Что нового в Python?

Выпуск 3.8.8

А. М. Кучлинг

августа 12, 2021

https://Digitology.tech
Email: tweakit@bk.ru

Содержание

1 Резюме - основные моменты выпуска **3**

2 Новые функции **3**
2.1 Выражения назначения . . . 3
2.2 Только позиционные параметры . . . 4
2.3 Параллельный кэш файловой системы для скомпилированных файлов байт-кодов . . . 5
2.4 В отладочной сборке используется тот же ABI, что и в версии build . . . 5
2.5 Поддержка f-строки = для самостоятельного документирования выражений и отладки . . . 6
2.6 PEP 578: Python хуки аудита времени выполнения . . . 6
2.7 PEP 587: Инициализация конфигурации Python . . . 6
2.8 Vectorcall: протокол быстрого вызова для CPython . . . 7
2.9 Pickle протокол 5 с внеполосными буферами данных . . . 8

3 Другие языковые изменения **8**

4 Новые модули **10**

5 Улучшение модулей **11**
5.1 ast . . . 11
5.2 asyncio . . . 11
5.3 builtins . . . 12
5.4 collections . . . 12
5.5 cProfile . . . 13
5.6 csv . . . 13
5.7 curses . . . 13
5.8 ctypes . . . 13
5.9 datetime . . . 13
5.10 functools . . . 13
5.11 gc . . . 14
5.12 gettext . . . 14
5.13 gzip . . . 14
5.14 IDLE и idlelib . . . 15
5.15 inspect . . . 15
5.16 io . . . 15

5.17 itertools . . . 16
5.18 json.tool . . . 16
5.19 logging . . . 16
5.20 math . . . 16
5.21 mmap . . . 17
5.22 multiprocessing . . . 17
5.23 os . . . 17
5.24 os.path . . . 17
5.25 pathlib . . . 18
5.26 pickle . . . 18
5.27 plistlib . . . 18
5.28 pprint . . . 18
5.29 py_compile . . . 19
5.30 shlex . . . 19
5.31 shutil . . . 19
5.32 socket . . . 19
5.33 ssl . . . 19
5.34 statistics . . . 19
5.35 sys . . . 20
5.36 tarfile . . . 20
5.37 threading . . . 20
5.38 tokenize . . . 20
5.39 tkinter . . . 21
5.40 time . . . 21
5.41 typing . . . 21
5.42 unicodedata . . . 21
5.43 unittest . . . 22
5.44 venv . . . 22
5.45 weakref . . . 22
5.46 xml . . . 22
5.47 xmlrpc . . . 23

6 Оптимизация **23**

7 Изменения C API и сборки **24**

8 Запрещённые **25**

9 Удаление API и функций **27**

10 Портирование на Python 3.8 **27**
10.1 Изменения в поведении Python . . . 28
10.2 Изменения в API Python . . . 28
10.3 Изменения в API C . . . 30
10.4 Изменения байткода CPython . . . 32
10.5 Демонстрации и инструменты . . . 33

11 Заметные изменения в Python 3.8.1 **34**

12 Заметные изменения в Python 3.8.2 **34**

13 Заметные изменения в Python 3.8.3 **34**

14 Заметные изменения в Python 3.8.8 **35**

Редактор Raymond Hettinger

Эта статья объясняет новые возможности в Python 3.8, по сравнению с 3.7. Подробную информацию см. в разделе changelog.

Python 3.8 вышел 14 октября 2019 года.

1 Резюме - основные моменты выпуска

2 Новые функции

2.1 Выражения назначения

Существует новый синтаксис `:=`, который присваивает значения переменным как части большего выражения. Он ласково известен как «оператор моржей» благодаря своему сходству с глазами и бивнями моржа.

В этом примере выражение назначения помогает постараться не называть `len()` дважды:

```
if (n := len(a)) > 10:
    print(f"List is too long ({n} elements, expected <= 10)")
```

Аналогичное преимущество возникает при сопоставлении регулярных выражений, когда объекты соответствия необходимы дважды, один раз для проверки того, произошло ли совпадение, а другой - для извлечения подгруппы:

```
discount = 0.0
if (mo := re.search(r'(\d+)% discount', advertisement)):
    discount = float(mo.group(1)) / 100.0
```

Оператор также полезен с контурами while-loops, которые вычисляют значение для проверки завершения цикла и затем снова нуждаются в том же самом значении в теле цикла:

```
# Loop over fixed length blocks
while (block := f.read(256)) != '':
    process(block)
```

Другой мотивирующий случай использования возникает в представлениях списка, где значение, вычисленное в условии фильтрации, также необходимо в теле выражения:

```
[clean_name.title() for name in names
 if (clean_name := normalize('NFC', name)) in allowed_names]
```

Постарайтесь ограничить использование оператора моржей чисткой случаев, которые уменьшают сложность и улучшают удобочитаемость.

Полное описание см. в разделе **PEP 572**.

(Предоставлено Emily Morehouse в bpo-35224.)

2.2 Только позиционные параметры

Существует новый синтаксис параметра функции `/`, указывающий, что некоторые параметры функции должны быть указаны позиционно и не могут использоваться в качестве ключевых аргументов. Это то же обозначение, которое показывает `help()` для функций C, аннотированных с помощью инструмента Клиники споров Ларри Хастингса.

