

Seznamy

IB113
Radek Pelánek

2019

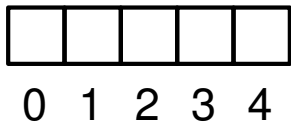
Seznamy (pole) – motivace

- řazení studentů podle bodů na písemce
- reprezentace herního plánu (piškvorky, šachy)
- frekvence písmen v textu

Frekvenční analýza nevhodně

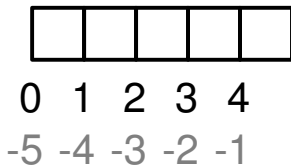
```
def frequency_analysis(text):  
    text = text.upper()  
    freqA = 0  
    freqB = 0  
    freqC = 0  
    for letter in text:  
        if letter == 'A':  
            freqA += 1  
        elif letter == 'B':  
            freqB += 1  
        elif letter == 'C':  
            freqC += 1  
    print('A', freqA)  
    print('B', freqB)  
    print('C', freqC)
```

Seznamy (pole)



- „více položek za sebou v pevném pořadí“
- indexováno od nuly
- základní koncept dostupný ve všech jazycích, běžně „pole“ (array), položky stejného typu, pevně daná délka
- seznamy v Pythonu – obecnější (ale pomalejší)
- Python a pole – knihovna NumPy (nad rámec IB113)

Seznamy v Pythonu



- variabilní délka
- položky mohou být různého typu
- indexování i od konce (pomocí záporných čísel)

Seznamy: použití v Pythonu

```
s = []          # deklarace prázdného seznamu
s = [3, 4, 1, 8]
s[2]           # indexace prvku, s[2] = 1
s[-1]         # indexace od konce, s[-1] = 8
s[2] = 15     # změna prvku
s.append(6)   # přidání prvku na konec
s[1:4]        # indexace intervalu, s[1:4] = [4, 15, 8]
len(s)        # délka seznamu, len(s) = 5
t = [3, "pes", [2, 7], -8.3]
              # seznam může obsahovat různé typy
```

list() – přetypování na seznam

Manipulace se seznamy

```
alist = [3, 8, 7]
alist.append(10)      # přidání na konec seznamu
alist.insert(1, 11)  # přidání na zadanou pozici
alist.remove(7)      # odstranění dané hodnoty
```

Seznamy: konvence zápisu (PEP8)

- mezera se dělá: za čárkou
- mezera se nedělá: před čárkou, na „okrajích“

Python: seznamy a cyklus for

- cyklus for – přes prvky seznamu*
- range – vrací seznam* čísel
- typické použití: `for i in range(n)`
- ale můžeme třeba:
 - `for animal in ["dog", "cag", "pig"]: ...`
 - `for letter in "hello world": ...`

(*) ne úplně přesně – z důvodu efektivity se používají generátory a speciální „range object“, v případě potřeby použijte explicitní přetypování na seznam: `list(range(10))`

Příklad: výpočet průměrné hodnoty

```
def average1(num_list):  
    total = 0  
    for i in range(len(num_list)):  
        total += num_list[i]  
    return total / len(num_list)
```

```
def average2(num_list):  
    total = 0  
    for x in num_list:  
        total += x  
    return total / len(num_list)
```

```
def average3(num_list):  
    return sum(num_list) / len(num_list)
```

Vestavěná podpora pro práci se seznamy

```
>>> numbers = [4, 1, 8, 12, 3]
>>> sum(numbers)
28
>>> min(numbers)
1
>>> max(numbers)
12
>>> 8 in numbers
True
```

Ilustrace práce se seznamem

```
def divisors_list(n):  
    divisors = []  
    for i in range(1, n+1):  
        if n % i == 0:  
            divisors.append(i)  
    return divisors
```

```
divisors24 = divisors_list(24)  
print(divisors24)  
print(len(divisors24))  
for x in divisors24: print(x**2)
```

Vytvoření seznamu

různé způsoby vytvoření seznamu písmen abecedy:

```
alist = list("abcdefghijklmnopqrstuvwxyz")
```

```
alist = []  
for i in range(26):  
    alist.append(chr(ord('a')+i))
```

```
alist = [chr(ord('a')+i) for i in range(26)]
```

