Programovanie v jazyku Python

Údajové štruktúry v Pythone
prednáška 3

Katedra kybernetiky a umelej inteligencie
Technická univerzita v Košiciach
Ing. Ján Magyar, PhD.

Reťazce (stringy) v Pythone

- nemeniteľná postupnosť znakov
- ohraničené úvodzovkami alebo apostrofom

```
"Hello World!"          'Hello World!'
```

- definícia reťazcov na viacerých riadkoch je možná pomocou trojitých úvodzoviek/apostrofov (medzery sa stanú súčasťou stringu)

```
"""Hello                '''Hello
   World!"""               World!'''
```

- stringy vieme vytvárať aj pretypovaním, napr.

```
str(5)
```

Zakódovanie stringov

- v Python 3 zvyčajne nie je potrebné
- možnosť konvertovania stringov na bajty a späť
  - encode() - string → bajty
  - decode() - bajty → string

```
nonlat = '字'
b = nonlat.encode()          b'\xe5\xad\x97'
b.decode()                   '字'
```

Modifikácia stringov

- pridanie stringu na koniec existujúceho reťazca

```
test = "Hello"
test += " World!"
```

- konkatenácia stringov

```
test = "Hello" + " World!"
test = "Hello" " World!"
```

- pomocou funkcie `join`

```
test = ' '.join(["Hello", "World!"])
```

Stringy ako zoznamy

- Python považuje stringy za špeciálny prípad zoznamov
- k jednotlivým znakom môžeme pristupovať pomocou indexov

```
test = "Hello World!"
test[0] -> 'H'
```

- Python podporuje aj záporné indexovanie (od zadu)

```
test[-1] -> '!'
```

Stringy ako zoznamy - slicing

- pre získanie časti stringu vieme použiť slicing

```
test[2:7] -> 'llo W'
test[:7] -> 'Hello W'
test[2:] -> 'llo World!'
test[2:-3] -> 'llo Wor'
test[:] -> 'Hello World!'
```

- tretí parameter stringu reprezentuje krok (každý i-tý znak)

```
test[2:7:2] -> 'loW'
```

Určenie dĺžky stringu/postupnosti

- `len(string)`
- prázdny string má dĺžku `0`
- platné indexy v reťazci: od `0` po `len(string) - 1`

Vybrané metódy stringov - vyhľadávanie

- `find(sub[, start[, end]])` / `rfind(sub[, start[, end]])`
  - nájde prvý výskyt časti reťazca (`sub`), v intervale `string[start:end]`
  - vráti hodnotu `-1` ak `sub` sa nenachádza v reťazci
- `index(sub[, start[, end]])` / `rindex(sub[, start[, end]])`
  - nájde prvý výskyt časti reťazca (`sub`), v intervale `string[start:end]`
  - interpreter vyhodí chybu `ValueError` ak sa `sub` nenachádza v reťazci
- `count(sub[, start[, end]])`
  - vráti počet výskytov časti reťazca `sub` v intervale `string[start:end]`
  - pretínajúce sa výskyty sa nerátajú

Vybrané metódy stringov - rozdelenie

- `split([sep[, maxsplit]]) / rsplit([sep[, maxsplit]])`
  - rozdelí reťazec na časti podľa separátora (`sep`)
  - `maxsplit` určuje maximálny počet rozdelení (počet častí: `maxsplit + 1`)
  - separátor je defaultne whitespace
- `partition(sep) / rpartition(sep)`
  - rozdelí string na tri časti pri prvom výskyte separátora (`sep`)
  - návratová hodnota je trojica: časť stringu pred separátorom, separátor, časť po separátoru
  - ak sa separátor nenachádza v reťazci, vráti trojicu so stringom a dvoma prázdnymi reťazcami
- `splitlines([keepends])`
  - rozdelí reťazec na riadky
  - ako separátor používa rôzne znaky, ktoré môžu reprezentovať koniec riadku, nie iba znak `\n`
  - ak `keepends` je `True`, výsledné stringy obsahujú znak pre koniec riadku

