ISYS90088 Introduction to Application Development

Week11 – Files – I/O processing

Dr Thomas Christy

Sem 2, 2017

Objectives

- Introduction to File Input and Output
- Using loops t process files
- Processing records

Introduction to File Input and Output

- So far: we have asked the user to provide data during run time. And its lost once the program stops running. This means we have to re-enter data every time.
- How do we re-use data especially large data sets so we can use them as often as possible?
  - When a program needs to save data for later use, it writes the data in a file. The data can be read from the file when you need them.
- In a business environment, they rely on large files of data – employee records, client information etc….
- When you save data on a file, we say, "writing data to a file" – **output file** is used to describe a file that data is written to.

Introduction to File Input and Output

Three steps to be taken when a file is used by a program:

1. Open the file:
   - Opening a file creates a connection between the file and the program.
   - Opening an output file usually creates the file and allows the program to write data on it.
   - Opening an file allows the user to read data from a file (for input).
2. Process the file:
   - In this step, data is either written to the file (if it is an output file) or read from the file (if it is an input file)
3. Close the file:
   - When the program finishes using a file, the file must be closed. Closing a file disconnects the file from the program.

Introduction to File I/O – Text files

- A text file contains data that has been encoded as text.
- Even if the file contains numbers, those numbers are stored in the file as a series of characters.
- This file can thus be opened and read from any text editor like Notepad for example.

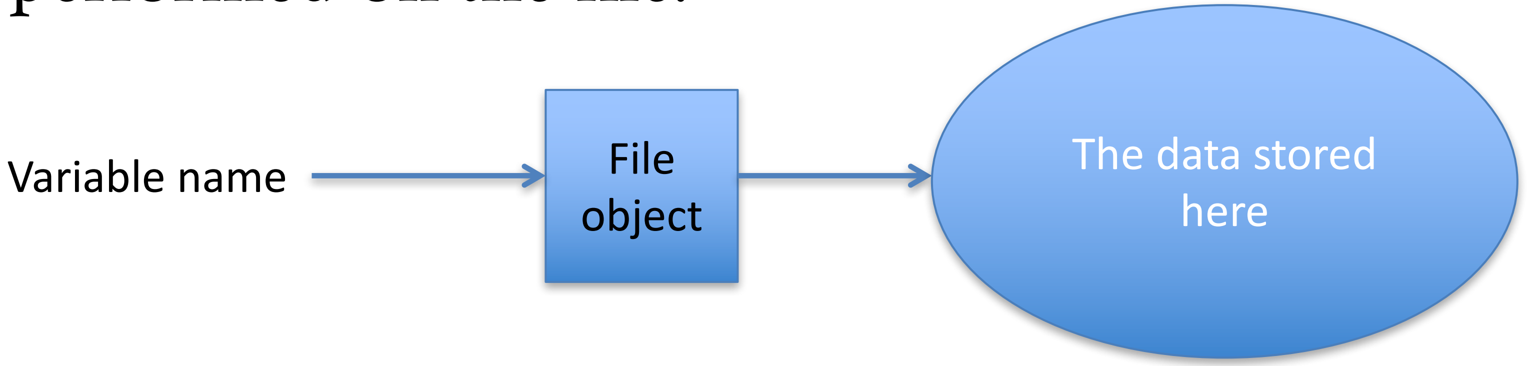
File names and file objects:

- Names given to files – and many times with an extension (eg. .txt, .doc, .csv, jpg etc…)
- File object – in order for a program to work on an environment, the program must create a *file object*

Introduction to File I/O: text files

- A *file object* is associated with a specific file for a program to work with it. In a program, we use a variable name /reference to reference the file object.

- This variable name is used to carry out operations that are performed on the file.



Introduction to File I/O: Opening a file

- Use the `open` `function` in python to open a file.

The open function creates a file object and associates it with a file.

Syntax:

```
    file_variable = open(filename, mode)
```

`file_variable:` is the name of the variable that will reference the file object.

`Filename:`is a string specifying the name of the file (.txt, .csv, etc…)

`mode:` is a string specifying the mode (ie., reading, writing or appending for example) in which the file will be opened.

Introduction to File I/O: Opening a file

- **Modes**

‘r’ - open a file for reading only. The file cannot be changed or written to.

‘w’ – open a file for writing. If the file already exists, erase its content. If it does not exist, create it.

‘a’ - open a file to be written to. All data written to the file will be appended to its end. If the file does not exist, create it.

