`Programování` **1 pro matematiky**

4. cvičení, 01-11-2023

---

tags: `Programovani 1 2021`, `čtvrtek`

Obsah:

0. Farní oznamy
1. Opakování
2. Seznamy
3. Domácí úkoly
4. 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 9 úkolů k prvním cvičením.

   Dostali jste domácí úkoly i na tento týden, i když minulý týden cvičení nebylo.

   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.

   K domácím úkolům a ReCodExu se ještě vrátím níže.

Kvíz

Co se vypíše?

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

1. Chyba
2. [3, 3], [3]
3. [3, 3], [3, 3]
4. Něco jiného.

To samé pro řetězce:

```
>>> x = y = "Python"
>>> x += " rocks!"
>>> print(x, y)
>>> ???
```

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

Euklidův algoritmus

Základní verze s odečítáním: $x > y : \gcd(x, y) = \gcd(x - y, y)$

```
#!/usr/bin/env python3
# Největší společný dělitel: Euklidův algoritmus s odčítáním
```

```
x = int(input())
y = int(input())

while x != y:
    if x > y:
        x -= y
    else:
        y -= x

print(x)
```

Ladící výpis: Pokud chceme vidět, jak si vedou čísla x a y v cyklu while, přidáme za rádek s `while` příkaz

```
print(f"{x=} {y=}")
```

Pokud je jedno z čísel o hodně menší než druhé, možná budeme opakovaně odečítat, a to nás spomaluje (náročnost algoritmu je lineární v n). Je proto lepší v jednom kroku odečítat kolikrát to jde: *odečítání nahradíme operací modulo*:

```
#!/usr/bin/env python3
# Největší společný dělitel: Euklidův algoritmus s modulem

x = int(input())
y = int(input())

while x > 0 and y > 0:
    if x > y:
        x %= y
    else:
        y %= x

if x > 0:
    print(x)
else:
    print(y)
```

Protože `x % y < y`, po každé operaci modulo víme, jaká je vzájemná velikost x a y. Kód tedy můžeme výrazně zdokonalit:

```python
#!/usr/bin/env python3
# Největší společný dělitel: Euklidův algoritmus s pár triky navíc

x = int(input())
y = int(input())

while y > 0:
    x, y = y, x%y

print(x)
```

Tady si všimneme přiřazení `x, y = y, x%y`. Je to dvojí přiřazení, ale nelze jej rozdělit na dvě přiřazení `x=y` a `y=x%y`, protože druhé přiřazení se po prvním změnilo na `y=y%y` a tedy y bude přiřazena 0.

1. Můžeme se ptát, proč to funguje (protože z dvojice na pravé straně se před přiřazením vytvoří neměnná - konstantní dvojice - *tuple* - a ten se při přiřazení "rozbalí" do x a y).
2. Jak byste takovéto přiřazení rozepsali na jednoduchá přiřazení, aby to fungovalo?

Toto je už celkem výkonný algoritmus, početní náročnost je $\sim \log n$ Teď můžeme dělat víc věcí, například spočíst Eulerovu funkci pro prvních milión čísel a podobně.

Fibonacciho čísla

$$\begin{aligned} Fib(0) &= 1, \\ Fib(1) &= 1, \\ Fib(n) &= Fib(n-1) + Fib(n-2), \quad = 2, 3, \ldots \end{aligned} \tag{1}$$

Úloha: Vypište prvních n Fibonacciho čísel.

Úvahy:

- potřebujeme vůbec seznam?

```
# Vypsat prvnich n Fibonacciho cisel
n = int(input())
a = 1
print(a, end = ", ")
b = 1
print(b, end = ", ")
for k in range(3,n+1):
    b, a = b+a, b
    print(b, end = ", ")
```

- Můžeme začít seznamem prvních dvou čísel, a pak dopočítávat a přidávat na konec další čísla:

```
# Vypsat prvnich n Fibonacciho cisel
n = int(input())
fibs = [1,1]
while len(fibs)<n:
    fibs.append(fibs[-1] + fibs[-2])
print(fibs)
```

- Pokud možno, nechcete alokovat paměť po malých kouscích. Když předem víme délku seznamu, je nejlepší zřídit ho celý a jenom ho naplnit.

Příklad: Erastothénovo síto

Úloha: Najděte všechna prvočísla menší nebo rovná n.

Úvahy:

- Musíme si nějak pamatovat, která čísla jsme už vyšktrtli a která nám ještě zůstala.
- Jedno řešení je, že vezmeme seznam všech čísel od 2 do n a budeme odstraňovat ty, které nejsou prvočísly. 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.

```
# vypiš všechna prvočísla menší nebo rovná n

n = int(input())
```

```
prvocisla = [True] * (n + 1) # včetně nuly a n
prvocisla[0] = False
prvocisla[1] = False
for i in range(2, n + 1):
    if prvocisla[i]:
        for j in range(i * i, n + 1, i):
            prvocisla[j] = False

print("Pocet: ", sum(prvocisla))
for i in range(n + 1):
    if prvocisla[i]:
        print(i, end = ', ')
```

Seznamy

```
>>> cisla = [1,2,3,4,5]
>>> type(cisla)
list
>>> cisla[0] # V Pythonu číslujeme od 0
1
>>> cisla[4] # takže poslední prvek je počet prvků - 1
5
>>> len(cisla) # počet prvků je len
5
>>> cisla[-1] # Indexování je velmi flexibilní
5
>>> cisla[1:3]
[2, 3]
>>> cisla[0:5]
[1, 2, 3, 4, 5]
>>> cisla[:3]
[1, 2, 3]
>>> cisla[3:]
[4, 5]
>>> cisla.append(6) # Přidání nového prvku do seznamu
>>> cisla
[1, 2, 3, 4, 5, 6]
```