Vybrané metódy stringov - úprava písmen

- `capitalize()`
  - vráti kópiu reťazca s veľkým začiatočným písmenom; ostatné písmená sú malé
- `title()`
  - vráti kópiu reťazca v ktorej každé slovo má veľké začiatočné písmeno
  - nevie spracovať apostrofy a pod., za slovo sa považuje skupina písmen
- `upper()`
  - vráti kópiu reťazca s veľkými písmenami*
- `lower()`
  - vráti kópiu reťazca s malými písmenami*
- `casefold()`
  - vráti kópiu reťazca s malými písmenami
  - špeciálne pre case-insensitive porovnanie

Vybrané metódy stringov - zarovnanie

- `lstrip([chars]) / rstrip([chars]) / strip([chars])`
  - vráti kópiu reťazca s odstránenými znakmi `chars` zo začiatku/konca
- `ljust(width[, fillchar]) / rjust(width[, fillchar]) / center(width[, fillchar])`
  - vráti reťazec s dĺžkou `width`, ktorý obsahuje pôvodný string zarovnaný doľava/doprava/v strede
  - `fillchar` je znak použitý na doplnenie prvkov - defaultne medzera
  - ak `width <= len(string)`, funkcia vráti pôvodný string
- `zfill(width)`
  - vráti kópiu reťazca s dĺžkou `width` reprezentujúcu číslo s pridanými nulami na začiatku (prípadne so znamienkom)
  - ak `width <= len(string)`, funkcia vráti pôvodný string

Vybrané metódy stringov - úprava

- `replace(old, new[, count])`
  - vráti kópiu stringu s vymenenými výskytmi časti `old` na `new`
  - `count` limituje počet vymenených výskytov
- `maketrans(x[, y[, z]])`
  - vytvorí tabuľku pre aktualizáciu niekoľkých znakov v stringu pre metódu `translate`
  - ak zadáme iba `x`, musí to byť dictionary mapujúci znaky na nové znaky
  - ak zadáme `x` a `y`, musia to byť stringy rovnakej dĺžky určujúce mapovanie
  - `z` je string znakov, ktoré budú vymazané z kópie
- `translate(table)`
  - vráti kópiu stringu s vymenenými znakmi

Vybrané metódy stringov - kontrola

- `endswith/startswith(sub[, start[, end]])`
  - vráti hodnotu `True` ak `string[start:end]` končí, resp. začína na `sub`
  - `sub` môže byť n-tica niekoľkých hľadaných podreťazcov
  - `start` a `end` sú nepovinné parametre
- `islower() / isupper()`
  - vráti hodnotu `True` ak všetky písmená v stringu sú malé, resp. veľké
  - ak string neobsahuje písmená, vráti `False`
- `istitle()`
  - vráti hodnotu `True` ak každé slovo v stringu začína na veľké písmeno a ostatné písmená sú malé
  - ak string neobsahuje znaky, vráti `False`

Vybrané metódy stringov - kontrola

- `isalpha()`
  - vráti hodnotu `True` ak každý znak v stringu je alfabetický podľa unicode databázy, pre prázdne stringy vráti `False`
- `isdecimal()`
  - vráti hodnotu `True` ak každý znak v stringu môže byť použitý v reprezentácii desiatkových čísel, pre prázdne stringy vráti `False`
- `isdigit()`
  - vráti hodnotu `True` ak každý znak v stringu je číslica, pre prázdne stringy vráti `False`
- `isnumeric()`
  - vráti hodnotu `True` ak každý znak v stringu je číslo, pre prázdne stringy vráti `False`
- `isalnum()`
  - vráti hodnotu `True` ak aspoň jedna podmienka platí pre všetky znaky v stringu

Formátovanie stringov

- `format()`
- placeholder sa označuje pomocou `{}`
- placeholder implicitne odkazuje na parametre a dodržuje ich poradie
  - `"Test {}".format("string")`
  - `"{} {}".format("Test", "string")`
  - `"Test {0}".format("string")`
  - `"Hey {name}".format(name="Jude")`
  - `"Object name: {0.name}".format(object)`
  - `"List head: {lst[0]}".format(lst=mylist)`

Predspracovanie stringov

- v rámci funkcie `format()` je možné predspracovať stringy pomocou funkcií
  - `str()` - pretypuje argument na string
    `"Make it a string: {!s}".format("test")`
  - `repr()` - pretypuje argument na string, pridá úvodzovky
    `"Add quotes: {!r}".format("test")`
  - `ascii()` - pretypuje argument na string ASCII znakov, pridá úvodzovky
    `"Make it ASCII: {!a}".format("test")`