Opening a file: example

- Suppose you have a customer data file called `customers.txt.` Lets assume we want to open it for reading information from it.
- Use `open` function :

```
customers_file = open('customers.txt', 'r')
```

After this statement executes, the file name `customers.txt` will be opened, and the variable `customers_file` will reference a file object that we can use to read data from a file.

- Suppose we want to create a file named `sales.txt` and write data to it, here is an example of how you could do it:

```
sales_file = open('sales.txt', 'w')
```

After this statement executes, the file name `sales.txt` will be created, and the variable `sales_file` will reference a file object that we can write data to the file.

Specifying the location of a file

- Suppose your program is located in the following folder on your computer:

```
C:\Users\mendozaa\Documents\Python
```

- Now lets say the program is running and it executes the following statement, the file test.txt is created in the same folder:

```
test_file = open ('test.txt', 'w')
```

- However, if you want to open a file in a different location, then you have to specify the entire path as well as the filename in the argument that you pass the open function.

```
test_file = open (r 'C:\Users\mendozaa\temp\test.txt', 'w')
```

Specifying the location of a file

Consider this statement:

```
test_file = open (r ‘C:\Users\mendozaa\temp\test.txt’, ‘w’)
```

This statement creates the `file test.txt` in the folder

```
        C:\Users\mendozaa\temp\
```

r – this prefix specifies that the string is a *raw string*. This causes python interpreter to read the backslash characters as literal backslashes. Without the prefix **r** the interpreter would assume that the backslash characters were part of escape character sequence, and error would occur .

Writing data to a file

If you wish to write data to a file, here is the general format:

```
file_variable.write (string)
```

- `file_variable` is the variable that references the file object, `string` is the text you want to write into the file.
- You must open the file using `'w' or 'a'` or an error will occur.
- Example: consider

```
customers_file = open ('customers.txt', 'w')
```

exists, and you want to add/write data 'Charles Dickens' to that file:

```
customers_file.write('Charles Dickens')
(or)
name = 'Charles Dickens'
customers_file.write(name)
```

Close a file

Once a program finishes working with a file, we are required to close the file.

For example:

```
customers_file.close()
```

Example 0: how to open a file, write into it and then close it?

# Write a program that writes three lines of data to a file

```
#example: write a program that writes three
lines of data into a file
```

Example 0: how to open a file, write into it and then close it?

# Write a program that writes three lines of data to a file

```
#example: write a program that writes three lines
of data into a file

def main():
    #open a file named customers.txt
    infile = open ('customers.txt', 'w')
    # write the names of three customers
    infile. write('Alice\n')
    infile.write('David\n')
    infile.write('Tim\n')
    # close the file
    infile.close()

main()
```

Example 1: Reading data from a file

# Write a program that reads the content from a file. Use the previous file that you just created and populated with data.

```
def main():
    #open the file
    infile = open ('customers.txt', ‘r')
    #process the file
    XXX = XXX.XXX()
    #close the file
    XXX.XXX()
    #print the data
    print(XXX)
main()
```

Example 2: Reading data one line at a time from a file

# Write a program that reads the content from a file one line at a time. Use the previous file that you created and populated with data.

```
def main():
    #open the file
    infile = open ('customers.txt', ‘r')
    #process the file
    data = infile.read()
    #close the file
    infile.close()
    #print the data
    print(data)
main()
```

Example 2: Reading data one line at a time from a file

# Write a program that reads the content from a file one line at a time. Use the previous file that you created and populated with data.

```
def main():
    #open the file
    infile = open ('customers.txt', ‘r')
    #process the file
    xxxx
    xxxx
    xxxx
    #close the file
    xxxx
    #print the data
    xxxx
    xxxx
    xxxx
main()
```

Example 2: Reading data one line at a time from a file

# Write a program that reads the content from a file one line at a time. Use the previous file that you created and populated with data.

```
def main():
    #open the file
    infile = open ('customers.txt', ‘r')
    #process the file
    line1 = infile.readline()
    line2 = infile.readline()
    line3 = infile.readline()
    #close the file
    infile.close()
    #print the data
    print(line1)
    print(line2)
    print(line3)
main()
```

Example 4: getting users to enter data and writing it into a file

Write a program that reads the three friends names from a user and write it into a file.

```
def main():
    # get the user to enter the name
    name1 = XXX
    name2 = XXX
    name3 = XXX
    #open the file or create one
    myfile = open('friends.txt', xxx)

    #process the file
    XXX.XXX (name1 + XXX)
    XXX.XXX (name2 + XXX)
    XXX.XXX (name3 + XXX)
    #close the file
    myfile.close()
main()
```

