Michal Kvasnica

Elektronický učebný text a materiály pre predmety Programovanie 1 a Programovanie 2

Fakulta chemickej a potravinárskej technológie
Slovenská technická univerzita v Bratislave
Bratislava
2021

Slovenská technická univerzita v Bratislave
Fakulta chemickej a potravinárskej technológie
Ústav informatizácie, automatizácie a matematiky

Elektronický učebný text a materiály pre predmety Programovanie 1 a Programovanie 2

Michal Kvasnica

Vydavateľstvo FCHPT
Bratislava, 2021
ISBN 978-80-8208-053-0

Tento učebný text je odporúčaný študentom bakalárskeho a inžinierskeho štúdia so zameraním na automatizáciu a riadenia procesov, ako i širokej verejnosti, ktorá má záujem zvládnuť základy programovania v jazyku Python.

Vydanie prvé
ISBN 978-80-8208-053-0

Obsah

Úvod

Programovací jazyk Python patrí medzi najpopulárnejšie programovacie jazyky na svete. Jeho obľúbenosť pramení z viacerých faktorov. V prvom rade ide o voľne dostupný programovací jazyk bez potreby zakúpenia komerčnej licencie. Syntax jazyka Python, teda spôsob jeho zápisu, je veľmi jednoduchá, ale pritom mocná, čo robí tento programovací jazyk ideálnym kandidátom nielen pre začínajúcich programátorov, ale i pre expertov v oblasti IT. Python obsahuje množstvo predpripravených knižníc a modulov, ktoré dovoľujú programátorovi pohodlne realizovať zložité úlohy, akými sú napríklad vedecké výpočty, tvorbu grafov, automatizáciu procesov, objektovo-orientované programovanie či programovanie webových aplikácií.

Z týchto dôvodov si získal programovací jazyk Python širokú obľubu medzi laickou i odbornou verejnosťou a v súčasnosti ho na tvorbu dvojich produktov a služieb využívajú snáď všetky úspešné korporácie, od Google, cez Microsoft, Apple, Dropbox, Amazon, Facebook, Instagram, či Netflix. Samotný tvorca programovacieho jazyky Google - holanďan Guido van Rossum - postupne pracoval pre Google, Dropbox a od roku 2020 aj pre Microsoft, pričom všetky tieto významné spoločnosti aktívne podporujú ďalší vývoj a využitie jazyka Python. Nie je bez zaujímavosti, že pre Google je Python jazykom č. 1 keď tvrdí "*Python používame tam, kde môžeme. Jazyk C++ zasa tam, kde musíme*".

Jazyk Python dovoľuje programátorovi elegantne transformovať algoritmus na riešenie konkrétnej úlohy do podoby počítačového kódu, pričom na zápis kódu často využíva formulácie známe z prirodzeného jazyka a matematiky. Tým sa výrazne odlišuje od iných programovacích jazykov (napr. C/C++ či Java), ktoré často používajú na zápis programu rôzne špeciálne znaky, ktoré môžu byť, hlavne pre začiatočníkov, pomerne mätúce. Python preferuje cestu čitateľnosti a porozumiteľnosti kódu na prvý pohľad. Naviac ide o tzv. dynamicky typovaný programovací jazyk, ktorý od programátora nevyžaduje deklarovanie premenných a určenie ich dátových typov. Tým dáva užívateľovi väčšiu flexibilitu pri tvorbe programu a ten sa môže viac sústrediť na algoritmické riešenie úlohy a nie na často otravné technické detaily. V neposlednom rade je Python tzv. interpretovaným jazykom, ktorý si nevyžaduje kompiláciu do spustiteľného kódu. Tým sa výrazne zrýchľuje vývoj programu a jeho odladovanie.

Tento učebný text ponúka základný prehľad možností programovacieho jazyka Python z pohľadu študenta bakalárskeho a inžinierskeho štúdia so zameraním na automatizáciu a informatizáciu, pričom sa od čitateľa požaduje iba základná znalosť programovania vo všeobecnosti, prípadne základné znalosti všeobecnej algoritmizácie úloh. Tento učebný text je teda určený aj úplným začiatočníkom, ktorí sa doteraz s programovacím jazykom Python nestretli vôbec, prípadne iba okrajovo.

Tento učebný text je prezentovaný formou snímkov, ktoré autor prezentoval v rámci prednášok z predmetov Programovanie 1 a Programovanie 2 na FCHPT STU v Bratislave v akedmických rokoch 2015/2016 až 2019/2020. Každá kapitola je venovaná jednej oblasti, od úplných základov spočívajúcich v používaní premenných a ich vypisovaní, cez podmienky v podobe AK/POTOM, prácu s pokročilými dátovými typmi, slučkami, metódami pracujúcimi s reťazcami, tvorbu vlastných funkcií, prácu so súbormi, využitie predpripravených modulov a knižníc, až po komplexné operácie venované vedeckým výpočtom.

Čitateľom autor želá hlavne veľa odhodlania pri prvých krokoch a spoznávaní programovacieho jazyka Python. Je totiž potrebné si uvedomiť, že hoci Python patrí podľa názoru autora medzi najjednoduhšie pochopiteľné programovacie jazyky, programovanie vo všeobecnosti môže byť občas frustrujúce. Programy totiž často obsahujú chyby, pričom nie vždy je na prvý pohľad zrejmé, v čom chyba spočíva. Preto tento učebný text na príslušných miestach zvýrazňuje načastejšie dôvody chybovosti kódu zapísaného v jazyky Python. Autor verí, že tento učebný text pomôže predovšetkým menej skúseným programátorom posunúť sa na vyššiu úroveň a spoznať aj menej známe zákutia jazyka Python.

Autor.

Programovanie v jazyku Python

Michal Kvasnica

Predpoklady

Základná znalosť programovania (Matlab, C/C++)

Trpezlivosť a odhodlanosť

- vaše programy budú obsahovať chyby
- vaše programy nebudú fungovať správne
- vaše programy budú padať
- absolútne nebudete vedieť, prečo sa to deje
- …ale aspoň sa to bude diať rýchlo

Prečo Python?

5. najpopulárnejší programovací jazyk na svete

Feb 2016	Feb 2015	Change	Programming Language	Ratings
1	2	↑	Java	21.145%
2	1	↓	C	15.594%
3	3		C++	6.907%
4	5	↑	C#	4.400%
5	8	↑	Python	4.180%
6	7	↑	PHP	2.770%
7	9	↑	Visual Basic .NET	2.454%
8	12	⇈	Perl	2.251%
9	6	↓	JavaScript	2.201%
10	11	↑	Delphi/Object Pascal	2.163%

Prečo Python?

5. najpopulárnejší programovací jazyk na svete

Zrejme 1. najpopulárnejší jazyk pre vedecké výpočty

NumPy
Base N-dimensional array package

SciPy library
Fundamental library for scientific computing

Matplotlib
Comprehensive 2D Plotting

IP[y]: IPython

IPython
Enhanced Interactive Console

Sympy
Symbolic mathematics

pandas
Data structures & analysis

Prečo Python?

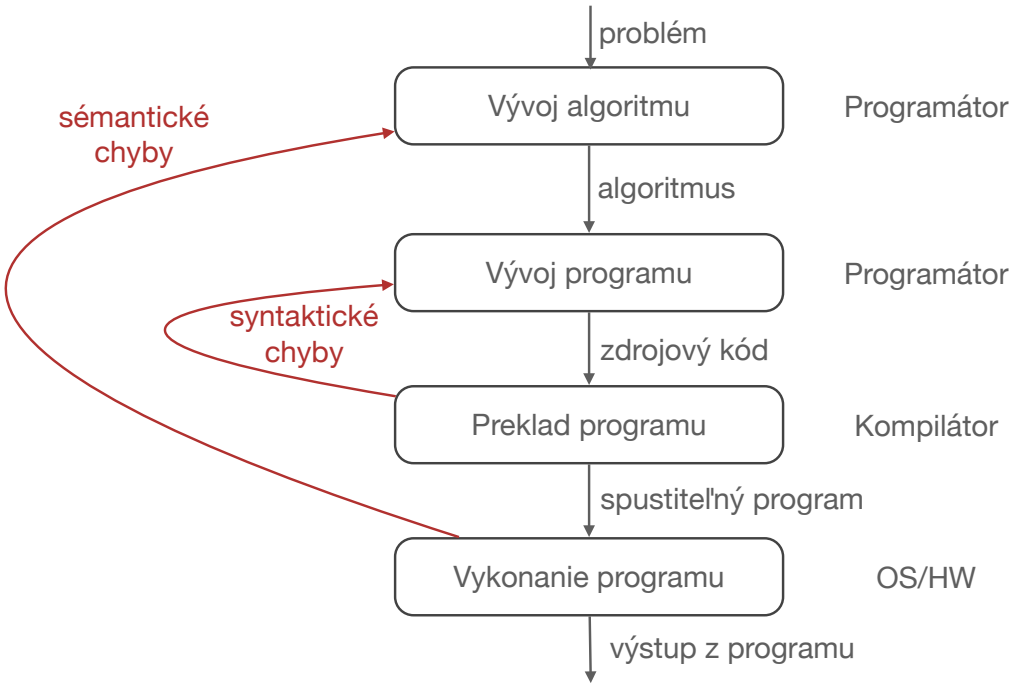
5. najpopulárnejší programovací jazyk na svete

Zrejme 1. najpopulárnejší jazyk pre vedecké výpočty

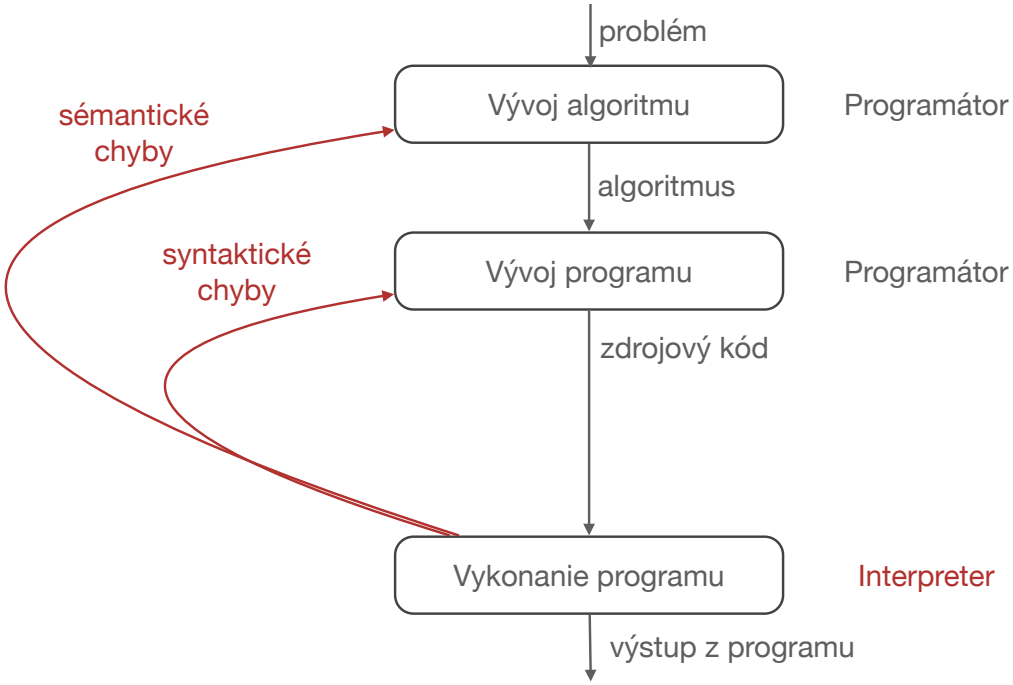
Voľne šíriteľný (=bezplatný)

Interpretovaný jazyk (=rýchlejšia a pohodlnejšia tvorba programu)

Programovanie s kompiláciou (napr. C)



Interpretovaný programovací jazyk



Prečo Python?

5. najpopulárnejší programovací jazyk na svete

Zrejme 1. najpopulárnejší jazyk pre vedecké výpočty

Voľne šíriteľný (=bezplatný)

Interpretovaný jazyk (=rýchlejšia a pohodlnejšia tvorba programu)

Dynamicky typovaný jazyk

Dokáže (takmer) všetko:

- objektovo-orientované programovanie
- práca s vektormi a maticami
- kreslenie grafov
- webové služby, databázy, ...

História jazyka Python

Vytvoril ho Guido van Rossum v r. 1989

Python 1.x (1994)

Python 2.x (2000)

- v sučasnosti najpoužívanejšia verzia
- podpora sa končí v roku 2020

Python 3.x (2008)

- Python tak, ako má byť
- čiastočne nekompatibilný s Python 2.x
- budúcnosť

BDFL
(Benevolent Dictator for Life)

Kedy použiť Python

Pre rýchly návrh a implementáciu algoritmov

- interpretovaný jazyk
- netreba vopred špecifikovať dátové typy
- veľké množstvo knižníc (netreba znovu vynájsť koleso)

Keď je na prvom mieste spoľahlivosť a rýchlosť programu je druhoradá

- typová bezpečnosť (napr. číslo + reťazec)
- kontrola indexovania polí
- zachytávanie chýb a práca s výnimkami
- ale aj Python kód môže byť rýchly (predkompilácia)

Keď potrebujeme open-source náhradu Matlabu

Inštalácia Pythonu

Budeme používať Python 3

Linux/Unix:

- zrejme už je nainštalovaný (spustiť `python3`)
- ak nie, doplnenie cez balíčkovacie systémy (`apt-get`, `emerge`, ...)

Windows/Mac:

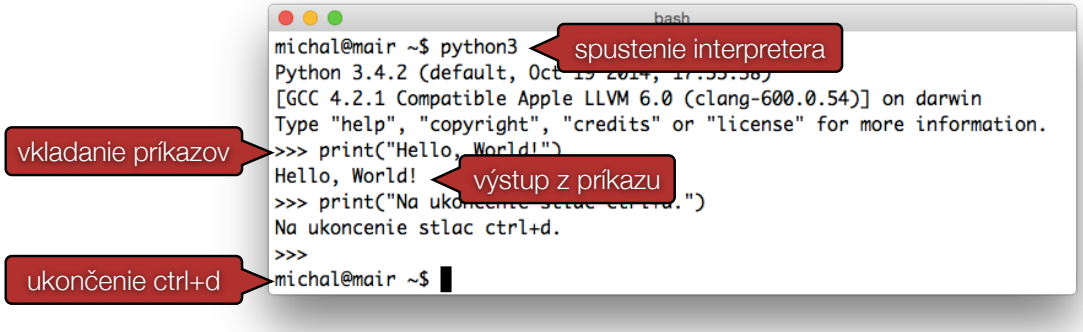
- najjednoduchšie cez Anacondu: https://www.continuum.io/downloads
- inštalovať Python 3.5

Web:

- https://jupyter.atm8.xyz
- https://ideone.com (zvoliť Python3)
- http://www.tutorialspoint.com/execute_python3_online.php

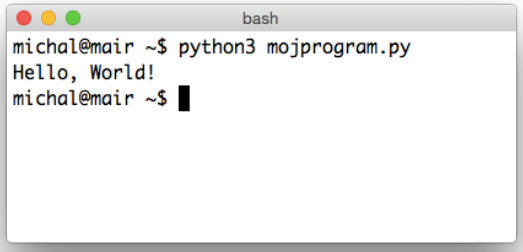
Spustenie Python programov

1. Priamo v interpreteri



Spustenie Python programov

1. Priamo v interpreteri
2. Zdrojový kód uložiť do súboru (napr. `mojprogram.py`) a potom ho spustiť cez `python3 mojprogram.py`



Prvý program

Python 3

```
print("Hello, World!")
```

Python 2

```
print "Hello, World!"
```

toto nefunguje v Python3!

Odteraz sa funkcia print() bude správať ako v Python3

```
from __future__ import print_function
print("Hello, World!")
```

Práca s premennými

Definovanie premenných (nie je potrebné dopredu špecifikovať ich dátový typ):

```
>>> cele_cislo = 1
>>> desatinne_cislo = 2.86
>>> retazec = "zhluk znakov"
>>> retazec2 = 'apostrofy'
```

Práca s premennými

Definovanie premenných (nie je potrebné dopredu špecifikovať ich dátový typ):

```
>>> cele_cislo = 1
>>> desatinne_cislo = 2.86
>>> retazec = "zhluk znakov"
>>> retazec2 = 'apostrofy'
```

Vypísanie hodnoty premennej:

Interpreter

```
>>> cele_cislo
1
>>> desatinne_cislo
2.86
>>> retazec
'zhluk znakov'
>>> retazec2
'apostrofy'
```

Program

```
print(cele_cislo)
print(desatinne_cislo)
print(retazec)
print(retazec2)
1
2.86
zhluk znakov
apostrofy
```

Dátové typy v jazyku Python

Čísla

- celé čísla (integer)
- čísla s desatinnou bodkou (float)
- Python3: delenie celých čísel dáva float
- Python2: delenie celých čísel dáva integer (zaokrúhlenie smerom nadol)

Python 3

```
>>> a = 3; b = 2;
>>> c = a/b
>>> print(c)
1.5
>>> type(c)
<class 'float'>
```

Python 2

```
>>> a = 3; b = 2;
>>> c = a/b
>>> print c
1
>>> type(c)
<type 'int'>
>>> from __future__ import division
>>> d = a/b
>>> print(d)
1.5
```

Dátové typy v jazyku Python

Čísla

- celé čísla (integer)
- čísla s desatinnou bodkou (float)
- Python3: delenie celých čísel dáva float
- Python2: delenie celých čísel dáva integer (zaokrúhlenie smerom nadol)
- prevody:
  - float na integer: `int(x)`
  - integer na float: `float(x)`
  - integer alebo float na retazec: `str(x)`
  - integer ASCII kód na znak: `chr(x)`
  - integer na hex: `hex(x)`

```
>>> int(1.56)
1
>>> float(1)
1.0
>>> str(1.56)
'1.56'
>>> str(1)
'1'
>>> chr(65)
'A'
>>> hex(123)
'0x7b'
```

Dátové typy v jazyku Python

Reťazce

- buď v úvodzovkách alebo v apostrofoch
  - `"Ahoj 'Python', ako sa mas?"`
  - `'Ahoj "Python", ako sa mas?'`
  - `'Ahoj \'Python\', ako sa mas?'`
  - `"Ahoj \"Python\", ako sa mas?"`

Dátové typy v jazyku Python

Reťazce

- buď v úvodzovkách alebo v apostrofoch:
- reťazce je možné spájať pomocou operátora "+" (plus)

```
>>> r1 = "Hello"
>>> r2 = "World"
>>> r1+r2
'HelloWorld'
>>> r1 + " " + r2 + "!"
'Hello World!'
```

Dátové typy v jazyku Python

Reťazce

- buď v úvodzovkách alebo v apostrofoch:
- reťazce je možné spájať pomocou operátora "+" (plus)
- dĺžku reťazca zistíme pomocou funkcie `len`

```
>>> len("1234567")
7
```

Dátové typy v jazyku Python

Reťazce

- buď v úvodzovkách alebo v apostrofoch:
- reťazce je možné spájať pomocou operátora "+" (plus)
- dĺžku reťazca zistíme pomocou funkcie `len`
- reťazec je pole znakov indexované od 0 po dĺžka-1

```
>>> r = "1234567"
>>> r[0]
'1'
>>> r[1]
'2'
>>> r[6]
'7'
```

Dátové typy v jazyku Python

Reťazce

- buď v úvodzovkách alebo v apostrofoch:
- reťazce je možné spájať pomocou operátora "+" (plus)
- dĺžku reťazca zistíme pomocou funkcie `len`
- reťazec je pole znakov indexované od 0 po dĺžka-1
- indexy môžu byť aj záporné

```
>>> r = "1234567"
>>> r[-1]
'7'
>>> r[-2]
'6'
>>> r[-7]
'1'
```

posledný prvok

predposledný prvok

prvý prvok

Dátové typy v jazyku Python

Reťazce

- buď v úvodzovkách alebo v apostrofoch:
- reťazce je možné spájať pomocou operátora "+" (plus)
- dĺžku reťazca zistíme pomocou funkcie `len`
- reťazec je pole znakov indexované od 0 po dĺžka-1
- indexy môžu byť aj záporné
- Python automaticky kontroluje indexovanie

posledný prvok má index 6

```
>>> r = "1234567"
>>> r[7]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> r[-8]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

Dátové typy v jazyku Python

Reťazce

- buď v úvodzovkách alebo v apostrofoch:
- reťazce je možné spájať pomocou operátora "+" (plus)
- dĺžku reťazca zistíme pomocou funkcie `len`
- reťazec je pole znakov indexované od 0 po dĺžka-1
- indexy môžu byť aj záporné
- Python automaticky kontroluje indexovanie
- časť reťazca získame pomocou rozsahu indexov
  - `r[x:y]` vráti všetky znaky od indexu "x" po index "y-1" vrátane
  - `r[:y]` vráti znaky od začiatku reťazca po index "y-1" vrátane
  - `r[x:]` vráti znaky od indexu "x" po koniec reťazca

