## Vállalati gyakorlat – 1.hét

### Python IDE választás:

A feladat elvégzéséhez a PyCharm-ot választottam. Ez az egyik legnépszerűbb, és legfejlettebb IDE a Pythonhoz. -letöltési oldal, telepítés

Néhány főbb funkciója:

* Intelligens kód-editor
* Debugger
* Teszt futtatás
* Refactor

A PyCharm elérhető az alábbi linken:

https://www.jetbrains.com/pycharm/

## Python átismétlése

#### Változók

A pythonban úgy adunk meg egy változót, hogy először megadjuk a nevét, majd egy „=” karaktert, utána pedig az értéket, amit a változóban tárolni szertnénk.

Néhány hasznos változó típus:

* int – egész számok
* float – tört számok
* str – szöveg, karaktersorozat, hosszabb elnevezése a string
* boolean – logiaki igaz / hamis (True/False)

#### String metódusok

A string típusú változók módosításához, vagy vizsgálatához többféle metódus áll rendelkezésünkre. Például:

* len(string) – megadja a string hosszát (karakterek számát)
* string.find( ”részlet”) – megkeresi az adott részletet a stringben
* string.count(”részlet”) – megszámolja az adott részlet előfordulásait a stringben

#### Típus konverzió

Egy változó típusát meg tudjuk változtatni úgy, hogy :

*változó\_neve = új\_típus(változó \_neve).*

Néhány konvertálás adatvesztéssel jár, például float típusból int-be konvertálásnál kerekítés történik egészre.

#### Felhasználói adatmegadás

Az input funkció használatával kérhetünk be adatokat a felhasználótól. Az input használata a következő:

*változó = input(”Szöveg, ami leírja milyen adatot kérünk, ez bármi lehet”)*

Ez alap esetben string típusú változót ad vissza, de a típuskonverzió használatával ezt egyszerűen tudjuk konvertálni.

#### Matematikai funkciók

A math könyvtár használatával különböző matematikai feladatokat tudunk egyszerűen elvégezni. Ehhez importálni kell a könyvtárat, amihez csak annyit kell tennünk, hogy a kód elején beírjuk a következőt:

*import math*

Néhány matematikai funkció ami így elérhetővé válik:

* fel és le kerekítés
* abszolút érték
* hatványozás
* gyökvonás
* maximum és minimum keresés

#### If statement

Az if használatával meg tudjuk határozni, hogy milyen feltételeknek kell teljesülni ahhoz, hogy egy kód lefusson. Használata:

if a > b:  
 print(str(a) + "nagyobb mint" + str(b))  
else:  
 print(str(b) + "nagyobb mint" + str(a))

Ez a kód eldönti, hogy melyik a nagyobb szám, és kiírja azt egy print() segítségével. Fontos, hogy a kódba meglegyen az if utáni sor behúzása, mivel a python ez alapján dönti el, hogy hova tartozik egy kódrészlet.

#### Logikai operátorok

A pythonban 3 logikai operátort tudunk használni:

* AND – ÉS operátor, akkor lesz „True” a kifejezés, ha mindkét oldala „True”
* OR – VAGY operátor, akkor lesz „True” a kifejezés, ha bármelyik oldala „True”
* NOT – NEM operátor, a „True” értékből „False” értéket, a „False” értéből „True”-t csinál.

#### Ciklusok

A pythonban kétféle egyszerű ciklus van, ezek a FOR és WHILE ciklusok. A FOR ciklus egy sorozat minden tagjához végre tudunk hajtani egy utasítást. Példa a FOR ciklusra:

szinek = ["piros", "kék", "zöld", "fehér", "fekete", "barna"]  
  
for i in range(0, len(szinek)):  
 print(szinek[i])

Ez a kód kiírja a szinek lista tartalmát.

A WHILE ciklus addig fut, ameddig a megadott feltétel igaz. Példa WHILE ciklusra:

while\_int = 0

while while\_int < 100:  
 while\_int += 1  
 print(while\_int)

Ez a kód kiírja a while\_int változót, és hozzáad egyet, amíg a változó kisebb mint 100.

