Függvények

2025

1 Függvények használata

A függvény egy olyan blokk, ami csak akkor fut le, amikor meghívják. A függvényhívás segítségével a blokk többször is felhasználható. A függvény deklarációja meg kell előzze a hívást.

1.1 Függvény létrehozása

A függvény a `def` kulcsszóval kezdődik, majd a neve következik (ami utal a feladatára). A név után zárójelek `()` között kerülnek felsorolásra a paraméterek, amelyek vesszővel `,` kerülnek elválasztásra. A sort kettőspont `:` zárja. A függvény törzsében szereplő utasításokat bejebb kell húzni. A törzs elején szerepelhet egy dokumentációs szöveg, ami segítséget nyújthat a függvény funkciójának és paramétereinek értelmezéséhez.

```
def <függvénynév>([paraméterek]):
    """ dokumentációs szöveg """
    utasítások
    return visszatérési érték
```

[ ]:
```
def fuggvenyem():
    """Ez egy példafüggvény"""
    print("Ez egy függvény")


fuggvenyem()  # függvény hívása, mindig kell a zárójel
```

A paramétereken keresztül adhatunk át információt a függvénynek. A paraméterek a függvény neve után, zárójelben vannak megadva. Tetszőleges számú paraméter megadható, vesszővel elválasztva kell felsorolni őket. Általában a függvényhívásakor az átadott argumentumok számának és sorrendjének meg kell egyeznie a paraméterek számával és sorrendjével, azonban a Python lehetőséget ad egyéb megadásokra is.
Az argumentum az az érték, amelyet a függvény a meghívásakor küld. A paraméter a függvénydefinícióban szereplő bemenet, amely a függvény törzsében változóként szerepel.

[ ]:
```
def koszon(nev):
    print(f"Hello {nev}!")


koszon("Tamás")
koszon("Anna")
```

```
koszon("Lajos")
```

1.2 Visszatérési érték

Ha a függvényünknek az a feladata, hogy adjon vissza értéket, akkor azt a `return` utasítással tehetjük meg. A `return` visszaad egy adatot, ha azt akarjuk jelezni, hogy nincs mivel visszatérni, akkor a `None` értéket adjunk vissza.

A függvény a `return` utasítással vissza tud adni értéket, amit függvényhíváskor felhasználhatunk. Az átadott argumentum típusát nem vizsgálja, ezért előfordulhat, hogy nem várt eredményt kapunk.

[ ]:
```
def otszoros(x):
    return 5 * x


print(otszoros(3))
print(otszoros(5.0))
print(otszoros("1"))
```

1.2.1 Több érték visszaadása

A `return`-nal egyetlen adattal térhetünk vissza. Azonban, ha a visszaadott adat egy tuple vagy lista, akkor azt szétszedhetjük fogadáskor több változóba.

[ ]:
```
def min_max(elemek):
    """Lista elemeinek minimumát és maximumát visszaadó függvény"""
    return (min(elemek), max(elemek))  # 2 érték becsomagolva egy tuple␣
↪szerkezetbe


szamok = [12, 45, 11, 47, 32, 26]
minimum_ertek, maximum_ertek = min_max(szamok)  # tuple szétszedése
```

1.3 Paraméterátadás

1.3.1 Referencia szerinti paraméterátadás

Az átadott argumentumok referencia szerint kerültek átadásra a paramétereknek. Ha az átadott objektum immutable (nem módosítható) típusú, akkor a függvény törzsén belül elvégzett módosítások nem hatnak vissza a híváskor használt változóra.

[ ]:
```
def novel(szam):
    szam += 1  # a szam csak lokálisan elérhető változó, ami híváskor kap egy␣
↪értéket


szamom = 0
novel(szamom)  # nem változik meg a szamom változó
```

```
print(szamom)
```

1.3.2 Lista használata paraméterként

Ha listát adunk át argumentumként, akkor egy mutable (módosítható) objektum referenciáját fogja a függvény megkapni, ami azt jelenti, hogy ebben az esetben a listán végzett összes változtatás kihat az átadott lista elemeire.

[ ]:
```
def skalaz(elemek, n):
    for index in range(len(elemek)):
        elemek[index] *= n


szamok = [1, 2, 3, 4]  # eredeti elemek
skalaz(szamok, 3)  # háromszoros
print(f"Átskálázott elemek: {szamok}")
```

1.4 Argumentumok megadási módjai

1.4.1 Alapértelmezett értékkel rendelkező paraméter

A függvény paramétere kaphat alapértélmezett értéket. Mindig jobbról haladva lehet alapértelmezett értéket megadni a paramétereknek egy = jellel, a többi paraméter kötelező jellegű. Ha a függvényt kevesebb argumentummal hívjuk meg, mint amennyi paramétere van, akkor az alapértelmezett értéket használja a hiányzó argumentumok helyett.