```
r = "1234567"
r[0:2], r[0:0], r[2:0]
r[:3],  r[:7],  r[:-1]
r[2:],  r[8:],  r[-2:]
```

Dátové typy v jazyku Python

Reťazce

- buď v úvodzovkách alebo v apostrofoch:
- reťazce je možné spájať pomocou operátora "+" (plus)
- dĺžku reťazca zistíme pomocou funkcie `len`
- reťazec je pole znakov indexované od 0 po dĺžka-1
- indexy môžu byť aj záporné
- Python automaticky kontroluje indexovanie
- časť reťazca získame pomocou rozsahu indexov
- prevody:
  - na integer: `int(r)`
  - na float: `float(r)`

```
>>> float("1.23")
1.23
>>> int("5")
5
>>> int("5.23")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
with base 10: '5.23'
>>> int(float("5.23"))
5
```

Dátové typy v jazyku Python

Reťazce

- buď v úvodzovkách alebo v apostrofoch:
- reťazce je možné spájať pomocou operátora "+" (plus)
- dĺžku reťazca zistíme pomocou funkcie `len`
- reťazec je pole znakov indexované od 0 po dĺžka-1
- indexy môžu byť aj záporné
- Python automaticky kontroluje indexovanie
- časť reťazca získame pomocou rozsahu indexov
- prevody
- násobenie číslo * reťazec spôsobí opakovanie reťazca

```
>>> print(4*"meno")
menomenomenomeno
>>> print(4*"meno ")
meno meno meno meno
>>> print("'" + 4*"meno " + "'")
'meno meno meno meno '
```

Dátové typy v jazyku Python

Pokročilé dátové typy

- zoznam (list)
- slovník (dictionary)
- n-tica (touple)

O nich však až neskôr…

Pár poznámok

Komentáre sa začínajú znakom mriežky `#`

V názvoch premenných nepoužívajte diakritiku

Záleží na prázdnych znakoch! (viac neskôr…)

Funkcia `print`

Požiadavky:

- v programe zadefinuj svoj vek ako číslo a svoje meno ako reťazec
- na obrazovku ich vypíš v tvare `Ahoj XXX, mas YYY rokov.`

```
# definovanie premennych
vek = 38
meno = "Michal"  # je tiez mozne pouzit apostrofy

# vypisanie hodnot
print("Ahoj " + meno + ", mas " + vek + " rokov.")
```

číslo sa nedá spojiť s reťazcom

Funkcia `print`

Požiadavky:

- v programe zadefinuj svoj vek ako číslo a svoje meno ako reťazec
- na obrazovku ich vypíš v tvare `Ahoj XXX, mas YYY rokov.`

```
# definovanie premennych
vek = 38
meno = "Michal"  # je tiez mozne pouzit apostrofy

# vypisanie hodnot
print("Ahoj " + meno + ", mas " + str(vek) + " rokov.")
```

ok, lebo teraz spájame reťazce

Funkcia `print`

Požiadavky:

- v programe zadefinuj svoj vek ako číslo a svoje meno ako reťazec
- na obrazovku ich vypíš v tvare `Ahoj XXX, mas YYY rokov.`

```
# definovanie premennych
vek = 38
meno = "Michal"  # je tiez mozne pouzit apostrofy

# vypisanie hodnot
print("Ahoj " + meno + ", mas " + str(vek) + " rokov.")

# alternativa
print("Ahoj %s, mas %d rokov." % (meno, vek))
```

sem celé číslo

zoznam hodnôt

sem sa dosadí reťazec

oddelovač

Formátovacie znaky vo funkcii `print`

Znak	Význam
%d, %i, %u	integer
%f	desatinné číslo (vrátane núl)
%g	desatinné číslo (bez núl na konci)
%e	desatinné číslo s exponentom
%s	reťazec
%c	znak
%%	znak percento

Postupné vypisovanie

Požiadavky:

- v programe zadefinuj svoj vek ako číslo a svoje meno ako reťazec
- na obrazovku ich vypíš v tvare `Ahoj XXX, mas YYY rokov.`

```
# definovanie premennych
vek = 38
meno = "Michal"  # je tiez mozne pouzit apostrofy

# postupne vypisovanie
print("Ahoj ")
print(meno)
print(", mas ")
print(vek)
print(" rokov.")
```

print automaticky pridá prázdny riadok

```
Ahoj
Michal
, mas
38
 rokov.
```

Postupné vypisovanie

Požiadavky:

- v programe zadefinuj svoj vek ako číslo a svoje meno ako reťazec
- na obrazovku ich vypíš v tvare `Ahoj XXX, mas YYY rokov.`

```
# definovanie premennych
vek = 38
meno = "Michal"  # je tiez mozne pouzit apostrofy

# postupne vypisovanie
print("Ahoj ", end="")
print(meno, end="")
print(", mas ", end="")
print(vek, end="")
print(" rokov.")
```

tým povieme, že riadok bude pokračovať ďalej

print bez end="" automaticky odriadkuje

Postupné vypisovanie v Python2

Požiadavky:

- v programe zadefinuj svoj vek ako číslo a svoje meno ako reťazec
- na obrazovku ich vypíš v tvare `Ahoj XXX, mas YYY rokov.`

```
from __future__ import print_function

# definovanie premennych
vek = 38
meno = "Michal"  # je tiez mozne pouzit apostrofy

# postupne vypisovanie
print("Ahoj ", end="")
print(meno, end="")
print(", mas ", end="")
print(vek, end="")
print(" rokov.")
```

Odteraz sa funkcia print() bude správať ako v Python3

toto nefunguje v Python2

Funkcia `print`

Požiadavky:

- v programe zadefinuj svoj vek ako číslo a svoje meno ako reťazec
- na obrazovku ich vypíš v tvare `Ahoj XXX, mas YYY rokov.`

```
# definovanie premennych
vek = 38
meno = "Michal"  # je tiez mozne pouzit apostrofy

# vypisanie hodnot
print("Ahoj", meno, "mas", vek, "rokov.")

Ahoj Michal mas 38 rokov.
```

po každom údaji sa pridá medzera

Funkcia `print`

Požiadavky:

- v programe zadefinuj svoj vek ako číslo a svoje meno ako reťazec
- na obrazovku ich vypíš v tvare `Ahoj XXX, mas YYY rokov.`

```
# definovanie premennych
vek = 38
meno = "Michal"  # je tiez mozne pouzit apostrofy

# vypisanie hodnot
print("Ahoj", meno, ", mas", vek, "rokov.")

Ahoj Michal , mas 38 rokov.
```

po každom údaji sa pridá medzera

Funkcia `print`

Požiadavky:

- v programe zadefinuj svoj vek ako číslo a svoje meno ako reťazec
- na obrazovku ich vypíš v tvare `Ahoj XXX, mas YYY rokov.`

```
# definovanie premennych
vek = 38
meno = "Michal"  # je tiez mozne pouzit apostrofy

# chybne:
print("Ahoj %s, mas %d rokov.", meno, vek)

Ahoj %s, mas %d rokov. Michal 38

# spravne:
print("Ahoj %s, mas %d rokov." % (meno, vek))

Ahoj Michal mas 38 rokov.
```

po každom údaji sa pridá medzera

Formátovaný výstup do reťazca

```
# definovanie premennych
vek = 38
meno = "Michal"  # je tiez mozne pouzit apostrofy

# programové vytvorenie reťazca:
vystup = "Ahoj %s, mas %d rokov." % (meno, vek)
print(vystup)

# alternatíva (automatické určenie formátu):
vystup = "Ahoj {}, mas {} rokov.".format(meno, vek)
print(vystup)
```

automatické formátovanie

dosadenie hodnôt

Vstup z klávesnice

Python 3:

```
meno = input("Vloz meno: ")
```

Python 2:

```
meno = raw_input("Vloz meno: ")
```

Vloženie čísla:

```
>>> udaj = input("Vloz cislo: ")
Vloz cislo: 123.45
>>> type(udaj)
<class 'str'>
>>> cislo = float(udaj)
>>> type(cislo)
<class 'float'>
>>> print(cislo)
123.45
```

alebo int(udaj)

Prvý program

Požiadavky:

- z klávesnice načítajte svoj vek a meno
- na obrazovku ich vypíš v tvare `Ahoj XXX, mas YYY rokov.`

```
# nacitanie z klavesnice
vek = int(input("Vloz vek: "))
meno = input("Vloz meno: ")

# vypis na obrazovku
print("Ahoj %s, mas %d rokov." % (meno, vek))
```

konverzia na integer

Dátový typ bool (logická premenná)

Označuje pravdu / nepravdu

- pravda: `True`
- nepravda: `False`
- case sensitive!

```
>>> plnolety = True
>>> print(plnolety)
True
>>> plnolety = False
>>> print(plnolety)
False
```

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

Dátový typ bool (logická premenná)

Vypisovanie:

```
>>> pravda = True
>>> nepravda = False
>>> print('Vysledok: %s' % (pravda))
Vysledok: True
>>> print('Vysledok: %s' % (nepravda))
Vysledok: False
```

```
>>> print('Vysledok: %d' % (pravda))
Vysledok: 1
>>> print('Vysledok: %f' % (nepravda))
Vysledok: 0.000000
```

```
>>> print('Vysledok: {}'.format(pravda))
Vysledok: True
>>> print('Vysledok: {}'.format(pravda))
Vysledok: False
```

Logické operácie a výrazy

Logické operácie:

- `x` **`and`** `y`
- `x` **`or`** `y`
- **`not`** `x`

```
moze_volit = plnolety and svojpravny
```

```
student = neplnolety or ma_ISIC
```

```
neplnolety = not plnolety
```

Logické operácie a výrazy

Logické operácie je možné ľubovolne kombinovať:

- poradie: `not`, `and`, `or`
- `and` a `or` majú skratové (short-circuit) správanie
- precvičenie:

```
x = T or F
x = F or T
x = T and F or T
x = T and F and F
x = T or not T
x = F or not F
x = F or not T
x = T or F and T
x = F or not T and T
x = T and F or T
x = not T and F
x = not (T and F)
x = (T or F) and not F
x = (T or F) and (F or T or not F) or not F
```

Logické operácie a výrazy

Porovnávacie operátory vrátia `bool`:

- striktné porovnanie: `a > b, a < b`
- nestriktné porovnanie: `a >= b, a <= b`
- rovnosť, nerovnosť: `a == b, a != b` — porovnáva hodnoty
- identita: `a is b, a is not b` — porovnáva hodnoty aj dátové typy

```
plnolety = (vek >= 18)
plnolety = (vek > 17)
```

```
>>> print(1 == 2)
False
>>> print(1 == 1.0)
True
>>> print(1 == True)
True
>>> print(2 != True)
True
>>> print(0 == False)
True
```

```
>>> print(3.5 is 3.5)
True
>>> print(3.5 is 2.6)
False
>>> print(3.5 is not 2.6)
True
>>> print(1 is 1.0)
False
>>> print(1 is True)
False
>>> print(1 is not True)
True
```

Logické operácie a výrazy

Kombinácie:

```
moze_volit = (vek>=18) and svojpravny
znamkaB = (percenta>=83) and (percenta<=92)
uspel = (percenta>55) or (opakuje and uspel_minule)
maly_integer = type(hodnota)==int and (hodnota<=1000)
velky_float = type(hodnota)==float and (hodnota>1000)
```

```
s = 'velmi dlhy retazec'
dlhy_string = len(s)>10 and type(s)==str
```

```
s = 1
dlhy_string = len(s)>10 and type(s)==str
TypeError: object of type 'int' has no len()
```

OK!

```
s = 'velmi dlhy retazec'
dlhy_string = type(s)==str and len(s)>10
```

Porovnávanie reťazcov

```
>>> "retazec" == "retazec"
True
```

```
>>> "retazec" == "retazec "
False
```

```
>>> "retazec" != "Retazec"
True
```

case sensitive

```
>>> "1" == 1
False
```

```
>>> x is y
```

bezpečné, porovnáva aj typ

Vyhľadávanie v reťazcoch

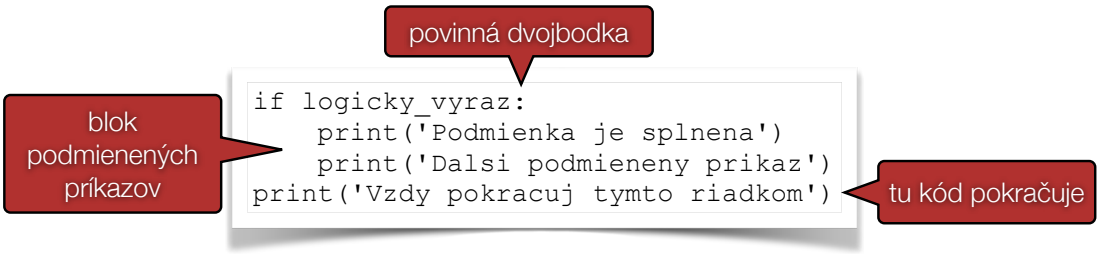
- vyhľadanie podreťazca v reťazci: `poderatazec` **`in`** `retazec`
- negácia vyhľadávania: `poderatazec` **`not in`** `retazec`
- vyhľadanie na začiatku reťazca: `retazec.`**`startswith`**`(podretazec)`
- vyhľadanie na konci reťazca: `retazec.`**`endswith`**`(podretazec)`

```
>>> "aze" in "retazec"
True
>>> "xyz" not in "retazec"
True
```

```
>>> "retazec".startswith("ret")
True
>>> "retazec".startswith("Ret")
False
```

```
>>> "retazec".endswith("ec")
True
```

IF-THEN podmienky v jazyku Python



Poznámky:

- dvojbodka za podmienkou znamená `then`
- podmienené príkazy je potrebné odsadiť medzerami (najčastejšie 4)
- IF-THEN blok sa končí keď "zmizne" odsadenie

IF-THEN podmienky v jazyku Python

```
vek = int(input('Vloz svoj vek: '))
if (vek>=18):
    print('Si plnolety')
```

```
vek = int(input('Vloz svoj vek: '))
svojpravny = bool(input('Si svojpravny? '))
if (vek>=18) and svojpravny:
    print('Mas volebne pravo')
```

berie iba True alebo False

```
vek = int(input('Vloz svoj vek: '))
svojpravny = bool(input('Si svojpravny? '))
moze_volit = (vek>=18) and svojpravny
if moze_volit:
    print('Mas volebne pravo')
```

IF-THEN podmienky v jazyku Python

```
vek = int(input('Vloz svoj vek: '))
s = input('Si svojpravny? ')
svojpravny = (s=='a') or (s=='A') or (s=='ano') or (s=='ANO')
moze_volit = (vek>=18) and svojpravny
if moze_volit:
    print('Mas volebne pravo')
```

```
vek = int(input('Vloz svoj vek: '))
s = input('Si svojpravny? ')
svojpravny = s.startswith('a') or s.startswith('A')
moze_volit = (vek>=18) and svojpravny
if moze_volit:
    print('Mas volebne pravo')
```

IF-THEN-ELSE podmienky v jazyku Python

povinná dvojbodka

```
if logicky_vyraz:
    print('Podmienka je splnena')
    print('Dalsi podmieneny prikaz')
else:
    print('Podmienka nie je splnena')
print('Vzdy pokracuj tymto riadkom')
```

blok sa vykoná ak nie je podmienka splnená

Poznámky:

- dvojbodka za podmienkou v `if` aj za `else`
- za `else` sa uz nepíše podmienka
- podmienené príkazy je potrebné odsadiť medzerami (najčastejšie 4)
- IF-THEN-ELSE blok sa končí keď "zmizne" odsadenie

IF-THEN-ELSE podmienky v jazyku Python

```
vek = int(input('Vloz svoj vek: '))
if (vek>=18):
    print('Si plnolety')
else:
    print('Nie si plnolety')
print('Koniec programu')
```

```
if (vek>=18):
    if svojpravny:
        print('Mas volebne pravo')
    else:
        print('Nemas volebne pravo')
else:
    print('Nie si plnolety')
print('Koniec programu')
```

Príklad

- načítajte z klávesnice vašu výšku ako float
- ak je väčšia ako 3, číslo považujte za centimetre, v opačnom prípade za metre
- na obrazovku vypíšte vašu výšku v príslušných jednotkách
  - centimetre vypíšte ako float bez desatinných miest
  - metre ako float s 2 desatinnými miestami
- príklad:
  - vstup: `177`, výstup: `Vyska je 177 cm`
  - vstup: `1.77`, výstup: `Vyska je 1.77 m`

```
vyska = float(input('Vloz svoju vysku: '))
if (vyska>3):
    print('Vyska je %.0f cm' % (vyska))
else:
    print('Vyska je %.2f m' % (vyska))
```

Príklad

- načítajte z klávesnice vašu výšku ako float
- ak je väčšia ako 3, číslo považujte za centimetre, v opačnom prípade za metre
- na obrazovku vypíšte vašu výšku v **centimetroch**
- príklad:
  - vstup: `177`, výstup: `Vyska je 177 cm`
  - vstup: `1.77`, výstup: `Vyska je 177 cm`

```
vyska = float(input('Vloz svoju vysku: '))
if (vyska>3):
    print('Vyska je %.0f cm' % (vyska))
else:
    print('Vyska je %.0f cm' % (vyska*100))
```

v princípe rovnaký príkaz

Príklad (lepšie riešenie)

- načítajte z klávesnice vašu výšku ako float
- ak je väčšia ako 3, číslo považujte za centimetre, v opačnom prípade za metre
- na obrazovku vypíšte vašu výšku v **centimetroch**
- príklad:
  - vstup: `177`, výstup: `Vyska je 177 cm`
  - vstup: `1.77`, výstup: `Vyska je 177 cm`

```
vyska = float(input('Vloz svoju vysku: '))
if (vyska<3):
    vyska = vyska*100  # odteraz to bude v cm
print('Vyska je %.0f cm' % (vyska))
```

Príklad

- načítajte z klávesnice vašu výšku ako float a jednotky ako string
- na obrazovku vypíšte vašu výšku v **centimetroch**
- ošetrite neplatné jednotky
- príklad:
  - vstup: `177 cm`, výstup: `Vyska je 177 cm`
  - vstup: `1.77 m`, výstup: `Vyska je 177 cm`
  - vstup: `1.77 km`, výstup: `Neplatna jednotka "km"`

```
vyska = float(input('Vloz svoju vysku: '))
jednotky = input('Jednotky: ')
if jednotky=='m':
    vyska = vyska*100  # odteraz to bude v cm
if jednotky=='m' or jednotky=='cm':
    print('Vyska je %d cm' % (vyska))
else:
    print('Neplatna jednotka "%s"' % (jednotky))
```

aj float sa dá vypísať ako integer

IF-THEN-ELIF-ELSE podmienky v jazyku Python

```
if prva_podmienka:
    print('Prva pdmienka je splnena')
elif druha_podmienka:
    print('Druha podmienka je splnena')
elif tretia_podmienka:
    print('Tretia podmienka je splnena')
else:
    print('Ziadna podmienka nie je splnena')
print('Vzdy pokracuj tymto riadkom')
```

Poznámky:

- dvojbodka za podmienkou v `if, elif` aj za `else`
- za `elif` sa píšu podmienky, za `else` nie
- `elif`-ov môže byť ľubovolný počet
- podmienené príkazy je potrebné odsadiť medzerami (najčastejšie 4)
- IF-THEN-ELIF-ELSE blok sa končí keď "zmizne" odsadenie

IF-THEN-ELIF-ELSE podmienky v jazyku Python

```
if (vek>=18) and svojpravny:
   print('Mas volebne pravo')
elif not svojpravny:
   print('Nie si svojpravny')
elif (vek>=18):
   print('Si plnolety')
else:
    print('Nie si plnolety')
print('Koniec programu')
```

Príklad

- načítajte z klávesnice dve čísla a znak (oddelené medzerami)
- ak znak je '+', na obrazovku vypíšte súčet načítaných čísel
- ak znak je '-', na obrazovku vypíšte rozdiel načítaných čísel
- ak znak je '*', na obrazovku vypíšte súčin načítaných čísel
- ak znak je '/', na obrazovku vypíšte podiel načítaných čísel
- v opačnom prípade sa vypíše, že znak nie je podporovaný

IF-THEN-ELIF-ELSE podmienky v jazyku Python

```
prve = float(input('Prve cislo: '))
druhe = float(input('Druhe cislo: '))
znak = input('Operacia: ')
if znak=='+':
    print('%g%s%g=%g' % (prve, znak, druhe, prve+druhe))
elif znak=='-':
    print('%g%s%g=%g' % (prve, znak, druhe, prve-druhe))
elif znak=='*':
    print('%g%s%g=%g' % (prve, znak, druhe, prve*druhe))
elif znak=='/':
    print('%g%s%g=%g' % (prve, znak, druhe, prve/druhe))
else:
    print('Nepodporovana operacia "%s"' % (znak))
```

IF-THEN-ELIF-ELSE podmienky v jazyku Python

```
prve = float(input('Prve cislo: '))
druhe = float(input('Druhe cislo: '))
znak = input('Operacia: ')

if znak=='+':
    vysledok = prve + druhe
elif znak=='-':
    vysledok = prve - druhe
elif znak=='*':
    vysledok = prve*druhe;
elif znak=='/':
    vysledok = prve/druhe;
```

IF-THEN-ELIF-ELSE podmienky v jazyku Python

```
prve = float(input('Prve cislo: '))
druhe = float(input('Druhe cislo: '))
znak = input('Operacia: ')

if znak=='+':
    vysledok = prve + druhe
elif znak=='-':
    vysledok = prve - druhe
elif znak=='*':
    vysledok = prve*druhe;
elif znak=='/':
    vysledok = prve/druhe;
if v premennej vysledok je nieco ulozene:
    print('%g%s%g=%g' % (prve, znak, druhe, vysledok))
else:
    print('Nepodporovana operacia "%s"' % (znak))
```

IF-THEN-ELIF-ELSE podmienky v jazyku Python

```
prve = float(input('Prve cislo: '))
druhe = float(input('Prve cislo: '))
znak = input('Operacia: ')
vysledok = None
if znak=='+':
    vysledok = prve + druhe
elif znak=='-':
    vysledok = prve - druhe
elif znak=='*':
    vysledok = prve*druhe;
elif znak=='/':
    vysledok = prve/druhe;
if vysledok is not None:
    print('%g%s%g=%g' % (prve, znak, druhe, prve+druhe))
else:
    print('Nepodporovana operacia "%s"' % (znak))
```

špeciálny dátový typ znamenajúci "nič"

máme niečo uložené v premennej?

Programovanie v jazyku Python

Michal Kvasnica

Dátový typ `list` (zoznam)

Reprezentuje jednorozmerové pole ľubovolných dát

```
zoznam = [ 1, 2, 12, 4, 0 ]
```

Heterogénny dátový typ (môžeme kombinovať dáta):

```
zoznam = [ 1, "retazec", 2.0, [0, 5] ]
```

Zoznam je možné indexovať podobne ako reťazce:

```
zoznam[0]
zoznam[1:3]
zoznam[:2]
zoznam[2:]
zoznam[-1]
zoznam[-3:]
```

Dátový typ `list` (zoznam)

Dĺžku zoznamu zistíme funkciou `len`:

```
dlzka = len(zoznam)
```

Prvky zoznamu môžeme meniť:

```
zoznam = [ 1, 2, 12, 4, 0 ]
zoznam[0] = 5
```

Pri výmene sa môže zmeniť aj dátový typ:

```
zoznam = [ 1, "retazec", 2.0, [0, 5] ]
zoznam[1] = 5.6
```

Dátový typ `list` (zoznam)

Metóda `append` vloží jeden prvok na koniec zoznamu:

```
>>> zoznam = []
>>> zoznam.append(1); print(zoznam)
[1]
```

```
>>> zoznam.append(12); print(zoznam)
[1, 12]
```

```
>>> zoznam.append([4.0, 5]); print(zoznam)
[1, 12, [4.0, 5]]
```

```
>>> zoznam.append("abc"); print(zoznam)
[1, 12, [4.0, 5], 'abc']
```

Dátový typ `list` (zoznam)

Metóda `extend` pridá všetky prvky zoznamu na koniec:

```
>>> zoznam = []; print(zoznam)
[]
```

nie je zoznam!

