# Podprogramy

Podprogramy můžeme chápat jako samostatné bloky kódu, které mají svůj název a spustí se právě při napsání tohoto názvu. Můžeme tak rozšířit Python o nové příkazy, kterým bude rozumět, až dosud jsme používali pouze jeho vestavěné příkazy.

Zápis podprogramu začíná definicí jeho názvu, pomocí klíčového slova **def**, a poté následuje odskočený blok příkazů, které se mají vykonat. Tyto části programu, nazýváme podprogramy, někdy také procedury nebo funkce.

Příklad

def vypistext():

print('toto je začátek podprogramu')

print('\*\*\* Python je moderní programovací jazyk \*\*\*')

print('toto je konec podprogramu')

Samotná definice podprogramu se v běhu programu nijak neprojeví, je nutné název podprogramu v kódu programu napsat. Toto napsání se nazývá **volání podprogramu**.

print('Vítej!')

vypistext()

Příklad: Pomocí podprogramu zajisti výpis textu: Python. Text bude pomocí cyklu vypsán vícekrát pod sebou, počet výpisů určí uživatel.

def vypistext():

print("Python")

pocet= int(input("Kolikrát chceš spustit podprogram?:"))

for i in range(0, pocet):

vypistext()

nebo řešení s cyklem v podprogramu

def vypistext():

pocet= int(input("Kolikrát chceš spustit podprogram?:"))

for i in range(0, pocet):

print("Python")

vypistext()

## Argumenty a parametry podprogramů, rozsah platnosti proměnných

Obecně v programování platí, že definovaná funkce (podprogram) dosáhne pouze na proměnné, které si **uvnitř sebe nadeklaruje**, na parametry **uvedené v hlavičce (argumenty)** a na **globální proměnné zpřístupněné deklarací global**. Ostatní proměnné jinde v programu či v jiných funkcích nevidí. Podívejme se podívat na to, jak to funguje v Pythonu, blíže.

Začněme parametry funkce. Ty uvádíme v hlavičce definice funkce v závorkách, jsou to jména proměnných, které můžeme uvnitř funkce používat. Jim se při volání funkce předává hodnota proměnné, kterou uvedeme na pozici parametru dané funkce. Předávané proměnné se při volání funkce říká **argument**.

def soucetcisel(prvni, druhe): # prvni a druhe jsou parametry funkce

soucet = prvni + druhe # používáme je jako deklarované proměnné uvnitř těla funkce

print(soucet)

soucetcisel(10, 5) # voláme funkci s **argumenty** prvni = 10 a druhe = 5

Funkce si může samozřejmě pro svoje vlastní potřeby nadefinovat vlastní lokální proměnné. Ty ale budou platit jen uvnitř funkce, jinde v programu jako by neexistovaly. Příkladem je proměnná **soucet** v předchozím příkladu. Kdybychom ji chtěli použít (např. vytisknout) mimo definici funkce, program by ohlásil chybu.

def soucetcisel(prvni, druhe): # prvni a druhe jsou parametry funkce

**soucet** = prvni + druhe # používáme je jako deklarované proměnné uvnitř těla funkce

print(soucet)

soucetcisel(10, 5) # voláme funkci s argumenty prvni = 10 a druhe = 5

print(**soucet**) # nelze!

Vstupní argumenty pro podprogram můžeme samozřejmě získat od uživatele.

Příklad: Program vyzve uživatele k zadání jeho oblíbeného čísla a následně mu je pomocí podprogramu vypíše.

def vypistext(jehocislo):

print("Tvé oblíbené číslo je: ", jehocislo)

cislo = int(input("Zadej své oblíbené číslo: "))

vypistext(cislo)

Jak jsou na tom **proměnné z vyšší úrovně uvnitř podprogramů?** Pokud nemáme v podprogramu proměnnou se stejným jménem, podprogram danou proměnnou vyšší úrovně vidí, ale nemůže ji změnit. Níže uvedený program sice proměnnou ***mojecislo*** vytiskne, ale když se jí pokusí změnit, ohlásí interpret chybu.

