

Funkcie

Poznámky k téme

Funkcia je pomenovaný kúsok programu, ktorý môžeme spustiť (zavolať) toľkokrát, koľkokrát potrebujeme. Funkcie už poznáš – `print`, `input`, `len`, `round` sú funkcie, ktoré pripravili autori Pythonu. V tejto téme sa naučíš písať vlastné. Vďaka funkciám sa program dá rozdeliť na menšie časti, z ktorých každá rieši jednu úlohu, a ten istý kód netreba písať viackrát.

Čo sa naučíš

- definovať funkciu príkazom `def` a zavolať ju,
- odovzdať funkcii údaje cez parametre,
- vrátiť výsledok príkazom `return` a odlíšiť ho od `print`,
- rozumieť lokálnym a globálnym premenným.

1 Definícia a volanie funkcie

```
def pozdrav():  
    print("Ahoj!")  
    print("Vitaj na hodine informatiky.")
```

```
pozdrav()  
pozdrav()
```

Výstup:

```
Ahoj!  
Vitaj na hodine informatiky.  
Ahoj!  
Vitaj na hodine informatiky.
```

- Riadok `def pozdrav():` funkciu **definuje**: `def` je kľúčové slovo, `pozdrav` meno funkcie, za ním zátvorky a dvojbodka.
- Odsadený blok pod ním je **telo funkcie**.
- Definícia funkciu **nespustí** – iba si ju Python zapamätá. Telo sa vykoná až pri **volaní** `pozdrav()`, a to pri každom volaní znova.

Funkcia musí byť definovaná **skôr**, než ju zavolaš (program sa vykonáva zhora nadol). Preto sa definície funkcií píšú na začiatok programu a samotný „hlavný program“ až pod ne.

```
vypis()

def vypis():
    print("Toto sa nevypíše.")
```

Výstup (program skončí chybou):

```
Traceback (most recent call last):
  File "program.py", line 1, in <module>
    vypis()
    ^^^^^
NameError: name 'vypis' is not defined
```

Pri volaní nezabudni na zátvorky: `pozdrav` bez zátvoriek funkciu nezavolá – iba sa na ňu odkáže.

2 Parametre

Funkcia je užitočnejšia, keď jej môžeš odovzdať údaje. V definícii ich zapíšeš ako **parametre** – mená v zátvorkách. Pri volaní do zátvoriek napíšeš **argumenty** – konkrétne hodnoty, ktoré sa do parametrov dosadia.

```
def pozdrav(meno):
    print("Ahoj,", mena + "!")

pozdrav("Ema")
pozdrav("Tomáš")
```

Výstup:

```
Ahoj, Ema!
Ahoj, Tomáš!
```

Pri prvom volaní má parameter `meno` hodnotu `"Ema"`, pri druhom `"Tomáš"`.

2.1 Viac parametrov

Parametre sa oddeľujú čiarkou. Argumenty sa do nich dosadia **podľa poradia**:

```
def obdlznik(sirka, vyska, znak):
    for _ in range(vyska):
        print(znak * sirka)

obdlznik(6, 2, "#")
print()
obdlznik(3, 3, "o")
```

Výstup:

```
#####
#####

ooo
ooo
ooo
```

Ak sa počet argumentov nezhoduje s počtom parametrov, vznikne chyba:

```
def obdlznik(sirka, vyska, znak):
    for _ in range(vyska):
        print(znak * sirka)

obdlznik(6, 2)
```

Výstup (program skončí chybou):

```
Traceback (most recent call last):
  File "program.py", line 5, in <module>
    obdlznik(6, 2)
TypeError: obdlznik() missing 1 required positional argument: 'znak'
```

2.2 Pomenované argumenty a predvolené hodnoty

Argument sa dá zadať aj **menom** parametra – potom na poradí nezáleží. Parameter môže mať aj **predvolenú hodnotu**, ktorá sa použije, ak argument chýba:

```
def obdlznik(sirka, vyska, znak="*"):
    for _ in range(vyska):
        print(znak * sirka)

obdlznik(4, 2)                # znak bude "*"
obdlznik(vyska=1, sirka=8, znak="=")
```

Výstup:

```
****
****
=====
```

Takto funguje aj `print(..., end="")` – `end` a `sep` sú pomenované parametre s predvolenou hodnotou.

