Cvičenie 5

Inštrukcie:

- Tie úlohy, ktoré nestihnete vypracovať na cvičení, vypracujte doma. Pýtajte sa cvičiacich, ak niečomu nerozumiete alebo si s úlohou neviete dať rady.
- **Pozor! Nevytvárajte rekurzívne funkcie pomocou globálnych premenných!** Je to zlý zvyk a na skúške bude za takéto riešenia 0 bodov!

Sekcia č. 1: Jednoduché rekurzívne úlohy

Úloha č. 1

Definujte funkciu *sucet_rekurzivne(n)* so vstupným parametrom, kladným celým číslom *n*. Funkcia vráti súčet 1+2+3+...+*n.* Funkciu naprogramujte tak, aby používala **rekurziu!**

Vstupy a výstupy:
Volanie *sucet_rekurzivne(1)* vráti 1.
Volanie *sucet_rekurzivne(2)* vráti 3.
Volanie *sucet_rekurzivne(3)* vráti 6.
Volanie *sucet_rekurzivne(10)* vráti 55.

Hint:
V duchu prednášky si položte 2 otázky:
1) (rekurzívna časť) Viem vyriešiť daný problém – súčet n čísiel 1+2+3+...+n tak, že budem predpokladať, že už poznám riešenie toho istého problému pre menší vstup? Napríklad čo ak už mám k dispozícii súčet čísiel od 1 po (n-1), t.j. už poznám 1+2+3+...+(n-1)? Takýto súčet by som predsa vedel získať volaním funkcie sucet_rekurzivne(n-1)

2) (ukončovacia podmienka) Aká je základná/najmenšia inštancia daného problému? Keďže chcem sčítať čísla od 1 po n, tak najmenšia inštancia je situácia, keď n = 1. Vtedy je predsa súčet hneď zrejmý a je to 1.

Úloha č. 2

Definujte funkciu *pocet_parnych_rekurzivne(n)*, ktorá má vstupný parameter, kladné celé číslo *n*. Funkcia načíta z klávesnice *n* čísiel a vráti počet, koľko z načítaných čísiel bolo párnych. Vo funkcii použite **rekurziu.**

Vstupy a výstupy:
Volanie *pocet_parnych_rekurzivne(3)* načíta 3 čísla. Ak čísla boli **10**, 5, 3, funkcia vráti 1.
Volanie *pocet_parnych_rekurzivne(5)* načíta 5 čísiel. Ak čísla boli **10**, 5, 3, **2, 4** funkcia vráti 3.
Volanie *pocet_parnych_rekurzivne(5)* načíta 5 čísiel. Ak čísla boli 1, 5, 3, 1, 9 funkcia vráti 0.

Úloha č. 3

Definujte funkciu *maximum_rekurzivne(n)*, ktorá má vstupný parameter kladné celé číslo *n*. Funkcia načíta *n* čísiel z klávesnice a vráti najväčšie z načítaných čísiel. Vo funkcii použite **rekurziu.**

Vstupy a výstupy:
Volanie *maximum_rekurzivne*(3) načíta 3 čísla. Ak čísla boli 5, 10, 3, funkcia vráti 10.
Volanie *maximum_rekurzivne*(4) načíta 4 čísla. Ak čísla boli -5, -10, -3, -4, funkcia vráti -3.

Úloha č. 4
Definujte funkciu *parita_suctu_rekurzivne(n)* so vstupným parametrom kladným celým číslom *n*. Funkcia načíta z klávesnice *n* čísiel a vráti hodnotu *True*, ak je súčet načítaných čísiel párny. V opačnom prípade vráti hodnotu *False.* Vo funkcii použite **rekurziu.**

Vstupy a výstupy:
Volanie *parita_suctu_rekurzivne(4)* načíta 4 čísla. Ak čísla sú napríklad 1,3,0,-2, funkcia vráti *True*, pretože 1+3+0+(-2) = 2 a 2 je párne číslo.

Hint:
Zamyslite sa, ako sa zmení parita čísla, ak k nemu pripočítam párne číslo a ako sa zmení parita čísla, ak k nemu pripočítam nepárne číslo.

Úloha č. 5
Definujte funkciu *sucet_prvocisiel(n)* so vstupným parametrom kladným celým číslom *n,* ktorá vráti súčet prvočísiel menších ako *n*. Vo funkcii pouzite **rekurziu**! Na testovanie prvočíselnosti môžete použiť funkciu *test_prvociselnosti()* z minulého cvičenia.

