

#1-----

Feladat: Első karakter

Írj egy függvényt `elso_karakter` néven amely visszatér az adott string első karakterével!

Természetesen, íme egy példa Python függvényre, amely visszatér az adott string első karakterével:

python

 Copy code

```
def elso_karakter(string):  
    return string[0]
```

Ez a függvény egy egyszerű indexelési műveletet használ, hogy megszerezze az első karaktert az adott stringből, majd visszatér ezzel az értékkel. A függvény hívása így néz ki:

python

 Copy code

```
elso_karakter("Hello World!")  
# eredmény: 'H'
```

#2-----

Feladat: Utolsó karakter

Írj egy függvényt utolso_karakter néven amely visszatér az adott string utolsó karakterével!

Természetesen, íme egy példa Python függvényre, amely visszaadja az adott string utolsó karakterét:

python

 Copy code

```
def utolso_karakter(string):
    return string[-1]
```

Ez a függvény szintén egy egyszerű indexelési műveletet használ, hogy megszerezze az utolsó karaktert az adott stringből, majd visszatér ezzel az értékkel. A függvény hívása így néz ki:

python

 Copy code

```
utolso_karakter("Hello World!")
# eredmény: '!'
```

```
# Feladat: Legkisebb szám a listában. [Programozási téTEL: Minimum kiválasztás]
# Készíts függvényt legkisebb néven, amely visszatér egy számokat tartalmazó lista legkisebb
számával.
# Figyelem! A feladat megoldása során nem használhatod a min függvényt!
```

Kiváló, hogy programozási tételeket is szeretnél bevonni a feladatokba! Itt van egy példa Python függvényre, amely visszaadja a számokat tartalmazó lista legkisebb elemét a Minimum kiválasztás algoritmus segítségével:

```
python
```

 Copy code

```
def legkisebb(lista):
    min_elem = lista[0]
    for elem in lista:
        if elem < min_elem:
            min_elem = elem
    return min_elem
```

A függvény létrehoz egy `min_elem` változót, amelyet inicializál az első listaelemmel. Ezután végigiterál a lista összes többi elemén, és ha talál egy olyan elemet, amely kisebb, mint a `min_elem`, akkor `min_elem` értékét az adott elemre állítja be. Végül a függvény visszatér a `min_elem` értékével.

Példa a függvény használatára:

```
python
```

 Copy code

```
lista = [4, 2, 8, 1, 5]
legkisebb_elem = legkisebb(lista)
print(legkisebb_elem) # eredmény: 1
```

```
# Feladat: Legnagyobb szám a listában. [Programozási téTEL: Maximum kiválasztás]
# Készíts függvényt legnagyobb néven, amely visszatér egy számokat tartalmazó lista
legnagyobb számával.
# Figyelem! A feladat megoldása során nem használhatod a max függvényt!
```

Nagyszerű, hogy egy másik programozási téTEL is választottál! Itt van egy példa Python függvényre, amely visszaadja a számokat tartalmazó lista legnagyobb elemét a Maximum kiválasztás algoritmus segítségével:

python
Copy code

```
def legnagyobb(lista):
    max_elem = lista[0]
    for elem in lista:
        if elem > max_elem:
            max_elem = elem
    return max_elem
```

A függvény ugyanúgy kezdi, mint az előző feladatnál a legkisebb elem meghatározásánál, tehát létrehoz egy `max_elem` változót az első listaelemmel. Ezután végigiterál a lista összes többi elemén, és ha talál egy olyan elemet, amely nagyobb, mint a `max_elem`, akkor `max_elem` értékét az adott elemre állítja be. Végül a függvény visszatér a `max_elem` értékével.

Példa a függvény használatára:

python
Copy code

```
lista = [4, 2, 8, 1, 5]
legnagyobb_elem = legnagyobb(lista)
print(legnagyobb_elem) # eredmény: 8
```

#5-----

Feladat: Listaban levő számok összege. [Programozási téTEL: Összegzés]

Készíts függvényt osszeg néven, amely visszatér egy számokat tartalmazó lista számainak összegével.

Figyelem! A feladat megoldása során nem használhatod a sum függvényt!

Ez is egy nagyszerű programozási téTEL, és egy jó gyakorlat lehet, ha megpróbáljuk implementálni a saját összegző függvényünket. Itt van egy példa Python függvényre, amely visszaadja a számokat tartalmazó lista összegét az Összegzés algoritmus segítségével:

```
python Copy code
def osszeg(lista):
    summa = 0
    for elem in lista:
        summa += elem
    return summa
```

A függvény létrehoz egy `summa` változót, amely az összeget fogja tárolni, és 0-ra állítja azt. Ezután végigiterál a lista minden elemén, és az `elem` értékét hozzáadja a `summa` értékéhez. Végül a függvény visszatér a `summa` értékével.

