Установка и использование Python

Выпуск 3.8.8

Гвидо ван Россум
и команда разработчиков Python

августа 12, 2021

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

Оглавление

Эта часть документации посвящена общей информации о настройке среды Python на различных платформах, вызове интерпретатора и вещах, которые облегчают работу с Python.

ГЛАВА 1

Командная строка и окружение

Интерпретатор CPython просматривает командную строку и окружение для различных параметров настройки.

Детали реализации CPython: Схемы командной строки других реализаций могут отличаться. Дополнительные ресурсы см. в разделе implementations.

1.1 Командная строка

При вызове Python можно указать любой из этих параметров:

```
python [-bBdEhiIOqsSuvVWx?] [-c command | -m module-name | script | - ] [args]
```

Самый распространенный вариант использования — это, конечно, простой вызов сценария:

```
python myscript.py
```

1.1.1 Интерфейсные опции

Интерфейс интерпретатора напоминает шелл интерфейс UNIX, но предоставляет некоторые дополнительный методы вызова:

- При вызове со стандартным входом, подключенным к устройству tty, он запрашивает команды и выполняет их до тех пор, пока не будет прочитан EOF (символ конца файла, который можно создать с помощью `Ctrl-D` в UNIX или `Ctrl-Z, Enter` в Windows).
- При вызове с аргументом имени файла или с файлом в качестве стандартного ввода он считывает и выполняет сценарий из этого файла.
- При вызове с аргументом имени каталога он считывает и выполняет сценарий с соответствующим именем из этого каталога.
- При вызове с помощью `-c command` он выполняет оператор(ы) Python, заданные как *command*. Здесь *command* может содержать несколько операторов, разделенных новыми строками. Ведущий пробел важен в Python операторах!

- При вызове с `-m module-name` модуль выполняется как сценарий, при условии что он доступен.

В неинтерактивном режиме весь ввод анализируется перед его выполнением.

Параметр интерфейса завершает список опций, используемых интерпретатором, все последовательные аргументы заканчиваются на `sys.argv`. Обратите внимание, что первый элемент — нижний нулевой индекс (`sys.argv[0]`), является строкой, отражающей источник программы.

`-c` <command>
: Выполнить команду Python кода в *command*. *command* может быть одним или несколькими операторами, разделенными новыми строками, с важным начальным пробелом, как в обычном модуле кода.

  Если параметр задан, первый элемент `sys.argv` будет `"-c"`, а текущий каталог будет добавлен к началу `sys.path` (что позволит импортировать модули в этот каталог как модули верхнего уровня).

  Raises an auditing event `cpython.run_command` with argument `command`.

`-m` <module-name>
: Ищет в `sys.path` вызываемый модуль и выполняет его содержание как модуль `__main__`.

  Т. к. аргумент — имя *module*, вы не должны передавать расширение файла (`.py`). Имя модуля должно быть действительным абсолютным именем модуля Python, но реализация может не всегда воплотить это в жизнь (например, она может позволить вам использовать имя, которое включает дефис).

  Также разрешены имена пакетов (включая пакеты пространства имён). Когда имя пакета будет поставляться вместо нормального модуля, интерпретатор выполнит `<pkg>.__main__` как главный модуль. Это поведение намеренно аналогично обработке каталогов и zipfiles, которые передаются интерпретатору в качестве аргумента сценария.

  Примечание: Этот параметр не может использоваться со встроенными модулями и модулями расширений, написанными на языке C, поскольку они не имеют файлов модулей Python. Однако это может использоваться для предварительно собранных модулей, даже если файл первоисточника не доступен.

  Если эта опция задана, первым элементом `sys.argv` будет полный путь к файлу модуля (во время нахождения файла модуля первый элемент будет установлен в `"-m"`). Как и в случае с опцией *-c*, текущий каталог будет добавлен к началу `sys.path`.

  Выбор *-I* может использоваться, чтобы управлять сценарием в изолированном режиме, где `sys.path` не содержит ни текущего каталога, ни справочника site-packages пользователя. Все переменные среды `PYTHON*` также игнорируются.

  Многие стандартные библиотечные модули содержат код, вызываемый при выполнении в качестве сценария. Примером может служить модуль `timeit`:

```
python -m timeit -s 'setup here' 'benchmarked code here'
python -m timeit -h # for details
```

  Raises an auditing event `cpython.run_module` with argument `module-name`.

