Lab 10: File I/O

We write programs to process data. Data stored in a program is temporary. Once we close the program, data is lost. To permanently store data, we need to use files. Data stored in a file can later be read into programs again.

File I/O stands for file input/output. To save data to a file is called "output" data to a file, AKA "write to file". The opposite process of write to file is "input", AKA, read from a file. Whether to read from a file, or write to a file, we need to obtain a file object that connects to the file. A file object is created using the "open" function.

The general form of creating a file object

<file_object> = open(<file_name>, <mode>)

Notes on the general form of obtaining a file object:

- The open function takes two values: <file_name>, and <mode>.
- <file_name> is a string value, as in the name of the file, with which <file_object> is associated with.
- <mode> is also a string value specifying the nature of the file object – either for reading or writing.

Example 1 – write data to file

Create a Python script with the name "write_data.py", and type up the following code.

```
def write_to_file():
    # Create a file object named "writefile"
    # The "writefile" object is associated with a file by the name "data.txt"
    # The "writefile" object is specified for the purpose of writing data
    # to the file "data.txt" (indicated by the parameter 'w')
    writefile = open('data.txt', 'w')

    # Write a line of string to the file, then start a new line ('\n')
    writefile.write("Hello, Python\n")
    # Write another line of string to the file
    writefile.write("Python is fun")

    writefile.close() #close the file object

    # Note, the file data.txt is created in the same folder
    # as the script write_data.py

def main():
    write_to_file()

main()
```

In this example, the function write_to_file first creates a file object "writefile" by calling the open() function. Points to note about the open() command:

- The file object "writefile" is connected to a file named data.txt

- The file object is specified for the purpose of writing data to the file
- Because there is no file with the name data.txt exist in this case, the open() function creates a file with the name data.txt
- The file data.txt is by default created in the current folder, i.e., the same folder as the Python script file write_data.py.

To use the "writefile" file object, the "write" function is called twice, and two lines of text are saved to the file. Note that, a new line character ('\n') was inserted to break the text into two lines in the file. Also note that, the "writefile" file object has to be closed in the end by calling the close() function on "writefile" object.

Running the above script, go to the folder where the script file write_data.py was created, and examine the content of the file data.txt.

Example 2 – the "with-as" statement to close file object automatically

The "with-as" statement creates the file object on the header of a code block. The file object is closed at the end of the code block, hence, calling the close() function on the file object is no longer needed.

Modify the previous script as follows:

```
def write_to_file():
    # Create a file object "writefile" on the "with-as" header
    with open('data.txt', 'w') as writefile:
        writefile.write("Hello, Python\n")
        writefile.write("Python is fun")

    ~~writefile.close()~~ # no longer needed

def main():
    write_to_file()

main()
```

Example 3 – read data from a file

In the same folder of write_data.py (the script file created in previous example), create a script read_and_display.py, and type up the following code. Caution: the following script must be created in the same folder as the text file data.txt, as Python by default searches for the text file in the current folder.

```
def read_and_display():
    # Create a file object "readfile" for reading (the mode parameter 'r')
    # object "readfile" is connected to the file data.txt
    with open('data.txt', 'r') as readfile:
       line = readfile.readline()
       print(line) # displays "Hello, Python"
       line = readfile.readline()
       print(line) # displays "Python is fun"

def main():
    read_and_display()

main()
```

The program uses the “with-as” statement to create a file object “readfile” associated with the file data.txt. The file object is specified for reading from the file (i.e., the mode parameter ‘r’). The readline() command called upon the file object reads a line of text from the file, and stores the text to the variable “line”, which is then printed to screen. The readline() command is called twice, because there are only two lines of text in the file.

Note that, calling the readline() the third time would cause error, because there is no more content to read, i.e., the end of the file has been reached.

Example 4 – read and write numeric data

By default, all data transferred to and from a file are in the format of string. Therefore, a conversion process needs to take place when processing numeric values. The following example first writes data of mixed types: string, integer, and float, to a text file. It then reads the file content back into the program.

Create another script read_write.py, and type up the code as follows:

```
def write_to_file():
    with open('num_data.txt', 'w') as writefile:
        writefile.write("Hello world\n") # write string
        writefile.write(str(10) + "\n") # write an integer
        writefile.write(str(20.5)) # write a float number

def read_from_file():
    with open('num_data.txt', 'r') as readfile:
        line = readfile.readline() # read as string
        print(line)

        num = int(readfile.readline()) # read as integer
        print("Read an integer:", num)

        read_num = float(readfile.readline()) # read as float number
        print("Read a float number:", real_num)

def main():
    write_to_file() # create the text file "num_data.txt"
    read_from_file() # read and display the file content

main()
```

In the write_to_file() function, the integer 10 and float number 20.5 were converted to string using the str() function before being written to the file. In the read_from_file() function, when the integer and float number were read from the file, they had to be converted back to their original data type by using int() and float() function respectively.

