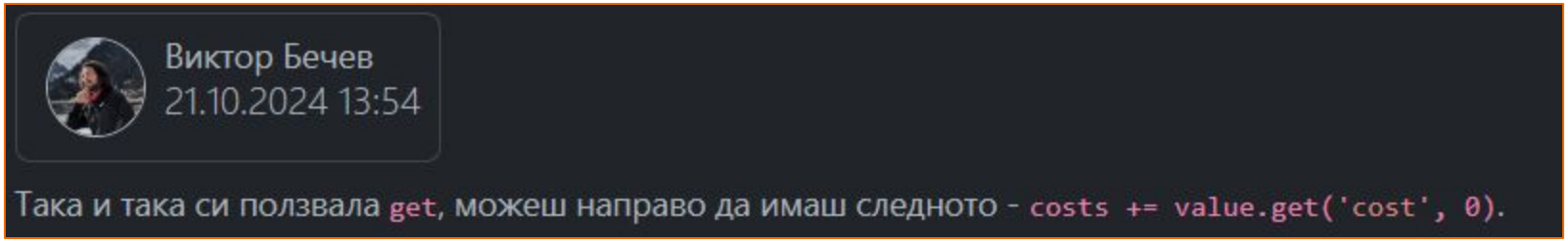
06. Домашни дискусии

11 март 2026

Не такива домашни дискусии

А такива домашни дискусии



Но преди това...

```
SPAM_LOG = {}

def spam(amount):
    def decorator(func):
        def decorated(order):
            spams = ", ".join(["spam"] * amount)
            SPAM_LOG[order] = spams
            return f"{func(order)}, {spams}"
        return decorated
    return decorator

@spam(3)
def order_food(order):
    return f"Here's your order of {order}"

print(order_food("eggs")) # ?  -> "Here's your order of eggs, spam, spam, spam"
print(SPAM_LOG)           # ?  -> {'eggs': 'spam, spam, spam'}
```

Кой иска да опита?

```
@(lambda spam, ham: lambda spam: lambda: spam * 2)("spam", "ham")
def eggs():
    return "eggs"

eggs() # ?
```

```
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    eggs()
  File "<pyshell#20>", line 1, in <lambda>
    @(lambda spam, ham: lambda spam: lambda: spam*2)("spam", "ham")
TypeError: unsupported operand type(s) for *: 'function' and 'int'
```

Втори опит

```
spam = "spam"
def spam(spam):
    def spam():
        global spam
        return spam
    return spam

@spam
def eggs():
    return "eggs"

print(eggs()) # ?  -> <function spam at 0x0000019F453FFCE0>
```

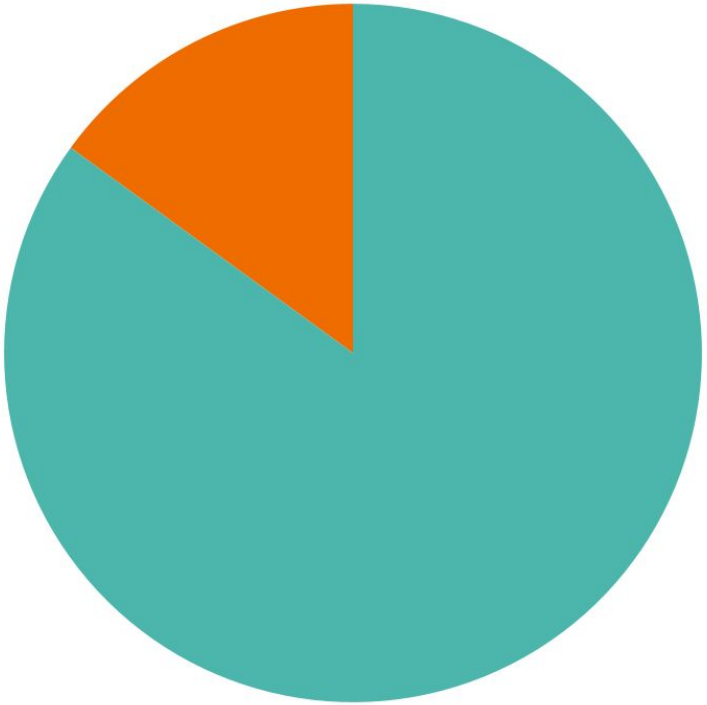
Аджендър (превъзходна степен на адженда)

/по-голямо от адженда, но не чак анджендест/

- Обща информация за процеса с домашните
- Стил
- Потенциални проблеми
- Излишъци
- Рефакториране/оптимизиране
- Шеги `# TODO: Да добавим забавни шеги, защото тези са тъпи`

Agender describes a person with no gender identity. This means the person feels like neither a man nor a woman.

