

ХОЧУ ПОМОЧЬ
ПРОЕКТУ

Практикум

Fullstack-разработчик

Я

РАБОТАЕТ НА
ТЕХНОЛОГИЯХ
ЯНДЕКСА

practicum.yandex.ru

Станьте Fullstack-разработчиком на курсе от Яндекса.

Программа 2023 года. Диплом о переподготовке и помощь с поиском работы. Рассрочка.

Узнать больше

РЕКЛАМА · 16+

Способы запуска скриптов Python

К Python, параметры и опции

При вызове Python [сканирует командную строку и переменную среду](#) с целью настройки параметров

Вызов параметров других реализаций могут отличаться. Смотрите [альтернативные реализации](#) языка

Python вызов параметров запуска конкретной реализации.

При вызове Python можно указать любой из следующих параметров:

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

Напишем и сохраним в файл test.py небольшую программку, которую будем запускать разными способами.

```
#!/usr/bin/env python3

def hello(text):
    print(f'Функция "{text}"')

print('Привет Мир')

if __name__ == '__main__':
    hello('hello')
```

Самым распространенным вариантом использования, конечно, является простой вызов скрипта:

```
$ python3 test.py
# Привет Мир
# Функция "hello"
```

Содержание:

- [Перенаправление вывода программы;](#)
- [Запуск скрипта как модуля с параметром -m;](#)
- [Запуск скрипта в интерактивном режиме с параметром -i;](#)
- [Использование имени файла скрипта;](#)
- [Запуск скрипта при выполнении импорта;](#)
- [Использование модуля importlib;](#)
- [Использование модуля runpy;](#)
- [Использование функции exec\(\).](#)

Перенаправление вывода программы.


Иногда бывает полезно сохранить вывод скрипта для последующего анализа. Вот как это сделать:

```
$ python3 test.py > output.txt
```

Вверх

Эта операция сохраняет вывод скрипта в файл `output.txt`, а не в стандартный системный вывод `stdout`. В результате на экран попадѣт только поток ошибок `stderr` (в случае их появления). Этот процесс широко известен как перенаправление потока и доступен как в Windows, так и в Unix-подобных системах. Если `output.txt` не существует, он создается автоматически. Если файл уже существует, то его содержимое будет заменено новым выводом.

РЕКЛАМА



Чт

ис

\$

Ну

пе

ре

\$

Ес

st

\$

или проще

\$ python3 test.py &>> output.log

вывод последовательных исполнений скрипта в конец файла `output.txt`, то нужно

ки `>>` вместо одной:

`txt`

в файл только вывода ошибок (в случае их появления). Для этого достаточно

ок ошибок, используя команду `2>` (`2` - это дескриптор стандартного потока ошибок). В

олько то, что команда отправляет в стандартный вывод:

`xt`

в файл вывод всего того, что появляется в консоли при выполнения скрипта (`stdout` и

ть вывод в файле, то запуск сценария должен выглядеть следующим образом:

`log 2>&1`

Как быть, если нужно сохранить результаты работы сценария Python в отдельный файл, не смешивая данные со сведениями об ошибках? Так как потоки можно перенаправлять независимо друг от друга, то можно добавить в команду запуска сценария перенаправления стандартного потока вывода в один файл, а ошибок в другой файл:

```
$ python3 test.py > output.txt 2> error.txt
# с добавлением результатов сценария
# в `output.txt` и перезаписью `error.txt`
$ python3 test.py >> output.txt 2> error.txt
```

Запуск скрипта как модуля с параметром `-m`.

Python предлагает ряд [параметров командной строки](#), которые можно использовать в соответствии со своими потребностями. Например, если нужно запустить модуль Python, то необходимо использовать команду `python -m <имя-модуля>`.

Опция `-m` ищет в [sys.path](#) имя модуля и запускает его содержимое как `__main__`:

```
$ python3 -m test
# Привет Мир
# Функция "hello"
```

Примечание: имя модуля должно быть именем объекта модуля, а не просто строкой.

Запуск скрипта в интерактивном режиме с параметром `-i`.

Иногда необходимо в ручную протестировать некоторые функции или классы скрипта, и здесь программиста выручает интерактивный режим работы интерпретатора Python. После того как скрипт отработает, интерпретатор Python перейдет в режим командной строки. В этом режиме, например, можно вызвать любую функцию, определенную в скрипте с другими аргументами.

```
$ python3 -i test.py
# Привет Мир
# Функция "hello"


# здесь можно запустить любую
# функцию с другими аргументами
>>> hello('печатает слово Привет!')
# Функция "печатает слово Привет!"
```

Вверх

Использование имени файла скрипта

В последних версиях Windows можно запускать сценарии Python, просто введя имя файла, содержащего код, в командной строке:

РЕКЛАМА



С:

#

#

Эт

ис

В

#!

пр

Эт

к

Ес

Windows использует системный реестр и ассоциацию файлов, чтобы определить, какую программу деленного файла.