```
>>> zoznam.extend(1)
TypeError: 'int' object is not iterable
```

ok, je zoznam

```
>>> zoznam.extend([1]); print(zoznam)
[1]
```

```
>>> zoznam.extend([2, 3]); print(zoznam)
[1, 2, 3]
```

```
>>> zoznam.extend("abc"); print(zoznam)
[1, 2, 3, 'a', 'b', 'c']
```

Dátový typ `list` (zoznam)

Zoznamy sa dajú spájať aj pomocou `'+'`:

```
>>> z1 = [1, 2, 3]; z2 = [4.0, "abc"]
>>> vysledok = z1 + z2; print(vysledok)
[1, 2, 3, 4.0, 'abc']
```

Pozor: oba spájané objekty musia byť zoznamy:

nie je zoznam!

```
>>> [1, 2, 3] + 4
TypeError: can only concatenate list (not "int") to list
```

ok, je zoznam

```
>>> [1, 2, 3] + [4]
[1, 2, 3, 4]
```

Dátový typ `list` (zoznam)

Zoznamy sa dajú násobiť celým číslom:

```
>>> 3*[1, 2, 3]
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
>>> 0*[1, 2, 3]
[]
```

```
>>> -1*[1, 2, 3]
[]
```

Dátový typ `list` (zoznam)

Metóda `insert(i,x)` vloží prvok `x` pred pozíciu `i`

```
>>> zoznam = [1, 2, 3, 4]
>>> zoznam.insert(3, 5.0); print(zoznam)
[1, 2, 3, 5.0, 4]
```

vlož na začiatok

```
>>> zoznam.insert(0, 0); print(zoznam)
[0, 1, 2, 3, 5.0, 4]
```

vlož na koniec

```
>>> zoznam.insert(len(zoznam), -1); print(zoznam)
[0, 1, 2, 3, 5.0, 4, -1]
```

vlož na predposledné miesto

```
>>> zoznam.insert(-1, 9); print(zoznam)
[0, 1, 2, 3, 5.0, 4, 9, -1]
```

Dátový typ `list` (zoznam)

Metóda `pop()` odoberie a vráti jeden prvok z konca zoznamu:

```
>>> zoznam = [1, 2, 3, 4]
>>> prvok = zoznam.pop()
>>> print(prvok)
4
>>> print(zoznam)
[1, 2, 3]
```

zmenil sa aj zoznam

```
>>> dalsi = zoznam.pop()
>>> print(dalsi)
3
>>> print(zoznam)
[1, 2]
```

```
>>> zoznam = []
>>> prvok = zoznam.pop()
IndexError: pop from empty list
```

nedá sa popovať z prázdneho zoznamu

Dátový typ `list` (zoznam)

Metóda `pop(i)` odoberie a vráti jeden prvok z danej pozície:

```
>>> zoznam = [1, 2, 3, 4]
>>> prvok = zoznam.pop(2)
>>> print(prvok)
3
>>> print(zoznam)
[1, 2, 4]
```

odober prvý prvok

```
>>> prvok = zoznam.pop(0)
>>> print(prvok)
1
>>> print(zoznam)
[2, 4]
```

odober posledný prvok

```
>>> prvok = zoznam.pop(-1)
>>> print(prvok)
4
>>> print(zoznam)
[2]
```

Dátový typ `list` (zoznam)

Metóda `index(x)` vráti index prvého výskytu prvku `x`:

```
>>> zoznam = [3, 1, 2, 1, 4]
>>> x = 1
>>> pozicia = zoznam.index(x)
>>> print(pozicia)
1
>>> print(zoznam[pozicia])
1
```

```
>>> zoznam = [3, 1, 2, 1, 4]
>>> pozicia = zoznam.index(0)
ValueError: 0 is not in list
```

prvok nie je v zozname

Dátový typ `list` (zoznam)

Metóda `count(x)` vráti počet výskytov prvku `x`:

```
>>> zoznam = [3, 1, 2, 1, 4]
>>> x = 1
>>> pocet = zoznam.count(x)
>>> print(pocet)
2
```

```
>>> pocet = zoznam.count(0)
>>> print(pocet)
0
```

```
>>> pocet = "abcdacd".count("a")
>>> print(pocet)
2
```

počet výskytov znaku v reťazci

```
>>> pocet = "abcdacd".count("ab")
>>> print(pocet)
1
```

počet výskytov podreťazca v reťazci

Dátový typ `list` (zoznam)

Metóda `sort()` usporiada pole od najmenšieho prvku po najväčší (priamo modifikuje zoznam):

```
>>> zoznam = [3, 1, 2, 1, 4]
>>> zoznam.sort()
>>> print(zoznam)
[1, 1, 2, 3, 4]
```

priamo mení zoznam

Metóda `sort(reverse=True)` usporiada pole od najväčšieho prvku po najmenší (priamo modifikuje zoznam):

```
>>> zoznam = [3, 1, 2, 1, 4]
>>> zoznam.sort(reverse=True)
>>> print(zoznam)
[4, 3, 2, 1, 1]
```

Dátový typ `list` (zoznam)

Funkcia `sorted()` vráti nový zoznam usporiadaný od najmenšieho prvku po najväčší

```
>>> zoznam = [3, 1, 2, 1, 4]
>>> novy = sorted(zoznam)
>>> print(novy)
[1, 1, 2, 3, 4]
>>> print(zoznam)
[3, 1, 2, 1, 4]
```

nemení pôvodný zoznam

```
>>> zoznam = [3, 1, 2, 1, 4]
>>> novy = sorted(zoznam, reverse=True)
>>> print(novy)
[4, 3, 2, 1, 1]
```

od najväčšieho po najmenší

Dátový typ `list` (zoznam)

Metóda `reverse()` prehodí poradie prvkov:

```
>>> zoznam = [3, 0, 2, 1, 4]
>>> zoznam.reverse()
>>> print(zoznam)
[4, 1, 2, 0, 3]
```

```
>>> zoznam = [3, 'abc', 2, [1, 5.0], 4]
>>> zoznam.reverse()
>>> print(zoznam)
[4, [1, 5.0], 2, 'abc', 3]
```

prehadzuje sa iba poradie hlavného zoznamu

Operácie so zoznamami čísel:

Funkcia `min(zoznam)` vráti najmenší prvok zoznamu:

```
>>> zoznam = [3, 0, 2, 1, 4]
>>> x = min(zoznam)
>>> print(x)
0
```

Funkcia `max(zoznam)` vráti najväčší prvok zoznamu:

```
>>> x = max(zoznam)
>>> print(x)
4
```

Funkcia `sum(zoznam)` vráti sumu prvkov zoznamu:

```
>>> s = sum(zoznam)
>>> print(x)
10
```

Operácie so zoznamami čísel:

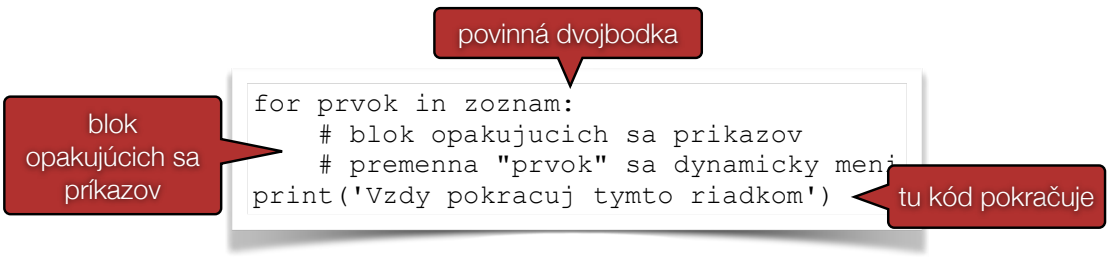
`min()` a `max()` pre zoznamy reťazcov:

```
>>> zoznam = ['z', 'aa', 'abc', 'abeceda', 'xyz']
>>> q = min(zoznam)
>>> print(q)
aa
>>> q = max(zoznam)
>>> print(q)
z
```

lingvisticky najmenší prvok

lingvisticky najväčší prvok

FOR slučky v jazyku Python



Poznámky:

- príkaz `for` postupne prechádza cez prvky zoznamu
- blok opakujúcich sa príkazov je potrebné odsadiť
- v každom priebehu for-slučky sa mení hodnota iteračnej premennej
- pozor: premenná prvok je hodnotou prvku, nie jeho indexom!

FOR slučky v jazyku Python

```
for prvok in [0, 1, 2, 3]:
    print(prvok)
```

```
0
1
2
3
```

FOR slučky v jazyku Python

```
for prvok in [0, [1, 2], "abc"]:
    print(prvok)
```

```
0
[1, 2]
abc
```

FOR slučky v jazyku Python

pole 3 znakov

```
for prvok in "abc":
    print(prvok)
```

```
a
b
c
```

pole s 1 prvkom

```
for prvok in ["abc"]:
    print(prvok)
```

```
abc
```

FOR slučky v jazyku Python

```
ludia = ['Jan', 'Marta', 'Zuzana']
for index in [0, 1, 2]:
    print("Ahoj {}".format(ludia[index]))
```

```
Ahoj Jan
Ahoj Marta
Ahoj Zuzana
```

```
ludia = ['Jan', 'Marta', 'Zuzana']
for clovek in ludia:
    print("Ahoj {}".format(clovek))
```

FOR slučky v jazyku Python

Funkcia `range(x)` vráti zoznam od `0` po `x-1`:

```
for i in range(4):
    print(i)
```

```
0
1
2
3
```

FOR slučky v jazyku Python

Funkcia `range(a, b)` vráti zoznam od `a` po `b-1`:

```
for i in range(1, 5):
    print(i)
```

```
1
2
3
4
```

FOR slučky v jazyku Python

Funkcia `range(a, b, c)` vráti zoznam od `a` po `b-1` s krokom `c`

```
for i in range(2, 7, 2):
    print(i)
```

```
2
4
6
```

Príklad

Vytvorte program, ktorý:

- načíta z klávesnice celé číslo
- na obrazovku vypíše do riadku príslušný počet hviezdičiek

```
opakovani = int(input("Pocet opakovani: "))
for i in range(opakovani):
    print("*", end="")
print()  # odriadkovanie
```

Čisté python riešenie:

```
opakovani = int(input("Pocet opakovani: "))
print("*"*opakovani)
```

Príklad

Vytvorte program, ktorý:

- načíta z klávesnice reťazec
- zistí počet výskytov znaku medzera

```
retazec = input("Vloz retazec: ")
medzery = 0
for i in range(len(retazec)):
    if retazec[i]==" ":
        medzery = medzery + 1
print("Pocet medzier: %d" % (medzery))
```

Lepšie python riešenie:

```
retazec = input("Vloz retazec: ")
medzery = 0
for znak in retazec:
    if znak==" ":
        medzery = medzery + 1
print("Pocet medzier: %d" % (medzery))
```

Príklad

Vytvorte program, ktorý:

- načíta z klávesnice reťazec
- zistí počet výskytov znaku medzera

```
retazec = input("Vloz retazec: ")
medzery = 0
for i in range(len(retazec)):
    if retazec[i]==" ":
        medzery = medzery + 1
print("Pocet medzier: %d" % (medzery))
```

Čisté python riešenie:

```
retazec = input("Vloz retazec: ")
medzery = retazec.count(" ")
print("Pocet medzier: %d" % (medzery))
```

Príklad

Vytvorte program, ktorý:

- načíta z klávesnice počet čísel
- z klávesnice načíta požadovaný počet čísel do poľa
- na obrazovku vypíše pole a priemer jeho prvkov

```
pocet = int(input("Pocet cisel: "))
pole = []
for i in range(pocet):
    prvok = float(input("%d. cislo: " % (i+1)))
    pole.append(prvok)
print("Pole: %s" % (pole))
print("Priemer: %f" % (sum(pole)/len(pole)))
```

FOR slučky v jazyku Python

Funkcia `enumerate(zoznam)` vráti dvojice `index,hodnota`

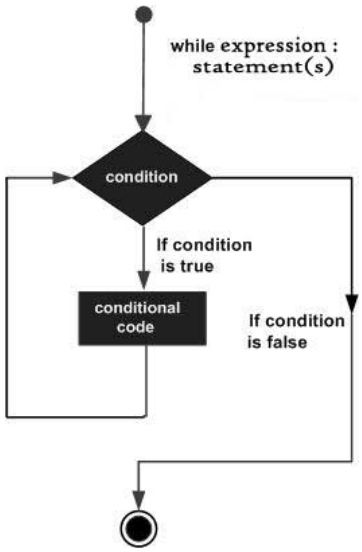
```
zoznam = [1, 5, "abc", [2, 3]]
for index, hodnota in enumerate(zoznam):
    print("index: {}, hodnota: {}".format(index, hodnota))
```

```
index: 0, hodnota: 1
index: 1, hodnota: 5
index: 2, hodnota: abc
index: 3, hodnota: [2, 3]
```

WHILE slučky v jazyku Python

povinná dvojbodka

```
while (podmienka je splnena):
    # blok opakujucich sa prikazov
print('Vzdy pokracuj tymto riadkom')
```



WHILE slučky v jazyku Python

```
pocet = 0
while (pocet < 4):
    print("Pocet: {}".format(pocet))
    pocet = pocet + 1
print("Dovidenia!")
```

```
Pocet: 0
Pocet: 1
Pocet: 2
Pocet: 3
Dovidenia!
```

Príklad

Vytvorte program, ktorý:

- načítava z klávesnice do poľa čísla
- po každom čísle sa spýta, či chceme načítať ďalšie
- na záver vypíše celé pole

```
pole = []
dalsie = "a"
while (dalsie=="a"):
    cislo = float(input("Cislo: "))
    pole.append(cislo)
    dalsie = input("Dalsie? [a/n]: ")
print("Pole: {}".format(pole))
```

Príklad

Vytvorte program, ktorý:

- načítava z klávesnice do poľa čísla
- po každom čísle sa spýta, či chceme načítať ďalšie
- na záver vypíše celé pole

predčasné ukončenie cyklu

```
pole = []
while True:
    cislo = float(input("Cislo: "))
    pole.append(cislo)
    dalsie = input("Dalsie? [a/n]: ")
    if dalsie=="n":
        break
print("Pole: {}".format(pole))
```

Výroba zoznamov

Majme zoznam čísel

Vyrobme nový zoznam obsahujúci druhé mocniny prvkov

```
zoznam = [ 1, 2, 3, 4, 5 ]
```

```
novy = []
for prvok in zoznam:
    novy.append(prvok**2)
```

```
novy = [ prvok**2 for prvok in zoznam ]
```

iterátor

čo spraviť s každým prvkom

nový zoznam

Výroba zoznamov

Majme zoznam čísel

Vyrobme nový zoznam obsahujúci kópiu pôvodného zoznamu

```
zoznam = [ 1, 2, 3, 4, 5 ]
```

```
novy = []
for prvok in zoznam:
    novy.append(prvok)
```

```
novy = [ prvok for prvok in zoznam ]
```

správny spôsob kópie zoznamu

```
novy = zoznam[:]
```

Výroba zoznamov

Majme zoznam čísel

Vyrobme nový zoznam obsahujúci prvky zväčšené o 1
v opačnom poradí

```
zoznam = [ 1, 2, 3, 4, 5 ]
```

```
novy = []
for prvok in zoznam.reverse():
    novy.append(prvok+1)
```

nebude fungovať, lebo metóda reverse() nič nevracia

```
novy = []
zoznam.reverse()
for prvok in zoznam:
    novy.append(prvok+1)
```

ok, ale stratíme pôvodné poradie zoznamu

Výroba zoznamov

Majme zoznam čísel

Vyrobme nový zoznam obsahujúci prvky zväčšené o 1
v opačnom poradí

```
zoznam = [ 1, 2, 3, 4, 5 ]
```

vráti nový zoznam v opačnom poradí

```
novy = []
for prvok in reversed(zoznam):
    novy.append(prvok+1)
```

```
novy = [ prvok+1 for prvok in reversed(zoznam) ]
```

Výroba zoznamov

Majme zoznam reťazcov

Vyrobme nový zoznam obsahujúci dĺžky jednotlivých reťazcov

```
ludia = [ 'Jan', 'Jozef', 'Zuzana' ]
```

```
dlzky = []
for meno in ludia:
    dlzky.append(len(meno))
```

```
dlzky = [ len(meno) for meno in ludia ]
```

Výroba zoznamov

Majme zoznam zoznamov v tvare [meno, vek]

Vyrobme nový zoznam obsahujúci iba veky

```
ludia = [ ['Jan',20], ['Jozef',25], ['Zuzana',22] ]
```

```
veky = []
for clovek in ludia:
    veky.append(clovek[1])
```

2. prvok vnútorného zoznamu

```
veky = [ clovek[1] for clovek in ludia ]
```

Výroba zoznamov

Majme zoznam zoznamov v tvare [meno, vek]

Zistite vek najstaršej osoby

```
ludia = [ ['Jan',20], ['Jozef',25], ['Zuzana',22] ]
```

```
najvacsi_vek = ludia[0][1]
for clovek in ludia[1:]:
    if clovek[1] > najvacsi_vek:
        najvacsi_vek = clovek[1]
```

2. prvok 1. vnútorného zoznamu

iteruj až od druhej osoby

```
veky = []
for clovek in ludia:
    veky.append(clovek[1])
najvacsi_vek = max(veky)
```

```
najvacsi_vek = max([ clovek[1] for clovek in ludia ])
```

Výroba zoznamov

Majme zoznam zoznamov v tvare [meno, vek]

Zistite meno najstaršej osoby

```
ludia = [ ['Jan',20], ['Jozef',25], ['Zuzana',22] ]
```

```
veky = [ clovek[1] for clovek in ludia ]
najvacsi_vek = max(veky)
index = veky.index(najvacsi_vek)
meno = ludia[index][0]
```

1. prvok vnútorného zoznamu na pozícii `index` v hlavnom zozname

Výroba zoznamov s podmienkami

Majme zoznam čísel

Vyrobme nový zoznam obsahujúci prvky väčšie ako 2

```
zoznam = [ 1, 5, 3, 2, 1, 4 ]
```

```
novy = []
for prvok in zoznam:
    if prvok > 2:
        novy.append(prvok)
```

```
novy = [ prvok for prvok in zoznam if prvok>2 ]
```

Výroba zoznamov s podmienkami

Majme zoznam čísel

Rozdeľte tento zoznam na dva: jeden obsahujúci párne čísla a druhý nepárne

```
zoznam = [ 1, 5, 3, 2, 1, 4 ]
```

```
parne = []
neparne = []
for prvok in zoznam:
    if prvok%2==0:
        parne.append(prvok)
    else:
        neparne.append(prvok)
```

