

# Programování 1 pro matematiky

## 4. cvičení, 20-10-2021

tags: Programování 1 2021, čtvrtek

### Obsah:

0. Farní oznamy
1. Opakování
2. Domácí úkoly
3. Programujeme: Třídění a binární vyhledávání

### Farní oznamy

1. **Materiály k přednáškám** najdete v GitHub repozitáři <https://github.com/PKvasnick/Programovani-1>. Najdete tam také kód ke cvičením.

2. **Domácí úkoly** Dostali jste zatím 7 úkolů k prvním třem cvičením.

Maximální počet bodů 70

Nominální počet bodů (bez "bonusových" úloh) - 100%: 50

Minimální počet bodů: 35

Pod touto hranicí se nachází pouze 1 člověk z každé skupiny. **Prosím ozvěte se, pokud máte pocit, že nestíháte nebo nezvládáte.**

**Důležité oznámení:** Počet vašich bodů **není definitivní**. Pokud máte ode mne vážné výhrady k řešení a nedojde k nápravě, po uplynutí posledního termínu na odevzdání úlohy vám můžu body odebrat.

**Termíny:** Budu se snažit dávat vám úkoly s termínem do večera před následujícím cvičením. Zatím ponechám také druhý, delší termín s nižším počtem bodů za příklad, ale to se může změnit.

K domácím úkolům a ReCodExu se ještě vrátím níže.

### Kvíz

Co se vypíše?

```
1 >>> x = y = [3]
2 >>> x += y
3 >>> print(x,y)
4 >>> ???
```

1. Chyba

2. [3, 3], [3]

3. [3, 3], [3, 3]

4. Něco jiného.

To samé pro řetězce:

```
1 >>> x = y = "Python"
2 >>> x += " rocks!"
3 >>> print(x, y)
4 >>> ???
```

Co se vypíše?

Řetězce sice mají některé prvky API podobné jako seznamy, ale je to fundamentálně odlišný objekt.

Funkce `id(objekt)` vrací adresu objektu, a operátor `is` porovnává adresy dvou objektů.

## Opakování

### Příklad: Erastothénovo síto

*Úloha:* Najděte všechna prvočísla menší nebo rovná n.

*Úvahy:*

- Musíme si nějak pamatovat, která čísla jsme už vyškrtli a která nám ještě zůstala.
- Jedno řešení je, že vezmeme seznam všech čísel od 2 do n a budeme odstraňovat ty, které nejsou prvočísla. To je ale pomalé a v proměnlivém poli se špatně iteruje - není snadné určit, kde právě v poli jsme.
- Lepší je vzít seznam logických hodnot. Index bude číslo, a hodnota bude označovat, jestli jej zatím považujeme za prvočíslo anebo už ne.

```
1 # vypiš všechna prvočísla menší nebo rovná n
2
3 n = int(input())
4
5 prvocisla = [True]*(n+1) # včetně nuly a n
6 prvocisla[0] = False
7 prvocisla[1] = False
8 for i in range(2,n+1):
9     if prvocisla[i]:
10         for j in range(i*i, n+1, i):
11             prvocisla[j]=False
12
13 print("Pocet: ", sum(prvocisla))
14 for i in range(n+1):
15     if prvocisla[i]:
16         print(i, end = ', ')
```

```
1 seznam = [1, 2, 3, 4]
2 slovo = ["p", "y", "t", "h", "o", "n"]
3 list = [i for i in range(10)]
```

Poslední příkaz: *list comprehension* - umožňuje vytvořit seznam z jiného seznamu.

### Přístup k položkám a řezy (slices) seznamů

Index vrací položku

Řez (slice) vrací seznam

```
1 P y t h o n
2 0 1 2 3 4 5 6 # řezy
3 0 1 2 3 4 5 # index
4
5 slovo[1] = "y" # prvek
6 slovo[1:2] = ["y"] # seznam
```

```
1 >>> s = [i for i in range(10)]
2 >>> s
3 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
4 >>> s[3]
5 3
6 >>> s[3:4]
7 [3]
8 >>> s[::3]
9 [0, 3, 6, 9]
10 >>> s[10::-1]
11 [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
12 >>> s[::-1]
13 [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Výrazy s indexem i řezy můžou také fungovat jako l-values - tedy jim můžeme něco přiřadit:

```
1 >>> s
2 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
3 >>> s[9] = 8
4 >>> s
5 [0, 1, 2, 3, 4, 5, 6, 7, 8, 8]
6 >>> s[9:10] = [9]
7 >>> s
8 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
9 >>> s[9:10] = []
10 >>> s
11 [0, 1, 2, 3, 4, 5, 6, 7, 8]
12 >>> s[2:4] = []
13 >>> s
14 [0, 1, 4, 5, 6, 7, 8]
15 >>> s[2:2] = [2,3]
16 >>> s
17 [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

## ☐ Metody seznamů

- `list.append(x)`  
Přidává položku na konec seznamu. Ekvivalent `a[len(a):] = [x]`.
- `list.extend(iterable)`  
Rozšíří seznam připojením všech prvků `iterable` na konec seznamu. Ekvivalent `a[len(a):] = iterable`.
- `list.insert(i, x)`  
Vloží položku na danou pozici. První argument je index prvku, před který se má vkládat, takže `a.insert(0, x)` vkládá na začátek seznamu a `a.insert(len(a), x)` je ekvivalentní `a.append(x)`.
- `list.remove(x)`  
Odstraní ze seznamu první položku s hodnotou `x`. Vyvolá `ValueError` pokud se taková položka v seznamu nenajde.
- `list.pop([i])`  
Odstraní položku na zadané pozici v seznamu a vrátí tuto položku. Pokud index není zadán, `a.pop()` odstraní a vrátí poslední hodnotu v seznamu.
- `list.clear()`  
Odstraní všechny položky ze seznamu. Ekvivalent: `del a[:]`.
- `list.index(x[, zacatek[, konec]])`  
Vrátí index (počítaný od 0) v seznamu, kde se nachází první položka s hodnotou rovnou `x`. Pokud taková hodnota v seznamu neexistuje, vyvolá `ValueError`. Volitelné argumenty `zacatek` a `konec` se interpretují jako v notaci řezů a používají se k omezení hledání na určitou oblast seznamu. Výsledný index vrácený funkcí se ale vždy počítá vzhledem k začátku seznamu a ne k poloze `zacatek`.
- `list.count(x)`  
Určí, kolikrát se `x` nachází v seznamu.
- `list.sort(, klíč=None, reverse=False)`  
Uřídí položky seznamu *na místě*. Argumenty mohou být použity na upřesnění požadovaného třídění.
- `list.reverse()`  
Na místě obrátí pořadí prvků v seznamu.
- `list.copy()`  
Vrací plytkou kopii seznamu. Ekvivalent `a[:]`.

## Logický operátor `in`

Zjišťuje, zda se v iterovatelném objektu nachází daná hodnota.

```
1 # Python program to illustrate
2 # 'in' operator
3 x = 24
4 y = 20
5 list = [10, 20, 30, 40, 50 ];
```

```

6
7 if ( y in list ):
8     print("y is present in given list")
9 else:
10    print("y is NOT present in given list")
11
12 if ( x not in list ):
13     print("x is NOT present in given list")
14 else:
15     print("x is present in given list")

```

## Domácí úkoly

### Všelíké býlí v kódu

V různých domácích úkolech jsem našel různé deklarace či figury, takže jsem se rozhodl je probrat.

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  if __name__=="__main__":
5      n = int(input())
6      m = int(input())
7      if n > m:
8          print("P")
9      elif n < m:
10         print("O")
11     else:
12         print("R")

```

1. **Shebang** `#!/usr/bin/env python3` - instrukce jak spustit soubor, má-li v operačním systému nastaveno "vykonatelný". Platí pouze pro linuxové systémy, ve Windows ji *nepotřebujete*.
2. **Deklarace kódování** zdrojového kódu `# -*- coding: utf-8 -*-`: UTF-8 je defaultní kódování, pokud z vážných důvodů nepotřebujete použít něco jiného, tuto deklaraci *nepotřebujete*.
3. `if __name__=="__main__":`: označuje blok kódu, který se vykoná pouze je-li daný soubor spuštěn přímo - tedy ne importován jiným souborem. V druhém případě dostane soubor jméno `__name__` souboru. Některá IDE (PyCharm) podle tohoto řádku určují místo, odkud spustit kód. *Nepotřebujete*, ale určitě nevádí, i když nevyvíjíme pythonovské moduly.

### (Samo)testování - v ReCodExu a jinde

Zdá se mi užitečné, abyste znali testy, kterým se váš kód podrobuje v ReCodExu. Příliš mnoha z vás jsem potřeboval sdělovat, kde má jejich kód problém a proč neprochází tím či oním testem.

U následujících úloh dostanete informaci o použitých testech, kromě případů, kde by k tomu byl vážný důvod.

Váš kód potřebuje správně zpracovat všechny možné výstupy, nejen testy v ReCodExu. Proto si můžete testy vytvořit sami:

- Místo načítání sekvencí (ukončené -1) zadejte hodnoty přímo v kódu anebo použijte náhodné posloupnosti. Nepoužívaný způsob vstupu můžete prostě zakomentovat.

### Pomůcka

Odkud vezmeme posloupnost?