Example 5 – read and write datetime information

Datatime values are stored in a text file in the format of string. When datetime value are read into program, they have to be converted to datetime objects for easier processing.

Modify the previous example by adding the **code in bold**, as follows:

```
def write_to_file():
    with open('num_data.txt', 'w') as writefile:
        writefile.write("Hello world\n") # write string
```

```
        writefile.write(str(10) + "\n") # write an integer
        writefile.write(str(20.5)) # write a float number

        writefile.write("23-4-2017") # write a date as string (dd-mm-yyyy)

def read_from_file():
    with open('num_data.txt', 'r') as readfile:
        line = readfile.readline() # read as string
        print(line)

        num = int(readfile.readline()) # read as integer
        print("Read an integer:", num)

        read_num = float(readfile.readline()) # read as float number
        print("Read a float number:", real_num)

        from datetime import datetime
        date_str = readfile.readline() # read a date as string
        # convert date string to date object
        # date object will later be used in program for processing
        date_obj = datetime.strptime(date_str, '%d-%m-%Y')

def main():
    write_to_file() # create the text file "num_data.txt"
    read_from_file() # read and display the file content

main()
```

In the function write_to_file(), a date string is written to the file in the format dd-mm-yyyy. In read_from_file(), the date is first read in as string (variable 'date_str'). It's then converted to a date object using strptime() function. The converted date object would be used in the program later.

Example 6 – split a line of text into a list of values

Data stored in text files normally contain multiple lines of records, each line of text contains multiple values, separated by a delimiter.

In the following example, values written to a file are added to the same line, separated by a comma. When the line is read into program, the line is split into a list of strings. The string values of the list are converted to their actual data types (int, float, or date); the actual data values are then added to a record (as a list).

Create another script split_text.py, and type up the code as follows:

```
from datetime import datetime
def write_to_file():
    with open('data_record.txt', 'w') as writefile:
        writefile.write("John Smith,20,19-5-1998") # write a record

def read_from_file():
    with open('data_record.txt', 'r') as readfile:
        line = readfile.readline() # read the record as string
        a_list = line.split(',') # split the line into a list of strings
        name = a_list[0] # extract name as string
        age = int(a_list[1]) # extract age and convert to integer
        # extract date info and convert from string to object
        date_obj = datetime.strptime(a_list[2], '%d-%m-%Y')
```

```
        # create a record as a list (with the actual data)
        rec = [name, age, date_obj]
        # display the list “rec” as is
        print(rec)

def main():
    write_to_file() # create the text file “num_data.txt”
    read_from_file() # read file content

main()
```

In the write_to_file() function, a single record is written to the file data_record.txt, with the values separated by comma. In the read_from_file() function, the line is split into a list (by the comma character). Each element of the list is converted to their actual data type – string, integer, and date object. The converted values are then used to create a list (variable “rec”).

Example 7 – read file content using *for* statement

In most cases, a text file contains many lines of records. Without having to manually count the exact number of lines in a file, the *for* statement is used to read all content from a file. When using the *for* statement, the reading stops automatically when the end of the file is reached, thus not causing any reading errors.

Modify the previous example by adding the **code in bold**, as follows:

```
from datetime import datetime
def write_to_file():
    with open('data_record.txt', 'w') as writefile:
        writefile.write('John Smith,20,19-5-1998' + '\n')
        writefile.write('Alan Paul,30,29-7-1990' + '\n')
        writefile.write('Ken Amos,40,17-2-1989' + '\n')

def read_from_file():
    records = [] # creates a container as a list of records
    with open('data_record.txt', 'r') as readfile:
        for line in readfile:
            line = line.strip()#strip off the newline character '\n'
            a_list = line.split(',') # split the line into a list of strings
            name = a_list[0] # extract name as string
            age = int(a_list[1]) # extract age and convert to integer
            # extract date info and convert from string to object
            date_obj = datetime.strptime(a_list[2], '%d-%m-%Y')

            # create a record as a list (with the actual data)
            rec = [name, age, date_obj]
            # add the list "rec" to "records"
            records.append(rec)

def main():
    write_to_file() # create the text file “num_data.txt”
    read_from_file() # read file content

main()
```

In the write_to_file() function, two more lines of records are written to the file, with a newline character (\n) padded at the end of each line.