В следующем примере параметры *a* и *b* являются только позиционными, в то время как *c* или *d* могут быть позиционными или ключевыми, а *e* или *f* должны быть только ключевыми:

```
def f(a, b, /, c, d, *, e, f):
    print(a, b, c, d, e, f)
```

Ниже приведен действительный вызов:

```
f(10, 20, 30, d=40, e=50, f=60)
```

Однако это недопустимые вызовы:

```
f(10, b=20, c=30, d=40, e=50, f=60)   # b не должен быть ключевым аргументом
f(10, 20, 30, 40, 50, f=60)           # e должен быть ключевым аргументом
```

Один из вариантов использования этой нотации заключается в том, что она позволяет чистым Python функциям полностью эмулировать поведение существующих C-кодированных функций. Например, встроенная функция `divmod()` не принимает аргументы ключевой:

```
def divmod(a, b, /):
    "Emulate the built in divmod() function"
    return (a // b, a % b)
```

Другой вариант использования - исключить аргументы ключевой, если имя параметра не полезно. Например, у встроенной функции `len()` есть сигнатура `len(obj, /)`. Это исключает неудобные вызовы, такие как:

```
len(obj='hello')  # Ключевой аргумент "obj" ухудшает читабельность
```

Еще одним преимуществом маркировки параметра как позиционного является то, что он позволяет изменять имя параметра в будущем без риска нарушения клиентского код. Например, в модуле `statistics` имя параметра *dist* может быть изменено в будущем. Это стало возможным с помощью следующей спецификации функции:

```
def quantiles(dist, /, *, n=4, method='exclusive')
    ...
```

Так как параметры слева от `/` не отображаются как возможные ключевые слова, имена параметров остаются доступными для использования в `**kwargs`:

```
>>> def f(a, b, /, **kwargs):
...     print(a, b, kwargs)
...
>>> f(10, 20, a=1, b=2, c=3)         # a и b используются двумя способами
10 20 {'a': 1, 'b': 2, 'c': 3}
```

Это значительно упрощает внедрение функций и методы, который должен принять произвольные аргументы ключевой. Например, вот выдержка из код в модуле `collections`:

```
class Counter(dict):

    def __init__(self, iterable=None, /, **kwds):
        # Примечание "iterable" является возможным ключевым аргументом
```

Полное описание см. в разделе **PEP 570**.

(Представлен Pablo Galindo в bpo-36540.)

2.3 Параллельный кэш файловой системы для скомпилированных файлов байт-кодов

Новая настройка `PYTHONPYCACHEPREFIX` (также доступный как `-X pycache_prefix`) настраивает неявный bytecode кэш, чтобы использовать отдельное параллельное дерево файловой системы, а не дефолт подкаталоги `__pycache__` в рамках каждого исходного справочника.

О местоположении кэша сообщают в `sys.pycache_prefix` (`None` указывает на местоположение по умолчанию в подкаталогах `__pycache__`).

(Вклад Carl Meyer в bpo-33499.)

2.4 В отладочной сборке используется тот же ABI, что и в версии build

Теперь Python использует тот же ABI, независимо от того, встроен ли он в режиме выпуска или отладки. В Unix, когда Python построен в режиме отладки, теперь можно загружать расширения C, построенные в режиме выпуска, и расширения C, построенные с использованием стабильного ABI.

Сборки версий и отладки теперь совместимы с ABI: определение макроса `Py_DEBUG` больше не подразумевает макрос `Py_TRACE_REFS`, который вводит единственную несовместимость ABI. Макрос `Py_TRACE_REFS`, который добавляет функцию `sys.getobjects()` и переменную окружения `PYTHONDUMPREFS`, может быть установлен, используя новый `./configure --with-trace-refs`, строят выбор. (Представлен Victor Stinner в bpo-36465.)

В Unix расширения C больше не связаны с libpython, кроме Android и Cygwin. Для статически связанного Python теперь возможно загрузить построенное использование расширения C общей библиотеки Python. (Представлен Victor Stinner в bpo-21536.)

На Unix, когда Python построен в режиме отладки, импорт теперь также ищет расширения C, собранные в режиме выпуска и для расширений C, собранных со стабильным ABI. (Представлен Victor Stinner в bpo-36722.)

Чтобы встроить Python в приложение, необходимо передать `python3-config --libs --embed` новый параметр `--embed`, чтобы получить `-lpython3.8` (связать приложение с libpython). Чтобы поддержать и 3.8 и более старый, попробуйте `python3-config --libs --embed` сначала и отступление к `python3-config --libs` (без `--embed`), если предыдущая команда терпит неудачу.

Добавьте pkg-config модуль `python-3.8-embed`, чтобы включить Python в применение: `pkg-config python-3.8-embed --libs` включает `-lpython3.8`. Чтобы поддержать и 3.8 и более старый, попробуйте `pkg-config python-X.Y-embed --libs` сначала и отступление к `pkg-config python-X.Y --libs` (без `--embed`), если предыдущая команда терпит неудачу (замените `X.Y` версией Python).

С другой стороны, `pkg-config python3.8 --libs` больше не содержит `-lpython3.8`. Расширения C не должны быть связаны с libpython (за исключением Android и Cygwin, случаи которых обрабатываются сценарием); это изменение не совместимо по назначению. (Представлен Victor Stinner в bpo-36721.)

2.5 Поддержка f-строки = для самостоятельного документирования выражений и отладки

Добавлен спецификатор = к f-строка. Такая f-строка, как `f'{expr=}'`, будет расширена до текста выражения, знака равенства, а затем представления вычисляемого выражения. Например:

```
>>> user = 'eric_idle'
>>> member_since = date(1975, 7, 31)
>>> f'{user=} {member_since=}'
"user='eric_idle' member_since=datetime.date(1975, 7, 31)"
```

Обычные спецификаторы формата f-строк позволяют больше управлять отображением результата выражения:

```
>>> delta = date.today() - member_since
>>> f'{user=!s}  {delta.days=:,d}'
'user=eric_idle  delta.days=16,075'
```

Спецификатор = отображает выражение целиком, чтобы можно было отобразить вычисления:

```
>>> print(f'{theta=}  {cos(radians(theta))=:.3f}')
theta=30  cos(radians(theta))=0.866
```

(Представлен Eric V. Smith и Larry Hastings в bpo-36817.)

2.6 PEP 578: Python хуки аудита времени выполнения

PEP добавляет Audit Hook и Verified Open Hook. Оба доступны от Python и собственного кода, позволяя заявлениям и фреймворков, написанному в чистом Python код использовать в своих интересах дополнительные уведомления, в то время как также разрешение embedders или системные администраторы, чтобы развернуться строит из Python, где ревизия всегда включается.