```
parne = [ prvok for prvok in zoznam if prvok%2==0 ]
neparne = [ prvok for prvok in zoznam if prvok%2!=0 ]
```

Výroba zoznamov s podmienkami

Majme zoznam reťazcov

Vyrobme nový zoznam obsahujúci iba mená dlhšie ako 4 znaky

```
ludia = [ 'Jan', 'Jozef', 'Zuzana' ]
```

```
dlhe = []
for meno in ludia:
    if len(meno)>4:
        dlhe.append(meno)
```

```
dlhe = [ meno for meno in ludia if len(meno)>4 ]
```

Výroba zoznamov s podmienkami

Majme zoznam zoznamov v tvare [meno, vek]

Vyrobme zoznam obsahujúci iba mená osôb starších ako 20 rokov

```
ludia = [ ['Jan',20], ['Jozef',25], ['Zuzana',22] ]
```

```
stari = [ clovek[0] for clovek in ludia if clovek[1]>20 ]
```

chceme iba meno človeka
(1. prvok vnútorného zoznamu)

tu je vek

Výroba zoznamov s podmienkami

Majme zoznam `[1, 2, 4, 3, 2, 1]`

Vyrobme zoznam obsahujúci iba unikátne hodnoty

Algoritmus

- ak je index prvku rovný pozícii jeho prvého výskytu, je unikátny
- v opačnom prípade nie je unikátny

FOR slučky v jazyku Python

Funkcia `enumerate(zoznam)` vráti dvojice `index,hodnota`

```
zoznam = [1, 5, "abc", [2, 3]]
for index, hodnota in enumerate(zoznam):
    print("index: {}, hodnota: {}".format(index, hodnota))
```

```
index: 0, hodnota: 1
index: 1, hodnota: 5
index: 2, hodnota: abc
index: 3, hodnota: [2, 3]
```

Výroba zoznamov s podmienkami

Majme zoznam `[1, 2, 4, 3, 2, 1]`

Vyrobme zoznam obsahujúci iba unikátne hodnoty

Algoritmus

- ak je index prvku rovný pozícii jeho prvého výskytu, je unikátny
- v opačnom prípade nie je unikátny

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

for i, prvok in enumerate(zoznam):
    print(i, prvok)
```

```
0 1
1 2
2 4
3 3
4 2
5 1
```

Výroba zoznamov s podmienkami

Majme zoznam `[1, 2, 4, 3, 2, 1]`

Vyrobme zoznam obsahujúci iba unikátne hodnoty

Algoritmus

- ak je index prvku rovný pozícii jeho prvého výskytu, je unikátny
- v opačnom prípade nie je unikátny

```
zoznam = [1, 2, 4, 3, 2, 1]
unikatne = []
for i, prvok in enumerate(zoznam):
    if i==zoznam.index(prvok):
        unikatne.append(prvok)
```

```
unikatne = [ prvok for i, prvok in enumerate(zoznam)
             if i==zoznam.index(prvok) ]
```

Výroba zoznamov s podmienkami

Majme zoznam `[1, 2, 4, 3, 2, 1]`

Vyrobme zoznam obsahujúci indexy všetkých výskytov prvku `x`

Algoritmus

- iteruj cez dvojicu `index, prvok`
- ak `prvok==x`, pridaj `index` do výsledného zoznamu

```
zoznam = [1, 2, 4, 3, 2, 1]
x = 1
indexy = []
for i, prvok in enumerate(zoznam):
    if prvok==x:
        indexy.append(i)
```

```
indexy = [ i for i, p in enumerate(zoznam) if p==x ]
```

Prienik dvoch zoznamov

Majme zoznamy

- $S = \{ x^2 : x \in \{0, \ldots, 9\} \}$
- $V = \{ 1, 2, 4, 8, \ldots, 2^{12} \}$

Vytvorte zoznam $M = \{ x : x \in S$ a zároveň $x \in V \}$

```
S = [ x**2 for x in range(10) ]
V = [ 2**x for x in range(13) ]
M = [ x for x in S if x in V ]
```

Disjunkcia dvoch zoznamov

Majme zoznamy

- S = { x^2 : x ∈ {0, …, 9} }
- V = { 1, 2, 4, 8, …, 2^{12} }

Vytvorte zoznam M = { x : x ∈ S a zároveň **x** ∉ **V** }

```
S = [ x**2 for x in range(10) ]
V = [ 2**x for x in range(13) ]
M = [ x for x in S if x not in V ]
```

```
# prvky vo V, ktoré nie sú v S:
M = [ x for x in V if x not in S ]
```

Zjednotenie dvoch zoznamov

Majme zoznamy

- S = { x^2 : x ∈ {0, …, 9} }
- V = { 1, 2, 4, 8, …, 2^{12} }

Vytvorte zoznam M = { x : x ∈ S alebo x ∈ V }

```
S = [ x**2 for x in range(10) ]
V = [ 2**x for x in range(13) ]
M = S + V
```

Metóda `split`

Metóda `split` rozdelí reťazec na prvky a vytvorí z nich zoznam:

```
>>> veta = "Ja som veta"
>>> slova = veta.split()
>>> print(slova)
['Ja', 'som', 'veta']
```

`split(znak)` rozdelí reťazec na prvky pri výskyte daného znaku:

tento znak sa zároveň odoberie

```
>>> veta = "Ja som veta"
>>> slova = veta.split('a')
>>> print(slova)
['J', ' som vet', '']
```

```
>>> "Jaa som vetaa".split('a')
['J', '', ' som vet', '', '']
>>> "Ja  som   veta  ".split(' ')
['Ja', '', 'som', '', '', 'veta', '', '']
```

Príklad: suma čísel z reťazca

Majme reťazec "1.0 2.4 3.5". Získajme sumu čísel v reťazci.

- najskôr reťazec rozdelíme na pole reťazcov splitom s medzerou
- pole reťazcov prekonvertujeme na pole čísel
- čísla spočítame

```
>>> r = "1.0 2.4 3.5"
>>> cisla = r.split()
['1.0', '2.4', '3.5']
```

zoznam sa nedá konvertovať na float!

```
>>> sum(float(cisla))
TypeError: float() argument must be a string
or a number, not 'list'
```

ok, konverzia po prvkoch

```
>>> sum([ float(x) for x in cisla ])
6.9
```

```
>>> sum([ float(x) for x in r.split() ])
```

Metóda `join`

Metóda `join` spojí zoznam reťazcov pomocou oddelovača

Oddelovačom môže byť ľubovolný reťazec:

```
>>> "_abc_".join(slova)
'Ja_abc_som_abc_veta'
```

Príklad: formátovanie mien

Majme zoznam mien, spojme ich dohromady

```
>>> mena = ['Jan', 'Jozef', 'Zuzana']
>>> r = ", ".join(mena)
>>> print(r)
Jan, Jozef, Zuzana
```

Reťazcové metódy

Vyhľadávanie

Úprava

Nahradenie

Zarovnávanie

Formátovaný výstup

Vyhľadávanie v reťazcoch

Nachádza sa podreťazec v reťazci?

```
>>> 'Py' in 'Python'
True
```

```
>>> 'C++' in 'Python'
False
```

Koľkokrát sa nachádza podreťazec v reťazci?

```
>>> 'Python je skvely'.count('y')
2
```

```
>>> 'Python je skvely'.count('C++')
0
```

Vyhľadávanie v reťazcoch

Na ktorej prvej pozícii sa nachádza podreťazec?

```
>>> 'Python je skvely'.index('y')
1
```

```
>>> 'Python je skvely'.index('C++')
ValueError: substring not found
```

```
>>> 'Python je skvely'.find('y')
1
```

```
>>> 'Python je skvely'.find('C++')
-1
```

Vyhľadávanie v reťazcoch

Na ktorej poslednej pozícii sa nachádza podreťazec?

```
>>> 'Python je skvely'.rindex('y')
15
```

```
>>> 'Python je skvely'.rindex('C++')
ValueError: substring not found
```

```
>>> 'Python je skvely'.rfind('y')
15
```

```
>>> 'Python je skvely'.rfind('C++')
-1
```

Úprava reťazcov

Všetky písmená malé:

```
>>> 'python JE Skvely'.lower()
'python je skvely'
```

Všetky písmená veľké:

```
>>> 'python JE Skvely'.upper()
'PYTHON JE SKVELY'
```

Úprava reťazcov

Kapitalizácia prvého písmena:

```
>>> 'python JE Skvely'.capitalize()
'Python je skvely'
```

Malé písmená na veľké a naopak:

```
>>> 'python JE Skvely'.swapcase()
'PYTHON je sKVELY'
```

Titulky:

```
>>> 'python JE Skvely'.title()
'Python Je Skvely'
```

Nahradenie

Nahradenie podreťazca v reťazci iným podreťazcom:

```
>>> 'python je skvely'.replace('y', 'i/y')
'pi/ython je skveli/y'
```

Nahradenie iba prvého výskytu:

```
>>> 'python je skvely'.replace('y', 'i/y', 1)
'pi/ython je skvely'
```

Odstraňovanie prázdnych znakov

Odstránenie prázdnych znakov zo začiatku:

```
>>> '   python   '.lstrip()
'python   '
```

Odstránenie prázdnych znakov z konca:

```
>>> '   python   '.rstrip()
'   python'
```

Odstránenie prázdnych znakov zo začiatku aj z konca:

```
>>> '   python   '.strip()
'python'
```

Odstraňovanie špecifických znakov

Odstránenie určených znakov zo začiatku:

```
>>> 'www.example.com'.lstrip('w.')
'example.com'
```

Odstránenie určených znakov z konca:

```
>>> 'www.example.com'.rstrip('mc.o')
'www.example'
```

Odstránenie prázdnych znakov zo začiatku aj z konca:

```
>>> 'www.example.com'.strip('cmowz.')
'example'
```

Zarovnávanie

Zarovnanie vľavo:

```
>>> 'python'.ljust(10)
'python    '
```

Zarovnanie vpravo:

```
>>> 'python'.rjust(10)
'    python'
```

Zarovnanie na stred:

```
>>> 'python'.center(10)
'  python  '
```

Formátovanie reťazcov

Dosadenie hodnôt:

```
>>> '{}, {}, {}'.format('a', 'b', 'c')
'a, b, c'
```

Číslované poradie dosaďovania:

```
>>> '{0}, {1}, {2}'.format('a', 'b', 'c')
'a, b, c'
```

```
>>> '{2}, {0}, {1}'.format('a', 'b', 'c')
'c, a, b'
```

```
>>> '{2}, {0}, {2}'.format('a', 'b', 'c')
'c, a, c'
```

Formátovanie reťazcov

"Číslovanie" podľa názvu

```
>>> '{prvy}, {druhy}'.format(prvy='a', druhy='b')
'a, b'
```

```
>>> '{druhy}, {prvy}'.format(prvy='a', druhy='b')
'b, a'
```

Prístup k atribútom argumentov:

```
>>> c = 3-5j
>>> 'r: {0.real}, i: {0.imag}'.format(c)
'r: 3.0, i: -5.0'
```

Formátovanie reťazcov

Zarovnanie vľavo:

```
>>> '{:<30}'.format('text')
'text                          '
```

Zarovnanie vpravo:

```
>>> '{:>30}'.format('text')
'                          text'
```

Zarovnanie na stred:

```
>>> '{:^30}'.format('text')
'             text             '
```

Formátovanie reťazcov

Výpis čísel na daný počet desatinných miest

```
>>> '{:.4f}, {}'.format(3.14, 3.14)
'3.1400, 3.14'
```

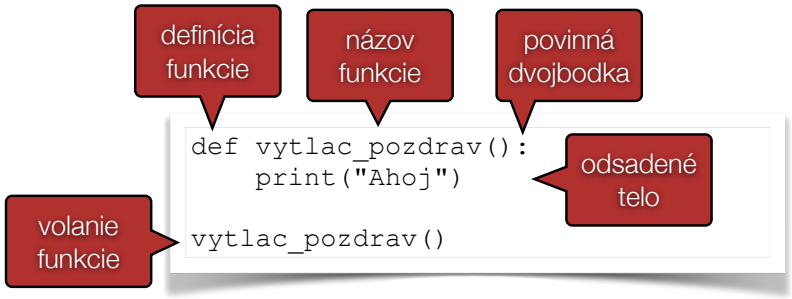
Výpis čísel aj so znamienkom

```
>>> '{:g}, {:g}, {:+g}'.format(3.14, -3.14, 3.14)
'3.14, -3.14, +3.14'
```

Automatický výpočet percent:

```
>>> 'Percenta: {:.1%}'.format(12/27)
'Percenta: 44.4%'
```

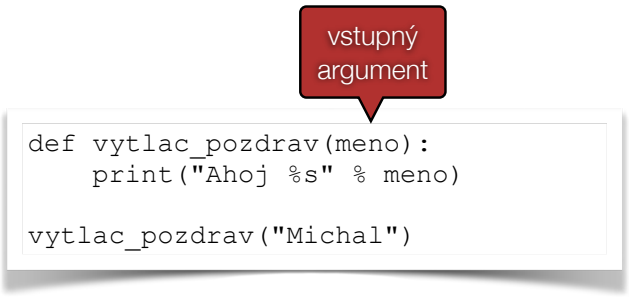
Funkcie bez vstupných argumentov



Poznámky:

- definícia funkcie začína klúčovým slovom `def`
- za názvom funkcie musia byť zátvorky a za nimi dvojbodka
- telo funkcie musí byť odsadené
- pri volaní funkcie musia byť použité zátvorky

Funkcie so vstupnými argumentami



Poznámky:

- do zátvorky píšeme vstupné argumenty oddelené čiarkou
- vstupné argumenty sú lokálne pre danú funkciu
- pri volaní funkcie musíme dodať hodnoty jednotlivých argumentov

Chybné príklady

Nedodaná hodnota:

```
def vytlac_pozdrav(meno):
    print("Ahoj %s" % meno)

vytlac_pozdrav()

TypeError: vytlac_pozdrav() missing 1 required positional
argument: 'meno'
```

```
def vytlac_pozdrav(meno):
    meno = "Michal"
    print("Ahoj %s" % meno)

vytlac_pozdrav()

TypeError: vytlac_pozdrav() missing 1 required positional
argument: 'meno'
```

Lokálnosť argumentov

Nedodaná hodnota:

```
def vytlac_pozdrav(meno):
    print("Ahoj %s" % meno)

meno = "Michal"
vytlac_pozdrav()

TypeError: vytlac_pozdrav() missing 1 required positional
argument: 'meno'
```

Globálne vs lokálne premenné

Globálne premenné je možné čítať:

```
def nieco_urob():
    print("Cislo vo funkcii: %d" % x)

x = 1
nieco_urob()
```

globálna premenná

vezme sa globálna hodnota

Ale nie je možné do nich zapisovať:

```
def nieco_urob():
    x = x + 1
    print("Cislo vo funkcii: %d" % x)

x = 1
nieco_urob()

UnboundLocalError: local variable 'x'
referenced before assignment
```

Globálne vs lokálne premenné

Explicitná deklarácia premennej ako globálnej:

```
def nieco_urob():
    global x
    x = x + 1
    print("Cislo vo funkcii: %d" % x)

x = 1
nieco_urob()
print("Cislo v programe: %d" % x)
```

globálna premenná

```
Cislo vo funkcii: 2
Cislo v programe: 2
```

Globálne vs lokálne premenné

Globálne sú iba premenné v hlavnom programe

```
def f1():
    print(x)

def f2():
    x = 1
    f1()

f2()
```

```
NameError: name 'x' is not defined
```

Globálne vs lokálne premenné

Alebo tie explicitne označené:

```
def f1():
    global x
    print(x)

def f2():
    global x
    x = 1
    f1()

f2()
```

Globálne vs lokálne premenné

Lokálne premenné sa navzájom neprepisujú:

```
def f1():
    x = 1
    print("f1: %d" % x)

def f2():
    x = 2
    print("f2: %d" % x)

def f3():
    x = 3
    print("f3 pred f1: %d" % x)
    f1()
    print("f3 po f1: %d" % x)
    f2()
    print("f3 po f2: %d" % x)

f1()
f2()
f3()
```

```
f1: 1
f2: 2
f3 pred f1: 3
f1: 1
f3 po f1: 3
f1: 2
f3 po f2: 3
```

Odovzdávanie argumentov hodnotou

Argumenty (okrem zoznamov) sa odovzdávajú hodnotou:

```
def vytlac_pozdrav(meno):
    meno = meno.upper()
    print("Ahoj %s" % meno)

meno = "Michal"
vytlac_pozdrav(meno)
print("Hodnota premennej meno: %s" % meno)
```

mení sa iba lokálna hodnota

pri volaní funkcie sa odovzdá hodnota

pôvodná premenná ostane zachovaná

```
Ahoj MICHAL
Hodnota premennej meno: Michal
```

Odovzdávanie argumentov referenciou

Zoznamy sa odovzdávajú referenciou:

premenná ukazuje na ten istý objekt

odovzdanie argumentu referenciou

```
def vytlac_zoznam(zoznam):
    zoznam.sort()
    print("Zoznam vo funkcii: %s" % zoznam)

z = [4, 1, 3, 2]
print("Zoznam v programe: %s" % z)
vytlac_zoznam(z)
print("Zoznam v programe: %s" % z)
```

```
Zoznam v programe: [4, 1, 3, 2]
Zoznam vo funkcii: [1, 2, 3, 4]
Zoznam v programe: [1, 2, 3, 4]
```

Odovzdávanie argumentov referenciou

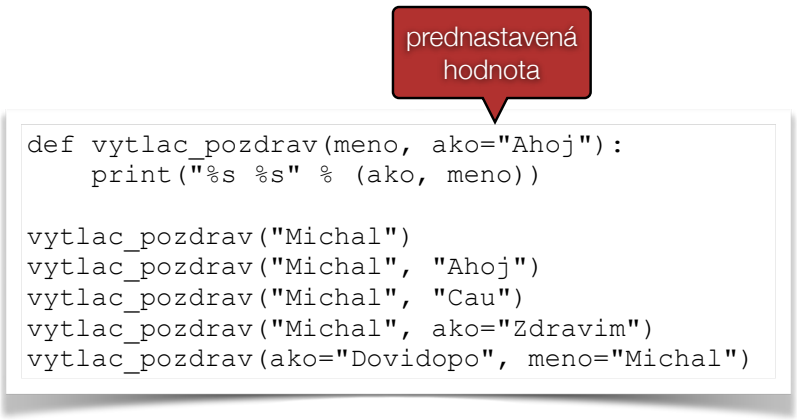
Meňte kópiu zoznamu:

```
def vytlac_zoznam(zoznam):
    zoznam = zoznam[:]
    zoznam.sort()
    print("Zoznam vo funkcii: %s" % zoznam)

z = [4, 1, 3, 2]
print("Zoznam v programe: %s" % z)
vytlac_zoznam(z)
print("Zoznam v programe: %s" % z)
```

vytvorenie kópie

```
Zoznam v programe: [4, 1, 3, 2]
Zoznam vo funkcii: [1, 2, 3, 4]
Zoznam v programe: [4, 1, 3, 2]
```

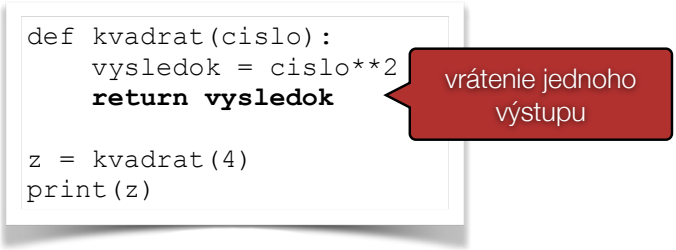
Prednastavené hodnoty vstupov



Poznámky:

- argumenty s prednastavenou hodnotou musia byť až na konci
- pri volaní funkcie sa takého argumenty nemusia špecifikovať
- poradie argumentov pri volaní funkcie môže byť zmenené ak explicitne použijeme názov argumentu

Výstup z funkcie



Poznámky:

- funkcia sa skončí keď program narazí na príkaz `return` bez ohľadu na to, že za ním nasledujú ďalšie príkazy
- ak sa príkaz `return` neaktivuje, výsledok je automaticky `None`

Výstup z funkcie

```
def odmocnina(cislo):
    if cislo >= 0:
        return cislo**0.5
    else:
        return 0

print(odmocnina(16))
print(odmocnina(-2.5))
```

```
4.0
0
```

Výstup z funkcie

```
def odmocnina(cislo):
    if cislo >= 0:
        return cislo**0.5
    else:
        return None

print(odmocnina(16))
print(odmocnina(-2.5))
```

znamená "nič"

```
4.0
None
```

Výstup z funkcie

```
def odmocnina(cislo):
    if cislo >= 0:
        return cislo**0.5

print(odmocnina(16))
print(odmocnina(-2.5))
```

```
4.0
None
```

dôsledok defaultného výstupu z funkcie

Vrátenie viacerých výstupov

```
def kvadrat(cislo):
    vysledok = cislo**2
    return vysledok, cislo

k, cislo = kvadrat(4)
```

2 výstupy z funkcie

"rozbalenie" výstupov

Poznámky:

- viaceré výstupy sú oddelené v `return` čiarkou
- v rovnakom poradí sa výstupy "rozbalia" v programe

