PROG1: Prednáška 2
Funkcie a for-cykly

Čo bolo minule...

- Minulý týždeň sme si ukázali, ako v Pythone naprogramovať veľmi jednoduché programy, ktoré pracujú s premennými, realizujú nejaký výpočet a vypíšu výsledok na obrazovku.

- Každý program, ktorý sme si ukázali, pozostával z niekoľkých príkazov, ako napríklad priradenie hodnoty do premennej, zmena hodnoty v premennej alebo výpis nejakej hodnoty na obrazovku.

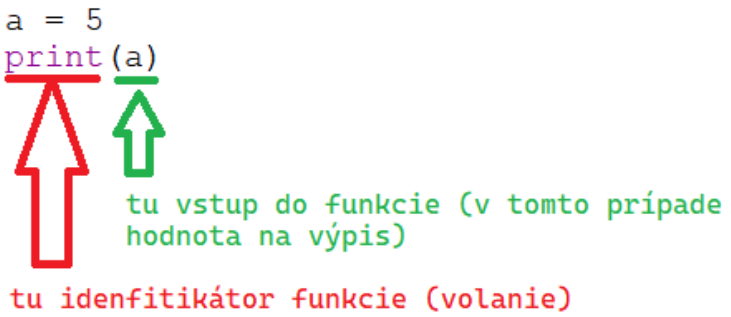
- Na úvod si vysvetlíme, ako vytvárať menšie *podprogramy* nazývané **funkcie,** ktoré následne môže programátor ďalej využívať.

Funkcie

- Na úvod sa teda budeme baviť o *funkciách*.
- **Funkcia** v programovaní, je *pomenovaná* postupnosť príkazov, ktorá vykonáva nejakú činnosť.
- V podstate aj každý program, ktorý sme minulý týždeň naprogramovali, bol taká malá funkcia (akurát, že sme mu nedali žiadne meno).
- Minulý týždeň sme sa už s niečím ako funkcia stretli – používali sme funkciu **print()**
- **print()** je príklad **vstavanej funkcie** v jazyku Python, t.j. ide o funkciu, ktorá tvorí súčasť jazyka Python a programátor ju v jazyku Python môže rovno používať.

Funkcia print()

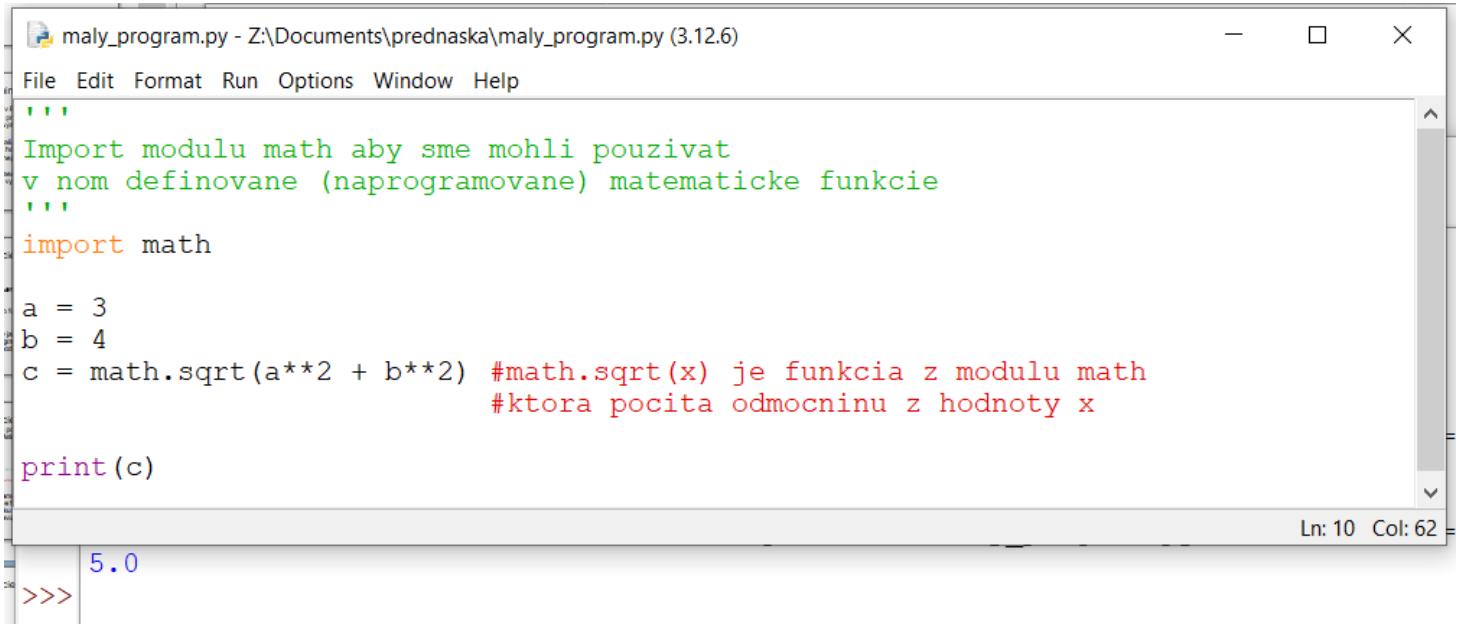
- Pripomeňme si, ako sa funkcia print() používala. Napríklad uvedený kód vytvorí premennú *a* s hodnotou 5 a následne ju vypíše na obrazovku



- Funkcia **print()** je v Pythone naprogramovaná tak, že keď ju programátor vo svojom programe použije – použitie funkcie nazývame jej **volanie (call)** - tak funkcia vypíše na obrazovku takú hodnotu, ktorú dostala ako svoj **vstup** – vstup do funkcie predstavuje tú hodnotu, ktorú uvedieme do **zátvoriek** pri jej volaní

Funkcie matematické

- Iný príklad funkcií sú predprogramované matematické funkcie z modulu *math* (pozri prvá prednáška):



Funkcie matematické

- Všimnite si na predchádzajúcom príklade, že funkcia *sqrt()* z modulu *math* mala vstup – v tomto prípade ním bola hodnota výrazu *a**2 + b**2*

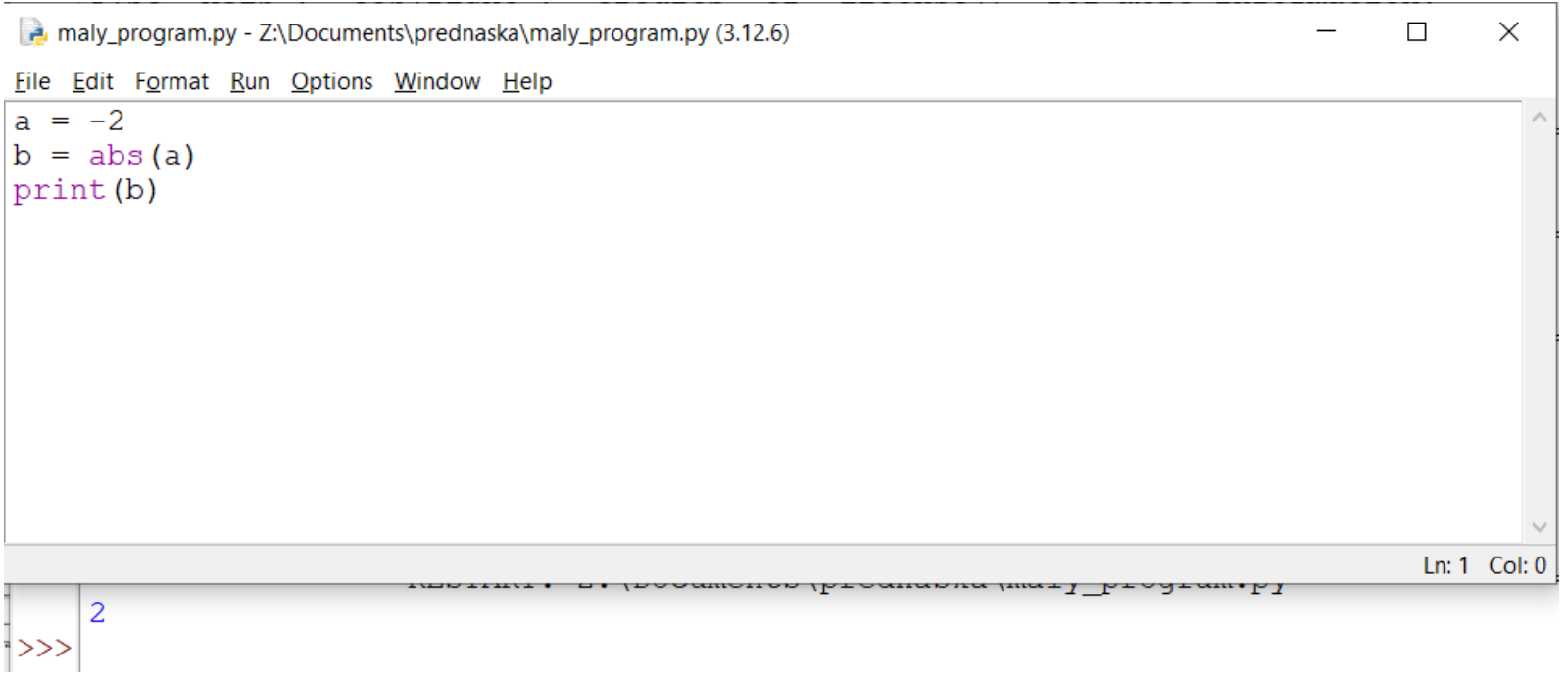
- A zároveň sme výsledok funkcie sqrt() priradili do premennej ***c***
- To znamená, že funkcia sqrt() vypočítala nejakú hodnotu, s ktorou sme ďalej mohli pracovať!

- V programovaní často používame aj také funkcie, ktoré nielen realizujú nejaký výpočet, ale následne chceme zistiť, aký bol výsledok daného výpočtu a neskôr s touto hodnotou pracovať **mimo danú funkciu**.

- O funkciách, ktoré vypočítajú nejakú hodnotu, ktorú následne **odovzdajú** programu na ďalšie spracovanie hovoríme, že **vracajú** tzv. **návratovú hodnotu.**

Funkcia abs()

- Ako ďalší príklad takejto funkcie je iná vstavaná funkcia v jazyku Python, funkcia **abs()**.
- abs() je funkcia, ktorá **vráti** absolútnu hodnotu svojho **vstupu**:



Funkcie

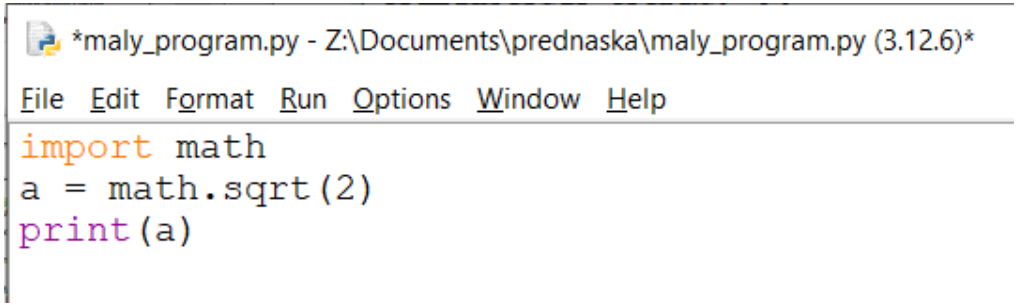
- Každá funkcia v jazyku Python teda má:
  - 1) Svoje meno (**identifikátor**), pomocou ktorého ju **voláme** (používame)
  - 2) Môže mať nejaké **vstupy**, s ktorými pracuje (vstup = hodnota/hodnoty vstupujúce do funkcie)
  - 3) Môže **vracať** nejakú hodnotu, t.j. funkcia po svojom skončení oznámi programu, ktorý ju volal, akú hodnotu vypočítala

Porovnanie print() vs abs()

- Funkcia print():
  - Jej **identifikátor**, pod ktorým ju voláme je **print**
  - Jej **vstupom** je nejaká hodnota (číslo/reťazec/premenná/výraz), ktorý funkcia vypíše na obrazovku
  - Funkcia print **nič nevracia!!!**

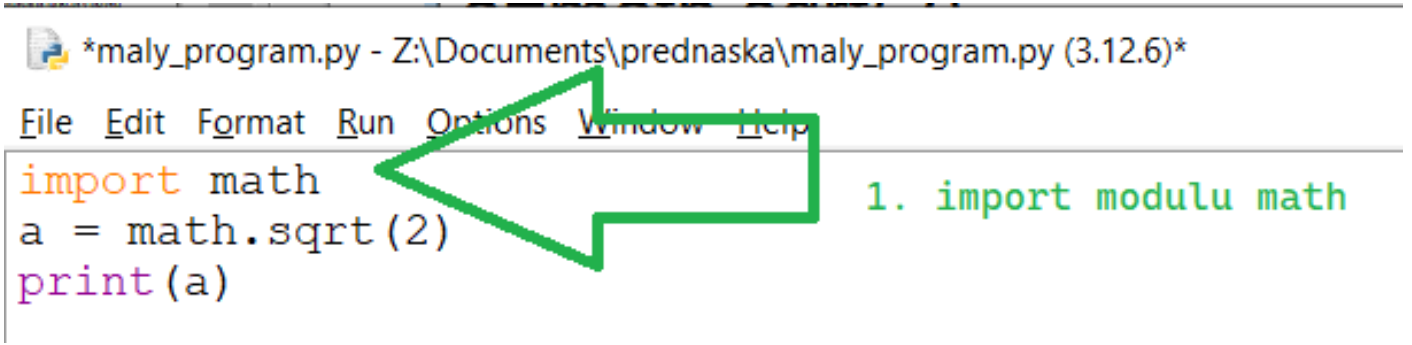
- Funkcia abs():
  - Jej **identifikátor**, pod ktorým ju voláme je **abs**
  - Jej **vstupom** je nejaká číselná hodnota, ktorej absolútnu hodnotu funkcia počíta
  - Funkcia **vráti absolútnu hodnotu vstupu!**

Príklad č. 1



- V druhom riadku používame (**voláme)** funkciu sqrt() z modulu math.
- Vidíme, že do funkcie sqrt vkladáme ako **vstup** číslo 2. Toto číslo predstavuje vstup do funkcie *odmocnina,* teda funkcia by mala vypočítať odmocninu z čísla 2.
- Funkcia sqrt funguje tak, že vypočítanú odmocninu, t.j. hodnotu 1.4142… **vráti** ako svoju návratovú hodnotu – preto túto hodnotu vieme vložiť do premennej *a* a následne s ňou pracovať.

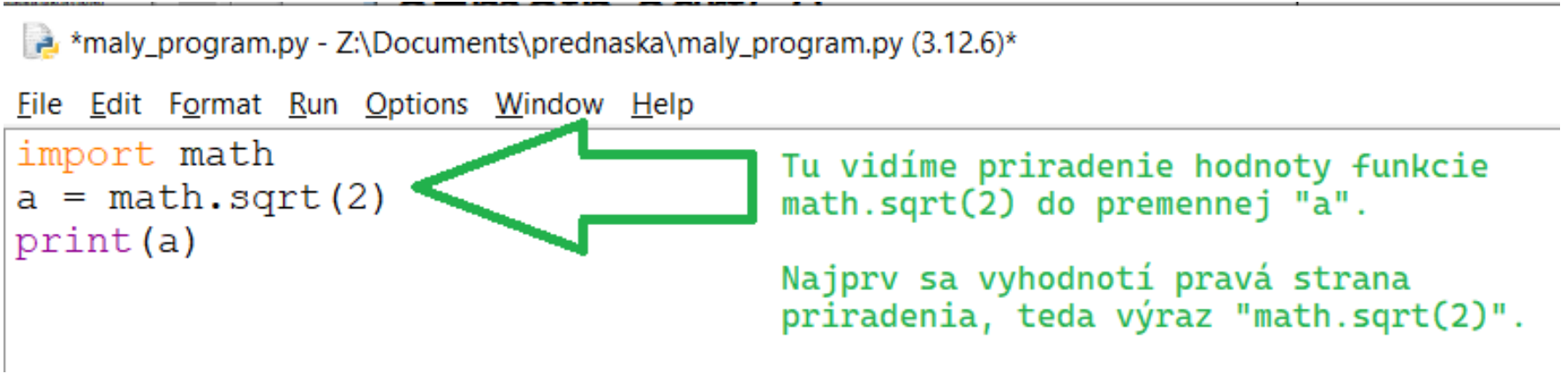
Ako to funguje?



Prejdime si krok-za-krokom, ako sa takýto program v jazyku Python vykonáva v počítači. Príkazy sú vykonávané postupne tak, ako sú napísané v zdrojovom kóde.

Teda prvý príkaz „import math“ urobí to, že hovorí Pythonu, že v našom programe budeme využívať modul *math*, teda predprogramované matematické objekty (funkcie).

