What's New in Python

Version 3.10.16

A. M. Kuchling

décembre 07, 2024

Python Software Foundation
Email : docs@python.org

Table des matières

Version 3.10.16
Date décembre 07, 2024
Rédacteur *Pablo Galindo Salgado*

This article explains the new features in Python 3.10, compared to 3.9. Python 3.10 was released on October 4, 2021. For full details, see the changelog.

1 Résumé – Points forts de la publication

Nouvelles fonctionnalités de syntaxe :
- **PEP 634**, Filtrage par motifs structurels : spécification ;
- **PEP 635**, Filtrage par motifs structurels : motivation et justification ;
- **PEP 636**, Filtrage par motifs structurels : tutoriel ;
- bpo-12782, Les gestionnaires de contextes entre parenthèses sont maintenant autorisés.

Nouvelles fonctionnalités dans la bibliothèque standard :
- **PEP 618**, ajout de la vérification optionnelle de taille dans `zip`.

Améliorations de l'interpréteur :
- **PEP 626**, numéros de lignes précis pour le débogage et les autres outils.

Nouvelles fonctionnalités de typage :
- **PEP 604**, autorise l'écriture d'union de types sous la forme X | Y ;
- **PEP 612**, variables de spécification de paramètres.
- **PEP 613**, alias de types explicites ;
- **PEP 647**, User-Defined Type Guards

Obsolescences, retraits ou restrictions :
- **PEP 644**, mise à jour de la version minimale d'*OpenSSL* à 1.1.1 ;
- **PEP 632**, obsolescence du module `distutils` ;
- **PEP 623**, obsolescence et préparation du retrait du membre `wstr` de `PyUnicodeObject` ;
- **PEP 624**, retrait des API d'encodage `Py_UNICODE` ;
- **PEP 597**, ajout de l'`EncodingWarning` optionnel

2 Nouvelles fonctionnalités

2.1 Gestionnaires de contextes entre parenthèses

Il est maintenant possible de mettre les gestionnaires de contexte entre parenthèses pour les répartir sur plusieurs lignes. Ceci permet le formatage d'une longue série de gestionnaires de contexte sur plusieurs lignes comme ce qui était préalablement possible avec les instructions d'importation. Par exemple, tous ces blocs sont maintenant valides :

```
with (CtxManager() as example):
    ...

with (
    CtxManager1(),
    CtxManager2()
):
    ...

with (CtxManager1() as example,
      CtxManager2()):
    ...

with (CtxManager1(),
      CtxManager2() as example):
    ...

with (
    CtxManager1() as example1,
    CtxManager2() as example2
):
    ...
```

il est aussi possible d'utiliser une virgule en fin de ligne à la fin du groupe entre parenthèses :

```
with (
    CtxManager1() as example1,
    CtxManager2() as example2,
    CtxManager3() as example3,
):
    ...
```

Cette nouvelle syntaxe utilise la capacité *non-LL(1)* du nouvel analyseur syntaxique. Voir **PEP 617** pour plus de détails.

(contribution de *Guido van Rossum*, *Pablo Galindo* et *Lysandros Nikolaou* dans bpo-12782 et bpo-40334).

2.2 Meilleurs messages d'erreurs

`SyntaxError`

Pendant l'analyse syntaxique de code qui contient des parenthèses ou autres balises ouvrantes et fermantes, l'interpréteur inclut maintenant l'emplacement de la balise ou parenthèse non fermée plutôt que d'afficher *SyntaxError : unexpected EOF while parsing* ou d'indiquer un emplacement incorrect. Par exemple, considérez le code suivant (remarquez l'accolade `'{'` non-fermée) :

```
expected = {9: 1, 18: 2, 19: 2, 27: 3, 28: 3, 29: 3, 36: 4, 37: 4,
            38: 4, 39: 4, 45: 5, 46: 5, 47: 5, 48: 5, 49: 5, 54: 6,
some_other_code = foo()
```

Les versions précédentes de l'interpréteur indiquaient des emplacements qui portaient à confusion pour l'erreur de syntaxe :

```
File "example.py", line 3
    some_other_code = foo()
                    ^
SyntaxError: invalid syntax
```

mais dans Python 3.10, une erreur plus informative est levée :

```
File "example.py", line 1
    expected = {9: 1, 18: 2, 19: 2, 27: 3, 28: 3, 29: 3, 36: 4, 37: 4,
               ^
SyntaxError: '{' was never closed
```

De façon similaire, les erreurs impliquant des chaînes littérales non-fermées (avec simples ou triples apostrophes) pointent maintenant vers le début de la chaîne plutôt de mentionner la fin de ligne ou la fin du fichier.

Ces améliorations sont inspirées par un travail préalable sur l'interpréteur *PyPy*

(contribution de *Pablo Galindo* dans bpo-42864 et *Batuhan Taskaya* dans bpo-40176).

Les exceptions `SyntaxError` levées par l'interpréteur soulignent maintenant toute la portée de l'expression qui constitue l'erreur de syntaxe plutôt que seulement la position où le problème a été détecté. De cette façon, plutôt que d'afficher (avant Python 3.10) :

```
>>> foo(x, z for z in range(10), t, w)
  File "<stdin>", line 1
    foo(x, z for z in range(10), t, w)
           ^
SyntaxError: Generator expression must be parenthesized
```

Python 3.10 affiche maintenant l'exception comme ceci :

```
>>> foo(x, z for z in range(10), t, w)
  File "<stdin>", line 1
    foo(x, z for z in range(10), t, w)
           ^^^^^^^^^^^^^^^^^^^^
SyntaxError: Generator expression must be parenthesized
```

Cette amélioration est une contribution de *Pablo Galindo* dans bpo-43914.

Un nombre considérable de nouveaux messages spécialisés pour les exceptions `SyntaxError` a été incorporé. Certains des plus notables sont :

— `:` manquant avant les blocs :

```
>>> if rocket.position > event_horizon
  File "<stdin>", line 1
    if rocket.position > event_horizon
                                      ^
SyntaxError: expected ':'
```

(Contributed by Pablo Galindo in bpo-42997.)

— Les *n*-uplets sans parenthèses dans les cibles de compréhensions :

```
>>> {x,y for x,y in zip('abcd', '1234')}
  File "<stdin>", line 1
    {x,y for x,y in zip('abcd', '1234')}
     ^
SyntaxError: did you forget parentheses around the comprehension target?
```

(Contributed by Pablo Galindo in bpo-43017.)

— virgules manquantes dans les littéraux de collections et entre les expressions ;

```
>>> items = {
... x: 1,
... y: 2
... z: 3,
  File "<stdin>", line 3
    y: 2
       ^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

(Contributed by Pablo Galindo in bpo-43822.)

— types multiples d'`Exception` sans parenthèses :

```
>>> try:
...     build_dyson_sphere()
... except NotEnoughScienceError, NotEnoughResourcesError:
  File "<stdin>", line 3
    except NotEnoughScienceError, NotEnoughResourcesError:
           ^
SyntaxError: multiple exception types must be parenthesized
```

(Contributed by Pablo Galindo in bpo-43149.)

— `:` et valeurs manquantes dans les littéraux de dictionnaires :

```
>>> values = {
... x: 1,
... y: 2,
... z:
... }
  File "<stdin>", line 4
    z:
     ^
SyntaxError: expression expected after dictionary key and ':'

>>> values = {x:1, y:2, z w:3}
  File "<stdin>", line 1
    values = {x:1, y:2, z w:3}
                        ^
SyntaxError: ':' expected after dictionary key
```

(Contributed by Pablo Galindo in bpo-43823.)

— blocs `try` sans blocs `except` ou `finally` :

```
>>> try:
...     x = 2
... something = 3
```

(suite sur la page suivante)

(suite de la page précédente)

```
  File "<stdin>", line 3
    something  = 3
    ^^^^^^^^^
SyntaxError: expected 'except' or 'finally' block
```

(Contributed by Pablo Galindo in bpo-44305.)

— utilisation de = au lieu de == dans les comparaisons :

```
>>> if rocket.position = event_horizon:
  File "<stdin>", line 1
    if rocket.position = event_horizon:
                       ^
SyntaxError: cannot assign to attribute here. Maybe you meant '==' instead␣
↪of '='?
```

(Contributed by Pablo Galindo in bpo-43797.)

— utilisation de * dans les chaînes formatées (*f-strings*) :

```
>>> f"Black holes {*all_black_holes} and revelations"
  File "<stdin>", line 1
    (*all_black_holes)
     ^
SyntaxError: f-string: cannot use starred expression here
```

(Contributed by Pablo Galindo in bpo-41064.)

IndentationError

Plusieurs exceptions `IndentationError` ont maintenant plus de contexte concernant le genre de bloc qui attendait une indentation. Ceci inclut l'emplacement de l'instruction :

```
>>> def foo():
...     if lel:
...     x = 2
  File "<stdin>", line 3
    x = 2
    ^
IndentationError: expected an indented block after 'if' statement in line 2
```

AttributeError

À l'impression d'une `AttributeError`, `PyErr_Display()` offre des suggestions de noms d'attributs similaires dans l'objet pour lequel l'exception a été levée :

```
>>> collections.namedtoplo
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module 'collections' has no attribute 'namedtoplo'. Did you mean:␣
↪namedtuple?
```

(contribution de *Pablo Galindo* dans bpo-38530).

Avertissement : notez que ceci ne fonctionne pas si `PyErr_Display()` n'est pas appelée pour afficher l'erreur, ce qui est le cas si une fonction d'affichage d'erreur personnalisée est utilisée. C'est un scénario typique avec certains interpréteurs interactifs comme *IPython*.

NameError

Quand elle affiche une `NameError` levée par l'interpréteur, `PyErr_Display()` offre des suggestions de variables qui ont un nom similaire dans la fonction de laquelle l'exception a été levée :

```
>>> schwarzschild_black_hole = None
>>> schwarschild_black_hole
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'schwarschild_black_hole' is not defined. Did you mean: schwarzschild_
↪black_hole?
```

(contribution de *Pablo Galindo* dans bpo-38530).

Avertissement : notez que ceci ne fonctionne pas si `PyErr_Display()` n'est pas appelée pour afficher l'erreur, ce qui est le cas si une fonction d'affichage d'erreur personnalisée est utilisée. C'est un scénario typique avec certains interpréteurs interactifs comme *IPython*.

2.3 PEP 626 : numéros de lignes précis pour le débogage et les autres outils

La PEP 626 apporte des numéros de lignes plus précis pour le débogage, le profilage et les outils de mesure de couverture. Les événements de traçage, avec les numéros de ligne corrects, sont générés pour toutes les lignes de code exécutées et seulement pour celles-ci.

L'attribut `f_lineno` des objets cadres contient dorénavant le numéro de ligne attendu.

L'attribut `co_lnotab` des objets code est obsolète et sera retiré dans 3.12. Tout code qui doit convertir d'un décalage (*offset*) vers des numéros de ligne doit plutôt utiliser la nouvelle méthode `co_lines()`.

2.4 PEP 634 : filtrage par motifs structurels

Le filtrage par motifs a été ajouté sous la forme d'une instruction *match* et d'instructions *case* pour les motifs avec des actions associées. Les motifs filtrent des séquences, des dictionnaires, des types de données et des instances de classes. Le filtrage par motifs permet aux programmes d'extraire de l'information de types de données complexes, faire du branchement selon la structure des données et réaliser des actions spécifiques en fonction des différentes formes des données.

Syntaxe et opérations

La syntaxe générique du filtrage par motifs est :

