1. Kör területének és kerületének számítása: Készíts egy **Kor** nevű osztályt, amelynek van egy **r** attribútuma (a kör sugara) és egy **pi** attribútuma (a pi értéke). Az osztály **\_\_init\_\_** metódusában kérj be egy valós számot a felhasználótól, és tároljuk el azt az **r** attribútumban. A **Kor** osztályon belül készíts két metódust: az egyik számítsa ki a kör területét, a másik pedig a kerületét. Mindkét metódus térjen vissza a kiszámolt értékkel. Írjuk ki a kiszámolt értékeket konzolra!

class Kor:

pi = 3.1415

def \_\_init\_\_(self, r):

self.r = r

def terulet(self):

return self.pi \* self.r \*\* 2

def kerulet(self):

return 2 \* self.pi \* self.r

r = float(input("Kérem a kör sugarát: "))

kor = Kor(r)

print(f"A kör területe: {kor.terulet()}")

print(f"A kör kerülete: {kor.kerulet()}")

1. Háromszög területének és kerületének számítása: Készíts egy **Haromszog** nevű osztályt, amelynek van három attribútuma (**a**, **b**, **c**) a háromszög oldalainak hosszát tároljuk. Az osztály **\_\_init\_\_** metódusában kérjünk be három valós számot a felhasználótól, és tároljuk el azokat az **a**, **b** és **c** attribútumokban. A **Haromszog** osztályon belül készítsünk két metódust: az egyik számítsa ki a háromszög területét, a másik pedig a kerületét. Mindkét metódus térjen vissza a kiszámolt értékkel. Írjuk ki a kiszámolt értékeket konzolra.

class Haromszog:

def \_\_init\_\_(self, a, b, c):

self.a = a

self.b = b

selfc. = c

def terulet(self):

s = (self.a + self.b + self.c) / 2

return (s \* (s - self.a) \* (s - self.b) \* (s - self.c)) \*\* 0.5

def kerulet(self):

return self.a + self.b + self.c

a = float(input("Kérem a háromszög 'a' oldalát: "))

b = float(input("Kérem a háromszög 'b' oldalát: "))

c = float(input("Kérem a háromszög 'c' oldalát: "))

h = Haromszog()

print(f"A háromszög területe: {h.terulet()}")

print(f"A háromszög kerülete: {h.kerulet()}")

1. Egy diák átlagának számítása és fájlba írása: Készíts egy **Diak** nevű osztályt, amelynek van egy **nev** attribútuma (a diák neve), és egy **jegyek** attribútuma (a diák jegyeinek listája). Az osztály **\_\_init\_\_** metódusában kérjük be a diák nevét a felhasználótól, és adjuk hozzá a **nev** attribútumhoz. Az osztály **jegyek** attribútumát kapjuk meg paraméterként a felhasználótól, így nem kell az **\_\_init\_\_** metódusban beolvasnunk. A **Diak** osztályon belül készítsünk egy **atlag** metódust, amely kiszámolja a diák jegyeinek átlagát, és egy **jegyek\_fajlba** metódust, amely kiírja a diák nevét és jegyeit egy fájlba. A fájlba íráshoz használjuk a Python beépített **open** függvényét, amelynek megadjuk a fájl nevét és a módját, amelyen keresztül a fájlt meg akarjuk nyitni (pl. írásra). Írjuk ki a diák átlagát konzolra és a fájlba is.

class Diak:

def \_\_init\_\_(self, nev, jegyek):

self.nev = nev

self.jegyek = jegyek

def atlag(self):

return sum(self.jegyek) / len(self.jegyek)

def jegyek\_fajlba(self, fajlnev):

with open(fajlnev, "w") as f:

f.write(f"{self.nev} jegyei:\n")

for jegy in self.jegyek:

f.write(str(jegy) + "\n")

nev = input("Kérem a diák nevét: ")

jegyek = []

while True:

jegy = input("Kérem a jegyet (vagy írja be a 'q'-t a kilépéshez): ")

if jegy == "q":

break

jegyek.append(int(jegy))

diak = Diak(nev, jegyek)

print(f"{diak.nev} átlaga: {diak.atlag()}")

diak.jegyek\_fajlba("jegyek.txt")

1. Szobakereső alkalmazás: Készíts egy **Szoba** nevű osztályt, amelynek van egy **azonosito** attribútuma (a szoba azonosítója), egy **ferohely** attribútuma (a szoba befogadóképessége), és egy **foglalt** attribútuma (a szoba foglaltsága). Az osztály **\_\_init\_\_** metódusában kérjük be a szoba azonosít óját a felhasználótól és adjuk hozzá az **azonosito** attribútumhoz. A **ferohely** attribútumot is kapjuk meg paraméterként a felhasználótól. A **foglalt** attribútum alapértelmezetten **False** legyen, tehát új szobák esetén még nincsenek foglalva. A **Szoba** osztályon belül készítsünk egy **foglal** metódust, amely lefoglalja a szobát (beállítja a **foglalt** attribútumot **True**-ra), és egy **szabad** metódust, amely felszabadítja a szobát (beállítja a **foglalt** attribútumot **False**-ra). A **Szoba** osztályon kívül kérjünk be a felhasználótól egy dátumot, amelyre szobát keresünk. Az alkalmazásunkban egy adott dátumra csak szabad szobák jelenjenek meg. Az alkalmazásunk a szabad szobákat egy fájlba írja ki, így könnyen áttekinthetjük a szabad helyeket.

