What's New in Python

发布 *3.10.5*

A. M. Kuchling

六月 14, 2022

Python Software Foundation
Email: docs@python.org

Contents

发布版本 3.10.5

日期 六月 14, 2022

编者 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 摘要 -- 发布重点

新的语法特性：

- **PEP 634**, 结构化模式匹配: 规范说明
- **PEP 635**, 结构化模式匹配: 动机与理由
- **PEP 636**, 结构化模式匹配: 教程
- bpo-12782，加圆括号的上下文管理器现在正式被允许使用。

标准库中的新特性：

- **PEP 618**，向 zip 添加可选的长度检查。

解释器的改进：

- **PEP 626**，在调试和其他工具中使用精确的行号。

新的类型标注特性：

- **PEP 604**，允许 X | Y 形式的联合类型写法
- **PEP 613**，显式类型别名
- **PEP 612**，形参规格变量

重要的弃用、移除或限制：

- **PEP 644**，要求 OpenSSL 1.1.1 或更新的版本
- **PEP 632**，弃用 distutils 模块。
- **PEP 623**，弃用并准备移除 PyUnicodeObject 中的 wstr 成员。
- **PEP 624**，移除 Py_UNICODE 编码器 API
- **PEP 597**，增加可选的 EncodingWarning

2 新的特性

2.1 带圆括号的上下文管理器

现在已支持使用外层圆括号来使多个上下文管理器可以连续多行地书写。这允许将过长的上下文管理器集能够以与之前 import 语句类似的方式格式化为多行的形式。例如，以下这些示例写法现在都是有效的:

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

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

with (CtxManager1() as example,
```

(下页继续)

```
      CtxManager2()):
    ...

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

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

在被包含的分组末尾过可以使用一个逗号作为结束：

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

这个新语法使用了新解析器的非 LL(1) 功能。请查看 **PEP 617** 来了解更多细节。

（由 Guido van Rossum, Pablo Galindo 和 Lysandros Nikolaou 在 bpo-12782 和 bpo-40334 中贡献。）

2.2 更清楚的错误消息

SyntaxError

现在当解析包含有未关闭括号的代码时解释器会包括未关闭括号的位置而不是显示 *SyntaxError: unexpected EOF while parsing* 并指向某个不正确的位置。例如，考虑以下代码（注意未关闭的“{ ”）：

```
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()
```

之前版本的解释器会报告令人迷惑的语法错误位置：

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

但在 Python 3.10 中则会发出信息量更多的错误提示：

```
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
```

类似地，涉及未关闭字符串字面值 (单重引号和三重引号) 的错误现在会指向字符串的开头而不是报告 EOF/EOL。

这些改进的灵感来自 PyPy 解释器之前所进行的工作。

（由 Pablo Galindo 在 bpo-42864 以及 Batuhan Taskaya 在 bpo-40176 中贡献。）

解释器所引发的 `SyntaxError` 异常现在将高亮构成语法错误本身的完整异常错误内容，而不是仅提示检测到问题的位置。这样，不再（同 Python 3.10 之前那样）仅显示：

```
>>> 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
```

现在 Python 3.10 将这样显示异常：

```
>>> 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
```

这个改进是由 Pablo Galindo 在 bpo-43914 中贡献的。

大量新增的专门化 `SyntaxError` 异常消息已被添加。其中最主要的一些如下所示：

- 在代码块之前缺失 `:`：

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

  （由 Pablo Galindo 在 bpo-42997 中贡献。）

- 在推导式的目标中有不带圆括号的元组：

  ```
  >>> {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?
  ```

  （由 Pablo Galindo 在 bpo-43017 中贡献。）

- 在多项集字面值中和表达式之间缺失逗号：

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

  （由 Pablo Galindo 在 bpo-43822 中贡献。）

- 多个异常类型不带圆括号：

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

  （由 Pablo Galindo 在 bpo-43149 中贡献。）

- 字典字面值中缺失 `:` 和值：

```
>>> 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
```

（由 Pablo Galindo 在 bpo-43823 中贡献）

- try 代码块不带 except 或 finally 代码块:

```
>>> try:
...     x = 2
... something = 3
  File "<stdin>", line 3
    something  = 3
    ^^^^^^^^^
SyntaxError: expected 'except' or 'finally' block
```

（由 Pablo Galindo 在 bpo-44305 中贡献。）

- 在比较中使用 = 而不是 ==:

```
>>> 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 '='?
```

（由 Pablo Galindo 在 bpo-43797 中贡献。）

- 在 f-字符串中使用 *:

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

（由 Pablo Galindo 在 bpo-41064 中贡献。）

IndentationError

许多 IndentationError 异常现在具有更多上下文来提示是何种代码块需要缩进，包括语句的位置:

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

AttributeError

当打印 `AttributeError` 时，`PyErr_Display()` 将提供引发异常的对象中类似属性名称的建议：

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

（由 Pablo Galindo 在 bpo-38530 中贡献。）

> **警告：** 请注意如果未调用 `PyErr_Display()` 来显示错误则此特性将没有效果，这可能发生在使用了某些其他自定义错误显示函数的时候。这在某些 REPL 例如 IPython 上是一种常见的情况。

NameError

当打印解释器所引发的 `NameError` 时，`PyErr_Display()` 将提供引发异常的函数中类似变量名称的建议：

```
>>> 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?
```

（由 Pablo Galindo 在 bpo-38530 中贡献。）

> **警告：** 请注意如果未调用 `PyErr_Display()` 来显示错误则此特性将没有效果，这可能发生在使用了某些其他自定义错误显示函数的时候。这在某些 REPL 例如 IPython 中是一种常见的情况。

2.3 PEP 626：在调试和其他工具中使用精确的行号

PEP 626 带来了更精确可靠的行号用于调试、性能分析和测试工具。所有被执行的代码行都会并且只有被执行的代码行才会生成带有正确行号的追踪事件。

帧对象的 `f_lineno` 属性将总是包含预期的行号。

代码对象的 `co_lnotab` 属性已被弃用并将在 3.12 中被移除。需要从偏移量转换为行号的代码应改用新的 `co_lines()` 方法。

2.4 PEP 634：结构化模式匹配

增加了采用模式加上相应动作的 *match* 语句和 *case* 语句的形式的结构化模式匹配。模式由序列、映射、基本数据类型以及类实例构成。模式匹配使得程序能够从复杂的数据类型中提取信息、根据数据结构实现分支，并基于不同的数据形式应用特定的动作。

语法与操作

模式匹配的通用语法如下:

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

match 语句接受一个表达式并将其值与以一个或多个 case 语句块形式给出的一系列模式进行比较。具体来说，模式匹配的操作如下:

1. 使用具有特定类型和形状的数据 (`subject`)
2. 针对 `subject` 在 `match` 语句中求值
3. 从上到下对 subject 与 `case` 语句中的每个模式进行比较直到确认匹配到一个模式。
4. 执行与被确认匹配的模式相关联的动作。
5. 如果没有确认到一个完全的匹配，则如果提供了使用通配符 _ 的最后一个 case 语句，则它将被用作已匹配模式。如果没有确认到一个完全的匹配并且不存在使用通配符的 case 语句，则整个 match 代码块不执行任何操作。

声明性方式

读者可能是通过 C, Java 或 JavaScript (以及其他许多语言) 中的 switch 语句将一个目标 (数据对象) 与一个字面值 (模式) 进行匹配的简单例子了解到模式匹配的概念的。switch 语句常常被用来将一个对象/表达式与包含在 case 语句中的字面值进行比较。

更强大的模式匹配例子可以在 Scala 和 Elixir 等语言中找到。这种结构化模式匹配方式是“声明性”的并且会显式地为所要匹配的数据指定条件（模式）。

虽然使用嵌套的“if”语句的“命令性”系列指令可以被用来完成类似结构化模式匹配的效果，但它没有“声明性”方式那样清晰。相反地，“声明性”方式指定了一个匹配所要满足的条件，并且通过其显式的模式使之更为易读。虽然结构化模式匹配可以采取将一个变量与一个 case 语句中的字面值进行比较的最简单形式来使用，但它对于 Python 的真正价值在于其针对目标类型和形状的处理操作。

简单模式：匹配一个字面值

让我们把这个例子看作是模式匹配的最简单形式：一个值，即主词，被匹配到几个字面值，即模式。在下面的例子中，`status` 是匹配语句的主词。模式是每个 case 语句，字面值代表请求状态代码。匹配后，将执行与该 case 相关的动作:

```
def http_error(status):
    match status:
        case 400:
            return "Bad request"
        case 404:
            return "Not found"
        case 418:
            return "I'm a teapot"
        case _:
            return "Something's wrong with the internet"
```

如果传给上述函数的 status 为 418，则会返回"I'm a teapot"。如果传给上述函数的 status 为 500，则带有 _ 的 case 语句将作为通配符匹配，并会返回"Something's wrong with the internet"。请注意最后一个代码块：变量名 _ 将作为 通配符并确保目标将总是被匹配。_ 的使用是可选的。

你可以使用 | （"or "）在一个模式中组合几个字面值:

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

无通配符的行为

如果我们修改上面的例子，去掉最后一个 case 块，这个例子就变成:

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

如果不在 case 语句中使用 _，可能会出现不存在匹配的情况。如果不存在匹配，则行为是一个 no-op。例如，如果传入了值为 500 的 status ，就会发生 no-op。

带有字面值和变量的模式

模式可以看起来像解包形式，而且模式可以用来绑定变量。在这个例子中，一个数据点可以被解包为它的 x 坐标和 y 坐标:

```
# 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")
```

第一个模式有两个字面值 (0, 0) ，可以看作是上面所示字面值模式的扩展。接下来的两个模式结合了一个字面值和一个变量，而变量 绑定了一个来自主词的值 (point)。第四种模式捕获了两个值，这使得它在概念上类似于解包赋值 (x, y) = point 。

模式和类

如果你使用类来结构化你的数据，你可以使用类的名字，后面跟一个类似构造函数的参数列表，作为一种模式。这种模式可以将类的属性捕捉到变量中:

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

def location(point):
    match point:
```

(下页继续)

(续上页)

```
        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")
```

带有位置参数的模式

你可以在某些为其属性提供了排序的内置类（例如 dataclass）中使用位置参数。你也可以通过在你的类中设置 `__match_args__` 特殊属性来为模式中的属性定义一个专门的位置。如果它被设为 ("x", "y")，则以下模式均为等价的（并且都是将 `y` 属性绑定到 `var` 变量）：

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

嵌套模式

模式可以任意地嵌套。例如，如果我们的数据是由点组成的短列表，则它可以这样被匹配:

```
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.")
```

复杂模式和通配符

到目前为止，这些例子仅在最后一个 case 语句中使用了 `_`。但通配符可以被用在更复杂的模式中，例如 `('error', code, _)`。举例来说:

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

在上述情况下，`test_variable` 将可匹配 ('error', code, 100) 和 ('error', code, 800)。

约束项

我们可以向一个模式添加 `if` 子句，称为“约束项”。如果约束项为假值，则 `match` 将继续尝试下一个 case 语句块。请注意值的捕获发生在约束项被求值之前。:

```
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.")
```

其他关键特性

一些其他关键特性:

- 类似于解包赋值，元组和列表模式具有完全相同的含义，而且实际上能匹配任意序列。从技术上说，目标必须为一个序列。因而，一个重要的例外是模式不能匹配迭代器。而且，为了避免一个常见的错误，序列模式不能匹配字符串。
- 序列模式支持通配符: `[x, y, *rest]` 和 `(x, y, *rest)` 的作用类似于解包赋值中的通配符。在 `*` 之后的名称也可以为 `_`，因此 `(x, y, *_)` 可以匹配包含两个条目的序列而不必绑定其余的条目。
- 映射模式: `{"bandwidth": b, "latency": l}` 会从一个字典中捕获 `"bandwidth"` 和 `"latency"` 值。与序列模式不同，额外的键会被忽略。也支持通配符 `**rest`。（但 `**_` 是冗余的，因而不被允许。）
- 子模式可使用 `as` 关键字来捕获:

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

  x1, y1, x2, y2 等绑定就如你在没有 `as` 子句的情况下所期望的，而 p2 会绑定目标的整个第二项。
- 大多数字面值是按相等性比较的。但是，单例对象 `True`, `False` 和 `None` 则是按标识号比较的。
- 命名常量也可以在模式中使用。这些命名常量必须为带点号的名称以防止常量被解读为捕获变量:

  ```
  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 :(")
  ```

完整规格说明参见 **PEP 634**。动机与理由参见 **PEP 635**，更详细的教程参见 **PEP 636**。

2.5 可选的 EncodingWarning 和 encoding="locale" 选项

TextIOWrapper 和 open() 的默认编码格式取决于具体的平台和语言区域设置。由于 UTF-8 被用于大多数 Unix 平台，当打开 UTF-8 文件 (例如 JSON, YAML, TOML, Markdown) 时省略 encoding 选项是一个非常常见的错误。例如:

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

为了便于查找此类错误，增加了可选的 EncodingWarning。它会在 sys.flags.warn_default_encoding 为真值并使用了语言区域指定的默认编码格式时被发出。

增加了 -X warn_default_encoding 选项和 PYTHONWARNDEFAULTENCODING 来启用相应警告。

请参阅 io-text-encoding 了解更多信息。

3 有关类型提示的新增特性

本节介绍了涉及 **PEP 484** 类型提示和 typing 模块的主要改变。

3.1 PEP 604: 新的类型联合运算符

引入了启用 X | Y 语法的类型联合运算符。这提供了一种表示' 类型 X 或类型 Y' 的相比使用 typing.Union 更清晰的方式，特别是在类型提示中。

在之前的 Python 版本中，要为可接受多种类型参数的函数应用类型提示，使用的是 typing.Union:

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

类型提示现在可以使用更简洁的写法:

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

这个新增语法也被接受作为 isinstance() 和 issubclass() 的第二个参数:

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

详情参见 types-union 和 **PEP 604**。

(由 Maggie Moss 和 Philippe Prados 在 bpo-41428 中贡献，并由 Yurii Karabas 和 Serhiy Storchaka 在 bpo-44490 中补充。)

3.2 PEP 612: 形参规格变量

在 typing 模块中新增了两个选项以改进用于 **PEP 484** 的 Callable 提供给静态类型检查器的信息。

第一个选项是形参规格变量。它们被用来将一个可调用对象的形参类型转发给另一个可调用对象——这种模式常见于高阶函数和装饰器。使用示例可在 typing.ParamSpec 中找到。在之前版本中，没有一种简单办法能以如此精确的方式对形参类型的依赖性进行类型标注。

第二个选项是新的 Concatenate 运算符。它与形参规格变量一起使用以便对增加或移除了其他可调用对象的高阶可调用对象进行类型标注。使用示例可以在 typing.Concatenate 中找到。

请参阅 typing.Callable, typing.ParamSpec, typing.Concatenate, typing.ParamSpecArgs, typing.ParamSpecKwargs 和 **PEP 612** 来了解更多细节。

(由 Ken Jin 在 bpo-41559 中贡献，并由 Jelle Zijlstra 在 bpo-43783 中加以少量改进。PEP 由 Mark Mendoza 撰写。)

3.3 PEP 613: 类型别名

PEP 484 引入了类型别名的概念，只要求它们是不带标注的最高层级赋值。这种简单性有时会使得类型检查器难以区分类型别名和普通赋值，特别是当涉及到前向引用或无效类型的时候。例如在比较:

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

现在 typing 模块具有一个特殊值 TypeAlias 可让你更明确地声明类型别名:

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

请参阅 **PEP 613** 了解详情。

(由 Mikhail Golubev 在 bpo-41923 中贡献。)

3.4 PEP 647: 用户自定义的类型保护器

TypeGuard 已被添加到 typing 模块用来标注类型保护器函数并改进在类型细化期间提供给静态类型分析器的信息。要了解更多信息，请参阅 TypeGuard 的文档以及 **PEP 647**。

(由 Ken Jin 和 Guido van Rossum 在 bpo-43766 中贡献。PEP 由 Eric Traut 撰写。)

4 其他语言特性修改

- int 类型新增了一个方法 int.bit_count()，返回给定整数的二进制展开中值为一的位数，或称“比特计量”。(由 Niklas Fiekas 在 bpo-29882 中贡献。)
- 现在 dict.keys(), dict.values() 和 dict.items() 所返回的视图都有一个 mapping 属性，它给出包装了原始字典的 types.MappingProxyType 对象。(由 Dennis Sweeney 在 bpo-40890 中贡献。)
- **PEP 618**: 现在 zip() 函数有一个可选的 strict 旗标，用于要求所有可迭代对象的长度都相等。
- 接受整数参数的内置和扩展函数不再接受 Decimal, Fraction 以及其他可被转换为整数但会丢失精度（即具有 __int__() 方法但没有 __index__() 方法）的对象。(由 Serhiy Storchaka 在 bpo-37999 中贡献。)
- 如果 object.__ipow__() 返回 NotImplemented，该运算符将正确地按照预期回退至 object.__pow__() 和 object.__rpow__()。(由 Alex Shkop 在 bpo-38302 中贡献。)
- 现在赋值表达式可以不带圆括号地在集合字面值和集合推导式中使用，也可以在序列索引号中使用（但不能用于切片）。
- 函数具有一个新的 __builtins__ 属性，当函数被执行时它会被用于查找内置符号，而不是在 __globals__['__builtins__'] 中查找。如果 __globals__["__builtins__"] 存在则该属性将基于它来初始化，否则将基于当前的内置函数。(由 Mark Shannon 在 bpo-42990 中贡献。)
- 增加了两个新的内置函数——aiter() 和 anext() 以分别提供与 iter() 和 next() 对应的异步版本。(由 Joshua Bronson, Daniel Pope 和 Justin Wang 在 bpo-31861 中贡献。)
- 静态方法 (@staticmethod) 和类方法 (@classmethod) 现在会继承方法属性 (__module__, __name__, __qualname__, __doc__, __annotations__) 并具有一个新的 __wrapped__ 属性。此外，静态方法现在还是与常规函数一样的可调用对象。(由 Victor Stinner 在 bpo-43682 中贡献。)

- 复杂目标的注解（**PEP 526** 定义的除 `simple name` 之外的一切复杂目标）在运行时不再受 `from __future__ import annotations` 的影响。（由 Batuhan Taskaya 在 bpo-42737 中贡献）。
- 类和模块对象现在可以按需创建空的注解字典。为保证向下兼容，这些注解数据将存储于对象的 `__dict__` 中。这改进了 `__annotations__` 的最佳用法；更多信息请参阅 annotations-howto 。（由 Larry Hastings 在 bpo-43901 中贡献）
- 由于会产生副作用，现在 `from __future__ import annotations` 时禁止使用包含 `yield`、`yield from`、`await` 或已命名表达式的注解。（由 Batuhan Taskaya 在 bpo-42725 中贡献）
- 未绑定变量、`super()` 和其他可能改变符号表处理的表达式，现在在 `from __future__ import annotations` 时不能用作注解。（由 Batuhan Taskaya 在 bpo-42725 中贡献）
- `float` 类型和 `decimal.Decimal` 类型的 NaN 值的哈希值现在取决于对象身份。以前，即便 NaN 值彼此不等，也都是哈希为 `0`。在创建包含多个 NaN 的字典和集合时，由于哈希冲突过度，导致了运行代价可能会二次方增长。（由 Raymond Hettinger 在 bpo-43475 中贡献）
- A `SyntaxError` (instead of a `NameError`) will be raised when deleting the `__debug__` constant. (Contributed by Dong-hee Na in bpo-45000.)
- `SyntaxError` exceptions now have `end_lineno` and `end_offset` attributes. They will be `None` if not determined. (Contributed by Pablo Galindo in bpo-43914.)

5 新增模块

- 无。

6 改进的模块

6.1 asyncio

加入了缺失的 `connect_accepted_socket()` 方法。（由 Alex Grönholm 在 bpo-41332 中贡献）

6.2 argparse

在 argparse 的帮助中，将“可选参数”这一误导性短语改为“可选项”。某些测试代码如果依赖精确的输出匹配，可能需要调整。（由 Raymond Hettinger 在 bpo-9694 中贡献）

6.3 array

现在，`array.array` 的 `index()` 方法拥有可选的 *start* 和 *stop* 参数。（由 Anders Lorentsen 和 Zackery Spytz 在 bpo-31956 中贡献）

6.4 asynchat、asyncore 和 smtpd

从 Python 3.6 开始，这些模块在其文档中已被标为废弃。现在这三个模块都增加了一个导入时警告 `DeprecationWarning`。

6.5 base64

增加 `base64.b32hexencode()` 和 `base64.b32hexdecode()` 以支持带有扩展十六进制字母的 Base32 编码。

6.6 bdb

增加 `clearBreakpoints()`，用于重置所有已设断点。(由 Irit Katriel 在 bpo-24160 中贡献)

6.7 bisect

Added the possibility of providing a *key* function to the APIs in the `bisect` module. (Contributed by Raymond Hettinger in bpo-4356.)

6.8 编码器

增加一个 `codecs.unregister()` 函数，用于取消对编解码器搜索函数的注册。(由 Hai Shi 在 bpo-41842 中贡献)

6.9 collections.abc

现在，parameterized generic 的 `collections.abc.Callable` 的 `__args__` 与 `typing.Callable` 一致了。`collections.abc.Callable` generic 现在将类型参数扁平化了，类似于 `typing.Callable` 当前的做法。这意味着 `collections.abc.Callable[[int, str], str]` 将带有 `(int, str, str)` 的参数 `__args__`；以前是 `([int, str], str)`。为了做到这一变化，`types.GenericAlias` 现在可以被子类化，当对 `collections.abc.Callable` 类型进行下标访问时，将返回一个子类。注意，`collections.abc.Callable` 非法的参数化形式可能会触发 `TypeError`，而在 Python 3.9 中可能就静默了。(由 Ken Jin 在 bpo-42195 中贡献)

6.10 contextlib

增加了一个上下文管理器 `contextlib.aclosing()`，以便能安全关闭异步生成器和代表异步释放资源的对象。(由 Joongi Kim 和 John Belmonte 在 bpo-41229 中贡献)

为 `contextlib.nullcontext()` 加入异步上下文管理器支持。由 Tom Gringauz 在 bpo-41543 中贡献)

加入 `AsyncContextDecorator`，以便支持用异步上下文管理器作为装饰器。

6.11 curses

在 ncurses 6.1 中增加的扩展颜色函数将会由 `curses.color_content()`、`curses.init_color()`、`curses.init_pair()` 和 `curses.pair_content()` 透明地使用。新增的函数 `curses.has_extended_color_support()` 将指明下层的 ncurses 库是否提供了扩展颜色支持。(由 Jeffrey Kintscher 和 Hans Petter Jansson 在 bpo-36982 中贡献)

现在常量 `BUTTON5_*` 如果是由底层的 curses 库提供的，则会在 `curses` 模块中体现。(由 Zackery Spytz 在 bpo-39273 中贡献)

6.12 dataclasses

__slots__

Added `slots` parameter in `dataclasses.dataclass()` decorator. (Contributed by Yurii Karabas in bpo-42269)

Keyword-only fields

dataclasses now supports fields that are keyword-only in the generated __init__ method. There are a number of ways of specifying keyword-only fields.

You can say that every field is keyword-only:

```
from dataclasses import dataclass

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

Both `name` and `birthday` are keyword-only parameters to the generated __init__ method.

You can specify keyword-only on a per-field basis:

```
from dataclasses import dataclass

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

Here only `birthday` is keyword-only. If you set `kw_only` on individual fields, be aware that there are rules about re-ordering fields due to keyword-only fields needing to follow non-keyword-only fields. See the full dataclasses documentation for details.

You can also specify that all fields following a KW_ONLY marker are keyword-only. This will probably be the most common usage:

```
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

整个 `distutils` 包已被废弃，将在 Python 3.12 中移除。其用指定包构建程序的功能已被第三方软件包 `setuptools` 和 `packaging` 完全取代，而且大多数其他常用的 API 在标准库的其他地方都可以调用（如 `platform` 、`shutil` 、`subprocess` 或 `sysconfig`）。目前没有迁移 `distutils` 其他功能的计划，用到其他功能的应用程序应该准备好自己保留一份拷贝。请参考 **PEP 632** 。

在 Python 3.8 中废弃的 `bdist_wininst` 命令已被移除。现在在 Windows 中发布二进制包推荐采用 `bdist_wheel` 命令。（由 Victor Stinner 在 bpo-42802 中贡献）

6.14 doctest

若模块中没有定义 `__loader__` ，则回退至使用 `__spec__.loader` 。（由 Brett Cannon 在 bpo-42133 中贡献）

6.15 encodings

现在 `encodings.normalize_encoding()` 会忽略非 ASCII 字符。（由 Hai Shi 在 bpo-39337 中贡献）

6.16 fileinput

在 `fileinput.input()` 和 `fileinput.FileInput` 中增加了 *encoding* 和 *errors* 形参。（由 Inada Naoki 在 bpo-43712 中贡献。）

现在 `fileinput.hook_compressed()` 会在 *mode* 为”r” 且文件被压缩时返回 `TextIOWrapper`，与未压缩文件一致。（由 Inada Naoki 在 bpo-5758 中贡献。）

6.17 faulthandler

现在 `faulthandler` 模块会检测在垃圾回收期间是否发生严重错误。（由 Victor Stinner 在 bpo-44466 中贡献）

6.18 gc

为 `gc.get_objects()`, `gc.get_referrers()` 和 `gc.get_referents()` 添加了审计钩子。（由 Pablo Galindo 在 bpo-43439 中贡献。）

6.19 glob

在 `glob()` 和 `iglob()` 中增加了 *root_dir* 和 *dir_fd* 形参，用于指定搜索的根目录。（由 Serhiy Storchaka 在 bpo-38144 中贡献）

6.20 hashlib

hashlib 模块要求 OpenSSL 1.1.1 或更新版本。（由 Christian Heimes 在 **PEP 644** 和 bpo-43669 中贡献。）

hashlib 模块已初步支持 OpenSSL 3.0.0。（由 Christian Heimes 在 bpo-38820 及其他问题事项中贡献。）

纯 Python 的回退版 `pbkdf2_hmac()` 已被弃用。未来 PBKDF2-HMAC 将仅在 Python 带有 OpenSSL 编译时才可用。（由 Christian Heimes 在 bpo-43880 中贡献）

6.21 hmac

现在 hmac 模块会在内部使用 OpenSSL 的 HMAC 实现。（由 Christian Heimes 在 bpo-40645 中贡献。）

6.22 IDLE 与 idlelib

Make IDLE invoke `sys.excepthook()` (when started without '-n'). User hooks were previously ignored. (Contributed by Ken Hilton in 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.)

The changes above were backported to a 3.9 maintenance release.

增加了 Shell 侧栏。将主提示符（“>>>”）移至侧栏。二级提示符（“...”）也加入侧栏。在编辑器的行号侧栏上鼠标单击和可选的拖动，会选定一行或多行文本。在选定文本行后右击将显示包含 “copy with prompts” 的上下文菜单。这会将侧栏的提示符与选定文本行合并。该选项也会在文本的上下文菜单中显示。（由 Tal Einat 在 bpo-37903 中贡献）

Use spaces instead of tabs to indent interactive code. This makes interactive code entries 'look right'. Making this feasible was a major motivation for adding the shell sidebar. (Contributed by Terry Jan Reedy in bpo-37892.)

