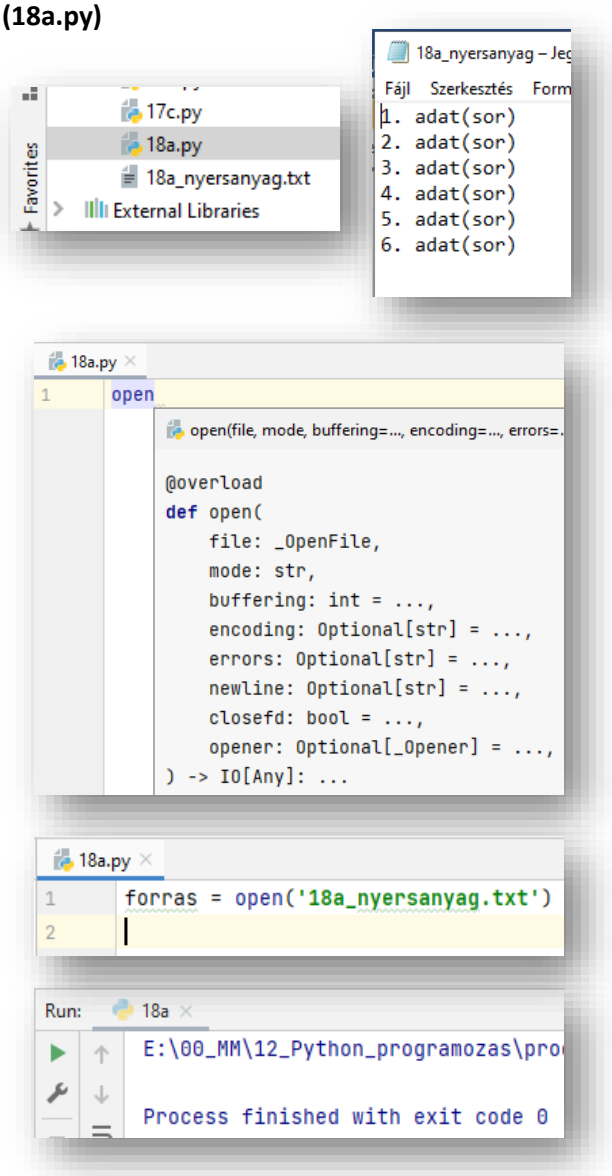
18.) FÁJLKEZELÉS PYTHONBAN

Az adatok tárolásának nagy jelentősége van a programozásban.
Eddig az adataink csak addig éltek, amíg a programunk futott. Amikor a program futását befejeztük, az addig használt adatok elvesztek.
Ha tartósan szeretnénk rögzíteni adatainkat, annak két módszere van. Az egyik, ha fájlban tároljuk ezeket, a másik, ha adatbázist használunk. Ebben a fejezetben az elsőt alkalmazzuk.
Nézzük meg, hogy egy python programból hogyan érhetünk el egy **txt** vagy **csv** fájlt, tehát hogyan tudunk beolvasni, majd hogyan tudjuk feladatunk végeztével kiíratni txt, csv fájlba a végeredményt.
(A következő feladatok megoldásához előkészített nyersanyagokat (txt, csv) használunk fel.)

Adatok beolvasása

(18a.py)

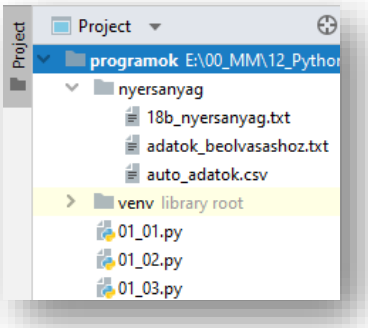


Készítsünk programot, melyben **beolvasunk** a programba egy előre elkészített txt fájlt!
Ennél a feladatnál a **forrásfájlt és a programfájlt azonos mappában helyezzük el**!