Frekvenční analýza nevhodně

```
def frequency_analysis(text):  
    text = text.upper()  
    freqA = 0  
    freqB = 0  
    freqC = 0  
    for letter in text:  
        if letter == 'A':  
            freqA += 1  
        elif letter == 'B':  
            freqB += 1  
        elif letter == 'C':  
            freqC += 1  
    print('A', freqA)  
    print('B', freqB)  
    print('C', freqC)
```

Frekvenční analýza lépe

```
def frequency_analysis(text):
    text = text.upper()
    frequency = [0 for i in range(26)]
    for letter in text:
        if ord(letter) >= ord('A') and\
            ord(letter) <= ord('Z'):
            frequency[ord(letter) - ord('A')] += 1
    for i in range(26):
        if frequency[i] != 0:
            print(chr(ord('A')+i), frequency[i])
```

Simulace volebního průzkumu – nevhodné řešení

```
def survey(size, pref1, pref2, pref3):  
    count1 = 0  
    count2 = 0  
    count3 = 0  
    for i in range(size):  
        r = random.randint(1, 100)  
        if r <= pref1: count1 += 1  
        elif r <= pref1 + pref2: count2 += 1  
        elif r <= pref1 + pref2 + pref3: count3 += 1  
    print("Party 1:", 100 * count1 / size)  
    print("Party 2:", 100 * count2 / size)  
    print("Party 3:", 100 * count3 / size)
```


Simulace volebního průzkumu – lepší řešení

```
def survey(size, pref):
    n = len(pref)
    count = [0 for i in range(n)]
    for _ in range(size):
        r = random.randint(1, 100)
        for i in range(n):
            if sum(pref[:i]) < r <= sum(pref[:i+1]):
                count[i] += 1
    for i in range(n):
        print("Party", i+1, 100*count[i]/size)
```

Toto řešení má stále nedostatky (po stránce funkčnosti) – zkuste dále vylepšit.

Převod do Morseovy abecedy

- vstup: řetězec
- výstup: zápis v Morseově abecedě
- příklad: PES \rightarrow .-- . | . | . . .

Převod do Morseovy abecedy nevhodně

```
def to_morse(text):  
    result = ''  
    for i in range(len(text)):  
        if text[i] == 'A': result += '.-|'  
        elif text[i] == 'B': result += '-...|'  
        elif text[i] == 'C': result += '-.-.|'  
        elif text[i] == 'D': result += '-..|'  
        # etc  
    return result
```

Převod do Morseovy abecedy: využití seznamu

```
morse = ['.-', '-...', '-.-.', '-..'] # etc
```

```
def to_morse(text):  
    result = ''  
    for i in range(len(text)):  
        if ord('A') <= ord(text[i]) <= ord('Z'):  
            c = ord(text[i]) - ord('A')  
            result += morse[c] + ' | '  
    return result
```

(ještě lepší řešení: využití slovníku – bude později)

Převod z Morseovy abecedy

```
def find_letter(sequence):
    for i in range(len(morse)):
        if morse[i] == sequence:
            return chr(ord('A') + i)
    return '?'

def from_morse(message):
    result = ''
    sequence = ''
    for symbol in message:
        if symbol == '|':
            result += find_letter(sequence)
            sequence = ''
        else:
            sequence += symbol
    return result
```

Split – seznam z řetězce

split – rozdělí řetězec podle zadaného oddělovače, vrátí seznam

```
>>> vowels = "a,e,i,o,u,y"
>>> vowels.split(",")
['a', 'e', 'i', 'o', 'u', 'y']
>>> message = ".-..|---|-.|.-"
>>> message.split("|")
['.-..', '---', '-.', '.-']
```

Příklad – načítání vstupu od uživatele

```
>>> input_string = input()
3 7
>>> xstring, ystring = input_string.split(" ")
>>> x = int(xstring)
>>> y = int(ystring)
```

Výškový profil



mapy.cz

Výškový profil

```
heights_profile([3, 4, 5, 3, 4, 3, 2, 4, 5, 6, 5])
```

```
                #
             #   # # #
          # #   #   # # # #
        # # # # # # # # # # #
      # # # # # # # # # # # #
    # # # # # # # # # # # #
```

Ascent 7

Descent 5

Výškový profil

```
def heights_profile(heights):  
    for v in range(max(heights)):  
        for i in range(len(heights)):  
            if heights[i] >= max(heights) - v:  
                print("#", end=" ")  
            else:  
                print(" ", end=" ")  
        print()  
    print()
```

