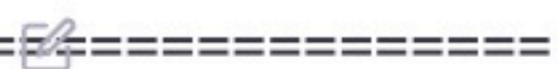


#01=====
Feladat: Karakterek száma a fájlban.
Írj egy függvényt karakterek_szama néven amely paraméterként egy fájlnevet kap és visszatér a fájlban levő karakterek számával. ('\n karakterekkel együtt'



Természetesen, íme egy Python függvény megoldás a problémára:



```
python  
  
def karakterek_szama(fajlnev):  
    try:  
        with open(fajlnev, 'r') as fajl:  
            szoveg = fajl.read()  
            karakterek_szama = len(szoveg)  
            return karakterek_szama  
    except FileNotFoundError:  
        print(f'Hiba: a(z) {fajlnev} fájl nem található.')
```

Copy code

Ez a függvény megnyitja a megadott fájlt olvasásra, beolvassa annak teljes tartalmát a `read()` metódus segítségével, majd visszaadja a karakterek számát, beleértve a sorvége karaktereket is.

Ha a fájl nem található, akkor a függvény `FileNotFoundException` kivételt dob és egy hibaüzenetet ír ki. Ez segít abban, hogy a felhasználók tudják, ha valami probléma van a fájl megnyitásával.

#01=====

```
# Feladat: Karakterek száma a fájlban.  
# Írj egy függvényt karakterek_szama néven amely paraméterként egy fájlnevet kap és visszatér a fájlban levő  
karakterek számával. ('\n karakterekkel együtt'  
# try nélkül!
```

Rendben, itt egy másik megoldás Pythonban, amely nem használja a `try` blokkot a hiba kezeléséhez:



```
python  
  
def karakterek_szama(fajlnev):  
    fajl = open(fajlnev, 'r')  
    szoveg = fajl.read()  
    fajl.close()  
    karakterek_szama = len(szoveg)  
    return karakterek_szama
```

Ez a függvény használja a `open()` beépített függvényt a fájl megnyitásához olvasásra, majd beolvassa annak teljes tartalmát a `read()` metódus segítségével. Ezután bezárja a fájlt a `close()` metódus segítségével, és visszaadja a karakterek számát, beleértve a sorvége karaktereket is.

Ez a megoldás ugyanazt a funkcionálitást biztosítja, mint a korábbi megoldás, azonban itt nincs hiba kezelés, így ha a fájl nem található vagy valamilyen más hiba történik, akkor a program el fog szakadni a hibaüzenet nélkül. Ennek elkerülése érdekében lehetőség van hiba kezelő mechanizmusok használatára a programban.

#02-----

Feladat: Szavak száma a fájlban.

Írj egy függvényt szavak_szama néven amely paraméterként egy fájlnevet kap és visszatér a fájlban levő szavak számával.

Itt egy Python függvény megoldás a feladatra:



python Copy code

```
def szavak_szama(fajlnev):
    with open(fajlnev, 'r') as fajl:
        szoveg = fajl.read()
        szavak = szoveg.split()
    return len(szavak)
```

Ez a függvény megnyitja a megadott fájlt olvasásra, beolvassa annak teljes tartalmát a `read()` metódus segítségével, majd a `split()` metódus segítségével darabolja szét a szöveget szavakra. A darabolás során a szóközök, tabulátorok és sorvége karakterek nem kerülnek be a szavak közé. A függvény visszatér a szavak számával, amelyet az `len()` függvénnyel határoz meg.

Ez a megoldás nem tartalmaz hiba kezelést a fájl megnyitásakor, mivel a `with` blokk automatikusan kezeli az ilyen hibákat, így a hibák automatikusan kezelésre kerülnek. A függvény csak akkor ad vissza helyes eredményt, ha a megadott fájl megtalálható és elérhető olvasásra.

#03-----

Feladat: Sorok száma a fájlban.

A sorok_szama(fname) függvény visszatér a fájlban levő sorok számával.

Itt egy Python függvény megoldás a feladatra:

python

 Copy code

```
def sorok_szama(fname):
    with open(fname, 'r') as fajl:
        sorok = fajl.readlines()
    return len(sorok)
```

Ez a függvény megnyitja a megadott fájlt olvasásra, majd beolvassa az összes sort a `readlines()` metódus segítségével. A függvény visszatér a sorok számával, amelyet az `len()` függvénnnyel határoz meg.

Ez a megoldás nem tartalmaz hiba kezelést a fájl megnyitásakor, mivel a `with` blokk automatikusan kezeli az ilyen hibákat, így a hibák automatikusan kezelésre kerülnek. A függvény csak akkor ad vissza helyes eredményt, ha a megadott fájl megtalálható és elérhető olvasásra.

#04-----

Feladat: r betük száma a fájlban.

Az r_betuk_szama(fname) függvény visszatér a fájlban levő 'r' betük számával.

Itt egy Python függvény megoldás a feladatra:

python

 Copy code

```
def r_betuk_szama(fname):
    with open(fname, 'r') as fajl:
        szoveg = fajl.read()
    return szoveg.count('r')
```

Ez a függvény megnyitja a megadott fájlt olvasásra, majd beolvassa annak teljes tartalmát a `read()` metódus segítségével. A függvény a `count()` metódus segítségével számolja meg az 'r' karakterek számát a szövegen, majd ezt a számot adja vissza.

Ez a megoldás nem tartalmaz hiba kezelést a fájl megnyitásakor, mivel a `with` blokk automatikusan kezeli az ilyen hibákat, így a hibák automatikusan kezelésre kerülnek. A függvény csak akkor ad vissza helyes eredményt, ha a megadott fájl megtalálható és elérhető olvasásra.

