Тема 1. Модули и пакеты в Python

Установка python-пакетов с помощью pip

pip - это система управления пакетами, которая используется для установки и управления программными пакетами, написанными на Python. Начиная с Python версии 3.4, pip поставляется вместе с интерпретатором python.

Начало работы

Попробуем с помощью pip установить какой-нибудь пакет, например, **numpy** (https://pythonworld.ru/numpy) в консоле (в режиме администратора):

```
Linux:
sudo pip3 install numpy

Windows:
pip3 install numpy
```

Может возникнуть ошибка: *"pip3" не является внутренней или внешней командой, исполняемой программой или пакетным файлом*.

Тогда необходимо указывать полный путь к программе, например (если Питон установлен в `C:\Python36\`):

```
C:\Python36\Tools\Scripts\pip3.exe install numpy
```

Либо добавлять папку `C:\Python36\Tools\Scripts\` в PATH вручную.

Либо сделав эту папку с pip3 текущей:

cd "`C:\Python36\Tools\Scripts\`"

```
pip3 install numpy
```

В последних версиях python вызов **pip** удобно выполнять из консоли:

```
python -m pip <command> [options]
```

Что ещё умеет делать pip3

Основные команды **pip3**:

pip help - помощь по доступным командам.

pip install package_name - установка пакета(ов).

pip uninstall package_name - удаление пакета(ов).

pip list - список установленных пакетов.

pip show package_name - показывает информацию об установленном пакете.

pip search - поиск пакетов по имени.

pip --proxy user:passwd@proxy.server:port - использование с прокси.

pip install -U - обновление пакета(ов).

pip install --force-reinstall - при обновлении, переустановить пакет, даже если он последней версии.

Пример проверки, установки и переустановки пакета matplotlib

```
"C:\Program Files\Python36-32\Scripts\pip" install -U matplotlib

Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 07:18:10) [MSC
v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import matplotlib as mpl
>>> print ('Current version on matplotlib library is',
                                                    mpl.__version__)
Current version on matplotlib library is 2.1.1
>>>
# де-факто стандарт вызова pyplot в python
import matplotlib.pyplot as plt
```

Модули в Python

Система модулей позволяет логически организовать ваш **код на Python**. Группирование кода в модули значительно облегчает процесс написания и понимания программы. **Модуль в Python** это просто файл, содержащий код на Python. Каждый **модуль в Python** может содержать переменные, объявления классов и функций. Кроме того, в модуле может находиться исполняемый код.

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:

```
>>> 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, возбудится 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
    math.Ë
AttributeError: 'module' object has no attribute 'Ë'
```

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 1> ],
                  [<Атрибут 2> [ as <Псевдоним 2> ] ...]

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, 00:03:43)
[MSC v.1600 32 bit (Intel)]'
>>> 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*.

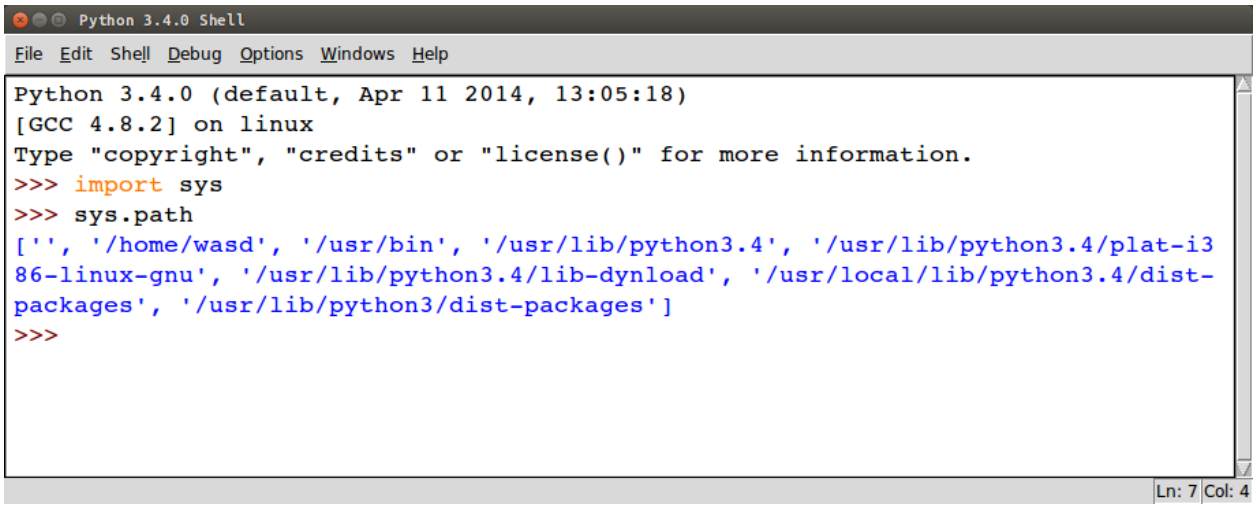
Кроме того, необходимо учитывать, что импортирование всех атрибутов из модуля может нарушить пространство имен главной программы, так как переменные, имеющие одинаковые имена, будут перезаписаны.

Местонахождение модулей в Python:

Когда вы импортируете модуль, интерпретатор Python ищет этот модуль в следующих местах:

1. Директория, в которой находится файл, в котором вызывается команда импорта
2. Если модуль не найден, Python ищет в каждой директории, определенной в консольной переменной PYTHONPATH.
3. Если и там модуль не найден, Python проверяет путь заданный по умолчанию

Путь поиска модулей сохранен в системном модуле sys в переменной path. Переменная **sys.path** содержит все три вышеописанных места поиска модулей.

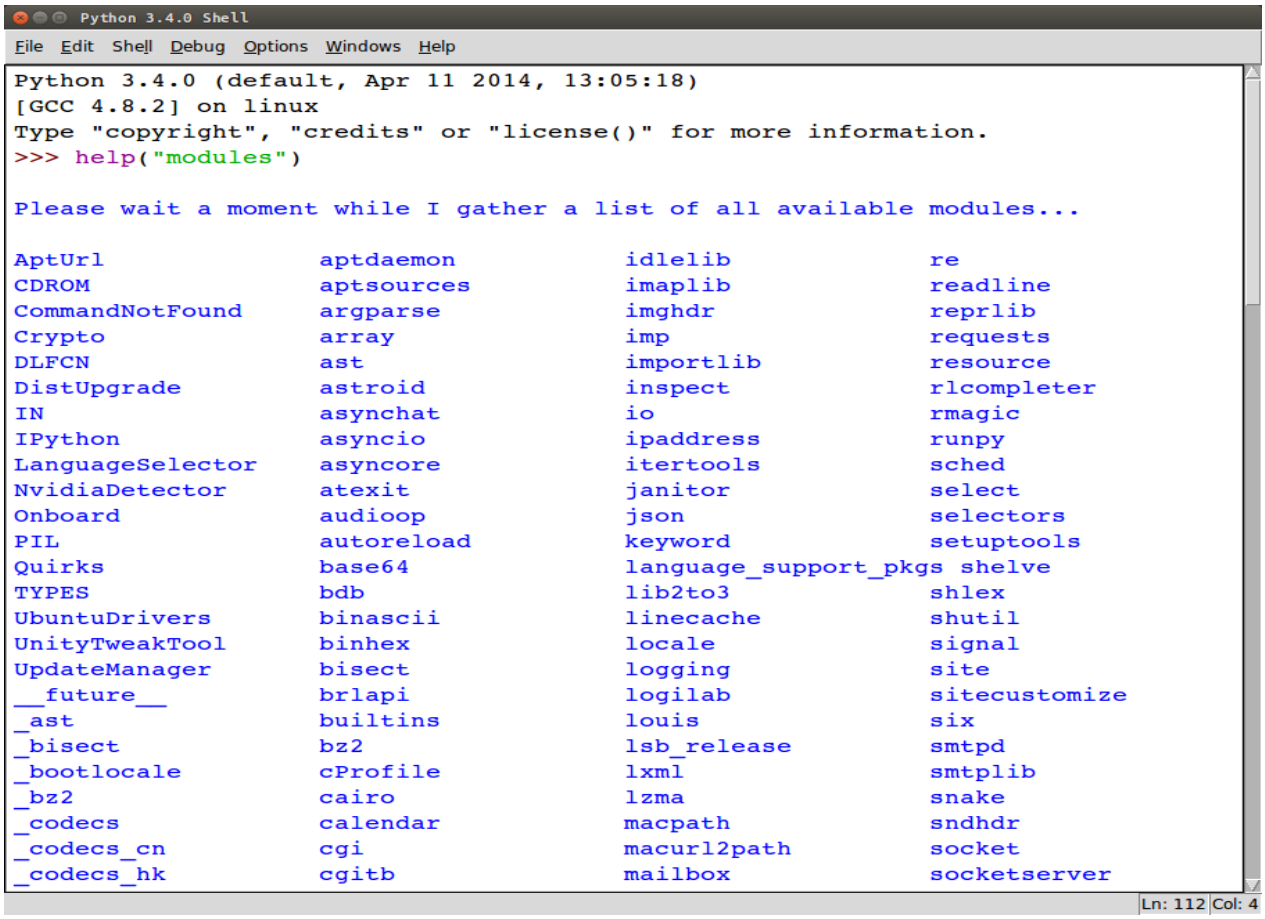


Получение списка всех модулей Python, установленных на компьютере

Для того, чтобы получить список всех модулей, установленных на вашем компьютере достаточно выполнить команду:

```
help("modules")
```

Через несколько секунд вы получите список всех доступных модулей.



