File Input/Output

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

Input for Programs

- Programs solve problems - for specific instance of the problem inputs for that instance are to be given as input to the program
- So far, we have seen only one way to give input - through terminal using input() statement/ function
- Very limiting - user has to type input data every time the program is to be used; data you can give is limited - one line
- Another way to give input is to have the inputs in a file, and then the program can read these inputs from it - we will look at it

Output

- A program must have some way to output the result to the user
- So far we have seen one way to output - on the terminal
- Very limiting - the output cannot be stored for future
- Need ways to have some permanent output
  - To a file on secondary storage (permanent)
  - To a printer - print it out
- Writing output on files is a common method for getting permanent output of programs

File

- A file is an object an operating system allows users to create which can hold persistent data, i.e. data remains after power off also
  - File is very different from main memory - which is volatile
- File is created by an Operating System- has location (directory), owner, etc. and OS allows only some operations

Files...

- Files are stored in a directory (folder) in the OS
- Directories are nested - the structure starts from root, i.e. /
- Full file name is
  - Directory path: The folder location in the file system
  - File name: User given file name
  - Extension: Often used to indicate the type of data
- To refer to a file (say within a program) - you can give the full path name of the file
- Can refer using "relative" path - relative to "current dir location" - for a program it often means where the program file resides
  - File name "myfile.txt" - if in the same directory as the program
  - subdir/myfile.txt - if it resides in subdir of the current dir
  - You can move up the hierarchy by using ".." operation

Files

- A file in any operating system generally consists of:
  - Header: Gives info about the file (size, type, owner, permissions,..)
  - Data: This is the bulk of the file
  - EOF: Special character that indicates the end of file
- Data in a file is conceptually contiguous - a sequence of bytes

File Encodings

- We will focus on text files, i.e. files whose data is text
- Encoding scheme decides how bytes are mapped to human readable characters
- ASCII (older - 128 chars) and unicode (utf-8) (new - 100K values) are the most common for text
- Text files - each line of text is terminated by new line ('\n')
- We also have binary files - no line terminator; data stored in machine readable binary format (cannot be read in an ordinary editor…)
- We limit to text files

File Access

- To access a file from a program, have to specify the location of the file to python program, so it can request the OS for the file
- To read from the file, the file has to be opened first (reqd by OS) - the owner of the program files should have rights to the file (your program can’t read someone else's file)
- On an open file, generally an OS allows - reading, writing, appending; Python provides functions for these
- After finishing with the file, the program should close it, so the OS resources for the file are freed

Opening a File in Python

- File opened by open() function; common approach:

  f = open(fname, mode)

- fname: string giving the file name - either relative to the directory where the program is, or full path

  "data.txt" or "/users/jalote/IP/DataDir/data.txt"

- ‘mode’ can be

  "w": write - file created; existing file emptied; written from top

  "a": append - written at the end, earlier contents remain

  "r": read ; can be omitted, i.e. this is the default assumed

- open() will give FileNotFoundError if file not found - execution terminates

Reading a file

- If file opened successfully, then it can be read (or written…), for text files file contents are treated as text
- Python provides a few different functions to read the file

```
f.read()  # reads the entire file, returns a string
f.read(size) # reads size bytes
f.readline() # reads one line, returns a string
f.readlines() # reads a list of strings, each item being a line
```

- To loop over lines in a file, it provides an efficient way

```
for line in f:
    <Code: can use line>
```

Reading a file

- Internally a current pointer in file is maintained
- Every time a program reads some amount of data the pointer is moved accordingly
- Further read starts from there
- Once the end of file reached - read returns an empty string
- (Can move the current file pointer by seek command - seek(0,0) will take it to the start)
- Best is to get into the habit of closing the file when done - seek needs to be used only for more advanced applications