Полную информацию см. в разделе **PEP 578**.

2.7 PEP 587: Инициализация конфигурации Python

PEP 587 добавляет новый API C, чтобы настроить инициализацию Python, обеспечивающую более прекрасный контроль над целой конфигурацией и лучшую ошибку при сообщении.

Новые структуры:

- `PyConfig`
- `PyPreConfig`
- `PyStatus`
- `PyWideStringList`

Новый функции:

- `PyConfig_Clear()`
- `PyConfig_InitIsolatedConfig()`
- `PyConfig_InitPythonConfig()`
- `PyConfig_Read()`
- `PyConfig_SetArgv()`

- `PyConfig_SetBytesArgv()`
- `PyConfig_SetBytesString()`
- `PyConfig_SetString()`
- `PyPreConfig_InitIsolatedConfig()`
- `PyPreConfig_InitPythonConfig()`
- `PyStatus_Error()`
- `PyStatus_Exception()`
- `PyStatus_Exit()`
- `PyStatus_IsError()`
- `PyStatus_IsExit()`
- `PyStatus_NoMemory()`
- `PyStatus_Ok()`
- `PyWideStringList_Append()`
- `PyWideStringList_Insert()`
- `Py_BytesMain()`
- `Py_ExitStatusException()`
- `Py_InitializeFromConfig()`
- `Py_PreInitialize()`
- `Py_PreInitializeFromArgs()`
- `Py_PreInitializeFromBytesArgs()`
- `Py_RunMain()`

Этот PEP также добавляет поля `_PyRuntimeState.preconfig` (тип `PyPreConfig`) и `PyInterpreterState.config` (тип `PyConfig`) к этим внутренним структурам. `PyInterpreterState.config` становится новой ссылочной конфигурацией, заменяя глобальные переменные конфигурации и другие частные переменные.

Документацию см. в разделе Конфигурация инициализации Python.

Полное описание см. в разделе **PEP 587**.

(Представлен Victor Stinner в bpo-36763.)

2.8 Vectorcall: протокол быстрого вызова для CPython

Протокол «vectorcall» добавляется в API Python/C. Предполагается формализовать существующие оптимизации, которые уже были сделаны для различных классы. Этот протокол может использоваться любым типом внутренней линии, реализующим вызываемый абонент.

В настоящее время этот документ носит предварительный характер. Цель состоит в том, чтобы сделать его полностью public в Python 3.9.

Полное описание см. в разделе **PEP 590**.

(Предоставлено Jeroen Demeyer и Mark Shannon в bpo-36974.)

2.9 Pickle протокол 5 с внеполосными буферами данных

Когда `pickle` - используемый, чтобы передать большие данные между процессами Python, чтобы использовать в своих интересах многоядерный или мультимашинная обработка, важно оптимизировать передачу, уменьшая копии памяти, и возможно применяя пользовательские методы, такие как зависимое от данных сжатие.

Протокол 5 `pickle` вводит поддержку внеполосных буферов, где **PEP 3118**-compatible данные могут передаваться отдельно от основного потока pickle, по усмотрению уровня связи.

Полное описание см. в разделе **PEP 574**.

(Представлен Antoine Pitrou в bpo-36785.)

3 Другие языковые изменения

- A `continue` инструкция был незаконным в `finally` клаузуле из-за проблемы с реализацией. В Python 3.8 это ограничение было снято. (Представлен Serhiy Storchaka в bpo-32489.)
- Типы `bool`, `int` и `fractions.Fraction` теперь имеют `as_integer_ratio()` метод, как в `float` и `decimal.Decimal`. Это дополнительное расширение API позволяет записывать `numerator, denominator = x.as_integer_ratio()` и работать в нескольких числовых типах. (Вклад Lisa Roach в bpo-33073 и Raymond Hettinger в bpo-37819.)
- Конструкторы `int`, `float` и `complex` теперь будут использовать специальный метод `__index__()`, если он доступен и соответствующие метод `__int__()`, `__float__()` или `__complex__()` недоступны. (Представлен Serhiy Storchaka в bpo-20092.)
- Добавлена поддержка `\N{name}` побегов в `regular expressions`:

```
>>> notice = 'Copyright © 2019'
>>> copyright_year_pattern = re.compile(r'\N{copyright sign}\s*(\d{4})')
>>> int(copyright_year_pattern.search(notice).group(1))
2019
```

  (Предоставлено Jonathan Eunice и Serhiy Storchaka в bpo-30688.)
- Dict и dictviews теперь могут повторяться в обратном порядке вставки с использованием `reversed()`. (Предоставлено R mi Lapeyre в bpo-33462.
- Синтаксис, разрешенный для имен ключевой в вызовах функций, был дополнительно ограничен. В частности, `f((keyword)=arg)` больше не допускается. Она никогда не предназначалась для того, чтобы разрешить более чем голое имя в левой части термина уступки аргумента ключевой. (Представлен Benjamin Peterson в bpo-34641.)
- Обобщенная повторяемая распаковка в `yield` и `return` инструкции больше не требует круглых скобок приложения. Это приводит синтаксис *yield* и *return* в лучшее соответствие с обычным синтаксисом назначения:

```
>>> def parse(family):
        lastname, *members = family.split()
        return lastname.upper(), *members

>>> parse('simpsons homer marge bart lisa sally')
('SIMPSONS', 'homer', 'marge', 'bart', 'lisa', 'sally')
```

  (Вклад David Cuthbert и Jordan Chapman в bpo-32117.)

- Если запятая пропущена в код, например `[(10, 20) (30, 40)]`, компилятор отображает `SyntaxWarning` с полезным предложением. Это улучшает наличие `TypeError`, указывающего на то, что первый кортеж не был вызываемым. (Представлен Serhiy Storchaka в bpo-15248.)