Zarovnanie stringov pri formátovaní

- stringy vieme zarovnať v podreťazci danej dĺžky
  - `"{:<30}".format('left aligned')`
  - `"{:>30}".format('right aligned')`
  - `"{:^30}".format('center aligned')`
  - `"{:-^30}".format('center with fill char')`

Práca so znamienkami

- `"Show always: {:+f}; {:+f}".format(19.09, -19.09)`
- `"Show space for positives: {: f}; {: f}".format(19.09, -19.09)`
- `"Show only minus: {:-f}; {:-f}".format(19.09, -19.09)`
- `"Show only minus: {:f}; {:f}".format(19.09, -19.09)`

Práca s číslami

```
"int: {0:d}, hex: {0:x}, oct: {0:o}, bin:
{0:b}".format(42)

"Add separator: {:,}".format(21081968)

"Round float numbers:
{0:.2f}".format(19.9499999999999)

"Percentage: {:.2%}".format(true/total)
```

Formátovanie pomocou dictionary

```
person_dict = {
     "name": "Roman",
     "age": 30
}
"{name} is {age} years old".format(**person_dict)
```

Zoznamy v Pythone

- množina zoradených prvkov
- prvky môžeme meniť
- môže obsahovať duplicitné prvky
- môže obsahovať prvky rôzneho typu (nie je typické)
- údajová štruktúra, nie primitívny typ

Základná práca so zoznamami

- vytvorenie zoznamu

```
lst = [1, 2, 3]
lst = []
lst = list()
```

- indexovanie a slicing - ako pri stringoch
- aktualizácia hodnoty

```
lst[0] = "a"
lst[1:3] = ["b", "c"]
```

Základná práca so zoznamami

- prechádzanie zoznamom

```
for element in lst:
    print(element)
for idx, element in enumerate(lst):
    print(idx, element)
```

- vyhľadávanie

```
element [not] in lst
```

- nájdenie indexu prvku v zozname (`ValueError` ak prvok nebol nájdený)

```
lst.index(elem[, start[, end]])
```

- určenie dĺžky zoznamu

```
len(lst)
```

- otočenie zoznamu

```
lst.reverse()
```

Pridávanie prvkov do zoznamu

- pridávanie prvku na koniec

```
lst.append(4)
```

- pridávanie prvku na danú pozíciu

```
lst.insert(2, "new element")
```

- konkatenácia zoznamov

```
lst = lst + [6, 7, 8]
lst.extend([6, 7, 8])
```

Vymazanie prvkov zo zoznamu

- vymazanie daného elementu (`ValueError` ak hodnota neexistuje)

```
lst.remove('a')
```

- vymazanie elementu z pozície

```
lst.pop(2)
del lst[2]
```

- vymazanie obsahu zoznamu

```
lst.clear()
del lst[:]
```

- vymazanie zoznamu

```
del lst
```

Zoradenie zoznamu

- prvky musia byť rovnakého typu
- `lst.sort(key=None, reverse=False)`
  - defaultne vzostupne (od najmenšieho po najväčšie)
  - zostupne - `reverse=True`
  - je možné zadefinovať kľúč zoradenia, zvyčajne pomocou lambda výrazov

Kopírovanie - elementárne typy vs. údajové štruktúry

```
x = 5
y = x
x = 6
```

Aká je hodnota `x` a `y`?

```
lst1 = ['a', 'b', 'c']
lst2 = lst1
lst1.append('d')
```

Aká je hodnota `lst1` a `lst2`?

```
lst1 = ['a', 'b', 'c']
lst2 = lst1
lst1 = [4, 5]
```

Aká je hodnota `lst1` a `lst2`?

Kopírovanie zoznamov

- `lst.copy()`
  - vráti kópiu zoznamu (iba vonkajšieho, prvky sa nekopírujú)
- `deepcopy()`
  - vráti hlbokú kópiu zoznamu
  - pre zoznamy zoznamov
  - `from copy import deepcopy`

List comprehensions

- pre jednoduché vytvorenie zoznamov, ktoré obsahujú prvky podľa určitého pravidla
- všeobecná syntax

```
lst = [element_tba for element in enumeration]
```

- tretia mocnina prvých 10 čísel

```
lst = []
for num in range(10):
    lst.append(num**3)

lst = [num**3 for num in range(10)]
```

