SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 120: Introduction to Computing Science and Programming 1

Using Files for Data Input and Output

Course Topics

1. General introduction
2. Algorithms, flow charts and pseudocode
3. Procedural programming in Python
4. Data types and Control Structures
5. Binary encodings
6. Fundamental algorithms
7. Basics of (Functions and) Recursion (Turtle Graphics)
8. **Basics of computability and complexity**
9. **Basics of Data File management**

Today's Topics

1. Introduction to File
2. Using File for Data Input (aside from using input())
3. Using Files for Data Output (aside from using print())

External storage

- When we shut down an application (e.g.: Python IDLE, Word or Excel) and/or turn off our computer, often we do not want our information (code, data) to disappear.
  - We want our information to **persist** until the next time we use it.
  - We achieve persistence by saving our information to **files** on external storage like **hard disk**, **flash memory**, etc…
  - We can use text files to store the input/output data.

Files

- **Text Files:**
  - The sequence of 0's and 1's represents human-readable characters, i.e., UNICODE/ASCII characters
  - To view the content of a text file, one needs to use the appropriate application such as a text editor (notepad).
  - Example:
  - In CMPT 120, we shall open or read text files to get data in to the program, or to write from a program.

Introduction to Recursion

```
# Hardcode data inside program.
```

```
# Get data using input() function.
```

```
# Get data from a text file.
# Opening a file for reading
```

Open a file in a Python program

- To use a file in our Python program, we must first open it in the appropriate mode:

Syntax: `<fileObject> = open(filename, <mode>)`

Optional **string** describing the way in which the file will be used.

- Where does the value of the variable **`filename`** come from?
- We can either ask the user to enter a filename (string) using `input()`, prior to the call to `open( )`
- OR
- We can assign a filename (string) to this variable at the top of our program, prior to the call to `open( )`

A word about the file named filename

- Python interpreter will look for a file with the **filename** in the **current directory**.
- **What is the current directory?**
  - The directory that contains the Python program we are currently running.
- If filename is stored in another directory, we must add the proper path to it:
  <**path**/filename>
  - **C:/my_folder/mark.txt**
- This **path** can be part of the value assigned to the variable filename.

```
filename = path + filename
```

A word about <mode>

- A mode describes the way in which a file is to be used
- Python considers the following modes:

1.
2.
3.
4.

Open a File for Reading

- To read from a file, we need to first open it in **read** mode with `'r'`:

Syntax:

- **`fileRead`** is (called) a file object.

- If the file does not exists in the current directory, then:
  - Python interpreter produces and prints an error.

```
FileNotFoundError: [Errno 2] No such file or
                  directory: 'fileDoesNotExist.txt'
```

Code Example

```
# Either ask user for a filename (and path, or set your
# filename variable once at the top of your program.
inputFile = “list_of_words.txt"

...

# Opening a file for reading
fileR = open(inputFile , 'r')
# or
fileR = open(inputFile )
```

Open a File for Writing

- To write to a file, we need to first open it in **write** mode with `'w'`:

Syntax:

- **fileWrite** is a file object, i.e., a variable of type class.
- If the file already exists in the directory, **its content is erased, ready to receive new data.**
- If the file does not exists in the directory, then, **it is created.**
- **Example:**

```
outputFile = "newFile.txt"
# Opening a file for writing
fileW = open(outputFile, 'w')
```

Open a File for Appending

Syntax: fileAppend = **open**(<filename>, **‘a'**)

- **fileReadWrite** is a file object, i.e., a variable of type class.
- If the file already exists in the directory, **new data will be automatically added to the end of the file, leaving the current content unaffected**
- If the file does not exists in the directory, then, **it is created.**
- **Example:**

```
appendFile = “savedFile.txt"
# Opening a file for appending
fileA = open(appendFile, ‘a')
```

Open a File for Reading and Writing

Syntax: fileReadWrite = **open**(<filename>, **‘r+'**)

- **fileReadWrite** is a file object, i.e., a variable of type class.
- If the file already exists in the directory, **new data will be automatically added to the end of the file, leaving the current content unaffected**
- If the file does not exists in the directory, then, **it is created.**
- **Example:**

```
scoreFile = “savedFile.txt"
# Opening a file for appending
fileRW = open(scoreFile, ‘r+')
```

Reading from a File

- **File object** provides methods for reading data from a file.
- To read a line from a file into a string:
  - **`readline( )`** : This method reads characters from the file until it reaches a **newline** character and returns the result as a string.
  - The file object keeps track of where it is in the file, so if we call `readline( )` again, we get the next line (i.e., 2nd line)
  - We can place the `readline( )` method inside a loop to read all the lines from a file – one by one.