  См.также:

  `runpy.run_module()` Эквивалентные функциональные возможности, непосредственно доступные для Python кода

PEP 338 – выполнение модулей в виде сценариев

Изменено в версии 3.1: Укажите имя пакета для запуска подмодуля `__main__`.

Изменено в версии 3.4: Поддерживаются также пакеты пространства имён

-

Считывание команд со стандартного ввода (`sys.stdin`). Если стандартный ввод является терминалом, то подразумевается *-i*.

Если этот параметр задан, первый элемент `sys.argv` будет `"-"`, а текущий каталог будет добавлен к началу `sys.path`.

Raises an auditing event `cpython.run_stdin` with no arguments.

<script>

Выполнить Python код, содержащийся в *script*, который должен быть путём файловой системы (абсолютным или относительным) со ссылкой на файл Python, каталог, содержащий файл `__main__.py`, или zipfile, содержащий файл `__main__.py`.

Если этот параметр задан, первым элементом `sys.argv` будет имя сценария, указанное в командной строке.

Если имя сценария ссылается непосредственно на файл Python, каталог, содержащий этот файл, добавляется к началу `sys.path`, и файл выполняется как модуль `__main__`.

Если имя сценария ссылается на каталог или zipfile, имя сценария добавляется к началу `sys.path`, и файл `__main__.py` в этом расположении выполняется как модуль `__main__`.

Опция *-I* может использоваться, чтобы управлять сценарием в изолированном режиме, где `sys.path` не содержит ни каталогов сценарных, ни каталога site-packages пользователя. Все переменные среды `PYTHON*` также игнорируются.

Raises an auditing event `cpython.run_file` with argument `filename`.

См.также:

`runpy.run_path()` эквивалентные функциональные возможности, непосредственно доступные для Python кода

Если параметр интерфейса не задан, подразумевается *-i*, `sys.argv[0]` является пустой строкой (`""`) и текущий каталог будет добавлен к началу `sys.path`. Кроме того, автоматически включаются функции автодополнения табов и редактирования истории, если они доступны на вашей платформе (см. rlcompleter-config).

См.также:

tut-invoking

Изменено в версии 3.4: Автоматическое включение автодополнения табов и редактирования истории.

1.1.2 Универсальные опции

-?
-h
--help

Напечатать краткое описание всех параметров командной строки.

-V
--version

Напечатать номер версии Python и выйти. Примером вывода может быть:

```
Python 3.8.0b2
```

При двойном вводе напечатает дополнительную информацию о сборке, например:

```
Python 3.8.0b2+ (3.8:0c076caaa8, Apr 20 2019, 21:55:00)
[GCC 6.2.0 20161005]
```

Добавлено в версии 3.6: Опция `-VV`.

1.1.3 Прочие опции

`-b`

Выдать предупреждение при сравнении `bytes` или `bytearray` с `str` или `bytes` с `int`. Выдать ошибку, если опция задана дважды (`-bb`).

Изменено в версии 3.5: Влияет на сравнение `bytes` с `int`.

`-B`

Если задано, Python не будет пытаться записать файлы `.pyc` при импорте исходных модулей. См. также раздел *PYTHONDONTWRITEBYTECODE*.

`--check-hash-based-pycs` `default|always|never`

Управляет поведением проверки файлов `.pyc` на основе хэшей. См. pyc-invalidation. Если установлено значение `default`, проверенные и непроверенные файлы кэша байт-кода на основе хеша проверяются в соответствии с их семантикой по умолчанию. Если установлено значение `always`, все файлы `.pyc` на основе хэшей, независимо от того, отмечены они или нет, проверяются на соответствие их соответствующему исходному файлу. Если задано значение `never`, файлы `.pyc` на основе хэшей не проверяются на соответствие их соответствующим исходным файлам.

Этот параметр не влияет на семантику файлов `.pyc` на основе временных меток.

`-d`

Включить отладочный вывод парсера (только для эксперта, в зависимости от вариантов компиляции). См. также раздел *PYTHONDEBUG*.

`-E`

Игнорировать все переменные среды `PYTHON*`, например, *PYTHONPATH* и *PYTHONHOME*, которые могут быть установлены.