Highlight the new soft keywords `match`, `case`, and _ in pattern-matching statements. However, this highlighting is not perfect and will be incorrect in some rare cases, including some _-s in `case` patterns. (Contributed by Tal Einat in bpo-44010.)

New in 3.10 maintenance releases.

Apply syntax highlighting to *.pyi* files. (Contributed by Alex Waygood and Terry Jan Reedy in bpo-45447.)

6.23 importlib.metadata

与 `importlib_metadata` 4.6（history）的功能一致。

importlib.metadata entry points now provide a nicer experience for selecting entry points by group and name through a new `importlib.metadata.EntryPoints` class. See the Compatibility Note in the docs for more info on the deprecation and usage.

添加了 `importlib.metadata.packages_distributions()`，用于将顶级 Python 模块和包解析为其 `importlib.metadata.Distributions`。

6.24 inspect

若模块中没有定义 `__loader__` ，则回退至使用 `__spec__.loader` 。（由 Brett Cannon 在 bpo-42133 中贡献）

加入了 `inspect.get_annotations()`，以便安全地对对象中定义的注解进行求值。`inspect.get_annotations()` 也可以正确地解析字符串化的注解。`inspect.get_annotations()` 现在应是访问任何 Python 对象注解字典的最佳实践；关于注解最佳用法的更多信息，请参见 annotations-howto。与之关联的，`inspect.signature()` 、`inspect.Signature.from_callable()` 和 `inspect.Signature.from_function()` 现在也调用 `inspect.get_annotations()` 来获取注解信息。这意味着 `inspect.signature()` 和 `inspect.Signature.from_callable()` 现在也可以解析字符串化的注解了。（由 Larry Hastings 在 bpo-43817 中贡献）

6.25 itertools

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

6.26 linecache

若模块中没有定义 `__loader__` ，则回退至使用 `__spec__.loader` 。（由 Brett Cannon 在 bpo-42133 中贡献）

6.27 os

为 VxWorks 实时操作系统加入 `os.cpu_count()` 的支持。（由 Peixing Xin 在:issue:*41440* 中贡献）

加入一个新函数 `os.eventfd()` 及其助手函数，以封装 Linux 的系统调用 `eventfd2` 。(由 Christian Heimes 在 bpo-41001 中贡献）

加入 `os.splice()`，以便在两个文件描述符之间移动数据，而无需在内核地址空间和用户地址空间之间进行复制，其中一个文件描述符必须指向某个管道。（由 Pablo Galindo 在 bpo-41625 中贡献）

为 macOS 加入 `O_EVTONLY` 、`O_FSYNC` 、`O_SYMLINK` 和 `O_NOFOLLOW_ANY` 。（由 Dong-hee Na 在 bpo-43106 中贡献）

6.28 os.path

现在 `os.path.realpath()` 可接受一个关键字参数 *strict*。若设为 `True` ，则在路径不存在或遭遇循环符号链接时，会触发 `OSError`。（由 Barney Gale 在 bpo-43757 中贡献。）

6.29 pathlib

为 `PurePath.parents` 加入切片支持。（由 Joshua Cannon 贡献于 bpo-35498）

为 `PurePath.parents` 加入负数索引支持。（由 Yaroslav Pankovych 贡献于 bpo-21041 ）

加入 `Path.hardlink_to` 方法，取代 `link_to()`。该新方法的参数顺序与 `symlink_to()` 相同。（由 Barney Gale 贡献于 bpo-39950 中）

现在 `pathlib.Path.stat()` 和 `chmod()` 接受一个关键字参数 *follow_symlinks* ，以便与 `os` 模块中的对应函数保持一致。（由 Barney Gale 贡献于 bpo-39906 ）

6.30 平台

加入 `platform.freedesktop_os_release()`，从标准文件 freedesktop.org os-release 中获取操作系统标识。（由 Christian Heimes 贡献于 bpo-28468 ）

6.31 pprint

现在 `pprint.pprint()` 接受一个新的关键字参数 `underscore_numbers`。（由 sblondon 贡献于 bpo-42914 ）

现在 `pprint` 可以完美打印 `dataclasses.dataclass` 实例。（由 Lewis Gaul 贡献于 bpo-43080 ）

6.32 py_compile

`py_compile' 的命令行界面加入 `--quiet“ 选项。（由 Gregory Schevchenko 贡献于 bpo-38731 ）

6.33 pyclbr

在 `pyclbr.readline()` 和 `pyclbr.readline_ex()` 返回的结果树中的 “Function“ 和 `Class` 对象上增加一个 `end_lineno` 属性。与现有的（开始）“lineno“ 相匹配。（由 Aviral Srivastava 贡献于 bpo-38307）

6.34 shelve

现在 `shelve` 模块在创建打包时，默认采用 `pickle.DEFAULT_PROTOCOL`，而不是 `pickle` 协议 `3` 。（由 Zackery Spytz 贡献于 bpo-34204 ）

6.35 statistics

加入 `covariance()` 、Pearson 的 `correlation()` 和简单的 `linear_regression()` 函数。（由 Tymoteusz Wołodźko 贡献于 bpo-38490 ）

6.36 site

若模块中没有定义 `__loader__` ，则回退至使用 `__spec__.loader` 。（由 Brett Cannon 在 bpo-42133 中贡献）

6.37 socket

现在异常 `socket.timeout` 是 `TimeoutError` 的别名。（由 Christian Heimes 在 bpo-42413 中贡献。）

加入用 `IPPROTO_MPTCP` 创建 MPTCP 套接字的选项（由 Rui Cunha 贡献于 bpo-43571 ）

加入 `IP_RECVTOS` 选项，以便接收服务类型（ToS）或 DSCP/ECN 字段（由 Georg Sauthoff 贡献于 bpo-44077 ）

6.38 ssl

ssl 模块要求 OpenSSL 1.1.1 或更新版本。(由 Christian Heimes 在 **PEP 644** 和 bpo-43669 中贡献。)

ssl 模块已初步支持 OpenSSL 3.0.0 和新选项 `OP_IGNORE_UNEXPECTED_EOF`。(由 Christian Heimes 在 bpo-38820, bpo-43794, bpo-43788, bpo-43791, bpo-43799, bpo-43920, bpo-43789 和 bpo-43811 中贡献。)

现在，已弃用函数和使用已弃用常量会导致 `DeprecationWarning`。`ssl.SSLContext.options` 默认设置了 `OP_NO_SSLv2` 和 `OP_NO_SSLv3`，因而设置此标记无法再次发出警告了。*弃用部分* 列出了已弃用的特性。(由 Christian Heimes 贡献于 bpo-43880)

现在，ssl 模块默认设置的安全性提高了。默认情况下，不具备前向安全性或 SHA-1 MAC 的加密算法会被禁用。二级安全禁止安全性低于 112 位的弱 RSA、DH 和 ECC 密钥。`SSLContext` 默认的最低版本协议为 TLS 1.2。这些设置是基于 Hynek Schlawack 的研究。(由 Christian Heimes 贡献于 bpo-43998)

已弃用的协议 SSL 3.0, TLS 1.0 和 TLS 1.1 不再受到官方支持。Python 不会直接禁用。但 OpenSSL 编译选项、发行版配置、厂商补丁和加密套件可能会阻止握手成功。

为 `ssl.get_server_certificate()` 函数加入 *timeout* 形参。(由 Zackery Spytz 贡献于 bpo-31870)

ssl 模块用到了堆类型和多阶段初始化。(由 Christian Heimes 贡献于 bpo-42333)

加入一个新的校验标记 `VERIFY_X509_PARTIAL_CHAIN`。(由 l0x 贡献于 bpo-40849)

6.39 sqlite3

为 `connect/handle()` 、`enable_load_extension()` 和 `load_extension()` 加入审计事件。(由 Erlend E. Aasland 贡献于 bpo-43762)

6.40 sys

加入了 `sys.orig_argv` 属性：传给 Python 可执行文件的初始命令行参数列表。(由 Victor Stinner 贡献于 bpo-23427)

添加了 `sys.stdlib_module_names`，包含标准库模块名称的列表。(由 Victor Stinner 在 bpo-42955 中贡献。)

6.41 _thread

现在，`_thread.interrupt_main()` 接受一个可选的信号值参数进行模拟（默认仍为 `signal.SIGINT`)。(由 Antoine Pitrou 贡献于 bpo-43356)

6.42 threading

加入 `threading.gettrace()` 和 `threading.getprofile()` ，分别用于获取 `threading.settrace()` 和 `threading.setprofile()` 设置的函数。(由 Mario Corchero 贡献于 bpo-42251)

加入 `threading.__excepthook__` ，用于获取 `threading.excepthook()` 的初始值，以防被设为一个差劲或其他的值。(由 Mario Corchero 贡献于 bpo-42308)

6.43 traceback

现在，format_exception() 、format_exception_only() 和 print_exception() 函数可以接受一个异常对象，作为唯一的位置参数。（由 Zackery Spytz 和 Matthias Bussonnier 贡献于 bpo-26389）

