What's New in Python

發🄵 *3.11.11*

A. M. Kuchling

12 月 07, 2024

Python Software Foundation
Email: docs@python.org

Contents

編輯者
Pablo Galindo Salgado

这篇文章介绍了 Python 3.11 相比 3.10 增加的新特性。Python 3.11 发布于 2022 年 10 月 24 日。要了解更详细的信息，可参阅 更新日志。

1 發布重點摘要

- Python 3.11 比 Python 3.10 快了 10-60%。我們使用了標准基准量測套裝軟體 (benchmark suite) 測得平均加速了 1.25x。細節請見*更快的 CPython*。

新增語法特性：

- *PEP 654*：*例外群組與 except**

新增棄建特性：

- *PEP 678*：*運用例外棄解使其更加詳盡*

新增標准函式庫模組：

- **PEP 680**：`tomllib` —在標准函式庫中支援 TOML 檔案的剖析

直譯器的改進：

- *PEP 657*：*回溯 (traceback) 中更細棄的錯誤位置*
- 新增 `-P` 命令列選項和 `PYTHONSAFEPATH` 環境變數以停用自動於 `sys.path` 的開頭加上一個有棄在安全問題的路徑

新增型棄特性：

- *PEP 646*：*可變參數泛型 (variadic generics)*
- *PEP 655*：*標記獨立 TypedDict 項目棄必要或不必要*
- *PEP 673*：*Self 型棄*
- *PEP 675*：*任意的文本字串型棄 (Arbitrary literal string type)*
- *PEP 681*：*資料類棄轉棄 (Data class transforms)*

重要的🅵用、移除與限制：

- **PEP 594**：許多標準函式庫中的遺留模組已被🅵用且將於 Python 3.13 移除
- **PEP 624**：*Py_UNICODE* 編碼器 *API* 已被移除
- **PEP 670**：轉🅵🅵🅵態行🅵函式的巨集

2 新增特性

2.1 PEP 657：回溯 (traceback) 中更細🅵的錯誤位置

當要印出回溯，直譯器現在會指出造成錯誤的確切運算式，而非只🅵明是哪一行。例如：

```
Traceback (most recent call last):
  File "distance.py", line 11, in <module>
    print(manhattan_distance(p1, p2))
          ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "distance.py", line 6, in manhattan_distance
    return abs(point_1.x - point_2.x) + abs(point_1.y - point_2.y)
                           ^^^^^^^^^
AttributeError: 'NoneType' object has no attribute 'x'
```

前一版本的直譯器只會標明是哪一行，無法辨認哪一個物件是 None。當處理多層的巢狀 dict 物件和多個函式呼叫時，這種🅵化錯誤提示也可能非常有用：

```
Traceback (most recent call last):
  File "query.py", line 37, in <module>
    magic_arithmetic('foo')
  File "query.py", line 18, in magic_arithmetic
    return add_counts(x) / 25
           ^^^^^^^^^^^^^
  File "query.py", line 24, in add_counts
    return 25 + query_user(user1) + query_user(user2)
                ^^^^^^^^^^^^^^^^
  File "query.py", line 32, in query_user
    return 1 + query_count(db, response['a']['b']['c']['user'], retry=True)
                           ~~~~~~~~~~~~~~~~~~~~^^^^^
TypeError: 'NoneType' object is not subscriptable
```

在🅵雜的計算運算式中也是：

```
Traceback (most recent call last):
  File "calculation.py", line 54, in <module>
    result = (x / y / z) * (a / b / c)
              ~~~~~~^~~
ZeroDivisionError: division by zero
```

此外，🅵化回溯特性所使用的資訊可以透過一般的 API 來取得，以用來使 bytecode 指示 (instruction) 與原始碼位置相互關聯。此項資訊可以用以下方式取得：

- Python 中的 `codeobject.co_positions()` 方法。
- C API 中的 `PyCode_Addr2Location()` 函式。

詳情請見 **PEP 657**。（由 Pablo Galindo、Batuhan Taskaya 與 Ammar Askar 於 bpo-43950 中所貢獻。）

備🄵: 這個特性必須要將欄的位置 (column position) 儲存於 codeobjects，這可能會導致直譯器用於編譯 Python 檔案的記憶體使用量與硬碟使用量增加。🄵了避免儲存多余的資訊且停用印出多余的回溯資訊，請用 `-X no_debug_ranges` 命令列選項或是 `PYTHONNODEBUGRANGES` 環境變數。

2.2 PEP 654：例外群組與 except*

PEP 654 引入了新的的語言特性，可讓程式同時引發🄵處理多個相互無關的例外。🄵建型🄵 `ExceptionGroup` 和 `BaseExceptionGroup` 使得程式可🄵多個例外組成群組🄵同時引發，新的 `except*` 語法也將 `except` 泛用化、能🄵比對例外群組的子群組。

詳情請見 **PEP 654**。

（由 Irit Katriel 於 bpo-45292 中所貢獻。PEP 由 Irit Katriel、Yury Selivanov 與 Guido van Rossum 撰寫。）

2.3 PEP 678：運用例外🄵解使其更加詳盡

新增 `add_note()` 方法到 `BaseException`。當上下文資訊在例外被引發時無法被取得，這個方法就可以用來🄵例外添加更多資訊。被添加的🄵解會在預設回溯中出現。

詳情請見 **PEP 678**。

（由 Irit Katriel 於 bpo-45607 中所貢獻。PEP 由 Zac Hatfield-Dodds 所撰寫）

2.4 Windows `py.exe` 🄵動程式 (launcher) 的改進

包括在 Python 3.11 中的 launcher 的副本已进行了重大更新。现在它支持 **PEP 514** 所定义的 company/tag 语法即使用 `-V:`*`<company>`*`/`*`<tag>`* 参数代替受限的 `-`*`<major>`*`.`*`<minor>`*。这允许启动托管在 python.org 上的 `PythonCore` 以外的其他发行版。

使用 `-V:` 選擇器時，可以省略公司或標🄵，但會搜索所有安裝。例如，`-V:OtherPython/` 將選擇 `OtherPython` 🄵🄵的「最佳」標🄵，而 `-V:3.11` 或 `-V:/3.11` 將選擇帶有 `3.11` 標🄵的「最佳」發行版。

在使用旧式的 `-`*`<major>`*、`-`*`<major>`*`.`*`<minor>`*、`-`*`<major>`*`-`*`<bitness>` 或 `-`*`<major>`*`.`*`<minor>`*`-`*`<bitness>` 参数时，应保留过去版本的所有已有行为，并只选择从 `PythonCore` 发布的版本。不过，`-64` 后缀现在表示“非 32 位”（不一定是 x86-64），因为有多种受支持的 64 位平台。32 位运行时是通过检查运行时的标签是否有 `-32` 后缀来检测的。自 Python 3.5 以来的所有版本都在其 32 位编译中包括了这个后缀。

3 型🄵提示相關的新特性

這個部分涵蓋影響 **PEP 484** 型🄵提示 (type hints) 與 `typing` 模組的重大變更。

3.1 PEP 646：可變參數泛型 (variadic generics)

PEP 484 先前引入了 `TypeVar`，開🄵了帶有單一型🄵的泛型參數化。**PEP 646** 新增 `TypeVarTuple`，開🄵了帶有任意數量型🄵的參數化 (parameterisation)。🄵句話🄵，`TypeVarTuple` 是可變的型🄵變數，🄵用了可變的泛型。

這使其有非常多用例，特🄴是它允許了像是 NumPy 和 Tensorflow 的數值運算函式庫中類似陣列結構的型🄵可用 *shape* 陣列來被參數化。🄵態型🄵檢查工具現在也能🄵🄵使用這些函式庫的程式捕捉到維度相關的錯誤。

詳情請見 **PEP 646**。

（由 Matthew Rahtz 於 bpo-43224 中所貢獻，由 Serhiy Storchaka 與 Jelle Zijlstra 協助。PEP 由 Mark Mendoza、Matthew Rahtz、Pradeep Kumar Srinivasan 與 Vincent Siles 所撰寫）

3.2 PEP 655：標記獨立 `TypedDict` 項目🄵必要或不必要

`Required` 與 `NotRequired` 提供了標記一個獨立項目在 `TypedDict` 中是否必須存在的直覺方法。在這之前，這只有透過繼承才有可能做得到。

所有欄位都預設🄴是必要的，除非 *total* 參數有被設🄴 `False`，那🄴所有欄位就會是非必要的。例如，這個範例指定了要有一個必要鍵與一個非必要鍵的 `TypedDict`：

```
class Movie(TypedDict):
   title: str
   year: NotRequired[int]

m1: Movie = {"title": "Black Panther", "year": 2018}  # OK
m2: Movie = {"title": "Star Wars"}  # OK (year is not required)
m3: Movie = {"year": 2022}  # ERROR (missing required field title)
```

以下定義等同於：

```
class Movie(TypedDict, total=False):
   title: Required[str]
   year: int
```

詳情請見 **PEP 655**。

（由 David Foster 與 Jelle Zijlstra 於 bpo-47087 中所貢獻。PEP 由 David Foster 所撰寫）

3.3 PEP 673：`Self` 型🄵

新的 `Self` 標🄴提供了一種簡單直觀的方法來標🄵那些會回傳其類🄵實例的方法。這與 **PEP 484** 中指定的基於 `TypeVar` 的方法相同，但更簡潔且更易於遵循。

常見用例包括作🄵 `classmethod` 的替代建構函式和會回傳 `self` 的 `__enter__()` 方法：

```
class MyLock:
    def __enter__(self) -> Self:
        self.lock()
        return self

    ...

class MyInt:
    @classmethod
```

(繼續下一頁)

(繼續上一頁)

```
    def fromhex(cls, s: str) -> Self:
        return cls(int(s, 16))

    ...
```

`Self` 也可用於標🄵與其封閉類類🄵 (enclosing class) 相同的方法參數或屬性。

詳情請見 **PEP 673**。

（由 James Hilton-Balfe 於 bpo-46534 中所貢獻。PEP 由 Pradeep Kumar Srinivasan 與 James Hilton-Balfe 所撰寫）

3.4 PEP 675：任意的文本字串型🄵 (Arbitrary literal string type)

新的 `LiteralString` 標🄵可用於標示一個函式參數可🄵任何文本字串型🄵 (literal string type)。這允許函式接受任意的文本字串型🄵，以及從其他文本字串創建的字串。型🄵檢查器就可以🄵制需審慎處理的函式（例如會執行 SQL 陳述式或 shell 命令的函式）僅會以🄵態引數呼叫，從而提供針對注入攻擊的保護。

例如一個 SQL 查詢函式 (query function) 可以被標🄵🄵：

```
def run_query(sql: LiteralString) -> ...
    ...

def caller(
    arbitrary_string: str,
    query_string: LiteralString,
    table_name: LiteralString,
) -> None:
    run_query("SELECT * FROM students")       # ok
    run_query(query_string)                   # ok
    run_query("SELECT * FROM " + table_name)  # ok
    run_query(arbitrary_string)               # type checker error
    run_query(                                # type checker error
        f"SELECT * FROM students WHERE name = {arbitrary_string}"
    )
```

詳情請見 **PEP 675**。

（由 Jelle Zijlstra 於 bpo-47088 中所貢獻。PEP 由 Pradeep Kumar Srinivasan 與 Graham Bleaney 所撰寫）

3.5 PEP 681：資料類🄵轉🄵 (Data class transforms)

`dataclass_transform` 可以用來裝飾一個類🄵、元類🄵 (metaclass)、或是一個本身就是裝飾器的函式。`@dataclass_transform()` 的存在會讓一個🄵態型🄵檢查器知道被裝飾物件會在運行程式做出轉🄵類🄵的「魔法」，賦予其類似 `dataclass` 的行🄵。

舉例來🄵：

```
# The create_model decorator is defined by a library.
@typing.dataclass_transform()
def create_model(cls: Type[T]) -> Type[T]:
    cls.__init__ = ...
    cls.__eq__ = ...
    cls.__ne__ = ...
    return cls
```

(繼續下一頁)

(繼續上一頁)

```
# The create_model decorator can now be used to create new model classes:
@create_model
class CustomerModel:
    id: int
    name: str

c = CustomerModel(id=327, name="Eric Idle")
```

詳情請見 **PEP 681**。

（由 Jelle Zijlstra 於 gh-91860 中所貢獻。PEP 由 Erik De Bonte 與 Eric Traut 所撰寫）

3.6 PEP 563 可能不是未來

PEP 563 標🄴的推遲求值 (Postponed Evaluation of Annotations)（`from __future__ import annotations` future 陳述式）最初計劃在 Python 3.10 中發布，但已被無限期擱置。請參🄵來自指導委員會 (Steering Counsil) 的訊息以獲得更多資訊。