Vstupy a výstupy:
Volanie *sucet_prvocisiel(10)* vráti číslo 17, pretože 2+3+5+7 = 17.
Volanie *sucet_prvocisiel(14)* vráti číslo 41, pretože 2+3+5+7+11+13 = 41.
Volanie *sucet_prvocisiel(13)* vráti číslo 28, pretože 2+3+5+7+11 = 28.

Sekcia č. 2: Pokročilé rekurzívne úlohy

Úlohy, ktoré ste riešili v sekcii č. 1 boli pomerne jednoduché a dajú sa hravo riešiť aj bez rekurzie. V tejto sekcii sú uvedené úlohy, kde je rekurzia šikovným nástrojom, ako ich riešiť. Navyše je v týchto úlohách nutné dobre sa zamyslieť, **aké argumenty** budú použité pri rekurzívnom volaní!

Úloha č. 1
Definujte funkciu *je_mocnina(a,b)*, ktorej vstupom sú celé čísla *a,b*. Funkcia vráti *True*, ak je číslo *a* mocninou čísla *b.* V opačnom prípade vráti *False*. Vo funkcii použite **rekurziu** podľa nasledovného pravidla:

Číslo *a* je mocninou čísla *b* vtedy, ak číslo *b* **delí** číslo *a* a zároveň aj *a/b* je mocninou čísla *b*.

Vstupy a výstupy:
Volanie *je_mocnina(8,2)* vráti *True*, pretože $8 = 2^3$
Volanie *je_mocnina(1,3)* vráti *True*, pretože $1 = 3^0$
Volanie *je_mocnina(81,3)* vráti *True*, pretože $81 = 3^4$
Volanie *je_mocnina(80,3)* vráti *False*, pretože 80 nie je mocnina 3 (neexistuje také celé číslo *n* že by platilo $80 = 3^n$)

Hint:
1) Všimnite si, ako je definované pravidlo, či je *a* mocnina *b*. Priamo v ňom sa nachádza návod na rekurziu.
2) Musíte si dobre premyslieť, aké budú ukončovacie podmienky!

Úloha č. 2
Definujte funkciu *GCD(a,b)*, ktorej vstupom sú celé čísla *a,b*. Funkcia vráti *Greatest Common Divisor*, teda **najväčšieho spoločného deliteľa** čísiel *a* a *b.* Vo funkcii použite **rekurziu** podľa nasledovného pravidla:

GCD(a,b) = GCD(b, r), kde *r* je zvyšok *a* po delení *b*, teda r = a % b (zapísané v Pythone)
Ako ukončovaciu podmienku uvažujte situáciu, že *GCD(a,0)* má hodnotu *a*.

Vstupy a výstupy:
Volanie *GCD(10,15)* vráti 5
Volanie *GCD(30,40)* vráti 10
Volanie *GCD(20,27)* vráti 1

Úloha č. 3
Definujte funkciu *dec_to_bin(n)*, ktorá pre vstupný parameter *n*, ktorým je nezáporné celé číslo, **vypíše na obrazovku** binárny rozvoj čísla *n*. Vo funkcii použite **rekurziu**!
(Táto úloha je inšpirovaná úlohou č. 175 z knihy Python Workbook od Bena Stephensona).

Vstupy a výstupy:
Volanie *dec_to_bin(10)* **vypíše** na obrazovku 1010
Volanie *dec_to_bin(0)* **vypíše** na obrazovku 0
Volanie *dec_to_bin(31)* **vypíše** na obrazovku 11111

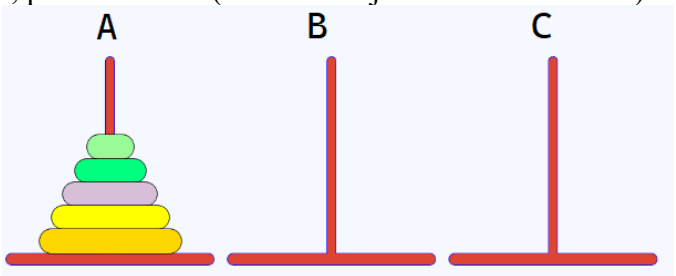
Úloha č. 4
Definujte funkciu *mince(suma, pocet, ...)* ktorá má **minimálne** 2 parametre, *suma* a *pocet*. Funkcia vráti *True* alebo *False* podľa toho, či je možné vytvoriť *sumu* v eurách pomocou zadaného *počtu mincí.* Uvažujme len centové mince, t.j. 0.01€, 0.02€, 0.05€, 0.10€, 0.20€, 0.50€. Mince sa môžu aj opakovať. Funkcia **musí** obsahovať minimálne parametre *suma* a *pocet*, avšak ak to uznáte za vhodné, môžete si pridať aj ďalšie parametre, ak Vám to pomôže naprogramovať danú funkciu.
(Táto úloha je inšpirovaná úlohou c. 181 z knihy Python Workbook od Bena Stephensona).