#### Listák

A listák egy változóban több elemet tárolnak. Egy listát úgy hozunk létre, hogy szögletes zárójelek közé, vesszővel elválasztva felsoroljuk az elemeket amiket a listába szeretnénk tenni. Példa:

szinek = ["piros", "kék", "zöld", "fehér", "fekete", "barna"]

Egy listán többféle műveletet is végezhetünk, például:

* lista.append(elem) – A listához hozzáadja a megadott elemet
* lista.remove(elem) – A listából eltávolítja a megadott elemet
* lista.insert(szám, elem) – A listába megadott helyre (szám) szúrja be az elemet
* lista.sort() – A listát rendezi
* lista.pop() – Az utolsó elem kivétele

Ezeknek a kódja az előbbi szinek listához:

szinek.append("lila")  
szinek.remove("barna")  
szinek.pop()  
szinek.insert(0, "sárga")  
szinek.sort()

#### Mátrixok

A pythonban a mátrixokat listák listájával tudjuk létrehozni. A kapott adatszerkezetből az adatokhoz indexek segítségével tudunk hozzáférni. Kódban ez így néz ki:

test\_matrix = [['Elem1', '001', 1], ['Elem2', '002', 2], ['Elem3', '003', 3], ['Elem4', '004', 4], ['Elem4', '004', 4]]

Ez a kód létrehozott egy mátrixot. A mátrix egy sorát így íratjuk ki (ez a kód a 3. sort írja ki):

print(test\_matrix[2])

Egy sor egy elemét így íratjuk ki (ez a kód a 3.sor első elemét írja ki):

print(test\_matrix[2][0])

## Vállalati gyakorlat – 2.hét

### Python átismétlésének folytatása

#### Funkciók

A funkció egy kód blokk, ami akkor fut le, amikor meghívják a programba. Egy funkció adatokat ad vissza eredményül. Ezekbe a funkciókba adatokat is be lehet adni.

Példa funkciókra:

def test\_function():  
 print("Ez a function meg lett hívva!")  
  
  
def osszeadas(szam1, szam2):  
 eredmeny = szam1 + szam2  
 print(str(eredmeny))

Ez a kód létrehozta a test\_function nevű funkciót, és az osszeadas nevű funkciót. Mivel még nem lettek meghívva, ezért ez a kód nem adna ki semmit. A meghívás kódja:

test\_function()  
osszeadas(15, 20)

Ezzel már lefutnak a funkciókban lévő kódok. Az osszeadas funkciónak meg kell adni két számot ahhoz, hogy működjön, míg a test\_function nem kér a adatokat.

#### Osztályok és eljárások

Az osztályok olyanok, mint az objektum konstruktorok. Az osztályok tartalmazhatnak metódusokat is, amik olyan funkciók, amik az osztályhoz tartoznak. Példa:

class Auto:  
 def \_\_init\_\_(self, evjarat, marka, uzemanyag):  
 self.evjarat = evjarat  
 self.marka = marka  
 self.uzemanyag = uzemanyag  
  
 def autokiiras(self):  
 print("Az auto evjarata: " + str(self.evjarat) + "\n" +  
 "Az auto markaja: " + self.marka + "\n" +  
 "Az auto uzemanyaga: " + self.uzemanyag)

Ez a kód létrehozza az Auto osztályt és az autokiiras metódust, amit ha meghívunk, akkor kiírja az objektum adatait. Egy új, Auto osztályú objektum létrehozása, és kiírása a metódus segítségével:

auto1 = Auto(2005, "Ford", "dízel")  
  
auto1.autokiiras()

#### Python adattípusok

A pythonba négy beépített adattípus van, amiket adatok tárolására használunk. Ezek a következőek:

* List – Lista
* Tuple – Rendezett véges lista, nem változtatható
* Set – Halmaz , nem indexelt, nem rendezett, és az elemei nem változtathatóak, de törölhetőek, és újak is hozzáadhatóak
* Dictionary – Szótár, az adatokat kulcs – érték párokban tárolja, az értékek változtathatóak, és az adatszerkezet nem engedi meg, hogy két kulcs ugyan az legyen. A Python 3.7 óta rendezett is

Ezeket az adattípusokat kódban így hozzuk létre:

lista = ["alma", "banán", "körte"]

rendezett\_veges\_lista = ("alma", "banán", "körte")

szotar = {"nev": "Neve", "kor": 21}

halmaz = {"alma", "banán", "körte"}

#### Fájlkezelés, beolvasás, és kiírás

A pythonba többféle műveletet is végre tudunk hajtani fájlokon. Először is meg kell nyitni, ehhez az open() funkciót használjuk. A megnyitásnak négy fajtája van:

* Read – Olvasás, ebben a módban nem tudjuk módosítani a fájlt
* Write – Írás, ebben a módban módosítani tudjuk a fájlt, ha nem létezik a megadott fájl, akkor létrehozza.
* Append – Hozzáadás a fájlhoz
* Create – Létrehoz egy fájlt