`-i`

Когда сценарий передаётся в качестве первого аргумента или используется параметр *-c*, после выполнения сценария или команды, даже если `sys.stdin` не является терминалом, следует перейти в интерактивный режим. Файл *PYTHONSTARTUP* не считан.

Может быть полезно для проверки глобальных переменных или трассировки стека, когда сценарий вызывает исключение. См. также раздел *PYTHONINSPECT*.

`-I`

Запустить Python в изолированном режиме. Также подразумевает -E и -s. В изолированном режиме `sys.path` не содержит ни каталог сценария, ни каталог site-packages пользователя. Все переменные среды `PYTHON*` также игнорируются. Дополнительные ограничения могут быть введены для предотвращения ввода пользователем вредоносного кода.

Добавлено в версии 3.4.

`-O`

Удалить операторы assert и любой код, обусловленный значением `__debug__`. Увеличить имя файла для скомпилированных (*байт-код*) файлов, добавив `.opt-1` перед расширением `.pyc` (см. **PEP 488**). См. также *PYTHONOPTIMIZE*.

Изменено в версии 3.5: Изменены имена файлов `.pyc` согласно **PEP 488**.

-OO

Сделать *-O*, а также отбросить строки документации. Увеличить имя файла для скомпилированных (*байт-код*) файлов, добавив `.opt-2` перед расширением `.pyc` (см. **PEP 488**).

Изменено в версии 3.5: Изменены имена файлов `.pyc` согласно **PEP 488**.

-q

Не отображать сообщения об авторских правах и версии даже в интерактивном режиме.

Добавлено в версии 3.2.

-R

Включить хеш-рандомизацию. Эта опция действует только в том случае, если для переменной среды *PYTHONHASHSEED* установлено значение `0`, поскольку по умолчанию хеш-рандомизация включена.

В предыдущих версиях Python этот параметр включает рандомизацию хеширования, так что значения `__hash__()` объектов str и bytes «засолены» непредсказуемым случайным значением. Хотя они остаются постоянными в рамках отдельного процесса Python, они не предсказуемы между повторными вызовами Python.

Рандомизация хеширования предназначена для обеспечения защиты от отказа в обслуживании, вызванного тщательно подобранными вводами, которые используют наихудшую производительность конструкции dict, сложность O(n^2). Дополнительные сведения см. на сайте.

PYTHONHASHSEED позволяет вам установить фиксированное значение для секретного хеш-кода.

Изменено в версии 3.7: Параметр больше не игнорируется.

Добавлено в версии 3.2.3.

-s

Не добавлять `каталог site-packages пользователя` в `sys.path`.

См.также:

PEP 370 – каталог site-packages для каждого пользователя

-S

Отключить импорт модуля `site` и связанные с ним манипуляции с `sys.path`, зависящие от сайта. Также отключить эти манипуляции, если `site` будет явно импортирован позже (вызовите `site.main()`, если вам нужно чтобы они запускались).

-u

Принудительно отключить буферизацию потоков stdout и stderr. Этот параметр не влияет на поток stdin.

См. также *PYTHONUNBUFFERED*.

Изменено в версии 3.7: Текстовый слой потоков stdout и stderr теперь не буферизован.

-v

Печать сообщения при каждой инициализации модуля с указанием места (имени файла или встроенного модуля), откуда оно загружается. Дважды (`-vv`) распечатает сообщение для каждого файла, который проверяется при поиске модуля. Также содержит информацию о очистке модуля на выходе. См. также раздел *PYTHONVERBOSE*.

-W `arg`

Управление предупреждением. Python механизм предупреждения по умолчанию печатает предупреждающие сообщения в `sys.stderr`. У типичного предупреждающего сообщения есть следующая форма:

```
file:line: category: message
```

По умолчанию каждое предупреждение печатается один раз для каждой исходной строки, где оно происходит. Этот параметр управляет частотой печати предупреждений.

Может быть указано несколько опций *-W*; когда предупреждение соответствует более чем одному параметру, выполняется действие для последнего совпадающего параметра. Недопустимые параметры *-W* игнорируются (хотя при первом предупреждении печатается предупреждающее сообщение о недопустимых параметрах).

