Aufgaben für die 11. Übung zur LV "Grundlagen der Informatik"
Thema: Python, Teil 6 (Dateiarbeit: EA in/aus Files / Liste von Listen)

****

→ In der 11. Übung benutzen wir die IDLE (Python GUI)

****

21) Dateiarbeit in Python (Ausgabe in Files / Lesen von Files) *(mit IDLE)*

a) Vorbemerkung zu Files und Übung im interaktiven Modus

****

→ Vorbemerkungen und Übungen im interaktiven Modus sind STETS als Vorbereitung VOR der Übung durchzuarbeiten.

****

Die Dateiarbeit (Arbeit mit Files) wurde in der 9. Vorlesung (Seite 10-14) behandelt. Dieser Stoff wird in den folgenden Aufgaben vorausgesetzt.

Python betrachtet Files (=Dateien) allgemein als Sequenz von Zeichen (1 Byte/Zeichen), die mit den *built-in*-Funktionen nur als Zeichenkette gelesen oder geschrieben werden können.
Mit *fileObjekt=open(<filebezeichner>,<modus>)* wird die Datei *<filebezeichner>* geöffnet und ein neues Fileobjekt mit dem Namen *fileObjekt* erzeugt, das mit dieser Datei verknüpft ist. Mit einem Fileobjekt ist stets ein „Dateizeiger“ verbunden, der nach dem Öffnen auf den Anfang der Datei (Position 0L) zeigt (zeigt stets auf das aktuelle Zeichen der Datei; die zeichenweise Zählung der Datenzeiger-Position beginnt bei 0). Die aktuelle Position des Dateizeigers kann man über die Funktion *fileObjekt.tell( )* abfragen. Man kann den aktuellen Dateizeiger über die Funktion *fileObjekt.seek(<zeichennummer>)* auf eine neue aktuelle Position einstellen.

Die wichtigsten eingebauten („*built-in*“) Funktionen zum Lesen/Schreiben von Dateien wie *read*, *readline*, *readlines* bzw. *write*, *writelines* wurden in der 9.Vorlesung behandelt. Auch auf die Ausgabe in Files mit *print* durch Umlenkung wurde eingegangen. Außerdem wurde auch das Lesen/Schreiben von Dateien mit den Funktionen *load/dump* aus dem Modul *pickle* vorgestellt (siehe 9. Vorlesung in Vorbereitung der Übung).

- Wir üben jetzt dazu kurz im interaktiven Modus der IDLE (hier ein File mit Textinhalt)**:**

```
>>> f=open(r'W:\test.out','w')     #Öffnen eines neues Files zum Schreiben ('w')
                                   #Filebez. ist wegen \ als raw-String zu schreiben !!!
>>> type(f)                        #f ist vom Typ file
>>> f.tell()       #Ausgabe der akt. Position des "Dateizeigers" (hier: 0L)
>>> f.write('Freiburk/Sachsen')  #Schreiben in File (mit Fehlern)
>>> f.tell()       # Ausgabe der akt. Position des "Dateizeigers" (hier:16L)
>>> f.seek(5)      #Einstellen der Position des Dateizeigers auf Zeichen "u"
>>> f.write('erg')#Überschreiben ab "u"
>>> f.close()      #Schließen des File
>>> f=open(r'W:\test.out','r')     #Öffnen eines existierenden Files zum Lesen ('r')
>>> text=f.read()
>>> text           #Ausgabe des korrigierten Textes: 'Freiberg/Sachsen'
>>> f.close()
```

#Öffnen Sie das geschriebene File *test.out* zur Kontrolle mit einem beliebigen Editorprogramm.

b) Aufgabe 21.1: Kopieren eines Festplattenfiles mit Hilfe von Python *(mit IDLE)*

- Im Übungsverzeichnis *Ueb11_Python6* finden Sie ein ASCII-Textfile *original.txt*. **Kopieren Sie dieses File nach W:**. Sehen Sie sich dann den Inhalt dieses Files mit einem beliebigen Editorprogramm an (z.B. Notepad/Editor).

