Szövegfájlok feldolgozása

2025

1 Szövegfájlok feldolgozása

Kétfajta adattárolási módszerrel tudunk fájlokat használni. - Az egyik az úgynevezett szövegfájl, amely szöveges adatokat tartalmaz, ez az emberek számára is olvasható formátum.
A szövegfájl sorokból áll, amelynek hossza előre nem ismert, de a végét egy nem látható sorvégjel zárja.
A szövegfájlokban az operációs rendszerektől függően eltérő a sorvégjel. Windows esetén a sorokat két escape karakter zárja `\r\n`, Unix esetén csak a `\n` és Machintos esetén pedig a `\r`.
A szövegfájlokat általában karakterenként vagy soronként kell feldolgozni, értelmezni.
A feldolgozás során szekvenciálisan kell haladni, mert nem lehet tudni, hogy hol lesz a szükséges információ. - A másik tárolási módszer a bináris tárolás, amely tetszőleges bájtokat tartalmazhat, az ember számára már olvashatatlan.
A bináris fájlok alkalmasak lehetnek például képek, hangfájlok vagy más bináris adatok tárolására.
A bájtok helyes értelmezése a programozó feladata. A struktúra ismerete esetén, nem kell végig nézni a teljes tartalmat, el lehet navigálni mindegyik adathoz külön-külön.

1.1 Fájl megnyitása

Az `open()` függvény segítségével lehet megnyitni a megadott fájlt. `open(filename, mode='r', encoding=None, newline=None)`

A fájl megnyitásához különböző módokat támogat a második argumentumként megadható karakterlánc segítségével: - **“r” (read):** Olvasás mód. A fájlt csak olvasásra nyitja meg, és a fájlmutató a fájl elején helyezkedik el. - **“w” (write):** Írás mód. Ha a fájl nem létezik, létrehozza; ha létezik, akkor a tartalmát felülírja. A fájlmutató a fájl elején helyezkedik - **“x” (create):** Létrehozás. Ha a fájl nem létezik, létrehozza; ha létezik, akkor egy hibát ad. - **“a” (append):** Fűzés mód. Ha a fájl nem létezik, létrehozza; ha létezik, akkor a tartalmát megőrzi, és a fájlmutatót a fájl végére helyezi. - **“rb”, “wb”, “ab”:** Bináris módok. Az előző módokban a fájlokat szöveges módúként kezeljük. Ha bináris adatokat szeretnél olvasni vagy írni, a fenti módokat “b” hozzáadásával érhető el. - **“r+”:** Olvasás és írás mód. - **“w+”:** Írás és olvasás mód; a fájl tartalmát felülírja. - **“a+”:** Fűzés és olvasás mód; a fájlmutató a fájl végén helyezkedik el.

A `encoding` paraméter határozza meg, hogy a Python milyen karakterkódolást használjon az írásnál vagy olvasásnál. Ékezetes karakterek használatánál gyakori az `utf-8`-as karakterkódolás.

A `newline` paraméter szabályozza, hogyan kezelje a Python a soremeléseket (`\n`, `\r`, `\r\n`) szöveges módban. Alapértelmezetten (`None`) a Python automatikusan konvertálja a platform szerinti soremelésre. Üres string `""` esetén nem történik semilyen konverzió, a megadott sorvégjel kerül felhasználásra.

[ ]:
```
f = open(
    "adatok.txt", mode="w", encoding="utf-8"
)  # fájl megnyitása, adott móddal és karakterkódolással
print(type(f))
```

1.2 Fájl bezárása

Ameddig a fájl meg van nyitva, addig a Python lefoglalva tartja. A fájlműveletek befejezését követően be kell zárni a fájlt a `close()` függvénnyel. A bezárás utasítás hatására meg fognak jelenni a fájlban a változtatások és a többi külső program is meg tudja majd nyitni saját használatra.

[ ]:
```
f.close()  # fájl bezárása
```

1.3 Kontextuskezelő alkalmazása

A `with` használatával a kódblokk végén automatikusan megtörténnek bizonyos takarítási műveletek, például a fájl zárása. A `with` kulcsszó nagyban megkönnyíti a fájlok, adatbázisok, hálózati kapcsolatok vagy más erőforrások használatát, és segít elkerülni a hibaforrásokat.
A fájlok esetén például a `with` blokk garantálja, hogy a fájl automatikusan lezáródik, még akkor is, ha kivétel keletkezik a kódblokkban. Így nem kell külön zárnunk a fájlt.

