Modules and Packages in Python

About me

Trayan Iliev

– CEO of **IPT – Intellectual Products & Technologies**
http://www.iproduct.org

– Oracle® certified programmer 15+ Y

– end-to-end reactive fullstack apps with Java, ES6+, TypeScript, Angular, React and Vue.js

– 12+ years IT trainer: Spring, Java EE, Node.js, Express, GraphQL, SOA, REST, DDD & Reactive Microservices

– Voxxed Days, jPrime, Java2Days, jProfessionals, BGOUG, BGJUG, DEV.BG speaker

– Organizer RoboLearn hackathons and IoT enthusiast

Where to Find The Code and Materials?

https://github.com/iproduct/intro-python

What You Will Learn?

- Програми на Python
- Модули
- Основни действия с модули
- Зареждане и изпълнение на модули
- Помощна информация за модули
- Синтаксис на командите за работа с модули
- Пакети
- Зареждане на пакети
- Видове пакети

Files and Programs

- Изучихме основните конструкции в езика
- Знаем как да създаваме програма за даден алгоритъм и да я запазваме във файл
- Всеки файл включва множество от команди и изрази
- Всеки файл има собствено пространство от обекти и свойства
- Програмите на Python са модули, създавани като текстови файлове с разширението .py
- Всеки модул в Python има свое пространство на имената
- Това пространство в един Python модул е глобално

Python Programs

- Програмите и модулите в езика Python се различават само по начина на използване:
- .py файловете, които се изпълняват от интерпретатора, са програми (често наричани скриптове)
- .py файлове, към които има обръщение с командата **import**, са модули
- Така един и същи файл може да бъде както програма (скрипт), така и модул

The Purpose of Modules

- Многократно използване на команди
- Всяка функция може да бъде извиквана многократно по време на изпълнение
- Функциите могат да бъдат извикани от различни програми
- Управление областите на имената
- Групиране заедно данните с функциите които ги използват
- Споделени услуги върху набор данни
- Глобална структура от данни достъпна от различни функции и програми

Using Modules

- Модулът, който се изпълни пръв (файлът който пръв се зареди в системата) се нарича главен (main)
- След това той може да зарежда други модули или от файлове, или от стандартната библиотека
- Всеки модул предоставя своите обекти като атрибути:

  <име_модул>.<име_атрибут>

- Тези атрибути най-често са функции, но могат да са произволни обекти

Loading Modules

При първо зареждане на модул, се изпълняват три отделни важни стъпки:

1. Откриване (намиране) на модула (файлът съдържащ неговите команди и дефиниции)
2. Компилиране на модула (ако е нужно)
3. Изпълнение на компилирания код за създаване на обектите, които модула предоставя

Loading Modules – sys.path

Python използва вграден път за търсене на модули който зависи от ОС и може да се допълва/променя. Освен пътя, в командата **import** се изпуска и типа на файла (използват се разширенията приети в ОС)

Кой е текущия път се вижда чрез извеждане на **sys.path** (след **import** на модула **sys**)

Пътят може да се променя чрез промяна стойността на този обект (тази променлива)

Modules Loading Paths – sys.path

Python използва следната йерархия от пътища:

- Текуща директория (където се намира стартирания файл, или от където се стартира Python)
- Пътища указани в променливата: **PYTHONPATH**
- Път към стандартната библиотека
- Пътища указани в **.pth** файлове
- Път към **site-packages** (други модули)

Първият, третия и последния елемент от йерархията от пътища е вграден в системата. Потребителят може да влияе чрез пътищата указани в променливата: PYTHONPATH, както и чрез .pth файлове

Compiling Modules

- При първо зареждане на модул, той трябва да е наличен в байт код компилиран формат. Освен текстовия файл с текста на програмата (разширение **.py**) системата съхранява и компилирана версия (с разширение **.pyc**)
- Системата проверява времената на последна промяна на двата варианта за модула, и ако **.py** не е по-нов от **.pyc** - зарежда **.pyc**
- Иначе, компилира модула и го запазва в **.pyc**, и после го зарежда. Всички файлове .pyc се намират в папката **__pycache__** в основната за **.py** папка

Compiling Modules - II

- След зареждане на компилирания модул, байт код командите му се изпълняват последователно една след друга за създаване на обектите, които модула предоставя.
- Например, дефинициите на функции def се изпълняват и новосъздадените обекти тип функция се записват като атрибути предоставяни от модула.
- Всяка следваща команда import за същия модул в същия процес (програма main) не прави нищо, а се използва каквото вече е налично в паметта.

