10. Programování - funkce, moduly

Funkce - výhody jejich použití

- Funkce jsou podprogramy, které vykonávají blok kódu a voláme je jménem.
- Mohou vracet užitečné hodnoty (např. nějakého výpočtu), vždy vrací alespoň None.
- Můžeme jim předat žádný, jeden nebo více parametrů.

Výhody:

- Přehlednost - umožňují oddělit určitou část kódu do samostatného bloku popsaného jménem funkce nebo komentářem
- Opakované použití - ušetříme psaní, poté je jen stačí volat (i s různými parametry)
- Snižuje se redundance
- Testování funkce nezávisle na ostatním kódu

definice funkce a její volání

Deklarace funkce začíná klíčovým slovem **def**, následuje jméno funkce (stejná omezení jak u proměnných) a v závorce pak argumenty. Více argumentů se odděluje čárkami. Poté dvojtečka a odsazený blok kódu.

```
def pozdrav(jmeno):
    print("Hello,", jmeno)
pozdrav(“Karel”)
```

typy parametrů

Parametry se uvádějí v závorce **v pořadí**:

1. **poziční**, při volání povinné
2. **implicitní** (výchozí, default, pojmenované), při volání nepovinné, mají výchozí hodnotu, která se použije v případě, že tento parametr nebude dosazen
3. **volitelné poziční** (*tuple) - proměnlivý počet **nepojmenovaných** parametrů "navíc"

   ```
   def prumer(*args):
       return sum(args)/len(args)
   ```

4. **volitelné pojmenované** (**slovník) - proměnlivý počet **pojmenovaných** parametrů "navíc"

```
def fce(poz, vych=1, *dalsi, **par): …
```

Pořadí musí být dodrženo! Vpravo od implicitního parametru nemohou stát poziční. Vpravo od volitelných pozičních mohou být jen volitelné pojmenované a za těmi už nesmí být nic.

Při **volání funkce** se hodnoty dosazují **zleva doprava**. Nechcete-li přemýšlet, v jakém pořadí jsou zapsány parametry v definici funkce, můžete je dosazovat jejich jmény. Tato možnost se také hodí pro implicitní parametry, chcete-li přiřadit hodnotu implicitnímu parametru a přitom nepřiřazovat hodnoty všem implicitním parametrům vlevo od něj.

návratová hodnota

Funkce v jazyce Python nemají předem určený datový typ návratové hodnoty.

Pokud funkce provede příkaz return, vrátí v něm uvedenou hodnotu (return může být i bez hodnoty - vrací None, to je užitečné pro předčasné ukončení fce).

Pokud neuvedeme return, vrací se vždy **None** - pythonovský ekvivalent hodnoty null, void, nic, žádná hodnota.

```
def Název(parametr1, parametr2, ...):
    blok příkazů
    return výsledek
```

tohle se už neprovede…

použití globálních a lokálních proměnných

V definici fce fungují **parametry** jako lokální proměnné.

Proměnné definované uvnitř funkcí jsou tzv. **lokální** - existují jen uvnitř funkce a po jejím skončení zmizí. Nemají nic společného s proměnnými vně.

Proměnné vně fce jsou tzv. **globální** - existují po celou dobu trvání programu. Ve funkcích je můžeme číst.

Pokud chceme uvnitř funkce zapisovat do globální proměnné, musíme ji na začátku bloku fce definovat: *global x*

Jestliže tato proměnná existovala, napojí se na ni, jinak se založí nová globální proměnná.

volání funkce komponentou

Např. v komponentách Tkinteru jsou parametry command, do kterých předáváme název funkce (bez závorek - jako referenci). Funkce/metody Tkinteru je pak mohou vnitřně podle reference volat.

V parametru command se očekává fce bez povinných parametrů, což se dá obejít lambda funkcí.

lambda funkce

Lambda je anonymní funkce (většinou nemá jméno) pro vykonávání jednoduchých výrazů. V Pythonu jsou tvořeny klíčovým slovem lambda, výčtem argumentů, dvojtečkou a samotným tělem funkce. Tělo funkce je tvořeno jediným výrazem. Ten je při zavolání funkce vyhodnocen a jeho hodnota je zároveň návratovou hodnotou lambda funkce.

f = lambda x,y: x%y # přiřazená proměnné - má jméno, vrací zbytek po dělení

Používá se především k vytvoření jednorázových funkcí nebo funkcí pro zpracování dat v seznamu - třeba s funkcemi **filter**(f, sek) nebo **map**(f, sek):

```
delitele_sesti=filter(lambda x: x%6==0, range(20,60))
```

Když potřebujeme dosadit jako parametr funkci s nesprávným počtem parametrů, můžeme dosadit lambda fci se správným počtem parametrů a v ní zavolat tu původní. Např. v modulu Tkinter u parametru **command**, kde nemůžeme používat funkce s parametry:

```
def Mocnina(x):
    print (x*x)
Button(hlavni, command=lambda:Mocnina(451))
```

moduly a jejich import, výhoda modulů

Moduly (knihovny) jsou další pythonovské soubory (nebo složitější adrésářové struktury), jejichž kód můžeme importovat a využívat v našem kódu.

Každý modul, který chceme použít, je nutné **importovat**:

1) import modul → modul.funkce()

2) import modul as m → m.funkce()

3) from modul import * → funkce()

POZOR! - může dojít k **přepsání** importovaných proměnných nebo fcí, pokud se v importovaných modulech jmenují stejně

funkce	mohu volat (spouštět)
proměnné	mohu použít
volné příkazy	se provedou

Jako **vlastní moduly** budou fungovat obyčejné soubory *.py. Pokud obsahují příkazy, které nechceme spoštět při importu, musíme je obalit ve speciální podmínce:

if __name__ == "__main__":

Příkazy…

Výhody:

- přehlednost kódu, organizace složitějších projektů do logických celků
- znovupoužití (recyklace) již napsaného kódu, např. často používané funkce
- jednoduché použití cizích knihoven (např. math)
- jednoduchá publikace vlastních modulů pro jiné programy

typování (navíc)

Python sice nemá pevné nebo explicitní datové typy proměnných a funkcí, přesto je ale můžeme uvádět - Python nám od verze 3.6 umožňuje psát **typové anotace**. Interpret Pythonu je ignoruje, ale dodávají kódu přehlednost a umožňují jednodušší hledání chyb.

K proměnným a argumentům funkcí píšeme typové anotace tak, že za jejich jméno napíšeme dvojtečku a daný typ. U funkcí můžeme anotovat návratovou hodnotu tak, že v hlavičce funkce vepíšeme před závěrečnou dvojtečku šipku (->) a za ní typ hodnoty, kterou funkce vrací.

def typed(arg1: str, arg2: float, arg3: int) -> bool:

return True

Funkce bere tři argumenty – řetězec, desetinné číslo, celé číslo a vrací logickou hodnotu.

Pro otypování složitějších datových struktur (seznamy, slovníky, n-tice atd.) se již neobejdeme bez standardního modulu **typing**.