[ ]:
```
with open("adatok.txt", "w") as f:
    pass  # ide kerülnek majd az utasítások
    # A fájl itt még mindig nyitva van

# A fájl a with blokk után automatikusan lezáródik
# Már nem kell külön f.close() hívást tennünk
```

A `with` blokk alapvetően egy olyan objektumot kezel, amely rendelkezik `__enter__` és `__exit__` metódusokkal. Amikor belépünk a `with` blokkba (`__enter__`), az objektum inicializálódik, és a blokk végén (`__exit__`) a takarítási műveletek történnek meg (például a fájl lezárása).

1.4 Fájlba történő írás művelete

A fájlba a `write()` függvény segítségével írhatunk ki adatot. Az adat lehet szöveg vagy egy bájtsorozat.
A szövegek egymás után kerülnek a fájlba, ha szeretnénk új sorokat létrehozni akkor oda `\n` sorvég jelet kell tenni.
A magyar ékezetes karakterek támogatása miatt a szövegfájlt, `utf-8`-as karakterkódolással érdemes létrehozni.

[ ]:
```
f = open("adatok.txt", "w")  # szövegfájl megnyitása írása
f.write("Egy egy szöveg, ami bekerül a fájlba.\n")  # sorok írása
f.write("Ez lesz a második sor.")  # sorok írása
f.close()  # fájl bezárása
```

Bináris fájl esetén a megnyitáskor a `b` módosítót is be kell írni, és a `write()` függvénynek egy bájtokból álló sorozatot kell átadni, amit szintén egy `b` betűvel kell jelölni a szöveg előtt.

```
[ ]: f = open("binaris_adatok.bin", "wb")  # bináris adatok megnyitása írása
     f.write(b"\x01\x02\x03\x04")  # bájtok sorozata
     f.close()  # fájl bezárása
```

A `writelines()` függvény segítségével egy lista tartalmát írathatjuk ki egy lépésbe. Az elemek egymás után kerülnek a szövegfájlba.

```
[ ]: nevek = ["Attila", "Antal", "Álmos"]

     with open("adatok.txt", "w") as f:
         # listaelemek kiírása
         f.writelines(nevek)
```

Ha szeretnénk az elemek között valamilyen határoló karaktert vagy sortörést elhelyezni, akkor célszerű a lista elemeit azzal összefűzni, és a kapott szöveget írjuk ki.

```
[ ]: nevek = ["Attila", "Antal", "Álmos"]

     with open("adatok.txt", "w") as f:
         # listaelemek összefűzése majd kiírása
         f.write(" ".join(nevek))
```

A lista elemeinek feldolgozásához készíthetünk egy egyszerű for ciklust is, amin belül elemenként kerül kiírásra a tartalom.

```
[ ]: nevek = ["Attila", "Antal", "Álmos"]

     with open("adatok.txt", "w") as f:
         # lista bejárása majd az elemek kiírása külön sorokba
         for nev in nevek:
             f.write(nev + "\n")
```

1.5 Fájl tartalmának beolvasása

A fájl teljes tartalmát a `read()` függvénnyel olvashatjuk be.

```
[ ]: # beolvasás egybe
     with open("adatok.txt", "r") as f:
         szoveg = f.read()
         print(szoveg)
```

A `read()` függvény argumentumában megadható, hogy mennyi karakter kerüljön beolvasásra.

```
[ ]: # beolvasás karakterenként
     with open("adatok.txt", "r") as f:
         print(f.read(3))  # 3 karakter
         print(f.read(2))  # 2 karakter
```

A fájl teljes tartalma beolvasható sorok listájaként a `readlines()` függvény segítségével.

[ ]:
```
# beolvasás listába
with open("adatok.txt", "r") as f:
    sorok = f.readlines()
    print(sorok)
```

A függvény argumentumában itt is meg lehet adni, hogy mennyi sort olvasson be.

[ ]:
```
# beolvasás listába 1 sort
with open("adatok.txt", "r") as f:
    print(f.readlines(1))  # 1 sor
```

A fájl teljes beolvasása nagyméretű fájlok esetén sokszor felesleges. Helyettes érdemes a fájlt soronként beolvasni és feldolgozni.
A `readline()` függvény segítségével beolvasható egy sor. A függvénynek ebben az esetben is meg lehet adni, hogy mennyi karakterig olvasson, ha egy sorban kevesebb karakter van, akkor csak a sorvégjelig olvas be.
A teljes fájl soronkénti feldolgozását egy `while` ciklus segítségével oldhatjuk meg. Mindig beolvasunk egy új sort, egészen addig, amíg a fájlvégjelig (EOF) el nem érkezünk.

