanceur Python. Malheureusement, il n'y a aucun moyen de les distinguer du code de sortie de Python lui-même.

Les noms des codes sont tels qu'utilisés dans les sources et ne sont donnés qu'à titre de référence. Il n'y a aucun moyen d'y accéder ou de les résoudre en dehors de la lecture de cette page. Les entrées sont classées par ordre alphabétique des noms.

Nom	Valeur	Description
RC_BAD_VENV_CFG	107	`pyvenv.cfg` a été trouvé mais est corrompu.
RC_CREATE_PROCESS	101	Échec du lancement de Python.
RC_INSTALLING	111	Une installation a commencé, mais la commande doit être relancée après qu'elle a terminé.
RC_INTERNAL_ERROR	109	Erreur non prévue. Merci de remonter un bogue.
RC_NO_COMMANDLI	108	Le système d'exploitation n'a pas fourni de ligne de commande.
RC_NO_PYTHON	103	Impossible de trouver la version demandée.
RC_NO_VENV_CFG	106	`pyvenv.cfg` est requis mais n'a pas été trouvé.

4.9 Recherche de modules

Ces notes complètent la description de sys-path-init avec des notes Windows détaillées.

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* figé, 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 `python37.dll` in your own executable, explicitly set `PyConfig.module_search_paths` before `Py_InitializeFromConfig()`.
- 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.)

Ceux-ci garantiront que les fichiers d'une installation à l'échelle du système n'auront pas la priorité sur la copie de la bibliothèque standard livrée avec votre application. Sinon, vos utilisateurs pourraient rencontrer des problèmes en utilisant votre application. Notez que la première suggestion est la meilleure, car les autres peuvent encore être sensibles aux chemins non-standard dans le registre et le *site-packages* utilisateur.

Modifié dans la version 3.6 : Add `._pth` file support and removes `applocal` option from `pyvenv.cfg`.

Modifié dans la version 3.6 : Add `pythonXX.zip` as a potential landmark when directly adjacent to the executable.

Obsolète depuis la version 3.6 : Modules specified in the registry under `Modules` (not `PythonPath`) may be imported by `importlib.machinery.WindowsRegistryFinder`. This finder is enabled on Windows in 3.6.0 and earlier, but may need to be explicitly added to `sys.meta_path` in the future.

4.10 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.

4.10.1 PyWin32

The PyWin32 module by Mark Hammond is a collection of modules for advanced Windows-specific support. This includes utilities for :

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

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

4.10.2 cx_Freeze

cx_Freeze 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.

4.11 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, 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.

4.12 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 n'est plus pris en charge https://github.com/python/cpython/issues/71542 depuis Python 3 (s'il l'a jamais été).
- Le programme d'installation de Cygwin propose d'installer également l'interpréteur Python.

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

CHAPITRE 5

Using Python on macOS

This document aims to give an overview of macOS-specific behavior you should know about to get started with Python on Mac computers. Python on a Mac running macOS is very similar to Python on other Unix-derived platforms, but there are some differences in installation and some features.

There are various ways to obtain and install Python for macOS. Pre-built versions of the most recent versions of Python are available from a number of distributors. Much of this document describes use of the Pythons provided by the CPython release team for download from the python.org website. See *Alternative Distributions* for some other options.

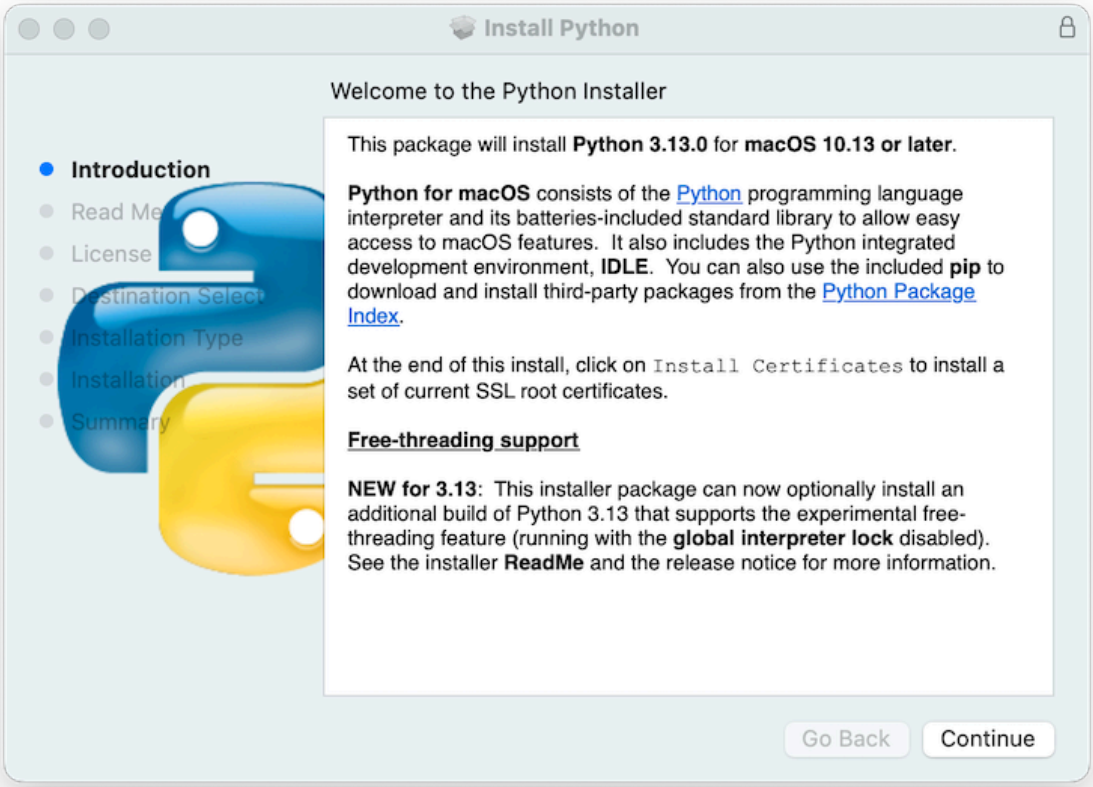
5.1 Using Python for macOS from `python.org`

5.1.1 Étapes d'installation

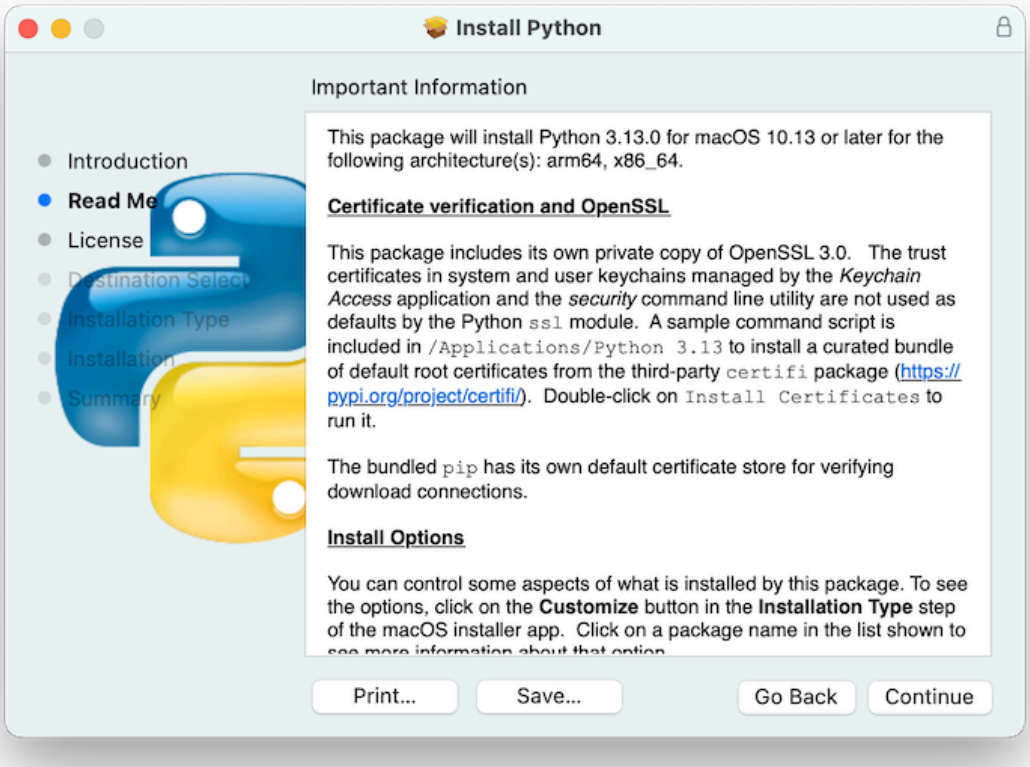
For current Python versions (other than those in `security` status), the release team produces a **Python for macOS** installer package for each new release. A list of available installers is available here. We recommend using the most recent supported Python version where possible. Current installers provide a universal2 binary build of Python which runs natively on all Macs (Apple Silicon and Intel) that are supported by a wide range of macOS versions, currently typically from at least **macOS 10.13 High Sierra** on.

The downloaded file is a standard macOS installer package file (`.pkg`). File integrity information (checksum, size, sigstore signature, etc) for each file is included on the release download page. Installer packages and their contents are signed and notarized with `Python Software Foundation` Apple Developer ID certificates to meet macOS Gatekeeper requirements.

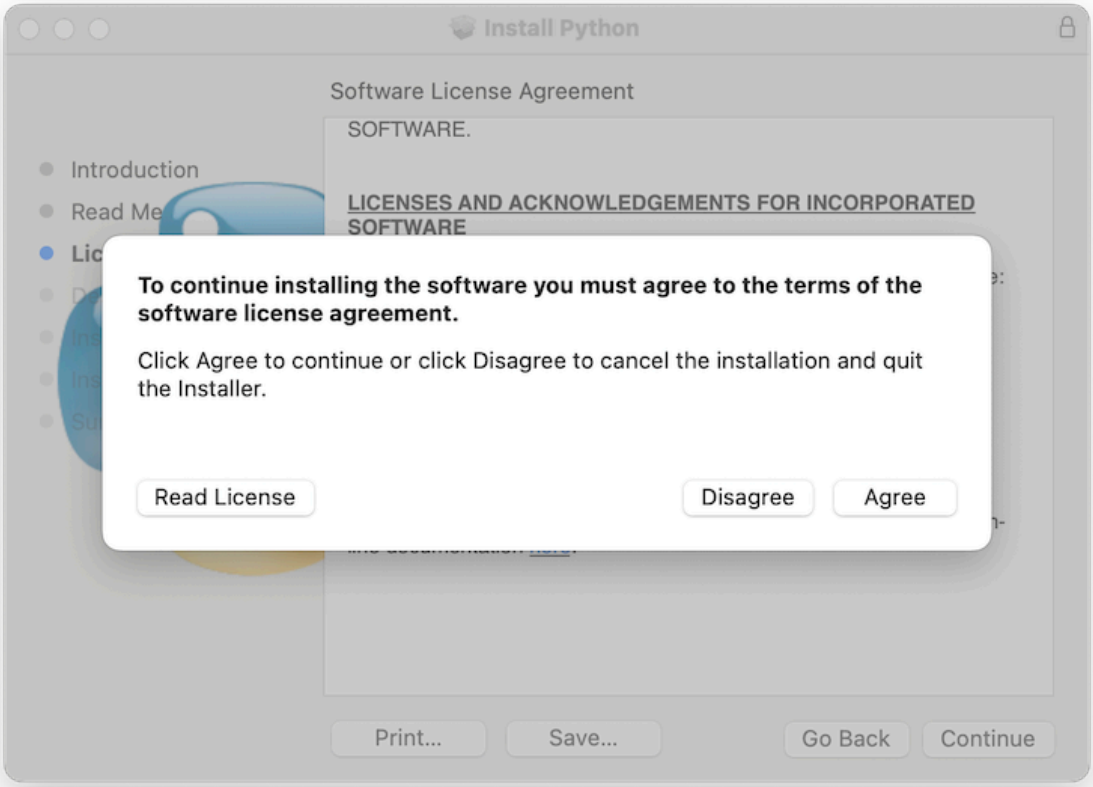
For a default installation, double-click on the downloaded installer package file. This should launch the standard macOS Installer app and display the first of several installer windows steps.



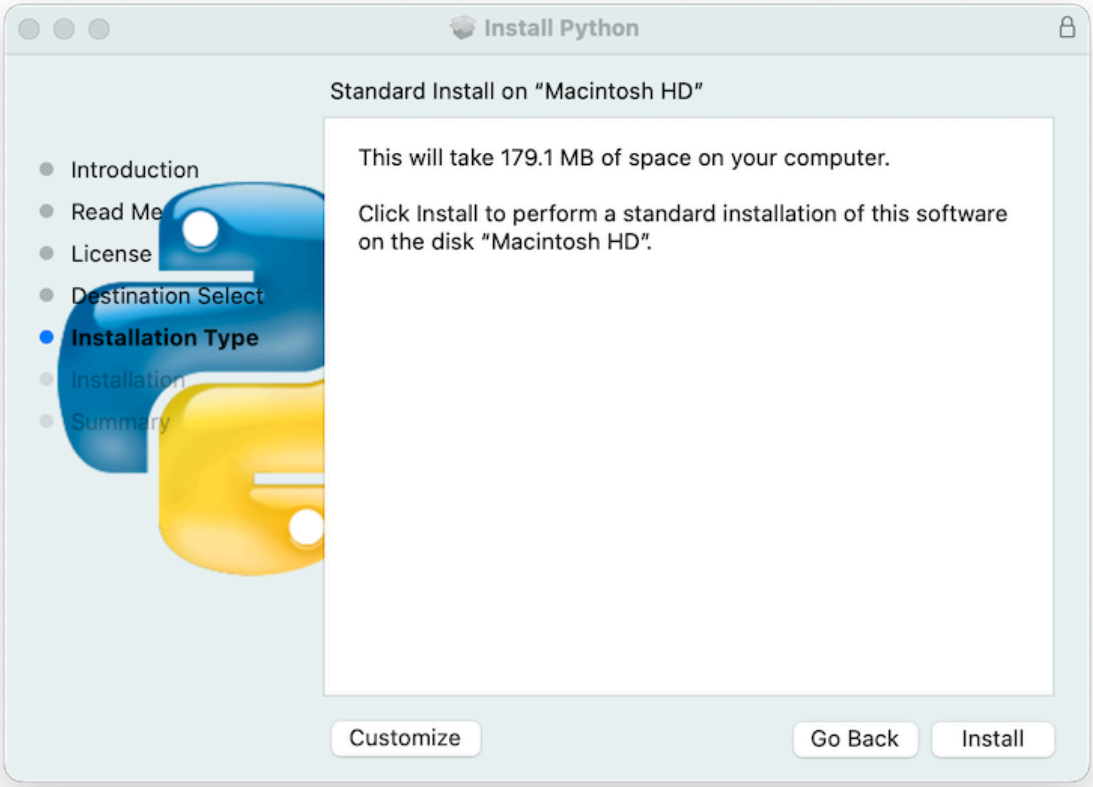
Clicking on the **Continue** button brings up the **Read Me** for this installer. Besides other important information, the **Read Me** documents which Python version is going to be installed and on what versions of macOS it is supported. You may need to scroll through to read the whole file. By default, this **Read Me** will also be installed in `/Applications/Python 3.13/` and available to read anytime.



