What's New in Python

Release 3.12.8

A. M. Kuchling

gennaio 19, 2025

Python Software Foundation
Email: docs@python.org

Indice

Editor
Adam Turner

Questo articolo spiega le nuove funzionalità in Python 3.12, rispetto alla versione 3.11. Python 3.12 è stato rilasciato il 2 ottobre 2023. Per i dettagli completi, consultare il changelog.

Vedi anche

PEP 693 – Programma di rilascio di Python 3.12

1 Sommario – Punti salienti del rilascio

Python 3.12 is a stable release of the Python programming language, with a mix of changes to the language and the standard library. The library changes focus on cleaning up deprecated APIs, usability, and correctness. Of note, the `distutils` package has been removed from the standard library. Filesystem support in `os` and `pathlib` has seen a number of improvements, and several modules have better performance.

Le modifiche al linguaggio si concentrano sull'usabilità, in quanto molte limitazioni dei f-strings sono state rimosse e i suggerimenti "Intendevi dire …" continuano a migliorare. La nuova *sintassi dei parametri di tipo* e l'istruzione `type` migliorano l'ergonomia per l'utilizzo di tipi generici e alias di tipo con i checker di tipo statico.

Questo articolo non tenta di fornire una specifica completa di tutte le nuove funzionalità, ma offre invece una panoramica. Per i dettagli completi, dovresti fare riferimento alla documentazione, come il Library Reference e il Language Reference. Se vuoi comprendere l'implementazione completa e la motivazione dietro una modifica, fai riferimento al PEP per una particolare nuova funzionalità; ma tieni presente che i PEP di solito non vengono aggiornati una volta che una funzionalità è stata completamente implementata.

Nuove funzionalità sintattiche:

- *PEP 695*, sintassi dei parametri di tipo e l'istruzione `type`

Nuove funzionalità grammaticali:

- *PEP 701*, f-strings nella grammatica

Miglioramenti dell'interprete:

- *PEP 684*, un GIL univoco per interprete
- *PEP 669*, monitoraggio a basso impatto
- *Suggerimenti migliorati "Intendevi dire …"* per eccezioni `NameError`, `ImportError`, e `SyntaxError`

Miglioramenti al modello di dati di Python:

- *PEP 688*, usando il protocollo del buffer da Python

Miglioramenti significativi nella libreria standard:

- La classe `pathlib.Path` ora supporta la sottoclassificazione

- Il modulo `os` ha ricevuto diversi miglioramenti per il supporto di Windows
- Un'interfaccia a riga di comando interface è stata aggiunta al modulo `sqlite3`
- I controlli `isinstance()` contro i protocolli `runtime-checkable` godono di un'accelerazione tra due e 20 volte
- Il pacchetto `asyncio` ha avuto numerosi miglioramenti di performance, con alcuni benchmark che mostrano un'accelerazione del 75%.
- Un'interfaccia a riga di comando interface è stata aggiunta al modulo `uuid`
- A causa delle modifiche nel *PEP 701*, la produzione di token tramite il modulo `tokenize` è fino al 64% più veloce.

Miglioramenti alla sicurezza:

- Sostituire le implementazioni integrate del modulo `hashlib` di SHA1, SHA3, SHA2-384, SHA2-512 e MD5 con codice formalmente verificato dal progetto HACL*. Queste implementazioni integrate rimangono come fallback che vengono utilizzate solo quando OpenSSL non le fornisce.

Miglioramenti all'API C:

- *PEP 697*, livello instabile dell'API C
- *PEP 683*, oggetti immortali

Miglioramenti all'implementazione di CPython:

- *PEP 709*, inlining delle comprensioni
- Supporto CPython per il profiler `perf` su Linux
- Implementare la protezione da overflow dello stack su piattaforme supportate

Nuove funzionalità di typing:

- *PEP 692*, utilizzo di `TypedDict` per annotare **kwargs
- *PEP 698*, decoratore `typing.override()`

Deprecazioni, rimozioni o restrizioni importanti:

- **PEP 623**: Rimuovere `wstr` dagli oggetti Unicode nell'API C di Python, riducendo la dimensione di ogni oggetto `str` di almeno 8 byte.
- **PEP 632**: Remove the `distutils` package. See **the migration guide** for advice replacing the APIs it provided. The third-party Setuptools package continues to provide `distutils`, if you still require it in Python 3.12 and beyond.
- gh-95299: Non preinstallare `setuptools` negli ambienti virtuali creati con `venv`. Ciò significa che `distutils`, `setuptools`, `pkg_resources` e `easy_install` non saranno più disponibili di default; per accedervi eseguire `pip install setuptools` nell'ambiente virtuale attivato.
- I moduli `asynchat`, `asyncore` e `imp` sono stati rimossi, insieme a diversi *alias di metodo* di `unittest.TestCase`.

2 Nuove Funzionalità

2.1 PEP 695: Sintassi dei Parametri di Tipo

Le classi generiche e le funzioni sotto **PEP 484** erano dichiarate utilizzando una sintassi verbosa che lasciava poco chiaro l'ambito dei parametri di tipo e richiedeva dichiarazioni esplicite di varianza.

PEP 695 introduce un nuovo modo, più compatto ed esplicito, per creare classi generiche e funzioni:

```
def max[T](args: Iterable[T]) -> T:
    ...

class list[T]:
    def __getitem__(self, index: int, /) -> T:
        ...

    def append(self, element: T) -> None:
        ...
```

Inoltre, il PEP introduce un nuovo modo di dichiarare alias di tipo utilizzando l'istruzione `type`, che crea un'istanza di `TypeAliasType`:

```
type Point = tuple[float, float]
```

Gli alias di tipo possono anche essere generici:

```
type Point[T] = tuple[T, T]
```

La nuova sintassi consente di dichiarare parametri `TypeVarTuple` e `ParamSpec`, nonché parametri `TypeVar` con limiti o vincoli:

```
type IntFunc[**P] = Callable[P, int]  # ParamSpec
type LabeledTuple[*Ts] = tuple[str, *Ts]  # TypeVarTuple
type HashableSequence[T: Hashable] = Sequence[T]  # TypeVar with bound
type IntOrStrSequence[T: (int, str)] = Sequence[T]  # TypeVar with constraints
```

Il valore degli alias di tipo e i limiti e vincoli delle variabili di tipo creati con questa sintassi sono valutati solo su richiesta (vedi lazy evaluation). Ciò significa che gli alias di tipo possono riferirsi ad altri tipi definiti successivamente nel file.

I parametri di tipo dichiarati attraverso una lista di parametri di tipo sono visibili all'interno dell'ambito della dichiarazione e in qualsiasi ambito annidato, ma non nell'ambito esterno. Ad esempio, possono essere utilizzati nelle annotazioni di tipo per i metodi di una classe generica o nel corpo della classe. Tuttavia, non possono essere utilizzati nell'ambito del modulo dopo che la classe è stata definita. Vedi type-params per una descrizione dettagliata della semantica di runtime dei parametri di tipo.

Per supportare questa semantica di ambito, è stato introdotto un nuovo tipo di ambito, l'"annotation scope. Gli ambiti di annotazione si comportano per lo più come gli ambiti delle funzioni, ma interagiscono in modo diverso con gli ambiti della classe che li contengono. In Python 3.13, le annotazioni saranno valutate negli ambiti di annotazione.

Vedi **PEP 695** per ulteriori dettagli.

(PEP scritto da Eric Traut. Implementazione di Jelle Zijlstra, Eric Traut e altri in gh-103764.)

2.2 PEP 701: Formalizzazione sintattica delle f-string

PEP 701 rimuove alcune restrizioni sull'uso delle f-string. Gli elementi dell'espressione all'interno delle f-string possono ora essere qualsiasi espressione Python valida, incluse stringhe che riutilizzano la stessa virgoletta della f-string contenente, espressioni multilinea, commenti, backslash e sequenze di escape unicode. Approfondiamo questi aspetti in dettaglio:

- Riutilizzo delle virgolette: in Python 3.11, riutilizzare le stesse virgolette della f-string contenente genera un'eccezione `SyntaxError`, costringendo l'utente a utilizzare altre virgolette disponibili (come l'uso delle virgolette doppie o delle virgolette triple se la f-string usa virgolette singole). In Python 3.12, è ora possibile fare cose come questa:

```
>>> songs = ['Take me back to Eden', 'Alkaline', 'Ascensionism']
>>> f"This is the playlist: {", ".join(songs)}"
'This is the playlist: Take me back to Eden, Alkaline, Ascensionism'
```

Prima di questa modifica non esisteva un limite esplicito su come le f-string potessero essere annidate, ma il fatto che le virgolette delle stringhe non potessero essere riutilizzate all'interno dell'elemento di espressione delle f-string rendeva impossibile annidare le f-string arbitrariamente. Infatti, questa è l'f-string più annidata che poteva essere scritta:

```
>>> f"""{f'''{f'{f"{1+1}"}'}'''}"""
'2'
```

Poiché ora le f-string possono contenere qualsiasi espressione Python valida all'interno degli elementi di espressione, è ora possibile annidare le f-string arbitrariamente:

```
>>> f"{f"{f"{f"{f"{f"{1+1}"}"}"}"}"}"
'2'
```

- Espressioni multilinea e commenti: in Python 3.11, le espressioni all'interno delle f-string devono essere definite in una singola riga, anche se l'espressione all'interno della f-string potrebbe normalmente estendersi su più righe (come le liste letterali definite su più righe), rendendole più difficili da leggere. In Python 3.12 è ora possibile definire f-string che si estendono su più righe e aggiungere commenti in linea:

```
>>> f"This is the playlist: {", ".join([
...     'Take me back to Eden',  # My, my, those eyes like fire
...     'Alkaline',              # Not acid nor alkaline
...     'Ascensionism'           # Take to the broken skies at last
... ])}"
'This is the playlist: Take me back to Eden, Alkaline, Ascensionism'
```

- Backslash e caratteri unicode: prima di Python 3.12, le espressioni f-string non potevano contenere alcun carattere `\`. Questo influenzava anche le sequenze di escape unicode (come `\N{snowman}`) poiché contengono la parte `\N` che prima non poteva far parte degli elementi di espressione delle f-string. Ora puoi definire espressioni come questa:

```
>>> print(f"This is the playlist: {"\n".join(songs)}")
This is the playlist: Take me back to Eden
Alkaline
Ascensionism
>>> print(f"This is the playlist: {"\N{BLACK HEART SUIT}".join(songs)}")
This is the playlist: Take me back to Eden♥Alkaline♥Ascensionism
```

Vedi **PEP 701** per ulteriori dettagli.

Come effetto collaterale positivo di come questa funzione è stata implementata (parsing delle f-string con **il PEG parser**), ora i messaggi di errore per le f-string sono più precisi e includono l'esatta posizione dell'errore. Ad esempio, in Python 3.11, la seguente f-string genera un'eccezione `SyntaxError`:

```
>>> my_string = f"{x z y}" + f"{1 + 1}"
  File "<stdin>", line 1
    (x z y)
     ^^^
SyntaxError: f-string: invalid syntax. Perhaps you forgot a comma?
```

ma il messaggio di errore non include l'esatta posizione dell'errore all'interno della riga e ha anche l'espressione circondata artificialmente da parentesi. In Python 3.12, poiché le f-string sono parse con il PEG parser, i messaggi di errore possono essere più precisi e mostrare l'intera riga:

```
>>> my_string = f"{x z y}" + f"{1 + 1}"
  File "<stdin>", line 1
    my_string = f"{x z y}" + f"{1 + 1}"
                   ^^^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

(Contributi da Pablo Galindo, Batuhan Taskaya, Lysandros Nikolaou, Cristián Maureira-Fredes e Marta Gómez in gh-102856. PEP scritto da Pablo Galindo, Batuhan Taskaya, Lysandros Nikolaou e Marta Gómez).

2.3 PEP 684: Un GIL per interprete

PEP 684 introduce un GIL per interprete, in modo che gli interpreti secondari possano ora essere creati con un GIL unico per interprete. Questo consente ai programmi Python di sfruttare appieno i core della CPU multipli. Attualmente è disponibile solo tramite la C-API, sebbene si **preveda una API Python per la versione 3.13**.

Usa la nuova funzione `Py_NewInterpreterFromConfig()` per creare un interprete con il proprio GIL:

```
PyInterpreterConfig config = {
    .check_multi_interp_extensions = 1,
    .gil = PyInterpreterConfig_OWN_GIL,
};
PyThreadState *tstate = NULL;
PyStatus status = Py_NewInterpreterFromConfig(&tstate, &config);
if (PyStatus_Exception(status)) {
    return -1;
}
/* The new interpreter is now active in the current thread. */
```

Per ulteriori esempi su come utilizzare la C-API per interpreti secondari con un GIL per interprete, vedi Modules/_xxsubinterpretersmodule.c.

(Contributi da Eric Snow in gh-104210, ecc.)

2.4 PEP 669: Monitoraggio a basso impatto per CPython

PEP 669 definisce una nuova `API` per profiler, debugger e altri strumenti per monitorare eventi in CPython. Copre una vasta gamma di eventi, inclusi chiamate, ritorni, righe, eccezioni, salti e altro. Questo significa che paghi solo per quello che usi, fornendo supporto per strumenti di debug e copertura a quasi zero overhead. Vedi `sys.monitoring` per i dettagli.

(Contributi da Mark Shannon in gh-103082.)

2.5 PEP 688: Rendere accessibile il protocollo buffer in Python

PEP 688 introduce un modo per utilizzare il protocollo buffer dal codice Python. Le classi che implementano il metodo `__buffer__()` sono ora utilizzabili come tipi di buffer.

Il nuovo ABC `collections.abc.Buffer` fornisce un modo standard per rappresentare oggetti buffer, ad esempio nelle annotazioni di tipo. Il nuovo enumerativo `inspect.BufferFlags` rappresenta i flag che possono essere utilizzati per personalizzare la creazione del buffer. (Contributo di Jelle Zijlstra in gh-102500.)

2.6 PEP 709: Inlining delle comprehension

Le comprehension di dizionari, liste e set sono ora inlined, piuttosto che creare un nuovo oggetto funzione monouso per ogni esecuzione della comprehension. Questo accelera l'esecuzione di una comprehension fino a due volte. Vedi **PEP 709** per ulteriori dettagli.

Le variabili di iterazione delle comprehension rimangono isolate e non sovrascrivono una variabile con lo stesso nome nell'ambito esterno, né sono visibili dopo la comprehension. L'inlining comporta alcuni cambiamenti di comportamento visibili:

- Nei traceback non c'è più un frame separato per la comprehension, e il tracciamento/profiling non mostra più la comprehension come una chiamata di funzione.
- Il modulo `symtable` non produrrà più tabelle simbolo figlie per ogni comprehension; invece, le variabili locali della comprehension saranno incluse nella tabella simbolo della funzione principale.
- Chiamando `locals()` all'interno di una comprehension ora include variabili dall'esterno della comprehension e non include più la variabile sintetica `.0` per «l'argomento» della comprehension.

- Una comprehension che itera direttamente su `locals()` (ad es. `[k for k in locals()]`) potrebbe vedere «RuntimeError: dimensione del dizionario cambiata durante l'iterazione» quando eseguita sotto tracciamento (ad es. misurazione della copertura del codice). Questo è lo stesso comportamento già visto, ad es. `for k in locals():`. Per evitare l'errore, prima crea una lista di chiavi da iterare: `keys = list(locals()); [k for k in keys]`.

(Contributi di Carl Meyer e Vladimir Matveev in **PEP 709**.)

2.7 Messaggi di errore migliorati

- I moduli della libreria standard sono ora potenzialmente suggeriti come parte dei messaggi di errore visualizzati dall'interprete quando un'eccezione `NameError` viene sollevata a livello superiore. (Contributi da Pablo Galindo in gh-98254.)