А какво е това?



- Хора, които са получили обратна връзка
- Също хора, които са получили обратна връзка, но в оранжево
- Да, обратна връзка има за всички!
- А и няма как иначе, все пак някои от вас много държат на това:

06.03.2026 00:40 (Check it out bitches)

Обратна връзка

- Стараем се да не даваме обратна връзка, която повлиява на резултата ви.
  - Помагаме да оптимизирате и да следвате стилистични практики.
  - Понякога ви подбутваме, ако грешката е много тъпа, или усетим неразбиране на условието.
- Ще даваме бонус точки по “субективна” преценка.

- Ще махаме точки за неспазване на правила, които сме казали 3+ пъти.

- Ако видим camelCase или кофти индентация, очаквайте червено човече.
- Можем да вземем и за други неща, ако вече сме ви връщали обратна връзка за тях.

Дисекция на домашните...

- В примерите показваме само фрагменти от код с цел визуализация.
- Някои са едно-към-едно от домашни от предните години, други са обобщени примери.

Стил - Излишни интервали

Какъв е проблемът?

```
def function_that_says_ni (*args, **kwargs):
    <някакъв код>
```

- Не слагайте интервали:
  - след име на функция
  - до друг интервал (двойни интервали)
  - преди двете точки (`:`)
  - в края на реда (trailing whitespace)
  - около символа равно, при дефиниране на параметри/аргументи на функция:
    - `sum(iterable, start=0)`

Стил - Липсващи интервали

Какъв е проблемът?

```
total=0
for currency,amount in amounts:
    if currency!="BGN":
        total+=amount*10000//1/10000
```

- Слагайте интервали:
  - след запетаи
  - около математическите и бинарните оператори (`=`, `!=`, `+`, `/`, etc.)
  - с изключение на споменатото в предния слайд:
    - `sum(iterable, start=0)`
  - след двуеточието при изброяване на `{ключ: стойност, ...}` в речник
  - генерално, погледнете това

Стил - Излишни празни редове

Какъв е проблемът?

```
def function_that_says_ni(*args, **kwargs):

    cost = 0


    for name, val in arg_list:
        if type(val) is not dict:
            continue
        <някакъв код>
```

- Не слагайте повече от един празен ред в тяло на функция (никога).
- Избягвайте използването на дори един празен ред.
  ”Extra blank lines may be used **(sparingly)** to separate groups of related functions.”.

Стил - Излишни скоби

Какъв е проблемът?

```
if (name in ("храст", "shrub", "bush")):
    total += cost.get('cost', 0)
return (total)
```

- Не слагайте скоби около условия на `if` (освен ако условието не е на няколко реда).
- Не слагайте скоби при `return`.

Стил - Константи

Какъв е проблемът?

```
def function_that_says_ni(*args, **kwargs):
    shrub_names = ("храст", "shrub", "bush")
    <някакъв код>
    if total_cost > 42:
        return "Ni!"
```

- Добре е да нямате хвърчащи литерали в кода, а да дефинирате константи в началото.
- Казахме ви, в Python константи няма, но по конвенция всяко име, дефинирано с главни букви, е константа (`SHRUB_NAMES`).

Стил - Имена на променливи/фунцкии

Какъв е проблемът?

```
def getTotalCost(random_dict):
    cost = 0
    for arg in random_dict:
        my_cost = arg.get('cost', 0)
        cost += my_cost
    return cost
```

- Именувайте функциите си и променливите си със snake_case. PascalCase ползваме само при имена на класове.
- Именувайте смислено, а не `i`, `md`, `s` и прочие. Някои неща са изключение, защото са очевидни, например `arg`.
- Именувайте разумно, а не `random_dict`, `my_dict`, `unique_simvoli`...
- В общия случай не е добра идея да използвате типа на променливата в нейното име.

