Основне академске студије
Информациони инжењеринг

Методе и технике науке о подацима

Основни концепти језика и окружења Python

(материјали за предавања)

Садржај

Увод

Настанак језика *Python*

- Гвидо ван Росум (*Guido van Rossum*)
  - аутор језика *Python*
    - 80-е године XX века
    - инспирација потиче од језика *ABC*
  - „добронамерни доживотни диктатор”

Увод

Развој језика *Python*

верзије

- Python 3.13.5 (јун 2025)
- Python 3.12.2 (фебруар 2024)
- Python 3.11.2 (фебруар 2023)
- Python 3.10.2 (јануар 2022)
- Python 3.9.2 (фебруар 2021)
- Python 3.8.1 (децембар 2019)
- Python 3.7.6 (децембар 2019)
- Python 3.8.0 (октобар 2019)
- Python 2.7.0 (јул 2010)
- Python 3.0.0 (децембар 2008)
- Python 2.6.0 (октобар 2008)
- Python 2.0.1 (јун 2001)
- Python 1.6.1 (септембар 2000)
- Python 1.5.2 (април 1999)
- Python 0.9.0–1.2 (1991–1995)

Увод

Развој језика *Python*

Python Software Foundation

https://www.python.org/psf/

непрофитна организација која промовише, штити и развија програмски језик *Python*

повлачење Гвида ван Росума

https://mail.python.org/pipermail/python-committers/2018-July/005664.html

12. јул 2018.

Python steering council

https://peps.python.org/pep-8100/

изабрани чланови (јануар–фебруар 2019)

Barry Warsaw
Brett Cannon
Carol Willing
Guido van Rossum
Nick Coghlan

Увод

Основне одлике

- интерпретиран
- строго типизован
- динамички типизован
  - *duck typing*
- разликовање малих и великих слова
- увлачење програмског кода помоћу одређених белих знакова

Пример почетног одзива у окружењу IDLE

- Python's Integrated Development and Learning Environment (IDLE)

```
Python 3.9.2 (default, Feb 20 2021, 00:00:00)
[GCC 10.2.1 20201125 (Red Hat 10.2.1-9)] on linux
Type "help", "copyright", "credits" or "license()" for more
information.
>>>
```

Пример

```
>>> import this
The Zen of Python, by Tim Peters

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!
>>>
```

КОНЗОЛА

Садржај

Објекти

Објекат

- одговара одређеним подацима у меморији
- поседује тип
  - тип као опис врсте података којима објекат одговара
  - типом одређено и које су операције могуће над одговарајућим подацима
- може поседовати назив (ознаку)
  - назив повезује с објектом кроз операцију доделе
  - придруживање више назива једном истом објекту не доводи до умножавања објекта
- променљивост објекта
  - променљиви објекти
  - непроменљиви објекти

Објекти

Основни типови

- бројчани тип
  - цели
  - реални
  - рационални
  - комплексни
- знаковни тип (стринг)
- логички тип (Булов)
- бинарни тип

Објекти

Напредни типови

- секвенца
  - листа
  - торка
  - распон
- скуп
- речник
- функција
- тип
- нулти тип (NULL)
- класа
- метода
- ...

Објекти

Називи (ознаке) објеката

- помоћу назива могуће користити објекат
- дозвољени знакови у називу
  - слова, цифре, доња црта
- забране у формирању назива
  - цифре нису дозвољене на почетку назива
  - резервисане речи не смеју бити називи
    - нпр. **`class`**, **`for`**, **`if`**, **`import`**, **`return`**

Објекти

Бројчани тип

- уграђени типови
  - цели бројеви (**`int`**)
  - реални бројеви (**`float`**)
  - рационални бројеви (**`Fraction`**)
  - комплексни бројеви (**`complex`**)
- непроменљивост бројева

Објекти

Бројчани тип – оператори

- промена знака **-**
- задржавање знака **+**
- сабирање **+**
- одузимање **-**
- множење *****
- дељење **/**
- целобројно дељење **//**
- остатак при дељењу **%**
- степеновање ******

Објекти

Бројчани тип – цели бројеви

```
>>> 1
1
>>> 1 + 2
3
>>> print(1)
1
>>> print(1 + 2)
3
>>> -3
-3
>>>
```

КОНЗОЛА

Бројчани тип – цели бројеви

```
>>> 5 - -3
8
>>> 5--3
8
>>> 5 - - 3
8
>>> --5--3
8
>>> ++5
5
>>> 5++
SyntaxError: invalid syntax
>>>
```

КОНЗОЛА

Бројчани тип – цели бројеви

```
>>> 6 // 3
2
>>> -5 // -3
1
>>> 5 % 3
2
>>> -5 % 3
1
>>> 5 % -3
-1
>>> -5 % -3
-2
>>> 2 ** 2
4
>>>
```

КОНЗОЛА

Објекти

Бројчани тип – цели бројеви

- произвољно велики
- подржани бројни системи
  - децимални
    - подразумевано користи
  - бинарни
    - означава помоћу префикса **0b** или **0B**
  - октални
    - означава помоћу префикса **0o** или **0O**
  - хексадецимални
    - означава помоћу префикса **0x** или **0X**

Бројчани тип – цели бројеви

подржани бројни системи

```
>>> 0b01 + 0b01
2
>>> 0o1 + 0o7
8
>>> 0xa + 0xb
21
>>> 0b10 + 0o10 + 0xA
20
>>>
```

КОНЗОЛА

Објекти

Бројчани тип – реални бројеви

- прецизност
  - до 15 децималних места
- начини представљања
  - помоћу децималне тачке
  - експоненцијални запис
    - мантиса
    - основа 10
    - експонент

Бројчани тип – реални бројеви

```
>>> 0.1
0.1
>>> .1
0.1
>>> 1e-1
0.1
>>> 1e10
10000000000.0
>>> 1e100
1e+100
>>> -1.23e123
-1.23e+123
>>> 0x1e2
482
>>>
```

КОНЗОЛА

Бројчани тип – реални бројеви

```
>>> -5 / -3
1.6666666666666667
>>> -5 / -5
1.0
>>> -5 // -3.0
1.0
>>> 2.0 ** 2
4.0
>>> 2.0 ** 2.5
5.65685424949238 1
>>>
```

КОНЗОЛА

Објекти

Бројчани тип – рационални бројеви

- ослањање на уграђени модул **`fractions`**
  - класа **`Fraction`**
    - инстанце формирају на основу задатих бројиоца и имениоца
    - операције над инстанцама могу довести до промене у имениоцу

Бројчани тип – рационални бројеви

```
>>> import fractions
>>> fractions.Fraction(5)
Fraction(5, 1)
>>> fractions.Fraction(5, 6)
Fraction(5, 6)
>>> fractions.Fraction("5/6")
Fraction(5, 6)
>>> fractions.Fraction(0.833333)
Fraction(7505996376551075, 9007199254740992)
>>>
```

КОНЗОЛА

Објекти

Бројчани тип – рационални бројеви

```
>>> import fractions
>>> x = fractions.Fraction(5, 6)
>>> x / 2
Fraction(5, 12)
>>> x ** 2
Fraction(25, 36)
>>> fractions.Fraction(5, 6) -
fractions.Fraction(3, 6)
Fraction(1, 3)
>>>
```

КОНЗОЛА

Бројчани тип – комплексни бројеви

- помоћу уграђене класе **`complex`**
  - инстанце формирају на основу задате реалне и имагинарне компоненте

Бројчани тип – комплексни бројеви

```
>>> complex(1)
(1+0j)
>>> complex(1, 1)
(1+1j)
>>> x = complex(1, 1)
>>> x
(1+1j)
>>> x ** 2
2j
>>> 1 + 1j
(1+1j)
>>>
```

КОНЗОЛА

Објекти

Знаковни тип (стринг)

- стринг – низ знакова
  - ознака почетка и краја стринга
    - знак апостроф (**'**) или знак наводник (**"**)
      - појединачно или троструко означавање
        - троструко означавање за стрингове у више редова
- непроменљивост стрингова

Знаковни тип

```
>>> 'tekst'
'tekst'
>>> "tekst"
'tekst'
>>> '"tekst"'
'"tekst"'
>>> "'tekst'"
"'tekst'"
>>> print('tekst')
tekst
>>> print("duži", "tekst")
duži tekst
>>>
```

КОНЗОЛА

Знаковни тип – посебни знакови

- обрнута коса црта (`\`)
  - посебна обрада пратећег карактера
  - примери
    - приказ видљивог знака који иначе има посебну улогу
      - \'
      - \”
      - \\
    - приказ контролних знакова
      - \n
      - \t
  - необрађени стринг (енгл. *raw*)
    - стринг означен помоћу префикса **`r`**
    - посебан знак обрнута коса црта не третира као посебан

Знаковни тип – посебни знакови

```
>>> '\'Sad!\''
"'Sad!'"
>>> print('\'Sad!\'')
'Sad!'
>>> r'\'Sad!\''
"\\'Sad!\\'"
>>>
```

КОНЗОЛА

Објекти

Знаковни тип – посебни знакови

```
>>> '\turban'
'\turban'
>>> r'\turban'
'\\turban'
>>> print('\turban')
   urban
>>> print(r'\turban')
\turban
>>>
```

КОНЗОЛА

Објекти

Знаковни тип

```
>>> x = '''Readability counts.
- The Zen of Python'''
>>> x
'Readability counts.\n- The Zen of Python'
>>> print(x)
Readability counts.
- The Zen of Python
>>> y = 'Readability counts.\n- The Zen of Python'
>>> y
'Readability counts.\n- The Zen of Python'
>>> print(y)
Readability counts.
- The Zen of Python
>>>
```

КОНЗОЛА

Објекти

Знаковни тип – оператори

- спајање **+**
- надодавање (додавање на крај) **+=**
- умножавање *****
- индексирање **[ ]**

Знаковни тип – оператори

```
>>> "pod" "atak"
'podatak'
>>> print("pod" "atak")
podatak
>>> print("pod", "atak")
pod atak
>>>
```

КОНЗОЛА

Знаковни тип – оператори

оператор **+**

```
>>> p = "pod"
>>> a = "atak"
>>> p + a
'podatak'
>>> p a
SyntaxError: invalid syntax
>>> print(p, a)
pod atak
>>>
```

КОНЗОЛА

Објекти

Знаковни тип – оператори

- оператор *
- оператор **+=**

```
>>> s = "s"
>>> s * 3 + "trava"
'ssstrava'
>>> s += "trava"
>>> s
'strava'
>>>
```

