

# Programování 1 pro matematiky

## 2. cvičení, 6,7-10-2021

tags: Programování 1 2021, středa, čtvrtek

### Obsah:

0. Farní oznamy
1. Opakování
2. Příkaz `if`
3. Příkaz `while`
4. Programujeme...

### Farní oznamy

1. **Materiály k přednáškám** najdete v GitHub repozitáři <https://github.com/PKvasnick/Programovani-1>. Najdete tam také kód ke cvičením.
  - Soubory si můžete číst přímo na GitHubu. Pokud si chcete stáhnout nebo zkopírovat kód, přepněte do *Raw* zobrazení (aby se vám nekopírovaly čísla řádků a pod.), Ctrl-A + Ctrl-C.
  - Windows: Nainstalujte si aplikaci GitHub Desktop a naklonujte si celý repozitář do svého počítače: Zelené tlačítko `Code`, z nabídky `open with GitHub Desktop`.
  - Pokud se v nějakém okamžiku neobejdete bez zřízení konta na GitHubu, zřídte si jej.
2. **Domácí úkoly** Tady se omlouvám, pral jsem se s ReCodExem, abych vám mohl zadat smysluplné domácí úkoly, ale nepovedlo se. Když budete psát víc kódu, budou snad omezení ReCodExu méně obtěžující.
3. **Cvičení pro pokročilé** - prosím ještě vydržte, oslovím vás v průběhu týdne. Zatím dostanete pár úkolů v průběhu přednášky.
4. **Každému vše chodí?** - přišel takovýto e-mail, pokud máte problémy, prosím hlašte se:

Zdravím,

Někteří z nás na cvičeních čelili problémům studentů s nemožností se přihlásit do počítačů v Impactu. Závěry z vyšetřování příčin a řešení posílám níže:

1. Kdo není studentem MFF, tak se nemůže přihlásit (systém uznává pouze jmenný login a ten studenti jiných fakult prý nemají). Řešením je zřídit jim lokální účet na Malé Straně. Jana zjistila, že je možné napsat email na [simunek@sisal.mff.cuni.cz](mailto:simunek@sisal.mff.cuni.cz) jméno s číslem osoby na kartičce UK (a pro úplnost i email) a Dr. Šimůnek jim to zařídí.

My z ÚČJF budeme posílat seznam studentů mimo MFF přes Janu hromadně. Pokud chcete, napište jí, nebo rovnou Šimůnkovi.

Petr Kos, správce Impactu, říkal, že i časem on bude moci lokální účet na Malé Straně zřídit, ale zatím to nejde.

\\2. Studenti MFF, kteří mají účet na Malé Straně, se musí mít na pozoru. Tento účet má prioritu před přihlašováním přes CAS. Takže pro přihlášení na počítače v Impactu musí! použít login a heslo z Malé Strany. Toto platí i pro toho, kdo si časem účet na Malé Straně ještě udělá.

\\3. Pro přihlášení pomocí CAS nefunguje číslo osoby. Je nutné zadat login, který se skládá z části příjmení, jména, čísla a nějakých písmen, typu roskb5am. Pro přihlášení je prý nutné použít malá písmena.

\\4. Studenti mohou mít ještě problémy s přihlášením, páč na počítači může být nastavená anglická klávesnice a ne česká.

Pokud máte ještě jiné postřehy, neváhejte je sdílet.

Zatím,

Beda Roskovec

---

## Opakování

- ✓ Základní instalace Pythonu
- ✓ Čísla a řetězce, aritmetické a logické operace
- ✓ Konverze `int()`, (a nejspíš vás nepřekvapí, že také existuje `float()`, `str()` a `bool()`).

```
1 In [4]: int(4.9)
2 Out[4]: 4
3
4 In [5]: round(4.9,0)
5 Out[5]: 5.0
6
7 In [6]: float(5)
8 Out[6]: 5.0
9
10 In [7]: bool(0.5)
11 Out[7]: True
12
13 In [8]: bool(-1.0)
14 Out[8]: True
15
16 In [9]: bool(0.0)
17 Out[9]: False
18
19 In [10]: str(4.6)
20 Out[10]: '4.6'
21
22 In [11]: str(True)
23 Out[11]: 'True'
24
25 In [12]: str(True)
26 Out[12]: 'True'
27
28 In [13]: str(False)
29 Out[13]: 'False'
```

