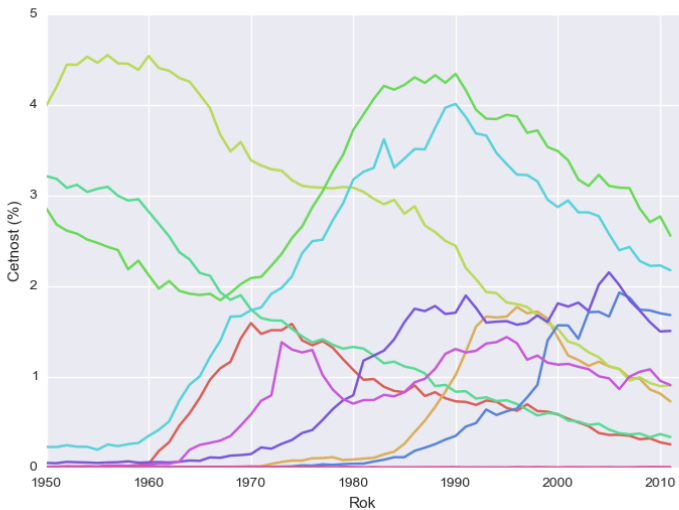


Regulární výrazy, práce s daty

IB113
Radek Pelánek

2019



JAN, JAROSLAV, JIŘÍ, MAREK, MATĚJ, NIKOLA,
 ONDŘEJ, RADEK, TOMÁŠ, VALDEMAR

Příklady a techniky relevantní k situaci:

- mám data (text, tabulková data)
- chci získat vhled, informace

Regulární výrazy: motivace

- vyhledání e-mailových adres v textu
- vyhledání odkazů v HTML dokumentu
- náhrada „jméno příjmení“ za „příjmení jméno“
- změna formátu datumů
- odstranění bílých znaků

nástroj pro hledání „vzorů“ v textu

- programování
- textové editory
- příkazová řádka: např. `grep`
- teorie: formální jazyky, konečné automaty

- obecně používaný nástroj
- syntax velmi podobná ve většině jazyků, prostředí
- bohatá syntax
- následuje „ochutnávka“, ukázky základního využití v Pythonu

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.

OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!



BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!



IT'S HOPELESS!

EVERYBODY STAND BACK.



I KNOW REGULAR EXPRESSIONS.



Naivní způsob hledání e-mailových adres v textovém souboru:

```
import re
f = open("testovaci-soubor.txt")
for line in f.readlines():
    if re.search(r'[a-z]+@[a-z]+\ .cz', line):
        print(line)
f.close()
```


Znaky a speciální znaky

- základní znak „vyhoví“ právě sám sobě
- speciální znaky: `. ^ $ * + ? { } [] \ | ()`
 - umožňují konstrukci složitějších výrazů
 - chceme, aby odpovídaly příslušnému symbolu \Rightarrow prefix `\`

Skupiny znaků

`[abc]` = jeden ze znaků a, b, c

`[^abc]` = cokoliv jiného než a, b, c

`\d` Čísla: `[0-9]`

`\D` Cokoliv kromě čísel: `[^0-9]`

`\s` Bílé znaky: `[\t\n\r\f\v]`

`\S` Cokoliv kromě bílých znaků: `[^\t\n\r\f\v]`

`\w` Alfnumerické znaky: `[a-zA-Z0-9_]`

`\W` Nealfnumerické znaky: `[^a-zA-Z0-9_]`

Speciální symboly

- . libovolný znak
- ^ začátek řetězce
- \$ konec řetězce
- | alternativa – výběr jedné ze dvou možností

Jaký je význam následujících výrazů?

- `kocka|pes`
- `^[Pp]rase$`
- `\d[A-Z]\d \d\d\d\d`

Opakování

| | |
|--------|------------------------|
| * | nula a více opakování |
| + | jedno a více opakování |
| ? | nula nebo jeden výskyt |
| {m, n} | m až n opakování |

Pozn. *, + jsou „hladové“, pro co nejmenší počet opakování *?, +?

Jaký je význam následujících výrazů?