```
match subject:
    case <pattern_1>:
        <action_1>
    case <pattern_2>:
        <action_2>
    case <pattern_3>:
        <action_3>
    case _:
        <action_wildcard>
```

Une instruction *match* prend une expression et compare successivement sa valeur à différents filtres qui sont donnés comme un ou plusieurs blocs *case*. Plus précisément, le filtrage par motifs s'effectue par :

1. l'utilisation de données qui ont un type et une forme (ici `subject`) ;
2. l'évaluation de `subject` dans une instruction `match` ;
3. l'application de chaque filtre sur `subject` dans des instructions `case` de haut en bas jusqu'à ce qu'un appariement soit confirmé ;
4. l'exécution de l'action associée au filtre s'il y a appariement ;
5. s'il n'y a aucun appariement et que le dernier `case` est un filtre *attrape-tout* _, l'action correspondante est exécutée. S'il n'y a aucun appariement ni filtre attrape-tout, le bloc `match` n'effectue aucune opération.

Approche déclarative

Le filtrage par motifs est peut-être connu des lecteurs par l'intermédiaire des langages *C*, *Java* ou *JavaScript* (et plusieurs autres langages), avec l'appariement simple d'un sujet (objet de données) à un littéral (filtre) avec l'instruction `switch`. Souvent, l'instruction `switch` est utilisée pour comparer des objets ou expressions à des instructions `case` qui contiennent des littéraux.

Des exemples plus puissants de filtrage par motifs sont présents dans des langages tels que *Scala* et *Elixir*. Avec le filtrage par motifs structurels, l'approche est « déclarative » et énonce les conditions (les motifs) pour apparier les données.

Bien qu'une série d'instructions « impératives » utilisant des instructions « if » imbriquées puisse être utilisée pour accomplir quelque chose de similaire au filtrage par motifs structurels, ceci est moins clair que l'approche « déclarative ». Cette dernière énonce les conditions à remplir pour réaliser un appariement grâce à ses filtres sur des motifs explicites. Bien que le filtrage par motifs structurels puisse être utilisé dans sa forme la plus simple pour comparer une variable à un littéral dans une instruction `case`, son intérêt réel en Python réside dans son traitement du sujet selon son type et sa forme.

Filtre simple : apparier à un littéral

Regardons cet exemple en tant que filtrage par motif dans sa forme la plus simple : une valeur, le sujet, est comparée à (ou « filtrée par ») plusieurs littéraux, les motifs. Dans l'exemple ci-dessous, `status` est le sujet de l'instruction `match`. Les filtres sont chacune des instructions `case`, où les littéraux représentent les codes de status des requêtes. Les actions associées au `case` sont exécutés suite à un appariement :

```
def http_error(status):
    match status:
        case 400:
            return "Bad request"
        case 404:
```

(suite sur la page suivante)

(suite de la page précédente)

```
            return "Not found"
        case 418:
            return "I'm a teapot"
        case _:
            return "Something's wrong with the internet"
```

Si on passe 418 à `status` dans la fonction ci-haut, elle renvoie `"I'm a teapot"`. Avec 500, l'instruction `case` qui contient _ apparie comme attrape-tout et la fonction renvoie `"Something's wrong with the internet"`. Prenez note que dans le dernier bloc, le nom de variable, _ agit comme un *attrape-tout* et garantit que le sujet sera toujours apparié. L'utilisation de _ est optionnelle.

Vous pouvez combiner plusieurs littéraux dans un seul filtre en utilisant `|` (qui se lit « ou ») :

```
case 401 | 403 | 404:
    return "Not allowed"
```

Comportement sans l'attrape-tout

Modifions l'exemple ci-haut en enlevant le dernier bloc `case`. Il devient :

```
def http_error(status):
    match status:
        case 400:
            return "Bad request"
        case 404:
            return "Not found"
        case 418:
            return "I'm a teapot"
```

Sans l'utilisation de _ dans une instruction `case`, il est possible de ne pas avoir d'appariement. Dans ce cas, il ne se passe rien. Par exemple, si 500 est passé à `status`, aucune opération n'est effectuée.

Filtres avec un littéral et une variable

Les motifs des filtres peuvent prendre une forme similaire aux affectations multiples et un filtre peut être utilisé pour lier plus d'une variable. Dans cet exemple, un point peut être dissocié entre son abscisse et son ordonnée :

```
# point is an (x, y) tuple
match point:
    case (0, 0):
        print("Origin")
    case (0, y):
        print(f"Y={y}")
    case (x, 0):
        print(f"X={x}")
    case (x, y):
        print(f"X={x}, Y={y}")
    case _:
        raise ValueError("Not a point")
```

Le motif du premier filtre comporte deux littéraux, `(0, 0)`, et peut être vu comme une extension du filtre sur un littéral montré plus haut. Les motifs des deux filtres suivants regroupent un littéral et une variable. De plus, la variable est liée à la valeur provenant du sujet (`point`). Le motif du quatrième filtre capture deux valeurs, ce qui le rend conceptuellement similaire à l'affectation multiple : `(x, y) = point`.

Filtres et classes

Si vous utilisez des classes pour structurer vos données, vous pouvez utiliser le nom de la classe suivie d'une liste d'arguments comme motif de filtre. Ceci ressemble à un appel du constructeur. Ce filtre peut capturer les attributs de la classe dans des variables :

```
class Point:
    x: int
    y: int

def location(point):
    match point:
        case Point(x=0, y=0):
            print("Origin is the point's location.")
        case Point(x=0, y=y):
            print(f"Y={y} and the point is on the y-axis.")
        case Point(x=x, y=0):
            print(f"X={x} and the point is on the x-axis.")
        case Point():
            print("The point is located somewhere else on the plane.")
        case _:
            print("Not a point")
```

Filtres avec arguments positionnels

Vous pouvez utiliser les arguments positionnels avec un certain nombre de classes natives qui définissent l'ordre de leurs attributs, notamment les classes de données (*dataclasses*). Vous pouvez aussi spécifier la position des attributs quand ils sont utilisés dans les filtres en définissant l'attribut spécial `__match_args__` dans vos classes. Par exemple, le mettre à `("x", "y")` rend tous les filtres ci-dessous équivalents (en particulier, tous provoquent la liaison de l'attribut `y` à la variable `var`) :

```
Point(1, var)
Point(1, y=var)
Point(x=1, y=var)
Point(y=var, x=1)
```

Filtres imbriqués

Les filtres peuvent être imbriqués de façon arbitraire. Par exemple, si nous avons une courte liste de points, elle peut être appariée comme ceci :

```
match points:
    case []:
        print("No points in the list.")
    case [Point(0, 0)]:
        print("The origin is the only point in the list.")
    case [Point(x, y)]:
        print(f"A single point {x}, {y} is in the list.")
    case [Point(0, y1), Point(0, y2)]:
        print(f"Two points on the Y axis at {y1}, {y2} are in the list.")
    case _:
        print("Something else is found in the list.")
```

Filtres complexes et attrape-tout

Jusqu'à maintenant, les exemples ont utilisé _ seul dans la dernière instruction `case`. Un attrape-tout peut être utilisé dans un filtre plus complexe tel que `('error', code, _)`. Par exemple :

```
match test_variable:
    case ('warning', code, 40):
        print("A warning has been received.")
    case ('error', code, _):
        print(f"An error {code} occurred.")
```

Dans l'exemple précédent, `test_variable` s'apparie à `('error', code, 100)` et à `('error', code, 800)`.

Garde

On peut ajouter une clause `if` à un filtre, ce qu'on appelle une « garde ». Si la garde s'évalue à faux, `match` poursuit avec la tentative d'appariement du prochain bloc `case`. Notez que la capture de valeur se produit avant l'évaluation de la garde :

```
match point:
    case Point(x, y) if x == y:
        print(f"The point is located on the diagonal Y=X at {x}.")
    case Point(x, y):
        print(f"Point is not on the diagonal.")
```

Autres fonctionnalités importantes

Plusieurs autres fonctionnalités importantes :

— Comme dans les affectations multiples, les motifs de filtres qui sont des *n*-uplets ou des listes sont totalement équivalents et autorisent tous les types de séquences. Techniquement, le sujet doit être une séquence. Ainsi, exception importante, les filtres n'autorisent pas les itérateurs. Aussi, pour prémunir d'une erreur commune, les filtres de séquences n'autorisent pas les chaînes ;

— les filtres de séquence peuvent faire intervenir l'affectation étoilée : `[x, y, *reste]` ou `(x, y, *reste)` ont le même sens que dans une affectation avec =. Le nom de variable après l'étoile peut aussi être l'attrape-tout _. Ainsi, `(x, y, *_)` est un motif de filtre qui reconnaît les séquences à deux éléments ou plus, en capturant les deux premiers et en ignorant le reste ;

— il existe les filtres d'association, qui ressemblent syntaxiquement aux dictionnaires. Par exemple, le filtre `{"bande_passante": b, "latence": l}` extrait les valeurs des clés `"bande_passante"` et `"latence"` dans un dictionnaire. Contrairement aux filtres de séquence, les clés absentes du filtre sont ignorées et le filtre réussit tout de même. L'affectation double-étoilée (`**reste`) fonctionne aussi (cependant, `**_` serait redondant et n'est donc pas permis) ;

— on peut capturer la valeur d'une partie du motif d'un filtre avec le mot-clé `as`, par exemple :

```
case (Point(x1, y1), Point(x2, y2) as p2): ...
```

Ceci lie `x1`, `y1`, `x2`, `y2` comme attendu sans la clause `as`, et lie `p2` à l'entièreté du second élément du sujet ;

— la plupart des littéraux sont comparés par égalité. Néanmoins, les singletons `True`, `False` et `None` sont comparés par identité ;

— les filtres peuvent contenir des noms qui se réfèrent à des constantes. Ces noms doivent impérativement être qualifiés (contenir au moins un point) pour ne pas être interprétés comme des variables de capture :

```
from enum import Enum
class Color(Enum):
    RED = 0
    GREEN = 1
    BLUE = 2

match color:
    case Color.RED:
        print("I see red!")
    case Color.GREEN:
        print("Grass is green")
    case Color.BLUE:
        print("I'm feeling the blues :(")
```

Pour la spécification complète, voir la **PEP 634**. La motivation et la justification des choix sont dans la **PEP 635**, et un tutoriel plus élaboré se trouve dans la **PEP 636**.

2.5 `EncodingWarning` et option `encoding="locale"` optionnels

L'encodage par défaut de `TextIOWrapper` et de `open()` dépend de la plateforme et des paramètres régionaux. Comme UTF-8 est utilisé sur la plupart des plateformes Unix, ne pas spécifier l'option `encoding` pour ouvrir des fichiers UTF-8 est une erreur courante. Par exemple :

```
# BUG: "rb" mode or encoding="utf-8" should be used.
with open("data.json") as f:
    data = json.load(f)
```

Pour trouver ce type de bogue, un `EncodingWarning` optionnel a été ajouté. Il est levé quand `sys.flags.warn_default_encoding` est `True` et quand l'encodage utilisé est celui défini par les paramètres régionaux.

L'option `-X warn_default_encoding` et la variable d'environnement `PYTHONWARNDEFAULTENCODING` ont été ajoutées pour activer cet avertissement.

Voir io-text-encoding pour plus d'informations.

3 Nouvelles fonctionnalités reliées aux indications de types

Cette section couvre les changements majeurs touchant le module `typing` et les indications de types de style **PEP 484**.

3.1 PEP 604 : nouvel opérateur d'union de types

Un nouvel opérateur d'union de types a été ajouté. Il permet la syntaxe `X | Y`. Ceci offre une façon plus propre d'exprimer « soit le type X, soit le type Y » plutôt que d'utiliser `typing.Union`, en particulier pour les indications de types.

Dans les versions précédentes de Python, pour appliquer une indication de type à des fonctions qui acceptent des arguments de plusieurs types, `typing.Union` était utilisée :

```
def square(number: Union[int, float]) -> Union[int, float]:
    return number ** 2
```

Les indications de type peuvent maintenant être écrites de façon plus courte :

```
def square(number: int | float) -> int | float:
    return number ** 2
```

Cette nouvelle syntaxe est aussi acceptée comme le second argument de `isinstance()` et de `issubclass()` :

