Тема 1. Модулі та пакети в Python

Встановлення python-пакетів за допомогою pip

pip - це система керування пакетами, яка використовується для встановлення та керування програмними пакетами, написаними на Python. З Python версії 3.4, pip поставляється разом з інтерпретатором python.

Початок роботи

Спробуємо за допомогою pip встановити якийсь пакет, наприклад, numpy (https://pythonworld.ru/numpy) у консолі (в режимі адміністратора):

```
Linux:
sudo pip3 install numpy

Windows:
pip3 install numpy
```

Може виникнути помилка: "pip3" не є внутрішньою чи зовнішньою командою, яку виконує програма або пакетний файл.

Тоді необхідно вказувати повний шлях до програми, наприклад (якщо Пітон встановлений у C: Python36):

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

Або додавати папку C:\Python36\Tools\Scripts\ в PATH вручну.

Або зробивши цю папку з pip3 поточної:

cd “C:\Python36\Tools\Scripts\”

```
pip3 install numpy
```

В останніх версіях python виклик pip зручно виконувати з консолі:

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

Що ще вміє робити

Основні команди pip3:

pip help - Допомога по доступним командам.

pip install package_name - Встановлення пакета(ів).

pip uninstall package_name - Видалення пакета(ів).

pip list - Перелік встановлених пакетів.

pip show package_name - Показує інформацію про встановлений пакет.

pip search - Пошук пакетів на ім'я.

pip --proxy user: passwd@proxy.server :port - Використання з проксі.

pip install -U - Оновлення пакета(ів).

pip install --force-reinstall - при оновленні, інсталювати пакет, навіть якщо він останньої версії.

Приклад перевірки, встановлення та 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
Тип "copyright", "credits" або "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 виклику pyplot у python
import matplotlib.pyplot as plt
```

Модулі в Python

Система модулів дозволяє логічно організувати ваш код Python. Групування коду в модулі значно полегшує процес написання та розуміння програми. Модуль Python це просто файл, що містить код Python. Кожен модуль Python може містити змінні, оголошення класів і функцій. Крім того, у модулі може знаходитися виконуваний код.

Кожна програма може імпортувати модуль та отримати доступ до його класів, функцій та об'єктів. Потрібно помітити, що модуль може бути написаний не лише на Python, а, наприклад, на C або C++.

Підключення модуля зі стандартної бібліотеки

Підключити модуль можна за допомогою інструкції import. Наприклад, підключимо модуль os для отримання поточної директорії:

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

Після ключового слова import вказується назва модуля. Однією інструкцією можна підключити кілька модулів, хоча цього не рекомендується робити, оскільки це знижує читання коду. Імпортуємо модулі time та random.

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

Після імпортування модуля його назва стає змінною, через яку можна отримати доступ до атрибутів модуля. Наприклад, можна звернутися до константи e, розташованої в модулі math:

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

Варто зазначити, що якщо зазначений атрибут модуля не буде знайдено, збудеться виняток AttributeError. Якщо ж не вдасться знайти модуль для імпортування, то ImportError.

```
>>>
>>> import notexist
Traceback (most recent call last):
  File "", line 1, in
    importnotexist
ImportError: No module named 'notexist'
>>> import math
>>> math.
Traceback (most recent call last):
  File "", line 1, in
    math.
AttributeError: 'module' object has no attribute 'E'
```

Використання псевдонімів

Якщо назва модуля занадто довга, або вона вам не подобається з якихось інших причин, то для нього можна створити псевдонім за допомогою ключового слова as.

```
>>>
>>> import math as m
>>> me
2.718281828459045
```

Тепер доступ до всіх атрибутів модуля math здійснюється тільки за допомогою змінної m, а змінної math у цій програмі вже не буде (якщо, звичайно, ви після цього не напишете import math, тоді модуль буде доступний як під ім'ям m, так і під ім'ям math).

Інструкція від

Підключити певні атрибути модуля можна за допомогою інструкції від. Вона має кілька форматів:

```
from<Назва модуля>import<Атрибут 1> [as<Псевдонім 1> ],
[<Атрибут 2> [as<Псевдонім 2> ] ...]
```

```
from<Назва модуля>import*
```

Перший формат дозволяє підключити з модуля лише вказані вами атрибути. Для довгих імен можна призначити псевдонім, вказавши його після ключового слова as.

