03. Въведение 2.1

04 март 2026

Какво видяхте в предишната лекция?

- Контролни структури - `if`, `while`, `for`, `switch` (`break`, `continue`)
- 4 интервала индентация и “:” за начало на блока
- Дефиниране на функции и аргументи на функции

Въпроси?

За какво не ни стигна времето?

- Да довършим последния слайд
- Супер готината новина, която ви обещахме
- Голите снимки, на които се надявахте
- Функции, които генерират колекции
- Функции, които използват колекции
- Функции, които едновременно използват и генерират колекции
- Бикове и крави

И в допълнение

- Добавили сме малко слайдове по някои от темите, които демонстрирахме предния път, но ги нямаше в презентациите
- Изрази, които генерират колекции

Но първо!!!

- Въпроси!
- *Някои от тях - courtesy of G.K.*


Коя колекция може да бъде итерирана с for?
A: Dict
B: Set
C: List
D: Tuple

Да итерираме тогава

```
for prime in [1, 2, 3, 5, 7]:
    if prime == 2 or prime % 2 != 0:
        print(prime * 2)
else:
    print("Oh no, what happens now?")
```

```
2
4
6
10
14
"Oh no, what happens now?"
```

Добре, без else

```
for prime in [1, 2, 3, 6, -7]:
    if prime < 0:
        print("Primes cannot be negative!")
        break
    elif prime == 2 or prime % 2 != 0:
        print(prime * 2)
        continue
    break
```

```
2
4
6
```

Побългарен код - какво ще изведе?

```
value = '_|_'
кажи = print
Вальо = value
тръбата = '|'
дължината = len

if тръбата in Вальо:                    # Ако тръбата е във Вальо
    if дължината(тръбата) > 20:         # Ако тръбата е по-дълга от 20
        _ = Вальо, кажи('Ooox!')        # Кажи, Вальо, Ooox!
    else:                               # Инак
        _ = Вальо, кажи('Eeex!')        # Кажи, Вальо, Eeex!
    _ = Вальо, кажи('Вух!')             # Кажи, Вальо, Вух!

>>> Eeex!
>>> Вух!
```

За десерт - една скучна функция

```
def boring_function(boring_argument, *args, **kwargs):
    return len(args) + len(kwargs)


print(boring_function() + boring_function(0) + boring_function(0, 0))


# TypeError: boring_function() missing 1 required positional
argument: 'boring_argument'
```

Последния слайд от предишната лекция

```
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 *(сега, може и да го направим в някакъв момент)*
- 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()
```

Кой какво сиренье има (всичко дотук - заедно)

```
data = [('John', 'Tilsit'), ('Eric', 'Cheshire'), ('Michael', 'Camembert'),
        ('Terry', 'Gouda'), ('Terry', 'Port Salut'), ('Michael', 'Edam'),
        ('Eric', 'Ilchester'), ('John', 'Fynbo')]

def cheeses_by_owner(cheeses_data):
    by_owner = {}
    for owner, cheese in cheeses_data: # <- tuple unpacking
        if owner in by_owner:
            by_owner[owner].append(cheese)
        else:
            by_owner[owner] = [cheese]
    return by_owner
```

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))
```

enumerate

```
necessities = ['Бира', 'Риба', 'Николай или Никола, или някое производно']
for index, necessity in enumerate(necessities, 1):
    print(f'{index}. {necessity}')

# 1. Бира
# 2. Риба
# 3. Николай или Никола, или някое производно
```

Map/Filter/Reduce/All/Any/Lambda

- `map(function, iterable)` създава колекция от резултатите от прилагането на function върху всеки елемент от iterable
- `filter(function, iterable)` създава колекция само с елементите, за които function върне `True`
- `reduce(function, iterable)` вика `function` с елементите на колекцията, докато сведе всичко до една стойност (във `functools` вж. тук)
- `all(iterable)` всички елементи се оценяват на истина
- `any(iterable)` поне един от елементите се оценява на истина
- `lambda` функции - анонимни функции, които често вървят ръка за ръка с горните

За любознателните: `map()` и `filter()` са мързеливи

Lambda функции

- Понякога не ви се занимава с това да пишете функции, а просто имате нужда от функция за един път
- Не искате да знаете как се казва, не ви интересува как ѝ е минала седмицата, просто искате да се позабавлявате веднъж и повече да не се занимавате с нея
- Не се притеснявайте, ние не съдим
- But remember kids, it must be a win-win
- Та, lambda функции

Lambda функции

- Например, ако искате да използвате `map`, можете да го направите по следния начин:

```
def squared(num):
    return num ** 2

squared_map = map(squared, range(0, 20, 3))
print(list(squared_map)) # [0, 9, 36, 81, 144, 225, 324]
```

- Можете обаче и по-лесно:

```
squared_map = map(lambda num: num ** 2, range(0, 20, 3))
```

Lambda функции

```
lambda pa1, pa2, ..., *args, kw1, kw2, ..., **kwargs: <some expression>
```

- Горното създава анонимна функция
- Може да се дефинира с параметри както нормална функция, но задължително се свежда до един израз и той е върнатата стойност
- Както всичко в Python - тя е обект *(преподавателите на този етап се дистанцират от всякакви предишни алегории)*
- Както всички обекти - когато вече няма референции към нея - маркира се за триене и в някакъв момент Python освобождава паметта
- В примера от предния слайд - когато `map`-а приключи - тя изчезва
- Но ако много ви хареса, винаги може да ѝ дадете име:

```
best_lambda_in_the_world = lambda x: x
```

Къде са голите снимки?

Имаме още

Тери Джоунс (да, от Монти Пайтън) в ролята на “Голият органист”

Comprehensions

- Изрази, които *генерират* колекции
- Елегантен заместител на `map()` и/или `filter()`
- Колекциите могат да са динамични

List comprehension

```
[израз for променлива in поредица if условие]
```

```
>>> [x * x for x in range(0, 10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

>>> [x * x for x in range(0, 10) if x % 2]
[1, 9, 25, 49, 81]
```

List comprehension

Един list comprehension може да се вложи в друг, защото връща нещо итерируемо

```
>>> [(x, y) for x in [1, 2, 3] for y in [3, 1, 4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

Generator expression

- Кръгли скоби вместо квадратни
- Като list comprehension, но се изпълнява динамично (lazy evaluation)
- На всяка стъпка итератора оценява условието и израза за следващата стойност

Lazy vs eager

- “Мързеливите” колекции не заемат *(толкова)* памет
- Но в същото време не се оценяват / изчисляват веднага
- Което пък създава трудности да достъпваме произволен елемент от колекцията
- И в общия случай мързеливите колекции могат да се обхождат само веднъж

Set comprehension

Като list comprehension, но с {}

```
>>> import math
>>> {int(math.sqrt(x)) for x in range(1, 100)}
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Dict comprehension

```
>>> {i: chr(65 + i) for i in range(10)}

{0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H', 8: 'I',
9: 'J'}
```

Колекции за хора без компания в петък вечер

- `defaultdict` - речник със стойност по подразбиране
- `deque` - двупосочни опашки
- `OrderedDict` - речник, който помни реда и някои други неща
- `Counter` - речник, който брои повтарящи се стойности
- `namedtuple` - кортеж с именувани полета

defaultdict

```
from collections import defaultdict

data = [('John', 'Tilsit'), ('Eric', 'Cheshire'), ('Michael', 'Camembert'),
        ('Terry', 'Gouda'), ('Terry', 'Port Salut'), ('Michael', 'Edam'),
        ('Eric', 'Ilchester'), ('John', 'Fynbo')]

def cheeses_by_owner(cheeses_data):
    by_owner = defaultdict(list)
    for owner, cheese in cheeses_data:
        by_owner[owner].append(cheese)
    return by_owner
```

deque

```
from collections import deque

people = deque()
people.append('John') # Отдясно
people.appendleft('Terry') # Отляво
people.append('Graham')
people.append('N/A')
people.appendleft('Eric')
people.appendleft('Not important')

print(people.popleft()) # 'Not important'
print(people.pop()) # 'N/A'
print(', '.join(people) + ', Michael, Terry') # John, Terry, Graham,
Eric, Michael, Terry
```

Упражнение - Бикове и Крави (1)

```
import random # Засега - магия. След няколко лекции - ежедневие.