Example 5: reads content one line at a time – strips the newlines from it

# check example 2 and redo it to give the correct output by stripping the \n

```
 def main():
    infile = open ('customers1.txt', 'r')
    #read the three lines from the file
    line1 = infile.readline()
    line2 = infile.readline()
    line3 = infile.readline()
    #strip the \n from each string
    line1 = line1.rstrip('\n')
    line2 = line2.rstrip('\n')
    line3 = line3.rstrip('\n')

    # close the file
    infile.close()
    #print the lines
    print(line1)
    print(line2)
    print(line3)

main()
```

Example 6: appending or adding data into an existing file of data

- Use ‘a’ to append data into an existing file.
- Check the `customers.txt` file that we already created with three customer names in it, add three more names to the `customers.txt` file

```
# open the file

# process the file

# close the file
```

Example 6: appending or adding data into an existing file of data

- Use ‘a’ to append data into an existing file.
- Check the `customers.txt` file that we already created with three customer names in it, add three more names to the `customers.txt` file

```
# open the file
   infile = open (‘customers.txt’, ‘a’)
# process the file
   infile.write(‘Jelly’)??
   infile.write(‘Jessica’)??
   infile.write(‘Chen’)??
# close the file
   infile.close()
??? Is there something else I missed????
```

Example 6: appending or adding data into an existing file of data

- Use ‘a’ to append data into an existing file.
- Check the `customers.txt` file that we already created with three customer names in it, add three more names to the `customers.txt` file

```
#open the file
   infile = open (‘customers.txt’, ‘a’)
# process the file
   infile.write(‘Jelly\n’)??
   infile.write(‘Jessica\n’)??
   infile.write(‘Chen\n’)??
#close the file
   infile.close()
??? Is there something else I missed????
```

Writing and reading Numeric data

- Strings can be written directly into files. But numeric data (numbers) must be converted to strings before they are written into a file.

- **`str`** converts a value into string

- Example:

```
 num = 99
    print(str(num))
```

Text files and their format

- Data in a text file can be characters, words, numbers, lines or text.
- When data are treated as numbers, they must be separated by white space characters – spaces, tabs, and newlines.
- For example, a text file that contains floating point numbers might look like:

```
34.6 22.33 66.75
77.12 21.44 99.01
```

Note:

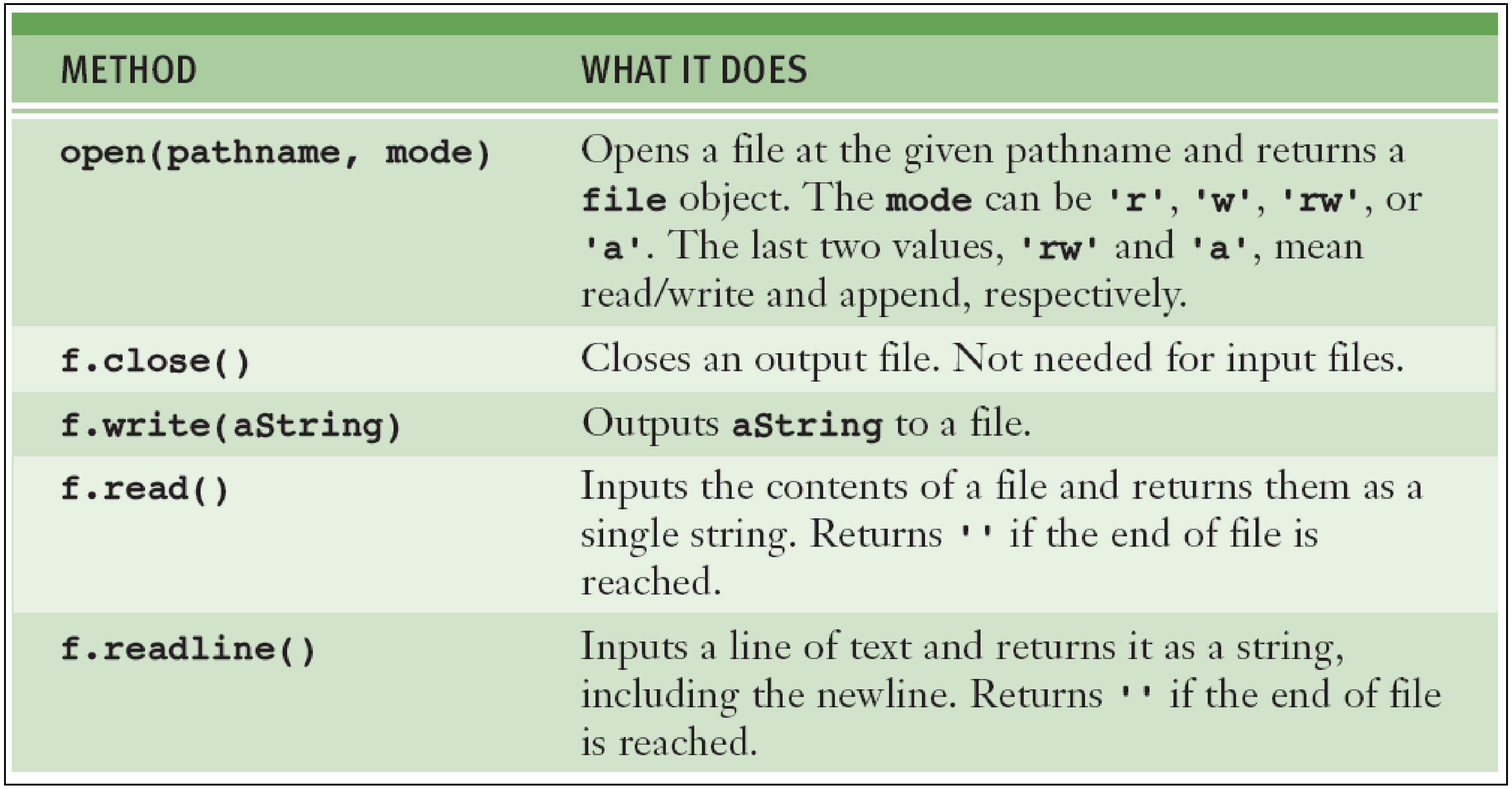
- This format includes a space or a newline separator of items in text.
- But remember: all input or output from a text file must be strings.

Writing and reading Numeric data

# write a program that converts numbers to string before writing into a file

NOTE: while reading from a file, watch out - you will reading the values as strings even if they are numbers. So make sure to convert it to int and then use in your program.

Reading Numbers from a File (continued)



METHOD	WHAT IT DOES
`open(pathname, mode)`	Opens a file at the given pathname and returns a **`file`** object. The **`mode`** can be **`'r'`**, **`'w'`**, **`'rw'`**, or **`'a'`**. The last two values, **`'rw'`** and **`'a'`**, mean read/write and append, respectively.
`f.close()`	Closes an output file. Not needed for input files.
`f.write(aString)`	Outputs **`aString`** to a file.
`f.read()`	Inputs the contents of a file and returns them as a single string. Returns **`''`** if the end of file is reached.
`f.readline()`	Inputs a line of text and returns it as a string, including the newline. Returns **`''`** if the end of file is reached.

Using loops to process files

- Files usually hold large amounts of data. It is therefore useful to use loops to process the data in files.

- For example, assume you are accessing customer details in a banking environment or sales data for a fortnight or even the census data in your assignment 2… large data sets.

- Consider an example where you write a program that prompts a data entry person to enter sales amounts and write those amounts to a `sales.txt` file

Example: Using loops to process files

```
def main():
    #get the number of days
    num_days = int(input(’how many days of sales figures?'))

    #open a file to create or inout data
    sales_file = open ('sales.txt', 'w')

    #get values from user for each day and write into file
    for count in range(1, num_days +1):
       sales = float(input('enter sales for day #' + str(count) + ':'))
       sales_file.write(str(sales) + '\n’)
    #  close the file
    sales_file.close()

main()
```

Using loops to process files: to read lines

- Using the `for` loop to read data line by line until the end of the file. For loops are more elegant for this.

Syntax:

```
for variable in file_object:
    statement
    statement
    statement
    etc….
```

Example:8 Using loops to process files: to read lines

#Using the `for` loop to read data line by line until the end of the file.

Example: Write a program that reads all the values in the `sales.txt` file

<check example 8 from the list of examples)

Example:8 Using loops to process files: to read lines

```
def main():
    #open the file to read values
    sales_file = open('sales.txt', 'r')

    # read the lines one by one
    for line in sales_file:
        #convert the values to float
        amount = float(line)
        print(format(amount, '0.2f'))

    #close the file
    sales_file.close()

main()
```

Some code: Reading Numbers from a File – check this out!!!

