Cvičenie 6

Inštrukcie:

- Úlohy riešte bez použitia zložených dátových typov (zoznamy, reťazce, množiny, slovníky, atď.). Zložene dátové typy sme ešte nepreberali.
- Dodržte interfejs funkcie, ktorý je v zadaní úlohy, t.j. nepridávajte/neuberajte parametre.

Úloha č. 1

Definujte funkciu *sucet_kladnych()* bez parametrov, ktorá bude načítavať čísla z klávesnice, kým používateľ nezadá záporné číslo alebo nulu. Po zadaní záporného čísla / nuly funkcia vráti súčet kladných načítaných čísiel.

Vstupy / výstupy:
Ak volanie *sucet_kladnych()* načíta čísla 10, 4, 0, funkcia **vráti** 14.
Ak volanie *sucet_kladnych()* načíta čísla 1, 2, 3, -1, funkcia **vráti** 6.
Ak volanie *sucet_kladnych()* načíta čísla 1, 5, -5, funkcia **vráti** 6.

Úloha č. 2

Definujte funkciu *stvorce_mensie_n(n)* s parametrom číslom *n*, ktorá **vypíše** všetky kladné štvorce menšie ako číslo *n*.
(Prirodzené číslo nazývame štvorec, ak je druhou mocninou nejakého prirodzeného čísla. Kladné štvorce sú teda napríklad čísla 1, 4, 9, 16, 25, 36, 49, 64, …).

Vstupy / výstupy:
Volanie *stvorce_mensie_n(10)* **vypíše** čísla 1, 4, 9.
Volanie *stvorce_mensie_n(25)* **vypíše** čísla 1, 4, 9, 16.
Volanie *stvorce_mensie_n(26)* **vypíše** čísla 1, 4, 9, 16, 25.

Úloha č. 3

Definujte funkciu *fejkovy_logaritmus(n)* s parametrom číslom *n*, ktorá **vráti** najväčšie prirodzené číslo *x* také, že 2^x je menšie ako *n*. Úlohu vyriešte bez použitia operátora umocnenia **.

Vstupy / výstupy:
Volanie *fejkovy_logaritmus(10)* **vráti** číslo 3.
Volanie *fejkovy_logaritmus(33)* **vráti** číslo 5.
Volanie *fejkovy_logaritmus(32)* **vráti** číslo 4.

Úloha č. 4

Definujte funkciu *index_najvacsieho()* bez parametrov. Funkcia načítava čísla od z klávesnice, kým používateľ nezadá nulu. Po zadaní nuly funkcia **vráti** poradové číslo (index) najväčšieho načítaného čísla, pričom prvé číslo má index 0, druhé číslo má index 1, atď.

Vstupy / výstupy:
Ak volanie *index_najvacsieho()* načíta čísla 10, 4, 0, funkcia **vráti** 0.
Ak volanie *index_najvacsieho()* načíta čísla 1, 2, 4, 0, funkcia **vráti** 2.
Ak volanie *index_najvacsieho()* načíta čísla -1, -2, -4, 0, funkcia **vráti** 3.
Ak volanie *index_najvacsieho()* načíta číslo 0, funkcia **vráti** 0.

Úloha č. 5
Definujte funkciu *vacsi_naslednik()* bez parametrov, ktorá bude načítavať čísla z klávesnice, kým používateľ nezadá nulu. Po zadaní nuly funkcia **vráti** počet koľkokrát sa stalo, že načítane číslo bolo väčšie ako číslo načítane pred ním.

Vstupy / výstupy:
Ak volanie *vacsi_naslednik()* načíta čísla 1,8,9,2,4,-1,0 funkcia **vráti** 4, lebo 1<8, 8<9, 2<4, -1<0.
Ak volanie *vacsi_naslednik()* načíta čísla 5,4,3,2,1,0 funkcia **vráti** 0.

Úloha č. 6
Definujte funkciu *pocet_maxim()* bez parametrov, ktorá bude načítavať čísla z klávesnice, kým používateľ nezadá nulu. Po zadaní nuly funkcia **vráti** koľko z načítaných čísiel je rovných najväčšiemu načítanému číslu.

Vstupy / výstupy:
Ak volanie *pocet_maxim()* načíta čísla 2,8,3,8,0 funkcia **vráti** 2.
Ak volanie *pocet_maxim()* načíta čísla 2,2,2,2,8,1,8,1,8,0 funkcia **vráti** 3.

Úloha č. 7
Definujte funkciu *najdlhsia_podpostupnost()* bez parametrov, ktorá bude načítavať čísla z klávesnice, kým používateľ nezadá nulu. Po zadaní nuly funkcia **vráti** dĺžku najdlhšej podpostupnosti po sebe idúcich rovnakých čísel z postupnosti načítaných čísel.

Vstupy / výstupy:
Ak volanie *najdlhsia_podpostupnost()* načíta čísla 2,2,3,3,3,3,1,3,3,0 funkcia **vráti** 4.