Function dir

- Използва се за получаване на списък от всички свойства и атрибути на даден обект.
- Когато се задава без аргумент връща списък на всички променливи в текущия контекст
- Типичен начин на използване – когато като аргумент се зададе име на модул, или име на вграден тип от данни
- За да се покаже тази информация за някакъв модул, той първо трябва да се зареди с import

Function dir – Example Usage

```
>>> import sys

>>> [a for a in dir(list) if not a.startswith('__')]

['append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove',
'reverse', 'sort']

>>> [a for a in dir(dict) if not a.startswith('__')]

['clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault',
'update', 'values']

>>> def dir1(x): return [a for a in dir(x) if not a.startswith('__')]

>>> dir1(tuple)

['count', 'index']
```

Function help

```
>>> help(dict)
Help on class dict in module builtins:
class dict(object)
 |  dict() -> new empty dictionary.
 |  dict(mapping) -> new dictionary initialized from a mapping object's
...
>>> help(ord)
Help on built-in function ord in module builtins:
ord(c, /)
    Return the Unicode code point for a one-character string.
```

Modules

- Групират функции и променливи във файлове
- Обектите са достъпни с команди from или import

```
from module import function
function()
```

```
import module
module.function()
```

- Всеки модул е отделна област на имена
- Може да се използва за организация на имената:

```
figure.position = figure.position - scene.position
```

Modules - II

- Всяка програма на Python има достъп само до базовите функции и класове

  (“int”, “dict”, “len”, “sum”, “range”, ...)

- Модулите съдържат допълнителна функционалност
- Модулът в Python е файл с име: **<име_модул> .py**
- Командата “import” казва на Python да зареди модул

```
>>> import math
>>> import numpy
```

“import” and “from ... import ...”

```
>>> import math
math.cos
>>> from math import cos, pi
cos
>>> from math import *
```

import ...

import somefile

- Всичко от файла с име somefile.py се зарежда, но в собствена област на имената
- За обръщение към обект от модула, се добавя “somefile.” пред името на обекта:

somefile.className.method(“abc”)

somefile.myFunction(34)

from … import *

from somefile import *

- Всичко от somefile.py се зарежда в текущото пространство на имената
- Обръщение към обект от модула става с името му
- Опасност! Чрез командата import в този вид лесно можем да изтрием функция или променлива от текущото пространство на имената!
- Този метод за зареждане не се препоръчва

className.method(“abc”)

myFunction(34)

from … import …

from somefile import Name

- Само обектът с име Name от модула somefile.py се зарежда
- След зареждане на обекта Name, можем да го използваме само по името му – той вече е част от текущото пространство на имената
- Внимание! Така заредения обект може да скрие обект със същото име от текущото пространство

Name.method(“abc”) imported

myFunction(34) Not imported

Module Execution

- Всяка следваща команда import за същия модул в същия процес (програма main) не прави нищо, а се използва каквото вече е налично в паметта.