def vypistext(jehocislo):

print("Tvé oblíbené číslo je: ", jehocislo)

print("Číslo z vyšší úrovně programu, které mi nebylo předáno jako argument, ale dostanu se k němu: ", mojecislo)

mojecislo = 15 # toto už nelze!

cislo = int(input("Zadej své oblíbené číslo: "))

mojecislo = 10

vypistext(cislo)

Pokud si v podprogramu nadefinujeme lokální proměnnou se stejným názvem, jako již má nějaká proměnná na vyšší úrovni, nic se neděje. Podprogram s ní pracuje v rámci vlastního měřítka, globální proměnné si nevšímá.

Pokud z nějakého důvodu opravdu potřebujeme pracovat s proměnnou z vyšší úrovně, měnit její hodnotu i uvnitř podprogramu, zpřístupníme si ji příkazem global. Tím z ní v rámci měřítka podprogramu děláme **globální proměnnou**, která je nám plně k dispozici.

def vypistext(jehocislo):

global mojecislo

print("Tvé oblíbené číslo je: ", jehocislo)

print("Mám právo vypsat číslo z hlavního programu: ", mojecislo)

mojecislo = 15 # nyní ho můžeme měnit, změna zde v podprogramu zajistí změnu i v hlavním programu, jedná se o tu samou proměnnou!

print ("Mohu ho ale i změnit!:", mojecislo)

cislo = int(input("Zadej své oblíbené číslo: "))

mojecislo = 10

vypistext(cislo)

print("Jsme zpět v hlavním programu, číslo je i tady už jiné: ", mojecislo)

Tento přístup s sebou ovšem nese riziko. Změnou globální proměnné někde uvnitř funkce si opět koledujeme o nevypočitatelné chování programu, o chyby, které budeme složitě odhalovat.

Pokud nepotřebujeme od podprogramu (funkce) žádnou návratovou hodnotu, jedná se o tzv. **proceduru**. V případě **funkce** naopak příkazem return říkáme, jakou hodnotu má podprogram vrátit.

Příklad: podprogram spočítá druhou odmocninu zadaného čísla, výsledek vrátí zpět do hlavního programu, kde bude následně zobrazen uživateli.

def Druha\_odmocnina(x):

odmocnina = x \*\* (0.5)

**return** odmocnina

c = float(input("Zadej číslo x pro výpočet jeho druhé odmocniny:"))

print("Druhá odmocnina z", c, "je:", Druha\_odmocnina(c))

Příklad: Zadejte věk psa a následně ho pomocí podprogramu vypište převedený na lidský věk, tj. vynásobený sedmi. Vyřešte pomocí procedury a pak i pomocí funkce.

Pomocí **procedury**

def prevedvek(vstup):

print("Zadaný psí věk " +str(vek)+ " odpovídá u člověka věku: ", vstup\*7)

vek = int(input("Zadej věk psa: "))

prevedvek(vek)

Pomocí **funkce**

def prevedvek(vstup):

return vstup\*7

vek = int(input("Zadej věk psa: "))

print("Zadaný psí věk " +str(vek)+ " odpovídá u člověka věku: ", prevedvek(vek))

**Proč používat funkce a procedury?**

U krátkých jednoduchých programů většinou podprogramy nepotřebujeme. U větších už ale ano. K čemu to bude dobré? Kusy kódu, které potřebujeme opakovaně, můžeme napsat pouze jednou a pak už je jen jako podprogram volat. Tím se nám celkový kód zkrátí a zpřehlední. Vyhneme se také chybám, které by kusy podobného kódu roztroušené všude možně po programu, mohly obsahovat.

Příklady k zopakování učiva:

Příklad 1: připravte funkci, která vrátí obsah obdélníka daných rozměrů.

Příklad 2: připravte funkci, která umocní zadané číslo na zadaný exponent.

Příklad 3: pomocí procedury připravte kreslení obdélníku, tvořeného znaky x. Délku a šířku stran zadá v hlavním programu uživatel.