Создание своего модуля в Python:

Чтобы создать свой **модуль в Python** достаточно сохранить ваш скрипт с расширением `.py` Теперь он доступен в любом другом файле. Например, создадим два файла: `module_1.py` и `module_2.py` и сохраним их в одной директории. В первом запишем:

```
def hello():
    print ("Hello from module_1")
```

А во втором вызовем эту функцию:

```
from module_1 import hello

hello()
```

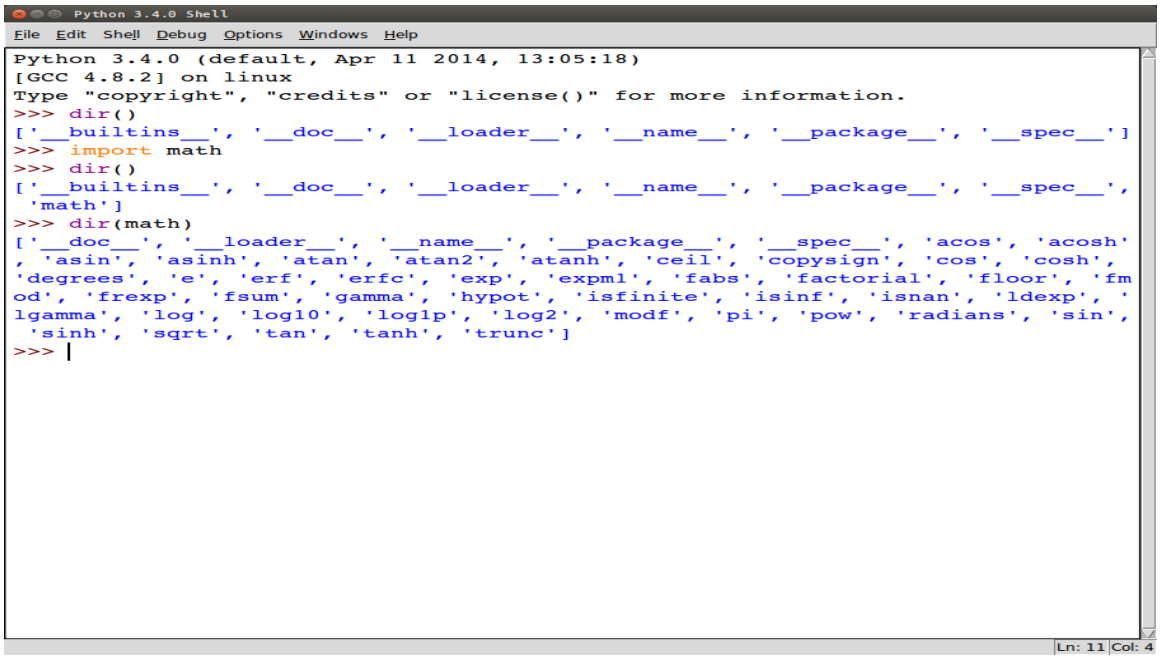
Выполнив код второго файла получим:

```
Hello from module_1
```

Функция dir():

Встроенная **функция `dir()`** возвращает отсортированный список строк, содержащих все имена, определенные в модуле.

```
# на данный момент нам доступны лишь
                                        встроенные функции
dir()
# импортируем модуль math
import math
# теперь модуль math в списке доступных имен
dir()
# получим имена, определенные в модуле math
dir(math)
```



Архитектура программы на Python

Код на Python может быть организован следующим образом:

1. Первый уровень это обычные команды на Python.
2. Команды на Python могут быть собраны в функции.
3. Функции могут быть частью класса.
4. Классы могут быть определены внутри модулей.
5. Наконец, модули могут составляться в пакеты модулей.

Пакеты модулей в Python

Отдельные файлы-модули с кодом на Python могут объединяться в **пакеты модулей**. Пакет это директория (папка), содержащая несколько отдельных файлов-скриптов.

Например, имеем следующую структуру:

```
|_ my_file.py
|_ my_package
          |_ __init__.py
          |_ inside_file.py
```

В файле `inside_file.py` определена некая функция `foo`. Тогда чтобы получить доступ к функции `foo`, в файле `my_file` следует выполнить следующий код:

```
from my_package.inside_file import foo
```

Так же обратите внимание на наличие внутри директории `my_package` файла `__init__.py`. Это может быть пустой файл, который сообщает **Python**, что данная директория является **пакетом модулей**. В Python 3.3 и выше включать файл `__init__.py` в **пакет модулей** стало необязательно, однако, рекомендуется делать это ради поддержки обратной совместимости.