```
>>>
```

```
>>> from math import e, ceil as c
>>> e
2.718281828459045
>>> c(4.6)
5
```

Атрибути, що імпортуються, можна розмістити на декількох рядках, якщо їх багато (для кращої читання коду):

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

Другий формат інструкції from дозволяє підключити всі (точніше, майже всі) змінні з модуля. Наприклад імпортуємо всі атрибути з 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)
```

Слід зазначити, що не всі атрибути імпортуватимуться. Якщо в модулі визначено змінну __all__ (список атрибутів, які можуть бути підключені), то будуть підключені лише атрибути цього списку. Якщо змінна __all__ не визначена, то буде підключено всі атрибути, що не починаються з нижнього підкреслення.

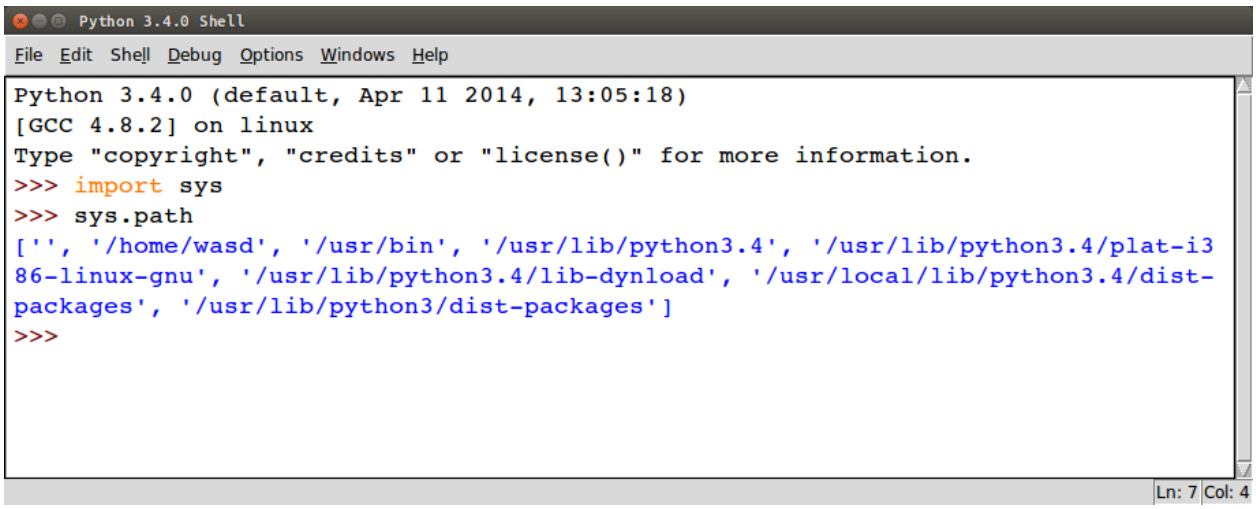
Крім того, необхідно враховувати, що імпортування всіх атрибутів з модуля може порушити простір імен головної програми, оскільки змінні, що мають однакові імена, будуть перезаписані.

Місцезнаходження модулів у Python:

Коли ви імпортуєте модуль, інтерпретатор Python шукає цей модуль у таких місцях:

1. Директорія, в якій знаходиться файл, у якому викликається команда імпорту
2. Якщо модуль не знайдено, Python шукає у кожній директорії, визначеній у консольній змінній PYTHONPATH.
3. Якщо і там модуль не знайдено, Python перевіряє шлях, заданий за замовчуванням.

Шлях пошуку модулів збережено в системному модулі sys змінної path. Змінна sys.path містить усі три вищеописані місця пошуку модулів.

