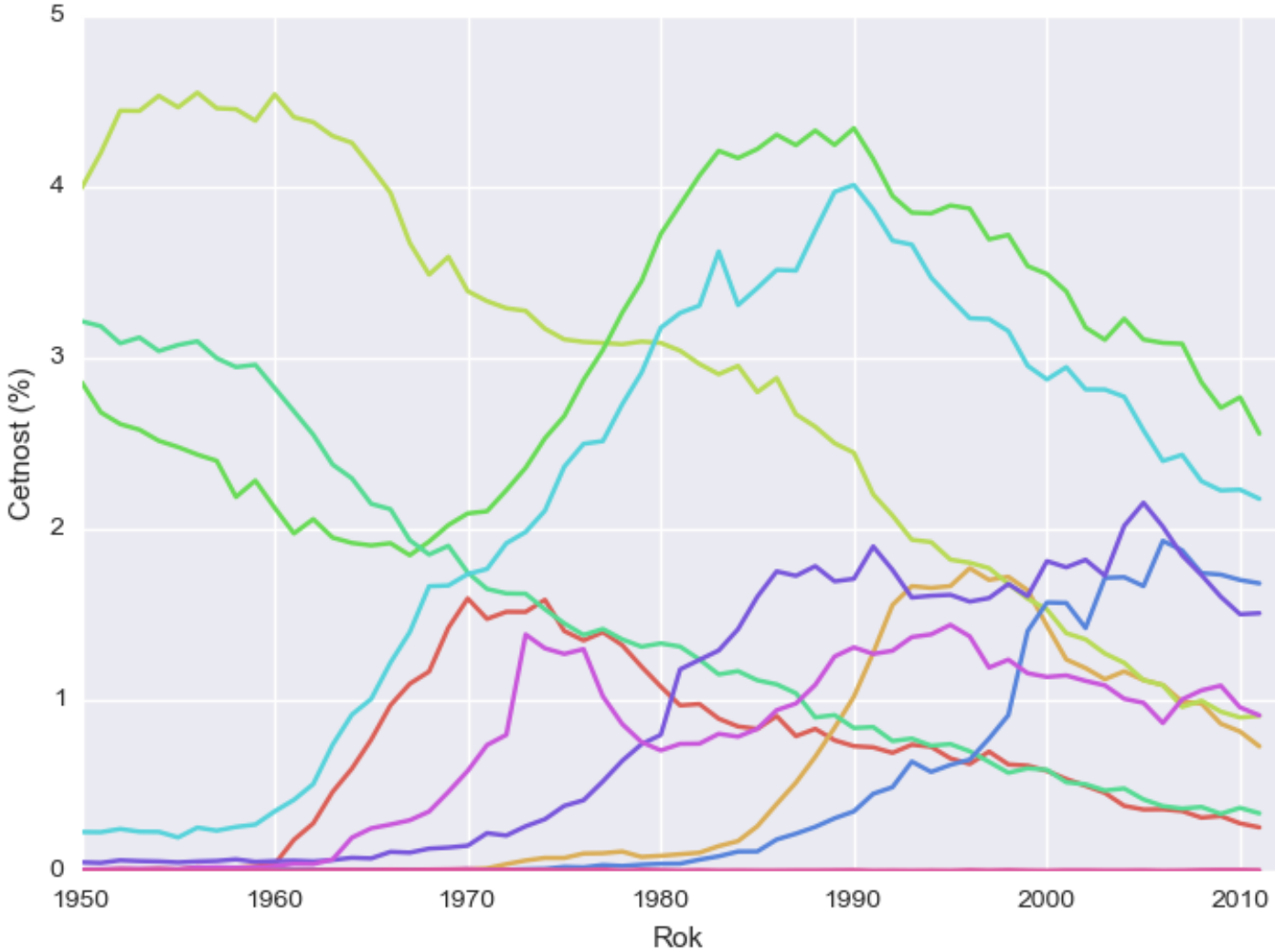
Práce s daty

IB111 Základy programování
Radek Pelánek

2018



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

Práce s daty

připomenutí

- Jaká data budu zpracovávat?
- Jaká data budu potřebovat k řešení problému?
- Jaké operace s daty budu chtít provádět?
- Jak rychle budu chtít, aby operace s daty fungovaly?

Dnešní přednáška

- použití datových struktur na praktických příkladech
- interakce volby dat a algoritmu
- vnořené datové struktury: seznam seznamů, slovník indexovaný dvojicí, slovník slovníků…
- práce s textem, soubory

Práce se soubory

Proč?

- vstupní data
- uložení výstupu programu
- zachování „stavu“ programu mezi jednotlivými běhy

větší projekty: databáze

Práce se soubory

základní postup:

- otevření souboru
- práce se souborem (čtení / zápis)
- zavření souboru

Práce se soubory: otevření, uzavření

- `f = open(filename, mode)`
- jméno souboru: řetězec
- způsob otevření:
  - čtení (`"r"`)
  - zápis (`"w"`) – přepíše soubor, pokud není, vytvoří jej
  - přidání na konec (`"a"`)
  - další možnosti: čtení i zápis, binární režim
- uzavření: `f.close()`

Práce se soubory: čtení, zápis

- `f.write(text)` – zapíše řetězec do souboru
  - neukončuje řádky, je třeba explicitně použít `'\n'`
- `f.readline()` – přečte jeden řádek
- `f.readlines()` – přečte všechny řádky, vrací seznam řádků
- `f.read(count)` – přečte daný počet znaků
- `f.read()` – přečte celý soubor, vrací řetězec
- `f.tell()` – aktuální pozice v souboru
- `f.seek(position)` – přesun pozice v souboru

Práce se soubory: iterování po řádcích

```
for line in f.readlines():
    print(line)
```

Alternativní způsob:

```
for line in my_file:
    print(line)
```

Frekvence slov v souboru

```
def word_freq_file(filename):
    f = open(filename)
    text = ""
    for line in f.readlines():
        text += line
    output_word_freq(text)     # -> previous lecture
    f.close()


word_freq_file("devatero_pohadek.txt")
```

Frekvence slov v souboru: úkol

Upravte výpis, aby vypisoval nejčastější slova zadané minimání délky.

a	2426	jsem	287
se	1231	jako	284
na	792	když	281
to	781	řekl	251
v	468	nebo	120
že	467	ještě	110
je	446	pane	109
si	401	toho	91
tak	384	povídá	83
do	365	král	80

Práce se soubory: with

Speciální blok `with`

- není třeba soubor zavírat (uzavře se automaticky po ukončení bloku)
- souvislost s výjimkami

```
with open("/tmp/my_file", "r") as my_file:
    lines = my_file.readlines()

print(lines)
```

Zápis do souboru

Přímočará realizace cenzury dlouhých slov:

```
def censorship(filename, limit=5):
    f_in = open(filename)
    f_out = open("censor_"+filename, "w")
    for line in f_in:
        out_line = ""
        for word in line.split():
            if len(word) < limit:
                out_line += word + " "
            else:
                out_line += "XXX "
        f_out.write(out_line+"\n")
    f_in.close()
    f_out.close()
```

Ukázka vstupu a výstupu

report.txt:

```
Climate models project robust 7 differences in regional climate
characteristics between present-day and global warming of 1.5 degree C,
and between 1.5 degree C and 2 degree C. These differences include
increases in: mean temperature in most land and ocean regions (high
confidence), hot extremes in most inhabited regions (high confidence),
heavy precipitation in several regions (medium confidence), and the
probability of drought and precipitation deficits in some regions (medium
confidence).
```

censor_report.txt:

```
XXX XXX XXX XXX 7 XXX in XXX XXX
XXX XXX XXX and XXX XXX of 1.5 XXX C,
and XXX 1.5 XXX C and 2 XXX C. XXX XXX XXX
XXX in: mean XXX in most land and XXX XXX XXX
XXX hot XXX in most XXX XXX XXX XXX
XXX XXX in XXX XXX XXX XXX and the
XXX of XXX and XXX XXX in some XXX XXX
XXX
```

Námět na cvičení: zachování interpunkce (tečky, čárky, závorky).

Případová studie: Číselné bludiště

jednoduchá logická úloha

Ilustrace pojmů a postupů:

- návrh algoritmu
- volba datových struktur
- seznam seznamů
- slovník
- fronta
- načítání ze souboru
- objekty

Číselné bludiště

levý horní roh $\rightarrow$ pravý dolní roh
skoky vertikálně a horizontálně, číslo = délka skoku

2	4	4	3	3
2	3	3	2	3
3	2	3	1	3
2	2	3	2	1
1	4	4	4	☆

3	3	1	3	2
1	1	2	1	3
1	3	1	2	4
4	3	4	1	4
4	1	4	4	☆

4	2	1	2	3
1	2	2	2	3
1	4	4	3	3
3	4	3	1	4
3	2	3	4	☆

K vyzkoušení: `www.umimematiku.cz/skakacka`

Variace

- nejkratší cesta
  - vzhledem k počtu skoků
  - vzhledem k celkové délce cesty
- kontrola jednoznačnosti nejkratšího řešení
- generování „co nejtěžšího“ zadání

Reprezentace bludiště

Jak budeme v programu reprezentovat bludiště?

2	4	4	3	3
2	3	3	2	3
3	2	3	1	3
2	2	3	2	1
1	4	4	4	☆

zadani.txt:

```
5
24433
23323
32313
22321
14440
```

Seznam seznamů

„klasická“ reprezentace – dvojrozměrné pole (seznam seznamů v Pythonu)

```
maze[x][y] = number
```

```
[[2, 2, 3, 2, 1], [4, 3, 2, 2, 4],
 [4, 3, 3, 3, 4], [3, 2, 1, 2, 4],
 [3, 3, 3, 1, 0]]
```

2	4	4	3	3
2	3	3	2	3
3	2	3	1	3
2	2	3	2	1
1	4	4	4	☆

Poznámka: `maze[x][y]` nebo `maze[y][x]`?
(častý zdroj chyb: nekonzistence)

Načítání ze souboru

```
def read_maze(filename="input.txt"):
    f = open(filename)
    n = int(f.readline())
    maze = [[0 for y in range(n)]
            for x in range(n)]
    for y in range(n):
        line = f.readline()
        for x in range(n):
            maze[x][y] = int(line[x])
    f.close()
    return maze
```

Algoritmus

Jak najít řešení?

2	4	4	3	3
2	3	3	2	3
3	2	3	1	3
2	2	3	2	1
1	4	4	4	☆

Algoritmus

speciální případ obecného „prohledávání do šířky“:

- systematicky zkoušíme všechny následníky
- pamatujeme si, kde už jsme byli
- pro každé pole si pamatujeme předchůdce (rekonstrukce cesty)

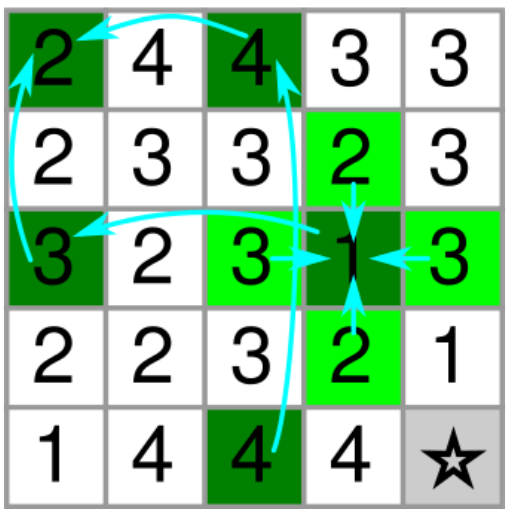
Algoritmus


2 4 4 3 3
2 3 3 2 3
3 2 3 1 3
2 2 3 2 1
1 4 4 4 ☆


2 4 4 3 3
2 3 3 2 3
3 2 3 1 3
2 2 3 2 1
1 4 4 4 ☆


2 4 4 3 3
2 3 3 2 3
3 2 3 1 3
2 2 3 2 1
1 4 4 4 ☆


2 4 4 3 3
2 3 3 2 3
3 2 3 1 3
2 2 3 2 1
1 4 4 4 ☆


2 4 4 3 3
2 3 3 2 3
3 2 3 1 3
2 2 3 2 1
1 4 4 4 ☆

...

Datové struktury

co si potřebujeme pamatovat:

- *fronta* polí, které musíme prozkoumat (světle zelená pole)
- *množina* polí, která už jsme navštívili (tmavě zelená pole)
- informace o předchůdcích (světle modré šipky)

optimalizace: podle informace o předchůdcích poznáme, kde už jsme byli

Zápis v Pythonu – přímočaré řešení

```
def solve(n, maze):
    start = (0, 0)
    queue = deque([start])
    pred = [[(-1, -1) for x in range(n)]
            for y in range(n)]

    while len(queue) > 0:
        (x, y) = queue.popleft()
        if maze[x][y] == 0:
            print_solution((x, y), pred)
            break
        k = maze[x][y]
        if x+k < n and pred[x+k][y] == (-1, -1):
            queue.append((x+k, y))
            pred[x+k][y] = (x, y)
        if x-k >= 0 and pred[x-k][y] == (-1, -1):
            queue.append((x-k, y))
            pred[x-k][y] = (x, y)
        if y+k < n and pred[x][y+k] == (-1, -1):
            queue.append((x, y+k))
            pred[x][y+k] = (x, y)
        if y-k >= 0 and pred[x][y-k] == (-1, -1):
            queue.append((x, y-k))
            pred[x][y-k] = (x, y)
```

Poznámky

- příkaz `break` – opuštění cyklu
- využití zkráceného vyhodnocování

Alternativní reprezentace: slovník

slovník indexovaný dvojicí
souřadnice $(x, y) \rightarrow$ číslo na dané souřadnici

```
maze[(x, y)] = number
```

```
{(1, 2): 2, (3, 2): 1, (0, 0): 2,
 (3, 0): 3, (2, 2): 3, (2, 1): 3,
 (1, 3): 2, (2, 3): 3, (1, 4): 4,
 (2, 4): 4, (4, 2): 3, (0, 3): 2,
 ... }
```

2	4	4	3	3
2	3	3	2	3
3	2	3	1	3
2	2	3	2	1
1	4	4	4	☆

N-tice a závorky

U n-tic většinou nemusíme psát závorky:

- `a, b = b, a`
  místo
  `(a, b) = (b, a)`
- `maze[x, y]`
  místo
  `maze[(x, y)]`

Slovník vs seznam seznamů

Pozor na rozdíl!

- `maze[x][y]` – seznam seznamů
- `maze[x, y]` – slovník indexovaný dvojicí

Další vylepšení

Zobecnění repetitivního kódu:

„copy&paste“ kód pro 4 směry

$\Downarrow$

`for` cyklus přes 4 směry

Předchůdci

- pamatujeme si rovnou celou cestu (*)
- také slovník

(*) to je sice „plýtvání pamětí“, ale vzhledem k velikosti zadání nás to vůbec nemusí trápit

Upravený kód

```
def solve(maze):
  start = (0, 0)
  queue = deque([start])
  pred = {}
  pred[start] = [start]
  while len(queue) > 0:
    (x, y) = queue.popleft()
    if maze[x, y] == 0:
      print(pred[x,y])
    for (dx, dy) in [(-1,0), (1,0), (0,-1), (0,1)]:
      newx = x + dx * maze[x, y]
      newy = y + dy * maze[x, y]
      if (newx, newy) in maze and \
         not (newx, newy) in pred:
        queue.append((newx, newy))
        pred[newx, newy] = pred[x, y] + [(newx, newy)]
```

Poznámky

Někdy jsou závorky důležité:

- `s = [1, 2]` – seznam obsahující dva prvky
  `s = [(1, 2)]` – seznam obsahující jeden prvek (dvojici)
- `s.append((1, 2))` – přidávám dvojici
  `s.append(1, 2)` – volám append se dvěma argumenty (chyba)

Objektová reprezentace

```
class Maze:

  def __init__(self):
    # initialize

  def read(self, filename):
    # read maze from file

  def solve(self):
    # find solution

  def print_solution(self):
    # text output

  def save_image(self, filename, with_solution=True):
    # save maze as an image
```

Objektová reprezentace

- pro základní úlohu není nezbytné
- velmi vhodné pro rozšíření, např. „hledání těžkých zadání“ (potřeba pracovat s více zadáními současně)
- *doporučené cvičení*

Piškvorky

herní plán:

- plán omezené velikosti
- neomezený plán

„chytrý“ algoritmus?

Práce s plánem

- globální proměnné
  - `draw_board()`
  - `make_move(x, y)`
- předávání funkcím
  - `draw_board(board, size)`
  - `make_move(board, size, x, y, player)`
- objektová reprezentace
  - `game.draw_board()`
  - `game.make_move(x, y)`

Implementace strategií: nevhodný styl

```
def strategy(num):
    if num == 0: # random strategy
        x = randint(1, N)
        y = randint(1, N)
    elif num == 1: # first empty
        for x in range(N):
            for y in range(N):
                s = board[x][y]
                # and so on
    elif num == 2: # clever
        # and so on...
```

```
def strategy_random(board):
    x = randint(1, board.size)
    y = randint(1, board.size)
    return x, y

def strategy_first_empty(board):
    for x in range(board.size):
        for y in range(board.size):
            s = board[x][y]
            # and so on...

strategies = {"random": strategy_random,
              "first": strategy_first_empty,
              "clever": strategy_clever}

x, y = strategies[player1_strategy](board)
```

Piškvorky: reprezentace strategií

ještě lepší přístup: objektová reprezentace

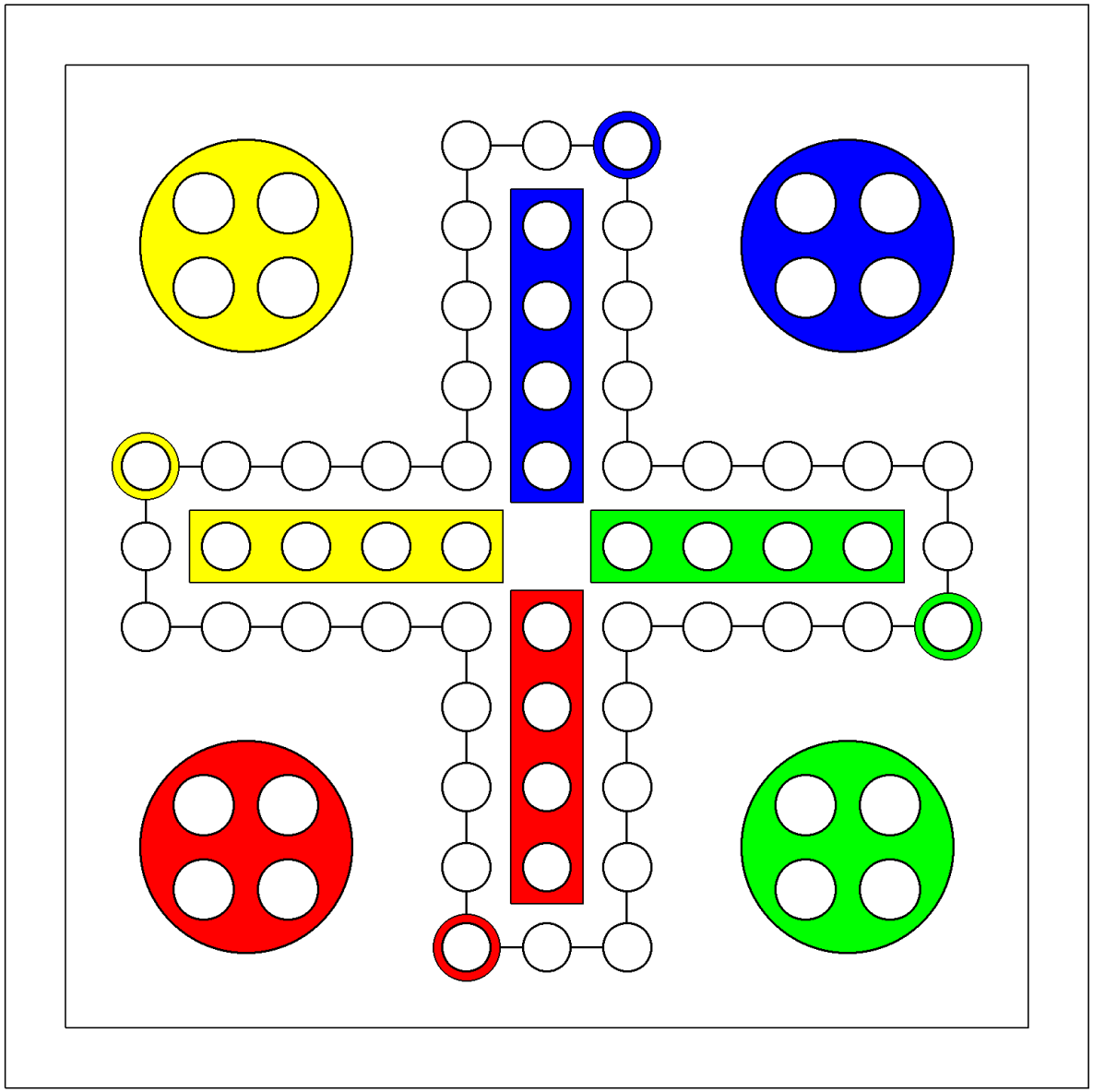
- reprezentace paměti, historie
- využití dědičnosti (pokročilejší koncept z OOP)
  - hierarchie strategií
  - eliminace repetitivního kódu

Volba datové struktury

Jaká datová struktura pro reprezentaci stavu?

- Člověče, nezlob se!
- Želví závody
- Pexeso

Člověče, nezlob se!



Želví závody

Pexeso – simulace paměti

- simulace hry pexeso
- hráči s různě dokonalou pamětí:
  - prázdná paměť (náhodná hra)
  - dokonalá paměť
  - omezená paměť – „buffer“ velikosti k
- pravděpodobnost výhry pro jednotlivé hráče

- reprezentace stavu hry (kartiček)?
- reprezentace paměti?

Kahoot

Kahoot otázky: 1, 2

```
a = [1, 2, 3]
b = [(1, 2, 3)]
c = [(1, 2), 3]
```

Kahoot otázky: 3, 4

```
data = {"a": [1, 3, 7, 12],
        "c": [2, 5],
        "x": [19, 2, 4]}
data["d"] = [1, 2]
data["c"] = [6, 7]
```

Kahoot otázky: 5, 6, 7

```
d1 = {"a": 5, "c": "R2D2"}
d2 = {"b": 8, "a": 7, "B": "C3PO"}
s = [d1, d2]
t = "BB8"
```

Kahoot otázky: 8

```
board = {(3, 0, 1): "X",
         (1, 2, 2): "O",
         (2, 4, 3): "X",
         (3, 0, 2): "O"}
```

Kahoot otázky: 9

```
x = []
nums = []
for i in range(3):
    nums.append(x)
    x.append(i)
```

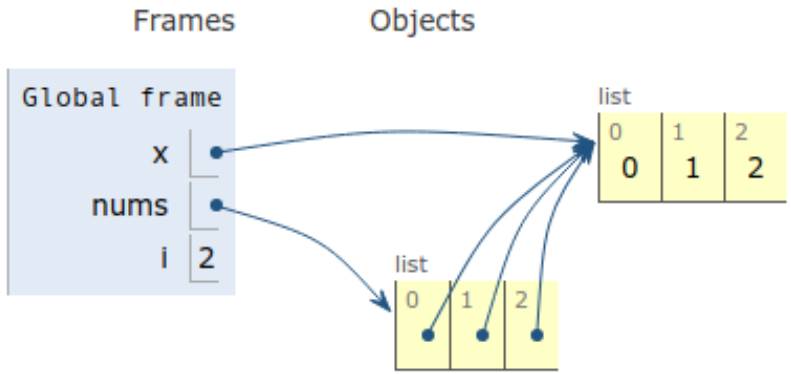
Vizualizace Python Tutor

Python 3.6

```
1  x = []
2  nums = []
3  for i in range(3):
4      nums.append(x)
5      x.append(i)
```

Edit code | Live programming

ıted



Práce s textem

- operace s textem – připomenutí a rozšíření
- ukázka zpracování

Rozdělení a spojení řetězce

- `split` – rozdělí řetězec podle zadaného podřetězce, vrací seznam částí
- `join` – spojení seznamu řetězců do jednoho

```
>>> text = "Holka modrooka nesedavej u potoka"
>>> text.split()
['Holka', 'modrooka', 'nesedavej', 'u', 'potoka']
>>> text.split('o')
['H', 'lka m', 'dr', '', 'ka nesedavej u p', 't', 'ka']
>>> text.split('ka')
['Hol', ' modroo', 'nesedavej u poto', '']
```

Změna vs. návratová hodnota

častá chyba (cvičení z předchozích let): použití `split`, ignorování návratové hodnoty

`split`:

- je metoda třídy řetězec
- nemění objekt
  - ani nemůže, řetězce jsou neměnné
- vrací jako návratovou hodnotu seznam řetězců

typický příklad užití:

```
word_list = sentence.split()
```

Řetězce: další funkce

- `find`, `count` – vyhledávání a počítání podřetězců
- `replace` – nahrazení podřetězce
- `lower`, `upper` – převod na malá/velká písmena
- `ljust`, `rjust`, `center` – zarovnání textu
- `lstrip`, `rstrip` – ořezání bílých znaků na začátku/konci

Pozn. objektová notace, metody

Analýza textu

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

$\Rightarrow$ *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

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 řečekovících 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?

Statistiky textu

- 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)

Implementace

- 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“

Reprezentace dat

- 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])
```

Imitace 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 sofistikovaněji

- Recurrent Neural Networks – dokáží postihnout i složitější 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.
```

Statistiky jmen

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

  `http://www.mvcr.cz/clanek/cetnost-jmen-a-prijmeni-722752.aspx`
- XLS – pro zpracování v Pythonu uložit jako CSV (comma-separated values)
- profiltrovaná data (nejčastější jména):

  `docs.google.com/spreadsheets/d/1tbhl-sHEnDOUnbxXOBXf9CaWJb50tyyOabpOHX6gioc`
- **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` $\rightarrow$ 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

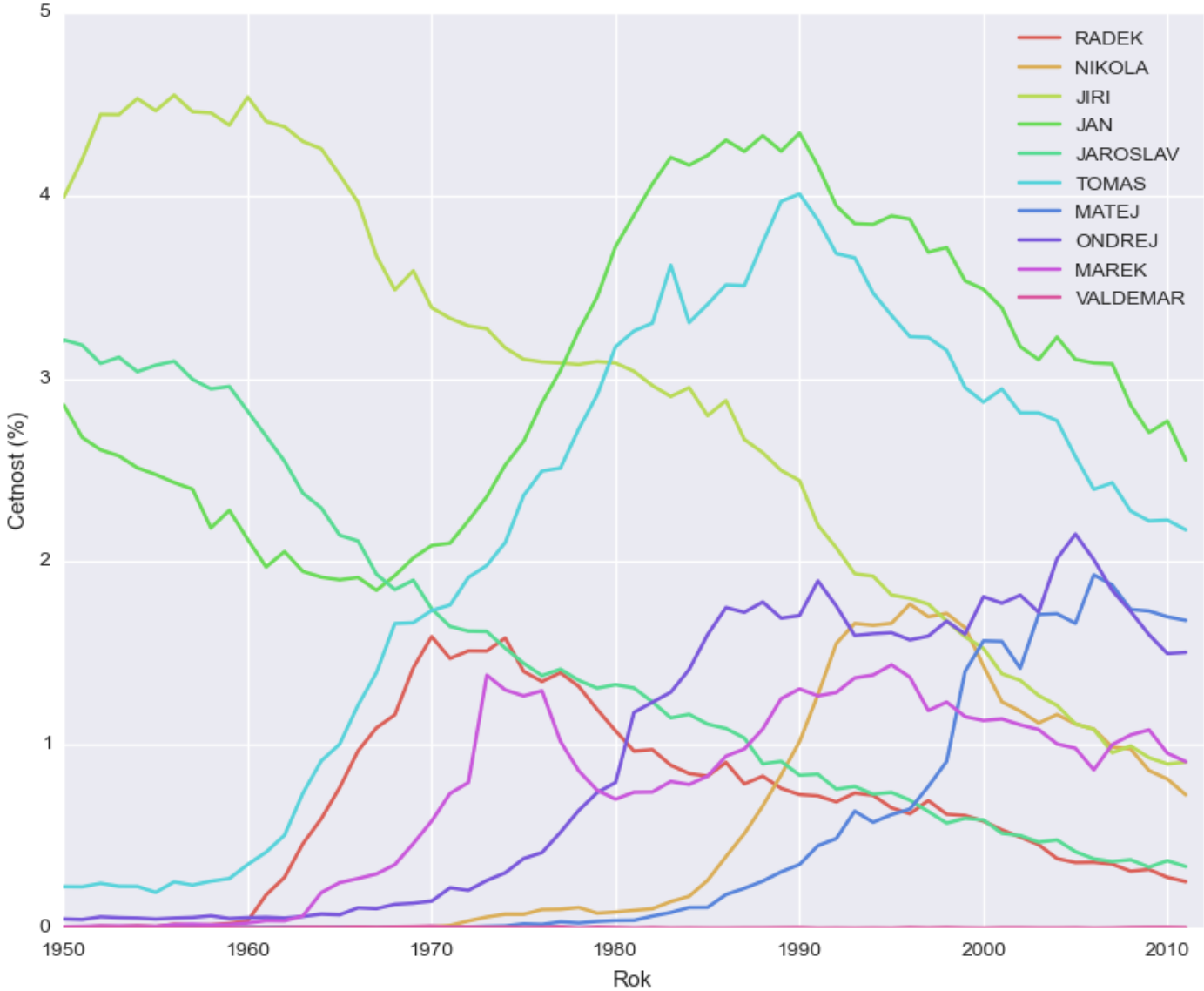
```
{'Alois': [332, 131, 112], 'Adam': [15, 10, 17]}
print(data[name][year-1950])
```

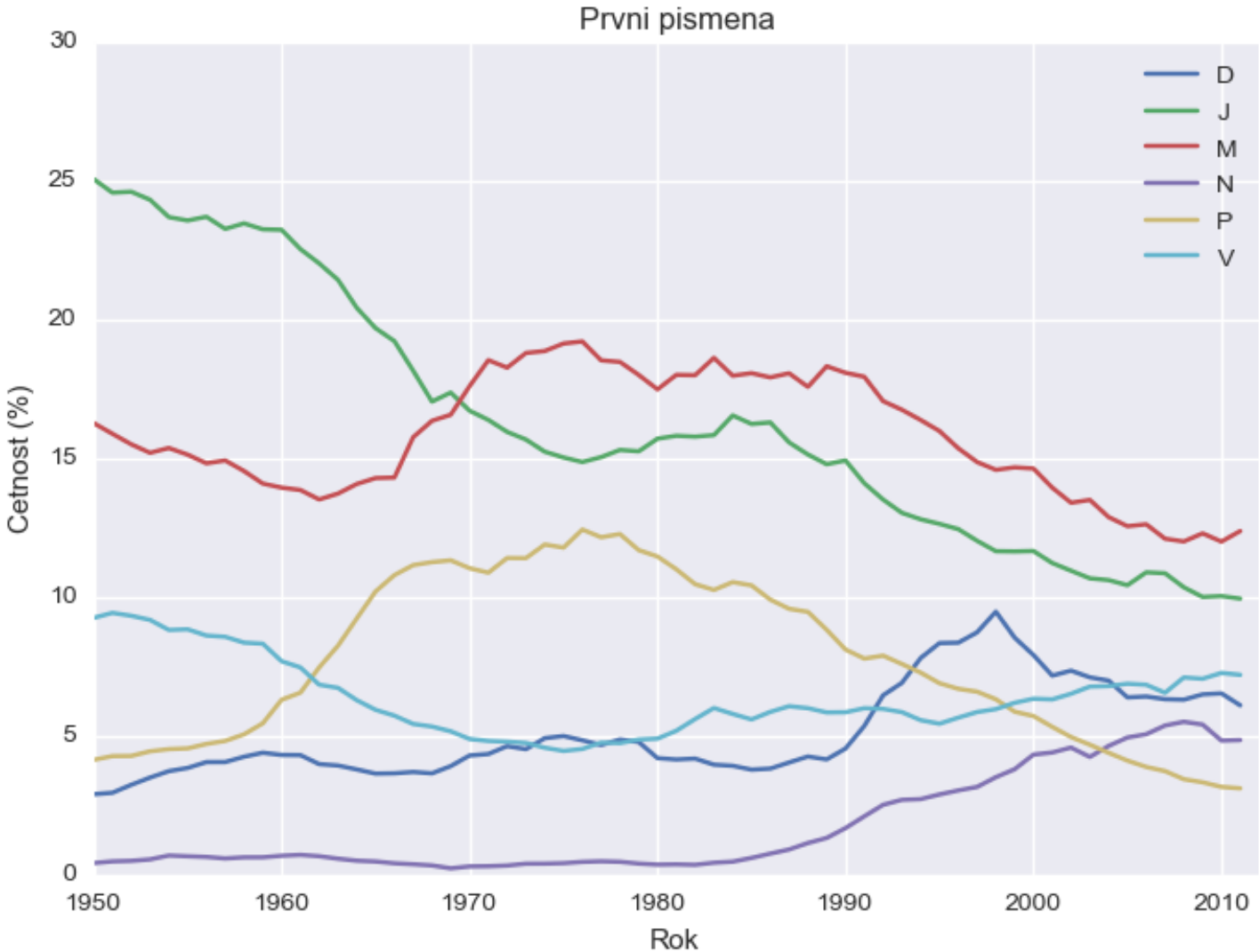
slovník mapující jména na slovníky mapující roky na počty

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

slovník mapující dvojici jméno+rok na počet výskytují

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


RADEK
NIKOLA
JIRI
JAN
JAROSLAV
TOMAS
MATEJ
ONDREJ
MAREK
VALDEMAR
Cetnost (%)
Rok
0
1
2
3
4
5
1950
1960
1970
1980
1990
2000
2010


Prvni pismena
Cetnost (%)
Rok
D
J
M
N
P
V
0
5
10
15
20
25
30
1950
1960
1970
1980
1990
2000
2010

Otázky

Co zajímavého můžeme z dat zjistit?

Kladení otázek – důležitá dovednost hodná tréninku.

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

Identifikace trendů

U kterých jmen nejvíce roste/klesá popularita?

- co to vlastně znamená?
- jak formalizovat?

Nejdelší růst/pokles

Kolik let v řadě roste popularita jména:

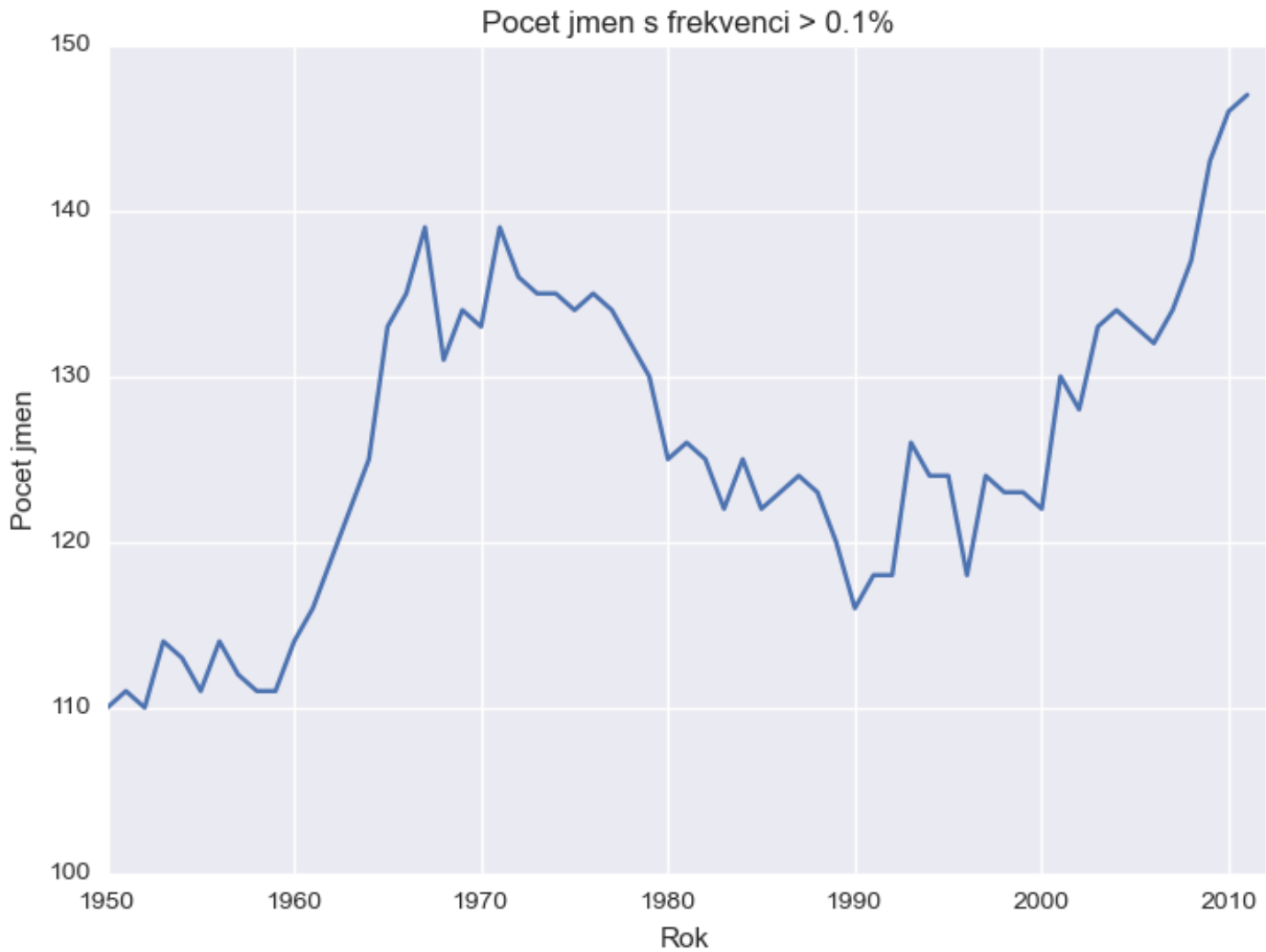
- Tobiáš – 14
- Viktorie, Ella, Sofie – 9
- Elen, Tobias – 8

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


Pocet jmen s frekvenci > 0.1%
150
140
130
120
110
100
Pocet jmen
1950
1960
1970
1980
1990
2000
2010
Rok

Zdroje zajímavých dat

Otevřená data / Open data

- `https://www.opendata.cz`
- `http://www.otevrenadata.cz`
- `https://www.data.gov`
- `https://data.gov.uk`

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`

Shrnutí

hry, zpracování textu

- jaká data potřebuji reprezentovat?
- jak budu data reprezentovat? (volba datových struktur)
- co se těmi daty budu dělat? (návrh algoritmu)

zmíněna řada doporučených cvičení – programujte!