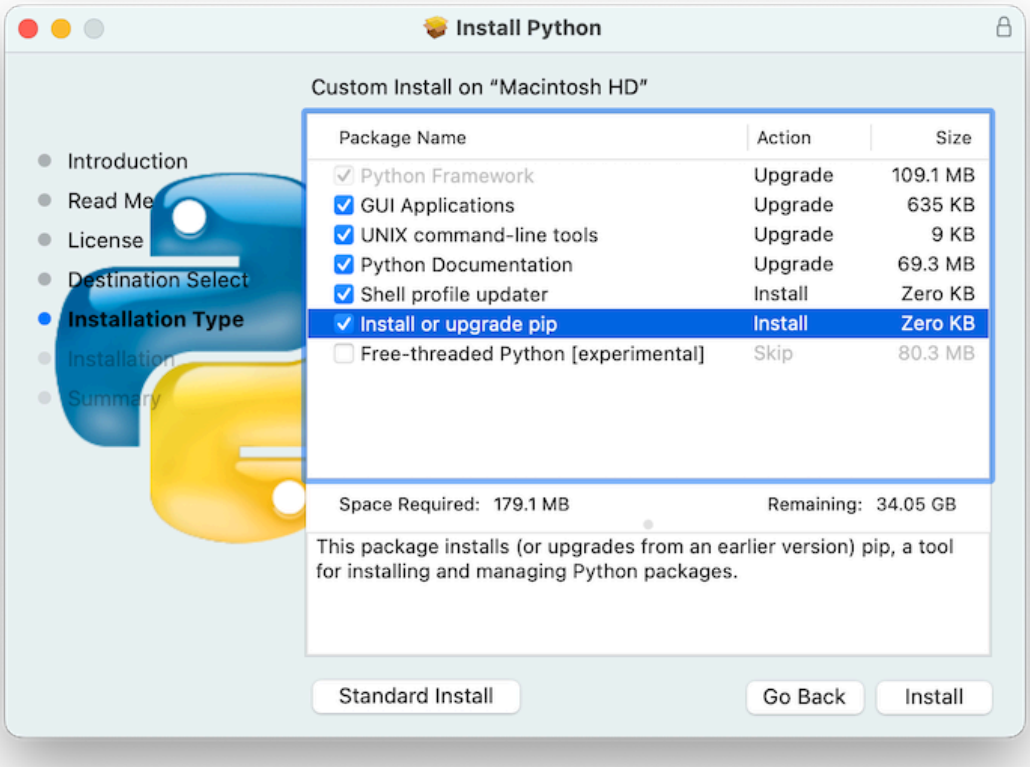
Clicking on **Continue** proceeds to display the license for Python and for other included software. You will then need to **Agree** to the license terms before proceeding to the next step. This license file will also be installed and available to be read later.



After the license terms are accepted, the next step is the **Installation Type** display. For most uses, the standard set of installation operations is appropriate.



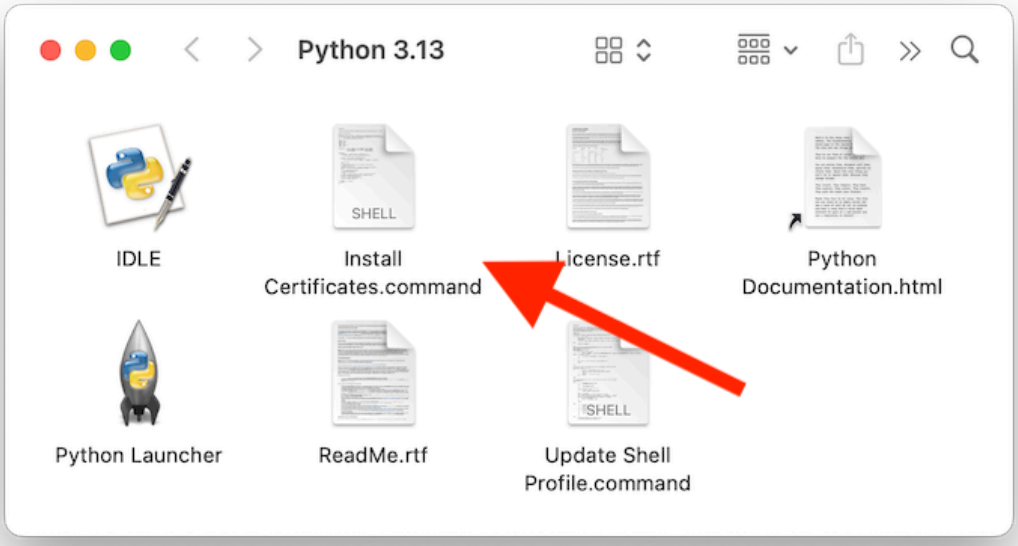
By pressing the **Customize** button, you can choose to omit or select certain package components of the installer. Click on each package name to see a description of what it installs. To also install support for the optional experimental free-threaded feature, see *Installing Free-threaded Binaries*.



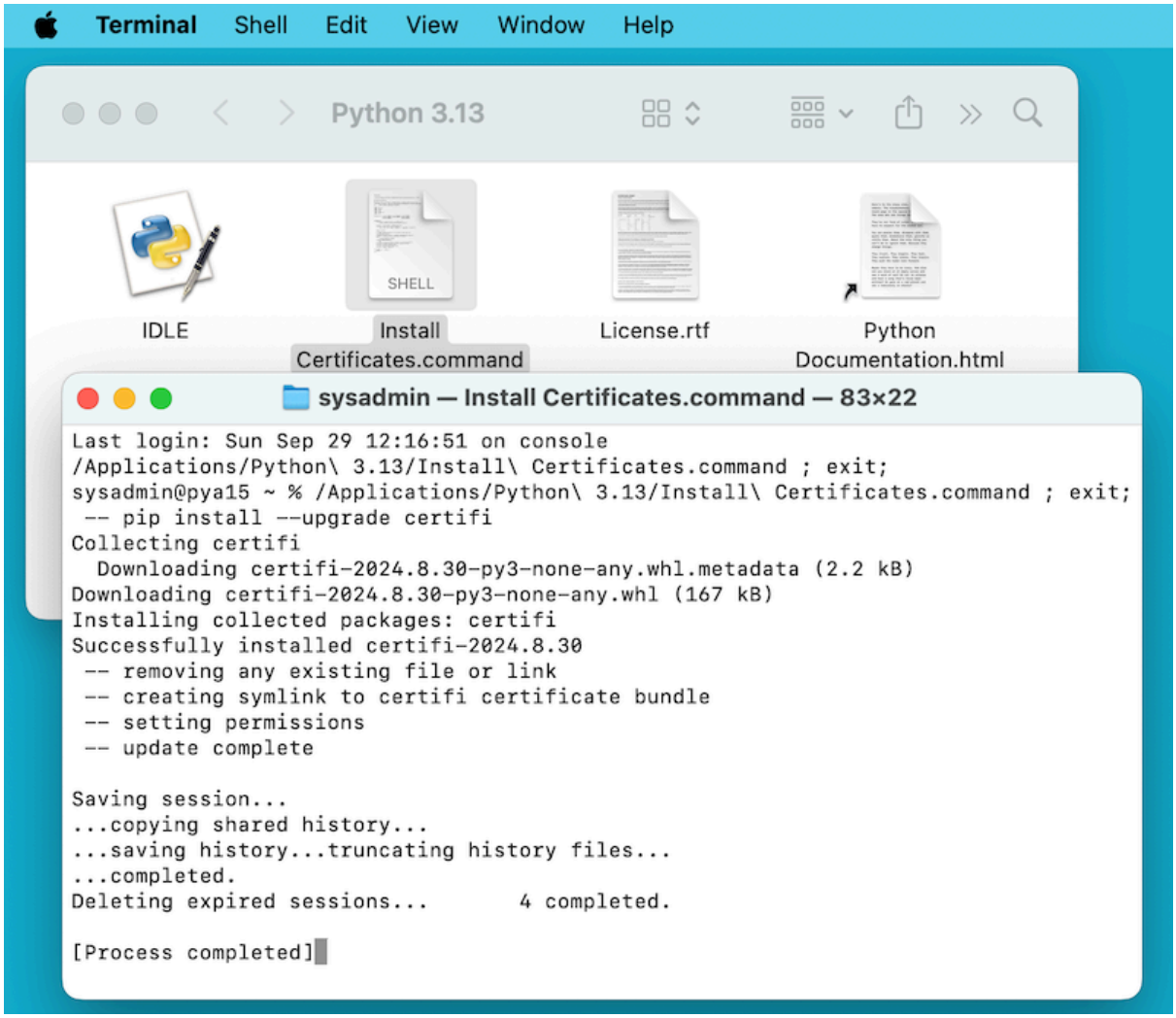
In either case, clicking **Install** will begin the install process by asking permission to install new software. A macOS user name with `Administrator` privilege is needed as the installed Python will be available to all users of the Mac.

When the installation is complete, the **Summary** window will appear.

Double-click on the **`Install Certificates.command`** icon or file in the `/Applications/Python 3.13/` window to complete the installation.



This will open a temporary **`Terminal`** shell window that will use the new Python to download and install SSL root certificates for its use.



If `Successfully installed certifi` and `update complete` appears in the terminal window, the installation is complete. Close this terminal window and the installer window.

A default install will include :

— A `Python 3.13` folder in your `Applications` folder. In here you find **IDLE**, the development environment that is a standard part of official Python distributions ; and **Python Launcher**, which handles double-clicking Python scripts from the macOS Finder.

— A framework `/Library/Frameworks/Python.framework`, which includes the Python executable and libraries. The installer adds this location to your shell path. To uninstall Python, you can remove these three things. Symlinks to the Python executable are placed in `/usr/local/bin/`.

Note

Recent versions of macOS include a **python3** command in `/usr/bin/python3` that links to a usually older and incomplete version of Python provided by and for use by the Apple development tools, **Xcode** or the **Command Line Tools for Xcode**. You should never modify or attempt to delete this installation, as it is Apple-controlled and is used by Apple-provided or third-party software. If you choose to install a newer Python version from `python.org`, you will have two different but functional Python installations on your computer that can co-exist. The default installer options should ensure that its **python3** will be used instead of the system **python3**.

5.1.2 Comment exécuter un script Python

There are two ways to invoke the Python interpreter. If you are familiar with using a Unix shell in a terminal window, you can invoke `python3.13` or `python3` optionally followed by one or more command line options (described in *Ligne de commande et environnement*). The Python tutorial also has a useful section on using Python interactively from a shell.

You can also invoke the interpreter through an integrated development environment. idle is a basic editor and interpreter environment which is included with the standard distribution of Python. **IDLE** includes a Help menu that allows you to access Python documentation. If you are completely new to Python, you can read the tutorial introduction in that document.

There are many other editors and IDEs available, see *Éditeurs et IDEs* for more information.

To run a Python script file from the terminal window, you can invoke the interpreter with the name of the script file :

```
python3.13 myscript.py
```

To run your script from the Finder, you can either :

- Drag it to **Python Launcher**.
- Select **Python Launcher** as the default application to open your script (or any `.py` script) through the Finder Info window and double-click it. **Python Launcher** has various preferences to control how your script is launched. Option-dragging allows you to change these for one invocation, or use its `Preferences` menu to change things globally.

Be aware that running the script directly from the macOS Finder might produce different results than when running from a terminal window as the script will not be run in the usual shell environment including any setting of environment variables in shell profiles. And, as with any other script or program, be certain of what you are about to run.

5.2 Alternative Distributions

Besides the standard `python.org` for macOS installer, there are third-party distributions for macOS that may include additional functionality. Some popular distributions and their key features :

ActivePython
: Installer with multi-platform compatibility, documentation

Anaconda
: Popular scientific modules (such as numpy, scipy, and pandas) and the `conda` package manager.

Homebrew
: Package manager for macOS including multiple versions of Python and many third-party Python-based packages (including numpy, scipy, and pandas).

MacPorts
: Another package manager for macOS including multiple versions of Python and many third-party Python-based packages. May include pre-built versions of Python and many packages for older versions of macOS.

Note that distributions might not include the latest versions of Python or other libraries, and are not maintained or supported by the core Python team.

5.3 Installation de paquets Python additionnels

Refer to the Python Packaging User Guide for more information.

5.4 GUI Programming

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

The standard Python GUI toolkit is `tkinter`, based on the cross-platform Tk toolkit (https://www.tcl.tk). A macOS-native version of Tk is included with the installer.

PyObjC is a Python binding to Apple's Objective-C/Cocoa framework. Information on PyObjC is available from pyobjc.

A number of alternative macOS GUI toolkits are available including :

- PySide : Official Python bindings to the Qt GUI toolkit.
- PyQt : Alternative Python bindings to Qt.
- Kivy : A cross-platform GUI toolkit that supports desktop and mobile platforms.
- Toga : Part of the BeeWare Project ; supports desktop, mobile, web and console apps.
- wxPython : A cross-platform toolkit that supports desktop operating systems.