List comprehensions s podmienkami

- párne čísla po 100

```
lst = [num for num in range(101) if num % 2 == 0]
```

- zoznam samohlások zo slova (prekonvertujeme na veľké písmená)

```
lst = [str.upper(c) for c in word
       if lower(c) in ['a', 'e', 'i', 'o', 'u']]
```

- zoznam súborov z priečinka

```
from os import listdir
from os.path import join, isfile
files = [join(dir, x) for x in listdir(dir)
         if isfile(join(dir, x))]
```

Vnorené list comprehensions

- všetky možné kombinácie prvkov z dvoch zoznamov

```
combinations = [(x, y) for x in [1, 2] for y in [3, 1]]
```

- transpozícia matice

```
matrix = [
    [1, 2, 3],
    [4, 5, 6]
]
i_matrix = [[row[i] for row in matrix]
            for i in range(len(matrix[0]))]
```

N-tica (tuple)

- množina zoradených prvkov
- prvky nemôžeme meniť
- môže obsahovať duplicitné prvky
- zvyčajne obsahuje prvky rôzneho typu
- údajová štruktúra, nie primitívny typ

Vytvorenie n-tice

- vymenovaním prvkov
```
sample_tuple = (1, 2, 3)
```
- vytvorenie n-tice s jedným prvkom
```
sample_tuple = (1, )
sample_tuple = 1,
```
- vytvorenie prázdnej n-tice
```
sample_tuple = tuple()
sample_tuple = ()
```

Operácia nad n-ticami

```
x in tuple
x not in tuple
tuple1 + tuple2
tuple * num
tuple[i]
tuple[start:end:step]
len(tuple)
min(tuple)/max(tuple)
tuple.index(element[, start[, end]])
tuple.count(element)
```

Dictionary

- údajová štruktúra s prvkami typu kľúč-hodnota
- prvky nie sú zoradené
- používa všeobecné indexovanie namiesto čísel
- typické prípady použitia
  - slovník
  - premena reprezentácie hodnoty (napr. 1 → one, 2 → two)
  - hašovacia tabuľka

Vytvorenie dictionary

- prázdny dictionary

```
my_dict = {}
my_dict = dict()
```

- vymenovaním prvkov

```
my_dict = {
    1: "one",
    2: "two"
}
```

- pridanie prvku

```
my_dict[3] = "three"
```

Pristupovanie ku prvkom dictionary

- `key [not] in dictionary`
  - určí, či sa kľúč nachádza/nenachádza v dictionary
- `my_dict[key]`
  - `KeyError` ak kľúč nie je v dictionary
- `my_dict.get(key[, default])`
  - vráti hodnotu `mydict[key]` ak existuje
  - ak kľúč nebol nájdený, vyhodí chybu `KeyError`
  - ak `default` bol zadaný a kľúč nie je v dictionary, vráti hodnotu `default`

Prechádzanie dictionary

- `for x in dictionary`
  - for cyklus pre kľúče v dictionary
- `dictionary.keys()`
  - vráti pohľad na kľúče dictionary, z ktorého vieme vytvoriť zoznam `list(dictionary.keys())`
- `dictionary.values()`
  - vráti pohľad na hodnoty dictionary, z ktorého vieme vytvoriť zoznam `list(dictionary.values())`
- `dictionary.items()`
  - vráti pohľad na dvojice kľúč-hodnota dictionary, z ktorého vieme vytvoriť zoznam `list(dictionary.items())`

Zmazanie prvkov z dictionary

- `del dictionary[key]`
  - vymaže dvojicu kľúč-hodnota z dictionary s kľúčom `key`; ak neexistuje, vyhodí `KeyError`
- `dictionary.pop(key[, default])`
  - vymaže a vráti hodnotu dvojice kľúč-hodnota z dictionary s kľúčom `key`
  - ak neexistuje, vráti `default` (ak `default` nebol zadaný, vyhodí `KeyError`)
- `dictionary.popitem()`
  - vymaže a vráti dvojicu kľúč-hodnota, ktorá bola naposledy pridaná do dictionary
- `dictionary.clear()`
  - vymaže obsah dictionary
- `del dictionary`
  - vymaže dictionary