Ako to funguje?

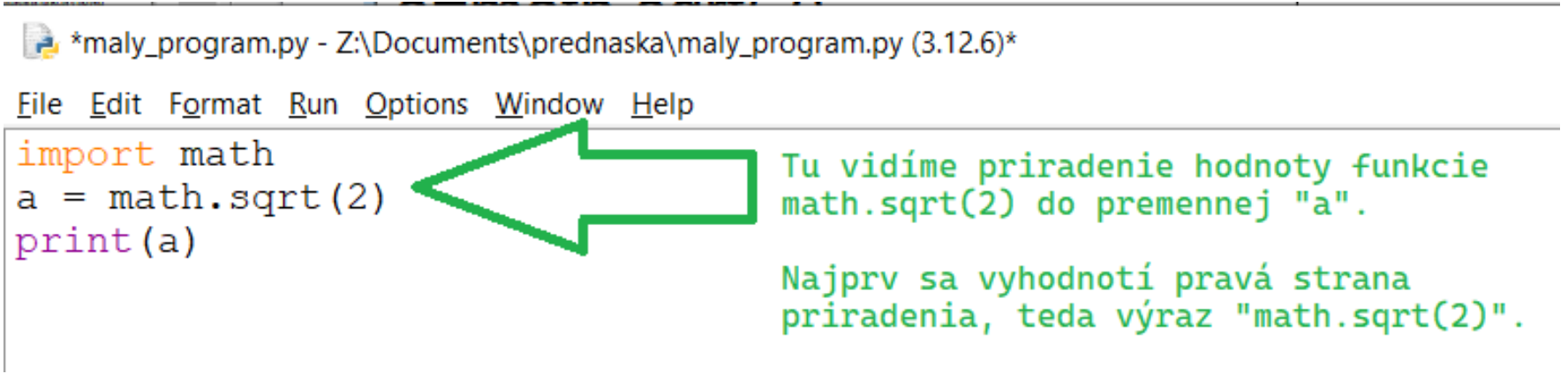


Na druhom riadku vidíme priradenie, kde sa do premennej „a“ priradí hodnota „math.sqrt(2)“.

Python **vie**, že math.sqrt() je funkcia. Urobí teda to, že spustí príslušnú funkciu a zároveň do nej vloží ako jej vstupnú hodnotu číslo 2.

To znamená, že niekde vo Vašom počítači už existuje naprogramovaná funkcia math.sqrt(), ktorú Python spustí pre vstupnú hodnotu 2. Váš program teraz **čaká, kým funkcia math.sqrt() so vstupom 2 skončí!**

Ako to funguje?

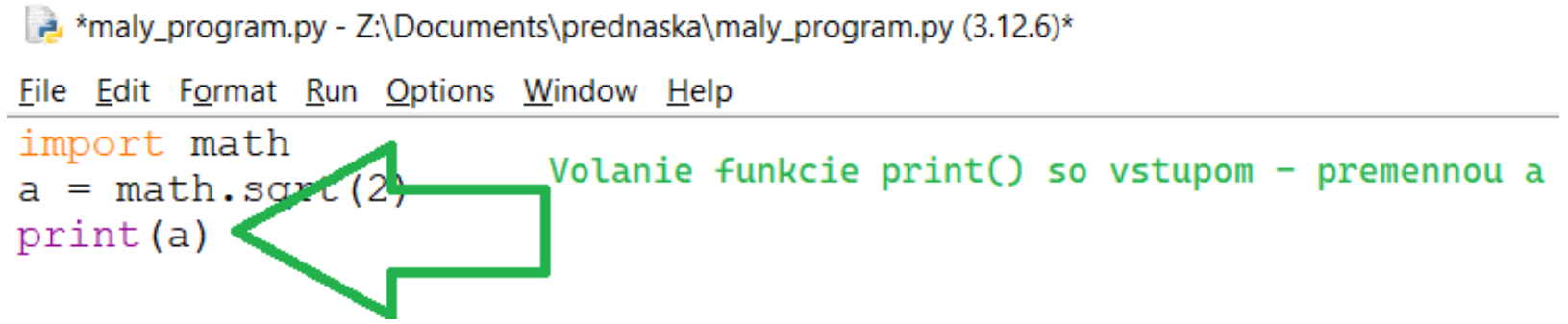


Akonáhle funkcia math.sqrt(2) skončí, tak táto funkcia **vráti návratovú hodnotu**.

Keďže funkcia math.sqrt() je naprogramovaná tak, aby ako návratovú hodnotu vrátila odmocninu svojho vstupu, v tomto prípade funkcia math.sqrt() pre vstup 2 vráti hodnotu približne 1.4142135623730951

A až táto hodnota sa **priradí do premennej „a“**.

Ako to funguje?



Posledný príkaz v programe je volanie funkcie print() so vstupnou hodnotou – obsahom premennej a

Keďže aktuálne sa v premennej a nachádza číslo 1.4142135623730951, tak sa **zavolá funkcia print()** - t.j. Python spustí túto funkciu a vloží do nej ako vstup hodnotu 1.4142135623730951. Funkcia print() je naprogramovaná tak, aby zobrazila svoj vstup na obrazovku – v tomto prípade teda zobrazí hodnotu 1.4142135623730951 na obrazovku.

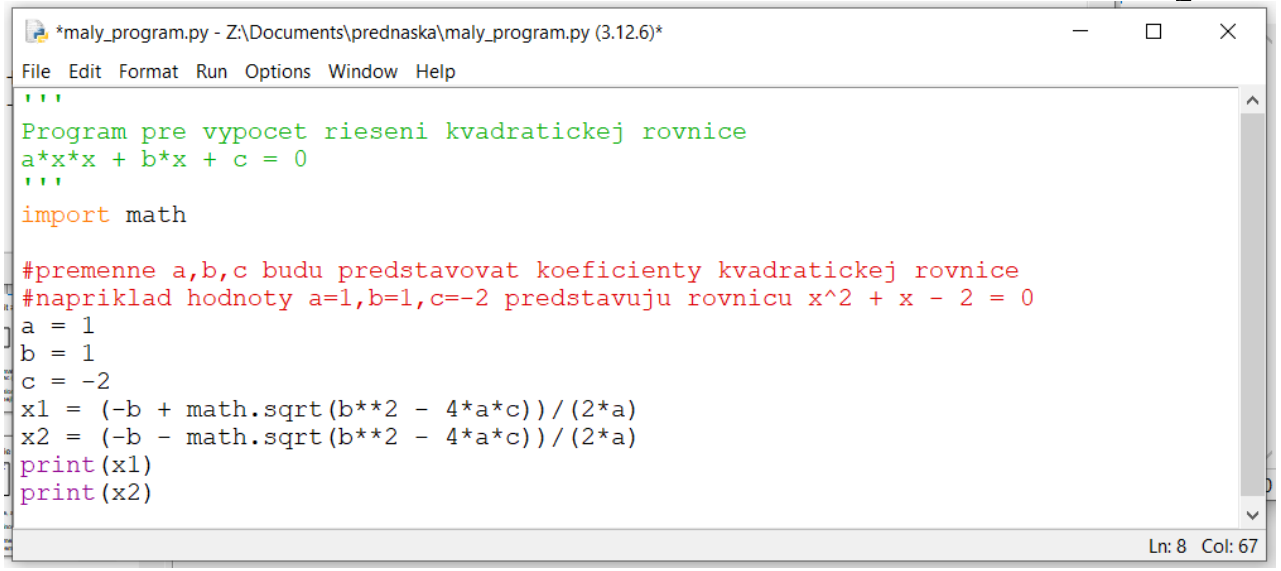
Volanie funkcií

- Vstup funkcie je nejaká hodnota – ako vstup môžeme teda v zdrojovom kóde uviesť aj nejaký výraz
  - V takom prípade Python najprv daný výraz vyhodnotí a až potom vloží do funkcie ako vstupnú hodnotu

- Podobne, keďže návratová hodnota z funkcie predstavuje nejakú hodnotu, môžeme volanie funkcie použiť ako súčasť nejakého výrazu / hodnoty.

- Pozri nasledovný príklad

Príklad č. 2 – riešenie kvadratickej rovnice



Všimnite si, **kde** je vložené volanie funkcie math.sqrt() a **čo** je uvedené ako vstup do volania math.sqrt():

1) najprv sa na základe hodnôt premenných a,b,c vyhodnotí výraz *b**2-4*a*c*
2) následne sa táto hodnota vloží do funkcie *math.sqrt(b**2-4*a*c)* ako vstup
3) výsledná návratová hodnota volania *math.sqrt(b**2-4*a*c)* sa potom použije ako hodnota vo výraze -b+math.sqrt(b**2-4*a*c)/(2*a)

Definovanie nových funkcií

- Doteraz sme sa bavili o funkciách, ktoré vytvoril už niekto iný – print(), abs(), matematické funkcie z modulu math, …

- V jazyku Python (a v podstate vo väčšine programovacích jazykov) si môže programátor vytvárať aj funkcie **vlastné!**

- Na začiatku sme si povedali, že každá funkcia je vlastne pomenovaná postupnosť príkazov.
- Teda každá funkcia má nejaký vlastný **zdrojový kód** (postupnosť príkazov), ktorý sa spustí, keď sa daná funkcia zavolá. Tieto príkazy nazývame **telo funkcie.** Zároveň má každá funkcia nejaké vlastné meno (identifikátor).

Definovanie funkcie bez vstupov

- Na úvod si povieme, ako vytvoríme jednoduchú funkciu, ktorá nemá žiadne vstupy.

- Vytvorenie novej funkcie v programovaní nazývame **definícia funkcie.**

- Novú funkciu, ktorá nemá vstupy, v jazyku Python definujeme takto:

def *meno_funkcie*(): #tento prvý riadok sa nazýva **hlavička funkcie**

prikaz
prikaz
….
prikaz

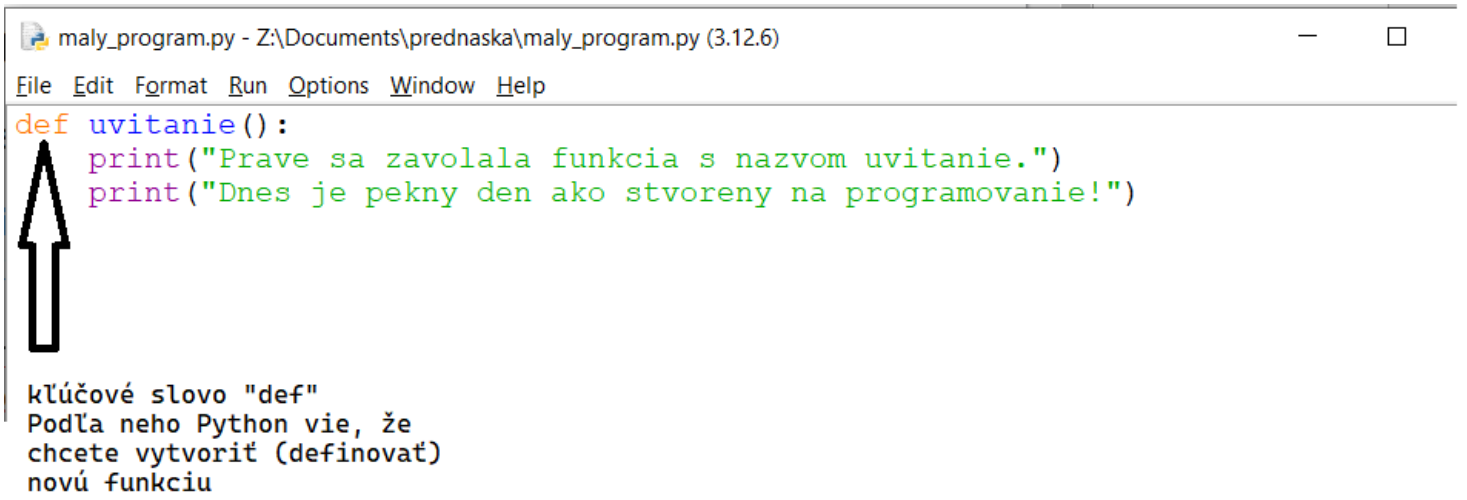
Tieto príkazy tvoria **telo funkcie** – teda príkazy, ktoré sa vykonajú pri volaní funkcie.
Každý príkaz musí byť odsadený nejakým počtom medzier (spravidla 4) od začiatku riadku, aby Python vedel, že ide o príkazy, ktoré tvoria **telo funkcie**.

Definovanie funkcie bez vstupov

- Definovanie novej funkcie bez vstupov si ukážeme na príklade.

- Predstavme si, že by sme chceli vytvoriť funkciu s názvom **uvitanie()**, ktorá nemá vstupné argumenty a ktorej funkcionalitou bude výpis 2 nasledovných textových reťazcov na obrazovku:
  - „Prave sa zavolala funkcia s nazvom uvitanie.“
  - „Dnes je pekny den ako stvoreny na programovanie!“

Definovanie funkcie bez vstupov



Pri definovaní novej funkcie musím ako prvé uviesť kľúčové slovo „def“.
Podľa neho Python vie, že ja ako programátor idem definovať novú funkciu.

Definovanie funkcie bez vstupov

identifikátor (meno funkcie)

```
def uvitanie():
    print("Prave sa zavolala funkcia s nazvom uvitanie.")
    print("Dnes je pekny den ako stvoreny na programovanie!")
```

Za kľúčovým slovom *def* sa nachádza identifikátor funkcie, t.j. jej meno, pod ktorým budem funkciu volať, v tomto prípade **uvitanie**

Definovanie funkcie bez vstupov

Definícia funkcie bez vstupov má za identifikátorom uvedené len prázdne okrúhle zátvorky a dvojbodku

```
def uvitanie():
    print("Práve sa zavolala funkcia s nazvom uvitanie.")
    print("Dnes je pekny den ako stvoreny na programovanie!")
```

Za identifikátorom funkcie v prípade funkcie bez vstupov uvedieme prázdne zátvorky a **dvojbodku:**

Definovanie funkcie bez vstupov

Tento prvý riadok sa nazýva hlavička funkcie

```
def uvitanie():
    print("Prave sa zavolala funkcia s nazvom uvitanie.")
    print("Dnes je pekny den ako stvoreny na programovanie!")
```

Prvý riadok definície funkcie, t.j. kľúčové slovo „def“ + identifikátor funkcie a zoznam vstupov (v tomto prípade prázdny) sa nazýva aj **hlavička funkcie.**

Definovanie funkcie bez vstupov

```
def uvitanie():
    print("Prave sa zavolala funkcia s nazvom uvitanie.")
    print("Dnes je pekny den ako stvoreny na programovanie!")
```

Všimnite si, že telo funkcie tvoria príkazy odsadené od začiatku riadku!
Štandardne sa používa odsadenie o 4 medzery.

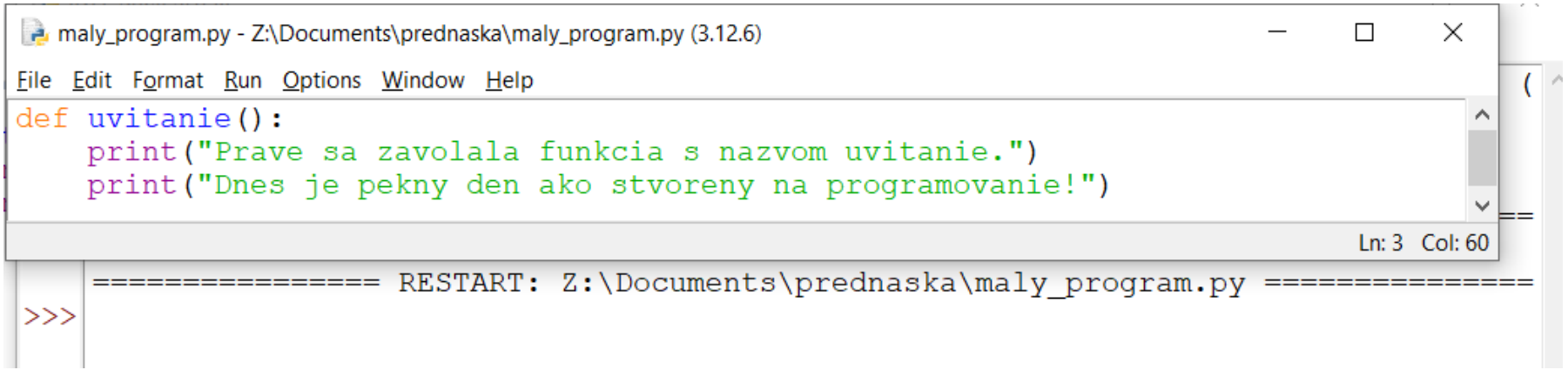
Telo funkcie
Prikazy, ktore sa vykonaju po zavolani funkcie

Za hlavičkou funkcie sa od ďalšieho riadku začína **telo funkcie**, t.j. príkazy, ktoré sa spustia v prípade volania funkcie**. Každý príkaz** v tele funkcie musí byť odsadený od začiatku riadku (spravidla sa používa odsadenie 4 medzery) – je to kvôli tomu, aby Python vedel, ktoré príkazy tvoria telo funkcie.