Példa a függvény használatára:

```
python Copy code
lista = [4, 2, 8, 1, 5]
osszeg = osszeg(lista)
print(osszeg) # eredmény: 20
```

#6-----

Feladat: Listában levő számok szorzata.

Készíts függvényt szorzat néven, amely visszatér egy számokat tartalmazó lista számainak szorzatával.

Ez egy másik nagyon hasznos függvény lehet, amely kiszámolja egy lista számainak szorzatát.

Itt van egy példa Python függvényre, amely ezt megteszi:

python

 Copy code

```
def szorzat(lista):
    szorzat = 1
    for elem in lista:
        szorzat *= elem
    return szorzat
```

A függvény létrehoz egy `szorzat` változót, amely az összes szám szorzatát fogja tárolni, és 1-re állítja azt. Ezután végigiterál a lista minden elemén, és a `szorzat` értékével megszorozza az `elem` értékét. Végül a függvény visszatér a `szorzat` értékével.

Példa a függvény használatára:

python

 Copy code

```
lista = [4, 2, 8, 1, 5]
szorzat = szorzat(lista)
print(szorzat) # eredmény: 320
```

Feladat: Páros számok száma a listában. [Programozási téTEL: Megszámolás]
Készíts függvényt parosok_szama néven, amely visszatér egy számokat tartalmazó lista páros számainak számával.

Az alábbi Python kóddal megoldható a feladat:

python

 Copy code

```
def parosok_szama(szamok):
    """
    Visszaadja a megadott számokat tartalmazó listában található páros számok számát.

    Args:
        szamok: A számokat tartalmazó lista

    Returns:
        A páros számok száma
    """
    parosok = 0
    for szam in szamok:
        if szam % 2 == 0:
            parosok += 1
    return parosok
```

A függvénynek egy számokat tartalmazó listát kell kapnia argumentumként, majd végigmegy a listán és minden páros számot megszámol. A megszámolt páros számok számát visszaadja a függvény.

Példa használat:

python

 Copy code

```
>>> szamok = [1, 2, 3, 4, 5, 6]
>>> parosok_szama(szamok)
3
```

Feladat: Páratlan számok száma a listában. [Programozási téTEL: Megszámolás]
Készíts függvényt paratlanok_szama néven, amely visszatér egy számokat tartalmazó lista páros számainak számával.

Az alábbi Python kóddal megoldható a feladat:

python

 Copy code

```
def paratlanok_szama(szamok):
    """
    Visszaadja a megadott számokat tartalmazó listában található páratlan számokat.

    Args:
        szamok: A számokat tartalmazó lista

    Returns:
        A páratlan számok száma
    """
    paratlanok = 0
    for szam in szamok:
        if szam % 2 == 1:
            paratlanok += 1
    return paratlanok
```

A függvénynek egy számokat tartalmazó listát kell kapnia argumentumként, majd végigmegy a listán és minden páratlan számot megszámol. A megszámolt páratlan számok számát visszaadja a függvény.

Példa használat:

python

 Copy code

```
>>> szamok = [1, 2, 3, 4, 5, 6]
>>> paratlanok_szama(szamok)
3
```

#9-----

Feladat: Pozitív számok száma a listában. [Programozási tétel: Megszámolás]
Készíts függvényt pozitivok_szama néven, amely visszatér egy számokat tartalmazó lista pozitív számainak számával.

Az alábbi Python kóddal megoldható a feladat:

python

 Copy code

```
def pozitivok_szama(szamok):
    """
    Visszaadja a megadott számokat tartalmazó listában található pozitív számokat.

    Args:
        szamok: A számokat tartalmazó lista

    Returns:
        A pozitív számok száma
    """
    pozitivok = 0
    for szam in szamok:
        if szam > 0:
            pozitivok += 1
    return pozitivok
```

A függvénynek egy számokat tartalmazó listát kell kapnia argumentumként, majd végigmegy a listán és minden pozitív számot megszámol. A megszámolt pozitív számok számát visszaadja a függvény.

Példa használat:

python

 Copy code

```
>>> szamok = [-3, 0, 5, -1, 10]
>>> pozitivok_szama(szamok)
2
```

#10-----

Feladat: Negatív számok száma a listában. [Programozási téTEL: Megszámolás]
Készíts függvényt negativok_szama néven, amely visszatér egy számokat tartalmazó lista negatív számainak számával.