#05-----

Feladat: lorem szavak száma a fájlban.

5. A lorem_szavak_szama(fname) függvény visszatér a fájlban levő "lorem" szavak számával.

Itt egy Python függvény megoldás a feladatra:

python

 Copy code

```
def lorem_szavak_szama(fname):
    with open(fname, 'r') as fajl:
        szoveg = fajl.read()
        szavak = szoveg.split()
    return szavak.count('lorem')
```

Ez a függvény megnyitja a megadott fájlt olvasásra, majd beolvassa annak teljes tartalmát a `read()` metódus segítségével. A függvény a `split()` metódus segítségével szétválasztja a szöveget szavakra, majd a `count()` metódus segítségével számolja meg a 'lorem' szavak számát a szövegben, majd ezt a számot adja vissza.

Ez a megoldás nem tartalmaz hiba kezelést a fájl megnyitásakor, mivel a `with` blokk automatikusan kezeli az ilyen hibákat, így a hibák automatikusan kezelésre kerülnek. A függvény csak akkor ad vissza helyes eredményt, ha a megadott fájl megtalálható és elérhető olvasásra.

#06-----

Feladat: A leggyakoribb karakter a fájlban.

A leggyakoribb_karakter(fname) függvény visszatér a fájlban leggyakrabban előforduló karakterrel.

Itt egy Python függvény megoldás a feladatra:

python

 Copy code

```
def leggyakoribb_karakter(fname):
    with open(fname, 'r') as fajl:
        szoveg = fajl.read()
        karakterek = list(szoveg)
        leggyakoribb = max(set(karakterek), key=karakterek.count)
    return leggyakoribb
```

Ez a függvény megnyitja a megadott fájlt olvasásra, majd beolvassa annak teljes tartalmát a `read()` metódus segítségével. A függvény először létrehoz egy listát a szövegen található összes karakterről, majd meghatározza a leggyakoribb karaktert a `max()` és `count()` metódusok segítségével. A `set()` metódus segítségével először egy egyedi karaktereket tartalmazó halmazt hozunk létre, amelynek minden eleme csak egyszer szerepel a listában, majd a `max()` függvény segítségével kiválasztjuk a leggyakoribb karaktert a listából.

Ez a megoldás nem tartalmaz hiba kezelést a fájl megnyitásakor, mivel a `with` blokk automatikusan kezeli az ilyen hibákat, így a hibák automatikusan kezelésre kerülnek. A függvény csak akkor ad vissza helyes eredményt, ha a megadott fájl megtalálható és elérhető olvasásra.

#06-----

Feladat: A leggyakoribb karakter a fájlban.

A leggyakoribb_karakter(fname) függvény visszatér a fájlban leggyakrabban előforduló karakterrel.

Szótár adattípus használatával!

Itt egy másik megoldás a leggyakoribb karakter meghatározására, amely szótár adattípust használ:

python

 Copy code

```
def leggyakoribb_karakter(fname):
    with open(fname, 'r') as fajl:
        szoveg = fajl.read()
        karakterek = {}
        for karakter in szoveg:
            if karakter in karakterek:
                karakterek[karakter] += 1
            else:
                karakterek[karakter] = 1
    leggyakoribb = max(karakterek, key=karakterek.get)
    return leggyakoribb
```

Ez a függvény szótárat használ a karakterek előfordulási gyakoriságainak számolására. Először létrehozunk egy üres szótárat, majd végigmegyünk a szöveg összes karakterén és hozzáadjuk azokat a szótárhoz. Ha egy karakter már szerepel a szótárban, növeljük annak előfordulási gyakoriságát, különben hozzáadjuk az új kulcs-érték párt a szótárhoz, ahol a kulcs a karakter, az érték pedig 1. Végül meghatározzuk a leggyakoribb karaktert a `max()` függvény és a `get()` metódus segítségével.

Ez a megoldás ismét nem tartalmaz hiba kezelést a fájl megnyitásakor, mivel a `with` blokk automatikusan kezeli az ilyen hibákat, így a hibák automatikusan kezelésre kerülnek. A függvény csak akkor ad vissza helyes eredményt, ha a megadott fájl megtalálható és elérhető olvasásra.

#06-----

Feladat: A leggyakoribb karakter a fájlban.

A leggyakoribb_karakter(fname) függvény visszatér a fájlban leggyakrabban előforduló karakterrel.

get() használatával.

Egy lehetséges megoldás a `get()` metódus használatával a következőképpen nézhet ki:

python

 Copy code

```
def leggyakoribb_karakter(fname):
    with open(fname, 'r') as f:
        freq = {}
        for line in f:
            for char in line:
                freq[char] = freq.get(char, 0) + 1
    return max(freq, key=freq.get)
```

Az első sorban megnyitjuk a fájlt olvasásra. A `freq` változó egy üres szótárat inicializál, amelyben tároljuk a karakterek előfordulási gyakoriságát.

Ezután iterálunk végig a fájl sorain és karakterein, majd a `get()` metódus segítségével növeljük a karakterek előfordulási gyakoriságát a `freq` szótárban. Az `get()` metódus visszatéríti a megadott kulcshoz tartozó értéket a szótárban, vagy ha a kulcs nem található a szótárban, akkor az általunk megadott alapértéket.

Végül a `max()` függvényt használjuk a `freq` szótár kulcsainak maximum értékét meghatározni, majd a `key` paraméter segítségével visszatérünk a kulcsával, amely az a karakter lesz, amely a leggyakrabban fordul elő a fájlban.

#07-----

Feladat: A leghosszabb sor hossza a fájlban.