Предупреждениями также можно управлять с помощью переменной среды *PYTHONWARNINGS* и из программы Python с помощью модуля `warnings`.

В простейших настройках определенное действие безоговорочно применяется ко всем предупреждениям, выдаваемым процессом (даже к тем, которые по умолчанию игнорируются):

```
-Wdefault  # Предупреждать один раз для каждого местоположения вызова
-Werror    # Преобразовать в исключения
-Walways   # Предупреждать каждый раз
-Wmodule   # Предупреждать один раз на каждый вызывающий модуль
-Wonce     # Предупреждать один раз для каждого процесса Python
-Wignore   # Никогда не предупреждать
```

Имена действий могут быть сокращены по желанию (например, `-Wi`, `-Wd`, `-Wa`, `-We`), и интерпретатор преобразует их в соответствующее имя действия.

Дополнительные сведения см. в разделах warning-filter и describing-warning-filters.

-x

Пропустить первую строку исходника, разрешив использование не Unix-форм `#!cmd`. Это предназначено только для хака DOS.

-X

Зарезервирован для различных опций реализации. CPython в настоящее время определяет следующие возможные значения:

- `-X faulthandler` для включения `faulthandler`;
- `-X showrefcount` для вывода общего счетчика ссылок и количества использованных блоков памяти при завершении программы или после каждого оператора в интерактивном интерпретаторе. Работает только в отладочных сборках.
- `-X tracemalloc``для начаа отслеживание распределения памяти Python с помощью модуля :mod:`tracemalloc`. По умолчанию в traceback трассировки сохраняется только самый последний фрейм. Используйте ``-X tracemalloc=NFRAME`, чтобы начать трассировку с пределом трассировки фреймов NFRAME. См. `tracemalloc.start()` для получения дополнительной информации.
- `-X showalloccount` для вывода общего количества выделенных объектов для каждого типа по завершении программы. Работает только, когда Python был собран с определенным `COUNT_ALLOCS`.
- `-X importtime`, чтобы показать, сколько времени занимает каждый импорт. Он показывает имя модуля, совокупное время (включая вложенный импорт) и собственное время (исключая вложенный импорт). Следует отметить, что его вывод может быть нарушен в многопоточном приложении. Типичное использование - `python3 -X importtime -c 'import asyncio'`. См. такж раздел *PYTHONPROFILEIMPORTTIME*.
- `-X dev`: включить «режим разработки» CPython, введя дополнительные проверки среды выполнения, которые слишком дороги для включения по умолчанию. Он не должен быть более

подробным, чем значение по умолчанию, если код является правильным: новые предупреждения выдаются только при обнаружении проблемы. Эффект от режима разработчика:

- Добавьте фильтр предупреждения `default` как *-W* `default`.
- Установить отладочные хуки на распределители памяти: см. функцию `PyMem_SetupDebugHooks()` C.
- Включить модуль `faulthandler` для сброса Python трейсбэка в случае сбоя.
- Включить asyncio режим отладки.
- Установить `dev_mode` атрибут `sys.flags` `True`.
- `io.IOBase` деструктор регистрирует исключения `close()`.

- `-X utf8` включает режим UTF-8 для интерфейсов операционной системы, переопределяя режим с учётом языковых стандартов по умолчанию. `-X utf8=0` явно отключает режим UTF-8 (даже если в противном случае он активируется автоматически). Дополнительные сведения см. в разделе *PYTHONUTF8*.
- `-X pycache_prefix=PATH` позволяет записывать файлы `.pyc` в параллельное дерево, коренящееся в предоставленном каталоге, а не в кодовом дереве. См. также раздел *PYTHONPYCACHEPREFIX*.

Также позволяет передавать произвольные значения и извлекать их через словарь `sys._xoptions`.

Изменено в версии 3.2: Добавлен параметр *-X*.

Добавлено в версии 3.3: Опция `-X faulthandler`.

Добавлено в версии 3.4: Параметры `-X showrefcount` и `-X tracemalloc`.

Добавлено в версии 3.6: Опция `-X showalloccount`.

Добавлено в версии 3.7: Опции `-X importtime`, `-X dev` и `-X utf8`.

Добавлено в версии 3.8: Опция `-X pycache_prefix`. Опция `-X dev` теперь регистрирует исключения `close()` в деструкторе `io.IOBase`.