Az alábbi Python kóddal megoldható a feladat:

python

 Copy code

```
def negativok_szama(szamok):
    """
    Visszaadja a megadott számokat tartalmazó listában található negatív számokat.

    Args:
        szamok: A számokat tartalmazó lista

    Returns:
        A negatív számok száma
    """
    negativok = 0
    for szam in szamok:
        if szam < 0:
            negativok += 1
    return negativok
```

A függvénynek egy számokat tartalmazó listát kell kapnia argumentumként, majd végigmegy a listán és minden negatív számot megszámol. A megszámolt negatív számok számát visszaadja a függvény.

Példa használat:

python

 Copy code

```
>>> szamok = [-3, 0, 5, -1, 10]
>>> negativok_szama(szamok)
2
```

#11-----

```
# Feladat: Benne van a szám a listában? [Programozási téTEL: Eldöntés]  
# Készíts függvényt benne_van_a_listaban néven, amelynek első paramétere egy számokat  
tartalmazó lista, a második paramétere egy szám.  
# A visszatérési érték True, ha a szám benne van a listában.  
# A visszatérési érték False, ha a szám nincs benne a listában.
```

Az alábbi Python kóddal megoldható a feladat:

python

 Copy code

```
def benne_van_a_listaban(szamok, keresett_szam):  
    """  
    Meghatározza, hogy a megadott számokat tartalmazó listában benne van-e a keresett szám.  
  
    Args:  
        szamok: A számokat tartalmazó lista  
        keresett_szam: A keresett szám  
  
    Returns:  
        True, ha a keresett szám benne van a listában, False egyébként  
    """  
    for szam in szamok:  
        if szam == keresett_szam:  
            return True  
    return False
```

A függvénynek egy számokat tartalmazó listát és egy keresett számot kell kapnia argumentumként. A függvény végigmegy a listán, és ha talál olyan számot, amely megegyezik a keresett számmal, akkor visszatér a True értékkel. Ha végigment a listán, és nem talált olyan számot, amely megegyezik a keresett számmal, akkor a függvény False értékkel tér vissza.

Példa használat:

python

 Copy code

```
>>> szamok = [1, 3, 5, 7, 9]  
>>> benne_van_a_listaban(szamok, 5)  
True  
>>> benne_van_a_listaban(szamok, 2)  
False
```

Az előző példában a `szamok` lista tartalmazza az 5-ös számot, ezért a `benne_van_a_listaban` függvény True értékkel tér vissza, ha az 5-öst keresi a függvény. Ha a 2-est keresi, amely nincs a listában, akkor a függvény False értékkel tér vissza.

#12-----

```
# Feladat: Benne van a betű a stringben? [Programozási téTEL: Eldöntés]
# Készíts függvényt benne_van_a_stringben néven, amelynek első paramétere egy string, a
# második paramétere egy betű.
# A visszatérési érték True, ha a betű benne van a stringben.
# A visszatérési érték False, ha a betű nincs benne a stringben.
```

Az alábbi Python kóddal megoldható a feladat:

```
python Copy code
def benne_van_a_stringben(string, betu):
    """
    Meghatározza, hogy a megadott stringben benne van-e a keresett betű.

    Args:
        string: A string, amiben keresünk
        betu: A keresett betű

    Returns:
        True, ha a keresett betű benne van a stringben, False egyébként
    """
    for karakter in string:
        if karakter == betu:
            return True
    return False
```

A függvénynek egy stringet és egy keresett betűt kell kapnia argumentumként. A függvény végigmegy a string karakterein, és ha talál olyan karaktert, amely megegyezik a keresett betűvel, akkor visszatér a True értékkel. Ha végigment a stringen, és nem talált olyan karaktert, amely megegyezik a keresett betűvel, akkor a függvény False értékkel tér vissza.

Példa használat:

```
python Copy code
>>> string = "hello world"
>>> benne_van_a_stringben(string, "l")
True
>>> benne_van_a_stringben(string, "z")
False
```

Az előző példában a `string` string tartalmazza az "l" betűt, ezért a `benne_van_a_stringben` függvény True értékkel tér vissza, ha az "l" betűt keresi a függvény. Ha a "z" betűt keresi, amely nincs a stringben, akkor a függvény False értékkel tér vissza.