- Арифметические операции между подклассы `datetime.date` или `datetime.datetime` и `datetime.timedelta` объектами теперь возвращают сущность подкласса, а не базовой класс. Это также затрагивает тип возвращения операций, внедрение которых (прямо или косвенно) использует арифметику `datetime.timedelta`, такую как `astimezone()`. (Предоставлено Paul Ganssle в bpo-32417.)

- Когда Python интерпретатор прерван Ctrl-C (SIGINT), и получающееся исключение `KeyboardInterrupt` не поймано, процесс Python теперь выходит через сигнал SIGINT или с правильным выходом код, таким образом, что процесс запроса может обнаружить, что это умерло из-за Ctrl-C. Оболочки POSIX и Windows используют это для правильного завершения сценариев в интерактивных сеансах. (Вклад Google через Gregory P. Smith в bpo-1054041.)

- Некоторые расширенные стили программирования требуют обновления объекта `types.CodeType` для существующей функции. Поскольку код объекты являются неизменяемыми, необходимо создать новый код объект, смоделированный на существующем код объекте. При 19 параметрах это было несколько утомительно. Теперь новый `replace()` метод позволяет создать клон с несколькими измененными параметрами.

  Вот пример, который изменяет функцию `statistics.mean()`, чтобы препятствовать тому, чтобы параметр *data* был используемый как аргументом ключевой:

```
>>> from statistics import mean
>>> mean(data=[10, 20, 90])
40
>>> mean.__code__ = mean.__code__.replace(co_posonlyargcount=1)
>>> mean(data=[10, 20, 90])
Traceback (most recent call last):
  ...
TypeError: mean() got some positional-only arguments passed as keyword␣
 ↪arguments: 'data'
```

  (Представлен Victor Stinner в bpo-37032.)

- Для целых чисел трехаргументная форма функции `pow()` теперь позволяет экспоненте быть отрицательной в случае, когда основание относительно простое к модулю. Затем он вычисляет модульную обратную к основанию, когда степень равна `-1`, и подходящую степень этой обратной для других отрицательных степеней. Например, чтобы вычислить модульный мультипликативный обратный 38 по модулю 137, запишите:

```
>>> pow(38, -1, 137)
119
>>> 119 * 38 % 137
1
```

  Модульные обратные возникают в решении linear Diophantine equations. Например, чтобы найти целочисленные решения для `4258␣ + 147␣ = 369`, сначала переписать как `4258␣ ≡ 369 (mod 147)` затем решить:

```
>>> x = 369 * pow(4258, -1, 147) % 147
>>> y = (4258 * x - 369) // -147
>>> 4258 * x + 147 * y
369
```

(Представлен Mark Dickinson в bpo-36027.)

- Понимания dict синхронизированы с литералами dict, так что ключ вычисляется первым, а значение вторым:

```
>>> # Dict comprehension
>>> cast = {input('role? '): input('actor? ') for i in range(2)}
role? King Arthur
actor? Chapman
role? Black Knight
actor? Cleese

>>> # Dict literal
>>> cast = {input('role? '): input('actor? ')}
role? Sir Robin
actor? Eric Idle
```

Гарантированный порядок выполнения полезен с выражениями назначения, поскольку переменные, назначенные в ключевом выражении, будут доступны в выражении значения:

```
>>> names = ['Martin von Löwis', 'Łukasz Langa', 'Walter Dörwald']
>>> {(n := normalize('NFC', name)).casefold() : n for name in names}
{'martin von löwis': 'Martin von Löwis',
 'łukasz langa': 'Łukasz Langa',
 'walter dörwald': 'Walter Dörwald'}
```

(Представлен J rn Heissler в bpo-35224.)

- Теперь `object.__reduce__()` метод может вернуть кортеж длиной от двух до шести элементов. Раньше предельным было пять. Новый опциональный шестой элемент является вызываемым с `(obj, state)` сигнатура. Это позволяет напрямую управлять поведением обновления состояния определенного объекта. Если не *None*, у этого подлежащего выкупу будет приоритет над `__setstate__()` метод объекта. (Представлен Pierre Glaser и Olivier Grisel в bpo-35900.)

4 Новые модули

- Новый модуль `importlib.metadata` обеспечивает (предварительную) поддержку для чтения метаданных из сторонних пакетов. Например, он может извлечь номер версии установленного пакета, список точек входа и многое другое:

```
>>> # Note following example requires that the popular "requests"
>>> # package has been installed.
>>>
>>> from importlib.metadata import version, requires, files
>>> version('requests')
'2.22.0'
>>> list(requires('requests'))
['chardet (<3.1.0,>=3.0.2)']
>>> list(files('requests'))[:5]
[PackagePath('requests-2.22.0.dist-info/INSTALLER'),
 PackagePath('requests-2.22.0.dist-info/LICENSE'),
 PackagePath('requests-2.22.0.dist-info/METADATA'),
 PackagePath('requests-2.22.0.dist-info/RECORD'),
 PackagePath('requests-2.22.0.dist-info/WHEEL')]
```

(Вклад Barry Warsaw и Jason R. Coombs в bpo-34632.)

5 Улучшение модулей

5.1 ast

Узлы AST теперь имеют атрибуты `end_lineno` и `end_col_offset`, которые дают точное расположение конца узла. (Это только относится к узлам, у которых есть `lineno` и `col_offset` атрибуты.)

Новая функция `ast.get_source_segment()` возвращает источник код для определенного узла AST.

(Представлен Ivan Levkivskyi в bpo-33416.)

Функция `ast.parse()` имеет несколько новых флагов:

- `type_comments=True` приводит к возврату текста комментариев типа **PEP 484** и **PEP 526**, связанных с определенными узлами AST;
- `mode='func_type'` может быть используемый для разбора **PEP 484** «сигнатура type comments» (возвращаемых для определения функции узлы AST);
- `feature_version=(3, N)` позволяет указать более раннюю версию Python 3. Например, `feature_version=(3, 4)` будет рассматривать `async` и `await` как незарезервированные слова.

(Представлен Guido van Rossum в bpo-35766.)