File Input

Let us just read from a file and print it line by line

We need the file name

(i) program directory - just file name

(ii) full path

(iii) relative to the program dir

```
f = open("data1.txt", "r")
for line in f:
    print(line, end='')
```

```
f = 
open("/Users/pankaj/Documents/Teaching/IntroToProg-2022Batch/TestPrograms/data1.txt")
for line in f:
    print(line, end='')
```

```
f = open("../TestPrograms/data1.txt")
for line in f:
    print(line, end='')
```

Writing to a file

- File must be opened in "w" mode (or "a"), generally

```
f = open(fname, "w")
```

- File is created if does not exist; cleared if it exists
- write operation to write on an open file, generally

```
f.write(string)
```

- Only strings can be written - int etc must be converted
- If you want a new line in the file, have to write it (newline is treated as a character in text)
- Append is similar - except it writes contents at the end of the file, that is appends to the file

Closing a file

- Once we are done with the reading and writing operations, we need to properly close the file.
- Closing a file free up the resources associated with the file.
- To close a file f, use f.close()
- If you don’t close the file - the OS will not know when to close it - it will do it sometime in future - strange behaviour possible
  - E.g. what you write to a file may be actually written on the disk at file closing time only
- Must close the file, if you want to ensure updated file contents, and to release OS resources

Writing to a File

- To write a file, must open it with "w"
- The file will get created, if it does not exist
- If it exists, it will be first erased and then written on to
- Writing on file is done by OS - often it first writes on buffer and then on file
- Closing ensures that all is written

```
fname = input("Give File name: ")
s = input("Give string to write: ")
f = open(fname, "w")
f.write(s)
f.close()
```

Quiz

Which of the following statements is/are correct?

(a) When you open a file for reading, if the file does not exist, an error occurs.

(b) When you open a file for writing, if the file does not exist, an error occurs.

(c) f.read(5) reads the first 5 characters of a file pointed to by file object ‘f’.

(d) close() method needs to be called to close the file after the code-block of the file opened using ‘with’.

Quiz

Which of the following statements is/are correct?

(a) When you open a file for reading, if the file does not exist, an error occurs. [FileNotFoundError occurs]

(b) When you open a file for writing, if the file does not exist, an error occurs. [No error will occur, a new file with the name is created]

(c) f.read(5) reads the first 5 characters of a file pointed to by file object ‘f’. [f.read(size) reads ‘size’ bytes; 1 char -> 1 byte]

(d) close() method needs to be called to close the file after the code-block of the file opened using ‘with’ [The file is automatically closed after leaving the block]

Solution: (a), (c)

Other Info On Handling Files

- Often when we read data from files, the lines have trailing whitespaces or newline characters.
- We can use the strip() function on individual line strings to remove all trailing and leading whitespaces and newline characters.
- lstrip() is used to remove whitespaces from only left side.
- rstrip() is used to remove whitespaces from only right side.

Example

- Given a file of marks for students, check if the total for each student is correct
- Structure of the file:

Name, q1,q2,q3,..., q15,Total
Student1, 3,4,3,1,...., 10,82
Student2, 3,5,1,1,....,4,56
Student3, 3,4,0,0,3,...,2,63
....

```
fname = "ExamMarks.txt"
f = open(fname, "r")
f.readline() #read the first line

for line in f:
    sl = line.split(",")
    ml = [int(sl[i]) for i in range(1, 
len(sl))]
    if sum(ml[:len(ml)-1]) != ml[-1]:
          print(f'For {sl[0]}, sum does not 
match with given total {ml[-1]}')
f.close()
```

Using with to work with files

- If you open a file, you should close it when done
- Closing essential, e.g. if you want to re-read a file from start
- Often programmers forget to close a file - then the file remains open and file resources remain occupied.
- The output written to the file might stay in buffer until file is closed and modifications might not be visible on disk.
- You can use "with" statement:

```
with open(<fname>) as f:
    <code-block>
```

- Code-block is where you use the opened file and save data in data structures; the file will be closed after the code-block.

Quiz – Single Correct

How many lines will be present in my_file.txt after the code execution?