```
>>> isinstance(1, int | str)
True
```

Voir types-union et la **PEP 604** pour plus de détails

(contribution de *Maggie Moss* et *Philippe Prados* dans bpo-41428, avec ajouts par *Yurii Karabas* et *Serhiy Storchaka* dans bpo-44490).

3.2 PEP 612 : variables de spécification de paramètres

Deux nouvelles options ont été ajoutées au module `typing` pour améliorer l'information donnée aux vérificateurs de types statiques pour les `Callable` définis dans la **PEP 484**.

La première est la variable de spécification de paramètre. Elles sont utilisées pour transférer le type d'un appelable à un autre – un patron de conception souvent utilisé dans les fonctions d'ordre supérieur et dans les décorateurs. Des exemples d'utilisation se trouvent dans `typing.ParamSpec`. Préalablement, il n'y avait pas de façon facile pour annoter les dépendances de types des paramètres de manière aussi précise.

La seconde option est le nouvel opérateur `Concatenate`. Il est utilisé en conjonction avec les variables de spécification de paramètres pour annoter les types d'un appelable d'ordre supérieur qui ajoute ou retire des paramètres à un autre appelable. Des exemples d'utilisation se trouvent dans `typing.Concatenate`.

Voir `typing.Callable`, `typing.ParamSpec`, `typing.Concatenate`, `typing.ParamSpecArgs`, `typing.ParamSpecKwargs` et la **PEP 612** pour plus de détails

(contribution de *Ken Jin* dans bpo-41559, avec améliorations mineures par *Jelle Zijlstra* dans bpo-43783 ; PEP écrite par *Mark Mendoza*).

3.3 PEP 613 : `TypeAlias`

La **PEP 484** a ajouté le concept d'alias de types, en exigeant seulement qu'ils soient des assignations non-annotées au niveau du module. Cette simplicité rendait parfois difficile pour les vérificateurs de types de distinguer entre un alias de type et une affectation ordinaire, en particulier quand des références postérieures ou des types invalides étaient impliqués. Comparez :

```
StrCache = 'Cache[str]'  # a type alias
LOG_PREFIX = 'LOG[DEBUG]'  # a module constant
```

Maintenant le module `typing` possède une valeur spéciale `TypeAlias` qui permet de déclarer des alias de types de façon plus explicite :

```
StrCache: TypeAlias = 'Cache[str]'  # a type alias
LOG_PREFIX = 'LOG[DEBUG]'  # a module constant
```

Voir la **PEP 613** pour plus de détails.

(contribution de *Mikhail Golubev* dans bpo-41923).

3.4 PEP 647 : gardes de type définies par l'utilisateur

`TypeGuard` a été ajouté au module `typing` pour annoter les fonctions de garde de type et améliorer l'information fournie aux vérificateurs de types statiques pendant le raffinage de types. Pour plus d'information, consultez la documentation de `TypeGuard` et la **PEP 647**.

(contribution de *Ken Jin* et *Guido van Rossum* dans bpo-43766. PEP écrit par *Eric Traut*).

4 Autres changements au langage

— Le type `int` a une nouvelle méthode `int.bit_count()`, qui renvoie le nombre de chiffres uns dans l'expansion binaire d'un entier donné, aussi connue sous le nom de chiffre de population (*population count*) (contribution de *Niklas Fiekas* dans bpo-29882).
— Les vues renvoyées par `dict.keys()`, `dict.values()` et `dict.items()` ont maintenant toutes un attribut `mapping` qui donne un objet `types.MappingProxyType` encapsulant le dictionnaire original (contribution de *Dennis Sweeney* dans bpo-40890).
— **PEP 618** : la fonction `zip()` a maintenant un argument optionnel `strict`, utilisé pour demander que tous les itérables soient de même longueur.
— Les fonctions d'extensions et les fonctions natives qui prennent des arguments entiers n'acceptent plus les `Decimal`, `Fraction` et autres objets qui ne peuvent pas être convertis en entiers sans perte de précision (par exemple ceux qui ont une méthode `__int__()` mais pas une méthode `__index__()`) (contribution de *Serhiy Storchaka* dans bpo-37999).
— Si `object.__ipow__()` renvoie `NotImplemented`, l'opérateur va correctement se rabattre sur `object.__pow__()` et `object.__rpow__()` tel qu'attendu (contribution d'*Alex Shkop* dans bpo-38302).
— Les expressions d'affectations peuvent maintenant être utilisées dans les littéraux d'ensembles, dans les compréhensions d'ensembles et dans les indices de séquences (mais pas dans les tranches).
— Les fonctions ont un nouvel attribut `__builtins__` qui est utilisé pour repérer les symboles natifs quand la fonction est exécutée, plutôt que de regarder dans `__globals__['__builtins__']`. L'attribut est initialisé à partir de `__globals__["__builtins__"]` s'il existe, ou des symboles natifs en cours autrement (contribution de *Mark Shannon* dans bpo-42990).
— Deux nouvelles fonctions natives – `aiter()` et `anext()` – ont été ajoutées pour fournir les équivalents asynchrones de `iter()` et de `next()`, respectivement (contribution de *Joshua Bronson*, *Daniel Pope* et *Justin Wang* dans bpo-31861).
— Les méthodes statiques (`@staticmethod`) et les méthodes de classes (`@classmethod`) héritent maintenant des attributs de méthodes (`__module__`, `__name__`, `__qualname__`, `__doc__`, `__annotations__`) et ont un nouvel attribut `__wrapped__`. De plus, les méthodes statiques sont maintenant appelables en tant que fonctions classiques (contribution de *Victor Stinner* dans bpo-43682).
— Les annotations des cibles complexes (toutes cibles autre que `simple name` telles que définies dans la **PEP 526**) n'ont plus d'effet à l'exécution avec `from __future__ import annotations` (contribution de *Batuhan Taskaya* dans bpo-42737).
— Les objets `class` et `module` créent maintenant paresseusement un dictionnaire d'annotations vide sur demande. Les dictionnaires d'annotations sont stockés dans le `__dict__` de l'objet pour garder la retro-compatibilité. Cela renforce les bonnes pratiques relatives à `__annotations__` ; pour plus d'information, voir annotations-howto (contribution de *Larry Hastings* dans bpo-43901).
— Les annotations qui contiennent `yield`, `yield from`, `await` ou une expression nommée ne sont plus permises sous `from __future__ import annotations` à cause de leurs effets secondaires (contribution de *Batuhan Taskaya* dans bpo-42725).
— L'utilisation de variables non liées, de `super()` et d'autres expressions qui peuvent changer le traitement de la table des symboles en tant qu'annotation n'ont plus d'effet sous `from __future__ import annotations` (contribution de *Batuhan Taskaya* dans bpo-42725).
— Le hachage des valeurs *NaN* des types `float` et `decimal.Decimal` dépend maintenant de l'identité de l'objet. Préalablement, cette valeur de hachage était toujours `0` alors que les valeurs *NaN* ne sont pas égales entre elles. Cela pouvait potentiellement entrainer un temps d'exécution quadratique en raison des collisions excessives des

valeurs de hachage lors de la création de dictionnaires et d'ensembles qui contiennent plusieurs *NaN* (contribution de *Raymond Hettinger* dans bpo-43475).
— Une `SyntaxError` (plutôt qu'une `NameError`) sera levée quand la constante `__debug__` est supprimée (contribution de *Dong-hee Na* dans bpo-45000).
— Les exceptions `SyntaxError` ont maintenant des attributs `end_lineno` et `end_offset`. La valeur est `None` quand ils ne peuvent pas être déterminés (contribution de *Pablo Galindo* dans bpo-43914).

5 Nouveaux modules

— Aucun pour le moment.

6 Modules améliorés

6.1 asyncio

Ajout de la méthode `connect_accepted_socket()` manquante (contribution d'*Alex Grönholm* dans bpo-41332).

6.2 argparse

L'expression `"optional arguments"` qui portait à confusion a été remplacée par `"options"` dans le message d'aide d'`argparse`. Certains tests devront être adaptés s'ils dépendent d'une comparaison exacte dans la sortie d'aide (contribution de *Raymond Hettinger* dans bpo-9694).

6.3 array

La méthode `index()` de la classe `array.array` a maintenant les paramètres optionnels *start* et *stop* (contribution d'*Anders Lorentsen* et de *Zackery Spytz* dans bpo-31956).

6.4 asynchat, asyncore, smtpd

Ces modules ont étés déclarés obsolètes dans leur documentation de modules depuis Python 3.6. Un `DeprecationWarning` au moment de l'importation est maintenant ajouté pour ces trois modules.

6.5 base64

Ajout de `base64.b32hexencode()` et de `base64.b32hexdecode()` pour prendre en charge l'encodage Base32 avec l'alphabet hexadécimal étendu.

6.6 bdb

Ajout de `clearBreakpoints()` pour supprimer tous les points d'arrêt (contribution d'*Irit Katriel* dans bpo-24160).

6.7 bisect

Ajout de la possibilité de fournir une fonction `key` aux API dans le module `bisect` (contribution de *Raymond Hettinger* dans bpo-4356).

6.8 codecs

Ajout d'une fonction `codecs.unregister()` pour désinscrire une fonction de recherche de codecs (contribution de *Hai Shi* dans bpo-41842).

6.9 collections.abc

L'attribut `__args__` des génériques paramétrables de `collections.abc.Callable` est maintenant cohérent avec `typing.Callable`. Les génériques de `collections.abc.Callable` aplatissent maintenant les types des paramètres à l'instar de `typing.Callable`. Ceci veut dire que `collections.abc.Callable[[int, str], str]` aura un attribut `__args__` de valeur `(int, str, str)` ; auparavant, il aurait été `([int, str], str)`. Pour permettre ce changement, `types.GenericAlias` peut être dérivée, et une sous-classe sera renvoyée lors de la surcharge du type `collections.abc.Callable`. Prenez note que `TypeError` sera levée pour des formes invalides de paramétrisation de `collections.abc.Callable`, ce qui aurait pu passer silencieusement dans Python 3.9 (contribution de *Ken Jin* dans bpo-42195).

6.10 contextlib

Ajout d'un gestionnaire de contexte `contextlib.aclosing()` pour fermer de façon sécurisée les générateurs asynchrones et les objets représentants des ressources libérées de façon asynchrone (contribution de *Joongi Kim* et *John Belmonte* dans bpo-41229).

Ajout d'un gestionnaire de contexte pour prendre en charge `contextlib.nullcontext()` (contribution de *Tom Gringauz* dans bpo-41543).

Ajout de `AsyncContextDecorator` pour prendre en charge les gestionnaires de contexte asynchrones en tant que décorateurs.

6.11 curses

Les fonctions de couleurs étendues ajoutées dans *ncurses* 6.1 sont utilisées de façon transparente par `curses.color_content()`, `curses.init_color()`, `curses.init_pair()` et `curses.pair_content()`. Une nouvelle fonction, `curses.has_extended_color_support()`, indique si la prise en charge des couleurs étendues est fournie par la bibliothèque *ncurses* sous-jacente (contribution de *Jeffrey Kintscher* et de *Hans Petter Jansson* dans bpo-36982).

Les constantes `BUTTON5_*` sont maintenant exposées dans le module `curses` si elles sont fournies par la bibliothèque *curses* sous-jacente (contribution de *Zackery Spytz* dans bpo-39273).

6.12 dataclasses

__slots__

Ajout du paramètre `slots` dans le décorateur `dataclasses.dataclass()` (contribution de *Yurii Karabas* dans bpo-42269).

Champs obligatoirement nommés

Les classes de données (`dataclass`) prennent maintenant en charge les champs obligatoirement nommés dans la méthode `__init__` générée. Il y a plusieurs façons de spécifier les champs obligatoirement nommés.

Vous pouvez indiquer que tous les champs sont obligatoirement nommés :

```
from dataclasses import dataclass

@dataclass(kw_only=True)
class Birthday:
    name: str
    birthday: datetime.date
```

Les paramètres `name` et `birthday` sont tous deux obligatoirement nommés dans la méthode `__init__` générée.

Pour chaque champ, vous pouvez indiquer s'il est obligatoirement nommé :

```
from dataclasses import dataclass

@dataclass
class Birthday:
    name: str
    birthday: datetime.date = field(kw_only=True)
```

Ici, seulement `birthday` est obligatoirement nommé. Si vous assignez `kw_only` sur un champ particulier, notez qu'il y a des règles concernant l'ordre des champs puisque les champs obligatoirement nommés doivent suivre les champs optionnellement nommés. Voir la documentation de `dataclass` pour plus de détails.

Vous pouvez aussi indiquer que tous les champs qui suivent un marqueur `KW_ONLY` sont obligatoirement nommés. Ceci sera probablement l'utilisation la plus commune :