5.5 Sujets avancés

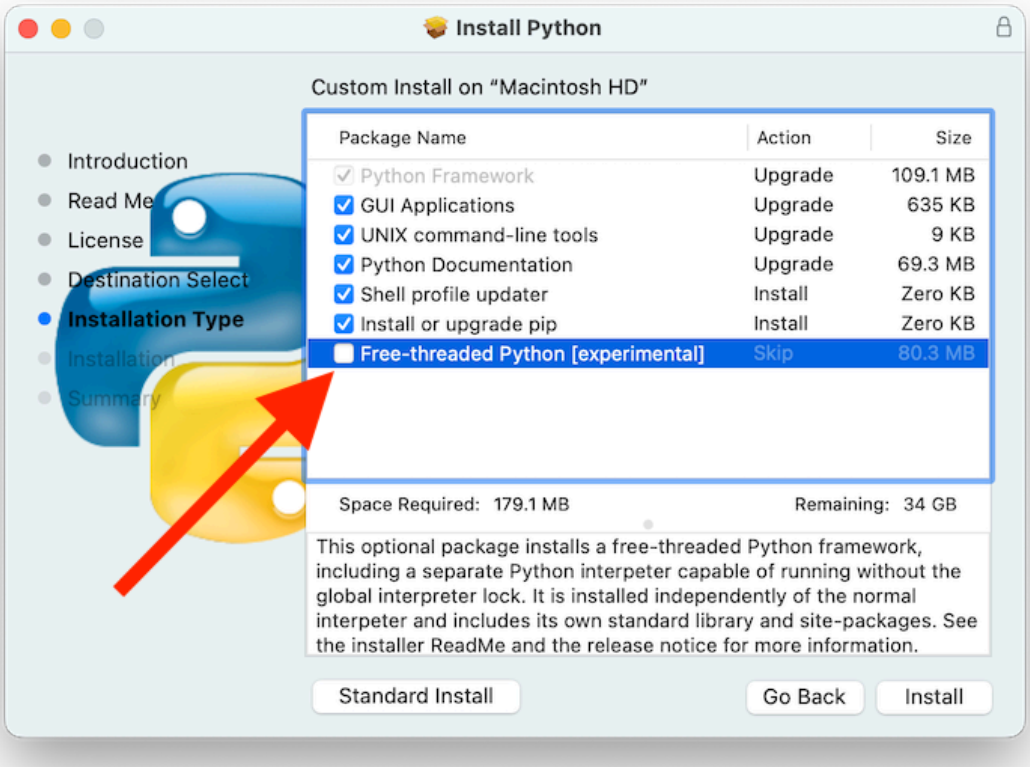
5.5.1 Installing Free-threaded Binaries

Ajouté dans la version 3.13 : (Experimental)

> **Note**
>
> Everything described in this section is considered experimental, and should be expected to change in future releases.

The `python.org` *Python for macOS* installer package can optionally install an additional build of Python 3.13 that supports **PEP 703**, the experimental free-threading feature (running with the *global interpreter lock* disabled). Check the release page on `python.org` for possible updated information.

Because this feature is still considered experimental, the support for it is not installed by default. It is packaged as a separate install option, available by clicking the **Customize** button on the **Installation Type** step of the installer as described above.



If the box next to the **Free-threaded Python** package name is checked, a separate `PythonT.framework` will also be installed alongside the normal `Python.framework` in `/Library/Frameworks`. This configuration allows a

free-threaded Python 3.13 build to co-exist on your system with a traditional (GIL only) Python 3.13 build with minimal risk while installing or testing. This installation layout is itself experimental and is subject to change in future releases.

Known cautions and limitations :

- The **UNIX command-line tools** package, which is selected by default, will install links in `/usr/local/bin` for `python3.13t`, the free-threaded interpreter, and `python3.13t-config`, a configuration utility which may be useful for package builders. Since `/usr/local/bin` is typically included in your shell `PATH`, in most cases no changes to your `PATH` environment variables should be needed to use `python3.13t`.
- For this release, the **Shell profile updater** package and the `Update Shell Profile.command` in `/Applications/Python 3.13/` do not support the free-threaded package.
- The free-threaded build and the traditional build have separate search paths and separate `site-packages` directories so, by default, if you need a package available in both builds, it may need to be installed in both. The free-threaded package will install a separate instance of **`pip`** for use with `python3.13t`.
  - To install a package using **`pip`** without a **`venv`** :

    ```
    python3.13t -m pip install <package_name>
    ```

- When working with multiple Python environments, it is usually safest and easiest to create and use virtual environments. This can avoid possible command name conflicts and confusion about which Python is in use :

  ```
  python3.13t -m venv <venv_name>
  ```

  then **`activate`**.
- To run a free-threaded version of IDLE :

  ```
  python3.13t -m idlelib
  ```

- The interpreters in both builds respond to the same *PYTHON environment variables* which may have unexpected results, for example, if you have `PYTHONPATH` set in a shell profile. If necessary, there are *command line options* like `-E` to ignore these environment variables.
- The free-threaded build links to the third-party shared libraries, such as `OpenSSL` and `Tk`, installed in the traditional framework. This means that both builds also share one set of trust certificates as installed by the **`Install Certificates.command`** script, thus it only needs to be run once.
- If you cannot depend on the link in `/usr/local/bin` pointing to the `python.org` free-threaded `python3.13t` (for example, if you want to install your own version there or some other distribution does), you can explicitly set your shell `PATH` environment variable to include the `PythonT` framework `bin` directory :

  ```
  export PATH="/Library/Frameworks/PythonT.framework/Versions/3.13/bin":"$PATH"
  ```

  The traditional framework installation by default does something similar, except for `Python.framework`. Be aware that having both framework `bin` directories in `PATH` can lead to confusion if there are duplicate names like `python3.13` in both ; which one is actually used depends on the order they appear in `PATH`. The `which python3.x` or `which python3.xt` commands can show which path is being used. Using virtual environments can help avoid such ambiguities. Another option might be to create a shell **`alias`** to the desired interpreter, like :

  ```
  alias py3.13="/Library/Frameworks/Python.framework/Versions/3.13/bin/python3.
  ↪13"
  alias py3.13t="/Library/Frameworks/PythonT.framework/Versions/3.13/bin/
  ↪python3.13t"
  ```

5.5.2 Installing using the command line

If you want to use automation to install the `python.org` installer package (rather than by using the familiar macOS **`Installer`** GUI app), the macOS command line **`installer`** utility lets you select non-default options, too. If you are not familiar with **`installer`**, it can be somewhat cryptic (see **`man installer`** for more information). As an example, the following shell snippet shows one way to do it, using the `3.13.0b2` release and selecting the free-threaded interpreter option :

```
RELEASE="python-3.13.0b2-macos11.pkg"
```

(suite sur la page suivante)

(suite de la page précédente)

```
# download installer pkg
curl -O https://www.python.org/ftp/python/3.13.0/${RELEASE}

# create installer choicechanges to customize the install:
#    enable the PythonTFramework-3.13 package
#    while accepting the other defaults (install all other packages)
cat > ./choicechanges.plist <<EOF
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
↪PropertyList-1.0.dtd">
<plist version="1.0">
<array>
        <dict>
                <key>attributeSetting</key>
                <integer>1</integer>
                <key>choiceAttribute</key>
                <string>selected</string>
                <key>choiceIdentifier</key>
                <string>org.python.Python.PythonTFramework-3.13</string>
        </dict>
</array>
</plist>
EOF

sudo installer -pkg ./${RELEASE} -applyChoiceChangesXML ./choicechanges.plist -
↪target /
```

You can then test that both installer builds are now available with something like :

```
$ # test that the free-threaded interpreter was installed if the Unix Command␣
↪Tools package was enabled
$ /usr/local/bin/python3.13t -VV
Python 3.13.0b2 experimental free-threading build (v3.13.0b2:3a83b172af, Jun  5␣
↪2024, 12:57:31) [Clang 15.0.0 (clang-1500.3.9.4)]
$ #    and the traditional interpreter
$ /usr/local/bin/python3.13 -VV
Python 3.13.0b2 (v3.13.0b2:3a83b172af, Jun  5 2024, 12:50:24) [Clang 15.0.0 (clang-
↪1500.3.9.4)]
$ # test that they are also available without the prefix if /usr/local/bin is on
↪$PATH
$ python3.13t -VV
Python 3.13.0b2 experimental free-threading build (v3.13.0b2:3a83b172af, Jun  5␣
↪2024, 12:57:31) [Clang 15.0.0 (clang-1500.3.9.4)]
$ python3.13 -VV
Python 3.13.0b2 (v3.13.0b2:3a83b172af, Jun  5 2024, 12:50:24) [Clang 15.0.0 (clang-
↪1500.3.9.4)]
```

Note

Current `python.org` installers only install to fixed locations like `/Library/Frameworks/`, `/Applications`, and `/usr/local/bin`. You cannot use the **`installer`** `-domain` option to install to other locations.

5.5.3 Distributing Python Applications

A range of tools exist for converting your Python code into a standalone distributable application :

- py2app : Supports creating macOS `.app` bundles from a Python project.
- Briefcase : Part of the BeeWare Project ; a cross-platform packaging tool that supports creation of `.app` bundles on macOS, as well as managing signing and notarization.
- PyInstaller : A cross-platform packaging tool that creates a single file or folder as a distributable artifact.

5.5.4 App Store Compliance

Apps submitted for distribution through the macOS App Store must pass Apple's app review process. This process includes a set of automated validation rules that inspect the submitted application bundle for problematic code.

The Python standard library contains some code that is known to violate these automated rules. While these violations appear to be false positives, Apple's review rules cannot be challenged. Therefore, it is necessary to modify the Python standard library for an app to pass App Store review.

The Python source tree contains a patch file that will remove all code that is known to cause issues with the App Store review process. This patch is applied automatically when CPython is configured with the `--with-app-store-compliance` option.

This patch is not normally required to use CPython on a Mac ; nor is it required if you are distributing an app *outside* the macOS App Store. It is *only* required if you are using the macOS App Store as a distribution channel.

5.6 Autres ressources

The python.org Help page has links to many useful resources. The Pythonmac-SIG mailing list is another support resource specifically for Python users and developers on the Mac.

CHAPITRE 6

Using Python on Android

Python on Android is unlike Python on desktop platforms. On a desktop platform, Python is generally installed as a system resource that can be used by any user of that computer. Users then interact with Python by running a **`python`** executable and entering commands at an interactive prompt, or by running a Python script.

On Android, there is no concept of installing as a system resource. The only unit of software distribution is an "app". There is also no console where you could run a **`python`** executable, or interact with a Python REPL.

As a result, the only way you can use Python on Android is in embedded mode – that is, by writing a native Android application, embedding a Python interpreter using `libpython`, and invoking Python code using the Python embedding API. The full Python interpreter, the standard library, and all your Python code is then packaged into your app for its own private use.

The Python standard library has some notable omissions and restrictions on Android. See the API availability guide for details.

6.1 Adding Python to an Android app

These instructions are only needed if you're planning to compile Python for Android yourself. Most users should *not* need to do this. Instead, use one of the following tools, which will provide a much easier experience :

- Briefcase, from the BeeWare project
- Buildozer, from the Kivy project
- Chaquopy
- pyqtdeploy
- Termux

If you're sure you want to do all of this manually, read on. You can use the testbed app as a guide ; each step below contains a link to the relevant file.

- Build Python by following the instructions in Android/README.md.
- Add code to your build.gradle file to copy the following items into your project. All except your own Python code can be copied from `cross-build/HOST/prefix/lib` :
  - In your JNI libraries :
    - `libpython*.*.so`
    - `lib*_python.so` (external libraries such as OpenSSL)
  - In your assets :
    - `python*.*` (the Python standard library)
    - `python*.*/site-packages` (your own Python code)
- Add code to your app to extract the assets to the filesystem.
- Add code to your app to start Python in embedded mode. This will need to be C code called via JNI.

CHAPITRE 7

Using Python on iOS

Authors
Russell Keith-Magee (2024-03)

Python on iOS is unlike Python on desktop platforms. On a desktop platform, Python is generally installed as a system resource that can be used by any user of that computer. Users then interact with Python by running a **`python`** executable and entering commands at an interactive prompt, or by running a Python script.

On iOS, there is no concept of installing as a system resource. The only unit of software distribution is an "app". There is also no console where you could run a **`python`** executable, or interact with a Python REPL.

As a result, the only way you can use Python on iOS is in embedded mode - that is, by writing a native iOS application, and embedding a Python interpreter using `libPython`, and invoking Python code using the Python embedding API. The full Python interpreter, the standard library, and all your Python code is then packaged as a standalone bundle that can be distributed via the iOS App Store.

If you're looking to experiment for the first time with writing an iOS app in Python, projects such as BeeWare and Kivy will provide a much more approachable user experience. These projects manage the complexities associated with getting an iOS project running, so you only need to deal with the Python code itself.

7.1 Python at runtime on iOS

7.1.1 iOS version compatibility

The minimum supported iOS version is specified at compile time, using the *`--host`* option to `configure`. By default, when compiled for iOS, Python will be compiled with a minimum supported iOS version of 13.0. To use a different minimum iOS version, provide the version number as part of the `--host` argument - for example, `--host=arm64-apple-ios15.4-simulator` would compile an ARM64 simulator build with a deployment target of 15.4.

7.1.2 Platform identification

When executing on iOS, `sys.platform` will report as `ios`. This value will be returned on an iPhone or iPad, regardless of whether the app is running on the simulator or a physical device.

Information about the specific runtime environment, including the iOS version, device model, and whether the device is a simulator, can be obtained using `platform.ios_ver()`. `platform.system()` will report `iOS` or `iPadOS`, depending on the device.

`os.uname()` reports kernel-level details ; it will report a name of `Darwin`.

7.1.3 Standard library availability

The Python standard library has some notable omissions and restrictions on iOS. See the API availability guide for iOS for details.

7.1.4 Binary extension modules

One notable difference about iOS as a platform is that App Store distribution imposes hard requirements on the packaging of an application. One of these requirements governs how binary extension modules are distributed.

The iOS App Store requires that *all* binary modules in an iOS app must be dynamic libraries, contained in a framework with appropriate metadata, stored in the `Frameworks` folder of the packaged app. There can be only a single binary per framework, and there can be no executable binary material outside the `Frameworks` folder.