Seznamy můžou obsahovat různé věci:

```
zaci = ["Honza", "Jakub", "Franta"]
matice = [[1,2,3],[2,3,1]] # Neužitečná implementace matice
matice[0]
>>> [1,2,3]
matice[1][1]
>>> 3
>>> [1,2] + [3,4] # aritmetika pro seznamy
[1, 2, 3, 4]
>>> [1,2]*3
[1, 2, 1, 2, 1, 2]
```

... ale také položky různého druhu:

```
>>> lst = [1,"Peter",True]
>>> lst
[1, 'Peter', True]
>>> del lst[0]
>>> lst
['Peter', True]
>>>
```

Pozor na kopírování seznamů:

```
>>> a = ['jeden', 'dva', 'tri']
>>> b = a
>>> a[0] = 'jedna'
>>> b
['jedna', 'dva', 'tri']
>>> c = [[1,2]]*3
>>> c
[[1, 2], [1, 2], [1, 2]]
>>> c[0][0] = 0
>>> c
[[0, 2], [0, 2], [0, 2]]
```

Seznam umíme lehce setřídit nebo obrátit:

```
>>> c = [2,4,1,3]
>>> sorted(c)
[1,2,3,4]
>>> reversed[c]
[3,1,4,2]
```

O třídění budeme mluvit na následujícím cvičení.

Cyklus `for`

```
In [9]: cisla = [1,1,2,3,5,8]

In [10]: for cislo in cisla:
    ...:     print(cislo, end = "-")
    ...:
1-1-2-3-5-8-
```

Cyklus for je *stručný* - na rozdíl od *while* nepotřebujeme inicializovat logickou podmínku ani inkrementovat či jinak měnit proměnné v cyklu.

Často chceme, aby cyklus probíhal přes jednoduchou číselnou řadu, např. $1, 2, \ldots, n$. Na generování takovýchto řad slouží funkce `range`:

```
>>> for i in range(5):
    print(i, end = ' ')

0 1 2 3 4
```

```
seznam = [1, 2, 3, 4]
slovo = ["P", "y", "t", "h", "o", "n"]
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

```
 P y t h o n
0 1 2 3 4 5 6 # řezy
 0 1 2 3 4 5  # index

slovo[1] = "y" # prvek
slovo[1:2] = ["y"] # seznam
```

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

```
>>> s
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> s[9] = 8
>>> s
[0, 1, 2, 3, 4, 5, 6, 7, 8, 8]
>>> s[9:10] = [9]
>>> s
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> s[9:10] = []
>>> s
[0, 1, 2, 3, 4, 5, 6, 7, 8]
>>> s[2:4] = []
>>> s
[0, 1, 4, 5, 6, 7, 8]
>>> s[2:2] = [2,3]
>>> s
[0, 1, 2, , 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 (počíítaný 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*)

  Utřídí položky seznamu *na místě*. Argumenty můžou být použity na upřesnění požadovaného třídení.

- `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.

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

if ( y in list ):
   print("y is present in given list")
else:
   print("y is NOT present in given list")

if ( x not in list ):
   print("x is NOT present in given list")
else:
   print("x is  present in given list")
```

Pozor To, že `in` je krátké slovíčko neznamená, že hledání, zda se nějaký prvek nachází v seznamu, je nějak zvlášt efektivní. Není, seznam se prohledá položku po položce. Pokud chcete kolekci s opravdu rychlým vyhledáváním, použijte množinu nebo slovník.

Domácí úkoly

(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 všech úloh dostáváte informaci o použitých testech*

Váš kód potřebuje správně zpracovat všechny možné výstupy, nejen testy v ReCodExcu. 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.

Binární vyhledáváni a třídění

V této části 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, abychom 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)

```
#!/usr/bin/env python3
# Binární vyhledávání v setříděném seznamu

kde = [11, 22, 33, 44, 55, 66, 77, 88]
co = int(input())

# Hledané číslo se nachazí v intervalu [l, p]
l = 0
p = len(kde) - 1

while l <= p:
    stred = (l+p) // 2
    if kde[stred] == co:   # Našli jsme
        print("Hodnota ", co, " nalezena na pozici", stred)
        break
    elif kde[stred] < co:
```

```
        l = stred + 1      # Jdeme doprava
    else:
        p = stred - 1      # Jdeme doleva
else:
    print("Hledaná hodnota nenalezena.")
```

Třídění

Opakovaný výběr minima

Opakovaně vybíráme minimum a příslušnou hodnotu umísťujeme na začátek seznamu.

```
# Třídění opakovaným výběrem minima

x = [31, 41, 59, 26, 53, 58, 97]

n = len(x)
for i in range(n):
    pmin = i
    for j in range(i+1, n):
        if x[j] < x[pmin]:
            pmin = j
    x[i], x[pmin] = x[pmin], x[i]

print(x)
```

Bublinové vyhledávání

Postupně "probubláváme" hodnoty směrem nahoru opakovaným srovnávaním se sousedy

```
# Třídění probubláváním

x = [31, 41, 59, 26, 53, 58, 97]

n = len(x)
for i in range(n-1):
    nswaps = 0
```

```
    for j in range(n-i-1):
        if x[j] > x[j+1]:
            x[j], x[j+1] = x[j+1], x[j]
        nswaps += 1
    if nswaps == 0:
        break

print(x)
```

Domácí úkoly

1. **Medián**: Načíst posloupnost končící -1 a vypočíst 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. **Emulace funkce list.insert()** Máte vytvořit kousek kódu, který na danou pozici v seznamu vloží novou položku.