- Schreiben Sie ein vollständiges Python-Programm, dass von diesem File *W:\original.txt* eine (Festplatten-)Kopie *W:\original.bac* anlegt.
(Lesen Sie dazu im Python-Programm *W:\original.txt* ein, und schreiben Sie die eingelesenen Daten als Kopie *W:\original.bac* in Ihr Arbeitsverzeichnis. Verwenden Sie hier die built-in Funktionen *read* und *write*.)

Sehen Sie sich danach zur Kontrolle den Inhalt der von Python erzeugten Kopie wieder mit einem beliebigen Editorprogramm an (z.B. Notepad/Editor).

P.S.: Natürlich könnte man die Kopie z.B. auch mit *readlines* und *write* anlegen.
---

c) Aufgabe 21.2: Einlesen eines Messwertdatenfiles von Festplatte, Bilden der Summe der Messwerte sowie Schreiben der eingelesenen Messwerte und der Summe in ein neues Festplattenfile *(mit IDLE)*

- **Vorbemerkung:** Wie weiter oben erwähnt und in der Vorlesung behandelt, betrachtet Python Files (=Dateien) als Sequenz von Zeichen, die mit den *built-in*-Funktionen nur als Zeichenkette (String) gelesen oder geschrieben werden können.
Will man also *Zahlen* aus einem File einlesen und dann verarbeiten, so muss man nach dem Lesen den Zahlenstring durch *int( )* bzw. *float( )* in den geforderten Zahlentyp konvertieren. Beim Schreiben von Zahlen in Files sind sie vorher mit *str( )* in eine Zeichenkette zu konvertieren.

- Im Übungsverzeichnis *Ueb11_Python6* finden Sie ein ASCII-Textfile *zahlen.txt,* das reelle Messwerte (1 Messwert pro Zeile) enthält. **Kopieren Sie dieses File nach W:**. Dieses File könnte von einem Datenerfasser eingegeben oder automatisch von einem Messgerät aufgezeichnet worden sein. Sehen Sie sich zunächst den Inhalt dieses Files mit einem beliebigen Editorprogramm an (z.B. Notepad/Editor).

- **Aufgabe**: Schreiben Sie ein vollständiges Python-Programm, dass alle Messwerte aus *W:\zahlen.txt* in eine Liste reeller Zahlen einliest und danach die Summe der reellen Zahlen dieser Liste bildet. Schließlich sind sowohl die reellen Messwerte als auch die berechnete Summe in ein neues File *W:\MesswSum.txt* zu schreiben.

Sehen Sie sich danach zur Kontrolle den Inhalt des Files *W:\MesswSum.txt* mit einem beliebigen Editorprogramm an (z.B. Notepad/Editor).
---

22) *(→ Wenn die Zeit in der Übung nicht reicht, ist der Rest Hausaufgabe)* **Einfache Matrizenberechnungen** *(mit IDLE)*

(vgl. Aufg.-komplex Struktogramme, Aufg.18)

- Vorbemerkung: Die Komponenten von Listen können nicht nur (wie bisher in einigen Aufgaben) von einfachem Typ wie *int* und *float* sein, sondern können wiederum Listen oder auch Dictionaries sein.
Für eine ***reelle Matrix*** eignet sich in Python als Datenstruktur eine ***Liste von Listen***. Jede (Unter-)Liste in der Liste stellt dabei eine Matrixzeile (oder auch eine Matrixspalte) dar.