Стил - Дълги условия в if

Какъв е проблемът?

```
if ("name" in dictionary and
(type(dictionary["name"]) is str)
and (dictionary["name"].lower() == "храст"
or dictionary["name"].lower() == "bush"
or dictionary["name"].lower() == "shrub")):
    <някакъв код>
```

- Опитайте да не го правите. Има алтернативи - вложени условия, отделяне на логиката във функция, по-кратък метод за да опишете логиката...
- Ако се наложи:
  - ограждате всичко в скоби;
  - логическите оператори се слагат в края на реда, а не в началото на следващия;
  - индентирате така, че да се чете лесно.
- НИКОГА не използвате наклонена черта за пренасяне!

Стил - Повтарящ се слайд

Какъв е проблемът?

```
if (name in ("храст", "shrub", "bush")):
    total += cost.get('cost', 0)
return (total)
```

- Проблемът е, че този слайд вече го видяхте. Само ви проверяваме.
- Но пък е хубаво да не правите същото с кода си.

Стил - Смесени кавички

Какъв е проблемът?

```
CURRENCY = "BGN"
PATRIOTIC = 'ПАТРИОТИЧНА!'
NON_PATRIOTIC = 'НЕПАТРИОТИЧНА!'
```

- Не смесвайте кавичките.
- Не е драма, и все пак.
- Изберете си един вид и ползвайте само тях **в целия проект**.
- Обикновено са двойните.

Стил - Докстрингове

```
def function_that_says_ni(*args, **kwargs):
    """Return the price of a shrubbery or 'Ni!'."""
    <някакъв код>
```

- Едно изречение до 72 символа, което описва какво прави функцията.
- Завършва с точка.
- Дефинира се, обградено от тройни кавички (не апострофи).
- Пишем го в “заповедна форма” (Return a value.), а не в трето лице (Return**s** a value).
- Ако един ред не стига, можем и в повече, но първият трябва да е завършена мисъл, защото понякога се визуализира само той.
- Ако имате повече редове, оставяте един празен ред след първия.
- Дисклеймър - ние ги пропускаме в слайдовете, за да пестим място.

Потенциални проблеми - Предефиниране на имена

Какъв е проблемът?

```
def function_that_says_ni(*args, **kwargs):
    sum = 0
    <някакъв код>
```

- Не предефинирайте built-in имена (`sum`, `dict`, `next`...).
- С времето ще се научите.
- Ако не - сложете си линтер. Той знае.

Потенциални проблеми - Скорост

Какъв е проблемът?

```
if name in ['shrub', 'bush', 'храст']:
    <някакъв код>
```

- Проверка дали стойност се среща в списък обхожда целият списък
- Използвайте `set`, които правят това с константна сложност.
  - Речникът също, но не е удачен за този пример.
    - `name in some_dict` не е същото като `name in some_dict.keys()`

Потенциални проблеми- Side effect

Какъв е проблемът?

```
def is_shrub(dictionary):
    <някакъв код>
    if "cost" in dictionary:
        dictionary["cost"] = 0
    <някакъв код>
```

- Не закачайте обектите, подадени като параметри.
- По-добре направете копие и го върнете.

Излишъци - Оператора in

Какъв е проблемът?

```
dictionary["name"].lower() == "bush" or dictionary["name"].lower() ==
"храст" or dictionary["name"].lower() == "shrub"
```

- Възползвайте се от `in`.
  - `dictionary["name"].lower() in {"храст", "shrub", "bush"}`
- Не само спестява многократното извикване на `.lower()`, но е и по-кратко и ясно.
- И по бързо, както вече видяхме.

Излишъци - Else след return

Какъв е проблемът?

```
if is_nice:
    return formatted_cost
else:
    return "Ni!"
```

- След като функцията срещне `return`, останалото не се изпълнява.
- Един ред, но “капка по капка - вир”.
- А най-добре - използвайте тернарни изрази!
  - `return formatted_cost if is_nice else "Ni!"`

Излишъци - if, bool, return

Какъв е проблемът?

```
if is_nice:
    return True
else:
    return False
```

- Ако правите проверка като горната с булева променлива, не ви трябва `if`:
  - `return is_nice` е същото

Излишъци - Променливи

Какъв е проблемът?

```
def get_cost_of_shrub(shrub):
    cost = shrub.get('cost', 0)
    return cost
```

- Ако ще ползвате променлива само веднъж, едва ли има смисъл от нея.
- Не само ако веднага я връщате, а ако я използвате веднъж след 1-2 реда и после никога повече.
- Изключение прави променлива, чиято дефиниция е дълга като код, но тази по-горе не е достатъчно дълга.