Vstupy a výstupy:
Volanie *mince(suma=0.04, pocet=4, ...)* vráti *True* pretože 0.04€ je možné zostrojiť pomocou 4 mincí, každá v hodnote 0.01€.
Volanie *mince(suma=0.60, pocet=2, ...)* vráti *True* pretože 0.60€ je možné zostrojiť pomocou 2 mincí ako 0.50€ + 0.10€
Volanie *mince(suma=0.31, pocet=2, ...)* vráti *False* pretože 0.31€ nie je možné zostrojiť pomocou 2 mincí žiadnym spôsobom.
Volanie *mince(suma=0.31, pocet=3, ...)* vráti *True* pretože 0.31€ je možné zostrojiť pomocou 3 mincí ako 0.20€ + 0.10€ + 0.01€.

Úloha č. 5
Upravte predošlú funkciu tak, aby funkcia okrem vrátenia *True/False* v prípade, že je sumu možne zostrojiť, vypísala, aké mince je potrebne použiť.

Úloha č. 6 – Hanojské veže – teoretický úvod
Hanojské veže patria k najkrajším úlohám na rekurziu. Výnimočnosť tejto úlohy spočíva v tom, že pomocou rekurzie je riešenie tohto problému možné implementovať pár riadkami. Ak by sme sa pokúsili túto úlohu riešiť bez rekurzie, program by bol komplikovanejší.

Hanojské veže sú nasledovný hlavolam. Predstavte si, že máte 3 stĺpy, označené A, B, C a na ľavom stĺpe A je niekoľko diskov, pričom disky sú rôznych veľkostí a sú zoradené zhora-nadol od najmenšieho po najväčší, pozri obrázok (na obrázku je verzia s 5 diskami):