КОНЗОЛА

Знаковни тип – оператори

оператор `[]`

```
>>> s = "oprst"
>>> s[0]
'o'
>>> s[1]
'p'
>>> s[4]
't'
>>> s[5]
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    s[5]
IndexError: string index out of range
>>>
```

КОНЗОЛА

Знаковни тип – оператори

оператор `[]`

```
>>> s = "oprst"
>>> s[-1]
't'
>>> s[-4]
'p'
>>> s[-5]
'o'
>>> s[-6]
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    s[-6]
IndexError: string index out of range
>>>
```

КОНЗОЛА

Знаковни тип – оператори

- оператор `[]` – сечење стринга

```
>>> tekst = "абвгдђежзи"
>>> tekst[0]
'а'
>>> tekst[0:5]
'абвгд'
>>> tekst[0:5:3]
'аг'
>>> tekst[0:10:3]
'агеи'
>>>
```

КОНЗОЛА

Знаковни тип – оператори

оператор `[]` – сечење стринга

```
>>> tekst = "абвгдђежзи"
>>> tekst[:10:3]
'аеи'
>>> tekst[::3]
'аеи'
>>> tekst[2::2]
'вдез'
>>> tekst[:2:]
'аб'
>>> tekst[::]
SyntaxError: invalid syntax
>>> tekst[:]
'абвгдђежзи'
>>>
```

КОНЗОЛА

Објекти

Знаковни тип – оператори

оператор `[]` – сечење стринга

```
>>> tekst = "абвгдђежзи"
>>> tekst[5:0:-1]
'ђдгвб'
>>> tekst[5::-1]
'ђдгвба'
>>> tekst[-5:-10:-1]
'ђдгвб'
>>> tekst[-5:-11:-1]
'ђдгвба'
>>> tekst[::-1]
'изжеђдгвба'
>>>
```

КОНЗОЛА

Објекти

Знаковни тип – оператори

оператор [] – сечење стринга

```
>>> tekst = "абвгдђежзи"
>>> tekst[-1::-1]
'изжеђдгвба'
>>> tekst[0::-1]
'а'
>>> tekst[1::-1]
'ба'
>>> tekst[-11::1]
'абвгдђежзи'
>>> tekst[2:-5:-1]
''
>>> tekst[-2:-5:1]
''
>>>
```

КОНЗОЛА

Објекти

Знаковни тип – оператори

- релациони оператори `<, <=, >, >=, !=, ==`

```
>>> "pravo" == "pravda"
False
>>> "pravo" != "pravda"
True
>>> "pravo" > "pravda"
True
>>> "право" > "правда"
True
>>>
```

КОНЗОЛА

Објекти

Знаковни тип – оператори

- оператори **`in, not in`**

```
>>> "n" in "nešto"
True
>>> "N" in "nešto"
False
>>> "to" in "nešto"
True
>>> "" not in "nešto"
False
>>>
```

КОНЗОЛА

Знаковни тип – помоћне функције

- функција **`len()`**
  - очитавање дужине стринга
- метода **`lower()`**
  - промена великих слова у мала
- метода **`upper()`**
  - промена малих слова у велика
- метода **`swapcase()`**
  - промена великих слова у мала и малих слова у велика
- метода **`capitalize()`**
  - промена почетног слова у велико а преосталих слова у мала
- метода **`title()`**
  - промена почетног слова сваке речи у велико
- метода **`replace()`**
  - промена једног подстринга другим

Знаковни тип – помоћне функције

```
>>> len("u buntu")
7
>>> capitalize("u buntu")
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    capitalize("u buntu")
NameError: name 'capitalize' is not defined
>>> "u buntu".capitalize()
'U buntu'
>>> "u buntu".replace(" ", "")
'ubuntu'
>>> "u buntu".title().replace(" ", "")
'UBuntu'
>>>
```

КОНЗОЛА

Знаковни тип – помоћне функције

- метода **`join()`**
  - спајање више стрингова у један
- метода **`split()`**
  - подела стринга на више стрингова
- метода **`splitlines()`**
  - подела стринга на више стрингова сагласно распореду знакова по редовима
- ...

Знаковни тип – помоћне функције

```
>>> "–".join("Dunav", "Tisa", "Dunav")
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    "–".join("Dunav", "Tisa", "Dunav")
TypeError: join() takes exactly one argument (3
given)
>>> "–".join(["Dunav", "Tisa", "Dunav"])
'Dunav–Tisa–Dunav'
>>> "Dunav–Tisa–Dunav".split("–")
['Dunav', 'Tisa', 'Dunav']
>>>
```

КОНЗОЛА

Логички тип (Булов)

- логичке константе
  - **`True`**
    - може бити третирана и као бројчана вредност **`1`**
  - **`False`**
    - може бити третирана и као бројчана вредност **`0`**

Логички тип – оператори

- дисјункција **`or`**
  - други операнд евалуира само по потреби
- конјункција **`and`**
  - други операнд евалуира само по потреби
- негација **`not`**

Логички тип

```
>>> True
True
>>> True and False or True
True
>>> True and not False
True
>>> True | False
True
>>> True & not False
SyntaxError: invalid syntax
>>> True & (not False)
True
>>> True == False
False
>>>
```

КОНЗОЛА

Логички тип

```
>>> True > False
True
>>> True - True
0
>>> True ** False
1
>>> True ** 2
1
>>> -True
-1
>>> False & True + True
0
>>>
```

КОНЗОЛА

Логички тип

- одређене појаве неких типова могу бити тумачене као **`False`**
  - може бити корисно при контроли тока
    - бројчана вредност **`0`**
    - празан стринг **`''`** или **`""`**
    - празне структуре података
      - празна торка **`()`**
      - празна листа **`[]`**
      - празан речник **`{}`**
      - празан скуп **`set()`**

Бинарни тип

- подршка за рад над бинарним садржајем
  - појединачни бајт дозвољава целе бројеве у распону од 0 до 255
- типови
  - **`bytes`**
    - непроменљива секвенца бајтова
  - **`bytearray`**
    - променљива секвенца бајтова

Бинарни тип

- представа вредности типа **`bytes`**
  - посебан стринг означен помоћу префикса **`b`**
    - задавање вредности бајта
      - као хексадецимална вредност
        - у формату **`\x__`**
      - као *ASCII* знак

Бинарни тип – тип **bytes**

```
>>> b"!"
b'!'
>>> b"\x43\x69\x61\x6f\x21"
b'Ciao!'
>>> bytes([0x43, 0x69, 0x61, 0x6f, 0x21])
b'Ciao!'
>>> x = bytes([0x43, 0x69, 0x61, 0x6f, 0x21])
>>> x[4]
33
>>> x[0:4]
b'Ciao'
>>>
```

КОНЗОЛА

Бинарни тип – тип **bytearray**

```
>>> x = bytearray([0x43, 0x69, 0x61, 0x6f, 0x21])
>>> x
bytearray(b'Ciao!')
>>> x[2] = 99
>>> x
bytearray(b'Cico!')
>>> x + x
bytearray(b'Cico!Cico!')
>>> x * 2
bytearray(b'Cico!Cico!')
>>>
```

КОНЗОЛА

Бинарни тип – помоћне функције

- конверзија података
  - функције из модула **`struct`**
    - функција **`pack()`**
      - конверзија у бинарни облик
    - функција **`unpack()`**
      - конверзија из бинарног облика
  - правила конверзије
    - опис редоследа бајтова
      - *little endian* (**`<`**)
      - *big endian* (**`>`**)
    - опис формата података
      - бајт за прескакање (**`x`**)
      - означени цео број (**`i`**)
      - неозначени цео број (**`I`**)
      - реалан број у систему покретног зареза (**`f`**)
      - знак (**`s`**)
      - …

Бинарни тип – помоћне функције

```
>>> import struct
>>> struct.pack(">I", 1024)
b'\x00\x00\x04\x00'
>>> struct.pack("<I", 1024)
b'\x00\x04\x00\x00'
>>> struct.pack(">i", 1024)
b'\x00\x00\x04\x00'
>>> struct.pack("<i", 1024)
b'\x00\x04\x00\x00'
>>> struct.pack("<i", -1025)
b'\xff\xfb\xff\xff'
>>>
```

КОНЗОЛА

Бинарни тип – помоћне функције

```
>>> import struct
>>> struct.unpack("<i", b"\xff\xfb\xff\xff")
(-1025,)
>>> struct.unpack("<I", b"\xff\xfb\xff\xff")
(4294966271,)
>>> struct.unpack(">i", b"\xff\xfb\xff\xff")
(-262145,)
>>> struct.unpack(">I", b"\xff\xfb\xff\xff")
(4294705151,)
>>>
```

КОНЗОЛА

Нулти тип (NULL)

- нулта константа
  - **`None`**
    - означава непостојећу вредност
      - није исто што и **`0`**
      - није исто што и **`False`**
    - може користити у логичком контексту
    - логичка негација вредности **`None`** је логичка вредност **`True`**

Нулти тип

```
>>> None
>>> None and None
>>> None or None
>>> None and False
>>> False and None
False
>>> True or None
True
>>> None or True
True
>>> not None
True
>>>
```

КОНЗОЛА

Објекти

Нулти тип – оператори

- оператор **`is`**
- оператор **`is not`**

Нулти тип – оператори

```
>>> x = None
>>> x
>>> x is None
True
>>> x is not None
False
>>> False is None
False
>>> None is False
False
>>> not None is True
True
>>>
```

КОНЗОЛА

Провера типа

- помоћу функције **`type()`**
  - аргумент
    - објекат чији тип треба утврдити
- помоћу функције **`isinstance()`**
  - аргументи
    - објекат за који треба урадити проверу
    - тип за који треба урадити проверу

Провера типа

```
>>> type(49)
<class 'int'>
>>> type(49e0)
<class 'float'>
>>> import fractions
>>> type(fractions.Fraction("22/7"))
<class 'fractions.Fraction'>
>>> type(complex(0, 1))
<class 'complex'>
>>> type("priča")
<class 'str'>
>>>
```

КОНЗОЛА

Провера типа

```
>>> type(True)
<class 'bool'>
>>> type(b"\x30")
<class 'bytes'>
>>> type(bytearray(b"\x48"))
<class 'bytearray'>
>>> type(None)
<class 'NoneType'>
>>>
```

КОНЗОЛА

Провера типа

```
>>> isinstance(49, int)
True
>>> isinstance(49, float)
False
>>> s = "string"
>>> isinstance(s, str)
True
>>> isinstance(None, bool)
False
>>> isinstance(not None, bool)
True
>>> isinstance(b"story", bytes)
True
>>>
```