4 其他語言更動

- 星號拆解 (starred unpacking) 運算式現在可以在 `for` 陳述式中使用。（詳情請見 bpo-46725。）
- 非同步綜合運算 (comprehension) 現在允許在非同步函式中的🄴部綜合運算 (inside comprehension)。在這種情🄵下，外部綜合運算 (outer comprehension) 隱晦地變成了非同步的了。（由 Serhiy Storchaka 在 bpo-33346 中貢獻。）
- 現在在不支援 context manager 協議的物件上使用 `with` 陳述式和 `contextlib.ExitStack.enter_context()` 或在不支援 asynchronous context manager 協議的物件上使用 `async with` 陳述式和 `contextlib.AsyncExitStack.enter_async_context()`，會引發 `TypeError` 而不是 `AttributeError`。（由 Serhiy Storchaka 在 bpo-12022 和 bpo-44471 中貢獻。）
- 增加了 `object.__getstate__()`，它提供 `__getstate__()` 方法的默认实现。`copy` 并 `pickle` 内置类型 `bytearray`, `set`, `frozenset`, `collections.OrderedDict`, `collections.deque`, `weakref.WeakSet` 和 `datetime.tzinfo` 的子类的实例现在将会拷贝并封存被实现为 槽位的实例属性。此项改变有一个意外的附带影响：它将扰乱少数不使用 `object.__getstate__()` 的现有 Python 项目。请参阅 gh-70766 上近期的评论了解有关此类代码所需处理的讨论。（由 Serhiy Storchaka 在 bpo-26579 中贡献。）
- 新增了一個 `-P` 命令列選項和一個 `PYTHONSAFEPATH` 環境變數，它們禁用了當使用 `-c` 和 `-m` 以在運行🄵本或當前目🄵時自動添加到🄵本目🄵的 `sys.path`。這確保只有 stdlib 和已安裝的模組會被 `import` 取用，以避免不小心或被惡意地將模組與本地（通常是使用者可寫入的）目🄵中的模組重🄵。（由 Victor Stinner 在 gh-57684 中貢獻。）
- `"z"` 選項被新增到 formatspec，它會🄴迫負的 0 在進位到格式精度後成🄴正的。更多詳情請見 **PEP 682**。（由 John Belmonte 於 gh-90153 中貢獻。）
- `sys.path` 不再接受位元組。支援已在 Python 3.2 和 3.6 之間的某個時間停止，直到 Python 3.10.0 發布後才引起人們的注意。此外，由於混合使用 `str` 和 `bytes` 鍵時 `-b` 和 `sys.path_importer_cache` 會出現的交互作用，恢復這項支援會出現問題。（由 Thomas Grainger 在 gh-91181 中貢獻。）

5 其他 CPython 實作更動

- 实现了用于 complex 的 __complex__() 和用于 bytes 的 __bytes__() 特殊方法以支持 typing.SupportsComplex 和 typing.SupportsBytes 协议。（由 Mark Dickinson 和 Donghee Na 在 bpo-24234 中贡献。）
- 新增 siphash13 以作F F部的雜F演算法，它有與 siphash24 相似的安全特性，但是在處理較長的輸入時會更快一些。現在是 str、bytes 和一些其他型F的 hash() 預設演算法。**PEP 552** 基於雜F的.pyc 檔案現在也使用 siphash13。（由 Inada Naoki 於 bpo-29410 中貢獻。）
- 當一個仍有效的例外被 raise 陳述式在F有參數的情F下重新引發，被附於該例外的追F資訊現在都會是 sys.exc_info()[1].__traceback__。這代表對於當前 except 子句的追F上做的改動會反映在被重F引發的例外上。（由 Irit Katriel 於 bpo-45711 中貢獻。）
- 有被處理的例外在直譯器狀態的表示（也就是 exc_info 或 _PyErr_StackItem）現在只會有 exc_value 欄位；exc_type 和 exc_traceback 已被移除，現在只能透過 exc_value 來取得它們。（由 Irit Katriel 於 bpo-45711 中貢獻。）
- 新增命令列選項 AppendPath，已增加於 Windows 安裝程式。它的行F類似於 PrependPath，但在安裝和F本目F後面附加而非新增於它們前面。（由 Bastian Neuburger 在 bpo-44934 中貢獻。）
- 初始化中若是要用 PyConfig.module_search_paths 來初始化 sys.path，則現在 PyConfig.module_search_paths_set 必須被設F 1。否則，初始化會重新計算路徑F取代所有被加到 module_search_paths 的值。
- --help 選項的輸出現在會在 50 列、80 欄的大小之F，Python 環境變數和 -X 選項的資訊現在能F分F透過 --help-env 和 --help-xoptions 旗標與 --help-all 一起使用來取得。（由 Éric Araujo 於 bpo-46142 中貢獻。）
- 在除 2（binary、二進制）、4、8（octal、八進制）、16（hexadecimal、十六進制）或 32 以外的基數中，例如以 10（decimal、十進制）F基數，進行 int 和 str 之間的轉F且字串形式的位數超過限制，現在會引發 ValueError，以避免由於演算法F雜性而導致的F在阻斷服務攻擊 (denial of service attacks)。這是針對 CVE-2020-10735 的緩解措施，可以透過環境變數、命令列旗標或 sys API 來設定或禁用此限制。請參F整數字串轉F長度限制文件。預設限制F字串形式的 4300 位數字。

6 新增模組

- tomllib：用於剖析 TOML。詳情請見 **PEP 680**。（由 Taneli Hukkinen 於 bpo-40059 中所貢獻。）
- wsgiref.types：**WSGI** 限定型F，用於F態型F檢查。（Sebastian Rittau 於 bpo-42012 中所貢獻。）

7 模組改進

7.1 asyncio

- 添加了 TaskGroup 類F，F一個會持有任務群組F在退出時等待全部完成的非同步情境管理器 (asynchronous context manager)。對於新程式碼，建議直接使用 create_task() 和 gather()。（由 Yury Selivanov 和其他人在 gh-90908 中貢獻。）
- 新增 timeout()，是一個用來F一個非同步操作設置超時的非同步情境管理器，新的程式建議直接使用它以取代 wait_for()。（由 Andrew Svetlov 於 gh-90927 貢獻。）
- 新增 Runner 類F，它會對外公布了 run() 的使用機制。（由 Andrew Svetlov 於 gh-91218 貢獻。）

- 於 asyncio 函式庫的同步化原始物件中新增 `Barrier` 類🄵與和其相關的 `BrokenBarrierError` 例外。(由 Yves Duprat 和 Andrew Svetlov in gh-87518 貢獻。)
- 在 `asyncio.loop.create_connection()` 新增關鍵字引數 *all_errors*，這樣多個連接錯誤就可以一起用一個 `ExceptionGroup` 來引發。
- 新增 `asyncio.StreamWriter.start_tls()` 方法，用來將已存在的串流連🄵升級至 TLS。(由 Ian Good 於 bpo-34975 中貢獻。)
- 在事件🄵圈增加原始資料元 (raw datagram) socket 函式：`sock_sendto()`、`sock_recvfrom()` 和 `sock_recvfrom_into()`。以上在 `SelectorEventLoop` 和 `ProactorEventLoop` 中都有實作。(由 Alex Grönholm 在 bpo-46805 中貢獻。)
- 於 `Task` 新增 `cancelling()` 和 `uncancel()` 方法。這些預期是只用於🄵部，尤其是 `TaskGroup`。

7.2 contextlib

- 添加了非平行安全的 `chdir()` 情境管理器來更改當前工作目🄵，然後在退出時恢復它。`chdir()` 的簡單包裝器。(由 Filipe Laíns 在 bpo-25625 中貢獻)

7.3 dataclasses

- 更改欄位預設的可變性檢查 (mutability check)，僅允許預設值是 hashable 而不是任何非 `dict`、`list` 或 `set` 實例的物件。(由 Eric V. Smith 在 bpo-44674 中貢獻。)

7.4 datetime

- 增加了 `datetime.UTC`，是 `datetime.timezone.utc` 的便捷別名。(由 Kabir Kwatra 在 gh-91973 中贡献。)
- `datetime.date.fromisoformat()`、`datetime.time.fromisoformat()` 和 `datetime.datetime.fromisoformat()` 現在可以用來剖析大部分的 ISO 8601 格式（除了那些支援分數形式的小時與分鐘）。(由 Paul Ganssle 於 gh-80010 中所貢獻。)

7.5 enum

- `EnumMeta` 更名🄵 `EnumType`（`EnumMeta` 保留🄵🄵名）。
- 增加 `StrEnum`，列舉 (enum) 🄵的成員必須是字串。
- 新增 `ReprEnum`，它只修改成員的 `__repr__()`，同時回傳成員的文本值 (literal value)（而不是名稱），以用於（🄵 `str()`、`format()` 和 f-string 所使用的）`__str__()` 和 `__format__()`。
- 修改了 `Enum.__format__()` (為 `format()`, `str.format()` 和 f-string 的默认值) 以便始终产生于 `Enum.__str__()` 相同的结果：对于继承自 `ReprEnum` 的枚举它将成为其成员的值；对于所有其他枚举它将为枚举和成员名称 (例如 `Color.RED`)。
- 新增 *boundary* 類🄵參數與其選項到 `Flag` 列舉和 `FlagBoundary` 列舉以控制處理超出範圍旗標數值的方法。
- 新增了 `verify()` 列舉裝飾器和 `EnumCheck` 列舉及其選項，以根據幾個特定限制檢查列舉類🄵。
- 新增 `member()` 與 `nonmember()` 裝飾器以確保被裝飾的物件會/不會被轉🄵成一個列舉成員。
- 新增 `property()` 裝飾器，它的作用類似 `property()` 但是是用於列舉，用以替代 `types.DynamicClassAttribute()`。

- 增加了 `global_enum()` 枚举装饰器，它会调整 `__repr__()` 和 `__str__()` 以将值显示为其模块的成员而不是枚举类的成员。例如，`'re.ASCII'` 是 `re.RegexFlag` 的 `ASCII` 成员而不是 `'RegexFlag.ASCII'`。
- 🄵化 `Flag` 以支援使用 `len()`、🄵代 (iteration) 和 `in`/`not in` 於其成員。例如，以下程式現在能🄵作用了：`len(AFlag(3)) == 2 and list(AFlag(3)) == (AFlag.ONE, AFlag.TWO)`
- 更改了 `Enum` 和 `Flag` 以在呼叫 `__init_subclass__()` 之前就定義成員；`dir()` 現在包括來自混合資料型🄵的方法。
- 更改 `Flag` 以僅考慮主要值 (2 的次方) 規範，而🄵合值 (`3`、`6`、`10` 等) 被視🄵🄵名；倒置旗標 (inverted flags) 會被🄵制轉🄵🄵正等價的值。

7.6 fcntl

- FreeBSD 上，`F_DUP2FD` 和 `F_DUP2FD_CLOEXEC` 旗標分🄵有被支援，前者等同於 `dup2` 用法，而後者設定了 `FD_CLOEXEC` 旗標。

7.7 fractions

- 支援有 **PEP 515** 風格的 `Fraction` 以字串初始化。(Sergey B Kirpichev 於 bpo-44258 中所貢獻。)
- `Fraction` 現在有實作了一個 `__int__` 方法，因此 `isinstance(some_fraction, typing.SupportsInt)` 的檢查會通過。(由 Mark Dickinson 在 bpo-44547 中貢獻。)

7.8 functools

- `functools.singledispatch()` 現在支援 `types.UnionType` 和 `typing.Union` 作🄵調度 (dispatch) 引數的標🄵。

```
>>> from functools import singledispatch
>>> @singledispatch
... def fun(arg, verbose=False):
...     if verbose:
...         print("Let me just say,", end=" ")
...     print(arg)
...
>>> @fun.register
... def _(arg: int | float, verbose=False):
...     if verbose:
...         print("Strength in numbers, eh?", end=" ")
...     print(arg)
...
>>> from typing import Union
>>> @fun.register
... def _(arg: Union[list, set], verbose=False):
...     if verbose:
...         print("Enumerate this:")
...     for i, elem in enumerate(arg):
...         print(i, elem)
...
```

(由 Yurii Karabas 於 bpo-46014 中所貢獻。)

7.9 gzip

- 现在 `gzip.compress()` 函数当传入 **mtime=0** 参数时会更快速因为它把压缩任务完全委托给单独的 `zlib.compress()` 操作。此项改变有一个附带影响：gzip 文件标头将在其标头中包含一个"OS" 字节。在传统上它总是会被 `gzip` 模块设为代表"unknown" 的值 255。现在，当使用 `compress()` 并传入 **mtime=0** 时，它可以被 Python 所链接的下层 zlib C 库设为不同的值。（请参阅 gh-112346 了解此附带影响的详情。）