```
>>> sys.version_info
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sys' is not defined. Did you forget to import 'sys'?
```

- Migliora il suggerimento di errore per le eccezioni `NameError` per le istanze. Ora, se un'eccezione `NameError` viene sollevata in un metodo e l'istanza ha un attributo esattamente uguale al nome nell'eccezione, il suggerimento includerà `self.<NAME>` invece della corrispondenza più vicina nell'ambito del metodo. (Contributi da Pablo Galindo in gh-99139.)

```
>>> class A:
...     def __init__(self):
...         self.blech = 1
...
...     def foo(self):
...         somethin = blech
...
>>> A().foo()
Traceback (most recent call last):
  File "<stdin>", line 1
    somethin = blech
               ^^^^^
NameError: name 'blech' is not defined. Did you mean: 'self.blech'?
```

- Migliora il messaggio di errore `SyntaxError` quando l'utente digita `import x from y` invece di `from y import x`. (Contributi da Pablo Galindo in gh-98931.)

```
>>> import a.y.z from b.y.z
Traceback (most recent call last):
  File "<stdin>", line 1
    import a.y.z from b.y.z
    ^^^^^^^^^^^^^^^^^^^^^^^
SyntaxError: Did you mean to use 'from ... import ...' instead?
```

- Le eccezioni `ImportError` sollevate da dichiarazioni di `from <module> import <name>` fallite includono ora suggerimenti per il valore di `<name>` basati sui nomi disponibili in `<module>`. (Contributi da Pablo Galindo in gh-91058.)

```
>>> from collections import chainmap
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: cannot import name 'chainmap' from 'collections'. Did you mean:
↪'ChainMap'?
```

3 Nuove funzionalità relative agli hint di tipo

Questa sezione copre le principali modifiche che riguardano **type hints** e il modulo `typing`.

3.1 PEP 692: Utilizzo di `TypedDict` per una digitazione `**kwargs` più precisa

La digitazione di `**kwargs` in una firma di funzione introdotta da **PEP 484** consentiva annotazioni valide solo nei casi in cui tutti i `**kwargs` erano dello stesso tipo.

PEP 692 specifica un modo più preciso di digitare `**kwargs` affidandosi a dizionari tipizzati:

```
from typing import TypedDict, Unpack

class Movie(TypedDict):
  name: str
  year: int

def foo(**kwargs: Unpack[Movie]): ...
```

Vedi **PEP 692** per ulteriori dettagli.

(Contributo di Franek Magiera in gh-103629.)

3.2 PEP 698: Decoratore Override per tipizzazione statica

Un nuovo decoratore `typing.override()` è stato aggiunto al modulo `typing`. Indica ai checker di tipo che il metodo è destinato a sovrascrivere un metodo in una superclasse. Questo consente ai checker di tipo di rilevare errori in cui un metodo che dovrebbe sovrascrivere qualcosa in una classe base in realtà non lo fa.

Esempio:

```
from typing import override

class Base:
  def get_color(self) -> str:
    return "blue"

class GoodChild(Base):
  @override  # ok: overrides Base.get_color
  def get_color(self) -> str:
    return "yellow"