КОНЗОЛА

Конверзија типа

- помоћу функције **`int()`**
  - генерише цео број
    - по потреби уклања део иза децималне запете
- помоћу функције **`float()`**
  - генерише реалан број
- помоћу функције **`str()`**
  - генерише стринг
- помоћу функције **`bool()`**
  - генерише логичку вредност
- помоћу функције **`bytearray()`**
  - генерише промењљиву секвенца бајтова

Конверзија типа

```
>>> int(4.3)
4
>>> int(-4.3)
-4
>>> int("4.3")
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    int("4.3")
ValueError: invalid literal for int() with base
10: '4.3'
>>> float(4.3)
4.3
>>> str(4.3)
'4.3'
>>>
```

КОНЗОЛА

Конверзија типа

```
>>> bool(0)
False
>>> bool(1)
True
>>> bool(2)
True
>>> bool("a")
True
>>> bool("")
False
>>> bool("0")
True
>>>
```

КОНЗОЛА

Конверзија типа

```
>>> bytearray(3)
bytearray(b'\x00\x00\x00')
>>> bytearray("ASCII", "utf-8")
bytearray(b'ASCII')
>>> bytearray(range(3))
bytearray(b'\x00\x01\x02')
>>>
```

КОНЗОЛА

Провера поклапања

- поклапање вредности
  - помоћу оператора **`==`** и **`!=`**
- поклапање објеката
  - помоћу оператора **`is`** и **`is not`**

Провера поклапања

```
>>> 3 == 1 + 2
True
>>> 3 is 1 + 2
True
>>> 3 == 3.0
True
>>> 3 is 3.0
False
>>> x = 3
>>> y = 3
>>> x == 3
True
>>> x is y
True
>>>
```

КОНЗОЛА

Провера поклапања

```
>>> a = "Ha!"
>>> b = "Ha!"
>>> a != b
False
>>> a is not b
True
>>>
```

КОНЗОЛА

Провера поклапања

```
>>> p = b"\x61"
>>> p
b'a'
>>> q = b"\x61"
>>> q
b'a'
>>> p == q
True
>>> p is q
False
>>> q is p
False
>>>
```

КОНЗОЛА

Провера поклапања – идентитет

- утврђивање идентитета
  - помоћу уграђене функције **`id()`**
- идентитет
  - целобројни идентификатор објекта
    - не мења током постојања објекта
  - два различита објекта чије постојање не преклапа у времену могу имати исти идентитет

Провера поклапања – идентитет

```
>>> x = 1
>>> id(x)
8790816114336
>>> id(1)
8790816114336
>>> p = b"\x01"
>>> q = b"\x01"
>>> id(p)
51286064
>>> id(q)
51286976
>>>
```

КОНЗОЛА

Оператори – проширена додела

- исказ проширене доделе **?=**
  - припада групи простих исказа
    - као и исказ доделе
  - обједињена примена додатног оператора и оператора доделе
  - **x ?= y** наспрам **x = x ? y**
    - **?** – додатни оператор
    - два исказа нису увек еквивалентна
      - различити токови евалуације
  - оператори проширене доделе
    - **+=**, **-=**, ***=**, **@=**, **/=**, **//=**, **%=**, ****=**, **>>=**, **<<=**, **&=**, **^=**, **|=**

Оператори – основни приоритети (од виших ка нижим)

Ниво	Оператори
1	(izr...), [izr...], {k: v...}, {izr...}
2	x[i], x[i:j], x(arg...), x.ob
3	await x
4	**
5	+x, -x, ~x
6	*, @, /, //, %
7	+, -
8	<<, >>
9	&

Ниво	Оператори
10	^
11	|
12	in, not in, is, is not, <, <=, >, >=, !=, ==
13	not x
14	and
15	or
16	if – else
17	lambda
18	:=

Садржај

Функција

- један вид организовања програмског кода ради његове поновне употребе
  - дефинисање функције
    - помоћу резервисане речи **def**
    - одређивање назива функције
    - одређивање параметара функције
    - одређивање тела функције
      - одређивање повратне вредности функције
  - позивање функције
    - позивање функције помоћу назива функције
    - прослеђивање аргумената при позиву функције
    - преузимање повратне вредности функције

```
def naziv(parametri):
    blok
```

Функција

```
>>> def jedan():
        return 1

>>> jedan()
1
>>> x = jedan()
>>> x
1
>>>
```

КОНЗОЛА

Функција

```
>>> def nema():
        pass

>>> nema()
>>> x = nema()
>>> type(x)
<class 'NoneType'>
>>>
```

КОНЗОЛА

Функција – аргументи

- задавање аргумената у редоследу параметара
- задавање аргумената преко назива параметара

Функција – аргументи

```
>>> def funk(pa, pb, pc):
        print('pa: ', pa)
        print('pb: ', pb)
        print('pc: ', pc)

>>> funk(1, 2, 3)
pa:  1
pb:  2
pc:  3
>>>
```

КОНЗОЛА

Функција – аргументи

```
>>> def funk(pa, pb, pc):
        print("pa: ", pa)
        print("pb: ", pb)
        print("pc: ", pc)

>>> r = funk(1, "b", True)
pa:  1
pb:  b
pc:  True
>>> r
>>>
```

КОНЗОЛА

Функција – аргументи

```
>>> def funk(pa, pb, pc):
        print("pa: ", pa)
        print("pb: ", pb)
        print("pc: ", pc)

>>> funk(pa = 1, pb = "b", pc = True)
pa:  1
pb:  b
pc:  True
>>>
```

КОНЗОЛА

Функција – аргументи

```
>>> def funk(pa, pb, pc):
        print("pa: ", pa)
        print("pb: ", pb)
        print("pc: ", pc)

>>> funk(pc = True, pa = 1, pb = "b")
pa:  1
pb:  b
pc:  True
>>> funk(1, pc = True, pb = "b")
pa:  1
pb:  b
pc:  True
>>>
```

КОНЗОЛА

Функција – аргументи

- подразумеване вредности параметара
  - користе ако при позиву нису задати аргументи за одговарајуће параметре
  - дефинишу при дефинисању функције
  - евалуирају при дефинисању функције

Функција – аргументи

```
>>> def sastavi(x, sep=" "):
        return sep.join(x)

>>> sastavi(["Rhein", "Main", "Donau"])
'Rhein Main Donau'
>>> sastavi(["Rhein", "Main", "Donau"], sep="–")
'Rhein–Main–Donau'
>>>
```

КОНЗОЛА

Функција – аргументи

```
>>> from datetime import datetime
>>> def loguj(x, vreme=datetime.now().time()):
        print(str(vreme) + ": " + x)

>>> loguj("onda")
17:18:59.096142: onda
>>> print(datetime.now().time())
17:19:06.984218
>>> loguj("sada")
17:18:59.096142: sada
>>>
```

КОНЗОЛА

Функција – тело

- тело функције
  - блок који садржи наредбе за извршавање
    - у посебном случају без икаквих редовних наредби
      - потребан нулти исказ **`pass`**
        - исказ без ефекта извршавања
  - обавезна примена увлачења програмског кода у блоку
    - препорука додавања четири размака по сваком нивоу увлачења
      - *PEP 8*: https://peps.python.org/pep-0008/

Функција – повратна вредност

- свака функција има повратну вредност
  - задавање повратне вредности помоћу исказа **`return`**
  - могуће прескочити експлицитно задавање повратне вредности
    - повратна вредност је **`None`**

Садржај

Гранање

- исказ **`if`**
  - припада групи сложених исказа
  - подржава различите гране
    - највише једна грана може бити активирана
    - одређеним врстама грана придружује израз
      - условни израз или ламбда израз
  - врсте грана
    - **`if`** грана
      - на основу придруженог израза одлучује о активацији гране
      - обавезна грана
    - **`elif`** грана
      - на основу придруженог израза одлучује о активацији гране
      - необавезна грана, може их бити произвољно много
    - **`else`** грана
      - долази до активације гране ако ниједна ранија грана није активирана
      - необавезна грана, може јавити највише једном
  - обавезна примена увлачења програмског кода у блоку
    - препорука додавања четири размака по сваком нивоу увлачења (*PEP 8*)

Контрола тока

Гранање

исказ **if**

```
if izraz:
    blok
```

```
if izraz1:
    blok1
elif izraz2:
    blok2
```

```
if izraz1:
    blok1
elif izraz2:
    blok2
elif izraz3:
    blok3
```

```
if izraz:
    blok1
else:
    blok2
```

```
if izraz1:
    blok1
elif izraz2:
    blok2
elif izraz3:
    blok3
else:
    blok4
```

Гранање – исказ **if**

```
>>> x = 3
>>> if isinstance(x, str):
    if x.isascii():
        print("string od ASCII znakova")
    else:
        print("string s posebnim znakovima")
elif isinstance(x, bool):
    print("logička vrednost")
else:
    print("nije ni string ni logička vrednost")

nije ni string ni logička vrednost
>>>
```

КОНЗОЛА

Гранање – исказ **if**

```
>>> def nema():
    pass

>>> if nema():
    print("if")
else:
    print("else")

else
>>>
```

КОНЗОЛА

Гранање – исказ **if**

```
>>> def nema():
    pass

>>> if nema() is None:
    print("if")
else:
    print("else")

if
>>>
```

КОНЗОЛА

Гранање

- условни израз
  - *PEP 308*: https://peps.python.org/pep-0308/
  - два облика
    - пуни облик
      - **`if`** варијанта условног израза (тернарни оператор)
    - скраћени облик

Гранање

условни израз

пуни облик – **`if`** варијанта условног израза (тернарни оператор)

```
a if u else b
```

на основу резултата евалуације за **`u`** одлучује о даљој евалуацији и вредности условног израза

ако резултат евалуације за **`u`** истина (**`True`**)

спроводи евалуација за **`a`** и добијени резултат је вредност условног израза

иначе

спроводи евалуација за **`b`** и добијени резултат је вредност условног израза

Гранање – условни израз

```
>>> a = "T"
>>> b = "s"
>>> c = a if a >= b else b
>>> c
's'
>>>
```

КОНЗОЛА

Гранање – условни израз

```
>>> if "if" if "if" else "else":
    print("if")
else:
    print("else")

if
>>>
```

КОНЗОЛА

Гранање

- условни израз
  - скраћени облик
    - у односу на пуни облик недостаје сегмент **`if`** – **`else`**
    - може послужити и у покушају да симулира тернарни оператор
      - употреба оператора **`and`** и **`or`**