Zároveň to robí kód ľahšie čitateľným.

Volanie funkcie

Čo sa stane, ak spustíme nasledovný kód?

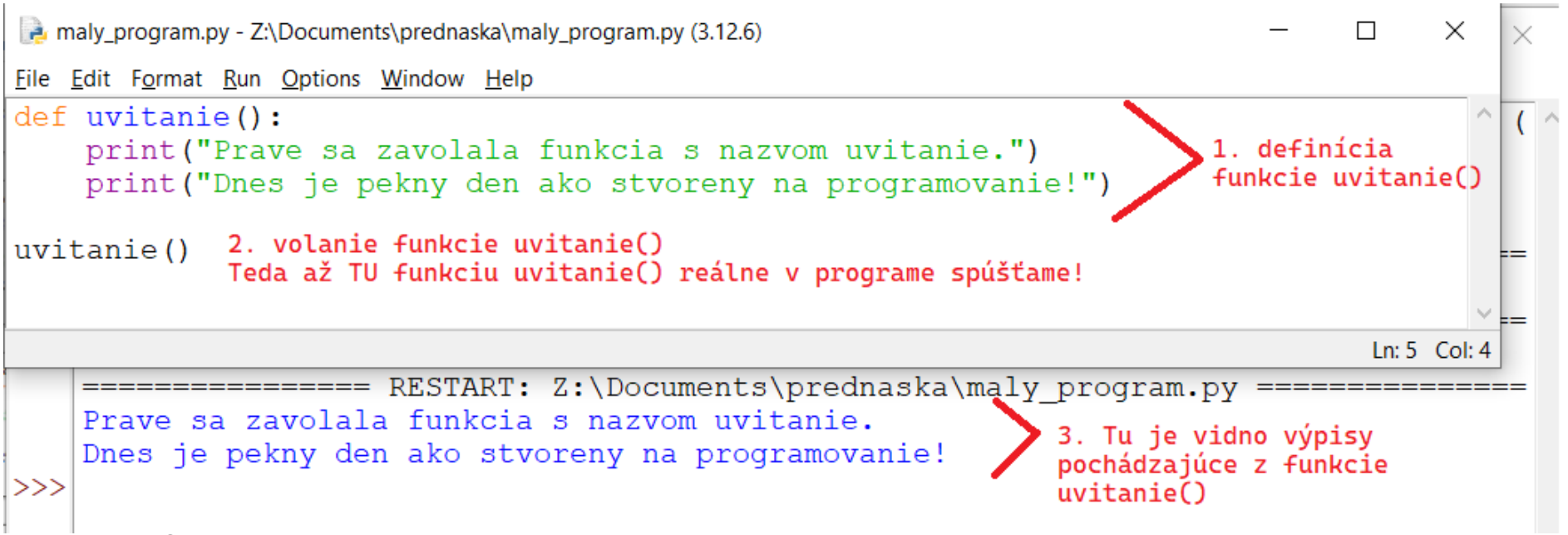


Navonok sa nestalo nič, nedošlo k výpisu uvedených správ.

Je to preto, že uvedený kód len **definuje funkciu uvitanie()**, t.j. obsahuje špecifikáciu, čo má funkcia robiť.

Avšak, aby sa funkcia vykonala, musíme ju **zavolať!**

Volanie funkcie

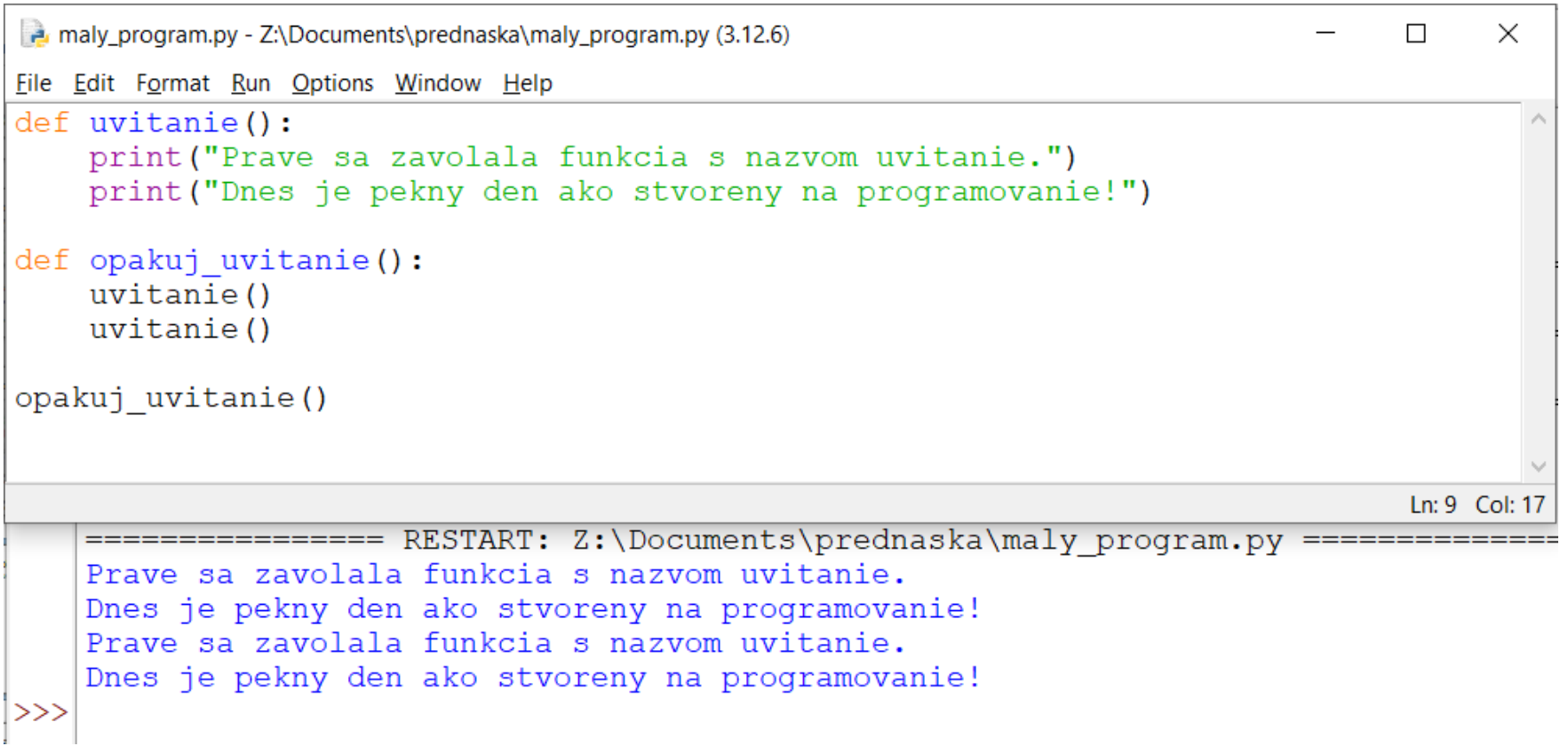


Volanie funkcie, t.j. jej spustenie realizujeme tak, že použijeme jej názov.
Ak ide o funkciu bez vstupných parametrov, stačí uviesť jej názov a prázdne zátvorky, t.j. *uvitanie*().

Všimnite si, že volanie funkcie už **nie je odsadené o 4 medzery!** Tým pádom volanie *uvitanie*() už **nie je súčasť tela funkcie** *uvitanie*().

Volanie funkcie

Po zadefinovaní novej funkcie ju môžeme použiť v tele inej funkcie!

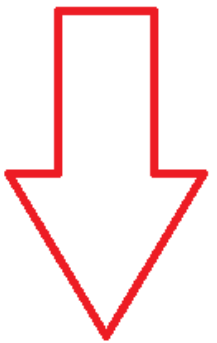


Ako Python spracuje definície a volanie funkcií?

maly_program.py - Z:\Documents\prednaska\maly_program.py (3.12.6)

File Edit Format Run Options Window Help

```
def uvitanie():
    print("Prave sa zavolala funkcia s nazvom uvitanie.")
    print("Dnes je pekny den ako stvoreny na programovanie!")

def opakuj_uvitanie():
    uvitanie()
    uvitanie()

opakuj_uvitanie()
```



Python číta a vykonáva zdrojový kód od prvého riadku po posledný

Ln: 9 Col: 17

Ako Python spracuje definície a volanie funkcií?

```
def uvitanie():
    print("Prave sa zavolala funkcia s nazvom uvitanie.")
    print("Dnes je pekny den ako stvoreny na programovanie!")

def opakuj_uvitanie():
    uvitanie()
    uvitanie()

opakuj_uvitanie()
```

1.

- 1) Ako prvé zistí, že je v kóde definovaná funkcia *uvitanie*() (to zistí podľa kľúčového slova „def“.
- 2) Následne Python vie, že každý príkaz, ktorý je odsadený od začiatku riadku, patrí do tela funkcie *uvitanie*()
- 3) Od tohto momentu teda Python vie, že v našom programe sme definovali funkciu *uvitanie*() a teda ak neskôr zistí, že sa táto funkcia niekde volá, bude ju považovať za známu a bude vedieť, aký kód sa má spustiť.

Ako Python spracuje definície a volanie funkcií?

```
def uvitanie():
    print("Prave sa zavolala funkcia s nazvom uvitanie.")
    print("Dnes je pekny den ako stvoreny na programovanie!")

def opakuj_uvitanie():
    uvitanie()
    uvitanie()

opakuj_uvitanie()
```

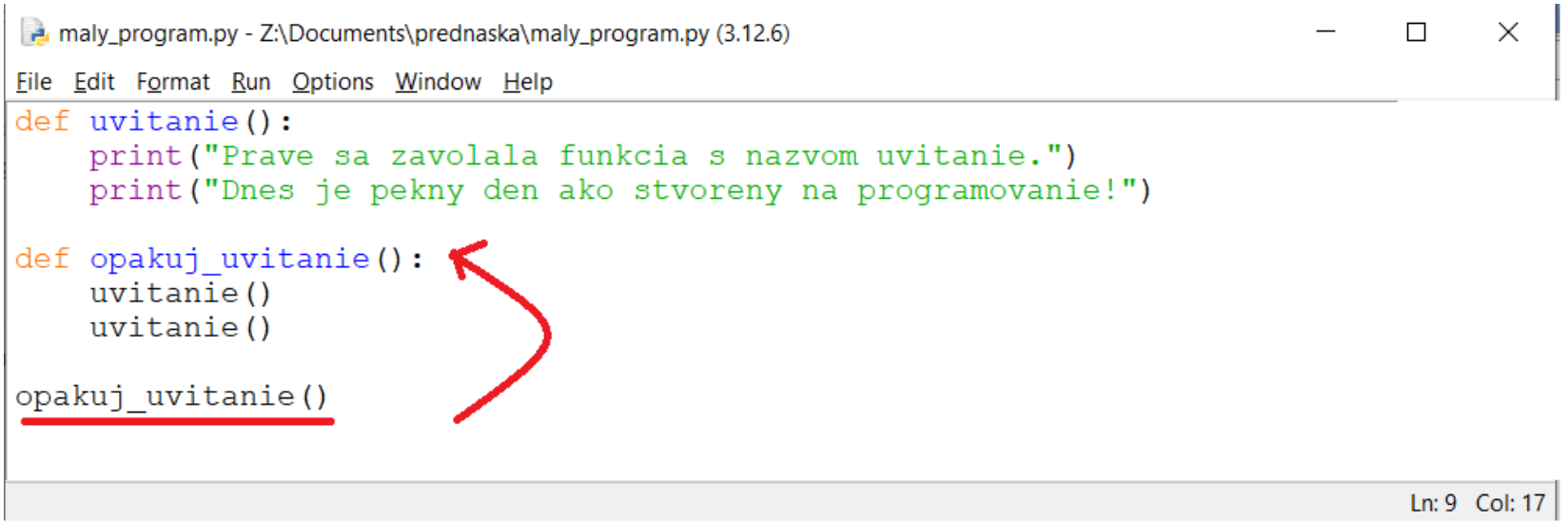
- 1) Ako druhé zistí, že je v kóde definovaná funkcia *opakuj_uvitanie*() (znovu to zistí podľa kľúčového slova „def“). Keďže slovo „def“ začína presne na začiatku riadku, Python zároveň vie, že táto definícia už **nie je súčasť definície** *uvitanie*()
- 2) Od tohto momentu teda Python vie, že v našom programe sme definovali funkciu *opakuj_uvitanie*() a teda ak neskôr zistí, že sa táto funkcia niekde volá, bude ju považovať za známu a bude vedieť, aký kód sa má spustiť.

Ako Python spracuje definície a volanie funkcií?

```
def uvitanie():
    print("Prave sa zavolala funkcia s nazvom uvitanie.")
    print("Dnes je pekny den ako stvoreny na programovanie!")

def opakuj_uvitanie():
    uvitanie()
    uvitanie()

opakuj_uvitanie()   3. volanie funkcie
                    "opakuj_volanie()"
```

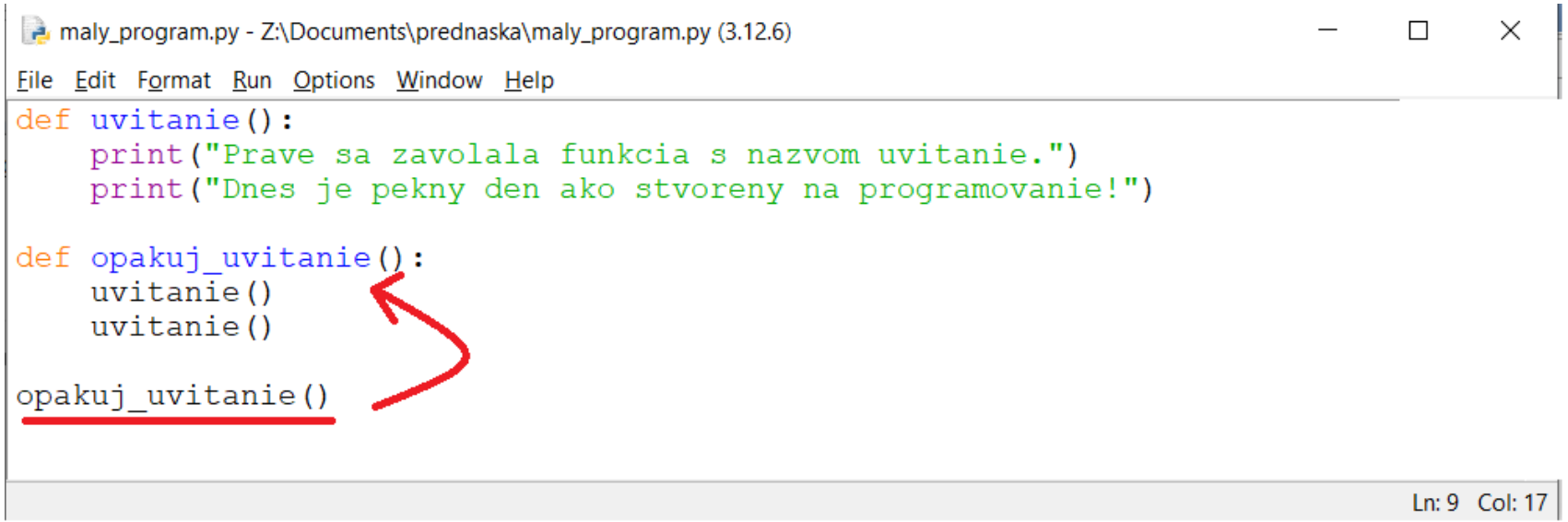
- Python teda postupne zistí, že v našom programe sú definované 2 nové funkcie, *uvitanie()* a *opakuj_uvitanie()*.
- Následne nájde (označené červenou farbou) **volanie** funkcie *opakuj_uvitanie*().
- V tomto momente teda spustí vykonávanie funkcie *opakuj_uvitanie()*.

Ako Python spracuje definície a volanie funkcií?