Example

```
# File_IO_Demo_Read_File.py

# Opening a file for reading

# Read its first line -> a string

print("\nfirst line: " , firstLine)
print("type(firstLine) is {}.
".format(type(firstLine)))

# Read its second line
# File object keeps track of the current line in file

print("\nsecond line: " , secondLine)

# Close the file
fileR.close( )
```

Quiz Example: Reading a Line (more values) At a Time

```
inputFile = 'mark_data.txt'

# Demo 1 - Reading a line (more than one value) at a time.
print("\nDemo 1 - Reading a line at a time from a file.")

# Open the file for reading
fileR = open(inputFile, 'r')

# Read its first line -> a string
firstLine = fileR.readline()

# Split the string into a list
mark_list = firstLine.split()

# Store marks into variables
quiz1 = int(mark_list[0])
quiz2 = int(mark_list[1])

# add marks
total = quiz1 + quiz2

print(total)

# Close the file
fileR.close( )
```

Reading From a File in a Loop

- Efficient way to read the content of a file using a loop.

```
for line in fileR:
    # strip whitespaces and newline character
    strippedLine = line.strip
    # process strippedLine
```

3. To read all lines from a file into a list:

```
myList = list(fileR)
fileR.readlines()
```

Code Example

```
…
# Opening a file for reading
fileR = open(inputFile, 'r’)

# Read all lines into list
myList1 = list(fileR)
print("\nfirst list: ", myList1)

# Close the file
fileR.close( )
```

```
# Opening a file for reading
fileR = open(inputFile, 'r')

# Read all lines into list
myList2 = fileR.readlines( )
print("\nsecond list: ", myList2)

# Close the file
fileR.close( )
```

Writing from a File

- **File object** provides methods for writing data into a file.
- **write()** method writes data to a file.

  numOfChars = fileWrite.write(aString)

  - writes the contents of **aString** to the file.
  - Stores number of characters written in numOfChars.
- To write something other than a string, convert it to a string first using:
  - **str( )**
  - **String formatting (e.g.: %d)**
  - **.format() method of string**

Code Examples

- See the following code files on our course web site:

1. File_IO_Demo_Write_to_File.py
2. File_IO_Demo_Read_File.py

Closing a file

- All the files must be closed:

```
<fileobject>.close( )
```

- Why?
  - To finalize the file.
  - To free up any system resources taken up by the open file.

  - After calling `close( )`, we cannot use the file object anymore (in our Python program).

Dealing with errors

- We saw that if the file does not exists, Python interpreter produces and prints an error.

```
FileNotFoundError: [Errno 2] No such file or
            directory: 'fileDoesNotExist.txt'
```

- We can write guardian code against this and other errors called “exceptions”.
  - “**exceptions**” to the normal flow of execution.

Catching exceptions

- Using the **try** statement (often called “try block”).

```
fileDoesNotExist = "fileDoesNotExist.txt"

 try:
    fin = open(fileDoesNotExist)
    for line in fin:
        print(line) # and other processing
    fin.close()
 except:
    print('\n%s not found' %fileDoesNotExist)
```

Appending to a non-existing file?

```
fileToAppendToDoesNotExist = "fileToAppendToDoesNotExist.txt"

# What happen when I append to a non-existing file?
fout = open(fileToAppendToDoesNotExist, 'a')
fout.write("Banana")
fout.close( )
```

Class Participation: Exercise 9.1 (Textbook Page 84)

- Post on the Canvas on Friday, June 13 by 11:59pm.
- **Think Python 2 - Exercise 9.1**: Write a program that reads `words.txt` and prints only the words with more than 20 characters (not counting whitespace). (Page 84, **Chapter 9. Case study: word play)**

Questions?