Programování není jenom znalost jazyka!

Jazyk je jenom nástroj!

(Ale dnes to ještě BUDE o jazyku.)

Podprogramy („funkce“)

Příklad:

Vytiskněte tabulku malé násobilky ve tvaru

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X    X    1    2    3    4    5    6    7    8    9    10 X
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
X 1  X    1    2    3    4    5    6    7    8    9    10 X
X 2  X    2    4    6    8   10   12   14   16   18    20 X
X 3  X    3    6    9.................................
X 4  X    4    8.....................................
X 5  X    5   10.....................................
.................................................
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Co je funkce (podprogram)

- Něco, co vrací výsledky
  (ne, nemusí vracet nic)
- Pojmenovaná část zdrojového kódu…
- …připravená na opakované použití
  (ale můžeme ji použít jen jednou nebo vůbec)

Co potřebujeme umět

- Jak definovat novou funkci
- Jak vracet výsledek
- Jak popsat parametry (více možností)
- Jak volat funkci
- Způsoby předávání parametrů
- Viditelnost proměnných (a funkcí)
- Funkce B definovaná uvnitř funkce A
- Pořadí hledání významu identifikátoru
- Kdy je funkce známá
- Jak vracet více výsledků najednou
- Jak předávat a dosazovat funkci (všechno je objekt)
- Closure

Proč funkce/podprogramy

- členění problému/programu na části,
  které můžeme řešit odděleně
- oddělení CO TO DĚLÁ od JAK TO DĚLÁ
- skrývání proměnných atd., které mají význam jen pro řešení určité části
- re-use – možnost jednou vytvořený podprogram použít víckrát (i v jiných programech)

Jak definovat novou funkci

```
def jméno_funkce(parametry):
    příkaz
    příkaz
```

Jak vracet výsledek

```
return hodnota
```

- příkaz return ukončuje provádění funkce
  a vrací výsledek (pokud ho uvedeme)
- ve funkci jich může být libovolný počet (i žádný)

Jak popsat parametry

- závorky se musí uvádět,
  i když je seznam parametrů prázdný
- parametry se oddělují čárkou
- parametry mohou být:
  - obyčejné (poziční)
  - pojmenované s přiřazenou výchozí hodnotou
  - násobné/sběrné

Jak volat funkci

- zapsáním jména a uvedením parametrů
- závorky se píší, i když žádné parametry nejsou
- možnost volat s pojmenovanými parametry…

```
print( 10, 20, 30, sep='/', end='.' )
```

- …bez dodržení pořadí

```
print( 10, 20, 30, end='.', sep='/' )
```

Způsoby předávání parametrů

a) hodnotou (většina jazyků)

---

b) odkazem (většina jazyků jiných než Python)

c) konstantní parametry (Pascal, C#...)

d) výstupní parametry (C#)

e) výsledkem

f) jménem

g) …

…hodnotou:

= nová proměnná, do které se na začátku dosadí hodnota skutečného parametru.

Skutečným parametrem může být jakýkoliv výraz.

Viditelnost proměnných (a funkcí)

- z funkce lze číst globální proměnné
- ALE dosazením vytváříme vlastní lokální proměnnou
- …pokud ji neoznačíme jako globální:

```
global proměnná
```

- lokální proměnné funkce jsou viditelné pouze uvnitř

Funkce B definovaná uvnitř funkce A

- je viditelná pouze uvnitř funkce A (je tam lokální)
- může číst proměnné funkce A
- může do nich dosazovat, pokud je označí jako

  nonlocal proměnná

Pořadí hledání významu identifikátoru

"The LEGB rule":

- Local
- Enclosed
- Global
- Built-in

Kdy je funkce známá

Špatná zpráva:

Funkci můžeme zavolat, jenom pokud byla předtím definovaná.

Dobrá zpráva:

Tělo funkce se zkoumá až při volání.

```
def f():
    return g()

def g():
    return 7
```