10. Модули

25 март 2026

Питанка

```
try:
    print([3, 4, 5][6])
except KeyError as e:
    print('I seem to have misplaced my keys')
    print(type(e))
except Exception as e:
    print('Oh man, oh man, oh man…')
    print(type(e))
except IndexError as e:
    print('It tends to happen with indexable stuff, bummer though.')
    print(type(e))
except:
    print('I have no idea what just happened!')
```

```
'Oh man, oh man, oh man…' <class 'IndexError'>
```

Питанка №2

```
class SoSmartContextManager:

    def __init__(self, obj):
        self._obj = obj

    __enter__ = lambda self: None

    def __exit__(self, type, value, tb):
        if type(self._obj) == int:
            print('This was an int')
        else:
            print('This was not an int')

class Notter:

    def __sub__(self, other):
        return other
```

```
Not = Notter()

with Not-SoSmartContextManager(42):
    pass

# Какво показва конзолата?
```

```
# TypeError: 'NoneType' object is not
callable
```

Модули 101

- Всяко нещо в Python е част от някой модул
- Дори нещата, които пишем в интерактивната конзола, са в модул
- Всеки файл с разширение *.py* е модул
- Името на модула е името на файла (без *.py*)
- Всяка директория може да бъде модул
- Разделянето на кода ни в модули ни позволява
  - хем да преизползваме код
  - хем да имаме еднакви имена, зад които стоят различни обекти

import

- Всеки модул се импортира само по веднъж
- При импортирането му се изпълнява целият файл!
- Важно е!

joey.py:

```
print("Hey, how you doin?")
```

Интерактивен интерпрататор:

```
>>> import joey
"Hey, how you doin?"
```

__name__

- Името на модула, duh:

```
>>> print(__name__) # __main__
```

```
>>> import unittest
>>> print(unittest.__name__) # unittest
```

- Модулът, от който е стартирана програмата, е с име `__main__`
- Дори и в интерпретатора?!

```
>>> print(__name__) # __main__
```

__name__ == "main"

- Помните ли това:

```
if __name__ == "__main__":
    ...
```

- Подсигурявате, че текущият файл е изпълнен, а не импортиран

"__main__" в наши модули

Преди: `import __hello__`

Сега: `$ python -m __hello__`

Защото...

```
def main():
    print("Hello world!")

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

"__main__" във вградени модули

- Можете да си пуснете уеб сървър

```
$ python -m http.server --directory .
```

- `$ python -m calendar`
- `$ python -m zipfile` # или tarfile/gzip за работа с архиви
- `$ python -m pdb` # за дебъгване
- `$ python -m unittest` # за тестване
- ... има още много
- ... ако ги import-нем, са доста по-интересни

Какво става при импортиране на модул?

```
import random

print(random) # <module 'random' from '<some_path>'>
```

- На името `random` се присвоява стойност - обект от тип модул
- Всички “глобални имена“ в този модул стават атрибути на обекта
  - достъпват се с точка (`random.shuffle`)
- Имената са “наистина глобални“ само ако сте в *`__main__`*
  - ...иначе стоят зад името на модула, от който идват
  - освен ако не импортирате само част от модула, но за това след малко
- Syntax sugar за:

```
random = __import__("random")
```

Още за import

```
>>> from random import shuffle


>>> import itertools as it
>>> it
<module 'itertools' (built-in)>
>>> it.__name__
'itertools'


>>> from matplotlib import pyplot as plt
```

Можете да бъркате навътре в модулите

- Пример:

```
from module.submodule.deeper.even_deeper.oh_god.please_stop import
oh_this_is_actually_not_so_bad
```

Всичко е просто namespace

- Вариант 1:

```
from django import db
my_model = db.models.Model()
```

- Вариант 2:

```
from django.db.models import Model
my_model = Model()
```

Кога какво?

- Ако ще използвате само едно-две-три неща от целия модул:
  ```
  from module import what_i_need, and_this_other_thing
  ```
- Ако ще използвате много неща, очевидно:
  ```
  import module
  ```
- Ако ще използвате едно нещо на много места:
  ```
  from module import what_i_need_a_lot
  ```
- Следното е напълно валидно:
  ```
  import module
  from module import what_i_need
  ```

Можете и да изсипете всичко с лопатата

```
from pandas import *
```

- По този начин всичко от `pandas` ще бъде "изсипано" в текущия скоуп
  - Това не включва имена, започващи с една или две долни черти
- Нямате добра причина да правите това извън интерактивна конзола или unit тестове
- Не го правете

dir()

```
import itertools
dir(itertools)
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', '_grouper',
'_tee', '_tee_dataobject', 'accumulate', 'chain', 'combinations',
'combinations_with_replacement', 'compress', 'count', 'cycle', 'dropwhile',
'filterfalse', 'groupby', 'islice', 'pairwise', 'permutations', 'product',
'repeat', 'starmap', 'takewhile', 'tee', 'zip_longest']
```

Search Order

- "вградените модули"
- текущата директория
- *sys.path* / *PYTHONPATH*

```
>>> import sys
>>> sys.path
['', '/usr/lib/python310.zip', '/usr/lib/python3.10',
'/usr/lib/python3.10/lib-dynload',
'/home/vbechev/.local/lib/python3.10/site-packages',
'/usr/local/lib/python3.10/dist-packages', '/usr/lib/python3/dist-packages']
```

Пакети

- Модули, изградени само от файлове, е твърде плоско решение
- Можем да групираме няколко файла (модули) в един свръх-модул (у-у-у-у)
- За да направим една директория модул, трябва да създадем *__init__.py* файл вътре
- Това е инициализаторът на свръх-модула (пакета) и се изпълнява преди всичко останало

fingers**/**
  __init__.py
  thumb.py
  index.py
  middle.py
  ...

`__all__`

- Можем в *__init__.py* да дефинираме списък от стрингове `__all__`
- Само имената в него ще бъдат импортнати, ако се използва `*`
    - същото можем да направим и в модул от само един файл

Absolute vs relative

- Ако пишем в пакет, можем да пропуснем търсенето в *sys.path* и директно да импортнем нещо от текущата директория:

```
from . import index
```

- Можем да търсим и в "горния" възел:

```
from .. import toes
```

- Или:

```
from ..toes import big_toe
```

- Правете го пестеливо

.pyc файлове

- *.pyc* са прекомпилирани версии на *.py* файловете ни
- Генерират се при импортиране на файла
- Питонска оптимизация
- В Python 3 стоят в директорията *__pycache__*
- Можем да ги деактивираме генерално
- Не мислим за тях
- (освен при version control)

Модули, които си струва да споменем

- os
- logging
- re
- sys
- json
- random
- itertools
- datetime
- unittest

__future__

- Дава достъп функционалности, които в бъдеще ще са стандартни
- Към момента (python 3.14) няма такива
- ... но всички предишно-бъдещи са още там (заради обратна съвместимост)

За всички фенове на Java

```
>>> from __future__ import braces

SyntaxError: not a chance
```

pip

- *pip* е мениджъра за пакети на Python и идва заедно с него по подразбиране
- *pip* използва централизирано онлайн хранилище, за да намери и инсталира пакета, който ви трябва – https://pypi.org/
- може да инсталира през git, локална файлова система, и други...
- също така се грижи да инсталира и евентуални зависимости на този пакет
- *pip* е модул и като всеки друг модул, той може да се изпълни чрез “`python -m <module>`”
- Възможно е да имате и глобална команда *pip*

```
$ python -m pip install some_module_present_on_pypi_org
```

Питонския зен

```
import this
```

Въпроси?