1.1.4 Опции, которые вы не должны использовать

`-J`
Зарезервировано для использования Jython.

1.2 Переменные окружения

Переменные среды влияют на поведение Python, они обрабатываются перед переключателями командной строки, отличными от -E или -I. Обычно переключатели командной строки переопределяют переменные среды в случае конфликта.

`PYTHONHOME`
Изменить расположение стандартных библиотек Python. По умолчанию библиотеки ищутся в *prefix*`/lib/python`*version* и *exec_prefix*`/lib/python`*version*, где *prefix* и *exec_prefix* — это каталоги, зависящие от установки, оба по умолчанию — `/usr/local`.

Если для *PYTHONHOME* задан один каталог, его значение заменяет как *prefix*, так и *exec_prefix*. Чтобы указать для них разные значения, установить *PYTHONHOME* на *prefix*`:`*exec_prefix*.

PYTHONPATH

Увеличить путь поиска по умолчанию для файлов модулей. Формат совпадает с форматом PATH оболочки: одно или несколько путей каталога, разделенных `os.pathsep` (например, двоеточия в Unix или точка с запятой в Windows). Несуществующие каталоги автоматически игнорируются.

В дополнение к обычным каталогам, отдельные записи *PYTHONPATH* могут относиться к zip-файлам, содержащим чистые модули Python (в исходной или скомпилированной форме). Модули расширений нельзя импортировать из zip-файлов.

Путь поиска по умолчанию зависит от установки, но обычно начинается с *prefix*`/lib/pythonversion` (см. *PYTHONHOME* выше). Он *всегда* добавляется к *PYTHONPATH*.

Дополнительный каталог будет вставлен в путь поиска перед *PYTHONPATH*, как описано выше в разделе *Интерфейсные опции*. Путь поиска можно изменить из программы Python с помощью переменной `sys.path`.

PYTHONSTARTUP

Если это имя читаемого файла, команды Python в этом файле выполняются до отображения первого приглашения в интерактивном режиме. Файл выполняется в том же пространстве имён, в котором выполняются интерактивные команды, так что определенные или импортированные в нём объекты могут использоваться без квалификации в интерактивном сеансе. Вы также можете изменить подсказки `sys.ps1` и `sys.ps2` и хук `sys.__interactivehook__` в этом файле.

Вызывает событие аудита `cpython.run_startup` с именем файла в качестве аргумента при запуске.

PYTHONOPTIMIZE

Если задана непустая строка, это эквивалентно указанию параметра *-O*. Если установлено целое число, это эквивалентно многократному указанию *-O*.

PYTHONBREAKPOINT

Если установлена, она именует вызываемый объект, используя нотацию пути с точками. Модуль, содержащий вызываемый объект, будет импортирован, а затем вызываемый объект будет запущен реализацией по умолчанию `sys.breakpointhook()`, которая сама вызывается встроенным `breakpoint()`. Если не задана или установлена пустая строка, она эквивалентен значению «pdb.set_trace». Установка значения в строку «0» приводит к тому, что реализация по умолчанию `sys.breakpointhook()` ничего не делает, кроме немедленного возврата.

Добавлено в версии 3.7.

PYTHONDEBUG

Если для неё задана непустая строка, это эквивалентно указанию параметра *-d*. Если установлено целое число, это эквивалентно многократному указанию *-d*.

PYTHONINSPECT

Если установлена в непустую строку, это эквивалентно определению опции *-i*.

Эта переменная также может быть изменена кодом Python с использованием `os.environ` для принудительного режима проверки при завершении программы.

PYTHONUNBUFFERED

Если для неё задана непустая строка, это эквивалентно указанию параметра *-u*.

PYTHONVERBOSE

Если для неё задана непустая строка, это эквивалентно указанию параметра *-v*. Если установлено целое число, это эквивалентно многократному указанию *-v*.

PYTHONCASEOK

Если установлена, Python игнорирует регистр в операторах `import`. Это работает только в Windows и OS X.

PYTHONDONTWRITEBYTECODE
Если для этого параметра установлена непустая строка, Python не будет пытаться записывать файлы `.pyc` при импорте исходных модулей. Это эквивалентно указанию параметра *-B*.