A leghosszabb_sor_hossza(fname) függvény visszatér a fájlban levő leghosszabb sor hosszával.

Az alábbiakban megadom a `leghosszabb_sor_hossza(fname)` függvény implementációját, amely visszatér a megadott fájlban található leghosszabb sor hosszával:

python

 Copy code

```
def leghosszabb_sor_hossza(fname):
    with open(fname, 'r') as f:
        max_hossz = 0
        for sor in f:
            hossz = len(sor)
            if hossz > max_hossz:
                max_hossz = hossz
    return max_hossz
```

Ez a függvény megnyitja a fájlt olvasásra, majd végigmegy a fájl sorain, és meghatározza a leghosszabb sort. A `len()` függvény segítségével számolja a sor hosszát, majd az `if` utasításban ellenőrzi, hogy ez a hossz nagyobb-e, mint a jelenlegi maximális hossz. Ha igen, akkor frissíti a maximális hosszt. A függvény visszatér a leghosszabb sor hosszával.

#08-----

```
# Feladat: Téglalap osztály definiálása. [Objektumorientált programozás]
# Hozz létre egy osztályt Teglalap néven.
# A Teglalap osztály lehetővé teszi a téglalap oldalhosszúságainak tárolását.
# A Teglalap osztály rendelkezik egy kerulet() nevű metódussal, amely az osztály segítségével
létrehozott objektum metódusaként visszaadja az adott objektum kerületét.
# A Teglalap osztály rendelkezik egy terulet() nevű metódussal, amely az osztály segítségével
létrehozott objektum metódusaként visszaadja az adott objektum területét.
```

Az alábbiakban látható a kért Teglalap osztály Python kódja:

```
python Copy code

class Teglalap:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def kerulet(self):
        return 2 * (self.a + self.b)

    def terulet(self):
        return self.a * self.b
```

Az osztály inicializálásakor (a `__init__()` metódusban) megadjuk a téglalap oldalhosszúságait ('a' és 'b'), majd a `kerulet()` és `terulet()` metódusokkal a téglalap kerületét és területét számolhatjuk ki. Az objektum saját attribútumai ('self.a' és 'self.b') tárolják az oldalhosszúságokat, amelyeket a metódusok használnak a számításokhoz.

Példa használat:

```
python Copy code

teglalap1 = Teglalap(5, 10)
print(teglalap1.kerulet()) # kimenet: 30
print(teglalap1.terulet()) # kimenet: 50

teglalap2 = Teglalap(3, 7)
print(teglalap2.kerulet()) # kimenet: 20
print(teglalap2.terulet()) # kimenet: 21
```

Az első példában egy 5×10 méretű téglalapot hozunk létre, majd kiszámítjuk annak kerületét és területét. A második példában egy 3×7 méretű téglalapot hozunk létre és számítunk neki kerületet és területet.

#09-----

Feladat: Négyzet osztály definiálása. [Objektumorientált programozás]
Hozz létre egy osztályt Negyzet néven.
A Negyzet osztály lehetővé teszi a negyzet oldalhosszúságának tárolását.
A Negyzet osztály rendelkezik egy kerulet() nevű metódussal, amely az osztály segítségével
létrehozott objektum metódusaként visszaadja az adott objektum kerületét.
A Negyzet osztály rendelkezik egy terulet() nevű metódussal, amely az osztály segítségével
létrehozott objektum metódusaként visszaadja az adott objektum területét.

Az alábbiakban látható a kért Negyzet osztály Python kódja:

```
python Copy code  
  
class Negyzet:  
    def __init__(self, a):  
        self.a = a  
  
    def kerulet(self):  
        return 4 * self.a  
  
    def terulet(self):  
        return self.a ** 2
```

Az osztály inicializálásakor (a `__init__()` metódusban) megadjuk a négyzet oldalhosszúságát (`a`), majd a `kerulet()` és `terulet()` metódusokkal a négyzet kerületét és területét számolhatjuk ki. Az objektum saját attribútuma (`self.a`) tárolja az oldalhosszúságot, amelyet a metódusok használnak a számításokhoz.

Példa használat:

```
python Copy code  
  
negyzet1 = Negyzet(5)  
print(negyzet1.kerulet()) # kimenet: 20  
print(negyzet1.terulet()) # kimenet: 25  
  
negyzet2 = Negyzet(7)  
print(negyzet2.kerulet()) # kimenet: 28  
print(negyzet2.terulet()) # kimenet: 49
```

Az első példában egy 5 egység oldalhosszúságú négyzetet hozunk létre, majd kiszámítjuk annak kerületét és területét. A második példában egy 7 egység oldalhosszúságú négyzetet hozunk létre és számítunk neki kerületet és területet.

#10-----

```
# Feladat: Kocka osztály definiálása. [Objektumorientált programozás]
# Hozz létre egy osztályt Kocka néven.
# A Kocka osztály lehetővé teszi a kocka oldalhosszúságának tárolását.
# A Kocka osztály rendelkezik egy terfogat() nevű metódussal, amely az osztály segítségével
létrehozott objektum metódusaként visszaadjja az adott objektum térfogatát.
# A Kocka osztály rendelkezik egy felszin() nevű metódussal, amely az osztály segítségével
létrehozott objektum metódusaként visszaadjja az adott objektum felszínét.
```

Az alábbiakban látható a kért Kocka osztály Python kódja:

```
python
Copy code

class Kocka:
    def __init__(self, a):
        self.a = a

    def terfogat(self):
        return self.a ** 3

    def felszin(self):
        return 6 * self.a ** 2
```

Az osztály inicializálásakor (a `__init__()` metódusban) megadjuk a kocka oldalhosszúságát ('a'), majd a `terfogat()` és `felszin()` metódusokkal a kocka térfogatát és felszínét számolhatjuk ki. Az objektum saját attribútuma ('self.a') tárolja az oldalhosszúságot, amelyet a metódusok használnak a számításokhoz.