This conflicts with the usual Python approach for distributing binaries, which allows a binary extension module to be loaded from any location on `sys.path`. To ensure compliance with App Store policies, an iOS project must post-process any Python packages, converting `.so` binary modules into individual standalone frameworks with appropriate metadata and signing. For details on how to perform this post-processing, see the guide for *adding Python to your project*.

To help Python discover binaries in their new location, the original `.so` file on `sys.path` is replaced with a `.fwork` file. This file is a text file containing the location of the framework binary, relative to the app bundle. To allow the framework to resolve back to the original location, the framework must contain a `.origin` file that contains the location of the `.fwork` file, relative to the app bundle.

For example, consider the case of an import `from foo.bar import _whiz`, where `_whiz` is implemented with the binary module `sources/foo/bar/_whiz.abi3.so`, with `sources` being the location registered on `sys.path`, relative to the application bundle. This module *must* be distributed as `Frameworks/foo.bar._whiz.framework/foo.bar._whiz` (creating the framework name from the full import path of the module), with an `Info.plist` file in the `.framework` directory identifying the binary as a framework. The `foo.bar._whiz` module would be represented in the original location with a `sources/foo/bar/_whiz.abi3.fwork` marker file, containing the path `Frameworks/foo.bar._whiz/foo.bar._whiz`. The framework would also contain `Frameworks/foo.bar._whiz.framework/foo.bar._whiz.origin`, containing the path to the `.fwork` file.

When running on iOS, the Python interpreter will install an `AppleFrameworkLoader` that is able to read and import `.fwork` files. Once imported, the `__file__` attribute of the binary module will report as the location of the `.fwork` file. However, the `ModuleSpec` for the loaded module will report the `origin` as the location of the binary in the framework folder.

7.1.5 Compiler stub binaries

Xcode doesn't expose explicit compilers for iOS ; instead, it uses an `xcrun` script that resolves to a full compiler path (e.g., `xcrun --sdk iphoneos clang` to get the `clang` for an iPhone device). However, using this script poses two problems :

- The output of `xcrun` includes paths that are machine specific, resulting in a sysconfig module that cannot be shared between users ; and
- It results in `CC`/`CPP`/`LD`/`AR` definitions that include spaces. There is a lot of C ecosystem tooling that assumes that you can split a command line at the first space to get the path to the compiler executable ; this isn't the case when using `xcrun`.

To avoid these problems, Python provided stubs for these tools. These stubs are shell script wrappers around the underingly `xcrun` tools, distributed in a `bin` folder distributed alongside the compiled iOS framework. These scripts are relocatable, and will always resolve to the appropriate local system paths. By including these scripts in the bin folder that accompanies a framework, the contents of the `sysconfig` module becomes useful for end-users to compile their own modules. When compiling third-party Python modules for iOS, you should ensure these stub binaries are on your path.

7.2 Installing Python on iOS

7.2.1 Tools for building iOS apps

Building for iOS requires the use of Apple's Xcode tooling. It is strongly recommended that you use the most recent stable release of Xcode. This will require the use of the most (or second-most) recently released macOS version, as Apple does not maintain Xcode for older macOS versions. The Xcode Command Line Tools are not sufficient for iOS development ; you need a *full* Xcode install.

If you want to run your code on the iOS simulator, you'll also need to install an iOS Simulator Platform. You should be prompted to select an iOS Simulator Platform when you first run Xcode. Alternatively, you can add an iOS Simulator Platform by selecting from the Platforms tab of the Xcode Settings panel.

7.2.2 Adding Python to an iOS project

Python can be added to any iOS project, using either Swift or Objective C. The following examples will use Objective C ; if you are using Swift, you may find a library like PythonKit to be helpful.

To add Python to an iOS Xcode project :

1. Build or obtain a Python `XCFramework`. See the instructions in iOS/README.rst (in the CPython source distribution) for details on how to build a Python `XCFramework`. At a minimum, you will need a build that supports `arm64-apple-ios`, plus one of either `arm64-apple-ios-simulator` or `x86_64-apple-ios-simulator`.
2. Drag the `XCframework` into your iOS project. In the following instructions, we'll assume you've dropped the `XCframework` into the root of your project ; however, you can use any other location that you want by adjusting paths as needed.
3. Drag the `iOS/Resources/dylib-Info-template.plist` file into your project, and ensure it is associated with the app target.
4. Add your application code as a folder in your Xcode project. In the following instructions, we'll assume that your user code is in a folder named `app` in the root of your project ; you can use any other location by adjusting paths as needed. Ensure that this folder is associated with your app target.
5. Select the app target by selecting the root node of your Xcode project, then the target name in the sidebar that appears.
6. In the "General" settings, under "Frameworks, Libraries and Embedded Content", add `Python.xcframework`, with "Embed & Sign" selected.
7. In the "Build Settings" tab, modify the following :
   - Build Options
     - User Script Sandboxing : No
     - Enable Testability : Yes
   - Search Paths
     - Framework Search Paths : `$(PROJECT_DIR)`
     - Header Search Paths : `"$(BUILT_PRODUCTS_DIR)/Python.framework/Headers"`
   - Apple Clang - Warnings - All languages
     - Quoted Include In Framework Header : No
8. Add a build step that copies the Python standard library into your app. In the "Build Phases" tab, add a new "Run Script" build step *before* the "Embed Frameworks" step, but *after* the "Copy Bundle Resources" step. Name the step "Install Target Specific Python Standard Library", disable the "Based on dependency analysis" checkbox, and set the script content to :

```
set -e

mkdir -p "$CODESIGNING_FOLDER_PATH/python/lib"
if [ "$EFFECTIVE_PLATFORM_NAME" = "-iphonesimulator" ]; then
    echo "Installing Python modules for iOS Simulator"
    rsync -au --delete "$PROJECT_DIR/Python.xcframework/ios-arm64_x86_64-
↪simulator/lib/" "$CODESIGNING_FOLDER_PATH/python/lib/"
else
```

(suite sur la page suivante)

(suite de la page précédente)

```
    echo "Installing Python modules for iOS Device"
    rsync -au --delete "$PROJECT_DIR/Python.xcframework/ios-arm64/lib/" "
↪$CODESIGNING_FOLDER_PATH/python/lib/"
fi
```

Note that the name of the simulator "slice" in the XCframework may be different, depending the CPU architectures your `XCFramework` supports.

9. Add a second build step that processes the binary extension modules in the standard library into "Framework" format. Add a "Run Script" build step *directly after* the one you added in step 8, named "Prepare Python Binary Modules". It should also have "Based on dependency analysis" unchecked, with the following script content :

```
set -e

install_dylib () {
    INSTALL_BASE=$1
    FULL_EXT=$2

    # The name of the extension file
    EXT=$(basename "$FULL_EXT")
    # The location of the extension file, relative to the bundle
    RELATIVE_EXT=${FULL_EXT#$CODESIGNING_FOLDER_PATH/}
    # The path to the extension file, relative to the install base
    PYTHON_EXT=${RELATIVE_EXT/$INSTALL_BASE/}
    # The full dotted name of the extension module, constructed from the file␣
↪path.
    FULL_MODULE_NAME=$(echo $PYTHON_EXT | cut -d "." -f 1 | tr "/" ".");
    # A bundle identifier; not actually used, but required by Xcode framework␣
↪packaging
    FRAMEWORK_BUNDLE_ID=$(echo $PRODUCT_BUNDLE_IDENTIFIER.$FULL_MODULE_NAME |␣
↪tr "_" "-")
    # The name of the framework folder.
    FRAMEWORK_FOLDER="Frameworks/$FULL_MODULE_NAME.framework"

    # If the framework folder doesn't exist, create it.
    if [ ! -d "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER" ]; then
        echo "Creating framework for $RELATIVE_EXT"
        mkdir -p "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER"
        cp "$CODESIGNING_FOLDER_PATH/dylib-Info-template.plist" "$CODESIGNING_
↪FOLDER_PATH/$FRAMEWORK_FOLDER/Info.plist"
        plutil -replace CFBundleExecutable -string "$FULL_MODULE_NAME" "
↪$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER/Info.plist"
        plutil -replace CFBundleIdentifier -string "$FRAMEWORK_BUNDLE_ID" "
↪$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER/Info.plist"
    fi

    echo "Installing binary for $FRAMEWORK_FOLDER/$FULL_MODULE_NAME"
    mv "$FULL_EXT" "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER/$FULL_MODULE_
↪NAME"
    # Create a placeholder .fwork file where the .so was
    echo "$FRAMEWORK_FOLDER/$FULL_MODULE_NAME" > ${FULL_EXT%.so}.fwork
    # Create a back reference to the .so file location in the framework
    echo "${RELATIVE_EXT%.so}.fwork" > "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_
↪FOLDER/$FULL_MODULE_NAME.origin"
 }

 PYTHON_VER=$(ls -1 "$CODESIGNING_FOLDER_PATH/python/lib")
```

(suite sur la page suivante)

(suite de la page précédente)

```
echo "Install Python $PYTHON_VER standard library extension modules..."
find "$CODESIGNING_FOLDER_PATH/python/lib/$PYTHON_VER/lib-dynload" -name "*.
↪so" | while read FULL_EXT; do
   install_dylib python/lib/$PYTHON_VER/lib-dynload/ "$FULL_EXT"
done

# Clean up dylib template
rm -f "$CODESIGNING_FOLDER_PATH/dylib-Info-template.plist"

echo "Signing frameworks as $EXPANDED_CODE_SIGN_IDENTITY_NAME ($EXPANDED_
↪CODE_SIGN_IDENTITY)..."
find "$CODESIGNING_FOLDER_PATH/Frameworks" -name "*.framework" -exec /usr/
↪bin/codesign --force --sign "$EXPANDED_CODE_SIGN_IDENTITY" ${OTHER_CODE_
↪SIGN_FLAGS:-} -o runtime --timestamp=none --preserve-metadata=identifier,
↪entitlements,flags --generate-entitlement-der "{}" \;
```

10. Add Objective C code to initialize and use a Python interpreter in embedded mode. You should ensure that :

    — UTF-8 mode (`PyPreConfig.utf8_mode`) is *enabled* ;
    — Buffered stdio (`PyConfig.buffered_stdio`) is *disabled* ;
    — Writing bytecode (`PyConfig.write_bytecode`) is *disabled* ;
    — Signal handlers (`PyConfig.install_signal_handlers`) are *enabled* ;
    — System logging (`PyConfig.use_system_logger`) is *enabled* (optional, but strongly recommended) ;
    — `PYTHONHOME` for the interpreter is configured to point at the `python` subfolder of your app's bundle ; and
    — The `PYTHONPATH` for the interpreter includes :
      — the `python/lib/python3.X` subfolder of your app's bundle,
      — the `python/lib/python3.X/lib-dynload` subfolder of your app's bundle, and
      — the `app` subfolder of your app's bundle

    Your app's bundle location can be determined using `[[NSBundle mainBundle] resourcePath]`.

Steps 8, 9 and 10 of these instructions assume that you have a single folder of pure Python application code, named `app`. If you have third-party binary modules in your app, some additional steps will be required :

— You need to ensure that any folders containing third-party binaries are either associated with the app target, or copied in as part of step 8. Step 8 should also purge any binaries that are not appropriate for the platform a specific build is targeting (i.e., delete any device binaries if you're building an app targeting the simulator).
— Any folders that contain third-party binaries must be processed into framework form by step 9. The invocation of `install_dylib` that processes the `lib-dynload` folder can be copied and adapted for this purpose.
— If you're using a separate folder for third-party packages, ensure that folder is included as part of the `PYTHONPATH` configuration in step 10.

7.2.3 Testing a Python package

The CPython source tree contains a testbed project that is used to run the CPython test suite on the iOS simulator. This testbed can also be used as a testbed project for running your Python library's test suite on iOS.

After building or obtaining an iOS XCFramework (See iOS/README.rst for details), create a clone of the Python iOS testbed project by running :

```
$ python iOS/testbed clone --framework <path/to/Python.xcframework> --app <path/to/
↪module1> --app <path/to/module2> app-testbed
```

You will need to modify the `iOS/testbed` reference to point to that directory in the CPython source tree ; any folders specified with the `--app` flag will be copied into the cloned testbed project. The resulting testbed will be created in the `app-testbed` folder. In this example, the `module1` and `module2` would be importable modules at runtime. If your project has additional dependencies, they can be installed into the `app-testbed/iOSTestbed/app_packages` folder (using `pip install --target app-testbed/iOSTestbed/app_packages` or similar).

You can then use the `app-testbed` folder to run the test suite for your app, For example, if `module1.tests` was the entry point to your test suite, you could run :

```
$ python app-testbed run -- module1.tests
```

This is the equivalent of running `python -m module1.tests` on a desktop Python build. Any arguments after the `--` will be passed to the testbed as if they were arguments to `python -m` on a desktop machine.

You can also open the testbed project in Xcode by running :

```
$ open app-testbed/iOSTestbed.xcodeproj
```

This will allow you to use the full Xcode suite of tools for debugging.

7.3 App Store Compliance

The only mechanism for distributing apps to third-party iOS devices is to submit the app to the iOS App Store ; apps submitted for distribution must pass Apple's app review process. This process includes a set of automated validation rules that inspect the submitted application bundle for problematic code.

The Python standard library contains some code that is known to violate these automated rules. While these violations appear to be false positives, Apple's review rules cannot be challenged ; so, it is necessary to modify the Python standard library for an app to pass App Store review.

The Python source tree contains a patch file that will remove all code that is known to cause issues with the App Store review process. This patch is applied automatically when building for iOS.

CHAPITRE 8

Éditeurs et IDEs

Il y a un grand nombre d'environnement de développement qui gèrent le langage de programmation Python. Beaucoup d'éditeurs et d'EDIs proposent une mise en surbrillance de la syntaxe, des outils de débogage et une vérification PEP-8.

8.1 IDLE --- Python editor and shell

IDLE is Python's Integrated Development and Learning Environment and is generally bundled with Python installs. If you are on Linux and do not have IDLE installed see *Installing IDLE on Linux*. For more information see the IDLE docs.

8.2 Other Editors and IDEs

Python's community wiki has information submitted by the community on Editors and IDEs. Please go to Python Editors and Integrated Development Environments for a comprehensive list.

ANNEXE A

Glossaire

>>>

The default Python prompt of the *interactive* shell. Often seen for code examples which can be executed interactively in the interpreter.

...