Контрола тока

Гранање – условни израз

```
>>> a = 1
>>> b = 2
>>> a > b and a or b
2
>>> a = 2
>>> b = 1
>>> a > b and a or b
2
>>> a = 0
>>> b = -1
>>> a > b and a or b
-1
>>>
```

КОНЗОЛА

Понављање

- подржане петље (циклуси)
  - исказ **`while`**
  - исказ **`for`**

Понављање

исказ **while**

- о понављању садржаних наредби одлучује на основу придруженог израза
  - понављање док евалуација придруженог израза враћа резултат **True**

```
while izraz:
    blok
```

- могуће придружити додатне наредбе кроз сегмент **else**
  - њихово извршавање може бити само једном и то ако евалуација придруженог израза врати резултат **False**

```
while izraz:
    blok1
else:
    blok2
```

- обавезна примена увлачења програмског кода у блоку
  - препорука додавања четири размака по сваком нивоу увлачења (*PEP 8*)

Понављање – исказ **while**

```
>>> x = 0
>>> while x < 3:
    print(x)
    x += 1
else:
    print(x * x)

0
1
2
9
>>>
```

КОНЗОЛА

Понављање

исказ `for`

извршавање садржаних наредби у виду обраде појединачних елемената од објекта који дозвољава итерирање

```
for x in izraz:
    blok
```

могуће придружити додатне наредбе кроз сегмент `else`

њихово извршавање може бити само једном и то након што сви елементи редовно обрађени без експлицитног прекида

```
for x in izraz:
    blok1
else:
    blok2
```

обавезна примена увлачења програмског кода у блоку

препорука додавања четири размака по сваком нивоу увлачења (*PEP 8*)

Понављање – исказ `for`

```
>>> for i in "321":
    print(i)
else:
    print("Sad!")

3
2
1
Sad!
>>>
```

КОНЗОЛА

Понављање – исказ `for`

```
>>> for i in range(3, 0, -1):
    print(i)
else:
    print("Sad!")

3
2
1
Sad!
>>>
```

КОНЗОЛА

Понављање – исказ `for`

```
>>> for a, b in enumerate(range(5, 11)):
print(a, b)

0 5
1 6
2 7
3 8
4 9
5 10
>>>
```

КОНЗОЛА

Понављање – исказ `for`

```
>>> for a, b in enumerate(range(5, 11), 1):
print(a, b)

1 5
2 6
3 7
4 8
5 9
6 10
>>>
```

КОНЗОЛА

Понављање

- додатна контрола понављања у исказима **`while`** и **`for`**
  - исказ **`break`**
    - прекид свих понављања
    - сегмент **`else`** не бива активиран
  - исказ **`continue`**
    - прекид тренутног понављања и прелазак на следеће

Понављање – исказ **break**

```
>>> for i in "C12":
    if i.isdigit():
        break
    print(i)
else:
    print(".")

C
>>>
```

КОНЗОЛА

Понављање – исказ **continue**

```
>>> for i in "C12":
    if i.isdigit():
        continue
    print(i)
else:
    print(".")

C
.
>>>
```

КОНЗОЛА

Садржај

Структуре података

Основне уграђене структуре података

- листа
- речник
- торка
- скуп

Структуре података

Листа

- линеарна структура података
- елементи листе не морају бити истог типа
- садржај листе променљив
- дужина листе променљива
- представа листе
  - ознака почетка и краја помоћу отворене и затворене угласте заграде `[ ]`
  - елементи раздвојени појединачним запетама `,`

Листа – формирање

```
>>> lst = []
>>> lst
[]
>>> lst = [1, 3, 6, 10, 15, 21, 28]
>>> lst
[1, 3, 6, 10, 15, 21, 28]
>>> lst = [1, "tekst", b"tekst"]
>>> lst
[1, 'tekst', b'tekst']
>>>
```

КОНЗОЛА

Листа – формирање

```
>>> lst = list()
>>> lst
[]
>>> lst = list(range(8))
>>> lst
[0, 1, 2, 3, 4, 5, 6, 7]
>>> lst = list("Ha!")
>>> lst
['H', 'a', '!']
>>>
```

КОНЗОЛА

Листа – формирање

„пуњач” (енгл. *comprehension*)

```
>>> str = "Zdravo!"
>>> lst = [c for c in str]
>>> lst
['Z', 'd', 'r', 'a', 'v', 'o', '!']
>>> lst = [c.upper() for c in str]
>>> lst
['Z', 'D', 'R', 'A', 'V', 'O', '!']
>>> lst = [c.upper() for c in str if c.isalpha()]
>>> lst
['Z', 'D', 'R', 'A', 'V', 'O']
>>>
```

КОНЗОЛА

Листа – формирање

„пуњач” (енгл. *comprehension*)

```
>>> lst = [d + c for c in "123" for d in "-+"]
>>> lst
['-1', '+1', '-2', '+2', '-3', '+3']
>>> lst = [(i, j) for i in range(6) if i % 2 == 1
for j in range (4) if j % 2 == 1]
>>> lst
[(1, 1), (1, 3), (3, 1), (3, 3), (5, 1), (5, 3)]
>>>
```

КОНЗОЛА

Листа – приступ елементима

- оператор `[]`
  - варијанте `[:]` и `[::]`

```
>>> lst = ["alfa", "beta", "gama", "delta"]
>>> lst[0]
'alfa'
>>> lst[3]
'delta'
>>> lst[-1]
'delta'
>>> lst[-4]
'alfa'
>>> lst[::3]
['alfa', 'delta']
>>>
```

КОНЗОЛА

Листа – додавање елемената

оператори **+** и **+=**

```
>>> lst = ["alfa", "beta"]
>>> lst = lst + ["gama", "delta"]
>>> lst
['alfa', 'beta', 'gama', 'delta']
>>> lst += ["epsilon"]
>>> lst
['alfa', 'beta', 'gama', 'delta', 'epsilon']
>>>
```

КОНЗОЛА

Листа – додавање елемената

- метода **extend()**
- метода **append()**

```
>>> lst = ["alfa", "beta"]
>>> lst.extend(["gama", "delta"])
>>> lst
['alfa', 'beta', 'gama', 'delta']
>>> lst.append("epsilon")
>>> lst
['alfa', 'beta', 'gama', 'delta', 'epsilon']
>>>
```

КОНЗОЛА

Листа – додавање елемената

- метода **extend()**
- метода **append()**

```
>>> lst = ["alfa", "beta"]
>>> lst.append(["gama", "delta"])
>>> lst
['alfa', 'beta', ['gama', 'delta']]
>>> lst.extend(["epsilon"])
>>> lst
['alfa', 'beta', ['gama', 'delta'], 'epsilon']
>>>
```

КОНЗОЛА

Листа – додавање елемената

метода **insert()**

```
>>> lst = ["alfa", "beta"]
>>> lst.insert(2, "gama")
>>> lst
['alfa', 'beta', 'gama']
>>> lst.insert(5, "epsilon")
>>> lst
['alfa', 'beta', 'gama', 'epsilon']
>>> lst.insert(3, "delta")
>>> lst
['alfa', 'beta', 'gama', 'delta', 'epsilon']
>>>
```

КОНЗОЛА

Листа – измена елемената

оператор `[]`

```
>>> lst = ["alfa", "beta"]
>>> lst.insert(2, "delta")
>>> lst
['alfa', 'beta', 'delta']
>>> lst[2] = "gama"
>>> lst
['alfa', 'beta', 'gama']
>>>
```

КОНЗОЛА

Листа – измена елемената

оператор **[]**

```
>>> lst = ["alfa", "beta"]
>>> lst += ["epsilon", "epsilon"]
>>> lst
['alfa', 'beta', 'epsilon', 'epsilon']
>>> lst[2:3] = ["gama", "delta"]
>>> lst
['alfa', 'beta', 'gama', 'delta', 'epsilon']
>>>
```

КОНЗОЛА

Листа – уклањање елемената

исказ **del**

```
>>> lst = ["alfa", "beta", "gama", "delta"]
>>> del lst[2]
>>> lst
['alfa', 'beta', 'delta']
>>> del lst[::2]
>>> lst
['beta']
>>> del lst[-1]
>>> lst
[]
>>>
```

КОНЗОЛА

Листа – уклањање елемената

метода **remove()**

```
>>> lst = ["alfa", "beta", "gama", "delta"]
>>> lst.remove("alfa")
>>> lst
['beta', 'gama', 'delta']
>>> lst.append("beta")
>>> lst
['beta', 'gama', 'delta', 'beta']
>>> lst.remove("beta")
>>> lst
['gama', 'delta', 'beta']
>>>
```

КОНЗОЛА

Листа – уклањање елемената

метода **remove()**

```
>>> lst = ["alfa", "beta", "gama", "delta"]
>>> lst.remove("epsilon")
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    lst.remove("epsilon")
ValueError: list.remove(x): x not in list
>>> lst
['alfa', 'beta', 'gama', 'delta']
>>>
```

КОНЗОЛА

Листа – уклањање елемената

метода **pop()**

```
>>> lst = ["alfa", "beta", "gama", "delta"]
>>> lst.pop()
'delta'
>>> lst
['alfa', 'beta', 'gama']
>>> lst.pop(0)
'alfa'
>>> lst
['beta', 'gama']
>>> lst.pop(1)
'gama'
>>> lst
['beta']
>>>
```

КОНЗОЛА

Листа – копирање

функција `list()`

```
>>> lst_a = ["Ana", "voli", "Milovana"]
>>> lst_b = list(lst_a)
>>> lst_b
['Ana', 'voli', 'Milovana']
>>> id(lst_a)
41865856
>>> id(lst_b)
31444288
>>>
```

КОНЗОЛА

Листа – копирање

оператор `[]` – варијанте `[:]` и `[::]`

```
>>> lst_a = ["Ana", "voli", "Milovana"]
>>> lst_b = lst_a[:]
>>> lst_b
['Ana', 'voli', 'Milovana']
>>> lst_c = lst_a[::1]
>>> lst_c
['Ana', 'voli', 'Milovana']
>>> id(lst_a)
41866176
>>> id(lst_b)
47998720
>>> id(lst_c)
48026624
>>>
```

КОНЗОЛА

Листа – копирање

- метода **`copy()`**
  - еквивалентно коришћењу оператора **`[:]`**

```
>>> lst_a = [b"ASCII", b"EBCDIC", b"Unicode"]
>>> lst_b = lst_a.copy()
>>> lst_b
[b'ASCII', b'EBCDIC', b'Unicode']
>>> id(lst_a)
47934464
>>> id(lst_b)
48026432
>>> id(lst_a[0])
47643744
>>> id(lst_b[0])
47643744
>>>
```