Rekurzívne volanie funkcií

```
def faktorial(cislo):
    if cislo == 1:
        return 1
    else:
        return cislo*faktorial(cislo-1)

print(faktorial(4))
```

Poznámky:

- rekurzia vzniká keď funkcia volá samu seba
- rekurzia sa vždy musí ukončiť
- nie je to najefektívnejší spôsob, ale niekedy sa mu nedá ujsť

Dokumentačné reťazce

```
def faktorial(cislo):
    """
    Volanie: y = faktorial(x)
    Vstup: x ako cele cislo
    Vystup: y ako cele cislo
    """
    if cislo == 1:
        return 1
    else:
        return cislo*faktorial(cislo-1)
```

Poznámky:

- dokumentácia sa píše medzi 3 úvodzovky
- daná časť je interpretovaná ako komentár
- je ekvivalentná použitiu mriežky (#)

Dokumentačné reťazce

```
def faktorial(cislo):
    """
    Volanie: y = faktorial(x)
    Vstup: x ako cele cislo
    Vystup: y ako cele cislo
    """
    if cislo == 1:
        return 1
    else:
        return cislo*faktorial(cislo-1)
```

```
help(faktorial)

faktorial(cislo)
    Volanie: y = faktorial(x)
    Vstup: x ako cele cislo
    Vystup: y ako cele cislo
```

Dátový typ slovník

Reprezentuje štruktúru v tvare `kľúč: hodnota`

```
osoba = { 'meno': 'Michal', 'priezvisko': 'Kvasnica' }
```

Alternatívny zápis:

```
osoba = dict(meno='Michal', priezvisko='Kvasnica')
```

Kľúčom môže byť aj číslo:

```
osoba = {1: 'Michal', 2.5: 'Kvasnica' }
```

Kľúče sú case-sensitive:

```
osoba = {'meno': 'Michal', 'Meno': 'Jan'}
```

Príklady slovníkov

Jedna osoba:

```
osoba = { 'meno': 'Michal', 'priezvisko': 'Kvasnica' }
```

Jedna osoba:

```
osoba = { 'Kvasnica': 'Michal', 'Drgona': 'Jan' }
```

Viacero osôb s hodnotami typu slovník:

```
osoby = { 1: {'meno': 'Michal', 'priezvisko': 'Kvasnica' },
          2: {'meno': 'Jan', 'priezvisko': 'Drgona'} }
```

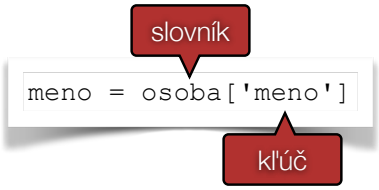
Viacero osôb ako zoznam slovníkov:

```
michal = { 'meno': 'Michal', 'priezvisko': 'Kvasnica' }
jan    = { 'meno': 'Jan', 'priezvisko': 'Drgona' }
osoby  = [ michal, jan ]
```

Prístup k dátam

```
osoba = { 'meno': 'Michal', 'priezvisko': 'Kvasnica' }
```

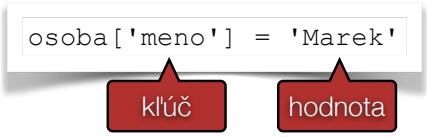
Čítanie hodnoty uloženej pod nejakým kľúčom:



Čítanie neexistujúceho kľúča vedie k chybe:

```
>>> meno = osoba['novy_kluc']
KeyError: 'novy_kluc'
```

Zmena/pridanie hodnoty:



Postupná tvorba slovníka

Inicializácia prázdneho slovníka:

```
osoba = {}
```

Postupné pridávanie kľúčov:

```
osoba['meno'] = 'Michal'
osoba['priezvisko'] = 'Kvasnica'
```

Pozor! Opätovné použitie kľúča prepíše pôvodnú hodnotu:

Operácie s kľúčmi

```
osoba = { 'meno': 'Michal', 'priezvisko': 'Kvasnica' }
```

Zistenie počtu kľúčov v slovníku:

```
dlzka = len(osoba)
```

Nachádza sa kľúč v slovníku?

```
>>> osoba.has_key('meno')
True
>>> 'meno' in osoba
True
>>> 'Michal' in osoba
False
```

Vymazanie kľúča (a príslušnej hodnoty) zo slovníka:

```
del osoba['meno']
```

Bezpečný prístup k hodnotám

Nebezpečný spôsob (nekontroluje, či kľúč existuje):

```
kluc = input('Kluc: ')
hodnota = slovnik[kluc]
```

Bezpečný spôsob s kontrolou:

```
kluc = input('Kluc: ')
if kluc in slovnik:
    hodnota = slovnik[kluc]
else:
    hodnota = None
```

Automatický spôsob:

```
hodnota = slovnik.get(kluc)
```

vráti None ak kľúč neexistuje

```
hodnota = slovnik.get(kluc, '')
```

vráti '' ak kľúč neexistuje

Iterovanie slovníkov

Iterovanie cez kľúče:

```
for kluc in osoba.keys():
    print(kluc)
```

```
priezvisko
meno
```

Iterovanie cez hodnoty:

```
for hodnota in osoba.values():
    print(hodnota)
```

```
Kvasnica
Michal
```

Iterovanie cez páry kľúč-hodnota:

```
priezvisko Kvasnica
meno Michal
```

```
for kluc, hodnota in osoba.items():
    print(kluc, hodnota)
```

```
('priezvisko', 'Kvasnica')
('meno', 'Michal')
```

```
for par in osoba.items():
    print(par)
```

Príklad: zoradený výpis

```
>>> s = {'c': 1, 'b': 2, 'aa': 3, 'a': 4}
>>> print(s)
{'aa': 3, 'b': 2, 'c': 1, 'a': 4}
```

Zoradenie podľa kľúča:

```
for k in sorted(s.keys()):
    print(k, s[k])
```

```
a 4
aa 3
b 2
c 1
```

Zoradenie podľa hodnoty:

```
for h in sorted(s.values()):
    print(h)
```

```
1
2
3
4
```

Príklady vyhľadávania

Jedna osoba:

```
osoba = { 'meno': 'Michal', 'priezvisko': 'Kvasnica' }
```

Je v slovníku kľúč?

```
>>> 'meno' in osoba
True
```

vyhľadáva iba v kľúčoch

```
>>> 'Michal' in osoba
False
```

Je v slovníku hodnota?

```
>>> 'Michal' in osoba.values()
True
```

Príklady vyhľadávania

Jedna osoba:

```
osoba = { 'Kvasnica': 'Michal', 'Drgona': 'Jan' }
```

Je v slovníku kľúč?

```
>>> 'Kvasnica' in osoba
True
```

vyhľadáva iba v kľúčoch

```
>>> 'Michal' in osoba
False
```

Je v slovníku hodnota?

```
>>> 'Michal' in osoba.values()
True
>>> 'Kvasnica' in osoba.values()
False
```

Príklady vyhľadávania

Viacero osôb s hodnotami typu slovník:

```
osoby = { 1: {'meno': 'Michal', 'priezvisko': 'Kvasnica' },
          2: {'meno': 'Jan', 'priezvisko': 'Drgona'} }
```

Je v slovníku kľúč?

```
>>> 1 in osoby
True
```

vyhľadáva iba v kľúčoch

```
>>> 'Michal' in osoby
False
```

Je v slovníku hodnota?

```
>>> 'Michal' in osoby.values()
False
```

hodnotami sú slovníky

Príklady vyhľadávania

Viacero osôb s hodnotami typu slovník:

```
osoby = { 1: {'meno': 'Jan', 'priezvisko': 'Kvasnica' },
          2: {'meno': 'Jan', 'priezvisko': 'Drgona'} }
```

Je v slovníku hodnota?

```
for kluc in osoby.keys():
    if 'Jan' in osoby[kluc].values():
        print(kluc)
```

1
2

```
for kluc in osoby.keys():
    if osoby[kluc]['priezvisko'] == 'Kvasnica':
        print(kluc)
```

1

Príklady vyhľadávania

Viacero osôb ako zoznam slovníkov:

```
michal = { 'meno': 'Michal', 'priezvisko': 'Kvasnica' }
jan    = { 'meno': 'Jan', 'priezvisko': 'Drgona' }
osoby  = [ michal, jan ]
```

Priezvisko osoby s menom Ján?

```
for prvok in osoby:
    if prvok['meno'] == 'Jan':
        print(prvok['priezvisko'])
```

Drgona

Index osoby s priezviskom Kvasnica?

```
for index, prvok in enumerate(osoby):
    if prvok['priezvisko'] == 'Kvasnica':
        print(index)
```

0

Priradenie a kópia slovníka

Priraďuje sa iba referencia:

```
>>> slovnik = {'meno': 'Michal'}
>>> druhy = slovnik
>>> druhy['meno'] = 'Jan'
>>> print(slovnik)
```

aj druhý slovník ukazuje na rovnaký obsah

```
{'meno': 'Jan'}
```

Vytvorenie kópie:

```
>>> slovnik = {'meno': 'Michal'}
>>> druhy = slovnik.copy()
>>> druhy['meno'] = 'Jan'
>>> print(druhy)
>>> print(slovnik)
```

vytvorenie kópie obsahu

```
{'meno': 'Jan'}
{'meno': 'Michal'}
```

Použitie slovníkov vo funkciách

Slovníky sa odovzdávajú referenciou:

```
def zostarni(osoba):
    osoba['vek'] = osoba['vek'] + 1

o = {'meno': 'Michal', 'vek': 38}
zostarni(o)
print(o)
```

bude sa modifikovať priamo premenná

```
{'vek': 39, 'meno': 'Michal'}
```

Príklad: prekladový slovník

```
slovnik = {}
slovnik['red'] = 'cerveny'
slovnik['blue'] = 'modry'
slovnik['green'] = 'zeleny'
slovnik['yellow'] = 'zlty'
```

```
slovo = input('Vloz anglicke slovo: ')
```

```
if slovo in slovnik:
    print('Preklad je', slovnik[slovo])
else:
    print('Slovo', slovo, 'nie je v slovniku')
```

Príklad: databáza osôb

```
zoznam_osob = []
osoba = {'meno': 'Michal', 'priezvisko': 'Kvasnica'}
zoznam_osob.append(osoba)
osoba = {'meno': 'Jan', 'priezvisko': 'Drgona'}
zoznam_osob.append(osoba)
```

```
>>> print(zoznam_osob)
[{'priezvisko': 'Kvasnica', 'meno': 'Michal'},
{'priezvisko': 'Drgona', 'meno': 'Jan'}]
```

```
>>> print(zoznam_osob[0])
{'priezvisko': 'Kvasnica', 'meno': 'Michal'}
```

```
>>> print(zoznam_osob[0]['meno'])
Michal
```

Otvorenie súboru

```
subor = open(nazov, mod)
```

Mód otvorenia:

- `'r'`: iba na čítanie
- `'w'`: iba na zápis (súbor sa vždy pri otvorení vymaže)
- `'a'`: na pridávanie (existujúci súbor ostane zachovaný)
- `'r+'`: na čítanie aj na zápis

Poznámky:

- `'r'` je defaultný mód pri `open(nazov)`
- je možné otvoriť viacero súborov naraz (každý musí mať svoju premennú)
- otvorenie neexistujúceho súboru vedie k chybe (okrem módu `'w'`)
- súbor/adresár musí mať príslušné prístupové práva

Cesty k súborom

Súbor v aktuálnom adresári:

```
subor = open('subor.txt')
```

Súbor v podadresári aktuálneho adresára:

```
subor = open('podadresar/subor.txt')
```

Súbor v adresári o úroveň vyššie:

```
subor = open('../subor.txt')
```

Absolútna cesta:

```
subor = open('/home/michal/subor.txt')
```

Konce riadkov

- Windows: `\r\n`
- Unix/Linux/Mac: `\n`

Čítanie zo súboru

Čítanie celého obsahu súboru do reťazca:

```
obsah = subor.read()
```

Čítanie iba daného počtu bajtov/znakov:

```
obsah = subor.read(pocet_bajtov)
```

Poznámky:

- je váš problém, ak sa pri `subor.read()` obsah nezmestí do pamäti
- vráti prázdny reťazec ak objekt ukazuje na koniec súboru
- vracia všetky znaky vrátane koncov riadkov

Čítanie zo súboru

Čítanie jedného riadka:

```
riadok = subor.readline()
```

Príklad:

```
>>> subor.readline()
'Toto je prvy riadok.\n'
>>> subor.readline()
'Druhy riadok\n'
>>> subor.readline()
''
```

Poznámky:

- vráti riadok vrátane znaku/znakov konca riadka
- na jeho/ich odstránenie použite `riadok.rstrip()`
- vráti prázdny reťazec ak objekt ukazuje na koniec súboru

Postupné čítanie zo súboru

```
for riadok in subor:
    print(riadok, end='')
```

Poznámky:

- iterovanie cez file objekt postupne načítava jednotlivé riadky
- riadky sú načítané vrátane znakov konca riadkov
- slučka sa ukončí, keď dosiahneme koniec súboru

Načítanie všetkých riadkov do zoznamu

1. spôsob:

```
subor = open('subor.txt', 'r')
riadky = list(subor)
```

2. spôsob:

```
subor = open('subor.txt', 'r')
riadky = subor.readlines()
```

Zápis do súboru

Zápis reťazca do súboru

```
subor.write('pridany riadok\n')
```

Poznámky:

- zapisovať môžeme iba reťazce, nie čísla, napr. `write(65)` nefunguje
- `write` na rozdiel od `print` nepridáva automaticky znak nového riadka
- metóda vracia počet zapísaných znakov
- v závislosti od platformy k skutočnému zápisu dôjde až pri zavretí alebo "spláchnutí" súboru cez `subor.flush()`

Zápis zoznamu reťazcov do súboru

```
subor.writelines(['prvy riadok\n', 'druhy riadok\n'])
```

Pohyb v súbore

Prechod na konkrétnu pozíciu v súbore od jeho začiatku:

```
subor.seek(pozicia)
```

Prechod na koniec súboru:

```
subor.seek(0, 2)
```

Zistenie aktuálnej pozície:

```
pozicia = subor.tell()
```

Poznámky:

- metóda `seek` tiež vráti polohu "kurzora" v súbore
- zistenie počtu znakov: `pocet = subor.seek(0, 2)`

Zavretie súboru

Spláchnutie:

```
subor.flush()
```

Zavretie:

```
subor.close()
```

Poznámky:

- po `flush()` sa ešte dá do súboru zapisovať aj z neho čítať
- po `close()` sa už do súboru nedá zapisovať
- zistenie, či je súbor zavretý: `zavrety = subor.closed`

Práca s modulmi

Importovanie všetkých funkcií z modulu:

```
import nazov_modulu
```

Volanie funkcie v module:

```
vystup = nazov_modulu.nazov_funkcie(vstup1, vstup2, ...)
```

Importovanie funkcie z modulu:

```
from nazov_modulu import nazov_funkcie
vystup = nazov_funkcie(vstup1, vstup2, ...)
```

Malý prehľad modulov

Matematika: `math, random, statistics`

Práca s dátumami a časmi: `time, calendar`

Služby operačného systému: `os, shutil`

Internetové služby: `urllib, email, ftplib`

Spracovanie HTML a XML: `html.parser, xml.dom`

Databázy: `pickle, sqlite3, dbm`

Multimédiá: `audioop, wave, imghdr, sndhdr`

Malý prehľad modulov

Matematika: **`math`**, `random`, `statistics`

Práca s dátumami a časmi: `time`, `calendar`

Služby operačného systému: `os`, `shutil`

Internetové služby: `urllib`, `email`, `ftplib`

Spracovanie HTML a XML: `html.parser`, `xml.dom`

Databázy: `pickle`, `sqlite3`, `dbm`

Multimédiá: `audioop`, `wave`, `imghdr`, `sndhdr`

Modul `math`

```
import math
```

Zaokrúhľovanie:

- `math.ceil(x)` – zaokrúhlenie smerom nahor
- `math.floor(x)` – zaokrúhlenie smerom nadol
- `math.trunc(x)` – vráti iba časť pred desatinnou čiarkou

Matematické funkcie:

- `math.fabs(x)` – absolútna hodnota
- `math.factorial(x)` – faktoriál daného čísla
- `math.fsum(zoznam)` – suma prvkov zoznamu
- `math.gcd(x, y)` – najväčší spoločný deliteľ

Modul `math`

Logaritmy a exponenty:

- `math.log(x)` – prirodzený logarimus
- `math.log(x, b)` – logaritmus v danej báze
- `math.log2(x)` – dvojkový logaritmus
- `math.log10(x)` – dekadický logaritmus
- `math.pow(x,y)` – umocnenie x^y
- `math.sqrt(x)` – druhá odmocnina

Trigonometria:

- `math.acos(x), math.asin(x), math.atan(x)`
- `math.cos(x), math.sin(x), math.tan(x)`
- `math.degrees(x)` – prevod radiánov na stupne
- `math.radians(x)` – prevod stupňov na radiány

Modul `math`

Konštanty:

- `math.pi`
- `math.e`
- `math.inf` – nekonečno (vhodné pri porovnávaní)
- `math.nan` – "not a number" (napr. chýbajúce merania)

Porovnávanie:

- `math.isclose(x, y, rel_tol=1e-9, abs_tol=0.0)`
- `math.isfinite(x)` – pravda ak je číslo konečné (nie inf a nan)
- `math.isinf(x)` – pravda ak je číslo +/- inf
- `math.isnan(x)` – pravda ak je číslo nan

Malý prehľad modulov

Matematika: `math`, **`random`**, `statistics`

Práca s dátumami a časmi: `time`, `calendar`

Služby operačného systému: `os`, `shutil`

Internetové služby: `urllib`, `email`, `ftplib`

Spracovanie HTML a XML: `html.parser`, `xml.dom`

Databázy: `pickle`, `sqlite3`, `dbm`

Multimédiá: `audioop`, `wave`, `imghdr`, `sndhdr`

Modul `random`

```
import random
```

Generovanie náhodných čísel:

- `random.random()` – náhodný float z intervalu [0.0, 1.0)
- `random.uniform(a, b)` – náhodný float z intervalu [a, b]
- `random.randint(a, b)` – náhodný integer z intervalu [a, b]

Práca so zoznamami čísel:

- `random.choice(zoznam)` – výber náhodného prvku
- `random.shuffle(zoznam)` – náhodné premiešanie poradia
- `random.sample(zoznam,k)` – vytvorenie populácie *k* prvkov

random.choice()

Výber náhodného prvku zo zoznamu čísel:

```
>>> zoznam = [1, 4, 8, 12, 16]
>>> prvok = random.choice(zoznam)
>>> print(prvok)
8
```

Výber náhodného prvku zo zoznamu reťazcov:

```
>>> zoznam = ['cervena', 'zelena', 'zlta', 'modra']
>>> prvok = random.choice(zoznam)
>>> print(prvok)
zelena
```

random.shuffle()

Premiešanie poradia zoznamu čísel:

```
>>> zoznam = [1, 4, 8, 12, 16]
>>> random.shuffle(zoznam)
>>> print(zoznam)
[12, 8, 1, 4, 16]
```

in-place správanie

Premiešanie poradia zoznamu reťazcov:

```
>>> zoznam = ['cervena', 'zelena', 'zlta', 'modra']
>>> random.shuffle(zoznam)
>>> print(zoznam)
['zlta', 'cervena', 'zelena', 'modra']
```

Premiešanie so zachovaním pôvodného zoznamu:

```
>>> zoznam = [1, 4, 8, 12, 16]
>>> kopia = zoznam[:]
>>> random.shuffle(kopia)
```

vytvorenie kópie

random.sample()

Výber náhodných trojíc zo zoznamu čísel:

```
>>> zoznam = [1, 4, 8, 12, 16]
>>> trojica = random.sample(zoznam, 3)
>>> print(trojica)
[8, 16, 1]
```

Výber dvoch farieb:

```
>>> zoznam = ['cervena', 'zelena', 'zlta', 'modra']
>>> farby = random.sample(zoznam, 2)
>>> print(farby)
['cervena', 'zlta']
```

Malý prehľad modulov

Matematika: `math, random, statistics`

Práca s dátumami a časmi: **`time`**`, calendar`

Služby operačného systému: `os, shutil`

Internetové služby: `urllib, email, ftplib`

Spracovanie HTML a XML: `html.parser, xml.dom`

Databázy: `pickle, sqlite3, dbm`

Multimédiá: `audioop, wave, imghdr, sndhdr`

Modul `time`

```
import time
```

Zistenie lokálneho času:

```
>>> cas = time.localtime()
>>> print(cas)
time.struct_time(tm_year=2016, tm_mon=5,    tm_mday=2,
                 tm_hour=10,   tm_min=31,   tm_sec=54,
                 tm_wday=0,    tm_yday=123, tm_isdst=1)
```

Prístup k jednotlivým atribútom:

- `cas.tm_year, cas.tm_mon, cas.tm_mday` – rok, mesiac, deň
- `cas.tm_hour, cas.tm_min, cas.tm_sec` – hodina, minúta, sekunda
- `cas.tm_wday` – číslo dňa (0=pondelok)
- `cas.tm_yday` – číslo dňa v roku
- `cas.tm_isdst` – letný čas

Formátovanie dátumu a času

Formátovanie času do reťazca

```
>>> print(time.strftime('%a, %d %b %Y %H:%M:%S'))
Mon, 02 May 2016 11:02:05
```