3 Návratová hodnota: return

Doteraz naše funkcie iba vypisovali. Často však chceme, aby funkcia niečo **vypočítala a výsledok odovzdala** ďalej – rovnako ako `len("abc")` odovzdá číslo 3. Na to slúži príkaz `return`:

```
def obsah_obdlznika(a, b):
    return a * b

s = obsah_obdlznika(4, 5)
print("Obsah je", s)
print("Dva také obdĺžniky:", 2 * obsah_obdlznika(4, 5))
```

Výstup:

```
Obsah je 20
Dva také obdĺžniky: 40
```

Volanie funkcie sa vo výraze **nahradí vrátenou hodnotou**: `obsah_obdlznika(4, 5)` sa správa ako číslo 20. S výsledkom môžeme ďalej počítat, uložiť ho do premennej alebo ho poslať inej funkcii.

3.1 return verzus print

Toto je jedna z najdôležitejších vecí v celej téme:

- `print` výsledok **ukáže na obrazovke** – program ho však nemá kde použiť.

- `return` výsledok **odovzdá** tomu, kto funkciu zavola – môže ho uložiť, porovnať, vypísať.

```
def druha_mocnina_print(x):  
    print(x * x)  
  
def druha_mocnina_return(x):  
    return x * x  
  
a = druha_mocnina_print(3)  
b = druha_mocnina_return(3)  
print("a =", a)  
print("b =", b)
```

Výstup:

```
9  
a = None  
b = 9
```

Prvá funkcia vypísala 9, ale do `a` uložila `None` – špeciálnu hodnotu, ktorá znamená „nič“. **Funkcia bez `return` vráti `None`.** Ak by sme chceli s `a` počítať (`a + 1`), vznikla by chyba.

Pravidlo: Funkcia, ktorá niečo **počíta**, má výsledok vracať cez `return`. Vypisovať ho má až hlavný program. Takú funkciu potom môžeš použiť kdekoľvek – aj tam, kde výsledok vypisovať nechceš.

3.2 `return` ukončí funkciu

Po vykonaní `return` sa funkcia okamžite skončí; ďalšie riadky jej tela sa už nevykonajú. To sa hodí vo funkciách s podmienkami:

```
def znamienko(x):  
    if x > 0:  
        return "kladné"  
    if x < 0:  
        return "záporné"  
    return "nula"  
  
print(znamienko(5), znamienko(-2), znamienko(0))
```

Výstup:

```
kladné záporné nula
```

Tu nepotrebujeme `elif` ani `else`: ak platí prvá podmienka, funkcia skončí a ďalšie riadky sa nevykonajú.

3.3 Funkcia vracajúca True alebo False

Funkcie, ktoré odpovedajú na otázku áno/nie, vracajú logickú hodnotu. Zvyknú sa pomenovať `je_...`:

```
def je_parne(n):  
    return n % 2 == 0  
  
for x in range(1, 6):  
    if je_parne(x):  
        print(x, "je párne")
```

Výstup:

```
2 je párne  
4 je párne
```

Zápis `return n % 2 == 0` je kratší ako `if n % 2 == 0: return True` a `else: return False` – porovnanie samo už je `True` alebo `False`.

3.4 Viac návratových hodnôt

Funkcia môže vrátiť aj viac hodnôt oddelených čiarkou. V skutočnosti vráti jednu n-ticu (téma N-tice), ktorú možno rozbiť do viacerých premenných:

```
def min_a_max(a, b, c):  
    return min(a, b, c), max(a, b, c)  
  
najmensie, najvacsie = min_a_max(7, 2, 9)  
print(najmensie, najvacsie)
```

Výstup:

```
2 9
```

4 Lokálne a globálne premenné

Premenné vytvorené **vo vnútri** funkcie (aj parametre) sú **lokálne**: existujú iba počas behu funkcie a mimo nej o nich nikto nevie. Premenné vytvorené v hlavnom programe sú **globálne**.

```
def vypocet():  
    vysledok = 42  
    print("vo funkcii:", vysledok)  
  
vypocet()  
print("mimo funkcie:", vysledok)
```

