

# Programování 1 pro matematiky

---

## 3. cvičení, 13,14-10-2021

---

tags: Programování 1 2021, středa, čtvrtek

---

### Obsah:

- 0. Farní oznamy
- 1. Opakování
- 2. Domácí úkoly
- 3. Seznamy
- 4. Programujeme...

### 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 6 úkolů k prvním dvěma cvičením.  
2 příklady jsou označené jako *bonusové* (Euklidova funkce a Pytagorejské trojice). **Počítání bodů:** Očekává se, že dosáhnete 70% bodového zisku za povinné příklady, a to *libovolnou kombinací bodů za povinné a bonusové příklady*.
3. **ReCodEx** Navzdory jistým omezením to je obrovská pomoc při správě domácích úkolů. Zatím se snažím průběžně kontrolovat každé odevzdané řešení, ale může se stát, že přestanu stíhat, když bude v oběhu víc příkladů najednou.
4. **Někteří z vás ještě neodevzdali ani jeden příklad.** To nemusí být kritické, máte k dispozici dost času i bodů, ale nespolehejte se, že to všechno doženete najednou. Úlohy mají *termíny*, protože po určité době ztrácejí opodstatnění.

*Máte s ReCodExem nějaké problémy? Funguje komunikace? Vídíte mé komentáře k řešením?*

K domácím úkolům a ReCodExu se ještě obšírněji vrátím.

---

### Opakování

☐ `if-elif-else:`

```
1  if podmínka1:
2      příkazy1
3  elif podmínka2:
4      příkazy2
5  else:
6      příkazy3
```

☐ Prázdný příkaz `pass`

☐ Cyklus `while`:

```
1 while podmínka:
2     příkazy
```

Příkazy pro kontrolu běhu cyklu:

`break` - v tomto místě opustit cyklus a pokračovat příkazem, následujícím za cyklem

`continue` - v tomto místě přejít na další iteraci cyklu (tedy na testování podmínky)

Větev `else`:

```
1 while podmínka:
2     příkazy
3 else
4     příkazy-jen-jestli-proběhl-celý-cyklus
```

☐ Vícenásobná přiřazení:

`a, b = b, a%b` (Krok modulárního Eulerova algoritmu)

`a, b = a+b, a` (Čeho je toto krok?)

---

## Domácí úkoly

### ReCodEx

Mnozí z vás svedli bitvu s ReCodExem, než se vám povedlo odevzdat své řešení. Je to dané několika specifiky:

ReCodEx používá pro testování vašeho kódu testy, které pozůstávají

1. ze souboru, ze kterého váš kód čte standardní vstup
2. ze souboru, se kterým ReCodEx porovnává standardní výstup vašeho kódu.

Porovnání výstupu vašeho programu s očekávaným výstupem je strojové a poměrně jednoduché.

**Cokoli, co váš program vypíše kromě očekávaného výstupu, povede k selhání v testu.**

Příkladem, který znáte všichni, kdo už něco na ReCodEx posílali, jsou příkazy `input()` - nikdy v nich prosím nepoužívejte výzvu! `input("Zadej číslo: ")` způsobí, že váš program neprojde testy, protože výzva se vypíše do standardního výstupu.

Z podobného důvodu vyžadují zadání nějaký **jednoduchý výstup**. Úloha o Pythagorejských trojicích například vyžaduje jen počet trojic a ne jejich vypsání, protože kontrola seznamu trojic není spolehlivá.

### Čistota a čitelnost kódu

Porovnejte prosím tyto dva kódy. Oba jsou správné a dělají správně to, co mají.