Úloha č. 8 – Fibonacciho postupnosť, úvod
Fibonacciho postupnosť je známa postupnosť čísiel, v ktorej je člen postupnosti rovný súčtu predchádzajúcich 2 členov. Ak a_i označuje i-ty člen postupnosti, potom teda $a_i = a_{i-1} + a_{i-2}$
Prvé 2 členy postupnosti $a_0 = 1$, $a_1 = 1$ – znovu budeme prvý člen postupnosti indexovať číslom 0.
Fibonacciho postupnosť teda tvoria hodnoty: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

Predpokladajme, že je daná funkcia *fibonacci(n)*, ktorá vráti člen Fibonacciho postupnosti s indexom *n,* t.j. volania *fibonacci(0) = 1, fibonacci(1) = 1, fibonacci(2) = 2, fibonacci(3) = 3, fibonacci(4) = 5, fibonacci(5) = 8*, atď.

Kód takejto funkcie by bol:

```
def fibonacci(n):
    if n == 0:
        return 1
    elif n == 1:
        return 1
    else: #ak n >= 2
        predposledny = 1
        posledny = 1
        aktualny = predposledny + posledny #vypocet noveho clena
        for i in range(n-2):
            predposledny = posledny
            posledny = aktualny
            aktualny = predposledny + posledny
        return a_i
```

Úloha č. 8 – Fibonacciho postupnosť, zadanie úlohy
Definujte funkciu *fibonacci_index(x)* s parametrom kladné celé číslo *x*. Funkcia **vráti** index čísla *x* vo Fibonacciho postupnosti, teda funkcia vráti také *i* že $a_i = x$. Ak sa číslo *x* vo Fibonacciho postupnosti nenachádza, funkcia vráti -1. Využite pritom funkciu *fibonacci(n)* definovanú vyššie. Pre hodnotu $x=1$ vráťte 0.

Vstupy a výstupy:
Volanie *fibonacci_index(1)* **vráti** 0.
Volanie *fibonacci_index(2)* **vráti** 2.
Volanie *fibonacci_index(3)* **vráti** 3.
Volanie *fibonacci_index(4)* **vráti** -1.
Volanie *fibonacci_index(5)* **vráti** 4.
Volanie *fibonacci_index(6)* **vráti** -*1*.
Volanie *fibonacci_index(7)* **vráti** -1.
Volanie *fibonacci_index(8)* **vráti** 5.

Úloha č. 9
Matematik *Srinivasa Ramanujan* (podľa jeho života bol natočený film *Génius z chatrče / The Man Who Knew Infinity)* objavil matematický rad, ktorý konverzuje k prevrátenej hodnote čísla pí:

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{k=0}^{\infty} \frac{(4k)!(1103 + 26390k)}{(k!)^4 396^{4k}}$$

Definujte funkciu *odhad_pi(epsilon)*s parametrom desatinným číslom *epsilon,* ktorá pomocou vyššie uvedeného radu odhadne hodnotu čísla pí tak, že rozdiel 2 po sebe idúcich členov vyššie uvedeného radu bude menší ako *epsilon* (viď prednáška a úlohy na Eulerove číslo a odmocninu).

Upozorňujem, že vyššie uvedený vzťah počíta **prevrátenú hodnotu pí** a vašou úlohou je to využiť na výpočet **hodnoty pí**.

Úloha č. 10
Definujte funkciu *kvocient(a, b)*, ktorá má 2 vstupy: nezáporne celé čísla *a* a *b.* Funkcia vráti dolnú celú časť po delení *a/b*, t.j. rovnakú hodnotu, ako by vrátil operátor *a* // *b*. Funkciu implementujte **bez použitia** operátorov //, /, %. (t.j. bez celočíselného delenia, klasického delenia a operátora modulo). V prípade, ze b == 0, funkcia vráti hodnotu None.

Vstupy a výstupy:
Volanie *kvocient(6,3)* **vráti** 2
Volanie *kvocient(1,4)* **vráti** 0
Volanie *kvocient(3,0)* **vráti** None
Volanie *kvocient(11,3)* **vráti** 3

Úloha č. 11
Definujte funkciu *modulo(a, b)*, ktorá má 2 vstupy: nezáporne celé čísla *a* a *b.* Funkcia vráti zvyšok *a* po delení číslom *b*, t.j. rovnakú hodnotu, ako by vrátil operátor *a* % *b*. Funkciu implementujte **bez použitia** operátorov //, /, %. (t.j. bez celočíselného delenia, klasického delenia a operátora modulo). V prípade, ze b == 0, funkcia vráti hodnotu None.

Vstupy a výstupy:
Volanie *modulo(6,3)* **vráti** 0
Volanie *modulo(1,4)* **vráti** 1
Volanie *modulo(3,0)* **vráti** None
Volanie *modulo(11,3)* **vráti** 2

Úloha č. 12
Definujte funkciu *faktorizacia(n)*, ktorá pre argument *n*, ktorým je kladné celé číslo väčšie ako 1, **vypíše** jeho faktorizáciu, t.j. rozklad čísla *n* na súčin prvočísiel. Pre jednoduchosť uvažujte, že program **vypíše** každý faktor na novy riadok. Na poradí vypísaných faktorov nezáleží.

Vstupy a výstupy
Volanie *faktorizacia(10)* postupne vypíše na obrazovku:
2
5
Volanie *faktorizacia*(100) postupne vypíše na obrazovku:
2
2
5
5
Volanie *faktorizacia*(13) postupne vypíše na obrazovku:
13
Volanie *faktorizacia*(80850) postupne vypíše na obrazovku:
2
3
5
5
7
7
11