[ ]:
```
# soronkénti beolvasás readline segítségével
with open("adatok.txt", "r") as f:
    sor = f.readline()
    while sor != "":  # A fájlvég (EOF) karakter egy üres string
        print(sor, end="")
        sor = f.readline()
```

A fenti módszer egy egyszerűsített használatát teszi lehetővé, ha a fájlobjektumot átadjuk egy `for` ciklusnak.

[ ]:
```
# soronkénti beolvasás for ciklus segítségével
with open("adatok.txt", "r") as f:
    for sor in f:
        print(sor, end="")
```

2 Gyakorló feladatok

1. Feladat

   Olvassa be a zh.txt fájl tartalmát és írja ki a neptunkódokat.

[ ]:
```
with open("./file/zh.txt", "r") as f:
    f.readline()  # fejléc kiolvasása
    for sor in f:
        sor = sor.strip()
        if sor == "":
            continue
        reszek = sor.split("\t")
        print(reszek[0])
```

2. Feladat

Olvassa be a zh.txt fájl tartalmát és írja ki azokat a neptunkódokat, akiknek nem sikerült a zh.

[ ]:
```
with open("./file/zh.txt", "r") as f:
    f.readline()  # fejléc kiolvasása
    for sor in f:
        sor = sor.strip()
        if sor == "":
            continue
        reszek = sor.split("\t")
        if int(reszek[1]) < 40:
            print(reszek[0])
```

3. Feladat

Olvassa be a zh.txt fájl tartalmát és írja ki a pontszámok átlagát.

[ ]:
```
with open("./file/zh.txt", "r") as f:
    f.readline()  # fejléc kiolvasása
    osszeg = 0
    db = 0
    for sor in f:
        sor = sor.strip()
        if sor == "":
            continue
        reszek = sor.split("\t")
        osszeg += int(reszek[1])
        db += 1

print(f"A pontszámok átlaga: {osszeg/db:.2f}")
```

3 CSV fájlok használata

A CSV (Comma-Separated Values) egy egyszerű szöveges adatfájlformátum, amelyet gyakran használnak táblázatok, táblázatos adatok vagy adatbázis rekordok tárolására. A CSV fájlokban az adatokat cellákból álló sorok alkotják, és a cellák közötti elválasztásált (delimiter) egy meghatározott karakter, leggyakrabban vessző (`,`), tabulátor (`\t`) vagy pontosvessző (`;`) adja.
Az Exelből könnyen exportálhatunk csv fájlt, azonban figyelnünk kell arra, hogy az elválasztó karakter függ az operációs rendszer nyelvétől. Angol területi beállításokat használva az exportált fájlban a vessző (`,`) lesz a határó karakter és a lebegőpontos számok tizedesének jelen a pont(`.`). Magyar területi beállítással rendelekező gép esetén a szeparáló karakter a pontosvessző (`;`) lesz és a tizedesvessző (`,`) fog megjeleni a lebegőpontos számoknál.

3.0.1 CSV fájl beolvasása

A `csv` modul a Python beépített modulja, amely lehetővé teszi a CSV fájlok könnyű kezelését. Segítségével könnyen olvashatunk és írhatunk CSV fájlokat, és számos beállítási lehetőséget biztosít a fájlok formázásához és feldolgozásához.
A `csv.reader()` osztály segítségével létrehozunk egy CSV olvasót, majd a `for` ciklussal soronként

beolvassuk a fájlt.
Minden sort szétszed a `delimiter`-ben megadott karakternél és egy listába adja vissza.

[ ]:
```python
import csv

with open("./file/zh.txt", "r", encoding="utf-8") as csv_file:
    csv_reader = csv.reader(csv_file, delimiter="\t")

    fejlec = next(csv_reader)  # 1 sor kiolvasása
    print(f"{fejlec[0]}, {fejlec[1]}")

    osszeg = 0
    sorszam = 0
    for sor in csv_reader:      # további sorok kiolvasása a végéig
        print(f"{sor[0]}, {sor[1]}")
        osszeg += int(sor[1])
        sorszam += 1

    print(f"Összesen {sorszam} adat")
    print(f"A pontok átlaga: {osszeg/sorszam:.2f}")
```

3.0.2 CSV fájl létrehozása

A `csv.writer()` függvény segítségével létrehozunk egy CSV írót, majd a `writerow()` vagy a `writerows()` metódusokkal írjuk be a sorokat a fájlba. Ezek a függvények automatikusan tartalmazzák a soremeléseket, ezért a fájl megnyitásakor a `newline` paraméternek adjunk meg egy üres stringet `""`, hogy ne történjen dupla soremelés.