Tento kód, včetně názvů identifikátorů, vyvinulo několik z vás:

```

1 n = int(input())
2 m = int(input())
3 if n > m:
4     print("P")
5 elif n < m:
6     print("O")
7 else:
8     print("R")

```

Porovnejte ho prosím s tímto kódem (to je také řešení, pocházející od jednoho z vás):

```

1 cislo_princezny = int(input())
2 cislo_obra = int(input())
3
4 if cislo_princezny > cislo_obra:
5     print("P")
6 elif cislo_princezny < cislo_obra:
7     print("O")
8 else:
9     print("R")

```

Problém s prvním kódem je, že si někde musíte zapamatovat přiřazení  $n \rightarrow$  číslo od princezny,  $m \rightarrow$  číslo od obra. Pak prvnímu kódu rozumíte stejně dobře jako druhému, ale v hlavě musíte držet náhodnou informaci, potřebnou k pochopení kódu.

Kód, který má krátké názvy proměnných, není v nijakém smyslu výkonnější nebo úspornější než kód s dlouhými proměnnými. Na druhou stranu, kód s příliš dlouhými identifikátory se může hůř číst.

**Jména objektů v Pythonu:** Počáteční písmeno, pak kombinace velkých i malých písmen, číslic a podtržítek.

**Snažte se prosím používat názvy proměnných, které reflektují jejich význam.**

## Prostředky Pythonu

Hodně z vás zná z Pythonu víc, než jsme zatím probrali. Přijímám i řešení, která používají prostředky jazyka, které jsme zatím neprobírali,

- pokud to není v rozporu s účelem zadání - tedy když máte sami naprogramovat něco, na co existuje v Pythonu knihovní funkce - například jsme programovali  $GCD(a,b)$ , zatímco máme funkci `math.gcd()`, která to udělá za vás.
- pokud to vede k čistšímu a efektivnímu kódu - mezi řešeními domácích úkolů se našly i takové, které využívali funkce, seznamy, množiny atd a to všechno spletené do ošklivého špatně čitelného kódu.

Cílem tohoto cvičení je naučit vás psát nejen správný, ale i čistý a dobře čitelný kód. Toho dosáhnete i účelným využíváním rozmanitých prostředků jazyka.

# Seznamy

```
1 >>> cisla = [1,2,3,4,5]
2 >>> type(cisla)
3 list
4 >>> cisla[0] # V Pythonu číslujeme od 0
5 1
6 >>> cisla[4] # takže poslední prvek je počet prvků - 1
7 5
8 >>> len(cisla) # počet prvků je len
9 5
10 >>> cisla[-1] # Indexování je velmi flexibilní
11 5
12 >>> cisla[1:3]
13 [2, 3]
14 >>> cisla[0:5]
15 [1, 2, 3, 4, 5]
16 >>> cisla[:3]
17 [1, 2, 3]
18 >>> cisla[3:]
19 [4, 5]
20 >>> cisla.append(6) # Přidání nového prvku do seznamu
21 >>> cisla
22 [1, 2, 3, 4, 5, 6]
```

Seznamy můžou obsahovat různé věci:

```
1 zaci = ["Honza", "Jakub", "Franta"]
2 matice = [[1,2,3],[2,3,1]] # Neužitečná implementace matice
3 matice[0]
4 >>> [1,2,3]
5 matice[1][1]
6 >>> 3
7 >>> [1,2] + [3,4] # aritmetika pro seznamy
8 [1, 2, 3, 4]
9 >>> [1,2]*3
10 [1, 2, 1, 2, 1, 2]
```

... ale také položky různého druhu:

```
1 >>> lst = [1,"Peter",True]
2 >>> lst
3 [1, 'Peter', True]
4 >>> del lst[0]
5 >>> lst
6 ['Peter', True]
7 >>>
```

Pozor na kopírování seznamů:

```

1 >>> a = ['jeden', 'dva', 'tri']
2 >>> b = a
3 >>> a[0] = 'jedna'
4 >>> b
5 ['jedna', 'dva', 'tri']
6 >>> c = [[1,2]]*3
7 >>> c
8 [[1, 2], [1, 2], [1, 2]]
9 >>> c[0][0] = 0
10 >>> c
11 [[0, 2], [0, 2], [0, 2]]

```

Seznam umíme lehce setřídít nebo obrátit:

```

1 >>> c = [2,4,1,3]
2 >>> sorted(c)
3 [1,2,3,4]
4 >>> reversed[c]
5 [3,1,4,2]

```

O třídění budeme mluvit na následujícím cvičení.