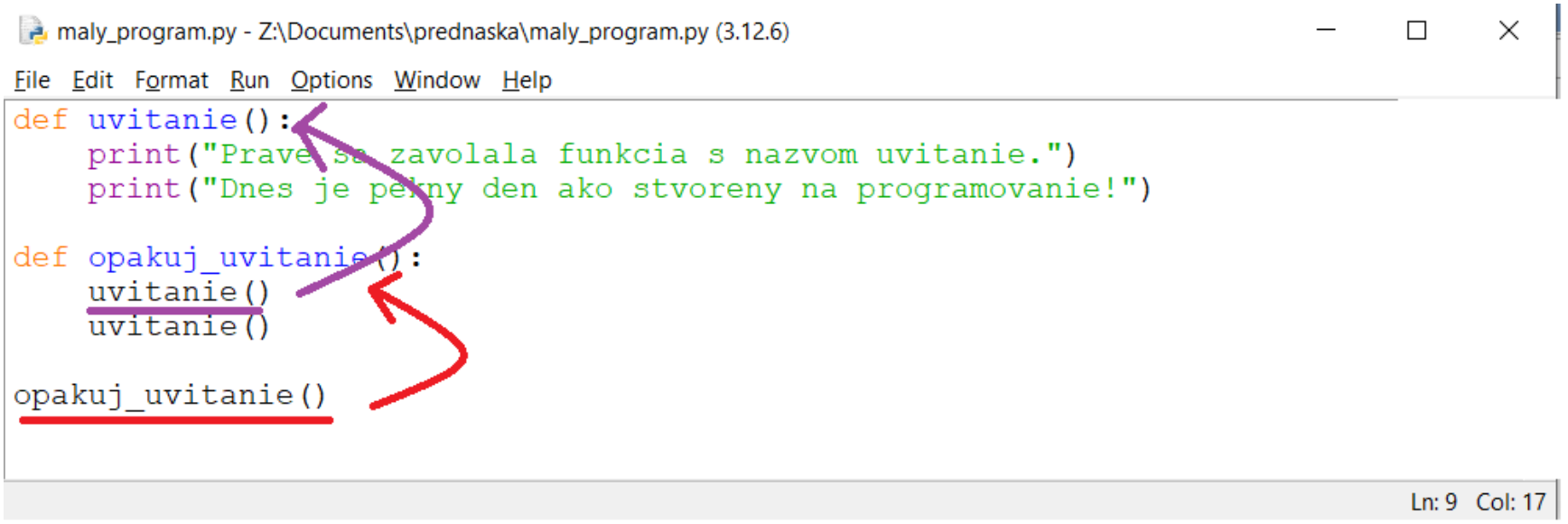
Pri volaní funkcie *opakuj_volanie()* Python „skočí“ na hlavičku príslušnej funkcie. Následne začne vykonávať jednotlivé príkazy v tele funkcie.

Ako Python spracuje definície a volanie funkcií?

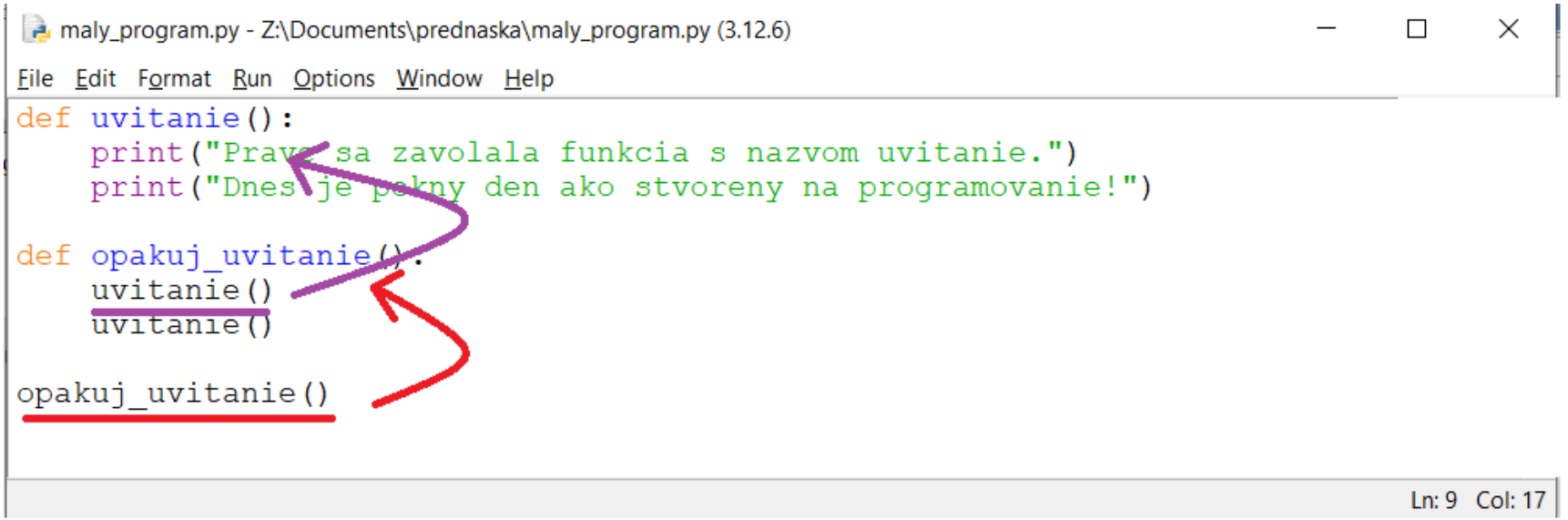


Keďže príkaz, ktorý sa nachádza v tele funkcie *opakuj_volanie()* je volaním funkcie *uvitanie()* následne Python „skočí“ na hlavičku funkcie *uvitanie()* a začne vykonávať telo funkcie *uvitanie().*

Ako Python spracuje definície a volanie funkcií?



Ako Python spracuje definície a volanie funkcií?



Výpis programu:

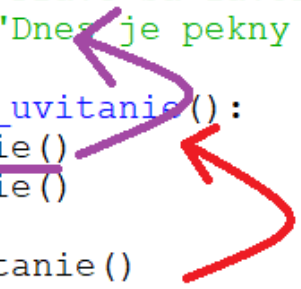
```
Prave sa zavolala funkcia s nazvom uvitanie.
```

Ako Python spracuje definície a volanie funkcií?

maly_program.py - Z:\Documents\prednaska\maly_program.py (3.12.6)

File Edit Format Run Options Window Help

```
def uvitanie():
    print("Prave sa zavolala funkcia s nazvom uvitanie.")
    print("Dnes je pekny den ako stvoreny na programovanie!")

def opakuj_uvitanie():
    uvitanie()
    uvitanie()

opakuj_uvitanie()
```



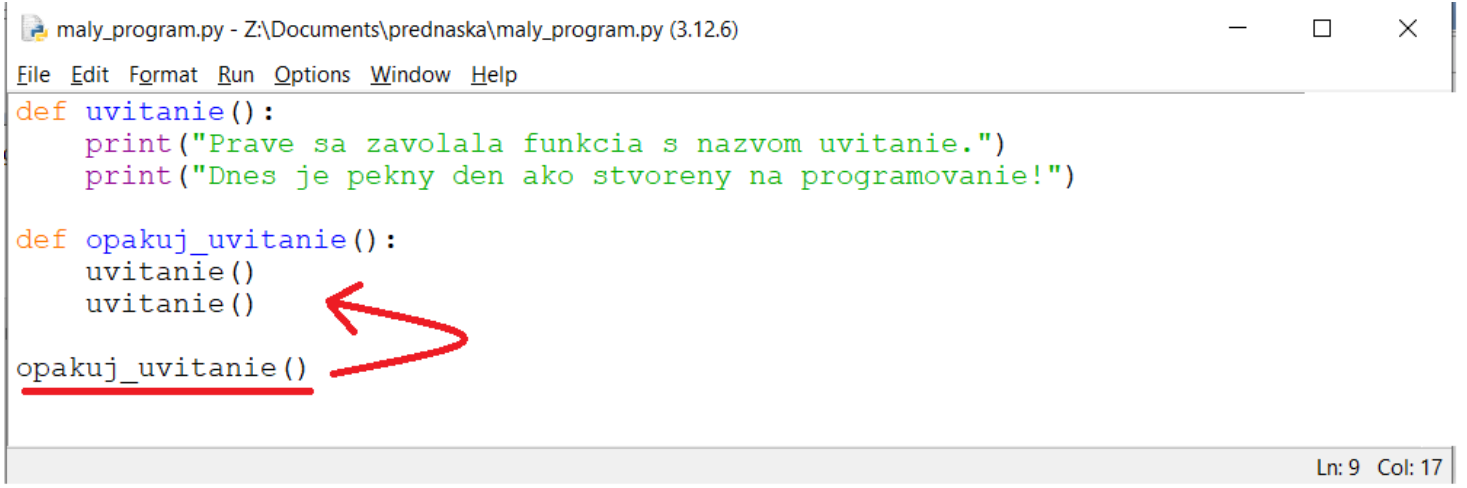
Ln: 9 Col: 17

Výpis programu:

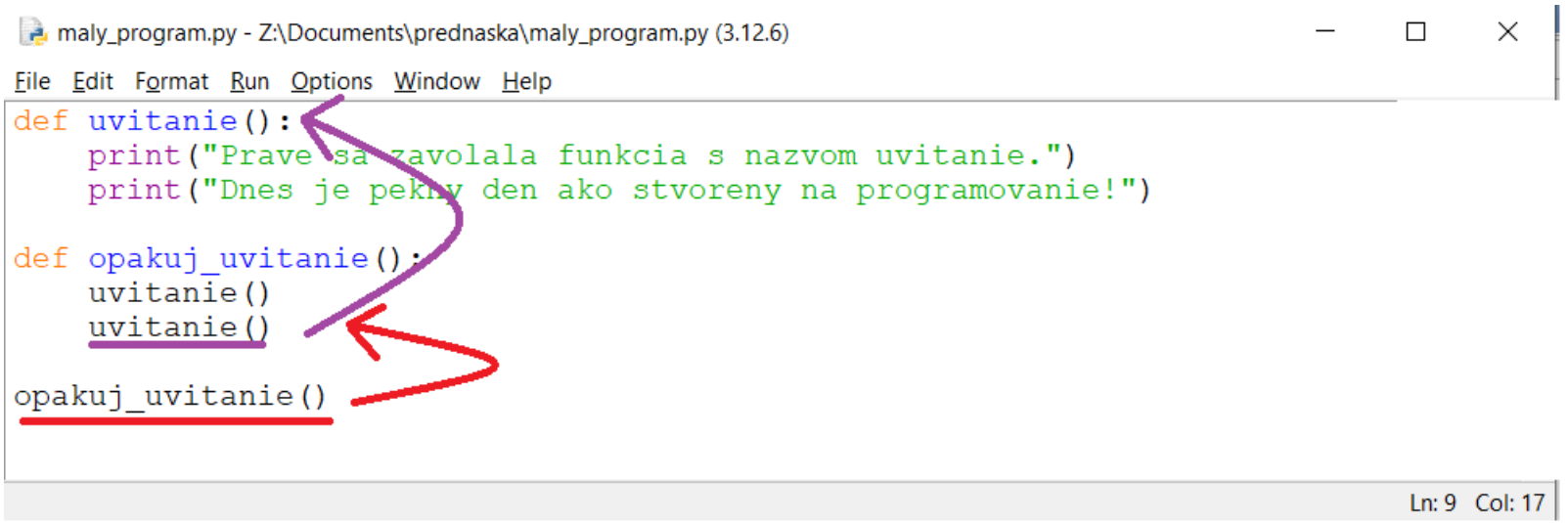
```
Prave sa zavolala funkcia s nazvom uvitanie.
Dnes je pekny den ako stvoreny na programovanie!
```

Ako Python spracuje definície a volanie funkcií?

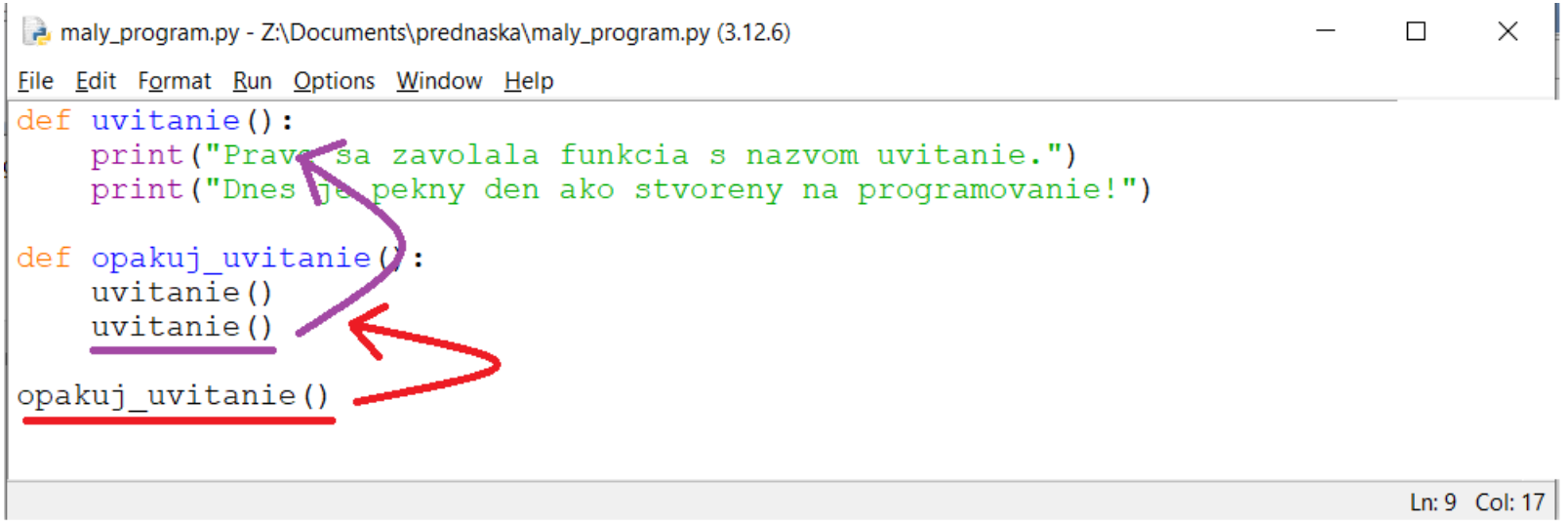
Keď sa vykonajú všetky príkazy v tele funkcie *uvitanie()*, príslušné volanie funkcie *uvitanie()* vo funkcii *opakuj_uvitanie()* sa ukončí a začne sa vykonávať ďalší príkaz vo funkcii *opakuj_uvitanie()*.



Ako Python spracuje definície a volanie funkcií?



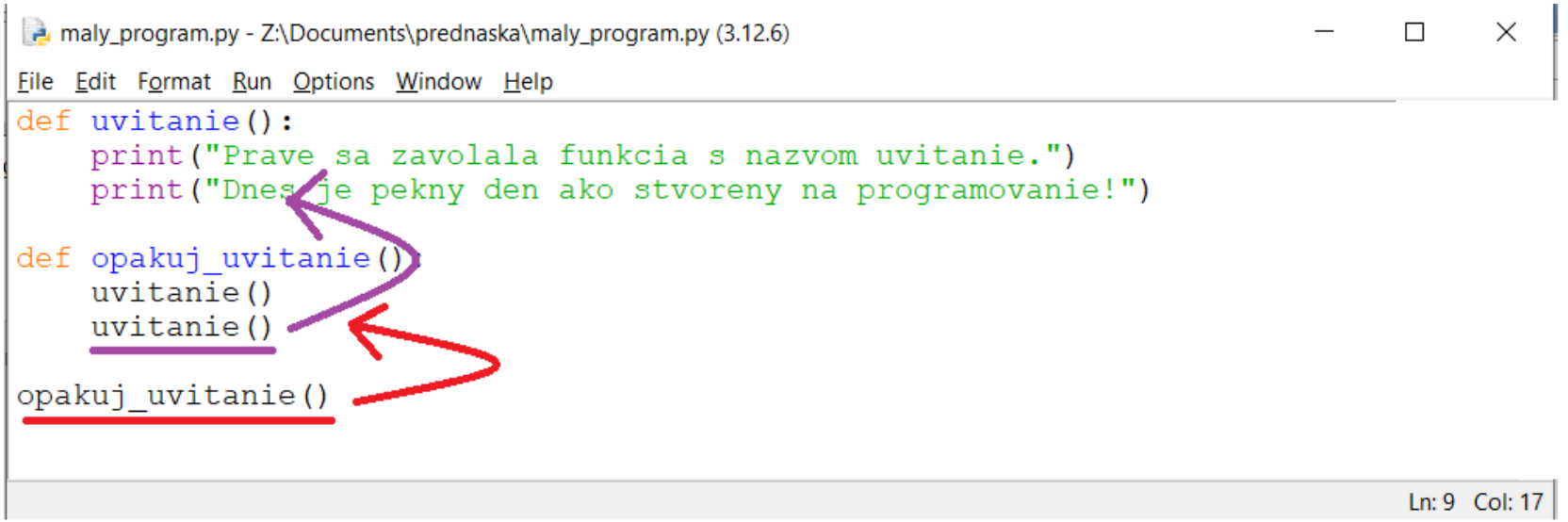
Ako Python spracuje definície a volanie funkcií?