Výškový profil

```
def elevation(heights):  
    ascent = 0  
    descent = 0  
    for i in range(len(heights)-1):  
        if heights[i] < heights[i+1]:  
            ascent += heights[i+1] - heights[i]  
        else:  
            descent += heights[i] - heights[i+1]  
    print("Ascent", ascent)  
    print("Descent", descent)
```

Objekty, hodnoty, aliasy – stručné varování

a = [1, 2, 3]
b = [1, 2, 3] nebo b = a[:]

a → [1, 2, 3]
b → [1, 2, 3]

a = [1, 2, 3]
b = a

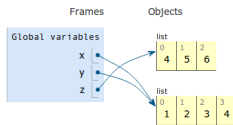
a → [1, 2, 3]
b ↗ [1, 2, 3]

- parametry funkcí – pouze volání hodnotou (na rozdíl např. od Pascalu: volání hodnotou a odkazem)
- měnitelné objekty (např. seznam) však funkce může měnit
- mělká vs hluboká kopie
- více později

Vizualizace běhu programu

<http://www.pythontutor.com/>

```
1 x = [1, 2, 3]
2 y = [4, 5, 6]
3 z = y
4 y = x
5 x = z
6
7 x = [1, 2, 3] # a different [1, 2, 3] list!
8 y = x
→ 9 x.append(4)
→ 10 y.append(5)
11 z = [1, 2, 3, 4, 5] # a different list!
12 x.append(6)
13 y.append(7)
14 y = "hello"
--
```



vhodné např. pokud je nejasný některý z příkladů ve slidech

Zdvojnásobení seznamu

Příklad: vstupem seznam čísel (např. [4, 1, 6]), chceme „získat dvojnásobky“ (tj. 8, 2, 12)

Důležité ujasnit, co přesně chceme:

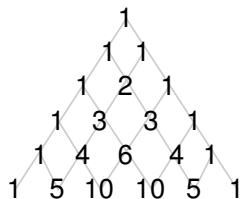
- vypsát dvojnásobky
- vrátit nový seznam, který obsahuje dvojnásobky
- změnit seznam, aby obsahoval dvojnásobky

Jak vypadají jednotlivé programy?

N-tice - stručné představení

- n-tice (tuples)
- neměnitelná varianta seznamů
- kulaté závorky místo hranatých (někdy lze vynechat):
 $t = (1, 2, 3)$
- neměnitelné podobně jako řetězce
- typické užití:
 - souřadnice: (x, y)
 - barva: (r, g, b)
- použití pro přiřazení: $a, b = b, a$

Pascalův trojúhelník



$$\begin{array}{c} \binom{0}{0} \\ \binom{1}{0} \quad \binom{1}{1} \\ \binom{2}{0} \quad \binom{2}{1} \quad \binom{2}{2} \\ \binom{3}{0} \quad \binom{3}{1} \quad \binom{3}{2} \quad \binom{3}{3} \end{array}$$

Explicitní vzorec

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

Rekurzivní vztah

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Pascalův trojúhelník

```
def get_next_row(row):  
    next_row = [1]  
    for i in range(len(row)-1):  
        next_row.append(row[i]+row[i+1])  
    next_row.append(1)  
    return next_row  
  
def pascal_triangle(n):  
    row = [1]  
    for i in range(n):  
        print(row)  
        row = get_next_row(row)
```

Kompaktní zápis

pro zajímavost:

```
def get_next_row(row):  
    return [1]+[sum(p) for p in zip(row, row[1:])] + [1]
```

- dělitelné jen 1 a sebou samým
- předmět zájmu matematiků od pradávna, cca od 70. let i důležité aplikace (moderní kryptologie)
- problémy s prvočísly:
 - výpis (počet) prvočísel v intervalu
 - test prvočíselnosti
 - rozklad na prvočísla (hledání dělitelů)

Výpis prvočísel přímočaře

```
def print_primes(how_many):  
    n = 1  
    while how_many > 0:  
        if len(divisors_list(n)) == 2:  
            print(n, end=" ")  
            how_many -= 1  
        n += 1  
    print()
```

Test prvočíselnosti:

- zkusíme všechny možné dělitele od 2 do $n - 1$
- vylepšení:
 - dělíme pouze dvojkou a lichými čísly
 - dělíme pouze dvojkou a čísly tvaru $6k \pm 1$
 - dělíme pouze do \sqrt{n}