#13-----

```
# Feladat: Hányadik a listában? [Programozási téTEL: Kiválasztás]
#       A listában a szám garantáltan megtalálható, nem kell vizsgálni a meglétét.
# Készíts függvényt hanyadik_a_listaban néven, amelynek első paramétere egy számokat
# tartalmazó lista, a második paramétere egy szám.
# A visszatérési érték a paraméterként megadott szám első előfordulási helye a listában.
```

Az alábbi függvény megoldja a feladatot:

python

 Copy code

```
def hanyadik_a_listaban(lista, szam):
    for i in range(len(lista)):
        if lista[i] == szam:
            return i+1
```

A függvény végigmegy a listán egy `for` ciklussal, és az első találat után visszatér a találat indexével, melyet a `return` utasítás ad vissza. Az indexet 1-gyel növeljük, mivel az feladat szerint a lista első elemének helye 1.

#14-----

```
# Feladat: Hányadik a stringben? [Programozási téTEL: Kiválasztás]
#       A betű garantáltan megtalálható a stringben, nem kell vizsgálnunk a meglétét.
# Készíts függvényt hanyadik_a_stringben néven, amelynek első paramétere egy string, a
# második paramétere egy betű.
# A visszatérési érték a paraméterként megadott betű első előfordulási helye a stringben.
```

Az alábbiakban látható a megoldás Python nyelven:

python

 Copy code

```
def hanyadik_a_stringben(string, betu):
    for i in range(len(string)):
        if string[i] == betu:
            return i
```

A fenti függvény egy egyszerű kiválasztó algoritmust használ, amely végigmegy a string karakterein, és az első olyan karakter esetén, amely megegyezik a paraméterként megadott betűvel, visszatér az indexével. Ha az adott betű nem található a stringben, akkor a függvény nem tér vissza semmilyen értékkel.

Példa használat:

python

 Copy code

```
string = "almafa"
betu = "a"
print(hanyadik_a_stringben(string, betu)) # kimenet: 0
```

#14-----

```
# Feladat: Hányadik a stringben? [Programozási téTEL: Kiválasztás]
#       A betű garantáltan megtalálható a stringben, nem kell vizsgálnunk a meglétét.
# Készíts függvényt hanyadik_a_stringben néven, amelynek első paramétere egy string, a
# második paramétere egy betű. range() függvény nélkül
# A visszatérési érték a paraméterként megadott betű első előfordulási helye a stringben.
```

Az alábbiakban látható a megoldás Python nyelven, a `range()` függvény nélkül:

python

 Copy code

```
def hanyadik_a_stringben(string, betu):
    i = 0
    while i < len(string):
        if string[i] == betu:
            return i
        i += 1
```

Ez a függvény is egy egyszerű kiválasztó algoritmust használ. Először az `i` változót inicializáljuk 0-ra, majd egy while ciklussal végigmegyünk a string karakterein. minden iterációban ellenőrizzük, hogy az aktuális karakter egyezik-e a paraméterként megadott betűvel. Ha igen, visszatérünk az indexével. Ha az adott betű nem található a stringben, akkor a függvény nem tér vissza semmilyen értékkel.

Példa használat:

python

 Copy code

```
string = "almafa"
betu = "a"
print(hanyadik_a_stringben(string, betu)) # kimenet: 0
```

#14-----

```
# Feladat: Hányadik a stringben? [Programozási téTEL: Kiválasztás]
#       A betű garantáltan megtalálható a stringben, nem kell vizsgálnunk a meglétét.
# Készíts függvényt hanyadik_a_stringben néven, amelynek első paramétere egy string, a
# második paramétere egy betű.
# range() függvény nélkül, while nélkül.
# A visszatérési érték a paraméterként megadott betű első előfordulási helye a stringben.
```

Az alábbiakban látható a megoldás Python nyelven, a `range()` és a while ciklus nélkül:

python

 Copy code

```
def hanyadik_a_stringben(string, betu):
    return string.index(betu)
```

A fenti függvény az `index()` metódust használja a string objektumon. Ez a metódus visszaadja az első olyan indexet, ahol a megadott érték megtalálható a stringben. Ha az adott betű nem található a stringben, akkor a metódus `ValueError` kivételt dob, amit a függvény nem kezel, hanem továbbadja.

