`Programování` 1 pro matematiky

4. cvičení, 20,21-10-2021

tags: `Programovani 1 2021`, `středa`, `čtvrtek`

Obsah:

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 třem cvičením.

   Maximální počet bodů 85

   Nominální počet bodů (bez "bonusových" úloh) - 100%: 55

   Minimální počet bodů: 38

   Chtěl bych slyšet od 30% z vás, kteří se nacházíte pod touto hranicí.

   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.

Opakování

- [ ] **Seznamy**

```
list = [1, 2, 3, 4]
slovo = "Python"
list = [i for i in range(10)]
```

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
 0 1 2 3 4 5

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, 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. removes and returns the last item in the list.

- `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")
```

Operátor `is`

Dáva `True`, keď objekty na dvoch stranách sú ten istý objekt.

```
# Python program to illustrate the use
# of 'is' identity operator
x = 5
if (type(x) is int):
    print("true")
else:
    print("false")
```

Domácí úkoly

Všelíké býlí v kódu

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

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

if __name__=="__main__":
    n = int(input())
    m = int(input())
    if n > m:
        print("P")
    elif n < m:
        print("O")
    else:
        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. Pro toto cvičení, kde nebudeme vyvíjet Pythonovské moduly,tento kód **nepotřebujete**.

(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 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.

Pomůcka

Odkud vezmeme posloupnost?

```
from random import randint

low = 0
high = 10
n = 10

print([randint(low, high) for i in range(10)])
```

Binární vyhledáváni 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 seznam.ů, my se je pokusíme naivně implementovat, aby sme si procivičili 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)

```

###