A `quoting` paraméter vezérli, hogy a CSV író mikor tegyen idézőjelet a mezők köré. Alapértelmezetten (`csv.QUOTE_MINIMAL`) csak olyan mezők köré ír ídézőjelet, amelyekben vesszők, soremelések vagy idézőjelek fordulnak elő. A `csv.QUOTE_ALL` minden mezőt idézőjelek közé tesz. A `csv.QUOTE_NONNUMERIC` minden nem numerikus mezőt idézőjelek közé tesz, a számokat automatikusan lebegőpontos értékké alakítja.

[ ]:
```python
import csv

with open("eredmenyek.csv", "w", newline="") as csv_file:
    csv_writer = csv.writer(csv_file, quoting=csv.QUOTE_NONNUMERIC)

    csv_writer.writerow(["Neptunkód", "Pontszám"]) # 1 sor kiírása

    csv_writer.writerow(["BATMAN", 15])
    csv_writer.writerow(["FGVDFS", 80])

    adatok = [["FGGB4F", 28], ["DCCCSD", 67], ["VBT3FT", 45], ["7TJ3AS", 79]]

    csv_writer.writerows(adatok)        # további sorok kiírása
```

További lehetőségek csv fájlok feldolgozására: https://realpython.com/python-csv/

Windows Excel-lel készített CSV fájl tartalmaz egy speciális bájtsorozat (BOM - Byte Order Mark) a fájl elején. Az `utf-8-sig` encoding használta automatikusan eltávolítja a BOM-ot a fájl elejéről, így nem kell manuálisan kezelni.

4 Gyakorló feladat (csv)

1. Feladat

   Excel segítségével készítsen egy olyan darabjegyzéket, amelyben M3, M4, M5, M6, M8, M10 méretű csavarok lesznek felsorolva tetszőleges sorrendben. Minden sor első oszlopában a csavar típusa lesz, a második oszlopban pedig a darabszám. Egy csavar több helyen is előfordul, különböző sorokban. Mentse ki a darabjegyzéket egy csavarok.csv kiterhesztésű fájlba. Határozza meg, hogy melyik méretű csavaból mennyit kell várásolni.
   Mentse ki az eredményt egy összesítő rendeles.csv fájlba.

[ ]:
```python
import csv

meretek = ["M3","M4","M5","M6","M8","M10"]
darabok = [0]*len(meretek)

with open("./file/csavarok.csv", encoding='utf-8-sig') as csv_file: # BOM␣
 ↪esetén kell az utf-8-sig
    csv_reader = csv.reader(csv_file, delimiter=";") # angol területen "," kell
    for sor in csv_reader:
        if sor[0] in meretek:
            index = meretek.index(sor[0])
            darabok[index] += int(sor[1])
        else:
            print("Hiba", sor[0], "nincs benne a listában")

with open("rendeles.csv", "w", newline="") as csv_file:
    csv_writer = csv.writer(csv_file, quoting=csv.QUOTE_NONNUMERIC, delimiter=";
 ↪")

    for index, meret in enumerate(meretek):
        print(f"{meret}: {darabok[index]} db")
        csv_writer.writerow([meret, darabok[index]])
```

5 Fájl- és mappakezelő funkciók

Az `os` modul hasznos műveleteket kínál az elérési útvonalak kezelésére és ellenőrzésére a Pythonban.

Az `os.path.join()` összefűzi a megadott elérési útvonalakat egyetlen elérési útvonallá, figyelembe véve a rendszerspecifikus elválasztókat.

[ ]:
```python
import os
```

```
path = os.path.join(".", "file", "zh.txt")
print(path)
```

Az `os.path.basename()` visszaadja az elérési útvonal utolsó elemét (a fájl vagy mappa nevét)

[ ]:
```
import os

path = os.path.join(".", "file", "zh.txt")
nev = os.path.basename(path)
print(nev)
```

Az `os.path.dirname()` visszaadja az elérési útvonal mappájának elérési útvonalát.

