

# PROG1: Prednáška 2

## Funkcie a for-cykly

# Domaca uloha

1. Vyriesit ulohy z Cvicenia 2.
2. Precitat kapitolu 3 z knihy. (ulohy na konci kapitoly riesit nemusite, namiesto nich je uloha 16 v cviceni 2)

# Funkcie

Funkcia = pomenovaná postupnosť príkazov, ktoré vykonávajú operácie so vstupnými argumentami funkcie

# Příklad 1

Uvažujme následovný skript:

```
import math  
  
a=math.sqrt(2)  
print(a)
```

- V druhom riadku používame funkciu `sqrt` z knižnice `math`.  
Hovoríme, že funkciu `sqrt` v druhom riadku **voláme**.
- Funkcia `sqrt` má jeden vstupný argument (číslo, ktorého druhú odmocninu chceme vypočítať).
- Funkcia `sqrt` funguje tak, že **vráti** hodnotu odmocniny vstupného argumentu

# Priklad 1 - pokracovanie

```
import math  
  
a=math.sqrt(2)  
print(a)
```

- Funkcia sqrt funguje tak, ze **vrati** hodnotu odmocniny vstupneho argumentu.
- Dolezite je tu slovo **vrati** !
- Co to znamena, ze funkcia nieco **vrati**?

# Priklad 1 - pokracovanie

```
import math  
  
a=math.sqrt(2)  
print(a)
```

- Funkcia sqrt funguje tak, ze **vrati** hodnotu odmocniny vstupneho argumentu.
- Dolezite je tu slovo **vrati** !
- Co to znamena, ze funkcia nieco **vrati**?
- Ak funkcia f **vrati** hodnotu h, znamena to, ze na mieste, kde sa vola funkcia f, Python toto volanie ohodnoti ako hodnotu h a bude s tymto volanim dalej pracovat ako s hodnotou h.

# Priklad 1 - pokračovanie

```
import math

a=math.sqrt(2)
print(a)
```

- Funkcia sqrt **vrati** hodnotu odmocniny vstupneho argumentu.
- Ak funkcia f **vrati** hodnotu h, znamena to, ze na mieste, kde sa vola funkcia f, Python toto volanie ohodnoti ako hodnotu h a bude s tymto volanim dalej pracovat ako s hodnotou h.
- V nasom priklade sa funkcia sqrt vola v riadku a=math.sqrt(2).
- Python rozpozna prikaz a=math.sqrt(2) ako prikaz priradenia.
- Python najprv ohodnoti pravu stranu prikazu priradenia.
- Na pravej strane je iba volanie funkcie sqrt so vstupnym argumentom 2.
- Python toto volanie ohodnoti ako 1.41... (priblizna hodnota odmocniny z 2) a cislo 1.41... priradi do premennej a.

# Vratit ≠ vypisat

Vyskusajte nasledovny skript:

```
import math  
  
math.sqrt(2)
```

- Nic sa nevypise.
- Ked Python pride na riadok `math.sqrt(2)`, ohodnoti volanie funkcie `sqrt` ako cislo 1,41... a nic viac (toto cislo neulozi do ziadnej premennej ani nevypise)
- Ak chceme, aby Python s cislom 1,41... dalej pracoval, musime ho ulozit do nejakej premennej! (tak ako sa to robi v prikaze `a=math.sqrt(2)` v Priklade 1)
- Ked mame cislo 1,41... ulozene v premennej `a`, mozeme s nim dalej pracovat (mozeme ho pouzit v dalsich vypoctoch v tom istom skripte) – **funkcie, ktore hodnotu vracaju, su oveľa uzitocnejšie ako funkcie, ktore hodnotu iba vypisuju !**



# Fruitful functions

- V knihe su funkcije, ktore vracaju nejaku hodnotu nazyvane ako **fruitful functions** (uroodne funkcije)
- Nie je to ale vseobecne zauzivane oznacenie

# Definovanie novych funkcii

```
def druha_mocnina(n):  
    a=n*n  
    return a
```

- V definicii funkcie definujeme, ako ma funkcia spracovat vstupne argumenty.
- Funkcia `druha_mocnina` definovana vyssie ma jeden vstupny argument (cislo `n`).
- Prikazom **return a** tu docielime, ze funkcia `druha_mocnina` bude **vracat** hodnotu ulozenu v premennej `a`.

# Prikaz return

Po vykonani prikazu **return**, beh funkcie vzdy skonci.

Napr. vo funkcii

```
def druha_mocnina(n):  
    a=n*n  
    return a  
    c=2*n  
    return c
```

sa riadky

```
    c=2*n  
    return c
```

nikdy nevykonaju.

# Definovanie novych funkcii (pokracovanie)

Po definovani funkcie, mozeme funkciu pouzivat.

```
def druha_mocnina(n):  
    a=n*n  
    return a  
  
b=druha_mocnina(2)  
print(b)
```

# Definovanie funkcie: Priklad 2

```
def priemer_dvoch(a,b):  
    p=(a+b)/2  
    return p
```

Funkcia `priemer_dvoch` definovana vyssie ma dva vstupne argumenty (cisla `a,b`).

# Definovanie funkcie: Priklad 3

```
def print_lyrics():  
    print("I'm a lumberjack, and I'm okay.")  
    print("I sleep all night and I work all day.")
```

Funkcia `print_lyrics` definovana vyssie ma 0 vstupnych argumentov.

# Lokálne premenné

Vyskúšajte skript:

```
def nasob5(n):  
    a=5  
    return a*n  
  
print(a)
```

# Lokálne premenné

Vyskúšajte skript:

```
def nasob5(n):  
    a=5  
    return a*n  
  
print(a)
```

Premenné definované v tele funkcie Python mimo funkcie nerozpoznáva!

Takéto premenné nazývame **lokálne**.



# for-cyklus

```
for i in range(3) :  
    print('Hello world!')
```

Ked napiseme

*for i in range(3):*

hovorime Pythonu nech vykona prikazy, ktore nasleduju za dvojbodkou a su odsadene o tabulator, pre kazde  $i$  z mnoziny  $\{0, 1, 2\}$ .

Python teda vypise *Hello world!* pre  $i=0$ , potom to vypise znovu pre  $i=1$  a potom este raz pre  $i=2$ .

# for-cyklus (pokracovanie)

Vyskúšajte skript:

```
for i in range(4) :  
    print(i)
```

# for-cyklus (pokracovanie)

Vyskúšajte skript:

```
for i in range(4) :  
    print(i)  
    print(i*i)  
    print()
```

# Dolezity priklad

Pomocou for-cyklu definujte funkciu s parametrom  $n$ , ktora vrati sucet cisel od 1 po  $n$ . (Na tento sucet existuje aj pekny vzorec, ale teraz chcem, aby sme to zratali pomocou for-cyklu.)

# Dolezity priklad

Pomocou for-cyklu definujte funkciu s parametrom n, ktora vrati sucet cisel od 1 po n.

Riesenie:

```
def sucet_po_hranicu(n):  
    sucet=0  
    for i in range(n) :  
        sucet=sucet+i+1  
    return sucet
```