Példa használat:

python

 Copy code

```
string = "almafa"
betu = "a"

print(hanyadik_a_stringben(string, betu)) # kimenet: 0
```

#14-----

```
# Feladat: Hányadik a stringben? [Programozási téTEL: Kiválasztás]
#      A betű garantáltan megtalálható a stringben, nem kell vizsgálnunk a meglétét.
# Készíts függvényt hanyadik_a_stringben néven, amelynek első paramétere egy string, a
# második paramétere egy betű.
# range() függvény nélkül, while nélkül, index() nélkül.
# A visszatérési érték a paraméterként megadott betű első előfordulási helye a stringben.
```

Az alábbiakban látható a megoldás Python nyelven, a `range()`, a while ciklus és az `index()` metódus nélkül:

```
python Copy code
def hanyadik_a_stringben(string, betu):
    for i, c in enumerate(string):
        if c == betu:
            return i
```

A fenti függvény egy egyszerű kiválasztó algoritmust használ. Az `enumerate()` függvénnnyel együtt a `for` ciklus minden iterációjában az aktuális indexet és karaktert kérdezzük le a string-ből. Ha az aktuális karakter egyezik a paraméterként megadott betűvel, akkor az index értékét adjuk vissza. Ha az adott betű nem található a stringben, akkor a függvény nem tér vissza semmilyen értékkel.

Példa használat:

```
python Copy code
string = "almafa"
betu = "a"
print(hanyadik_a_stringben(string, betu)) # kimenet: 0
```

#15-----

```
# Feladat: Benne van-e a listában és hanyadik helyen? [Programozási téTEL: Keresés]
# Készíts függvényt kereses_a_listaban néven, amelynek első paramétere egy számokat
# tartalmazó lista, a második paramétere egy szám.
# Ha a szám benne van a listában, akkor a visszatérési érték a paraméterként megadott szám
# első előfordulási helye a listában.
# A visszatérési érték None, ha a szám nincs benne a listában.
```

Az alábbiakban látható a megoldás Python nyelven:

```
python Copy code

def kereses_a_listaban(lista, szam):
    for i, elem in enumerate(lista):
        if elem == szam:
            return i
    return None
```

A fenti függvény egy egyszerű lineáris keresést alkalmaz a listában. Az `enumerate()` függvénnyel együtt a `for` ciklus minden iterációjában az aktuális indexet és elemet kérdezzük le a listából. Ha az aktuális elem egyezik a paraméterként megadott számmal, akkor az index értékét adjuk vissza. Ha az adott szám nem található a listában, akkor a függvény `None` értékkel tér vissza.

Példa használat:

```
python Copy code

lista = [1, 2, 3, 4, 5, 6]
szam = 4
print(kereses_a_listaban(lista, szam)) # kimenet: 3

szam = 7
print(kereses_a_listaban(lista, szam)) # kimenet: None
```

#16-----

```
# Feladat: Hányadik a stringben? [Programozási téTEL: Keresés]
# Készíts függvényt kereses_a_stringben néven, amelynek első paramétere egy string, a
# második paramétere egy betű.
# A visszatérési érték a paraméterként megadott betű első előfordulási helye a stringben.
# A visszatérési érték None, ha a betű nincs benne a stringben.
```

Lehetőségünk van az előző feladatban bemutatott keresési algoritmusok alkalmazására, de ha egy adott betű pozícióját szeretnénk megtudni egy string-ben, akkor egyszerűbb megoldás lehet a következő:

```
python Copy code
def kereses_a_stringben(string, betu):
    if betu in string:
        return string.index(betu)
    else:
        return None
```

Ez a függvény az `'index()'` metódust használja, amely visszaadja az első előfordulási helyet a string-ben. Ha a betű nem található a string-ben, akkor a függvény `'None'`-t ad vissza.

#16-----

```
# Feladat: Hányadik a stringben? [Programozási téTEL: Keresés]
# Készíts függvényt kereses_a_stringben néven, amelynek első paramétere egy string, a
# második paramétere egy betű.
# A visszatérési érték a paraméterként megadott betű első előfordulási helye a stringben.
# A visszatérési érték None, ha a betű nincs benne a stringben.
# index() nem használható!
```

Egy megoldás a feladatra a következő:

python

 Copy code

```
def kereses_a_stringben(string, betu):
    i = 0
    while i < len(string):
        if string[i] == betu:
            return i
        i += 1
    return None
```

Ez a megoldás használ egy `while` ciklust, hogy végigiteráljon a string karakterein, és a `==` operátorral ellenőrizze, hogy az aktuális karakter egyezik-e a keresett betűvel. Ha megtalálta az első előfordulást, akkor az indexsel tér vissza, ha nem találta meg, akkor pedig `None` értékkel tér vissza. Az `index()` függvényt itt sem használtuk.