Výpis programu:

```
Prave sa zavolala funkcia s nazvom uvitanie.
Dnes je pekny den ako stvoreny na programovanie!
Prave sa zavolala funkcia s nazvom uvitanie.
```

Ako Python spracuje definície a volanie funkcií?



Výpis programu:

```
Prave sa zavolala funkcia s nazvom uvitanie.
Dnes je pekny den ako stvoreny na programovanie!
Prave sa zavolala funkcia s nazvom uvitanie.
Dnes je pekny den ako stvoreny na programovanie!
```

Ako Python spracuje definície a volanie funkcií?

- Po ukončení druhého príkazu, t.j. volania *uvitanie()* volanie funkcie *opakuj_volanie()* končí.

- Tým pádom končí aj celý program a výsledný výpis na obrazovku je:

```
Prave sa zavolala funkcia s nazvom uvitanie.
Dnes je pekny den ako stvoreny na programovanie!
Prave sa zavolala funkcia s nazvom uvitanie.
Dnes je pekny den ako stvoreny na programovanie!
```

Definovanie funkcie so vstupnými parametrami

- V jazyku Python je možné definovať aj funkcie, ktoré obsahujú *vstupy*, teda majú tzv. ***vstupné parametre*.**

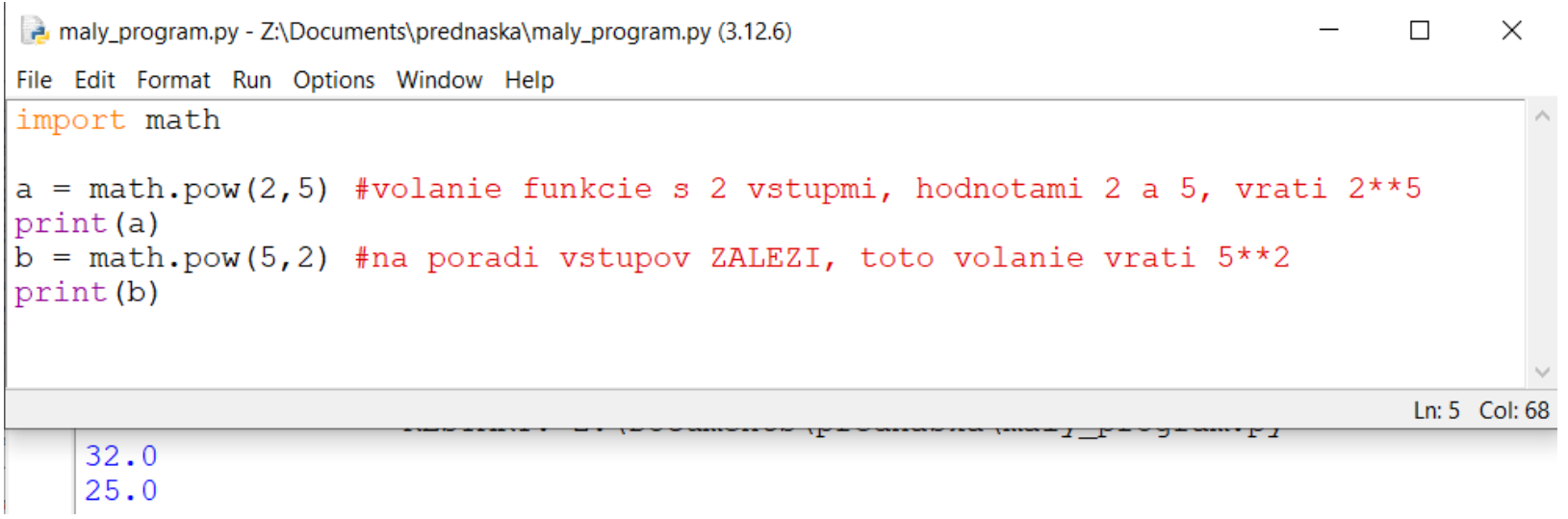
- Vstupné parametre funkcie predstavujú hodnoty, ktoré vstupujú do volania funkcie, s ktorými funkcia ďalej pracuje a spracúva ich.

- Príkladom funkcie so vstupmi je napríklad funkcia *abs()*, kde sme videli, že ak funkciu zavoláme so vstupom, napríklad *abs(-4)*, tak funkcia vráti absolútnu hodnotu vstupu, v tomto prípade 4.

- Funkcia print() zase pracuje tak, že svoj vstup vypíše na obrazovku, napr. *print(“ahoj“*) vypíše na obrazovku reťazec ahoj. V tomto volaní predstavuje “ahoj“ (spolu s úvodzovkami) vstup funkcie print().

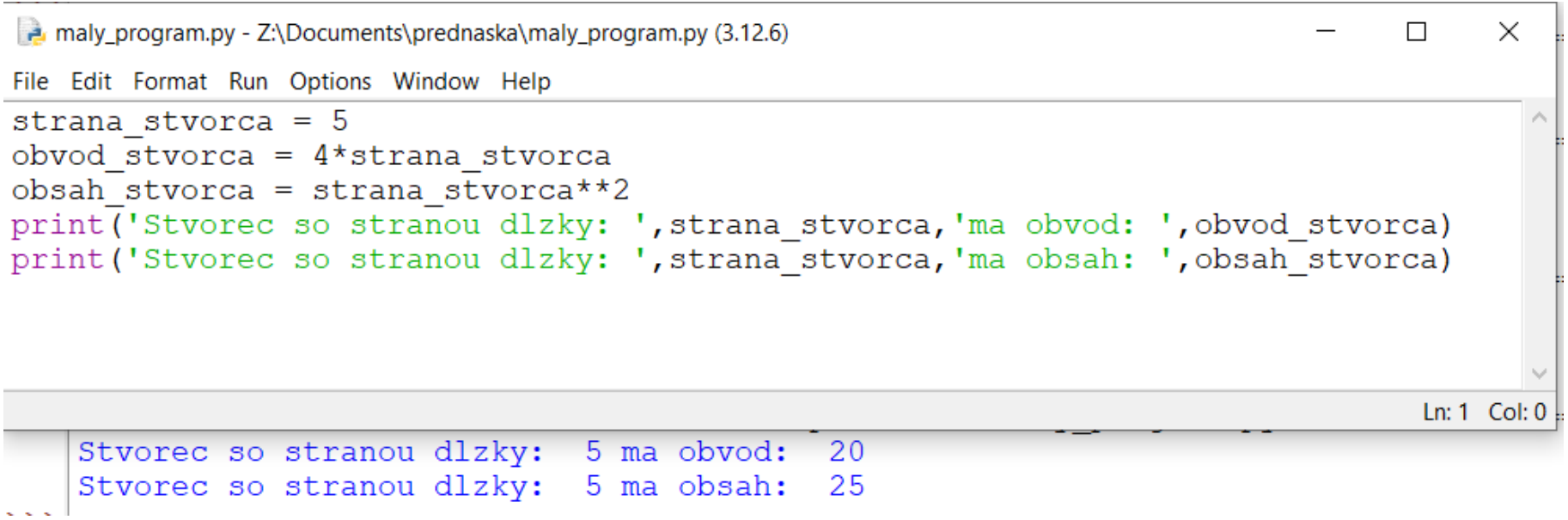
Definovanie funkcie so vstupnými parametrami

- Funkcie môžu mať aj viacero vstupných parametrov, napríklad funkcia *pow()* z modulu *math* môže mať 2 parametre, hodnoty *x* a *y*, pričom funkcia vráti hodnotu x^y.



Definovanie funkcie so vstupnými parametrami

- Aj funkcia *print()* môže mať viacero vstupov – v takom prípade ich postupne vypíše na obrazovku na 1 riadok. Všimnite si, ako to funguje, na príklade:



Definovanie funkcie so vstupnými parametrami

- Definícia funkcie **so vstupnými parametrami** sa líši od definície funkcie bez parametrov v tom, že pri definícii uvedieme, **koľko má funkcia parametrov a ako sa nazývajú** tak, že ich vymenujeme v hlavičke funkcie v zátvorke za identifikátorom:

def *meno_funkcie*(param1): #táto funkcia má 1 vstupný parameter s názvom *param1*
    prikaz
    prikaz
    ….
    prikaz

Telo funkcie

Definovanie funkcie so vstupnými parametrami

- Ak má funkcia mať viacero parametrov, uvedieme ich postupne a oddelíme ich čiarkami:

```
def meno_funkcie(param1, param2):      #táto funkcia má 2 vstupné parametre: param1 a param2
    prikaz
    prikaz
    ….
    prikaz
```

Telo funkcie

Definovanie funkcie so vstupnými parametrami

- **Vstupné parametre** funkcie zároveň fungujú ako **premenné** vo vnútri funkcie.
- Hodnoty týchto vstupných parametrov sa získajú pri **volaní funkcie** tak, že sa do nich odovzdajú tie **hodnoty**, ktoré sa použijú pri volaní funkcie.
- Hodnoty, ktoré použijeme pri **volaní funkcie** nazývame **vstupné argumenty pri volaní funkcie.**
- Pri volaní funkcie sa teda vezme hodnota **vstupných argumentov** a skopíruje sa do príslušných **vstupných parametrov**.

Definovanie funkcie so vstupnými parametrami

Ukážka funkcie, ktorá **vypíše obvod a obsah** obdĺžnika, pričom veľkosti strán obdĺžnika, *a* a *b*, predstavujú 2 vstupné parametre funkcie:

maly_program.py - Z:\Documents\prednaska\maly_program.py (3.12.6)

File Edit Format Run Options Window Help

```
def obvod_obsah_obdlznik(strana_a, strana_b):
    obvod_obdlznika = 2*strana_a + 2*strana_b
    obsah_obdlznika = strana_a*strana_b
    print('Strana a obdlznika: ',strana_a)
    print('Strana b obdlznika: ',strana_b)
    print('Obvod obdlznika: ', obvod_obdlznika)
    print('Obsah obdlznika: ', obsah_obdlznika)

obvod_obsah_obdlznik(5,13)
```

```
Strana a obdlznika:  5
Strana b obdlznika:  13
Obvod obdlznika:  36
Obsah obdlznika:  65
```

Ako vstupné argumenty pri volaní funkcie možno použiť ľubovoľné hodnoty – napr. aj iné premenné (v uvedenom príklade ako vstupné argumenty do volania funkcie *obvod_obsah_obdlznik(strana_a,strana_b)* používame premenné *a* a *b*:

maly_program.py - Z:\Documents\prednaska\maly_program.py (3.12.6)

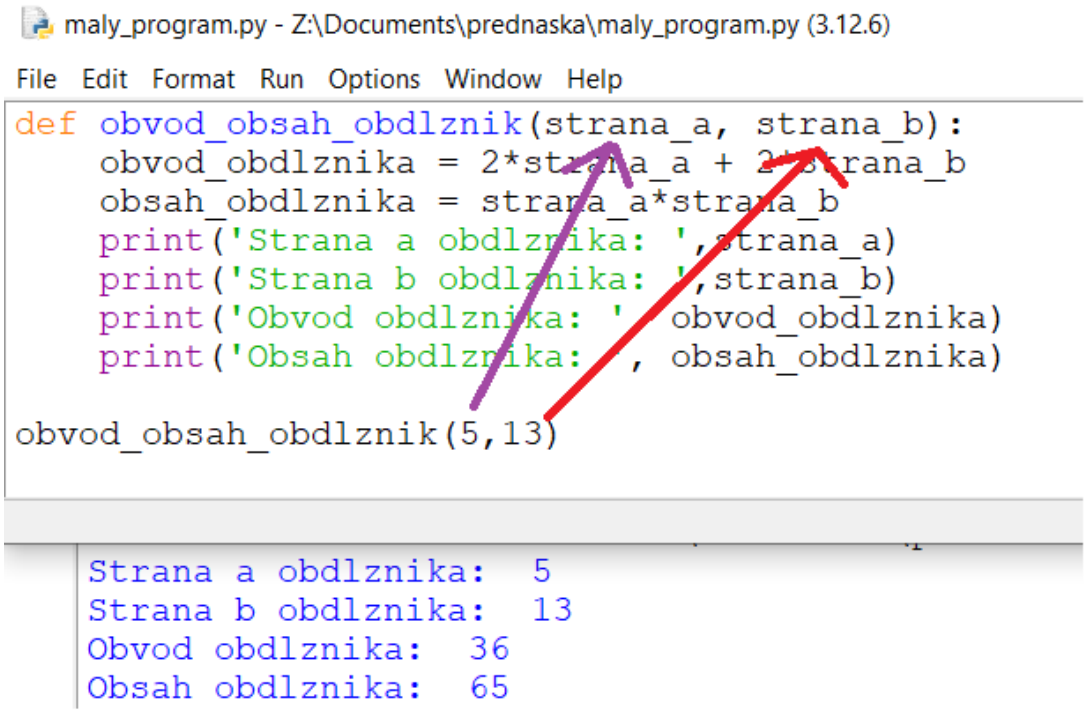
File Edit Format Run Options Window Help

```
def obvod_obsah_obdlznik(strana_a, strana_b):
    obvod_obdlznika = 2*strana_a + 2*strana_b
    obsah_obdlznika = strana_a*strana_b
    print('Strana a obdlznika: ',strana_a)
    print('Strana b obdlznika: ',strana_b)
    print('Obvod obdlznika: ', obvod_obdlznika)
    print('Obsah obdlznika: ', obsah_obdlznika)

a = 4
b = 7
obvod_obsah_obdlznik(a,b)
```

```
Strana a obdlznika:  4
Strana b obdlznika:  7
Obvod obdlznika:  22
Obsah obdlznika:  28
```

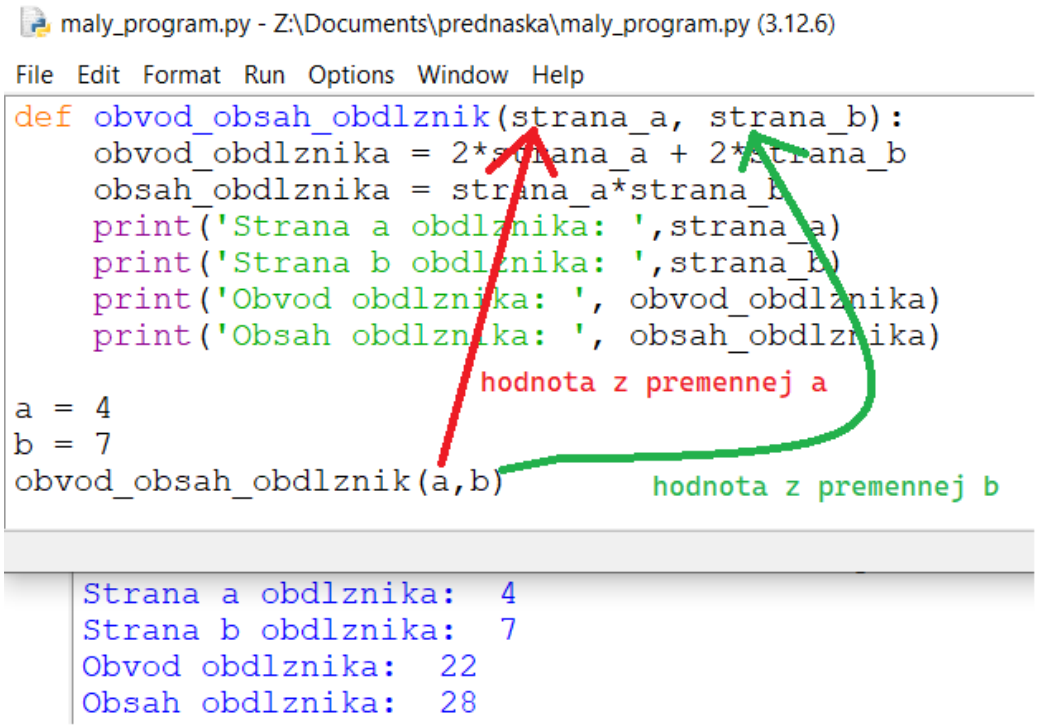
Ako funguje vzťah argument -> parameter

Pri volaní sa vezmú hodnoty argumentov a vložia sa do parametrov **v takom poradí, v akom sú zapísané pri volaní.**



Ako funguje vzťah argument -> parameter

Pri volaní sa vezmú hodnoty argumentov a vložia sa do parametrov **v takom poradí, v akom sú zapísané pri volaní.**



Ako funguje vzťah argument -> parameter

Teda, na poradí argumentov pri volaní funkcie **záleží!!!**

maly_program.py - Z:\Documents\prednaska\maly_program.py (3.12.6)

File Edit Format Run Options Window Help

```
def objem_valca(polomer_podstavy, vyska):
    pi = 3.14
    objem = pi*(polomer_podstavy**2)*vyska
    print('Polomer podstavy valca: ',polomer_podstavy)
    print('Vyska valca: ',vyska)
    print('Objem valca: ',objem)

r = 5
v = 7
objem_valca(r,v)
objem_valca(v,r)
```