КОНЗОЛА

Листа – помоћне радње

функција **len()** и метода **count()**

```
>>> lst = ["alfa", "beta", "gama", "delta"]
>>> type(lst)
<class 'list'>
>>> len(lst)
4
>>> lst.count("alfa")
1
>>> lst.append("alfa")
>>> lst.count("alfa")
2
>>> lst.count("")
0
>>>
```

КОНЗОЛА

Листа – помоћне радње

- оператори **`in`** и **`not in`**

```
>>> lst = ["alfa", "beta", "gama", "delta"]
>>> "alfa" in lst
True
>>> "epsilon" in lst
False
>>> "delta" not in lst
False
>>>
```

КОНЗОЛА

Листа – помоћне радње

метода **index()**

```
>>> lst = ["alfa", "beta", "gama", "delta"]
>>> lst.index("gama")
2
>>> lst.append("gama")
>>> lst
['alfa', 'beta', 'gama', 'delta', 'gama']
>>> lst.index("gama")
2
>>> lst.index("psi")
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    lst.index("psi")
ValueError: 'psi' is not in list
>>>
```

КОНЗОЛА

Листа – помоћне радње

метода `sort()`

```
>>> lst = ["alfa", "beta", "gama", "delta"]
>>> lst.sort()
>>> lst
['alfa', 'beta', 'delta', 'gama']
>>> lst = ["алфа", "бета", "гама", "делта"]
>>> lst.sort()
>>> lst
['алфа', 'бета', 'гама', 'делта']
>>> lst.sort(reverse=True)
>>> lst
['делта', 'гама', 'бета', 'алфа']
>>>
```

КОНЗОЛА

Листа – помоћне радње

функција **sorted()**

```
>>> lst = ["alfa", "beta", "gama", "delta"]
>>> sorted(lst)
['alfa', 'beta', 'delta', 'gama']
>>> lst
['alfa', 'beta', 'gama', 'delta']
>>> sorted(lst, reverse=True)
['gama', 'delta', 'beta', 'alfa']
>>> lst
['alfa', 'beta', 'gama', 'delta']
>>>
```

КОНЗОЛА

Листа – помоћне радње

метода **reverse()** и функција **reversed()**

```
>>> lst = ["alfa", "gama", "epsilon"]
>>> lst.reverse()
>>> lst
['epsilon', 'gama', 'alfa']
>>> type(reversed(lst))
<class 'list_reverseiterator'>
>>> for e in reversed(lst):
    print(e)

alfa
gama
epsilon
>>>
```

КОНЗОЛА

Листа – помоћне радње

функција **zip()**

```
>>> lst_gr = ["alfa", "lambda", "pi"]
>>> lst_rs = ["А", "Л", "П"]
>>> for gr, rs in zip(lst_gr, lst_rs):
    "{:^12} : {:^5}".format(gr, rs)

'    alfa       :   А   '
'   lambda      :   Л   '
'     pi        :   П   '
>>>
```

КОНЗОЛА

Речник

- структура података чији елементи су парови *кључ–садржај*
- кључ
  - вредност кључа је јединствена на нивоу речника
  - тип кључа је неки од непроменљивих типова
  - тип кључа не мора бити исти у свим елементима
- садржај
  - вредност садржаја не мора бити јединствена на нивоу речника
  - садржају може приступити на основу вредности придруженог кључа
  - тип садржаја не мора бити исти у свим елементима
- садржај речника променљив
- дужина речника променљива
- представа речника
  - ознака почетка и краја помоћу отворене и затворене витичасте заграде **{ }**
  - елементи раздвојени појединачним запетама **,**
  - унутар елемента, кључ и садржај раздвојени двотачком **:**

Речник – формирање

```
>>> rcnk = {}
>>> rcnk
{}
>>> rcnk = {"ao": "Angola", "ie": "Ireland", "rs":
"Serbia"}
>>> rcnk
{'ao': 'Angola', 'ie': 'Ireland', 'rs': 'Serbia'}
>>> rcnk = {"a": 10, "b": 11.0, "c": "12"}
>>> rcnk
{'a': 10, 'b': 11.0, 'c': '12'}
>>> rcnk = {10: "a", 11.0: "b", "12": 0xC}
>>> rcnk
{10: 'a', 11.0: 'b', '12': 12}
>>>
```

КОНЗОЛА

Речник – формирање

```
>>> rcnk = dict([["BČ", "Bečej"], ["RU", "Ruma"]])
>>> rcnk
{'BČ': 'Bečej', 'RU': 'Ruma'}
>>> rcnk = dict((["BČ", "Bečej"], ["RU", "Ruma"]))
>>> rcnk
{'BČ': 'Bečej', 'RU': 'Ruma'}
>>> rcnk = dict(["1A", "2B", "3C", "4D", "5E"])
>>> rcnk
{'1': 'A', '2': 'B', '3': 'C', '4': 'D', '5': 'E'}
>>>
```

КОНЗОЛА

Речник – формирање

„пуњач” (енгл. *comprehension*)

```
>>> rcnk = {i: hex(i) for i in range(50, 56)}
>>> rcnk
{50: '0x32', 51: '0x33', 52: '0x34', 53: '0x35',
54: '0x36', 55: '0x37'}
>>> rcnk = {i: j for i, j in enumerate("αβγδ", 1)}
>>> rcnk
{1: 'α', 2: 'β', 3: 'γ', 4: 'δ'}
>>>
```

КОНЗОЛА

Речник – приступ елементима

оператор `[]`

```
>>> r = {"а": "α", "б": "β", "г": "γ"}
>>> r
{'а': 'α', 'б': 'β', 'г': 'γ'}
>>> r["б"]
'β'
>>> r["д"]
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    r["д"]
KeyError: 'д'
>>>
```

КОНЗОЛА

Речник – приступ елементима

метода **get()**

```
>>> r = {"а": "α", "б": "β", "г": "γ"}
>>> r
{'а': 'α', 'б': 'β', 'г': 'γ'}
>>> r.get("г")
'γ'
>>> r.get("д")
>>> r.get("д", "?")
'?'
>>>
```

КОНЗОЛА

Речник – додавање елемената

оператор `[]`

```
>>> r = {"а": "α", "б": "β", "г": "γ"}
>>> r
{'а': 'α', 'б': 'β', 'г': 'γ'}
>>> r["д"] = "Δ"
>>> r
{'а': 'α', 'б': 'β', 'г': 'γ', 'д': 'Δ'}
>>> r["д"] = "б"
>>> r
{'а': 'α', 'б': 'β', 'г': 'γ', 'д': 'б'}
>>>
```

КОНЗОЛА

Речник – додавање елемената

метода **update()**

```
>>> r1 = {"BČ": "Bečej", "RU": "Ruma"}
>>> r2 = {"NI": "Niš", "PI": "Pirot"}
>>> r1.update(r2)
>>> r1
{'BČ': 'Bečej', 'RU': 'Ruma', 'NI': 'Niš', 'PI':
'Pirot'}
>>>
```

КОНЗОЛА

Речник – уклањање елемената

исказ **del**

```
>>> r = {"BČ": "Bečej", "NI": "Niš", "PI":
"Pirot", "RU": "Ruma"}
>>> del r["NI"]
>>> r
{'BČ': 'Bečej', 'PI': 'Pirot', 'RU': 'Ruma'}
>>> del r["pi"]
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    del r["pi"]
KeyError: 'pi'
>>>
```

КОНЗОЛА

Речник – уклањање елемената

метода **pop()**

```
>>> r = {"BČ": "Bečej", "NI": "Niš", "PI":
"Pirot", "RU": "Ruma"}
>>> r.pop("NI")
'Niš'
>>> r.pop("IN", "?")
'?'
>>> r
{'BČ': 'Bečej', 'PI': 'Pirot', 'RU': 'Ruma'}
>>>
```

КОНЗОЛА

Речник – уклањање елемената

метода **popitem()**

```
>>> r = {"BČ": "Bečej", "NI": "Niš", "PI":
"Pirot", "RU": "Ruma"}
>>> r["IN"] = "Indija"
>>> r
{'BČ': 'Bečej', 'NI': 'Niš', 'PI': 'Pirot', 'RU':
'Ruma', 'IN': 'Indija'}
>>> r.popitem()
('IN', 'Indija')
>>> r
{'BČ': 'Bečej', 'NI': 'Niš', 'PI': 'Pirot', 'RU':
'Ruma'}
>>>
```

КОНЗОЛА

Речник – копирање

функција `dict()`

```
>>> r_a = {"A": "amper", "T": "tesla", "W": "vat"}
>>> r_b = dict(r_a)
>>> r_b
{'A': 'amper', 'T': 'tesla', 'W': 'vat'}
>>> id(r_a)
47672320
>>> id(r_b)
46223104
>>>
```

КОНЗОЛА

Речник – копирање

метода **copy()**

```
>>> r_a = {"A": "amper", "T": "tesla", "W": "vat"}
>>> r_b = r_a.copy()
>>> r_b
{'A': 'amper', 'T': 'tesla', 'W': 'vat'}
>>> id(r_a)
47672000
>>> id(r_b)
47796928
>>>
```

КОНЗОЛА

Речник – помоћне радње

- функција **`len()`**
- оператори **`in`** и **`not in`**

```
>>> r = {"A": "amper", "T": "tesla", "W": "vat"}
>>> r
{'A': 'amper', 'T': 'tesla', 'W': 'vat'}
>>> len(r)
3
>>> "T" in r
True
>>> "Hz" in r
False
>>>
```

КОНЗОЛА

Речник – помоћне радње

метода **items()**

резултат је поглед на елементе речника

```
>>> r = {"A": "amper", "T": "tesla", "W": "vat"}
>>> r.items()
dict_items([('A', 'amper'), ('T', 'tesla'), ('W',
'vat')])
>>> type(r.items())
<class 'dict_items'>
>>> for i in r.items():
    print(type(i), i)

<class 'tuple'> ('A', 'amper')
<class 'tuple'> ('T', 'tesla')
<class 'tuple'> ('W', 'vat')
>>>
```

КОНЗОЛА

Речник – помоћне радње

метода `items()`

```
>>> r = {"A": "amper", "T": "tesla", "W": "vat"}
>>> for k, s in r.items():
    print(k, "::", s)

A :: amper
T :: tesla
W :: vat
>>>
```

КОНЗОЛА

Речник – помоћне радње

метода **keys()**

резултат је поглед на кључеве речника

```
>>> r = {"A": "amper", "T": "tesla", "W": "vat"}
>>> r.keys()
dict_keys(['A', 'T', 'W'])
>>> type(r.keys())
<class 'dict_keys'>
>>> for k in r.keys():
    print(k)

A
T
W
>>>
```