Formátovacie znaky:

`%a`	skrátený názov dňa	`%S`	sekunda [00, 61]
`%A`	plný názov dňa	`%W`	číslo týždňa
`%b`	skrátený názov mesiaca	`%w`	číslo dňa v týždni
`%B`	plný názov mesiaca	`%y`	rok bez letopočtu [00, 99]
`%d`	deň v mesiaci [01, 31]	`%Y`	rok aj s letopočtom
`%H`	hodina [00, 23]	`%z`	offset časovej zóny
`%m`	mesiac [01, 12]	`%Z`	názov časovej zóny
`%M`	minúta [00, 59]	`%%`	znak percento

Formátovanie dátumu a času

Rýchle formátovanie dátumu aj času:

```
>>> print(time.strftime('%c'))
Mon May  2 11:14:49 2016
```

Rýchle formátovanie dátumu:

```
>>> print(time.strftime('%x'))
05/02/16
```

Rýchle formátovanie času:

```
>>> print(time.strftime('%X'))
11:15:27
```

Parsovanie dátumu a času

Parsovanie:

```
>>> time.strptime('03.05.2016', '%d.%m.%Y')
time.struct_time(tm_year=2016, tm_mon=5,    tm_mday=3,
                 tm_hour=0,    tm_min=0,    tm_sec=0,
                 tm_wday=1,    tm_yday=124, tm_isdst=-1)
```

Formátovacie znaky:

%a	skrátený názov dňa	%S	sekunda [00, 61]
%A	plný názov dňa	%W	číslo týždňa
%b	skrátený názov mesiaca	%w	číslo dňa v týždni
%B	plný názov mesiaca	%y	rok bez letopočtu [00, 99]
%d	deň v mesiaci [01, 31]	%Y	rok aj s letopočtom
%H	hodina [00, 23]	%z	offset časovej zóny
%m	mesiac [01, 12]	%Z	názov časovej zóny
%M	minúta [00, 59]	%%	znak percento

Malý prehľad modulov

Matematika: `math, random, statistics`

Práca s dátumami a časmi: `time, calendar`

Služby operačného systému: `os, shutil`

Internetové služby: `urllib, email, ftplib`

Spracovanie HTML a XML: `html.parser, xml.dom`

Databázy: `pickle, sqlite3, dbm`

Multimédiá: `audioop, wave, imghdr, sndhdr`

Modul `calendar`

```
import calendar
```

Vytlačenie kalendára na daný rok:

```
>>> calendar.prcal(2016)
```

```
                                  2016

      January                   February                  March
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                1  2  3       1  2  3  4  5  6  7          1  2  3  4  5  6
 4  5  6  7  8  9 10       8  9 10 11 12 13 14       7  8  9 10 11 12 13
11 12 13 14 15 16 17      15 16 17 18 19 20 21      14 15 16 17 18 19 20
18 19 20 21 22 23 24      22 23 24 25 26 27 28      21 22 23 24 25 26 27
25 26 27 28 29 30 31      29                        28 29 30 31

      April                     May                       June
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                1  2  3                            1          1  2  3  4  5
 4  5  6  7  8  9 10       2  3  4  5  6  7  8       6  7  8  9 10 11 12
11 12 13 14 15 16 17       9 10 11 12 13 14 15      13 14 15 16 17 18 19
18 19 20 21 22 23 24      16 17 18 19 20 21 22      20 21 22 23 24 25 26
25 26 27 28 29 30         23 24 25 26 27 28 29      27 28 29 30
                          30 31

       July                    August                  September
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                1  2  3       1  2  3  4  5  6  7                1  2  3  4
 4  5  6  7  8  9 10       8  9 10 11 12 13 14       5  6  7  8  9 10 11
11 12 13 14 15 16 17      15 16 17 18 19 20 21      12 13 14 15 16 17 18
18 19 20 21 22 23 24      22 23 24 25 26 27 28      19 20 21 22 23 24 25
25 26 27 28 29 30 31      29 30 31                  26 27 28 29 30

      October                   November                  December
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                1  2          1  2  3  4  5  6                1  2  3  4
 3  4  5  6  7  8  9       7  8  9 10 11 12 13       5  6  7  8  9 10 11
10 11 12 13 14 15 16      14 15 16 17 18 19 20      12 13 14 15 16 17 18
17 18 19 20 21 22 23      21 22 23 24 25 26 27      19 20 21 22 23 24 25
24 25 26 27 28 29 30      28 29 30                  26 27 28 29 30 31
31
```

Modul `calendar`

```
import calendar
```

Vytlačenie kalendára na daný rok:

```
>>> calendar.prcal(2016)
```

Vytlačenie kalendára na daný mesiac/rok:

```
>>> calendar.prmonth(2016, 5)
```

```
     May 2016
Mo Tu We Th Fr Sa Su
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

Modul `calendar`

Je rok priestupný?

```
>>> print(calendar.isleap(2014))
False
>>> print(calendar.isleap(2016))
True
```

Aké je číslo dňa v týždni? (0=pondelok, 6=nedeľa)

```
>>> print(calendar.weekday(2016, 5, 3))
1
>>> print(calendar.weekday(2016, 5, 1))
6
```

Výnimky

Dva druhy chýb:

- syntaktické chyby (zachytené ešte pred spustením programu)
- výnimky (chyby pri vykonávaní jednotlivých príkazov)

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

```
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined
```

```
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

Zachytávanie výnimiek

Príklad:

```
x = int(input("Please enter a number: "))
Please enter a number: asd
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'asd'
```

Zachytenie výnimky cez `try` - `except`:

```
try:
    x = int(input("Please enter a number: "))
except ValueError:
    print("Chyba! Nezadal si platne cislo.")
```

Lepšie riešenie

Opakuj načítavanie, dokým užívateľ nevloží platné číslo:

```
while True:
    try:
        x = int(input("Please enter a number: "))
        break
    except ValueError:
        print("Chyba! Nezadal si platne cislo.")
```

Jeden `try` môže mať viacero `except` častí

Individuálne reakcie na výnimky:

```
try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except OSError:
    print("OS error.")
except ValueError:
    print("Could not convert data to an integer.")
except:
    print("Unexpected error.")
```

Spoločná reakcia na viaceré výnimky:

```
try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except (OSError, ValueError):
    print("Vyskytla sa nejaka chyba.")
```

Upratovanie cez konštrukciu `finally`

Blok finally sa vykoná vždy po ukončení `try` - `except` bloku:

```
try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except:
    print("Vyskytla sa chyba.")
finally:
    f.close()
```

Slúži na upratovanie:

- uzavretie otvorených súborov
- ukončenie externých procesov
- oznámanie o ukončení programu
- ...

Upratovanie cez konštrukciu `finally`

Ak počas behu try bloku vznikne nezachytená výnimka, znova sa vyvolá po skončení `finally` bloku:

```
try:
    x = int(input("Vloz cislo: "))
finally:
    print("Dovidenia.")
```

```
Vloz cislo: asd
Dovidenia.
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
ValueError: invalid literal for int() with base 10: 'asd'
```

Upratovanie cez konštrukciu `finally`

Zachytené chyby sa znova nevyvolávajú:

```
try:
    x = int(input("Vloz cislo: "))
except:
    print("Vyskytla sa chyba.")
finally:
    print("Dovidenia.")
```

```
Vloz cislo: asd
Vyskytla sa chyba.
Dovidenia.
```

Dnes

Výnimky (exceptions)

Knižnica `os`

Knižnica `os`

Obsahuje funkcie na prácu s operačným systémom:

- zistenie aktuálneho pracovného adresára: `os.getcwd()`
- zmena adresára: `os.chdir()`
- vytvorenie adresára: `os.mkdir()`
- výpis súborov v adresári: `os.listdir()`
- zistenie údajov o súbore (napr. veľkosť): `os.stat()`
- premenovanie súboru: `os.rename()`
- vymazanie súboru: `os.remove()`
- vymazanie adresára: `os.rmdir()`
- `...`

Zistenie aktuálneho pracovného adresára

Formát volania:

- `os.getcwd()`
- vstupy: žiadne
- výstup: reťazec

Príklad:

```
import os
adresar = os.getcwd()
print("Aktualny adresar: {}".format(adresar))

Aktualny adresar: /notebooks/prg2/2017/Kvasnica
```

Zmena aktuálneho pracovného adresára

Formát volania:

- `os.chdir(cesta)`
- vstup: reťazec s novou cestou (môže byť relatívna alebo absolútna)
- výstup: `FileNotFoundError` ak cesta neexistuje

Príklad:

```
import os
print(os.getcwd())
os.chdir('..')
print(os.getcwd())

/notebooks/prg2/2017/Kvasnica
/notebooks/prg2/2017
```

```
os.chdir('/neexistujuca/abolutna/cesta')
FileNotFoundError: [Errno 2] No such file or
directory: '/neexistujuca/abolutna/cesta'
```

Vytvorenie adresára

Formát volania:

- `os.mkdir(cesta)`
- vstup: relatívny alebo absolútny názov adresára
- výstup: `FileExistsError` ak cesta už existuje

Príklad:

```
import os
print(os.getcwd())
os.mkdir("novy_adresar") # relativna cesta
os.chdir("novy_adresar")
print(os.getcwd())

/notebooks/prg2/2017
/notebooks/prg2/2017/novy_adresar
```

Výpis súborov v adresári

Formát volania:

- `os.listdir(cesta)`
- vstup: reťazec s novou cestou (defaultne `cesta='.'`)
- výstup: zoznam názvov súborov a adresárov

Príklad:

```
import os
print(os.listdir())

['03_precvicovanie.ipynb',
'.ipynb_checkpoints', '02_emaily.txt']
```

```
import os
print(os.listdir('..'))

['Furka', 'Fedorkovic', 'ntbcp.sh', 'Morozov',
'Beno', 'Horvathova', 'zaloha_2016', 'Holaza',
'Boros', 'Kis', 'Kvasnica', 'Bacikova',
'Dugat']
```

Zistenie údajov o súbore

Formát volania:

- `os.stat(cesta)`
- vstup: reťazec s cestou k súboru (absolútna alebo relatívna cesta)
- výstup: štruktúra s detailami

Príklad:

```
import os
detaily = os.stat('01_precvicovanie.ipynb')
print(detaily)

os.stat_result(st_mode=33188, st_ino=1465732, st_dev=64769, st_nlink=1,
st_uid=0, st_gid=0, st_size=10074, st_atime=1489032609,
st_mtime=1486707155, st_ctime=1486707155)
```

```
print('Velkost suboru v bajtoch {}'.format(detaily.st_size))

Velkost suboru v bajtoch: 10074
```

Premenovanie súboru

Formát volania:

- `os.rename(zdroj, ciel)`
- vstup: reťazce so starou a s novou cestou (môže byť aj adresár)
- výstup: `OSError` ak došlo ku chybe

Príklad:

```
import os
subor = open('docasny.txt', 'w')
subor.close()
os.rename('docasny.txt', 'novy.dat')
print(os.listdir())
```

```
['03_precvicovanie.ipynb', '.ipynb_checkpoints', 'novy.dat',
'02_emaily.txt', '02_precvicovanie.ipynb', '02_vety_kapitalizacia.txt',
'02_subory.txt', '01_precvicovanie.ipynb', '02_logs.txt']
```

Vymazanie súboru

Formát volania:

- `os.remove(cesta)`
- vstup: reťazec s cestou k súboru (absolútna alebo relatívna)
- výstup: `IsADirectoryError` ak je vstupom cesta k adresáru

Príklad:

```
import os
subor = open('docasny.txt', 'w')
subor.close()
os.remove('docasny.txt')
```

```
import os
os.mkdir('docasny_adresar')
os.remove('docasny_adresar')

IsADirectoryError: [Errno 21] Is a directory:
'docasny_adresar'
```

Vymazanie adresára

Formát volania:

- `os.rmdir(cesta)`
- vstup: reťazec s cestou k adresáru (absolútna alebo relatívna)
- výstup: `FileNotFoundError` ak cesta nie je adresár alebo `OSError` ak adresár nie je prázdny

Príklad:

```
import os
os.mkdir('docasny')
os.rmdir('docasny') # OK
os.mkdir('docasny')
subor = open('docasny/test.txt', 'w'); subor.close()
os.rmdir('docasny/test.txt')

NotADirectoryError: [Errno 20] Not a directory:
'docasny/test.txt'

os.rmdir('docasny')
OSError: [Errno 39] Directory not empty: 'docasny'
```

`numpy`

`numpy` je knižnica ja vedecké výpočty v Pythone

V princípe funguje podobne ako MATLAB

Podporuje:

- vektory a matice
- lineárnu algebru (riešenie systémov rovníc)
- vykresľovanie (cez knižnicu `matplotlib`)

Importovanie knižnice `numpy`

Dlhý zápis

```
import numpy
vektor = numpy.array(...)
```

Skrátený zápis

```
import numpy as np
vektor = np.array(...)
```

Vektory a matice

Sú reprezentované triedou **`array`**

Vektory aj matice sú definované ako zoznam zoznamov

- vonkajší zoznam definuje riadky
- vnútorný zoznam definuje stĺpce

Riadkový vektor 1x3:

```
v1 = np.array( [ [1., 2., 3.] ] )
```

Stĺpcový vektor 3x1:

```
v2 = np.array( [ [1.], [2.], [3.] ] )
```

Matica 2x3:

```
m = np.array( [ [1., 2., 3.], [4., 5., 6.] ] )
```

Vypisovanie vektorov a matíc

Realizuje sa cez funkciu `print()` alebo metódu `format()`

```
import numpy as np
v1 = np.array([[1., 2., 3.]])
v2 = np.array([[1.], [2.], [3.]])
m = np.array([ [1., 2., 3.], [4., 5., 6.] ])
```

```
print(v1)
print('Vektor je %s' % (v2) )
print('Matica je {}'.format(m))
```

Zistenie veľkosti

Atribút `shape` obsahuje riadkovú a stĺpcovú veľkosť, vracia dvojicu, ktorú je možné "rozbaliť":

```
r, s = v1.shape
print('v1 ma {} riadkov a {} stlpcov'.format(r,s))
```

Atribút `size` obsahuje počet prvkov (riadky*stĺpce):

```
pocet_prvkov = m.size
print('Matica ma %d prvkov' % pocet_prvkov)
```

Tvorba špeciálnych matíc

Nulovú maticu vytvoríme funkciou `zeros()`,
vstupom je **dvojica** (počet riadkov a počet stĺpcov)

```
nulova = np.zeros( (3, 4) )
```

Maticu jednotiek vytvoríme funkciou `ones()`

```
jednotky = np.ones( (2, 1) )
```

Jednotkovú maticu vytvoríme funkciou `eye()`

```
I = np.eye(3)
```

Indexovanie a rezanie

Vektory a matice indexujeme ako zoznamy cez hranaté zátvorky

Nezabudnite, že Python indexuje od nuly

```
M = np.array( [ [1., 2., 3.], [4., 5., 6.] ])
print(M[0, 0])
```

Posledný riadok/stĺpec majú index -1

```
print(M[-1, -1])
```

Vektor a maticu je možné "rozrezať" na časti:

```
M = np.array( [ [1., 2., 3.], [4., 5., 6.] ])
prvy_riadok = M[0, :]
print(prvy_riadok)
```

Kopírovanie

Priradenie `y=x` vytvára iba **odkaz**!

```
x = np.array( [ [1.] ] )
y = x
y[0, 0] = 2
```

Kópia vektora/matice sa spraví cez `copy()`:

```
y = x.copy()
y[0] = 3
```

Poznámka k indexovaniu a rezaniu

Pri indexovaní a rezaní sa vracia **pohľad** (view)

Pri zmene pohľadu sa mení aj pôvodný objekt:

```
M = np.array([ [ 1., 2., 3. ] ])
p = M[0, 1:3]
p[1] = 4
```

Ak chceme modifikovať iba pohľad, musíme ho kopírovať:

```
q = M[0, 1:3].copy()
q[1] = 5
```

Matematické operácie

Transpozícia vektora/matice je cez metódu `transpose()`:

```
M = np.array( [ [1., 2., 3.], [4., 5., 6.] ])
trans = M.transpose()
```

Skratka cez atribút `T`:

```
trans = M.T
```

Násobenie vektorov a matíc (musia sedieť rozmery):

```
M = np.array( [ [1., 2., 3.], [4., 5., 6.] ])
V = np.array( [ [1.], [2.], [3.] ])
Z = M.dot(V)
```

Operácie po prvkoch

Sčitovanie, odčitovanie, násobenie, delenie:

```
M1 = np.array( [ [1., 2., 3.], [4., 5., 6.] ])
M2 = np.array( [ [7., 8., 9.], [10., 11., 12.] ])
Q1 = M1+M2; Q2 = M1-M2; Q3 = M1*M2; Q4 = M1/M2
```

Sčitovanie, odčitovanie, násobenie, delenie so skalárom:

```
P1 = M1+2; P2 = M1-4; P3 = M1/5; P4 = M1*6
```

Umocnenie každého prvku:

```
Q5 = M1**3
```

Vyhľadávanie (vracia maticu obsahujúcu `True` a `False`):

```
idx = M1>3.5
```

Lineárna algebra

Je potrebné najskôr importovať knižnizu `numpy.linalg`:

```
import numpy.linalg as linalg
```

Inverzia regulérnej matice:

```
M = np.array( [ [1., 2. ], [5., 6.] ] )
I = linalg.inv(M)
```

Hodnosť matice:

```
r = linalg.matrix_rank(M)
```

Zistenie vlastných čísel a vektorov:

```
vlastne_cisla,vlastne_vektory = linalg.eig(A)
```

Riešenie systému rovníc

Ax=b (matica A štvorcová):

```
A = np.array( [ [1., 2. ], [5., 6.] ] )
b = np.array( [ [3.], [4.] ])
x = linalg.solve(A, b)
```

Ax=b (matica A neštvorcová):

```
x = linalg.lstsq(A, b)
```

Dôležité odkazy

Hlavná stránka:

- http://www.numpy.org

`numpy` pre používateľov MATLABu:

- https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html

Tutoriál pre `numpy`:

- https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

`matplotlib`

`matplotlib` je knižnica ja vykresľovanie grafov v Pythone

Akceptuje dáta reprezentované ako `array` z `numpy`

Podporuje podobné vlastnosti grafov ako MATLAB

Vektory v `numpy`

Doteraz sme používali dvojrozmerové vektory:

```
riadkovy = np.array( [ [1., 2., 3.] ] )
stlpcovy = np.array( [ [1.], [2.], [3.] ] )
```

`numpy` podporuje aj jednorozmerové vektory:

```
vektor = np.array( [1., 2., 3.] )
```

Jednorozmerový vektor z matice/riadkového vektora:

```
vektor = riadkovy[0, :]
```

Jednorozmerové vektory nemenia svoj rozmer pri transpozícii:

```
print(vektor.T)  ->   [ 1.  2.  3.]
```

Skalárny súčin si automaticky prispôsobí rozmery:

```
print(vektor.dot(vektor))  ->  14
```

Matematické funkcie v `numpy`

Postupnosť pomocou `range()`:

```
x = np.array( range(10) )
x = np.array( range(2, 10) )
x = np.array( range(2, 10, 4) )
```

Postupnosť cez `numpy.arange(start, stop, krok)`:

```
x = np.arange(10.0)
x = np.arange(2.0, 10.0)
x = np.arange(2.0, 10.0, 0.1)
```

Matematické funkcie v `numpy`

Trigonometrické funkcie (sin, cos, tan, arcsin, arccos, arctan):

```
x = np.array( [1., 2., 3.] )
y = np.sin(x)
```

Prevod stupňov na radiány:

```
s = np.deg2rad(x)
r = np.rad2deg(x)
```

Zaokrúhl'ovanie:

```
z = around(x)
z = around(x, 3)
z = floor(x)
z = ceil(x)
z = trunc(x)
```

Matematické funkcie v `numpy`

Suma prvkov vektora:

```
y = np.sum(x)
```

Kumulovaná suma (vracia vektor):

```
y = np.cumsum(x)
```

Produkt prvkov vektora (vracia skalár $x_1*x_2*...*x_n$):

```
y = np.prod(x)
```

Diferencia prvkov vektora (vracia vektor $[x_2-x_1, x_3-x_2, ..., x_n-x_{n-1}]$):

```
y = np.diff(x)
```

Matematické funkcie v `numpy`

Exponenty:

```
z = np.exp(x)
```

Logaritmy:

```
z = np.log(x)
z = np.log10(x)
z = np.log2(x)
```

Rôzne:

```
z = np.sqrt(x)
z = np.cbrt(x)
z = np.absolute(x)
z = np.maximum(x)
z = np.minimum(x)
```

Knižnica `matplotlib`

Importovanie knižnice:

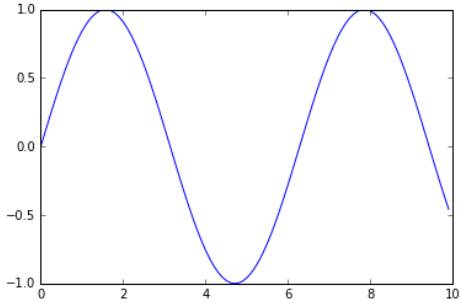
```
import matplotlib.pyplot as plt
```

Vstupom na vykreslovanie musia byť jednorozmerové vektory:

```
x = np.arange(0, 10, 0.1)
y = np.sin(x)
```