Излишъци - Кратки функции

Какъв е проблемът?

```
def get_cost_of_shrub(shrub):
    return shrub.get('cost', 0)
```

- Ако ще ползвате функция само веднъж и съдържанието ѝ е толкова просто, колкото е горното, едва ли има смисъл от нея.

Излишъци - .get()

Какъв е проблемът?

```
if 'cost' in shrub:
    cost += shrub['cost']
```

- Речниците имат метод `.get()`, с който можете да дефинирате резултат, ако поискате несъществуващ ключ:
    - `cost += shrub.get('cost', 0)`

Излишъци - Java

Какъв е проблемът?

```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

- Тук има прекалено много излишъци. Ползвайте Python:
    - `print("Hello, World!")`

Рефакториране/оптимизиране - речници

Какъв е проблемът?

```
for item in items:
    for key in some_things_in_a_dict:
        if item == key:
            ...
```

- Това е начин да търсите в речник екстремно бавно - *O(n)* vs **O(1)**.
- На практика е същото като:
  `some_things_in_a_dict.get(item)`

Рефакториране/оптимизиране - “излишен” for

Какъв е проблемът?

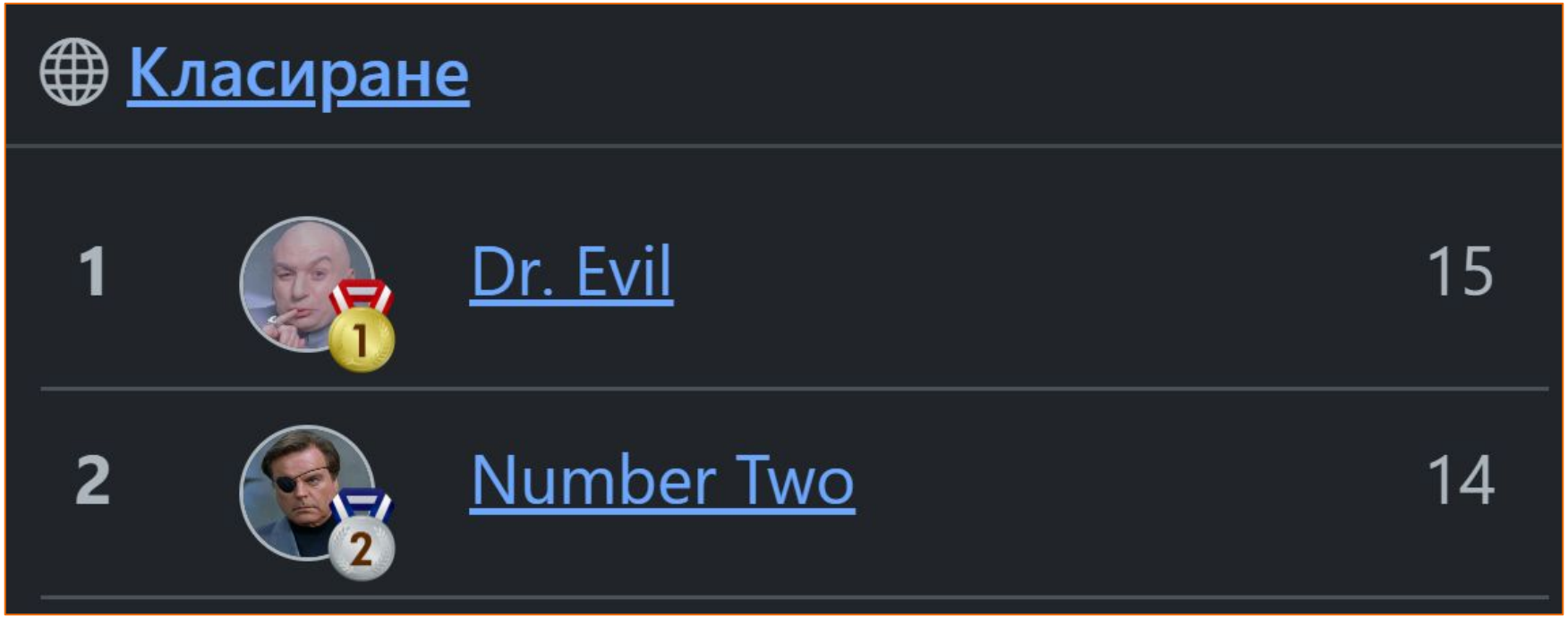
```
words_to_censor = []
for word in words:
    if "кор" in word:
        words_to_censor.append(word)
```

- `for`-овете са окей, но често има по-добри инструменти.
- Понякога това са вградени функции, понякога са просто comprehensions:
  `words_to_censor = [word for word in words if "кор" in word]`

Защо да ни пука?

- По-четим код.
- Спестяване на излишни операции - т.е. на време и място.
- Спестяване на време за писане.
- По-дълбоко разбиране на езика.
- PEP8
- Останалите програмисти ви обичат.
- Ние предупредихме още в нулевата лекция: Grammar Nazis.

Защо да ни пука?




“And now for something completely different.”
Monty Python

06½. (.)(.) P

11 март 2026

От идейна гледна точка

Разбирайте „за програмирането генерално“

- Абстракция/ Abstraction
- Енкапсулация/ Encapsulation
- Модулярност/ Modularity

Абстракция / Abstraction

Според википедия:

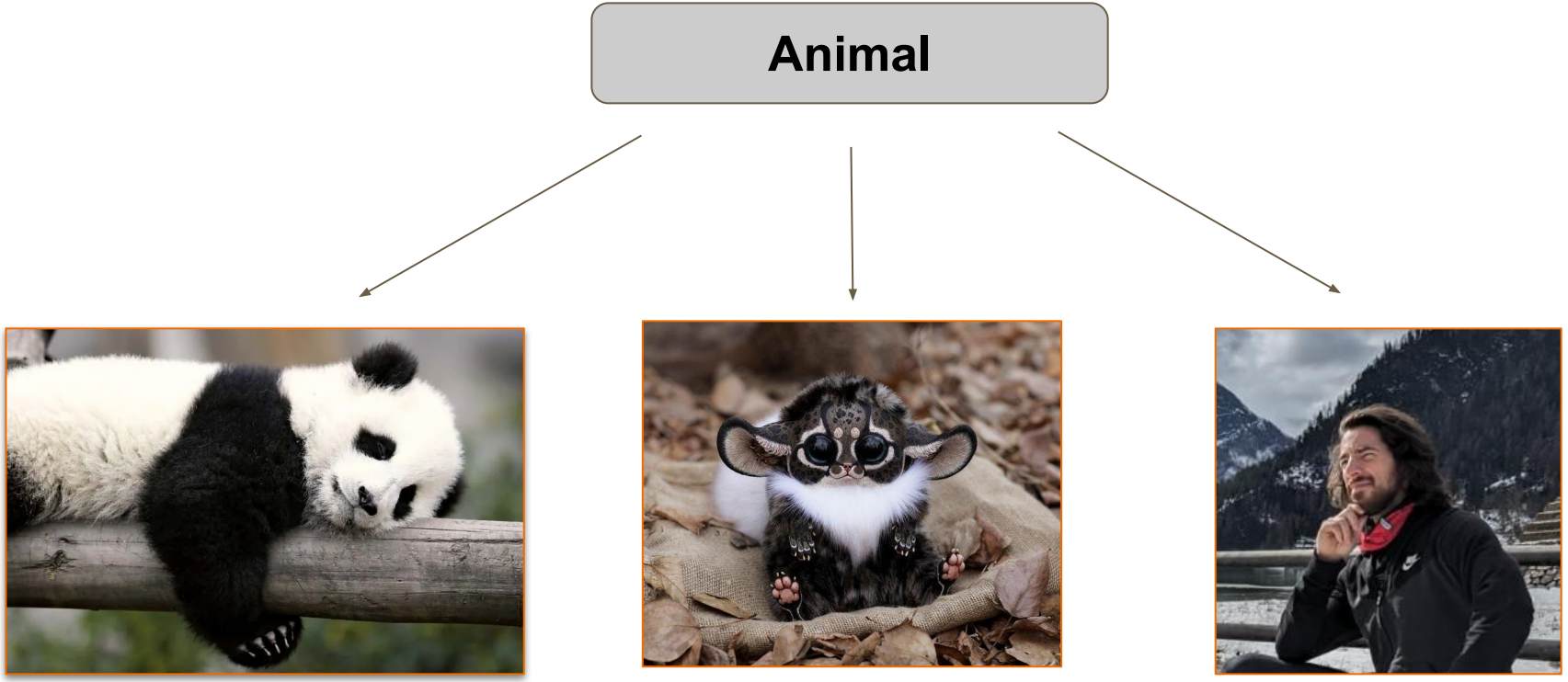
Abstraction in its main sense is a conceptual process by which general rules and concepts are derived from the usage and classification of specific examples, literal ("real" or "concrete") signifiers, first principles, or other methods. "An abstraction" is the product of this process—a concept that acts as a super-categorical noun for all subordinate concepts, and connects any related concepts as a group, field, or category.

Абстракция / Abstraction

В нашия случай:

[...] a technique for managing complexity of computer systems. It works by establishing a level of complexity on which a person interacts with the system, suppressing the more complex details below the current level. The programmer works with an idealized interface (usually well defined) and can add additional levels of functionality that would otherwise be too complex to handle.

Визуален пример



Енкапсулация / Encapsulation

Encapsulation is the packing of data and functions into a single component.

Има и други начини за реализиране, но ООП е далеч най-популярният.

Information Hiding

Abstraction and encapsulation are complementary concepts: abstraction focuses on the observable behavior of an object... encapsulation focuses upon the implementation that gives rise to this behavior... encapsulation is most often achieved through information hiding, which is the process of hiding all of the secrets of object that do not contribute to its essential characteristics.

Модулярност / Modularity

Механизъм за организиране на сходна логика работеща върху свързани видове данни, обработваща сходни видове процеси от моделирания свят в добре обособени и ясно разделени парчета от кода ни.

Сега сериозно

Разбирайте „конкретно за python“

1. Всичко е обект
2. Обектите са отворени
3. Класовете са отворени

Последните две с някои малки уговорки, които обаче рядко ще ви интересуват

Абсолютно всичко е обект... голяма изненада за вас