6.44 types

重新引入 types.EllipsisType 、types.NoneType 和 types.NotImplementedType 类，以提供一套新的类型，可供类型检查程序解释。（由 Bas van Beek 贡献于 bpo-41810）

6.45 typing

For major changes, see 有关类型提示的新增特性.

typing.Literal 的行为被改为遵循 **PEP 586** 并匹配该 PEP 所描述的静态类型检查器的行为。

1. Literal 现在将是去重后的形参。
2. Literal 对象的相等性比较现在将与顺序无关。
3. Literal comparisons now respect types. For example, Literal[0] == Literal[False] previously evaluated to True. It is now False. To support this change, the internally used type cache now supports differentiating types.
4. 现在，如果 Literal 对象的任何参数都不是 hashable ，在相等性比较时将引发 TypeError 异常。请注意，在声明 Literal 时，参数不可哈希不会抛出错误：

```
>>> 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'
```

（由 Yurii Karabas 在 bpo-42345 中贡献。）

加入新函数 typing.is_typeddict()，以内部检查注解是否为 typing.TypedDict。（由 Patrick Reader 贡献于 bpo-41792 ）

现在，如果 typing.Protocol 的子类只声明了数据变量，那么在用 isinstance 检查时会引发 TypeError，除非是用 runtime_checkable() 装饰的。以前，这是静默通过检查的。如果用户需要运行时协议，应该用 runtime_checkable() 装饰器来装饰其子类。（由 Yurii Karabas 贡献于 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)

6.46 unittest

加入新方法 assertNoLogs() ，以补充现有的 assertLogs()。（由 Kit Yan Choi 贡献于 bpo-39385 ）

6.47 urllib.parse

Python 3.10 以下版本允许在 `urllib.parse.parse_qs()` 和 `urllib.parse.parse_qsl()` 中同时使用 `;` 和 `&` 作为查询参数分隔符。出于安全考虑，并符合 W3C 最新的建议，这已经被修改为只允许用一种分隔符，默认值为 `&`。这一改变也影响到了 `cgi.parse()` 和 `cgi.parse_multipart()`，因为他们内部用到了这些函数。更多细节，请参阅各自的文档。（由 Adam Goldschmidt 、Senthil Kumaran 和 Ken Jin 贡献于 bpo-42967 ）

在 URL 中存在换行符或制表符，可能会导致某些形式的攻击。根据 WHATWG 的规范更新了:rfc:*3986*，`urllib.parse` 中的解析器将从 URL 中移除 ASCII 换行符 `\n` 、`\r` 和 `\t`` 字符，以防止这种攻击。将移除的字符由一个新的模块级变量 ``urllib.parse._UNSAFE_URL_BYTES_TO_REMOVE` 控制。（参阅 bpo-43882 ）

6.48 xml

在 `xml.sax.handler` 模块中加入一个 `LexicalHandler` 类。（由 Jonathan Gossage 和 Zackery Spytz 贡献于 bpo-35018 ）

6.49 zipimport

加入 **PEP 451** 相关的方法：`find_spec()` 、`zipimport.zipimporter.create_module()` 和 `zipimport.zipimporter.exec_module()`。（由 Brett Cannon 贡献于 bpo-42131 ）

加入 `invalidate_caches()` 方法。（由 Desmond Cheong 贡献于 bpo-14678 ）

7 性能优化

- 现在，构造函数 `str()` 、`bytes()` 和 `bytearray()` 速度更快了（小对象大约提速 30-40%）。（由 Serhiy Storchaka 贡献于 bpo-41334 ）
- 现在，`runpy` 导入的模块变少了。`python3 -m module-name` 命令的启动时间平均加快 1.4 倍。在 Linux 上，Python 3.9 的 `python3 -I -m module-name` 导入了 69 个模块，而 Python 3.10 只导入了 51 个模块（少了 18 个）。（由 Victor Stinner 贡献于 bpo-41006 和 bpo-41718）
- 现在，`LOAD_ATTR` 指令会使用新的“单独操作码缓存”机制。对于常规属性大约会提速 36%，而对于槽位属性会加快 44%。（由 Pablo Galindo 和 Yury Selivanov 贡献于 bpo-42093 ），并由 Guido van Rossum 贡献于 bpo-42927，基于最初在 PyPy 和 MicroPython 中实现的思路。）
- 现在，当用 `--enable-optimizations` 构建 Python 时，会在编译和链接命令行中添加 `-fno-semantic-interposition`。这会让用带参数 `--enable-shared` 的 `gcc` 构建 Python 解释器时提速 30%。详情请参阅 ‘这篇文章 <https://developers.redhat.com/blog/2020/06/25/red-hat-enterprise-linux-8-2-brings-faster-python-3-8-run-speeds/>‘_ 。（由 Victor Stinner 和 Pablo Galindo 贡献于 bpo-38980 ）
- `bz2` / `lzma` / `zlib` 模块用了新的输出缓冲区管理代码，并在 `_compression.DecompressReader` 类中添加 “.readall()“ 函数。现在，bz2 解压过程提速了 1.09 倍 ~ 1.17 倍，lzma 解压快了 1.20 倍 ~ 1.32 倍，`GzipFile.read(-1)` 快了 1.11 倍 ~ 1.18 倍。（由 Ma Lin 贡献，由 Gregory P. Smith 审查，bpo-41486）
- 在使用字符串式的注解时，函数的注解字典不再是在函数创建时建立了。取而代之的是，注解被存储为一个字符串元组，而函数对象在需要时再延迟转换为注解字典。这一优化将定义带注解函数的 CPU 时间减少了一半。（由 Yurii Karabas 和 Inada Naoki 贡献于 bpo-42202）
- 现在，子串搜索函数，如 `str1 in str2` 和 `str2.find(str1)` ，有时会采用 Crochemore & Perrin 的“二路归并”字符串搜索算法，以避免长字符串的二次检索行为。（由 Dennis Sweeney 贡献于 bpo-41972 ）
- 为 `_PyType_Lookup()` 加入了少许优化，以提高类型属性缓存查询在常见命中情况下的性能。这使得解释器的平均速度提高了 1.04 倍。（由 Dino Viehland 贡献于 bpo-43452 ）

- 现在，以下内置函数支持更快的 **PEP 590** vectorcall 调用约定：`map()`、`filter()`、`reversed()`、`bool()` 和 `float()`。(由 Dong-hee Na 和 Jeroen Demeyer 贡献于 bpo-43575、bpo-43287、bpo-41922、bpo-41873 和 bpo-41870)
- 通过移除内部的 `RLock`，`BZ2File` 的性能得以改善。这使得 `BZ2File` 在面对多个同时读写线程时变得不再安全，类似一直如此的 `gzip` 和 `lzma` 中的对应类。(由 Inada Naoki 贡献于 bpo-43785)

8 弃用

- 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).
- 本版本将开始发起一次协同任务，清理为兼容 Python 2.7 而保留的旧导入语义。具体包括：`find_loader()`/`find_module()`（被 `find_spec()` 取代），`load_module()`（被 `exec_module()` 取代），`module_repr()`（将由导入系统处理），`__package__` 属性（被 `__spec__.parent` 取代）、`__loader__` 属性（被 `__spec__.loader` 取代），以及 `__cached__` 属性（被 `__spec__.cached` 取代），都将逐渐被移除（还包括 `importlib` 中的其他类和方法）。`ImportWarning` 和/或 `DeprecationWarning` 将在适当时刻触发，以助在过渡期内识别出那些需要更新的代码。
- 整个 `distutils` 命名空间已被弃用，并将在 Python 3.12 中被移除。请参阅模块的变化 一节了解更多信息。
- `random.randrange()` 的非整数参数已被弃用。`ValueError` 已被弃用而应改用 `TypeError`。(由 Serhiy Storchaka 和 Raymond Hettinger 贡献于 bpo-37319)
- `importlib` 的各种 `load_module()` 方法自 Python 3.6 起就已被记录为弃用，现在还会触发 `DeprecationWarning`。请改用 `exec_module()`。(由 Brett Cannon 贡献于 bpo-26131)
- `zimport.zipimporter.load_module()` 已弃用并被 `exec_module()` 代替。(由 Brett Cannon 在 bpo-26131 中贡献。)
- 现在导入时使用 `load_module()` 会引发 `ImportWarning`，应改用 `exec_module()`。(由 Brett Cannon 贡献于 bpo-26131)
- 现在导入系统使用 `importlib.abc.MetaPathFinder.find_module()` 和 `importlib.abc.PathEntryFinder.find_module()` 会引发 `ImportWarning` 而应分别改用 `importlib.abc.MetaPathFinder.find_spec()` 和 `importlib.abc.PathEntryFinder.find_spec()`。你可以使用 `importlib.util.spec_from_loader()` 来协助移植。(由 Brett Cannon 在 bpo-42134 中贡献。)
- 现在导入系统使用 `importlib.abc.PathEntryFinder.find_loader()` 会引发 `ImportWarning` 而应改用 `importlib.abc.PathEntryFinder.find_spec()`。你可以使用 `importlib.util.spec_from_loader()` 来协助移植。(由 Brett Cannon 在 bpo-43672 中贡献。)
- The various implementations of `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()`, `importlib.abc.MetaPathFinder.find_module()`), `importlib.abc.PathEntryFinder.find_module()` (`importlib.machinery.FileFinder.find_module()`), and `importlib.abc.PathEntryFinder.find_loader()` (`importlib.machinery.FileFinder.find_loader()`) now raise `DeprecationWarning` and are slated for removal in Python 3.12 (previously they were documented as deprecated in Python 3.4). (Contributed by Brett Cannon in bpo-42135.)