A. 0
B. 5
C. 3
D. 1

Initial state of my_file.txt —>

```
hello world
#2022
```

```
with open("my_file.txt","r") as f:
    data = f.readlines()
    lines = [i.strip() for i in data]

with open("my_file.txt","w") as f:
    f.write("#Introduction to programming\n")
    f.write("#Linear Algebra")
    f.write("#Digital Circuits\n")

with open("my_file.txt","a") as f:
    f.write("".join(lines))
```

Quiz – Single Correct

How many lines will be present in my_file.txt after the code execution?

A. 0
B. 5
C. 3
D. 1

Initial state of my_file.txt —>

```
hello world
#2022
```

```
with open("my_file.txt","r") as f:
    data = f.readlines()
    lines = [i.strip() for i in data]

with open("my_file.txt","w") as f:
    f.write("#Introduction to programming\n")
    f.write("#Linear Algebra")
    f.write("#Digital Circuits\n")

with open("my_file.txt","a") as f:
    f.write("".join(lines))
```

Final state of my_file.txt

```
#Introduction to programming
#Linear Algebra#Digital Circuits
hello world#2022
```

Example

(i) A file of numbers has multiple lines, each a list of integers. Create another file "squares" which has squares of them. (ii) maintaining the line structure (i.e. same no's squares in each line)

1. open file, read line by line in a loop
2. Split the line (gives list of strings), use list comprehension to get an int list
3. Use list comprehensions to create a list of squares
4. Convert this list of ints to list of strings using list comprehension
5. Join these strings using "separator".join(str_lst) function
6. Write this string in the file and add newline

```
f = open(fname, "r")
f2 = open("data3.txt", "w")
items = f.read()
ilst = [int(i) for i in items.split()]
sqlst = [i*i for i in ilst]
sqstr = [str(i) for i in sqlst]
outstr = " ".join(sqstr)
f2.write(outstr)
f.close()
f2.close()
```

```
with open("data1.txt") as f,
open("data3.txt", "w") as f2:
    items = f.read()
    ilst = [int(i) for i in items.split()]
    sqlst = [i*i for i in ilst]
    sqstrlst = [str(i) for i in sqlst]
    outstr = " ".join(sqstrlst)
    f2.write(outstr)
print("String written: ", outstr)
```

(ii) Maintaining lines

```
fname = "data1.txt"
f = open(fname, "r")
f2 = open("data4.txt", "w")

for line in f:
    ilst = [int(item) for item in line.split()]
    sqlst = [i*i for i in ilst]
    sqstr = [str(i) for i in sqlst]
    line = " ".join(sqstr)
    f2.write(line+'\n')
f.close()
f2.close()
```

Example...

Read student marks - check the total

File structure:

```
Name,q1,q2,q3,q4,q5,q6,q7,q8,q9,q10,q11,
q12,q13,q14,q15,Total
student1,3,4,3,1,3,3,5,5,5,10,6,6,8,10,10,82
student2,3,5,1,1,3,2,5,5,5,10,0,0,2,10,4,56
student3,3,4,0,0,3,3,5,5,4,10,5,4,5,10,2,63
student4,2,4,1,1,3,3,5,5,5,10,4,6,5,10,6,71
student5,4,4,2,3,3,5,5,5,5,10,8,6,0,0,0,60
```

```
fname = "ExamMarks.txt"
f = open(fname, "r")

f.readline() #read the first line
for line in f:
    sl = line.split(",")
    ml = [int(sl[i]) for i in range(1, len(sl))]
    if sum(ml[:len(ml)-1]) != ml[-1]:
        print(f'For {sl[0]}, sum does not
match with given total {ml[-1]}')

f.close()
```

Summary

- Files make it easier to give inputs to the program and record outputs produced by the program.
- Opening a file, f = open(file_name, "rwa")
- Reading a file f.read(), f.readline(), f.readlines(), f.read(n)
- Writing a file : f.write(s) # string s is written
- Appending a file : f = open(<file_name>,“a”)
- Closing a file : f.close()

Minute Paper

Take a few minutes to summarize your learning for today

And the issue that is still not clear