7.10 hashlib

- `hashlib.blake2b()` 與 `hashlib.blake2s()` 現在偏好使用 libb2 多於 Python 自發行版的🄵🄵。（由 Christian Heimes 於 bpo-47095 中所貢獻。）
- 帶有 SHA3 和 SHAKE 演算法的🄵部 `_sha3` 模組現在使用 *tiny_sha3* 而不是 *Keccak 程式碼套件*來🄵少程式碼和二進位檔案大小。`hashlib` 模組更喜歡來自 OpenSSL 的 SHA3 和 SHAKE 最佳化實作。此更改僅影響🄵有 OpenSSL 支援的安裝。（由 Christian Heimes 在 bpo-47098 中貢獻。）
- 新增 `hashlib.file_digest()`，是個能🄵🄵檔案或類檔案物件做高效率雜🄵的幫助函式。（由 Christian Heimes 於 gh-89313 中貢獻。）

7.11 IDLE 與 idlelib

- 在 *.pyi* 檔案施用語法突顯 (syntax highlight)。（由 Alex Waygood 與 Terry Jan Reedy 於 bpo-45447 中所貢獻。）
- 當帶有輸入與輸出地儲存 Shell 時，也會包含提示字元。（由 Terry Jan Reedy 於 gh-95191 中所貢獻。）

7.12 inspect

- 添加 `getmembers_static()` 以回傳所有成員，而不會通過描述器協議 (descriptor protocol) 觸發動態查找。（由 Weipeng Hong 在 bpo-30533 中貢獻。）
- 新增 `inspect.ismethodwrapper()`，用來檢查一個物件的型🄵是否🄵 `MethodWrapperType`。（由 Hakan Çelik 於 bpo-29418 中所貢獻。）
- 更改 `inspect` 模組中與幀相關的函式以回傳新的 `FrameInfo` 和 `Traceback` 類🄴實例（向後相容之前類似於 named tuple 的介面），包括擴充的 **PEP 657** 位置資訊（結束行號、欄和結束欄）。受影響的功能是：
  - `inspect.getframeinfo()`
  - `inspect.getouterframes()`
  - `inspect.getinnerframes()`,
  - `inspect.stack()`
  - `inspect.trace()`

  （由 Pablo Galindo 於 gh-88116 中所貢獻。）

7.13 locale

- 新增 `locale.getencoding()` 以取得當前的區域編碼 (locale encoding)。和 `locale.getpreferredencoding(False)` 類似但不考慮 Python UTF-8 模式。

7.14 logging

- 新增 `getLevelNamesMapping()` 以回傳一個日🄵級🄵名稱（例如 `'CRITICAL'`）指到對應的 levels 數值（例如，預設🄵 `50`）的映射。(由 Andrei Kulakovin 於 gh-88024 中貢獻。)
- 添加了一個 `createSocket()` 方法到 `SysLogHandler`，以匹配 `SocketHandler.createSocket()`。如果🄵有已🄵用的 socket，它會在處理程式初始化期間和發出一個事件時自動呼叫。(由 Kirill Pinchuk 在 gh-88457 中貢獻。)

7.15 math

- 新增 `math.exp2()`：回傳 2 的 x 次方。(由 Gideon Mitchell 於 bpo-45917 中所貢獻。)
- 新增 `math.cbrt()`：回傳 x 的立方根。(由 Ajith Ramachandran 於 bpo-44357 中所貢獻。)
- 🄵了與 IEEE 754 規範保持一致，更改了兩個 `math.pow()` 邊角案例 (corner case) 的行🄵。`math.pow(0.0, -math.inf)` 和 `math.pow(-0.0, -math.inf)` 現在回傳 `inf`，之前它們會引發 `ValueError`。(由 Mark Dickinson 在 bpo-44339 中貢獻。)
- `math.nan` 現🄵隨時可用。(由 Victor Stinner 於 bpo-46917 中所貢獻。)

7.16 operator

- 新增 `operator.call` 函式，使得 `operator.call(obj, *args, **kwargs) == obj(*args, **kwargs)`。(由 Antony Lee 於 bpo-44019 中所貢獻。)

7.17 os

- 在 Windows 上，`os.urandom()` 現在將使用 `BCryptGenRandom()`，而不是已被弃用的 `CryptGenRandom()`。(由 Donghee Na 在 bpo-44611 中贡献。)
- As of 3.11.10, `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.)

7.18 pathlib

- 如果 *pattern* 以路徑名稱元件分隔符號 `sep` 或 `altsep` 結尾，`glob()` 和 `rglob()` 只回傳目🄵。(由 Eisuke Kawasima 於 bpo-22276 與 bpo-33392 中所貢獻。)

7.19 re

- 現在規則運算式 (regular expression) 是有支援原子性群組 (atomic grouping) (`(?>...)`) 和🄵有性量詞 (possessive quantifier) (`*+`, `++`, `?+`, `{m,n}+`) 的。(由 Jeffrey C. Jacobs 和 Serhiy Storchaka 在 bpo-433030 中貢獻。)

7.20 shutil

- 新增 `shutil.rmtree()` 的可選參數 *dir_fd*。(由 Serhiy Storchaka 於 bpo-46245 中所貢獻。)

7.21 socket

- 新增 NetBSD 對於 CAN Socket 的支援。(由 Thomas Klausner 於 bpo-30512 中所貢獻。)
- 當連接失敗時，`create_connection()` 有個選項可以引發一個包含所有錯誤的 `ExceptionGroup`，而非只引發最後一個錯誤。(由 Irit Katriel 於 bpo-29980 中貢獻。)

7.22 sqlite3

- 現在可以透過將 `None` 傳遞給 `set_authorizer()` 來停用 authorizer。(由 Erlend E. Aasland 於 bpo-44491 中貢獻。)
- 定序 (collation) 名稱 `create_collation()` 現在可以包含任何 Unicode 字元。帶有無效字元的定序名稱現在會引發 `UnicodeEncodeError` 而不是 `sqlite3.ProgrammingError`。(由 Erlend E. Aasland 在 bpo-44688 中貢獻。)
- `sqlite3` 例外現在包含 SQLite 擴充錯誤碼和 SQLite 錯誤名稱 (分🄵🄵 `sqlite_errorcode` 和 `sqlite_errorname`)。(由 Aviv Palivoda、Daniel Shahaf 和 Erlend E. Aasland 在 bpo-16379 和 bpo-24139 中貢獻。)
- 將 `setlimit()` 和 `getlimit()` 新增到 `sqlite3.Connection` 以根據連🄵來設定和取得 SQLite 限制。(由 Erlend E. Aasland 在 bpo-45243 中貢獻。)
- `sqlite3` 現在會基於底層 SQLite 函式庫編譯時所使用的預設執行緒模式來設定 `sqlite3.threadsafety`。(由 Erlend E. Aasland 在 bpo-45613 中貢獻。)
- `sqlite3` 如果🄵用回呼回溯 (callback traceback)，C 回呼現在使用無法被引發的例外。使用者現在可以🄵🄵一個無法被引發的🄵鉤處理程式 (`unraisable hook handler`) 來改善他們的除錯體驗。(由 Erlend E. Aasland 在 bpo-45828 中貢獻。)
- 跨越不同回🄵 (rollback) 的拿取動作不再引發 `InterfaceError`，我們將其留給 SQLite 函式庫來處理這些情🄵。(由 Erlend E. Aasland 在 bpo-44092 中貢獻。)
- 將 `serialize()` 和 `deserialize()` 新增到 `sqlite3.Connection` 以用於序列化和反序列化資料庫。(由 Erlend E. Aasland 在 bpo-41930 中貢獻。)
- 於 `sqlite3.Connection` 加入 `create_window_function()` 已建立聚合視窗函式 (aggregate window function)。(由 Erlend E. Aasland 於 bpo-34916 中貢獻。)
- 在 `sqlite3.Connection` 新增 `blobopen()`。`sqlite3.Blob` 允許對 blob 進行增量 I/O 操作 (incremental I/O operations)。(由 Aviv Palivoda 和 Erlend E. Aasland 在 bpo-24905 中貢獻。)

7.23 string

- 新增 `get_identifiers()` 和 `is_valid()` 於 `string.Template`，分🄵能🄵回傳所有合格的預留位置 (placeholder) 與是否有任何不合格的預留位置存在。（由 Ben Kehoe 於 gh-90465 中貢獻。）

7.24 sys

- `sys.exc_info()` 現在從 `value`（例外實例）衍生出 `type` 和 `traceback` 欄位，因此當例外在處理過程中被修改時，變更會反映在 `exc_info()` 後續呼叫的結果中。（由 Irit Katriel 在 bpo-45711 中貢獻。）
- 新增會回傳活躍例外實例 (active exception instance) 的 `sys.exception()`（等價於 `sys.exc_info()[1]`）。（由 Irit Katriel 於 bpo-46328 中所貢獻。）
- 新增 `sys.flags.safe_path` 旗標。（由 Victor Stinner 於 gh-57684 中所貢獻。）

7.25 sysconfig

- 新增了三個安裝方案（*posix_venv*、*nt_venv* 和 *venv*），它們在 Python 建立新的🄴擬環境或在🄴擬環境中執行環境使用。前兩個方案（*posix_venv* 和 *nt_venv*）是非 Windows 和 Windows 作業系統所特有的，*venv* 本質上會根據 Python 運行的操作系統來做🄴其中之一的🄴名。這對修改 `sysconfig.get_preferred_scheme()` 的下游發布者很有用。建立新🄴擬環境的第三方程式碼應該使用新的 *venv* 安裝方案來確定路徑，就像 `venv` 一樣。（由 Miro Hrončok 在 bpo-45413 中貢獻。）

7.26 tempfile

- `SpooledTemporaryFile` 物件現在完整實作了 `io.BufferedIOBase` 或 `io.TextIOBase` 的方法（取🄵於檔案模式），這使它們能🄵正確地使用需要類檔案物件的 API，例如壓縮模組。（由 Carey Metcalfe 在 gh-70363 中貢獻。）
- As of 3.11.10 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.)

7.27 threading

- 在 Unix 上，如果 `sem_clockwait()` 函数存在于 C 库中 (即 glibc 2.30 及更新的版本)，则 `threading.Lock.acquire()` 方法现在将使用单调时钟 (`time.CLOCK_MONOTONIC`) 来计算超时，而不使用系统时钟 (`time.CLOCK_REALTIME`)，以不受系统时钟修改的影响。（由 Victor Stinner 在 bpo-41710 中贡献。）

7.28 time

- 在 Unix 上，如果可用的話，`time.sleep()` 現在會使用 `clock_nanosleep()` 或 `nanosleep()` 函式，其解析度🄵 1 納秒（10^{-9} 秒），而不是使用解析度🄵 1 微秒（10^{-6} 秒）的 `select()`。（由 Benjamin Szőke 和 Victor Stinner 在 bpo-21302 中貢獻。）
- 在 Windows 8.1 或更新版本上，现在 `time.sleep()` 会使用一个基于 高精度计时器 的可等待计时器，其精度为 100 纳秒 (10^{-7} 秒)。在之前版本中，其精度为 1 毫秒 (10^{-3} 秒)。（由 Benjamin Szőke, Donghee Na, Eryk Sun 和 Victor Stinner 在 bpo-21302 和 bpo-45429 中贡献。）

7.29 tkinter

- 新增了 `info_patchlevel()` 方法，它會回傳 Tcl 函式庫的確切版本以作🄵類似於 `sys.version_info` 的附名元組。（由 Serhiy Storchaka 在 gh-91827 中貢獻。）

7.30 traceback

- 新增 `traceback.StackSummary.format_frame_summary()` 以允許使用者覆蓋回溯中出現的幀及它們的格式。（由 Ammar Askar 在 bpo-44569 中貢獻。）
- 新增 `traceback.TracebackException.print()`，它會印出格式化的 `TracebackException` 實例至一個檔案。（由 Irit Katriel 在 bpo-33809 中貢獻。）

