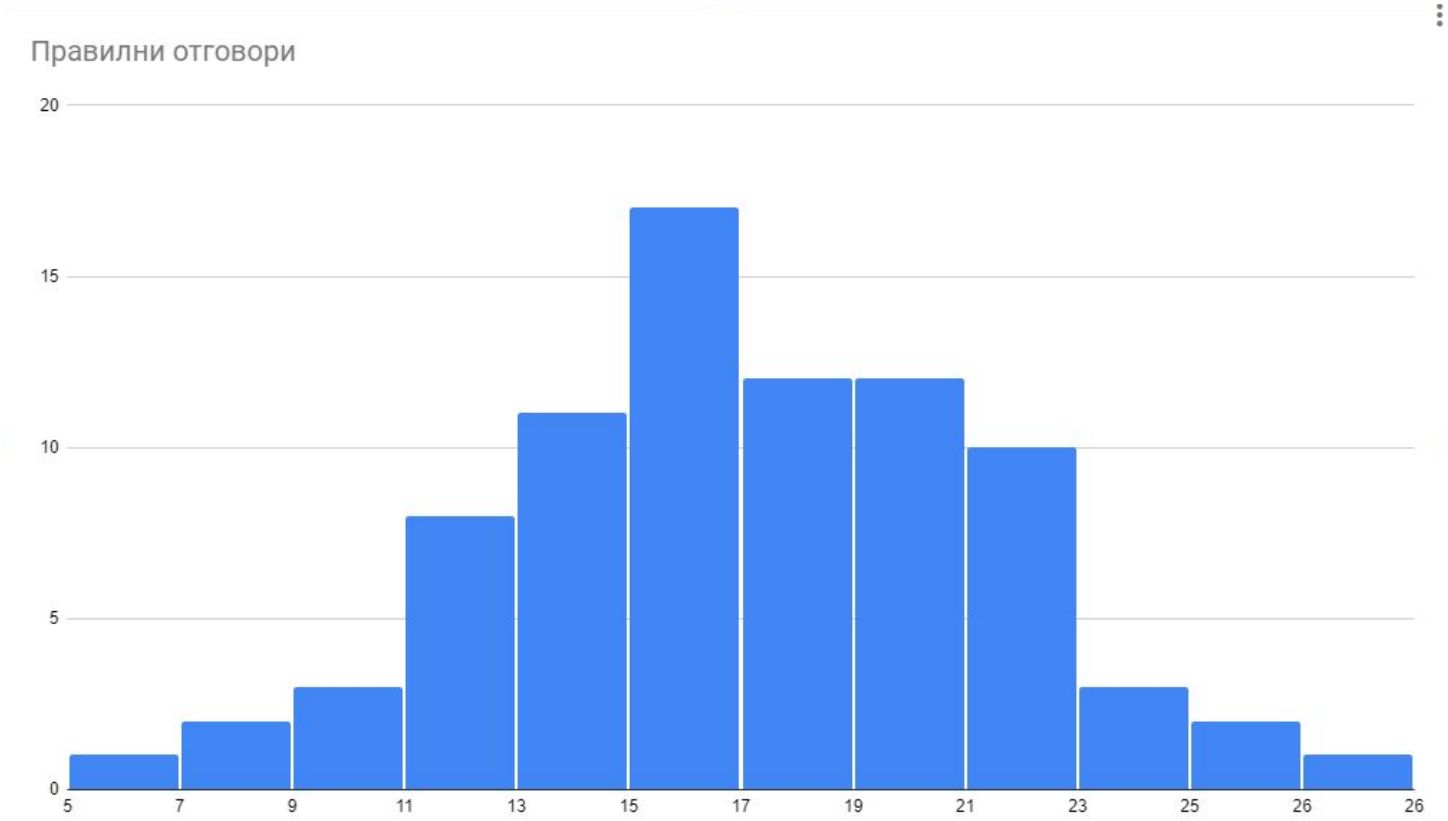
10. Модули

17 ноември 2022

Но първо!

Контролното

Каква е статистиката?



И в крайна сметка какво значи това?

