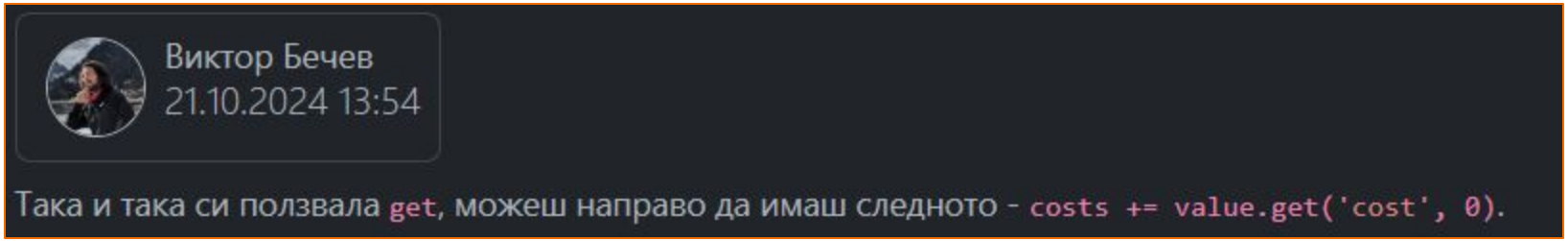
05. Домашни дискусии

22 октомври 2024

Не такива домашни дискусии

А такива домашни дискусии



Но преди това!

Преводи

- Да речем, че сме понаписали определен брой функции, които връщат като резултат низ
- Обаче ни идва нагости братовчед ни от Щатите
- А низовете ни са на Български!
- Нека да го улесним
- *П.П. Това далеч не е най-добрият вариант за имплементиране на internationalization / localization*

Нашата функция

Нека да дефинираме проста функция, която връща низ:

```
def кон():
    return "отиде коня у ряката"

кон() # 'отиде коня у ряката'
```

Искаме следното:

```
@translate
def кон():
    return "отиде коня у ряката"

кон() # 'went the horse into the river'
```

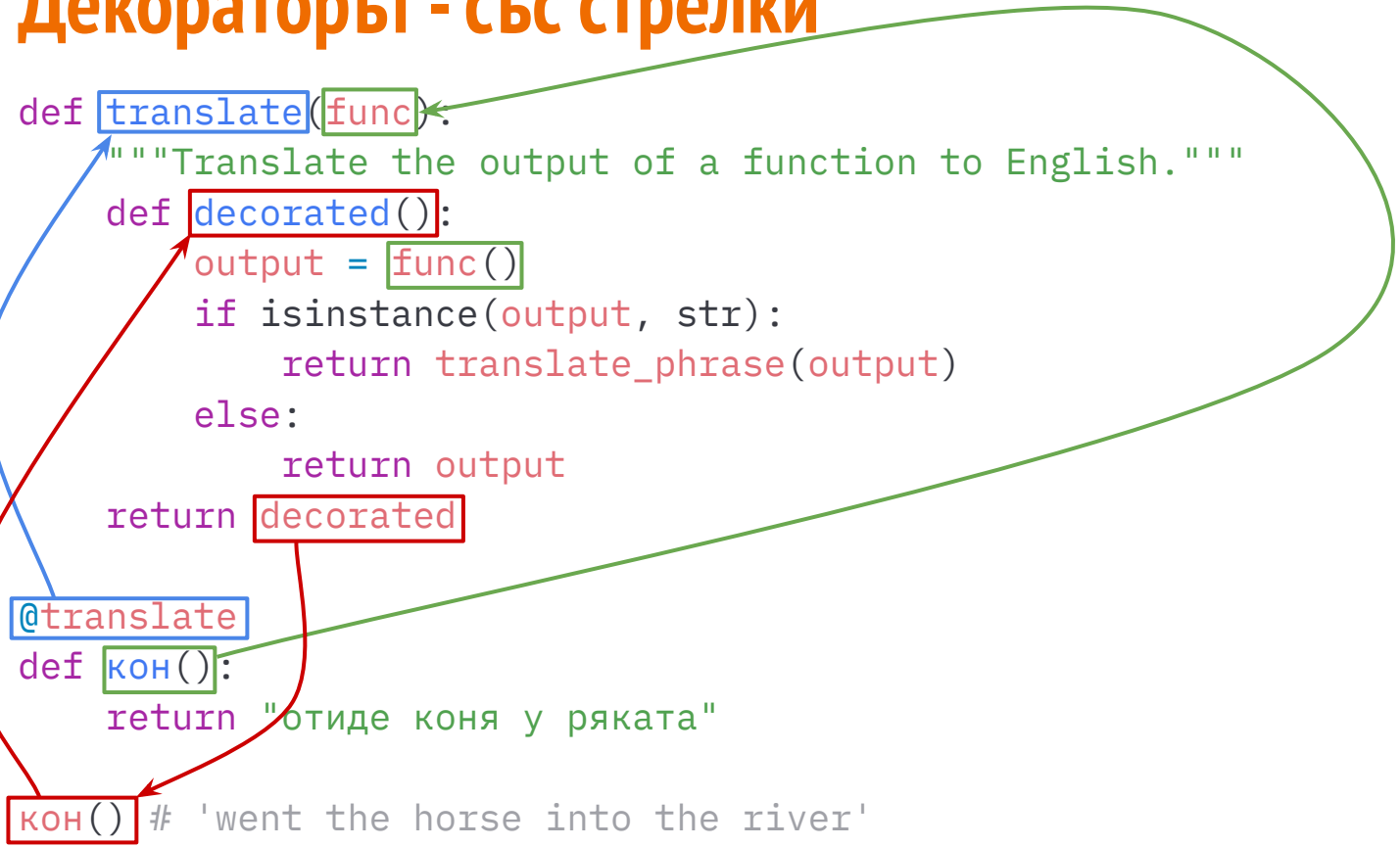
Декораторът

```
def translate(func):
    """Translate the output of a function to English."""
    def decorated():
        output = func()
        if isinstance(output, str):
            return translate_phrase(output)
        else:
            return output
    return decorated

@translate
def кон():
    return "отиде коня у ряката"

кон() # 'went the horse into the river'
```