5.2 asyncio

`asyncio.run()` вышел из предварительного в стабильный API. Эта функция может быть используемый, чтобы выполнить корутину и возвратить результат, автоматически управляя событийный цикл. Например:

```
import asyncio

async def main():
    await asyncio.sleep(0)
    return 42

asyncio.run(main())
```

Это *roughly* эквивалентно:

```
import asyncio

async def main():
    await asyncio.sleep(0)
    return 42

loop = asyncio.new_event_loop()
asyncio.set_event_loop(loop)
try:
    loop.run_until_complete(main())
finally:
    asyncio.set_event_loop(None)
    loop.close()
```

Фактическая реализация значительно сложнее. Таким образом, `asyncio.run()` должен быть предпочтительным способом запуска программ asyncio.

(Внесено Yury Selivanov в bpo-32314.)

При запуске `python -m asyncio` запускается асинхронный REPL. Это позволяет быстро экспериментировать с код, который имеет верхний уровень `await`. Больше нет необходимости напрямую вызывать `asyncio.run()`, что породило бы новую событийный цикл при каждом вызове:

```
$ python -m asyncio
asyncio REPL 3.8.0
Use "await" directly instead of "asyncio.run()".
Type "help", "copyright", "credits" or "license" for more information.
>>> import asyncio
>>> await asyncio.sleep(10, result='hello')
hello
```

(Представлен Yury Selivanov в bpo-37028.)

Исключение `asyncio.CancelledError` теперь наследуется от `BaseException`, а не от `Exception`, и больше не наследуется от `concurrent.futures.CancelledError`. (Внесено Юрием Селивановым в bpo-32528.)

На Windows дефолт событийный цикл - теперь `ProactorEventLoop`. (Представлен Victor Stinner в bpo-34687.)

`ProactorEventLoop` теперь также поддерживает UDP. (Предоставлено Adam Meily и Andrew Svetlov в bpo-29883.)

Теперь `ProactorEventLoop` может прерываться командой `KeyboardInterrupt` («CTRL+C»). (Представлен Vladimir Matveev в bpo-23057.)

Добавлено `asyncio.Task.get_coro()` для получения обернутой корутины в пределах `asyncio.Task`. (Вклад Alex Gr nholm в bpo-36999.)

Теперь задачи Asyncio можно именовать либо путем передачи аргумента `name` ключевой в `asyncio.create_task()` или метод `create_task()` событийный цикл, либо путем вызова метода `set_name()` метод в объекте задачи. Имя задачи видимо в продукции `repr()` `asyncio.Task` и может также быть восстановлено, используя `get_name()` метод. (Представлен Alex Gr nholm в bpo-34270.)

Добавлена поддержка Happy Eyeballs в `asyncio.loop.create_connection()`. Для задания поведения были добавлены два новых параметра: *happy_eyeballs_delay* и *interleave*. Алгоритм Happy Eyeballs повышает быстродействие приложений, поддерживающих IPv4 и IPv6, путем одновременной попытки подключения с использованием обоих. (Представлен twisteroid ambassador в bpo-33530.)

5.3 builtins

Встроенное `compile()` было улучшено, чтобы принять флаг `ast.PyCF_ALLOW_TOP_LEVEL_AWAIT`. Если этот новый флаг передан, `compile()` будет разрешать конструкции верхнего уровня `await`, `async for` и `async with`, которые обычно считаются недопустимым синтаксисом. Затем может быть возвращен асинхронный объект код, помеченный флагом `CO_COROUTINE`. (Предоставлено Matthias Bussonnier в bpo-34616)

5.4 collections

Теперь `_asdict()` метод для `collections.namedtuple()` возвращает `dict` вместо `collections.OrderedDict`. Это работает, потому что регулярные документы гарантировали порядок с Python 3.7. Если требуются дополнительные функции `OrderedDict`, предлагаемое исправление должно привести

результат к требуемому типу: `OrderedDict(nt._asdict())`. (Представлен Raymond Hettinger в bpo-35864.)

5.5 cProfile

`cProfile.Profile` класс может теперь быть используемый как менеджером контекст. Выполните профилирование блока код:

```
import cProfile

with cProfile.Profile() as profiler:
      # code to be profiled
      ...
```

(Предоставлено Scott Sanderson в bpo-29235.)

5.6 csv

Теперь `csv.DictReader` возвращает сущности `dict` вместо `collections.OrderedDict`. Инструмент теперь быстрее и использует меньше памяти, сохраняя при этом порядок полей. (Представлен Michael Selik в bpo-34003.)

5.7 curses

Добавлена новая переменная, содержащая информацию о структурированной версии для библиотеки ncurses: `ncurses_version`. (Представлен Serhiy Storchaka в bpo-31680.)

5.8 ctypes

В Windows `CDLL` и подклассы теперь принимают параметр *winmode*, чтобы указать флаги для базового вызова `LoadLibraryEx`. Флаги по умолчанию устанавливаются для загрузки только зависимостей DLL из надежных расположений, включая путь, по которому хранится DLL (если для загрузки начальной DLL-библиотеки используемый полный или частичный путь), и пути, добавляемые `add_dll_directory()`. (Предоставлено Steve Dower в bpo-36085.)

5.9 datetime

Добавлены новые альтернативные конструкторы `datetime.date.fromisocalendar()` и `datetime.datetime.fromisocalendar()`, которые строят объекты `date` и `datetime` соответственно из года ISO, номера недели и дня недели; это обратная величина для каждого класса `isocalendar` метод. (Предоставлено Paul Ganssle в bpo-36004.)

5.10 functools

`functools.lru_cache()` может теперь быть используемый как прямым декоратором, а не как функция, возвратив декоратор. Так что оба они теперь поддерживаются:

```
@lru_cache
def f(x):
    ...

@lru_cache(maxsize=256)
def f(x):
    ...
```

(Представлен Raymond Hettinger в bpo-36772.)