```
from dataclasses import dataclass, KW_ONLY

@dataclass
class Point:
    x: float
    y: float
    _: KW_ONLY
    z: float = 0.0
    t: float = 0.0
```

Here, `z` and `t` are keyword-only parameters, while `x` and `y` are not. (Contributed by Eric V. Smith in bpo-43532.)

6.13 distutils

Le paquet `distutil` est entièrement obsolète et sera retiré dans Python 3.12. Ses fonctionnalités pour spécifier la construction de paquets sont déjà complètement remplacées par les paquets tierce-parties `setuptools` et `packaging`. Les autres API régulièrement utilisées sont disponibles ailleurs dans la bibliothèque standard (tel que dans `platform`, `shutil`, `subprocess` ou `sysconfig`). Il n'est pas prévu de migrer d'autres fonctionnalités de `distutils`, aussi les applications qui utilisent d'autres fonctions du module doivent faire des copies privées du code. Voir la discussion dans la **PEP 632**.

La commande `bdist_wininst` qui est obsolète depuis Python 3.8 a été retirée. La commande `bdist_wheel` est maintenant recommandée pour distribuer des paquets binaires sous Windows (contribution de *Victor Stinner* dans bpo-42802).

6.14 doctest

Quand un module ne définit pas `__loader__`, `__spec__.loader` est utilisé (contribution de *Brett Cannon* dans bpo-42133).

6.15 encodings

La fonction `encodings.normalize_encoding()` ignore maintenant les caractères non ASCII (contribution de *Hai Shi* dans bpo-39337).

6.16 fileinput

Ajout des paramètres `encoding` et `errors` dans `fileinput.input()` et dans `fileinput.FileInput` (contribution d'*Inada Naoki* dans bpo-43712).

`fileinput.hook_compressed()` renvoie maintenant un objet `TextIOWrapper` quand `mode` est `"r"` et que le fichier est compressé, comme pour les fichiers non compressés (contribution d'*Inada Naoki* dans bpo-5758).

6.17 faulthandler

Le module `faulthandler` détecte maintenant si une erreur fatale s'est produite pendant un passage du ramasse-miettes (contribution de *Victor Stinner* dans bpo-44466).

6.18 gc

Ajout de fonctions de rappels d'audit pour `gc.get_objects()`, `gc.get_referrers()` et `gc.get_referents()` (contribution de *Pablo Galindo* dans bpo-43439).

6.19 glob

Ajout des paramètres `root_dir` et `dir_fd` dans `glob()` et dans `iglob()` pour permettre la spécification du répertoire racine utilisé pour la recherche (contribution de *Serhiy Storchaka* dans bpo-38144).

6.20 hashlib

La version minimale d'*OpenSSL* pour le module `hashlib` est maintenant 1.1.1 (contribution de *Christian Heimes* dans **PEP 644** et dans bpo-43669).

Le module *hashlib* offre une prise en charge préliminaire pour *OpenSSL* 3.0.0 (contribution de *Christian Heimes* dans bpo-38820 et autres tickets).

Le repli vers la version en pur Python de `pbkdf2_hmac()` est obsolète. Dorénavant, `PBKDF2-HMAC` n'est disponible que si Python a été compilé avec la prise en charge d'*OpenSSL* (contribution de *Christian Heimes* dans bpo-43880).

6.21 hmac

L'implémentation interne du module `hmac` utilise maintenant *HMAC* d'*OpenSSL* (contribution de *Christian Heimes* dans bpo-40645).

6.22 IDLE et idlelib

IDLE invoque maintenant `sys.excepthook()` (lorsque lancé sans `'-n'`). Auparavant, les fonctions de rappel définies par l'utilisateur étaient ignorées (contribution par *Ken Hilton* dans bpo-43008).

Rearrange the settings dialog. Split the General tab into Windows and Shell/Ed tabs. Move help sources, which extend the Help menu, to the Extensions tab. Make space for new options and shorten the dialog. The latter makes the dialog better fit small screens. (Contributed by Terry Jan Reedy in bpo-40468.) Move the indent space setting from the Font tab to the new Windows tab. (Contributed by Mark Roseman and Terry Jan Reedy in bpo-33962.)

Les changements ci-dessus ont été rétro-portés dans le cadre de la maintenance de 3.9.

Ajout d'une barre de côté à la console. Déplacement de l'invite de commande principale (>>>) vers cette barre. Ajout des invites secondaires (`...`) à la barre de côté. Le clic de gauche et le glisser optionnel sélectionnent une ou plusieurs lignes de texte, tout comme dans la barre de numéros de ligne de la fenêtre d'édition. Le clic droit après la sélection de lignes de texte affiche un menu contextuel pour copier avec invites (*'copy with prompts'*). Ceci combine les invites de la barre de côté avec le texte sélectionné. Cette option apparait aussi dans le menu contextuel pour le texte (contribution de *Tal Einat* dans bpo-37903).

Utilisation d'espaces plutôt que des tabulations pour indenter le code interactif. Ceci donne au code interactif la « bonne apparence ». Cette fonctionnalité est fortement corrélée à l'ajout de la barre de côté de la console (contribution de *Terry Jan Reedy* dans bpo-37892).

Coloration des nouveaux mots clés ad-hoc `match`, `case` et _ dans les instructions de filtrage par motifs (*pattern matching*). Par contre, cette coloration n'est pas parfaite et elle sera incorrecte dans quelques rares cas, y-compris certains _ dans des instructions `case` (contribution de Tal Einat dans bpo-44010).

Nouveau dans les versions de maintenance 3.10.

Applique la coloration syntaxique aux fichiers `.pyi` (contribution d'*Alex Waygood* et de *Terry Jan Reedy* dans bpo-45447).

Include prompts when saving Shell with inputs and outputs. (Contributed by Terry Jan Reedy in gh-95191.)

6.23 importlib.metadata

Équivalence de fonctionnalités avec importlib_metadata 4.6 (historique).

Les points d'entrée de importlib.metadata offrent maintenant une interface plus agréable pour choisir les points d'entrée par groupe et par nom par l'intermédiaire d'une nouvelle classe importlib.metadata.EntryPoints. Voir la rubrique Note sur la Compatibilité (*Compatibility Note*) dans la documentation du module importlib.metadata pour plus d'information sur son utilisation et ce qui devient obsolète.

Ajout de importlib.metadata.packages_distributions() pour résoudre les modules et paquets Python du niveau racine vers leur importlib.metadata.Distribution.

6.24 inspect

Quand un module ne définit pas __loader__, __spec__.loader est utilisé (contribution de *Brett Cannon* dans bpo-42133).

Ajout de inspect.get_annotations() qui calcule les annotations définies sur un objet de façon sécuritaire. Elle contourne les difficultés d'accéder aux annotations de différents types d'objets et présume très peu de l'objet examiné. inspect.get_annotations() peut également convertir pour vous les annotations contenues dans des chaînes de caractères en objets. inspect.get_annotations() est maintenant considérée comme la meilleure pratique pour accéder au dictionnaire d'annotations de tout objet qui prend en charge les annotations ; pour d'autre information sur les meilleures pratiques relatives aux annotations, voir annotations-howto. Dans le même esprit, inspect.signature(), inspect.Signature.from_callable() et inspect.Signature.from_function() appellent maintenant inspect.get_annotations() pour récupérer les annotations. Ceci veut dire que inspect.signature() et inspect.Signature.from_callable() peuvent maintenant convertir pour vous les annotations contenues dans des chaînes de caractères en objets (contribution de *Larry Hastings* dans bpo-43817).

6.25 itertools

Add itertools.pairwise(). (Contributed by Raymond Hettinger in bpo-38200.)

6.26 linecache

Quand un module ne définit pas __loader__, __spec__.loader est utilisé (contribution de *Brett Cannon* dans bpo-42133).

6.27 os

Ajout de la prise en charge de *VxWorks RTOS* dans os.cpu_count() (contribution de *Peixing Xin* dans bpo-41440).

Ajout d'une nouvelle fonction os.eventfd() et des fonctions utilitaires associées pour encapsuler l'appel système eventfd2 sur Linux (contribution de *Christian Heimes* dans bpo-41001).

Ajout de os.splice() qui permet de déplacer des données entre deux descripteurs de fichiers sans copie entre les espaces noyau et utilisateur. Un des descripteurs de fichiers doit référencer un tube (*pipe*) (contribution de *Pablo Galindo* dans bpo-41625).

Ajout de O_EVTONLY, O_FSYNC, O_SYMLINK et O_NOFOLLOW_ANY sur macOS (contribution de *Dong-hee Na* dans bpo-43106).

As of 3.10.15, `os.mkdir()` and `os.makedirs()` on Windows now support passing a *mode* value of `0o700` to apply access control to the new directory. This implicitly affects `tempfile.mkdtemp()` and is a mitigation for CVE-2024-4030. Other values for *mode* continue to be ignored. (Contributed by Steve Dower in gh-118486.)

6.28 os.path

`os.path.realpath()` accepte maintenant l'argument nommé *strict*. Lorsqu'il est `True`, une `OSError` est levée si le chemin n'existe pas ou si une boucle de liens symboliques est rencontrée (contribution de *Barney Gale* dans bpo-43757).

6.29 pathlib

Add slice support to `PurePath.parents`. (Contributed by Joshua Cannon in bpo-35498.)

Add negative indexing support to `PurePath.parents`. (Contributed by Yaroslav Pankovych in bpo-21041.)

Ajout de la méthode `Path.hardlink_to` qui remplace `link_to()`. L'ordre des arguments de la nouvelle méthode est le même que `symlink_to()` (contribution de *Barney Gale* dans bpo-39950).

`pathlib.Path.stat()` et `chmod()` acceptent maintenant l'argument nommé *follow_symlinks* pour rester cohérent avec les fonctions correspondantes dans le module `os` (contribution de *Barney Gale* dans bpo-39906).

6.30 platform

Add `platform.freedesktop_os_release()` to retrieve operation system identification from freedesktop.org os-release standard file. (Contributed by Christian Heimes in bpo-28468.)

6.31 pprint

`pprint.pprint()` accepte maintenant le nouvel argument nommé `underscore_numbers` (contribution de *sblondon* dans bpo-42914).

`pprint` peut maintenant afficher élégamment les instances de `dataclasses.dataclass` (contribution de *Lewis Gaul* dans bpo-43080).

6.32 py_compile

Ajout de l'option de ligne de commande `--quiet` à l'interface de `py_compile` (contribution de *Gregory Schevchenko* dans bpo-38731).

6.33 pyclbr

Ajout d'un attribut `end_lineno` aux objets `Function` et `Class` dans l'arbre renvoyé par `pyclbr.readline()` et `pyclbr.readline_ex()`. Il complète l'attribut `lineno` déjà existant (contribution d'*Aviral Srivastava* dans bpo-38307).

6.34 shelve

Le module `shelve` utilise maintenant `pickle.DEFAULT_PROTOCOL` par défaut plutôt que le protocole `3` du module `pickle` à la création des objets `shelf` (contribution de *Zackery Spytz* dans bpo-34204).

6.35 statistics

Ajout des fonctions `covariance()`, `correlation()` (corrélation de Pearson) et `linear_regression()` (régression linéaire simple) (contribution de *Tymoteusz Wołodźko* dans bpo-38490).

6.36 site

Quand un module ne définit pas `__loader__`, `__spec__.loader` est utilisé (contribution de *Brett Cannon* dans bpo-42133).

6.37 socket

L'exception `socket.timeout` est maintenant un alias de `TimeoutError` (contribution de *Christian Heimes* dans bpo-42413).

Ajout d'une option pour créer des connecteurs (*socket*) *MPTCP* avec `IPPROTO_MPTCP` (contribution de *Rui Cunha* dans bpo-43571).

Ajout de l'option `IP_RECVTOS` pour récupérer la valeur des champs *type of service (ToS)* ou *differentiated services code point (DSCP/ECN)* (contribution de *Georg Sauthoff* dans bpo-44077).

6.38 ssl

La version minimale d'*OpenSSL* pour le module `ssl` est maintenant 1.1.1 (contribution de *Christian Heimes* dans **PEP 644** et dans bpo-43669).

Le module `ssl` offre une prise en charge préliminaire pour *OpenSSL* 3.0.0 et la nouvelle option `OP_IGNORE_UNEXPECTED_EOF` (contribution de *Christian Heimes* dans bpo-38820, bpo-43794, bpo-43788, bpo-43791, bpo-43799, bpo-43920, bpo-43789 et bpo-43811).

Les fonctions obsolètes et l'accès aux constantes obsolètes lèvent maintenant un `DeprecationWarning`. `ssl.SSLContext.options` a options `OP_NO_SSLv2` et `OP_NO_SSLv3` définies par défaut ; par conséquent, il ne peut pas avertir si l'option est définie de nouveau. La section *obsolescence* contient la liste des fonctionnalités obsolètes (contribution de *Christian Heimes* dans bpo-43880).

La configuration du module `ssl` est plus sécuritaire par défaut. Les algorithmes de chiffrement sans confidentialité persistante (*forward secrecy*) et le code d'authentification de message (MAC) *SHA-1* sont désactivés par défaut. Le niveau 2 de sécurité ne permet pas les clés faibles de moins de 112 bits pour *RSA*, *DH* et *ECC*. La version minimale du protocole *TLS* pour `SSLContext` est 1.2. La configuration est basée sur la recherche de *Hynek Schlawack* (contribution de *Christian Heimes* dans bpo-43998).

Les protocoles obsolètes *SSL* 3.0, *TLS* 1.0 et *TLS* 1.1 ne sont plus officiellement pris en charge. Python ne les bloque pas activement, mais les options de compilation d'*OpenSSL*, les configurations du système d'exploitation, les rustines des fabricants et les suites d'algorithmes de chiffrement peuvent empêcher l'établissement d'une connexion.

Ajout d'un paramètre *timeout* à la fonction `ssl.get_server_certificate()` (contribution de *Zackery Spytz* dans bpo-31870).

Le module `ssl` utilise les types du tas et l'initialisation multi-phase (contribution de *Christian Heimes* dans bpo-42333).

Ajout d'un nouveau drapeau de vérification `VERIFY_X509_PARTIAL_CHAIN` (contribution de *l0x* dans bpo-40849).

6.39 sqlite3

Ajout d'événements d'audit pour `connect/handle()`, `enable_load_extension()` et `load_extension()` (contribution d'*Erlend E. Aasland* dans bpo-43762).

6.40 sys

Ajout de l'attribut `sys.orig_argv` : la liste des arguments de ligne de commande originalement passée à l'exécutable Python (contribution de *Victor Stinner* dans bpo-23427).

Ajout de `sys.stdlib_module_names` qui contient la liste des noms des modules de la bibliothèque standard (contribution de *Victor Stinner* dans bpo-42955).

6.41 tempfile

As of 3.10.15 on Windows, the default mode `0o700` used by `tempfile.mkdtemp()` now limits access to the new directory due to changes to `os.mkdir()`. This is a mitigation for CVE-2024-4030. (Contributed by Steve Dower in gh-118486.)

6.42 _thread

`_thread.interrupt_main()` prend maintenant l'argument optionnel `signum` : le numéro de signal à simuler (la valeur par défaut est toujours `signal.SIGINT`) (contribution d'*Antoine Pitrou* dans bpo-43356).

6.43 threading

Ajout des fonctions `threading.gettrace()` et `threading.getprofile()` pour récupérer les fonctions assignées par `threading.settrace()` et `threading.setprofile()` respectivement (contribution de *Mario Corchero* dans bpo-42251).

Ajout de `threading.__excepthook__` pour permettre la récupération de `threading.excepthook()` au cas où elle serait affectée à une valeur incorrecte ou différente (contribution de *Mario Corchero* dans bpo-42308).

6.44 traceback

Les fonctions `format_exception()`, `format_exception_only()` et `print_exception()` peuvent maintenant prendre un objet exception en argument positionnel (contribution de *Zackery Spytz* et *Matthias Bussonnier* dans bpo-26389).

6.45 types

Les classes `types.EllipsisType`, `types.NoneType` et `types.NotImplementedType` sont ajoutées de nouveau. Elles fournissent un nouvel ensemble de types facilement interprétable par les vérificateurs de types (contribution de *Bas van Beek* dans bpo-41810).

6.46 typing

Pour les changements majeurs, voir *Nouvelles fonctionnalités reliées aux indications de types*.

Le comportement de `typing.Literal` a été changé pour être conforme avec la **PEP 586** et pour correspondre au comportement des vérificateurs de types statiques défini dans la PEP.

1. `Literal` dé-duplique maintenant les paramètres.
2. La comparaison d'égalité entre objets `Literal` ne tient plus compte de l'ordre.
3. Les comparaisons de `Literal` respectent maintenant les types. Par exemple, `Literal[0] == Literal[False]` était préalablement évaluée à `True`. Elle est maintenant `False`. Pour prendre en charge ce changement, le cache des types de l'implémentation interne peut maintenant différencier entre les types.
4. Les objets `Literal` lèvent maintenant une exception `TypeError` pendant la comparaison d'égalité si au moins un de leurs paramètres n'est pas hachable. Notez que déclarer un `Literal` avec des paramètres non hachables ne lève pas d'erreur :