- `^\s*Nadpis`
- `^a.+a$`
- `\d{3}\s?\d{3}\s?\d{3}`
- `[a-z]+@[a-z]+\.``cz`
- `^To:\s*(fi|kit)(-int)?@fi\.``muni\.``cz`

Která z následujících slov vyhoví jednotlivým výrazům?

| | <code>p[ars]e</code> | <code>p[ars]*e</code> | <code>p[[^]ars]e</code> |
|---------|----------------------|-----------------------|----------------------------------|
| ps | | | |
| pes | | | |
| pse | | | |
| poe | | | |
| prase | | | |
| poklice | | | |

Která z následujících slov vyhoví jednotlivým výrazům?

| | <code>p[ars]e</code> | <code>p[ars]*e</code> | <code>p[~ars]e</code> |
|----------------------|----------------------|-----------------------|-----------------------|
| <code>ps</code> | × | × | × |
| <code>pes</code> | × | ✓ | × |
| <code>pse</code> | ✓ | ✓ | × |
| <code>poe</code> | × | × | ✓ |
| <code>prase</code> | × | ✓ | × |
| <code>poklice</code> | × | × | × |

Kontrola tabulky v Pythonu

```
import re
texts = ["ps", "pes", "pse", "poe", "prase", "poklice"]
regexps = [ r'p[ars]e', r'p[ars]*e', r'p[^\ars]e' ]

for text in texts:
    print(text, end=" ")
    for regexp in regexps:
        if re.search(regexp, text):
            print(1, end=" ")
        else:
            print(0, end=" ")
    print()
```

- „odkaz“ na předchozí část textu
- pomocí \1, \2, \3, ...
- příklad: slova, která obsahují opakovaně stejnou trojici písmen (jinde než) na začátku a konci
 - **starosta**
 - **periferie**
- jak zapsat regulární výraz?
- jaká další slova splňují?

Backreference a trojice písmen

- seznam slov, např.

https://wiki.korpus.cz/doku.php/seznamy:srovnavaci_seznamy

- regulární výrazy:
 - $\text{^\text{(}\dots\text{)}\text{.*}\backslash\text{1}\text{\$}$
 - $\text{.\text{(}\dots\text{)}\text{.*}\backslash\text{1}\text{.}$
- výstupy např.:
 - **ostrost**
 - **skaliska**
 - **bramborami**
 - **multikulturni**

Bez regulárních výrazů

Varianta „uvnitř slova“:

```
def triple_match(word):  
    for i in range(1, len(word)-3):  
        for j in range(i+1, len(word)-3):  
            if word[i:i+3] == word[j:j+3]:  
                return True  
    return False
```

Je to ekvivalentní?

Detekce opakovaných trojic

- regulární výraz: `.(...)*\1.`
- kontrola podmínky: `word[i:i+3] == word[j:j+3]`
- rozdílné chování: sklen**ene**ho, kov**ov**ou, koc**ic**ich

Co je správně?

Nejasná specifikace problému...

Regulární výrazy: xkcd



<http://xkcd.com/1313/>
http://www.explainxkcd.com/wiki/index.php/1313:_Regex_Golf
<https://regex.alf.nu/>

Regulární výrazy v Pythonu

- knihovna `re` (`import re`)
- `re.match` – hledá shodu na začátku řetězce
- `re.search` – hledá shodu kdekoliv v řetězci
- (`re.compile` – pro větší efektivitu)
- `re.sub` – nahrazení
- „raw string“ – `r'vyraz'` – nedochází k interpretaci speciálních znaků jako u běžných řetězců v Pythonu

Regulární výrazy v Pythonu: práce s výsledkem

- `match/search` vrací „MatchObject“ pomocí kterého můžeme s výsledkem pracovat
- pomocí kulatých závorek `()` označíme, co nás zajímá

Regulární výrazy v Pythonu: práce s výsledkem

```
>>> m = re.match(r'(\w+) (\w+)', \
                  'Isaac Newton, fyzik')
>>> m.group(0)
'Isaac Newton'
>>> m.group(1)
'Isaac'
>>> m.group(2)
'Newton'
```