КОНЗОЛА

Речник – помоћне радње

метода **values()**

резултат је поглед на садржаје речника

```
>>> r = {"A": "amper", "T": "tesla", "W": "vat"}
>>> r.values()
dict_values(['amper', 'tesla', 'vat'])
>>> type(r.values())
<class 'dict_values'>
>>> for s in r.values():
    print(s)

amper
tesla
vat
>>>
```

КОНЗОЛА

Структуре података

Торка

- линеарна структура података
  - одговара концепту н-торке
- елементи торке не морају бити истог типа
- садржај торке непроменљив
- дужина торке непроменљива
- представа торке
  - ознака почетка и краја помоћу отворене и затворене заграде `( )`
  - елементи раздвојени појединачним запетама `,`

Торка – формирање

```
>>> t = ()
>>> t
()
>>> type(t)
<class 'tuple'>
>>> t = tuple()
>>> t
()
>>>
```

КОНЗОЛА

Торка – формирање

```
>>> u = "DDR",
>>> u
('DDR',)
>>> u = ("DDR")
>>> u
'DDR'
>>> type(u)
<class 'str'>
>>> u = ("DDR",)
>>> u
('DDR',)
>>> type(u)
<class 'tuple'>
>>>
```

КОНЗОЛА

Торка – формирање

```
>>> v = "DDR", "DDR2"
>>> v
('DDR', 'DDR2')
>>> v = "DDR", "DDR2",
>>> v
('DDR', 'DDR2')
>>> v = ("DDR", "DDR2", "DDR3")
>>> v
('DDR', 'DDR2', 'DDR3')
>>>
```

КОНЗОЛА

Торка – приступ елементима

- оператор `[]`
  - варијанте `[:]` и `[::]`

```
>>> d = ("DDR", "DDR2", "DDR3", "DDR4")
>>> d[1]
'DDR2'
>>> d[-2]
'DDR3'
>>> d[2:3]
('DDR3',)
>>> d[::2]
('DDR', 'DDR3')
>>>
```

КОНЗОЛА

Торка – приступ елементима

```
>>> d = ("DDR", "DDR2", "DDR3", "DDR4")
>>> e = tuple(d)
>>> f = d[:]
>>> id(d), id(e), id(f)
(46130576, 46130576, 46130576)
>>>
```

КОНЗОЛА

Торка – спајање и умножавање

оператори **+**, *, **+=** и ***=**

```
>>> m = ("0",)
>>> m *= 2
>>> m
('0', '0')
>>> m += ("1",)
>>> m
('0', '0', '1')
>>> m + m
('0', '0', '1', '0', '0', '1')
>>> m * 2
('0', '0', '1', '0', '0', '1')
>>>
```

КОНЗОЛА

Торка – додела

оператор **=**

```
>>> c = (4.85, 45.75)
>>> lat, long = c
>>> lat
4.85
>>> long
45.75
>>> long, lat = lat, long
>>> c = lat, long
>>> c
(45.75, 4.85)
>>>
```

КОНЗОЛА

Торка – помоћне радње

- оператори **in** и **not in**
- функција **len()**
- метода **index()**

```
>>> d = ("DDR", "DDR2", "DDR3", "DDR4")
>>> "DDR" in d
True
>>> "DDR1" not in d
True
>>> len(d)
4
>>> d.index("DDR4")
3
>>>
```

КОНЗОЛА

Структуре података

Скуп

- структура података
  - одговара концепту математичког скупа
- није дозвољено да два елемента скупа буду једнака по вредности
- елементи скупа не морају бити истог типа
- садржај скупа променљив
- величина скупа променљива
- представа скупа
  - ознака почетка и краја помоћу отворене и затворене витичасте заграде `{ }`
  - елементи раздвојени појединачним запетама `,`

Скуп – формирање

```
>>> s = set()
>>> s
set()
>>> type(s)
<class 'set'>
>>> type({})
<class 'dict'>
>>> s = {15, 30, 45}
>>> s
{45, 30, 15}
>>> s = {0xA, "B", True}
>>> s
{True, 10, 'B'}
>>>
```

КОНЗОЛА

Скуп – формирање

```
>>> s = set("albatros")
>>> s
{'r', 'b', 't', 'o', 'a', 'l', 's'}
>>> s = set(("a", "l", "b", "a", "t", "r", "o",
"s"))
>>> s
{'r', 'b', 't', 'o', 'a', 'l', 's'}
>>> s = set({"A": 65, "a": 97})
>>> s
{'A', 'a'}
>>> s = set([i**2 for i in range(-3, 4)])
>>> s
{0, 9, 4, 1}
>>>
```

КОНЗОЛА

Скуп – формирање

„пуњач” (енгл. *comprehension*)

```
>>> s = {i**2 for i in range(-3, 4)}
>>> s
{0, 9, 4, 1}
>>> s = {i for i in {1, 1.0}}
>>> s
{1}
>>> s = {i for i in {1.0, 1}}
>>> s
{1.0}
>>>
```

КОНЗОЛА

Скуп – додавање елемената

метода **add()**

```
>>> vp = {"Балакирјев", "Бородин", "Кјуј",
"Мусоргски"}
>>> vp
{'Бородин', 'Кјуј', 'Мусоргски', 'Балакирјев'}
>>> vp.add("Римски-Корсаков")
>>> vp
{'Бородин', 'Кјуј', 'Балакирјев', 'Мусоргски',
'Римски-Корсаков'}
>>> vp.add("Римски-Корсаков")
>>> vp
{'Бородин', 'Кјуј', 'Балакирјев', 'Мусоргски',
'Римски-Корсаков'}
>>>
```

КОНЗОЛА

Скуп – додавање елемената

метода **update()**

```
>>> vp = {"Балакирјев"}
>>> vp.update("Бородин")
>>> vp
{'Б', 'о', 'р', 'Балакирјев', 'и', 'н', 'д'}
>>> vp.update({"Кјуј"})
>>> vp
{'Б', 'о', 'р', 'Балакирјев', 'Кјуј', 'и', 'н',
'д'}
>>>
```

КОНЗОЛА

Скуп – додавање елемената

метода **update()**

```
>>> vp = {"Балакирјев"}
>>> vp.update({"Бородин", "Кјуј"})
>>> vp
{'Бородин', 'Кјуј', 'Балакирјев'}
>>>
```

КОНЗОЛА

Скуп – додавање елемената

метода **update()**

```
>>> vp = {"Балакирјев"}
>>> vp.update({"Бородин", "Кјуј"}, {"Мусоргски",
"Римски-Корсаков"})
>>> vp
{'Бородин', 'Кјуј', 'Мусоргски', 'Балакирјев',
'Римски-Корсаков'}
>>>
```

КОНЗОЛА

Скуп – уклањање елемената

метода **remove()**

```
>>> vp = {"Балакирјев", "Бородин", "Кјуј",
"Мусоргски", "Римски-Корсаков"}
>>> vp.remove("Kjyj")
>>> vp
{'Римски-Корсаков', 'Бородин', 'Балакирјев',
'Мусоргски'}
>>> vp.remove("Kjyj")
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    vp.remove("Kjyj")
KeyError: 'Kjyj'
>>>
```

КОНЗОЛА

Скуп – уклањање елемената

метода **discard()**

```
>>> vp = {"Балакирјев", "Бородин", "Кјуј",
"Мусоргски", "Римски-Корсаков"}
>>> vp.discard("Kjyj")
>>> vp
{'Балакирјев', 'Римски-Корсаков', 'Мусоргски',
'Бородин'}
>>> vp.discard("Kjyj")
>>> vp
{'Балакирјев', 'Римски-Корсаков', 'Мусоргски',
'Бородин'}
>>>
```

КОНЗОЛА

Скуп – уклањање елемената

метода **pop()**

```
>>> vp = {"Балакирјев", "Бородин", "Кјуј",
"Мусоргски", "Римски-Корсаков"}
>>> vp.pop()
'Кјуј'
>>> vp
{'Балакирјев', 'Бородин', 'Мусоргски', 'Римски-
Корсаков'}
>>> vp.pop()
'Балакирјев'
>>> vp.pop()
'Бородин'
>>> vp.pop()
'Мусоргски'
>>>
```

КОНЗОЛА

Скуп – уклањање елемената

метода **pop()**

```
>>> vp = {"Балакирјев", "Бородин", "Кјуј",
"Мусоргски", "Римски-Корсаков"}
>>> for i in vp:
    print(i)

Кјуј
Балакирјев
Бородин
Мусоргски
Римски-Корсаков
>>> vp.pop()
'Кјуј'
>>>
```

КОНЗОЛА

Скуп – копирање

функција **`set()`**

```
>>> s = {"H", "He", "Li", "Be"}
>>> t = set(s)
>>> id(s), id(t)
(47894080, 47894304)
>>> s
{'H', 'Be', 'Li', 'He'}
>>> t
{'H', 'Be', 'Li', 'He'}
>>>
```

КОНЗОЛА

Скуп – копирање

метода **copy()**

```
>>> s = {"H", "He", "Li", "Be"}
>>> t = s.copy()
>>> id(s), id(t)
(47894080, 47892960)
>>> s
{'He', 'Be', 'H', 'Li'}
>>> t
{'He', 'Be', 'H', 'Li'}
>>>
```

КОНЗОЛА

Скуп – помоћне радње

- функција **`len()`**
- оператори **`in`** и **`not in`**

```
>>> s = {"H", "He", "Li", "Be"}
>>> len(s)
4
>>> "" in s
False
>>> "He" in s
True
>>> "B" not in s
True
>>>
```

КОНЗОЛА

Структуре података

Скуп – радње над скуповима

- унија
  - оператор `|`
  - метода **`union()`**
- пресек
  - оператор **`&`**
  - метода **`intersection()`**
- разлика
  - оператор **`-`**
  - метода **`difference()`**
- симетрична разлика
  - оператор **`^`**
  - метода **`symmetric_difference()`**

Скуп – радње над скуповима

унија

```
>>> s = {"H", "He", "Li", "Be"}
>>> t = {"He", "Ne", "Ar", "Kr"}
>>> s | t
{'H', 'Ar', 'Kr', 'He', 'Be', 'Ne', 'Li'}
>>> s.union(t)
{'H', 'Ar', 'Kr', 'He', 'Be', 'Ne', 'Li'}
>>> s
{'Li', 'H', 'Be', 'He'}
>>> t
{'Ne', 'Ar', 'Kr', 'He'}
>>>
```

