Programování 1 pro matematiky

2. cvičení, 08-10-2023

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** Až dnes dostanete první domácí úkoly.
3. **Každému vše chodí?** - Python? ReCodEx?

Kvíz

Které z následujících řetězců jsou správnými názvy proměnné v Pythonu?

```
a123
MamHlad
mám_hlad
délka.chodidla
β
5tibeťanů
```

Opakování + něco nové k tomu

Matematické funkce

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.sin(math.pi / 3)
0.8660254037844386
```

math je modul. Modul je něco jako adresář plný Pythonského kódu, který si přitáhnete do svého programu, a získáte tak přístup k funkcím a třídám modulu.

Nápověda

```
>>> help(math.sin)
>>> help(math)
```

Logické výrazy

```
>>> 5**7 > 7**5
True
>>> math.cos(0) < 0
False
>>> 0.8 <= sin(pi/3) <= 0.9
True
>>> pi>3 and pi<4
True
>>> x>0 or not x>0
True
>>> 1 == 1
True
>>> 1 != 2
True
```

Seznamy, množiny, slovníky

```
>>> seznam = [1, 2, 3]
>>> seznam[0]
1
>>> seznam[1]
2
>>> seznam.append(4)
>>> seznam
[1, 2, 3, 4]
>>> seznam.pop()
4
>>> seznam
[1, 2, 3]
```

```
>>> ovoce = {"jablka", "hrušky", "pomeranče"}
>>> ovoce.add("švestky")
>>> ovoce
{"jablka", "hrušky", "švestky", "pomeranče"}
>>> "hrušky" in ovoce
True
>>> "ananas" in ovoce
False
>>> ovoce.add("hrušky")
>>> ovoce
{"jablka", "hrušky", "švestky", "pomeranče"}
```

```
>>> číslice = {"jedna" : 1, "dva" : 2, "tři" : 3}
>>> číslice["tři"]
3
>>> číslice["čtyři"] = 4
>>> číslice
{"jedna" : 1, "dva" : 2, "tři" : 3, "čtyři" : 4}
```

Znakové řetězce

```
s = 'Hello, world'
s
s[2]
s[2:5]
"wor" in s
```

Můžeme používat jednoduché i dvojité uvozovky, i když jednoduché jsou pro Python typičtější. Indexování začíná nulou. Řetězce můžeme také sečítat:

```
s1='Hello'
s2 = 'world'
s1 + ' ' + s2
```

Náš první program: počítáme od 1 do 10

```
i = 1
while i <= 10:
   print(i)
   i += 1
```

Odsazení funguje jako programovací závorky a je v Pythonu nekompromisně vyžadováno. Musí být konzistentní, tedy stejné a není povoleno střídat mezery a tabulátory. Tedy pokaždé pro stejnou úroveň stejné odsazení.

```
i = 1
while i <= 10:
   if i%2 == 0:
      print(i)
   i += 1
```

Teď se ještě zeptáme, do kolika se má počítat:

```
n = int(input("Do kolika chceš počítat? "))
i = 1
while i <= n:
   if i%2 == 0:
      print(i)
   i += 1
```

Nakonec můžeme přidat do textu komentáře: Python ignoruje znaky za `#` až do konce řádku. Komentář s vykřičníkem v prvním řádku, `#!/usr/bin/env python3`, se nazývá *shebang* a v unixových systémech informuje, jak se má soubor spustit.

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

# Nejprve zjistíme, do kolika počítat
n = int(input("Do kolika chceš počítat? "))

# Aktuální číslo
i = 1
while i <= n:           # Ještě pokračovat?
   if i%2 == 0:         # Je číslo sudé?
      print(i)
   i += 1               # Další, prosím!
```

Radši nepoužívejte v zdrojovém kódu a v komentářích diakritiku, pokud to není nevyhnutné. Můžete občas narazit na nepříjemné problémy.

Vstup z konzole

`print` nám tiskne věci z programu, `input` nám umožňuje načíst z konzole vstup:

```
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
```

input() a ReCodEx

1. Funkce `input()` funguje v ReCodExu ponékud jinak než na vaši konzole. ReCodEx při hodnocení vašich úloh přesměruje soubor se vstupními údaji pro běh programu na standardní vstup programu. Zatímco při použití `input()` na konzoli je výstupem znakový řetězec, který jste zadali, bez koncového znaku nového řádku (\n), v ReCodExu bude vstup obsahovat i koncové \n.
2. Pro úlohy odevzdávané v ReCodExu **nikdy** nepoužívejte výzvu ve funkci `input()`. Tedy vždy volejte funkci bez parametrů, `input()` a ne `input("Zadej cislo: ")` nebo něco podobného.

Introspekce

Objekty v Pythonu vědí, jakého jsou typu, a různé typy můžeme konvertovat na jiné:

```
a = 3.1
b = 4.5
a+b
type(a)
type(b)
c = int(a)
c
type(c)
```

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")
```

Příkazy `if` a `while`

- [ ] 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 `match case`

```
match term:
    case pattern-1:
         action-1
    case pattern-2:
         action-2
    case pattern-3:
         action-3
    case _:
        action-default
```

Tento příkaz nám umožňuje nahradit strukturu `if-elif-else` v případech, kdy vybíráme z většího množství voleb:

Namísto

```
lang = input("What's the programming language you want to learn? ")

if lang == "JavaScript":
    print("You can become a web developer.")
elif lang == "PHP":
    print("You can become a backend developer.")
elif lang == "Python":
    print("You can become a Data Scientist")
elif lang == "Solidity":
    print("You can become a Blockchain developer.")
elif lang == "Java":
    print("You can become a mobile app developer")
else:
    print("The language doesn't matter, what matters is solving problems.")

print(switch("JavaScript"))
print(switch("PHP"))
print(switch("Java"))

"""
Output:
You can become a web developer.
You can become a backend developer.
You can become a mobile app developer
"""
```

můžeme psát

```
lang = input("What's the programming language you want to learn? ")

match lang:
    case "JavaScript":
        print("You can become a web developer.")

    case "Python":
        print("You can become a Data Scientist")

    case "PHP":
        print("You can become a backend developer")

    case "Solidity":
        print("You can become a Blockchain developer")

    case "Java":
        print("You can become a mobile app developer")
    case _:
        print("The language doesn't matter, what matters is solving problems.")
```

V Pythonu existují ještě jiné způsoby implementace mnohonásobného větvení, např. pomocí slovníku.

`match case` ale umí přiřazovat složitější vzory:

```
point = (1, 2)

match point:
    case (0, 0):
        result = "Origin"
    case (x, 0):
        result = f"X-axis at {x}"
    case (0, y):
        result = f"Y-axis at {y}"
    case (x, y):
        result = f"Point at {x}, {y}"
```

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 všechna prvočísla, které jsme dosud nalezli.

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

Součet posloupnosti čísel

```
#!/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._

Domácí úkol na příští týden:

- Obr a princezna (velice snadný úkol pro otestování práce s ReCodExem)
- Spočíst a vypsat počet cifer zadaného celého čísla
- Vypsat zadané celé číslo jako součin prvočinitelů

10 bodů / úkol do pondělí, pak 5 do další neděle. Příští úterý dám návody k věcem, kde uvidím problémy, řešení zveřejním v neděli.