#16-----

```
# Feladat: Hányadik a stringben? [Programozási téTEL: Keresés]
# Készíts függvényt kereses_a_stringben néven, amelynek első paramétere egy string, a
# második paramétere egy betű.
# A visszatérési érték a paraméterként megadott betű első előfordulási helye a stringben.
# A visszatérési érték None, ha a betű nincs benne a stringben.
# index() nem használható!
```

Az alábbi megoldás használja a `enumerate()` függvényt, hogy bejárja a karaktereket a stringen és ellenőrizze, hogy megegyeznek-e a keresett karakterrel. Ha igen, akkor visszaadja a karakter indexét, ha pedig nem találja meg a karaktert, akkor None értéket ad vissza.

python

 Copy code

```
def kereses_a_stringben(string, char):
    for i, c in enumerate(string):
        if c == char:
            return i
    return None
```

Példa használat:

python

 Copy code

```
>>> kereses_a_stringben('almafa', 'm')
2
>>> kereses_a_stringben('banán', 'x')
None
```

#17-----

Feladat: Páros számok kiválogatása egy listából. [Programozási téTEL: Kiválogatás]
Készíts függvényt parosok_kivalogatasa néven, amely visszatér egy listával amely a
paraméterként átadott számokat tartalmazó lista páros számait tartalmazza.

Mutasd az megoldást.

python

 Copy code

```
def parosok_kivalogatasa(szamok):
    parosok = []
    for szam in szamok:
        if szam % 2 == 0:
            parosok.append(szam)
    return parosok
```

Ez a függvény végigmegy a kapott `szamok` listán és minden páros számot hozzáad a `parosok` listához. A páros számokat az osztás maradéka alapján ellenőrzi. A végén a `parosok` listát adja vissza.

#17-----

Feladat: Páros számok kiválogatása egy listából. [Programozási tétel: Kiválogatás]
Készíts függvényt parosok_kivalogatasa néven, amely visszatér egy listával amely a
paraméterként átadott számokat tartalmazó lista páros számait tartalmazza.
Megoldás List comprehension segítségével.

Egy lehetséges megoldás a feladatra a következő:

python

 Copy code

```
def parosok_kivalogatasa(szamok):  
    return [szam for szam in szamok if szam % 2 == 0]
```

Ez a függvény egy list comprehension-t használ, amely végigmegy a paraméterként kapott `szamok` listán, és kiválogatja azokat a számokat, amelyek párosak (`szam % 2 == 0`). Az így kiválogatott számokat egy új listába teszi, és ezt a listát adja vissza a függvény.

#18-----

```
# Feladat: Páratlan számok számok kiválogatása egy listából. [Programozási téTEL: Kiválogatás]  
# Készíts függvényt paratlanok_kivalogatasa néven, amely visszatér egy listával amely a  
paraméterként átadott számokat tartalmazó lista páratlan számait tartalmazza.
```

Az alábbiakban egy lehetséges megoldást mutatok list comprehension segítségével a feladatra:

python

 Copy code

```
def paratlanok_kivalogatasa(szamok):  
    return [szam for szam in szamok if szam % 2 == 1]
```

A függvény egy számokat tartalmazó listát vár paraméterként, majd egy új listával tér vissza, amely a páratlan számokat tartalmazza. A list comprehension használata során az eredeti lista minden elemét bejárjuk, és csak azokat tartjuk meg, amelyek maradékának osztása 2-vel 1-et ad (tehát páratlanok).

#19-----

Feladat: Pozitív számok kiválogatása egy listából. [Programozási téTEL: Kiválogatás]
Készíts függvényt pozitivok_kivalogatasa néven, amely visszatér egy listával amely a paraméterként átadott számokat tartalmazó lista pozitív számait tartalmazza.

Az alábbiakban egy lehetséges megoldás található a pozitív számok kiválogatására:

python

 Copy code

```
def pozitivok_kivalogatasa(szamok):  
    pozitivak = [szam for szam in szamok if szam > 0]  
    return pozitivak
```

A függvény egy list comprehension segítségével végigmegy a kapott számok listáján, és minden pozitív számot hozzáad egy új listához ('**pozitivak**'). Végül a függvény visszatér ezzel a listával, amely a pozitív számokat tartalmazza.

#20-----

Feladat: Negatív számok számok kiválogatása egy listából. [Programozási téTEL: Kiválogatás]
Készíts függvényt negativok_kivalogatasa néven, amely visszatér egy listával amely a paraméterként átadott számokat tartalmazó lista negatív számait tartalmazza.