Декораторът - със стрелки



```
def translate(func):
    """Translate the output of a function to English."""
    def decorated():
        output = func()
        if isinstance(output, str):
            return translate_phrase(output)
        else:
            return output
    return decorated

@translate
def кон():
    return "отиде коня у ряката"

кон() # 'went the horse into the river'
```

translate_phrase

За пълнота, ето ви спомагателната функция:

```
SENTENCE_TRANSLATIONS = {"отиде коня у ряката": "went the horse into the river"}
WORD_TRANSLATIONS = {"отиде": "went", "коня": "the horse",
                     "у": "into", "ряката": "the river"}

def translate_phrase(phrase):
    full_translation = SENTENCE_TRANSLATIONS.get(phrase)
    if full_translation is None:
        words = phrase.split()
        words = [WORD_TRANSLATIONS.get(word, word) for word in words]
        full_translation = " ".join(words)
    return full_translation
```

Забележка - спрямо промените, които ще правим, тази функция също ще се променя. Приемайте я като абстракция, за целите ни не е важна нейната имплементация.

По-универсално решение

Добре, а ако искаме да декорираме функции, които приемат входни параметри?

```
@translate
def кон(number_of_horses):
    return f"отидоха {number_of_horses} коня у ряката"

кон(5) # ???
```

```
Traceback (most recent call last):
  File "D:/Dev/FMI/Python/decorators_xample_args_unfixed.py", line 27, in
<module>
    print(кон(5))
TypeError: translate.<locals>.decorated() takes 0 positional arguments but 1
was given
```

Защо?

```
def translate(func):
    """Translate the output of a function to English."""
    def decorated():
        output = func()
        if isinstance(output, str):
            return translate_phrase(output)
        else:
            return output
    return decorated

@translate
def кон(number_of_horses):
    return f"отидоха {number_of_horses} коня у ряката"

кон(5) # ....decorated() takes 0 positional arguments but 1 was given
```

Решението

```
def translate(func):
    """Translate the output of a function to English."""
    def decorated(*args, **kwargs):
        output = func(*args, **kwargs)
        if isinstance(output, str):
            return translate_phrase(output)
        else:
            return output
    return decorated

@translate
def кон(number_of_horses):
    return f"отидоха {number_of_horses} коня у ряката"

кон(5) # 'went 5 the horse into the river'
```