Példa fájlok beolvasására, és fájlba kiírásra (a minta.txt a program futása előtt is létezett, és számokat tartalmaz):

file = open("minta.txt", "r")  
  
szamok = file.read().split()  
  
for i in range(len(szamok)):  
 szamok[i] = int(szamok[i])  
  
print(szamok)  
  
file.close()  
  
file2 = open("letrehozott.txt", "w+")  
for i in range(len(szamok)):  
 dupla = szamok[i]\*2  
 file2.write(str(dupla) + " ")

file2.close()

A kód először megnyitotta a minta.txt fájlt olvasás módba, majd a tartalmát a szamok nevű tömbbe rakta, ezután a minta.txt file be lett zárva. Mivel nem létezett a letrehozott.txt fájl a program futása előtt, ezért létrejött, és a számok listában lévő számok duplája egy space karakterrel elválasztva belekerültek a letrehozott.txt fájlba. Végül a letrehozott.txt fájl is be lett zárva.

#### Adatbázis kapcsolat

A python sqlite3 modulja lehetővé teszi, hogy SQLite adatbázisokkal dolgozzunk egy Python programból.

Kapcsolódás egy adatbázishoz (amennyiben nem létezik, létrehozza):

import sqlite3  
  
conn = sqlite3.connect('test.db')  
  
print("Opened database successfully")

A test.db adatbázisban egy termekek nevű tábla létrehozása, és egy sor beszúrása:

conn.execute(  
 '''  
 CREATE TABLE TERMEKEK  
 (  
 ID INT PRIMARY KEY NOT NULL,  
 NAME CHAR(50) NOT NULL,  
 PRICE INT NOT NULL,  
 STOCK INT,  
 SOLD INT  
 );  
 '''  
)

conn.execute("INSERT INTO TERMEKEK (ID,NAME,PRICE,STOCK,SOLD) \  
 VALUES (1, 'Toll', 200, 1568, 259 )")

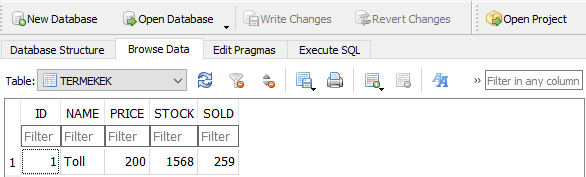
conn.commit()

conn.close()

Ezek után ki tudjuk írni az adatbázis tartalmát is:

conn = sqlite3.connect('test.db')  
  
cursor = conn.execute("SELECT ID,NAME,PRICE,STOCK,SOLD from TERMEKEK")  
for row in cursor:  
 print("ID = ", row[0])  
 print("NAME = ", row[1])  
 print("PRICE = ", row[2])  
 print("STOCK = ", row[3])  
 print("SOLD = ", row[4], "\n")  
conn.close()

A program működésének ellenőrzésére a DB Browser for SQLite nevű programot használtam (a képernyőképen látható, hogy az adatbázis, és a tábla létrejött, és az adatok beszúrásra kerültek).



A DB Browser for SQLite program elérhető itt: <https://sqlitebrowser.org/>

#### Vállalati gyakorlat – 3.hét

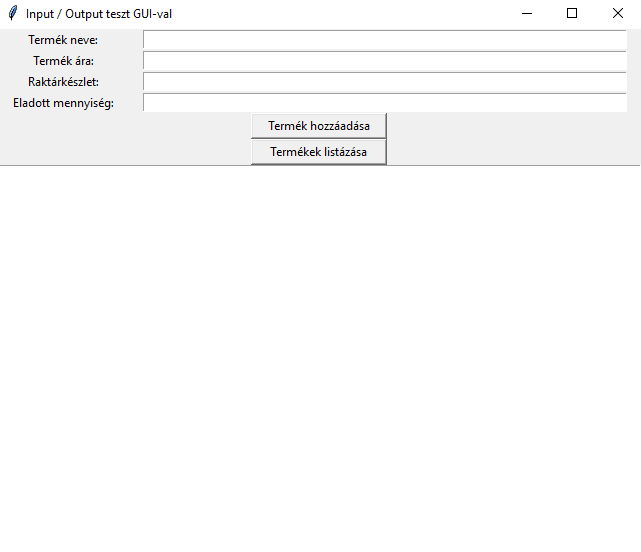
#### Tkinter GUI

A tkinter az alap Python GUI keretrendszer. Mivel alapból bent van a Pythonba, ezért minden rendszeren működni fog, és a rendszernek megfelelően fog megjelenni.