Példa használat:

```
python
Copy code

kocka1 = Kocka(3)
print(kocka1.terfogat()) # kimenet: 27
print(kocka1.felszin()) # kimenet: 54

kocka2 = Kocka(5)
print(kocka2.terfogat()) # kimenet: 125
print(kocka2.felszin()) # kimenet: 150
```

Az első példában egy 3 egység oldalhosszúságú kockát hozunk létre, majd kiszámítjuk annak térfogatát és felszínét. A második példában egy 5 egység oldalhosszúságú kockát hozunk létre és számítunk neki térfogatot és felszínt.

#11-----

```
# Feladat: String fájlba írása
# Készíts függvényt string_fajlba néven, amely az első paraméterként kapott sztringet fájlba írja.
# A fájl nevét második paraméterként kapja meg a függvény.
```

Az alábbiakban látható a kért `string_fajlba` függvény Python kódja:

```
python Copy code
def string_fajlba(sztring, fajlnev):
    with open(fajlnev, 'w') as f:
        f.write(sztring)
```

A függvény a `with` kulcsszó segítségével megnyit egy fájlt a megadott névvel (`fajlnev`) és a `'w'` opcióval megnyitja írásra. Ezután a fájlba írja a kapott sztringet (`sztring`) a `write()` metódus segítségével. A `with` blokk végén a fájl automatikusan bezáródik, így nem kell külön bezárni a fájlt.

Példa használat:

```
python Copy code
szoveg = "Ez egy teszt szöveg."
string_fajlba(szoveg, "teszt.txt")
```

Ebben a példában a `szoveg` változó tartalmát írjuk a `teszt.txt` nevű fájlba a `string_fajlba()` függvény segítségével. Ha a fájl már létezik, akkor az írás felülírja annak tartalmát, ha nem létezik, akkor létrehozza a fájlt és beleírja a szöveget.

#12-----

Feladat: Számtani sorozat fájlba írása

Készíts függvényt szaz_szam_fajlba néven, amely 1-től 100-ig egyesével kiírja a számokat egy fájlba.

minden szám kerüljön új sorba.

A fájl nevét paraméterként kapja meg a függvény.

python

 Copy code

```
def szaz_szam_fajlba(fajlnev):
    with open(fajlnev, 'w') as f:
        for i in range(1, 101):
            f.write(str(i) + '\n')
```

Ez a függvény egy adott fájlba írja a számokat 1-től 100-ig, minden számot új sorba. A `with` blokk segítségével a fájl lezárása automatikusan megtörténik a függvény végrehajtása után. A `range` függvény segítségével generáljuk a számokat, majd a `write` metódussal írjuk ki a fájlba. A `\n` karakterrel új sort kezdünk, hogy minden szám külön sorba kerüljön.

#12-----

```
# Feladat: Számtani sorozat fájlba írása
# Készíts függvényt szaz_szam_fajlba néven, amely 1-től 100-ig egyesével kiírja a számokat egy
fájlba.
# minden szám kerüljön új sorba.
# A fájl nevét paraméterként kapja meg a függvény.
# Ne használd a write() metódust!
```

Az alábbiakban található egy megoldás a kért feladatra a `'print()'` függvény `'file'` paraméterének használatával:

```
python Copy code
def szaz_szam_fajlba(filename):
    with open(filename, 'w') as file:
        for i in range(1, 101):
            print(i, file=file)
```

A függvény egy `'with'` blokkban megnyitja a megadott fájlt írásra (`'w'` mód), majd végigmegy egy `'for'` ciklusban az 1-től 100-ig terjedő számokon. A `'print()'` függvény segítségével minden számot külön sorba ír a megadott fájlba. A `'file'` paraméteren keresztül lehet megadni, hogy a `'print()'` függvény melyik fájlba írja az adott szöveget.

Például a következőképpen hívható meg a függvény a `'szamok.txt'` nevű fájl létrehozására:

```
python Copy code
szaz_szam_fajlba('szamok.txt')
```

#13-----

```
# Feladat: Első karakter a szövegfájlban  
# Írj egy függvényt elso_karakter_a_fajlban néven, amely visszatér egy szövegfájl első  
karakterével.  
# A függvény bemenő paramétere a fájl neve.
```

Az alábbi Python kóddal meg lehet valósítani a kért függvényt:

python

 Copy code

```
def elso_karakter_a_fajlban(fajlnev):  
    with open(fajlnev, 'r') as f:  
        elso_karakter = f.read(1)  
    return elso_karakter
```

A függvény `open` függvénnyel megnyitja a megadott fájlt olvasásra (''r''), majd a `read` függvénnyel kiolvassa az első karaktert. Végül a függvény visszatér az első karakterrel.