[ ]:
```
import os

path = os.path.join(".", "file", "zh.txt")
dir = os.path.dirname(path)
print(dir)
```

Az `os.path.exists()` ellenőrzi, hogy a megadott elérési útvonal létezik-e.

[ ]:
```
import os

path = os.path.join(".", "file", "zh.txt")
out = os.path.exists(path)
print(out)
```

Az `os.path.isfile()` ellenőrzi, hogy a megadott elérési útvonal egy fájl-e.

[ ]:
```
import os

path = os.path.join(".", "file", "zh.txt")
out = os.path.isfile(path)
print(out)
```

Az `os.path.isdir()` ellenőrzi, hogy a megadott elérési útvonal egy mappa-e.

[ ]:
```
import os

path = os.path.join(".", "file", "zh.txt")
out = os.path.isdir(path)
print(out)
```

Az `os.path.split()` kettéválasztja az elérési útvonalat és az utolsó mappát vagy fájlt, és egy tuple-t ad vissza.

[ ]:
```
import os

path = os.path.join(".", "file", "zh.txt")
path, file = os.path.split(path)
```

```
print(path)
print(file)
```

A fájl méretének meghatározásához használható az `os.path.getsize()` függvény, amely az adott fájl méretét bájtban adja vissza.

[ ]:
```
import os

path = os.path.join(".", "file", "zh.txt")
meret = os.path.getsize(path)

print(f"Mérete: {meret} byte")
```

A könyvtárban található fájlok és mappák listázásához használható az `os.listdir()` függvény. Ez a függvény egy listát ad vissza, amely tartalmazza a megadott könyvtárban található összes fájl és mappa nevét.

[ ]:
```
import os

path = ".."
elemek = os.listdir(path)
print(elemek)
```

A mappákat és a fájlokat egy `for` cikluson belül az `os.path.isdir()` függvény segítségével szét lehet válogatni.

[ ]:
```
import os

path = ".."
elemek = os.listdir(path)

mappak = []
fajlok = []
for elem in elemek:
    path = os.path.join("..", elem)
    if os.path.isdir(path):
        mappak.append(elem)
    else:
        fajlok.append(elem)

print(mappak)
print(fajlok)
```

Fájl törlésére használható az `os.remove()` függvény.

[ ]:
```
import os

if os.path.exists("demofile.txt"):
    os.remove("demofile.txt")
```

```
else:
    print("A fájl nem létezik")
```

Könyvtár létrehozása az `os.mkdir()` függvény használható.

[ ]:
```
import os

if os.path.exists("teszt"):
    print("A könyvtár már létezik")
else:
    os.mkdir("teszt")
```

Üres könyvtár törlése az `os.rmdir()` függvény használható.

[ ]:
```
import os

path = "teszt"
if os.path.exists(path):
    files = os.listdir(path)
    if files.count == 0:
        os.rmdir("teszt")
    else:
        print("A könyvtár nem üres")
else:
    print("A könyvtár nem létezik")
```

Fájl vagy mappa átnevezése (átmozgatása) az `os.rename()` függvénnyel történik.

[ ]:
```
import os

if os.path.exists("teszt"):
    os.rename("teszt", "tesztElek")
```

Fájl vagy mappa másolása, a shutil modul `shutil.copy()` függvényével történik.

[ ]:
```
import shutil

shutil.copy("file/zh.txt", "tesztElek/zh.txt")
```

6 Gyakorló feladat (path)

1. Feladat

   Írjuk ki az összes fájlunk nevét és méretét a szülőkönyvtárunk (`.`) szintjétől. Állapítsuk meg, mennyi helyet foglalnak el összesen.

[ ]:
```
import os

path = "."
```

```
elemek = os.listdir(path)

osszmeret = 0
for elem in elemek:
    path = os.path.join(".", elem)
    if os.path.isdir(path):
        gyerek_elemek = os.listdir(path)
        for gyerek in gyerek_elemek:
            gyerek_path = os.path.join(".", elem, gyerek)
            meret = os.path.getsize(gyerek_path)
            print(f"{elem}\\{gyerek}: {meret} byte")
            osszmeret += meret
    elif elem[0] != ".":
        meret = os.path.getsize(path)
        print(f"{elem}: {meret} byte")
        osszmeret += meret

print("Összesen:", osszmeret, "byte")
```