Egy példa egy Tkinter GUI-ra (ez a rész még nincsen kész):

import tkinter as tk  
  
  
def hozzaad():  
 nev = nev\_entry.get()  
 ar = ar\_entry.get()  
 raktar = raktar\_entry.get()  
 eladott = eladott\_entry.get()  
 aru = [nev , ar, raktar, eladott]  
 print(aru)  
 arulista.append(aru)  
 print(arulista)  
  
def kiiras():  
 kiir\_text\_field.delete(1.0, tk.END)  
 for i in arulista:  
 for e in range(0, 4):  
 kiir\_text\_field.insert(tk.END, i[e] + " ")  
 if e == 3:  
 kiir\_text\_field.insert(tk.END, i[e] + "\n")  
  
  
window = tk.Tk()  
window.title("Input / Output teszt GUI-val")  
arulista = []  
tk.Label(window, text="Termék neve:").grid(row=0)  
tk.Label(window, text="Termék ára:").grid(row=1)  
tk.Label(window, text="Raktárkészlet:").grid(row=2)  
tk.Label(window, text="Eladott mennyiség:").grid(row=3)  
  
nev\_entry = tk.Entry(window, width=80)  
ar\_entry = tk.Entry(window, width=80)  
raktar\_entry = tk.Entry(window, width=80)  
eladott\_entry = tk.Entry(window, width=80)  
btn\_hozzaad = tk.Button(window,text="Termék hozzáadása",width=18,command=hozzaad)  
btn\_kiir = tk.Button(window,text="Termékek listázása",width=18,command=kiiras)  
kiir\_text\_field = tk.Text(window)  
  
nev\_entry.grid(row=0, column=1)  
ar\_entry.grid(row=1, column=1)  
raktar\_entry.grid(row=2, column=1)  
eladott\_entry.grid(row=3, column=1)  
btn\_hozzaad.grid(row=4, columnspan=2)  
btn\_kiir.grid(row=5, columnspan=2)  
kiir\_text\_field.grid(row=6, columnspan=2)  
  
window.mainloop()

Ez a kód a következő GUI-t hozza létre:



## Vállalati gyakorlat – 4.hét

### Adatbázis feltöltése adatokkal, és az adatok lekérése Python alkalmazás segítségével.

Az előző GUI példát módosítottam úgy, hogy es SQLite adatbázisba mentse el az adatokat. Ezek az adatok a program futtatása után is megmaradnak. Ez a változat hozzáadni, és kiírni tudja az adatokat. (A kódban kikommentelt rész a táblát hozza létre, amibe az adatokat menti)

#### Példa:

import tkinter as tk  
import sqlite3  
  
  
def ab\_hozzaad():  
 conn = sqlite3.connect('test.db')  
  
 nev = nev\_entry.get()  
 ar = ar\_entry.get()  
 raktar = raktar\_entry.get()  
 eladott = eladott\_entry.get()  
  
 script = "INSERT INTO TERMEKEK (NAME,PRICE,STOCK,SOLD) \  
 VALUES ('{}', {}, {}, {})".format(nev, ar, raktar, eladott)  
  
 conn.execute(script)  
  
 conn.commit()  
 conn.close()  
  
  
def ab\_kiiras():  
 conn = sqlite3.connect('test.db')  
  
 cursor = conn.execute("SELECT NAME,PRICE,STOCK,SOLD from TERMEKEK")  
 kiir\_text\_field.delete(1.0, tk.END)  
 for row in cursor:  
 print("NAME = ", row[0])  
 print("PRICE = ", row[1])  
 print("STOCK = ", row[2])  
 print("SOLD = ", row[3], "\n")  
 kiir\_text\_field.insert(tk.END, "Termék neve: " + str(row[0]) + ", termék ára: " + str(row[1]) + ", raktárkészlet: " + str(row[  
 2]) + ", eladott mennyiség: " + str(row[3]) + "\n")  
  
 conn.close()  
  
  
# conn = sqlite3.connect('test.db')  
#  
# conn.execute(  
# '''  
# CREATE TABLE TERMEKEK  
# (  
# NAME CHAR(50) NOT NULL,  
# PRICE INT NOT NULL,  
# STOCK INT,  
# SOLD INT  
# );  
# '''  
# )  
  
  
window = tk.Tk()  
window.title("Input / Output teszt GUI-val")  
arulista = []  
tk.Label(window, text="Termék neve:").grid(row=0)  
tk.Label(window, text="Termék ára:").grid(row=1)  
tk.Label(window, text="Raktárkészlet:").grid(row=2)  
tk.Label(window, text="Eladott mennyiség:").grid(row=3)  
  