- importlib.abc.Finder 已被弃用 (包括它唯一的方法 find_module())。importlib.abc.MetaPathFinder 和 importlib.abc.PathEntryFinder 都不再继承该类。用户应当改为继承这两个类中的一个。(由 Brett Cannon 在 bpo-42135 中贡献。)
- imp, importlib.find_loader(), importlib.util.set_package_wrapper(), importlib.util.set_loader_wrapper(), importlib.util.module_for_loader(), pkgutil.ImpImporter 和 pkgutil.ImpLoader 的弃用状态已被更新以将 Python 3.12 列为预定要移除它们的版本 (它们在之前的 Python 版本中已开始引发 DeprecationWarning)。(由 Brett Cannon 在 bpo-43720 中贡献。).)
- 现在导入系统会先使用模块上的 __spec__ 属性再回退到 module_repr() 来使用模块的 __repr__() 方法。对 module_repr() 的使用预定在 Python 3.12 中移除。(由 Brett Cannon 在 bpo-42137 中贡献。)
- importlib.abc.Loader.module_repr(), importlib.machinery.FrozenLoader.module_repr() 和 importlib.machinery.BuiltinLoader.module_repr() 已被弃用并预定在 Python 3.12 中移除。(由 Brett Cannon 在 bpo-42136 中贡献。)
- sqlite3.OptimizedUnicode 自 Python 3.3 起就被移出文档并设为过时，当时它是被设为 str 的别名。现在它已被弃用，预定在 Python 3.12 中移除。(由 Erlend E. Aasland 在 bpo-42264 中贡献。)
- asyncio.get_event_loop() now emits a deprecation warning if there is no running event loop. In the future it will be an alias of get_running_loop(). asyncio functions which implicitly create Future or Task objects now emit a deprecation warning if there is no running event loop and no explicit *loop* argument is passed: ensure_future(), wrap_future(), gather(), shield(), as_completed() and constructors of Future, Task, StreamReader, StreamReaderProtocol. (Contributed by Serhiy Storchaka in bpo-39529.)
- 未记入文档的内置函数 sqlite3.enable_shared_cache 现在已被弃用，预定在 Python 3.12 中移除。SQLite3 强烈不建议使用它。请参阅 SQLite3 文档 了解详情。如果必须要使用共享缓冲区，请使用 cache=shared 查询参数来以 URI 模式打开数据库。(由 Erlend E. Aasland 在 bpo-24464 中贡献。)
- 以下 threading 方法已被弃用:
  - 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

  (由 Jelle Zijlstra 在 bpo-21574 中贡献。)
- pathlib.Path.link_to() 已被弃用并预定在 Python 3.12 中移除。请改用 pathlib.Path.hardlink_to()。(由 Barney Gale 在 bpo-39950 中贡献。)
- cgi.log() 已被弃用并预定在 Python 3.12 中移除。(由 Inada Naoki 在 bpo-41139 中贡献。)
- 以下 ssl 特性自 Python 3.6, Python 3.7 或 OpenSSL 1.1.0 起已被弃用并将在 3.11 中移除:
  - OP_NO_SSLv2, OP_NO_SSLv3, OP_NO_TLSv1, OP_NO_TLSv1_1, OP_NO_TLSv1_2 及 OP_NO_TLSv1_3 会被 sslSSLContext.minimum_version 和 sslSSLContext.maximum_version 代替。
  - PROTOCOL_SSLv2, PROTOCOL_SSLv3, PROTOCOL_SSLv23, PROTOCOL_TLSv1, PROTOCOL_TLSv1_1, PROTOCOL_TLSv1_2 及 PROTOCOL_TLS 已被弃用并会被 PROTOCOL_TLS_CLIENT 和 PROTOCOL_TLS_SERVER 代替
  - wrap_socket() 会被 ssl.SSLContext.wrap_socket() 代替

  - `match_hostname()`
  - `RAND_pseudo_bytes()`, `RAND_egd()`
  - NPN 特性如 `ssl.SSLSocket.selected_npn_protocol()` 和 `ssl.SSLContext.set_npn_protocols()` 会被 ALPN 代替。

- 线程调试 (`PYTHONTHREADDEBUG` 环境变量) 在 Python 3.10 中已被弃用并将在 Python 3.12 中移除。此特性需要 Python 的调试编译版。(由 Victor Stinner 在 bpo-44584 中贡献。)
- 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 移除

- 移除了 `complex` 类的特殊方法 `__int__`, `__float__`, `__floordiv__`, `__mod__`, `__divmod__`, `__rfloordiv__`, `__rmod__` 和 `__rdivmod__`。它们总是会引发 `TypeError`。(由 Serhiy Storchaka 在 bpo-41974 中贡献。)
- `ParserBase.error()` 方法 (来自私有且未记入文档的 `_markupbase` 模块) 已被移除。`html.parser.HTMLParser` 是 `ParserBase` 的唯一子类并且它的 `error()` 实现在 Python 3.5 中已被移除。(由 Berker Peksag 在 bpo-31844 中贡献。)
- 移除了 `unicodedata.ucnhash_CAPI` 属性，它是一个内部 PyCapsule 对象。相关联的私有 `_PyUnicode_Name_CAPI` 结构体已被移至内部 C API。(由 Victor Stinner 在 bpo-42157 中贡献。)
- 移除了 `parser` 模块，它在 3.9 中由于切换到新的 PEG 解析器而与仅被旧解析器所使用的 C 源文件和头文件一起被弃用，包括 `node.h`, `parser.h`, `graminit.h` 和 `grammar.h`。
- 移除了公有 C API 函数 `PyParser_SimpleParseStringFlags`, `PyParser_SimpleParseStringFlagsFilename`, `PyParser_SimpleParseFileFlags` 和 `PyNode_Compile`，它们在 3.9 中由于切换到新的 PEG 解析器而被弃用。
- 移除了 `formatter` 模块，它在 Python 3.4 中已被弃用。它相当过时、极少被使用，并且未经测试。它最初计划在 Python 3.6 中移除，但此移除被改为延迟到 Python 2.7 生命期结束之后。现有用户应当将他们用到的所有类都拷贝到自己的代码中。(由 Dong-hee Na 和 Terry J. Reedy 在 bpo-42299 中贡献。)
- 移除了 `PyModule_GetWarningsModule()` 函数，现在它由于 _warnings 模块在 2.6 中被转换为内置模块而变得没有用处。(由 Hai Shi 在 bpo-42599 中贡献。)
- 从 `collections` 模块中移除了已被弃用的 collections-abstract-base-classes 的别名。(由 Victor Stinner 在 bpo-37324 中贡献。)
- `loop` 形参已从大部分 `asyncio` 的高层级 API 中被移除，之前它们在 Python 3.8 中已被弃用。这一改变的动机是多方面的:

  1. 这简化了高层级 API。
  2. 高层级 API 中的这些函数自 Python 3.7 起已经会隐式地获取当前线程正在运行的事件循环。在大多数正常使用场景中都没有必要向 API 传入事件循环。
  3. 在处理不同线程中运行的事件循环时传递事件循环特别容易产生错误。

  Note that the low-level API will still accept `loop`. See *Python API 的变化* for examples of how to replace existing code.

  (由 Yurii Karabas, Andrew Svetlov, Yury Selivanov 和 Kyle Stanley 在 bpo-42392 中贡献。)

10 移植到 Python 3.10

本节列出了先前描述的更改以及可能需要更改代码的其他错误修正.

10.1 Python 语法中的变化

- 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 Python API 的变化

- `traceback` 模块中的 `format_exception()` 、`format_exception_only()` 和 `print_exception()` 函数的 *etype* 参数已更名为 *exc* 。（由 Zackery Spytz 和 Matthias Bussonnier 贡献于 bpo-26389 ）
- `atexit`：在 Python 退出时，若用 `atexit.register()` 注册的回调失败，现在会记录其异常。以前，只有部分异常被记录，最后一个异常总是被静默忽略。（由 Victor Stinner 贡献于 bpo-42639 ）
- 现在，泛型 `collections.abc.Callable` 的类型参数扁平化了，类似于 `typing.Callable` 目前的做法。这意味着 `collections.abc.Callable[[int, str], str]` 的 `__args__` 将为 `(int, str, str)`；而以前是 `([int, str], str)`。通过 `typing.get_args()` 或 `__args__` 访问参数的代码需要考虑到这一变化。此外，为 `collections.abc.Callable` 给出无效参数可能会引发 `TypeError` ，而在 Python 3.9 中则可能会静默传入。（由 Ken Jin 贡献于 bpo-42195）
- 现在，如果给定的形参不是 16 位无符号整数，`socket.htons()` 和 `socket.ntohs()` 会引发 `OverflowError` 而非 `DeprecationWarning`。（由 Erlend E. Aasland 在 bpo-42393 中贡献。）
- The `loop` parameter has been removed from most of `asyncio`'s high-level API following deprecation in Python 3.8.