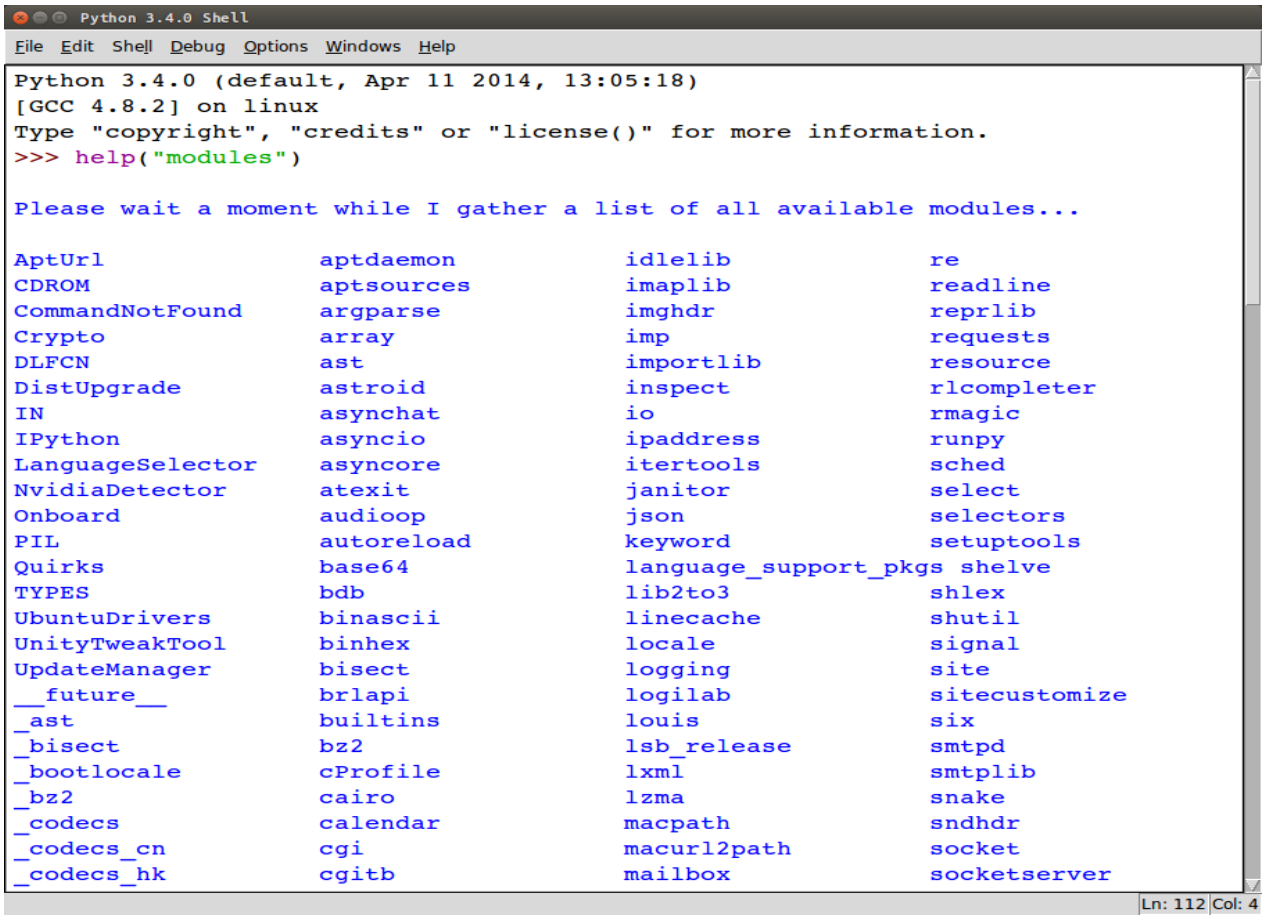


Отримання списку всіх модулів Python, встановлених на комп'ютері

Для того, щоб отримати список усіх модулів, встановлених на вашому комп'ютері, достатньо виконати команду:

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

За кілька секунд ви отримаєте список усіх доступних модулів.



