COMPSCI 101

Principles of Programming

Lecture 20 – Open files, read from files, write to files, 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 19: 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.

```
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()
```

```
Balance $4950, deposits $1250, withdrawals $1700
```


Data which is processed in a Python program

Data processed in a program exists while the program is running but it is lost when the program terminates.

```
import random

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

  print(my_list)

main()
```

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

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

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.

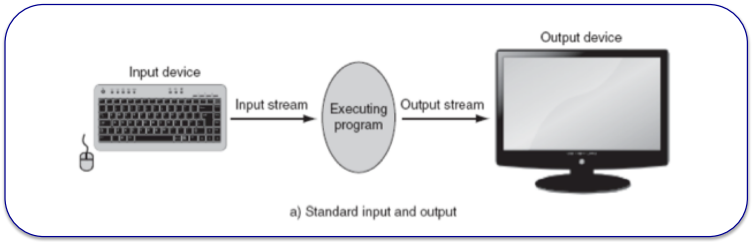
Accessing a file means establishing a connection between the file and a program and moving data between the two.

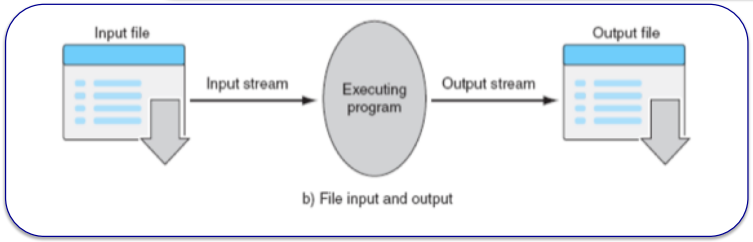
We need to be able to:

- 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:

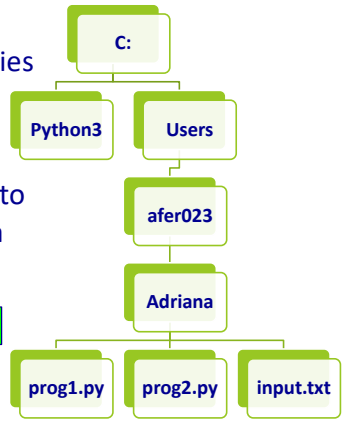




Accessing a file

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/afer023/Adriana/input.txt
```

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

Path of a file

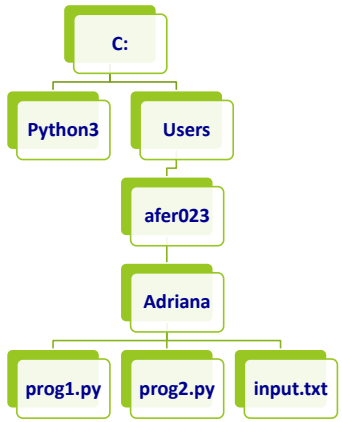
The file path is the '/' separated sequence of directories which need to be visited in order to reach the file. For example, if the input.txt file needs to be accessed from inside the prog2.py program.

This file can be accessed using the **absolute path**:

```
C:/Users/afer023/Adriana/input.txt
```

or using the **relative path**:

```
'input.txt'
```

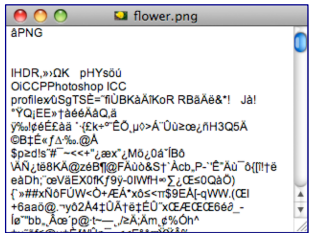


Binary vs text files

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.

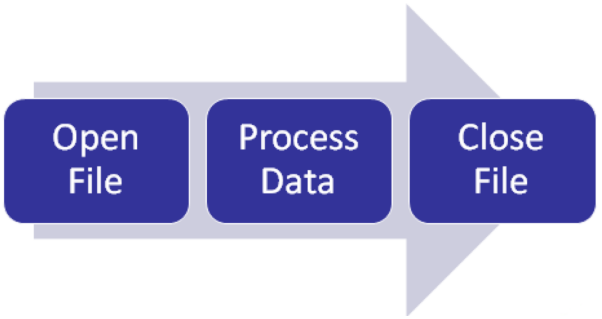
Image file displayed by a text editor



Same file displayed by an image viewer

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:

```
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:

```
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:

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

Mode	Description
'r'	Opens a file for reading.
'w'	Opens a file for writing.
#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, then any as yet unwritten content is flushed to the file.

For example:

```
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:

```
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.

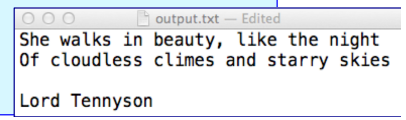
The syntax for writing to a file:

```
output_file.write(a_string_of_text)
```

For example:

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

main()
```



Writing to a file continued

The syntax for writing to a file:

```
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,

```
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()
```

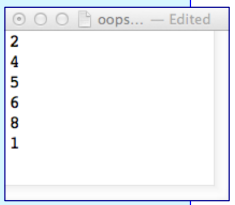

Program with 3 errors

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

```
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()
```



Reading From a File

First, the file needs to be opened for reading:

```
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:

```
input_file.read()
input_file.read(an_integer)
input_file.readline()
input_file.readlines()
```


The `read()` method

The **`read()`** method returns the entire contents of the file. This method returns a string.

```
all_contents = input_file.read()
```

The **`read(an_integer)`** method returns the specified number of characters (a string) from the file.

```
some_characters = input_file.read(an_integer)
```


The `read()` method - examples

Both the following sections of code use the file below:

poem.txt — Edited

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

John Keats
```

```
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 Keat
```

```
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()` method

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

poem.txt — Edited

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

John Keats
```

```
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 Keat
```

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.

```
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).

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


`readline()`/`readlines()` - examples

Both the following sections of code use the file below:

RedHerring.txt — Edited

```
A Red Herring: A distraction from the main issue.
Origin: It comes from fox hunting.
A red herring has a strong odour.
Hounds chasing a fox could be distracted by
the smell of the herring and start
following that instead.
```

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

```
A Red Herring: A distraction from the main issue.
```

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

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 ": ".

```
def write_to_file(filename, list1, list2):



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()
```

Assume the two lists have exactly the same number of elements and that each element is an integer.

combi... — Edited

```
2: 123
4: 54
5: 58
6: 106
8: 87
1: 206
```


Show the contents of the OutBoard.txt file

```
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()
```

AboveBoard.txt — Edited

```
Above board: Legitimate.
Origin: Cardsharps
place their hands
under the 'board' or
table to stack the deck.
If they keep
their hands above the
board, they
can be presumed to be
performing without
trickery.
```

OutBoard.txt

Complete the function

Complete the `get_percent_vowels()` function which returns the percentage of letters in the text (rounded to a whole number) which are vowels. Ignore all non alphabetic characters (use `isalpha()` to check).

PoetryPrize.txt — Edited

```
Hello Everyone
txtin iz messin,
mi headn'me englis,
try2rite essays,
they all come out txtis.
gran not plsed w/letters shes getn,
swears i wrote better
b4 comin2uni.
&she's african
```

```
def get_percent_vowels(filename):
    vowels = "aeiouAEIOU"



def main():
    input_f = "PoetryPrize.txt"
    percent_vowels = get_percent_vowels(input_f)
    print(str(percent_vowels) + "% are vowels")
main()
```

```
35% are vowels
```


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 four characters followed by the last four characters in the file.

FreeAdviceIn.txt — Edited

```
Free advice

is worth
 what you paid

   for it!
```

FreeAdviceOut.txt — Edited

```
Free advice

is worth
 what you paid

   for it!
```

```
def copy_file(filename_in, filename_out):



def main():
    input_f = "FreeAdviceIn.txt"
    output_f = "FreeAdviceOut.txt"
    first_last_chars = copy_file(input_f, output_f)
    print(first_last_chars)
main()
```

```
Free it!
```


Summary

In a Python program:

- a 'data stream' can be created between the program and a file
- data can be written to a file
- data can be read from a file
- a file should be closed once the program has finished reading or writing to the file

In CompSci 101 we are dealing with text files only.

The file system is a hierarchical structure

Examples of Python features used in this lecture

```
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(": ")
        output_file.write(str(list2[i]) + "\n")

    output_file.close()
```