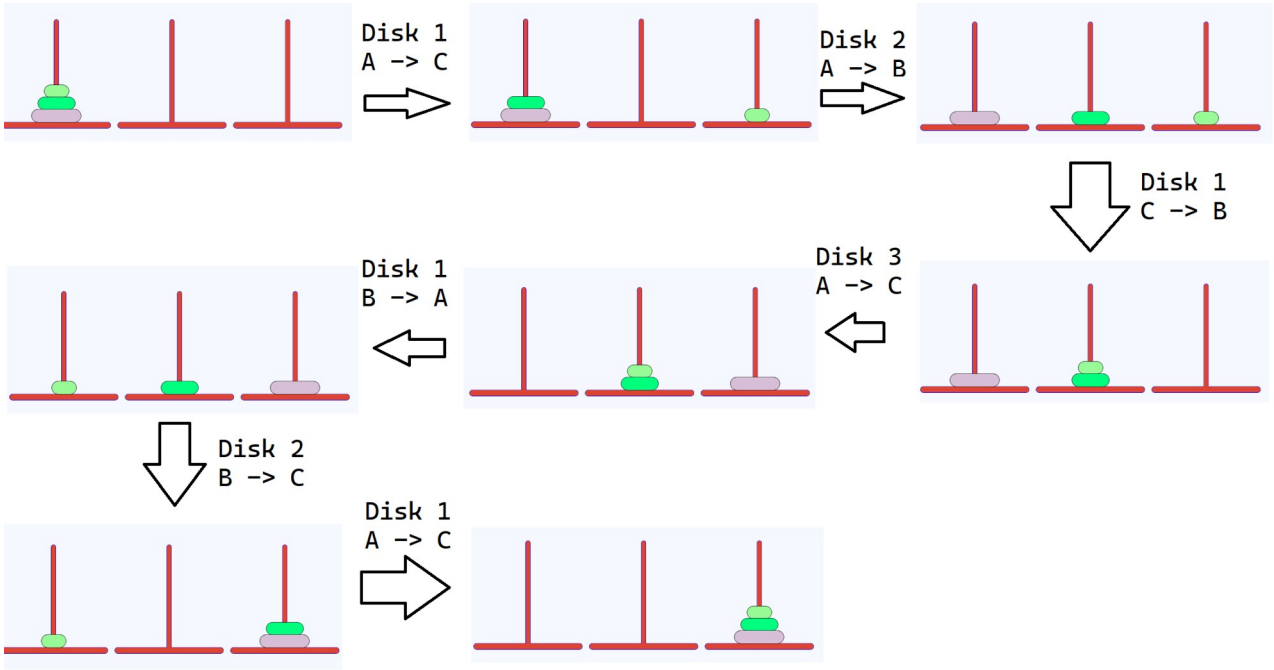


Úlohou je presunúť všetky disky zo stĺpu A na stĺp C tak, aby aj v stĺpe C boli disky uložené tak, že sú zoradené zhora-nadol od najmenšieho po najväčší. Zároveň sa disky môžu presúvať **len podľa nasledovných pravidiel:**
1) V jednom okamihu je možné presúvať len 1 disk.
2) Nie je dovolené položiť väčší disk na menší disk.

Stĺp B je možné počas presúvania používať ako pomocné úložisko diskov.

Samotný hlavolam si viete skúsiť zahrať tu:
https://www.mathsisfun.com/games/towerofhanoi.html

Napríklad ak by sme mali verziu s 3 diskami, tak riešenie uvedeného hlavolamu by mohlo vyzerať nasledovne. Stĺpy sú označené zľava-doprava A,B,C; disky sú očíslované 1,2,3, pričom 1 je najmenší, 2 stredný a 3 najväčší:



Vyššie uvedené riešenie vieme popísať nasledovne:
Disk 1, A -> C
Disk 2, A -> B
Disk 1, C -> B
Disk 3, A -> C
Disk 1, B -> A
Disk 2, B -> C
Disk 1, A -> C

Vidíme teda, že verziu s 3 diskami by sme vedeli vyriešiť a vieme ich presunúť zo stĺpu A na stĺp C, pričom stĺp B použijeme ako pomocný stĺp pri presúvaní.

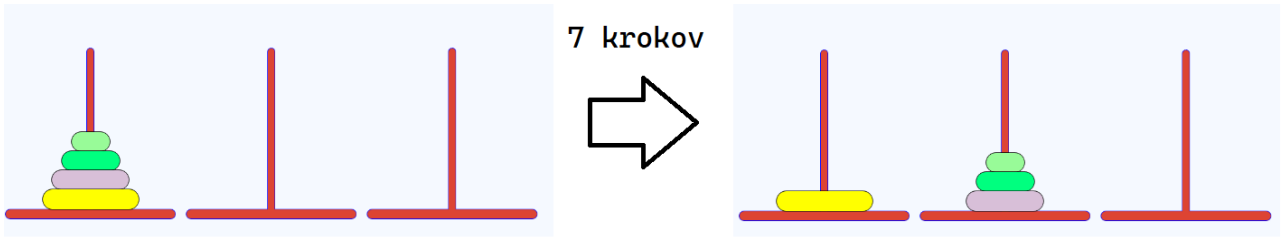
Ako by sme riešili verziu so 4 diskami?

Pri hľadaní postupnosti, ako presunúť 4 disky zo stĺpu A na stĺp C by sme mohli **vychádzať z toho, že vieme riešiť verziu s 3 diskami!**
V predošlej verzii s 3 diskami sme našli postupnosť krokov, ako presunúť 3 disky zo stĺpu A na stĺp C, pričom stĺp B použijeme ako pomocný. To ale znamená, že jednoduchou zámenou B <-> C by sme vedeli nájsť aj postup, ako presunúť 3 disky zo stĺpu A na stĺp B, pričom stĺp C použijeme ako pomocný!

Predstavme si teda, že by sme vo verzii so 4 diskami presunuli prvé 3 disky zo stĺpu A na stĺp B, pričom použijeme stĺp C ako pomocný. Ako píšeme vyššie, taký postup určite existuje:

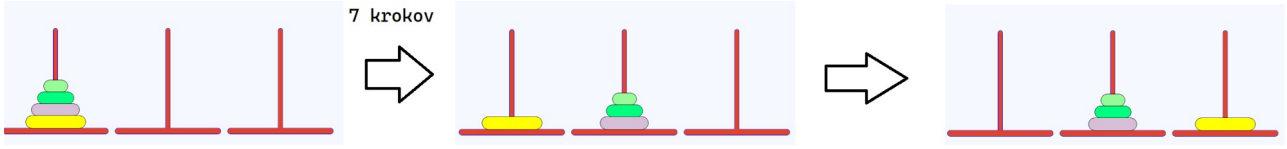
Disk 1, A -> B
Disk 2, A -> C
Disk 1, B -> C
Disk 3, A -> B
Disk 1, C -> A
Disk 2, C -> B
Disk 1, A -> B