Ďalšie funkcie dictionary

- `dictionary.copy() / deepcopy()`
  - kopírovanie dictionary ako pri zoznamoch
- `dictionary.setdefault(key[, default])`
  - pridá dvojicu kľúč hodnota `dictionary[key] = default` ak kľúč ešte neexistuje v dictionary a vráti hodnotu `default`
  - ak kľúč už existuje, vráti jeho hodnotu
  - `default` je defaultne `None`
- `dictionary.update([other])`
  - aktualizuje dictionary podľa dvojíc kľúč hodnota v `other`
  - `other` môže byť
    - dictionary
    - množina n-tíc alebo iných postupností s dĺžkou 2
    - vieme odovzdať dvojice aj ako parametre

      `dictionary.update(one=1)`

Pole (Array)

- (viacrozmerné) usporiadanie hodnôt rovnakého typu
- k údajom pristupujeme pomocou indexov - na základe ukladania v pamäti
- v Pythone dve možnosti:
  - zoznam zoznamov (zoznamov zoznamov...)
  - pomocou knižnice `numpy`
    - C-čková implementácia polí
    - vyšší výkon a rýchlejšie výpočty ako pomocou zoznamov

Numpy

- implementácia polí v Pythone (v skutočnosti sú to matice)
- podobné, ako zoznamy, iba nemeniteľné (hodnoty však vieme aktualizovať)
- vytvorenie polí:

```
import numpy as np
my_array = np.array([2, 3, 4])
```

- konvertovanie vstupu na pole:

```
numpy.asarray(a, dtype=None, order=None)
```

Typ polí

- každé pole má typ `ndarray`, jednotlivé prvky musia byť rovnakého typu `my_array.dtype`

- typy prvkov nie sú pythonovské primitívne typy, sú to vlastné implementácie knižnice `numpy`

Tvar polí

- každé pole má jednu alebo viac dimenzií
- tvar poľa je definovaný rozmermi

```
my_array.shape
```

- vracia n-ticu s rozmermi, napr.: `(2, 3)`, kde prvá hodnota je vonkajší rozmer, a postupujeme smerom dnu
- ak je pole dvojdimenzionálne, prvý rozmer je počet riadkov, druhý je počet stĺpcov*

Úprava rozmerov poľa

- `numpy.reshape(array, newshape, order='C')`
  - upraví tvar poľa na požadované rozmery
  - počet prvkov v poli a počet prvkov v novom tvare musí byť rovnaký
  - order určí poradie pridávania prvkov (`C` alebo `F`)
  - vracia nové pole

- `numpy.flatten(order='C')`
  - vracia nové pole - vektorová reprezentácia (jeden riadok, resp. stĺpec)

zeros(), ones() a full()

- metódy slúžia na inicializáciu matice s hodnotami 0, 1 alebo vlastnou hodnotou

- `numpy.zeros(shape, dtype=float, order='C')`
- `numpy.ones(shape, dtype=float, order='C')`
- `numpy.full(shape, fill_value, dtype=None, order='C')`

Pridávanie prvkov

- implementuje konkatenáciu polí

- `numpy.hstack(tup)`
  `numpy.hstack((array1, array2))`
- `numpy.vstack(tup)`
  `numpy.vstack((array1, array2))`

- pre vytvorenie polí dynamicky je lepšie použiť zoznamy, a následne vygenerovať pole

Indexovanie a slicing v numpy

- zásady sú rovnaké ako v Pythone, numpy ale umožňuje skrátený zápis

```
np.array([(1,2,3), (4,5,6)])
print('First row:', e[0])
print('Second column:', e[:,1])
print('Second row, first two values:', e[1, :2])
```

Aritmetické operácie nad maticami

- primitívne operácie fungujú element-wise

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

test [2, 3, 4]
```

Štatistické metódy

- `np.min(array)`
- `np.max(array)`
- `np.mean(array)`
- `np.median(array)`
- `np.std(array)`

- všetky podporujú parameter `axis`, ktorý určí podľa ktorej osi sa má vypočítať štatistická metrika

Maticové operácie

- `np.transpose(array, axes=None)`
- `np.dot(array1, array2)`
- `np.matmul(array1, array2)`
  - ak polia sú dvojrozmerné - štandardné násobenie
  - ak polia majú jeden rozmer - vygeneruje sa z nich dvojrozmerné pole
  - ak polia majú viac rozmerov, ako dva - považujú sa za zásobník dvojrozmerných matíc

Generovanie rozsahov

- `numpy.arange(start, stop, step)`
  - funguje rovnako ako `range()`
- `numpy.linspace(start, stop, num=50, endpoint=True)`
  - vygeneruje `num` hodnôt v rovnakej vzdialenosti od seba v intervale `[start, stop]`
  - `endpoint` určí, či interval má obsahovať hodnotu stop

Pandas

- knižnica pre dátovú analytiku
- nadstavba nad `numpy`
- pre tabuľkové údaje a časové rady
- umožňuje všeobecné indexovanie (cez názvy stĺpcov)
- zvyčajne sa používa importom