[ ]:
```
def koszon(name="Kata"):
    print(f"Hello {name}!")


koszon("Tamás")
koszon()  # az alapértelmezett értéket veszi fel
koszon("Lajos")
```

1.4.2 Pozíció szerinti paramétermegadás

Az argumentumok kitöltése ugyanolyan sorrend szerint történik, mint amilyen a függvény definíciójában szereplő paraméterek sorrendje.

[ ]:
```
def koszon(vezeteknev, keresztnev, elotag=""):
    print(f"Hello {elotag} {vezeteknev} {keresztnev}!")


koszon("Nagy", "Tamás")
koszon("Kiss", "Lajos", "ifj.")
koszon("Kovács", "László", "Dr.")
```

1.4.3 Név szerinti paramétermegadás

Az argumentumokat átadhatjuk `kulcs = érték` szintaxissal. Így az argumentumok sorrendje nem számít.

[ ]:
```
def koszon(vezeteknev, keresztnev):
    print(f"Hello {vezeteknev} {keresztnev}!")


koszon(keresztnev="Balázs", vezeteknev="Kiss")
```

1.4.4 Pozíció szerinti megadás előírása

Ha szeretnénk előírni, hogy a paramétereket csak pozíciójuk sorrendjébe lehessen megadni, akkor a paraméterek után a `, /` szimbólummal jelölhetjük. Ilyenkor ezek a paraméterek nem használhatók a név szerinti (`kulcs = érték`) megadással.

[ ]:
```
def koszon(vezeteknev, keresztnev, /):
    print(f"Hello {vezeteknev} {keresztnev}!")


# koszon(keresztnev = "Balázs", vezeteknev = "Kiss" ) # nincs engedélyezve
koszon("Kiss", "Balázs")  # csak a pozíció alapján tölthető ki
```

1.4.5 Név szerinti megadás előírása

Ha azonban azt szeretnénk előírni, hogy csak a név szerinti (`kulcs = érték`) megadást lehessen használni, akkor a paraméterek elött ezzel a `* ,` szimbólummal jelölhetjük.
Ilyen esetben az alapértelmezett értékekre már nem vonatkozik a jobbról történő kitöltés megszorítása, bármelyik név szerinti paraméter kaphat alapértelmezett értéket.

[ ]:
```
def koszon(*, vezeteknev="Nagy", keresztnev):
    print(f"Hello {vezeteknev} {keresztnev}!")


# koszon("Kiss", "Balázs") # nincs engedélyezve
koszon(keresztnev="Balázs", vezeteknev="Kiss")  # csak a kulcs alapján tölthető␣
↪ki
```

1.4.6 Pozíció és név szerinti megadás kombinálása

A kettő megadási mód kombinálható előírt pozíció szerinti argumentumokra és kötelezően név szerint megadható argumentumokra. A kettő jelölés közötti argumentumok bármelyik módszer szerint megadhatók.

[ ]:
```
def szamol_fuggveny(a, b, /, c, *, d=6, e):
    print(a + b + c + d + e)
```

```
szamol_fuggveny(5, 6, 10, e=8)
szamol_fuggveny(5, 6, e=8, c=10)
```

1.4.7 Változó elemszámú argumentumlista feldolgozása

Ha nem tudjuk előre, hogy a függvény mennyi értékkel megadott argumentumot fog kapni híváskor (pl. a `print()` függvény), akkor az egyik paraméter elé írjunk `*` jelet, ezzel jelezve, hogy ez a paraméter több értéket is kaphat, amit egy tuple elemeiként érhetünk el. Az ezt követő paraméterek már csak név alapján kaphatnak értéket.

[ ]:
```
def szamol_fuggveny(*arg, oszto):
    print(type(arg))
    print(f"Összeg: {sum(arg)}")
    print(f"Normál: {sum(arg)/oszto}")


szamol_fuggveny(3, 4, 5, oszto=3)
```

1.4.8 Változó kulcsszámú argumentumlista feldolgozása

Ha nem tudjuk előre, hogy a függvény mennyi kulccsal megadott argumentumot fog kapni híváskor, akkor az utolsó paraméter elé írjunk `**` jelet, ezzel jelezve, hogy ez a paraméter több értéket is kaphat, amit egy szótár elemeiként érhetünk el.

[ ]:
```
def nev_fuggveny(**karg):
    print(type(karg))
    if "keresztnev" in karg:  # megvizsgáljuk, hogy van-e keresztvény kulcsú␣
 ↪argumentum
        print("Keresztnév: " + karg["keresztnev"])


nev_fuggveny(keresztnev="Balázs", vezeteknev="Kiss")
```

1.5 Változók hatóköre a függvényen belül

1.5.1 Lokális változó

A függvénytörzsben létrehozott változó lokálisan jön létre, ami azt jelenti, hogy csak a függvényen belül létezik, és más függvények vagy a program más részei nem férhetnek hozzá.