- Знаем, че контролното беше “гадно”
- Приемете го като reality check
- Нашата цел не е да ви оценим, назидаем или скъсаме
- А да научите Python :)
- Така че сме скалирали точките - човекът с 26 верни отговора има 30 точки
- Всички останали скалират спрямо това

Какво знаете (статистически погледнато)?

- Protected & private - 1 грешен отговор
- Могат ли да се дефинират наименувани функции в тялото на друга функция - 82% **верни** отговори
- Колко е средната скорост на ненатоварена лястовица - 81% верни отговори
- Как се създават изключения - 78% верни отговори
- Mutability в Python - 76.5% верни отговори

Какво е добре да си припомните?

- *Забележка: Изключваме въпросите, които просто са по-завъртяни. :)*

- Какво прави ключовата дума raise сама на ред в блока на except - 49% **грешни** отговори
- Как точно работят генераторите - 58% грешни отговори
- Какво представляват контекстните мениджъри - 59% грешни отговори
- __getattr__ и __getattribute__ - 61% грешни отговори (и то средно от 2 въпроса)

По-сериозните “проблеми”

- С кои методи правим инстанциите на клас итеруеми?
- Какво можем да ползваме за ключове на речник?
- Какво се оценява до истина / неистина?
- Декоратори?

Итеруеми обекти - 73% грешни отговори

- Верен отговор:

  Трябват ни __iter__ и __next__.

- __iter__ се вика от iter(x)
- __next__ се вика от next(x)
- Двете в комбинация правят един обект итеруем
- Припомнете си ги

Ключове на речник? - 77% грешни отговори

- Ключовете на речник трябва да са immutable, това го знаете
- На базата на изключване остават само В и Д
- Ако решите, че функциите са immutable обекти - печелите

А. `{}, 1, True`

Б. `dict(), 'абвг', set()`

В. `lambda x: x + 1, (1, 2, 3), 10`

Г. `[1, 2, 3], "abcd", None`

Д. Никое от изброените

Truthiness? - 85% грешни отговори

- Защо?
- Защото:

```
>>> (i for i in range(0))
<generator object <genexpr>
at 0x7fa7d812f270>
>>> bool(_)
True
```

А. None
Б. ""
В. 0j
Г. set([])
Д. (i for i in range(0))
Е. bool([i for i in dict()])
Ж. {}

Декоратори? - само 11 верни отговора

7. Кое от следните твърдения **не** е вярно?

А. Декораторите са функции, които връщат други функции

Б. Декораторите получават само един аргумент, когато се използват

В. Декораторите са обикновени функции

Г. Можем да декорираме само методи и функции

Д. `property`, `classmethod` и `staticmethod` са "вградени"декоратори

Е. Можем да използваме `type` като декоратор

Нашите изводи

- Справили сте се добре (верните отговори не са показателни)
- Защо?
- Защото сте объркали предимно:
  - Неща, за които не сме ви давали да пишете домашни *(генератори, дъндър методи, итератори, изключения, контекстни мениджъри)*
  - Неща, които не сме ви казали в прав текст *(което не значи, че не е имало как да стигнете до верният отговор с мисъл / изключване)*
  - Откровенно шибани въпроси :p
- Подточка “първа” ще я поправим скоро
- Стига толкова, хайде да си говорим за модули

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
- Файлът, от който е стартирана програмата, е с име __main__
- Дори и в интерпретатора?!
- Същото е:

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

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?”
```

В този ред на мисли

```
import __hello__
```

__name__

- Името на модула, duh:

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

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

- Помните ли това:

```
if __name__ == "__main__": …
```

- Е, целта е да се подсигурите, че текущият файл е изпълнен, а не импортиран

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()
```


gospodari.com
КОГА?!

Кога?

- 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 panda import *
```

- По този начин всичко от panda ще бъде "изсипано" в текущия скоуп
- Нямате добра причина да правите това извън интерактивна конзола
- 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 модула 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 *

Absolute vs relative

- 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
- Питонска оптимизация
- В Python 3 стоят в директорията __pycache__
- Можем да ги деактивираме генерално
- Не мислим за тях
- (освен при version control)

import pdb;

- Понякога се налага да дебъгваме с нещо по-мощно от принтове
- Enter pdb:

```
import pdb; pdb.set_trace()
```

- p
- n
- s
- ll / l
- b
- Абе - ?

Въпроси?