```
import pandas as pd
```

Základné objekty

- postupnosť prvkov – `Series`

```
s = pd.Series([1, 3, 5, np.nan, 6, 8])
```

- údajový rámec – `DataFrame`

```
my_ids = pd.date_range("20220214", periods=4)

df = pd.DataFrame(
    np.random.randn(4, 4),
    index=my_ids,
    columns=['A', 'B', 'C', ,D'])
```

Generovanie údajových rámcov

- z dictionary

```
df2 = pd.DataFrame({
      "A": 1.0,
      "B": "2022-02-14",
      "C": ["test", "train", "eval", "train"]
})
```

- načítanie zo súboru, napr.

```
df3 = pd.read_csv(path, sep=',')
```

Indexovanie v dataframeoch

- funguje všeobecné Pythonovské indexovanie
- ďalšie pomocné možnosti prístupu: `at`, `iat`, `loc`, `iloc`
- výber stĺpca (vráti postupnosť `Series`)
  `df["A"]`
- výber riadkov
  `df[0:3]`
  podľa indexov
  `df["20220214":"20220216"]`
- výber riadkov a stĺpcov
  `df.loc["20220214":"20220216", ["A", "B"]]`
- `iat` a `iloc` fungujú podobne ale iba s číselnými indexmi

Podmienený výber

- pomocou podmienok (podmieňovacie operátory)
  `df[df["A"] > 0]`

- pre filtrovanie môžeme použiť metódu `isin()`
  `df[df["C"].isin(["test", "train"])]`

Nastavenie hodnôt

- rozšírenie o stĺpec:
  `df["F"] = "new"`

- nastavenie hodnoty podľa (číselného) indexu
  `df.at[0, "A"] = 0 / df.iat[0, "A"] = 0`

- nastavenie niekoľkých hodnôt pomocou polí
  `df.loc[:, "D"] = np.array([5] * len(df))`

- podmienené nastavenie (napr. nastavenie na absolútne hodnoty)
  `df[df < 0] = -df`

Základná práca s dataframeami

- ukážkové riadky zo začiatku/konca dataframeu
  `df.head(n=g) / df.tail(n=5)`
- získanie zoznamu indexov a stĺpcov
  `df.index / df.columns`
- konvertovanie na `numpy` polia (iba pri rovnakých údajových typoch)
  `df.to_numpy()`
- prehľad štatistických metrík
  `df.describe()`
- transpozícia
  `df.T`

Triedenie údajov v dataframeoch

- triedenie indexov
  ```
  df.sort_index(axis=1, ascending=False)
  ```

- triedenie podľa hodnôt
  ```
  df.sort_values(by="B")
  ```

Spracovanie chýbajúcich hodnôt

- vymazanie riadkov s chýbajúcimi hodnotami
  `df.dropna(how="any")`

- doplnenie chýbajúcich hodnôt
  `df.fillna(value=0)`

- získanie prehľadu o prítomnosti chýbajúcich hodnôt
  `pd.isna(df)`

Štatistické metódy

- priemer podľa stĺpcov / riadkov
```
df.mean() / df.mean(1)
```

- aplikácia ľubovoľnej funkcie
```
df.apply(np.cumsum)  # suma
df.apply(lambda x: x.max() - x.min())
```

- početnosť hodnôt
```
df["C"].value_counts()
```

Ďalšie funkcionality

- konkatenácia – `concat()`
- spojenie ako v SQL – `join()`
- zgrupovanie – `groupby()`
  `df.groupby("A") / df.groupby(["A", "B"])`
- zhustenie / rozbalenie dimenzií
  `df.stack() / df.unstack()`

Zhrnutie

- práca s reťazcami
- práca so zoznamami
- práca s n-ticami
- práca s dictionary
- rozdiel medzi priradením a kopírovaním
- `numpy` polia a `pandas` dataframey