## Cyklus `for`

```

1 In [9]: cisla = [1,1,2,3,5,8]
2
3 In [10]: for cislo in cisla:
4     ...:     print(cislo, end = "-")
5     ...:
6 1-1-2-3-5-8-

```

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

```

1 >>> for i in range(5):
2     print(i, end = ' ')
3
4 0 1 2 3 4

```

`range` respektuje číslovací konvence Pythonu a podporuje ještě další argumenty: začátek sekvence a krok:

```

1 >>> for i in range(2,10,3):
2     ...     print(i)
3     ...
4 2
5 5
6 8

```

Argumenty nemusí být kladná čísla, takže můžeme lehko iterovat pozpátku:

```

1 >>> for i in range(9,-1,-1):
2     print(i, end = ' ')
3     ---
4 9 8 7 6 5 4 3 2 1 0 >>>

```

Tedy chceme-li iterovat pozpátku přes `range(n)`, použijeme `range(n-1,-1,-1)`. Abychom se nespletli, nabízí Python elegantnější řešení:

```

1 >>> for i in reversed(range(5)):
2     print(i, end = " ")
3
4 4 3 2 1 0

```

a tuto funkci můžeme použít na libovolný seznam:

```

1 >>> cisla = ["jedna", "dvě", "tři", "čtyři"]
2 >>> for cislo in reversed(cisla):
3     print(cislo, end = " ")
4
5 čtyři tři dvě jedna

```

`range(n)` není seznam, i když podporuje přístup k polžkám přes index:

```

1 >>> range(10)
2 range(0, 10)
3 >>> type(range(10))
4 <class 'range'>
5 >>> range(10)[-1]
6 9
7 >>> list(range(10))
8 [0,1,2,3,4,5,6,7,8,9]
9 >>> [i for i in range(10)]
10 [0,1,2,3,4,5,6,7,8,9]

```

Také `reversed(seznam)` není seznam, ale reverzní iterátor:

```

1 >>> reversed(cisla)
2 <list_reverseiterator object at 0x000002AE55D45370>
3 >>> [s for s in reversed(cisla)]
4 ['čtyři', 'tři', 'dvě', 'jedna']

```

ale `sorted(seznam)` je seznam:

```

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

```

## Fibonacciho čísla

$$Fib(0) = 1, \quad Fib(1) = 1, \quad Fib(n) = Fib(n-1) + Fib(n-2), \quad n = 2, 3, \dots$$

*Úloha:* Vypište prvních  $n$  Fibonacciho čísel.

Úvahy:

- potřebujeme vůbec seznam?

```
1 # Vypsát prvních n Fibonacciho čísel
2 n = int(input())
3 a = 1
4 print(a, end = ", ")
5 b = 1
6 print(b, end = ", ")
7 for k in range(3,n+1):
8     b, a = b+a, b
9     print(b, end = ", ")
```

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

```
1 # Vypsát prvních n Fibonacciho čísel
2 n = int(input())
3 fibs = [1,1]
4 while len(fibs)<n:
5     fibs.append(fibs[-1] + fibs[-2])
6 print(fibs)
```

- Nikdy 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 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 = ', ')
```

---

## Další úkoly

Je dána posloupnost čísel...

- zjistěte, jestli jsou všechna navzájem různá.
- najděte dvojici s co nejmenším rozdílem.
- vypište všechna opakující se čísla (ale každé jen jednou).
- Umíte předchozí úkoly vyřešit efektivněji, pokud víte, že všechna zadaná čísla leží od 1 do 100?

### 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)])
```