Výstup (program skončí chybou):

```
vo funkcii: 42  
Traceback (most recent call last):  
  File "program.py", line 6, in <module>  
    print("mimo funkcie:", vysledok)  
                          ^^^^^^^  
NameError: name 'vysledok' is not defined
```

Lokálna premenná môže mať aj rovnaké meno ako globálna – sú to však dve rôzne premenné. Priradenie vo funkcii zmení iba tú lokálnu:

```
x = 5

def zmen():
    x = 10
    print("vo funkcii x =", x)

zmen()
print("mimo funkcie x =", x)
```

Výstup:

```
vo funkcii x = 10
mimo funkcie x = 5
```

Globálnu premennú funkcia **čítať môže**, ak lokálnu s rovnakým menom nevytvorí. Zmeniť ju by mohla iba po príkaze `global x`. To sa však **neodporúča** – program, v ktorom funkcie menia globálne premenné, sa ťažko číta aj opravuje. Lepšie je údaje funkcii odovzdať parametrami a výsledok vrátiť cez `return`.

5 Ako písať dobré funkcie

- **Jedna funkcia = jedna úloha.** Funkcia `nacitaj_cisla`, ktorá iba načíta, `spocitaj_priemer`, ktorá iba počíta.
- **Výstižné meno** so slovesom: `vypocitaj_obsah`, `je_prvocislo`, `vypis_tabulku`.
- **Údaje dnu parametrami, výsledok von cez return** – nie cez globálne premenné ani `input/print` vo vnútri, ak to nie je úlohou funkcie.
- Na začiatok tela môžeš napísať **dokumentačný reťazec** (*docstring*) – krátky popis v trojitých úvodzovkách:

```
def obsah_kruhu(r):
    """Vráti obsah kruhu s polomerom r."""
    return 3.14159 * r ** 2

print(round(obsah_kruhu(2), 2))
```

Výstup:

```
12.57
```

6 Riešené úlohy

6.1 Je číslo prvočíslo?

Zadanie: Napíš funkciu `je_prvocislo(n)`, ktorá vráti `True`, ak je `n` prvočíslo, inak `False`. Pomocou nej vypíš prvočísla do 30.

Riešenie: Čísla menšie ako 2 nie sú prvočísla. Inak skúsime delitele od 2; keď nájdeme deliteľa, môžeme hneď vrátiť `False`. Ak cyklus prebehne celý, deliteľa niet a vrátime `True`.

```
def je_prvocislo(n):  
    if n < 2:  
        return False  
    for d in range(2, n):  
        if n % d == 0:  
            return False  
    return True  
  
for x in range(31):  
    if je_prvocislo(x):  
        print(x, end=" ")  
print()
```

Výstup:

```
2 3 5 7 11 13 17 19 23 29
```

Všimni si, kde je posledné `return True`: **za** cyklom, nie v ňom. Keby bolo v cykle v `else` vetve, funkcia by sa rozhodla už pri prvom deliteľovi.

6.2 Faktoriál

Zadanie: Napíš funkciu `faktorial(n)`, ktorá vráti $n! = 1 \cdot 2 \cdot \dots \cdot n$ (a $0! = 1$).

```
def faktorial(n):  
    vysledok = 1  
    for i in range(2, n + 1):  
        vysledok *= i  
    return vysledok  
  
print(faktorial(0), faktorial(5), faktorial(10))
```

Výstup:

```
1 120 3628800
```

6.3 Najväčší spoločný deliteľ

Zadanie: Napíš funkciu `nsd(a, b)`, ktorá vráti najväčší spoločný deliteľ dvoch kladných čísel, a pomocou nej skráť zlomok $84/126$.