Základné vykreslenie:

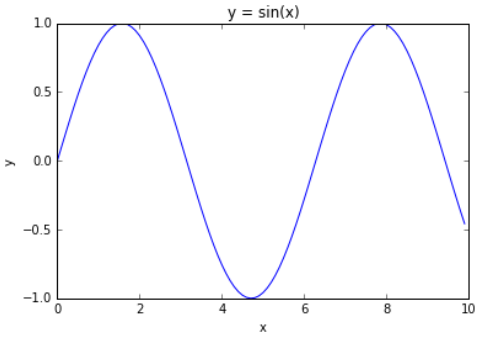
```
plt.plot(x, y)
plt.show()
```



Knižnica `matplotlib`

Označenie osí a pridanie titulku:

```
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('y = sin(x)')
plt.show()
```

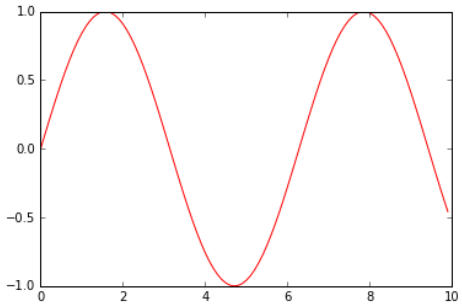


Všetko musí byť realizované ešte pred `plt.show()`

Knižnica `matplotlib`

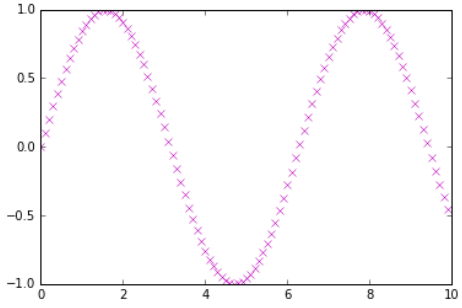
Zmena farby čiary:

```
plt.plot(x, y, 'r')
plt.show()
```



Vykresľovanie po bodoch:

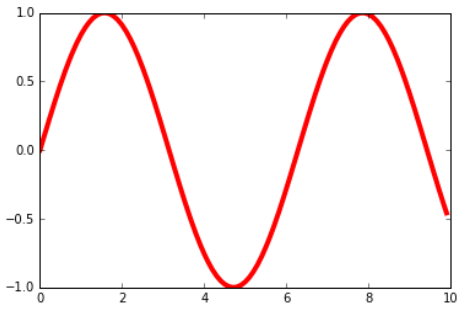
```
plt.plot(x, y, 'mx')
plt.show()
```



Knižnica `matplotlib`

Zmena hrúbky čiary:

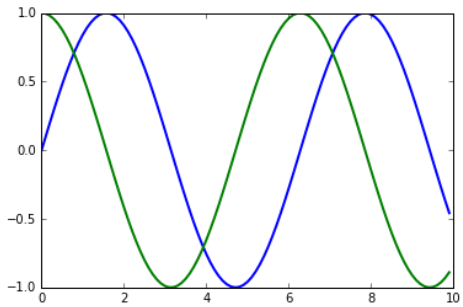
```
plt.plot(x, y, color='r', linewidth=4)
plt.show()
```



Knižnica `matplotlib`

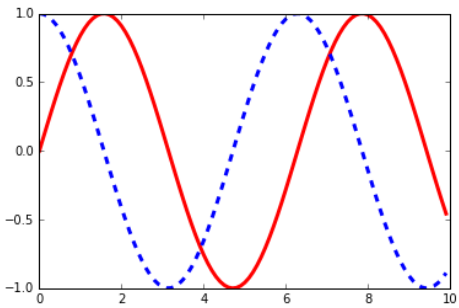
Vykreslenie viacerých dát:

```
y1 = np.sin(x)
y2 = np.cos(x)
plt.plot(x, y1, x, y2)
plt.show()
```

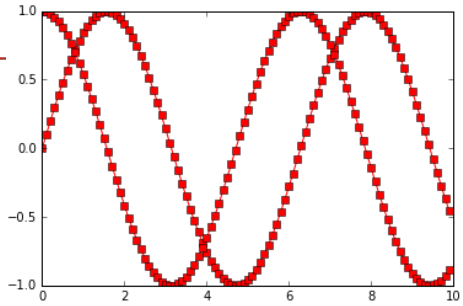


Farbu je možné špecifikovať zvlášť, hrúbku iba spoločne:

```
plt.plot(x, y1, 'r', x, y2, 'b--', linewidth=3)
plt.show()
```



Knižnica matplotlib

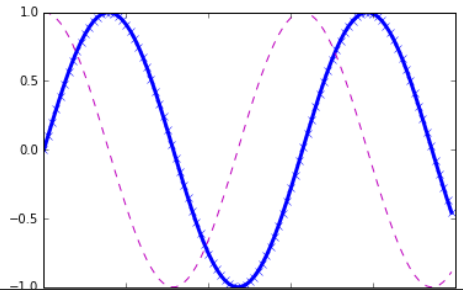


Dodatočná zmena vlastností:

```
y1 = np.sin(x)
y2 = np.cos(x)
ciary = plt.plot(x, y1, x, y2)
plt.setp(ciary, color='r', marker='s')
plt.show()
```

Individuálna zmena:

```
plt.setp(ciary[0], marker='x', linewidth=3)
plt.setp(ciary[1], linestyle='--', color='m')
```



Knižnica matplotlib

Výpis podporovaných vlastností:

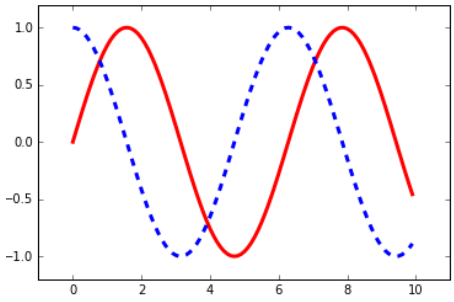
```
print(plt.setp(ciary))
```

```
  alpha: float (0.0 transparent through 1.0 opaque)
  animated: [True | False]
  antialiased or aa: [True | False]
  axes: an :class:`~matplotlib.axes.Axes` instance
  clip_box: a :class:`matplotlib.transforms.Bbox` instance
  clip_on: [True | False]
  clip_path: [ (:class:`~matplotlib.path.Path`, :class:`~matplotlib.transforms.Transform`)
| :class:`~matplotlib.patches.Patch` | None ]
  color or c: any matplotlib color
  contains: a callable function
  dash_capstyle: ['butt' | 'round' | 'projecting']
  dash_joinstyle: ['miter' | 'round' | 'bevel']
  dashes: sequence of on/off ink in points
  drawstyle: ['default' | 'steps' | 'steps-pre' | 'steps-mid' | 'steps-post']
  figure: a :class:`matplotlib.figure.Figure` instance
  fillstyle: ['full' | 'left' | 'right' | 'bottom' | 'top' | 'none']
  gid: an id string
  label: string or anything printable with '%s' conversion.
  linestyle or ls: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) | ```-```
| ```--``` | ```-.``` | ```:``` | ```None``` | ``` ``` | ```'']
  linewidth or lw: float value in points
  marker: :mod:`A valid marker style <matplotlib.markers>`
  markeredgecolor or mec: any matplotlib color
  markeredgewidth or mew: float value in points
  markerfacecolor or mfc: any matplotlib color
  markerfacecoloralt or mfcalt: any matplotlib color
  markersize or ms: float
```

Knižnica matplotlib

Zmena osí:

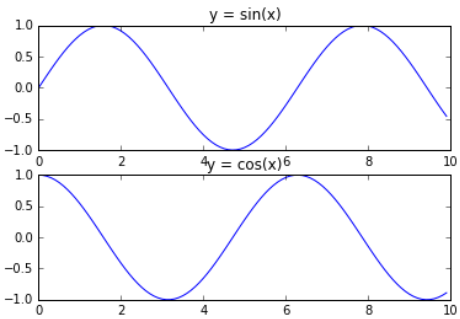
```
y1 = np.sin(x)
y2 = np.cos(x)
plt.plot(x, y1, x, y2)
plt.axis([-1, 11, -1.2, 1.2])
plt.show()
```



Knižnica matplotlib

Subploty:

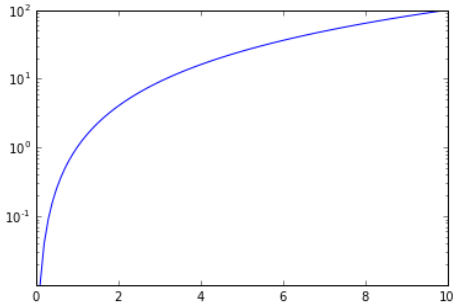
```
plt.figure(1)
plt.subplot(211)
plt.plot(x, np.sin(x))
plt.title('y = sin(x)')
plt.subplot(212)
plt.plot(x, np.cos(x))
plt.title('y = cos(x)')
plt.show()
```



Knižnica `matplotlib`

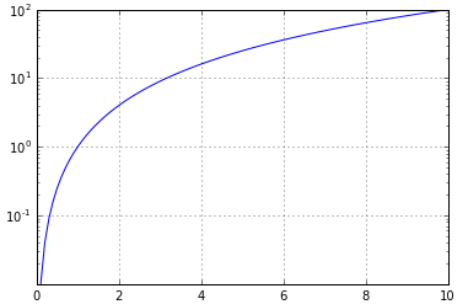
Logaritmické osy:

```
plt.plot(x, x**2)
plt.xscale('linear')
plt.yscale('log')
plt.show()
```



Zapnutie gridu:

```
plt.plot(x, x**2)
plt.xscale('linear')
plt.yscale('log')
plt.grid(True)
plt.show()
```



Triedenie v `numpy`

Zotriedenie prvkov vektora:

```
v = np.array([4.2, 1.8, 0.3, 3.2])
z = np.sort(v) # [0.3, 1.8, 3.2, 4.2]
```

Triedenie v opačnom poradí:

```
v = np.array([4.2, 1.8, 0.3, 3.2])
z = np.sort(v)    # [0.3, 1.8, 3.2, 4.2]
opacne = z[::-1] # [4.2, 3.2, 1.8, 0.3]
```

Zistenie indexov:

```
v = np.array([4.2, 1.8, 0.3, 3.2])
i = np.argsort(v)  # [2 1 3 0]
```

Vyhľadávanie v `numpy`

Minimum a maximum vektora:

```
v = np.array([4.2, 1.8, 0.3, 3.2])
max = np.max(v) # 4.2
min = np.min(v) # 0.3
```

Zistenie indexov:

```
v = np.array([4.2, 0.0, -1.8, 4.2, 0.0])
imax = np.argmax(v)       # 0
imin = np.argmin(v)       # 2
inonzero = np.nonzero(v) # [0, 2, 3]
```

Načítavanie dát z textového súboru

Formát načítavaného súboru:

```
# viacriadkovy komentar
# sa preskoci
1.0 2.5 3.2
2.0 -1.5 0.5
3.0 0.3 0.2
...
```

Načítanie dát do matice:

```
data = np.loadtxt(nazov)
data = np.loadtxt(nazov, delimiter=', ')
data = np.loadtxt(nazov, comments='#')
data = np.loadtxt(nazov, skiprows=[1, 4, 5])
```

Načítavanie dát z textového súboru

Formát načítavaného súboru:

```
# viacriadkovy komentar
# sa preskoci
1.0 2.5 3.2
2.0 -1.5 0.5
3.0 0.3 0.2
...
```

Načítané stĺpce sa ukladajú ako riadky matice!

```
x = data[0, :] # prvy stlpec, hoci indexujeme riadok
y = data[1, :] # druhy stlpec, hoci indexujeme riadok
```

Načítanie dát do vektorov po stĺpcoch:

```
x, y, z = np.loadtxt(nazov, unpack=True)
```

Zápis matice do textového súboru

Matice sa do súboru zapisujú po riadkoch:

```
M = np.array([ [1., 2., 3.], [4., 5., 6.]])
np.savetxt('matica.txt', M)
```

Voľby:

```
np.savetxt('matica.txt', M, fmt='%10.5f')
np.savetxt('matica.txt', M, delimiter=',')
```

Zobrazenie obsahu súboru:

```
f = open('matica.txt', 'r')
print(f.read())
f.close()
```

Modul `numpy.polynomial.polynomial`

Poskytuje funkcie a objekty na prácu s polynómami:

- hodnota polynómu: `polyval(x, coeffs)`
- korene polynómu: `polyroots(coeffs)`
- polynóm z koreňov: `polyfromroots(roots)`
- derivácia: `polyder(coeffs)`
- integrál: `polyint(coeffs)`
- algebra polynómov (+, -, *, /, ^)
- fitovanie dát polynómom: `polyfit(x, y, deg)`
- objekt reprezentujúci polynóm: `Polynomial(coeffs)`

Importovanie knižnice:

```
import numpy.polynomial.polynomial as poly
```

Hodnota polynómu: `polyval(x, coeffs)`

Hodnota polynómu $a_3s^3+a_2s^2+a_1s+a_0$ v bode $s=x$:

```
koeficienty = np.array([a0, a1, a2, a3])
hodnota = poly.polyval(x, koeficienty)
```

Koeficienty môžu byť aj obyčajný zoznam:

```
hodnota = poly.polyval(x, [a0, a1, a2, a3])
```

Pozor, v `numpy` idú koeficienty od najmenšej mocniny po najväčšiu, Matlab to má naopak:

```
>> hodnota = polyval([a3 a2 a1 a0], x)
```

Vyhodnocovať môžeme aj vektor:

```
vektor = [1.1, 2.3] # môže byť aj np.array([...])
hodnoty = poly.polyval(vektor, koeficienty)
```

Korene polynómu: `polyroots(coeffs)`

Korene polynómu s^2+4s+2:

```
korene = poly.polyroots([2, 4, 1])
[-3.41421356 -0.58578644]
```

Korene polynómu s^2+s+1:

```
korene = poly.polyroots([1, 1, 1])
[-0.5-0.8660254j -0.5+0.8660254j]
```

Prístup k reálnej a imaginárnej zložke:

```
real1 = korene[0].real
imag1 = korene[0].imag
```

Je číslo komplexné? (podporuje aj vektory):

```
je_komplexne = np.iscomplex(korene)
[ True  True]
```

Polynóm z koreňov: `polyfromroots(roots)`

Polynóm, ktorý má korene -3, 1.5 a 2.8:

```
p = poly.polyfromroots([-3, 1.5, 2.8])
[ 12.6  -8.7  -1.3   1. ] # 12.6-8.7*s-1.3*s^2+s^3
```

Poznámka: polynóm je vždy znormovaný tak, že posledný koeficient pri najvyššej mocnine je 1

Polynóm s koreňmi -1, 0 a 1:

```
p = poly.polyfromroots([-1, 0, 1])
[ 0. -1.  0.  1.] # = -x+x^3
```

Komplexné korene *-i* a *+i*:

```
i = complex(0,1)
p = poly.polyfromroots([-i, i])
[ 1.+0.j  0.+0.j  1.+0.j ] # nulove imag. casti
```

Derivácia polynómu: `polyder(coeffs)`

Prvá derivácia polynómu $3x^2 + 2x + 1$:

```
p = poly.polyder([1, 2, 3])
[ 2.  6.] # = 2+6x
```

Druhá derivácia polynómu $x^3 + 2x^2 + 3x + 4$:

```
stupen = 2
p = poly.polyder([4, 3, 2, 1], stupen)  # 4+6x
```

Integrovanie polynómu: `polyint(coeffs)`

Integrál polynómu $3x^2 + 2x + 1$:

```
p = poly.polyint([1, 2, 3])
[ 0.  1.  1.  1.] # = x+x^2+x^3
```

Viacnásobný integrál polynómu x^3+2x:

```
stupen = 2
p = poly.polyint([0, 2, 0, 1], stupen)
[ 0. 0. 0. 0.33 0. 0.05 ] # = 0.33x^3 + 0.05x^5
```

Algebra polynómov

Sčitovanie dvoch polynómov, napr. $p_1=3x^2+1$, $p_2=x-2$

```
p1 = [1, 0, 3]; p2 = [-2, 1]
p = poly.polyadd(p1, p2)
[-1.  1.  3.] # = -1+x+3x^2
```

Odčitovanie:

```
p = poly.polysub(p1, p2)
[3. -1. 3.] # = 3-x+3x^2
```

Násobenie:

```
p = poly.polymul(p1, p2)
[-2.  1. -6.  3.] # = -2+x-6x^2+3x^3
```

Delenie:

```
podiel, zvysok = poly.polydiv(p1, p2)
```

Algebra polynómov

Delenie $p_1=3x^2+1$, $p_2=x-2$

```
p1 = [1, 0, 3]; p2 = [-2, 1]
podiel, zvysok = poly.polydiv(p1, p2)
[ 6.  3.]  # podiel
[ 13.]     # zvysok po deleni
```

Umocnenie polynómu:

```
mocnina = 3
p = poly.polypow(p2, mocnina)
[ -8.  12.  -6.   1.] # -8+12x-6x^2+x^3
```

Fitovanie dát polynómom: `polyfit(x,y,deg)`

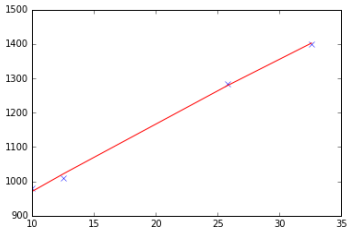
Fitovací polynóm je v tvare $p(x) = c_0+c_1x+...+c_nx^n$

Pri fitovaní sa minimalizuje $\sum (y_j - p(x_j))^2$

x_j sú experimenty (napr. teplota) a y_j ich výsledky (napr. vodivosť)

Vstupom môžu byť zoznamy alebo `numpy.array`:

```
x = [10.0, 12.5, 25.8, 32.6]
y = np.array([980, 1010, 1284.5, 1400])
p = poly.polyfit(x, y, 2) # 760+21.6x-0.006x^2
plt.plot(x, y, 'x', x, poly.polyval(x, p))
plt.show()
```



`Polynomial(coeffs)`

Zjednodušená reprezentácia koeficientov:

```
p = poly.Polynomial([1, 2, 0, 3]) # 1+2x+3x^3
```

Vyhodnotenie polynómu:

```
h = p(2.5)          # 52.875
h = p([2.5, 4.6]) # [  52.875  302.208]
```

Zistenie stupňa polynómu: `d = p.degree()`

Korene polynómu: `r = p.roots()`

Derivovanie: `d = p.deriv()` alebo `d = p.deriv(stupen)`

Integrovanie: `i = p.integ()` alebo `i = p.integ(stupen)`

Polynomial(coeffs)

Algebra:

```
p1 = poly.Polynomial([1, 2, 0, 3]) # 1+2x+3x^3
p2 = poly.Polynomial([0, 1])       # x
s = p1+p2                          # x+3x+3x^3
r = p1-p2                          # 1+x+3x^3
n = p1*p2                          # x+2x^2+3x^4
```

Vykreslenie polynómu:

```
x, y = p.linspace(pocet_bodov, [xmin, xmax])
plt.plot(x, y)
```

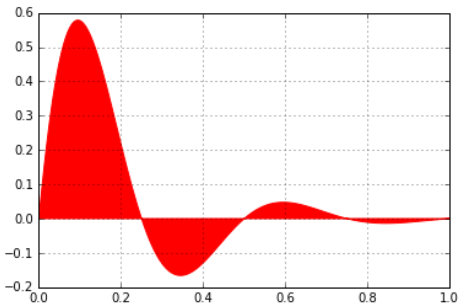
Alternatíva:

```
x, _ = p.linspace(pocet_bodov, [xmin, xmax])
plt.plot(x, p(x))
```

2D grafy

Graf s výplňou:

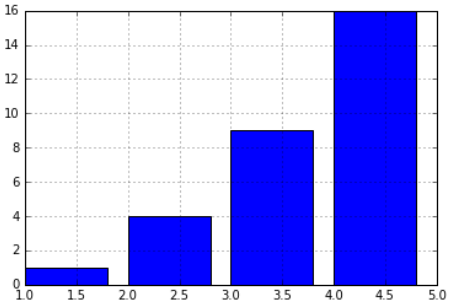
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 1, 500)
y = np.sin(4 * np.pi * x) * np.exp(-5 * x)

plt.fill(x, y, color='r')
plt.grid(True)
plt.show()
```



2D grafy

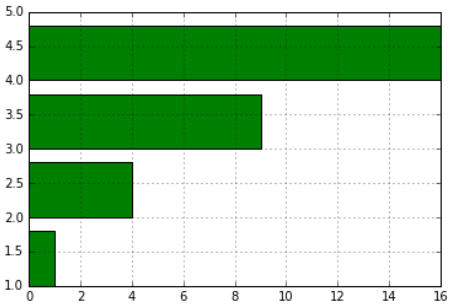
Stĺpcový graf (vertikálne stĺpce):