class Szoba:

def \_\_init\_\_(self, azonosito, ferohely):

self.azonosito = azonosito

self.ferohely = ferohely

self.foglalt = False

def foglal(self):

self.foglalt = True

def szabad(self):

self.foglalt = False

szobak = [

Szoba("101", 2),

Szoba("102", 1),

Szoba("103", 4),

Szoba("201", 3),

Szoba("202", 2),

Szoba("203", 1),

]

datum = input("Kérem a dátumot (YYYY-MM-DD): ")

szabad\_szobak = []

for szoba in szobak:

if not szoba.foglalt:

szabad\_szobak.append(szoba)

szabad\_szobak\_fajlba = []

for szoba in szabad\_szobak:

szabad\_szobak\_fajlba.append(f"{szoba.azonosito} ({szoba.ferohely} fő)")

with open("szabad\_szobak.txt", "w") as f:

f.write(f"Szabad szobák a következő dátumra: {datum}\n\n")

if szabad\_szobak\_fajlba:

f.write("\n".join(szabad\_szobak\_fajlba))

else:

f.write("Nincsenek szabad szobák.")

1. Az alábbiakban egy olyan példa feladatot mutatok be, amelyben egy egyszerű webáruház termékeit kell kezelnünk. Az alkalmazásunkban egy **Termek** osztály reprezentálja a terméket, amelynek az árát és az elérhető készletmennyiségét is tároljuk. A felhasználótól bekérjük a rendelni kívánt termék nevét, majd ellenőrizzük, hogy az adott termék elérhető-e a készleten. Ha igen, bekérjük a rendelni kívánt mennyiséget, majd kiszámoljuk a rendelés összegét és az elérhető készletmennyiséget csökkentjük a rendelt mennyiséggel. Az alkalmazásunk egy fájlba írja ki az elérhető készletmennyiségeket.

class Termek:

def \_\_init\_\_(self, nev, ar, keszlet):

self.nev = nev

self.ar = ar

self.keszlet = keszlet

termekek = [

Termek("Telefon", 49990, 5),

Termek("Laptop", 149990, 3),

Termek("Tablet", 79990, 2),

]

rendeles = input("Kérem a rendelni kívánt termék nevét: ")

for termek in termekek:

if termek.nev.lower() == rendeles.lower():

if termek.keszlet > 0:

mennyiseg = int(input("Kérem a rendelt mennyiséget: "))

if mennyiseg <= termek.keszlet:

osszeg = mennyiseg \* termek.ar

termek.keszlet -= mennyiseg

print(f"A rendelés összege: {osszeg}")

else:

print("Nincs elegendő készlet.")

else:

print("Nincs készleten.")

break

else:

print("Nem található ilyen termék.")

keszlet\_fajlba = []

for termek in termekek:

keszlet\_fajlba.append(f"{termek.nev}: {termek.keszlet} db")

with open("keszlet.txt", "w") as f:

f.write("\n".join(keszlet\_fajlba))

1. Az alábbiakban egy egyszerű példa feladatot mutatok be, amelyben egy autógyár autóinak adatait kell kezelnünk. Az alkalmazásunkban egy **Auto** osztály reprezentálja az autót, amelynek a típusát, az árát és az elérhető darabszámát is tároljuk. A felhasználótól bekérjük, hogy milyen típusú autót szeretne vásárolni, majd ellenőrizzük, hogy az adott típusú autó elérhető-e a készleten. Ha igen, bekérjük a vásárolni kívánt darabszámot, majd kiszámoljuk a vásárlás összegét és az elérhető darabszámot csökkentjük a vásárolt darabszámmal. Az alkalmazásunk egy fájlba írja ki az elérhető darabszámokat.

class Auto:

def \_\_init\_\_(self, tipus, ar, db):

self.tipus = tipus

self.ar = ar

self.db = db

autok = [

Auto("Suzuki Swift", 3490000, 5),

Auto("Opel Corsa", 3790000, 3),

Auto("Toyota Yaris", 4190000, 2),

]

vasarlas = input("Kérem a vásárolni kívánt autó típusát: ")

for auto in autok:

if auto.tipus.lower() == vasarlas.lower():

if auto.db > 0:

db = int(input("Kérem a vásárolni kívánt darabszámot: "))

if db <= auto.db:

osszeg = db \* auto.ar

auto.db -= db

print(f"A vásárlás összege: {osszeg}")

else:

print("Nincs elegendő készlet.")

else:

print("Nincs készleten.")

break

else:

print("Nem található ilyen autó típus.")

darabszamok\_fajlba = []

for auto in autok:

darabszamok\_fajlba.append(f"{auto.tipus}: {auto.db} db")

with open("darabszamok.txt", "w") as f:

f.write("\n".join(darabszamok\_fajlba))

1. Az alábbi egyszerű feladat során a felhasználótól kérünk be egy számot, majd kiszámoljuk ennek a számnak az osztóit, majd a szám prímszám-e vagy sem. A kiszámolt eredményeket kiírjuk a képernyőre.

class Osztok:

def \_\_init\_\_(self, szam):

self.szam = szam

def osztok(self):

osztok = []

for i in range(1, self.szam + 1):

if self.szam % i == 0:

osztok.append(i)

return osztok

def prim(self):

if self.szam < 2:

return False

for i in range(2, int(self.szam \*\* 0.5) + 1):

if self.szam % i == 0:

return False

return True

szam = int(input("Kérem adjon meg egy egész számot: "))

osztok = Osztok(szam).osztok()

if Osztok(szam).prim():

print(f"A(z) {szam} prímszám.")

else:

print(f"A(z) {szam} nem prímszám.")

print(f"A(z) {szam} osztói: {osztok}")