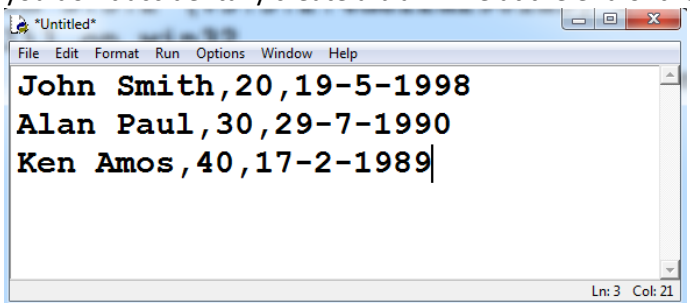
In the read_from_file() function, the for statement header reads each line of the file content into the variable “line”. The variable “line” represents each line of the file content, in the order from the beginning to the end. The strip() function was used to remove the newline character. The for loop stops when the end of the file is reached. Within the loop body, each line is split, and values converted, and a record of the actual data types is created out of each line. In the end, all records are added to the pre-created container “records”, i.e., the file content are transformed to a list of records.

Example 8 – put everything together

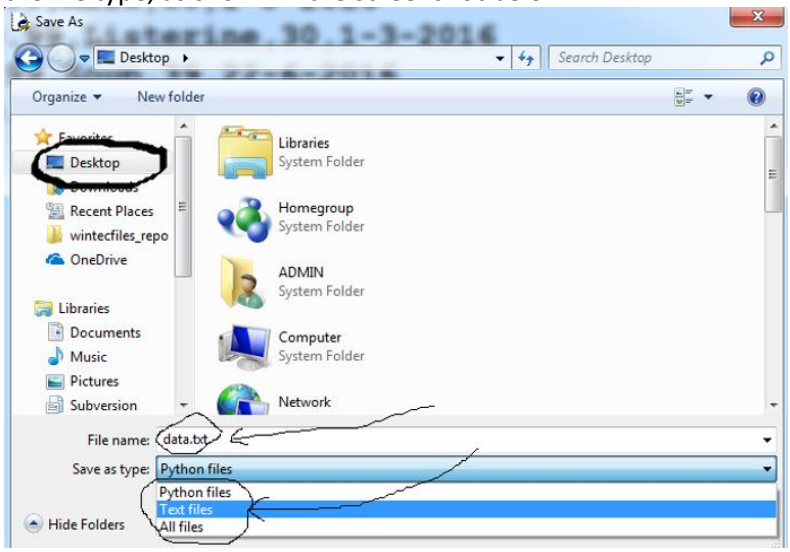
In this example, a text file is first manually created. The content of the file is then read into program to create a list of records. The list of records are written to another file as backup.

Following the steps below to create a text file “data.txt”:

1. File, New File
2. Type the data you intend to use in the editor, for example as in the screenshot below. Make sure you don’t accidentally create a blank line at the end of the file.



3. File, Save (or use keyboard shortcut: Ctrl + s)
4. In the popup dialog, select the same location as you have saved your Python script file (e.g., Desktop). Choose a name for the text file (e.g., data.txt). More importantly, choose “Text files” as the file type, as shown in the screenshot below.



Create a script input_output.py, with the following code:

```
from datetime import datetime
def read_data():
    records = [] # creates a container as a list of records
    with open('data.txt', 'r') as readfile:
        for line in readfile:
```

```
            line = line.strip()
            a_list = line.split(‘,’) # split the line into a list of strings
            name = a_list[0] # extract name as string
            age = int(a_list[1]) # extract age and convert to integer
            # extract date info and convert from string to object
            date_obj = datetime.strptime(a_list[2], ‘%d-%m-%Y’)

            # create a record as a list (with the actual data)
            rec = [name, age, date_obj]
            # add the list “rec” to “records”
            records.append(rec)

    return records # returns “records” from the function

def write_data(data):
    with open(‘backup_data.txt’, ‘w’) as writefile:
        for rec in data:
            name = rec[0]
            age = str(rec[1])
            datestr = rec[2].strftime('%d-%m-%Y')
            new_rec = [name, age, datestr]
            # conversion: [‘John Smith’,20,19-5-1998] -> ‘John Smith,20,19-5-1998’
            line = ','.join(new_rec)
            writefile.write(line + '\n')

def main():

    rec_list = read_data() # read_data() function returns a list of records
    write_data(rec_list) # rec_list is passed to write_data() function for backup

main()
```

In this example, the read_data() function is the same as the read_from_file() function, except it returns from the function the list of records composed from the file content. When the read_data() function is invoked in the main(), the returned list of records is assigned to the variable “rec_list”. The “rec_list” variable is then passed as an argument to the write_data() function. In the write_data() function, the parameter “data” (which represents the list of records passed from the main) is then processed by the for statement. In the for loop body, each record (i.e., the “rec” variable) itself is a list, elements of which are then converted to their string representation before being joined by the comma character as delimiter and written to the file backup_data.txt.