Добавлен новый декоратор `functools.cached_property()` для вычисляемых свойств, кэшированных для жизни сущности.:

```
import functools
import statistics

class Dataset:
   def __init__(self, sequence_of_numbers):
      self.data = sequence_of_numbers

   @functools.cached_property
   def variance(self):
      return statistics.variance(self.data)
```

(Представлен Carl Meyer в bpo-21145)

Добавлен новый декоратор `functools.singledispatchmethod()`, который преобразует методы в общие функции с помощью одиночной диспетчеризации:

```
from functools import singledispatchmethod
from contextlib import suppress

class TaskManager:

    def __init__(self, tasks):
        self.tasks = list(tasks)

    @singledispatchmethod
    def discard(self, value):
        with suppress(ValueError):
            self.tasks.remove(value)

    @discard.register(list)
    def _(self, tasks):
        targets = set(tasks)
        self.tasks = [x for x in self.tasks if x not in targets]
```

(Представлен Ethan Smith в bpo-32380)

5.11 gc

Теперь `get_objects()` может получить необязательный параметр *generation*, указывающий генерацию для получения объектов. (Представлен Pablo Galindo в bpo-36016.)

5.12 gettext

Добавлены `pgettext()` и его варианты. (Вклад Franz Glasner, ric Araujo и Cheryl Sabella в bpo-2504.)

5.13 gzip

Добавлен параметр *mtime* в `gzip.compress()` для воспроизводимого вывода. (Внесён Guo Ci Teo в bpo-34898.)

Исключение `BadGzipFile` теперь поднято вместо `OSError` для определенных типов инвалида или повреждённых gzip файлов. (Вклад Filip Gruszczy ski, Michele Orr и Zackery Spyt в bpo-6584.)

5.14 IDLE и idlelib

Вывод по N линиям (по умолчанию 50) сжимается до кнопки. N можно изменить в разделе PyShell на странице общие диалогового окна настройки. Меньше, но, возможно, слишком длинных линий можно сжать, щелкнув правой кнопкой мыши на выходе. Сжатые выходные данные можно развернуть по месту, дважды нажав кнопку или в буфер обмена или в отдельном окне, щелкнув кнопку правой кнопкой мыши. (Внесен Tal Einat в bpo-1529353.)

Добавьте «Run Customized» в меню Run для запуска модуля с настроенными настройками. Все введенные аргументы командной строки добавляются в sys.argv. Они также повторно появляются в поле для следующего настраиваемого прогона. Можно также подавить обычный перезапуск основного модуля Shell. (Вклад Cheryl Sabella, Terry Jan Reedy и других в bpo-5680 и bpo-37627.)

Добавлены дополнительные номера строк для окон редактора IDLE. Окна открываются без номеров строк, если не указано иное на вкладке «Общие» диалогового окна конфигурации. Номера строк для существующего окна отображаются и скрываются в меню «Параметры». (Предоставлено Tal Einat и Saimadhav Heblikarв bpo-17535.)

Уроженцем OS кодировка является теперь используемый для преобразования между объектами Python строки и Tcl. Это позволяет IDLE работать с эмодзи и другими символами, отличными от BMP. Эти символы могут отображаться или копироваться и вставляться в буфер обмена или из него. Преобразование строки из Tcl в Python и обратно теперь никогда не завершается неудачно. (Многие люди работали над этим восемь лет, но проблема была окончательно решена Serhiy Storchaka в bpo-13153.)

Новое в 3.8.1:

Добавлена опция для отключения мигания курсора. (Внесен Закери Спицем в bpo-4603.)

Клавиша Escape теперь закрывает окна завершения простоя. (Вклад Джонни Наджеры в bpo-38944.)

Вышеприведенные изменения были перенесены обратно в версии технического обслуживания 3.7.

Добавлены ключевые слова в списоковое включение имени модуля. (Вклад Терри Дж. Риди в bpo-37765.)

5.15 inspect

Функция `inspect.getdoc()` может теперь найти докстринги для `__slots__`, если это, атрибут - `dict`, где значения - докстринги. Здесь представлены варианты документации, аналогичные уже имеющимся у нас для `property()`, `classmethod()` и `staticmethod()`:

```
class AudioClip:
    __slots__ = {'bit_rate': 'expressed in kilohertz to one decimal place',
                 'duration': 'in seconds, rounded up to an integer'}
    def __init__(self, bit_rate, duration):
        self.bit_rate = round(bit_rate / 1000.0, 1)
        self.duration = ceil(duration)
```

(Представлен Raymond Hettinger в bpo-36326.)

5.16 io

В режиме разработки (`-X env`) и при отладочной сборке финализатор `io.IOBase` регистрирует исключение в случае сбоя `close()` метод. Исключение по умолчанию игнорируется без предупреждения при построении выпуска. (Представлен Victor Stinner в bpo-18748.)

5.17 itertools

Функция `itertools.accumulate()` добавила аргумент option *initial* ключевой для указания начального значения:

```
>>> from itertools import accumulate
>>> list(accumulate([10, 5, 30, 15], initial=1000))
[1000, 1010, 1015, 1045, 1060]
```

(Внесена Lisa Roach в bpo-34659.)

5.18 json.tool

Добавьте опцию `--json-lines`, чтобы разобрать каждую входную строку как отдельный объект JSON. (Представлен Weipeng Hong в bpo-31553.)

5.19 logging

Добавлен ключевой аргумент *force* `logging.basicConfig()`, когда установлено в истинный, любые существующие обработчики были свойственны корневому логгеру удалены и закрыты перед выполнением конфигурации, определенной другими аргументами.

Это решает давнюю проблему. Как только вызов логгер или *basicConfig()* был вызван, последующие звонки в *basicConfig()* были молча проигнорированы. Это затрудняло обновление, экспериментирование или обучение различных опций конфигурации логирование с помощью интерактивного приглашения или блокнота Jupyter.

(Предложено Raymond Hettinger, реализовано Dong-hee Na и рассмотрено Vinay Sajip в bpo-33897.)