Peut faire référence à :

— The default Python prompt of the *interactive* shell when entering the code for an indented code block, when within a pair of matching left and right delimiters (parentheses, square brackets, curly braces or triple quotes), or after specifying a decorator.

— La constante `Ellipsis`.

classe mère abstraite

Les classes mères 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`.

annotate function

A function that can be called to retrieve the *annotations* of an object. This function is accessible as the `__annotate__` attribute of functions, classes, and modules. Annotate functions are a subset of *evaluate functions*.

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*.

Annotations of local variables cannot be accessed at runtime, but annotations of global variables, class attributes, and functions can be retrieved by calling `annotationlib.get_annotations()` on modules, classes, and functions, respectively.

See *variable annotation*, *function annotation*, **PEP 484**, **PEP 526**, and **PEP 649**, which describe this functionality. Also see annotations-howto for best practices on working with annotations.

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 *paramètre* 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 `async with` en définissant les méthodes `__aenter__()` et `__aexit__()`. A été Introduit par la **PEP 492**.

générateur asynchrone
Fonction qui renvoie un *itérateur de générateur asynchrone*. 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`.

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

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

itérateur de générateur asynchrone
Objet créé par un *générateur asynchrone*.

C'est un *asynchronous iterator* qui, lorsqu'il est appelé via la méthode `__anext__()` renvoie un objet *awaitable* qui exécute le corps de la fonction du générateur asynchrone jusqu'au prochain `yield`.

Each `yield` temporarily suspends processing, remembering the execution state (including local variables and pending try-statements). When the *asynchronous generator iterator* effectively resumes with another awaitable returned by `__anext__()`, it picks up where it left off. See **PEP 492** and **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
Objet qui implémente les méthodes `__aiter__()` et `__anext__()`. `__anext__()` doit renvoyer un objet *awaitable*. Tant que la méthode `__anext__()` produit des objets *awaitable*, le `async for` appelant les consomme. L'itérateur asynchrone lève une exception `StopAsyncIteration` pour signifier la fin de l'itération. A été introduit par la **PEP 492**.

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

Il est possible de donner à un objet un attribut dont le nom n'est pas un identifiant tel que défini pour les identifiers, par exemple en utilisant `setattr()`, si l'objet le permet. Un tel attribut ne sera pas accessible à l'aide d'une expression pointée et on devra y accéder avec `getattr()`.

attendable (*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

A *file object* able to read and write *bytes-like objects*. Examples of binary files are files opened in binary mode (`'rb'`, `'wb'` or `'rb+'`), `sys.stdin.buffer`, `sys.stdout.buffer`, and instances of `io.BytesIO` and `gzip.GzipFile`.

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

référence empruntée

In Python's C API, a borrowed reference is a reference to an object, where the code using the object does not own the reference. It becomes a dangling pointer if the object is destroyed. For example, a garbage collection can remove the last *strong reference* to the object and so destroy it.

Il est recommandé d'appeler `Py_INCREF()` sur la *référence empruntée*, ce qui la transforme *in situ* en une *référence forte*. Vous pouvez faire une exception si vous êtes certain que l'objet ne peut pas être supprimé avant la dernière utilisation de la référence empruntée. Voir aussi la fonction `Py_NewRef()`, qui crée une nouvelle *référence forte*.

objet octet-compatible

Un objet gérant le protocole tampon et pouvant exporter un tampon (*buffer* en anglais) C-*contigu*. Cela inclut les objets `bytes`, `bytearray` et `array.array`, ainsi que beaucoup d'objets `memoryview`. Les objets octets-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 (« *objets octets-compatibles en lecture seule* »), par exemple des `bytes` ou des `memoryview` d'un objet `bytes`.

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.

appelable (*callable*)

Un appelable est un objet qui peut être appelé, éventuellement avec un ensemble d'arguments (voir *argument*), avec la syntaxe suivante :

```
callable(argument1, argument2, argumentN)
```

Une *fonction*, et par extension une *méthode*, est un appelable. Une instance d'une classe qui implémente la méthode `__call__()` est également un appelable.

fonction de rappel (*callback*)

Une fonction (classique, par opposition à une coroutine) passée en argument pour être exécutée plus tard.

classe

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

variable de classe

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

closure variable

A *free variable* referenced from a *nested scope* that is defined in an outer scope rather than being resolved at runtime from the globals or builtin namespaces. May be explicitly defined with the `nonlocal` keyword to allow write access, or implicitly defined if the variable is only being read.

For example, in the `inner` function in the following code, both `x` and `print` are *free variables*, but only `x` is a *closure variable* :

```
def outer():
    x = 0
    def inner():
        nonlocal x
        x += 1
        print(x)
    return inner
```

Due to the `codeobject.co_freevars` attribute (which, despite its name, only includes the names of closure variables rather than listing all referenced free variables), the more general *free variable* term is sometimes used even when the intended meaning is to refer specifically to closure variables.

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.

context
This term has different meanings depending on where and how it is used. Some common meanings :
— The temporary state or environment established by a *context manager* via a `with` statement.
— The collection of keyvalue bindings associated with a particular `contextvars.Context` object and accessed via `ContextVar` objects. Also see *context variable*.
— A `contextvars.Context` object. Also see *current context*.

context management protocol
The `__enter__()` and `__exit__()` methods called by the `with` statement. See **PEP 343**.

gestionnaire de contexte
An object which implements the *context management protocol* and controls the environment seen in a `with` statement. See **PEP 343**.

variable de contexte
A variable whose value depends on which context is the *current context*. Values are accessed via `contextvars.ContextVar` objects. Context variables are primarily used to isolate state between concurrent asynchronous tasks.

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

coroutine
Les coroutines sont une forme généralisée des fonctions. On entre dans une fonction en un point et on en sort en un autre point. On peut entrer, sortir et reprendre l'exécution d'une coroutine en plusieurs points. Elles peuvent être implémentées en utilisant l'instruction `async def`. 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*.

current context
The *context* (`contextvars.Context` object) that is currently used by `ContextVar` objects to access (get or set) the values of *context variables*. Each thread has its own current context. Frameworks for executing asynchronous tasks (see `asyncio`) associate each task with a context which becomes the current context whenever the task starts or resumes execution.

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(arg):
    ...
f = staticmethod(f)

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

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

Any object which defines the methods `__get__()`, `__set__()`, or `__delete__()`. When a class attribute is a descriptor, its special binding behavior is triggered upon attribute lookup. Normally, using *a.b* to get, set or delete an attribute looks up the object named *b* in the class dictionary for *a*, but if *b* is a descriptor, the respective descriptor method gets called. Understanding descriptors is a key to a deep understanding of Python because they are the basis for many features including functions, methods, properties, class methods, static methods, and reference to super classes.

Pour plus d'informations sur les méthodes des descripteurs, consultez descriptors ou le guide pour l'utilisation des descripteurs.

dictionnaire

An associative array, where arbitrary keys are mapped to values. The keys can be any object with `__hash__()` and `__eq__()` methods. Called a hash in Perl.

dictionnaire en compréhension (ou dictionnaire en intension)

Écriture concise pour traiter tout ou partie des éléments d'un itérable et renvoyer un dictionnaire contenant les résultats. `results = {n: n ** 2 for n in range(10)}` génère un dictionnaire contenant des clés `n` liées à leurs valeurs `n ** 2`. Voir compréhensions.

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.

chaîne de documentation (*docstring*)

A string literal which appears as the first expression in a class, function or module. While ignored when the suite is executed, it is recognized by the compiler and put into the `__doc__` attribute of the enclosing class, function or module. Since it is available via introspection, it is the canonical place for documentation of the object.

typage canard (*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 mère 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.

evaluate function
A function that can be called to evaluate a lazily evaluated attribute of an object, such as the value of type aliases created with the `type` statement.

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 `while`. Les affectations sont également des instructions et non des expressions.

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

f-string
Chaîne littérale préfixée de `'f'` ou `'F'`. Les "f-strings" sont un raccourci pour formatted string literals. Voir la **PEP 498**.

objet fichier
An object exposing a file-oriented API (with methods such as `read()` or `write()`) to an underlying resource. Depending on the way it was created, a file object can mediate access to a real on-disk file or to another type of storage or communication device (for example standard input/output, in-memory buffers, sockets, pipes, etc.). File objects are also called *file-like objects* or *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*.

encodage du système de fichiers et gestionnaire d'erreurs associé
Encodage et gestionnaire d'erreurs utilisés par Python pour décoder les octets fournis par le système d'exploitation et encoder les chaînes de caractères Unicode afin de les passer au système.

L'encodage du système de fichiers doit impérativement pouvoir décoder tous les octets jusqu'à 128. Si ce n'est pas le cas, certaines fonctions de l'API lèvent `UnicodeError`.

Cet encodage et son gestionnaire d'erreur peuvent être obtenus à l'aide des fonctions `sys.getfilesystemencoding()` et `sys.getfilesystemencodeerrors()`.

L'*encodage du système de fichiers et gestionnaire d'erreurs associé* sont configurés au démarrage de Python par la fonction `PyConfig_Read()` : regardez `filesystem_encoding` et `filesystem_errors` dans les membres de `PyConfig`.

Voir aussi *encodage régional*.

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

There are two types of finder : *meta path finders* for use with `sys.meta_path`, and *path entry finders* for use with `sys.path_hooks`.

See finders-and-loaders and `importlib` for much more detail.

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**.

free threading
A threading model where multiple threads can run Python bytecode simultaneously within the same interpreter. This is in contrast to the *global interpreter lock* which allows only one thread to execute Python bytecode at a time. See **PEP 703**.

free variable
Formally, as defined in the language execution model, a free variable is any variable used in a namespace which is not a local variable in that namespace. See *closure variable* for an example. Pragmatically, due to the name of the `codeobject.co_freevars` attribute, the term is also sometimes used as a synonym for *closure variable*.

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.

Les annotations de fonctions sont généralement utilisées pour des *indications de types* : par exemple, cette fonction devrait prendre deux arguments `int` et devrait également avoir une valeur de retour de type `int` :

```
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 *annotation de variable* et la **PEP 484**, qui décrivent cette fonctionnalité. Voir aussi annotations-howto sur les bonnes pratiques concernant les annotations.

__future__
Une importation depuis le futur s'écrit `from __future__ import <fonctionnalité>`. Lorsqu'une importation du futur est active dans un module, Python compile ce module avec une certaine modification de la syntaxe ou du comportement qui est vouée à devenir standard dans une version ultérieure. Le module `__future__` documente les possibilités pour *fonctionnalité*. L'importation a aussi l'effet normal d'importer une variable du module. Cette variable contient des informations utiles sur la fonctionnalité en question, notamment la version de Python dans laquelle elle a été ajoutée, et celle dans laquelle elle deviendra standard :

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

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

Each `yield` temporarily suspends processing, remembering the execution state (including local variables and pending try-statements). When the *generator iterator* resumes, it picks up where it left off (in contrast to functions which start fresh on every invocation).

expression génératrice
An *expression* that returns an *iterator*. It looks like a normal expression followed by a `for` clause defining a loop variable, range, and an optional `if` clause. The combined expression generates values for an enclosing function :

```
>>> 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**.

type générique
Un *type* qui peut être paramétré ; généralement un conteneur comme `list` ou `dict`. Utilisé pour les *indications de type* et les *annotations*.

Pour plus de détails, voir types alias génériques et le module `typing`. On trouvera l'historique de cette fonctionnalité dans les **PEP 483**, **PEP 484** et **PEP 585**.

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.

As of Python 3.13, the GIL can be disabled using the `--disable-gil` build configuration. After building Python with this option, code must be run with `-X gil=0` or after setting the `PYTHON_GIL=0` environment variable. This feature enables improved performance for multi-threaded applications and makes it easier to use multi-core CPUs efficiently. For more details, see **PEP 703**.

***pyc* utilisant le hachage**

Un fichier de cache de code intermédiaire (*bytecode* en anglais) qui utilise le hachage plutôt que l'heure de dernière modification du fichier source correspondant pour déterminer sa validité. Voir pyc-invalidation.

hachable

An object is *hashable* if it has a hash value which never changes during its lifetime (it needs a `__hash__()` method), and can be compared to other objects (it needs an `__eq__()` method). Hashable objects which compare equal must have the same hash value.

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

La plupart des types immuables natifs de Python sont hachables, mais les conteneurs mutables (comme les listes ou les dictionnaires) ne le sont pas ; les conteneurs immuables (comme les n-uplets ou les ensembles figés) ne sont hachables que si leurs éléments sont hachables. Les instances de classes définies par les utilisateurs sont hachables par défaut. Elles sont toutes considérées différentes (sauf avec elles-mêmes) et leur valeur de hachage est calculée à partir de leur `id()`.

IDLE

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

immortal

Immortal objects are a CPython implementation detail introduced in **PEP 683**.

If an object is immortal, its *reference count* is never modified, and therefore it is never deallocated while the interpreter is running. For example, `True` and `None` are immortal in CPython.

Immortal objects can be identified via `sys._is_immortal()`, or via `PyUnstable_IsImmortal()` in the C API.

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.

importation

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 has an interactive interpreter which means you can enter statements and expressions at the interpreter prompt, immediately execute them and see their results. Just launch `python` with no arguments (possibly by selecting it from your computer's main menu). It is a very powerful way to test out new ideas or inspect modules and packages (remember `help(x)`). For more on interactive mode, see tut-interac.

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

An object capable of returning its members one at a time. Examples of iterables include all sequence types (such as `list`, `str`, and `tuple`) and some non-sequence types like `dict`, *file objects*, and objects of any classes you define with an `__iter__()` method or with a `__getitem__()` method that implements *sequence* semantics.

Iterables can be used in a `for` loop and in many other places where a sequence is needed (`zip()`, `map()`, ...). When an iterable object is passed as an argument to the built-in function `iter()`, it returns an iterator for the object. This iterator is good for one pass over the set of values. When using iterables, it is usually not necessary to call `iter()` or deal with iterator objects yourself. The `for` statement does that automatically for you, creating a temporary unnamed variable to hold the iterator for the duration of the loop. See also *iterator*, *sequence*, and *generator*.

itérateur

An object representing a stream of data. Repeated calls to the iterator's `__next__()` method (or passing it to the built-in function `next()`) return successive items in the stream. When no more data are available a `StopIteration` exception is raised instead. At this point, the iterator object is exhausted and any further calls to its `__next__()` method just raise `StopIteration` again. Iterators are required to have an `__iter__()` method that returns the iterator object itself so every iterator is also iterable and may be used in most places where other iterables are accepted. One notable exception is code which attempts multiple iteration passes. A container object (such as a `list`) produces a fresh new iterator each time you pass it to the `iter()` function or use it in a `for` loop. Attempting this with an iterator will just return the same exhausted iterator object used in the previous iteration pass, making it appear like an empty container.