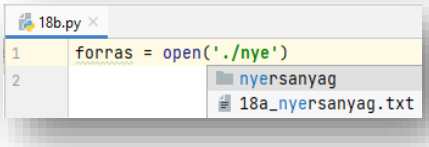
- Először másoljuk a nyersanyagot abba a mappába, ahol a programfájljainkat tároljuk.
- Majd hozzuk létre mellé a 18a.py program fájlt!
- A fájl elérése és megnyitása python programozásban az **open függvény** segítségével történik.
- Ha PyCharb-ban dolgozunk, akkor Ctrl + Shift +I billentyűkombinációval elérhetjük az **esetleges paraméterek**et, amelyeket felhasználhatunk.
- Mi a file, a mode, és az encoding –ot fogjuk használni elsősorban.
- Tehát az **open parancs megadása után zárójelek között először sztring formátumban megadjuk a fájl pontos nevét**. Mivel azonos mappában van a nyersanyag és a forrásfájl, ezért csak aposztrófok között megadjuk a fájl nevét, kiterjesztéssel együtt!
- Az előző parancs létrehoz egy objektumot. Ahhoz hogy dolgozni tudjuk **egy változóba kell tennünk ezt az objektumot**. Ezért elé írunk egy változó nevet, jelen esetben „forras” néven.
- Az, hogy sikeres volt a megnyitás, az úgy fog kiderülni számunkra, ha lefuttatjuk a programot, és nem kapunk hibajelzést.
- Tehát ebben az esetben semmi látványos dolog nem történik, csak egyszerűen megnyitottunk egy txt fájlt!

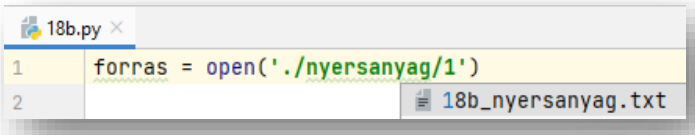
Az előző példában a forrásfájl azonos mappában volt a programfájllal. De programozás során általában több nyersanyaggal, fájlal dolgozunk. Ezért célszerű külön almappában tárolni ezeket a fájlokat. Nézzünk erre példákat!

(18b.py)



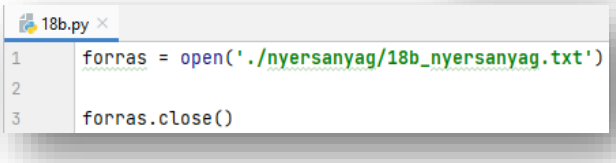
- Hozzunk létre egy „nyersanyagok” nevű mappát az eddigi programfájljaink mellé!
- Tegyük bele a 18b_nyersanyag.txt nevű fájlt a nyersanyag mappába!
- Ebben az esetben már nem elég csak a fájl nevét megadni, hanem ki kell egészíteni a **fájl elérési útjá**val!
- A könyvtárstruktúra arra a szintjére kell hivatkozni, ahol a fájl most található.
- A PyCharm segít a nyersanyag nevének kiválasztásában. **A hely megadásának az elején „./” után adjuk meg az almappa nevét, majd „/” karakter után a fájl nevét!**





Találkozhatunk olyan esettel, hogy felfelé kell haladnunk a mappánkban. Ez akkor fordulhat elő, ha programfájlunk helyezkedik el az almappában és a nyersanyag a főmappában van. Ilyen esetben „./../valami.txt” módon érhetjük el a nyersanyagunkat. Tehát két pont karakterrel haladunk egy mappányit felfelé!

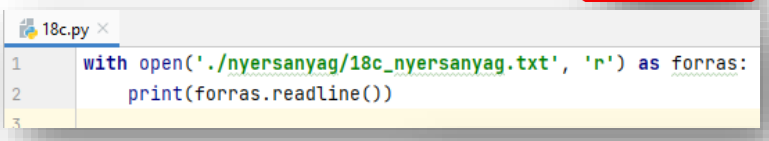
Nagyon fontos, ha megnyitunk egy nyersanyag fájlt, azt be is kell zárni.
Ennek akkor van jelentősége, ha nem csak olvassuk a fájlt, hanem írjuk is.

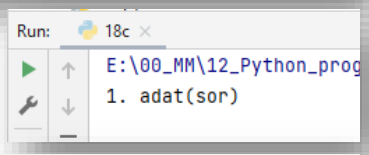


(18c.py)

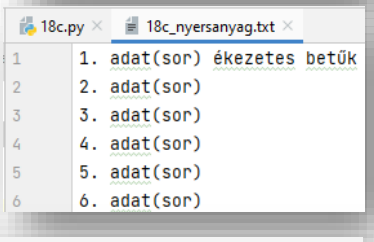
A pythonban létezik egy úgynevezett **context manager**, amellyel **egyszerűbben dolgozhatunk, ezért mi a későbbiekben ezzel fogunk dolgozni**!

- Ebben az esetben a „**with**” kulcsszóval kezdünk.
- Majd jöhet az **open függvény** az elérési úttal, mely után beírjuk az **„as” utasítást és az objektum nevét**.

