02. Въведение 2.0

10 октомври 2023

Какво видяхте в предишната лекция?

- Свалете си Python и изберете IDE
- Кодът седи в `.py` файлове
- Основни типове данни - `int, float, complex, str, bool, None` (проверете типа с `type(x)`)
- Променливите сочат към стойност, те не са стойност
- Колекции - `list, tuple, dict, set`
- Mutable vs Immutable

Нямате въпроси? (или имате, няма значение)

Е, ние имаме...

Какъв ще е резултатът от следните операции?

```
3 / 2                   # 1.5
3 // 2                  # 1
0.1 + 0.2               # 0.30000000000000004
0.1 + 0.1               # 0.2
0.5 - 0.2               # 0.3
0.3 / 3                 # 0.09999999999999999
```

И още един

Какъв е типът?

```
>>> things = ['eggs', ('spam', 'spam', 'spam'), 'ham']
>>> type(things[1][2][3])

<class 'str'>

>>> things[1][2][3]
'm'
```

А така?

Малко по-различна ситуация:

```
>>> things = ('eggs', ['spam', 'spam', 'spam'], 'ham')
>>> things[1][2][3] = 'h'
>>> type(things[1][2][3])

Traceback (most recent call last):
  File "<pyshell#69>", line 1, in <module>
    things[1][2][3] = 'h'
TypeError: 'str' object does not support item assignment
```

Какво ще видите днес?

- Контролни структури - `if`, `while`, `for`, `switch` (`break`, `continue`)
- Как се дефинират блокове код в Python
- Дефиниране на функции и аргументи на функции
- Малко по-задълбочено колекциите `list` и `tuple`
- Супер готина новина

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 (например `0, 0.0, 0j`)
- празният низ
- празни контейнери (`tuple, list, dict, set`)
- наши типове могат да дефинират как да бъдат оценявани като булеви променливи

Всички останали стойности се интерпретират като истина.

В този ред на мисли...

- Можем да “cast”-ваме стойности към различни типове.
- В кавички, защото не е баш кастване и определено не е като в статично типизираните езици.

```
int('5') # 5

b = 10
str(b) # '10'

nums = (1, 2, 3)
nums = list(nums)
nums # [1, 2, 3]

str(['baba', 2]) # "['baba', 2]"
```

Разбира се в някакви граници

```
int('диевиетстотин и пидисе')

Traceback (most recent call last):
  File "<pyshell#950>", line 1, in <module>
    int('диевиетстотин и пидисе')
ValueError: invalid literal for int() with base 10: 'диевиетстотин и
пидисе'
```

Тогава - въпрос

На базата на последните 3 слайда, до какво ще се оцени долното:

```
>>> question = "Питона искаш ли да ти покажа?"
>>> bool(question)

True
```

В почивката.

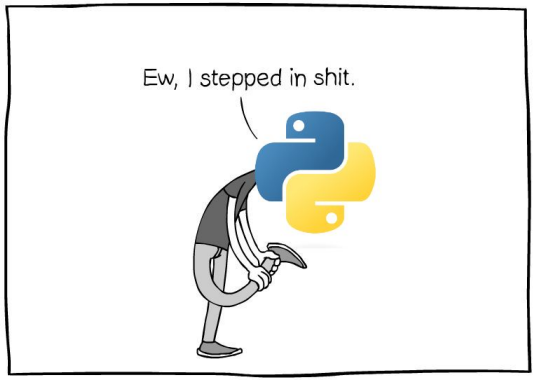
Също така, цитат от преди 3 слайда:
Всички останали стойности се интерпретират като истина.

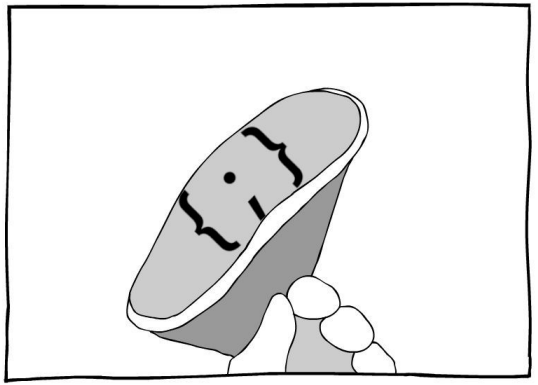
if (тестове за принадлежност)

```
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>





Индентация


C#
Java
C++
New Programmer
Missing ;
Python
IdentationError:


TabError: inconsistent use of tabs and spaces in indentation
Python Programmer

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, side):
    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 - позиционни

- Параметрите в предният пример са позиционни
- Ако функцията има 3 позиционни параметъра - трябва да я извикате с 3 аргумента
- Иначе - грешка
- Параметри vs аргументи?!

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

multiply(5, 10) # 50

multiply(5)
# TypeError: multiply() missing 1 required positional argument: 'b'
```

Аргументи на функции - именувани

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

- Именувани, по подразбиране, опционални
- Могат да бъдат изрично упоменати, ако не - взимат стойността по подразбиране
- Стига позиционните да са подадени в правилен ред, именуваните могат да бъдат в какъвто и да е ред

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**
- **Редът е важен!**

Аргументи на функции - позиционни

- Има и други шано неща като positional only, keyword only, optional positional, required keyword и прочие параметри
- Няма да навлизаме в тях
- Повярвайте ни, ще ви заболи главата, а е малко вероятно да ви трябват
- За когато се наложи - знайте, че ги има

```
def baba(a, b, /, c, d, *, e=2.72, f, **kwargs):
    do_stuff()
```