LENGTH = 4 # Константа за дължина на числата

def get_secret():
    """Generate random 4-digit number with unique digits (no zero)."""
    all_digits = list(map(str, range(1, 10)))
    random.shuffle(all_digits)
    return ''.join(all_digits[:LENGTH])
```

Упражнение - Бикове и Крави (2)

```
def compare(num1, num2):
    """Return bulls/cows count between two numbers. See the rules."""
    bulls, cows = 0, 0
    for dig1, dig2 in zip(num1, num2):
        if dig1 == dig2:
            bulls += 1
        elif dig1 in num2:
            cows += 1
    return bulls, cows
```

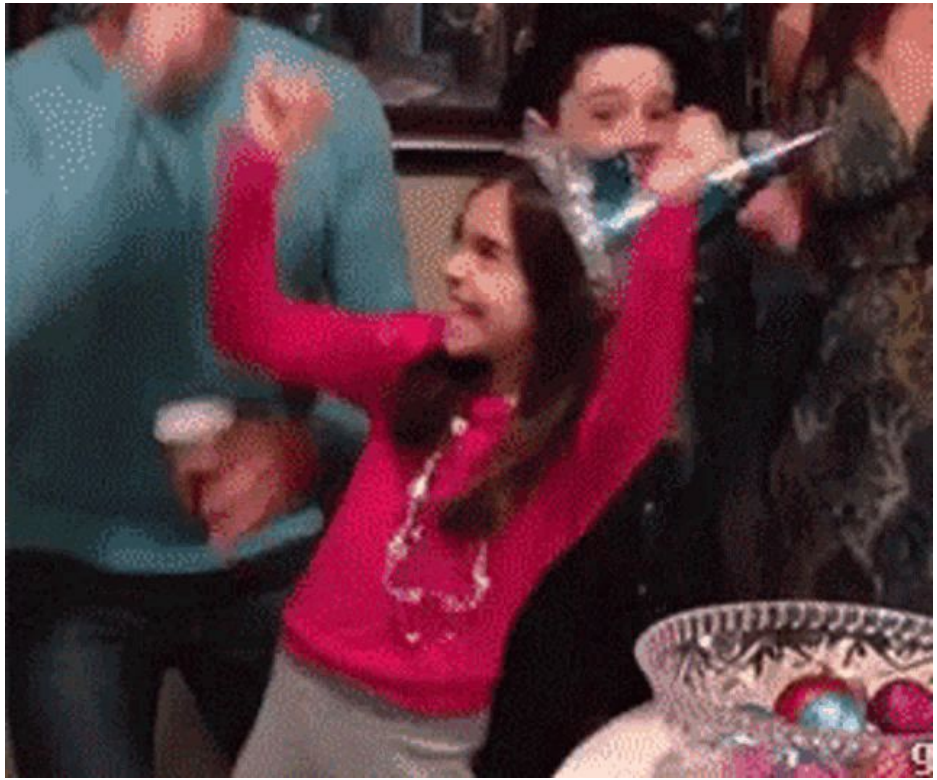
Упражнение - Бикове и Крави (3)

```
SECRET = get_secret()
print('Намислих си четирицифрено число без повторение на цифрите и без
нули. Опитай да познаеш...')
while True:
    guess = input('Въведи предположение: ')
    bulls, cows = compare(SECRET, guess)
    if bulls == LENGTH:
        print(f'Позна, машино!')
        break
    else:
        print(f'Имаш {bulls} бика и {cows} крави.')
```

Упражнение - Бикове и Крави (4)

```
while True:
    guess = input('Въведи предположение: ')
    # Вмъкваме малко валидация в while цикъла
    if not guess.isdigit():
        print('Това май не е число.')
        continue
    if len(guess) != LENGTH:
        print('Числото трябва да е четирицифрено.')
        continue
    if len(set(guess)) != LENGTH:
        print('Числото трябва да съдържа само уникални цифри.')
        continue
    if '0' in guess:
        print('Числото не може да съдържа нула.')
        continue
    bulls, cows = compare(SECRET, guess)
    …
```

Първо домашно!



Първото домашно

- До 18:00 на 11.03 (следващата сряда)
- Вече започва да става интересно
- Знаете, ако имате въпроси - в коментарите

По темата с домашните

Въпроси?