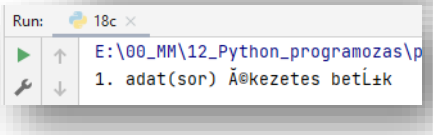




- Ebben az esetben nem kell bezárnunk a megnyitott fájlt, akár olvasásról, akár írásról van szó.
- Ha **csak olvasásra (read)** szeretnénk megnyitni a nyersanyagunkat, akkor az open utasításon belül, a fájl helyének megadása után vesszővel elválasztva megadjuk az **’r’ paramétert**.
- Aztán **olvassuk ki az első sor tartalmát a fájlból a readline() utasítással!**



- Ha a 18c_nyersanyag.txt fájl első sorába beírunk olyan szöveget, amely ékezetet tartalmaz, majd bezárás után futtatjuk a programunkat, akkor látszik, hogy hibásan értelmezi az ékezetes betűket.



- Az előző probléma megoldására egy **encoding='utf-8'** paraméterrel egészítjük ki az open függvényünket! (Ez Windows operációs rendszernél okoz problémát.)
- Az előző readline() utasítást másoljuk még le háromszor és futtasuk újból a programot!
- Itt látszik a, hogy a txt-ben lévő sor végi (most felesleges „/n') enterket is beolvassa. Üres sorokat is kapunk. (A későbbiekben, majd nézünk erre megoldást.)
- Van egy úgynevezett **fájlmutató,** ami „megmondja" nekünk bájtban, hogy éppen hol állunk a program futása során. Ehhez a **tell()** metódust használjuk. Ezeket az utasításokat másoljuk be a mintán látható helyekre, és futtassuk a programot!
- Azt látjuk, hogy kiírja a képernyőre, hogy hányadik karakternél járunk. Az eredmények bitszámokat jelentenek.
- Nem csak kiírathatjuk a helyeket, hanem **„ugrálhatunk" a pozíciók között a seek() utasítással**! Írjuk be a negyedik sorba az utasítást, és adjuk meg, hogy ugorjon vissza a 0. karakterpozícióba!
- Így ha futtatjuk a programot láthatjuk, hogy kétszer lesz kiírva az első sor!

18c.py
```
with open('./nyersanyag/18c_nyersanyag.txt', 'r', encoding='utf-8') as forras:
    print(forras.readline())
```

Run: 18c
```
E:\00_MM\12_Python_programozas\
1. adat(sor) ékezetes betűk
```

18c.py
```
with open('./nyersanyag/18c_nyer
    print(forras.tell())
    print(forras.readline())
    forras.seek(0)
    print(forras.tell())
    print(forras.readline())
    print(forras.tell())
    print(forras.readline())
    print(forras.readline())
```

Run: 18c
```
E:\00_MM\12_Python_programoz
0
1. adat(sor) ékezetes betűk

0
1. adat(sor) ékezetes betűk

31
2. adat(sor)
```

(18d.py)

A következő programban nyissuk meg a 18d_nyersanyag.txt fájlt a tanult módon.
A **readlines()** utasítással az összes sor beolvasásra kerül és egy listában megjelenítődik. Egy-egy sor, egy-egy lista elem, sztring formátumban. (A sor végi „/n" vezérlő karakterek is látszódnak.)

19d.py
```
with open('./nyersanyag/18d_nyersanyag.txt', 'r', encoding='utf-8') as forras:
    print(forras.readlines())
```

Run: 19d
```
E:\00_MM\12_Python_programozas\programok\venv\Scripts\python.exe E:/00_MM/12_Python_programozas/pro
['Kiss Balázs\n', 'Nagy Imre\n', 'Horváth Béla\n', 'Kovács Éva\n', 'Soós Mária\n', 'Balogh Ágota']
```

(18e.py)

Mentsük le másként az előző programot, és csak írjuk át a readlines() utasítást **read()**-re! Ebben az esetben egymás alá íratjuk ki a fájl sorait!
Ezen **utasításoknak a hátránya, hogy memória túlcsorduláshoz is vezethet**. **Ezért** nem ezeket az utasításokat fogjuk használni, hanem a **for ciklus segítségével soronként dolgozunk**!

19e.py
```
with open('./nyersanyag/18d_nyersanyag.txt', 'r', encoding='utf-8') as forras:
    print(forras.read())
```

Run: 19e
```
E:\00_MM\12_Pythor
Kiss Balázs
Nagy Imre
Horváth Béla
Kovács Éva
Soós Mária
Balogh Ágota
```