PYTHONPYCACHEPREFIX
Если установлено, Python будет записывать файлы `.pyc` в зеркальном дереве каталогов по этому пути, а не в каталогах `__pycache__` в исходном дереве. Это эквивалентно указанию опции *-X* `pycache_prefix=PATH`.

Добавлено в версии 3.8.

PYTHONHASHSEED

Если переменная не установлена или содержит значение `random`, для заполнения хэшей объектов str и bytes используется случайное значение.

Если для *PYTHONHASHSEED* задано целочисленное значение, оно используется в качестве фиксированного начального числа для генерации hash() типов, охватываемых рандомизацией хэша.

Её цель — разрешить повторяемое хеширование, например, для самотестирования самого интерпретатора, или позволить кластеру процессов Python совместно использовать хеш-значения.

Целое число должно быть десятичным числом в диапазоне [0,4294967295]. Указание значения 0 отключит рандомизацию хэша.

Добавлено в версии 3.2.3.

PYTHONIOENCODING
Если она установлена до запуска интерпретатора, то переопределяет кодировку, используемую в stdin/stdout/stderr, в синтаксисе `encodingname:errorhandler`. И `encodingname`, и `:errorhandler` не являются обязательными и содержат то же значение, что и `str.encode()`.

Для stderr часть `:errorhandler` игнорируется; обработчик всегда будет `'backslashreplace'`.

Изменено в версии 3.4: Часть `encodingname` теперь не является обязательной.

Изменено в версии 3.6: Кодировка в Windows, указанная этой переменной, игнорируется для буферов интерактивной консоли, если также не указана *PYTHONLEGACYWINDOWSSTDIO*. Файлы и каналы, перенаправленные через стандартные потоки, не затрагиваются.

PYTHONNOUSERSITE
Если установлено, Python не будет добавлять `пользовательский каталог site-packages` в `sys.path`.

См.также:

PEP 370 — каталог site-packages для каждого пользователя

PYTHONUSERBASE
Определяет `базыовый каталог пользователей`, используемый для вычисления пути `пользовательской директории site-packages` и пути установки Distutils для `python setup.py install --user`.

См.также:

PEP 370 — каталог site-packages для каждого пользователя

PYTHONEXECUTABLE
Если переменная среды установлена, для `sys.argv[0]` будет установлено её значение вместо значения, полученного через среду выполнения C. Работает только в Mac OS X.

PYTHONWARNINGS

Эквивалент опции *-W*. Если задана строка, разделенная запятыми, это эквивалентно многократному указанию *-W*, при этом фильтры, расположенные позже в списке, приоритетнее над фильтрами ранее в списке.

В простейших настройках определенное действие безоговорочно применяется ко всем предупреждениям, выдаваемым процессом (даже к тем, которые по умолчанию игнорируются):

```
PYTHONWARNINGS=default  # Предупреждать один раз для каждого
↪местоположения вызова
PYTHONWARNINGS=error    # Преобразовать в исключения
PYTHONWARNINGS=always   # Предупреждать каждый раз
PYTHONWARNINGS=module   # Предупреждать один раз для каждого вызывающего
↪модуля
PYTHONWARNINGS=once     # Предупреждать один раз для каждого процесса
↪Python
PYTHONWARNINGS=ignore   # Никогда не предупреждать
```

Подробнее см. warning-filter и describing-warning-filters.

PYTHONFAULTHANDLER

Если для этой переменной среды задана непустая строка, при запуске вызывается `faulthandler.enable()`: установить обработчик для сигналов `SIGSEGV`, `SIGFPE`, `SIGABRT`, `SIGBUS` и `SIGILL` для дампа трассировки Python. Это эквивалентно опции *-X* `faulthandler`.

Добавлено в версии 3.3.

PYTHONTRACEMALLOC

Если для этой переменной среды задана непустая строка, начинать отслеживать выделение памяти Python с помощью модуля `tracemalloc`. Значение переменной — это максимальное количество фреймов, сохраняемых в обратной трассировке trace. Например, `PYTHONTRACEMALLOC=1` хранит только самый последний фрейм. См. `tracemalloc.start()` для получения дополнительной информации.

Добавлено в версии 3.4.

PYTHONPROFILEIMPORTTIME

Если для переменной среды задана непустая строка, Python покажет, сколько времени занимает каждый импорт. Это в точности эквивалентно настройке `-X importtime` в командной строке.