class BadChild(Base):
  @override  # type checker error: does not override Base.get_color
  def get_colour(self) -> str:
    return "red"
```

Vedere **PEP 698** per maggiori dettagli.

(Contributo di Steven Troxler in gh-101561.)

4 Altre modifiche al Linguaggio

- Il parser ora solleva `SyntaxError` quando analizza il codice sorgente contenente byte nulli. (Contributo di Pablo Galindo in gh-96670.)
- Una coppia carattere-barra rovesciata che non è una sequenza di escape valida ora genera un `SyntaxWarning`, invece di un `DeprecationWarning`. Ad esempio, `re.compile("\d+\.\d+")` ora

emette un `SyntaxWarning` (`"\d"` è una sequenza di escape non valida, utilizzare stringhe raw per le espressioni regolari: `re.compile(r"\d+\.\d+")`). In una futura versione di Python, verrà infine sollevato un `SyntaxError`, invece di un `SyntaxWarning`. (Contributo di Victor Stinner in gh-98401.)

- Le sequenze di escape ottali con valore maggiore di `0o377` (es: `"\477"`), deprecate in Python 3.11, ora producono un `SyntaxWarning`, invece di un `DeprecationWarning`. In una futura versione di Python, diventeranno infine un `SyntaxError`. (Contributo di Victor Stinner in gh-98401.)
- Le variabili utilizzate nella parte target delle comprensioni che non vengono memorizzate possono ora essere utilizzate nelle espressioni di assegnazione (`:=`). Ad esempio, in `[(b := 1) for a, b.prop in some_iter]`, l'assegnazione a `b` è ora consentita. Si noti che l'assegnazione a variabili memorizzate nella parte target delle comprensioni (come `a`) è ancora vietata, come per **PEP 572**. (Contributo di Nikita Sobolev in gh-100581.)
- Le eccezioni sollevate nel metodo `__set_name__` di una classe o di un tipo non vengono più avvolte da un `RuntimeError`. Le informazioni di contesto vengono aggiunte all'eccezione come nota **PEP 678**. (Contributo di Irit Katriel in gh-77757.)
- Quando una costruzione `try-except*` gestisce l'intero `ExceptionGroup` e solleva un'altra eccezione, quella eccezione non è più avvolta in un `ExceptionGroup`. Anche modificato nella versione 3.11.4. (Contributo di Irit Katriel in gh-103590.)
- Il Garbage Collector ora viene eseguito solo sul meccanismo eval breaker del ciclo di valutazione del bytecode Python invece delle allocazioni di oggetti. Il GC può anche essere eseguito quando viene chiamato `PyErr_CheckSignals()` in modo che le estensioni C che devono essere eseguite per lungo tempo senza eseguire alcun codice Python abbiano anche la possibilità di eseguire periodicamente il GC. (Contributo di Pablo Galindo in gh-97922.)
- Tutti i callable builtin e delle estensioni che si aspettano parametri booleani ora accettano argomenti di qualsiasi tipo invece di soli `bool` e `int`. (Contributo di Serhiy Storchaka in gh-60203.)
- `memoryview` supporta ora il tipo half-float (il codice di formato «e»). (Contributo di Donghee Na e Antoine Pitrou in gh-90751.)
- Gli oggetti `slice` sono ora hashabili, consentendo loro di essere utilizzati come chiavi di dizionari e elementi di insiemi. (Contributo di Will Bradshaw, Furkan Onder e Raymond Hettinger in gh-101264.)
- `sum()` utilizza ora la sommatoria Neumaier per migliorare la precisione e la commutatività quando si sommano float o mix di int e float. (Contributo di Raymond Hettinger in gh-100425.)
- `ast.parse()` solleva ora `SyntaxError` invece di `ValueError` quando analizza il codice sorgente contenente byte nulli. (Contributo di Pablo Galindo in gh-96670.)
- I metodi di estrazione in `tarfile`, e `shutil.unpack_archive()`, hanno un nuovo argomento *filter* che consente di limitare le funzionalità tar che possono essere sorprendenti o pericolose, come la creazione di file al di fuori della directory di destinazione. Vedere tarfile extraction filters per ulteriori dettagli. In Python 3.14, il valore predefinito passerà a `'data'`. (Contributo di Petr Viktorin in **PEP 706**.)
- Le istanze di `types.MappingProxyType` sono ora hashabili se la mappa sottostante è hashabile. (Contributo di Serhiy Storchaka in gh-87995.)
- Aggiungo supporto per il profiler perf attraverso la nuova variabile di ambiente `PYTHONPERFSUPPORT` e l'opzione da riga di comando `-X perf`, così come le nuove funzioni `sys.activate_stack_trampoline()`, `sys.deactivate_stack_trampoline()` e `sys.is_stack_trampoline_active()`. (Progettazione di Pablo Galindo. Contributo di Pablo Galindo e Christian Heimes con contributi di Gregory P. Smith [Google] e Mark Shannon in gh-96123.)

5 Nuovi Moduli

- Nessuno.

6 Moduli Migliorati

6.1 array

- La classe `array.array` ora supporta il subscript, rendendola un generic type. (Contributo di Jelle Zijlstra in gh-98658.)

6.2 asyncio

- Le prestazioni della scrittura sui socket in `asyncio` sono state notevolmente migliorate. `asyncio` ora evita copie non necessarie durante la scrittura sui socket e utilizza `sendmsg()` se la piattaforma lo supporta. (Contributo di Kumar Aditya in gh-91166.)
- Aggiungo le funzioni `asyncio.eager_task_factory()` e `asyncio.create_eager_task_factory()` per permettere a un event loop di optare per l'esecuzione rapida dei task, rendendo alcuni casi d'uso 2x a 5x più veloci. (Contributo di Jacob Bower & Itamar Oren in gh-102853, gh-104140 e gh-104138)
- Su Linux, `asyncio` utilizza `asyncio.PidfdChildWatcher` di default se `os.pidfd_open()` è disponibile e funzionale invece di `asyncio.ThreadedChildWatcher`. (Contributo di Kumar Aditya in gh-98024.)
- L'event loop ora utilizza il miglior watcher per figli disponibile per ogni piattaforma (`asyncio.PidfdChildWatcher` se supportato e `asyncio.ThreadedChildWatcher` altrimenti), quindi configurare manualmente un watcher per figli non è consigliato. (Contributo di Kumar Aditya in gh-94597.)
- Aggiungo il parametro *loop_factory* a `asyncio.run()` per consentire di specificare una factory per event loop personalizzata. (Contributo di Kumar Aditya in gh-99388.)
- Aggiungo l'implementazione in C di `asyncio.current_task()` per un miglioramento delle prestazioni da 4x a 6x. (Contributo di Itamar Oren e Pranav Thulasiram Bhat in gh-100344.)
- `asyncio.iscoroutine()` ora restituisce `False` per i generatori poiché `asyncio` non supporta le coroutine basate su generatori legacy. (Contributo di Kumar Aditya in gh-102748.)
- `asyncio.wait()` e `asyncio.as_completed()` accettano ora generatori che generano task. (Contributo di Kumar Aditya in gh-78530.)

6.3 calendar

- Aggiungo gli enum `calendar.Month` e `calendar.Day` che definiscono i mesi dell'anno e i giorni della settimana. (Contributo di Prince Roshan in gh-103636.)

6.4 csv

- Aggiungo le flag `csv.QUOTE_NOTNULL` e `csv.QUOTE_STRINGS` per fornire un controllo più fine di `None` e delle stringhe vuote da parte degli oggetti `csv.writer`.

6.5 dis

- Gli opcode delle pseudo-istruzioni (che sono utilizzati dal compilatore ma non compaiono nel bytecode eseguibile) sono ora esposti nel modulo `dis`. `HAVE_ARGUMENT` è ancora rilevante per i reali opcode, ma non è utile per le pseudo-istruzioni. Usare la nuova collezione `dis.hasarg` al suo posto. (Contributo di Irit Katriel in gh-94216.)
- Aggiungo la collezione `dis.hasexc` per indicare le istruzioni che impostano un gestore di eccezioni. (Contributo di Irit Katriel in gh-94216.)

6.6 fractions

- Gli oggetti di tipo `fractions.Fraction` ora supportano il formato float-style. (Contributo di Mark Dickinson in gh-100161.)

6.7 importlib.resources

- `importlib.resources.as_file()` supporta ora le directory di risorse. (Contributo di Jason R. Coombs in gh-97930.)
- Rinominare il primo parametro di `importlib.resources.files()` in *anchor*. (Contributo di Jason R. Coombs in gh-100598.)

6.8 inspect

- Aggiungo `inspect.markcoroutinefunction()` per contrassegnare le funzioni sync che restituiscono una coroutine per l'uso con `inspect.iscoroutinefunction()`. (Contributo di Carlton Gibson in gh-99247.)
- Aggiungo `inspect.getasyncgenstate()` e `inspect.getasyncgenlocals()` per determinare lo stato corrente dei generatori asincroni. (Contributo di Thomas Krennwallner in gh-79940.)
- Le prestazioni di `inspect.getattr_static()` sono state notevolmente migliorate. La maggior parte delle chiamate a questa funzione dovrebbe essere almeno 2x più veloce rispetto a Python 3.11. (Contributo di Alex Waygood in gh-103193.)

6.9 itertools

- Add `itertools.batched()` for collecting into even-sized tuples where the last batch may be shorter than the rest. (Contributed by Raymond Hettinger in gh-98363.)

6.10 matematica

- Aggiungi `math.sumprod()` per calcolare una somma di prodotti. (Contributo da Raymond Hettinger in gh-100485.)
- Estendere `math.nextafter()` per includere un argomento *steps* per muoversi su o giù di più passi alla volta. (Contributo da Matthias Goergens, Mark Dickinson, e Raymond Hettinger in gh-94906.)

6.11 os

- Aggiungi `os.PIDFD_NONBLOCK` per aprire un file descriptor per un processo con `os.pidfd_open()` in modalità non bloccante. (Contributo da Kumar Aditya in gh-93312.)
- `os.DirEntry` ora include un metodo `os.DirEntry.is_junction()` per verificare se l'elemento è una giunzione. (Contributo da Charles Machalow in gh-99547.)
- Aggiungi le funzioni `os.listdrives()`, `os.listvolumes()` e `os.listmounts()` su Windows per enumerare unità, volumi e punti di montaggio. (Contributo da Steve Dower in gh-102519.)
- Le funzioni `os.stat()` e `os.lstat()` sono ora più precise su Windows. Il campo `st_birthtime` verrà ora riempito con il tempo di creazione del file e `st_ctime` è deprecato ma contiene ancora il tempo di creazione (ma in futuro restituirà l'ultima modifica ai metadati, per coerenza con altre piattaforme). `st_dev` può essere fino a 64 bit e `st_ino` fino a 128 bit a seconda del file system, e `st_rdev` è sempre impostato su zero anziché valori errati. Entrambe le funzioni possono essere significativamente più veloci nelle nuove versioni di Windows. (Contributo da Steve Dower in gh-99726.)
- A partire dalla versione 3.12.4, `os.mkdir()` e `os.makedirs()` su Windows ora supportano il passaggio di un valore *mode* di `0o700` per applicare il controllo degli accessi alla nuova directory. Questo influisce implicitamente su `tempfile.mkdtemp()` ed è una mitigazione per **CVE 2024-4030**. Altri valori per *mode* continuano ad essere ignorati. (Contributo da Steve Dower in gh-118486.)

6.12 os.path

- Aggiungi `os.path.isjunction()` per verificare se un determinato percorso è una giunzione. (Contributo da Charles Machalow in gh-99547.)
- Aggiungi `os.path.splitroot()` per dividere un percorso in una triade `(drive, root, tail)`. (Contributo da Barney Gale in gh-101000.)

6.13 pathlib

- Aggiungi il supporto per la sottoclassificazione di `pathlib.PurePath` e `pathlib.Path`, oltre alle loro varianti specifiche Posix e Windows. Le sottoclassi possono sovrascrivere il metodo `pathlib.PurePath.with_segments()` per passare informazioni tra le istanze del cammino.
- Aggiungi `pathlib.Path.walk()` per attraversare gli alberi delle directory e generare tutti i nomi di file o directory al loro interno, simile a `os.walk()`. (Contributo da Stanislav Zmiev in gh-90385.)
- Aggiungi il parametro opzionale *walk_up* a `pathlib.PurePath.relative_to()` per permettere l'inserimento di voci `..` nel risultato; questo comportamento è più coerente con `os.path.relpath()`. (Contributo da Domenico Ragusa in gh-84538.)
- Aggiungi `pathlib.Path.is_junction()` come proxy per `os.path.isjunction()`. (Contributo da Charles Machalow in gh-99547.)
- Aggiungi il parametro opzionale *case_sensitive* a `pathlib.Path.glob()`, `pathlib.Path.rglob()` e `pathlib.PurePath.match()` per abbinare la sensibilità del percorso, consentendo un controllo più preciso sul processo di abbinamento.

6.14 pdb

- Aggiungi variabili di convenienza per tenere temporaneamente i valori per la sessione di debug e fornire un accesso rapido a valori come il frame corrente o il valore di ritorno. (Contributo da Tian Gao in gh-103693.)

6.15 random

- Aggiungi `random.binomialvariate()`. (Contributo da Raymond Hettinger in gh-81620.)
- Aggiungi un default di `lambd=1.0` a `random.expovariate()`. (Contributo da Raymond Hettinger in gh-100234.)

6.16 shutil

- `shutil.make_archive()` ora passa l'argomento *root_dir* agli archiviatori personalizzati che lo supportano. In questo caso, non cambia più temporaneamente la directory di lavoro corrente del processo in *root_dir* per eseguire l'archiviazione. (Contributo da Serhiy Storchaka in gh-74696.)
- `shutil.rmtree()` ora accetta un nuovo argomento *onexc* che è un gestore di errori simile a *onerror* ma che si aspetta un'istanza di eccezione piuttosto che un terzetto *(typ, val, tb)*. *onerror* è deprecato. (Contributo da Irit Katriel in gh-102828.)
- `shutil.which()` ora consulta la variabile d'ambiente *PATHEXT* per trovare corrispondenze all'interno di *PATH* su Windows anche quando il *cmd* dato include un componente di directory. (Contributo da Charles Machalow in gh-103179.)

  `shutil.which()` chiamerà `NeedCurrentDirectoryForExePathW` quando interroga per eseguibili su Windows per determinare se la directory di lavoro corrente dovrebbe essere aggiunta al percorso di ricerca. (Contributo da Charles Machalow in gh-103179.)

  `shutil.which()` restituirà un percorso corrispondente al *cmd* con un componente da `PATHEXT` prima di una corrispondenza diretta altrove nel percorso di ricerca su Windows. (Contributo da Charles Machalow in gh-103179.)

6.17 sqlite3

- Aggiungi una interfaccia da linea di comando. (Contributo da Erlend E. Aasland in gh-77617.)
- Aggiungi l'attributo `sqlite3.Connection.autocommit` a `sqlite3.Connection` e il parametro *autocommit* a `sqlite3.connect()` per controllare la **PEP 249**-compliant gestione delle transazioni. (Contributo da Erlend E. Aasland in gh-83638.)
- Aggiungi il parametro keyword-only *entrypoint* a `sqlite3.Connection.load_extension()`, per sovrascrivere il punto di ingresso dell'estensione SQLite. (Contributo da Erlend E. Aasland in gh-103015.)

- Aggiungi `sqlite3.Connection.getconfig()` e `sqlite3.Connection.setconfig()` a `sqlite3.Connection` per apportare modifiche di configurazione a una connessione al database. (Contribuito da Erlend E. Aasland in gh-103489.)

6.18 statistica

- Estendere `statistics.correlation()` per includere un metodo `ranked` per calcolare la correlazione di Spearman di dati classificati. (Contribuito da Raymond Hettinger in gh-95861.)

6.19 sys

- Aggiungi lo spazio dei nomi `sys.monitoring` per esporre la nuova API di monitoraggio *PEP 669*. (Contribuito da Mark Shannon in gh-103082.)
- Aggiungi `sys.activate_stack_trampoline()` e `sys.deactivate_stack_trampoline()` per attivare e disattivare i trampolini del profiler di stack, e `sys.is_stack_trampoline_active()` per verificare se i trampolini del profiler di stack sono attivi. (Contribuito da Pablo Galindo e Christian Heimes con contributi di Gregory P. Smith [Google] e Mark Shannon in gh-96123.)
- Aggiungi `sys.last_exc` che contiene l'ultima eccezione non gestita che è stata sollevata (per casi d'uso di debug post-mortem). Deprecati i tre campi che contengono le stesse informazioni nella loro forma legacy: `sys.last_type`, `sys.last_value` e `sys.last_traceback`. (Contribuito da Irit Katriel in gh-102778.)
- `sys._current_exceptions()` ora restituisce una mappa da thread-id a un'istanza di eccezione, anziché a una tupla `(typ, exc, tb)`. (Contribuito da Irit Katriel in gh-103176.)
- `sys.setrecursionlimit()` e `sys.getrecursionlimit()`. Il limite di ricorsione ora si applica solo al codice Python. Le funzioni built-in non utilizzano il limite di ricorsione, ma sono protette da un diverso meccanismo che impedisce alla ricorsione di causare il crash della macchina virtuale.

6.20 tempfile

- La funzione `tempfile.NamedTemporaryFile` ha un nuovo parametro opzionale *delete_on_close* (Contribuito da Evgeny Zorin in gh-58451.)
- `tempfile.mkdtemp()` ora ritorna sempre un percorso assoluto, anche se l'argomento fornito al parametro *dir* è un percorso relativo.
- A partire dalla versione 3.12.4 su Windows, la modalità predefinita `0o700` utilizzata da `tempfile.mkdtemp()` ora limita l'accesso alla nuova directory a causa di modifiche a `os.mkdir()`. Questa è una mitigazione per **CVE 2024-4030**. (Contribuito da Steve Dower in gh-118486.)

6.21 threading

- Aggiungi `threading.settrace_all_threads()` e `threading.setprofile_all_threads()` che permettono di impostare funzioni di traccia e di profilazione in tutti i thread in esecuzione oltre a quello chiamante. (Contribuito da Pablo Galindo in gh-93503.)

6.22 tkinter

- `tkinter.Canvas.coords()` ora appiattisce i suoi argomenti. Ora accetta non solo coordinate come argomenti separati (`x1, y1, x2, y2, ...`) e una sequenza di coordinate (`[x1, y1, x2, y2, ...]`), ma anche coordinate raggruppate in coppie (`(x1, y1), (x2, y2), ...` e `[(x1, y1), (x2, y2), ...]`), come i metodi `create_*()`. (Contributo di Serhiy Storchaka in gh-94473.)

6.23 tokenize

- Il modulo `tokenize` include le modifiche introdotte in **PEP 701**. (Contributo di Marta Gómez Macías e Pablo Galindo in gh-102856.) Vedi *Porting a Python 3.12* per maggiori informazioni sulle modifiche al modulo `tokenize`.

6.24 types

- Aggiungi `types.get_original_bases()` per consentire ulteriori introspezioni dei user-defined-generics quando sottoclassati. (Contributo di James Hilton-Balfe e Alex Waygood in gh-101827.)

6.25 typing

- I controlli `isinstance()` contro i `runtime-checkable protocols` ora utilizzano `inspect.getattr_static()` anziché `hasattr()` per verificare l'esistenza degli attributi. Ciò significa che i descrittori e i metodi `__getattr__()` non vengono più valutati inaspettatamente durante i controlli `isinstance()` contro i protocolli verificabili a runtime. Tuttavia, ciò può anche significare che alcuni oggetti che in precedenza erano considerati istanze di un protocollo verificabile a runtime potrebbero non essere più considerati tali con Python 3.12+ e viceversa. La maggior parte degli utenti non dovrebbe essere influenzata da questo cambiamento. (Contributo di Alex Waygood in gh-102433.)

- I membri di un protocollo verificabile a runtime sono ora considerati «congelati» a runtime non appena la classe è stata creata. Il monkey-patching degli attributi in un protocollo verificabile a runtime funzionerà ancora, ma non avrà alcun impatto sui controlli `isinstance()` che confrontano oggetti con il protocollo. Per esempio:

```
>>> from typing import Protocol, runtime_checkable
>>> @runtime_checkable
... class HasX(Protocol):
...     x = 1
...
>>> class Foo: ...
...
>>> f = Foo()
>>> isinstance(f, HasX)
False
>>> f.x = 1
>>> isinstance(f, HasX)
True
>>> HasX.y = 2
>>> isinstance(f, HasX)  # unchanged, even though HasX now also has a "y"␣
↪attribute
True
```

  Questa modifica è stata apportata per velocizzare i controlli `isinstance()` contro i protocolli verificabili a runtime.

- Il profilo delle prestazioni dei controlli `isinstance()` contro i `runtime-checkable protocols` è cambiato significativamente. La maggior parte dei controlli `isinstance()` contro i protocolli con pochi membri dovrebbe essere almeno 2 volte più veloce rispetto alla versione 3.11, e alcuni potrebbero essere 20 volte più veloci o più. Tuttavia, i controlli `isinstance()` contro i protocolli con molti membri potrebbero risultare più lenti rispetto a Python 3.11. (Contributo di Alex Waygood in gh-74690 e gh-103193.)

- Tutte le classi `typing.TypedDict` e `typing.NamedTuple` ora possiedono l'attributo `__orig_bases__`. (Contributo di Adrian Garcia Badaracco in gh-103699.)

- Aggiungi il parametro `frozen_default` a `typing.dataclass_transform()`. (Contributo di Erik De Bonte in gh-99957.)

6.26 unicodedata

- Il database Unicode è stato aggiornato alla versione 15.0.0. (Contributo di Benjamin Peterson in gh-96734).

6.27 unittest

Aggiungi l'opzione di linea di comando `--durations`, che mostra i N casi di test più lenti:

```
python3 -m unittest --durations=3 lib.tests.test_threading
.....
Slowest test durations
----------------------------------------------------------------------
1.210s     test_timeout (Lib.test.test_threading.BarrierTests)
1.003s     test_default_timeout (Lib.test.test_threading.BarrierTests)
0.518s     test_timeout (Lib.test.test_threading.EventTests)

(0.000 durations hidden.  Use -v to show these durations.)
----------------------------------------------------------------------
Ran 158 tests in 9.869s

