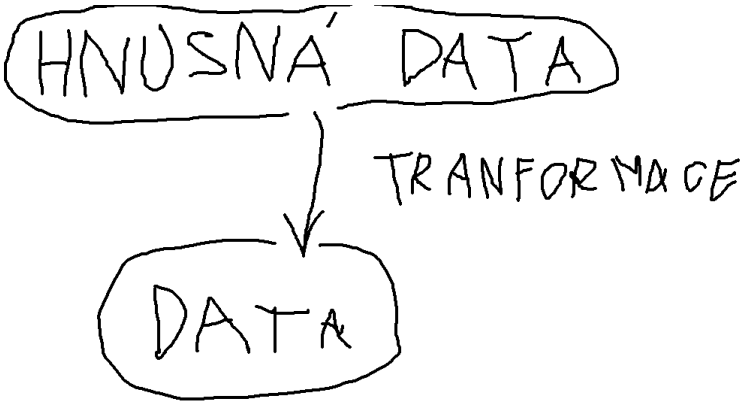
lekce

May 13, 2020

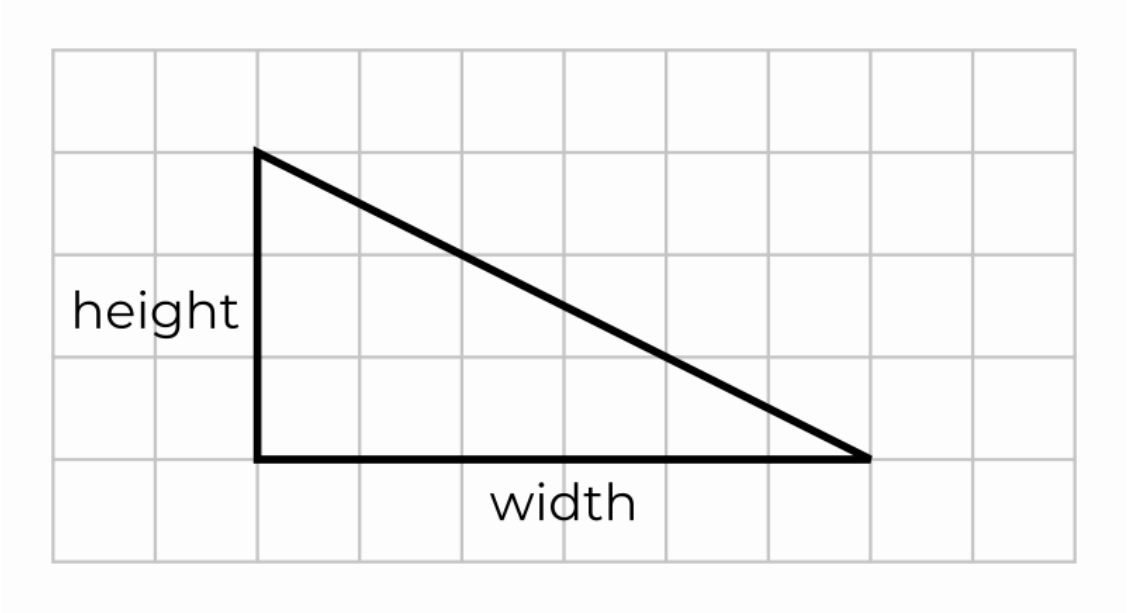
1 Transformace



1.1 1. Vlastní funkce

Příklad: funkce `round`, `len`, `random`, `open`.

- Funkce je poměrně základním konceptem v Pythonu a programování vůbec.
- Dosud jsme jich potkali spoustu, jak přímo z Pythonu nebo z nějaké knihovny (`requests`, `pandas`, …).
- Je možné definovat si vlastní funkce.
- Umožňuje definovat blok kódu provádějící určitou specializovanou činnost, která se opakuje.
- Přesné chování funkce je závislé na vstupních parametrech.
- Vrací hodnotu jako výsledek.