Regulární výrazy v Pythonu: nahrazení

```
import re

text = 'Petr Novak, 329714, FI B-AP BcAP'
print(re.sub(r'(\w*) (\w*), (\d*)',
             r'\2 \1, UCO=\3',
             text))

# 'Novak Petr, UCO=329714, FI B-AP BcAP'
```


Kahoot: program 1

```
import re

def filter_list(regex, alist):
    output = []
    for word in alist:
        if re.search(regex, word):
            output.append(word)
    return output

fruits = ['apple', 'banana', 'pear', 'blackberry',
          'fig', 'peach']

print(filter_list(r'a.*e', fruits))
```

Kahoot: program 2

```
import re

def filter_list(regex, alist):
    output = []
    for word in alist:
        if re.search(regex, word):
            output.append(word)
    return output

nums = ['1.2', '3', '5.832', '428', '1.55']

print(len(filter_list(r'\d.\d', nums)))
```

Kahoot: program 3

```
import re

def filter_list(regex, alist):
    output = []
    for word in alist:
        if re.search(regex, word):
            output.append(word)
    return output

names = ['Kremilek a Vochomurka', 'Maxipes Fik',
         'Rakosnicek', 'Mala carodejnice',
         'Tom a Jerry']
print(len(filter_list(r'^\w*\s\w*$', names)))
```

Kahoot: program 4

```
import re

def decide(regex, string):
    if re.search(regex, string):
        return 'Y'
    return 'N'

for regex in [r'A.I', r'A.*I', r'A.*I$']:
    print(decide(regex, 'PARDUBICE'), end="")
```

Kahoot: program 5

```
import re

text = 'FI MU, Botanicka 68a, 602 00 Brno'

m = re.match(r'.*, (\w*).*, [\s\d]*(\w*)$', text)

print(m.group(1)+ ' ' + m.group(2))
```


Kahoot: program 6

```
import re

text = 'Petr /vykopal#sloveso/ diru.'

text = re.sub(r'/(.*)#(.*)/',
              r'<span cat="\2">\1</span>',
              text)

print(text)
```

Kahoot: program 7

```
text = 'one two three'
```

```
text.split('e')
```

```
print(len(text))
```

Kahoot: program 8

```
def process(alist):  
    output = []  
    for x in alist:  
        if isinstance(x, str):  
            output.append(x)  
    return output  
  
alist = [3, '2.4', '1', 8.2, '2']  
  
print("+".join(process(alist)))
```

Kahoot: program 9

```
def magic(alist):  
    for i in range(len(alist)):  
        alist[i] = alist[i].count('a')  
  
fruits = ['apple', 'banana', 'fig']  
magic(fruits)  
print(fruits)
```

Analýza textu

- vstup: text (v textovém souboru)
- výstup: zajímavé statistiky
 - délka vět, slov
 - nejčastější slova (určité minimální délky)
 - frekvence písmen, digramy, trigramy

⇒ *cvičení*

Analýza textu

statistiky délky slov a vět:

- \bar{x} – průměr
- s – směrodatná odchylka (míra variability)

| | slova | | věty | |
|-------------|-----------|-----|-----------|------|
| | \bar{x} | s | \bar{x} | s |
| Starý zákon | 4.3 | 2.3 | 14.9 | 7.8 |
| Čapek | 4.5 | 2.5 | 14.9 | 13.3 |
| Pelánek | 5.9 | 3.6 | 13.5 | 6.9 |
| Wikipedie | 5.6 | 3.0 | 14.8 | 8.3 |

Analýza textu – dekompozice problému

- 1 text → seznam délek slov (vět)
- 2 seznam délek → statistiky
- 3 statistiky → výpis

Imitace textu

- vstup: rozsáhlý text
- výstup: náhodně generovaný text, který má „podobné charakteristiky“ jako vstupní text
- imitace na úrovni písmen nebo slov

Náhodnostní imitace vstupního textu

I špiské to pole kavodali pamas ne nebo kdy v Dejný Odm sem uvalini se zabijí s Pan stěží ře, a silobe lo v ne řečekovicích blova v nadrá těly jakvěmutelaji rohnutkohonebout anej Fravinci V A pěk finé houty. zal Jírakočítencej ské žil, kdDo jak a to Lorskříže si tomůžu schno mí, kto.

Kterak král kočku kupoval V zemi Taškářů panoval král a zapřisáhl se velikou přísahou že bude pochválena První pán si jí ani nevšimnul zato druhý se rychle shýbl a Jůru pohladil Aha řekl sultán a bohatě obdaroval pana Lustiga koupil od něho telegram z Bombaje v Indii není o nic horší člověk nežli někdo z mých hraček Kdepak mávl Vašek rukou

Základní přístup

- 1 vstupní text \Rightarrow statistiky textu
- 2 statistiky \Rightarrow generování náhodného textu

Co jsou vhodné statistiky?

- základ: frekvence písmen (slov)
- rozšíření: korelace mezi písmeny (slovy)

příklad: pokud poslední písmeno bylo **a**:

- **e** velmi nepravděpodobné (méně než obvykle)
- **l, k** hodně pravděpodobná (více než obvykle)

- základní frekvenční analýza – datová struktura seznam nebo slovník
písmeno \Rightarrow frekvence
- rozšířená analýza – seznam seznamů nebo slovník slovníků
písmeno \Rightarrow { písmeno \Rightarrow frekvence }

generování

- podle aktuálního písmene získám frekvence
- vyberu náhodné písmeno podle těchto frekvencí – „vážená ruleta“

- seznam seznamů – tabulka 26×26 , pouze pro malá písmena anglické abecedy
- slovník slovníků – pro libovolné symboly

Korelace písmen: seznamy

```
def letter_ord(char):  
    return ord(char) - ord('a')  
  
def letter_correlations(text):  
    counts = [[0 for a in range(26)]  
              for b in range(26)]  
    for i in range(len(text)-1):  
        a = letter_ord(text[i])  
        b = letter_ord(text[i+1])  
        if 0 <= a <= 26 and 0 <= b <= 26:  
            counts[a][b] += 1  
    return counts
```

Výpis nejčastějších následujících písmen

```
def most_common_after(letter, counts, top_n=5):  
    i = letter_ord(letter)  
    letter_counts = [(counts[i][j], chr(ord('a')+j))  
                     for j in range(26)]  
    letter_counts.sort(reverse=True)  
    for count, other_letter in letter_counts[:top_n]:  
        print(other_letter, count)
```

Korelace písmen: slovníky

```
def symbol_correlation(text):
    counts = {}
    last = " "
    for symbol in text:
        if last not in counts:
            counts[last] = {}
        counts[last][symbol] = counts[last].get(symbol, 0)
        last = symbol
    return counts
```

Tip pro kratší kód: defaultdict

Výpis nejčastějších následujících písmen

```
def most_common_after_symbol(symbol, counts, top_n=5):  
    print(sorted(counts[symbol].keys(),  
                 key=lambda s:  
                 -counts[symbol][s])[:top_n])
```

Imitate textu

```
def get_next_letter(current, counts):
    total = sum(counts[current].values())
    r = random.randint(0, total-1)
    for symbol in counts[current].keys():
        r -= counts[current][symbol]
        if r < 0:
            return symbol
    return " "
```



```
def imitate(counts, length=100):
    current = " "
    for _ in range(length):
        print(current, end="")
        current = get_next_letter(current, counts)
```

Imitace textu: rozšíření

- nebrat v potaz pouze předcházející písmeno, ale k předcházejících písmen
- *doporučené cvičení*

Imitace sofistikovanejji

- Recurrent Neural Networks – dokážj postihnout i složitějji aspekty jazyka
- básně, recepty, Wikipedia články, zdrojové kódy, ...

<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

PANDARUS:

Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

- data: četnosti jmen, příjmení podle roků, krajů, ...
- zdroj: Ministerstvo vnitra ČR

Akt. stav: „Ministerstvo vnitra není oprávněno ke zveřejňování statistik / přehledů s charakterem statistik.

Proto bylo zveřejňování těchto přehledů zrušeno.“

- profiltrovaná data (nejčastější jména):

docs.google.com/spreadsheets/d/1tbh1-sHEndOUnbxXOBXf9CaWJb50tyy0abpOHX6gioc

- **doporučené cvičení**

- snadno zpracovatelné
 - zajímavá data
 - cvičení na vymýšlení otázek
- následuje několik ukázek pro inspiraci ...

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N |
|----|-----------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 1 | JMÉNO | 1950 | 1951 | 1952 | 1953 | 1954 | 1955 | 1956 | 1957 | 1958 | 1959 | 1960 | 1961 | 1962 |
| 2 | ADAM | 15 | 10 | 17 | 17 | 7 | 10 | 9 | 11 | 9 | 11 | 7 | 6 | 10 |
| 3 | ADÉLA | 13 | 16 | 23 | 30 | 19 | 28 | 19 | 22 | 21 | 8 | 12 | 15 | 8 |
| 4 | ADRIANA | 2 | 1 | 3 | 3 | 2 | 3 | 5 | 2 | 7 | 8 | 10 | 6 | 11 |
| 5 | ALENA | 2804 | 3041 | 3033 | 3387 | 3481 | 3611 | 3560 | 3290 | 2978 | 2743 | 2600 | 2651 | 2474 |
| 6 | ALEŠ | 100 | 164 | 170 | 222 | 222 | 231 | 255 | 271 | 251 | 280 | 302 | 317 | 397 |
| 7 | ALEXANDR | 66 | 65 | 72 | 67 | 57 | 70 | 47 | 54 | 62 | 57 | 57 | 90 | 56 |
| 8 | ALEXANDRA | 90 | 107 | 78 | 102 | 80 | 87 | 70 | 63 | 58 | 54 | 70 | 60 | 69 |
| 9 | ALICE | 72 | 78 | 72 | 95 | 91 | 89 | 109 | 88 | 98 | 91 | 103 | 100 | 95 |
| 10 | ALOIS | 332 | 312 | 271 | 303 | 282 | 283 | 229 | 215 | 218 | 195 | 148 | 163 | 144 |
| 11 | ALŽBĚTA | 126 | 131 | 112 | 110 | 89 | 83 | 75 | 77 | 78 | 55 | 45 | 43 | 34 |
| 12 | AMÁLIE | 6 | 4 | 4 | 2 | 0 | 1 | 2 | 1 | 1 | 1 | 1 | 3 | 0 |
| 13 | ANDREA | 11 | 7 | 9 | 9 | 12 | 13 | 4 | 14 | 11 | 10 | 25 | 18 | 26 |
| 14 | ANETA | 0 | 0 | 2 | 1 | 0 | 3 | 0 | 2 | 1 | 2 | 1 | 1 | 1 |
| 15 | ANEŽKA | 271 | 232 | 220 | 170 | 188 | 157 | 128 | 127 | 106 | 91 | 77 | 52 | 52 |
| 16 | ANNA | 3163 | 2993 | 2812 | 2534 | 2352 | 2184 | 2114 | 1993 | 1832 | 1489 | 1377 | 1158 | 1024 |
| 17 | ANTONIE | 131 | 156 | 124 | 127 | 114 | 122 | 106 | 74 | 74 | 63 | 51 | 41 | 32 |
| 18 | ANTONÍN | 1355 | 1308 | 1254 | 1163 | 1114 | 1054 | 1068 | 1010 | 881 | 771 | 748 | 790 | 690 |
| 19 | BARBORA | 22 | 31 | 30 | 23 | 31 | 30 | 22 | 25 | 32 | 39 | 35 | 56 | 91 |
| 20 | BEDŘICH | 157 | 153 | 170 | 146 | 138 | 130 | 144 | 155 | 99 | 108 | 93 | 99 | 88 |
| 21 | BLANKA | 592 | 685 | 701 | 704 | 734 | 692 | 718 | 761 | 675 | 575 | 615 | 685 | 674 |
| 22 | BLAŽENA | 176 | 199 | 172 | 152 | 156 | 124 | 114 | 116 | 112 | 65 | 60 | 65 | 58 |
| 23 | BOHUMIL | 552 | 576 | 547 | 479 | 487 | 441 | 424 | 449 | 374 | 299 | 327 | 259 | 284 |
| 24 | BOHUMILA | 262 | 238 | 269 | 265 | 218 | 218 | 190 | 188 | 171 | 159 | 127 | 113 | 93 |
| 25 | BOHUSLAV | 415 | 362 | 360 | 357 | 332 | 357 | 318 | 342 | 268 | 245 | 236 | 195 | 177 |
| 26 | BOŽENA | 1364 | 1173 | 1080 | 990 | 946 | 964 | 871 | 758 | 681 | 534 | 492 | 420 | 366 |

Poznámky ke zpracování

- načtení CSV
 - funkce `split` → seznam hodnot
 - použití knihovny pro práci s CSV (pro složitější data)
- normalizace (relativní výskyty jmen) – podělit součtem (pro daný rok)
 - různě velké ročníky
 - neúplná data u starých ročníků

Reprezentace dat

slovník mapující jména na seznam počtů výskytů

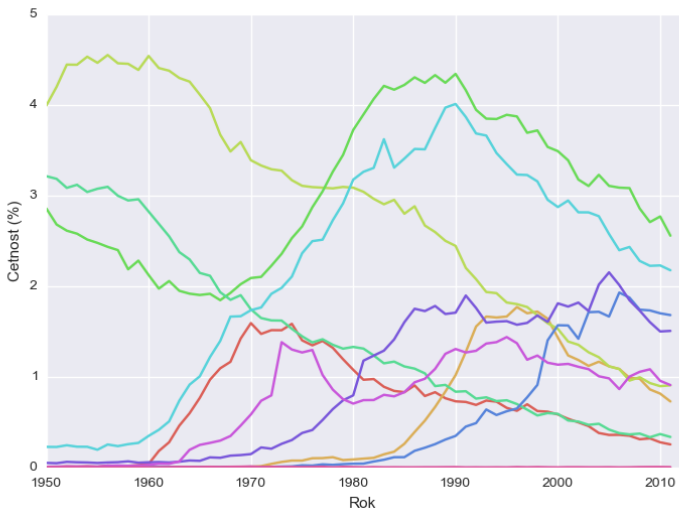
```
{'Alois': [332, 131, 112], 'Adam': [15, 10, 17]}  
print(data[name][year-1950])
```

slovník mapující jména na slovníky mapující roky na počty

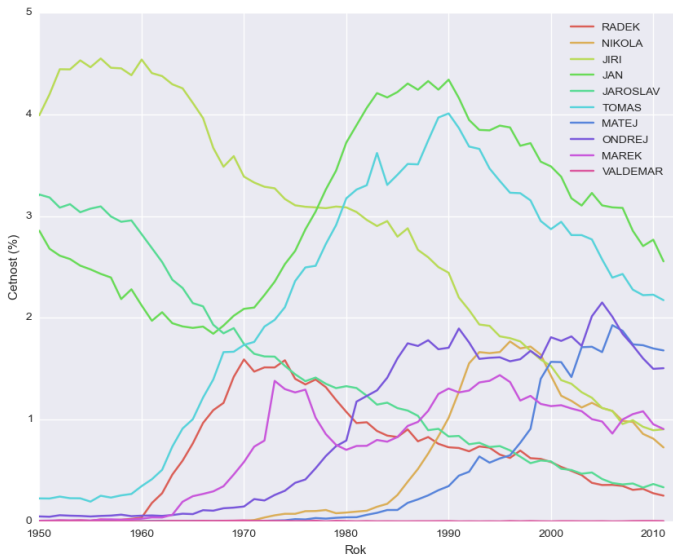
```
{'Adam': {1950: 15, 1951: 10},  
 'Alois': {1950: 332, 1951: 131}}  
print(data[name][year])
```

slovník mapující dvojici jméno+rok na počet výskytů

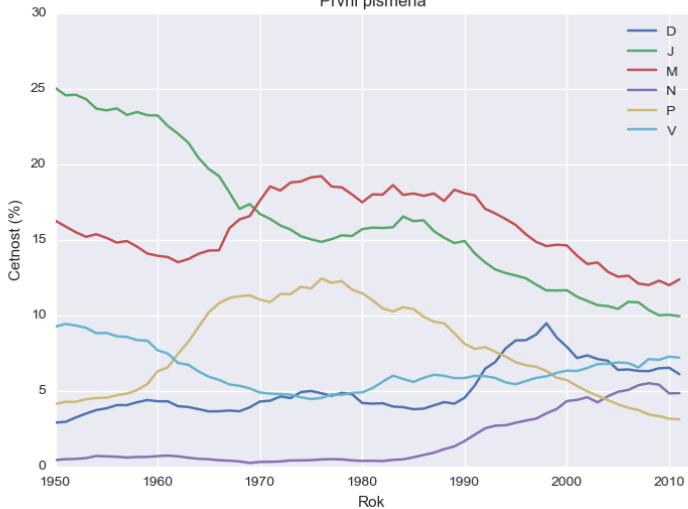
```
{('Alois', 1951): 131, ('Adam', 1950): 15,  
 ('Alois', 1950): 332, ('Adam', 1951): 10}  
print(data[name, year])
```

JAN, JAROSLAV, JIŘÍ, MAREK, MATĚJ, NIKOLA,
 ONDŘEJ, RADEK, TOMÁŠ, VALDEMAR



Prvni pismena



Co zajímavého můžeme z dat zjistit?

Kladení otázek – důležitá dovednost hodná tréninku.

Computers are useless. They can only give you answers. (Pablo Picasso)

U kterých jmen nejvíce roste/klesá popularita?

- co to vlastně znamená?
- jak formalizovat?

Nejdelší růst/pokles

Kolik let v řadě roste popularita jména:

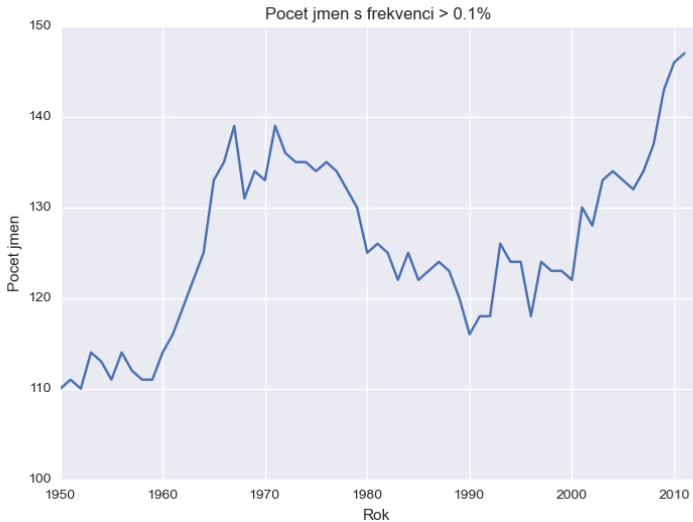
- Tobiáš – 14
- Viktorie, Ella, Sofie – 9
- Elen, Tobias – 8

Kolik let v řadě klesá popularita jména:

- Jana – 26
- Martin – 21
- Petra – 11
- Zdeněk – 9

Největší skok v popularitě za 10 let

- alespoň desetinásobný nárůst popularity:
Sofie, Elen, Amálie, Ella, Nicol, Nella, Tobias
- pokles alespoň o 60 %:
Petra, Pavlína, Martina



Zpracování dat seriózněji

využití existujících knihoven:

- načítání dat ve standardních formátech: HTML, XML, JSON, CSV, ...
- operace s daty: numpy, pandas
- vizualizace: matplotlib
- interaktivní prozkoumávání dat: ipython, jupyter

Doporučené procvičování

<https://www.umimeprogramovat.cz/regularni-vyrazy>

Kontrolní otázky

- K čemu slouží regulární výrazy?
- Jaký je v regulárních výrazech význam +, *, ., \s, \d, \w, [ab], \$?
- Jak používáme regulární výrazy v Pythonu?
- Pokud mám tabulku číselných údajů, jak ji mohu načíst, reprezentovat a zpracovat?

- regulární výrazy
- práce s textem, daty
- využití datových struktur