Dostaneme situáciu:



Po tomto presune 3 diskov zo stĺpu A na stĺp B sme si „oslobodili“ najväčší disk číslo 4, ktorý presunieme na stĺp C, t.j.:

Disk 4, A -> C



Následne už len potrebujeme presunúť znovu 3 disky, tentokrát zo stĺpu B na stĺp C, pričom máme voľný stĺp A, ktorý vieme využiť ako pomocný. Ale ako sme povedali, my **poznáme postup** ako presunúť 3 disky! Tentokrát len potrebujeme presunúť disky zo stĺpu B na stĺp C za pomoci stĺpu A:
Disk 1, B -> C
Disk 2, B -> A
Disk 1, C -> A
Disk 3, B -> C
Disk 1, A -> B
Disk 2, A -> C
Disk 1, B -> C

Ak si to teda zhrnieme. **Vedeli sme, ako presunúť 3 disky** zo stĺpu A na stĺp C za pomoci B. Tento postup sme **využili pri presune 4 diskov** nasledovným spôsobom:

1) presuň 3 disky z A na B za pomoci C
2) presuň 4-tý disk z A na C
3) presuň 3 disky z B na C za pomoci A.

pomocou postupnosti krokov:

Disk 1, A -> B
Disk 2, A -> C
Disk 1, B -> C
Disk 3, A -> B
Disk 1, C -> A
Disk 2, C -> B
Disk 1, A -> B
Disk 4, A -> C
Disk 1, B -> C
Disk 2, B -> A
Disk 1, C -> A
Disk 3, B -> C
Disk 1, A -> B
Disk 2, A -> C
Disk 1, B -> C

Ak teraz teda poznáme postupnosť krokov, ktoré presunú 4 disky zo stĺpu A na stĺp C za pomoci stĺpu B, ako by sme presunuli 5 diskov?

1) presuň 4 disky zo stĺpu A na stĺp B za pomoci stĺpa C
2) presuň 5-ty disk zo stlpu A na stlp C
3) presuň 4 disky zo stĺpu B na stĺp C za pomoci stĺpa A

VŠIMNITE SI!!!

Presúvame *n* diskov zo stĺpu A na stĺp C za pomoci stĺpca B tak, že :
1) presunieme *n-1* diskov podľa známeho postupu zo stĺpu A na stĺp B za pomoci stĺpu C
2) presunieme *n*-tý disk zo stĺpu A na stĺp C
3) presunieme *n-1* diskov podľa známeho postupu zo stlpu B na stĺp C za pomoci stĺpu A.

V prvom a treťom kroku je **REKURZIA!!!** Riešime problém presunu *n* diskov tak, že **predpokladáme,** že máme riešenie presunu menšieho počtu diskov, konkrétne *n-1* a následne z tohto riešenia jednoducho nájdeme riešenie pre *n* diskov!

A čo by bola ukončovacia podmienka? Čo je tá najmenšia inštancia problému presunu diskov, ktorú vieme riešiť priamo hneď?

No predsa presun $n = 1$ disku, ktorý priamo preložíme z počiatočného stĺpu na konečný.

Napíšme teraz v jazyku Python program, ktorý postupne vypíše na obrazovku inštrukcie, ktorý disk kam treba presunúť, aby sa vyriešil problém Hanojských veží pre *n* diskov.

Definujeme funkciu *hanojske_veze(n, start, pomocny, koniec)*, ktorá má 4 parametre:
n – počet diskov
start – označenie štartovacieho stĺpu (t.j. odkiaľ presúvame)
pomocny – označenie pomocného stĺpu
koniec – označenie konečného stĺpu (t.j. kam presúvame)

```
def hanojske_veze(n, start, pomocny, koniec):
    if n == 1:                                  } Ukončovacia podmienka. Ak presúvame len n=1 disk, presúvame
        print(n,start,'->',koniec)              } ho priamo zo štartu na koniec
    else:                                                Rekurzívna časť:
        hanojske_veze(n-1, start, koniec, pomocny)       1) Najprv presunieme n-1 diskov zo štartu na
        print(n,start,'->',koniec)                       pomocný stĺp cez konečný stĺp
        hanojske_veze(n-1, pomocny, start, koniec)       2) Potom presunieme n-tý disk zo štartu na
                                                         konečný stĺp
                                                         3) Na záver presunieme n-1 diskov z pomocného
                                                         stĺpu na konečný cez počiatočný stĺp
```