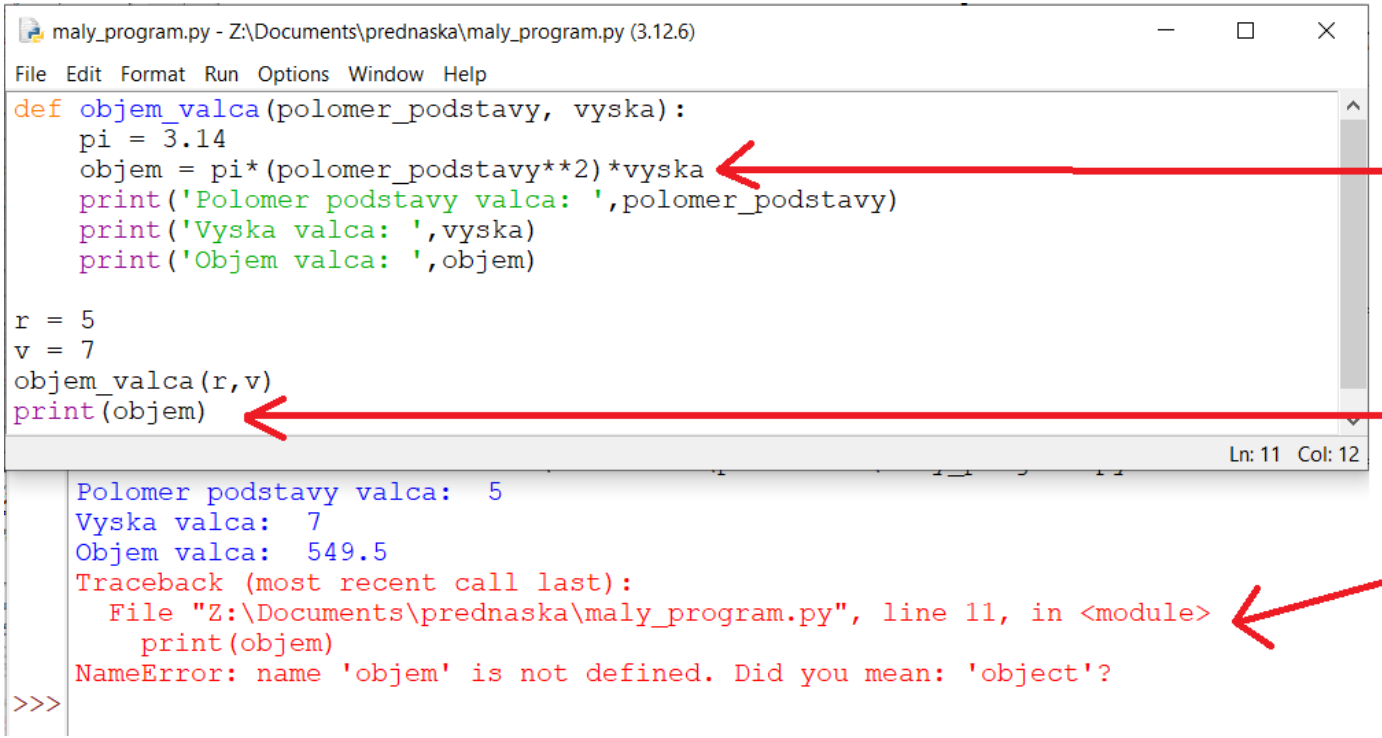
```
Polomer podstavy valca:  5
Vyska valca:  7
Objem valca:  549.5
Polomer podstavy valca:  7
Vyska valca:  5
Objem valca:  769.3000000000001
```

Lokálne premenné

- Každá premenná, ktorá je vytvorená vo vnútri tela nejakej funkcie, existuje **len počas vykonávania danej funkcie** a je taktiež prístupná **len v danej funkcii.**
- Rovnako **vstupné parametre** funkcie predstavujú **lokálne premenné.**
- Preto, ak sa pokúsime použiť lokálnu premennú (vstupný parameter) mimo príslušnú funkciu, Python vypíše chybu!

Lokálne premenné

Chybový výpis o neexistencii lokálnej premennej *objem* pri pokuse o výpis jej hodnoty mimo telo funkcie, kde bola premenná vytvorená.



Tu vytvorená lokálna premenná "objem", preto táto premenná neexistuje mimo telo funkcie "objem_valca"

Preto tento pokus o vypísanie hodnoty premennej "objem" na obrazovku skončí neúspechom a Python vypíše chybovú hlášku, že nepozná premennú "objem"

Lokálne premenné

Analogická situácia nastane pri pokuse o prístup k premennej (vstupnému parametru) *polomer_podstavy*:

maly_program.py - Z:\Documents\prednaska\maly_program.py (3.12.6)

File Edit Format Run Options Window Help

```
def objem_valca(polomer_podstavy, vyska):
    pi = 3.14
    objem = pi*(polomer_podstavy**2)*vyska
    print('Polomer podstavy valca: ',polomer_podstavy)
    print('Vyska valca: ',vyska)
    print('Objem valca: ',objem)

r = 5
v = 7
objem_valca(r,v)
print(polomer_podstavy)
```

Ln: 11 Col: 22

```
Polomer podstavy valca:  5
Vyska valca:  7
Objem valca:  549.5
Traceback (most recent call last):
  File "Z:\Documents\prednaska\maly_program.py", line 11, in <module>
    print(polomer_podstavy)
NameError: name 'polomer_podstavy' is not defined
```

Návratová hodnota funkcie

- Okrem toho, že funkcia môže mať vstupné parametre, teda to, čo vykonáva, závisí od vstupných hodnôt, môže funkcia na záver svojej činnosti aj **vrátiť** nejakú **návratovú hodnotu**.

- Štandardne sa to používa pri takých funkciách, ktoré realizujú nejaký výpočet, ktorého výsledok chceme použiť v programe na ďalšie spracovanie / účel.

- Napríklad sme sa stretli s funkciou *sqrt()* z modulu *math*, ktorá vráti ako svoju návratovú hodnotu odmocninu zo svojho vstupu.

- Ak chceme, aby aj nami definovaná funkcia vrátila nejakú hodnotu, použijeme na to v tele funkcie príkaz

  return *hodnota*

  kde *hodnota* je hodnota, ktorú chceme vrátiť ako návratovú hodnotu. Ako neskôr uvidíme, návratovou hodnotou môže byť v jazyku Python nielen číslo či reťazec, ale aj iné typy hodnôt...

Návratová hodnota funkcie

- V nasledovnom príklade upravíme program pre výpočet objemu valca tak, že si vytvoríme novú funkciu *obsah_podstavy(polomer_podstavy)*, ktorá počíta obsah podstavy:
  - Vstupom funkcie bude polomer podstavy
  - Výstupom funkcie (návratovou hodnotou) bude obsah podstavy

- Následne túto funkciu využijeme vo funkcii pre výpočet objemu valca a upravíme funkciu pre výpočet objemu valca tak, aby výsledný objem vrátila ako svoju návratovú hodnotu.

Návratová hodnota funkcie

maly_program.py - Z:\Documents\prednaska\maly_program.py (3.12.6)

File Edit Format Run Options Window Help

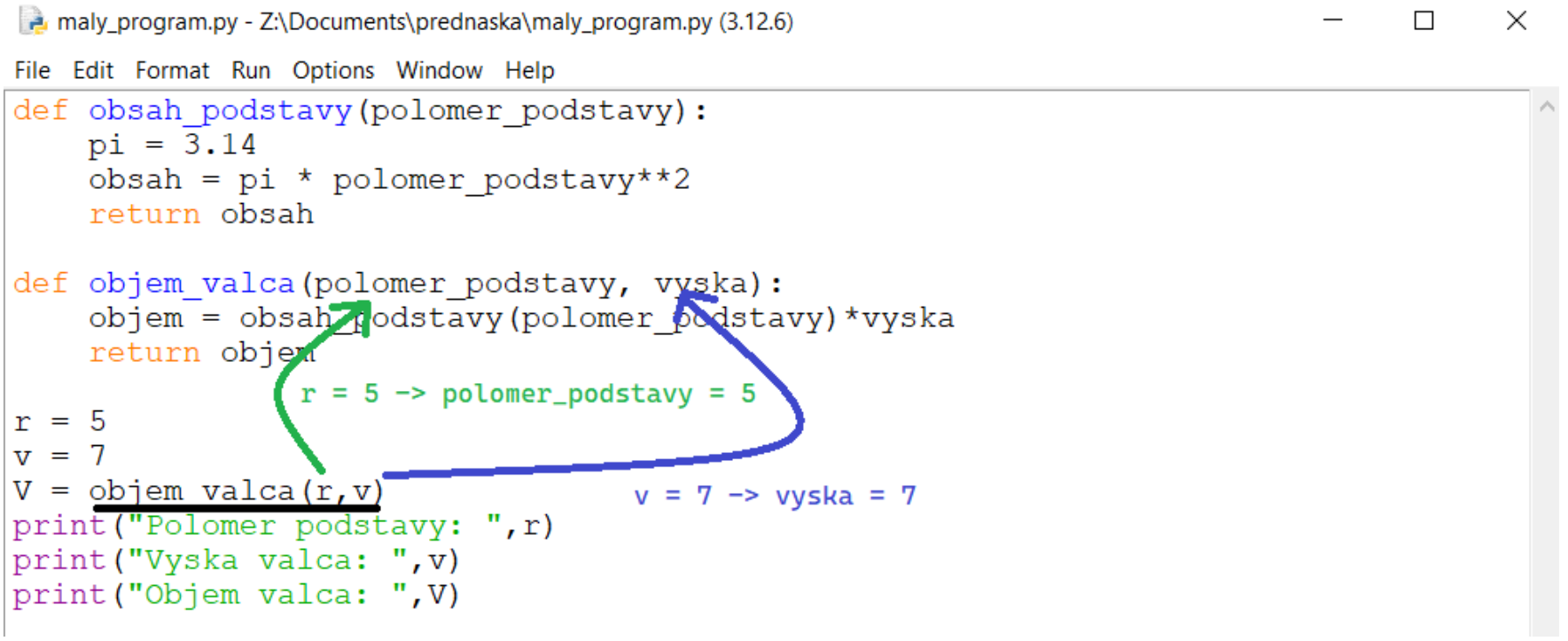
```
def obsah_podstavy(polomer_podstavy):
    pi = 3.14
    obsah = pi * polomer_podstavy**2
    return obsah

def objem_valca(polomer_podstavy, vyska):
    objem = obsah_podstavy(polomer_podstavy)*vyska
    return objem

r = 5
v = 7
V = objem_valca(r,v)
print("Polomer podstavy: ",r)
print("Vyska valca: ",v)
print("Objem valca: ",V)
```

Ln: 16 Col: 0

```
Polomer podstavy:  5
Vyska valca:  7
Objem valca:  549.5
```

Návratová hodnota funkcie

Pri volaní funkcie *objem_valca* sa hodnota argumentu *r* (teda číslo 5) vloží do vstupného parametra *polomer_podstavy*. Podobne sa hodnota argumentu *v* (číslo 7) vloží do vstupného parametra *vyska*.



Návratová hodnota funkcie

Pozor! Obe funkcie obsah_podstavy / objem_valca obsahujú vstupný parameter s názvom *polomer_podstavy*. Formálne však ide o 2 rôzne premenné (obe sú to lokálne premenné v príslušných funkciách).

maly_program.py - Z:\Documents\prednaska\maly_program.py (3.12.6)

File Edit Format Run Options Window Help

```
def obsah_podstavy(polomer_podstavy):
    pi = 3.14
    obsah = pi * polomer_podstavy**2
    return obsah

def objem_valca(polomer_podstavy, vyska):
    objem = obsah_podstavy(polomer_podstavy)*vyska
    return objem

r = 5
v = 7
V = objem_valca(r,v)
print("Polomer podstavy: ",r)
print("Vyska valca: ",v)
print("Objem valca: ",V)
```

```
polomer_podstavy(parameter funkcie objem_valca) = 5 ->
polomer_podstavy (parameter_funkcie obsah_podstavy) = 5
```

Návratová hodnota funkcie

Vo volaní funkcie *obsah_podstavy* (predošlý slajd) bola hodnota vstupného argumentu 5, preto aj lokálna premenná *polomer_podstavy* má vo volaní funkcie *obsah_podstavy* hodnotu 5. Do premennej *obsah* sa teda vloží hodnota 78.5

maly_program.py - Z:\Documents\prednaska\maly_program.py (3.12.6)

File Edit Format Run Options Window Help

```
def obsah_podstavy(polomer_podstavy):
    pi = 3.14
    obsah = pi * polomer_podstavy**2
    return obsah

def objem_valca(polomer_podstavy, vyska):
    objem = obsah_podstavy(polomer_podstavy)*vyska
    return objem

r = 5
v = 7
V = objem_valca(r,v)
print("Polomer podstavy: ",r)
print("Vyska valca: ",v)
print("Objem valca: ",V)
```

Do premennej obsah sa uloží hodnota 3.14 * 5**2 = 78.5

Návratová hodnota funkcie

Vo volaní funkcie *obsah_podstavy* ku ktorému došlo vo funkcii *objem_valca*, nastal príkaz **return** s hodnotou 78.5 (premenná *obsah*). Preto na mieste kde bola volaná funkcia *obsah_podstavy* dôjde k odovzdaniu hodnoty 78.5 do príslušného výrazu!

maly_program.py - Z:\Documents\prednaska\maly_program.py (3.12.6)

File Edit Format Run Options Window Help

```
def obsah_podstavy(polomer_podstavy):
    pi = 3.14
    obsah = pi * polomer_podstavy**2
    return obsah

def objem_valca(polomer_podstavy, vyska):
    objem = obsah_podstavy(polomer_podstavy)*vyska
    return objem

r = 5
v = 7
V = objem_valca(r,v)
print("Polomer podstavy: ",r)
print("Vyska valca: ",v)
print("Objem valca: ",V)
```

volanie obsah_podstavy so vstupným argumentom 5 vráti návratovú hodnotu 78.5 (hodnotu z premennej "obsah")

Návratová hodnota funkcie

maly_program.py - Z:\Documents\prednaska\maly_program.py (3.12.6)

File Edit Format Run Options Window Help

```
def obsah_podstavy(polomer_podstavy):
    pi = 3.14
    obsah = pi * polomer_podstavy**2
    return obsah

def objem_valca(polomer_podstavy, vyska):
    objem = obsah_podstavy(polomer_podstavy)*vyska
    return objem

r = 5
v = 7
V = objem_valca(r,v)
print("Polomer podstavy: ",r)
print("Vyska valca: ",v)
print("Objem valca: ",V)
```

Do premennej"objem"sa vloží hodnota 78.5 * 7 = 549.5, pretože 78.5 je návratová hodnota z volania "obsah_podstavy(5)" a 7 je hodnota parametra "vyska"

Návratová hodnota funkcie

maly_program.py - Z:\Documents\prednaska\maly_program.py (3.12.6)

File Edit Format Run Options Window Help

```
def obsah_podstavy(polomer_podstavy):
    pi = 3.14
    obsah = pi * polomer_podstavy**2
    return obsah

def objem_valca(polomer_podstavy, vyska):
    objem = obsah_podstavy(polomer_podstavy)*vyska
    return objem

r = 5
v = 7
V = objem_valca(r,v)
print("Polomer podstavy: ",r)
print("Vyska valca: ",v)
print("Objem valca: ",V)
```

Funkcia "objem_valca" príkazom return vráti hodnotu premennej "objem", teda aktuálne 549.5. Dôjde teda k odovzdaniu hodnoty 549.5 do výrazu, kde bola volaná funkcia "objem_valca", teda sa hodnota 549.5 priradí do premennej "V".

Čo vracajú funkcie, ktoré nič nevracajú?

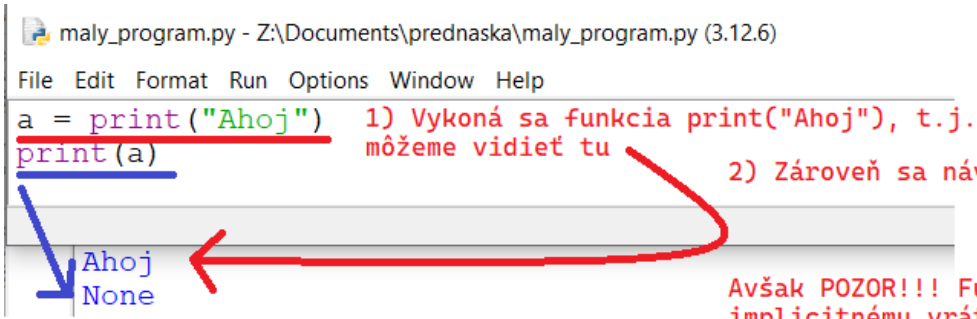
- Ak teda počas vykonávania funkcie dôjde k príkazu **return**, funkcia skončí svoju činnosť a vráti príslušnú hodnotu, uvedenú pri príkaze **return.**

- Čo sa však v jazyku Python udeje v situácii, že funkcia nič nevracia??? Teda, že počas behu funkcie **nedôjde** k použitiu príkazu **return?**

- Takéto funkcie **v jazyku Python** nazývame ako tzv. **void** funkcie (angl. *void* = prázdny, prázdnota, prázdne miesto).

