Programování NENÍ 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 i jen jednou nebo vůbec)
- nástroj pro strukturování programu
  (rozkouskování, dekompozici)

Co potřebujeme umět/vědět

- Jak definovat novou funkci
- Jak vracet výsledek
- Jak popsat parametry (více možností)
- Jak volat funkci
- Způsoby předávání parametrů
- Viditelnost proměnných (a funkcí) (zvenku a zevnitř)
- 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

(Funkční) dekompozice programu

- 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

= To je těžké a budeme to zkoušet pořád!

- 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 (to jsou všechny)
  - 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='/' )
```

Tohohle si nevšímejte

```
def H(a,b,/, *z,c=77,d=88):
    print( f"a={a} b={b} c={c} d={d} z={z}" )


H(1,2,3,4,5,c=11,d=22)
a=1 b=2 c=11 d=22 z=(3, 4, 5)
H(1,2,3,4,5,d=22)
a=1 b=2 c=77 d=22 z=(3, 4, 5)
H(1,2,3,4,5,c=11)
a=1 b=2 c=11 d=88 z=(3, 4, 5)
```

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 (Algol 60)

g) ...

Předávání parametrů 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 (a funkce) jsou viditelné pouze uvnitř

To dovoluje brát funkci jako černou skříňku (CO to dělá)
a nestarat se o to, co je uvnitř (JAK to dělá).

Funkce ale může mít vedlejší efekty!
(Třeba číst vstup nebo měnit obsah vnějších proměnných.)

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 __builtins__

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
```

Příklad funkce

Potřebujeme postupně číst čísla,
která ale mohou být zapsaná tak, že je

- více čísel na jedné řádce
- vstup na více řádek

Takže pro čtení jednoho čísla musíme přečíst celou řádku
a zároveň nesmíme ztratit ta čísla, která tam ještě budou.

```
def PrectiJednoCisloZeVstupu():
    ...
```

Problém

Když si funkce potřebuje něco pamatovat mezi svými voláními:

- nemůže to být lokální proměnná (ta přestane existovat)
- dosazením do globální proměnné může přepsat obsah proměnné která se používá k jinému účelu
- ...nebo naopak jiná část programu může poškodit proměnnou, kterou využívá ta funkce.

Částečné řešení

```
def PrectiJednoCisloZeVstupu(kamUkladatZbytek):
```

Úplné řešení

objekty (nebo Closure) (obojí uvidíme později).

Příklad

Napište program, který přečte text ze vstupu ukončený slovem KONEC a vypíše dvacet slov s největším počtem výskytů.

Funkční dekompozice

= rozdělení práce pomocí funkcí

Programování shora (dolů)

= řešení úlohy rozkladem na pod-úlohy

= využívá volání (zatím neexistujících) podprogramů

Programování zdola (nahoru)

= vytváření podprogramů (řešení pod-úloh),

o kterých myslíme, že je budeme potřebovat

a následně z nich skládáme řešení větších

pod-úloh, až k hlavní úloze

Ladění zdola

testovací (pod)programy

Ladění shora

náhradní obsah podprogramů

Testovací podprogramy, vyhodnocování správnosti, počítání chyb, automatické testování – později.