```
1 from random import randint
2
3 low = 0
4 high = 10
5 n = 10
6
7 print([randint(low, high) for i in range(10)])
```

## Binární vyhledávání a třídění

Na dnešním cvičení nezavedeme žádnou novou část jazyka, ale budeme cvičit práci se seznamy na dvou důležitých příkladech. Obě funkce jsou součástí API seznamů, my se je pokusíme naivně implementovat, aby sme si procvičili programovací svaly.

- **Binární vyhledávání**
- **Třídění seznamů**

### Vyhledávání v setříděném seznamu

To je to, co potřebují dělat funkce `index` a `count` - najít hodnotu v setříděném seznamu, nebo zjistit, jestli se tam nachází, nebo v kolikrát.

Algoritmus: Půlení intervalu (proto *binární*).

Náročnost:  $\log(n)$

```
1  #!/usr/bin/env python3
2  # Binární vyhledávání v setříděném seznamu
3
4  kde = [11, 22, 33, 44, 55, 66, 77, 88]
5  co = int(input())
6
7  # Hledané číslo se nachází v intervalu [l, p]
8  l = 0
9  p = len(kde) - 1
10
11 while l <= p:
12     stred = (l+p) // 2
13     if kde[stred] == co:    # Našli jsme
14         print("Hodnota ", co, " nalezena na pozici", stred)
15         break
16     elif kde[stred] < co:
17         l = stred + 1      # Jdeme doprava
18     else:
19         p = stred - 1      # Jdeme doleva
20 else:
```

```
21 print("Hledaná hodnota nenalezena.")
22
```

## Třídění

### Opakovaný výběr minima

Opakovaně vybíráme minimum a příslušnou hodnotu umísťujeme na začátek seznamu.

```
1 # Třídění opakovaným výběrem minima
2
3 x = [31, 41, 59, 26, 53, 58, 97]
4
5 n = len(x)
6 for i in range(n):
7     pmin = i
8     for j in range(i+1, n):
9         if x[j] < x[pmin]:
10             pmin = j
11     x[i], x[pmin] = x[pmin], x[i]
12
13 print(x)
14
```

### Bublinové vyhledávání

Postupně "probubláváme" hodnoty směrem nahoru opakovaným srovnáváním se sousedy

```
1 # Třídění probubláváním
2
3 x = [31, 41, 59, 26, 53, 58, 97]
4
5 n = len(x)
6 for i in range(n-1):
7     nswaps = 0
8     for j in range(n-i-1):
9         if x[j] > x[j+1]:
10             x[j], x[j+1] = x[j+1], x[j]
11             nswaps += 1
12     if nswaps == 0:
13         break
14
15 print(x)
16
```

## Domácí úkoly

1. **Medián:** Načíst posloupnost končící -1 a vypočítat a vypsat medián, tedy hodnotu, od které je polovina hodnot v seznamu větších a polovina menších.
2. **Pascalův trojúhelník** - pro dané n máte vypsat seznam seznamů, obsahující prvních n řádek Pascalova trojúhelníku.

3. **Fibonacciho algoritmus pro rozklad zlomků** - dostanete čítele a jmenovatele zlomku, a máte pomocí Fibonacciho algoritmu převést zlomek na součet zlomků se jmenovatelem 1 (= do "egyptského tvaru"). Příklad:

$$\frac{17}{36} = \frac{1}{3} + \frac{1}{8} + \frac{1}{72}$$

Pomůcka: `"1 3".split()` dává `["1", "3"]`.