Добавлено в версии 3.7.

PYTHONASYNCIODEBUG

Если для этой переменной среды задана непустая строка, включить режим отладки модуля `asyncio`.

Добавлено в версии 3.4.

PYTHONMALLOC

Установить распределители памяти Python и/или установить отладочные хуки.

Установить семейство распределителей памяти, используемых Python:

- `default`: использовать распределители памяти по умолчанию.
- `malloc`: использовать функцию `malloc()` библиотеки C для всех доменов (`PYMEM_DOMAIN_RAW`, `PYMEM_DOMAIN_MEM`, `PYMEM_DOMAIN_OBJ`).
- `pymalloc`: использовать pymalloc распределитель для доменов `PYMEM_DOMAIN_MEM` и `PYMEM_DOMAIN_OBJ` и использовать функцию `malloc()` для домена `PYMEM_DOMAIN_RAW`.

Установить отладочные хуки:

- `debug`: установить хуки отладки поверх распределителей памяти по умолчанию.
- `malloc_debug`: то же, что и `malloc`, но с установкой отладочных хуков.
- `pymalloc_debug`: то же, что и `pymalloc`, но с установкой отладочных хуков.

См. распределителей памяти по умолчанию и функцию `PyMem_SetupDebugHooks()` (установить отладочные хуки на распределителях памяти Python).

Изменено в версии 3.7: Добавлен распределитель `"default"`.

Добавлено в версии 3.6.

PYTHONMALLOCSTATS

Если установлена непустая строка, Python будет печатать статистику pymalloc распределителя памяти каждый раз, когда создаётся новая объектная арена pymalloc, а также при завершении работы.

Эта переменная игнорируется, если переменная среды *PYTHONMALLOC* используется для принудительного использования распределителя `malloc()` библиотеки C или если Python настроен без поддержки `pymalloc`.

Изменено в версии 3.6: Переменную теперь также можно использовать в Python, скомпилированном в режиме релиза. Теперь она не действует, если определена пустой строкой.

PYTHONLEGACYWINDOWSFSENCODING

Если задана непустая строка, кодировка файловой системы по умолчанию и режим ошибок вернутся к своим значениям до 3.6 «mbcs» и «replace» соответственно. В противном случае используются новые значения по умолчанию «utf-8» и «surrogatepass».

Также можно включить во время выполнения с `sys._enablelegacywindowsfsencoding()`.

Availability: Windows.

Добавлено в версии 3.6: См. **PEP 529** для получения более подробной информации.

PYTHONLEGACYWINDOWSSTDIO

Если задана непустая строка, новые средства чтения и записи консоли не используются. Это означает, что Юникод символы будут кодироваться в соответствии с активной кодовой страницей консоли, а не с использованием utf-8.

Эта переменная игнорируется, если стандартные потоки перенаправляются (в файлы или каналы), а не ссылаются на буферы консоли.

Availability: Windows.

Добавлено в версии 3.6.

PYTHONCOERCECLOCALE

Если установлено значение `0`, основное приложение командной строки Python пропускает приведение устаревших локалей C и POSIX на основе ASCII к более функциональной альтернативе на основе UTF-8.

Если эта переменная *не* установлена (или ей присвоено значение, отличное от `0`), переменная среды переопределения локали `LC_ALL` также не устанавливается, а текущая локаль, указанная для категории `LC_CTYPE`, является либо локалью по умолчанию `C`, либо явно ASCII-основанной локали `POSIX`, то Python CLI попытается настроить следующие локали для категории `LC_CTYPE` в порядке, указанном перед загрузкой среды выполнения интерпретатора:

- `C.UTF-8`
- `C.utf8`
- `UTF-8`

Если установка одной из этих категорий локали прошла успешно, то переменная среды `LC_CTYPE` также будет соответственно установлена в текущей среде процесса перед инициализацией среды выполнения Python. Это гарантирует, что обновленный параметр будет виден как самому интерпретатору, так и другим компонентам, зависящим от локали, работающим в одном процессе (например, библиотеке GNU `readline`), и в подпроцессах (независимо от того, являются ли эти процессы запущенными интерпретатором Python или нет), а также в операциях, которые запрашивают среду, а не текущую локаль C (например, собственная Python `locale.getdefaultlocale()`).