```
>>> type(42)
<class 'int'>

>>> type([])
<class 'list'>

>>> type([]) is list
True

>>> type(list)
<class 'type'>

>>> type(list) is type
True
```

Дали?

```
>>> type(type)
???
<class 'type'>
```

Класове

Класове се създават с ключовата дума `class`, след което всяка функция, дефинирана в тялото на класа, е метод, а всяка променлива е клас променлива.

Примерен Клас

```
class Vector:

    def __init__(self, x, y):
        self.x = x
        self.y = y

spam = Vector(1.0, 1.0)

print(spam.x)
```

1. „Конструкторът“ се казва `__init__`, той не връща стойност
2. Първият аргумент на методите винаги е инстанцията, върху която се извикват, той може да се казва всякак, но винаги се казва `self`, иначе губите огромни количества точки/колегите ви ви мразят/никой не иска да си играе с вас в пясъчника
3. Атрибутите („член-променливите“/„член-данните“) не се нуждаят от декларации (обектите са отворени)
4. Инстанцираме клас, като го „извикаме“ със съответните аргументи, които очаква `__init__` метода му и като резултат получаваме новоконстурирания обект

Забележка

„Конструктор“ е думата, с която сте свикнали, но в случая далеч по-подходяща е „инициализатор“, както си личи от името

Въпреки, че при инстанциране на обект подаваме аргументите, които очаква инциализатора, той не е първия метод, който се извиква в този момент, но засега не влагайте много мисъл в това, просто го имайте предвид

Инстанцирането на обект синтактично е еднакво с викане на функция (кръгли скоби с аргументи). Някои неща, които мислите че са викания на функции, могат да се окажат инстанцирания на обекти.

И последно, преди да се разотидем

- По темата с контролното...
- За момента можем да ви ориентираме с точност до 8 дни - между **25.03** и **01.04**.
- С други думи, възможни дати са **25.03**, **30.03** или **01.04**.
- Повече детайли - като наближи *(но така или иначе основното го знаете)*

Въпроси?