Створення свого модуля в Python:

Щоб створити свій модуль у Python, достатньо зберегти ваш скрипт з розширенням .py Тепер він доступний в будь-якому іншому файлі. Наприклад,

створимо два файли: module_1.py та module_2.py і збережемо їх в одній директорії. У першому запишемо:

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

А в другому викличемо цю функцію:

```
from module_1 import hello

hello()
```

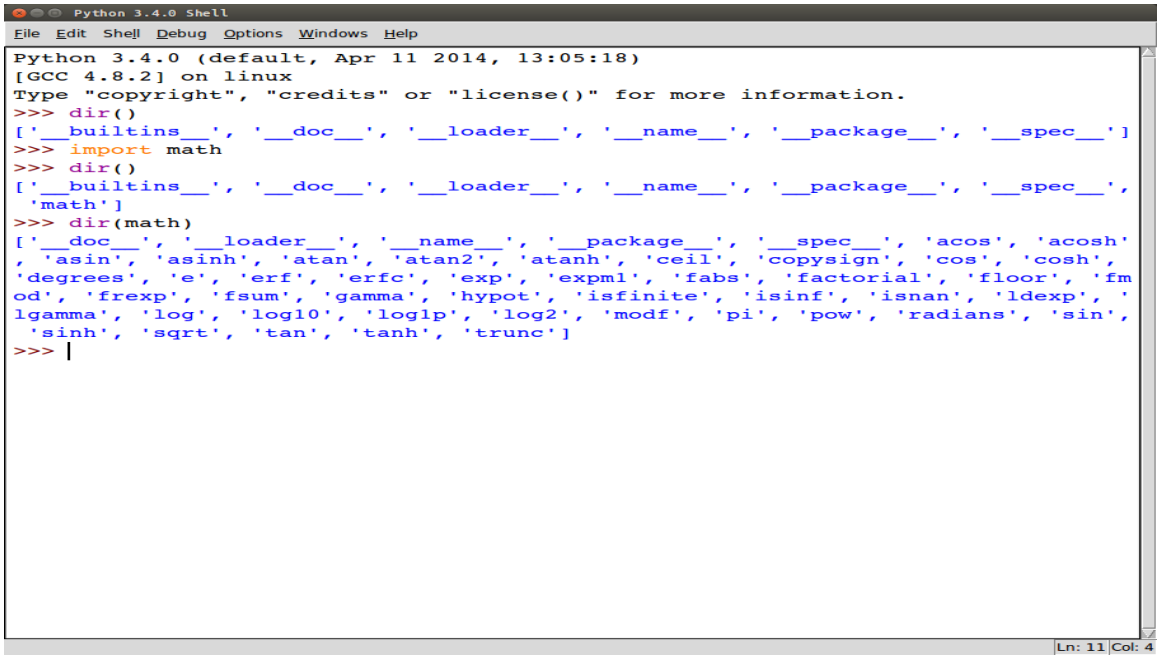
Виконавши код другого файлу отримаємо:

```
Hello from module_1
```

Функція dir():

Вбудована функція dir() повертає відсортований список рядків, що містять усі імена, визначені у модулі.

```
# на даний момент нам доступні лише вбудовані функції
dir()
# Імпортуємо модуль math
Import math
# тепер модуль math у списку доступних імен
dir()
# отримаємо імена, визначені в модулі math
dir(math)
```



Архітектура програми на Python

Код Python може бути організований таким чином:

1. Перший рівень - це звичайні команди на Python.

2. Команди на Python можуть бути зібрані у функції.
3. Функції може бути частиною класу.
4. Класи можна визначити всередині модулів.
5. Нарешті, модулі можуть складатися пакети модулів.

Пакети модулів у Python

Окремі файли-модулі з кодом Python можуть об'єднуватися в пакети модулів. Пакет це директорія (папка), що містить кілька окремих файлів-скриптів.

Наприклад, маємо таку структуру:

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

У файлі inside_file.py визначено певну функцію foo. Тоді, щоб отримати доступ до функції foo, у файлі my_file слід виконати наступний код:

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

Також зверніть увагу на наявність усередині директорії my_package файлу __init__.py. Це може бути порожній файл, який повідомляє Python, що ця директорія є пакетом модулів. У Python 3.3 і вище включати файл __init__.py в пакет модулів стало необов'язково, проте рекомендується робити це задля підтримки зворотної сумісності.