Chapter 5: File I/O

CSCI-UA 0002 – Introduction to Computer Programming
Mr. Joel Kemp
Our New Task!

Surf-A-Thon

The scoreboard is currently empty. Wonder who won today’s contest?
Scores!

- We need to process the scores from the competition.
  - The judges stored their results in a text file.

Uhh, so how are we supposed to use this file?
File Handling

• We can manipulate the file using `open()` and `close()` functions.

• If you need to open a file in Python, use the `open()` method.
  – When you’re done with the file, be sure to `close()` it so that we release that memory!

• Example:

```python
file_handle = open("results.txt")
# Load the data from the file...
# Process the data somehow...
file_handle.close()
```
for loop to the rescue!

- We need to go through the file line by line.
  - We can actually achieve this by looping!

Refer to: fileIO.py
Back to the Competition!

• So we now know how to read from a text file.
• Our goal is to have the top/highest three scores on the podium.
• Let’s start by getting the first highest score.
• **Exercise:**
  – Given a list of numbers, write an algorithm to find the highest/largest number!

| 8.65 | 9.12 | 8.45 | 7.81 | 8.05 | 7.21 | 8.31 |
One solution

Computing the highest score:

For each element that we read

- Check to see if it is larger than the highest score seen so far.
- If yes, then store the current element as the highest score.

Continue reading until the end of the file.
Exercise

• Using the algorithm that you came up with, write a program that finds the highest score in the results.txt file.
  – Remember, the `for` loop creates a string from each line in the file.
  
    • **Hint:** You’ll need to convert the string into a number!

```python
file_handle = open("results.txt")

highest_score = 0

for line in file_handle:
    line_float = float(line)

    if(line_float > highest_score):
        highest_score = line_float

file_handle.close()

print("The highest score was", highest_score)
```

Refer to: highestScore.py
Hurray! It works!

The results are in! Or are they...?

Surf-A-Thon
1. 9.12
2. 
3. 

Ummm ... those are the top scores?!?!
The Top 3

• We gave the highest score, but what about the 2\textsuperscript{nd} and 3\textsuperscript{rd} place finishers?

• Our program needs to keep track of the three highest scores!!
Brainstorming
What would make the solution easier?

☐ If the data were ordered highest to lowest

☐ If you knew exactly how many lines are in the file

Ordered data makes things much easier!

☑ If the data were ordered highest to lowest

☐ If you knew exactly how many lines are in the file

• Ordering data in either ascending or descending order is known as sorting!

How would this help in our program?
The Sorted Idea

The unsorted data

- 8.65
- 9.12
- 8.45
- 7.81
- 8.05
- 7.21
- 8.31

An ordered (or sorted) copy of the same data

- 9.12
- 8.65
- 8.45
- 8.31
- 8.05
- 7.81
- 7.21

The top three scores are just the first three values in the sorted data, extracted to variables. Easy!
But wait!
• We don’t actually remember all of the numbers...
  – So how can we compute the top three if we only remember one number at a time?
  – We’ll need to store them! But where?
    • Variables!
• So how many variables do we need?
  – 7 variables in our situation.
A Variable Per Line

- That wasn’t so bad for three scores, but how about four scores? Or five? Or even worse, what about 10,000 scores? Hmm...
  - Good luck typing out 10,000 variable names!

- To deal with lots of data all at once, Python provides you with a container called an **array**!
  - Also known as a “list.”
Arrays

- **Definition**: a programming construct that groups similar types of data, referenced as sequential locations in memory.

- Think of an array as a data train where each car is called an **array element** and can store a single piece of data!

- Example:

  ```python
  my_array = [7, "24", "Fish", "hat stand"]
  ```

  – We have a single variable that represents the train carrying the data!
Example Array

```
my_array = [7, "24", "Fish", "hat stand"]
```

We can just treat `my_array` like any other variable.