5.20 math

Добавлена новая функция `math.dist()` для вычисления евклидового расстояния между двумя точками. (Представлен Raymond Hettinger в bpo-33089.

Расширена функция `math.hypot()` для обработки нескольких размеров. Ранее она поддерживала только дело 2-D. (Представлен Raymond Hettinger в bpo-33089.

Добавлена новая функция `math.prod()`, аналогичная функции `sum()`, которая возвращает произведение начального значения (по умолчанию: 1), умноженное на интератор чисел:

```
>>> prior = 0.8
>>> likelihoods = [0.625, 0.84, 0.30]
>>> math.prod(likelihoods, start=prior)
0.126
```

(Представлен Pablo Galindo в bpo-35606.

Добавлены две новые комбинаторные функции `math.perm()` и `math.comb()`:

```
>>> math.perm(10, 3)    # Permutations of 10 things taken 3 at a time
720
>>> math.comb(10, 3)    # Combinations of 10 things taken 3 at a time
120
```

(Вклад Yash Aggarwal, Keller Fuchs, Serhiy Storchaka и Raymond Hettinger в bpo-37128, bpo-37178 и bpo-35431.)

Добавлена новая функция `math.isqrt()` для вычисления точных целых квадратных корней без преобразования в плавающую точку. Новая функция поддерживает произвольно большие целые числа. Он быстрее `floor(sqrt(n))`, но медленнее `math.sqrt()`:

```
>>> r = 650320427
>>> s = r ** 2
>>> isqrt(s - 1)         # correct
650320426
>>> floor(sqrt(s - 1))   # incorrect
650320427
```

(Представлен Mark Dickinson в bpo-36887.)

Функция `math.factorial()` больше не принимает аргументы, не похожие на int. (Внесено Pablo Galindo в bpo-33083.)

5.21 mmap

Теперь `mmap.mmap` класс имеет `madvise()` метод для доступа к системному вызову `madvise()`. (Представлен Zackery Spytz в bpo-32941.)

5.22 multiprocessing

Добавлен новый модуль `multiprocessing.shared_memory`. (Представлен Davin Potts в bpo-35813.)

На macOS начинаются *spawn*, метод - теперь используемый по умолчанию. (Представлен Victor Stinner в bpo-33725.)

5.23 os

Добавлена новая функция `add_dll_directory()` в Windows для предоставления дополнительных путей поиска собственных зависимостей при импорте модулей расширений или загрузке DLL с помощью `ctypes`. (Предоставлено Steve Dower в bpo-36085.)

Добавлена новая функция `os.memfd_create()` для переноса `memfd_create()` syscall. (Представлен Zackery Spytz и Christian Heimes в bpo-26836.)

В Windows большая часть ручной логики для обработки точек повторной обработки (включая симссылки и соединения каталогов) была делегирована операционной системе. В частности, теперь `os.stat()` будет пересекать все, что поддерживается операционной системой, в то время как `os.lstat()` будет открывать только точки повторной обработки, которые идентифицируются как «суррогаты имени», в то время как другие открываются как для `os.stat()`. Во всех случаях у `stat_result.st_mode` только будет набор `S_IFLNK` для символических ссылок и не других видов точек переразбора. Чтобы определить другие виды точки переразбора, проверьте новый `stat_result.st_reparse_tag` атрибут.

Теперь в Windows `os.readlink()` может читать узлы каталогов. Обратите внимание, что `islink()` возвратит `False` для директивных соединений, и таким образом, код, который проверяет `islink` сначала, продолжит рассматривать соединения как справочники, в то время как код, который обрабатывает ошибки от `os.readlink()`, может теперь рассматривать соединения как ссылки.

(Предоставлено Steve Dower в bpo-37834.)

5.24 os.path

`os.path` функции, возвращающие логический результат, такие как `exists()`, `lexists()`, `isdir()`, `isfile()`, `islink()` и `ismount()`, теперь возвращают `False` вместо увеличения `ValueError` или

его подклассы `UnicodeEncodeError` и `UnicodeDecodeError` для путей, которые содержат символы или байты, непредставимые на уровне OS. (Представлен Serhiy Storchaka в bpo-33721.)

`expanduser()` в Windows теперь предпочитает переменную среды `USERPROFILE` и не использует `HOME`, которая обычно не устанавливается для обычных учетных записей пользователей. (Представлен Anthony Sottile в bpo-36264.)

`isdir()` в Windows больше не возвращает `True` для ссылки на несуществующий каталог.

`realpath()` в Windows теперь разрешает точки повторной обработки, включая ссылки и соединения каталогов.

(Предоставлено Steve Dower в bpo-37834.)

5.25 pathlib

`pathlib.Path` методы, возвращающие логический результат, такой как `exists()`, `is_dir()`, `is_file()`, `is_mount()`, `is_symlink()`, `is_block_device()`, `is_char_device()`, `is_fifo()`, `is_socket()`, теперь возвращают `False` вместо увеличения `ValueError` или его подкласс `UnicodeEncodeError` для путей, которые содержат символы, непредставимые на уровне OS. (Представлен Serhiy Storchaka в bpo-33721.)

Добавлено `pathlib.Path.link_to()`, создающее жесткую связь, указывающую на контур. (Представлен Joannah Nanjekye в bpo-26978)

5.26 pickle

Расширения `pickle`, подклассирующие C-оптимизированные `Pickler`, теперь могут переопределять логику pickling функций и классы, определяя специальные `reducer_override()` метод. (Вклад Pierre Glaser и Olivier Grisel в bpo-35900.)

5.27 plistlib

Добавленный новый `plistlib.UID` и включил поддержку чтения и написания набора из двух предметов NSKeyedArchiver-кодированный plists. (Представлен Jon Janzen в bpo-26707.)

5.28 pprint

Модуль `pprint` добавил параметр *sort_dicts* к нескольким функциям. По умолчанию эти функции продолжают сортировать словари перед визуализацией или печатью. Однако, если *sort_dicts* установлен в ложный, словари сохраняют порядок, что ключи были вставлены. Это может быть полезно для сравнения с входными данными JSON во время отладки.