КОНЗОЛА

Скуп – радње над скуповима

пресек

```
>>> s = {"H", "He", "Li", "Be"}
>>> t = {"He", "Ne", "Ar", "Kr"}
>>> s & t
{'He'}
>>> s.intersection(t)
{'He'}
>>> s
{'Be', 'Li', 'H', 'He'}
>>> t
{'Ne', 'Kr', 'He', 'Ar'}
>>>
```

КОНЗОЛА

Скуп – радње над скуповима

разлика

```
>>> s = {"H", "He", "Li", "Be"}
>>> t = {"He", "Ne", "Ar", "Kr"}
>>> s - t
{'Li', 'Be', 'H'}
>>> s.difference(t)
{'Li', 'Be', 'H'}
>>> s
{'Be', 'He', 'Li', 'H'}
>>> t
{'Ne', 'He', 'Ar', 'Kr'}
>>>
```

КОНЗОЛА

Скуп – радње над скуповима

симетрична разлика

```
>>> s = {"H", "He", "Li", "Be"}
>>> t = {"He", "Ne", "Ar", "Kr"}
>>> s ^ t
{'Ar', 'H', 'Ne', 'Be', 'Kr', 'Li'}
>>> s.symmetric_difference(t)
{'Ar', 'H', 'Ne', 'Be', 'Kr', 'Li'}
>>> s
{'Be', 'H', 'He', 'Li'}
>>> t
{'Kr', 'Ar', 'He', 'Ne'}
>>>
```

КОНЗОЛА

Скуп – односи између скупова

- подскуп
  - оператор **<=**
  - метода **issubset()**
- прави подскуп
  - оператор **<**
- надскуп
  - оператор **>=**
  - метода **issuperset()**
- прави надскуп
  - оператор **>**

Скуп – односи између скупова

подскуп и прави подскуп

```
>>> u = {"He", "Ne"}
>>> v = {"He", "Ne", "Ar", "Kr"}
>>> u <= v
True
>>> u.issubset(v)
True
>>> u < v
True
>>> v < u
False
>>>
```

КОНЗОЛА

Скуп – односи између скупова

надскуп и прави надскуп

```
>>> u = {"He", "Ne"}
>>> v = {"He", "Ne", "Ar", "Kr"}
>>> u >= v
False
>>> u.issuperset(v)
False
>>> u > v
False
>>> v > u
True
>>>
```

КОНЗОЛА

Уграђени типови секвенци

листа

`list` – променљива секвенца

торка

`tuple` – непроменљива секвенца

распон

`range` – непроменљива секвенца бројева

текстуална секвенца (стринг)

`str` – непроменљива секвенца кодних тачака у систему *Unicode*

бинарне секвенце

`bytes` – непроменљива секвенца бајтова

`bytearray` – променљива секвенца бајтова

Уграђени типови секвенци

- опште радње над секвенцама
  - оператори **`in`** и **`not in`**
  - оператори **`+`** и **`*`**
  - оператори **`[]`**, **`[:]`** и **`[::]`**
  - функције **`len()`**, **`min()`**, **`max()`**, **`index()`** и **`count()`**

Уграђени типови секвенци

- радње над променљивим секвенцама
  - оператори **`+=`** и **`*=`**
  - додела уз коришћење оператора **`[]`**, **`[:]`** и **`[::]`**
  - исказ **`del`**
    - уз употребу оператора **`[]`**, **`[:]`** и **`[::]`**
  - методе **`append()`**, **`extend()`** и **`insert()`**
  - методе **`pop()`**, **`remove()`** и **`clear()`**
  - методе **`reverse()`**
  - метода **`copy()`**

Садржај

Библиотеке

Основни концепти у раду с библиотекама

- простор назива
- модул
- пакет

Простор назива

- евиденција назива и одговарајућих објеката који су на располагању приликом извршења програмског кода
  - енгл. *namespace*
  - техничка изведба
    - речник парова *назив објекта – објекат којем је придружен назив*

Простор назива

основне врсте простора назива

- **уграђени**
  - настаје покретањем интерпретера
    - обухвата креирање простора назива за уграђене идентификаторе који постоје у модулу **`builtins`**
      - омогућује употребу уграђених константи и уграђених функција
- **глобални**
  - настаје покретањем програма, односно модула
    - ажурира се током извршавања програма
  - преглед садржаја помоћу функције **`globals()`**
- **локални**
  - настаје покретањем посебних делова кода – пример тога је функција
    - ажурира се током извршавања посебних делова кода
  - преглед садржаја помоћу функције **`locals()`**
    - на нивоу модула, резултат позива се поклапа с резултатом позива функције **`globals()`**

Простор назива

```
>>> globals()
{'__name__': '__main__', '__doc__': None,
'__package__': None, '__loader__': <class
'_frozen_importlib.BuiltinImporter'>, '__spec__':
None, '__annotations__': {}, '__builtins__':
<module 'builtins' (built-in)>}
>>> b = True
>>> globals()
{'__name__': '__main__', '__doc__': None,
'__package__': None, '__loader__': <class
'_frozen_importlib.BuiltinImporter'>, '__spec__':
None, '__annotations__': {}, '__builtins__':
<module 'builtins' (built-in)>, 'b': True}
>>>
```

КОНЗОЛА

Простор назива

```
>>> locals()
{'__name__': '__main__', '__doc__': None,
'__package__': None, '__loader__': <class
'_frozen_importlib.BuiltinImporter'>, '__spec__':
None, '__annotations__': {}, '__builtins__':
<module 'builtins' (built-in)>}
>>> gr = 8
>>> locals()
{'__name__': '__main__', '__doc__': None,
'__package__': None, '__loader__': <class
'_frozen_importlib.BuiltinImporter'>, '__spec__':
None, '__annotations__': {}, '__builtins__':
<module 'builtins' (built-in)>, 'gr': 8}
>>>
```

КОНЗОЛА

Простор назива

```
>>> p = "globalni"
>>> def flokalni():
    plokalni = "lokalni"
    print("flokalni:", locals().keys())

    def flokalniji():
        plokalniji = "lokalniji"
        print("flokalniji:", locals().keys())

    flokalniji()

>>> print("globalni:", globals().keys())
globalni: dict_keys(['__name__', '__doc__', '__package__', '__loader__',
'__spec__', '__annotations__', '__builtins__', 'p', 'flokalni'])
>>> flokalni()
flokalni: dict_keys(['plokalni'])
flokalniji: dict_keys(['plokalniji'])
>>>
```

КОНЗОЛА

Простор назива

- блок програмског кода (блок)
  - програмски код који се извршава као целина
  - примери блокова
    - модул
    - тело функције
    - дефиниција класе
    - наредба издата у интерактивном режиму
    - скрипт датотека
    - скрипт наредба
    - аргумент уграђене функције **`eval()`**
    - аргумент уграђене функције **`exec()`**

Простор назива

- увођење назива објекта
  - повезивање назива с објектом
  - назив који је уведен у неком блоку је локални назив за тај блок
    - осим ако није посебно означен као нелокални или глобални
      - помоћу исказа **`nonlocal`** или **`global`**
  - назив који је уведен на нивоу модула је глобални назив

Простор назива

- препознавање назива објекта
  - сваки назив који се појављује у програмском коду потребно је препознати
    - утврдити на који се објекат дати назив односи
  - назив који се уводи у програмски код не мора бити видљив свуда у програмском коду
    - ако је назив видљив у одређеном делу програмског кода онда је могуће преко тог назива користити њему придружени објекат у датом делу програмског кода
  - досег назива представља видљивост назива

Простор назива

- препознавање назива објекта
  - дати назив тражи се у доступним просторима назива
    - прво се тражи у локалном простору назива
      - онда се редом тражи у надређеним локалним просторима назива
        - од најближег па до све даљих надређених локалних простора назива
    - потом у глобалном простору назива
    - потом у уграђеном простору назива
  - то што назив постоји у доступним просторима назива не мора значити да га је могуће произвољно користити

Простор назива

```
>>> p = "globalni"
>>> def flokalni():
    plokalni = "lokalni"
    print("flokalni: ", p)
    print("flokalni: ", plokalni)

    def flokalniji():
        plokalniji = "lokalniji"
        print("flokalniji: ", p)
        print("flokalniji: ", plokalni)
        print("flokalniji: ", plokalniji)

    flokalniji()

>>> flokalni()
flokalni:  globalni
flokalni:  lokalni
flokalniji:  globalni
flokalniji:  lokalni
flokalniji:  lokalniji
>>>
```

КОНЗОЛА

Простор назива

```
>>> r = list(range(1, 6))
>>> def prosiri(x):
    r += [x]

>>> prosiri(6)
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    prosiri(6)
  File "<pyshell#3>", line 2, in prosiri
    r += [x]
UnboundLocalError: local variable 'r' referenced
before assignment
>>> r
[1, 2, 3, 4, 5]
>>>
```

КОНЗОЛА

Простор назива

препознавање назива објекта

исказ **`global`**

- назнака да се у контексту матичног блока исказа називи наведени у исказу односе на називе из највиших простора назива (глобалног или уграђеног)
  - прво су наведени називи тражени у глобалном па у уграђеном простору назива
  - ако је назив пронађен онда се његова даља употреба у матичном блоку исказа повезује с простором назива у којем је пронађен
  - ако неки назив није пронађен онда је омогућено да касније увођење тог назива у оквиру матичног блока исказа буде повезано с глобалним простором назива

Простор назива

```
>>> r = list(range(1, 6))
>>> def prosiri(x):
    global r
    r += [x]

>>> prosiri(6)
>>> r
[1, 2, 3, 4, 5, 6]
>>>
```

КОНЗОЛА

Простор назива

```
>>> r = list(range(1, 6))
>>> def prosiri(x):
    r.append(x)

>>> prosiri(6)
>>> r
[1, 2, 3, 4, 5, 6]
>>>
```

КОНЗОЛА

Простор назива

```
>>> def prosiri(x):
    global r
    r = [x]

>>> r
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    r
NameError: name 'r' is not defined
>>> prosiri(1)
>>> r
[1]
>>>
```

КОНЗОЛА

Простор назива

- препознавање назива објекта
  - исказ **`nonlocal`**
    - назнака да се називи наведени у исказу односе на постојеће називе који нису ни у непосредно локалном ни у највишим просторима назива
      - наведени називи су тражени у надређеним локалним просторима назива
        - од најближег па до све даљих надређених локалних простора назива
    - након извршења исказа могуће је даље у матичном блоку користити наведене називе по потреби