Z kodim.cz ukradneme nápad na funkci pro výpočet obsahu trojúhelníku.

[1]:
```
height1 = 3
width1 = 6
area1 = (height1 * width1) / 2
print(area1)

height2 = 4
width2 = 3
area2 = (height2 * width2) / 2
print(area2)
```

```
9.0
6.0
```

Pokud potřebujeme spočítat obsah mnoha trojúhelníků, začne být otravné vypisovat pořád ten samý kód dokola. Navíc při častém opisování toho samého snadno uděláme chybu.

Definujeme si funkci, která v parametrech dostane rozměry trojúhelníku a jako výsledek vrátí jeho obsah.

[2]:
```
def triangle_area(height, width):
    return (height * width) / 2
```

[3]:
```
area1 = triangle_area(3, 6)
print(area1)

area2 = triangle_area(4, 3)
print(area2)
```

```
9.0
6.0
```

Formát definice funkce je následující.

```
def <nazev_funkce>(<parametry>):
    <libovolny kod>
```

- Názvy funkcí se řídí stejnými pravidly, jako názvy proměnných.
- Funkce může mít libovolný počet parametrů. Pokud nemá žádné, kulaté závorky necháme prázdné (být tam ale musejí).
- Uvnitř těla funkce (odsazené pod jejím názvem a parametry) může být libovolný kód.
- Proměnné definované uvnitř funkce existují pouze tam, zvenčí nejsou viditelné.
- Pokud funkce vrací nějaký výsledek, uvedeme na konci `return <vysledek>`.

1.1.1 Součet kladných hodnot

[4]:
```python
def sum_positive(values):
    result = 0

    for value in values:
        if value > 0:
            result += value

    return result
```

[5]:
```python
test1 = [1, 2, 3, 4, 5]
python_sum1 = sum(test1)
my_sum1 = sum_positive(test1)
print(f"Python sum: {python_sum1}")
print(f"My sum: {my_sum1}")
```

```
Python sum: 15
My sum: 15
```

[6]:
```python
test2 = [-1, 2, 3, -4, 5]
python_sum2 = sum(test2)
my_sum2 = sum_positive(test2)
print(f"Python sum: {python_sum2}")
print(f"My sum: {my_sum2}")
```

```
Python sum: 5
My sum: 10
```

1.1.2 Zápis do CSV

Funkce bez návratové hodnoty. Pozor, je velmi zjednodušená, v praxi by se měly použít pandas nebo pythonovský modul pro práci s CSV soubory.

[7]:
```
def write_to_csv(path, nested_list):
    f = open(path, mode="w", encoding="utf-8")

    for inner_list in nested_list:
        str_list = [str(v) for v in inner_list]
        f.write(",".join(str_list))
        f.write("\n")

    f.close()
```

[8]:
```
data = [["hello", "python"], ["goodbye", "python"], [123, 456]]
write_to_csv("test.csv", data)
```

[9]:
```
!cat test.csv
```

```
hello,python
goodbye,python
123,456
```

1.1.3 Převod školních známek

Funkce mohou také zpřehlednit kód. Použijeme příklad z dřívejška s převodem známek z písmen na čísla.

[10]:
```
grades = ["A", "A", "B", "A", "C", "D", "E", "C", "B", "A", "C"]
```

[11]:
```
converted1 = [grade.replace("A", "1").replace("B", "2").replace("C", "3").
 ↪replace("D", "4").replace("E", "5")
              for grade in grades]
print(converted1)
```

```
['1', '1', '2', '1', '3', '4', '5', '3', '2', '1', '3']
```

[12]:
```
def convert_grade(grade):
    return grade.replace("A", "1").replace("B", "2").replace("C", "3").
 ↪replace("D", "4").replace("E", "5")
```

[13]:
```
converted2 = [convert_grade(grade) for grade in grades]
print(converted2)
```

```
['1', '1', '2', '1', '3', '4', '5', '3', '2', '1', '3']
```