Vous trouverez davantage d'informations dans typeiter.

Particularité de l'implémentation CPython : CPython does not consistently apply the requirement that an iterator define `__iter__()`. And also please note that the free-threading CPython does not guarantee the thread-safety of iterator operations.

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])`. Par ailleurs `attrgetter()`, `itemgetter()` et `methodcaller()` permettent de créer des fonctions clés. Voir le guide pour le tri 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.

liste
A built-in Python *sequence*. Despite its name it is more akin to an array in other languages than to a linked list since access to elements is $O(1)$.

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
An object that loads a module. It must define the `exec_module()` and `create_module()` methods to implement the `Loader` interface. A loader is typically returned by a *finder*. See also :
— finders-and-loaders
— `importlib.abc.Loader`
— **PEP 302**

encodage régional
Sous Unix, il est défini par la variable régionale LC_CTYPE. Il peut être modifié par `locale.setlocale(locale.LC_CTYPE, new_locale)`.

Sous Windows, c'est un encodage ANSI (par ex. : `"cp1252"`).

Sous Android et VxWorks, Python utilise `"utf-8"` comme encodage régional.

`locale.getencoding()` can be used to get the locale encoding.

Voir aussi l'*encodage du systèmes de fichiers et gestionnaire d'erreurs associé*.

méthode magique
Un synonyme informel de *special method*.

tableau de correspondances (*mapping* en anglais)
Conteneur permettant de rechercher des éléments à partir de clés et implémentant les méthodes spécifiées dans les classes mères abstraites des `tableaux de correspondances` (immuables) ou `tableaux de correspondances mutables` (voir les classes mères abstraites). 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
Method Resolution Order is the order in which base classes are searched for a member during lookup. See python_2.3_mro for details of the algorithm used by the Python interpreter since the 2.3 release.

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`.

See also module-specs.

MRO
Voir *ordre de résolution des méthodes*.

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

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

Plusieurs types natifs sont appelés n-uplets, y compris les valeurs retournées par `time.localtime()` et `os.stat()`. Un autre exemple est `sys.float_info` :

```
>>> sys.float_info[1]                    # indexed access
1024
>>> sys.float_info.max_exp               # named field access
1024
>>> isinstance(sys.float_info, tuple)    # kind of tuple
True
```

Some named tuples are built-in types (such as the above examples). Alternatively, a named tuple can be created from a regular class definition that inherits from `tuple` and that defines named fields. Such a class can be written by hand, or it can be created by inheriting `typing.NamedTuple`, or with the factory function `collections.namedtuple()`. The latter techniques also add some extra methods that may not be found in hand-written or built-in named tuples.

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
A *package* which serves only as a container for subpackages. Namespace packages may have no physical representation, and specifically are not like a *regular package* because they have no `__init__.py` file.

Namespace packages allow several individually installable packages to have a common parent package. Other-

wise, it is recommended to use a *regular package*.

For more information, see **PEP 420** and reference-namespace-package.

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
Old name for the flavor of classes now used for all class objects. In earlier Python versions, only new-style classes could use Python's newer, versatile features like `__slots__`, descriptors, properties, `__getattribute__()`, class methods, and static methods.

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*.

optimized scope
A scope where target local variable names are reliably known to the compiler when the code is compiled, allowing optimization of read and write access to these names. The local namespaces for functions, generators, coroutines, comprehensions, and generator expressions are optimized in this fashion. Note : most interpreter optimizations are applied to all scopes, only those relying on a known set of local and nonlocal variable names are restricted to optimized scopes.

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* : définit un argument qui ne peut être fourni que par position. Les paramètres *positional-only* peuvent être définis en insérant un caractère "/" dans la liste de paramètres de la définition de fonction après eux. Par exemple : *posonly1* et *posonly2* dans le code suivant :

```
def func(posonly1, posonly2, /, positional_or_keyword): ...
```

— *keyword-only* : l'argument ne peut être fourni que nommé. Les paramètres *keyword-only* peuvent être définis en utilisant un seul paramètre *var-positional*, ou en ajoutant une étoile (`*`) seule dans la liste des paramètres avant eux. Par exemple, *kw_only1* et *kw_only2* dans le code suivant :

```
def func(arg, *, kw_only1, kw_only2): ...
```

— *var-positional* : une séquence d'arguments positionnels peut être fournie (en plus de tous les arguments positionnels déjà acceptés par d'autres paramètres). Un tel paramètre peut être défini en préfixant son nom par une `*`. Par exemple *args* ci-après :

```
def func(*args, **kwargs): ...
```

— *var-keyword* : une quantité arbitraire d'arguments peut être passée, chacun étant nommé (en plus de tous les arguments nommés déjà acceptés par d'autres paramètres). Un tel paramètre est défini en préfixant le nom du paramètre par `**`. Par exemple, *kwargs* ci-dessus.

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

Voir aussi *argument* 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*
A callable on the `sys.path_hooks` list which returns a *path entry finder* if it knows how to find modules on a specific *path entry*.

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 de Python). Une 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 PEP sont censées 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 la **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

The number of references to an object. When the reference count of an object drops to zero, it is deallocated. Some objects are *immortal* and have reference counts that are never modified, and therefore the objects are never deallocated. Reference counting is generally not visible to Python code, but it is a key element of the *CPython* implementation. Programmers can call the `sys.getrefcount()` function to return the reference count for a particular object.

paquet classique

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

Voir aussi *paquet-espace de nommage*.

REPL

An acronym for the "read–eval–print loop", another name for the *interactive* interpreter shell.

__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

An *iterable* which supports efficient element access using integer indices via the `__getitem__()` special method and defines a `__len__()` method that returns the length of the sequence. Some built-in sequence types are `list`, `str`, `tuple`, and `bytes`. Note that `dict` also supports `__getitem__()` and `__len__()`,

but is considered a mapping rather than a sequence because the lookups use arbitrary *hashable* keys rather than integers.

The `collections.abc.Sequence` abstract base class defines a much richer interface that goes beyond just `__getitem__()` and `__len__()`, adding `count()`, `index()`, `__contains__()`, and `__reversed__()`. Types that implement this expanded interface can be registered explicitly using `register()`. For more documentation on sequence methods generally, see Common Sequence Operations.

ensemble en compréhension (ou ensemble en intension)
Une façon compacte de traiter tout ou partie des éléments d'un itérable et de renvoyer un *set* avec les résultats. `results = {c for c in 'abracadabra' if c not in 'abc'}` génère l'ensemble contenant les lettres « r » et « d » `{'r', 'd'}`. Voir comprehensions.

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.

soft deprecated
A soft deprecated API should not be used in new code, but it is safe for already existing code to use it. The API remains documented and tested, but will not be enhanced further.

Soft deprecation, unlike normal deprecation, does not plan on removing the API and will not emit warnings.

See PEP 387 : Soft Deprecation.

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`.

static type checker
An external tool that reads Python code and analyzes it, looking for issues such as incorrect types. See also *type hints* and the `typing` module.

référence forte
In Python's C API, a strong reference is a reference to an object which is owned by the code holding the reference. The strong reference is taken by calling `Py_INCREF()` when the reference is created and released with `Py_DECREF()` when the reference is deleted.

Une référence forte est créée à l'aide de la fonction `Py_NewRef()`. Il faut normalement appeler `Py_DECREF()` dessus avant de sortir de sa portée lexicale, sans quoi il y a une fuite de référence.

Voir aussi *référence empruntée*.

encodages de texte
Une chaîne de caractères en Python est une suite de points de code Unicode (dans l'intervalle `U+0000`--`U+10FFFF`). Pour stocker ou transmettre une chaîne, il est nécessaire de la sérialiser en suite d'octets.

Sérialiser une chaîne de caractères en une suite d'octets s'appelle « encoder » et recréer la chaîne à partir de la suite d'octets s'appelle « décoder ».

Il existe de multiples codecs pour la sérialisation de texte, que l'on regroupe sous l'expression « encodages de texte ».

fichier texte
Objet fichier 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'*encodage de texte* 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 *fichier binaire* pour un objet fichier capable de lire et d'écrire des *objets octets-compatibles*.

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

The type of a Python object determines what kind of object it is ; every object has a type. An object's type is accessible as its `__class__` attribute or can be retrieved with `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 :

```
def remove_gray_shades(
        colors: list[tuple[int, int, int]]) -> list[tuple[int, int, int]]:
    pass
```

pourrait être rendu plus lisible comme ceci :

```
Color = tuple[int, int, int]

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

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

indication de type

L'*annotation* qui spécifie le type attendu pour une variable, un attribut de classe, un paramètre de fonction ou une valeur de retour.

Type hints are optional and are not enforced by Python but they are useful to *static type checkers*. They can also aid IDEs with code completion and refactoring.

Les indications de type de variables globales, d'attributs de classe et de fonctions, mais pas de variables locales, peuvent être consultées en utilisant `typing.get_type_hints()`.

Voir `typing` et la **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 variable est expliquée dans la section annassign.

Reportez-vous à *annotation de fonction*, à la **PEP 484** et à la **PEP 526** qui décrivent cette fonctionnalité. Voir aussi annotations-howto sur les bonnes pratiques concernant les annotations.

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 *code intermédiaire* 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

About this documentation

Python's documentation is generated from reStructuredText sources using Sphinx, a documentation generator originally created for Python and now maintained as an independent project.

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., the creator of the original Python documentation toolset and author of much of the content ;
- 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 Contributors to the Python documentation

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 was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see https://www.cwi.nl) in the Netherlands as a successor of a language called ABC. Guido remains Python's principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see https://www.cnri.reston.va.us) in Reston, Virginia where he released several versions of the software.

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, which became Zope Corporation. 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 was a sponsoring member of the PSF.

All Python releases are Open Source (see https://opensource.org for the Open Source Definition). Historically, most, but not all, Python releases have also been GPL-compatible ; the table below summarizes the various releases.

Version	Dérivé de	Année	Propriétaire	GPL-compatible ? (1)
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	yes (2)
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

(1) GPL-compatible doesn't mean that we're distributing Python under the GPL. All Python licenses, unlike the GPL, let you distribute a modified version without making your changes open source. The GPL-compatible licenses make it possible to combine Python with other software that is released under the GPL ; the others don't.

(2) According to Richard Stallman, 1.6.1 is not GPL-compatible, because its license has a choice of law clause. According to CNRI, however, Stallman's lawyer has told CNRI's lawyer that 1.6.1 is "not incompatible" with the GPL.

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

Python software and documentation are licensed under the Python Software Foundation License Version 2.

Starting with Python 3.8.6, examples, recipes, and other code in the documentation are dual licensed under the PSF License Version 2 and the *Zero-Clause BSD license*.

Certains logiciels faisant partie de Python sont soumis à d'autres licences. Ces licences sont incluses avec le code lié à celles-ci. Voir *Licences et remerciements pour les logiciels tiers* pour une liste non exhaustive de ces licences.

C.2.1 PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2

```
1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and
   the Individual or Organization ("Licensee") accessing and otherwise using this
   software ("Python") 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 alone or in any derivative
   version, provided, however, that PSF's License Agreement and PSF's notice of
   copyright, i.e., "Copyright © 2001 Python Software Foundation; All Rights
   Reserved" are retained in Python 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 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.

4. PSF is making Python 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 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

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

(suite sur la page suivante)

(suite de la page précédente)

```
   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, 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
   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
```

(suite sur la page suivante)

(suite de la page précédente)

```
   ("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
   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.2.5 ZERO-CLAUSE BSD LICENSE FOR CODE IN THE PYTHON DOCUMENTATION

```
Permission to use, copy, modify, and/or distribute this software for any
purpose with or without fee is hereby granted.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH
REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT,
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

The `_random` C extension underlying the `random` module includes code based on a download from http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html. The following are the verbatim comments from the original code :

```
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).

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

Redistribution and use in source and binary forms, with or without
```

(suite sur la page suivante)

(suite de la page précédente)

```
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*)

The `socket` module uses the functions, `getaddrinfo()`, and `getnameinfo()`, which are coded in separate source files from the WIDE Project, https://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.

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
```

(suite sur la page suivante)

(suite de la page précédente)

```
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 Interfaces de connexion asynchrones

The `test.support.asynchat` and `test.support.asyncore` modules contain the following notice :

```
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.3.4 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
```

(suite sur la page suivante)

(suite de la page précédente)

```
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
```

C.3.5 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.3.6 Les fonctions UUencode et UUdecode

The `uu` codec contains the following notice :

```
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.
```

(suite sur la page suivante)

(suite de la page précédente)

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

C.3.7 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.3.8 test_epoll

The `test.test_epoll` module contains the following notice :

```
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.3.9 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.3.10 SipHash24

Le fichier `Python/pyhash.c` contient 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>

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

(suite sur la page suivante)

(suite de la page précédente)

```
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.3.11 *strtod* et *dtoa*

The file `Python/dtoa.c`, which supplies C functions dtoa and strtod for conversion of C doubles to and from strings, is derived from the file of the same name by David M. Gay, currently available from https://web.archive.org/web/20220517033456/http://www.netlib.org/fp/dtoa.c. The original file, as retrieved on March 16, 2009, contains the following copyright and licensing notice :

```
/****************************************************************
 *
 * 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.3.12 OpenSSL

The modules `hashlib`, `posix` and `ssl` use the OpenSSL library for added performance if made available by the operating system. Additionally, the Windows and macOS installers for Python may include a copy of the OpenSSL libraries, so we include a copy of the OpenSSL license here. For the OpenSSL 3.0 release, and later releases derived from that, the Apache License v2 applies :

```
                              Apache License
                        Version 2.0, January 2004
                     https://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
```

(suite sur la page suivante)