Кроме того, есть удобство новая функция, `pprint.pp()`, который похож на `pprint.pprint()`, но с *sort_dicts*, не выполняющим своих обязательств к `False`:

```
>>> from pprint import pprint, pp
>>> d = dict(source='input.txt', operation='filter', destination='output.txt')
>>> pp(d, width=40)                  # Original order
{'source': 'input.txt',
 'operation': 'filter',
 'destination': 'output.txt'}
>>> pprint(d, width=40)              # Keys sorted alphabetically
{'destination': 'output.txt',
 'operation': 'filter',
 'source': 'input.txt'}
```

(Вклад R mi Lapeyre в bpo-30670.)

5.29 py_compile

Теперь `py_compile.compile()` поддерживает бесшумный режим. (Представлен Joannah Nanjekye в bpo-22640.)

5.30 shlex

Новая функция `shlex.join()` действует как инверсия `shlex.split()`. (Представлен Bo Bayles в bpo-32102.)

5.31 shutil

Теперь `shutil.copytree()` принимает новый ключевой аргумент `dirs_exist_ok`. (Предоставлено Josh Bronson в bpo-20849.)

`shutil.make_archive()` теперь не выполняют своих обязательств к современному миру (POSIX.1-2001) формат, чтобы новые архивы улучшили мобильность и соответствие стандартов, унаследованное от соответствующего изменения до модуля `tarfile`. (Автор: C.A.M. Gerlach в bpo-30661.)

`shutil.rmtree()` в Windows теперь удаляет соединения каталогов без рекурсивного удаления их содержимого. (Предоставлено Steve Dower в bpo-37834.)

5.32 socket

Добавлены функции удобства `create_server()` и `has_dualstack_ipv6()` для автоматизации необходимых задач, обычно задействованных при создании серверного сокета, включая прием как IPv4, так и IPv6 подключений на одном и том же сокет. (Автор - Giampaolo Rodol в bpo-17561.)

Функции `socket.if_nameindex()`, `socket.if_nametoindex()` и `socket.if_indextoname()` реализованы в Windows. (Представлен Zackery Spytz в bpo-37007.)

5.33 ssl

Добавленный `post_handshake_auth`, чтобы включить и `verify_client_post_handshake()`, чтобы начать идентификацию TLS 1.3 пострукопожатия. (Представлен Christian Heimes в bpo-34670.)

5.34 statistics

Добавленный `statistics.fmean()` как более быстрый вариант с плавающей запятой `statistics.mean()`. (Представлен Raymond Hettinger и Steven D'Aprano в bpo-35904.)

Добавлено `statistics.geometric_mean()` (внесено Raymond Hettinger в bpo-27181.)

Добавленный `statistics.multimode()`, который возвращает список из наиболее распространенных ценностей. (Представлен Raymond Hettinger в bpo-35892.)

Добавлено `statistics.quantiles()`, которое делит данные или распределение в к настраиваемым интервалам (например, квартили, децили или процентили). (Представлен Raymond Hettinger в bpo-36546.)

Добавленный `statistics.NormalDist`, инструмент для создания и управления нормальными распределениями случайной переменной. (Представлен Raymond Hettinger в bpo-36018.)

```
>>> temperature_feb = NormalDist.from_samples([4, 12, -3, 2, 7, 14])
>>> temperature_feb.mean
6.0
>>> temperature_feb.stdev
6.356099432828281

>>> temperature_feb.cdf(3)           # Chance of being under 3 degrees
0.3184678262814532
>>> # Relative chance of being 7 degrees versus 10 degrees
>>> temperature_feb.pdf(7) / temperature_feb.pdf(10)
1.2039930378537762

>>> el_niño = NormalDist(4, 2.5)
>>> temperature_feb += el_niño       # Add in a climate effect
>>> temperature_feb
NormalDist(mu=10.0, sigma=6.830080526611674)

>>> temperature_feb * (9/5) + 32     # Convert to Fahrenheit
NormalDist(mu=50.0, sigma=12.294144947901014)
>>> temperature_feb.samples(3)       # Generate random samples
[7.672102882379219, 12.000027119750287, 4.647488369766392]
```

5.35 sys

Добавьте новую функцию `sys.unraisablehook()`, которая может быть переопределена для управления обработкой «неразрешимых исключений». Это называют, когда исключение произошло, но нет никакого пути к Python, чтобы обращаться с ним. Например, когда деструктор вызывает исключение или во время сборки мусора (`gc.collect()`). (Представлен Victor Stinner в bpo-36829.)

5.36 tarfile

Модуль `tarfile` теперь по умолчанию использует современный формат pax (POSIX.1-2001) для новых архивов вместо предыдущего формата GNU. Это повышает переносимость между платформами с помощью согласованного кодировка (UTF-8) в стандартизированном и расширяемом формате и предлагает ряд других преимуществ. (Автор: C.A.M. Gerlach в bpo-36268.)

5.37 threading

Добавьте новую функцию `threading.excepthook()`, которая обрабатывает необученное исключение `threading.Thread.run()`. Она может быть переопределена для управления обработкой необнаруженных исключений `threading.Thread.run()`. (Представлен Victor Stinner в bpo-1230540.)

Добавьте новую функцию `threading.get_native_id()` и функцию `native_id` атрибут к `threading.Thread` класс. Они возвращают собственный интегральный идентификатор потока текущего поток, назначенного ядром. Эта функция доступна только на определенных платформах, см. раздел `get_native_id` для получения дополнительной информации. (Представлен Jake Tesler в bpo-36084.)

5.38 tokenize

Модуль `tokenize` теперь неявно выдает токен `NEWLINE`, когда ему предоставляется вход, который не имеет конечной новой строки. Это поведение теперь совпадает с тем, что делает внутренний токенизатор C. (Представлен Ammar Askar в bpo-33899.)