Példa használat:

python

 Copy code

```
elso_karakter = elso_karakter_a_fajlban('teszt.txt')  
print(elso_karakter) # kiírja a szövegfájl első karakterét
```

#14-----

```
# Feladat: Utolsó karakter a szövegfájlban  
# Írj egy függvényt utolso_karakter_a_fajlban néven, amely visszatér egy szövegfájl utolsó  
karakterével.  
# A függvény bemenő paramétere a fájl neve.
```

Az alábbiakban látható egy lehetséges megoldás Python nyelven:

```
python Copy code  
  
def utolso_karakter_a_fajlban(fajlnev):  
    with open(fajlnev, 'r') as fajl:  
        tartalom = fajl.read()  
    return tartalom[-1]
```

Ez a függvény megnyitja a megadott fájlt olvasásra, majd beolvassa annak teljes tartalmát a `read()` metódus segítségével. Végül visszatér az utolsó karakterrel, amely a `tartalom` string utolsó karaktere (`tartalom[-1]`). A `with` blokk használata garantálja, hogy a fájl automatikusan lezárásra kerül, amikor a blokk véget ér, így nem kell külön lezárni a fájlt.

Használata például így történhet:

```
python Copy code  
  
utolso_karakter = utolso_karakter_a_fajlban('teszt.txt')  
print(utolso_karakter)
```

Ez az előző példát feltételezi, hogy a 'teszt.txt' nevű fájl a futtatható kód ugyanabban a mappájában található. Ha a fájl elérési útvonala eltérő, azt megfelelően kell megadni a fájlnévét tartalmazó stringben.

#15-----

```
# Feladat: Számok összege egy csak számokat tartalmazó szövegfájlban.  
# Írj egy függvényt szamok_osszege_a_fajlban néven amely visszatér egy szövegfájlban levő  
számok összegével.  
# A függvény bemenő paramétere a fájl neve.
```

Az alábbi függvény visszaadja egy csak számokat tartalmazó szövegfájlban lévő összes szám összegét:

python

 Copy code

```
def szamok_osszege_a_fajlban(fajlnev):  
    osszeg = 0  
    with open(fajlnev, 'r') as f:  
        for sor in f:  
            osszeg += int(sor.strip())  
    return osszeg
```

A függvény az adott fájlt megnyitja olvasásra, és soronként végigmegy rajta a `for sor in f:` ciklus segítségével. minden sorban számot feltételezünk, amelynek értékét az `int()` függvényel alakítjuk át. Az összeget minden egyes sor után frissítjük, majd visszaadjuk az összes összeget.

Használata például így történhet:

python

 Copy code

```
osszeg = szamok_osszege_a_fajlban('teszt.txt')  
print(osszeg)
```

A 'teszt.txt' fájlnak tartalmaznia kell csak számokat, különben a függvény az adatokat nem helyesen dolgozza fel.

#16-----

Feladat: Számok átlaga egy szövegfájlban.

Írj egy függvényt szamok_atlaga_a_fajlban néven, amely visszatér egy szövegfájlban levő számok átlagával.

A függvény bemenő paramétere a fájl neve.

Az alábbi függvény visszaadja egy szövegfájlban lévő összes szám átlagát:

python

 Copy code

```
def szamok_atlaga_a_fajlban(fajlnev):
    osszeg = 0
    darab = 0
    with open(fajlnev, 'r') as f:
        for sor in f:
            szam = int(sor.strip())
            osszeg += szam
            darab += 1
    if darab == 0:
        return 0
    return osszeg / darab
```

A függvény az adott fájlt megnyitja olvasásra, és soronként végigmegy rajta a `for sor in f:` ciklus segítségével. minden sorban számot feltételezünk, amelynek értékét az `int()` függvénnnyel alakítjuk át. Az összeget minden egyes sor után frissítjük, és a darabszámot is növeljük. Ha a fájl üres, akkor az átlag 0 lesz. Végül az összeget elosztjuk a darabszámmal, és visszaadjuk az átlagot.

Használata például így történhet:

python

 Copy code

```
atlag = szamok_atlaga_a_fajlban('teszt.txt')
print(atlag)
```

A 'teszt.txt' fájlnak tartalmaznia kell csak számokat, különben a függvény az adatokat nem helyesen dolgozza fel.

#17-----

Feladat: Páros számok száma egy szövegfájlban.

Írj egy függvényt paros_szamok_szama_a_fajlban néven, amely visszatér egy szövegfájlban levő páros számok számával.

A függvény bemenő paramétere a fájl neve.

Az alábbi függvény visszaadja egy szövegfájlban lévő páros számok számát:

python

 Copy code

```
def paros_szamok_szama_a_fajlban(fajlnev):
    db = 0
    with open(fajlnev, 'r') as f:
        for sor in f:
            szam = int(sor.strip())
            if szam % 2 == 0:
                db += 1
    return db
```

A függvény megnyitja az adott fájlt olvasásra, és soronként végigmegy rajta a `for sor in f:` ciklus segítségével. minden sorban feltételezzük, hogy szám található, amelynek értékét az `int()` függvénnnyel alakítjuk át. Ha a szám osztható kettővel (azaz páros), akkor növeljük a `db` változó értékét. Végül a függvény visszaadja a páros számok számát.

Használata például így történhet:

python

 Copy code

```
db = paros_szamok_szama_a_fajlban('teszt.txt')
print(db)
```

A 'teszt.txt' fájlnak tartalmaznia kell csak számokat, különben a függvény az adatokat nem helyesen dolgozza fel.

#17-----

```
# Feladat: Páros számok száma egy szövegfájlban.  
# Írj egy függvényt paros_szamok_szama_a_fajlban néven, amely visszatér egy szövegfájlban  
levő páros számok számával.  
# A függvény bemenő paramétere a fájl neve.  
# Használj list comprehension módszert.
```

Az alábbi függvény használja a list comprehension módszert, és visszaadja a páros számok számát egy szövegfájlban:

python

 Copy code

```
def paros_szamok_szama_a_fajlban(fajlnev):  
    with open(fajlnev, 'r') as f:  
        szamok = [int(sor.strip()) for sor in f if int(sor.strip()) % 2 == 0]  
    return len(szamok)
```

A függvény megnyitja az adott fájlt olvasásra, és a list comprehension segítségével végigmegy rajta soronként. A `int(sor.strip())` kifejezéssel az aktuális sorban lévő számot alakítjuk át integer típusúvá, majd megvizsgáljuk, hogy osztható-e kettővel (`% 2 == 0`). Ha igen, akkor hozzáadjuk a számok listájához. Végül a függvény visszaadja a számok listájának hosszát, azaz a páros számok számát.