[ ]:
```
def minta_fuggveny():
    lokalis_valtozo = 10  # függvényen belül létrehozott változó, lokális
    print(f"Lokális változó a függvényben: {lokalis_valtozo}")


minta_fuggveny()
# print(lokalis_valtozo) # Hiba, a lokalis_valtozo nem elérhető itt
```

1.5.2 Globális változó

A globális változót a függvényen kívül azt megelőzően kell definiálni. A globális változó olyan változó, amely mindenhol elérhető a programban, így a függvényen belül is hozzáférhető, de csak olvasásra. Ha a függvényen belül értéket adunk neki, akkor lokális változóként létre fog jönni egy másik változó, így a globális változó nem módosul.

[ ]:
```
globalis_valtozo = 5  # globális változó


def minta_fuggveny():
    lokalis_valtozo = 10  # függvényen belül létrehozott változó, lokális
    print(f"Lokális változó a függvényben: {lokalis_valtozo}")
    # globalis_valtozo = 50 # Ez egy lokális változó
    print(f"Globális változó a függvényben: {globalis_valtozo}")


minta_fuggveny()
print(f"Globális változó: {globalis_valtozo}")
globalis_valtozo = 30
minta_fuggveny()
print(f"Globális változó: {globalis_valtozo}")
```

Ha egy függvényen belül módosítani szeretnénk egy globális változót, akkor az első használat elött egy `global` kulcsszóval jelöljük azt, így nem lokálisként fogja használni.

[ ]:
```
globalis_valtozo = 5  # globális változó


def minta_fuggveny():
    global globalis_valtozo  # Ez jelöli, hogy globálisként lehet erre a névre␣
 ↪hivatkozni
    print(f"Globális változó a függvényben: {globalis_valtozo}")
    globalis_valtozo = 50  # Ez már nem lokális
    print(f"Globális változó a függvényben: {globalis_valtozo}")


minta_fuggveny()
print(f"Globális változó: {globalis_valtozo}")
```

Fontos, hogy a globális változók használata néha nehezen karbantartható kódot eredményezhet, mivel a változók állapotát nehéz követni. Ajánlott a változókat a lehető legkisebb szintű hatókörön belül definiálni és használni.
Elkerülendő a globális változók túlzott használata, mivel könnyen okozhatnak problémákat, például függvények közötti függőségek növelését és hibakeresési nehézségeket. Ideális esetben inkább paramétereket kell átadni és visszatérési értékeket kell használni a függvények közötti információátadásra.

2 Gyakorló feladatok

1. Feladat

   Készítsen egy olyan függvényt, amely kiszámolja a `a*log(x^2)+b^(3/2)=0` egyenlethez tartozó x értéket, ha `a` és `b` bemenő paraméterek.

[ ]:
```python
import math


def fgv(a, b):
    if a == 0 or b < 0:
        return None
    x = math.sqrt(math.exp((-(b ** (3 / 2))) / a))
    return x


print(fgv(-1, 2))
```

2. Feladat

   Készítsen egy olyan függvényt, amely egy lista tartalmát kimenti egy csv fájlba, amelynek első oszlopában a sorszám, a második oszlopában a lista eleme szerepeljen.

[ ]:
```python
import os

def kiir(lista, filenev):
    if not os.path.exists("tmp"):
        os.mkdir("tmp")
    with open("./tmp/"+filenev, "w", encoding="utf-8") as f:
        for index, elem in enumerate(lista, 1):
            f.write(f"{index};{elem}\n")


listam = ["alma", "körte", "banán"]
kiir(listam, "gyumolcsok.csv")
```

3. Feladat

   Egy szótárban szereplő termékek és hozzájuk tartozó mennyiségek (kg) adatait írjuk ki egy csv fájlba úgy, hogy az elemek pontosvesszővel legyenek elválasztva és a mennyiségek pedig tizedesvesszővel jelenjenek meg.

[ ]:
```python
import os

def kiir(szotar, filenev):
    if not os.path.exists("tmp"):
        os.mkdir("tmp")
    with open("./tmp/"+filenev, "w") as f:
        for kulcs, ertek in szotar.items():
```

```
            f.write(f"{kulcs};{str(ertek).replace(".",",")}\n")

bevasarlo_lista = {"alma":5.2, "korte":1.2, "banan":3.5}

kiir(bevasarlo_lista,"bevasarlo_lista.csv")
```

4. Feladat

   Írja ki egy listában szereplő gyümölcsök neve elé a határozott névelőt egy külön fájlba. Mindezt külön sorokba.