7.31 typing

重大變更請見型🄵提示相關的新特性。

- 新增 `typing.assert_never()` 和 `typing.Never`。`typing.assert_never()` 可用於要型🄵檢查器確認某行程式碼是否不可觸及。在執行環境，它會引發 `AssertionError`。（由 Jelle Zijlstra 在 gh-90633 中貢獻。）
- 新增 `typing.reveal_type()`，這可用於請求型🄵檢查器🄵給定運算式推斷出什🄵型🄵。在執行環境它會印出接收到的值的型🄵。（由 Jelle Zijlstra 在 gh-90572 中貢獻。）
- 新增 `typing.assert_type()`，這可用於要型🄵檢查器確認它🄵給定運算式推斷的型🄵是否與給定型🄵相符。在執行環境，它只會回傳接收到的值。（由 Jelle Zijlstra 在 gh-90638 中貢獻。）
- `typing.TypedDict` 型🄵現可🄵泛型。（由 Samodya Abeysiriwardane 於 gh-89026 中所貢獻。）
- `NamedTuple` 型🄵現可🄵泛型。（由 Serhiy Storchaka 於 bpo-43923 中所貢獻。）
- 允許繼承 `typing.Any`，這能有效避免與高度動態類🄴（例如 mock）相關的型🄵檢查器錯誤。（由 Shantanu Jain 在 gh-91154 中貢獻。）
- `typing.final()` 裝飾器現在會在被裝飾的物件上設定 `__final__` 屬性。（由 Serhiy Storchaka 於 gh-90500 中所貢獻。）
- `typing.get_overloads()` 函式可用於自我檢查 (introspect) 一個函式的過載 (overload)。`typing.clear_overloads()` 可用於清除一個函式的所有已🄵🄵過載。（由 Jelle Zijlstra 在 gh-89263 中貢獻。）
- `Protocol` 子類🄴的 `__init__()` 方法現在被保留。（由 Adrian Garcia Badarasco 在 gh-88970 中貢獻。）
- 空元組型🄵 (`Tuple[()]`) 的表示法得到簡化，這會影響自我檢查 (introspection)，例如 `get_args(Tuple[()])` 的求值現在會是 `()` 而不是 `((),)`。（由 Serhiy Storchaka 在 gh-91137 中貢獻。）

- 通過🄵除私有 `typing._type_check` 函式中的可呼叫檢查，放寬型🄵標🄵的執行環境要求。（由 Gregory Beauregard 在 gh-90802 中貢獻。）
- 作🄵 PEP 585 泛化🄵名中的前向參照，`typing.get_type_hints()` 現支援了🄵字串求值 (evaluate)。（由 Niklas Rosenstein 在 gh-85542 中貢獻。）
- `typing.get_type_hints()` 不再將 `Optional` 新增到預設🄵 `None` 的參數中。（由 Nikita Sobolev 在 gh-90353 中貢獻。）
- `typing.get_type_hints()` 現在支援🄵無修飾 (bare) 字串化 (stringified) 的 `ClassVar` 標🄵來求值。（由 Gregory Beauregard 在 gh-90711 中貢獻。）
- `typing.no_type_check()` 不再修改外部類🄵和函式。它現在也正確地將類🄵方法標記🄵不需進行型🄵檢查。（由 Nikita Sobolev 在 gh-90729 中貢獻。）

7.32 unicodedata

- Unicode 資料庫被更新🄵 14.0.0 版本。（Benjamin Peterson 於 bpo-45190 中所貢獻。）

7.33 unittest

- 新增 `TestCase` 類🄵的 `enterContext()` 與 `enterClassContext()` 方法、`IsolatedAsyncioTestCase` 類🄵的 `enterAsyncContext()` 方法、`unittest.enterModuleContext()` 函式。（由 Serhiy Storchaka 於 bpo-45046 貢獻。）

7.34 venv

- 建立新的 Python 🄵擬環境時，*venv* sysconfig 安裝方案會被用於確定環境🄵的路徑。當 Python 在🄵擬環境中運行時，預設使用相同的安裝方案。這意味著下游發布者可以在不改變🄵擬環境行🄵的情🄵下更改預設的 sysconfig 安裝方案。建立新🄵擬環境的第三方程式碼也應該這樣做。（由 Miro Hrončok 在 bpo-45413 中貢獻。）

7.35 warnings

- `warnings.catch_warnings()` 現在接受 `warnings.simplefilter()` 的引數，提供了一種更簡潔的方法來在本地端忽略警告或將它們轉🄵🄵錯誤。（由 Zac Hatfield-Dodds 在 bpo-47074 中貢獻。）

7.36 zipfile

- 新增了對指定成員名稱編碼的支援，以便在 `ZipFile` 的目🄵和檔案標頭中讀取元資料 (metadata)。（由 Stephen J. Turnbull 和 Serhiy Storchaka 在 bpo-28080 中貢獻。）
- 新增 `ZipFile.mkdir()` 以在 ZIP 歸檔中建立新的目🄵。（由 Sam Ezeh 於 gh-49083 貢獻。）
- 於 `zipfile.Path` 新增 `stem`、`suffix` 和 `suffixes`。（由 Miguel Brito 於 gh-88261 貢獻。）

8 最佳化

這個部分會涵蓋到特定的最佳化，但獨立於擁有自己一個🄵明的更快的 *CPython* 計畫。

- 編譯器現在對僅包含格式程式碼 `%s`、`%r` 和 `%a` 的字串文本 (string literal) 進行簡單的 printf 風格% 格式化 (printf-style % formatting) 最佳化🄵使其與相應的 f-string 運算式一樣快。（由 Serhiy Storchaka 在 bpo-28307 中貢獻。）
- 整數除法 (`//`) 🄵了編譯器最佳化而被調校過。現在將 `int` 除以小於 `2**30` 的值時，在 x86-64 上快了大約 20%。（由 Gregory P. Smith 和 Tim Peters 在 gh-90564 中貢獻。）
- 針對小於 `2**30` 的整數，`sum()` 現在快了將近 30%。（由 Stefan Behnel 於 gh-68264 中所貢獻。）
- 調整 list 大小在常見情🄴下增進了效能，🄵 `list.append()` 加快了約 15% 🄵🄵簡單的 list comprehension 加快了高達 20-30%（由 Dennis Sweeney 在 gh-91165 中貢獻。）
- 當所有鍵都是 Unicode 物件時，字典不存儲雜🄵值，🄵少了 `dict` 的大小。例如，`sys.getsizeof(dict.fromkeys("abcdefg"))` 在 64-bit 平台上從 352 位元組🄴少到 272 位元組（🄵少 23%）。（由 Inada Naoki 在 bpo-46845 中貢獻。）
- 使用 `asyncio.DatagramProtocol` 以透過 UDP 傳輸大文件時，現在速度提高了幾個數量級，傳輸 ≈60 MiB 檔案的速度提高了 100 多倍。（由 msoxzw 在 gh-91487 中貢獻。）
- `math` 函式 `comb()` 和 `perm()` 針對較大引數現在快了 ≈10 倍（對於更大的 *k* 有更大的加速）。（由 Serhiy Storchaka 在 bpo-37295 中貢獻。）
- `statistics` 函式 `mean()`、`variance()` 和 `stdev()` 現在會一次性的消耗🄴代器，而不是先將它們轉🄵🄵 `list`，這讓速度提升🄵兩倍🄴可以節省大量記憶體空間。（由 Raymond Hettinger 在 gh-90415 中貢獻。）
- 現在 `unicodedata.normalize()` 将在固定时间内正规化纯 ASCII 字符串。（由 Donghee Na 在 bpo-44987 中贡献。）

9 更快的 CPython

當使用基准量測套裝軟體 pyperformance 量測🄵以 GCC 於 Ubuntu Linux 上編譯，Python 3.11 平均比 Python 3.10 快了 25%。根據程式工作量可能有所不同，整體加速程度可達 10-60%。

此計畫專注在 Python 的更快的🄵動 和更快的運行程式。不在此專案🄵的最佳化被獨立列出在最佳化。

9.1 更快的🄵動

凍結引入 (Frozen imports) / 🄵態程式碼物件 (Static code objects)

Python 將位元組碼於 __pycache__ 目🄵中存🄵快取來加速模組的載入。

在先前的 3.10 中，執行 Python 模組會像是這樣：

```
Read __pycache__ -> Unmarshal -> Heap allocated code object -> Evaluate
```

在 Python 3.11 中，核心模組在 Python 🄵動時必須被「凍結」，這意味著它們的程式碼物件（和位元組碼）是由直譯器🄵態分配的。這將模組執行過程中的步驟🄴少🄵：

```
Statically allocated code object -> Evaluate
```

在 Python 3.11 中直譯器🄵動速度快了 10-15%。這對於使用 Python 所撰寫的短暫程式有著巨大影響。

（由 Eric Snow、Guido van Rossum 與 Kumar Aditya 於多個 issue 中貢獻。）

9.2 更快的運行程式

所需資源更少 (cheaper) 且惰性的 (lazy)) Python 幀 (frame)

每當 Python 呼叫 Python 函式時，就會建立保存執行資訊的 Python 幀。以下是針對幀而做的新最佳化：

- 使幀的建立過程更有效率。
- 在 C 堆🄵 (stack) 中盡量重🄵利用幀的空間來避免記憶體分配。
- 讓🄵部幀結構只包含必要資訊，使其更加精簡。在過去，幀必須帶有額外的偵錯與記憶體管理的資訊。

舊式幀物件現在僅在除錯器或 Python 自我檢查函式（例如 `sys._getframe()` 或 `inspect.currentframe()`）請求時才建立。對於大多數使用者程式碼，根本不會建立任何幀物件。結果幾乎所有 Python 函式呼叫都顯著加速。我們以 pyperformance 測得了 3-7% 的加速。

（由 Mark Shannon 於 bpo-44590 中所貢獻。）

行🄵 Python 函式呼叫

在 Python 函式呼叫期間，Python 將呼叫一個正在求值的 C 函式來直譯該函式的程式碼，這有效地將純 Python 遞🄵限制在對 C 堆🄵的安全範圍🄵。

在 3.11 中，當 CPython 檢測到 Python 程式碼呼叫另一個 Python 函式時，它會設立一個新框架 (frame)，🄵「跳轉」到新框架🄵的新程式碼，這避免了呼叫整個 C 直譯函式。

現在大多數 Python 函式的呼叫因🄵不會🄵用 C 堆🄵空間而被加速。在斐波那契 (fibonacci) 或階乘等簡單遞🄵函式中，觀察到 1.7 倍的加速。這也意味著遞🄵函式可以遞🄵得更深（如果使用者有增加遞🄵限制）。我們在 pyperformance 測得 1-3% 的改進。

（由 Pablo Galindo 與 Mark Shannon 於 bpo-45256 中所貢獻。）

PEP 659：特化的適應性直譯器

PEP 659 是加速 CPython 專案的關鍵部分之一。一般的想法是，雖然 Python 是一種動態語言，但大多數程式碼都有物件和型🄵很少去更改的區域。這個概念被稱🄵型🄵穩定 *(type stability)*。

在執行環境，Python 將嘗試在執行中的程式碼🄵尋找常用模式和型🄵穩定，然後 Python 將用更特化的操作替🄵當前操作。這種特化操作運用了僅適用於那些用例/型🄵的快速路徑，這通常優於它們的泛用對應 (generic counterparts)。這也引入了另一個稱🄵行🄵快取 *(inline caching)* 的概念，其中 Python 將繁重操作的結果直接快取在位元組碼中。

特化程式 (specializer) 還將某些常用指示 (common instruction) 組合成一個超級指示 (superinstruction)，這🄵少了執行期間的開銷。

Python 只會在看到「熱」（被多次執行的）程式碼時特化，這可以防止 Python 將時間浪費在只運行一次的程式碼上。當程式碼過於動態或用途發生變化時，Python 也可以去特化 (de-specialize)。特化會定期被嘗試執行，而嘗試的成本也不會太高，這讓特化得以適應新的環境。

（PEP 由 Mark Shannon 撰寫、概念🄵發自 Stefan Brunthaler。詳情請見 **PEP 659**。由 Mark Shannon 和 Brandt Bucher 實作，Irit Katriel 和 Dennis Sweeney 亦提供了額外的幫助。）

操作	形式	特化	操作加速程度（上限）	貢獻者

二元操作	`x + x`
	`x - x`
	`x * x`
		常見型🄵如 `int`、`float` 與 `str` 的二元加法、乘法與🄵法，🄵底層型🄵🄵取了特🄵的快速路徑。	10%	Mark Shannon, Donghee Na, Brandt Bucher, Dennis Sweeney