их как GNU/Linux, можно добиться такого поведения добавив первую строку с текстом ython это простой комментарий, а для операционной системы эта строка указывает, какую вать для запуска файла.

нация символов #!, которая обычно называется "хеш-банг" или "шебанг", и указывает путь к интерпретатору:

абсолютный путь к интерпретатору: #!/usr/bin/python3.

- использование команды env операционной системы, которая находит и запускает Python путем поиска в переменной окружения PATH: #!/usr/bin/env python3.

И наконец, чтобы выполнить скрипт в Linux, необходимо назначить ему права на выполнение.

Вот пример того, как это сделать:

```
# Даем права на выполнение
$ chmod +x test.py
# Запускаем скрипт, используя его имя файла
$ ./test.py
# Привет Мир
# Функция "hello"
```

Обратите внимание, что если скрипт не находится в текущем рабочем каталоге, то необходимо указать путь к файлу скрипта, чтобы он запустился.

Запуска скриптов из интерактивного сеанса.

Также можно запускать сценарии и модули Python из интерактивного сеанса. Этот вариант предлагает множество возможностей.

Запуск скрипта при выполнении импорта.

В момент импорта модуля происходит загрузка его содержимого для последующего доступа и использования. Самое интересное в этом процессе то, что на последнем этапе [import](#) запускает код.

Когда модуль содержит только определения классов, функций, переменных и констант, то не будет видно, что код действительно выполнялся, но когда модуль включает в себя вызовы функций, методов или других инструкций, которые генерируют видимые результаты и расположенные вне [сценария верхнего уровня](#) `'__main__'`, тогда станете свидетелем его исполнения.

```
>>> import test
# Привет Мир

# вызов функции `hello()`
>>> test.hello('запускается как test.hello()')
# Функция "запускается как test.hello()"

# выполним импорт только функции `hello()`
>>> from test import hello
>>> hello('запускается как hello()')
# Вверх "запускается как hello()"
```

Необходимо отметить, что код, который выполняется вне сценария верхнего уровня `'__main__'` - работает только один раз за сеанс. После первого импорта последующие импорты ничего не делают, даже если изменить содержимое модуля. Это связано с тем, что операции импорта являются дорогостоящими и поэтому выполняются только один раз.

Ис РЕКЛАМА : importlib.

Ст | содержит [модуль importlib](#), который предоставляет функцию [importlib.reload\(\)](#). С
по. | поставить интерпретатор повторно импортировать модуль и, следовательно, выполнить код

06 `importlib.reload()` должен быть именем объекта модуля, а не строкой!

 раз

```
ir
#
>>
>>
#
#
/lyzlov/test.py'>
```

Функция `update_requirements` также будет полезна, если вы отредактировали исходный файл модуля с помощью внешнего редактора и хотите опробовать новую версию, не выходя из интерпретатора Python.

Использование модуля `groupy`.

Стандартная библиотека включает модуль `runpy`, которая имеет функцию `runpy.run_module()`, позволяющая запускать модули без их предварительного импорта. Эта функция возвращает словарь глобальных переменных исполняемого модуля.

```
>>> import runpy
>>> runpy.run_module(mod_name='test')
# Привет Мир
# {'__name__': 'test', '__file__': ...
# ... ВЫВОД сокращен ...
# 'hello': <function hello at 0x7f3cdf6a65e0>}
```

Запускаемый модуль ищется с использованием стандартного механизма импорта, а затем выполняется в новом пространстве имен модулей. Первым аргументом `runpy.run_module()` должна быть [строка](#) с абсолютным именем выполняемого модуля без расширения `.py`.

Модуль `runpy` также предоставляет функцию `runpy.run_path()`, которая позволит запустить модуль, указав его расположение в файловой системе:

```
>>> import runpy
>>> runpy.run_path(path_name='test.py')
# Привет Мир
# {'__name__': 'test', '__file__': ...
# ... ВЫВОД сокращен ...
# 'hello': <function hello at 0x7f3cdf6a65e0>}
```

Как и `runpy.run_module()`, так и `runpy.run_path()` возвращает глобальный словарь исполняемого модуля.

Аргумент `path_name` должен быть строкой и может ссылаться на следующее:

- Расположение исходного файла Python
- Расположение скомпилированного файла байт-кода
- Значение допустимой записи в `sys.path`, содержащей модуль `__main__` (файл `__main__.py`)

Использование функции `exec()` .

Встроенная функция `exec()` поддерживает динамическое выполнение кода Python, тем самым предоставляет альтернативный способ запуска скриптов:

```
exec(open('test.py').read())
# Привет Мир
# Вверх "hello"
```


Приведенный выше пример немного не соответствует действительности. Это просто "ХАК", который показывает, что можно использовать один и тот же код для работы с разными типами данных. Это универсальный и гибкий подход.



[ора Python](#)
[ой строки Python](#)
[thon](#)

[@docs_python_ru](#)