```
x = np.array([1, 2, 3, 4])
y = x**2
plt.bar(x, y)
plt.show()
```



Stĺpcový graf (horizontálne stĺpce):

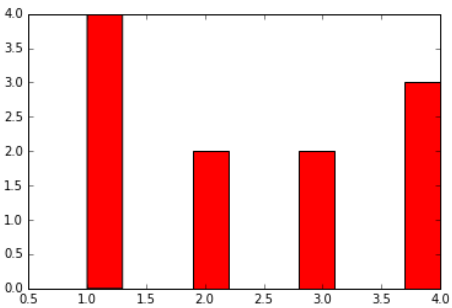
```
x = np.array([1, 2, 3, 4])
y = x**2
plt.barh(x, y, color='g')
plt.show()
```



2D grafy

Histogram:

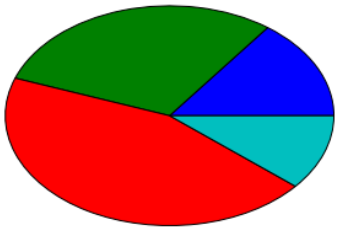
```
x = np.array([1, 2, 3, 4, 1, 1, 2, 1, 3, 4, 4])
plt.hist(x, color='r')
plt.show()
```



2D grafy

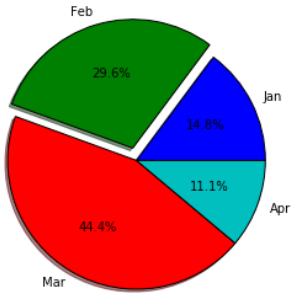
Koláčový graf:

```
x = [4, 8, 12, 3]
plt.pie(x)
plt.axis('equal')
plt.show()
```



Ďalšie možnosti:

```
kategorie = ['Jan', 'Feb', 'Mar', 'Apr']
plt.pie(x, labels=kategorie,
           explode= [0, 0.1, 0, 0],
           shadow=True,
           autopct='%1.1f%%')
plt.axis('equal')
plt.show()
```



Tvorba mriežky cez `meshgrid`

Rovnaké volanie ako v Matlabe:

```
x = np.arange(-5, 5, 0.1)
y = np.arange(-5, 5, 0.1)
xx, yy = np.meshgrid(x, y)
```

Generuje matice:

```
[[-5.  -4.9 -4.8 ...,  4.7  4.8  4.9]
 [-5.  -4.9 -4.8 ...,  4.7  4.8  4.9]
 [-5.  -4.9 -4.8 ...,  4.7  4.8  4.9]
 ...,
 [-5.  -4.9 -4.8 ...,  4.7  4.8  4.9]
 [-5.  -4.9 -4.8 ...,  4.7  4.8  4.9]
 [-5.  -4.9 -4.8 ...,  4.7  4.8  4.9]]
```

Práca s mriežkou

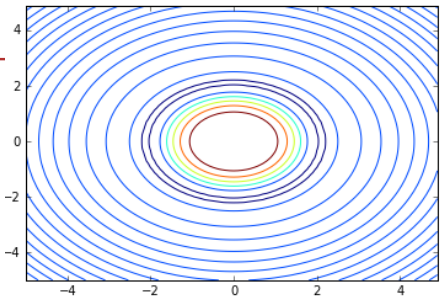
Rovnaké volanie ako v Matlabe:

```
x = np.arange(-5, 5, 0.1)
y = np.arange(-5, 5, 0.1)
xx, yy = np.meshgrid(x, y)
```

Prvky matíc vieme indexovať:

```
zz = np.zeros(xx.shape)                 # nulova matica
for i = range(xx.shape[0]):             # po riadkoch
    for j = range(xx.shape[1]):         # po stlpcoch
        zz[i, j] = xx[i, j]*yy[i, j]    # zz=xx.*yy
```

Vrstevnicový graf

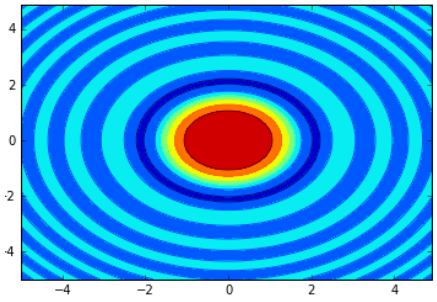


Rovnaké volanie ako v Matlabe:

```
x = np.arange(-5, 5, 0.1)
y = np.arange(-5, 5, 0.1)
xx, yy = np.meshgrid(x, y)
zz = np.sin(xx**2 + yy**2) / (xx**2 + yy**2)
plt.contour(xx, yy, zz)
```

Prípadne s výplňou:

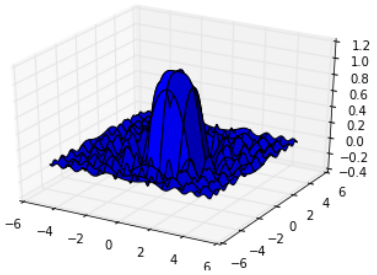
```
plt.contourf(xx, yy, zz)
```



3D graf

Importovanie knižníc:

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
```

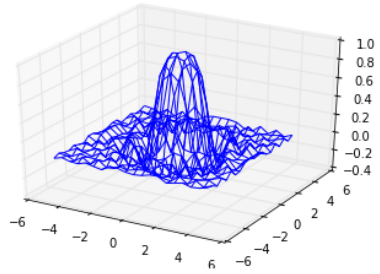


Príprava nového obrázka:

```
fig = plt.figure()
ax = fig.gca(projection='3d')
```

Vykreslenie (rovnaký formát ako `surf` v Matlabe):

```
ax.plot_surface(xx, yy, zz)
```



Prípadne bez výplne:

```
ax.plot_wireframe(xx, yy, zz)
```

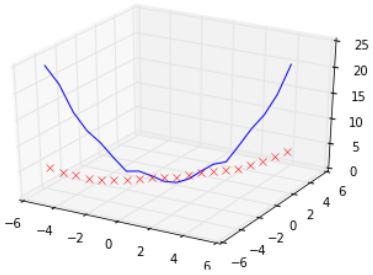
3D čiary a body

Rovnaké volanie ako `plot3` v Matlabe:

```
plt.plot(x, y, z)
```

Predpríprava:

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
ax = plt.gca(projection='3d')
```



Príklad:

```
x = np.arange(-5, 5, 0.5)
y = np.arange(-5, 5, 0.5)
plt.plot(x, y, x*y+np.sin(x**2))
plt.plot(x, y, np.abs(x)+np.cos(y), 'rx')
plt.show()
```

Export obrázkov do súboru:

Exportovanie aktuálneho obrázka:

```
plt.savefig(nazov)
```

Typ sa automaticky určí podľa prípony (.png, .jpg, atď)

Príklad:

```
x = np.arange(-5, 5, 0.5)
y = np.arange(-5, 5, 0.5)
ax = plt.gca(projection='3d')
plt.plot(x, y, x*y+np.sin(x**2))
plt.savefig('obrazok.png')
```

Knižnica `scipy`

Knižnica na vedecké a inžinierske výpočty

Podknižnice:

- interpolácia (`scipy.interpolate`)
- optimalizácia (`scipy.optimize`)
- filtrácia signálov a dynamické systémy (`scipy.signal`)
- numerická integrácia (`scipy.integrate`)
- štatistika (`scipy.stats`)
- ...

Knižnica `scipy.interpolate`

Základná funkcia `interp1d` interpoluje dáta splinom

https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.interpolate.interp1d.html

Knižnica `scipy.interpolate`

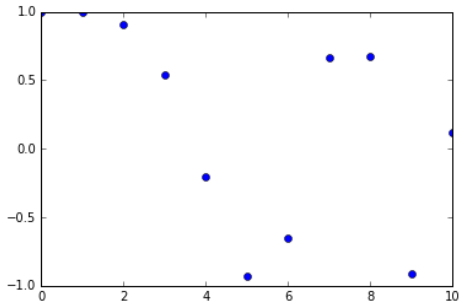
Základná funkcia `interp1d` interpoluje dáta splinom

Importovanie funkcie z knižnice:

```
from scipy.interpolate import interp1d
```

Príklad:

```
x = np.linspace(0, 10, 11)
y = np.cos(-x**2/9.0)
plt.plot(x, y, 'o')
```



https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.interpolate.interp1d.html

Knižnica `scipy.interpolate`

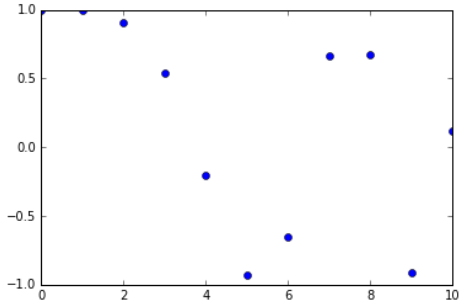
Základná funkcia `interp1d` interpoluje dáta splinom

Importovanie funkcie z knižnice:

```
from scipy.interpolate import interp1d
```

Príklad:

```
x = np.linspace(0, 10, 11)
y = np.cos(-x**2/9.0)
plt.plot(x, y, 'o')
f1 = interp1d(x, y)
f2 = interp1d(x, y, kind=2)
```



https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.interpolate.interp1d.html

Knižnica `scipy.interpolate`

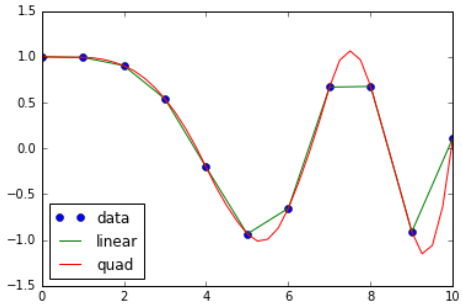
Základná funkcia `interp1d` interpoluje dáta splinom

Importovanie funkcie z knižnice:

```
from scipy.interpolate import interp1d
```

Príklad:

```
x = np.linspace(0, 10, 11)
y = np.cos(-x**2/9.0)
plt.plot(x, y, 'o')
f1 = interp1d(x, y)
f2 = interp1d(x, y, kind=2)
xn = np.linspace(0, 10, 41)
plt.plot(xn, f1(xn), xn, f2(xn))
plt.legend(['data', 'linear', 'quad'], loc='best')
```



https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.interpolate.interp1d.html

Knižnica `scipy.optimize`

Základné funkcie

- hľadanie lokálneho minima (rýchle): `minimize()`
- hľadanie globálneho minima (pomalé): `basinhopping()`
- lineárne programovanie: `linprog()`
- fitovanie ľubovolnou funkciou: `curve_fit()`

Importovanie danej funkcie z knižnice:

```
from scipy.optimize import minimize
```

Importovanie všetkých funkcií z knižnice:

```
from scipy.optimize import *
```

Importovanie s prefixom:

```
import scipy.optimize as opt
```

Hľadanie lokálneho minima: `minimize()`

Hľadá lokálne optimum problému *min f(x) v.n.* $g_i(x) \geq 0$, $h_j(x) = 0$

Základné volanie pri optimalizácii bez ohraničení:

```
vysledok = minimize(funkcia, x0)
```

Príklad:

```
def ucelovka(x):
    return (x[0]-2.0)**2 + (x[1]+3.0)**2
x0 = [0.0, 0.0]
vysledok = minimize(ucelovka, x0)
print(vysledok)
optimum = vysledok.x
```

```
      fun: 4.3136529750579557e-16
 message: 'Optimization terminated successfully.'
     nfev: 16
      nit: 2
     njev: 4
   status: 0
  success: True
        x: array([ 1.99999998, -3.00000001])
```

https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.optimize.minimize.html

Hľadanie lokálneho minima: `minimize()`

Hľadá lokálne optimum problému *min f(x) v.n.* $g_i(x) \geq 0$, $h_j(x) = 0$

Optimilizácia s ohraničeniami:

```
vysledok = minimize(funkcia, x0, constraints=ohr)
```

Príklad:

```
def nerovnost(x):
    return x[0] - x[1]    # x_1-x_2 >= 0
def rovnost(x):
    return 2*x[0] - x[1] # 2*x_1-x_2 == 0
ohr = [{'type': 'ineq', 'fun': nerovnost },
       {'type': 'eq', 'fun': rovnost}]
x0 = [0.0, 0.0]
vysledok = minimize(ucelovka, x0, constraints=ohr)
print(vysledok.x) # [-0.8 -1.6]
```

https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.optimize.minimize.html

Hľadanie globálneho minima: `basinhopping()`

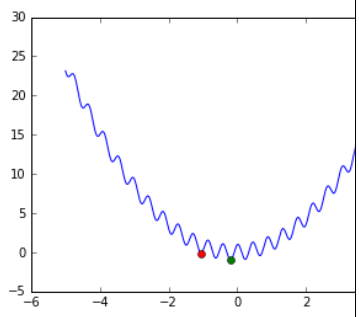
Hľadá globálne optimum funkcie *f(x)* bez ohraničení

Základné volanie:

```
vysledok = basinhopping(funkcia, x0, niter=100)
```

Príklad:

```
def f(x):
    return np.cos(14.5*x-0.3)+(x+0.2)*x
x0 = 2.5
lok = minimize(f, x0)
glob = basinhopping(f, x0)

x = np.linspace(-5, 5, 1000)
plt.plot(x, f(x), lok.x, lok.fun, 'ro',
         glob.x, glob.fun, 'go')
```



https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.optimize.basinhopping.html

Lineárne programovanie: `linprog()`

Nájde globálne optimum problému *min* $c^T x$ *v.n.* $Ax \leq b$, $Cx = d$

Základné volanie:

```
vysledok = linprog(c, A_ub=A, b_ub=b, A_eq=C, b_eq=d)
```

Príklad:

```
c = [1, -4] # moze byt aj np.array(...)
A = [ [-3, 1], [1, 2], [0, -1] ]
b = [6, 4, 3]
vysledok = linprog(c, A_ub=A, b_ub=b)
print(vysledok)
```

```
     fun: -8.0
 message: 'Optimization terminated successfully.'
     nit: 1
   slack: array([ 4.,  0.,  5.])
  status: 0
 success: True
       x: array([ 0.,  2.])
```

https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linprog.html

Fitovanie ľubovoľnou funkciou: `curve_fit()`

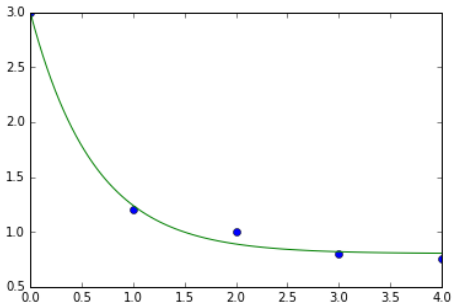
Hľadá parametre funkcie f(x, p) také, že *min* $\sum [f(x_i,p) - y_i]^2$

Základné volanie:

```
popt, kovariancia = curve_fit(f, x, y)
```

Príklad:

```
def funkcia(x, a, b, c):
    return a*np.exp(-b*x)+c

x = [0.0, 1.0, 2.0, 3.0, 4.0]
y = [3.0, 1.2, 1.0, 0.8, 0.75]
p, c = curve_fit(funkcia, x, y)
xn = np.linspace(0, 4, 100)
plt.plot(x, y, 'o', xn, funkcia(xn, *p))
```



https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.optimize.curve_fit.html

Knižnica `scipy.signal`

Obsahuje funkcie na filtráciu signálov a prácu s dynamickými systémami

Importovanie knižnice:

```
from scipy import signal
```

Podporované typy systémov:

- prenosové funkcie (`signal.TransferFunction`)
- stavové opisy (`signal.StateSpace`)
- reprezentácia pomocou núl, pólov a zosilnenia (`signal.zpk`)

https://docs.scipy.org/doc/scipy-0.19.0/reference/tutorial/signal.html

Prenosové funkcie

Vytvorenie objektu reprezentujúceho prenosovú funkciu $H(s) = \frac{s^2 + 3s + 3}{s^2 + 2s + 1}$

```
citatel = [1, 3, 3]   # od najvyssej mocniny
menovatel = [1, 2, 1]
tf = signal.TransferFunction(citatel, menovatel)
```

Prístup k čitateľu a menovateľu:

```
cit = tf.num      # priamy pristup cez atribut
men = tf.den      # nie je to volanie funkcie!
```

Výpočet núl a pólov:

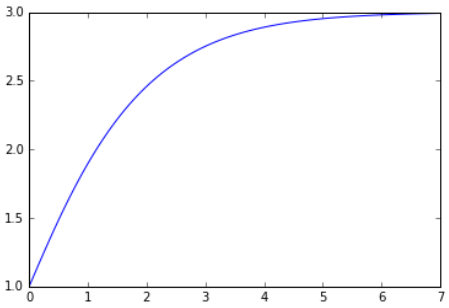
```
poly = tf.poles   # priamy pristup cez atribut
nuly = tf.zeros   # nie je to volanie funkcie!
```

https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.TransferFunction.html

Prenosové funkcie

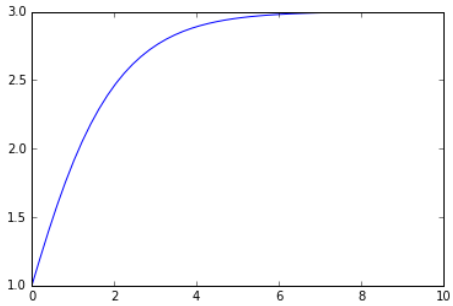
Získanie prechodovej charakteristiky ako odozvy na jednotkový skok:

```
t, y = signal.step(tf)
plt.plot(t, y)
```



Voliteľné špecifikovanie časového rozsahu:

```
cas = np.linspace(0, 10)
t, y = signal.step(tf, T=cas)
plt.plot(t, y)
```

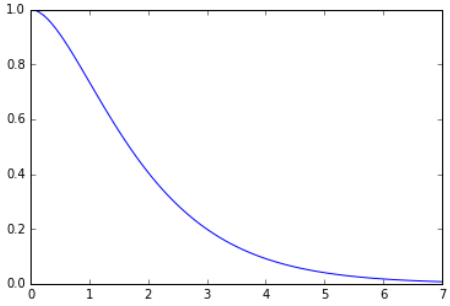


https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.step.html

Prenosové funkcie

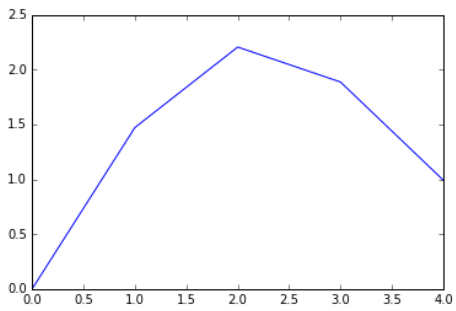
Získanie impulznej prechodovej charakteristikv:

```
t, y = signal.impulse(tf)
plt.plot(t, y)
```



Odozva na špecifické vstupy:

```
U = [0.0, 1.0, 1.0, 0.5, 0.0]
T = [0, 1, 2, 3, 4]
tout,y,x = signal.lsim(tf, U, T)
plt.plot(tout, y)
```



https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.impulse.html
https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.lsim.html

Stavové opisy

Rovnice stavového opisu:

- stavová rovnica: dx = Ax + Bu
- výstupná rovnica: y = Cx + Du

Vytvorenie objektu:

```
A = [ [0, 1], [0, 0] ]
B = [ [0], [1] ]
C = [ [1, 0] ]
D = [ [0] ]
sys = signal.StateSpace(A, B, C, D)
```

https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.StateSpace.html

Stavové opisy

Prístup k maticiam:

- `sys.A, sys.B, sys.C, sys.D`

Konverzie objektov:

- stavový opis na prenos: `tf = signal.TransferFunction(sys)`
- prenos na stavový opis: `sys = signal.StateSpace(tf)`

Simulácie:

- skoková prechodová charakteristika: `signal.step(sys)`
- impulzná prechodová charakteristika: `signal.impulse(sys)`
- simulácia s danými vstupmi: `signal.lsim(sys, U, T)`

https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.StateSpace.html

Knižnica `scipy.integrate`

Obsahuje funkcie na numerické integrovanie funkcií

Importovanie knižnice:

```
from scipy import integrate
```

Základné volanie pri integrovaní funkcie *f* v hraniciach *[a, b]*:

```
integral, chyba = integrate.quad(f, a, b)
```

Príklad:

```
def kvadrat(x):
    return x**2
integral, chyba = integrate.quad(kvadrat, 1.5, 5)
```

https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html

Zoznam literatúry

[1] Summerfield, M. (2014). Python 3. Albatros Media as.
[2] HORÁČKOVÁ, H. (2014). Výuka programování v jazyce Python pro střední školy.
[3] KAUKIČ, M. (2008). Python ako prvý programovací jazyk na VŠ. In 7th internacional conference. APLIMAT.
[4] Van Rossum, G., & Drake, F. L. (2003). Python language reference manual.
[5] Kiusalaas, J. (2013). Numerical methods in engineering with Python 3. Cambridge university press.
[6] Lutz, M. (2013). Learning python: Powerful object-oriented programming. " O'Reilly Media, Inc.".
[7] Cielen, D., Meysman, A., & Ali, M. (2016). Introducing data science: big data, machine learning, and more, using Python tools. Manning Publications Co.
[8] Martelli, A. (2006). Python in a Nutshell. " O'Reilly Media, Inc.".
[9] Oliphant, T. E. (2006). A guide to NumPy (Vol. 1, p. 85). USA: Trelgol Publishing.
[10] Bressert, E. (2012). SciPy and NumPy: an overview for developers. " O'Reilly Media, Inc.".
[11] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., ... & van Mulbregt, P. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. Nature methods, 17(3), 261-272.
[12] Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in science & engineering, 9(3), 90-95.
[13] Tosi, S. (2009). Matplotlib for Python developers. Packt Publishing Ltd.