下標	`a[i]`	下標容器型🄵如 `list`、`tuple` 和 `dict` 直接索引底層的資料結構。 下標自定義 `__getitem__()` 也是行🄵的，類似於行🄵 *Python 函式呼叫*。	10-25%	Irit Katriel, Mark Shannon
儲存下標	`a[i] = z`	類似於上面的下標特化。	10-25%	Dennis Sweeney
呼叫	`f(arg)` `C(arg)`	常見🄵建 (C) 函式和型🄵的呼叫，例如 `len()` 和 `str`，會直接呼叫它們的 C 版本底層，這避免了通過🄵部呼叫的慣例。	20%	Mark Shannon, Ken Jin
載入全域變數	`print` `len`	全域/🄵建之命名空間🄵的物件索引被快取起來。載入全域與🄵建變數不需要任何命名空間的查找。	[1]	Mark Shannon
載入屬性	`o.attr`	和載入全域變數類似，類🄵/物件之命名空間🄵的屬性索引被快取起來。在大部分情🄵中，載入屬性不需要任何命名空間的查找。	[2]	Mark Shannon
載入要呼叫的方法	`o.meth()`	方法的真實記憶體地址被快取 (cache) 起來，方法的載入現在不需要命名空間的查找 -- 即便有很長繼承鏈結的類🄵也是。	10-20%	Ken Jin, Mark Shannon
儲存屬性	`o.attr = z`	和載入屬性的最佳化相似。	2% 於 pyperformance 中	Mark Shannon
拆解 (unpack) 序列	`*seq`	🄵像是 `list` 和 `tuple` 的常見容器所特化，避免了🄵部呼叫慣例。	8%	Brandt Bucher

9.3 雜項

- 物件現在因🄵使用了惰性建立的物件命名空間所以需要更少的記憶體。它們的命名空間字典現在也更自由地共享鍵。（由 Mark Shannon 於 bpo-45340 和 bpo-40116 貢獻。）
- 實作了「無代價 (Zero-cost)」的例外，消除了在🄵有例外被引發時的 `try` 陳述式開銷。（由 Mark Shannon 於 bpo-40222 貢獻。）
- 在直譯器🄵使用更簡潔的例外表示法將捕獲一個例外所需的時間🄵少了大約 10%。由 Irit Katriel 在 bpo-45711 中貢獻。）
- `re` 的正則表達式比對引擎部分被重構，且現在會有支援的平台上使用 computed gotos（或者「執行緒程式碼 (threaded code)」），因此 Python 3.11 在執行 pyperformance 正則表達式基准量測的表現上比起 Python 3.10 快了 10%。（由 Brandt Bucher 於 gh-91404 中貢獻。）

[1] 類似的最佳化自從 Python 3.8 就存在。3.11 特🄵處理了更多形式🄵🄵少效能開銷 (overhead)。
[2] 類似的最佳化自從 Python 3.10 就存在。3.11 特🄵處理了更多形式。此外，所有屬性載入也被 bpo-45947 所加速。

9.4 FAQ

我該如何在程式碼中獲取這些加速？

撰寫符合 Python 風格 (Pythonic) 且依循常見最佳實踐的程式碼就好，你不需要改變你的程式碼。CPython 加速計畫中，我們🄵所觀察到的常見程式編寫模式來做最佳化。

Python 3.11 會不會使用更多記憶體？

也許不會。我們預期不會有超出 3.10 20% 的記憶體使用量。這數字會和上述🄵物件與物件字典的記憶體最佳化而有所偏差。

我在我的程式當中🄵感覺到任何加速，🄵什🄵？

某些程式中不會有顯著的好處。如果你的程式花了大部分的時間在 I/O 操作上，或已經將大部分計算用像是 numpy 的 C 擴充函式庫處理，那就不會有明顯的加速。這個計畫是對純 Python 的工作負荷最有幫助。

此外，pyperformance 數值🄵一個幾何平均數 (geometric mean)。即便在 pyperformance 基准量測中，某些測試稍微慢了一些，但其他加快了將近兩倍！

有用到 JIT 編譯器嗎？

🄵有，我們還在探索其他最佳化方式。

9.5 關於

CPython 加速計畫探索了各種 CPython 最佳化的可能性。主要團隊由微軟 (microsoft) 所資助以全職發展該計畫，Pablo Galindo Salgado 亦由彭博有限合夥企業 (Bloomberg LP) 資助來兼職開發，更有許許多多來自社群的自發性貢獻者。

10 CPython 位元組碼 (bytecode) 變更

位元組碼現在包含行🄵快取條目，它們🄵用新添加的 `CACHE` 指示的形式。許多操作碼預期後面要有確切數量的快取，🄵指示直譯器在執行環境跳過它們。傳遞的 (populated) 快取看起來像任意指示，因此在讀取或修改包含加速資料的原始且適應 (adaptive) 位元組碼時應格外小心。

10.1 新增 opcode

- `ASYNC_GEN_WRAP`、`RETURN_GENERATOR` 和 `SEND` 被用於🄵生器與協程。
- `COPY_FREE_VARS`，避免了🄵閉包 (closure) 而生的特殊呼叫方 (caller-side) 程式碼的需求。
- `JUMP_BACKWARD_NO_INTERRUPT`，用於某些不需要處理中斷的循環。
- `MAKE_CELL` 被用於建立 cell-objects。
- `CHECK_EG_MATCH` 和 `PREP_RERAISE_STAR`，處理 **PEP 654** 所加入的*新增例外群組和 except**。
- `PUSH_EXC_INFO` 被用於例外處理函式。
- `RESUME`，🄵無操作 (no-po)，用於🄵部追查、除錯和最佳化檢查。

10.2 被取代的操作碼 (opcode)

被取代的操作碼	新的操作碼	🄵記
BINARY_* INPLACE_*	BINARY_OP	以單一一個操作碼來取代所有數值的、二進位/原位 (in-place) 操作碼
CALL_FUNCTION CALL_FUNCTION_KW CALL_METHOD	CALL KW_NAMES PRECALL PUSH_NULL	將方法的引數搬移 (argument shifting) 與關鍵字引數的處理分離開來；允許更好的呼叫特化
DUP_TOP DUP_TOP_TWO ROT_TWO ROT_THREE ROT_FOUR ROT_N	COPY SWAP	堆🄵操作指示
JUMP_IF_NOT_EXC_MATCH	CHECK_EXC_MATCH	現在執行檢查但不跳位 (jump)
JUMP_ABSOLUTE POP_JUMP_IF_FALSE POP_JUMP_IF_TRUE	JUMP_BACKWARD POP_JUMP_BACKWARD_IF_* POP_JUMP_FORWARD_IF_*	參見[3]；每個方向的 TRUE、FALSE、NONE 和 NOT_NONE 變體
SETUP_WITH SETUP_ASYNC_WITH	BEFORE_WITH	with 區塊設置

10.3 有更動/被移除的 opcode

- 更改了 MATCH_CLASS 和 MATCH_KEYS，不再推送一個額外的布林值來表示成功/失敗。取而代之的是會在失敗時推送 None，而非一個包含提取值的元組。
- 更改了運作於例外的操作碼以反映它們現在在堆🄵中的表示🄵一項而不是三項（請參🄵 gh-89874）。
- 🄵除 COPY_DICT_WITHOUT_KEYS、GEN_START、POP_BLOCK、SETUP_FINALLY 和 YIELD_FROM。

[3] 所有跳位操作碼 (jump opcode) 現在都是相對的，包括現有的 JUMP_IF_TRUE_OR_POP 和 JUMP_IF_FALSE_OR_POP。該引數現在是當前指示 (instruction) 的偏移量而不是🄵對位置。

11 已🄵用

這個部分列出了在 Python 3.11 中🄵用的 Python API。

被🄵用的 C API 被獨立列出。

11.1 語言/🄵建

- 鏈接 `classmethod` 描述器（在 bpo-19072 中引入）現已🄵用。它不能再用於包裝其他描述器，例如 `property`。此功能的核心設計存在缺陷，🄵導致了許多下游問題。要「傳遞通過 (pass-through)」`classmethod`，請考慮使用 Python 3.10 中添加的 `__wrapped__` 屬性。（由 Raymond Hettinger 在 gh-89519 中貢獻。）
- 值大於 `0o377`（十進位🄴 255）的字串和位元組文本值 (bytes literal) 中的八進位跳🄴 (octal escape) 現在會🄵生 `DeprecationWarning`。在未來的 Python 版本中，他們將引發一個 `SyntaxWarning` 🄵最終引發一個 `SyntaxError`。（由 Serhiy Storchaka 在 gh-81548 中貢獻。）
- `int()` 到 `__trunc__()` 的授權 (delegation) 現已🄵用。當 `type(a)` 有實作 `__trunc__()` 但🄵有 `__int__()` 或 `__index__()`，呼叫 `int(a)` 現在會引發一個 `DeprecationWarning`。（由 Zackery Spytz 在 bpo-44977 中貢獻。）

11.2 模組

- **PEP 594** 引領下列模組的🄵用，🄴排訂於 Python 3.13 移除：

`aifc`	`chunk`	`msilib`	`pipes`	`telnetlib`
`audioop`	`crypt`	`nis`	`sndhdr`	`uu`
`cgi`	`imghdr`	`nntplib`	`spwd`	`xdrlib`
`cgitb`	`mailcap`	`ossaudiodev`	`sunau`	

（由 Brett Cannon 和 Victor Stinner 分🄵於 bpo-47061 與 gh-68966 中所貢獻。）

- `asynchat`、`asyncore` 和 `smtpd` 至少在 Python 3.6 以前就被🄵用，它們的文件與🄵用警告現在已被更新🄵會提示它們即將於 Python 3.12 中移除。（由 Hugo van Kemenade 於 bpo-47022 中貢獻。）
- `lib2to3` 套件和 2to3 工具現已🄵用，可能無法剖析 Python 3.10 或更新版本。有關詳細資訊請參🄵 **PEP 617**，它引入了新的 PEG 剖析器。（由 Victor Stinner 在 bpo-40360 中貢獻。）
- 未被記🄵於文件中的 `sre_compile`、`sre_constants` 和 `sre_parse` 模組現在已被🄵用。（由 Serhiy Storchaka 在 bpo-47152 中貢獻。）

11.3 標準函式庫

- 以下 `configparser` 相關項目已在 Python 3.2 中🄵用，它們的🄵用警告現在會提示它們即將於 Python 3.12 中移除：
  - `configparser.SafeConfigParser` 類🄵
  - `configparser.ParsingError.filename` 屬性
  - `configparser.RawConfigParser.readfp()` 方法

  （由 Hugo van Kemenade 於 bpo-45173 中所貢獻。）

- configparser.LegacyInterpolation 自 Python 3.2 起已在文件字串中🄵用，🄵且未在 configparser 文檔中列出。它現在會發出一個 DeprecationWarning 🄵將在 Python 3.13 中🄵除。請改用 configparser.BasicInterpolation 或 configparser.ExtendedInterpolation。(由 Hugo van Kemenade 在 bpo-46607 中貢獻。)
- 舊的 importlib.resources 函式集因🄵不支援定位套件子目🄵中的資源而被🄵用、🄵將在未來的 Python 版本中🄵除，取而代之的是在 Python 3.9 中添加的替代方案：
  - importlib.resources.contents()
  - importlib.resources.is_resource()
  - importlib.resources.open_binary()
  - importlib.resources.open_text()
  - importlib.resources.read_binary()
  - importlib.resources.read_text()
  - importlib.resources.path()
- locale.getdefaultlocale() 函数已被弃用并将在 Python 3.15 中移除。请改用 locale.setlocale(),locale.getpreferredencoding(False) 和 locale.getlocale() 函数。(由 Victor Stinner 在 gh-90817 中贡献。)
- locale.resetlocale() 函式已🄵用🄵將於 Python 3.13 中移除，請改用 locale.setlocale(locale.LC_ALL, "")。(由 Victor Stinner 於 gh-90817 中所貢獻。)
- 現在將對規則運算式中的數值群組參照 (numerical group references) 和群組名稱套用更嚴格的規則。現在只接受 ASCII 數字序列作🄴數值參照，🄴且 bytes 模式 (pattern) 的群組名稱和替🄴字串中只能包含 ASCII 字母、數字和底🄵。目前，會針對違反這些規則的語法發出🄵用警告。(由 Serhiy Storchaka 在 gh-91760 中貢獻。)
- 在 re 模块中，re.template() 函数和相应的 re.TEMPLATE 和 re.T 旗标已被弃用，因为它们未被写入文档并缺少明显的目的。它们将在 Python 3.13 中移除。(由 Serhiy Storchaka 和 Miro Hrončok 在 gh-92728 由贡献。)
- turtle.settiltangle() 自 Python 3.1 以來已被🄵用；它現在會發出🄵用警告，🄵將在 Python 3.13 中🄵除。請改用 turtle.tiltangle()(它之前被錯誤地標記🄵已🄵用，其文件字串現在已更正)。(由 Hugo van Kemenade 在 bpo-45837 中貢獻。)
- 僅用於支援 Python 2 和 Python 3 程式碼間相容性的 typing.Text 現已🄵用。目前未計劃🄵除它，但鼓勵用🄵盡可能使用 str 代替。(由 Alex Waygood 在 gh-92332 中貢獻。)
- 用於建構 typing.TypedDict 型🄵的關鍵字引數語法現已🄵用。將在 Python 3.13 中停止支援。(由 Jingchen Ye 在 gh-90224 中貢獻。)
- webbrowser.MacOSX 已被弃用并将在 Python 3.13 中移除。它未经测试，未写入文档，也未被 webbrowser 本身所使用。(由 Donghee Na 在 bpo-42255 中贡献。)
- 回傳從 TestCase 和 IsolatedAsyncioTestCase 測試方法（預設的 None 值除外）給定值的行🄵現已🄵用。
- 🄵用以下🄵🄵有正式紀🄵於文件中的 unittest 函式，🄵預計於 Python 3.13 中移除：
  - unittest.findTestCases()
  - unittest.makeSuite()
  - unittest.getTestCaseNames()