  现在如下协程：

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

  应替换为：

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

  如果 `foo()` 被特别设计成 不运行于当前线程的运行事件循环中（比如运行在另一个线程的事件循环中），请考虑使用 `asyncio.run_coroutine_threadsafe()` 来代替。

  （由 Yurii Karabas, Andrew Svetlov, Yury Selivanov 和 Kyle Stanley 在 bpo-42392 中贡献。）
- The `types.FunctionType` constructor now inherits the current builtins if the *globals* dictionary has no `"__builtins__"` key, rather than using `{"None": None}` as builtins: same behavior as `eval()` and `exec()` functions. Defining a function with `def function(...): ...` in Python is not affected, globals cannot be overridden with this syntax: it also inherits the current builtins. (Contributed by Victor Stinner in bpo-42990.)

10.3 C API 的变化

- 由于换成了新的 PEG 解析程序，C 语言 API 函数 “PyParser_SimpleParseStringFlags“、PyParser_SimpleParseStringFlagsFilename、PyParser_SimpleParseFileFlags、PyNode_Compile``以及这些函数用到的类型 ``struct _node 已被删除。

  现在应该用 Py_CompileString() 将源代码直接编译为代码对象。然后可以用 PyEval_EvalCode() 之类的东西来对其求值。

  特别地：

  - 先 PyParser_SimpleParseStringFlags 再 PyNode_Compile 的调用，可以由 Py_CompileString() 代替。
  - PyParser_SimpleParseFileFlags 没有直接替代品。要从 FILE * 参数编译代码，需要先用 C 语言读取文件，然后将结果缓冲区传给 Py_CompileString()。
  - 要编译一个 char * 给定文件名的文件，先显式打开该文件，再读取并进行编译。一种方法是利用 io 模块的 PyImport_ImportModule() 、PyObject_CallMethod() 、PyBytes_AsString() 和 Py_CompileString()，如下图所示。(省略了声明和错误处理部分)

    ```
    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);
    ```

  - For FrameObject objects, the f_lasti member now represents a wordcode offset instead of a simple offset into the bytecode string. This means that this number needs to be multiplied by 2 to be used with APIs that expect a byte offset instead (like PyCode_Addr2Line() for example). Notice as well that the f_lasti member of FrameObject objects is not considered stable: please use PyFrame_GetLineNumber() instead.

11 CPython 字节码的改变

- MAKE_FUNCTION 指令现在可以接受一个字典或一个字符串元组作为函数的标注。(由 Yurii Karabas 和 Inada Naoki 在 bpo-42202 中贡献。)

12 编译版的变化

- **PEP 644**：Python 现在要求 OpenSSL 1.1.1 以上版本。不再支持 OpenSSL 1.0.2。(由 Christian Heimes 贡献于 bpo-43669)
- 编译 Python 现在需要用到 C99 函数 snprintf() 和 vsnprintf() 。(由 Victor Stinner 贡献于 bpo-36020)
- sqlite3 requires SQLite 3.7.15 or higher. (Contributed by Sergey Fedoseev and Erlend E. Aasland in bpo-40744 and bpo-40810.)
- 现在，atexit 模块必须编译为内置模块。(由 Victor Stinner 贡献于 bpo-42639)
- 在 configure 脚本中加入 --disable-test-modules 选项：不编译也不安装 test 模块。(由 Xavier de Gaye、Thomas Petazzoni 和 Peixing Xin 贡献于 bpo-27640)
- 在 ./configure 脚本中加入 --with-wheel-pkg-dir=PATH 选项。如果指定了该选项，ensurepip 模块会在该目录下查找 setuptools 和 pip 包：如果两者都存在，就会使用这些包，而不是 surepip 绑定的包。

某些 Linux 发行版的打包策略建议不要绑定依赖关系。比如 Fedora 在 “/usr/share/python-wheels/“ 目录下安装 wheel 包，而不安装 “ensurepip._bundled“ 包。

（由 Victor Stinner 贡献于 bpo-42856）

- 增加了新的 `configure --without-static-libpython` 选项，用于标明不编译 `libpythonMAJOR.MINOR.a` 静态库并且不安装 `python.o` 对象文件。

  （由 Victor Stinner 在 bpo-43103 中贡献。）

- 现在，`configure` 脚本会利用 `pkg-config` 工具检测 Tcl/Tk 头文件和库的位置。此前，这些文件的位置可通过 `--with-tcltk-includes` 和 `--with-tcltk-libs` 选项显式指定。（由 Manolis Stamatogiannakis 贡献于 bpo-42603 ）

- 为 `configure` 脚本加入 `--with-openssl-rpath` 选项。该选项简化了用定制版本 OpenSSL 编译 Python 的过程，例如 `./configure --with-openssl=/path/to/openssl --with-openssl-rpath=auto`。（由 Christian Heimes 贡献于 bpo-43466 ）

13 C API 的变化

13.1 PEP 652：稳定版 ABI 的维护

现在，用于扩展模块或嵌入 Python 的稳定版 ABI （应用程序二进制接口）已有显式的定义。stable 描述了 C API 和 ABI 稳定性保证和稳定版 ABI 的最佳实践。

（由 Petr Viktorin 在 **PEP 652** 和 bpo-43795 中贡献。）.)

13.2 新的特性

- 现在 `PyNumber_Index()` 的结果一定是 `int` 类型。此前可能是 `int` 的子类实例。（由 Serhiy Storchaka 贡献于 bpo-40792 ）
- Add a new `orig_argv` member to the `PyConfig` structure: the list of the original command line arguments passed to the Python executable. (Contributed by Victor Stinner in bpo-23427.)
- 加入宏 `PyDateTime_DATE_GET_TZINFO()` 和 `PyDateTime_TIME_GET_TZINFO()` ，用于访问 `datetime.datetime` 和 `datetime.time` 对象的 `tzinfo` 属性。（由 Zackery Spytz 贡献于 bpo-30155 ）
- 加入 `PyCodec_Unregister()` 函数，用于注销编解码器检索函数。（由 Hai Shi 贡献于 bpo-41842 ）
- 加入 `PyIter_Send()` 函数，可不触发 `StopIteration` 异常地向迭代器发送数据。（由 Vladimir Matveev 贡献于 bpo-41756 ）
- 受限 C API 中加入了 `PyUnicode_AsUTF8AndSize()` 。（由 Alex Gaynor 贡献于 bpo-41784 ）
- 加入 `PyModule_AddObjectRef()` 函数：类似于 `PyModule_AddObject()` 但在成功后不会偷取参数对象的引用计数。（由 Victor Stinner 贡献于 bpo-1635741 ）
- 加入 `Py_NewRef()` 和 `Py_XNewRef()` 函数，用于递增指定对象的引用计数并返回该对象。（由 Victor Stinner 贡献于 bpo-42262 ）
- 现在，`PyType_FromSpecWithBases()` 和 `PyType_FromModuleAndSpec()` 函数可接受一个类作为 *bases* 参数。（由 Serhiy Storchaka 贡献于 bpo-42423 ）
- `PyType_FromModuleAndSpec()` 函数现在接受 NULL `tp_doc` 槽位。（由 Hai Shi 在 bpo-41832 中贡献。）
- `PyType_GetSlot()` 函数现在可以接受 静态类型。（由 Hai Shi 和 Petr Viktorin 在 bpo-41073 中贡献。）

- 新增 PySet_CheckExact() 函数到 C-API 用于检查一个对象是否是 set 的实例但不是其子类型的实例。(由 Pablo Galindo 在 bpo-43277 中贡献。)
- 增加了 PyErr_SetInterruptEx()，它允许传入一个信号序号用于进行模拟。(由 Antoine Pitrou 在 bpo-43356 中贡献。)
- 现在，Python 以调试模式编译时也支持受限 C API 的使用了（需先定义 Py_DEBUG 宏）。在受限 C API 中，如果 Python 是以调试模式编译的，且 Py_LIMITED_API 宏以 Python 3.10 以上版本为目标，那么现在 Py_INCREF() 和 Py_DECREF() 函数实现为非透明的函数调用，而非直接访问 PyObject.ob_refcnt 成员。在调试模式下支持受限 C API 成为可能，是因为自 Python 3.8 起 PyObject 结构体在发布模式和调试模式下是相同的（参见 bpo-36465 ）。

  在 --with-trace-refs 特殊编译方式下 (Py_TRACE_REFS 宏)，仍不支持使用受限 C API 。(由 Victor Stinner 贡献于 bpo-43688)
- 加入 Py_Is(x, y) 函数，用于测试 *x* 对象是否是 *y* 对象，等价于 Python 中的 x is y。还加入了 Py_IsNone() 、Py_IsTrue() 、Py_IsFalse() 函数，分别用于测试某对象是否为 None 单例、True 单例或 False 单例。(由 Victor Stinner 贡献于 bpo-43753)
- 新增由 C 代码控制垃圾回收器的函数: PyGC_Enable(), 、PyGC_Disable() 、PyGC_IsEnabled()。这些函数允许从 C 代码激活、停止和查询垃圾回收器的状态，而不必导入 gc 模块。
- 新增 Py_TPFLAGS_DISALLOW_INSTANTIATION 类型标记，用于禁止创建类型实例。(由 Victor Stinner 贡献于 bpo-43916)
- 新增 Py_TPFLAGS_IMMUTABLETYPE 类型标记，用于创建不可变类型对象：类型的属性不可设置或删除。(由 Victor Stinner 和 Erlend E. Aasland 贡献于 bpo-43908)