OK (skipped=3)
```

(Contributo di Giampaolo Rodola in gh-48330)

6.28 uuid

- Aggiungi una interfaccia della linea di comando. (Contributo di Adam Chhina in gh-88597.)

7 Ottimizzazioni

- Rimuovi i membri `wstr` e `wstr_length` dagli oggetti Unicode. Ciò riduce la dimensione degli oggetti di 8 o 16 byte sulla piattaforma a 64 bit. (**PEP 623**) (Contributo di Inada Naoki in gh-92536.)
- Aggiungi il supporto sperimentale per l'uso dell'ottimizzatore binario BOLT nel processo di build, che migliora le prestazioni dell'1-5%. (Contributo di Kevin Modzelewski in gh-90536 e ottimizzato da Donghee Na in gh-101525)
- Velocizza la sostituzione delle espressioni regolari (funzioni `re.sub()` e `re.subn()` e i corrispondenti metodi di `re.Pattern`) per le stringhe di sostituzione contenenti riferimenti di gruppo di 2-3 volte. (Contributo di Serhiy Storchaka in gh-91524.)
- Velocizza la creazione di `asyncio.Task` rimandando la formattazione delle stringhe costosa. (Contributo di Itamar Oren in gh-103793.)
- Le funzioni `tokenize.tokenize()` e `tokenize.generate_tokens()` sono fino al 64% più veloci come effetto collaterale delle modifiche necessarie per coprire **PEP 701** nel modulo `tokenize`. (Contributo di Marta Gómez Macías e Pablo Galindo in gh-102856.)
- Velocizza le chiamate al metodo `super()` e i caricamenti degli attributi tramite la nuova istruzione `LOAD_SUPER_ATTR`. (Contributo di Carl Meyer e Vladimir Matveev in gh-103497.)

8 Modifiche al bytecode di CPython

- Rimuovi l'istruzione `LOAD_METHOD`. È stata unita a `LOAD_ATTR`. Ora `LOAD_ATTR` si comporterà come la vecchia istruzione `LOAD_METHOD` se il bit inferiore del suo oparg è impostato. (Contributo di Ken Jin in gh-93429.)
- Rimuovi le istruzioni `JUMP_IF_FALSE_OR_POP` e `JUMP_IF_TRUE_OR_POP`. (Contributo di Irit Katriel in gh-102859.)
- Rimuovi l'istruzione `PRECALL`. (Contributo di Mark Shannon in gh-92925.)
- Aggiungi le istruzioni `BINARY_SLICE` e `STORE_SLICE`. (Contributo di Mark Shannon in gh-94163.)
- Aggiungi le istruzioni `CALL_INTRINSIC_1`. (Contributo di Mark Shannon in gh-99005.)

- Aggiungi l'istruzione `CALL_INTRINSIC_2`. (Contributo di Irit Katriel in gh-101799.)
- Aggiungi l'istruzione `CLEANUP_THROW`. (Contributo di Brandt Bucher in gh-90997.)
- Aggiungi l'istruzione `END_SEND`. (Contributo di Mark Shannon in gh-103082.)
- Aggiungi l'istruzione `LOAD_FAST_AND_CLEAR` come parte dell'implementazione di **PEP 709**. (Contributo di Carl Meyer in gh-101441.)
- Aggiungi l'istruzione `LOAD_FAST_CHECK`. (Contributo di Dennis Sweeney in gh-93143.)
- Aggiungi le istruzioni `LOAD_FROM_DICT_OR_DEREF`, `LOAD_FROM_DICT_OR_GLOBALS`, e `LOAD_LOCALS` come parte dell'implementazione di **PEP 695**. Rimuovi l'opcode `LOAD_CLASSDEREF`, che può essere sostituito con `LOAD_LOCALS` più `LOAD_FROM_DICT_OR_DEREF`. (Contributo di Jelle Zijlstra in gh-103764.)
- Aggiungi l'istruzione `LOAD_SUPER_ATTR`. (Contributo di Carl Meyer e Vladimir Matveev in gh-103497.)
- Aggiungi l'istruzione `RETURN_CONST`. (Contributo di Wenyang Wang in gh-101632.)

9 Demos e Tools

- Rimuovi la directory `Tools/demo/` che conteneva vecchi script demo. Una copia può essere trovata nel progetto old-demos. (Contributo di Victor Stinner in gh-97681.)
- Rimuovi script di esempio obsoleti dalla directory `Tools/scripts/`. Una copia può essere trovata nel progetto old-demos. (Contributo di Victor Stinner in gh-97669.)

10 Deprecato

- `argparse`: I parametri *type*, *choices* e *metavar* di `argparse.BooleanOptionalAction` sono deprecati e saranno rimossi nella versione 3.14. (Contributo di Nikita Sobolev in gh-92248.)
- `ast`: Le seguenti caratteristiche di `ast` sono deprecate nella documentazione da Python 3.8, ora causano un'emissione di `DeprecationWarning` al runtime quando vengono accedute o utilizzate, e saranno rimosse in Python 3.14:
  - `ast.Num`
  - `ast.Str`
  - `ast.Bytes`
  - `ast.NameConstant`
  - `ast.Ellipsis`

  Utilizzare `ast.Constant` al suo posto. (Contributo di Serhiy Storchaka in gh-90953.)
- `asyncio`:
  - Le classi watcher dei processi figli `asyncio.MultiLoopChildWatcher`, `asyncio.FastChildWatcher`, `asyncio.AbstractChildWatcher` e `asyncio.SafeChildWatcher` sono deprecate e saranno rimosse in Python 3.14. (Contributo di Kumar Aditya in gh-94597.)
  - Le funzioni `asyncio.set_child_watcher()`, `asyncio.get_child_watcher()`, `asyncio.AbstractEventLoopPolicy.set_child_watcher()` e `asyncio.AbstractEventLoopPolicy.get_child_watcher()` sono deprecate e saranno rimosse in Python 3.14. (Contributo di Kumar Aditya in gh-94597.)
  - Il metodo `get_event_loop()` della politica del ciclo di eventi predefinito ora emette un `DeprecationWarning` se non c'è alcun ciclo di eventi corrente impostato e decide di crearne uno. (Contributo di Serhiy Storchaka e Guido van Rossum in gh-100160.)
- `calendar`: Le costanti `calendar.January` e `calendar.February` sono deprecate e sostituite da `calendar.JANUARY` e `calendar.FEBRUARY`. (Contributo di Prince Roshan in gh-103636.)

- `collections.abc`: Deprecata `collections.abc.ByteString`. Preferire `Sequence` o `collections.abc.Buffer`. Per l'uso nei tipi, preferire una unione, come `bytes | bytearray`, o `collections.abc.Buffer`. (Contributo di Shantanu Jain in gh-91896.)

- `datetime`: I metodi `utcnow()` e `utcfromtimestamp()` della classe `datetime.datetime` sono deprecati e saranno rimossi in una futura versione. Invece, utilizzare oggetti timezone-aware per rappresentare i datetime in UTC: rispettivamente, chiamare `now()` e `fromtimestamp()` con il parametro *tz* impostato su `datetime.UTC`. (Contributo di Paul Ganssle in gh-103857.)

- `email`: Deprecato il parametro *isdst* in `email.utils.localtime()`. (Contributo di Alan Williams in gh-72346.)

- `importlib.abc`: Deprecate le seguenti classi, pianificate per la rimozione in Python 3.14:

  - `importlib.abc.ResourceReader`
  - `importlib.abc.Traversable`
  - `importlib.abc.TraversableResources`

  Utilizzare invece le classi del modulo `importlib.resources.abc`:

  - `importlib.resources.abc.Traversable`
  - `importlib.resources.abc.TraversableResources`

  (Contributo di Jason R. Coombs e Hugo van Kemenade in gh-93963.)

- `itertools`: Deprecato il supporto per operazioni di copy, deepcopy e pickle, che è non documentato, inefficiente, storicamente soggetto a bug e inconsistente. Questo sarà rimosso in 3.14 per una significativa riduzione del volume di codice e del carico di manutenzione. (Contributo di Raymond Hettinger in gh-101588.)

- `multiprocessing`: In Python 3.14, il metodo di avvio predefinito di `multiprocessing` cambierà in uno più sicuro su Linux, BSD e altre piattaforme POSIX non macOS dove `'fork'` è attualmente il predefinito (gh-84559). Aggiungere un avviso in fase di runtime su questo cambiamento è stato ritenuto troppo dirompente poiché la maggioranza del codice non dovrebbe risentirne. Utilizzare le API `get_context()` o `set_start_method()` per specificare esplicitamente quando il codice *richiede* `'fork'`. Vedi contexts and start methods.

- `pkgutil`: Le funzioni `pkgutil.find_loader()` e `pkgutil.get_loader()` sono deprecate e saranno rimosse in Python 3.14; utilizzare invece `importlib.util.find_spec()`. (Contributo di Nikita Sobolev in gh-97850.)

- `pty`: Il modulo ha due funzioni non documentate `master_open()` e `slave_open()` che sono deprecate da Python 2 ma hanno ottenuto un vero e proprio `DeprecationWarning` solo in 3.12. Rimuoverle in 3.14. (Contributo di Soumendra Ganguly e Gregory P. Smith in gh-85984.)

- `os`:

  - I campi `st_ctime` restituiti da `os.stat()` e `os.lstat()` su Windows sono deprecati. In una futura versione, conterranno l'ora dell'ultima modifica dei metadati, in linea con altre piattaforme. Per ora, contengono ancora l'ora di creazione, che è anche disponibile nel nuovo campo `st_birthtime`. (Contributo di Steve Dower in gh-99726.)
  - On POSIX platforms, `os.fork()` can now raise a `DeprecationWarning` when it can detect being called from a multithreaded process. There has always been a fundamental incompatibility with the POSIX platform when doing so. Even if such code *appeared* to work. We added the warning to raise awareness as issues encountered by code doing this are becoming more frequent. See the `os.fork()` documentation for more details along with this discussion on fork being incompatible with threads for *why* we're now surfacing this longstanding platform compatibility problem to developers.

  Quando questo avviso appare a causa dell'uso di `multiprocessing` o `concurrent.futures` la soluzione è di utilizzare un diverso metodo di avvio di `multiprocessing` come `"spawn"` o `"forkserver"`.

- `shutil`: L'argomento *onerror* di `shutil.rmtree()` è deprecato; utilizzare *onexc* invece. (Contributo di Irit Katriel in gh-102828.)

- `sqlite3`:

  - default adapters and converters sono ora deprecati. Invece, utilizzare sqlite3-adapter-converter-recipes e adattarli alle tue esigenze. (Contributo di Erlend E. Aasland in gh-90016.)
  - In `execute()`, ora viene emesso un `DeprecationWarning` quando named placeholders sono utilizzati insieme a parametri forniti come una sequence invece che come un `dict`. A partire da Python 3.14, l'uso di named placeholders con parametri forniti come una sequence genererà un `ProgrammingError`. (Contributo di Erlend E. Aasland in gh-101698.)
- `sys`: I campi `sys.last_type`, `sys.last_value` e `sys.last_traceback` sono deprecati. Utilizzare `sys.last_exc` invece. (Contributo di Irit Katriel in gh-102778.)
- `tarfile`: L'estrazione degli archivi tar senza specificare *filter* è deprecata fino a Python 3.14, quando il filtro `'data'` diventerà il predefinito. Vedere tarfile-extraction-filter per i dettagli.
- `typing`:
  - `typing.Hashable` e `typing.Sized`, alias rispettivamente di `collections.abc.Hashable` e `collections.abc.Sized`, sono deprecati. (gh-94309.)
  - `typing.ByteString`, deprecata da Python 3.9, ora genera un `DeprecationWarning` quando viene utilizzata. (Contributo di Alex Waygood in gh-91896.)
- `xml.etree.ElementTree`: Il modulo ora emette `DeprecationWarning` quando si testa il valore di verità di un `xml.etree.ElementTree.Element`. Prima, l'implementazione in Python emetteva `FutureWarning`, e l'implementazione in C non emetteva nulla. (Contributo di Jacob Walls in gh-83122.)
- Le firme a 3 argomenti (type, value, traceback) di `coroutine throw()`, `generator throw()` e `async generator throw()` sono deprecate e potrebbero essere rimosse in una futura versione di Python. Utilizzare invece le versioni a un singolo argomento di queste funzioni. (Contributo di Ofey Chan in gh-89874.)
- `DeprecationWarning` is now raised when `__package__` on a module differs from `__spec__.parent` (previously it was `ImportWarning`). (Contributed by Brett Cannon in gh-65961.)
- Setting `__package__` or `__cached__` on a module is deprecated, and will cease to be set or taken into consideration by the import system in Python 3.14. (Contributed by Brett Cannon in gh-65961.)
- The bitwise inversion operator (`~`) on bool is deprecated. It will throw an error in Python 3.16. Use `not` for logical negation of bools instead. In the rare case that you really need the bitwise inversion of the underlying `int`, convert to int explicitly: `~int(x)`. (Contributed by Tim Hoffmann in gh-103487.)
- Accessing `co_lnotab` on code objects was deprecated in Python 3.10 via **PEP 626**, but it only got a proper `DeprecationWarning` in 3.12. May be removed in 3.15. (Contributed by Nikita Sobolev in gh-101866.)

10.1 Rimozione pianificata in Python 3.13

Moduli (vedi **PEP 594**):

- `aifc`
- `audioop`
- `cgi`
- `cgitb`
- `chunk`
- `crypt`
- `imghdr`
- `mailcap`
- `msilib`
- `nis`
- `nntplib`
- `ossaudiodev`

- `pipes`
- `sndhdr`
- `spwd`
- `sunau`
- `telnetlib`
- `uu`
- `xdrlib`

Altri moduli:

- `lib2to3`, e il programma **`2to3`** (gh-84540)

API:

- `configparser.LegacyInterpolation` (gh-90765)
- `locale.resetlocale()` (gh-90817)
- `turtle.RawTurtle.settiltangle()` (gh-50096)
- `unittest.findTestCases()` (gh-50096)
- `unittest.getTestCaseNames()` (gh-50096)
- `unittest.makeSuite()` (gh-50096)
- `unittest.TestProgram.usageExit()` (gh-67048)
- `webbrowser.MacOSX` (gh-86421)
- Catena di descrittori `classmethod` (gh-89519)
- metodi deprecati `importlib.resources`:
  - `contents()`
  - `is_resource()`
  - `open_binary()`
  - `open_text()`
  - `path()`
  - `read_binary()`
  - `read_text()`

  Use `importlib.resources.files()` instead. Refer to importlib-resources: Migrating from Legacy (gh-106531)

10.2 Rimozione prevista in Python 3.14

- `argparse`: I parametri *type*, *choices* e *metavar* di `argparse.BooleanOptionalAction` sono deprecati e saranno rimossi nella versione 3.14. (Contributo di Nikita Sobolev in gh-92248.)
- `ast`: The following features have been deprecated in documentation since Python 3.8, now cause a `DeprecationWarning` to be emitted at runtime when they are accessed or used, and will be removed in Python 3.14:
  - `ast.Num`
  - `ast.Str`
  - `ast.Bytes`
  - `ast.NameConstant`
  - `ast.Ellipsis`

Utilizzare `ast.Constant` al suo posto. (Contributo di Serhiy Storchaka in gh-90953.)

- `asyncio`:
  - The child watcher classes `MultiLoopChildWatcher`, `FastChildWatcher`, `AbstractChildWatcher` and `SafeChildWatcher` are deprecated and will be removed in Python 3.14. (Contributed by Kumar Aditya in gh-94597.)
  - Le funzioni `asyncio.set_child_watcher()`, `asyncio.get_child_watcher()`, `asyncio.AbstractEventLoopPolicy.set_child_watcher()` e `asyncio.AbstractEventLoopPolicy.get_child_watcher()` sono deprecate e saranno rimosse in Python 3.14. (Contributo di Kumar Aditya in gh-94597.)
  - Il metodo `get_event_loop()` della politica del ciclo di eventi predefinito ora emette un `DeprecationWarning` se non c'è alcun ciclo di eventi corrente impostato e decide di crearne uno. (Contributo di Serhiy Storchaka e Guido van Rossum in gh-100160.)
- `collections.abc`: Deprecated `ByteString`. Prefer `Sequence` or `Buffer`. For use in typing, prefer a union, like `bytes | bytearray`, or `collections.abc.Buffer`. (Contributed by Shantanu Jain in gh-91896.)
- `email`: Deprecated the *isdst* parameter in `email.utils.localtime()`. (Contributed by Alan Williams in gh-72346.)
- `importlib`: `__package__` and `__cached__` will cease to be set or taken into consideration by the import system (gh-97879).
- `importlib.abc` deprecated classes:
  - `importlib.abc.ResourceReader`