- Hoci tieto funkcie **explicitne** nevracajú žiadnu hodnotu (t.j. nedôjde počas ich behu k príkazu **return**), **implicitne** vracajú špeciálnu hodnotu, tzv. *None* hodnotu.
- *None* je špeciálna hodnota v jazyku Python, špeciálneho dátového typu *NoneType*. Ide o špecialitu jazyku Python, iné programovacie jazyky túto hodnotu nemusia používať.

Ukážka vrátenia hodnoty *None*



1) Vykoná sa funkcia print("Ahoj"), t.j. dôjde k výpisu reťazca Ahoj na obrazovku, čo môžeme vidieť tu

2) Zároveň sa návratová hodnota volania print("Ahoj") uloží do premennej "a".

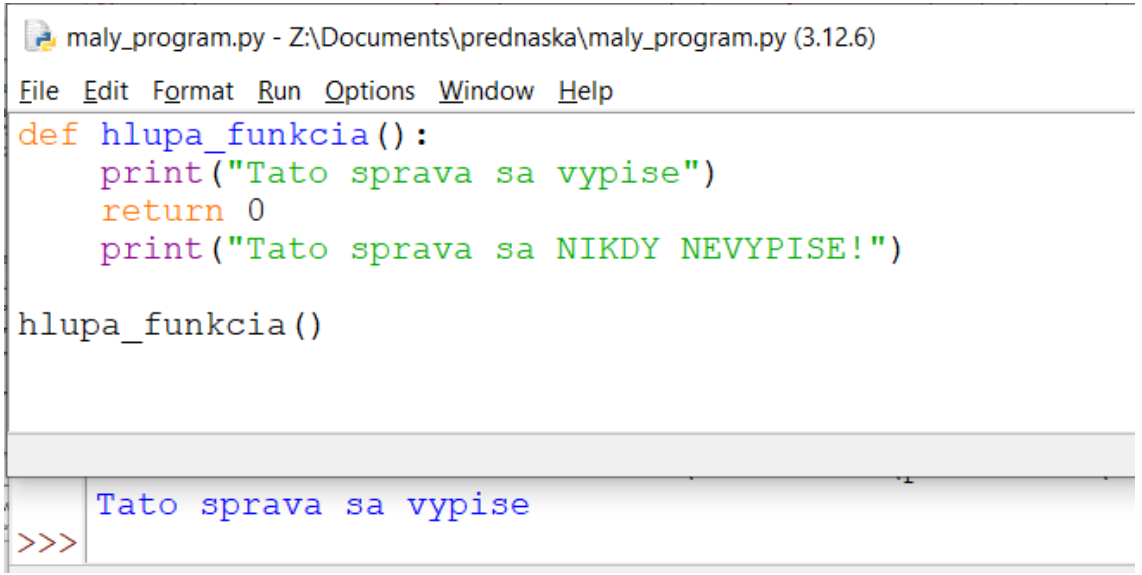
Avšak POZOR!!! Funkcia print() v jazyku Python oficiálne nič nevracia! Teda dôjde k implicitnému vráteniu hodnoty None, čiže do premennej "a" sa vloží hodnota None

3) Preto vidíme, že výpis hodnoty v premennej "a" na obrazovku pomocou print(a) vypísal na obrazovku hodnotu None.

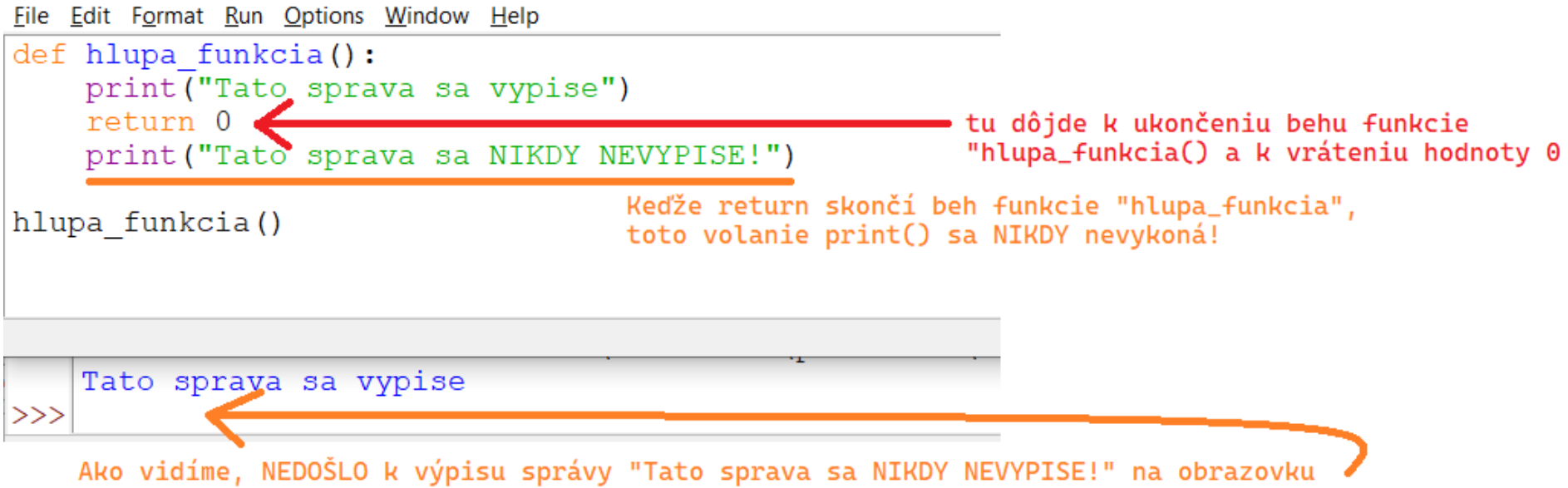
Veľmi dôležité upozornenie

- Príkaz **return** spôsobí, že funkcia, v ktorej bol príkaz použitý, vráti požadovanú návratovú hodnotu a zároveň **skončí svoju činnosť!**

- To znamená, že v momente vrátenia návratovej hodnoty funkcia končí a ak sa v jej kóde nachádzali ešte nejaké ďalšie príkazy, tak tie sa **nevykonajú!**



Veľmi dôležité upozornenie



Načo funkcie???

Použitie funkcií má v programovaní niekoľko významov:

- Rozdelenie programu do funkcií zlepšuje čitateľnosť a prehľadnosť kódu
- Funkcie dokážu eliminovať opakovanie kódu – ak by som napríklad na viacerých miestach v kóde potreboval realizovať rovnaký výpočet, je lepšie si preň vytvoriť samostatnú funkciu a tú potom podľa potreby volať.
- Rozdelenie programu na jednotlivé funkcie uľahčuje hľadanie chýb a ladenie kódu
- Rozdelenie programu na funkcie uľahčuje spoluprácu medzi programátormi – napríklad ak pracujem na programe, v ktorom potrebujem najprv utriediť dáta, iný programátor môže napísať funkciu, ktorá dáta utriedi a ja sa môžem sústrediť na zvyšok programu a jeho funkciu len zavolám, aby mi dáta utriedila.

Jednoduché for-cykly

Jednoduchý for-cyklus (for-loop)

- V mnohých programovacích jazykoch (vrátane Pythonu) existujú príkazy, ktoré Vám umožnia zapísať opakovanie nejakej postupnosti príkazov.
- Lepšie povedané, v programovaní / algoritmizácii existuje koncept **opakovania postupnosti príkazov**.
- Príkazy, pomocou ktorých vieme počítaču prikázať, aby nejakú postupnosť príkazov zopakoval, nazývame **cykly** alebo aj **iterácie.**
- Príkaz *for* je základným typom iterácií – ide o tzv. **iteráciu s vopred známym počtom opakovaní**.

Iterácia s vopred známym počtom opakovaní

- V jazyku Python je predstaviteľom iterácií s vopred známym počtom opakovaní tzv. *for-cyklus* (angl. *for-loop*).
- Pozrime sa na nasledovný príklad

Iterácia s vopred známym počtom opakovaní

Program:

maly_program.py - Z:\Documents\prednaska\maly_program.py (3.12.6)

File Edit Format Run Options Window Help

```
for i in range(4):
    print("Ahojte")
    print("studentky a studenti")

for i in range(2):
    print("Dovidenia")
```

Výstup:

```
Ahojte
studentky a studenti
Ahojte
studentky a studenti
Ahojte
studentky a studenti
Ahojte
studentky a studenti
Dovidenia
Dovidenia
```

Iterácia s vopred známym počtom opakovaní

Program:

maly_program.py - Z:\Documents\prednaska\maly_program.py (3.12.6)

File Edit Format Run Options Window Help

```
for i in range(4):
    print("Ahojte")
    print("studentky a studenti")

for i in range(2):
    print("Dovidenia")
```

Sa teda správa rovnako ako:

maly_program.py - Z:\Documents\prednaska\maly_program.py (3.12.6)

File Edit Format Run Options Window Help

```
print("Ahojte")
print("studentky a studenti")
print("Ahojte")
print("studentky a studenti")
print("Ahojte")
print("studentky a studenti")
print("Ahojte")
print("studentky a studenti")

print("Dovidenia")
print("Dovidenia")
```

Iterácia s vopred známym počtom opakovaní

maly_program.py - Z:\Documents\prednaska\maly_program.py (3.12.6)

File Edit Format Run Options Window Help

```
print("Ahojte")
print("studentky a studenti")
print("Ahojte")
print("studentky a studenti")
print("Ahojte")
print("studentky a studenti")
print("Ahojte")
print("studentky a studenti")

print("Dovidenia")
print("Dovidenia")
```

4-krat zopakovane:
print("Ahojte")
print("studentky a studenti")

2-krat zopakovane:
print("Dovidenia")

<--- Verzia vľavo má všetky príkazy vypísané postupne

Verzia vpravo --->
Používa zápis pomocou príkazu for, kde je uvedené, koľkokrát sa majú príslušné príkazy vykonať! Všimnite si, že počet opakovaní je možné vidieť v zátvorke za volaním funkcie *range*()

maly_program.py - Z:\Documents\prednaska\maly_program.py (3.12.6)

File Edit Format Run Options Window Help

```
for i in range(4):
    print("Ahojte")
    print("studentky a studenti")

for i in range(2):
    print("Dovidenia")
```

Iterácia s vopred známym počtom opakovaní

Ak teda chceme v jazyku Python zapísať, aby sa nejaká postupnosť príkazov vykonala opakovane, pričom **vieme špecifikovať, koľkokrát sa má daná postupnosť opakovať – napríklad nejakou konštantou, alebo aj hodnotou nejakej premennej**, môžeme použiť for-cyklus, ktorého syntax v jazyku Python nasledovná:

for *premenná* **in range***(počet_opakovaní)*:

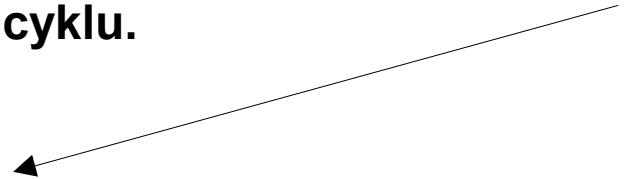
príkaz
príkaz
…
príkaz

Tieto príkazy tvoria tzv. **telo cyklu**. Ide o príkazy, ktoré sa budú iterovane (opakovane) vykonávať.

Podobne ako pri definícii funkcie, do **tela cyklu** patria len tie príkazy, ktoré sú **odsadené od začiatku riadku**, štandardne o 4 medzery.

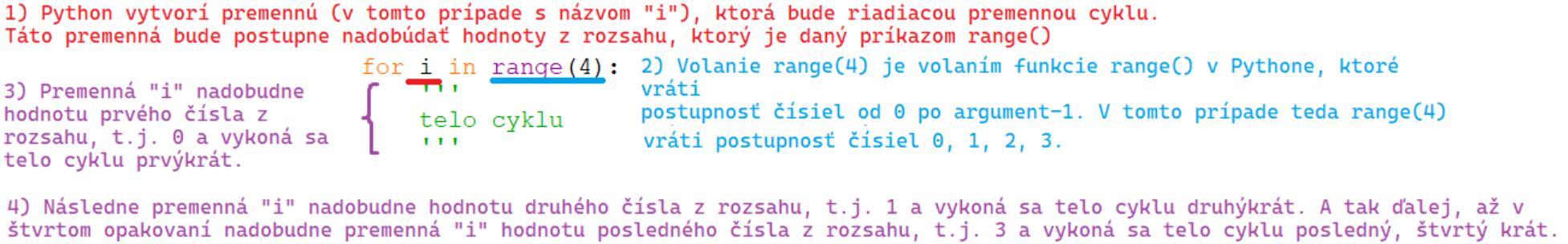
Iterácia s vopred známym počtom opakovaní

Premenná, ktorá je uvedená na tomto mieste, sa nazýva aj **riadiaca premenná cyklu.**



for *premenná* **in range**(*počet_opakovaní)*:
    príkaz
    príkaz
    …
    príkaz

Ako to funguje?



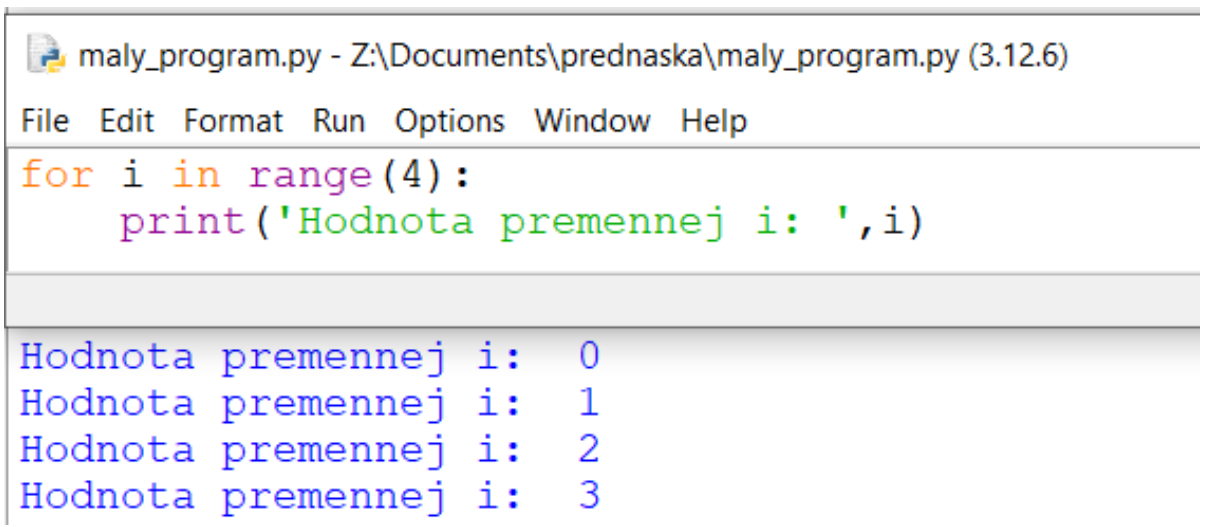
1) Funkcia **range**(*hodnota*) teda v jazyku Python vráti postupnosť čísiel, vždy od 0 po *hodnota-1*. Teda napríklad range(4) vráti postupnosť čísiel 0,1,2,3; volanie range(10) by vrátilo postupnosť čísiel 0,1,2,3,4,5,6,7,8,9 atď. Všimnite si, že to v tomto prípade **vždy začína nulou** a že dokopy je tých hodnôt **práve toľko**, aká *hodnota* bola vložená ako argument do funkcie **range**.

2) Riadiaca premenná cyklu teda **postupne** nadobúda hodnoty z tejto postupnosti. Najprv nadobudne prvú hodnotu a vykoná sa celé telo cyklu prvýkrát. Po spustení príkazov tela cyklu sa hodnota *i* zmení na ďalšiu hodnotu z postupnosti a telo cyklu sa vykoná druhýkrát, a tak ďalej, kým sa nevyčerpajú hodnoty z postupnosti.

Ako to funguje?