[14]:
```
def convert_grade_better(grade):
    if grade == "A":
        return "1"
    elif grade == "B":
        return "2"
    elif grade == "C":
```

```
        return "3"
    elif grade == "D":
        return "4"
    else:
        return "5"
```

[15]:
```
converted3 = [convert_grade_better(grade) for grade in grades]
print(converted3)
```

```
['1', '1', '2', '1', '3', '4', '5', '3', '2', '1', '3']
```

1.2 Cvičení

[ ]:
```

```

1.3 2. Transformace dat v Pandas

Budeme se snažit převést špatně formátovaná data do tvaru, se kterým se lépe pracuje. Něco takového řešíme v praxi v jednom kuse – dostaneme horu dat, nic o nich nevíme a potřebujeme v nich udělat pořádek.

Pracujeme s daty o hmotnosti Kristiána během 14 dnů, kdy se snažil zhubnout.

[16]:
```
!cat vaha.txt
```

```
den     váha    běh     týden
pá 3.   75,6 kg 3 km    1
so 4.   75,3 kh pauza   1
ne 5.   75,9kg  pauza   1
po 6.   76,1 kg 2 km    1
út 7    75,4 kg paza    1
st 8.   75kg    pauza   1
čt 9.   74,9 kg 3       1
pá 10.  74,8 k  pauza   2
so 11.  74,3kg  3 km    2
ne 12   75,2 kg 4 km    2
po 13.  74,5 kg         2
út 14.  74,2 kg pauza   2
st 15.  74,1 kg 3 km    2
čt 16   73,8 kg 3km     2
```

[17]:
```
import pandas
```

[18]:
```
vaha = pandas.read_csv("vaha.txt", encoding="utf-8", sep="\t")
vaha
```

[18]:

	den	váha	běh	týden
0	pá 3.	75,6 kg	3 km	1

```
1    so 4.  75,3 kh  pauza      1
2    ne 5.   75,9kg  pauza      1
3    po 6.  76,1 kg   2 km      1
4     út 7  75,4 kg   paza      1
5    st 8.     75kg  pauza      1
6    čt 9.  74,9 kg      3      1
7   pá 10.   74,8 k  pauza      2
8   so 11.   74,3kg   3 km      2
9    ne 12  75,2 kg   4 km      2
10  po 13.  74,5 kg    NaN      2
11  út 14.  74,2 kg  pauza      2
12  st 15.  74,1 kg   3 km      2
13   čt 16  73,8 kg    3km      2
```

[19]:
```
vaha.dtypes
```

[19]:
```
den      object
váha     object
běh      object
týden     int64
dtype: object
```

Conclusion: Kristián je trochu prase, aneb data v praxi.

Nejprve transformujeme sloupec `den` na číslo dne. Sloupec `den` je uložen vždy jako `2pismenny_nazev_dne cislo_dne`, některá čísla navíc končí tečkou.

1.3.1 Zahodíme názvy dnů

[20]:
```
dny = vaha["den"]
dny
```

[20]:
```
0      pá 3.
1      so 4.
2      ne 5.
3      po 6.
4       út 7
5      st 8.
6      čt 9.
7     pá 10.
8     so 11.
9      ne 12
10    po 13.
11    út 14.
12    st 15.
13     čt 16
Name: den, dtype: object
```

[21]:
```
cislo_dne = dny.str[3:]
cislo_dne
```

[21]:
```
0      3.
1      4.
2      5.
3      6.
4       7
5      8.
6      9.
7     10.
8     11.
9      12
10    13.
11    14.
12    15.
13     16
Name: den, dtype: object
```

1.3.2 Zahodíme tečky

[22]:
```
cislo_dne = cislo_dne.str.replace(".", "")
cislo_dne
```

[22]:
```
0      3
1      4
2      5
3      6
4      7
5      8
6      9
7     10
8     11
9     12
10    13
11    14
12    15
13    16
Name: den, dtype: object
```

