

# Python 3 для начинающих

[Книги](#)[Самоучитель](#)[Курсы](#)[Категории ↓](#)[Все статьи](#)

## Работа с модулями: создание, подключение инструкциями `import` и `from`



### Онлайн-тренажер Python 3 для начинающих

Теория без воды. Задачи с автоматической проверкой. Подсказки на русском языке. Работает в любом современном браузере.

[НАЧАТЬ БЕСПЛАТНО](#)

Модулем в Python называется любой файл с программой (да-да, все те программы, которые вы писали, можно назвать модулями). В этой статье мы поговорим о том, как создать модуль, и как подключить модуль, из [стандартной библиотеки](#) или написанный вами.

Каждая программа может импортировать модуль и получить доступ к его классам, функциям и объектам. Нужно заметить, что модуль может быть написан не только на Python, а например, на C или C++.

## Подключение модуля из стандартной библиотеки

Подключить модуль можно с помощью инструкции `import`. К примеру, подключим [модуль os](#) для получения текущей директории:

`>>>`

### Свежее

- [Модуль csv - чтение и запись CSV файлов](#)
- [Создаём сайт на Django, используя хорошие практики. Часть 1: создаём проект](#)
- [Онлайн-обучение Python: сравнение популярных программ](#)

### Категории

- [Книги о Python](#)
- [GUI \(графический интерфейс пользователя\)](#)
- [Курсы Python](#)
- [Модули](#)
- [Новости мира Python](#)
- [NumPy](#)
- [Обработка данных](#)
- [Основы программирования](#)
- [Примеры программ](#)
- [Типы данных в Python](#)
- [Видео](#)

```
>>> import os
>>> os.getcwd()
'C:\\Python33'
```

После ключевого слова **import** указывается название модуля. Одной инструкцией можно подключить несколько модулей, хотя этого не рекомендуется делать, так как это снижает читаемость кода.

Импортируем модули **time** и **random**.

```
>>> import time, random
>>> time.time()
1376047104.056417
>>> random.random()
0.9874550833306869
```


После импортирования модуля его название становится переменной, через которую можно получить доступ к атрибутам модуля. Например, можно обратиться к константе **e**, расположенной в модуле **math**:

```
>>> import math
>>> math.e
2.718281828459045
```


Стоит отметить, что если указанный атрибут модуля не будет найден, возникнет **исключение** `AttributeError`. А если не удастся найти модуль для импортирования, то `ImportError`.

```
>>> import notexist
Traceback (most recent call last):
  File "", line 1, in
    import notexist
ImportError: No module named 'notexist'
>>> import math
>>> math.Ë
Traceback (most recent call last):
  File "", line 1, in
```

 [Python для Web](#)

 [Работа для Python-программистов](#)

## Полезные материалы

 [Сделай свой вклад в развитие сайта!](#)

 [Самоучитель Python](#)

 [Карта сайта](#)

 [Отзывы на книги по Python](#)

 [Реклама на сайте](#)

## Мы в соцсетях

Подпишись на обновления [по RSS](#) или по почте!

[Подписаться!](#)

```
math.E
```

```
AttributeError: 'module' object has no attribute
```

## Использование псевдонимов

Если название модуля слишком длинное, или оно вам не нравится по каким-то другим причинам, то для него можно создать псевдоним, с помощью ключевого слова `as`.

```
>>>
```

```
>>> import math as m
```

```
>>> m.e
```

```
2.718281828459045
```

Теперь доступ ко всем атрибутам модуля `math` осуществляется только с помощью переменной `m`, а переменной `math` в этой программе уже не будет (если, конечно, вы после этого не напишете `import math`, тогда модуль будет доступен как под именем `m`, так и под именем `math`).

## Инструкция `from`

Подключить определенные атрибуты модуля можно с помощью инструкции `from`. Она имеет несколько форматов:

```
from <Название модуля> import <Атрибут 1> [ as <псевдоним> ]
```

```
from <Название модуля> import *
```

Первый формат позволяет подключить из модуля только указанные вами атрибуты. Для длинных имен также можно назначить псевдоним, указав его после ключевого слова `as`.

```
>>>
```

```
>>> from math import e, ceil as c
```

```
>>> e
```

```
2.718281828459045
```

```
>>> c(4.6)
```

```
5
```

Импортируемые атрибуты можно разместить на нескольких строках, если их много, для лучшей читаемости кода:

```
>>>
>>> from math import (sin, cos,
...                  tan, atan)
```

Второй формат инструкции from позволяет подключить все (точнее, почти все) переменные из модуля. Для примера импортируем все атрибуты из модуля `sys`:

```
>>>
>>> from sys import *
>>> version
'3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:00:00)
>>> version_info
sys.version_info(major=3, minor=3, micro=2, releaselevel='final', serial=0)
```

Следует заметить, что не все атрибуты будут импортированы. Если в модуле определена переменная `__all__` (список атрибутов, которые могут быть подключены), то будут подключены только атрибуты из этого списка. Если переменная `__all__` не определена, то будут подключены все атрибуты, не начинающиеся с нижнего подчёркивания. Кроме того, необходимо учитывать, что импортирование всех атрибутов из модуля может нарушить пространство имен главной программы, так как переменные, имеющие одинаковые имена, будут перезаписаны.

## Создание своего модуля на Python

Теперь пришло время создать свой модуль. Создадим файл `mymodule.py`, в которой определим какие-нибудь функции:

```
def hello():
    print('Hello, world!')

def fib(n):
    a = b = 1
```

```
for i in range(n - 2):  
    a, b = b, a + b  
  
return b
```

Теперь в этой же папке создадим другой файл, например, main.py:

```
import mymodule  
  
mymodule.hello()  
print(mymodule.fib(10))
```

Выведет:

```
Hello, world!  
55
```

Поздравляю! Вы **сделали свой модуль**! Напоследок отвечу ещё на пару вопросов, связанных с созданием модулей:

## Как назвать модуль?

Помните, что вы (или другие люди) будут его импортировать и использовать в качестве переменной. Модуль нельзя именовать также, как и ключевое слово (их список можно посмотреть [тут](#)). Также имена модулей нельзя начинать с цифры. И не стоит называть модуль также, как какую-либо из [встроенных функций](#). То есть, конечно, можно, но это создаст большие неудобства при его последующем использовании.

## Куда поместить модуль?

Туда, где его потом можно будет найти. Пути поиска модулей указаны в переменной sys.path. В него включены текущая директория (то есть модуль можно оставить в папке с основной программой), а также директории, в которых установлен python. Кроме того, переменную sys.path можно изменять вручную, что позволяет положить модуль в любое удобное для вас

место (главное, не забыть в главной программе модифицировать `sys.path`).

## Можно ли использовать модуль как самостоятельную программу?

Можно. Однако надо помнить, что при импортировании модуля его код выполняется полностью, то есть, если программа что-то печатает, то при её импортировании это будет напечатано. Этого можно избежать, если проверять, запущен ли скрипт как программа, или импортирован. Это можно сделать с помощью переменной `__name__`, которая определена в любой программе, и равна `"__main__"`, если скрипт запущен в качестве главной программы, и имя, если он импортирован. Например, `mymodule.py` может выглядеть вот так:

```
def hello():
    print('Hello, world!')

def fib(n):
    a = b = 1
    for i in range(n - 2):
        a, b = b, a + b
    return b

if __name__ == "__main__":
    hello()
    for i in range(10):
        print(fib(i))
```

Ошибка в тексте?  
Выделите её мышкой!  
И нажмите:



Для вставки кода на Python в комментарий заключайте его в теги `<pre>`

`<code class="python3">Ваш код</code></pre>`