Második megoldás List comprehension segítségével:

python

 Copy code

```
def negativok_kivalogatasa(szamok):  
    return [szam for szam in szamok if szam < 0]
```

Első megoldás ciklussal:

python

 Copy code

```
def negativok_kivalogatasa(szamok):  
    negativ_szamok = []  
    for szam in szamok:  
        if szam < 0:  
            negativ_szamok.append(szam)  
    return negativ_szamok
```

#21-----

```
# Feladat: Két halmaz metszete. [Programozási téTEL: Metszet]
# Készíts függvényt halmazok_metszete néven, amely két paraméterként átadott halmaz
azonos elemeivel mint halmazzal tér vissza.
# & operátorral.
```

Az alábbiakban látható a kért függvény Python nyelven:

python

 Copy code

```
def halmazok_metszete(halmaz1, halmaz2):
    return halmaz1 & halmaz2
```

A fenti kódrészlet definiál egy `halmazok_metszete` nevű függvényt, amely két halmazt kap paraméterként, és azok metszetét adja vissza. A `&` operátor a Python nyelvben a halmazmetszetet jelöli.

Például, ha az alábbi két halmazzal hívjuk meg a függvényt:

python

 Copy code

```
halmaz1 = {1, 2, 3, 4}
halmaz2 = {3, 4, 5, 6}
print(halmazok_metszete(halmaz1, halmaz2))
```

A kimenet:

```
{3, 4}
```

 Copy code

Mivel csak a 3 és 4 szerepel mindenkor halmazban, ezért ezek alkotják a metszetüket.

#21-----

Feladat: Két halmaz metszete. [Programozási téTEL: Metszet]
Készíts függvényt halmazok_metszete néven, amely két paraméterként átadott halmaz
azonos elemeivel mint halmazzal tér vissza.

Az alábbiakban látható egy implementáció, amely a Python beépített halmaz típusát használja:

python

 Copy code

```
def halmazok_metszete(halmazi, halmaz2):
    return halmazi.intersection(halmaz2)
```

Az `intersection()`` metódus a halmazok metszetét adja vissza. A függvény egyszerűen
meghívja ezt a metódust a két paraméterként átadott halmazra, és visszaadja az eredményt.

#21-----

Feladat: Két halmaz metszete. [Programozási tétel: Metszet]

Készíts függvényt halmazok_metszete néven, amely két paraméterként átadott halmaz azonos elemeivel mint halmazzal tér vissza.

intersection() nélkül!

Egy lehetőség a metszet kiszámolására az, hogy először átalakítjuk a két halmazt listává, majd bejárjuk az egyik listát, és minden olyan elemet hozzáadunk a végeredmény listához, amely megtalálható a másik listában is.

Példa implementáció:

python Copy code

```
def halmazok_metszete(halmaz1, halmaz2):
    lista1 = list(halmaz1)
    lista2 = list(halmaz2)
    eredmény = []
    for elem in lista1:
        if elem in lista2 and elem not in eredmény:
            eredmény.append(elem)
    return set(eredmény)
```

Az implementáció átalakítja a halmazokat listává a `list()` függvénnnyel, majd bejárja az első listát, és minden olyan elemet, amely megtalálható a második listában is, hozzáadja az eredmény listához. A `not in` feltétellel elkerüljük, hogy az eredményben többször szerepeljen ugyanaz az elem. Végül az eredményt visszaalakítjuk halmazzá a `set()` függvénnnyel.

#22-----

Feladat: Két lista metszete. [Programozási téTEL: Metszet]

Készíts függvényt listak_metszete néven, amely két paraméterként átadott lista azonos elemeivel mint listával tér vissza.

A megoldáshoz használhatjuk az előző feladatban bemutatott halmazok metszetét számoló függvényt, majd az eredmény halmzból előállítjuk a listát.

python

 Copy code

```
def listak_metszete(lista1, lista2):
    halmazi = set(lista1)
    halmaz2 = set(lista2)
    metszet_halmaz = halmazi & halmaz2
    return list(metszet_halmaz)
```

Példa:

python

 Copy code

```
>>> listak_metszete([1, 2, 3], [2, 3, 4])
[2, 3]
```

#23-----

Feladat: Két halmaz uniója. [Programozási téTEL: Unió]

Készíts függvényt halmazok_unioja néven, amely két paraméterként átadott halmaz minden elemeivel mint halmazzal tér vissza.