[ ]:
```
import os

def nevelo(lista:list, filenev:str = "valami.txt"):
    '''Egy listában megadott gyümökcsnevek elé kiteszi a magánhangzókat'''
    maganhangzo = "aáeéiíoóöőuúüű"
    if not os.path.exists("tmp"):
        os.mkdir("tmp")
    with open("./tmp/"+filenev, "w", encoding="utf-8") as f:
        for elem in lista:
            if elem[0].lower() in maganhangzo:
                f.write(f"az {elem.lower()}\n")
            else:
                f.write(f"a {elem.lower()}\n")


gyumolcsok = ["alma", "körte", "banán", "Eper"]
nevelo(gyumolcsok, "gyumolcsok.txt")
nevelo(gyumolcsok)
nevelo(filenev="gyumolcsok.txt", lista=gyumolcsok)
nevelo("alma","valami.txt") # rossza paraméterrekkel történő függvényhívás,␣
 ↪mégis lefut
```

5. Feladat

   Készítsen egy olyan függvényt, amely 2db 3 dimenziós vektor keresztszorzatát képes kiszámolni. A vektor egy 3 elemű (x,y,z) lista legyen.

$$V = \frac{4R^3\pi}{3}$$

[ ]:
```
def kereszt(A, B):
    k = [0] * 3
    k[0] = A[1] * B[2] - A[2] * B[1]
    k[1] = A[2] * B[0] - A[0] * B[2]
    k[2] = A[0] * B[1] - A[1] * B[0]
    return k
```

```
v1 = [4, -2, 1]
v2 = [1, -1, 3]

print(kereszt(v1, v2))
```

6. Feladat

   Készítsen egy olyan függvényt, amely meghatározza egy gömb tömegét és térfogatát, ha ismerjük a gömb átmérőjét és anyagának sűrűségét.

   $$V = \frac{4R^3\pi}{3}$$

   Határozza meg a 70mm átmérőjű gömb tömegét, ha az anyaga vas! Mekkora a tömege egy ugyanekkora aluminium gömbnek?

[ ]:
```
import math


def gomb(atmero, suruseg):
    V = (4 * (atmero / 2) ** 3 * math.pi) / 3
    m = V * suruseg
    return (V, m)


d = 70          # mm
vas_ro = 7870   # kg/m^3
alu_ro = 2700   # kg/m^3

terfogat, tomeg = gomb(d/1000, vas_ro)
print(f"A gömb térfogata: {terfogat:.2e} m^3")
print(f"A gömb tömege, ha az anyaga vas: {tomeg:.2f} kg")
_, tomeg = gomb(d/1000, alu_ro)
print(f"A gömb tömege, ha az anyaga alu: {tomeg:.2f} kg")
```

7. Feladat

   Írjon egy függvényt, amely ellenőrzi, hogy egy adott szám prímszám-e.

[ ]:
```
import math

def isprime(szam):

    szam = abs(szam)

    if szam == 0 or szam == 1:
        return False

    for i in range(2, int(math.sqrt(szam))):
        if (szam % i) == 0:
```

```
            return False

    return True


print(isprime(42))
print(isprime(13))
```

8. Feladat

   Írjon egy függvényt, amely meghatározza egy adott év szökőév-e vagy sem. Szökőév minden negyedik, nem szökőév minden századik, mégis az minden 400-adik.

[ ]:
```
def szokoev(ev):
    if ev % 400 == 0:
        return True
    if ev % 4 == 0 and ev % 100 != 0:
        return True
    return False

print(szokoev(2000))
print(szokoev(2013))
```

9. Feladat

   Írjon egy függvényt, amely ellenőrzi, hogy egy szó vagy mondat palindrom-e (azaz ugyanaz, ha visszafelé olvasva). Működjön a függvény számokkal is.

[ ]:
```
def isPalindrome(s):
    s = str(s)
    return s == s[::-1]

print(isPalindrome("almamla"))
print(isPalindrome(12321))
```

10. Feladat

    Írjon egy függvényt, amely visszaadja két szám közös osztóinak listáját.

[ ]:
```
def kozos(a, b):
    l = []
    for i in range(2, min(a, b)):
        if a % i == 0 and b % i == 0:
            l.append(i)
    return l

print(kozos(40, 140))
```

11. Feladat

    Írjon egy függvényt, ami kiszámolja egy adott háromszög területét a megadott oldalak

hosszai alapján (Hérón képlet)!

[ ]:
```python
def Heron(a, b, c):
    s = (a + b + c) / 2
    terulet = math.sqrt(s * (s - a) * (s - b) * (s - c))
    kerulet = a + b + c
    return (kerulet, terulet)

kerulet, terulet = Heron(3, 4, 5)
print("Kerület:", kerulet)
print("Terület:", terulet)
```