- ☐ Operátory `+, -, *, /, **, //, %, ==, and, or, not`
- ☐ Přiřazení `=` a přiřazení s operací `+=, -=, *=, /=`, ale také třeba `%=` - operátor *vymodulení*, s kterým se dnes setkáme.
- ☐ Matematické funkce z balíku *math*, `math.*`, např. `math.sin()`.
- ☐ Funkce pro čtení řetězce ze standardního vstupu `input(výzva)` a funkce pro tisk do standardního výstupu `print(objekt1, objekt2, ...)`
- ☐ Podmíněný příkaz

```
1 if podmínka:
2     příkazy
```

- ☐ Příkaz cyklu

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

kde *příkazy* mohou být příkazy přiřazení, volání funkce, další podmíněné příkazy nebo příkazy cyklu, a dnes se naučíme, že také příkazy `pass` (nedělej nic), `break` (opuštění cyklu) a `continue` (přechod na další iteraci cyklu).

---

## Příkaz `if`

Úplná syntaxe příkazu `if`:

```
1 if podmínka:
2     příkazy
3 else:           # volitelně
4     příkazy
```

Větev `else` je nepovinná; když chceme vynechat příkazy ve větvi `if`, musíme použít prázdný příkaz `pass`.

Větve `elif`: V případě řetězcích příkazů `if` můžeme namísto konstrukce

```
1 if podmínka1:
2     příkazy
3 else:
4     if podmínka2:
5         příkazy
6     else:
7         příkazy
8
```

psát

```
1 if podmínka1:
2     příkazy
3 elif podmínka2:
4     příkazy
5 else:
6     příkazy
```

což je o něco přehlednější - mén slov a hlavně ploché odsazení.

## Příkaz `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)

Nekonečný cyklus: podmínka stále platí, a o ukončení cyklu rozhodneme v těle za použití příkazu `break`:

```
1 while True:
2     příkazy
3     if podmínka:
4         break
```

Příkaz `while` má také volitelnou větev `else`. Příkazy v této větvi se vykonají, pokud cyklus řádně skončí (tedy ne v případě opuštění cyklu příkazem `break`).

```
1 while podmínka:
2     příkazy1
3 else:
4     příkazy2
```

## Příklady

### Test prvočísel

Chceme otestovat, zda je číslo  $n$  ze vstupu prvočíslo.

Metoda: U všech čísel  $d < n$  prověřím, zda jsou děliteli  $n$ .

```
1  #!/usr/bin/env python3
2
3  # Otestuje, zda číslo je prvočíslem
4
5  n = int(input())
6  d = 2
7  mam_delitele = False
8
9  while d < n:
10     if n%d == 0:
11         print("Číslo", n, "je dělitelné", d)
12         mam_delitele = True
13         break
14     d += 1
15
16  if not mam_delitele:
```

```
17 | print("Číslo", n, "je prvočíslo")
```

To není nijak zvlášť efektivní metoda, ale to nám nevadí, my jsme celí rádi, že umíme napsat něco, co v zásadě funguje.

Pojďme opatrně vylepšovat. Zásadní vylepšení kódu by bylo, kdybychom "nahý" cyklus `while` uměli celý zapouzdřit do jediného příkazu.

☺ Pokročilé kolegy poprosím o tvar onoho jediného příkazu.

Asi první věc, která nám vadí, je stavová proměnná `mam_delitele`. A té se v prvním kroku zbavíme za použití větve `else`:

```
1  #!/usr/bin/env python3
2
3  # Otestuje, zda číslo je prvočíslem (2. pokus)
4
5  n = int(input())
6  d = 2
7
8  while d < n:
9      if n%d == 0:
10         print("Číslo", n, "je dělitelné", d)
11         break
12     d += 1
13 else:
14     print("Číslo", n, "je prvočíslo")
```

Jak bychom mohli dál vylepšit náš test?