Všimnite si, ako sa správa nasledovný kód aj s výpisom:

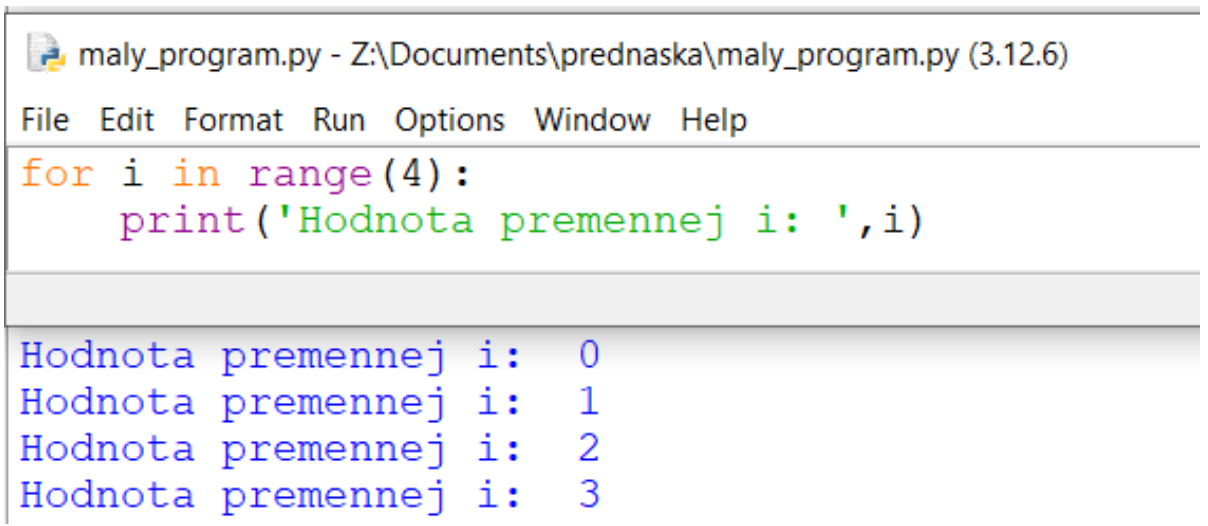
1) range(4) vráti postupnosť čísiel 0,1,2,3
2) Python vytvorí riadiacu premennú cyklu s názvom *i* a vloží do nej **prvú hodnotu** z postupnosti, t.j. nastaví *i* = 0
3) Vykoná sa telo cyklu, t.j. príkaz, ktorý vypíše na obrazovku hodnotu premennej *i*, teda sa vypíše, že premenná *i* má hodnotu 0.



Ako to funguje?

Všimnite si, ako sa správa nasledovný kód aj s výpisom:

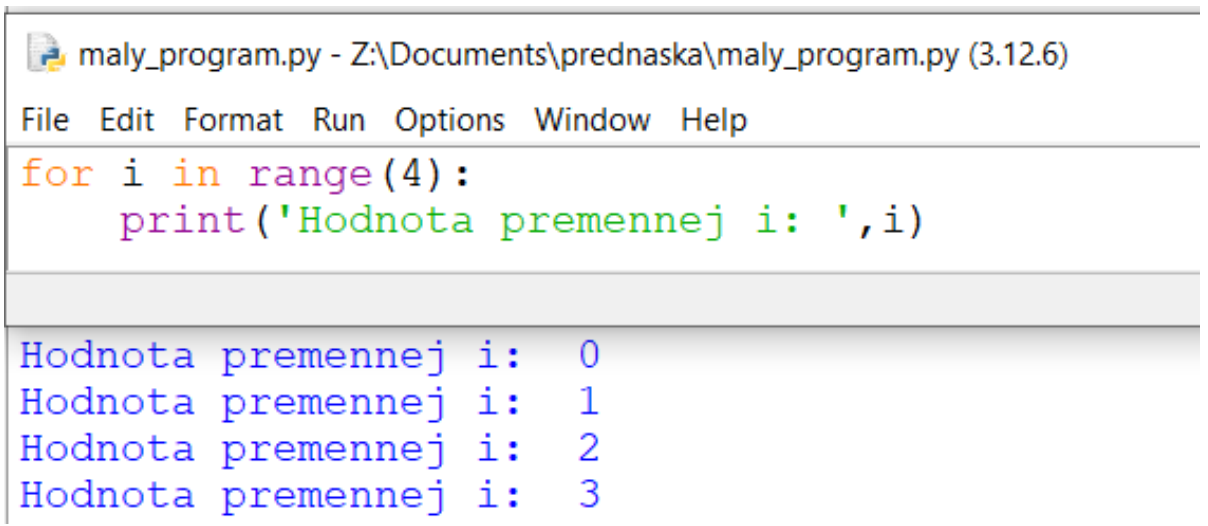
4) Po vykonaní tela cyklu (výpise hodnoty premennej *i*) sa zmení hodnota premennej *i* na ďalšiu hodnotu z postupnosti 0,1,2,3, t.j. na hodnotu *i* = 1 a znovu sa vykoná telo cyklu, t.j. sa vypíše obsah premennej *i*, tentokrát číslo 1.
5) Toto sa zopakuje aj pre *i* = *2*, aj pre *i* = *3*.



Ako to funguje?

Všimnite si, ako sa správa nasledovný kód aj s výpisom:

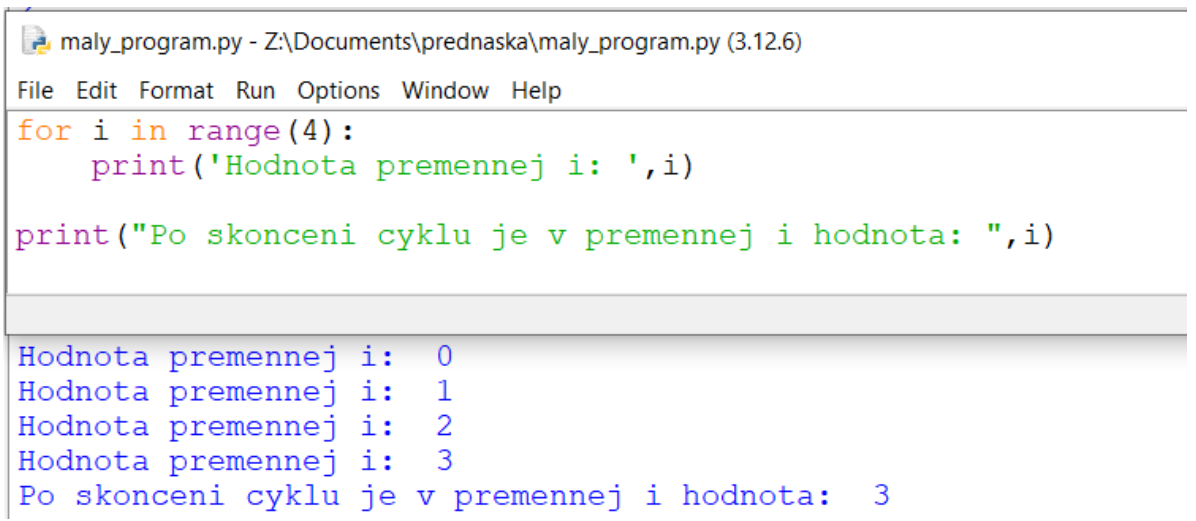
6) Keď sa vykoná štvrtá iterácia, kde má premenná *i* hodnotu 3, tak Python zistí, že už premenná *i* nadobudla **všetky** hodnoty z vygenerovanej postupnosti 0,1,2,3. Tým pádom príslušný cyklus **skončí**, pretože už neexistuje ďalšia hodnota, ktorú by premenná *i* mala nadobudnúť.
7) Dokopy sa teda cyklus zopakoval štyrikrát, pre hodnoty *i* = 0, 1, 2, 3.



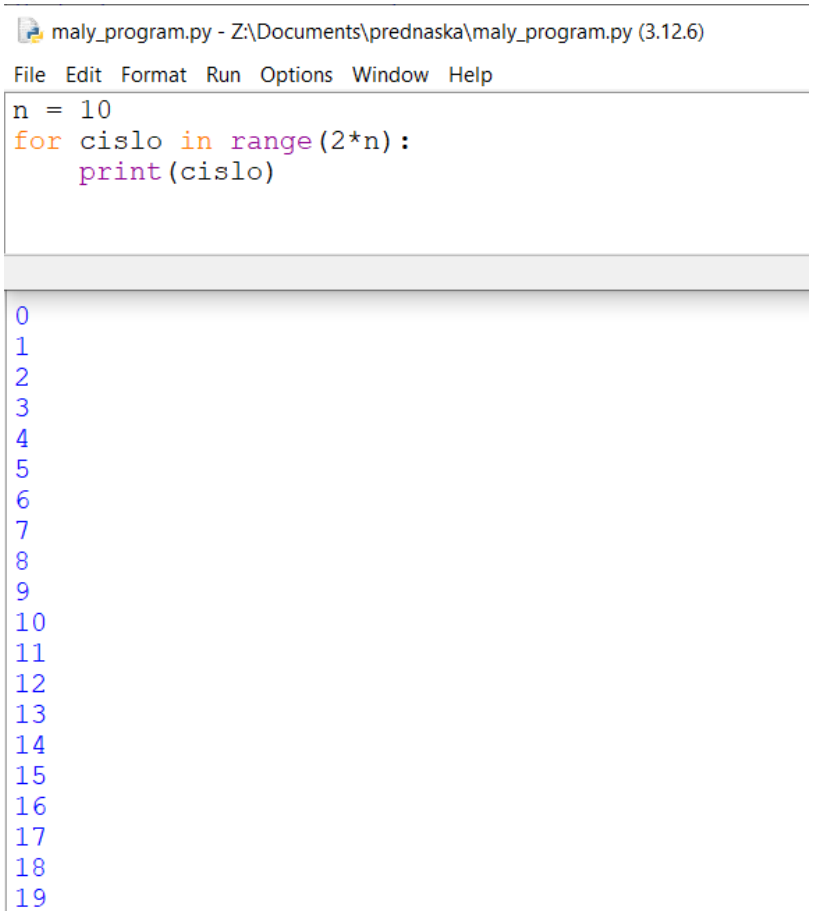
Ako to funguje?

Do tela for-cyklu patria **len tie príkazy,** ktoré sú odsadené voči príslušnému **for** príkazu! V našom príklade sú odsadené o 4 medzery.

Na ďalšom príklade si všimnite, že premenná *i* existuje **aj po skončení cyklu** a že k výpisu poslednej správy nedošlo v rámci cyklu, ale až po ňom, pretože príslušné volanie funkcie print() už nie je odsadené o 4 medzery voči **for** príkazu, teda je **mimo telo cyklu.**



Ako to funguje?



Na príklade vľavo si všimnite:

1) Vo funkcii *range*() môžeme používať ľubovoľné **celočíselné** výrazy. Napríklad vľavo je v premennej *n* uložená hodnota 10 a teda *range(2*n)* v danej situácii vygeneruje postupnosť čísiel 0,1,2,…,18,19

2) Riadiaca premenná cyklu môže mať v princípe ľubovoľné meno, v príklade vľavo som zvolil názov *cislo*

Prečo cykly?

1) Výhodou cyklov je jednoduchý zápis opakovania postupnosti príkazov.

2) Často sa stretnete v praxi so situáciou, že chcete opakovať nejakú činnosť, avšak počet opakovaní nejakým spôsobom závisí od počtu dát, ktoré máte k dispozícii. Napríklad sa stretneme so situáciou, že budeme chcieť pracovať so skupinou čísel, ktorých počet sa môže meniť a opakovanie nejakej činnosti bude závisieť práve od počtu čísiel v skupine!

3) Cykly sú taktiež užitočné v situácii, že potrebujete – z nejakého dôvodu – pracovať s postupne sa meniacimi číslami (napríklad potrebujete generovať čísla 1, 2, 3, … atď.). Cykly predstavujú veľmi jednoduchý spôsob, ako takéto čísla generovať.

4) A ešte veľa ďalších aplikácií...

Dôležitý príklad

Majme nasledovnú úlohu:

Pomocou for-cyklu napíšte funkciu *scitaj_n* so vstupným parametrom *n*, ktorá vráti **súčet** čísiel od 1 po *n*. (Na tento súčet existuje v matematike aj pekný vzorec, ale teraz to skúsime spraviť pomocou for-cyklu).

Príklady vstupu a výstupu:

- Volanie *scitaj_n(1)* vráti hodnotu 1.
- Volanie *scitaj_n(2)* vráti hodnotu 3, pretože 3 = 1 + 2.
- Volanie *scitaj_n(3)* vráti hodnotu 6, pretože 6 = 1 + 2 + 3.
- Volanie *scitaj_n(4)* vráti hodnotu 10, pretože 6 = 1 + 2 + 3 + 4.
- Volanie *scitaj_n(5)* vráti hodnotu 15, pretože 15 = 1 + 2 + 3 + 4 + 5.

Dôležitý príklad

Riešenie:
- chceme sčítať čísla 1 + 2 + … + *n* pre nejakú hodnotu parametra *n*.
- funkcia *range(n)* nám vie vyrobiť postupnosť čísiel 0, 1, …, n-1.
- Pomocou for-cyklu teda vieme vyrobiť premennú *i*, ktorá bude postupne nadobúdať hodnoty 0, 1, 2, …, n-1. Keďže chceme postupne sčítať hodnoty 1,2,3, …, n, vždy vezmeme hodnotu (i+1) a pripočítame ju do premennej *sucet*.

```
def scitaj_n(n):
    '''
    Funkcia scitaj_n(n) vrati sucet cisiel od 1 po n,
    teda vrati hodnotu 1+2+3+...+n.
    '''
    sucet = 0 #v premennej sucet budeme postupne vytvarat vysledny sucet
              #vsimnite si, ze ju MUSIME pred pouzitim nastavit v pripade suctu na NULA
    for i in range(n):  #toto vytvori premennu i s hodnotami 0, 1, 2, ..., n-1
        sucet = sucet + (i+1)  #takto zabezpecime, ze scitame hodnoty 1, 2, ..., n
    return sucet  #na zaver je v premennej sucet ulozeny pozadovany vysledok
```

Dôležitý príklad

Riešenie:
- všimnite si, že premenná *sucet* slúži na postupné pripočítavanie hodnôt premennej *i*, teda po poslednej iterácii for-cyklu v nej bude uložený súčet 1+2+3+...+n.
- pozrime sa, aké hodnoty bude funkcia vracať pre rôzne hodnoty vstupného argumentu:

maly_program.py - Z:\Documents\prednaska\maly_program.py (3.12.6)

File Edit Format Run Options Window Help

```
def scitaj_n(n):
    '''
    Funkcia scitaj_n(n) vrati sucet cisiel od 1 po n,
    teda vrati hodnotu 1+2+3+...+n.
    '''
    sucet = 0
    for i in range(n):
        sucet = sucet + (i+1)
    return sucet

for vstup in range(6):
    print("Pre vstup n = ",vstup," funkcia vratila hodnotu: ",scitaj_n(vstup))
```

```
Pre vstup n =  0  funkcia vratila hodnotu:  0
Pre vstup n =  1  funkcia vratila hodnotu:  1
Pre vstup n =  2  funkcia vratila hodnotu:  3
Pre vstup n =  3  funkcia vratila hodnotu:  6
Pre vstup n =  4  funkcia vratila hodnotu:  10
Pre vstup n =  5  funkcia vratila hodnotu:  15
```

Dôležitý príklad

Dôležité upozornenie:

- premenná *sucet* slúži vo funkcii na to, že do nej postupne pripočítavame jednotlivé sčítance.

- Ak by sme ju pred začiatkom cyklu **nenastavili na 0 (neinicializovali)**, program by nefungoval!!!

- pretože aby sa korektne vykonal príkaz v cykle:

    sucet = sucet + 1

  tak je nutné, aby aj v prvej iterácii for-cyklu bola už premennej *sucet* nejaká hodnota!

Dôležitý príklad

maly_program.py - Z:\Documents\prednaska\maly_program.py (3.12.6)

File Edit Format Run Options Window Help

```
def scitaj_n(n):
    for i in range(n):
        sucet = sucet + (i+1)
    return sucet

vstup = 5
print("Pre vstup n = ",vstup," funkcia vratila hodnotu: ",scitaj_n(vstup))
```

Ln: 6 Col: 9

```
Traceback (most recent call last):
  File "Z:\Documents\prednaska\maly_program.py", line 7, in <module>
    print("Pre vstup n = ",vstup," funkcia vratila hodnotu: ",scitaj_n(vstup))
  File "Z:\Documents\prednaska\maly_program.py", line 3, in scitaj_n
    sucet = sucet + (i+1)
UnboundLocalError: cannot access local variable 'sucet' where it is not associated with a value
```

Domáca úloha

1. Vyriešte úlohy z Cvičenia 2.

2. Odporúčam tiež, aby ste si prečítali kapitolu 3 z knihy. (úlohy na konci kapitoly riešiť nemusíte, namiesto nich je úloha č.9 v Cvičení 2)