  - `importlib.abc.Traversable`
  - `importlib.abc.TraversableResources`

  Utilizzare invece le classi del modulo `importlib.resources.abc`:
  - `importlib.resources.abc.Traversable`
  - `importlib.resources.abc.TraversableResources`

  (Contributo di Jason R. Coombs e Hugo van Kemenade in gh-93963.)
- `itertools` had undocumented, inefficient, historically buggy, and inconsistent support for copy, deepcopy, and pickle operations. This will be removed in 3.14 for a significant reduction in code volume and maintenance burden. (Contributed by Raymond Hettinger in gh-101588.)
- `multiprocessing`: The default start method will change to a safer one on Linux, BSDs, and other non-macOS POSIX platforms where `'fork'` is currently the default (gh-84559). Adding a runtime warning about this was deemed too disruptive as the majority of code is not expected to care. Use the `get_context()` or `set_start_method()` APIs to explicitly specify when your code *requires* `'fork'`. See multiprocessing-start-methods.
- `pathlib`: `is_relative_to()` and `relative_to()`: passing additional arguments is deprecated.
- `pkgutil`: `find_loader()` and `get_loader()` now raise `DeprecationWarning`; use `importlib.util.find_spec()` instead. (Contributed by Nikita Sobolev in gh-97850.)
- `pty`:
  - `master_open()`: use `pty.openpty()`.
  - `slave_open()`: use `pty.openpty()`.
- `sqlite3`:
  - `version` and `version_info`.
  - `execute()` and `executemany()` if named placeholders are used and *parameters* is a sequence instead of a `dict`.

- typing: ByteString, deprecated since Python 3.9, now causes a DeprecationWarning to be emitted when it is used.
- urllib: urllib.parse.Quoter is deprecated: it was not intended to be a public API. (Contributed by Gregory P. Smith in gh-88168.)

10.3 Rimozione in sospeso in Python 3.15

- http.server.CGIHTTPRequestHandler will be removed along with its related --cgi flag to python -m http.server. It was obsolete and rarely used. No direct replacement exists. *Anything* is better than CGI to interface a web server with a request handler.
- locale: locale.getdefaultlocale() was deprecated in Python 3.11 and originally planned for removal in Python 3.13 (gh-90817), but removal has been postponed to Python 3.15. Use locale.setlocale(), locale.getencoding() and locale.getlocale() instead. (Contributed by Hugo van Kemenade in gh-111187.)
- pathlib: pathlib.PurePath.is_reserved() is deprecated and scheduled for removal in Python 3.15. From Python 3.13 onwards, use os.path.isreserved to detect reserved paths on Windows.
- platform: java_ver() is deprecated and will be removed in 3.15. It was largely untested, had a confusing API, and was only useful for Jython support. (Contributed by Nikita Sobolev in gh-116349.)
- threading: Passing any arguments to threading.RLock() is now deprecated. C version allows any numbers of args and kwargs, but they are just ignored. Python version does not allow any arguments. All arguments will be removed from threading.RLock() in Python 3.15. (Contributed by Nikita Sobolev in gh-102029.)
- typing.NamedTuple:
  - The undocumented keyword argument syntax for creating NamedTuple classes (NT = NamedTuple("NT", x=int)) is deprecated, and will be disallowed in 3.15. Use the class-based syntax or the functional syntax instead.
- types:
  - types.CodeType: Accessing co_lnotab was deprecated in **PEP 626** since 3.10 and was planned to be removed in 3.12, but it only got a proper DeprecationWarning in 3.12. May be removed in 3.15. (Contributed by Nikita Sobolev in gh-101866.)
- typing:
  - When using the functional syntax to create a NamedTuple class, failing to pass a value to the *fields* parameter (NT = NamedTuple("NT")) is deprecated. Passing None to the *fields* parameter (NT = NamedTuple("NT", None)) is also deprecated. Both will be disallowed in Python 3.15. To create a NamedTuple class with 0 fields, use class NT(NamedTuple): pass or NT = NamedTuple("NT", []).
- typing.TypedDict: When using the functional syntax to create a TypedDict class, failing to pass a value to the *fields* parameter (TD = TypedDict("TD")) is deprecated. Passing None to the *fields* parameter (TD = TypedDict("TD", None)) is also deprecated. Both will be disallowed in Python 3.15. To create a TypedDict class with 0 fields, use class TD(TypedDict): pass or TD = TypedDict("TD", {}).
- wave: Deprecate the getmark(), setmark() and getmarkers() methods of the wave.Wave_read and wave.Wave_write classes. They will be removed in Python 3.15. (Contributed by Victor Stinner in gh-105096.)

10.4 Pending Removal in Python 3.16

- The import system:
  - Setting __loader__ on a module while failing to set __spec__.loader is deprecated. In Python 3.16, __loader__ will cease to be set or taken into consideration by the import system or the standard library.
- array: array.array 'u' type (wchar_t): use the 'w' type instead (Py_UCS4).
- builtins: ~bool, bitwise inversion on bool.

- `symtable`: Deprecate `symtable.Class.get_methods()` due to the lack of interest. (Contributed by Bénédikt Tran in gh-119698.)

10.5 Rimozione in sospeso nelle versioni future

The following APIs will be removed in the future, although there is currently no date scheduled for their removal.

- `argparse`: Nesting argument groups and nesting mutually exclusive groups are deprecated.
- Il codice di formato `'u'` di `array` (gh-57281)
- `builtins`:
  - `bool(NotImplemented)`.
  - Generators: `throw(type, exc, tb)` and `athrow(type, exc, tb)` signature is deprecated: use `throw(exc)` and `athrow(exc)` instead, the single argument signature.
  - 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]`). A syntax warning is raised if the numeric literal is immediately followed by one of keywords `and`, `else`, `for`, `if`, `in`, `is` and `or`. In a future release it will be changed to a syntax error. (gh-87999)
  - Support for `__index__()` and `__int__()` method returning non-int type: these methods will be required to return an instance of a strict subclass of `int`.
  - Support for `__float__()` method returning a strict subclass of `float`: these methods will be required to return an instance of `float`.
  - Support for `__complex__()` method returning a strict subclass of `complex`: these methods will be required to return an instance of `complex`.
  - Delegation of `int()` to `__trunc__()` method.
  - Passing a complex number as the *real* or *imag* argument in the `complex()` constructor is now deprecated; it should only be passed as a single positional argument. (Contributed by Serhiy Storchaka in gh-109218.)
- `calendar`: Le costanti `calendar.January` e `calendar.February` sono deprecate e sostituite da `calendar.JANUARY` e `calendar.FEBRUARY`. (Contributo di Prince Roshan in gh-103636.)
- `codeobject.co_lnotab`: use the `codeobject.co_lines()` method instead.
- `datetime`:
  - `utcnow()`: use `datetime.datetime.now(tz=datetime.UTC)`.
  - `utcfromtimestamp()`: use `datetime.datetime.fromtimestamp(timestamp, tz=datetime.UTC)`.
- `gettext`: Plural value must be an integer.
- `importlib`:
  - `load_module()` method: use `exec_module()` instead.
  - `cache_from_source()` *debug_override* parameter is deprecated: use the *optimization* parameter instead.
- `importlib.metadata`:
  - `EntryPoints` tuple interface.
  - Implicit `None` on return values.
- `mailbox`: Use of StringIO input and text mode is deprecated, use BytesIO and binary mode instead.
- `os`: Calling `os.register_at_fork()` in multi-threaded process.
- `pydoc.ErrorDuringImport`: A tuple value for *exc_info* parameter is deprecated, use an exception instance.

- `re`: More strict rules are now applied for numerical group references and group names in regular expressions. Only sequence of ASCII digits is now accepted as a numerical reference. The group name in bytes patterns and replacement strings can now only contain ASCII letters and digits and underscore. (Contributed by Serhiy Storchaka in gh-91760.)
- `sre_compile`, `sre_constants` and `sre_parse` modules.
- `shutil`: `rmtree()`'s *onerror* parameter is deprecated in Python 3.12; use the *onexc* parameter instead.
- `ssl` options and protocols:
  - `ssl.SSLContext` without protocol argument is deprecated.
  - `ssl.SSLContext`: `set_npn_protocols()` and `selected_npn_protocol()` are deprecated: use ALPN instead.
  - `ssl.OP_NO_SSL*` options
  - `ssl.OP_NO_TLS*` options
  - `ssl.PROTOCOL_SSLv3`
  - `ssl.PROTOCOL_TLS`
  - `ssl.PROTOCOL_TLSv1`
  - `ssl.PROTOCOL_TLSv1_1`
  - `ssl.PROTOCOL_TLSv1_2`
  - `ssl.TLSVersion.SSLv3`
  - `ssl.TLSVersion.TLSv1`
  - `ssl.TLSVersion.TLSv1_1`
- `sysconfig.is_python_build()` *check_home* parameter is deprecated and ignored.
- `threading` methods:
  - `threading.Condition.notifyAll()`: use `notify_all()`.
  - `threading.Event.isSet()`: use `is_set()`.
  - `threading.Thread.isDaemon()`, `threading.Thread.setDaemon()`: use `threading.Thread.daemon` attribute.
  - `threading.Thread.getName()`, `threading.Thread.setName()`: use `threading.Thread.name` attribute.
  - `threading.currentThread()`: use `threading.current_thread()`.
  - `threading.activeCount()`: use `threading.active_count()`.
- `typing.Text` (gh-92332).
- `unittest.IsolatedAsyncioTestCase`: it is deprecated to return a value that is not `None` from a test case.
- `urllib.parse` deprecated functions: `urlparse()` instead
  - `splitattr()`
  - `splithost()`
  - `splitnport()`
  - `splitpasswd()`
  - `splitport()`
  - `splitquery()`
  - `splittag()`
  - `splittype()`

  - `splituser()`
  - `splitvalue()`
  - `to_bytes()`

- `urllib.request`: `URLopener` and `FancyURLopener` style of invoking requests is deprecated. Use newer `urlopen()` functions and methods.
- `wsgiref`: `SimpleHandler.stdout.write()` should not do partial writes.
- `xml.etree.ElementTree`: Testing the truth value of an `Element` is deprecated. In a future release it will always return `True`. Prefer explicit `len(elem)` or `elem is not None` tests instead.
- `zipimport.zipimporter.load_module()` is deprecated: use `exec_module()` instead.

11 Rimosso

11.1 asynchat e asyncore

- Questi due moduli sono stati rimossi secondo il programma in **PEP 594**, essendo stati deprecati in Python 3.6. Utilizzare `asyncio` invece. (Contribuito da Nikita Sobolev in gh-96580.)

11.2 configparser

- Diversi nomi deprecati nel `configparser` fin dalla versione 3.2 sono stati rimossi per gh-89336:
  - `configparser.ParsingError` non ha più un attributo o argomento `filename`. Utilizzare al suo posto l'attributo e l'argomento `source`.
  - `configparser` non ha più una classe `SafeConfigParser`. Utilizzare al suo posto il nome più breve `ConfigParser`.
  - `configparser.ConfigParser` non ha più un metodo `readfp`. Utilizzare al suo posto `read_file()`.

11.3 distutils

- Rimozione del pacchetto `distutils`. È stato deprecato in Python 3.10 da **PEP 632** «Deprecazione del modulo distutils». Per i progetti che ancora utilizzano `distutils` e non possono essere aggiornati ad altro, il progetto `setuptools` può essere installato: fornisce ancora `distutils`. (Contribuito da Victor Stinner in gh-92584.)

11.4 ensurepip

- Rimozione della ruota setuptools inclusa in `ensurepip`, e cessazione dell'installazione di setuptools negli ambienti creati da `venv`.

  `pip (>= 22.1)` non richiede che setuptools sia installato nell'ambiente. I pacchetti basati su `setuptools` (e su `distutils`) possono ancora essere utilizzati con `pip install`, poiché pip fornirà `setuptools` nell'ambiente di build che utilizza per costruire un pacchetto.

  `easy_install`, `pkg_resources`, `setuptools` e `distutils` non sono più forniti di default negli ambienti creati con `venv` o bootstrap con `ensurepip`, poiché fanno parte del pacchetto `setuptools`. Per i progetti che dipendono da questi a runtime, il progetto `setuptools` dovrebbe essere dichiarato come una dipendenza e installato separatamente (tipicamente, utilizzando pip).

  (Contribuito da Pradyun Gedam in gh-95299.)

11.5 enum

- Rimozione di `EnumMeta.__getattr__` di `enum`, che non è più necessario per l'accesso agli attributi degli enum. (Contribuito da Ethan Furman in gh-95083.)

11.6 ftplib

- Rimozione dell'attributo di classe `FTP_TLS.ssl_version` di `ftplib`: utilizzare invece il parametro *context*. (Contributo da Victor Stinner in gh-94172.)

11.7 gzip

- Rimozione dell'attributo `filename` di `gzip.GzipFile` di `gzip`, deprecato dalla versione Python 2.6, utilizzare invece l'attributo `name`. In modalità scrittura, l'attributo `filename` aggiungeva l'estensione di file `'.gz'` se non era presente. (Contributo da Victor Stinner in gh-94196.)

11.8 hashlib

- Remove the pure Python implementation of `hashlib`'s `hashlib.pbkdf2_hmac()`, deprecated in Python 3.10. Python 3.10 and newer requires OpenSSL 1.1.1 (**PEP 644**): this OpenSSL version provides a C implementation of `pbkdf2_hmac()` which is faster. (Contributed by Victor Stinner in gh-94199.)

11.9 importlib

- Molti di precedenti deprecazioni in `importlib` sono ora state completate:
  - References to, and support for `module_repr()` has been removed. (Contributed by Barry Warsaw in gh-97850.)
  - `importlib.util.set_package`, `importlib.util.set_loader` e `importlib.util.module_for_loader` sono stati tutti rimossi. (Contributo da Brett Cannon e Nikita Sobolev in gh-65961 e gh-97850.)
  - Il supporto per le API `find_loader()` e `find_module()` è stato rimosso. (Contributo da Barry Warsaw in gh-98040.)
  - `importlib.abc.Finder`, `pkgutil.ImpImporter` e `pkgutil.ImpLoader` sono stati rimossi. (Contributo da Barry Warsaw in gh-98040.)

11.10 imp

- Il modulo `imp` è stato rimosso. (Contributo da Barry Warsaw in gh-98040.)

  Per migrare, consultare la seguente tabella di corrispondenza:

imp	importlib
`imp.NullImporter`	Inserire `None` in `sys.path_importer_cache`
`imp.cache_from_s`	`importlib.util.cache_from_source()`
`imp.find_module()`	`importlib.util.find_spec()`
`imp.get_magic()`	`importlib.util.MAGIC_NUMBER`
`imp.get_suffixes`	`importlib.machinery.SOURCE_SUFFIXES`, `importlib.machinery.EXTENSION_SUFFIXES`, e `importlib.machinery.BYTECODE_SUFFIXES`
`imp.get_tag()`	`sys.implementation.cache_tag`
`imp.load_module()`	`importlib.import_module()`
`imp.new_module(na`	`types.ModuleType(name)`
`imp.reload()`	`importlib.reload()`
`imp.source_from_`	`importlib.util.source_from_cache()`
`imp.load_source()`	*Vedi sotto*

Sostituire `imp.load_source()` con:

```
import importlib.util
import importlib.machinery

def load_source(modname, filename):
    loader = importlib.machinery.SourceFileLoader(modname, filename)
    spec = importlib.util.spec_from_file_location(modname, filename,␣
↪loader=loader)
    module = importlib.util.module_from_spec(spec)
    # The module is always executed and not cached in sys.modules.
    # Uncomment the following line to cache the module.
    # sys.modules[module.__name__] = module
    loader.exec_module(module)
    return module
```

- Rimuovere le funzioni e gli attributi di `imp` senza sostituzioni:
  - Funzioni non documentate:
    * `imp.init_builtin()`
    * `imp.load_compiled()`
    * `imp.load_dynamic()`
    * `imp.load_package()`
  - `imp.lock_held()`, `imp.acquire_lock()`, `imp.release_lock()`: lo schema di blocco è cambiato in Python 3.3 a blocchi per modulo.
  - Costanti di `imp.find_module()`: `SEARCH_ERROR`, `PY_SOURCE`, `PY_COMPILED`, `C_EXTENSION`, `PY_RESOURCE`, `PKG_DIRECTORY`, `C_BUILTIN`, `PY_FROZEN`, `PY_CODERESOURCE`, `IMP_HOOK`.

11.11 io

- Rimuovere `io.OpenWrapper` di `io` e `_pyio.OpenWrapper`, deprecati in Python 3.10: utilizzare semplicemente `open()`. La funzione `open()` (`io.open()`) è una funzione integrata. Dalla versione 3.10 di Python, `_pyio.open()` è anche un metodo statico. (Contributo di Victor Stinner in gh-94169.)

11.12 locale

- Rimuovere la funzione `locale.format()` di `locale`, deprecata in Python 3.7: usare invece `locale.format_string()`. (Contributo di Victor Stinner in gh-94226.)

11.13 smtpd

- Il modulo `smtpd` è stato rimosso secondo il programma di **PEP 594**, essendo stato deprecato in Python 3.4.7 e 3.5.4. Utilizzare il modulo PyPI aiosmtpd o qualsiasi altro server basato su `asyncio`. (Contributo di Oleg Iarygin in gh-93243.)

11.14 sqlite3

- Le seguenti funzionalità non documentate di `sqlite3`, deprecate in Python 3.10, sono ora rimosse:
  - `sqlite3.enable_shared_cache()`
  - `sqlite3.OptimizedUnicode`

  Se è necessario utilizzare una cache condivisa, aprire il database in modalità URI utilizzando il parametro di query `cache=shared`.

  La text factory `sqlite3.OptimizedUnicode` è stata un alias per `str` fin dalla versione 3.3 di Python. Il codice che precedentemente impostava la text factory su `OptimizedUnicode` può utilizzare esplicitamente `str`, o affidarsi al valore predefinito che è anch'esso `str`.

  (Contributo di Erlend E. Aasland in gh-92548.)

11.15 ssl

- Rimuovere la funzione `ssl.RAND_pseudo_bytes()` di `ssl`, deprecata in Python 3.6: utilizzare invece `os.urandom()` o `ssl.RAND_bytes()`. (Contributo di Victor Stinner in gh-94199.)
- Rimuovere la funzione `ssl.match_hostname()`. È stata deprecata in Python 3.7. OpenSSL esegue il confronto degli hostname dalla versione 3.7 di Python, Python non utilizza più la funzione `ssl.match_hostname()`. (Contributo di Victor Stinner in gh-94199.)
- Rimuovere la funzione `ssl.wrap_socket()`, deprecata in Python 3.7: creare invece un oggetto `ssl.SSLContext` e chiamare il suo metodo `ssl.SSLContext.wrap_socket`. Qualsiasi pacchetto che utilizza ancora `ssl.wrap_socket()` è rotto e non sicuro. La funzione non invia né un'estensione TLS SNI né valida l'hostname del server. Il codice è soggetto a **CWE 295** (Validazione Impropria del Certificato). (Contributo di Victor Stinner in gh-94199.)

11.16 unittest

- Rimuovere molte funzionalità di `unittest` deprecate da lungo tempo:
  - Un certo numero di alias di metodo di `TestCase`:

Alias deprecato	Nome del Metodo	Deprecato in
`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

Puoi usare https://github.com/isidentical/teyit per modernizzare automaticamente i tuoi test unitari.

- Metodo `TestCase` `assertDictContainsSubset` non documentato e malfunzionante (deprecato in Python 3.2).
- Parametro non documentato `TestLoader.loadTestsFromModule` *use_load_tests* (deprecato e ignorato da Python 3.5).
- Un alias della classe `TextTestResult`: `_TextTestResult` (deprecato in Python 3.2).

(Contributo da Serhiy Storchaka in gh-89325.)

11.17 webbrowser

- Rimuovi il supporto per i browser obsoleti da `webbrowser`. I browser rimossi includono: Grail, Mosaic, Netscape, Galeon, Skipstone, Iceape, Firebird e le versioni di Firefox fino alla 35 (gh-102871).

11.18 xml.etree.ElementTree

- Rimuovi il metodo `ElementTree.Element.copy()` dell'implementazione pura in Python, deprecato in Python 3.10, usa invece la funzione `copy.copy()`. L'implementazione in C di `xml.etree.ElementTree` non ha metodo `copy()`, ma solo un metodo `__copy__()`. (Contributo da Victor Stinner in gh-94383.)

11.19 zipimport

- Rimuovi i metodi `find_loader()` e `find_module()` di `zipimport`, deprecati in Python 3.10: usa invece il metodo `find_spec()`. Vedi **PEP 451** per la spiegazione. (Contributo da Victor Stinner in gh-94379.)

11.20 Altri

- Rimuovi la regola `suspicious` dal `Makefile` della documentazione e da `Doc/tools/rstlint.py`, entrambe a favore di sphinx-lint. (Contributo da Julien Palard in gh-98179.)
- Rimuovi i parametri *keyfile* e *certfile* dai moduli `ftplib`, `imaplib`, `poplib` e `smtplib`, e i parametri *key_file*, *cert_file* e *check_hostname* dal modulo `http.client`, tutti deprecati da Python 3.6. Usa invece il parametro *context* (*ssl_context* in `imaplib`). (Contributo da Victor Stinner in gh-94172.)
- Rimuovi hack di compatibilità con `Jython` da diversi moduli e test della stdlib. (Contributo da Nikita Sobolev in gh-99482.)
- Rimuovi il flag `_use_broken_old_ctypes_structure_semantics_` dal modulo `ctypes`. (Contributo da Nikita Sobolev in gh-99285.)

12 Porting a Python 3.12

Questa sezione elenca i cambiamenti precedentemente descritti e altre correzioni di bug che potrebbero richiedere modifiche al tuo codice.

12.1 Modifiche nell'API di Python

- Ora si applicano regole più severe per i riferimenti di gruppo numerici e i nomi di gruppo nelle espressioni regolari. Sono accettati solo riferimenti numerici che sono una sequenza di cifre ASCII. Il nome del gruppo nei pattern in bytes e nelle stringhe di sostituzione ora può contenere solo lettere e cifre ASCII e il carattere di sottolineatura. (Contribuito da Serhiy Storchaka in gh-91760.)

- Rimosso il funzionamento di `randrange()` deprecato da Python 3.10. In precedenza, `randrange(10.0)` veniva convertito senza perdita a `randrange(10)`. Ora solleva un'eccezione `TypeError`. Inoltre, l'eccezione sollevata per valori non interi come `randrange(10.5)` o `randrange('10')` è cambiata da `ValueError` a `TypeError`. Questo impedisce anche bug dove `randrange(1e25)` selezionava silenziosamente da un intervallo più ampio rispetto a `randrange(10**25)`. (Originariamente suggerito da Serhiy Storchaka gh-86388.)

- `argparse.ArgumentParser` ha cambiato la codifica e il gestore di errori per la lettura degli argomenti dai file (ad esempio l'opzione `fromfile_prefix_chars`) dalla codifica di testo predefinita (ad esempio `locale.getpreferredencoding(False)`) a filesystem encoding and error handler. I file degli argomenti dovrebbero essere codificati in UTF-8 invece che in ANSI Codepage su Windows.

- Rimosso il modulo `smtpd` basato su `asyncore` deprecato in Python 3.4.7 e 3.5.4. Una sostituzione raccomandata è il modulo PyPI `asyncio`-based aiosmtpd.

- `shlex.split()`: Passare `None` per l'argomento *s* ora solleva un'eccezione, invece di leggere da `sys.stdin`. La funzionalità è stata deprecata in Python 3.9. (Contribuito da Victor Stinner in gh-94352.)

- Il modulo `os` non accetta più percorsi di tipo bytes, come `bytearray` e `memoryview`: solo il tipo esatto `bytes` è accettato per stringhe di tipo bytes. (Contribuito da Victor Stinner in gh-98393.)

- `syslog.openlog()` e `syslog.closelog()` ora falliscono se utilizzati nei subinterpreti. `syslog.syslog()` può ancora essere usato nei subinterpreti, ma solo se `syslog.openlog()` è stato già chiamato nel main interpreter. Queste nuove restrizioni non si applicano al main interpreter, quindi solo un insieme molto ristretto di utenti potrebbe essere interessato. Questa modifica aiuta con l'isolamento degli interpreti. Inoltre, `syslog` è un wrapper attorno a risorse globali del processo, che sono meglio gestite dal main interpreter. (Contribuito da Donghee Na in gh-99127.)

- Il comportamento di blocco non documentato di `cached_property()` è stato rimosso, perché bloccava tutte le istanze della classe, portando a una forte contention dei lock. Questo significa che ora una funzione getter della proprietà in cache potrebbe essere eseguita più di una volta per una singola istanza, se due thread sono in competizione. Per la maggior parte delle proprietà in cache semplici (ad esempio quelle che sono idempotenti e calcolano semplicemente un valore basato su altri attributi dell'istanza) questo sarà accettabile. Se è necessaria la sincronizzazione, implementare il locking all'interno della funzione getter della proprietà in cache o intorno ai punti di accesso multi-thread.

- `sys._current_exceptions()` ora restituisce una mappa da thread-id a un'istanza di eccezione, anziché a una tupla `(typ, exc, tb)`. (Contribuito da Irit Katriel in gh-103176.)

- Quando si estraggono file tar usando `tarfile` o `shutil.unpack_archive()`, passare l'argomento *filter* per limitare le funzionalità che possono essere sorprendenti o pericolose. Vedere tarfile-extraction-filter per i dettagli.

- L'output delle funzioni `tokenize.tokenize()` e `tokenize.generate_tokens()` è ora cambiato a causa delle modifiche introdotte in **PEP 701**. Questo significa che i token `STRING` non sono più emessi per le f-string e i token descritti in **PEP 701** sono ora prodotti al loro posto: `FSTRING_START`, `FSTRING_MIDDLE` e `FSTRING_END` sono ora emessi per le parti «string» delle f-string in aggiunta ai token appropriati per la tokenizzazione nei componenti di espressione. Ad esempio, per la f-string `f"start {1+1} end"` la vecchia versione del tokenizer emetteva:

```
1,0-1,18:            STRING         'f"start {1+1} end"'
```

mentre la nuova versione emette:

```
1,0-1,2:             FSTRING_START  'f"'
1,2-1,8:             FSTRING_MIDDLE 'start '
1,8-1,9:             OP             '{'
1,9-1,10:            NUMBER         '1'
1,10-1,11:           OP             '+'
1,11-1,12:           NUMBER         '1'
1,12-1,13:           OP             '}'
1,13-1,17:           FSTRING_MIDDLE ' end'
1,17-1,18:           FSTRING_END    '"'
```

Inoltre, potrebbero esserci alcuni cambiamenti comportamentali minori come conseguenza delle modifiche necessarie per supportare **PEP 701**. Alcuni di questi cambiamenti includono:

- L'attributo `type` dei token emessi quando si tokenizzano alcuni caratteri Python non validi come `!` è cambiato da `ERRORTOKEN` a `OP`.
- Le stringhe a riga singola incomplete ora sollevano `tokenize.TokenError` come fanno le stringhe multilinea incomplete.
- Alcuni codici Python incompleti o non validi ora sollevano `tokenize.TokenError` invece di restituire token `ERRORTOKEN` arbitrari durante la tokenizzazione.
- La miscelazione di tabulazioni e spazi come indentazione nello stesso file non è più supportata e genererà un'eccezione `TabError`.

- Il modulo `threading` ora si aspetta che il modulo `_thread` abbia un attributo `_is_main_interpreter`. È una funzione senza argomenti che restituisce `True` se l'interprete corrente è il main interpreter.

  Qualsiasi libreria o applicazione che fornisce un modulo `_thread` personalizzato dovrebbe fornire `_is_main_interpreter()`. (Vedere gh-112826.)

13 Modifiche alla Build

- Python non utilizza più `setup.py` per costruire moduli di estensione C condivisi. I parametri di build come intestazioni e librerie sono rilevati nello script `configure`. Le estensioni sono costruite da `Makefile`. La maggior parte delle estensioni utilizza `pkg-config` e ricorre al rilevamento manuale. (Contribuito da Christian Heimes in gh-93939.)
- `va_start()` con due parametri, come `va_start(args, format)`, è ora richiesto per costruire Python. `va_start()` non viene più chiamato con un solo parametro. (Contribuito da Kumar Aditya in gh-93207.)
- CPython ora utilizza l'opzione ThinLTO come politica predefinita di ottimizzazione del tempo di link se il compilatore Clang accetta il flag. (Contribuito da Donghee Na in gh-89536.)
- Aggiunta la variabile `COMPILEALL_OPTS` in `Makefile` per sovrascrivere le opzioni di `compileall` (predefinito: `-j0`) in `make install`. Inoltre, combinati i 3 comandi `compileall` in un unico comando per costruire i file .pyc per tutti i livelli di ottimizzazione (0, 1, 2) in una volta. (Contribuito da Victor Stinner in gh-99289.)
- Aggiunti triplette di piattaforma per LoongArch a 64 bit:
  - loongarch64-linux-gnusf
  - loongarch64-linux-gnuf32
  - loongarch64-linux-gnu

  (Contribuito da Zhang Na in gh-90656.)
- `PYTHON_FOR_REGEN` ora richiede Python 3.10 o versioni successive.

- Autoconf 2.71 e aclocal 1.16.4 sono ora richiesti per rigenerare `!configure`. (Contribuito da Christian Heimes in gh-89886.)
- Le build di Windows e gli installer per macOS da python.org ora utilizzano OpenSSL 3.0.

14 Modifiche all'API C

14.1 Nuove Funzionalità

- **PEP 697**: Introdotta la Unstable C API tier, destinata a strumenti di basso livello come debugger e compilatori JIT. Questa API può cambiare in ogni rilascio minore di CPython senza avvisi di deprecazione. I suoi contenuti sono contrassegnati dal prefisso `PyUnstable_` nei nomi.

  Costruttori di oggetti Code:

  - `PyUnstable_Code_New()` (rinominato da `PyCode_New`)
  - `PyUnstable_Code_NewWithPosOnlyArgs()` (rinominato da `PyCode_NewWithPosOnlyArgs`)

  Archiviazione extra per oggetti code (**PEP 523**):

  - `PyUnstable_Eval_RequestCodeExtraIndex()` (rinominato da `_PyEval_RequestCodeExtraIndex`)
  - `PyUnstable_Code_GetExtra()` (rinominato da `_PyCode_GetExtra`)
  - `PyUnstable_Code_SetExtra()` (rinominato da `_PyCode_SetExtra`)

  I nomi originali continueranno a essere disponibili fino alla rispettiva modifica dell'API.

  (Contribuito da Petr Viktorin in gh-101101.)

- **PEP 697**: Aggiunta un'API per estendere i tipi la cui struttura di memoria delle istanze è opaca:

  - `PyType_Spec.basicsize` può essere zero o negativo per specificare l'ereditarietà o l'estensione della dimensione della classe base.
  - `PyObject_GetTypeData()` e `PyType_GetTypeDataSize()` aggiunti per consentire l'accesso ai dati di istanza specifici della sottoclasse.
  - `Py_TPFLAGS_ITEMS_AT_END` e `PyObject_GetItemData()` aggiunti per consentire l'estensione sicura di determinati tipi a dimensione variabile, incluso `PyType_Type`.
  - `Py_RELATIVE_OFFSET` aggiunto per consentire la definizione di `membri` in termini di una struttura specifica della sottoclasse.

  (Contribuito da Petr Viktorin in gh-103509.)

- Aggiunta la nuova funzione della limited C API `PyType_FromMetaclass()`, che generalizza l'esistente `PyType_FromModuleAndSpec()` utilizzando un argomento metaclasse aggiuntivo. (Contribuito da Wenzel Jakob in gh-93012.)
- È stata aggiunta un'API per creare oggetti che possono essere chiamati utilizzando il protocollo vectorcall alla Limited API:

  - `Py_TPFLAGS_HAVE_VECTORCALL`
  - `PyVectorcall_NARGS()`
  - `PyVectorcall_Call()`
  - `vectorcallfunc`

  Il flag `Py_TPFLAGS_HAVE_VECTORCALL` viene ora rimosso da una classe quando il metodo della classe `__call__()` viene riassegnato. Questo rende sicuro l'uso di vectorcall con tipi mutabili (cioè tipi di heap senza il flag immutabile, `Py_TPFLAGS_IMMUTABLETYPE`). I tipi mutabili che non sovrascrivono `tp_call` ora ereditano il flag `Py_TPFLAGS_HAVE_VECTORCALL`. (Contributo di Petr Viktorin in gh-93274.)

The `Py_TPFLAGS_MANAGED_DICT` and `Py_TPFLAGS_MANAGED_WEAKREF` flags have been added. This allows extensions classes to support object `__dict__` and weakrefs with less bookkeeping, using less memory and with faster access.

- È stata aggiunta all" Limited API l'API per eseguire chiamate utilizzando il protocollo vectorcall:
  - `PyObject_Vectorcall()`
  - `PyObject_VectorcallMethod()`
  - `PY_VECTORCALL_ARGUMENTS_OFFSET`

  Questo significa che entrambe le estremità, in entrata e in uscita, del protocollo vectorcall sono ora disponibili nella Limited API. (Contributo di Wenzel Jakob in gh-98586.)
- Aggiunte due nuove funzioni pubbliche, `PyEval_SetProfileAllThreads()` e `PyEval_SetTraceAllThreads()`, che permettono di impostare funzioni di tracciamento e profilazione in tutti i thread in esecuzione oltre a quello chiamante. (Contributo di Pablo Galindo in gh-93503.)
- Aggiunta la nuova funzione `PyFunction_SetVectorcall()` alla C API che imposta il campo vectorcall di un dato `PyFunctionObject`. (Contributo di Andrew Frost in gh-92257.)
- La C API ora consente di registrare callback tramite `PyDict_AddWatcher()`, `PyDict_Watch()` e API correlate che vengono chiamate ogni volta che un dizionario viene modificato. Questo è inteso per l'uso da parte di interpreti ottimizzanti, compilatori JIT o debugger. (Contributo di Carl Meyer in gh-91052.)
- Aggiunte le API `PyType_AddWatcher()` e `PyType_Watch()` per registrare callback che ricevono notifiche sulle modifiche a un tipo. (Contributo di Carl Meyer in gh-91051.)
- Aggiunte le API `PyCode_AddWatcher()` e `PyCode_ClearWatcher()` per registrare callback che ricevono notifiche sulla creazione e distruzione di oggetti di codice. (Contributo di Itamar Oren in gh-91054.)
- Aggiunte le funzioni `PyFrame_GetVar()` e `PyFrame_GetVarString()` per ottenere una variabile di frame per nome. (Contributo di Victor Stinner in gh-91248.)
- Aggiunte `PyErr_GetRaisedException()` e `PyErr_SetRaisedException()` per salvare e ripristinare l'eccezione corrente. Queste funzioni restituiscono e accettano un singolo oggetto eccezione, anziché i tre argomenti della ormai deprecata `PyErr_Fetch()` e `PyErr_Restore()`. Questo è meno soggetto a errori e un po" più efficiente. (Contributo di Mark Shannon in gh-101578.)
- Aggiunto `_PyErr_ChainExceptions1`, che accetta un'istanza di eccezione, per sostituire l'API legacy `_PyErr_ChainExceptions`, che ora è deprecata. (Contributo di Mark Shannon in gh-101578.)
- Aggiunte `PyException_GetArgs()` e `PyException_SetArgs()` come funzioni di comodo per recuperare e modificare gli `args` passati al costruttore dell'eccezione. (Contributo di Mark Shannon in gh-101578.)
- Aggiunta `PyErr_DisplayException()`, che accetta un'istanza di eccezione, per sostituire l'API legacy `PyErr_Display()`. (Contributo di Irit Katriel in gh-102755).
- **PEP 683**: Introdurre *Immortal Objects*, che consente agli oggetti di bypassare i conti di riferimento, e modifiche correlate alla C-API:
  - **`_Py_IMMORTAL_REFCNT`: Il conteggio dei riferimenti che definisce un oggetto** come immortale.
  - `_Py_IsImmortal` Controlla se un oggetto ha il conteggio dei riferimenti immortali.
  - **`PyObject_HEAD_INIT` Ora inizializzerà il conteggio dei riferimenti a** `_Py_IMMORTAL_REFCNT` quando usato con `Py_BUILD_CORE`.
  - **`SSTATE_INTERNED_IMMORTAL` Un identificatore per oggetti Unicode internati** che sono immortali.
  - **`SSTATE_INTERNED_IMMORTAL_STATIC` Un identificatore per oggetti Unicode internati** che sono immortali e statici.

- **`sys.getunicodeinternedsize` Restituisce il numero totale di oggetti Unicode** che sono stati internati. Questo è ora necessario per `refleak.py` per monitorare correttamente i conti di riferimenti e i blocchi allocati.

(Contributo di Eddie Elizondo in gh-84436.)

- **PEP 684**: Aggiunta la nuova funzione `Py_NewInterpreterFromConfig()` e `PyInterpreterConfig`, che possono essere utilizzate per creare sotto-interpreti con i propri GIL. (Vedi *PEP 684: Un GIL per interprete* per maggiori informazioni.) (Contributo di Eric Snow in gh-104110.)
- Nella versione 3.12 della limited C API, le funzioni `Py_INCREF()` e `Py_DECREF()` sono ora implementate come chiamate di funzione opache per nascondere i dettagli dell'implementazione. (Contributo di Victor Stinner in gh-105387.)

14.2 Porting a Python 3.12

- Le API Unicode legacy basate sulla rappresentazione `Py_UNICODE*` sono state rimosse. Si prega di migrare alle API basate su UTF-8 o `wchar_t*`.
- Le funzioni di analisi degli argomenti come `PyArg_ParseTuple()` non supportano più i formati basati su `Py_UNICODE*` (ad esempio `u`, `Z`). Si prega di migrare ad altri formati per Unicode come `s`, `z`, `es` e `U`.
- `tp_weaklist` per tutti i tipi built-in statici è sempre `NULL`. Questo è un campo solo interno su `PyTypeObject`, ma segnaliamo la modifica nel caso qualcuno lo stia accedendo direttamente. Per evitare interruzioni, si consiglia di utilizzare invece la C-API pubblica esistente o, se necessario, la macro (solo interna) `_PyObject_GET_WEAKREFS_LISTPTR()`.
- Questo campo solo interno `PyTypeObject.tp_subclasses` potrebbe ora non essere un puntatore a oggetto valido. Il suo tipo è stato cambiato in `void*` per riflettere questo. Menzioniamo questa modifica nel caso qualcuno stia accedendo direttamente al campo solo interno.

  To get a list of subclasses, call the Python method `__subclasses__()` (using `PyObject_CallMethod()`, for example).
- Aggiunto supporto per più opzioni di formattazione (allineamento a sinistra, ottali, esadecimali maiuscoli, `intmax_t`, `ptrdiff_t`, stringhe C `wchar_t`, larghezza e precisione variabili) in `PyUnicode_FromFormat()` e `PyUnicode_FromFormatV()`. (Contributo di Serhiy Storchaka in gh-98836.)
- Un carattere di formato non riconosciuto in `PyUnicode_FromFormat()` e `PyUnicode_FromFormatV()` ora imposta una `SystemError`. Nelle versioni precedenti, causava la copia invariata del resto della stringa di formato nella stringa di risultato e scartava eventuali argomenti extra. (Contributo di Serhiy Storchaka in gh-95781.)
- Corretta la posizione errata del segno in `PyUnicode_FromFormat()` e `PyUnicode_FromFormatV()`. (Contributo di Philip Georgi in gh-95504.)
- Extension classes wanting to add a `__dict__` or weak reference slot should use `Py_TPFLAGS_MANAGED_DICT` and `Py_TPFLAGS_MANAGED_WEAKREF` instead of `tp_dictoffset` and `tp_weaklistoffset`, respectively. The use of `tp_dictoffset` and `tp_weaklistoffset` is still supported, but does not fully support multiple inheritance (gh-95589), and performance may be worse. Classes declaring `Py_TPFLAGS_MANAGED_DICT` should call `_PyObject_VisitManagedDict()` and `_PyObject_ClearManagedDict()` to traverse and clear their instance's dictionaries. To clear weakrefs, call `PyObject_ClearWeakRefs()`, as before.
- La funzione `PyUnicode_FSDecoder()` non accetta più percorsi simili a byte, come `bytearray` e tipi `memoryview`: solo il tipo esatto `bytes` è accettato per le stringhe di byte. (Contributo di Victor Stinner in gh-98393.)
- Le macro `Py_CLEAR`, `Py_SETREF` e `Py_XSETREF` ora valutano i loro argomenti solo una volta. Se un argomento ha effetti collaterali, questi effetti collaterali non vengono più duplicati. (Contributo di Victor Stinner in gh-98724.)

- L'indicatore di errore dell'interprete è ora sempre normalizzato. Questo significa che `PyErr_SetObject()`, `PyErr_SetString()` e le altre funzioni che impostano l'indicatore di errore ora normalizzano l'eccezione prima di memorizzarla. (Contributo di Mark Shannon in gh-101578.)