nev\_entry = tk.Entry(window, width=80)  
ar\_entry = tk.Entry(window, width=80)  
raktar\_entry = tk.Entry(window, width=80)  
eladott\_entry = tk.Entry(window, width=80)  
btn\_hozzaad = tk.Button(window, text="Termék hozzáadása", width=18, command=ab\_hozzaad)  
btn\_kiir = tk.Button(window, text="Termékek listázása", width=18, command=ab\_kiiras)  
kiir\_text\_field = tk.Text(window)  
  
nev\_entry.grid(row=0, column=1)  
ar\_entry.grid(row=1, column=1)  
raktar\_entry.grid(row=2, column=1)  
eladott\_entry.grid(row=3, column=1)  
btn\_hozzaad.grid(row=4, columnspan=2)  
btn\_kiir.grid(row=5, columnspan=2)  
kiir\_text\_field.grid(row=6, columnspan=2)  
  
window.mainloop()

## Vállalati gyakorlat – 5.hét

### Adatok frissítése, és adatok törlése

Az előző kódot bővítve lehetővé tettem az adatok módosítását, és törlését is név alapján. Ezek a változások az adatbázisban módosítják a táblákat.

### Példa:

def ab\_termek\_torles():  
 conn = sqlite3.connect('test.db')  
  
 nev = nev\_entry.get()  
  
 script = """DELETE FROM TERMEKEK WHERE NAME = ?"""  
  
 conn.execute(script, (nev,))  
  
 conn.commit()  
 conn.close()

def ab\_termek\_update():  
 conn = sqlite3.connect('test.db')  
  
 nev = nev\_entry.get()  
 ar = ar\_entry.get()  
 raktar = raktar\_entry.get()  
 eladott = eladott\_entry.get()  
 idnum = id\_entry.get()  
 kategoria\_id = kategoria\_id\_entry.get()  
  
 script = """UPDATE TERMEKEK SET   
 NAME = ? ,  
 PRICE = ? ,  
 STOCK = ? ,  
 SOLD = ? ,  
 KATEGORIA\_ID = ?  
 WHERE ID = ?"""  
  
 if nev != "" and ar != 0 and raktar != "" and eladott != 0 and kategoria\_id != 0 and idnum != 0:  
 conn.execute(script, (nev, ar, raktar, eladott, kategoria\_id, idnum))  
 conn.commit()  
  
 conn.close()

## Vállalati gyakorlat – 6.hét

### Több ablak használata, több tábla összekapcsolása

A tkinter segítségével több ablakot is létre lehet hozni, ezért a könnyebb használat érdekében a külön az adatbázis külön tábláinak módosítására, és kiírására külön ablakokat hoztam létre. (Ebben a szakaszban a két tábla: TERMEKEK, KATEGORIAK) Az sqlite3 használatával könnyen lehet több táblát, és az azok közötti kapcsolatot is létrehozni. A kapcsolat létrehozásához a termékek táblába került egy KATEGORIA\_ID nevű külső kulcs (FOREIGN\_KEY).

### Példa

Új tábla létrehozása az adatbázisban:

conn = sqlite3.connect('test.db')

conn.execute(  
 '''  
 CREATE TABLE KATEGORIAK  
 (  
 ID INT NOT NULL PRIMARY KEY,  
 NAME CHAR(50) NOT NULL  
 );  
 '''  
)

## Vállalati gyakorlat – 7.hét

### A program részekre bontása, tkinter gemoetry manager

A teljes program eddig a „main.py” nevű fájlba volt írva. Ezt most szétszedtem, a könnyebb átláthatóság miatt. A „main.py” a program indításakor létrejövő ablakért felelős, onnan lehet kiválasztani a többi opciót. Az első a „termekablak.py”, ami a TERMEKEK tábla adatainak kezelésére használható, új ablakot nyit meg. A második a „kategoriaablak.py” ami a KATEGORIAK tábla adatainak a kezelésére használható.

Azért, hogy ez működjön, importálni kellett a „termekablak.py” és a „kategoriaablak.py”-t a „main.py” fájlba, a következő kóddal:

import termekablak  
import kategoriaablak

A „graph.py” jelenleg nem használt, később az adatok grafikus ábrázolásáért lesz felelős.

A GUI az egyszerűbb használat érdekében egy kicsit változtatva lett, a tkinter grid geometry manager használatával. Bővebb információ erről: https://www.pythontutorial.net/tkinter/tkinter-grid/

## Felhasznált források:

<https://www.tutorialspoint.com/sqlite/sqlite_python.htm>

<https://www.w3schools.com/>

<https://realpython.com/python-gui-tkinter/>

<https://docs.python.org/3/>