Списъци (List)

```
nice_things = ['coffee', 'cheese', 'crackers', 'tea']
for thing in nice_things:
    print(f'I tend to like {thing}')
```

Можем и просто да ги индексираме:

```
print(nice_things[1]) # cheese
print(nice_things[-1]) # tea
print(nice_things[-3]) # cheese
```

Списъци (List) - slicing

```
cute_animals = ['cat', 'raccoon', 'panda', 'red panda', 'marmot']

cute_animals[1:3]  # ['raccoon', 'panda']
cute_animals[-1]  # 'marmot'
cute_animals[1:-1]  # ['raccoon', 'panda', 'red panda']
cute_animals[::-1]  # ['marmot', 'red panda', 'panda', 'raccoon', 'cat']
cute_animals[-1:0:-1]  # ['marmot', 'red panda', 'panda', 'raccoon']
cute_animals[-1:0:-2]  # ['marmot', 'panda']
```

Списъци (List)

Списъците съдържат "референции" към елементи.

```
coffee, cheese, crackers, tea = 'coffee', 'cheese', 'crackers', 'tea'
# unpacking

things_i_like = [coffee, cheese, crackers]
things_you_like = [crackers, coffee, tea]

things_i_like[0] == things_you_like[1] # True
things_i_like[0] is things_you_like[1] # True
```

Списъци (List)

Това позволява някои интересни неща:

```
cheeses = ['brie', 'bergkäse', 'kashkaval', 'leipäjuusto']
cheeses.append(cheeses)

cheeses[-1] is cheeses # True
print(cheeses) # ['brie', 'bergkäse', 'kashkaval', 'leipäjuusto', [...]]
```

Списъци(-ception)

```
cheeses = ['brie', 'bergkäse', 'kashkaval', 'leipäjuusto']
teas = ['chai', 'earl grey', 'jasmine', 'oolong']

breakfast = [cheeses, teas]
print(breakfast[0][1]) # bergkäse

breakfast[1][2] = ['шкембе', 'люти чушки', 'оцет с чесън']
print(teas) # ?

['chai', 'earl grey', ['шкембе', 'люти чушки', 'оцет с чесън'],
'oolong']
```

Методи на списъци

- `.index(element)` - Индекса на първото срещане на element в списъка или гърми с ValueError
- `.count(element)` - Броят срещания на element в списъка
- `.append(element)` - Добавя element в края на списъка
- `.extend(elements)` - Добавя елементите на elements в списъка (като + ама по-бързо)
- `.sort()` - Сещате се
- ...

Range

range връща итерируемо за интервал от числа

```
numbers = range(3)
for number in numbers:
    print('We can count to ' + str(number))
```

Range

```
numbers = range(10, 13)
for number in numbers:
    print('We can count to ' + str(number))
```

Range

range може и в обратен ред

```
numbers = range(13, 0, -1)
for number in numbers:
    print('We can count to ' + str(number))
```

Tuple

Миналия път казахме, че може и без скобите:

```
people = 'Niki', 'Kiro', 'Genata'
people = 'Niki',

people = ('Niki') # най-вероятно не е каквото очаквате
```

- Имат методите `index` и `count` като на списъците
- Скучно

Любопитни неща

Ако имате n-торка, съдържаща само имена от лявата страна на присвояване, може да постигнете интересни ефекти:

```
(a, b) = 1, 2

print(a)  # 1
```

Още по-любопитни неща

Всъщност скобите изобщо не са задължителни:

```
a, b = 1, 2

print(a)  # 1
```

Още по-по-любопитни неща

```
numbers = (1, 2, 3)

a, b, c = numbers
```

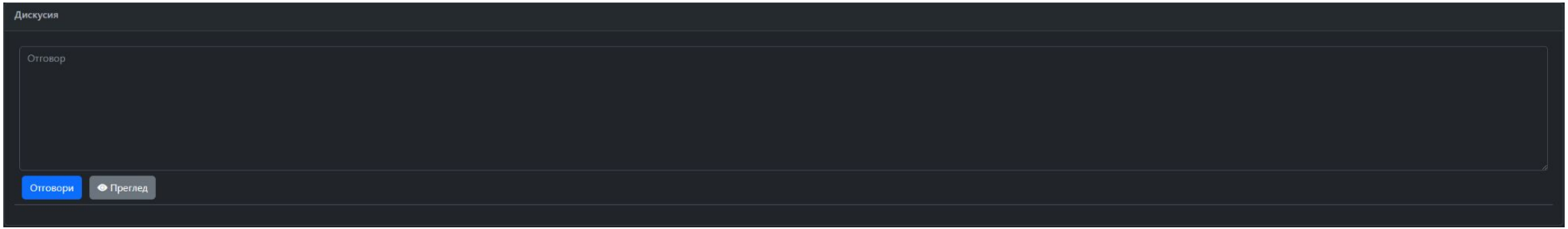
Най-любопитни неща

```
a, *b, c = 1, 2, 3, 4, 5

a = 1
b = [2, 3, 4]
c = 5
```

И сега за новината, която нямате търпение да чуете!

Имате първо домашно...

- Сега!
- Срокът е до 18:00 на 17.10 (следващият вторник)
- Внимателно се запознайте с това как да (не) си изпращате задачите
- Отдолу има поле за дискусия, ако имате въпроси - ползвайте го



- Знаем, че няма да ни послушате, но...
- Пробвайте да не го оставяте за последния момент, за да можем да ви върнем обратна връзка

Въпроси?