Popřemýšlíme, a zatím vymyslíme, jak bychom vypsali všechna prvočísla menší nebo rovná  $n$ . Nejjednodušší metoda bude projít všechna čísla od 2 do  $n$ , u každého rozhodnout, zda je prvočíslem, a jestli ano, vypsát ho.

```
1  #!/usr/bin/env python3
2  # vypíše všechna prvočísla od 1 do n
3
4  n = int(input())
5
6  x = 2
7  while x <= n:
8      d = 2
9      while d < x:
10         if x%d == 0:
11             break
12         d += 1
13     else:
14         print(x)
15
16     x += 1
17
```

Optimalizace je v tomto případě ještě více nasnadě, jenomže si zatím neumíme pamatovat věci - například všechny prvočísla, které jsme dosud našli.

☺ Pokročilé kolegy poprosím o algoritmus s memoizací a/nebo Erastovenovo síto.

# Euklidův algoritmus

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

```
1  #!/usr/bin/env python3
2  # Největší společný dělitel: Euklidův algoritmus s odčítáním
3
4  x = int(input())
5  y = int(input())
6
7  while x != y:
8      if x > y:
9          x -= y
10     else:
11         y -= x
12
13 print(x)
14
```

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*:

```
1  #!/usr/bin/env python3
2  # Největší společný dělitel: Euklidův algoritmus s modulem
3
4  x = int(input())
5  y = int(input())
6
7  while x > 0 and y > 0:
8      if x > y:
9          x %= y
10     else:
11         y %= x
12
13 if x > 0:
14     print(x)
15 else:
16     print(y)
```

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

```
1  #!/usr/bin/env python3
2  # Největší společný dělitel: Euklidův algoritmus s pár triky navíc
3
4  x = int(input())
5  y = int(input())
6
7  while y > 0:
8      x, y = y, x%y
9
10 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 spočítat Eulerovu funkci pro prvních milión čísel a podobně.

---

## Součet posloupnosti čísel

```
1  #!/usr/bin/env python3
2
3  # Načteme ze vstupu posloupnost čísel, ukončenou -1.
4  # Vypíšeme jejich součet.
5
6  s = 0
7  while True:
8      n = int(input())
9      if n == -1:
10         break
11     s += n
12 print(s)
```

Proč nemůžeme na konci jenom stisknout Enter a nezadat nic?

☺ Pokročilé kolegy poprosím

- o variantu se stiskem Enter
- a pro vypísání aritmetického průměru a standardní odchylky.

## Druhé největší číslo posloupnosti

Načtěte ze vstupu posloupnost čísel ukončenou -1. Pak vypište

- druhé největší číslo posloupnosti
- jeho polohu v posloupnosti

Abychom pochopili, jak to udělat, přemýšlíme v termínech stavu našeho pátrání po druhém největším čísle. Abychom správně naložili s novým číslem, musíme si pamatovat aktuálně největší a druhé největší číslo posloupnosti  $m_1$  a  $m_2$ . Když nám přijde nový člen posloupnosti  $m$ , musíme tento stav - tedy čísla  $m_1$  a  $m_2$  - aktualizovat podle toho, jaká je jeho velikost. Pokud přijde  $m = -1$ , vypíšeme  $m_2$ .

```
1  #!/usr/bin/env python3
2  # Načítá čísla ze vstupu ukončená -1,
3  # vypíše druhé největší z nich
4
5  m1 = 0      # Zatím největší číslo
6  m2 = 0      # Zatím druhé největší
7
8  while True:
9      n = int(input())
10     if n == -1:
```

```
11         break
12
13     if n >= m1:
14         m1, m2 = n, m1
15     elif n >= m2:
16         m2 = n
17
18 print(m2)
```

---

## Další úlohy

Další úkoly:

- Spočítejte, kolik má zadané číslo cifer.
- Najděte číslo zapsané samými jedničkami (v desítkové soustavě), které je dělitelné zadaným  $K$ . Jak se včas zastavit, když neexistuje?
- Najděte číslo mezi 1 a  $N$  s co nejvíce děliteli.
- Vypočtěte Eulerovu funkci (*Euler's totient function*)  $\phi(n)$ , která je rovna počtu s  $n$  nesoudělných (*relatively coprime*) čísel menších než  $n$ .