  改用 TestLoader 方法：
  - unittest.TestLoader.loadTestsFromModule()

- unittest.TestLoader.loadTestsFromTestCase()
- unittest.TestLoader.getTestCaseNames()

（由 Erlend E. Aasland 於 bpo-5846 中所貢獻。）

- unittest.TestProgram.usageExit() 被标记为已弃用，将在 3.13 中被移除。（由 Carlos Damázio 在 gh-67048 中贡献。）

12 Python 3.12 中待🄵議的移除項目

下列 API 已在先前的 Python 發布版本中🄴用，🄵將於 Python 3.12 中移除。

待定的 C API 移除項目🄵獨立列出的。

- asynchat 模組
- asyncore 模組
- 整個 distutils 套件
- imp 模組
- typing.io 命名空間
- typing.re 命名空間
- cgi.log()
- importlib.find_loader()
- importlib.abc.Loader.module_repr()
- importlib.abc.MetaPathFinder.find_module()
- importlib.abc.PathEntryFinder.find_loader()
- importlib.abc.PathEntryFinder.find_module()
- importlib.machinery.BuiltinImporter.find_module()
- importlib.machinery.BuiltinLoader.module_repr()
- importlib.machinery.FileFinder.find_loader()
- importlib.machinery.FileFinder.find_module()
- importlib.machinery.FrozenImporter.find_module()
- importlib.machinery.FrozenLoader.module_repr()
- importlib.machinery.PathFinder.find_module()
- importlib.machinery.WindowsRegistryFinder.find_module()
- importlib.util.module_for_loader()
- importlib.util.set_loader_wrapper()
- importlib.util.set_package_wrapper()
- pkgutil.ImpImporter
- pkgutil.ImpLoader
- pathlib.Path.link_to()

- `sqlite3.enable_shared_cache()`
- `sqlite3.OptimizedUnicode()`
- `PYTHONTHREADDEBUG` 環境變數
- `unittest` 中被🄵用的🄵名：

已🄵用的🄵名	方法名懲	🄵用於
`failUnless`	`assertTrue()`	3.1
`failIf`	`assertFalse()`	3.1
`failUnlessEqual`	`assertEqual()`	3.1
`failIfEqual`	`assertNotEqual()`	3.1
`failUnlessAlmostEqual`	`assertAlmostEqual()`	3.1
`failIfAlmostEqual`	`assertNotAlmostEqual()`	3.1
`failUnlessRaises`	`assertRaises()`	3.1
`assert_`	`assertTrue()`	3.2
`assertEquals`	`assertEqual()`	3.2
`assertNotEquals`	`assertNotEqual()`	3.2
`assertAlmostEquals`	`assertAlmostEqual()`	3.2
`assertNotAlmostEquals`	`assertNotAlmostEqual()`	3.2
`assertRegexpMatches`	`assertRegex()`	3.2
`assertRaisesRegexp`	`assertRaisesRegex()`	3.2
`assertNotRegexpMatches`	`assertNotRegex()`	3.5

13 已移除

此部分列出 Python 3.11 中移除的 Python API。

被移除的 C API 被獨立列出。

- 🄵除了 `@asyncio.coroutine()` decorator 使遺留的基於生成器的協程 (generator-based coroutine) 與 `async` / `await` 程式碼相容。該函式自 Python 3.8 起已被🄵用，計劃於 Python 3.10 🄵除。請改用 `async def`。（由 Illia Volochii 在 bpo-43216 中貢獻。）
- 移除除錯模式中用於包裝遺留基於🄵生器之協程物件的 `asyncio.coroutines.CoroWrapper`。（由 Illia Volochii 於 bpo-43216 中貢獻。）
- 因🄵有重大的安全性考量，Python 3.9 中停用的 `asyncio.loop.create_datagram_endpoint()` 之 *reuse_address* 參數目前已經移除。這是因🄵 UDP socket 選項 `SO_REUSEADDR` 的行🄵所致。（由 Hugo van Kemenade 於 bpo-45129 中貢獻。）
- 移除 Python 3.9 中🄵用的 `binhex` 模組，與其相關且相似的 `binascii` 函式也一🄵被移除：
  - `binascii.a2b_hqx()`
  - `binascii.b2a_hqx()`
  - `binascii.rlecode_hqx()`
  - `binascii.rldecode_hqx()`

  `binascii.crc_hqx()` 維持可用。

  （由 Victor Stinner 於 bpo-45085 中所貢獻。）
- 移除 Python 3.9 中🄵用的 `distutils bdist_msi` 命令。請改用 `bdist_wheel`（wheel 套件）。（由 Hugo van Kemenade 於 bpo-45124 中貢獻。）

- 將 `xml.dom.pulldom.DOMEventStream`、`wsgiref.util.FileWrapper` 和 `fileinput.FileInput` 自 Python 3.9 中🄵用的 `__getitem__()` 方法移除。(由 Hugo van Kemenade 在 bpo-45132 中貢獻。)
- 移除了已弃用的 `gettext` 函数 `lgettext()`, `ldgettext()`, `lngettext()` 和 `ldngettext()`。并移除了 `bind_textdomain_codeset()` 函数、`NullTranslations.output_charset()` 和 `NullTranslations.set_output_charset()` 方法，以及 `translation()` 和 `install()` 的 *codeset* 形参，因为它们仅被用于 `l*gettext()` 函数。(由 Donghee Na 和 Serhiy Storchaka 在 bpo-44235 中贡献。)
- 於 `inspect` 模組中移除：
  - Python 3.0 中🄵用的 `getargspec()`；改用 `inspect.signature()` 或 `inspect.getfullargspec()`。
  - Python 3.5 中🄵用的 `formatargspec()` 函式；請直接用 `inspect.signature()` 函式或 `inspect.Signature` 物件。
  - Python 3.5 中停用且🄵有被紀🄵於文件上的 `Signature.from_builtin()` 和 `Signature.from_function()` 方法；改用 `Signature.from_callable()` 方法。

  (由 Hugo van Kemenade 於 bpo-45320 中所貢獻。)
- 自 `pathlib.PurePath` 中移除 `__class_getitem__()` 方法，因🄵它是前一版本中誤加且🄵被使用。(由 Nikita Sobolev 於 bpo-46483 中所貢獻。)
- 移除了 `smtpd` 模块中的 `MailmanProxy` 类，因为它在没有外部 `mailman` 包时是无法使用的。(由 Donghee Na 在 bpo-35800 中贡献。)。
- 移除 `_tkinter.TkappType` 已被🄵用的 `split()` 方法。(由 Erlend E. Aasland 於 bpo-38371 貢獻。)
- 從 `unittest` 中🄵除了命名空間套件支援。它在 Python 3.4 中引入，但自 Python 3.7 以來已無法運作。(由 Inada Naoki 在 bpo-23882 中貢獻。)
- 將未被記🄵於文件中的私有方法 `float.__set_format__()` 移除，它過去是 Python 3.7 中的 `float.__setformat__()`。它的文件字串 (docstring) 🄵到：「你大概不會想要使用這個函式，它只🄵了讓 Python 測試系列套件 (suite) 使用而存在。」(由 Victor Stinner 於 bpo-46852 中貢獻。)
- 移除 `--experimental-isolated-subinterpreters` 配置旗標 (與對應的 `EXPERIMENTAL_ISOLATED_SUBINTERPRETERS` 巨集)。
- Pynche --- Python 風格的自然色彩與色調編輯器 --- 已被移出 `Tools/scripts`，獨立開發於 Python 原始碼之外。

14 移植至 Python 3.11

本部分列出了之前描述的 Python API 中可能需要你去更改 Python 程式碼的變更和其他錯誤修復。

C API 的移植被獨立列出。

- `open()`、`io.open()`、`codecs.open()` 和 `fileinput.FileInput` 不再接受 `'U'`（"universal newline"，通用🄵行符）文件模式。在 Python 3 中，每當以文本模式 (text mode) 打開檔案時，預設情🄵下會使用「通用🄵行符」模式，🄵且自 Python 3.3 以來就不推薦使用 `'U'` 旗標。這些函式的 newline 參數控制了通用🄵行符的作用方式。(由 Victor Stinner 在 bpo-37330 中貢獻。)
- `ast.AST` 節點位置現在會在提供給 `compile()` 和其他相關函式時被驗證。如果檢測到無效位置，則會引發 `ValueError`。(由 Pablo Galindo 在 gh-93351 中貢獻)
- 在 Python 3.8 中🄵用後，禁止將非 `concurrent.futures.ThreadPoolExecutor` 執行器傳遞給 `asyncio.loop.set_default_executor()`。(由 Illia Volochii 在 bpo-43234 中貢獻。)

- `calendar`：`calendar.LocaleTextCalendar` 和 `calendar.LocaleHTMLCalendar` 類[F]如果[F]有指定語言環境，現在會使用 `locale.getlocale()` 而非 `locale.getdefaultlocale()`。（由 Victor Stinner 在 bpo-46659 中貢獻。）
- `pdb` 模組現在會讀取 `'UTF-8'` 編碼的 `.pdbrc` 配置檔案。（Srinivas Reddy Thatiparthy (□□□□□□□□□□ □□□□□ □□□□□□□□□□) 於 bpo-41137 貢獻。）
- `random.sample()` 的 *population* 參數必須是一個序列，不再支援 `set` 到 `list` 的自動轉[F]。此外，如果抽樣大小大於總體大小，則會引發 `ValueError`。（由 Raymond Hettinger 在 bpo-40465 中貢獻。）
- [F]除了 `random.shuffle()` 的 *random* 可選參數。它以前是用於隨機排列 (shuffle) 的任意隨機函式；現在都會使用 `random.random()`（這是它以前的預設值）。
- 在 `re` re-syntax 中，全域行[F]旗標（例如 `(?i)`）現在只能在規則運算式的開頭使用。自 Python 3.6 以來，在其他地方使用它們已被[F]用。（由 Serhiy Storchaka 在 bpo-47066 中貢獻。）
- 在 `re` 模組中修復了幾個長期存在的錯誤，在極少數情[F]下，這些錯誤可能會導致捕獲群組 (capture group) 得到錯誤的結果。因此，這可能會在這些情[F]下更改捕獲的輸出。（Ma Lin 在 bpo-35859 中貢獻。）

15 建置變更

- CPython 現在有 **PEP 11 Tier 3 支援** 以用於交叉編譯至 WebAssembly 平台 Emscripten (`wasm32-unknown-emscripten`，即[F]覽器中的 Python) 和 WebAssembly 系統介面 (WASI) (`wasm32-unknown-wasi`)。這個靈感來自過往的貢獻，例如 Pyodide。這些平台提供了有限的 POSIX API 子集；Python 標准函式庫功能和與網路、行程、執行緒、訊號、mmap、用[F]/群組相關的模組不開放使用或無法正常使用。（Emscripten 由 Christian Heimes 和 Ethan Smith 在 gh-84461 貢獻，WASI 由 Christian Heimes 在 gh-90473 貢獻；平台在 gh-95085 中推廣）
- 建置 CPython 現在必須要有：
  - C11 編譯器與標准函式庫。可選的 C11 特性 [F]非必要。（由 Victor Stinner 於 bpo-46656、bpo-45440 和 bpo-46640 中貢獻。）
  - 對 IEEE 754 浮點數的支援（由 Victor Stinner 於 bpo-46917 中所貢獻。）
- `Py_NO_NAN` 巨集已被移除。因[F] CPython 現在需要 IEEE 754 浮點數，NaN 數值皆[F]可得的。（由 Victor Stinner 在 bpo-46656 中貢獻。）
- `tkinter` 套件現在必須要有 Tcl/Tk 8.5.12 或更新的版本。（由 Serhiy Storchaka 於 bpo-46996 中所貢獻。）
- 大多數 stdlib 擴充模組的依賴套件、編譯器旗標 (compiler flag) 和鏈接器旗標 (linker flags) 現在可以透過 **`configure`** 檢測出來。（當可用時）pkg-config 會檢測出 libffi、libnsl、libsqlite3、zlib、bzip2、liblzma、libcrypt、Tcl/Tk 和 uuid 旗標。`tkinter` 現在需要一個 pkg-config 命令來檢測 Tcl/Tk 標頭檔和函式庫的開發設定。（由 Christian Heimes 和 Erlend Egeberg Aasland 在 bpo-45847、bpo-45747 和 bpo-45763 中貢獻。）
- libpython 不再鏈接到 libcrypt。（由 Mike Gilbert 在 bpo-45433 中貢獻。）
- 现在 CPython 可以通过向 `--with-lto` 传入 `thin`，即 `--with-lto=thin` 在编译时启用 ThinLTO 选项。（由 Donghee Na 和 Brett Holman 在 bpo-44340 中贡献。）
- 物件結構的空[F]列表現在可被禁用。一個新的 **`configure`** 選項 —`without-freelists` 可用於禁用除空元組單例之外的所有空[F]列表。（由 Christian Heimes 在 bpo-45522 中貢獻。）
- `Modules/Setup` 和 `Modules/makesetup` 已得到改進和綁定。現在可以通過 `makesetup` 建置擴充模組。除了一些測試模組外，所有模組都可以[F]態鏈接到主要的二進制文件或函式庫中。（由 Brett Cannon 和 Christian Heimes 在 bpo-45548、bpo-45570、bpo-45571 和 bpo-43974 中貢獻。）