Használata például így történhet:

python

 Copy code

```
db = paros_szamok_szama_a_fajlban('teszt.txt')  
print(db)
```

A 'teszt.txt' fájlnak tartalmaznia kell csak számokat, különben a függvény az adatokat nem helyesen dolgozza fel.

#18-----

Feladat: Páratlan számok száma egy szövegfájlban.

Írj egy függvényt paratlan_szamok_szama_a_fajlban néven, amely visszatér egy szövegfájlban levő páratlan számok számával.

A függvény bemenő paramétere a fájl neve.

Az alábbiakban megadom a megoldást a feladatra:

python

 Copy code

```
def paratlan_szamok_szama_a_fajlban(fajlnev):
    with open(fajlnev, "r") as f:
        return len([int(line) for line in f if int(line) % 2 == 1])
```

A függvény megnyitja a fájlt olvasásra ('"r"' mód), majd végigmegy rajta soronként. Ha a sorban lévő szám páratlan, akkor az átalakítjuk `int` típusúvá, és hozzáadjuk egy listához, amit a `list comprehension`-ben készítünk. Végül a lista hosszát adjuk vissza, ami a páratlan számok száma lesz.

#19-----

Feladat: Pozitív számok száma egy szövegfájlban.

Írj egy függvényt pozitiv_szamok_szama_a_fajlban néven, amely visszatér egy szövegfájlban levő pozitív számok számával.

A függvény bemenő paramétere a fájl neve.

Egy lehetséges megoldás a következő:

python

 Copy code

```
def pozitiv_szamok_szama_a_fajlban(fajlnev):
    with open(fajlnev, "r") as fajl:
        szamok = [int(s) for s in fajl.read().split() if s.isdigit()]
        pozitiv_szamok = [szam for szam in szamok if szam > 0]
    return len(pozitiv_szamok)
```

A függvény megnyitja a fájlt olvasásra, majd a fájlban talál

#19-----

```
# Feladat: Pozitív számok száma egy csak számokat tartalmazó szövegfájlban.  
# Írj egy függvényt pozitiv_szamok_szama_a_fajlban néven, amely visszatér egy szövegfájlban  
levő pozitív számok számával.  
# A függvény bemenő paramétere a fájl neve.  
# list comprehension nélkül!
```

Természetesen! Íme a megoldás:

python

 Copy code

```
def pozitiv_szamok_szama_a_fajlban(fajlnev):  
    with open(fajlnev, "r") as f:  
        szamok = f.read().split()  
        pozitiv_szamok = 0  
        for szam in szamok:  
            if float(szam) > 0:  
                pozitiv_szamok += 1  
    return pozitiv_szamok
```

Ez a függvény megegyezik az előző verzióval, kivéve azt, hogy itt nem használunk list comprehension-t, hanem egy egyszerű for loop-ot használunk a pozitív számok megszámlálásához.

#20-----

```
# Feladat: Negatív számok száma egy fájlban.  
# Írj egy függvényt negativ_szamok_szama_a_fajlban néven, amely visszatér egy fájlban levő  
negativ számok számával.  
# A függvény bemenő paramétere a fájl neve.
```

A megoldáshoz beolvassuk a fájl tartalmát, majd végigiterálunk rajta, és minden negatív szám esetén növeljük egy számláló változót. Végül visszatérünk a számláló értékével.

Itt a megoldás Python nyelven:

python

 Copy code

```
def negativ_szamok_szama_a_fajlban(fajlnev):  
    szamlalo = 0  
    with open(fajlnev, 'r') as f:  
        for sor in f:  
            szam = int(sor.strip())  
            if szam < 0:  
                szamlalo += 1  
    return szamlalo
```

Ez a függvény a megadott fájlban található negatív számok számával tér vissza. Ha például a "szamok.txt" fájl a következő számokat tartalmazza:

diff

 Copy code

```
1  
-5  
7  
-3  
0
```

akkor a függvény meghívása a következő eredményt adja vissza:

python

 Copy code

```
negativ_szamok_szama_a_fajlban('szamok.txt') # eredmény: 2
```

#21-----

```
# Feladat: Legkisebb szám egy fájlban.  
# Írj egy függvényt legkisebb_szam_a_fajlban néven, amely visszatér egy fájlban levő lekisebb  
számmal.  
# A függvény bemenő paramétere a fájl neve.  
# list comprehension nélkül!  
# minden sor csak egy számot tartalmaz.
```

A következőkben látható a megoldás list comprehension nélkül, amely minden sort végig iterál és a lekisebb értéket kiválasztja.

python

 Copy code

```
def legkisebb_szam_a_fajlban(fajl_neve):  
    with open(fajl_neve, 'r') as f:  
        legkisebb = None  
        for sor in f:  
            szam = int(sor.strip())  
            if legkisebb is None or szam < legkisebb:  
                legkisebb = szam  
    return legkisebb
```