```
>>> from typing import Literal
>>> Literal[{0}]
>>> Literal[{0}] == Literal[{False}]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'set'
```

(contribution de *Yurii Karabas* dans bpo-42345).

Add new function `typing.is_typeddict()` to introspect if an annotation is a `typing.TypedDict`. (Contributed by Patrick Reader in bpo-41792.)

Subclasses of `typing.Protocol` which only have data variables declared will now raise a `TypeError` when checked with `isinstance` unless they are decorated with `runtime_checkable()`. Previously, these checks passed silently. Users should decorate their subclasses with the `runtime_checkable()` decorator if they want runtime protocols. (Contributed by Yurii Karabas in bpo-38908.)

Importing from the `typing.io` and `typing.re` submodules will now emit `DeprecationWarning`. These submodules have been deprecated since Python 3.8 and will be removed in a future version of Python. Anything belonging to those submodules should be imported directly from `typing` instead. (Contributed by Sebastian Rittau in bpo-38291.)

6.47 unittest

Ajout d'une nouvelle méthode `assertNoLogs()` pour complémenter la méthode existante `assertLogs()` (contribution de *Kit Yan Choi* dans bpo-39385).

6.48 urllib.parse

Les versions préalables à Python 3.10 acceptaient les deux caractères `;` et `&` comme séparateurs de paramètres de requêtes dans `urllib.parse.parse_qs()` et `urllib.parse.parse_qsl()`. Pour prévenir des failles de sécurité potentielles et pour respecter les dernières recommandations du W3C, ceci a été changé pour ne permettre qu'une seule clé de séparation, avec `&` par défaut. Ce changement affecte aussi l'implémentation interne de `cgi.parse()` et de `cgi.parse_multipart()` puisqu'elles utilisent les fonctions affectées. Pour plus de détails, voir la documentation respective de ces fonctions (contribution d'*Adam Goldschmidt*, de *Senthil Kumaran* et de *Ken Jin* dans bpo-42967).

La présence de caractères de saut de ligne ou de tabulation dans une URL permet certaines formes d'attaques. En conformité avec la spécification du *WHATWG* qui met à jour la **RFC 3986**, les caractères *ASCII* de saut de ligne `\n`, `\r` et de tabulation `\t` sont retirés des URL par l'analyseur dans `urllib.parse` pour contrer ces attaques. Le jeu de caractères à retirer est contrôlé par une nouvelle variable de module `urllib.parse._UNSAFE_URL_BYTES_TO_REMOVE` (voir bpo-43882).

6.49 xml

Ajout d'une classe `LexicalHandler` au module `xml.sax.handler` (contribution de *Jonathan Gossage* de *Zackery Spytz* dans bpo-35018).

6.50 zipimport

Ajout des méthodes relatives à la **PEP 451** : `find_spec()`, `zipimport.zipimporter.create_module()` et `zipimport.zipimporter.exec_module()` (contribution de *Brett Cannon* dans bpo-42131).

Ajout de la méthode `invalidate_caches()` (contribution de *Desmond Cheong* dans bpo-14678).

7 Optimisations

- Les constructeurs de `str()`, `bytes()` et `bytearray()` sont maintenant plus rapides (environ 30—40 % pour les petits objets) (contribution de *Serhiy Storchaka* dans bpo-41334).
- Le module `runpy` importe maintenant moins de modules. Le temps de démarrage de la commande `python3 -m module-name` est en moyenne 1,4 fois plus rapide. Sur Linux `python3 -I -m module-name` importe 69 modules dans Python 3.9, alors qu'elle n'en importe que 51 (18 de moins) dans Python 3.10 (contribution de *Victor Stinner* dans bpo-41006 et dans bpo-41718).
- L'instruction `LOAD_ATTR` utilise maintenant le mécanisme de cache par code d'opération. Elle est environ 36 % plus rapide pour les attributs ordinaires et 44 % plus rapide pour les *slots* (contribution de *Pablo Galindo* et de *Yury Selivanov* dans bpo-42093 et *Guido van Rossum* dans bpo-42927, basée sur des idées originalement mises en œuvre dans *PyPy* et dans *MicroPython*).
- Lors de la construction de Python avec `--enable-optimizations`, `-fno-semantic-interposition` est maintenant ajouté aux lignes de commande de compilation et d'édition des liens. Ceci accélère les binaires de l'interpréteur Python créés avec `--enable-shared` et `gcc` jusqu'à 30 %. Voir cet article (en anglais) pour plus de détails (contribution de *Victor Stinner* et *Pablo Galindo* dans bpo-38980).
- Ré-usinage du code assurant la gestion du tampon de sortie pour les modules `bz2`, `lzma` et `zlib`. Ajout de la fonction `.readall()` à la classe `_compression.DecompressReader`. La décompression *bz2* est 1,09 à 1,17 fois plus rapide, la décompression *lzma* est 1,20 à 1,32 fois plus rapide, `GzipFile.read(-1)` est 1,11 à 1,18 fois plus rapide (contribution de *Ma Lin*, révision par *Gregory P. Smith* dans bpo-41486).
- When using stringized annotations, annotations dicts for functions are no longer created when the function is created. Instead, they are stored as a tuple of strings, and the function object lazily converts this into the annotations

dict on demand. This optimization cuts the CPU time needed to define an annotated function by half. (Contributed by Yurii Karabas and Inada Naoki in bpo-42202.)

— Les fonctions de recherche de sous-chaînes telles que `str1 in str2` et `str2.find(str1)` peuvent utiliser maintenant l'algorithme « bidirectionnel » de recherche de chaînes *Crochemore et Perrin* pour éviter le comportement quadratique sur les longues chaînes (contribution de *Dennis Sweeney* dans bpo-41972).

— Add micro-optimizations to `_PyType_Lookup()` to improve type attribute cache lookup performance in the common case of cache hits. This makes the interpreter 1.04 times faster on average. (Contributed by Dino Viehland in bpo-43452.)

— The following built-in functions now support the faster **PEP 590** vectorcall calling convention : `map()`, `filter()`, `reversed()`, `bool()` and `float()`. (Contributed by Dong-hee Na and Jeroen Demeyer in bpo-43575, bpo-43287, bpo-41922, bpo-41873 and bpo-41870.)

— `BZ2File` performance is improved by removing internal `RLock`. This makes `BZ2File` thread unsafe in the face of multiple simultaneous readers or writers, just like its equivalent classes in `gzip` and `lzma` have always been. (Contributed by Inada Naoki in bpo-43785.)

8 Obsolescence

— Currently Python accepts numeric literals immediately followed by keywords, for example `0in x`, `1or x`, `0if 1else 2`. It allows confusing and ambiguous expressions like `[0x1for x in y]` (which can be interpreted as `[0x1 for x in y]` or `[0x1f or x in y]`). Starting in this release, a deprecation warning is raised if the numeric literal is immediately followed by one of keywords `and`, `else`, `for`, `if`, `in`, `is` and `or`. In future releases it will be changed to syntax warning, and finally to syntax error. (Contributed by Serhiy Storchaka in bpo-43833.)

— À partir de la version présente, il y aura un effort coordonné pour commencer le nettoyage des anciennes sémantiques d'importation qui étaient gardées pour la compatibilité avec Python 2.7. Plus spécifiquement, `find_loader()`/`find_module()` (remplacées par `find_spec()`), `load_module()` (remplacée par `exec_module()`), `module_repr()` (qui est géré par le système d'importation pour vous), l'attribut `__package__` (remplacé par `__spec__.parent`), l'attribut `__loader__` (remplacé par `__spec__.loader`) et l'attribut `__cached__` (remplacé par `__spec__.cached`) seront progressivement retirés (de même que d'autres classes et méthodes dans `importlib`). Des `ImportWarning` ou `DeprecationWarning` seront levés le cas échéant pendant cette transition pour aider à identifier le code qui doit être mis à jour.

— L'espace de noms `distutils` est complètement obsolète et sera retiré dans Python 3.12. Voir la section sur les *changements aux modules* pour plus d'information.

— Les arguments non entiers de `random.randrange()` sont obsolètes. La levée de `ValueError` dans ce cas est obsolète et est remplacée par `TypeError` (contribution de *Serhiy Storchaka* et *Raymond Hettinger* dans bpo-37319).

— Les multiples méthodes `load_module()` de `importlib` étaient documentées comme obsolètes depuis Python 3.6, mais elles vont maintenant aussi lever un `DeprecationWarning`. Utilisez plutôt `exec_module()` (contribution de *Brett Cannon* dans bpo-26131).

— `zimport.zipimporter.load_module()` est obsolète. Utilisez plutôt `exec_module()` (contribution de *Brett Cannon* dans bpo-26131).

— L'utilisation de `load_module()` par le système d'importation lève maintenant un `ImportWarning` car `exec_module()` est recommandée (contribution de (*Brett Cannon* dans bpo-26131).

— L'utilisation de `importlib.abc.MetaPathFinder.find_module()` et de `importlib.abc.PathEntryFinder.find_module()` par le système d'importation lève maintenant un `ImportWarning` car `importlib.abc.MetaPathFinder.find_spec()` et `importlib.abc.PathEntryFinder.find_spec()` sont recommandées, respectivement. Vous pouvez utiliser `importlib.util.spec_from_loader()` pour faciliter le portage (contribution de *Brett Cannon* dans bpo-42134).

— L'utilisation de `importlib.abc.PathEntryFinder.find_loader()` par le système d'importation lève maintenant un `ImportWarning`. Son remplacement par `importlib.abc.PathEntryFinder.`

`find_spec()` est recommandé. Vous pouvez utiliser `importlib.util.spec_from_loader()` pour faciliter le portage (contribution de *Brett Cannon* dans bpo-43672).

— Plusieurs implémentations de `importlib.abc.MetaPathFinder.find_module()` (`importlib.machinery.BuiltinImporter.find_module()`, `importlib.machinery.FrozenImporter.find_module()`, `importlib.machinery.WindowsRegistryFinder.find_module()`, `importlib.machinery.PathFinder.find_module()` et `importlib.abc.MetaPathFinder.find_module()`), `importlib.abc.PathEntryFinder.find_module()` (`importlib.machinery.FileFinder.find_module()`) et `importlib.abc.PathEntryFinder.find_loader()` (`importlib.machinery.FileFinder.find_loader()`) lèvent maintenant `DeprecationWarning` et seront retirées dans Python 3.12 (elles étaient documentées comme obsolètes depuis Python 3.4) (contribution de *Brett Cannon* dans bpo-42135).