- Examples of code :

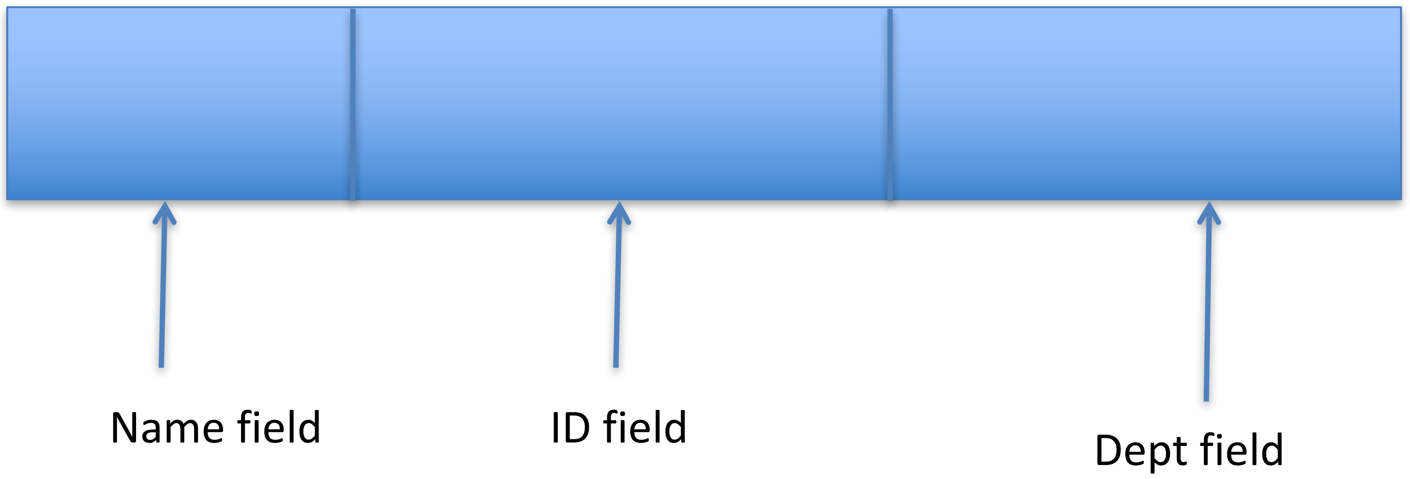
```
f = open("integers.txt", 'r')
sum = 0
for line in f:
    line = line.strip()
    number = int(line)
    sum += number
print("The sum is", sum)
```

```
f = open("integers.txt", 'r')
sum = 0
for line in f:
    wordlist = line.split()
    for word in wordlist:
        number = int(word)
        sum += number
print("The sum is", sum)
```

Processing records

- The data stored in a file is frequently organized in records.
- A *record* would be a complete set of data about an item, and a *field* is an individual piece of data with a record.
- For example: suppose we want to store data about employees in a file. The file will contain records for each employee. Each record will be a collection of fields, such as name, ID, and dept.

One record in a file: Employee_record



Note: We will many such records in a sequential fashion in a file

Example9: Processing records

Lets write a program that gets employee data from users and saves it as records in the employee.txt file

```
def main():
    # get the total number of employees recorsd you want to create
    num_emps = int(input('how many employee records:'))
    #open a file to write the information
    employee_file = open ('employee.txt', 'w')

    # get the data from user
    for count in range(1, num_emps+1):
        print('enter data for employee #', count)
        name = input('name:')
        id_num = input('ID:')
        dept = input('department:')

        # write the data - record into the file
        employee_file.write(name + '\n')
        employee_file.write(id_num + '\n')
        employee_file.write(dept + '\n')

        print()
    employee_file.close()

main()
```

Example10: Processing records

Lets write a program that reads from the employee.txt that your just created to get the employee details record by record

```
def main():
    employee_file = open ('employee.txt', 'r’)
    #read first line from file, which is the name filed of first record
    name = employee_file.readline()
    while name != '':
        id_num = employee_file.readline()
        dept = employee_file.readline()

        #strip the newlines fro the field
        name = name.rstrip('\n')
        id_num = id_num.rstrip('\n')
        dept = dept.rstrip('\n')

        #display records
        print('name:', name)
        print('id:', id_num)
        print('dept:', dept)
        print()
        name = employee_file.readline()

    employee_file.close()

main()
```