Ez a megoldás kinyitja a fájlt olvasásra és végig iterálja a sorokat. minden sor először átalakításra kerül egy egész számmá `int(sor.strip())`, majd összehasonlításra kerül az aktuális `legkisebb` értékkel. Ha az aktuális szám kisebb, akkor az lesz a legkisebb érték. A függvény a lekisebb értékkel tér vissza.

```
# Feladat: Legnagyobb szám egy fájlban.  
# Írj egy függvényt legnagyobb_szam_a_fajlban néven, amely visszatér egy fájlban levő  
legnagyobb számmal.  
# A függvény bemenő paramétere a fájl neve.  
# minden sor egy számot tartalmaz
```

Az alábbiakban egy megoldást javaslok a feladatra:

python

 Copy code

```
def legnagyobb_szam_a_fajlban(fajlnev):  
    legnagyobb = None  
    with open(fajlnev) as f:  
        for sor in f:  
            szam = int(sor.strip())  
            if legnagyobb is None or szam > legnagyobb:  
                legnagyobb = szam  
    return legnagyobb
```

Az `open()` függvénnnyel megnyitjuk a fájlt, majd egy `for` ciklusban végigmegyünk a fájl összes során. minden sorban egy szám van, amit a `strip()` metódussal eltávolítunk a sort végző újsor karaktertől, majd az `int()` függvénnnyel átalakítjuk egész számmá.

Ezután ellenőrizzük, hogy a jelenlegi szám nagyobb-e, mint az eddig talált legnagyobb szám. Ha igen, akkor frissítjük a `legnagyobb` változó értékét.

Végül a függvény visszatér az eddig talált legnagyobb számmal.

Fontos megjegyezni, hogy a függvény csak akkor működik helyesen, ha minden sorban pontosan egy szám van. Ha a fájl tartalmaz bármilyen más típusú adatot vagy a sorokban több szám is szerepel, akkor a függvény hibát dobhat. A fájl ilyen esetekben előzetes ellenőrzése és a megfelelő hibakezelés implementálása javasolt.

#23-----

Feladat: Párosok egy szövegfájlból.

Írj egy függvényt parosok_a_fajlban néven, amely visszatér a szövegfájlban levő páros számokkal mint listával.

A függvény bemenő paramétere a fájl neve.

Az alábbiakban látható a kért függvény megvalósítása Python nyelven:

python

 Copy code

```
def parosok_a_fajlban(fajlnev):
    with open(fajlnev, 'r') as f:
        return [int(x) for x in f.read().split() if int(x) % 2 == 0]
```

A függvény megnyitja a bemeneti fájlt olvasásra, majd végigmegy az összes szóközzel elválasztott számon a fájlban. Amennyiben a szám páros, hozzáadja egy listához, ami a végén lesz a függvény kimenete. Az így kapott listát a függvény visszatérési értékeként adja vissza.

#24-----

Feladat: Páratlanok egy szövegfájlból.
Írj egy függvényt paratlanok_a_fajlbol néven, amely visszatér a szövegfájlban levő páratlan számokkal mint listával.
A függvény bemenő paramétere a fájl neve.

Egy megoldás a feladatra:

python

 Copy code

```
def paratlanok_a_fajlbol(fajlnev):
    with open(fajlnev, 'r') as f:
        lines = f.readlines()
    paratlanok = []
    for line in lines:
        szam = int(line.strip())
        if szam % 2 != 0:
            paratlanok.append(szam)
    return paratlanok
```

Ez a függvény ugyanúgy működik, mint az előző feladatban bemutatott `parosok_a_fajlbol` függvény, de itt a páratlan számokat gyűjtjük össze egy listába.

#25-----

```
# Feladat: Pozitívok egy szövegfájlból.  
# Írj egy függvényt pozitiv_a_fajlban néven, amely visszatér a szövegfájlban levő pozitív  
számokkal mint listával.  
# A függvény bemenő paramétere a fájl neve.
```

Az alábbiakban látható egy lehetséges megoldás a feladatra:

python

 Copy code

```
def pozitiv_a_fajlban(fajlnev):  
    pozitiv_szamok = []  
    with open(fajlnev, 'r') as f:  
        for sor in f:  
            szam = int(sor.strip())  
            if szam > 0:  
                pozitiv_szamok.append(szam)  
    return pozitiv_szamok
```

A függvény létrehoz egy üres listát a pozitív számok tárolására, majd megnyitja a fájlt olvasásra. Ezután soronként beolvassa a fájlt, és minden sorból eltávolítja a felesleges whitespace karaktereket (pl. sortörés). A `strip()` metódus használatával ezt könnyedén megtehetjük.

A következő lépésben az `int()` függvényel átalakítjuk az aktuális sort számmá. Ha a szám pozitív, akkor hozzáadjuk a `pozitiv_szamok` listához.

Végül a függvény visszatér a `pozitiv_szamok` listával, amely tartalmazza az összes pozitív számot a fájlból.

#26-----

Feladat: Negatívok egy szövegfájlból.

Írj egy függvényt negativok_a_fajlbol néven, amely visszatér a szövegfájlban levő negatív számokkal mint listával.

A függvény bemenő paramétere a fájl neve.

Az alábbiakban egy megoldást találsz a kért feladatra:

python

 Copy code

```
def negativok_a_fajlbol(fajlnev):
    with open(fajlnev, 'r') as f:
        negativok = [int(line.strip()) for line in f if int(line) < 0]
    return negativok
```

Ez a függvény hasonló a pozitívokat kereső függvénnnyel, de itt a feltételben az `int(line) < 0` jelenti, hogy csak azokat a számokat fogjuk beolvasni a fájlból, amelyek negatívak. Az így beolvasott számokat egy listába gyűjtjük, majd azt a listát adjuk vissza a függvény meghívónak.