— `importlib.abc.Finder` est obsolète (incluant sa seule méthode, `find_module()`). Les deux classes `importlib.abc.MetaPathFinder` et `importlib.abc.PathEntryFinder` n'héritent plus de `Finder`. Les utilisateurs doivent hériter d'une de ces deux classes le cas échéant (contribution de *Brett Cannon* dans bpo-42135).

— Les déclarations d'obsolescence de `imp`, `importlib.find_loader()`, `importlib.util.set_package_wrapper()`, `importlib.util.set_loader_wrapper()`, `importlib.util.module_for_loader()`, `pkgutil.ImpImporter` et `pkgutil.ImpLoader` ont étés mises à jour pour indiquer Python 3.12 comme la version où elles seront retirées (elles lèvent un `DeprecationWarning` depuis les précédentes versions de Python) (contribution de *Brett Cannon* dans bpo-43720).

— Le système d'importation utilise maintenant l'attribut `__spec__` des modules avant de se rabattre sur `module_repr()` pour la méthode `__repr__()` des modules. Le retrait de l'utilisation de `module_repr()` est planifié pour Python 3.12 (contribution de *Brett Cannon* dans bpo-42137).

— `importlib.abc.Loader.module_repr()`, `importlib.machinery.FrozenLoader.module_repr()` et `importlib.machinery.BuiltinLoader.module_repr()` sont obsolètes et seront retirées dans Python 3.12 (contribution de *Brett Cannon* dans bpo-42136).

— `sqlite3.OptimizedUnicode` est non-documenté et désuète depuis Python 3.3, quand elle devenue un alias de `str`. Elle est maintenant déclarée obsolète et sera retiré dans Python 3.12 (contribution de *Erlend E. Aasland* dans bpo-42264).

— La fonction native non-documentée `sqlite3.enable_shared_cache` est maintenant obsolète et sera retirée dans Python 3.12. Son utilisation est fortement déconseillée dans la documentation de SQLite3. Voir la documentation de SQLite3 (en anglais) pour plus de détails. Si un cache partagé est nécessaire, ouvrez la base de données en mode URI en utilisant le paramètre de requête `cache=shared` (contribution de *Erlend E. Aasland* dans bpo-24464).

— Les méthodes suivantes du module `threading` sont maintenant obsolètes :
  — `threading.currentThread` => `threading.current_thread()`;
  — `threading.activeCount` => `threading.active_count()`;
  — `threading.Condition.notifyAll` => `threading.Condition.notify_all()`;
  — `threading.Event.isSet` => `threading.Event.is_set()`;
  — `threading.Thread.setName` => `threading.Thread.name`;
  — `threading.thread.getName` => `threading.Thread.name`;
  — `threading.Thread.isDaemon` => `threading.Thread.daemon`;
  — `threading.Thread.setDaemon` => `threading.Thread.daemon`.

  (Contributed by Jelle Zijlstra in gh-87889.)

— `pathlib.Path.link_to()` est obsolète et sera retirée dans Python 3.12. Utilisez plutôt `pathlib.Path.hardlink_to()` (contribution de *Barney Gale* dans bpo-39950).

— `cgi.log()` est obsolète et sera retirée dans Python 3.12 (contribution de *Inada Naoki* dans bpo-41139).

— Les fonctionnalités suivantes de `ssl` sont obsolètes depuis Python 3.6, Python 3.7 ou *OpenSSL* 1.1.0 et seront retirées dans 3.11 :
  — `OP_NO_SSLv2`, `OP_NO_SSLv3`, `OP_NO_TLSv1`, `OP_NO_TLSv1_1`, `OP_NO_TLSv1_2` et `OP_NO_TLSv1_3` sont remplacés par `sslSSLContext.minimum_version` et `sslSSLContext.maximum_version`;
  — `PROTOCOL_SSLv2`, `PROTOCOL_SSLv3`, `PROTOCOL_SSLv23`, `PROTOCOL_TLSv1`, `PROTOCOL_TLSv1_1`, `PROTOCOL_TLSv1_2` et `PROTOCOL_TLS` sont obsolètes et remplacés

par `PROTOCOL_TLS_CLIENT` et `PROTOCOL_TLS_SERVER`;
  - `wrap_socket()` est remplacée par `ssl.SSLContext.wrap_socket()`;
  - `match_hostname()`;
  - `RAND_pseudo_bytes()`, `RAND_egd()`;
  - Les fonctionnalités NPN (*Next Protocol Negotiation*) telles que `ssl.SSLSocket.selected_npn_protocol()` et `ssl.SSLContext.set_npn_protocols()` sont remplacées par ALPN (*Application Layer Protocol Negotiation*).
- Le débogage des fils d'exécution (variable d'environnement `PYTHONTHREADDEBUG`) est obsolète dans Python 3.10 et sera retiré dans Python 3.12. Cette fonctionnalité nécessite un interpréteur compilé avec les paramètres de débogage (contribution de *Victor Stinner* dans bpo-44584).
- Importing from the `typing.io` and `typing.re` submodules will now emit `DeprecationWarning`. These submodules will be removed in a future version of Python. Anything belonging to these submodules should be imported directly from `typing` instead. (Contributed by Sebastian Rittau in bpo-38291.)

9 Retrait

- Les méthodes spéciales `__int__`, `__float__`, `__floordiv__`, `__mod__`, `__divmod__`, `__rfloordiv__`, `__rmod__` et `__rdivmod__` sont retirées de la classe `complex`. Elles levaient toujours une `TypeError` (contribution de *Serhiy Storchaka* dans bpo-41974).
- La méthode `ParserBase.error()` du module privé et non-documenté `_markupbase` a été retirée. `html.parser.HTMLParser` est la seule sous-classe de `ParserBase` et son implémentation de la méthode `error()` a été retirée dans Python 3.5 (contribution de *Berker Peksag* dans bpo-31844).
- Retrait de l'attribut `unicodedata.ucnhash_CAPI` qui était un objet `PyCapsule` interne. La structure privée associée `_PyUnicode_Name_CAPI` a été déplacée vers l'API C interne (contribution de *Victor Stinner* dans bpo-42157).
- Retrait du module `parser` qui était obsolète depuis 3.9 étant donné la transition vers le nouvel analyseur syntaxique PEG. Retrait aussi des fichiers source et en-tête C qui n'étaient utilisés que par l'ancien analyseur, ceci comprend `node.h`, `parser.h`, `graminit.h` et `grammar.h`.
- Retrait des fonctions `PyParser_SimpleParseStringFlags`, `PyParser_SimpleParseStringFlagsFilename`, `PyParser_SimpleParseFileFlags` et `PyNode_Compile` de l'API C publique. Elles étaient obsolètes depuis 3.9 étant donné la transition vers l'analyseur syntaxique *PEG*.
- Retrait du module `formatter` qui était obsolète depuis Python 3.4. Il était quelque peu désuet, peu utilisé et non-testé. Il était originalement planifié pour le retrait dans Python 3.6, mais ces retraits ont été retardés jusqu'à la fin de la prise en charge de Python 2.7. Les utilisateurs de ce module doivent copier les classes qu'ils utilisent dans leur propre code (contribution de *Dong-hee Na* et *Terry J. Reedy* dans bpo-42299).
- Retrait de la fonction `PyModule_GetWarningsModule()` qui était maintenant inutile depuis la conversion de `_warnings` en module natif dans 2.6 (contribution de *Hai Shi* dans bpo-42599).
- Retrait des alias obsolètes collections-abstract-base-classes du module `collections` (contribution de *Victor Stinner* dans bpo-37324).
- Le paramètre `loop` a été retiré de la majorité de l'API haut-niveau d'`asyncio`. Il était obsolète depuis Python 3.8. La motivation pour ce changement avait plusieurs facettes :

  1. ceci simplifie l'API haut-niveau ;
  2. les fonctions dans l'API haut niveau reçoivent implicitement la boucle d'évènements qui tourne dans le présent fil d'exécution depuis Python 3.7. Il n'y a aucun besoin de passer la boucle d'évènements à l'API dans la majorité de son utilisation normale ;
  3. le passage de la boucle d'évènements cause facilement des erreurs quand plusieurs boucles qui tournent dans plusieurs fils d'exécution sont impliquées.

  Prenez note que l'API bas-niveau accepte toujours le paramètre `loop`. Voir *Changements dans l'API Python* pour plus d'exemples sur comment remplacer le code existant

  (contribution de *Yurii Karabas*, *Andrew Svetlov*, *Yury Selivanov* et *Kyle Stanley* dans bpo-42392).

10 Portage vers Python 3.10

Cette section liste les changements mentionnés préalablement et autres améliorations qui peuvent demander des changements à votre code.

10.1 Changements à la syntaxe de Python

— Deprecation warning is now emitted when compiling previously valid syntax if the numeric literal is immediately followed by a keyword (like in `0in x`). In future releases it will be changed to syntax warning, and finally to a syntax error. To get rid of the warning and make the code compatible with future releases just add a space between the numeric literal and the following keyword. (Contributed by Serhiy Storchaka in bpo-43833.)

10.2 Changements dans l'API Python

— Les paramètres *etype* des fonctions `format_exception()`, `format_exception_only()` et `print_exception()` du module `traceback` ont étés renommés *exc* (contribution de *Zackery Spytz* et *Matthias Bussonnier* dans bpo-26389).

— `atexit` : à la fin de l'exécution de Python, si une fonction de rappel qui a été enregistrée avec `atexit.register()` échoue, son exception est maintenant journalisée. Préalablement, seulement certaines exceptions étaient journalisées et la dernière exception était toujours ignorée de façon silencieuse (contribution de *Victor Stinner* dans bpo-42639).

— Les génériques de `collections.abc.Callable` aplatissent maintenant les paramètres de types tel qu'il est fait par `typing.Callable`. Ceci veut dire que `collections.abc.Callable[[int, str], str]` aura un attribut `__args__` de valeur `(int, str, str)` ; préalablement, il aurait été `([int, str], str)`. Les codes qui accèdent aux arguments par `typing.get_args()` ou `__args__` doivent tenir compte de ce changement. De plus, `TypeError` sera levée pour des formes invalides de paramétrisation de `collections.abc.Callable`, ce qui aurait pu passer silencieusement dans Python 3.9 (contribution de *Ken Jin* dans bpo-42195).

— `socket.htons()` et `socket.ntohs()` lèvent maintenant `OverflowError` plutôt que `DeprecationWarning` si le paramètre passé est trop grand pour être stocké dans un entier 16-bit non-signé (contribution de *Erlend E. Aasland* dans bpo-42393).

— Le paramètre `loop` a été retiré de la majorité de l'API haut-niveau d'`asyncio` suite à la déclaration de son obsolescence dans Python 3.8.

Une coroutine qui ressemble actuellement à ceci :