(suite de la page précédente)

```
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications,
   including but not limited to software source code, documentation
   source, and configuration files.

   "Object" form shall mean any form resulting from mechanical
   transformation or translation of a Source form, including but
   not limited to compiled object code, generated documentation,
   and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or
   Object form, made available under the License, as indicated by a
   copyright notice that is included in or attached to the work
   (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object
   form, that is based on (or derived from) the Work and for which the
   editorial revisions, annotations, elaborations, or other modifications
   represent, as a whole, an original work of authorship. For the purposes
   of this License, Derivative Works shall not include works that remain
   separable from, or merely link (or bind by name) to the interfaces of,
   the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including
   the original version of the Work and any modifications or additions
   to that Work or Derivative Works thereof, that is intentionally
   submitted to Licensor for inclusion in the Work by the copyright owner
   or by an individual or Legal Entity authorized to submit on behalf of
   the copyright owner. For the purposes of this definition, "submitted"
   means any form of electronic, verbal, or written communication sent
   to the Licensor or its representatives, including but not limited to
   communication on electronic mailing lists, source code control systems,
   and issue tracking systems that are managed by, or on behalf of, the
   Licensor for the purpose of discussing and improving the Work, but
   excluding communication that is conspicuously marked or otherwise
   designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity
   on behalf of whom a Contribution has been received by Licensor and
   subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   copyright license to reproduce, prepare Derivative Works of,
   publicly display, publicly perform, sublicense, and distribute the
   Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of
   this License, each Contributor hereby grants to You a perpetual,
```

(suite sur la page suivante)

(suite de la page précédente)

```
   worldwide, non-exclusive, no-charge, royalty-free, irrevocable
   (except as stated in this section) patent license to make, have made,
   use, offer to sell, sell, import, and otherwise transfer the Work,
   where such license applies only to those patent claims licensable
   by such Contributor that are necessarily infringed by their
   Contribution(s) alone or by combination of their Contribution(s)
   with the Work to which such Contribution(s) was submitted. If You
   institute patent litigation against any entity (including a
   cross-claim or counterclaim in a lawsuit) alleging that the Work
   or a Contribution incorporated within the Work constitutes direct
   or contributory patent infringement, then any patent licenses
   granted to You under this License for that Work shall terminate
   as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the
   Work or Derivative Works thereof in any medium, with or without
   modifications, and in Source or Object form, provided that You
   meet the following conditions:

   (a) You must give any other recipients of the Work or
       Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices
       stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works
       that You distribute, all copyright, patent, trademark, and
       attribution notices from the Source form of the Work,
       excluding those notices that do not pertain to any part of
       the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its
       distribution, then any Derivative Works that You distribute must
       include a readable copy of the attribution notices contained
       within such NOTICE file, excluding those notices that do not
       pertain to any part of the Derivative Works, in at least one
       of the following places: within a NOTICE text file distributed
       as part of the Derivative Works; within the Source form or
       documentation, if provided along with the Derivative Works; or,
       within a display generated by the Derivative Works, if and
       wherever such third-party notices normally appear. The contents
       of the NOTICE file are for informational purposes only and
       do not modify the License. You may add Your own attribution
       notices within Derivative Works that You distribute, alongside
       or as an addendum to the NOTICE text from the Work, provided
       that such additional attribution notices cannot be construed
       as modifying the License.

   You may add Your own copyright statement to Your modifications and
   may provide additional or different license terms and conditions
   for use, reproduction, or distribution of Your modifications, or
   for any such Derivative Works as a whole, provided Your use,
   reproduction, and distribution of the Work otherwise complies with
   the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
```

(suite sur la page suivante)

(suite de la page précédente)

```
   any Contribution intentionally submitted for inclusion in the Work
   by You to the Licensor shall be under the terms and conditions of
   this License, without any additional terms or conditions.
   Notwithstanding the above, nothing herein shall supersede or modify
   the terms of any separate license agreement you may have executed
   with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
   names, trademarks, service marks, or product names of the Licensor,
   except as required for reasonable and customary use in describing the
   origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
   agreed to in writing, Licensor provides the Work (and each
   Contributor provides its Contributions) on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
```

C.3.13 expat

The `pyexpat` extension is built using an included copy of the expat sources unless the build is configured `--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
```

(suite sur la page suivante)

(suite de la page précédente)

```
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.3.14 libffi

The `_ctypes` C extension underlying the `ctypes` module is built using an included copy of the libffi sources unless the build is configured `--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.3.15 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.
```

(suite sur la page suivante)

(suite de la page précédente)

```
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.3.16 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
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.3.17 libmpdec

The `_decimal` C extension underlying the `decimal` module is built using an included copy of the libmpdec library unless the build is configured `--with-system-libmpdec` :

```
Copyright (c) 2008-2020 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.
```

C.3.18 Ensemble de tests C14N du W3C

Les tests de C14N version 2.0 du module `test` (`Lib/test/xmltestdata/c14n-20/`) proviennent du site du W3C à l'adresse https://www.w3.org/TR/xml-c14n2-testcases/ et sont distribués sous licence BSD modifiée :

```
Copyright (c) 2013 W3C(R) (MIT, ERCIM, Keio, Beihang),
All Rights Reserved.

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

* Redistributions of works must retain the original copyright notice,
  this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the original 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 W3C nor the names of its contributors may be
  used to endorse or promote products derived from this work 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,
```

(suite sur la page suivante)

(suite de la page précédente)

```
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.3.19 mimalloc

MIT License :

```
Copyright (c) 2018-2021 Microsoft Corporation, Daan Leijen

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.3.20 *asyncio*

Parts of the `asyncio` module are incorporated from uvloop 0.16, which is distributed under the MIT license :

```
Copyright (c) 2015-2021 MagicStack Inc.  http://magic.io

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.3.21 Global Unbounded Sequences (GUS)

The file `Python/qsbr.c` is adapted from FreeBSD's "Global Unbounded Sequences" safe memory reclamation scheme in subr_smr.c. The file is distributed under the 2-Clause BSD License :


```
Copyright (c) 2019,2020 Jeffrey Roberson <jeff@FreeBSD.org>

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 unmodified, 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 "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 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 :

Copyright © 2001 Python Software Foundation. All rights reserved.

Copyright © 2000 *BeOpen.com*. Tous droits réservés.

Copyright © 1995-2000 *Corporation for National Research Initiatives*. Tous droits réservés.

Copyright © 1991-1995 *Stichting Mathematisch Centrum*. Tous droits réservés.

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

Index

Non alphabétique

...., **87**
-?
    option de ligne de commande, 5
%APPDATA%, 49
>>>, **87**
__future__, **93**
__slots__, **100**

A

alias de type, **102**
annotate function, **87**
annotation, **87**
annotation de fonction, **93**
annotation de variable, **102**
API provisoire, **99**
appelable (*callable*), **89**
argument, **87**
argument nommé, **96**
argument positionnel, **99**
arrêt de l'interpréteur, **95**
attendable (*awaitable*), **88**
attribut, **88**

B

-B
    option de ligne de commande, 6
-b
    option de ligne de commande, 6
BDFL, **88**
BOLT_APPLY_FLAGS
    option de ligne de commande, 30
BOLT_INSTRUMENT_FLAGS
    option de ligne de commande, 30
--build
    option de ligne de commande, 35
BZIP2_CFLAGS
    option de ligne de commande, 27
BZIP2_LIBS
    option de ligne de commande, 27

C

-c
    option de ligne de commande, 3
CC
    option de ligne de commande, 27
C-contiguous, 90
CFLAGS, 29, 3840
    option de ligne de commande, 27
CFLAGS_NODIST, 3840
chaîne de documentation (*docstring*), **91**
chaîne entre triple guillemets, **101**
chargeur, **96**
--check-hash-based-pycs
    option de ligne de commande, 6
chemin des importations, **94**
chercheur, **92**
chercheur basé sur les chemins, **99**
chercheur dans les méta-chemins, **96**
chercheur de chemins, **99**
classe, **89**
classe mère abstraite, **87**
closure variable, **89**
code intermédiaire (*bytecode*), **89**
CONFIG_SITE
    option de ligne de commande, 35
context, **90**
context management protocol, **90**
contigu, **90**
coroutine, **90**
CPP
    option de ligne de commande, 27
CPPFLAGS, 38, 40
    option de ligne de commande, 27
CPython, **90**
current context, **90**
CURSES_CFLAGS
    option de ligne de commande, 27
CURSES_LIBS
    option de ligne de commande, 27

D

-d
    option de ligne de commande, 6
décorateur, **91**
descripteur, **91**
dictionnaire, **91**

dictionnaire en compréhension (*ou dictionnaire en intension*), **91**
--disable-gil
    option de ligne de commande, 26
--disable-ipv6
    option de ligne de commande, 24
--disable-safety
    option de ligne de commande, 34
--disable-test-modules
    option de ligne de commande, 29
distribution simple, **101**
division entière, **92**

E

-E
    option de ligne de commande, 6
EAFP, **91**
--enable-big-digits
    option de ligne de commande, 24
--enable-bolt
    option de ligne de commande, 30
--enable-experimental-jit
    option de ligne de commande, 26
--enable-framework
    option de ligne de commande, 34, 35
--enable-loadable-sqlite-extensions
    option de ligne de commande, 24
--enable-optimizations
    option de ligne de commande, 29
--enable-profiling
    option de ligne de commande, 31
--enable-pystats
    option de ligne de commande, 25
--enable-shared
    option de ligne de commande, 32
--enable-slower-safety
    option de ligne de commande, 34
--enable-universalsdk
    option de ligne de commande, 34
--enable-wasm-dynamic-linking
    option de ligne de commande, 28
--enable-wasm-pthreads
    option de ligne de commande, 29
encodage du système de fichiers et gestionnaire d'erreurs associé, **92**
encodage régional, **96**
encodages de texte, **101**
ensemble en compréhension (*ou ensemble en intension*), **101**
entrée de chemin, **99**
environnement virtuel, **102**
espace de nommage, **97**
evaluate function, **92**
--exec-prefix
    option de ligne de commande, 29
expression, **92**
expression génératrice, **93**

F

f-string, **92**
fichier binaire, **89**
fichier texte, **101**
fonction, **93**
fonction clé, **95**
fonction coroutine, **90**
fonction de rappel (*callback*), **89**
fonction générique, **93**
Fortran contiguous, 90
free threading, **92**
free variable, **92**

G

GDBM_CFLAGS
    option de ligne de commande, 27
GDBM_LIBS
    option de ligne de commande, 27
générateur, **93**
générateur asynchrone, **88**
gestionnaire de contexte, **90**
gestionnaire de contexte asynchrone, **88**
GIL, **94**

H

-h
    option de ligne de commande, 5
hachable, **94**
--help
    option de ligne de commande, 5
--help-all
    option de ligne de commande, 5
--help-env
    option de ligne de commande, 5
--help-xoptions
    option de ligne de commande, 5
--host
    option de ligne de commande, 35
HOSTRUNNER
    option de ligne de commande, 35

I

-I
    option de ligne de commande, 6
-i
    option de ligne de commande, 6
IDLE, **94**
immortal, **94**
immuable, **94**
importateur, **94**
importation, **94**
indication de type, **102**
instruction, **101**
interactif, **94**
interprété, **95**
itérable, **95**
itérable asynchrone, **88**
itérateur, **95**

itérateur asynchrone, **88**
itérateur de générateur, **93**
itérateur de générateur asynchrone, **88**

J

-J
    option de ligne de commande, 10

L

lambda, **96**
LBYL, **96**
LDFLAGS, 38, 40
    option de ligne de commande, 27
LDFLAGS_NODIST, 40
Le zen de Python, **103**
LIBB2_CFLAGS
    option de ligne de commande, 27
LIBB2_LIBS
    option de ligne de commande, 27
LIBEDIT_CFLAGS
    option de ligne de commande, 27
LIBEDIT_LIBS
    option de ligne de commande, 27
LIBFFI_CFLAGS
    option de ligne de commande, 28
LIBFFI_LIBS
    option de ligne de commande, 28
LIBLZMA_CFLAGS
    option de ligne de commande, 28
LIBLZMA_LIBS
    option de ligne de commande, 28
LIBMPDEC_CFLAGS
    option de ligne de commande, 28
LIBMPDEC_LIBS
    option de ligne de commande, 28
LIBREADLINE_CFLAGS
    option de ligne de commande, 28
LIBREADLINE_LIBS
    option de ligne de commande, 28
LIBS
    option de ligne de commande, 27
LIBSQLITE3_CFLAGS
    option de ligne de commande, 28
LIBSQLITE3_LIBS
    option de ligne de commande, 28
LIBUUID_CFLAGS
    option de ligne de commande, 28
LIBUUID_LIBS
    option de ligne de commande, 28
liste, **96**
liste en compréhension (*ou liste en intension*), **96**

M

-m
    option de ligne de commande, 4
MACHDEP
    option de ligne de commande, 27
machine virtuelle, **103**
magic
    méthode, 96
métaclasse, **96**
méthode, **97**
    magic, 96
    special, 101
méthode magique, **96**
méthode spéciale, **101**
module, **97**
module d'extension, **92**
MRO, **97**
mutable, **97**

N

n-uplet nommé, **97**
nom qualifié, **100**
nombre complexe, **90**
nombre de références, **100**
nouvelle classe, **98**

O

-O
    option de ligne de commande, 7
objet, **98**
objet fichier, **92**
objet fichier-compatible, **92**
objet octet-compatible, **89**
objet simili-chemin, **99**
-OO
    option de ligne de commande, 7
OPT, 32
optimized scope, **98**
option de ligne de commande
    -?, 5
    -B, 6
    -b, 6
    BOLT_APPLY_FLAGS, 30
    BOLT_INSTRUMENT_FLAGS, 30
    --build, 35
    BZIP2_CFLAGS, 27
    BZIP2_LIBS, 27
    -c, 3
    CC, 27
    CFLAGS, 27
    --check-hash-based-pycs, 6
    CONFIG_SITE, 35
    CPP, 27
    CPPFLAGS, 27
    CURSES_CFLAGS, 27
    CURSES_LIBS, 27
    -d, 6
    --disable-gil, 26
    --disable-ipv6, 24
    --disable-safety, 34
    --disable-test-modules, 29
    -E, 6
    --enable-big-digits, 24
    --enable-bolt, 30