#27-----

Feladat: Leggyakoribb szám a szövegfájlban.

Írj egy függvényt leggyakoribb_szam_a_fajlban néven, amely visszatér a szövegfájlban levő leggyakoribb számmal.

A függvény bemenő paramétere a fájl neve.

A feladat megoldásához szükség lesz az előfordulási gyakoriságok számítására. Ezt a Counter osztály segítségével tehetjük meg. Az alábbi függvényben először beolvassuk a számokat a fájlból, majd létrehozzuk a Counter objektumot, amely a számok előfordulási gyakoriságát számolja ki. Végül visszaadjuk a leggyakoribb számot, amelyet a Counter objektum most_common metódusának segítségével kaphatunk meg.

python

 Copy code

```
from collections import Counter

def leggyakoribb_szam_a_fajlban(fajlnev):
    with open(fajlnev, 'r') as f:
        szamok = [int(line.strip()) for line in f]
    szam_gyakorisagok = Counter(szamok)
    return szam_gyakorisagok.most_common(1)[0][0]
```

Teszteljük a függvényt a következő fájllal:

szamok.txt:

```
2
3
2
4
5
2
6
```

python

 Copy code

```
print(leggyakoribb_szam_a_fajlban('szamok.txt')) # kimenet: 2
```

A függvény sikeresen visszaadta a leggyakoribb számot, ami a fájlból szereplő 2-es szám volt.

#27-----

Feladat: Leggyakoribb szám a szövegfájlban.

Írj egy függvényt leggyakoribb_szam_a_fajlban néven, amely visszatér a szövegfájlban levő leggyakoribb számmal.

A függvény bemenő paramétere a fájl neve.

```
def leggyakoribb_szam_a_fajlban(fname):
    with open(fname) as f:
        lista = [int(sor) for sor in f]
    szotar = dict()
    for i in lista:
        szotar[i] = szotar.get(i, 0) + 1
    res = max(szotar, key=szotar.get)
    return res
```

A függvény jól működik és hatékonyan használja a szótárat a számok előfordulási gyakoriságának meghatározására. Az `'open'` függvény paraméterként átadott fájlnévvel megnyitja a fájlt, majd a `'with'` blokk lezárása után automatikusan bezárja azt, így nem kell külön figyelni a fájl bezárására.

A `'szotar.get(i, 0)'` kifejezés az adott kulcshoz (számhoz) tartozó értéket adja vissza a szótárból, vagy ha az adott kulcs még nem létezik a szótárban, akkor az alapértelmezett értéknek (0) megfelelő értéket.

A `'max()'` függvény a szótárban lévő kulcs-érték párok közül azokat az értékeket vizsgálja, amelyekhez az adott kulcs tartozik, és visszaadja a legnagyobb értékkel rendelkező kulcsot. A `'key=szotar.get'` paraméter megadja, hogy a `'max()'` függvény az értékek helyett azok kulcsait használja a maximum kereséséhez.

A függvény jól használja a Python beépített függvényeit, így hatékonyan és röviden oldja meg a feladatot.

#28-----

```
# Feladat: Hárrommal osztható számok a szövegfájlban.  
# Írj egy függvényt harommal_oszthato_szamok_a_fajlban néven, amely visszatér a  
szövegfájlban levő hárrommal osztható számok listájával.  
# A függvény bemenő paramétere a fájl neve.
```

Az alábbi függvényt használhatod a feladat megoldására:

python

 Copy code

```
def harommal_oszthato_szamok_a_fajlban(fajlnev):  
    eredmény = []  
    with open(fajlnev, 'r') as fajl:  
        for sor in fajl:  
            szám = int(sor.strip())  
            if szám % 3 == 0:  
                eredmény.append(szám)  
    return eredmény
```

Ez a függvény végigiterál a megadott fájl sorain, és minden sorban lévő számot átalakít int típusúvá. Ha a szám osztható 3-mal, akkor hozzáadja az eredmény listához. Végül az eredmény listát adja vissza.

A függvény használatához meg kell adni a fájl nevét a bemeneti paraméterként. Például:

python

 Copy code

```
eredmény = harommal_oszthato_szamok_a_fajlban('szamok.txt')  
print(eredmény)
```

A fenti kódrészlet a "szamok.txt" fájlból kigyűjt az összes hárrommal osztható számot, majd kiírja őket a képernyőre.

#29-----

Feladat: Neggyel osztható számok a szövegfájlban.

Írj egy függvényt neggyel_oszthato_szamok_a_fajlban néven, amely visszatér a szövegfájlban levő neggyel osztható számok listájával.

A függvény bemenő paramétere a fájl neve.

Az alábbi függvényt használhatod a feladat megoldására:

python

 Copy code

```
def neggyel_oszthato_szamok_a_fajlban(fajlnev):
    eredmény = []
    with open(fajlnev, 'r') as fajl:
        for sor in fajl:
            szám = int(sor.strip())
            if szám % 4 == 0:
                eredmény.append(szám)
    return eredmény
```

Ez a függvény végigiterál a megadott fájl sorain, és minden sorban lévő számot átalakít int típusúvá. Ha a szám osztható 4-gyel, akkor hozzáadja az eredmény listához. Végül az eredmény listát adja vissza.

A függvény használatához meg kell adni a fájl nevét a bemeneti paraméterként. Például:

python

 Copy code

```
eredmény = neggyel_oszthato_szamok_a_fajlban('szamok.txt')
print(eredmény)
```

A fenti kódrészlet a "szamok.txt" fájlból kigyűjt az összes neggyel osztható számot, majd kiírja őket a képernyőre.