Настройка одной из этих локалей (явно или через указанное выше неявное принуждение языковых стандартов) автоматически включает `surrogateescape` обработчик ошибки для `sys.stdin` и `sys.stdout` (`sys.stderr` продолжает использовать `backslashreplace`, как и в любой другой локали). Это поведение обработки потока может быть изменено, как обычно, с помощью *`PYTHONIOENCODING`*.

Для целей отладки установка `PYTHONCOERCECLOCALE=warn` приведёт к тому, что Python будет выдавать предупреждающие сообщения на `stderr`, если либо активируется принуждение локали, либо если языковой стандарт, который мог бы вызвать приведение, всё ещё активен при инициализации среды выполнения Python.

Также обратите внимание, что даже если принуждение локали отключено или не удается найти подходящую целевую локаль, *`PYTHONUTF8`* по-прежнему будет активироваться по умолчанию в устаревших языковых стандартах на основе ASCII. Обе функции должны быть отключены, чтобы интерпретатор использовал `ASCII` вместо `UTF-8` для системных интерфейсов.

Availability: *nix.

Добавлено в версии 3.7: См. **PEP 538** для получения более подробной информации.

`PYTHONDEVMODE`

Если для этой переменной среды задана непустая строка, включается «режим разработки» CPython. См. опцию *`-X`* `dev`.

Добавлено в версии 3.7.

`PYTHONUTF8`

Если установлено значение `1`, включает режим интерпретатора UTF-8, где `UTF-8` используется в качестве кодировки текста для системных интерфейсов, независимо от текущей настройки локали.

Это значит, что:

- `sys.getfilesystemencoding()` возвращает `'UTF-8'` (кодировка локали игнорируется).
- `locale.getpreferredencoding()` возвращает `'UTF-8'` (кодировка локали игнорируется, а параметр функции `do_setlocale` не действует).
- `sys.stdin`, `sys.stdout` и `sys.stderr` все используют UTF-8 в качестве кодировки текста, при этом `surrogateescape` обработчик ошибки включён для `sys.stdin` и `sys.stdout` (`sys.stderr` продолжает использовать `backslashreplace`, как и в режиме с учётом локали по умолчанию)

Вследствие изменений в этих API нижнего уровня другие API более высокого уровня также демонстрируют другое поведение по умолчанию:

- Аргументы командной строки, переменные среды и имена файлов декодируются в текст с использованием кодировки UTF-8.
- `os.fsdecode()` и `os.fsencode()` используют кодировку UTF-8.
- `open()`, `io.open()` и `codecs.open()` по умолчанию используют кодировку UTF-8. Однако они по-прежнему используют строгий обработчик ошибок по умолчанию, так что попытка открыть двоичный файл в текстовом режиме может вызвать исключение, а не создавать бессмысленные данные.

Обратите внимание, что стандартные настройки потока в режиме UTF-8 могут быть отменены *PYTHONIOENCODING* (так же, как они могут быть в режиме с учётом локали по умолчанию).

Если установлено значение `0`, интерпретатор работает в режиме с учётом локали по умолчанию.

Установка любой другой непустой строки вызывает ошибку при инициализации интерпретатора.

Если эта переменная среды вообще не задана, то интерпретатор по умолчанию использует текущие настройки локали, если текущая локаль *не определена* как устаревшая локаль на основе ASCII (как описано для *PYTHONCOERCECLOCALE*), и принуждение локали отключено или не работает. В таких устаревших региональных стандартах интерпретатор по умолчанию включает режим UTF-8, если явно не указано иное.

Также доступена как опция *-X* `utf8`.

Добавлено в версии 3.7: См. **PEP 540** для получения более подробной информации.

1.2.1 Переменные режима отладки

Установка этих переменных влияет только на отладочную сборку Python.

PYTHONTHREADDEBUG
Если установлена, Python будет печатать отладочную информацию о потоках.

Нужен Python, настроенный с опцией сборки `--with-pydebug`.

PYTHONDUMPREFS
Если установлено, Python будет сбрасывать объекты и счётчики ссылок, все ещё живые после завершения работы интерпретатора.

Нужен Python, настроенный с опцией сборки `--with-trace-refs`.