- `_Py_RefTotal` non è più autorevole e viene mantenuto solo per la compatibilità ABI. Nota che è una globale interna e disponibile solo build di debug. Se stai usando questa variabile, dovrai iniziare a usare `_Py_GetGlobalRefTotal()`.

- Le seguenti funzioni ora selezionano una metaclasse appropriata per il tipo appena creato:

  - `PyType_FromSpec()`
  - `PyType_FromSpecWithBases()`
  - `PyType_FromModuleAndSpec()`

  Creare classi la cui metaclasse sovrascrive `tp_new` è deprecato e in Python 3.14+ sarà disabilitato. Nota che queste funzioni ignorano `tp_new` della metaclasse, possibilmente permettendo un'inizializzazione incompleta.

  Nota che `PyType_FromMetaclass()` (aggiunto in Python 3.12) già impedisce la creazione di classi la cui metaclasse sovrascrive `tp_new` (`__new__()` in Python).

  Poiché `tp_new` sovrascrive quasi tutto ciò che fanno le funzioni `PyType_From*`, i due sono incompatibili tra loro. Il comportamento esistente — ignorare la metaclasse per diversi passaggi della creazione del tipo — è generalmente insicuro, poiché (meta)classi assumono che `tp_new` sia stato chiamato. Non esiste una soluzione generale semplice. Una delle seguenti opzioni potrebbe funzionare per te:

  - Se controlli la metaclasse, evita di usare `tp_new` in essa:
    * Se l'inizializzazione può essere saltata, può essere fatta al contrario in `tp_init`.
    * Se la metaclasse non ha bisogno di essere istanziata da Python, imposta il suo `tp_new` su `NULL` utilizzando il flag `Py_TPFLAGS_DISALLOW_INSTANTIATION`. Ciò la rende accettabile per le funzioni `PyType_From*`.
  - Evita le funzioni `PyType_From*`: se non ti servono funzionalità specifiche di C (slot o impostazione della dimensione dell'istanza), crea tipi tramite la chiamata della metaclasse.
  - Se *sai* che `tp_new` può essere saltato in sicurezza, filtra l'avviso di deprecazione usando `warnings.catch_warnings()` da Python.

- `PyOS_InputHook` e `PyOS_ReadlineFunctionPointer` non vengono più chiamati in subinterpreters. Questo perché i client generalmente si affidano allo stato globale del processo (poiché questi callback non hanno modo di recuperare lo stato del modulo di estensione).

  Questo evita anche situazioni in cui le estensioni potrebbero trovarsi in esecuzione in un subinterprete che non supportano (o non sono ancora state caricate in). Vedi gh-104668 per ulteriori informazioni.

- `PyLongObject` ha avuto i suoi interni cambiati per migliorare le prestazioni. Sebbene gli interni di `PyLongObject` siano privati, vengono utilizzati da alcuni moduli di estensione. I campi interni non devono più essere acceduti direttamente, ma devono essere utilizzate le funzioni API che iniziano con `PyLong_...`. Vengono fornite due nuove funzioni API *instabili* per un accesso efficiente al valore dei `PyLongObject`s che rientrano in una sola parola macchina:

  - `PyUnstable_Long_IsCompact()`
  - `PyUnstable_Long_CompactValue()`

- Gli allocatori personalizzati, impostati tramite `PyMem_SetAllocator()`, sono ora richiesti per essere thread-safe, indipendentemente dal dominio di memoria. Gli allocatori che non hanno uno stato proprio, compresi i «ganci», non sono interessati. Se il tuo allocatore personalizzato non è già thread-safe e hai bisogno di assistenza, crea un nuovo problema su GitHub e CC `@ericsnowcurrently`.

14.3 Deprecato

- In conformità con **PEP 699**, il campo `ma_version_tag` in `PyDictObject` è deprecato per i moduli di estensione. L'accesso a questo campo genererà un avviso del compilatore al momento della compilazione. Questo campo sarà rimosso in Python 3.14. (Contribuito da Ramvikrams e Kumar Aditya in gh-101193. PEP di Ken Jin.)
- Deprecare la variabile di configurazione globale:
  - `Py_DebugFlag`: usa `PyConfig.parser_debug`
  - `Py_VerboseFlag`: usa `PyConfig.verbose`
  - `Py_QuietFlag`: usa `PyConfig.quiet`
  - `Py_InteractiveFlag`: usa `PyConfig.interactive`
  - `Py_InspectFlag`: usa `PyConfig.inspect`
  - `Py_OptimizeFlag`: usa `PyConfig.optimization_level`
  - `Py_NoSiteFlag`: usa `PyConfig.site_import`
  - `Py_BytesWarningFlag`: usa `PyConfig.bytes_warning`
  - `Py_FrozenFlag`: usa `PyConfig.pathconfig_warnings`
  - `Py_IgnoreEnvironmentFlag`: usa `PyConfig.use_environment`
  - `Py_DontWriteBytecodeFlag`: usa `PyConfig.write_bytecode`
  - `Py_NoUserSiteDirectory`: usa `PyConfig.user_site_directory`
  - `Py_UnbufferedStdioFlag`: usa `PyConfig.buffered_stdio`
  - `Py_HashRandomizationFlag`: usa `PyConfig.use_hash_seed` e `PyConfig.hash_seed`
  - `Py_IsolatedFlag`: usa `PyConfig.isolated`
  - `Py_LegacyWindowsFSEncodingFlag`: usa `PyPreConfig.legacy_windows_fs_encoding`
  - `Py_LegacyWindowsStdioFlag`: usa `PyConfig.legacy_windows_stdio`
  - `Py_FileSystemDefaultEncoding`: usa `PyConfig.filesystem_encoding`
  - `Py_HasFileSystemDefaultEncoding`: usa `PyConfig.filesystem_encoding`
  - `Py_FileSystemDefaultEncodeErrors`: usa `PyConfig.filesystem_errors`
  - `Py_UTF8Mode`: usa `PyPreConfig.utf8_mode` (vedi `Py_PreInitialize()`)

  L'API `Py_InitializeFromConfig()` dovrebbe essere usata con `PyConfig` invece. (Contribuito da Victor Stinner in gh-77782.)
- Creare `tipi immutabili` con basi mutabili è deprecato e verrà disabilitato in Python 3.14. (gh-95388)
- L'intestazione `structmember.h` è deprecata, sebbene continui a essere disponibile e non ci siano piani per rimuoverla.

  I suoi contenuti sono ora disponibili semplicemente includendo `Python.h`, con un prefisso `Py` aggiunto se mancava:
  - `PyMemberDef`, `PyMember_GetOne()` e `PyMember_SetOne()`
  - Macro di tipo come `Py_T_INT`, `Py_T_DOUBLE`, ecc. (precedentemente `T_INT`, `T_DOUBLE`, ecc.)
  - Le bandiere `Py_READONLY` (precedentemente `READONLY`) e `Py_AUDIT_READ` (precedentemente tutte maiuscole)

  Diversi elementi non sono esposti da `Python.h`:
  - `T_OBJECT` (usa `Py_T_OBJECT_EX`)
  - `T_NONE` (precedentemente non documentato, e piuttosto strano)

  - La macro `WRITE_RESTRICTED` che non fa nulla.
  - Le macro `RESTRICTED` e `READ_RESTRICTED`, equivalenti di `Py_AUDIT_READ`.
  - In alcune configurazioni, `<stddef.h>` non è incluso da `Python.h`. Va incluso manualmente quando si utilizza `offsetof()`.

  L'header deprecato continua a fornire i suoi contenuti originali sotto i nomi originali. Il tuo vecchio codice può rimanere invariato, a meno che l'inclusione extra e le macro non spaziati ti diano molto fastidio.

  (Contribuito in gh-47146 da Petr Viktorin, basato su lavori precedenti di Alexander Belopolsky e Matthias Braun.)

- `PyErr_Fetch()` e `PyErr_Restore()` sono deprecate. Usa `PyErr_GetRaisedException()` e `PyErr_SetRaisedException()` invece. (Contribuito da Mark Shannon in gh-101578.)
- `PyErr_Display()` è deprecata. Usa `PyErr_DisplayException()` invece. (Contribuito da Irit Katriel in gh-102755).
- `_PyErr_ChainExceptions` è deprecata. Usa `_PyErr_ChainExceptions1` invece. (Contribuito da Irit Katriel in gh-102192.)
- Usare `PyType_FromSpec()`, `PyType_FromSpecWithBases()` o `PyType_FromModuleAndSpec()` per creare una classe la cui metaclasse sovrascrive `tp_new` è deprecato. Chiama direttamente la metaclasse invece.

Rimozione prevista in Python 3.14

- Il campo `ma_version_tag` in `PyDictObject` per moduli di estensione (**PEP 699**; gh-101193).
- Creare `tipi immutabili` con basi mutabili (gh-95388).
- Functions to configure Python's initialization, deprecated in Python 3.11:
  - `PySys_SetArgvEx()`: Set `PyConfig.argv` instead.
  - `PySys_SetArgv()`: Set `PyConfig.argv` instead.
  - `Py_SetProgramName()`: Set `PyConfig.program_name` instead.
  - `Py_SetPythonHome()`: Set `PyConfig.home` instead.

  L'API `Py_InitializeFromConfig()` dovrebbe essere usata con `PyConfig`.
- Variabili di configurazione globali:
  - `Py_DebugFlag`: Use `PyConfig.parser_debug` instead.
  - `Py_VerboseFlag`: Use `PyConfig.verbose` instead.
  - `Py_QuietFlag`: Use `PyConfig.quiet` instead.
  - `Py_InteractiveFlag`: Use `PyConfig.interactive` instead.
  - `Py_InspectFlag`: Use `PyConfig.inspect` instead.
  - `Py_OptimizeFlag`: Use `PyConfig.optimization_level` instead.
  - `Py_NoSiteFlag`: Use `PyConfig.site_import` instead.
  - `Py_BytesWarningFlag`: Use `PyConfig.bytes_warning` instead.
  - `Py_FrozenFlag`: Use `PyConfig.pathconfig_warnings` instead.
  - `Py_IgnoreEnvironmentFlag`: Use `PyConfig.use_environment` instead.
  - `Py_DontWriteBytecodeFlag`: Use `PyConfig.write_bytecode` instead.
  - `Py_NoUserSiteDirectory`: Use `PyConfig.user_site_directory` instead.
  - `Py_UnbufferedStdioFlag`: Use `PyConfig.buffered_stdio` instead.
  - `Py_HashRandomizationFlag`: Use `PyConfig.use_hash_seed` and `PyConfig.hash_seed` instead.

- `Py_IsolatedFlag`: Use `PyConfig.isolated` instead.
- `Py_LegacyWindowsFSEncodingFlag`: Use `PyPreConfig.legacy_windows_fs_encoding` instead.
- `Py_LegacyWindowsStdioFlag`: Use `PyConfig.legacy_windows_stdio` instead.
- `Py_FileSystemDefaultEncoding`: Use `PyConfig.filesystem_encoding` instead.
- `Py_HasFileSystemDefaultEncoding`: Use `PyConfig.filesystem_encoding` instead.
- `Py_FileSystemDefaultEncodeErrors`: Use `PyConfig.filesystem_errors` instead.
- `Py_UTF8Mode`: Use `PyPreConfig.utf8_mode` instead. (see `Py_PreInitialize()`)

L'API `Py_InitializeFromConfig()` dovrebbe essere usata con `PyConfig`.

Rimozione in sospeso in Python 3.15

- The bundled copy of `libmpdecimal`.
- The `PyImport_ImportModuleNoBlock()`: Use `PyImport_ImportModule()` instead.
- `PyWeakref_GetObject()` and `PyWeakref_GET_OBJECT()`: Use `PyWeakref_GetRef()` instead.
- `Py_UNICODE` type and the `Py_UNICODE_WIDE` macro: Use `wchar_t` instead.
- Funzioni di inizializzazione di Python:
  - `PySys_ResetWarnOptions()`: Clear `sys.warnoptions` and `warnings.filters` instead.
  - `Py_GetExecPrefix()`: Get `sys.exec_prefix` instead.
  - `Py_GetPath()`: Get `sys.path` instead.
  - `Py_GetPrefix()`: Get `sys.prefix` instead.
  - `Py_GetProgramFullPath()`: Get `sys.executable` instead.
  - `Py_GetProgramName()`: Get `sys.executable` instead.
  - `Py_GetPythonHome()`: Get `PyConfig.home` or the `PYTHONHOME` environment variable instead.

Rimozione in sospeso nelle versioni future

Le seguenti API sono deprecate e verranno rimosse, anche se attualmente non è prevista una data di rimozione.

- `Py_TPFLAGS_HAVE_FINALIZE`: Unneeded since Python 3.8.
- `PyErr_Fetch()`: Use `PyErr_GetRaisedException()` instead.
- `PyErr_NormalizeException()`: Use `PyErr_GetRaisedException()` instead.
- `PyErr_Restore()`: Use `PyErr_SetRaisedException()` instead.
- `PyModule_GetFilename()`: Use `PyModule_GetFilenameObject()` instead.
- `PyOS_AfterFork()`: Use `PyOS_AfterFork_Child()` instead.
- `PySlice_GetIndicesEx()`: Use `PySlice_Unpack()` and `PySlice_AdjustIndices()` instead.
- `PyUnicode_AsDecodedObject()`: Use `PyCodec_Decode()` instead.
- `PyUnicode_AsDecodedUnicode()`: Use `PyCodec_Decode()` instead.
- `PyUnicode_AsEncodedObject()`: Use `PyCodec_Encode()` instead.
- `PyUnicode_AsEncodedUnicode()`: Use `PyCodec_Encode()` instead.
- `PyUnicode_READY()`: Unneeded since Python 3.12
- `PyErr_Display()`: Use `PyErr_DisplayException()` instead.
- `_PyErr_ChainExceptions()`: Use `_PyErr_ChainExceptions1()` instead.

- `PyBytesObject.ob_shash` member: call `PyObject_Hash()` instead.
- `PyDictObject.ma_version_tag` member.
- API del Thread Local Storage (TLS):
  - `PyThread_create_key()`: Use `PyThread_tss_alloc()` instead.
  - `PyThread_delete_key()`: Use `PyThread_tss_free()` instead.
  - `PyThread_set_key_value()`: Use `PyThread_tss_set()` instead.
  - `PyThread_get_key_value()`: Use `PyThread_tss_get()` instead.
  - `PyThread_delete_key_value()`: Use `PyThread_tss_delete()` instead.
  - `PyThread_ReInitTLS()`: Unneeded since Python 3.7.

14.4 Rimosso

- Rimuovi il file di intestazione `token.h`. Non c'è mai stata un'API pubblica per il tokenizer C. Il file di intestazione `token.h` è stato progettato solo per essere usato dagli interni di Python. (Contribuito da Victor Stinner in gh-92651.)
- Le API Unicode legacy sono state rimosse. Vedi **PEP 623** per i dettagli.
  - `PyUnicode_WCHAR_KIND`
  - `PyUnicode_AS_UNICODE()`
  - `PyUnicode_AsUnicode()`
  - `PyUnicode_AsUnicodeAndSize()`
  - `PyUnicode_AS_DATA()`
  - `PyUnicode_FromUnicode()`
  - `PyUnicode_GET_SIZE()`
  - `PyUnicode_GetSize()`
  - `PyUnicode_GET_DATA_SIZE()`
- Rimuovi la macro della funzione `PyUnicode_InternImmortal()`. (Contributo di Victor Stinner in gh-85858.)

15 Cambiamenti notevoli in 3.12.4

15.1 ipaddress

- Risolto il comportamento di `is_global` e `is_private` in `IPv4Address`, `IPv6Address`, `IPv4Network` e `IPv6Network`.

16 Notable changes in 3.12.5

16.1 email

- Headers with embedded newlines are now quoted on output.

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

17 Notable changes in 3.12.6

17.1 email

- `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.)

18 Notable changes in 3.12.8

18.1 sys

- The previously undocumented special function `sys.getobjects()`, which only exists in specialized builds of Python, may now return objects from other interpreters than the one it's called in.

Indice

C

P

V