1.3.3 Převedeme na čísla

[23]:
```
cislo_dne = pandas.to_numeric(cislo_dne)
cislo_dne
```

[23]:
```
0      3
1      4
2      5
3      6
4      7
5      8
6      9
7     10
8     11
9     12
10    13
11    14
12    15
13    16
Name: den, dtype: int64
```

1.3.4 Uložíme ještě název dne

[24]:
```
dny
```

[24]:
```
0      pá 3.
1      so 4.
2      ne 5.
3      po 6.
4       út 7
5      st 8.
6      čt 9.
7     pá 10.
8     so 11.
9      ne 12
10    po 13.
11    út 14.
12    st 15.
13     čt 16
Name: den, dtype: object
```

[25]:
```
nazev_dne = dny.str[:2]
nazev_dne
```

[25]:
```
0     pá
1     so
2     ne
3     po
4     út
5     st
6     čt
7     pá
```

```
8     so
9     ne
10    po
11    út
12    st
13    čt
Name: den, dtype: object
```

1.3.5 Obojí uložíme do dataframe

[26]:
```
vaha.drop("den", axis="columns", inplace=True)
vaha
```

[26]:

	váha	běh	týden
0	75,6 kg	3 km	1
1	75,3 kh	pauza	1
2	75,9kg	pauza	1
3	76,1 kg	2 km	1
4	75,4 kg	paza	1
5	75kg	pauza	1
6	74,9 kg	3	1
7	74,8 k	pauza	2
8	74,3kg	3 km	2
9	75,2 kg	4 km	2
10	74,5 kg	NaN	2
11	74,2 kg	pauza	2
12	74,1 kg	3 km	2
13	73,8 kg	3km	2

[27]:
```
vaha["číslo_dne"] = cislo_dne
vaha["název_dne"] = nazev_dne
vaha
```

[27]:

	váha	běh	týden	číslo_dne	název_dne
0	75,6 kg	3 km	1	3	pá
1	75,3 kh	pauza	1	4	so
2	75,9kg	pauza	1	5	ne
3	76,1 kg	2 km	1	6	po
4	75,4 kg	paza	1	7	út
5	75kg	pauza	1	8	st
6	74,9 kg	3	1	9	čt
7	74,8 k	pauza	2	10	pá
8	74,3kg	3 km	2	11	so
9	75,2 kg	4 km	2	12	ne
10	74,5 kg	NaN	2	13	po
11	74,2 kg	pauza	2	14	út
12	74,1 kg	3 km	2	15	st

```
13  73,8 kg    3km      2         16        čt
```

[28]:
```
vaha.dtypes
```

[28]:
```
váha         object
běh          object
týden         int64
číslo_dne     int64
název_dne    object
dtype: object
```

Tím máme slušně uložené údaje o dnech.

1.4 3. Chroustání Series

Zde budeme transformovat sloupec **váha**.

[29]:
```
vaha
```

[29]:

	váha	běh	týden	číslo_dne	název_dne
0	75,6 kg	3 km	1	3	pá
1	75,3 kh	pauza	1	4	so
2	75,9kg	pauza	1	5	ne
3	76,1 kg	2 km	1	6	po
4	75,4 kg	paza	1	7	út
5	75kg	pauza	1	8	st
6	74,9 kg	3	1	9	čt
7	74,8 k	pauza	2	10	pá
8	74,3kg	3 km	2	11	so
9	75,2 kg	4 km	2	12	ne
10	74,5 kg	NaN	2	13	po
11	74,2 kg	pauza	2	14	út
12	74,1 kg	3 km	2	15	st
13	73,8 kg	3km	2	16	čt

Jak se píše na kodim.cz: *Pokud se nám podaří rozdělit hodnotu podle mezery, můžeme první část převést na číslo. Pokud se to nepovede, můžeme dělit podle písmenka 'k'*

Tento postup si definujeme jako funkci, kterou aplikujeme na sloupec **váha** prvek po prvku.