--enable-experimental-jit, 26
--enable-framework, 34, 35
--enable-loadable-sqlite-extensions, 24
--enable-optimizations, 29
--enable-profiling, 31
--enable-pystats, 25
--enable-shared, 32
--enable-slower-safety, 34
--enable-universalsdk, 34
--enable-wasm-dynamic-linking, 28
--enable-wasm-pthreads, 29
--exec-prefix, 29
GDBM_CFLAGS, 27
GDBM_LIBS, 27
-h, 5
--help, 5
--help-all, 5
--help-env, 5
--help-xoptions, 5
--host, 35
HOSTRUNNER, 35
-I, 6
-i, 6
-J, 10
LDFLAGS, 27
LIBB2_CFLAGS, 27
LIBB2_LIBS, 27
LIBEDIT_CFLAGS, 27
LIBEDIT_LIBS, 27
LIBFFI_CFLAGS, 28
LIBFFI_LIBS, 28
LIBLZMA_CFLAGS, 28
LIBLZMA_LIBS, 28
LIBMPDEC_CFLAGS, 28
LIBMPDEC_LIBS, 28
LIBREADLINE_CFLAGS, 28
LIBREADLINE_LIBS, 28
LIBS, 27
LIBSQLITE3_CFLAGS, 28
LIBSQLITE3_LIBS, 28
LIBUUID_CFLAGS, 28
LIBUUID_LIBS, 28
-m, 4
MACHDEP, 27
-O, 7
-OO, 7
-P, 7
PANEL_CFLAGS, 28
PANEL_LIBS, 28
PKG_CONFIG, 27
PKG_CONFIG_LIBDIR, 27
PKG_CONFIG_PATH, 27
--prefix, 29
-q, 7
-R, 7
-S, 7
-s, 7
TCLTK_CFLAGS, 28
TCLTK_LIBS, 28
-u, 7
-V, 5
-v, 8
--version, 5
-W, 8
--with-address-sanitizer, 32
--with-app-store-compliance, 35
--with-assertions, 32
--with-build-python, 35
--with-builtin-hashlib-hashes, 33
--with-computed-gotos, 30
--with-dbmliborder, 25
--with-dtrace, 32
--with-ensurepip, 29
--with-framework-name, 35
--with-hash-algorithm, 33
--with-libc, 33
--with-libm, 33
--with-libs, 32
--with-lto, 30
--with-memory-sanitizer, 32
--with-openssl, 33
--with-openssl-rpath, 33
--without-c-locale-coercion, 25
--without-decimal-contextvar, 25
--without-doc-strings, 31
--without-mimalloc, 30
--without-pymalloc, 30
--without-readline, 33
--without-static-libpython, 32
--with-pkg-config, 25
--with-platlibdir, 25
--with-pydebug, 31
--with-readline, 33
--with-ssl-default-suites, 34
--with-strict-overflow, 31
--with-suffix, 24
--with-system-expat, 32
--with-system-libmpdec, 33
--with-tail-call-interp, 30
--with-thread-sanitizer, 32
--with-trace-refs, 31
--with-tzpath, 24
--with-undefined-behavior-sanitizer, 32
--with-universal-archs, 34
--with-valgrind, 32
--with-wheel-pkg-dir, 25
-X, 9
-x, 9
ZLIB_CFLAGS, 28
ZLIB_LIBS, 28
ordre de résolution des méthodes, **97**

P

-P

option de ligne de commande, 7
PANEL_CFLAGS
option de ligne de commande, 28
PANEL_LIBS
option de ligne de commande, 28
paquet, **98**
paquet classique, **100**
paquet provisoire, **99**
paquet-espace de nommage, **97**
paramètre, **98**
PATH, 11, 21, 44, 46, 5254, 56
PATHEXT, 46
PEP, **99**
PKG_CONFIG
option de ligne de commande, 27
PKG_CONFIG_LIBDIR
option de ligne de commande, 27
PKG_CONFIG_PATH
option de ligne de commande, 27
point d'entrée pour la recherche dans path, **99**
portée imbriquée, **98**
portion, **99**
--prefix
option de ligne de commande, 29
PROFILE_TASK, 29
PY_PYTHON, 57
pyc utilisant le hachage, **94**
PYLAUNCHER_ALLOW_INSTALL, 58
PYLAUNCHER_ALWAYS_INSTALL, 58
PYLAUNCHER_DEBUG, 58
PYLAUNCHER_DRYRUN, 58
PYLAUNCHER_NO_SEARCH_PATH, 56
Python 3000, **99**
Python Enhancement Proposals
PEP 1, 99
PEP 7, 23
PEP 11, 23, 43, 61
PEP 278, 102
PEP 302, 96
PEP 328, 92
PEP 338, 4
PEP 343, 90
PEP 362, 88, 99
PEP 370, 7, 13
PEP 397, 54
PEP 411, 99
PEP 420, 98, 99
PEP 443, 93
PEP 483, 94
PEP 484, 87, 93, 94, 102
PEP 488, 7
PEP 492, 88, 90
PEP 498, 92
PEP 514, 55
PEP 519, 99
PEP 525, 88
PEP 526, 87, 102
PEP 528, 54
PEP 529, 14, 54
PEP 538, 15, 25
PEP 585, 94
PEP 649, 87
PEP 683, 94
PEP 703, 48, 72, 92, 94
PEP 3116, 102
PEP 3155, 100
PYTHON_COLORS, 10
PYTHON_CPU_COUNT, 10
PYTHON_FROZEN_MODULES, 9
PYTHON_GIL, 10, 94
PYTHON_PERF_JIT_SUPPORT, 10
PYTHON_PRESITE, 10
PYTHONCOERCECLOCALE, 25
PYTHONDEBUG, 6, 31
PYTHONDEVMODE, 9
PYTHONDONTWRITEBYTECODE, 6
PYTHONDUMPREFS, 32
PYTHONFAULTHANDLER, 9
PYTHONHASHSEED, 7, 12
PYTHONHOME, 6, 11, 59
PYTHONINSPECT, 6
PYTHONINTMAXSTRDIGITS, 9
PYTHONIOENCODING, 15
Pythonique, **100**
PYTHONLEGACYWINDOWSSTDIO, 13
PYTHONMALLOC, 14, 30, 31
PYTHONNODEBUGRANGES, 9
PYTHONNOUSERSITE, 7
PYTHONOPTIMIZE, 7
PYTHONPATH, 6, 11, 53, 59
PYTHONPERFSUPPORT, 10
PYTHONPROFILEIMPORTTIME, 9
PYTHONPYCACHEPREFIX, 9
PYTHONSAFEPATH, 7
PYTHONSTARTUP, 6, 12
PYTHONTRACEMALLOC, 9
PYTHONUNBUFFERED, 8
PYTHONUTF8, 9, 15, 53
PYTHONVERBOSE, 8
PYTHONWARNDEFAULTENCODING, 9
PYTHONWARNINGS, 8

Q

-q
option de ligne de commande, 7

R

-R
option de ligne de commande, 7
ramasse-miettes, **93**
référence empruntée, **89**
référence forte, **101**
REPL, **100**
retours à la ligne universels, **102**

S

-S
    option de ligne de commande, 7
-s
    option de ligne de commande, 7
séquence, **100**
soft deprecated, **101**
special
    méthode, 101
spécificateur de module, **97**
static type checker, **101**

T

tableau de correspondances (*mapping en anglais*), **96**
TCLTK_CFLAGS
    option de ligne de commande, 28
TCLTK_LIBS
    option de ligne de commande, 28
TEMP, 49
tranche, **101**
typage canard (*duck-typing*), **91**
type, **102**
type générique, **93**

U

-u
    option de ligne de commande, 7

V

-V
    option de ligne de commande, 5
-v
    option de ligne de commande, 8
variable de classe, **89**
variable de contexte, **90**
variable d'environnement
    %APPDATA%, 49
    BASECFLAGS, 39
    BASECPPFLAGS, 38
    BLDSHARED, 40
    CC, 38
    CCSHARED, 39
    CFLAGS, 29, 3840
    CFLAGS_ALIASING, 39
    CFLAGS_NODIST, 3840
    CFLAGSFORSHARED, 39
    COMPILEALL_OPTS, 39
    CONFIGURE_CFLAGS, 39
    CONFIGURE_CFLAGS_NODIST, 39
    CONFIGURE_CPPFLAGS, 38
    CONFIGURE_LDFLAGS, 40
    CONFIGURE_LDFLAGS_NODIST, 40
    CPPFLAGS, 38, 40
    CXX, 38
    EXTRA_CFLAGS, 39
    LDFLAGS, 38, 40
    LDFLAGS_NODIST, 40
    LDSHARED, 40
    LIBS, 40
    LINKCC, 40
    OPT, 32, 39
    PATH, 11, 21, 44, 46, 5254, 56
    PATHEXT, 46
    PROFILE_TASK, 29, 30
    PURIFY, 39
    PY_BUILTIN_MODULE_CFLAGS, 39
    PY_CFLAGS, 39
    PY_CFLAGS_NODIST, 39
    PY_CORE_CFLAGS, 39
    PY_CORE_LDFLAGS, 40
    PY_CPPFLAGS, 38
    PY_LDFLAGS, 40
    PY_LDFLAGS_NODIST, 40
    PY_PYTHON, 57
    PY_STDMODULE_CFLAGS, 39
    PYLAUNCHER_ALLOW_INSTALL, 58
    PYLAUNCHER_ALWAYS_INSTALL, 58
    PYLAUNCHER_DEBUG, 58
    PYLAUNCHER_DRYRUN, 58
    PYLAUNCHER_NO_SEARCH_PATH, 56
    PYTHON_BASIC_REPL, 16
    PYTHON_COLORS, 10, 16
    PYTHON_CPU_COUNT, 10, 16
    PYTHON_FROZEN_MODULES, 9, 16
    PYTHON_GIL, 10, 16, 94
    PYTHON_HISTORY, 16
    PYTHON_PERF_JIT_SUPPORT, 10, 16
    PYTHON_PRESITE, 10, 17
    PYTHONASYNCIODEBUG, 14
    PYTHONBREAKPOINT, 11
    PYTHONCASEOK, 12
    PYTHONCOERCECLOCALE, 14, 25
    PYTHONDEBUG, 6, 11, 31
    PYTHONDEVMODE, 9, 15
    PYTHONDONTWRITEBYTECODE, 6, 12
    PYTHONDUMPREFS, 17, 32
    PYTHONDUMPREFSFILE, 17
    PYTHONEXECUTABLE, 13
    PYTHONFAULTHANDLER, 9, 13
    PYTHONHASHSEED, 7, 12
    PYTHONHOME, 6, 10, 11, 59
    PYTHONINSPECT, 6, 11
    PYTHONINTMAXSTRDIGITS, 9, 12
    PYTHONIOENCODING, 12, 15
    PYTHONLEGACYWINDOWSFSENCODING, 14
    PYTHONLEGACYWINDOWSSTDIO, 13, 14
    PYTHONMALLOC, 14, 30, 31
    PYTHONMALLOCSTATS, 14
    PYTHONNODEBUGRANGES, 9, 15
    PYTHONNOUSERSITE, 7, 13
    PYTHONOPTIMIZE, 7, 11
    PYTHONPATH, 6, 11, 53, 59
    PYTHONPERFSUPPORT, 10, 16
    PYTHONPLATLIBDIR, 11

PYTHONPROFILEIMPORTTIME, 9, 13
PYTHONPYCACHEPREFIX, 9, 12
PYTHONSAFEPATH, 7, 11
PYTHONSTARTUP, 6, 11, 12
PYTHONTRACEMALLOC, 9, 13
PYTHONUNBUFFERED, 8, 12
PYTHONUSERBASE, 13
PYTHONUTF8, 9, 15, 53
PYTHONVERBOSE, 8, 12
PYTHONWARNDEFAULTENCODING, 9, 15
PYTHONWARNINGS, 8, 13
TEMP, 49
verrou global de l'interpréteur, **94**
--version
option de ligne de commande, 5
vue de dictionnaire, **91**

W

-W
option de ligne de commande, 8
--with-address-sanitizer
option de ligne de commande, 32
--with-app-store-compliance
option de ligne de commande, 35
--with-assertions
option de ligne de commande, 32
--with-build-python
option de ligne de commande, 35
--with-builtin-hashlib-hashes
option de ligne de commande, 33
--with-computed-gotos
option de ligne de commande, 30
--with-dbmliborder
option de ligne de commande, 25
--with-dtrace
option de ligne de commande, 32
--with-ensurepip
option de ligne de commande, 29
--with-framework-name
option de ligne de commande, 35
--with-hash-algorithm
option de ligne de commande, 33
--with-libc
option de ligne de commande, 33
--with-libm
option de ligne de commande, 33
--with-libs
option de ligne de commande, 32
--with-lto
option de ligne de commande, 30
--with-memory-sanitizer
option de ligne de commande, 32
--with-openssl
option de ligne de commande, 33
--with-openssl-rpath
option de ligne de commande, 33
--without-c-locale-coercion
option de ligne de commande, 25
--without-decimal-contextvar
option de ligne de commande, 25
--without-doc-strings
option de ligne de commande, 31
--without-mimalloc
option de ligne de commande, 30
--without-pymalloc
option de ligne de commande, 30
--without-readline
option de ligne de commande, 33
--without-static-libpython
option de ligne de commande, 32
--with-pkg-config
option de ligne de commande, 25
--with-platlibdir
option de ligne de commande, 25
--with-pydebug
option de ligne de commande, 31
--with-readline
option de ligne de commande, 33
--with-ssl-default-suites
option de ligne de commande, 34
--with-strict-overflow
option de ligne de commande, 31
--with-suffix
option de ligne de commande, 24
--with-system-expat
option de ligne de commande, 32
--with-system-libmpdec
option de ligne de commande, 33
--with-tail-call-interp
option de ligne de commande, 30
--with-thread-sanitizer
option de ligne de commande, 32
--with-trace-refs
option de ligne de commande, 31
--with-tzpath
option de ligne de commande, 24
--with-undefined-behavior-sanitizer
option de ligne de commande, 32
--with-universal-archs
option de ligne de commande, 34
--with-valgrind
option de ligne de commande, 32
--with-wheel-pkg-dir
option de ligne de commande, 25

X

-X
option de ligne de commande, 9
-x
option de ligne de commande, 9

Z

ZLIB_CFLAGS
option de ligne de commande, 28
ZLIB_LIBS
option de ligne de commande, 28