備🄵: 使用環境變數 TCLTK_CFLAGS 和 TCLTK_LIBS 以手動指定 Tcl/Tk 標頭檔和函式庫的位置。**configure** 選項 —with-tcltk-includes 和 —with-tcltk-libs 已被🄵除。

RHEL 7 和 CentOS 7 上的開發套件🄵無提供 tcl.pc 和 tk.pc；要使用 TCLTK_LIBS="-ltk8.5 -ltkstub8.5 -ltcl8.5"。目🄵 Misc/rhel7 包含 .pc 檔案與如何使用 RHEL 7 和 CentOS 7 的 Tcl/Tk 與 OpenSSL 建置 Python 的指示。

- CPython 現在預設使用 30-bit 數字來實作 Python int。以前，在 SIZEOF_VOID_P >= 8 的平台上預設使用 30-bit 數字，否則使用 15-bit 數字，但仍能通過配置🄵本的 --enable-big-digits 選項或（於 Windows）PC/pyconfig.h 中的 PYLONG_BITS_IN_DIGIT 變數來明確請求使用 15-bit 數字，但此選項可能會在將來的某個時候被🄵除。（由 Mark Dickinson 在 bpo-45569 中貢獻。）

16 C API 變更

16.1 新增特性

- 新增 PyType_GetName() 函式來取得型🄵的短名。（由 Hai Shi 於 bpo-42035 中所貢獻。）
- 新增 PyType_GetQualName() 函式來取得型🄵的合格名稱 (qualified name)。（由 Hai Shi 於 bpo-42035 中所貢獻。）
- 在受限 C API (limited C API) 中新增 PyThreadState_EnterTracing() 和 PyThreadState_LeaveTracing() 函式來中止和繼續追🄵 (tracing) 和性能分析 (profiling)。（由 Victor Stinner 於 bpo-43760 中貢獻。）
- 添加了 Py_Version 常數，其值與 PY_VERSION_HEX 相同。（由 Gabriele N. Tornetta 在 bpo-43931 中貢獻。）
- Py_buffer 與 API 目前是受限 API 與穩定 ABI 中的一部分：
  - PyObject_CheckBuffer()
  - PyObject_GetBuffer()
  - PyBuffer_GetPointer()
  - PyBuffer_SizeFromFormat()
  - PyBuffer_ToContiguous()
  - PyBuffer_FromContiguous()
  - PyObject_CopyData()
  - PyBuffer_IsContiguous()
  - PyBuffer_FillContiguousStrides()
  - PyBuffer_FillInfo()
  - PyBuffer_Release()
  - PyMemoryView_FromBuffer()
  - bf_getbuffer 與 bf_releasebuffer 型🄵插槽 (type slot)

  （由 Christian Heimes 於 bpo-45459 中所貢獻。）
- 增加了 PyType_GetModuleByDef() 函数，用于在无法直接获取信息的情况下 (通过 PyCMethod) 获取方法定义所在的模块。（由 Petr Viktorin 在 bpo-46613 中贡献。）

- 新增函式以打包 (pack) 和取出 (unpack) C double（序列化和反序列化）：`PyFloat_Pack2()`、`PyFloat_Pack4()`、`PyFloat_Pack8()`、`PyFloat_Unpack2()`，`PyFloat_Unpack4()` 和 `PyFloat_Unpack8()`。(由 Victor Stinner 在 bpo-46906 中貢獻。)
- 新增取得幀物件屬性的函式：`PyFrame_GetBuiltins()`、`PyFrame_GetGenerator()`、`PyFrame_GetGlobals()`、`PyFrame_GetLasti()`。
- 添加了兩個新函式來獲得和設置仍在作用的例外實例：`PyErr_GetHandledException()` 和 `PyErr_SetHandledException()`。這些是 `PyErr_SetExcInfo()` 和 `PyErr_GetExcInfo()` 的替代品，它們與例外的遺留三元組表示法一起作用。(由 Irit Katriel 在 bpo-46343 中貢獻。)
- 新增 `PyConfig.safe_path` 成員。(由 Victor Stinner 於 gh-57684 中所貢獻。)

16.2 移植至 Python 3.11

- 一些巨集已轉Ⓕ行Ⓕ態函式以避免巨集陷阱 (macro pitfalls)。這種變化對用Ⓕ來Ⓕ應該是透明的，因Ⓕ替Ⓕ函式會將它們的引數轉ⒻⒻ預期的型Ⓕ，以避免由於Ⓕ態型Ⓕ檢查而Ⓕ生的編譯器警告。但是，當受限 C API 設置Ⓕ >=3.11 時，這些轉Ⓕ不會完成，使用者需要將引數轉ⒻⒻ他們期望的型Ⓕ。有關更多詳細資訊，請參Ⓕ **PEP 670**。(由 Victor Stinner 和 Erlend E. Aasland 在 gh-89653 中貢獻。)
- `PyErr_SetExcInfo()` 不再使用 `type` 和 `traceback` 引數，直譯器現在從例外實例（`value` 引數）中獲得這些值。該函式仍會偷用這三個引數的參照。(由 Irit Katriel 在 bpo-45711 中貢獻。)
- `PyErr_GetExcInfo()` 現在從例外實例的結果（`value` 欄位）中導出 `type` 和 `traceback` 欄位。(由 Irit Katriel 在 bpo-45711 中貢獻。)
- `_frozen` 有一個新的 `is_package` 欄位來表示凍結模組是否Ⓕ一個套件。以前 `size` 欄位中的負值是指標，現在只有非負值可用於 `size`。(由 Kumar Aditya 在 bpo-46608 中貢獻。)
- `_PyFrameEvalFunction()` 現在將 `_PyInterpreterFrame*` 作Ⓕ其第二個參數，而不是 `PyFrameObject*`。有關如何使用此函式指標型Ⓕ的更多詳細資訊，請參Ⓕ **PEP 523**。
- `PyCode_New()` 和 `PyCode_NewWithPosOnlyArgs()` 現在Ⓕ用額外的 `exception_table` 引數。如果可能的話應該避免使用這些函式。要獲取自定義程式碼物件，使用編譯器建立一個程式碼物件，然後使用 `replace` 方法來得到修改後的版本。
- `PyCodeObject` 不再會有 `co_code`、`co_varnames`、`co_cellvars` 和 `co_freevars` 欄位。分Ⓕ被改Ⓕ透過 C API 的 `PyCode_GetCode()`、`PyCode_GetVarnames()`、`PyCode_GetCellvars()` 和 `PyCode_GetFreevars()` 來存取。(由 Brandt Bucher 在 bpo-46841、Ken Jin 在 gh-92154 與 gh-94936 中貢獻。)
- 舊的回收筒巨集 (trashcan macro) (`Py_TRASHCAN_SAFE_BEGIN`/`Py_TRASHCAN_SAFE_END`) 現在已經被Ⓕ用，它們應被新的巨集 `Py_TRASHCAN_BEGIN` 和 `Py_TRASHCAN_END` 所取代。

  一個用到老舊巨集的 tp_dealloc 函式，像是：

```
static void
mytype_dealloc(mytype *p)
{
    PyObject_GC_UnTrack(p);
    Py_TRASHCAN_SAFE_BEGIN(p);
    ...
    Py_TRASHCAN_SAFE_END
}
```

  應該要搬遷到新的巨集，如下所示：

```
static void
mytype_dealloc(mytype *p)
{
    PyObject_GC_UnTrack(p);
    Py_TRASHCAN_BEGIN(p, mytype_dealloc)
    ...
    Py_TRASHCAN_END
}
```

請注意 Py_TRASHCAN_BEGIN 有第二個引數，它應該是它所在的釋放函式 (deallocation function)。

F支援舊版 Python 在同一份程式碼中，你可以定義以下巨集F在程式碼中使用它們（要歸功於 mypy 程式碼，這些是從那邊FF過來的）：

```
#if PY_VERSION_HEX >= 0x03080000
#  define CPy_TRASHCAN_BEGIN(op, dealloc) Py_TRASHCAN_BEGIN(op, dealloc)
#  define CPy_TRASHCAN_END(op) Py_TRASHCAN_END
#else
#  define CPy_TRASHCAN_BEGIN(op, dealloc) Py_TRASHCAN_SAFE_BEGIN(op)
#  define CPy_TRASHCAN_END(op) Py_TRASHCAN_SAFE_END(op)
#endif
```

- 现在如果一个类型定义了 Py_TPFLAGS_HAVE_GC 旗标但没有遍历函数 (PyTypeObject.tp_traverse) 则 PyType_Ready() 函数将引发一个错误。（由 Victor Stinner 在 bpo-44263 中贡献。）
- 带有 Py_TPFLAGS_IMMUTABLETYPE 旗标的堆类型现在可以继承 **PEP 590** vectorcall 协议。在之前版本中，这只适用于 静态类型。（由 Erlend E. Aasland 在 bpo-43908 中贡献。）
- 由於 Py_TYPE() 更改F行FF態函式 (inline static function)，因此 Py_TYPE(obj) = new_type 必須替FF Py_SET_TYPE(obj, new_type)：參見 Py_SET_TYPE() 函式（自 Python 3.9 起可用）。F了向後相容，可以使用這個巨集：

```
#if PY_VERSION_HEX < 0x030900A4 && !defined(Py_SET_TYPE)
static inline void _Py_SET_TYPE(PyObject *ob, PyTypeObject *type)
{ ob->ob_type = type; }
#define Py_SET_TYPE(ob, type) _Py_SET_TYPE((PyObject*)(ob), type)
#endif
```

（由 Victor Stinner 於 bpo-39573 中所貢獻。）

- 由於 Py_SIZE() 更改F行FF態函式，因此 Py_SIZE(obj) = new_size 必須替FF Py_SET_SIZE(obj, new_size)：參見 Py_SET_SIZE() 函式（自 Python 3.9 起可用）。F了向後相容，可以使用這個巨集：

```
#if PY_VERSION_HEX < 0x030900A4 && !defined(Py_SET_SIZE)
static inline void _Py_SET_SIZE(PyVarObject *ob, Py_ssize_t size)
{ ob->ob_size = size; }
#define Py_SET_SIZE(ob, size) _Py_SET_SIZE((PyVarObject*)(ob), size)
#endif
```

（由 Victor Stinner 於 bpo-39573 中所貢獻。）

- 當 Py_LIMITED_API 巨集被設定F 0x030b0000（Python 3.11）或以上，<Python.h> 不再會包含標頭檔 <stdlib.h>、<stdio.h>、<errno.h> 和 <string.h>。C 擴充程式應該要清楚的在 #include <Python.h> 之後引入標頭檔案。（由 Victor Stinner 於 bpo-45434 中貢獻。）
- 非受限 API (non-limited API) 檔案 cellobject.h、classobject.h、code.h、context.h、funcobject.h、genobject. h 和 longintrepr.h 已移至 Include/cpython 目F。此外，

eval.h 標頭檔已被🄵除。不能直接引入這些文件，因🄵它們已被包含在 Python.h 中：引入檔案。如果它們已被直接引入，請考慮改🄵引入 Python.h。（由 Victor Stinner 在 bpo-35134 中貢獻。）

- PyUnicode_CHECK_INTERNED() 巨集已從受限 C API 中移出，它從來🄵辦法被使用，因🄵它使用了受限 C API 不提供的🄵部結構。（由 Victor Stinner 於 bpo-46007 中所貢獻。）
- 以下用於幀 (frame) 的函式與型🄵現在可直接透過 #include <Python.h> 來使用，不必再加上 #include <frameobject.h>：
  - PyFrame_Check()
  - PyFrame_GetBack()
  - PyFrame_GetBuiltins()
  - PyFrame_GetGenerator()
  - PyFrame_GetGlobals()
  - PyFrame_GetLasti()
  - PyFrame_GetLocals()
  - PyFrame_Type