[30]:
```
def prevod_vahy(vaha):
    casti = vaha.split(" ")

    if len(casti) < 2:
        casti = vaha.split("k")

    desetinna_tecka = casti[0].replace(",", ".")
    return float(desetinna_tecka)
```

Takto definovanou funkci nyní použijeme jako parametr funkce `apply` volané na sloupci `váha`. Funkce `apply` vezme námi definovanou funkci, a aplikuje ji na každý prvek daného sloupce. Výsledkem je sloupec výsledků naší funkce.

[31]:
```
vaha["váha"] = vaha["váha"].apply(prevod_vahy)
vaha
```

[31]:

	váha	běh	týden	číslo_dne	název_dne
0	75.6	3 km	1	3	pá
1	75.3	pauza	1	4	so
2	75.9	pauza	1	5	ne
3	76.1	2 km	1	6	po
4	75.4	paza	1	7	út
5	75.0	pauza	1	8	st
6	74.9	3	1	9	čt
7	74.8	pauza	2	10	pá
8	74.3	3 km	2	11	so
9	75.2	4 km	2	12	ne
10	74.5	NaN	2	13	po
11	74.2	pauza	2	14	út
12	74.1	3 km	2	15	st
13	73.8	3km	2	16	čt

[32]:
```
vaha.dtypes
```

[32]:
```
váha         float64
běh           object
týden          int64
číslo_dne      int64
název_dne     object
dtype: object
```

Ještě by bylo fajn upravit sloupec `běh`.

[33]:
```
def prevod_behu(beh):
    if beh == "pauza" or beh == "paza":
        return 0
    elif pandas.isnull(beh):
        return beh
    else:
        casti = beh.split(" ")

        if len(casti) < 2:
            casti = beh.split("k")

        return float(casti[0])
```

[34]:
```
vaha["běh"] = vaha["běh"].apply(prevod_behu)
vaha
```

[34]:

	váha	běh	týden	číslo_dne	název_dne
0	75.6	3.0	1	3	pá
1	75.3	0.0	1	4	so
2	75.9	0.0	1	5	ne
3	76.1	2.0	1	6	po
4	75.4	0.0	1	7	út
5	75.0	0.0	1	8	st
6	74.9	3.0	1	9	čt
7	74.8	0.0	2	10	pá
8	74.3	3.0	2	11	so
9	75.2	4.0	2	12	ne
10	74.5	NaN	2	13	po
11	74.2	0.0	2	14	út
12	74.1	3.0	2	15	st
13	73.8	3.0	2	16	čt

[35]:
```
vaha.dtypes
```

[35]:
```
váha         float64
běh          float64
týden          int64
číslo_dne      int64
název_dne     object
dtype: object
```

1.5 4. Vlastní agregační funkce

Definujeme si vlastní agregační funkci, podobně jako jsme posledně viděli funkce `mean`, `std` a podobné, volané na výsledku operace `groupby`.

Data seskupíme podle týdne a v každém týdnu napočítáme nějakou funkci váhy. Zde např. průměrnou váhu v každém týdnu.

[36]:
```
vaha.groupby("týden")["váha"].mean()
```

[36]:
```
týden
1    75.457143
2    74.414286
Name: váha, dtype: float64
```

Budeme počítat tzv. rozpětí (spread) – rozdíl mezi maximální a minimální hodnotou. Takovou funkci pandas nenabízí, musíme si ji tedy napsat sami.

Funkce jako argument dostane pandas series (sloupec) a jako výsledek vrátí jeho rozpětí.

[37]:
```
def spread(series):
    return series.max() - series.min()
```

Nejprve funkci aplikujeme na jediný sloupec. Tím dostaneme rozpětí vah za celé sledované období.

[38]:
```
vaha["váha"].agg(spread)
```

[38]:
```
2.299999999999997
```

A teď na groupby objekt – za každý týden dostaneme rozpětí vah.

[39]:
```
vaha.groupby("týden")["váha"].agg(spread)
```

[39]:
```
týden
1    1.2
2    1.4
Name: váha, dtype: float64
```

[ ]:
```

```