Riešenie: Euklidov algoritmus: kým `b` nie je 0, nahradíme dvojicu `(a, b)` dvojicou `(b, a % b)`. Keď je `b` nula, v `a` je výsledok.

```
def nsd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

citateľ, menovateľ = 84, 126
d = nsd(citateľ, menovateľ)
print("NSD =", d)
print(f"{citateľ}/{menovateľ} = {citateľ // d}/{menovateľ // d}")
```

Výstup:

```
NSD = 42
84/126 = 2/3
```

6.4 Program rozdelený na funkcie

Zadanie: Načítaj body troch písomiek (max. 50 bodov každá), vypočítaj priemerné percento a vypíš známku.

Riešenie: Program rozdelíme na funkcie podľa úloh: výpočet percent, výpočet známky a hlavný program, ktorý ich použije. Každú funkciu možno samostatne vyskúšať.

```
def percenta(body, maximum):
    return body / maximum * 100

def znamka(p):
    if p >= 90:
        return 1
    if p >= 75:
        return 2
    if p >= 50:
        return 3
    if p >= 30:
        return 4
    return 5

spolu = 0
for i in range(1, 4):
    spolu += int(input(f"Body z písomky {i}: "))
p = percenta(spolu, 3 * 50)
print(f"Úspešnosť: {p:.1f} %, známka: {znamka(p)}")
```

Výstup:

```
Body z písomky 1: 41
Body z písomky 2: 35
Body z písomky 3: 47
Úspešnosť: 82.0 %, známka: 2
```

7 Časté chyby

- **Volanie bez zátvoriek:** `pozdrav` namiesto `pozdrav()` nič nespustí; `print(obsah)` vypíše niečo ako `<function obsah at 0x...>`.

- **print namiesto return:** funkcia vypíše výsledok, ale vráti `None`. Výraz `vysledok + 1` potom skončí chybou `TypeError`.
- **Volanie pred definíciou:** `NameError` – definície funkcií patria na začiatok.
- **Zlý počet argumentov:** `TypeError: ... missing 1 required positional argument`.
- **return v cykle priskoro:** `return` v tele cyklu ukončí celú funkciu už pri prvom opakovaní.
- **Chýbajúci return v niektorej vetve:** ak funkcia v niektorom prípade nedôjde k `return`, vráti `None`.

```
def absolutna(x):  
    if x > 0:  
        return x  
    elif x < 0:  
        return -x  
  
print(absolutna(-4), absolutna(0))
```

Výstup:

```
4 None
```

Pre `0` neplatí žiadna podmienka, funkcia skončí bez `return` a vráti `None`. Oprava: `if x >= 0`.

Zhrnutie

- `def meno(parametre):` + odsadené telo; funkcia sa vykoná až pri volaní `meno(argumenty)`.
- Argumenty sa dosadia do parametrov podľa poradia alebo podľa mena; parameter môže mať predvolenú hodnotu.
- `return hodnota` funkciu ukončí a vráti hodnotu; volanie sa vo výraze nahradí touto hodnotou.
- Funkcia bez `return` vráti `None`. Počítajúce funkcie majú výsledok vracat', nie vypisovať.
- Premenné vo funkcii (aj parametre) sú lokálne; globálne premenné vo funkcii nemeň.
- Jedna funkcia = jedna úloha, výstižné meno, údaje dnu parametrami a výsledok von cez `return`.

Úlohy na precvičenie

1. Napíš funkciu `obvod_obdlznika(a, b)` a použi ju pre tri rôzne obdĺžniky.
2. Napíš funkciu `max3(a, b, c)`, ktorá vráti najväčšie z troch čísel bez použitia `max`.
3. Napíš funkciu `je_priestupny(rok)` a vypíš všetky priestupné roky od 1990 do 2030.
4. Napíš funkciu `ciferny_sucet(n)` a nájdi trojciferné číslo s najväčším ciferným súčtom.
5. Napíš funkciu `riadok(n, znak)`, ktorá vypíše `n` znakov za sebou, a pomocou nej nakresli štvorec.
6. Napíš funkciu `nsn(a, b)` – najmenší spoločný násobok. Pomôcka: `nsn(a, b) = a * b // nsd(a, b)`.

Úlohy k tejto téme nájdesh na learnwithmente.com v časti Informatika → Python.