  （由 Victor Stinner 於 gh-93937 中所貢獻。）
- PyFrameObject 結構成員已經從公開的 C API 中移除。

  雖然文件指出 PyFrameObject 欄位隨時可能發生變化，但它們已經穩定了很長時間，🄵被用於幾個流行的擴充套件中。

  Python 3.11 中，幀的結構被重新編制來🄵性能做最佳化，有些作🄵舊版實作細節的欄位被整個移除。

  PyFrameObject 欄位：
  - f_back：使用 PyFrame_GetBack()。
  - f_blockstack：已移除。
  - f_builtins：使用 PyFrame_GetBuiltins()。
  - f_code：使用 PyFrame_GetCode()。
  - f_gen：使用 PyFrame_GetGenerator()。
  - f_globals：使用 PyFrame_GetGlobals()。
  - f_iblock：已移除。
  - f_lasti：使用 PyFrame_GetLasti()。程式碼中 f_lasti 有與 PyCode_Addr2Line() 同時使用的部分應該改用 PyFrame_GetLineNumber()；它可能會更快。
  - f_lineno：使用 PyFrame_GetLineNumber()
  - f_locals：使用 PyFrame_GetLocals()。
  - f_stackdepth：已移除。
  - f_state：無公開 API（重新命名🄵 f_frame.f_state）。
  - f_trace：無公開 API。
  - f_trace_lines：使用 PyObject_GetAttrString((PyObject*)frame, “f_trace_lines”)。
  - f_trace_opcodes：使用 PyObject_GetAttrString((PyObject*)frame, “f_trace_opcodes”)。

  - `f_localsplus`：無公開 API（重新命名🄵 `f_frame.localsplus`）。
  - `f_valuestack`：已移除。

  現在 Python 帧对象是惰性地创建的。一个附带影响是 `f_back` 成员不可被直接访问，因为现在它的值也是惰性地计算的。必须改为调用 `PyFrame_GetBack()` 函数。

  直接访问 `f_locals` 的调试器必须改为调用 `PyFrame_GetLocals()`。它们不再需要调用 `PyFrame_FastToLocalsWithError()` 或 `PyFrame_LocalsToFast()`，实际上它们不应调用这些函数。现在帧所需要的更新将由虚拟机来管理。

  `PyFrame_GetCode()` 在 Python 3.8 以前的程式定義：

```
#if PY_VERSION_HEX < 0x030900B1
static inline PyCodeObject* PyFrame_GetCode(PyFrameObject *frame)
{
    Py_INCREF(frame->f_code);
    return frame->f_code;
}
#endif
```

  `PyFrame_GetBack()` 在 Python 3.8 以前的程式定義：

```
#if PY_VERSION_HEX < 0x030900B1
static inline PyFrameObject* PyFrame_GetBack(PyFrameObject *frame)
{
    Py_XINCREF(frame->f_back);
    return frame->f_back;
}
#endif
```

  或是使用 pythoncap_compat 計畫來在舊版 Python 函式中取得這兩個函式。

- `PyThreadState` 結構成員的改動：
  - `frame`：已移除，改用 `PyThreadState_GetFrame()`（bpo-40429 於 Python 3.9 新增的函式）。警告：會回傳 strong reference 的函式必須呼叫 `Py_XDECREF()`。
  - `tracing`：已變更，改用 `PyThreadState_EnterTracing()` 和 `PyThreadState_LeaveTracing()`（bpo-43760 於 Python 3.11 中新增的函式）。
  - `recursion_depth`：已移除，請改用 `(tstate->recursion_limit - tstate->recursion_remaining)`。
  - `stackcheck_counter`：已移除。

  `PyThreadState_GetFrame()` 在 Python 3.8 以前的程式定義：

```
#if PY_VERSION_HEX < 0x030900B1
static inline PyFrameObject* PyThreadState_GetFrame(PyThreadState *tstate)
{
    Py_XINCREF(tstate->frame);
    return tstate->frame;
}
#endif
```

  `PyThreadState_EnterTracing()` 與 `PyThreadState_LeaveTracing()` 在 Python 3.10 以前的程式定義：

```
#if PY_VERSION_HEX < 0x030B00A2
static inline void PyThreadState_EnterTracing(PyThreadState *tstate)
```

(繼續下一頁)

(繼續上一頁)

```
{
    tstate->tracing++;
#if PY_VERSION_HEX >= 0x030A00A1
    tstate->cframe->use_tracing = 0;
#else
    tstate->use_tracing = 0;
#endif
}

static inline void PyThreadState_LeaveTracing(PyThreadState *tstate)
{
    int use_tracing = (tstate->c_tracefunc != NULL || tstate->c_profilefunc !=␣
↪NULL);
    tstate->tracing--;
#if PY_VERSION_HEX >= 0x030A00A1
    tstate->cframe->use_tracing = use_tracing;
#else
    tstate->use_tracing = use_tracing;
#endif
}
#endif
```

或是使用 pythoncap-compat 計畫來在舊版 Python 函式中取得它們。

- 鼓勵發布者們使用最佳化過的 Blake2 函式庫 libb2 來建置 Python。
- 初始化中若是要用 `PyConfig.module_search_paths` 來初始化 `sys.path`，則現在 `PyConfig.module_search_paths_set` 必須被設🄵 1。否則，初始化會重新計算路徑🄵取代所有被加到 `module_search_paths` 的值。
- `PyConfig_Read()` 不再計算初始搜索路徑，🄵且不會將任何值填充到 `PyConfig.module_search_paths` 中。若要計算預設路徑然後修改它們，完成初始化🄵使用 `PySys_GetObject()` 以取得 `sys.path` 作🄵 Python 列表物件🄵直接修改它。

16.3 已🄵用

- 🄵用以下用來配置 Python 初始化的函式：
  - `PySys_AddWarnOptionUnicode()`
  - `PySys_AddWarnOption()`
  - `PySys_AddXOption()`
  - `PySys_HasWarnOptions()`
  - `PySys_SetArgvEx()`
  - `PySys_SetArgv()`
  - `PySys_SetPath()`
  - `Py_SetPath()`
  - `Py_SetProgramName()`
  - `Py_SetPythonHome()`
  - `Py_SetStandardStreamEncoding()`
  - `_Py_SetProgramFullPath()`

請改用 Python 初始化配置中新的 `PyConfig` API。(由 Victor Stinner 於 gh-88279 中所貢獻。)

- 🄵用 `PyBytesObject` 中的 `ob_shash` 成員。請改用 `PyObject_Hash()`。(由 Inada Naoki 於 bpo-46864 中所貢獻。)

16.4 Python 3.12 中待🄵議的移除項目

以下 C API 已於先前 Python 發布版本中🄵用，🄵將於 Python 3.12 中移除。

- `PyUnicode_AS_DATA()`
- `PyUnicode_AS_UNICODE()`
- `PyUnicode_AsUnicodeAndSize()`
- `PyUnicode_AsUnicode()`
- `PyUnicode_FromUnicode()`
- `PyUnicode_GET_DATA_SIZE()`
- `PyUnicode_GET_SIZE()`
- `PyUnicode_GetSize()`
- `PyUnicode_IS_COMPACT()`
- `PyUnicode_IS_READY()`
- `PyUnicode_READY()`
- `PyUnicode_WSTR_LENGTH()`
- `_PyUnicode_AsUnicode()`
- `PyUnicode_WCHAR_KIND`
- `PyUnicodeObject`
- `PyUnicode_InternImmortal()`

16.5 已移除

- `PyFrame_BlockSetup()` 和 `PyFrame_BlockPop()` 已被移除。(由 Mark Shannon 在 bpo-40222 中貢献。)
- 移除以下使用到 `errno` 變數的數學巨集:
  - `Py_ADJUST_ERANGE1()`
  - `Py_ADJUST_ERANGE2()`
  - `Py_OVERFLOWED()`
  - `Py_SET_ERANGE_IF_OVERFLOW()`
  - `Py_SET_ERRNO_ON_MATH_ERROR()`

  (由 Victor Stinner 於 bpo-45412 中所貢獻。)
- 移除在 Python 3.3 中🄵用的 `Py_UNICODE_COPY()` 和 `Py_UNICODE_FILL()`。請改用 `PyUnicode_CopyCharacters()` 或 `memcpy()` (`wchar_t*` 字串) 和 `PyUnicode_Fill()` 函式。(由 Victor Stinner 於 bpo-41123 中所貢獻。)

- 移除 `pystrhex.h` 標頭檔案。它只有包含私有函式。C 的擴充應該只要引入主要的 `<Python.h>` 標頭檔案。（由 Victor Stinner 於 bpo-45434 中所貢獻。）
- 移除 `Py_FORCE_DOUBLE()` 巨集。它先前被用於 `Py_IS_INFINITY()` 巨集。（由 Victor Stinner 於 bpo-45440 中所貢獻。）
- 當 `Py_LIMITED_API` 有被定義時，以下項目將無法被取得：
  - `PyMarshal_WriteLongToFile()`
  - `PyMarshal_WriteObjectToFile()`
  - `PyMarshal_ReadObjectFromString()`
  - `PyMarshal_WriteObjectToString()`
  - `Py_MARSHAL_VERSION` 巨集

  这些不是 受限 API 的组成部分。

  （由 Victor Stinner 於 bpo-45474 中所貢獻。）
- 將 `PyWeakref_GET_OBJECT()` 排除在受限 C API 之外。由于 `PyWeakReference` 结构体在受限 C API 中被屏蔽因此它从未发挥作用。（由 Victor Stinner 在 bpo-35134 中贡献。）
- 移除 `PyHeapType_GET_MEMBERS()` 巨集，它是不小心才被放到公開的 C API 中，應該只能被 Python 🄵部所使用。請改用 `PyTypeObject.tp_members`。（由 Victor Stinner 於 bpo-40170 中所貢獻。）
- 移除 `HAVE_PY_SET_53BIT_PRECISION` 巨集（移動至🄵部 C API）。（由 Victor Stinner 於 bpo-45412 中所貢獻。）
- 移除 `Py_UNICODE` 編碼器 API，它們自從 Python 3.3 就被🄵用，非常少用且和推薦的替代方案已無太大關聯。

  被移除的函式🄵：
  - `PyUnicode_Encode()`
  - `PyUnicode_EncodeASCII()`
  - `PyUnicode_EncodeLatin1()`
  - `PyUnicode_EncodeUTF7()`
  - `PyUnicode_EncodeUTF8()`
  - `PyUnicode_EncodeUTF16()`
  - `PyUnicode_EncodeUTF32()`
  - `PyUnicode_EncodeUnicodeEscape()`
  - `PyUnicode_EncodeRawUnicodeEscape()`
  - `PyUnicode_EncodeCharmap()`
  - `PyUnicode_TranslateCharmap()`
  - `PyUnicode_EncodeDecimal()`
  - `PyUnicode_TransformDecimalToASCII()`

  詳情請見 **PEP 624** 與**搬遷指南**。（由 Inada Naoki 於 bpo-44029 中所貢獻。）

17 3.11.4 中值得注意的變更

17.1 tarfile

- `tarfile` 和 `shutil.unpack_archive()` 中的提取方法有一個新的 *filter* 引數，它僅允許有限的 tar 功能、停用一些危險的功能，例如在目標目🄵之外建立檔案。詳細資訊請參🄵 tarfile-extraction-filter。在 Python 3.12 中，不帶 *filter* 引數使用將顯示 `DeprecationWarning`。在 Python 3.14 中會將預設切🄵🄵 `'data'`。（由 Petr Viktorin 在 **PEP 706** 中貢獻。）

18 3.11.5 中的重要变化

18.1 OpenSSL

- 来自 python.org 的 Windows 版本和 macOS 安装程序现在使用 OpenSSL 3.0。

19 Notable changes in 3.11.10

19.1 ipaddress

- 修正了 `IPv4Address`，`IPv6Address`，`IPv4Network` 和 `IPv6Network` 中的 `is_global` 和 `is_private` 行为。

19.2 email

- 带有嵌入的换行符的标头现在输出时会加引号。

  现在 `generator` 会拒绝序列化（写入）不正确地折叠或分隔的标头，例如将被解析为多个标头或与相邻数据合并的标头等。如果你需要禁用此安全特性，请设置 `verify_generated_headers`。（由 Bas Bloemsaat 和 Petr Viktorin 在 gh-121650 中贡献。）

- `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.)

索引

非依字母順序

P