Lecture 18 – Open, read, write and close files

Learning outcomes

- At the end of this lecture, students should be able to:
  - understand the file system structure
  - open and close a file
  - write data to a file
  - read data from a file

- In CompSci 101 we are dealing with text files only.

Recap

- From lecture 17: complete the carry_out_transactions() function which is passed an initial balance and a tuple of transactions (positive and negative amounts). The function returns a tuple made up of three values: the final balance, the sum of all the deposits and the sum of all the withdrawals.

```
import random

def carry_out_transactions(balance, transactions_tuple):
    withdrawals = 0
    deposits = 0
    for trans in transactions_tuple:
        if trans < 0:
            withdrawals = withdrawals + abs(trans)
        elif trans > 0:
            deposits = deposits + trans
    balance = balance + trans
    return (balance, deposits, withdrawals)

def main():
    results = carry_out_transactions(5400, (100, -400, 500, -800, 600, -100, -200, 50, 0, -200))
    print("Balance $", results[0], ", deposits $", results[1], ", withdrawals $", results[2], sep="")

main()
```

Program data

- Data used in a program is temporary, and is lost when the program terminates.

```
import random

def main():
    my_list = []
    for num in range(20):
        my_list.append(random.randrange(10, 100))

main()
```

- To permanently store the data created in a program, we need to save it on a physical storage device.

```
[67, 53, 35, 39, 89, 44, 73, 86, 48, 69, 74, 97, 60, 64, 72, 56, 88, 80, 39, 69]
```
**Files**

- A file is a collection of bytes of information that usually resides permanently on a disk.
- The data in a file can be used later by other programs.
- Accessing a file means establishing a connection between the file and a program and moving data between the two.
- We need to be able to:
  - read data from a file into a program
  - write data from a program to a file

**Accessing a file**

- When a connection has been set up between a Python program and a file, a 'stream of data' is established between the two:

**Path of a file**

- The file system of a computer organises files in a hierarchical (tree) structure.
- Files are placed inside directories. Directories can contain files or other directories.
- A complete description of which directories to visit in order to reach a certain file is called a path, e.g.,

\[ C:/Users/Adriana/Documents/prog1.py \]

Each path to a file or a directory must be unambiguous.

- The file path is the '/' separated list of directories which need to be visited in order to reach the file. For example, if the input.txt file needs to be accessed from inside the prog2.py program. This file can be accessed using the **absolute path:**

\[ C:/Users/Adriana/Documents/input.txt \]

or using the **relative path:**

\[ input.txt \]
Binary vs text files

- Python files are classified into two categories, i.e., text and binary.
- Text files can be processed using a text editor.
- Binary files, e.g., images, audio, video files are designed to be read by special applications which 'understand' their format.
- If you open a binary file using a text editor, the editor tries to match the binary information to text characters but mostly the file information will be gobbledygook.

Processing files

- To use Python's built in file processing functions you must first open the file. Once open, data within the file is processed using functions provided by Python, and finally the file is closed. Always remember to close the file when you're done so that the resources can be released.

Opening a file

- The Python syntax for opening a file is:
  
  ```python
  file_variable = open(filename, mode)
  ```

  The variable, file_variable, is now the connection between the program and the file, and this variable can now be used to read/write to the file.

- For example:

  ```python
  def main():
      input_file = open("stocks.txt", "r")
  main()
  ```

  Note that the filename is the path of the file. In this case the file, "stocks.txt" is in the same directory as the program, i.e., the file path used is the relative path.

File access modes

- The Python syntax for opening a file is:
  
  ```python
  file_variable = open(filename, mode)
  ```

- Mode Description
  
<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'r'</td>
<td>Opens a file for reading.</td>
</tr>
<tr>
<td>'w'</td>
<td>Opens a file for writing.</td>
</tr>
<tr>
<td>'a'</td>
<td>Opens a file for appending data. Data is written to the end of the file.</td>
</tr>
<tr>
<td>'rb'</td>
<td>Opens a file for reading binary data.</td>
</tr>
<tr>
<td>'wb'</td>
<td>Opens a file for writing binary data.</td>
</tr>
</tbody>
</table>

#The following modes are not used in CompSci 101

- 'a' Opens a file for appending data. Data is written to the end of the file.
- 'rb' Opens a file for reading binary data.
- 'wb' Opens a file for writing binary data.
Closing a file

- The Python syntax for **closing a file** is: `file_variable.close()`
- The `close()` method closes the file (i.e., releases the file resources). After a file has been closed, access to the file contents is no longer available until the file is opened again.
- If the mode is write mode, any as-yet-unwritten content is flushed to the file.
- For example:
  ```python
def main():
    input_file = open("stocks.txt", "r")
    # process the file
    input_file.close()
main()
```  

Writing to a file

- First, the file needs to be opened for writing:
  ```python
  output_file = open("output.txt", "w")
  ```
- If the output.txt file does not exist then the open() function creates the file.
- If the output.txt file exists then the open() function erases the contents of the file.
- The syntax for writing to a file:
- For example:
  ```python
  output_file.write(a_string_of_text)
  ```

  ```python
def main():
    output_file = open("output.txt", "w")
    output_file.write("She walks in beauty, like the night in
    Of cloudless climes and starry skies
    Lord Tennyson")
    output_file.close()
main()
```  

Writing to a file continued

- The syntax for writing to a file:
  ```python
  output_file.write(a_string_of_text)
  ```
  and the parameter passed to the write() function is a string. Any numbers need to be converted using the `str()` function. Any new lines need to be written to the file ("\n"). For example:

  ```python
def main():
    output_file = open("output.txt", "w")
    sum_of_nums = int(input("Enter num: "))
    sum_of_nums = sum_of_nums + int(input("Enter num: "))
    output_file.write(str(sum_of_nums) + "\n")
    output_file.close()
main()
```  

Program with 3 errors

- Find the three errors in the following code. The file which should be created by the following code is shown below:

  ```python
  def three_errors(list1):
    output_file = open("oops.txt", "w")
    for num in list1:
      output_file.write(num)
  def main():
    a_list1 = [2, 4, 5, 6, 8, 1]
    three_errors(a_list1)
main()
```
Complete the function

- Complete the write_to_file() function which writes the elements of the two parameter lists (one element from both files per line) to the file (given by the parameter, filename). The elements are separated by ",": ".

```python
def write_to_file(filename, list1, list2):
    # Assume the two files have exactly the same number of elements and
    # that each element is an integer.

def main():
    a_list1 = [2, 4, 5, 6, 8, 1]
    a_list2 = [123, 54, 58, 106, 87, 206]
    filename = "combined_lists.txt"
    write_to_file(filename, a_list1, a_list2)

main()
```

Reading a file

- First, the file needs to be opened for reading:
  ```python
  input_file = open("input.txt", "r")
  ```
  If the input.txt file does not exist then an error occurs.

- The four ways characters can be read from a file:
  ```python
  input_file.read()
  input_file.read(an_integer)
  input_file.readline()
  input_file.readlines()
  ```

The read() functions

- The read() method returns the entire contents of the file. This method returns a string.
  ```python
  all_contents = input_file.read()
  ```

- The read(an_integer) method returns the specified number of characters (a string) from the file.
  ```python
  some_characters = input_file.read(an_integer)
  ```

The read() functions - examples

- Both the following sections of code use the file below:
  ```python
  input_file = open("poem.txt", "r")
  all_contents = input_file.read()
  input_file.close()
  print(all_contents)
  A thing of beauty is a joy for ever:
  Its loveliness increases; it will never
  Pass into nothingness; ...
  John Keats
  ```

- The following code uses the file with a 10-character limit.
  ```python
  input_file = open("poem.txt", "r")
  some_contents = input_file.read(10)
  input_file.close()
  print(some_contents)
  print(len(some_contents))
  A thing of
  10
  ```
A note about the read() functions

- Note that the file variable reads from whichever position in the file it is currently pointing to, e.g.,

```python
input_file = open("poem.txt", "r")
some_characters = input_file.read(10)
print(some_characters)
print()
all_contents = input_file.read()
input_file.close()
print(all_contents)
```

A thing of beauty is a joy for ever:
Its loveliness increases; it will never
Pass into nothingness; ...
John Keats

The readline()/readlines() methods

- The readline() method returns the next line of the file. This method returns a string. The new line character is the last character of the string returned.

```python
next_line = input_file.readline()
```

- The readlines() method returns a list of the remaining lines of the file. This method returns a list of strings. The new line character is the last character of each string in the list (except for the last element).

```python
list_of_lines = input_file.readlines()
```

The readline() and readlines() methods - examples

- Both the following sections of code use the file below:

```python
input_file = open("RedHerring.txt", "r")
one_line = input_file.readline()
print(one_line)
```

A Red Herring: A distraction from the main issue.

```python
input_file = open("RedHerring.txt", "r")
list_of_lines = input_file.readlines()
print(len(list_of_lines))
print(len(list_of_lines[2]))
print(len(list_of_lines[4]))
print(len(list_of_lines))
```

A red herring has a strong odour.
The smell of the herring and start

6 Note that the string read from the text contains the newline character.

Show the contents of the OutBoard.txt file

```python
def main():
    input_file = open("AboveBoard.txt", "r")
    output_file = open("OutBoard.txt", "w")
    line_list = input_file.readlines()
    for line in line_list:
        if line[0] == 'p' or line[0] == 'A':
            output_file.write(line)
    input_file.close()
    output_file.close()
```

main()
Complete the function

Complete the function

- The copy_file() function takes the names of an input file and an output file, copies data from the input file to the output file and returns a string made up of the first and last characters in the file.

```python
def copy_file(filename_in, filename_out):
    first_last_chars = copy_file(input_f, output_f)
    print(first_last_chars)
main()
```

Examples of Python features used in this lecture

```python
def read_poem():
    input_file = open("poem.txt", "r")
    all_contents = input_file.read()
    input_file.close()
    print(all_contents)
    print()

def write_to_file(filename, list1, list2):
    output_file = open(filename, "w")
    for i in range(len(list1)):
        output_file.write(str(list1[i]) + ":")
        output_file.write(str(list2[i]) + "\n")
    output_file.close()
```