```
async def foo(loop):
    await asyncio.sleep(1, loop=loop)
```

Doit être remplacée par ceci :

```
async def foo():
    await asyncio.sleep(1)
```

Si `foo()` a été spécifiquement conçue pour *ne pas* s'exécuter dans le fil d'exécution courant, (par exemple pour tourner dans la boucle d'exécution d'un autre fil), l'utilisation de `asyncio.run_coroutine_threadsafe()` est probablement plus appropriée.

(contribution de *Yurii Karabas*, *Andrew Svetlov*, *Yury Selivanov* et *Kyle Stanley* dans bpo-42392).

— Le constructeur de `types.FunctionType` hérite maintenant des définitions natives si le dictionnaire *globals* n'a pas de clé `"__builtins__"`, plutôt que d'utiliser `{"None": None}` pour les définitions natives : même comportement que celui des fonctions `eval()` et `exec()`. Définir une fonction avec `def function(...`

`):  ...` en Python n'est pas affecté, les globales ne peuvent pas être changées avec cette syntaxe : elles héritent aussi des définitions natives courantes (contribution de *Victor Stinner* dans bpo-42990).

10.3 Changements dans l'API C

— Les fonctions de l'API C `PyParser_SimpleParseStringFlags`, `PyParser_SimpleParseStringFlagsFilename`, `PyParser_SimpleParseFileFlags` et `PyNode_Compile` ainsi que `struct _node`, le type utilisé par ces fonctions, ont été retirés suite à la transition vers le nouvel analyseur syntaxique *PEG*.

Le code source doit maintenant être compilé directement en un objet code, en utilisant par exemple `Py_CompileString()`. L'objet code ainsi produit peut ensuite être évalué, en utilisant par exemple `PyEval_EvalCode()`.

Spécifiquement :

— Un appel à `PyParser_SimpleParseStringFlags` suivi de `PyNode_Compile` peut être remplacé par `Py_CompileString()` ;

— Il n'y a pas de remplacement immédiat pour `PyParser_SimpleParseFileFlags`. Pour compiler du code à partir d'un argument `FILE *`, vous devez lire le fichier en C et passer le tampon résultant à `Py_CompileString()`.

— Pour compiler un fichier à partir du `char *` de son nom, ouvrez le fichier de façon explicite, lisez son contenu et compilez le résultat. Une façon de faire est d'utiliser le module `io` en conjonction avec `PyImport_ImportModule()`, `PyObject_CallMethod()`, `PyBytes_AsString()` et `Py_CompileString()`, comme dans l'esquisse qui suit (les déclarations et la gestion d'erreurs ne sont pas incluses) :

```
io_module = Import_ImportModule("io");
fileobject = PyObject_CallMethod(io_module, "open", "ss", filename, "rb");
source_bytes_object = PyObject_CallMethod(fileobject, "read", "");
result = PyObject_CallMethod(fileobject, "close", "");
source_buf = PyBytes_AsString(source_bytes_object);
code = Py_CompileString(source_buf, filename, Py_file_input);
```

— Pour les objets `FrameObject`, le champ `f_lasti` représente maintenant en décalage par *wordcode* plutôt qu'un simple décalage dans la chaîne de code intermédiaire. Il est donc nécessaire de multiplier ce nombre par 2 pour l'utiliser dans les API qui attendent un décalage par octet (tel que `PyCode_Addr2Line()`, par exemple). Prenez aussi note que le champ `f_lasti` de `FrameObject` n'est pas considéré stable : utilisez plutôt `PyFrame_GetLineNumber()`.

11 Changements au code intermédiaire CPython

— The `MAKE_FUNCTION` instruction now accepts either a dict or a tuple of strings as the function's annotations. (Contributed by Yurii Karabas and Inada Naoki in bpo-42202.)

12 Changements à la compilation

— **PEP 644** : la version minimale d'*OpenSSL* pour Python est maintenant 1.1.1. *OpenSSL* 1.0.2 n'est plus prise en charge (contribution de *Christian Heimes* dans bpo-43669).
— Les fonctions de *C99* `snprintf()` et `vsnprintf()` sont maintenant nécessaires à la compilation de Python (contribution de *Victor Stinner* dans bpo-36020).
— La version minimale de SQLite pour le module `sqlite3` est maintenant 3.7.15 (contribution de *Sergey Fedoseev* et *Erlend E. Aasland* dans bpo-40744 et bpo-40810).
— Le module `atexit` doit maintenant toujours être compilé en tant que module natif (contribution de *Victor Stinner* dans bpo-42639).
— Ajout de l'option `--disable-test-modules` au script `configure` : ne pas compiler ni installer les modules de tests (contribution de *Xavier de Gaye*, *Thomas Petazzoni* et *Peixing Xin* dans bpo-27640).
— Ajout de l'option `--with-wheel-pkg-dir=PATH` au script `./configure`. Si elle est passée, le module `ensurepip` recherche les paquets *wheel* de `setuptools` et de `pip` dans ce répertoire : si les deux sont présents, ces paquets *wheel* sont utilisés plutôt que ceux inclus avec *ensurepip*.

  La politique de paquets de certaines distributions Linux déconseille l'inclusion des dépendances. Par exemple, Fedora installe les paquets *wheel* dans le répertoire `/usr/share/python-wheels/` et n'installe pas le paquet `ensurepip._bundled`

  (contribution de *Victor Stinner* dans bpo-42856).
— Ajout d'une nouvelle option `--without-static-libpython` au script `configure` pour ne pas compiler la bibliothèque statique `libpythonMAJOR.MINOR.a` et ne pas installer le fichier objet `python.o`

  (contribution de *Victor Stinner* dans bpo-43103).
— Le script `configure` utilise maintenant l'utilitaire `pkg-config` lorsqu'il est disponible pour repérer l'emplacement des en-têtes et des bibliothèques *Tcl/Tk*. Comme auparavant, ces emplacements peuvent être spécifiés de façon explicite par les options de configuration `--with-tcltk-includes` et `--with-tcltk-libs` (contribution de *Manolis Stamatogiannakis* et bpo-42603).
— Ajout de l'option `--with-openssl-rpath` au script `configure`. Cette option simplifie la compilation de Python avec une installation personnalisée d'OpenSSL, par exemple : `./configure --with-openssl=/path/to/openssl --with-openssl-rpath=auto` (contribution de *Christian Heimes* dans bpo-43466).

13 Changements à l'API C

13.1 PEP 652 : maintenance d'une ABI stable

L'interface binaire-programme (*Application Binary Interface*, *ABI* en anglais) stable pour les modules d'extension ou pour l'intégration de Python est maintenant définie de façon explicite. Le document stable décrit les garanties de stabilité pour l'API C et pour l'interface binaire-programme ainsi que les bonnes pratiques pour travailler avec l'interface binaire-programme stable

(contribution de *Petr Viktorin* dans **PEP 652** et dans bpo-43795).

13.2 Nouvelles fonctionnalités

— Le résultat de `PyNumber_Index()` est maintenant exactement de type `int`. Auparavant, le résultat pouvait être une instance d'une classe dérivée de `int` (contribution de *Serhiy Storchaka* dans bpo-40792).
— Ajout d'un nouveau champ `orig_argv` à la structure `PyConfig` : la liste des arguments de ligne de commande originalement passée à l'exécutable Python (contribution de *Victor Stinner* dans bpo-23427).
— Les macros `PyDateTime_DATE_GET_TZINFO()` et `PyDateTime_TIME_GET_TZINFO()` ont été ajoutées pour récupérer l'attribut `tzinfo` des objets `datetime.datetime` et `datetime.time` (contribution de *Zackery Spytz* dans bpo-30155).
— Ajout d'une fonction `PyCodec_Unregister()` pour désinscrire une fonction de recherche de codecs (contribution de *Hai Shi* dans bpo-41842).
— Ajout de la fonction `PyIter_Send()` pour acheminer une valeur dans un itérateur sans lever une exception `StopIteration` (contribution de *Vladimir Matveev* dans bpo-41756).
— Ajout de `PyUnicode_AsUTF8AndSize()` à l'API C limitée (contribution d'*Alex Gaynor* dans bpo-41784).
— Ajout de la fonction `PyModule_AddObjectRef()` : comparable à `PyModule_AddObject()` mais ne s'accapare pas une référence à la valeur s'il y a réussite (contribution de *Victor Stinner* dans bpo-1635741).
— Ajout des fonctions `Py_NewRef()` et `Py_XNewRef()` pour incrémenter le compteur des références d'un objet et renvoyer l'objet (contribution de *Victor Stinner* dans bpo-42262).
— Les fonctions `PyType_FromSpecWithBases()` et `PyType_FromModuleAndSpec()` n'acceptent maintenant qu'une seule classe pour l'argument *bases* (contribution *Serhiy Storchaka* dans bpo-42423).
— La fonction `PyType_FromModuleAndSpec()` accepte maintenant `NULL` dans l'emplacement `tp_doc` (contribution de *Hai Shi* dans bpo-41832).
— La fonction `PyType_GetSlot()` peut maintenant accepter des types statiques (contribution de *Hai Shi* et *Petr Viktorin* dans bpo-41073).
— Ajout d'une nouvelle fonction `PySet_CheckExact()` à l'API C pour vérifier si un objet est une instance de `set` mais pas une instance d'un sous-type (contribution de *Pablo Galindo* dans bpo-43277).
— Ajout de `PyErr_SetInterruptEx()` qui accepte le numéro d'un signal à simuler (contribution d'*Antoine Pitrou* dans bpo-43356).
— L'API C limitée est maintenant prise en charge si Python est compilé en mode débogage (si la macro `Py_DEBUG` est définie). Dans l'API C limitée, les fonctions `Py_INCREF()` et `Py_DECREF()` sont maintenant implémentées en appel de fonctions opaques, plutôt que d'accéder directement au champ `PyObject.ob_refcnt`. C'est le cas si Python est compilé en mode débogage et que la macro `Py_LIMITED_API` vise Python 3.10 ou plus récent. Il est possible de prendre en charge l'API C limitée en mode débogage, car la structure `PyObject` est la même dans les versions standard et dans les versions débogage depuis Python 3.8 (voir bpo-36465).

  L'API C limitée n'est toujours pas prise en charge dans les compilations spéciales avec l'option `--with-trace-refs` (macro `Py_TRACE_REFS`) (Contribution de *Victor Stinner* dans bpo-43688).
— Ajout de la fonction `Py_Is(x, y)` pour vérifier si l'objet *x* est l'objet *y*, tout comme `x is y` en Python. Ajout aussi des fonctions `Py_IsNone()`, `Py_IsTrue()` et `Py_IsFalse()` pour vérifier si l'objet est le singleton `None`, le singleton `True` ou le singleton `False`, respectivement (contribution de *Victor Stinner* dans bpo-43753).
— Ajout de fonctions pour contrôler le ramasse-miettes en C : `PyGC_Enable()`, `PyGC_Disable()` et `PyGC_IsEnabled()`. Ces fonctions permettent d'activer, de désactiver et d'interroger l'état du ramasse-miettes dans du code C sans avoir à importer le module `gc`.
— Ajout d'une nouvelle option `Py_TPFLAGS_DISALLOW_INSTANTIATION` pour les types. Elle empêche la création d'instances de ce type (contribution de *Victor Stinner* dans bpo-43916).
— Ajout d'une nouvelle option `Py_TPFLAGS_IMMUTABLETYPE` pour les types qui permet de créer des objets de types immuable : les attributs du type ne peuvent pas être affectés ou supprimés (contribution de *Victor Stinner* et d'*Erlend E. Aasland* dans bpo-43908).

13.3 Portage vers Python 3.10

— The `PY_SSIZE_T_CLEAN` macro must now be defined to use `PyArg_ParseTuple()` and `Py_BuildValue()` formats which use `# : es#, et#, s#, u#, y#, z#, U#` and `Z#`. See arg-parsing and **PEP 353**. (Contributed by Victor Stinner in bpo-40943.)

— Étant donné la conversion de `Py_REFCNT()` en fonction statique en-ligne, `Py_REFCNT(obj) = new_refcnt` doit être remplacé par `Py_SET_REFCNT(obj, new_refcnt)` : voir `Py_SET_REFCNT()` (disponible depuis Python 3.9). Pour la rétro-compatibilité, cette macro peut être utilisée :