Az alábbiakban egy lehetséges megoldás található Python nyelven:

python

 Copy code

```
def halmazok_unioja(h1, h2):
    return h1.union(h2)
```

Ez a függvény az `'union()'` metódust használja a halmazok uniójának meghatározására, amelynek eredménye a két halmaz elemeinek összefűzése egy új halmazba. A függvény paraméterei két halmaz (`'h1'` és `'h2'`) és a visszatérési értéke is egy halmaz.

Feladat: Két halmaz uniója. [Programozási téTEL: Unió]

Készíts függvényt halmazok_unioja néven, amely két paraméterként átadott halmaz minden elemeivel mint halmazzal tér vissza.

union() nélkül!

A halmazok uniójának meghatározásához végig kell menni az egyik halmazon és minden elemet hozzá kell adni az eredmény halmazhoz. Ezután ugyanezt kell tenni a másik halmazzal is, de azokat az elemeket, amelyek már szerepelnek az eredmény halmazban, nem kell hozzáadni újra. Az így kapott eredmény halmaz lesz a két halmaz uniója.

Íme egy ilyen függvény implementációja:

python

 Copy code

```
def halmazok_unioja(h1, h2):
    eredmeny = set()
    for elem in h1:
        eredmeny.add(elem)
    for elem in h2:
        if elem not in eredmeny:
            eredmeny.add(elem)
    return eredmeny
```

Ez a függvény visszatér egy halmazzal, amely tartalmazza a két paraméterként átadott halmaz összes elemét. Az első ciklusban az összes elemet hozzáadjuk az eredmény halmazhoz. A második ciklusban pedig az összes elemet hozzáadjuk a második halmazból is, kivéve azokat, amelyek már szerepelnek az eredmény halmazban. Ezt az `if elem not in eredmeny` feltétellel érhetjük el.

#23-----

Feladat: Két halmaz uniója. [Programozási téTEL: UniÓ]

Készíts függvényt halmazok_unioja néven, amely két paraméterként átadott halmaz minden elemeivel mint halmazzal tér vissza.

| operátorral!

A `|` operátorral lehet két halmazt összefűzni, tehát az uniójukat létrehozni. Így a `halmazok_unioja` függvény implementációja a következőképpen nézhet ki:

python

 Copy code

```
def halmazok_unioja(h1, h2):
    return h1 | h2
```

Az `h1` és `h2` paraméterek a két halmaz, amelyek unióját szeretnénk létrehozni. Az `|` operátorral egyszerűen összeadjuk a két halmazt, és ezt a műveletet a függvény egyszerűen végrehajtja és visszatér az eredménnyel.

#24-----

Feladat: Listaban levő számok átlaga (számtani közepe).

Készíts függvényt lista_atlag néven, amely visszatér egy számokat tartalmazó lista számainak átlagával.

Itt egy példa implementáció a feladathoz:

python

 Copy code

```
def lista_atlag(lst):
    if not lst:
        return 0
    return sum(lst) / len(lst)
```

A függvény egyszerűen kiszámolja a lista elemeinek összegét, majd elosztja a lista hosszával, hogy megkapja az átlagot. Az első sorban ellenőrizzük, hogy a lista nem üres-e, hogy ne legyen osztás nullával hiba. Ha az üres, akkor visszaadunk egy nullát, de ez nem feltétlenül szükséges a feladat specifikációja szerint.

#25-----

Feladat: Faktoriális.

Készíts függvényt faktorialis néven, amely visszatér a paraméterként megkapott szám faktoriálisával.

Rekurzió nélkül!

Itt egy lehetséges megoldás, amely egy egyszerű ciklussal számolja ki a faktoriálist:

python

 Copy code

```
def faktorialis(n):
    fakt = 1
    for i in range(2, n+1):
        fakt *= i
    return fakt
```

Az függvényt a következőképpen használhatjuk:

python

 Copy code

```
print(faktorialis(5)) # kimenet: 120
```

#25-----

Feladat: Faktoriális.

Készíts függvényt faktoriális néven, amely visszatér a paraméterként megkapott szám faktoriálisával.

Íme egy példa implementáció a faktoriális függvénynek Pythonban:

python

 Copy code

```
def faktorialis(n):
    if n == 0:
        return 1
    else:
        return n * faktorialis(n-1)
```

Ez a függvény rekurzívan számolja ki a faktoriálist. Ha a bemeneti érték 0, akkor az eredmény 1, mert a 0 faktoriálisa 1. Ha a bemeneti érték nagyobb, akkor a faktoriális értéke a bemeneti érték és az egy kisebb faktoriálisa szorzata. Ez azért működik, mert a faktoriális definíciója szerint $n! = n * (n-1) * (n-2) * \dots * 1$.