File I/O Part I

October 19, 2020

First, some software development tips

TEST AS YOU GO!!!

That means if you're developing a program using multiple functions, write a function and test it. Make sure it works before moving on and writing other functions

- If you write multiple functions at a time, you'll never be able to debug it properly!!!

An example

Suppose that you need to write a program that prompts a user to enter a string of names, then finds the number of letters in each name, and then sums the numbers of letters. You use functions. Let’s write and test each function as we go.

And now to File I/O

Input and Output in Python

Up until now, all input has been from the keyboard and all output has been to the screen

X = input(“kindly hand over the data I need.”)

- Everything comes in as a string; you have to cast it to another type if you need integers, floats, etc.

print (“the output is”, output)

That has limited usefulness

It’s a pain in the neck to ask the user to type in data, over and over and over…

You can’t easily preserve your program results for later use, unless you do a whole bunch of screenshots

It would be far better if you could have all the data stored in a file that was saved on a computer, so it didn’t have to be typed in

And it would help if you could save your output in a file that you could reference later on, or even use for further processing

Fortunately, you can!!!

File Input in Python

You can read from any file that can be found on the computer - that is in the “Python path”

- Exactly how this works varies by operating system, but there is an environment variable that can be set that tells the Python interpreter where to look for any files you reference

The easiest thing to do is just make sure that your data files are in the directory in which you’re working

- Or you can specify the full path in the **open** statement

Opening a file

Before you can read from or write to a file in Python, you have to open in

There are several different open statements; we'll use

```
with open("filename", "mode") as f:
	#rest of your code indented one tab
```

"

"Mode" is one of:

- "r" if you just want to read from the file
- "r+" if you want to both read from and write to the file - modify existing contents
- "w" if you want to write to the file - ERASE EXISTING CONTENTS!!
- "a" if you just want to append to the file - add new content without modifying existing content

Modes

- If you leave off the mode value, it defaults to “r” - you can read from the file but you can’t modify it
- Using “w” gives you a new, empty file!! If the file previously existed, any content that was there before will be erased.
- Using “a” does not erase existing content; it just moves the “stream pointer” to the end of the existing file and adds from there

Text mode

For our purposes, all files are just text. They consist of one or more strings and lines of characters. All read commands return strings.

There are “binary” files in Python that are different, but we’re not going to talk about them right now.

Reading from a file

File name in the current directory or in the path

```
with open(“test_data.txt”, “r”) as f:

        data = f.read()  #reads in all data from the file as a single string

        data_lines = f.readlines() # reads in all data from the file

                                        # returns a list of strings - each element in the

                                        #list is one line from the file

        next_line = f.readline() #reads in one line from the file; returns it as a string
```

Use one of these three

How it works

There is a ***file pointer*** for any open file, that points to the next character to be read or the place where the next item will be written.

When you open a file for reading, the file pointer starts at the beginning of the file.

Each time you read a line, the file pointer is advanced to the start of the next line.

When you open a file for writing, the file pointer starts at the beginning of a new file.

When you open a file for appending, the file pointer starts at the end of the current content

Reading a file - an example

```
# read in a file of integers, and print the largest
integer

with open(“integers.txt”, “r”) as f:
        #read the whole file in as a single string
        data = f.read()
        #split the string into components at
whitespace
        numbers = data.split()
        #we have a list of strings, each of which
must be converted to integers
        for i in range(len(numbers)):
                numbers[i] = int(numbers[i])
```

```
        #now find the largest integer. Start by
        #setting the “largest” to the first value
        Largest = numbers[0]
        #now see what’s bigger
        for i in range(len(numbers)):
                if numbers[i] > largest:
                        largest = numbers[i]
        # now print out the largest value
        print(“The largest integer in the file was”,
largest)
```

What if we wanted to write the result to a file?

“print” puts the output on the screen. What if we wanted it to go to a file?

The file.write() method will work

```
with open(“results.txt”, “w”) as out:
        s = ‘ ‘.join(“The largest integer in the file is”, largest)
        out.write(“The largest integer in the file is”, largest)
```

Next example: read from & write to the same file

Read data from a file

Manipulate the data - replace every occurrence of the word “dog” with the word “cat”

Write the modified data back out to the same file we started with

Be careful how you do this! You don’t want to erase all the data in the file before you’ve finished reading it!!!