Братовчедът от Япония

Добре, а ако имаме функции, изходът от които искаме да превеждаме на различен език?

```
@translate("jp")
def кон(number_of_horses):
    return f"отидоха {number_of_horses} коня у ряката"

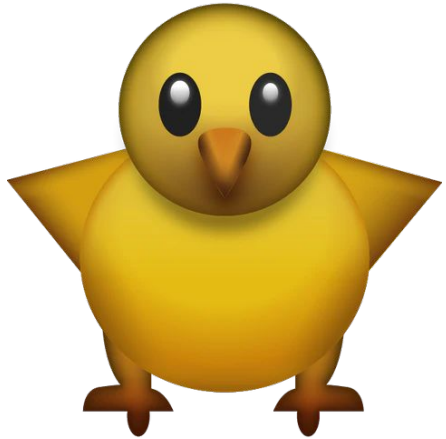
кон(5) # ???
```

```
Traceback (most recent call last):
  ...
  File "D:/Dev/FMI/Python/decorators_xample_jp_not_callable.py", line 16, in
decorated
    output = func(*args, **kwargs)
TypeError: 'str' object is not callable
```

Защо?

```
def translate(func):
    """Translate the output of a function to English."""
    def decorated():
        output = func()
        if isinstance(output, str):
            return translate_phrase(output)
        else:
            return output
    return decorated

@translate("jp")
def кон(number_of_horses):
    return f"отидоха {number_of_horses} коня у ряката"
```



Решението

```
def translate(language="en"):
    """Translate the output of a function to a specific language."""
    def translate_to(func):
        def decorated(*args, **kwargs):
            output = func(*args, **kwargs)
            if isinstance(output, str):
                return translate_phrase(output, language)
            else:
                return output
        return decorated
    return translate_to

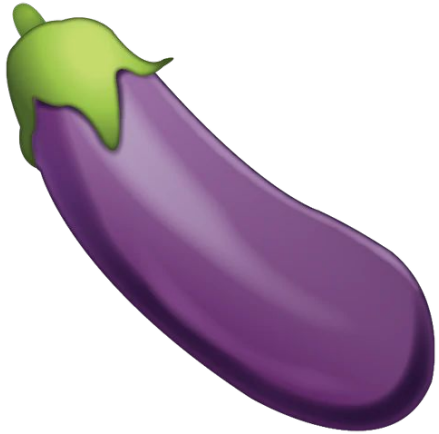
@translate("jp")
def кон(number_of_horses):
    return f"отидоха {number_of_horses} коня у ряката"


кон(5) # '行った 5 馬 で 川'
```

И за финал

```
@translate("вр")
def народна_мъдрост():
    return "Не се опитвай да угодиш на всички."

народна_мъдрост()
# 'Не моа са аресаш на сички. Не си рикия!'
```



Един бонус - истинският деКОРатор

```
def de_кор_ator(fun):
    def clean_fun():
        return fun().replace('кор', '***')
    return clean_fun

@de_кор_ator
def student_statement():
    return "Да търпиш глупостите на лекторите е голям кор!"

print(student_statement()) # 'Да търпиш глупостите на лекторите е голям ***!'
```

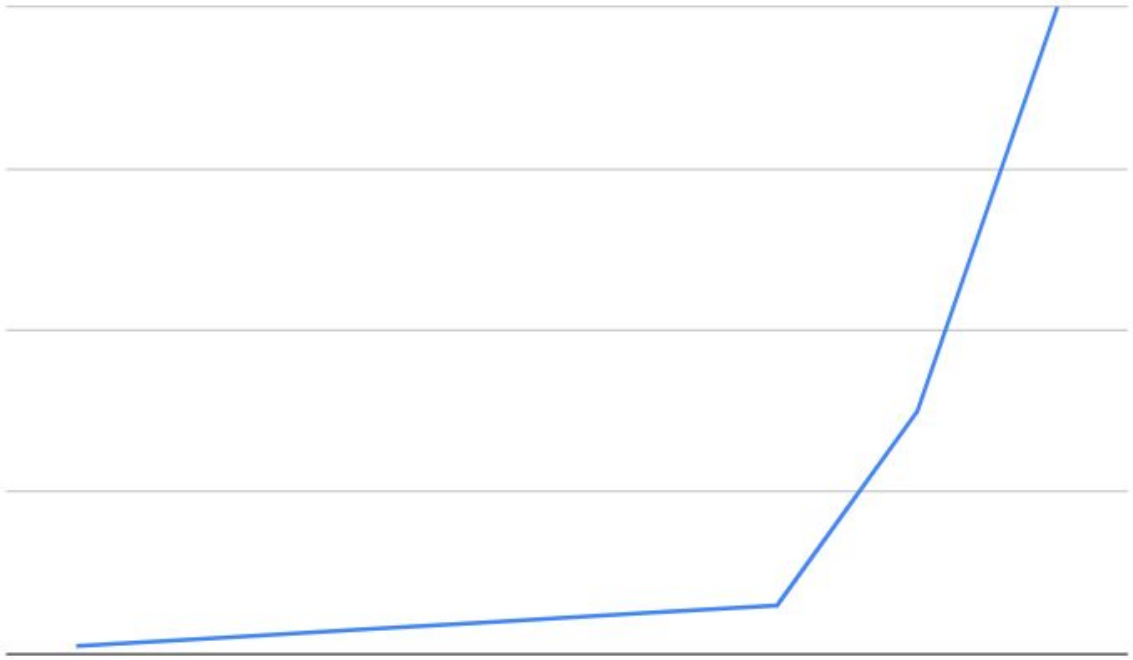
Аджендър (превъзходна степен на адженда)

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

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

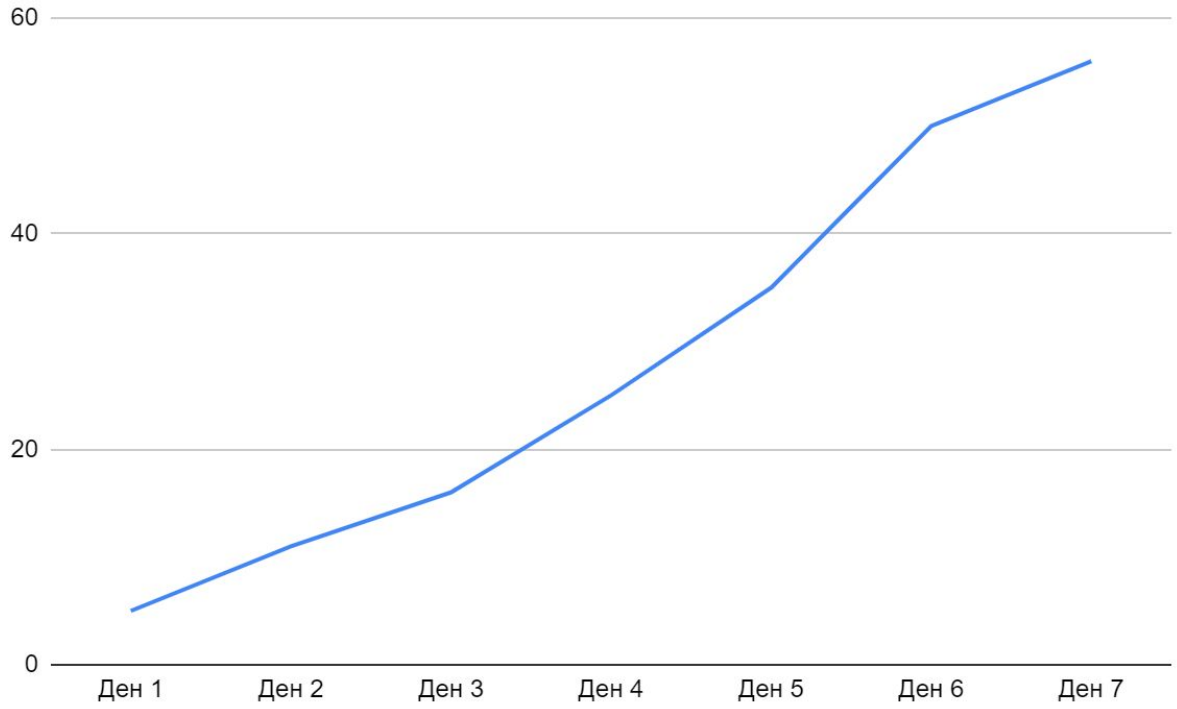
Agender describes a person with no gender identity. This means the person feels like neither a man nor a woman.

Преди да започнем, един въпрос. Какво е това?

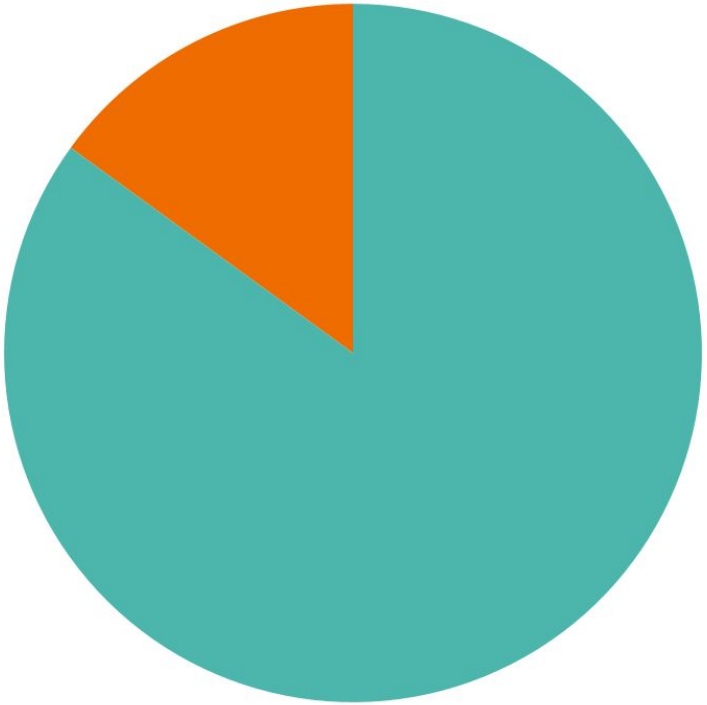


Прогресът по предаване на първото домашно от зимния семестър на 22/23

Тази година е далеч по-добре. Браво!



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!

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.
- Даваме бонус точки по “субективна” преценка.

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

- Ако видим CamelCase или използване на табулации, очаквайте червено човече.
  - Можем да вземем и за други неща, ако вече сме ви връщали обратна връзка за тях.
- Четете внимателно условието. Няколко човека има проблем с гранична стойност при сума=42.

Дисекция на домашните...

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

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.

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.

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.
- 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 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 проверявам.

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.

Потенциални проблеми - Идентитет

Какъв е проблемът?

```
if type(arg) == dict:
    <някакъв код>
```

- Обекти, които съществуват само веднъж, се сравняват по идентитет (`id()`):
  - Built-in класове;
  - Сингълтъни като True, False, None;
  - Ваши собствени дефиниции.
- Сравнението по идентитет се имплементира от оператора `is`.
- В бъдеще ще говорим, че е по-добре да се използва `isinstance()`.

Излишъци - Float вместо int

Какъв е проблемът?

```
total_cost = 0.0
```

- Можете да започнете и с целочислена нула (0).
- Ако добавите `float` към нея, резултатът ще е `float`.
- Не е голяма оптимизация, но е от сърце!

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”.

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 - Цикли

Какъв е проблемът?

```
# words == ['name', 'george', 'bush']
unique_letters = set()
for word in words:
    for char in word:
        unique_letters.add(char)
```

- Доста от алгоритмите, които сте свикнали да имплементирате с цикли, имат по-удобни варианти в Python.
- Горното може да се напише и така:

```
for word in words:
    unique_letters.update(word)
```

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.
- 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 - литерали и индекси

Какъв е проблемът?

```
bush = (name, cost)
no_name_bushes.append(bush)
<някакъв код>
get_sum_of_elements_at_index(no_name_bushes, 1)
```

- В горния пример правим `tuple`-и, които после индексираме - трудно се следи кой индекс за какво е.
- Избягвайте използване на литерали в кода си (експлицитно написани стойности като индекси, стрингове за ключове или магически числа, чието значение никой не знае).
- Дефинирайте константи и пазете данните по по-разумен начин.

Рефакториране/оптимизиране - планувайте

Какъв е проблемът?

```
# words == ['name', 'george', 'bush']
unique_letters = set()
for word in words:
    unique_letters.update(word)
```

- Ако не събирате имената като списък от стрингове, а просто ги конкатенирате, няма да се налага после да циклите в тях:
    - ```
      # words == 'namegeorgebush'
      unique_letters = set(words)
      ```
- Разбира се, можете още при събиране, вместо конкатенация, да ги добавяте в предефинират `set`.
      ```

Рефакториране/оптимизиране - обединение

Какъв е проблемът?

```
for element in args:
    if type(element) is dict and is_a_valid_bush_dict(element):
        total_cost += element.get("cost", 0)

for key, value in kwargs.items():
    if type(value) is dict and is_a_valid_bush_dict(value):
        total_cost += value.get("cost", 0)
        shrub_names.append(key)
```

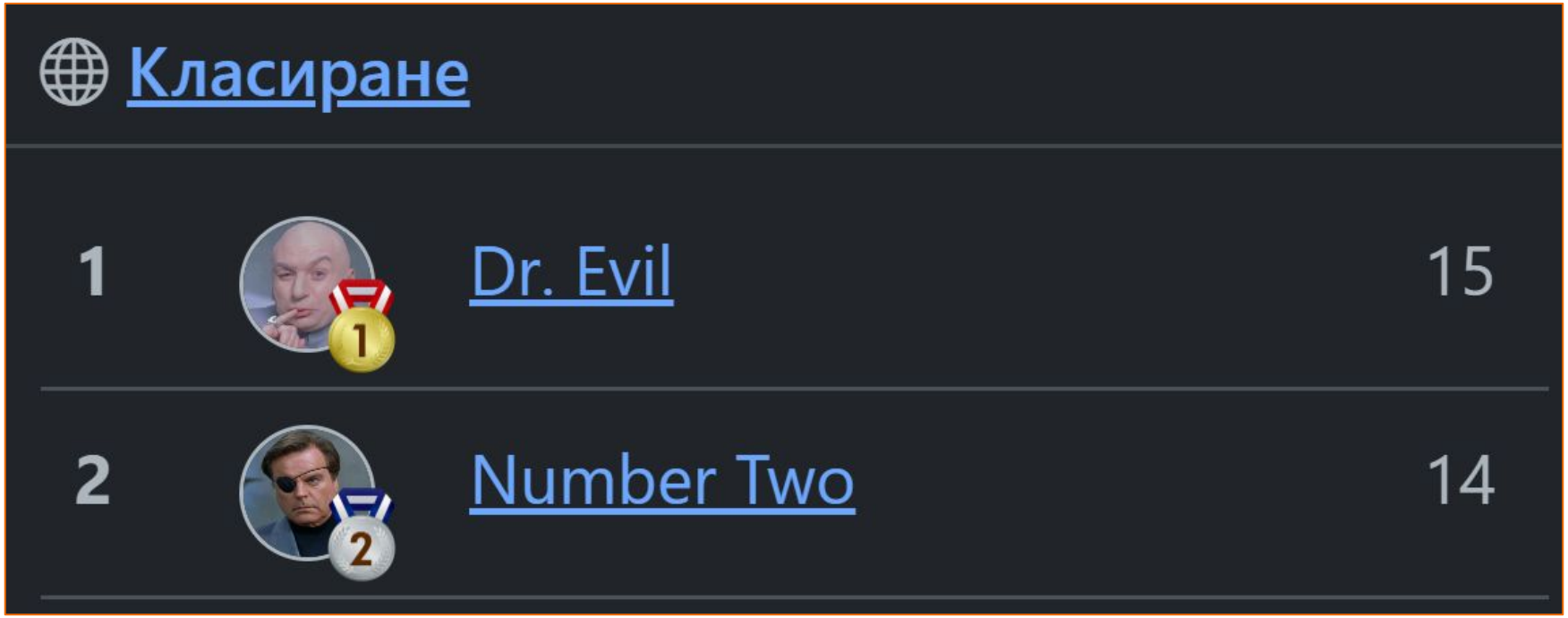
- Това са два цикъла, които правят горе-долу едно и също нещо.
- Ако унифицирате данните, върху които итерирате, можете да ги обедините:

```
arg_list = list(map(lambda x: ('', x), args)) + list(kwargs.items())
for key, value in arg_list:
    <някакъв код>
```

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?



Въпроси?