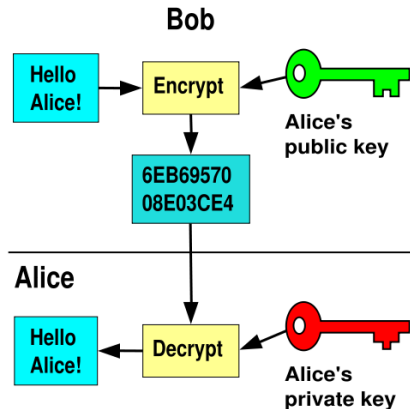
Test prvočíselnosti: chytřejší algoritmy

- náhodnostní algoritmy
- polynomiální deterministický algoritmus (objeven 2002)
- (vysoce) nad rámec tohoto kurzu
- **umí se to** dělat rychle

Rozklad na prvočísla

- rozklad na prvočísla = faktorizace
- naivní algoritmy:
 - průchod všech možných dělitelů
 - zlepšení podobně jako u testů prvočíselnosti
- chytřejší algoritmy:
 - složitá matematika
 - aktivní výzkumná oblast
 - neumí se to dělat rychle
 - max cca 200 ciferná čísla

Příklad aplikace: asymetrická kryptologie



http://en.wikipedia.org/wiki/Public-key_cryptography

Asymetrická kryptologie: realizace

- jednosměrné funkce
 - jednoduché vypočítat jedním směrem
 - obtížné druhým (inverze)
 - ilustrace: míchání barev
- RSA (Rivest, Shamir, Adleman) algoritmus
 - jednosměrná funkce: násobení prvočísel (inverze = faktorizace)
 - veřejný klíč: součin velkých prvočísel
 - bezpečnost \sim nikdo neumí provádět efektivně faktorizaci
 - využití modulární aritmetiky, Eulerovy věty, ...

Eratosthenovo síto

- problém: výpis prvočísel od 2 do n
- algoritmus: opakovaně provádíme
 - označ další neškrtnuté číslo na seznamu jako prvočíslo
 - všechny násobky tohoto čísla vyškrtni

Eratosthenovo síto

1. krok

| | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| | 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 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |

2. krok

| | | | | | | | | | | | | | | | | | | | |
|---------------|---------------|----|---------------|---------------|---------------|---------------|---------------|---------------|---------------|----|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
| | 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 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |

3. krok

| | | | | | | | | | | | | | | | | | | | |
|---------------|---------------|----|---------------|---------------|---------------|---------------|---------------|---------------|---------------|----|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
| | 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 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |

4. krok

| | | | | | | | | | | | | | | | | | | | |
|---------------|---------------|----|---------------|---------------|---------------|---------------|---------------|---------------|---------------|----|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|
| | 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 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |

Eratosthenovo síto

```
def reset_multiples(is_candidate, i):  
    k = 0  
    while k < len(is_candidate):  
        is_candidate[k] = 0  
        k += i  
  
def eratosthenes(n):  
    is_candidate = [1 for _ in range(n)]  
    for i in range(2, n):  
        if is_candidate[i]:  
            print(i, end=" ")  
            reset_multiples(is_candidate, i)
```

Pozn. Všimněte si, že funkce mění seznam.

- prvočísla – Ulamova spirála
- Pascalův trojúhelník – obarvení podle sudosti – Sierpiňského trojúhelník

Vi Hart: Doodling in math: Sick number games

<https://www.khanacademy.org/math/recreational-math/vi-hart/doodling-in-math/v/>

`doodling-in-math-sick-number-games`

Kontrolní otázky

- Je u datové struktury seznam důležité pořadí prvků?
- Může v Pythonu seznam obsahovat položky různého typu?
- Jakým příkazem přidáme do seznamu nový prvek?
- Jak zjistíme délku seznamu?
- Proč není dobrý nápad dát proměnné obsahující seznam jméno list?
- Řetězec je v mnoha ohledech podobný jako „seznam znaků“. V čem se liší?
- Jak vytvořit seznam obsahující čísla od 1 do 5? (uveďte několik různých způsobů)
- Jak zjistíme poslední prvek seznamu? Zkuste najít 3 různé způsoby.

Doporučené procvičování

<https://www.umimeprogramovat.cz/rozhodovacka>

<https://www.umimeprogramovat.cz/porozumeni>

<https://www.umimeprogramovat.cz/vystup-programu>

⇒ sada „Seznamy“

- datová struktura seznam
- základní operace se seznamy
- příklady

příště: algoritmy se seznamy – vyhledávání, řazení