13.3 移植到 Python 3.10

- 现在必须定义 PY_SSIZE_T_CLEAN 才能使用 PyArg_ParseTuple() 和 Py_BuildValue()，该格式会使用 #: es#, et#, s#, u#, y#, z#, U# 和 Z#。参见 解析参数和构造值以及 **PEP 353**。(由 Victor Stinner 在 bpo-40943 中贡献。)
- 由于 Py_REFCNT() 已改为内联静态函数，Py_REFCNT(obj) = new_refcnt 必须换成 Py_SET_REFCNT(obj, new_refcnt): 参见 Py_SET_REFCNT() （自 Python 3.9 起提供)。为保持向下兼容，可用此宏:

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

  (由 Victor Stinner 在 bpo-39573 中贡献。)
- 由于历史原因，曾经允许调用 PyDict_GetItem() 时不带 GIL 。现在则不行了。(由 Victor Stinner 贡献于 bpo-40839)
- 现在,PyUnicode_FromUnicode(NULL, size) 和 PyUnicode_FromStringAndSize(NULL, size) 会引发 DeprecationWarning。请利用 PyUnicode_New() 获得不带初始数据的 Unicode 对象。(由 Inada Naoki 贡献于 bpo-36346)
- 私有结构体 _PyUnicode_Name_CAPI （PyCapsule API unicodedata.ucnhash_CAPI ）已被移入内部 C API。(由 Victor Stinner 贡献于 bpo-42157)
- 现在，若是在 Py_Initialize() 之前去调用 （在 Python 被初始化之前） ，Py_GetPath(),Py_GetPrefix() 、Py_GetExecPrefix() 、Py_GetProgramFullPath() 、Py_GetPythonHome() 和 Py_GetProgramName() 将返回 NULL。请用新的 Python 初始化配置 API 来获取 Python 路径配置。(由 Victor Stinner 贡献于 bpo-42260)
- 宏 PyList_SET_ITEM() 、PyTuple_SET_ITEM() 和 PyCell_SET() 不可再用作左值或右值。例如，现在 “x = PyList_SET_ITEM(a, b, c)“ 和 PyList_SET_ITEM(a, b, c) = x 会失败并提示编译器错误。这可以防止 if (PyList_SET_ITEM (a, b, c) < 0) ... 之类的检测发生问题。(由 Zackery Spytz 和 Victor Stinner 贡献于 bpo-30459)

- 非受限 API 文件 `odictobject.h`、`parser_interface.h`、`picklebufobject.h`、`pyarena.h`、`pyctype.h`、`pydebug.h`、`pyfpe.h` 和 `pytime.h` 已被移至 `Include/cpython` 目录。这些文件不能被直接包含，因为 `Python.h` 中已包含过了：Include 文件。如果已被直接包含，请考虑改为包含 `Python.h`。（由 Nicholas Sim 贡献于 bpo-35134 ）
- 请用 `Py_TPFLAGS_IMMUTABLETYPE` 类型标记来创建不可变对象。不要依赖 `Py_TPFLAGS_HEAPTYPE` 来确定类型对象是否可变；请改为检查是否设置了 `Py_TPFLAGS_IMMUTABLETYPE`。（由 Victor Stinner 和 Erlend E. Aasland 贡献于 bpo-43908 ）
- 未记入文档的函数 `Py_FrozenMain` 已从受限 API 中移除。该函数主要适用于 Python 的定制版。（由 Petr Viktorin 贡献于 bpo-26241 ）

13.4 弃用

- 现在 `PyUnicode_InternImmortal()` 函数已被弃用，并将在 Python 3.12 中移除：请改用 `PyUnicode_InternInPlace()`。（由 Victor Stinner 贡献于 bpo-41692 ）

13.5 移除

- 移除了 `Py_UNICODE_str*` 函数，它被用于控制 `Py_UNICODE*` 字符串。（由 Inada Naoki 在 bpo-41123 中贡献。）
  - `Py_UNICODE_strlen`: 使用 `PyUnicode_GetLength()` 或 `PyUnicode_GET_LENGTH`
  - `Py_UNICODE_strcat`: 使用 `PyUnicode_CopyCharacters()` 或 `PyUnicode_FromFormat()`
  - `Py_UNICODE_strcpy`, `Py_UNICODE_strncpy`: 使用 `PyUnicode_CopyCharacters()` 或 `PyUnicode_Substring()`
  - `Py_UNICODE_strcmp`: 使用 `PyUnicode_Compare()`
  - `Py_UNICODE_strncmp`: 使用 `PyUnicode_Tailmatch()`
  - `Py_UNICODE_strchr`, `Py_UNICODE_strrchr`: 使用 `PyUnicode_FindChar()`
- 移除了 `PyUnicode_GetMax()`。请迁移到新的 (**PEP 393**) API。（由 Inada Naoki 在 bpo-41103 中贡献。）
- 移除了 `PyLong_FromUnicode()`。请迁移到 `PyLong_FromUnicodeObject()`。（由 Inada Naoki 在 bpo-41103 中贡献。）
- 移除了 `PyUnicode_AsUnicodeCopy()`。请使用 `PyUnicode_AsUCS4Copy()` 或 `PyUnicode_AsWideCharString()` （由 Inada Naoki 在 bpo-41103 中贡献。）
- 移除了 `_Py_CheckRecursionLimit` 变量：它已被 `PyInterpreterState` 结构体的 `ceval.recursion_limit` 所取代。（由 Victor Stinner 在 bpo-41834 中贡献。）
- 移除了未记入文档的宏 `Py_ALLOW_RECURSION` 和 `Py_END_ALLOW_RECURSION` 以及 `PyInterpreterState` 结构体的 `recursion_critical` 字段。（由 Serhiy Storchaka 在 bpo-41936 中贡献。）
- 移除了未记入文档的 `PyOS_InitInterrupts()` 函数。Python 初始化时已隐式安装了信号处理 handler：参见 `PyConfig.install_signal_handlers`。（由 Victor Stinner 贡献于 bpo-41713 ）
- 移除了 `PyAST_Validate()` 函数。不能再使用公有 C API 来构建 AST 对象（`mod_ty` 类型）了。该函数已不属于受限 C API（**PEP 384** ）。（由 Victor Stinner 贡献于 bpo-43244 ）
- 移除了 `symtable.h` 头文件及未写入文档的函数:
  - `PyST_GetScope()`
  - `PySymtable_Build()`
  - `PySymtable_BuildObject()`

  - `PySymtable_Free()`
  - `Py_SymtableString()`
  - `Py_SymtableStringObject()`

  `Py_SymtableString()` 函数误为稳定版 ABI 却无法使用，因为 `symtable.h` 头文件不属于受限 C API。

  请改用 Python `symtable` 模块。（由 Victor Stinner 在 bpo-43244 中贡献。）

- `PyOS_ReadlineFunctionPointer()` 已从受限 C API 头文件和 `python3.dll` 中移除，此 dll 为 Windows 中的稳定版 ABI 库。由于该函数可接受一个 `FILE*` 参数，所以无法保证其 ABI 稳定性。（由 Petr Viktorin 贡献于 bpo-43868 ）
- 移除了 `ast.h`, `asdl.h` 和 `Python-ast.h` 头文件。这些函数未入文档且不属于受限 C API。这些头文件中定义的大多数名称都不带 `Py` 前缀，因此可能会造成命名冲突。比如 `Python-ast.h` 定义了一个 `Yield` 宏，就会与另一个 Windows `<winbase.h>` 头文件中的 `Yield` 冲突。请改用 Python `ast` 模块。（由 Victor Stinner 贡献于 bpo-43244 ）
- 移除了用到 `struct _mod` 类型的编译器和解析器函数，因为公共的 AST C API 已被移除：
  - `PyAST_Compile()`
  - `PyAST_CompileEx()`
  - `PyAST_CompileObject()`
  - `PyFuture_FromAST()`
  - `PyFuture_FromASTObject()`
  - `PyParser_ASTFromFile()`
  - `PyParser_ASTFromFileObject()`
  - `PyParser_ASTFromFilename()`
  - `PyParser_ASTFromString()`
  - `PyParser_ASTFromStringObject()`

  这些函数未入文档且不属于受限 C API。（由 Victor Stinner 贡献于 bpo-43244 ）

- 移除了包含下列函数的头文件 `pyarena.h`：
  - `PyArena_New()`
  - `PyArena_Free()`
  - `PyArena_Malloc()`
  - `PyArena_AddPyObject()`

  这些函数未记入文档，且不属于受限 C API，仅由编译器内部使用。（由 Victor Stinner 贡献于 bpo-43244 ）

- The `PyThreadState.use_tracing` member has been removed to optimize Python. (Contributed by Mark Shannon in bpo-43760.)

索引

非字母

P