More on Multi-level lists and File I/O

October 23, 2023

Administrative Notes

Homework 5 is due

Project 1 is out - due November 6 (the Monday of Exam week)

Where we are

Last Wednesday I tried to simultaneously introduce two major topics:

- Multi-level lists
- File I/O

Let’s try to dive into each, with a bit more organization

Multilevel Lists

A two-dimensional, or 2D, list, is just a list whose elements are themselves lists

E.g.,

```
#2D lists
results = [
   [1, "USA", 8, 3, 11],
   [2, "GB", 7, 2, 9]
   [3, "Kenya", 5, 3, 8]
   ]
```

You can have lists as many levels deep as you want

A 3-D list

```
#3D lists
heat_results = [
   [
   [1, "USA", 8, 3, 11],
   [2, "GB", 7, 2, 9],
   [3, "Kenya", 5, 3, 8]
   ],
    [
   [4, "Nigeria", 1,2, 3],
   [5, "China", 4, 0, 4]
      ]
   ]
But in this class we won't really use things beyond 2D lists - they're
Just not practical
```

So what do you need to know about 2D lists

1. Syntax:
    a. An element is identified by two subscripts:
        i. Row comes first
        ii. Column comes second
        iii. Each subscript is contained in its own square brackets
    b. If there is only one subscript, it refers to a row in the 2D list
2. 2D lists in Python are row-oriented
    a. One subscript => row. There is no easy way to refer to a column; you have to use a loop
    b. You can add a row by appending it or inserting it the same way you do with a one-dimensional list
    c. The same with remove, pop or del
    d. You can’t do that with a column. You have to use a loop to add or remove (or do anything to) all the elements in a column

What you need to know about 2D lists (cont.)

3. Do all rows in a 2D- list have to be the same? No

E.g.,

```
results = [
   [1, "USA", 8, 3, 11],
   “vacated”,
   [3, "Kenya", 5, 3, 8]
   ]
```

If you do this with your data structures, you have to be very careful with your code.

If you tried to do something with a row or column and all that was there was one string, your program could fail in interesting ways.

2D lists

If you know those three big things about multi-dimensional lists, you’re in good shape.

File I/O

So why did I try to introduce File I/O and 2D lists at the same time?

- Because most 2D lists in your programs are going to be loaded from or output to files; you won't ask users for that much data

But can't file I/O be done without worrying about 2D lists?

- Yes, of course. Let's look at that.

Suppose we have a file of integers

“Our_integers.txt”

1
2
3
4
5
6
7
8
9
10

We want to read in these numbers, and calculate their sum, their product; and their average.

We have three statements we can use to read data from files:

read() : reads the entire contents of the file in as a single string

readlines() : reads the entire contents of the file in as a list of strings; each string is the contents of one line in the file

readline() : reads one line of hte file as a string; does not touch the rest of the file

Opening and closing files

Before you can use a file in Python (either read or write), you must open it.

After you are done, you must close the file to ensure that the system understands exactly what the status is.

There are several ways to do this in Python, but in this class we teach:

    with open (“filename”, “mode”) as internal_variable_name:

Why? Because doing it this way has the easiest syntax, AND it automatically closes the file for you when you’re done - you don’t have to remember anything.

Three ways to read in the file:

```
with
open("our_integers.
txt", "r") as
infile:

   data =
infile.read()
```

```
with
open("our_integers.t
xt", "r") as infile:

   data_list =
infile.readlines()
```

```
with open("our_integers.txt", "r") as
infile:

   data = infile.read()

   print(data)
```

```
with open ("our_integers.txt", "r") as
infile:

   nums = []

   line = infile.readline()

   while line != "":

       nums.append(int(line))

       line = infile.readline()
```

The difference is what you have after reading

:with “read” you have a single string; so you have to split it up and then convert each entry into an integer

```
nums = data.split()
for i in range(len(nums)):
    nums[i] = int(nums[i])
```

With “readlines” you have a list of strings; you need to convert each list item into an integer

```
for i in range(len(data_list)):
    data_list[i] = int(data_list[i]
```

With “readline” the code you’ve written has already done the splitting and conversion you need

File Output

There are only two options for writing to a file:

“write” - write one string to a file; with no newline character at the end

“writelines” - write one or more strings to a file; adding a newline after each string

There is no “writeline” (singular) method in Python.

Which you use is really up to you . If you want to do all output with one write statement, it’s better to use “write()”

If you want to output a little bit at a time, use “writelines” in a loop or at multiple points in your program

Remember that regardless of “write” or “writelines” you can ONLY output strings.