- Това води до проблем, ако искаме да заредим нова версия на програмата, създадена и вече променена във файла с модула

- Решение – с функцията **reload**

reload

- Потребител зарежда модул, променя кода му в текстов редактор и го зарежда отново. Това се случва ако се работи интерактивно или с голяма програма, в която модулите се зареждат периодично.

```
import module # първоначален import

...use module.attributes...

...

from imp import reload #  зареждане на reload

reload(module) # зареждане на нова версия
```

reload

- Зарежда новия код и връща неговото пространство. Променя модула на място в паметта:
- Новият код и обекти се зареждат в и изменят текущото пространство на модула
- Имената се заместват с нови обекти. Например, def командата заменя функцията с нов обект
- При пълен import достъпа е до новите обекти
- При клауза from достъпът остава до старите обекти
- **reload** се използва само за един модул

Importing Multiple Modules

```
# Multiple modules
>>> import time, sockets, random
>>> # Multiple functions
>>> from math import sin, cos, tan
>>> # Multiple constants
>>> from math import pi, e
>>> print(pi)
3.141592653589793
>>> print(cos(45))
0.5253219888177297
>>> print(time.time())
1482807222.7240417
```

Special Variable __all__

С помощта на променливата __all__ се ограничава кои променливи от модул да са достъпни след from … import *

```
# mymodule.py
__all__ = ['imported_by_star']
imported_by_star = 42
not_imported_by_star = 21
>>> from mymodule import *
>>> imported_by_star
42
>>> not_imported_by_star
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'not_imported_by_star' is not defined
```

Special Variable __all__

Въпреки това, променливите които не са изрично указани в променливата __all__ също могат да бъдат заредени, но само ако бъдат явно указани:

```
>>> from mymodule import not_imported_by_star
>>> not_imported_by_star
21
```

Special Variable __all__

Въпреки това, променливите които не са изрично указани в променливата __all__ също могат да бъдат заредени, но само ако бъдат явно указани:

```
>>> from mymodule import not_imported_by_star
>>> not_imported_by_star
21
```

PEP 8 Usage Guidelines

1. Всяка клауза import на отделен ред :

```
from math import sqrt, ceil  # Не се препоръчва
from math import sqrt        # Препоръчва се
from math import ceil
```

2. Всички клаузи import да бъдат в началото подредени в следния ред:

- От стандартната библиотека
- От други общоприети библиотеки
- Локални / специфични за дадена библиотека

3. Да се избягва клауза from module import *

4. Да се избягват относителни имена; да се дават явни

Packages

- Това са специални обединения от модули, групирани в структури от папки, за съвместно използване.
- Използването на папки помага за структуриране на модулите по групи функции и улеснява използването на пакета от даден потребител в зависимост от това какви функции и модули от пакета са му нужни.
- При тях се използва зареждане от вида:

```
import sound.effects.echo          #зарежда модул Echo
sound.effects.echo.echofunc()      #обръщение към обект
```

Loading Packages

```
import sound.effects.echo                    #зарежда модул Echo
sound.effects.echo.echofunc()                #обръщение към обект

from sound.effects import echo               #зарежда модул Echo
echo.echofunc()                              #обръщение към обект

from sound.effects.echo import echofunc   #зарежда Echo
# Улеснява обръщението към обектите от модула Echo
echofunc(input, output, delay=0.7, atten=4)
```

Loading Packages – II

from <пакет> import <елемент>

<елемент> може да бъде под-пакет, модул, обект

Първо се търси като обект, и само ако не се намери, се търси после като модул или под-пакет

import <елем1.елем2.елем3>

Тук <елем1> и <елем2> са задължително под-пакети

<елем3> може да бъде модул или под-пакет, но не може да бъде обект

Identifing Python Packages

Една папка се възприема като (под)пакет, ако в нея има специален файл с името: __init__.py

Когато се използва зареждане от вида:

>>> import <елем1.елем2.елем3>

- <елем1> трябва да бъде под-папка на папка от текущия път (sys.path)
- <елем1> и <елем2> да съдържат в себе си задължително файл с име __init__.py
- Ако <елем3> е папка (под-пакет), също трябва да съдържа файл с такова име
- Ако <елем3> е име на файл се възприема като модул

Loading Packages

Когато се използва зареждане от вида:

>>> import <елем1.елем2.елем3>

се зареждат последователно (под)пакетите <елем1> (т.е. се изпълнява нейния файл __init__.py), <елем2> и накрая или (под)пакета <елем3> (т.е. неговия файл __init__.py), или модула <елем3> (т.е. файла с име <елем3>)

Друга разлика между пакет и модул е наличието на променливата __path__ само в дефиницията на пакет, която съдържа списък с един елемент името на папката в която е дефиниран пакета.