```
#if PY_VERSION_HEX < 0x030900A4
#  define Py_SET_REFCNT(obj, refcnt) ((Py_REFCNT(obj) = (refcnt)), (void)0)
#endif
```

(contribution de *Victor Stinner* dans bpo-39573).

— Appeler `PyDict_GetItem()` sans avoir obtenu le GIL était autorisé pour des raisons historiques. Ceci n'est plus permis (contribution de *Victor Stinner* dans bpo-40839).

— `PyUnicode_FromUnicode(NULL, size)` et `PyUnicode_FromStringAndSize(NULL, size)` lèvent maintenant un `DeprecationWarning`. Vous devez utiliser `PyUnicode_New()` pour allouer un objet Unicode sans donnée initiale (contribution de *Inada Naoki* dans bpo-36346).

— La structure privée `_PyUnicode_Name_CAPI` de l'API PyCapsule `unicodedata.ucnhash_CAPI` a été déplacée dans l'API C interne. (Contribution par Victor Stinner dans bpo-42157.)

— `Py_GetPath()`, `Py_GetPrefix()`, `Py_GetExecPrefix()`, `Py_GetProgramFullPath()`, `Py_GetPythonHome()` and `Py_GetProgramName()` functions now return `NULL` if called before `Py_Initialize()` (before Python is initialized). Use the new init-config API to get the init-path-config. (Contributed by Victor Stinner in bpo-42260.)

— `PyList_SET_ITEM()`, `PyTuple_SET_ITEM()` and `PyCell_SET()` macros can no longer be used as l-value or r-value. For example, `x = PyList_SET_ITEM(a, b, c)` and `PyList_SET_ITEM(a, b, c) = x` now fail with a compiler error. It prevents bugs like `if (PyList_SET_ITEM (a, b, c) < 0) ...` test. (Contributed by Zackery Spytz and Victor Stinner in bpo-30459.)

— The non-limited API files `odictobject.h`, `parser_interface.h`, `picklebufobject.h`, `pyarena.h`, `pyctype.h`, `pydebug.h`, `pyfpe.h`, and `pytime.h` have been moved to the `Include/cpython` directory. These files must not be included directly, as they are already included in `Python.h`; see api-includes. If they have been included directly, consider including `Python.h` instead. (Contributed by Nicholas Sim in bpo-35134.)

— Use the `Py_TPFLAGS_IMMUTABLETYPE` type flag to create immutable type objects. Do not rely on `Py_TPFLAGS_HEAPTYPE` to decide if a type object is mutable or not; check if `Py_TPFLAGS_IMMUTABLETYPE` is set instead. (Contributed by Victor Stinner and Erlend E. Aasland in bpo-43908.)

— The undocumented function `Py_FrozenMain` has been removed from the limited API. The function is mainly useful for custom builds of Python. (Contributed by Petr Viktorin in bpo-26241.)

13.4 Obsolescence

— La fonction `PyUnicode_InternImmortal()` est maintenant obsolète et sera retirée dans Python 3.12 : utilisez plutôt `PyUnicode_InternInPlace()` (contribution de *Victor Stinner* dans bpo-41692).

13.5 Retrait

— Retrait des fonctions `Py_UNICODE_str*` manipulant des chaînes `Py_UNICODE*` (contribution de *Inada Naoki* dans bpo-41123) :
  — `Py_UNICODE_strlen` : utilisez `PyUnicode_GetLength()` ou `PyUnicode_GET_LENGTH` ;
  — `Py_UNICODE_strcat` : utilisez `PyUnicode_CopyCharacters()` ou `PyUnicode_FromFormat()` ;
  — `Py_UNICODE_strcpy`, `Py_UNICODE_strncpy` : utilisez `PyUnicode_CopyCharacters()` ou `PyUnicode_Substring()` ;
  — `Py_UNICODE_strcmp` : utilisez `PyUnicode_Compare()` ;
  — `Py_UNICODE_strncmp` : utilisez `PyUnicode_Tailmatch()` ;
  — `Py_UNICODE_strchr`, `Py_UNICODE_strrchr` : utilisez `PyUnicode_FindChar()`.

— Retrait de `PyUnicode_GetMax()`. Vous devez migrer vers les nouvelles API de la **PEP 393** (contribution de *Inada Naoki* dans bpo-41103).

— Retrait de `PyLong_FromUnicode()`. Vous devez migrer vers `PyLong_FromUnicodeObject()` (contribution de *Inada Naoki* dans bpo-41103).

— Retrait de `PyUnicode_AsUnicodeCopy()`. Vous devez utiliser `PyUnicode_AsUCS4Copy()` ou `PyUnicode_AsWideCharString()` (contribution de *Inada Naoki* dans bpo-41103).

— Retrait de la variable `_Py_CheckRecursionLimit` : elle a été remplacée par le champ `ceval.recursion_limit` de la structure `PyInterpreterState` (contribution de *Victor Stinner* dans bpo-41834).

— Retrait des macros non-documentées `Py_ALLOW_RECURSION` et `Py_END_ALLOW_RECURSION` et du champ `recursion_critical` de la structure `PyInterpreterState` (contribution de *Serhiy Storchaka* dans bpo-41936).

— Retrait de la fonction non-documentée `PyOS_InitInterrupts()`. L'initialisation de Python installe déjà les gestionnaires de signaux de façon implicite : voir `PyConfig.install_signal_handlers` (contribution de *Victor Stinner* dans bpo-41713).

— Retrait de la fonction `PyAST_Validate()`. Il n'est plus possible de construire un objet *AST* (type `mod_ty`) avec l'API C publique. La fonction était déjà retirée de l'API C limitée (**PEP 384**) (contribution de *Victor Stinner* dans bpo-43244).

— Retrait du fichier d'en-tête `symtable.h` et des fonctions non-documentées :
  — `PyST_GetScope()` ;
  — `PySymtable_Build()` ;
  — `PySymtable_BuildObject()` ;
  — `PySymtable_Free()` ;
  — `Py_SymtableString()` ;
  — `Py_SymtableStringObject()`.

  Par erreur, la fonction `Py_SymtableString()` faisait partie de l'*ABI* stable, mais elle ne pouvait pas être utilisée, car le fichier d'en-tête `symtable.h` était exclu de l'API C limitée.

  Utilisez plutôt le module Python `symtable` (contribution de *Victor Stinner* dans bpo-43244).

— Retrait de `PyOS_ReadlineFunctionPointer()` des fichiers d'en-tête de l'API C limitée et de `python3.dll`, la bibliothèque qui fournit l'*ABI* stable sur Windows. Étant donné que la fonction prend un `FILE*` en argument, la stabilité de son interface binaire-programme ne peut pas être garantie (contribution de *Petr Viktorin* dans bpo-43868).

— Retrait des fichiers d'en-tête `ast.h`, `asdl.h` et `Python-ast.h`. Ces fonctions étaient non-documentées et exclues de l'API C limitée. La majorité des noms définis dans ces fichiers d'en-tête n'avaient pas le préfixe `Py` et pouvaient ainsi causer des conflits de noms. Par exemple, `Python-ast.h` définit une macro `Yield` qui était

en conflit avec le nom `Yield` utilisé dans le fichier d'en-tête `<winbase.h>` de Windows. Utilisez plutôt le module Python `ast` (contribution de *Victor Stinner* dans bpo-43244).

— Retrait des fonctions d'analyse syntaxique et de compilation qui utilisent le type `struct _mod` car l'API C publique pour manipuler les *AST* a été retirée :
  — `PyAST_Compile()`;
  — `PyAST_CompileEx()`;
  — `PyAST_CompileObject()`;
  — `PyFuture_FromAST()`;
  — `PyFuture_FromASTObject()`;
  — `PyParser_ASTFromFile()`;
  — `PyParser_ASTFromFileObject()`;
  — `PyParser_ASTFromFilename()`;
  — `PyParser_ASTFromString()`;
  — `PyParser_ASTFromStringObject()`.

  Ces fonctions étaient non-documentées et exclues de l'API C limitée (contribution de *Victor Stinner* dans bpo-43244).
— Retrait du fichier d'en-tête `pyarena.h` et des fonctions :
  — `PyArena_New()`;
  — `PyArena_Free()`;
  — `PyArena_Malloc()`;
  — `PyArena_AddPyObject()`.

  Ces fonctions étaient non-documentées et exclues de l'API C limitée ; elles n'étaient utilisées que par le compilateur dans l'implémentation interne (contribution de *Victor Stinner* dans bpo-43244).
— Retrait du champ `PyThreadState.use_tracing` pour optimiser Python (contribution de *Mark Shannon* dans bpo-43760).

14 Notable security feature in 3.10.7

Converting between `int` and `str` in bases other than 2 (binary), 4, 8 (octal), 16 (hexadecimal), or 32 such as base 10 (decimal) now raises a `ValueError` if the number of digits in string form is above a limit to avoid potential denial of service attacks due to the algorithmic complexity. This is a mitigation for CVE-2020-10735. This limit can be configured or disabled by environment variable, command line flag, or `sys` APIs. See the integer string conversion length limitation documentation. The default limit is 4300 digits in string form.

15 Notable security feature in 3.10.8

The deprecated `mailcap` module now refuses to inject unsafe text (filenames, MIME types, parameters) into shell commands. Instead of using such text, it will warn and act as if a match was not found (or for test commands, as if the test failed). (Contributed by Petr Viktorin in gh-98966.)

16 Notable Changes in 3.10.12

16.1 tarfile

— The extraction methods in `tarfile`, and `shutil.unpack_archive()`, have a new a *filter* argument that allows limiting tar features than may be surprising or dangerous, such as creating files outside the destination directory. See tarfile-extraction-filter for details. In Python 3.12, use without the *filter* argument will show a `DeprecationWarning`. In Python 3.14, the default will switch to `'data'`. (Contributed by Petr Viktorin in **PEP 706**.)

17 Notable changes in 3.10.15

17.1 ipaddress

— Fixed `is_global` and `is_private` behavior in `IPv4Address`, `IPv6Address`, `IPv4Network` and `IPv6Network`.

17.2 email

— Headers with embedded newlines are now quoted on output.

The `generator` will now refuse to serialize (write) headers that are improperly folded or delimited, such that they would be parsed as multiple headers or joined with adjacent data. If you need to turn this safety feature off, set `verify_generated_headers`. (Contributed by Bas Bloemsaat and Petr Viktorin in gh-121650.)

— `email.utils.getaddresses()` and `email.utils.parseaddr()` now return `('', '')` 2-tuples in more situations where invalid email addresses are encountered, instead of potentially inaccurate values. An optional *strict* parameter was added to these two functions : use `strict=False` to get the old behavior, accepting malformed inputs. `getattr(email.utils, 'supports_strict_parsing', False)` can be used to check if the *strict* paramater is available. (Contributed by Thomas Dwyer and Victor Stinner for gh-102988 to improve the CVE-2023-27043 fix.)

Index

P

R

V