But, how do we use it?
Python Arrays

• Arrays in Python are actually called **Lists**.

```python
my_words = ['Dudes', 'and']
```

• To get access to the individual array elements:
  – We use the element’s **index** (i.e., position in the list)!

```python
>>> print(my_words[0])
Dudes
>>> print(my_words[1])
and
```
Playing with Lists!

• How do we add data to the list?
  – The `append()` function!

    ```python
    >>> my_words.append("Bettys")
    >>> print(my_words[2])
    Bettys
    ```

Okay, let’s do an exercise!
Easy Exercise

• Write a program that keeps asking the user to input data while their input is not “-1”.
  – Add each piece of data to a list named data.
  – Stop the program and print out the list when the user enters -1.

• Note: You need to have a list before you can append to it!
  – Initialize an empty list!
    ```python
    data = []
    ```
  – Reminder: appending to a list
    ```python
    data.append(“Booyah!”)
    ```

Refer to: easyExercise.py
More Exercises

• Given the list, \texttt{words}:

<table>
<thead>
<tr>
<th>“The”</th>
<th>“quick”</th>
<th>“brown”</th>
<th>“fox”</th>
<th>“jumps”</th>
<th>“over”</th>
<th>“the”</th>
<th>“lazy”</th>
<th>“dog”</th>
</tr>
</thead>
</table>

1. Write a program that computes and prints the average word length!
   • \textbf{Tip:} \texttt{len(some\_string)} gives you the number of chars!
   • \textbf{Tip:} \texttt{len(some\_list)} gives you the number of elements!
   • \textbf{Tip:} Use a \texttt{for} loop to iterate over the list!

```python
for word in words:  
    print(word)
for i in range(len(words)):  
    print(words[i])
```

2. Write another program that reverses the list of words!
   • \textbf{Hint:} Create another list to hold the reversed elements or swap elements using a temporary variable!

Refer to: moreExercises.py
Back to the Competition!

• So now we know how to read the scores from a file **and** store the scores in a list!
• But how do we **sort** the list of numbers?
# List Methods! Yay!

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>append</td>
<td>Adds an item at the end of the list.</td>
</tr>
<tr>
<td>count</td>
<td>Tells you how many times a value is in the list.</td>
</tr>
<tr>
<td>extend</td>
<td>Adds a list of items to another list.</td>
</tr>
<tr>
<td>index</td>
<td>Looks for an item and returns its index value.</td>
</tr>
<tr>
<td>insert</td>
<td>Adds an item at any index location.</td>
</tr>
<tr>
<td>pop</td>
<td>Removes and returns the last list item.</td>
</tr>
<tr>
<td>remove</td>
<td>Remove the first item from the list of a given value.</td>
</tr>
<tr>
<td>reverse</td>
<td>Reverses the order of a list.</td>
</tr>
<tr>
<td>sort</td>
<td>Sorts the list in ascending order (smallest to largest).</td>
</tr>
</tbody>
</table>

Refer to: listMethods.py
So... how do we sort?

• We’ll definitely need the `sort()` function
  – But it defaults to ascending order 😞
  
  `scores.sort()` would give us this list:

| 7.21 | 7.81 | 8.05 | 8.31 | 8.45 | 8.65 | 9.12 |

  – Getting the top 3 scores is still feasible...

    • How do we print the last three numbers?
      
      ```python
      print(scores[len(scores)-1])
      print(scores[len(scores)-2])
      print(scores[len(scores)-3])
      ```

• Can we do better?
  – What if we had the list in descending order?
reverse()

• What does it do?
  – Reverses a list!

• Example:
  a = [1,2,3]
  a.reverse()
  print(a) #yields [3,2,1]

• So how can we use this?
  scores.sort()
  scores.reverse()

• So how do we print out the top three scores now?
  print(scores[0])
  print(scores[1])
  print(scores[2])

Refer to:
threeHighest.py
And the winners are!

Surf-A-Thon

1. 9.12
2. 8.65
3. 8.45

Dude, those are some sweet scores.