Loading Packages - II

from <пакет> import *

В този случай в папката на последния под-пакет от името зададено с <пакет> се търси файла __init__.py и в него променливата от тип списък __all__ , в която се съдържат имената на модулите (файловете) които да се заредят при * от този под-пакет (папка)

Ако имаме командата:

>>> from sound.effects import *

във файла sounds/sffects/__init__.py трябва да имаме:

__all__ = ["echo", "surround", "reverse"]

Ако __all__ не е дефинирана в __init__.py, ще се заредят само обекти създавани в или зареждани от инициализиращия файл __init__.py (евентуално и заредени преди това с предишни команди import)

Types of Packages

- Регулярен (нормален) пакет - това са пакетите за които говорихме до момента, които имат специалния файл __init__.py в своята папка, и който се използва за инициализирането на пакета (зареждане на всички обекти дефинирани в този файл в пространството на имена свързано с пакета)

- Пакет дефиниращ пространство на имена - това са пакети, в папките на които няма файл с името __init__.py

Namespace Packages

- Задали сме import foo - търсим модул или пакет с име "foo" във всяка папка в родителската папка:
- Ако намерим <dir>/foo/__init__.py, зареждаме нормален пакет и прекратяваме.
- Ако няма, но има <dir>/foo.{py,pyc,so,pyd}, зареждаме модул и прекратяваме.
- Ако няма, но има папка <dir>/foo, тя се отбелязва и се продължава
- Ако няма, търсенето се продължава.
- Ако не се намери пакет или модул, но има отбелязана поне една папка, се създава пакет с пространство на имена, за който атрибута __path__ се свързва с итератор, с елементи всички отбелязани папки

Differences between Two Types of Packages

- Обикновените пакети са в една структура от папки, докато частите от пространствата на другите пакети идват от произволни източници
- Пакетите с порции от пространства имат атрибут __path__ , който не може да се променя и се създава автоматично като итератор включващ отделните порции пространства
- Пакетите с порции от пространства нямат __init__.py модул
- Пакетите с порции от пространства имат различен метод на зареждане

Realtive and Absolute Package Imports

- Абсолютно – указва се пълното име включващо всички родителски пакети:

import sounds.effect.echo или алтернативно

from sounds.effect import echo

- Относително – използва се само във варианта с from и използва . или .. за обозначаване на текущата или на родителската папка:
- from .import echo
- from ..import formats

Naming Conventions

- import <име_без_точки> - търси се модул като се търси в пътя дефиниран в променливата sys.path
- from<име_без_точки> import <елем> - аналогично
- from .effect import echo – търси локално в пакет
- Не се допуска използване на . в import
- При намиране на файл и на папка с едно и също име, предимство има папката, ако в нея има файл __init__.py
- Ако в папката няма такъв файл, тогава с предимство е файла.

Installing Packages

- За инсталиране на пакет, който не е наличен в дистрибуцията, изпълнете в команден режим:

pip install package

- За проверка дали pip е наличен:

pip --version

- За да се инсталира pip, първо се изтегля от:

https://pypi.org/project/pip/

- За разглеждане на налични пакети:

https://pypi.python.org/pypi

Installing Packages

- За получаване на списък с наличните пакети:

```
pip list
Result:
Package         Version
-----------------------
mysqlclient 1.3.12
pip             18.1
pymongo     3.6.1
setuptools    39.0.1
```

Installing pip

- Ако pip не е наличен:

```
python -m ensurepip --default-pip
```

- За инсталиране на последна налична версия:

```
python -m pip install --upgrade pip
```

Installing Packages - III

```
pip install "SomeProject"

pip install "SomeProject==1.4"

pip install "SomeProject>=1,<2"

pip install "SomeProject~=1.4.2"
```

Thank's for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD