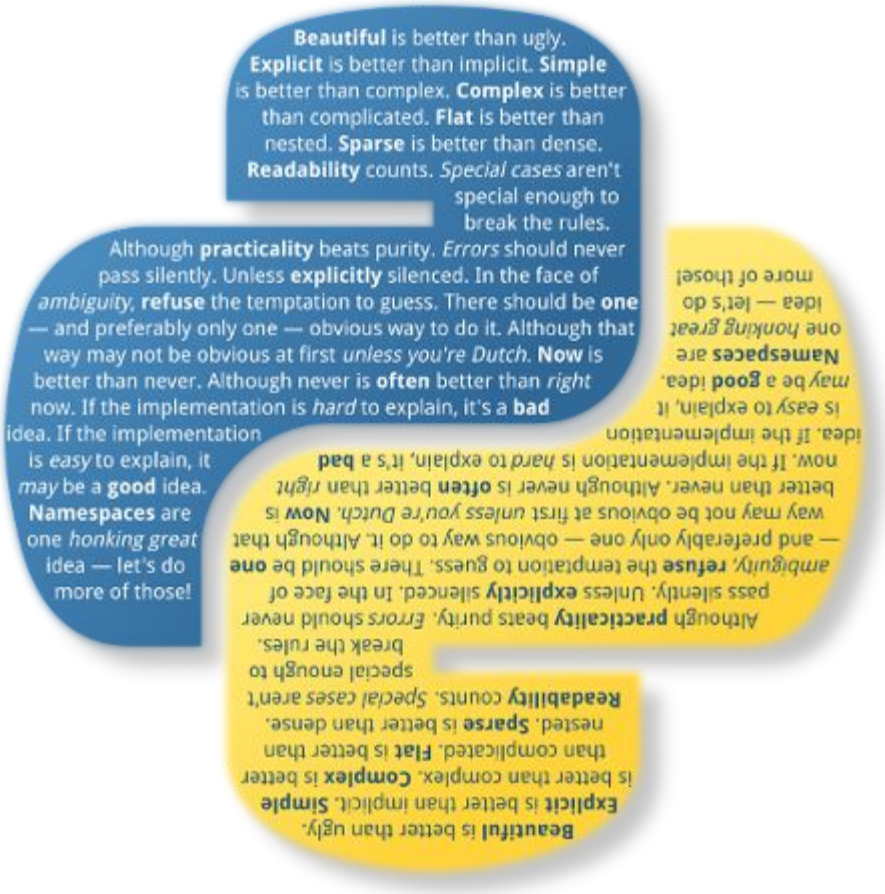
01. Въведение в Python

11 октомври 2022

Първо, организационен слайд

- Сайт още няма, ще ви кажем като има


Beautiful is better than ugly. **Explicit** is better than implicit. **Simple** is better than complex. **Complex** is better than complicated. **Flat** is better than nested. **Sparse** is better than dense. **Readability** counts. *Special cases* aren't special enough to break the rules.
Although **practicality** beats purity. *Errors* should never pass silently. Unless **explicitly** silenced. In the face of *ambiguity*, **refuse** the temptation to guess. There should be **one** — and preferably only one — obvious way to do it. Although that way may not be obvious at first *unless you're Dutch*. **Now** is better than never. Although never is **often** better than *right* now. If the implementation is *hard* to explain, it's a **bad** idea. If the implementation is *easy* to explain, it *may* be a **good** idea. **Namespaces** are one *honking great* idea — let's do more of those!

Среда за програмиране

- Който е с Windows - светият граал е наличен на https://www.python.org/downloads/
- Или “download python”@Google
- Повреме на инсталация - “Add Python to PATH“
- Който е с Linux - `sudo apt install python3` (или еквивалента на това)

Среда за програмиране

- PyCharm
- VSCode
- Sublime
- IDLE
- Notepad (++)
- CMD / Terminal

- *vim (само за смелите)*
- *emacs (само за тези с железни кутрета)*

Къде отива кода?

- Код се пише в `.py` файлове (например `gameoflife.py`).
- Изпълнява се с `python gameoflife.py`
- Можем да пишем код интерактивно като пуснем `python` без аргументи.

Python е предсказуем

Когато не сте сигурни, просто пробвайте.

```
$ python
Python 3.10.7 (tags/v3.10.7:6cc6b13, Sep  5 2022, 14:08:36)
[MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 13 + 10
23
>>> a = 13
>>> b = a + 10
>>> print(b)
23
>>> a * 2
26
>>> a ** 2
169
>>> "hello" + ', ' + "world"
'hello, world'
```

Първа помощ

В интерактивната конзола, `help()` показва документацията на всяка функция, клас или тип.

```
>>> help(str)
>>> help(5)
>>> help(SomeClass)
>>> help(some_function)
```

Един ред код

```
>>> my_var = 'spam'.upper()
>>> print(my_var)
SPAM
```

- Съдържа един израз
- **никога не завършва с** `;`
- Всичко след `#` е коментар

Типове: int

- Цели числа, положителни и отрицателни
- Стандартни операции: `+`, `-`, `*`, `/`, `%`, `**` (степенуване)
- Без максимален размер
- Може да пробваме `2 ** 4 ** 8`

Типове: float

- Числа с плаваща запетая(точка?) `3.1452`
- По всичко друго приличат на целите числа
- `0.1 + 0.2` = ?

Типове: complex

- Още един вид число - комплексно
- Пишат се така: `(2+3j)`
- Да, `j`, а не `i`

Типове: complex

```
>>> a = 1j * 1j
(-1+0j)
```

Типове: str

```
>>> "hello".upper()
"HELLO"
>>> len("абвгдеж")
7
>>> "hello"[1]
"E"
>>> help(str)
```

- Текстови низове с произволна дължина
- Единични или двойни кавички
- Unicode навсякъде!!!!
- Поддържат `\n`, `\t` и пр.

Типове: bool

- `True` и `False`
- **NB!** главните букви

Типове: None

- Като `null` в другите езици.
- Когато една функция не върне нищо, тя връща `None`.
- Използвайте го за да кажете "нищо" или "няма"

Типове

- Всяка стойност има тип

```
>>> type(5.5)
<class 'float'>
>>> type("баба")
<class 'str'>
```

- Включително и функциите

```
>>> type(len)
<class 'builtin_function_or_method'>
```

Типове

- Всяка стойност е обект и има клас, включително функциите
- **Всичко** в python е обект, включително функциите **и типовете**!
- Можем да проверим типа на един обект с функцията `type()`

Типове

`type` е функция

⇒ `type` е обект

⇒ `type` си има тип

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

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

It's turtles all the way down...

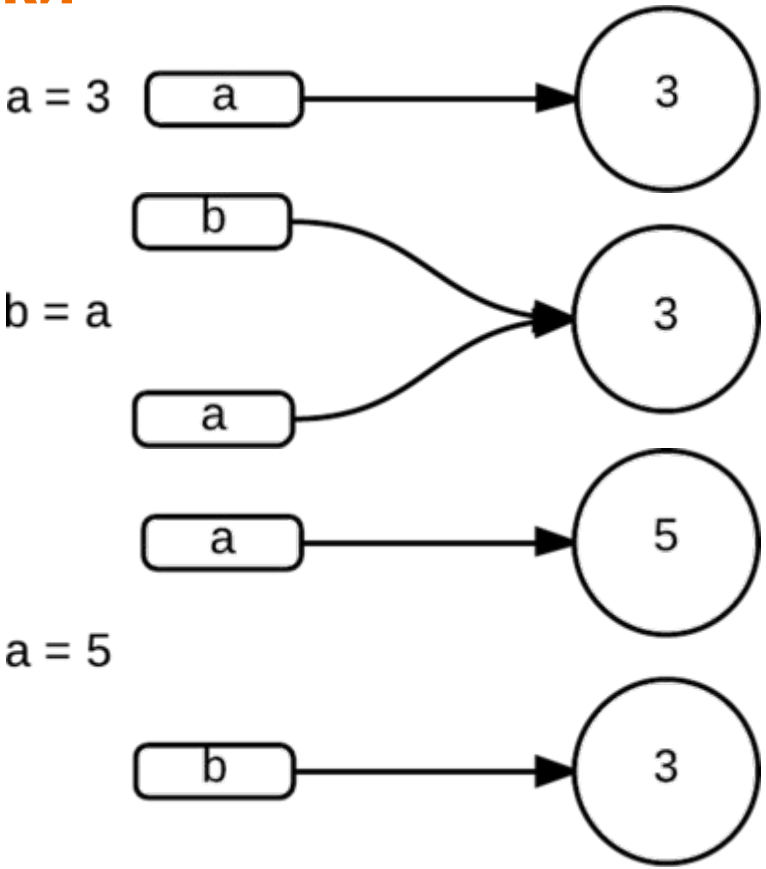
Имена

Можем да присвоим стойност на име, така създаваме променлива

Python е динамичен език, стойностите имат тип, но не и имената

```
>>> a = 5
>>> type(a)
<class 'int'>
>>> a = 'test'
>>> type(a)
<class 'str'>
```

Имена в картинки



Структури от данни

- Списък - `list`
- Речник - `dict`
- Tuple - `tuple` a.k.a кортеж
- Множество - `set`
- `help` е ваш пръв приятел!

Списъци

```
>>> my_list = []         # препоръчително!
>>> my_list = list()   # иначе може и така
```

- Списък = `list` = масив = array
- Mutable и без фиксирана дължина
- Бързи за търсене по индекс, бавни за търсене по стойност
- Гарантиран ред
- Не е нужно елементите да са от еднакъв тип (т.е. списъците са хетерогенни)

Списъци

```
>>> my_list = []
>>> my_list.append('word')
>>> my_list.append(5)
>>> my_list.append(False)


>>> my_list[1] == 5
True
```

Списъци

```
>>> my_other_list = ['foo', 'bar', 'quux']
>>> len(my_other_list)
3
>>> del my_other_list[1]

>>> my_other_list
['foo', 'quux']

>>> 'foo' in my_other_list
True
>>> False in my_list
True
>>> 'spam' in my_list
False
```

Речник (dict)

```
>>> ages = {'Кай': 2, 'Бобо': 3}
>>> ages['Стефан'] = 3
>>> ages['Кирил'] = 42
>>> ages['Николай'] = 23

>>> ages['Кирил']
42
>>> 'Николай' in ages
True

>>> ages.get('Стамат')
None
>>> ages.get('Стамат', 'няма такъв')
няма такъв
```

Речник (dict)

- Речник = `dict` = hashtable = associative array
- Реда не е гарантиран
- Асоциира ключ със стойност

tuple

```
>>> args = (9.8, 3.14, 2.71)
>>> args[2]
2.71
>>> args[1] = 22/7
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

tuple

- `tuple` = кортеж = n-торка
- Immutable
- Гарантиран ред
- Ползват за да подадете или върнете няколко стойности от функция, когато специален клас би бил твърде много
- Tuple от един елемент - със запетайка на края:
  `('This is the tale of captain Jack Sparrow.',)`
- Може и без скобите

Структури от данни: set

```
>>> unique_numbers = {2, 3, 5, 6}
>>> unique_numbers
{2, 3, 5, 6}
>>> unique_numbers.add(5)
>>> unique_numbers
{2, 3, 5, 6}
>>> unique_numbers.remove(5)
>>> unique_numbers
{2, 3, 6}
>>> my_list = [5, 1, 6, 6, 2, 3, 5, 5]
>>> set(my_list)
{1, 2, 3, 5, 6}
```

Структури от данни: set

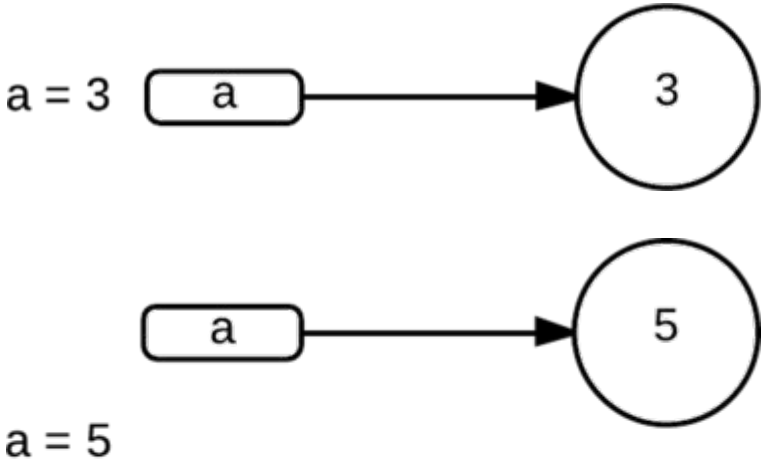
- `set` = множество = колекция без повтарящи се елементи
- Редът не е гарантиран
- Нямаме пряк достъп до конкретен елемент
- Можем да проверяваме за принадлежност
- Можем да обхождаме всичките(след малко ще видим как)

Mutable vs immutable

```
a = 3
a += 2
a # 5
```

- Immutable са стойностите, които не могат да бъдат променяни.
- Този код не променя стойността на `3`, а кара `a` да сочи към друга стойност - `5`.
- Immutable са числата, низовете, `tuple`-ите, `True`, `False`, `None` etc.

Имена в картинки (пак)



(Или в нашия случай - `a+=2`)

Mutable vs immutable

```
a = [1, 2, 3]
a.append(4)
a # [1, 2, 3, 4]
```

- Този код променя списъка, към който сочи `a`. Списъците са mutable.
- Всичко останало е mutable.
- Като ключ на `dict` или елемент на `set` могат да се ползват само immutable стойности.
- Защо?
- hashmap

Контролни структури

- `if .. elif .. else`
- `while`
- `for`

if

```
if a == 5:
    print("a is five")
elif a == 3 and not b == 2:
    print("a is three, but b is not two")
else:
    print("a is something else or b is two")
```

- Точно каквото очаквахте.
- Не слагайте скоби около условията.
- `and`, `or` и `not`
- **НЕ** `&&`, `||`, `!`

if (с булеви променливи)

```
a = True

if a:
    print("a is True")
if not a:
    print("a is not True")
```

истина и лъжа

В контекста на булевите операции като лъжа се интерпретират следните стойности:

- `False`
- `None`
- числото 0 независимо от типа числа (на пример `0`, `0.0`, `0j`)
- празният низ
- празни контейнери (`tuple`, `list`, `dict`, `set`, `frozenset`)
- наши типове могат да дефинират как да бъдат оценявани като булеви променливи

Всички останали стойности се интерпретират като истина.

if (тестове за принадлежност)

```
my_list = [1, 2, 3, 4]

if 1 in my_list:
    print('1 is in my list')

if 5 not in my_list:
    print('5 is not in my list')
```

Индентация

- Къде са къдравите скоби?!
- Всеки блок код (тяло на `if`, тяло на функция, и т.н.) се определя с индентацията му спрямо обгръщащия го блок.
- Всеки блок код започва само след двоеточие в края на предишния ред.
- Блокът свършва, когато се върнете към предишната индентация.
- 4 празни места = нов блок.
- **Не 2, не 3, не 8, не табулация**
- Дресирайте редактора си да слага 4 празни места когато натиснете `<Tab>`

while

```
a = 10
while a > 5:
    a -= 1
    print(f"a is {a}")
```

for

```
primes = [3, 5, 7, 11]
for e in primes:
    print(e ** 2) # 9 25 49 121

people = {'bob': 25, 'john': 22, 'mitt': 56}
for name, age in people.items():
    print("{} is {} years old".format(name, age))
    # bob is 25 years old
    # john is 22 years old
    # ...
```

- `for` е като `foreach` в другите езици
- Няма инициализация, стъпка и проверка, не е fancy `while`
- Обхожда структури от данни

for като в C

```
for i in range(0, 20):
    print(i)
    # 0 1 2 3 4 5 6 .. 19

for i in range(0, 20, 3):
    print(i)
    # 0 3 6 9 12 15 18
```

Може и наобратно

```
for i in range(20, 0, -1):
    print(i)
    # 20 19 18 17 16 15 .. 1

for i in range(20, 0, -3):
    print(i)
    # 20 17 14 11 8 5 2
```

break и continue

- Работят както очаквате във `for` и `while`.
- Афектират само най-вътрешния цикъл.

switch/case

- Няма…
- Добре де, нямаше…
- Вече има (`Python >= 3.10`).
- Все още не сме решили дали е добра идея.
- Засега можете да си поиграете с него.
- И в Python е `match/case`.

match/case

```
http_status = 400

match http_status:
    case 400:
        print("Bad request")
    case 401 | 403: # 401 OR 403
        print("Authentication error")
    case 404:
        print("Not found")

# Bad request
```

match/case

```
http_status = 9001

match http_status:
    case 400:
        print("Bad request")
    ...
    case _: # Default
        print("Other error")

# Other error
```

Има и още хиляда синтактични конструкции свързани с `match/case`, за момента толкоз.

Функции

```
def say_hello(name, from):
    return "Hello… It's me…"
```

- Функцията приема аргументи
- Функцията може да върне нещо с `return`, а ако няма `return` връща `None`
- Не се описват типовете на аргументите, нито типа на резултата

Аргументи на функции

```
def multiply(a, b=2):
    return a * b

multiply(5) # 10
multiply(5, 10) # 50

def is_pythagorean(a=2, b=3, c=4):
    return a * a + b * b == c * c

is_pythagorean(b=5) # a = 2, c = 4
is_pythagorean(1, c=3) # a = 1, b = 3
```

Променлив брой аргументи

```
def varfunc(some_arg, *args, **kwargs):
    #...

varfunc('hello', 1, 2, 3, name='Bob', age=12)
    # some_arg == 'hello'
    # args = (1, 2, 3)
    # kwargs = {'name': 'Bob', 'age': 12}
```

- Функциите могат да приемат произволен брой аргументи
- Позиционните аргументи (тези без име) отиват в `args`, което е `tuple` от аргументи
- Именуваните аргументи отиват в `kwargs`, което е `dict` от имена на аргументи и съответните им стойности
- Имената `args` и `kwargs` не са специални, **но са наложена конвенция**

First-class citizens

В python функциите са обекти!

```
def baba():
    print('баница')

def call(function, times):
    for _ in range(times):
        function()

call(baba, 5)
# баница
# баница
# баница
# баница
# баница
```

First-class citizens

Всяка функция може да приема като аргумент обекти от всякакъв тип, включително други функции, вградени типове, наши типове

Въпроси?