Простор назива

```
>>> r = list(range(1, 6))
>>> def prosiri(x):
    nonlocal r
    r += [x]

SyntaxError: no binding for nonlocal 'r' found
>>> def prosiri(x):
    nonlocal x

SyntaxError: name 'x' is parameter and nonlocal
>>>
```

КОНЗОЛА

Простор назива

```
>>> p = "globalni"
>>> def flokalni():
    plokalni = "lokalni"

    def flokalniji():
        plokalni = "/"

    flokalniji()

    print("plokalni:", plokalni)

>>> flokalni()
plokalni: lokalni
>>>
```

КОНЗОЛА

Простор назива

```
>>> p = "globalni"
>>> def flokalni():
    plokalni = "lokalni"

    def flokalniji():
        nonlocal plokalni
        plokalni = "/"

    flokalniji()

    print("plokalni:", plokalni)

>>> flokalni()
plokalni: /
>>>
```

КОНЗОЛА

Библиотеке

Модул

- датотека која садржи програмски код писан у језику *Python*
  - екстензија датотеке је *py*
  - може служити као библиотека
    - програмски код намењен употреби у другим програмима
    - обично садржи дефиниције константи, функција и класа
    - постоји велики број уграђених модула у језику *Python*
  - може служити као скрипта
    - програм који комплетно смештен у једну датотеку
- модул у систему типова
  - модул је типа **`module`**

Модул

```
1  def izdvajanje(rec):
2      klj = list(rec.keys())
3      sad = list(rec.values())
4      return klj, sad
5
6  def spajanje(klj, sad):
7      rec = dict(zip(klj, sad))
8      return rec
9
10 def zamena(rec):
11     klj, sad = izdvajanje(rec)
12     rec_nov = spajanje(sad, klj)
13     return rec_nov
14
```

modul.py

Модул – учитавање

- стављање садржаја модула на располагање
  - садржај модула бива извршен приликом учитавања
  - простор назива бива креиран за модул приликом учитавања
  - модул мора бити доступан на очекиваним путањама (до директоријума или датотека у формату *zip*)
    - на подешеним путањама
      - објекат **`sys.path`**
        - радни директоријум
        - променљива окружења **`PYTHONPATH`**
      - …
- након учитавања модул и његови делови доступни или преко назива или преко псеудонима који задат приликом учитавања
  - садржај модула ставља се на располагање само по првом учитавању
    - накнадна учитавања промењеног модула немају ефекта у истој сесији

Модул – учитавање

рад с радним директоријумом

помоћу функција из уграђеног модула **os**

очитавање путање радног директоријума

os.getcwd()

промена путање радног директоријума

os.chdir(putanja)

приказ садржаја директоријума

os.listdir(putanja)

ако путања није наведена, приказује се садржај радног директоријума

Модул – учитавање

```
>>> import os
>>> os.chdir("/home/prog/moduli")
>>> os.getcwd()
'/home/prog/moduli'
>>> os.listdir()
['modul.py']
>>>
```

КОНЗОЛА

Модул – учитавање

исказ **import**

учитавање једног или више модула преко назива модула (без екстензије)

```
import modul
import modul1, modul2
```

учитавање модула под псеудонимом

```
import modul as m
import modul1 as m1, modul2 as m2
```

учитавање дела модула

```
from modul import deo
from modul import deo1, deo2
from modul import deo as d
from modul import deo1 as d1, deo2 as d2
```

учитавање свих јавних делова модула

```
from modul import *
```

унутар модула је посебно назначено који су делови јавни (називи у објекту **__all__**)

ако није назначено онда су јавни сви делови чији назив не почиње знаком **_**

Модул

учитавање и коришћење корисничког модула

потребно модул *modul.py* поставити у радни директоријум

```
>>> import modul
>>> a = range(1, 6)
>>> b = range(-1, -6, -1)
>>> modul.spajanje(a, b)
{1: -1, 2: -2, 3: -3, 4: -4, 5: -5}
>>> modul.izdvajanje(modul.spajanje(a, b))
([1, 2, 3, 4, 5], [-1, -2, -3, -4, -5])
>>> modul.zamena(modul.spajanje(a, b))
{-1: 1, -2: 2, -3: 3, -4: 4, -5: 5}
>>>
```

КОНЗОЛА

Модул

учитавање и коришћење уграђених модула

```
>>> import random
>>> import statistics
>>> random.seed(11)
>>> s = random.sample(range(1, 51), 10)
>>> s
[29, 36, 30, 50, 33, 38, 13, 12, 46, 31]
>>> statistics.quantiles(s)
[25.0, 32.0, 40.0]
>>>
```

КОНЗОЛА

Модул

учитавање и коришћење уграђених модула

```
>>> import random as rand
>>> from statistics import quantiles as qua
>>> rand.seed(11)
>>> s = rand.sample(range(1, 51), 10)
>>> s
[29, 36, 30, 50, 33, 38, 13, 12, 46, 31]
>>> qua(s)
[25.0, 32.0, 40.0]
>>>
```

КОНЗОЛА

Пакет

- један вид организације модула
  - хијерархијска организација пакета и модула
    - један пакет може обухватити више модула и пакета (потпакета)
- уобичајена имплементација пакета
  - пакет као директоријум
    - неопходно да у директоријуму постоји датотека **`__init__.py`**
      - датотека садржи програмски код који треба извршити приликом учитавања пакета
        - дозвољено је да датотека буде празна
    - може садржати датотеке које одговарају модулима
    - може садржати поддиректоријуме који одговарају пакетима

Пакет

- пакет у систему типова
  - пакет је типа **`module`**
    - у односу на обичне модуле разликује се по присуству атрибута **`__path__`**
- учитавање пакета и садржаних модула
  - помоћу исказа **`import`**
    - користи се знак **`.`** за раздвајање нивоа у хијерархијама које су састављене од пакета и модула

```
import paket.modul
import paket.potpaket.modul
from paket import modul
from paket.potpaket import modul
```

Пакет

- учитавање и коришћење корисничког модула који је у пакету
  - потребно директоријум *moduli* поставити у радни директоријум
    - потребно модул *modul.py* поставити у директоријум *moduli*
    - потребно празну датотеку *__init__.py* поставити у директоријум *moduli*

```
>>> import moduli.modul
>>> moduli.modul.zamena({})
{}
>>>
```

КОНЗОЛА

Библиотеке за језик *Python*

- расположиве библиотеке
  - бројне уграђене библиотеке за језик *Python*
  - репозиторијум *The Python Package Index (PyPI)*
    - https://pypi.org/
  - репозиторијуми на сервису *GitHub*
    - https://github.com/topics/python
  - репозиторијум *Anaconda Repository*
    - https://repo.anaconda.com/
- инсталација библиотека
  - помоћу алата *pip*
    - прикључен уз новије инсталације за језик *Python*
  - помоћу алата *conda*
    - https://conda.org/
    - укључен у дистрибуцију *Anaconda*

Садржај

Синтакса

- разликовање малих и великих слова
- називи
  - подршка за карактере из система *Unicode*
    - подржани карактери из распона *ASCII*
      - мала и велика слова
      - доња црта
      - цифре (осим као почетни знак у називу)
    - подржани карактери изван распона *ASCII*
      - *PEP 3131*: https://peps.python.org/pep-3131/
  - резервисане речи не могу бити коришћене као називи
  - називи посебног значења
    - називи с префиксом `_`
    - називи с префиксом `__`
    - називи с префиксом `__` и суфиксом `__`
- увлачење програмског кода
  - коришћењем знакова размак и табулатор
    - исказ **`if`**, исказ **`while`**, исказ **`for`**, дефиниција функције, дефиниција класе, ...

Синтакса

Резервисане речи

`False`	`None`	`True`	`and`	`as`
`assert`	`async`	`await`	`break`	`class`
`continue`	`def`	`del`	`elif`	`else`
`except`	`finally`	`for`	`from`	`global`
`if`	`import`	`in`	`is`	`lambda`
`nonlocal`	`not`	`or`	`pass`	`raise`
`return`	`try`	`while`	`with`	`yield`

Стил писања програмског кода – неке препоруке

PEP 8: https://peps.python.org/pep-0008/

- изворне датотеке да буду у складу са системом кодирања *UTF-8*
- максимална дужина реда је 79 знакова
- коришћење четири знака размака по нивоу увлачења програмског кода
- постављање исказа **`import`** на почетак изворне датотеке
  - после коментара и документационих описа а пре глобалних објеката и константи
- коментари пожељно у виду целих реченица
- називи
  - не користити називе **`l`** (мало Л), **`O`** (велико О) и **`I`** (велико И)
  - називи модула и пакета треба да буду кратки и састављени од малих слова
  - називи класа треба да буду у складу с камиљом нотацијом (енгл. c*amel case*)
    - спојене речи, свака реч почиње великим словом
  - називи функција треба да буду састављени од малих слова, при чему су речи у називу раздвојене доњом цртом
  - назив првог аргумента код методе инстанце да буде **`self`**
  - назив првог аргумента код методе класе да буде **`cls`**
  - називи константи великим словима, при чему су речи у називу раздвојене доњом цртом

Садржај

Извори и литература

Основни извори и литература

- Lubanovic B. Uvod u Python: Moderno računarstvo u jednostavnim paketima. O'Reilly (Sebastopol, CA, USA), CET (Beograd, Srbija), Računarski fakultet (Beograd, Srbija); 2015.
- Kovačević MA. Osnove programiranja u Pajtonu. Akademska misao (Beograd, Srbija); 2017.
- Pilgrim M. Dive Into Python 3. Apress (New York, NY, USA); 2009. Internet: https://diveintopython3.net/
- Python Software Foundation. Welcome to Python.org. Internet: https://www.python.org/
- Python Software Foundation. PEP 8 – Style Guide for Python Code. Internet: https://peps.python.org/pep-0008/
- Python Software Foundation. PEP 308 – Conditional Expressions. Internet: https://peps.python.org/pep-0308/

Извори и литература

Основни извори и литература

- Python Software Foundation. PEP 3131 – Supporting Non-ASCII Identifiers. Internet: https://peps.python.org/pep-3131/
- Python Software Foundation. Python 3.9.22 documentation. Internet: https://docs.python.org/3.9/

Извори и литература

Додатни извори и литература

- Downey AB. Think Python: How to Think Like a Computer Scientist. 2nd edition. Green Tea Press (Needham, MA, USA); 2015. Internet: https://greenteapress.com/wp/think-python-2e/

Основне академске студије
Информациони инжењеринг

Методе и технике науке о подацима

Основни концепти језика и окружења Python

(материјали за предавања)