Ak by sme teraz chceli, aby program vypísal riešenie pre povedzme 4 disky a stĺpy by boli označené: A, B, C, kde A je štartovací, B je pomocný a C je konečný, funkciu zavoláme:

```
hanojske_veze(4,'A','B','C')
1 A -> B
2 A -> C
1 B -> C
3 A -> B
1 C -> A
2 C -> B
1 A -> B
4 A -> C
1 B -> C
2 B -> A
1 C -> A
3 B -> C
1 A -> B
2 A -> C
1 B -> C
```

Ak stále nerozumiete, ako sme úlohu vyriešili, skúste si pozrieť nasledovné video:

https://www.youtube.com/watch?v=8lhxIOAfDss

Komentár č. 1:
Vo videu používa profesor Altenkirch prostredie Jupyter. Rovnaké funkcie, ktoré vytvára a spúšťa prof Altenkirch vo videu, si ale môžete vytvoriť a spustiť aj v prostredí Idle.

Komentár č. 2
Vo videu vytvori prof Altenkirch funkciu

```
def move(f,t):
    print("Move a disc from {} to {}!".format(f,t))
```

V tejto funkcii využíva prof Altenkirch metódu format() , s ktorou sme sa ešte na predmete nestretli. Ak by sme sa chceli vyhnúť metóde format(), mohli by sme funkciu move definovať nasledovne

```
def move(f,t):
    print("Move a disc from "+f+" to "+t+"!")
```

Takto definovaná funkcia move ma rovnaký efekt ako funkcia move od prof Altenkircha. Namiesto metódy format() sa v nej využíva iba spájanie reťazcov, ktoré sme preberali ešte na prvej prednáške.

Úloha č. 6 – Hanojské veže – úloha pre Vás!
Definujte funkciu *hanojske_veze_pocet_presunov(n)*, ktorá má 1 vstupný parameter *n*, ktorým je kladné celé číslo udávajúce počet diskov. Funkcia vráti **počet presunov diskov**, ktorý je potrebný na vyriešenie hlavolamu, t.j. presun *n* diskov zo štartovacieho na konečný stĺp.

Vstupy a výstupy:
Volanie *hanojske_veze_pocet_presunov(3)*vráti číslo 7, pretože ako môžete vidieť vyššie, na presun 3 diskov je potrebné vykonať 7 presunov.
Volanie *hanojske_veze_pocet_presunov(4)*vráti číslo 15, pretože ako môžete vidieť vyššie, na presun 4 diskov je potrebné vykonať 15 presunov.

Doplnok k úlohe:
Legenda hovorí, že v indickom meste Váránasí (alebo Benáres) je chrám zasvätený bohu Brahma, v ktorom je verzia hry Hanojské veže so zlatými 64 diskami. Miestni mnísi postupne každý deň presunú 1 disk. V momente, keď presunú všetkých 64 diskov na cieľový stĺp, nastane koniec sveta.

Napíšte program, ktorý postupne vypočíta, koľko rokov by trvalo vyriešiť verziu Hanojských veží s 1, 2, 3, 4, 5, 6, … 64 diskami, ak predpokladáme, že mnísi za 1 deň vykonajú 1 presun disku a rok má 365 dní.

Sekcia č. 3 – Debugger v prostredí IDLE

1. Vývojové prostredia pre tvorbu počítačových programov spravidla obsahujú nastroj zvaný Debugger. Pomocou tohto nástroja môžete prechádzať vašim programom krok po kroku a môžete sledovať, ako sa menia hodnoty premenných vo vašom programe. Toto je veľmi užitočne pri hľadaní chýb vo vašom programe. Prejdite si tutorial o používaní Debuggera vo vývojovom prostredí IDLE dostupný na:

https://www.cs.uky.edu/~keen/help/debug-tutorial/debug.html

Osvojte si prácu s Debuggerom v IDLE. **Bude to pre vás užitočné!** Ak radšej používate iné vývojové prostredie ako IDLE, osvojte si prácu s Debuggerom vo vašom preferovanom vývojovom prostredí!