- Wir üben dazu zunächst kurz im **interaktiven Modus** (***vor der Übung abzuarbeiten***):
Im Folgenden soll interaktiv folgende reelle (2x4)-Matrix als Liste von (Zeilen-)Listen eingegeben und zur Kontrolle wieder ausgegeben werden:

```
                        1.0   2.0   3.0   4.0
                        5.0   6.0   7.0   8.0
```

```
------------
>>> matrix=[]               #Matrix als leere Liste initialisieren
>>> zeile=[1.0,2.0,3.0,4.0]       #Zuweisung der 1.Matrixzeile
>>> matrix.append(zeile)   #Anhaengen der 1.Zeile an (leere) Matrix
>>> matrix                        #Unformatierte Kontrollausgabe
>>> zeile=[5.0,6.0,7.0,8.0]       #Zuweisung der 2.Matrixzeile
>>> matrix.append(zeile)          #Anhaengen der 2.Zeile an Matrix
>>> matrix          #Unformatierte Ausgabe der (2x4)-Matrix (Liste von Listen!)
>>> for i in range(2):       #Formatierte Ausgabe der (2x4)-Matrix
        for j in range(4):
            print "%.2f " %matrix[i][j], #Komma-->kein Zeilenwechsel
        print                #Zeilenwechsel nach Ausgabe einer Zeile
-----------
```

- **Aufgabe:** Eine reelle (n, m)-Matrix mit n Zeilen und m Spalten ist in einer gesonderten Funktion zeilenweise von Tastatur einzulesen (n>=2; m>=3). Die Zeilenanzahl n und die Spaltenanzahl m seien dem Nutzer bekannt und sind im Hauptprogramm vor dem Aufruf der Eingabefunktion von Tastatur einzulesen.
- Zur Kontrolle (und zur Übung) ist die eingelesene Matrix in einer gesonderten Funktion auf den Bildschirm zeilenweise auszugeben.
- Danach ist in einer gesonderten Funktion die Summe über alle Elemente der 2. und 3. Spalte der Matrix zu ermitteln, die berechnete Summe als Resultat an das Hauptprogramm zurückzugeben und dort auf den Bildschirm auszugeben.
Es ist nicht mit globalen Variablen zu arbeiten.
(In der oben genannten Struktogrammaufgabe wurde außerdem noch die transponierte Matrix berechnet. Dieser Teil entfällt hier aus Zeitgründen, kann aber fakultativ behandelt werden.)

→ Vorgehen zur Lösung:
Öffnen Sie aus dem Übungsverzeichnis das unvollständige **Teilprogramm** *Y:\Lehre\LV_Gr_d_Inf_WS0607\Ueb11_Python6\matrixberechStartfile.py* . Die Funktion zur Matrixeingabe und ihr Aufruf sind darin schon vollständig enthalten.
Ergänzen Sie dieses Startfile zum vollständigen Programm.

Es folgt der Ausdruck des unvollständigen Startfiles *matrixberechStartfile.py*

```
# -*- coding: cp1252 -*-
# Aufg. 19: Eingabe einer reellen (nxm)-Matrix und Kontrollausgabe
#           sowie Summe der 2. und 3. Spalte
##########################################
def matrixEingabe(n,m):
    #Zeilenweise Eingabe einer (nxm)-Matrix
    matrix=[]
    for i in range(n):
        zeile=[]
        print "\nEingabe der %i. Zeile:"%(i+1) #Zeilenindex von 0 bis n-1
        #Nach außen wegen besserer Verständlichkeit: Zeilenindex von 1 bis n
        for j in range (m):
            print "   matrix[%i][%i] = " %(i+1,j+1),
            zeile.append(input())
        matrix.append(zeile)        # Anfuegen als UNTERliste
    return matrix
##########################################
#Funktion zur Berechnung der Summe der 2. und 3. Matrixspalte
#SELBST SCHREIBEN
##########################################
#Kontrollausgabe der eingelesen Matrix auf den Bildschirm
#SELBST SCHREIBEN
##########################################
#Hauptprogramm:
print "Eingabe der Zeilen- und Spaltenanzahl der Matrix:"
n=input("   Zeilenanzahl  (>=2): ")  #ohne Eingabeüberprüfung
m=input("   Spaltenanzahl (>=3): ")  #ohne Eingabeüberprüfung
a=matrixEingabe(n,m)                 #rechte Seite: Funktionsaufruf
print "\nKontrollausgabe der eingelesenen Matrix:"
#SELBST HIER FUNKTIONSAURUF EINFÜGEN
#Resultat (Summe der 2. und 3.Spalte):
#SELBST HIER AUFRUF DER FUNKTION UND RESULTATAUSGABE SCHREIBEN
```
