1. # Programování 1 pro matematiky

2. cvičení, 6-10-2022

tags: `Programovani 1 2022`, `čtvrtek 1`, `čtvrtek 2`

Obsah:

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 nezkopírovaly 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, zřiďte si jej.
2. **Domácí úkoly** Doiterovali jsme se k docela velké úspěšnosti u jediného úkolu, který jsem zadával na předchozím cvičení. Více v osobitné části.

   Pokud ještě nemáte správné řešení, vězte, že dostanete ještě nějaké body i když ho odevzdáte do příští středy.
3. **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) jméno s číslem osoby na kartičce UK (a pro úplnost i email) a Dr. Šimůnek jim to zařídí.
>
> \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

- [x] Základní instalace Pythonu
- [x] Čísla a řetězce, aritmetické a logické operace
- [x] **Domácí úkol:** Konverze `int()`

```
a = input()
b = input()
print(a, b, a>b)

=============== RESTART: C:\Users\kvasn\Dropbox\Python_MFF\sk.py
===============
2
13
2 13 True
```

```
a = int(input())
b = int(input())
print(a, b, a>b)

=============== RESTART: C:\Users\kvasn\Dropbox\Python_MFF\sk.py
===============
2
13
2 13 False
```

Nejspíš vás nepřekvapí, že také existuje `float()`, `str()` a `bool()`

```
In [3]: int(4.9)
Out[3]: 4

In[4]: int("Petr")
Traceback (most recent call last):
  File "<pyshell#72>", line 1, in <module>
    int("Petr")
ValueError: invalid literal for int() with base 10: 'Petr'

In [5]: round(4.9,0)
```

```
Out[5]: 5.0

In [6]: float(5)
Out[6]: 5.0

In [7]: bool(0.5)
Out[7]: True

In [8]: bool(-1.0)
Out[8]: True

In [9]: bool(0.0)
Out[9]: False

In [10]: str(4.6)
Out[10]: '4.6'

In [11]: str(True)
Out[11]: 'True'

In [12]: str(False)
Out[12]: '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*, `import math` a pak `math.*`, např. `math.sin()`.
- [ ] Funkce pro čtení řetězce ze standardního vstupu `input(výzva)` a funkce pro tisk do standardního výstupu `print(objekt1, objekt2, ...)`

  Print podrobněji:

  ```
  print(1,2,3); print(4,5,6)
  1 2 3
  4 5 6
  ```

  Konverze do řetězcové reprezentace, položky oddělené mezerami, na konci znak nového řádku.

  ```
  print(1, 2, 3, sep = "-", end = "!!!\n")
  ```

  Formátování výstupu:

  ```
  jmeno = "Petr"
  vaha = 100
  print(jmeno, "váží", vaha, "kilogramů")
  print(f"{jmeno} váží {vaha} kilogramů")
  ```

- [ ] Podmíněný příkaz

```
if podmínka:
   příkazy
```

- [ ] Příkaz cyklu

```
while podmínka:
   příkazy
```

kde *příkazy* můžou 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`

Úplnější syntaxe příkazu `if`:

```
if podmínka:
   příkazy
else:           # volitelně
   příkazy
```

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

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

```
if podmínka1:
   příkazy
else:
   if podmínka2:
      příkazy
   else:
      příkazy

```

psát

```
if podmínka1:
   příkazy
elif podmínka2:
   příkazy
else:
   příkazy
```

což je o něco přehlednější - hlavně díky plochému (nerostoucímu) odsazení.

Příkaz `while`

```
while podmínka:
   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`:

```
while True:
   příkazy
   if podmínka:
      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`).

```
while podmínka:
   příkazy1
else:
   příkazy2
```

Příklady

Test prvočísel

Chceme otestovat, zda je číslo n ze vstupu prvočíslo.

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

```
#!/usr/bin/env python3

# Otestuje, zda číslo je prvočíslem

n = int(input())
d = 2
mam_delitele = False

while d < n:
    if n%d == 0:
        print("Číslo", n, "je dělitelné", d)
        mam_delitele = True
        break
    d += 1

if not mam_delitele:
    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`:

```
#!/usr/bin/env python3

# Otestuje, zda číslo je prvočíslem (2. pokus)

n = int(input())
d = 2

while d < n:
    if n%d == 0:
        print("Číslo", n, "je dělitelné", d)
        break
    d += 1
else:
    print("Číslo", n, "je prvočíslo")
```

Jak bychom mohli dál vylepšit náš test?

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

```
#!/usr/bin/env python3
# Vypíše všechna prvočísla od 1 do n

n = int(input())

x = 2
while x <= n:
    d = 2
    while d < x:
        if x%d == 0:
            break
        d += 1
    else:
        print(x)

    x += 1

```

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

😎 *Pokročilé kolegy* poprosím o optimalizovaný algoritmus, např. Erastothenovo síto.

Euklidův algoritmus

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

```python
#!/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)

```

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*:

```python
#!/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.

Součet posloupnosti čísel

```python
#!/usr/bin/env python3

# Načteme ze vstupu posloupnost čísel, ukončenou -1.
# Vypíšeme jejich součet.

s = 0
while True:
    n = int(input())
    if n == -1:
        break
    s += n
print(s)
```

Proč nemůžeme na konci jenom stisknout Enter a nezadat nic?

😎 *Pokročilé kolegy poprosím*

- o variantu se stiskem Enter
- a pro vypsání 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.

```
#!/usr/bin/env python3
# Načítá čísla ze vstupu ukončená -1,
# vypíše druhé největší z nich

m1 = 0      # Zatím největší číslo
m2 = 0      # Zatím druhé největší

while True:
    n = int(input())
    if n == -1:
        break

    if n >= m1:
        m1, m2 = n, m1
    elif n >= m2:
        m2 = n

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.

Domácí úkoly na příští týden:

- Spočíst a vypsat počet cifer zadaného celého čísla
- Vypsat zadané číslo jako součin prvočinitelů
- Vypočtěte Eulerovu funkci (*Euler's totient function*) $\phi(n)$, která je rovna počtu s n nesoudělných (*relatively coprime*) čísel menších než n.