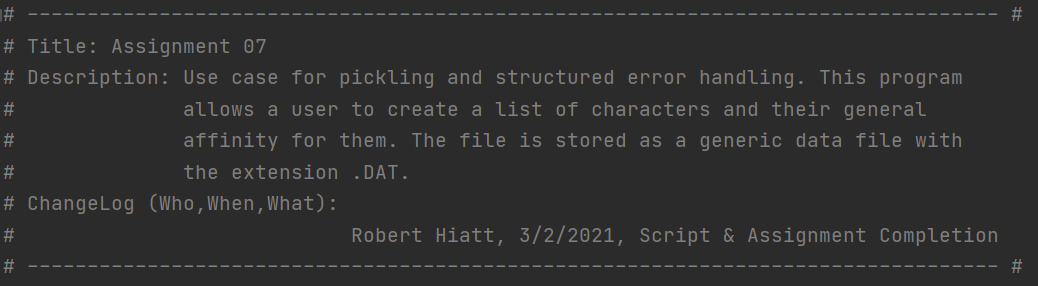
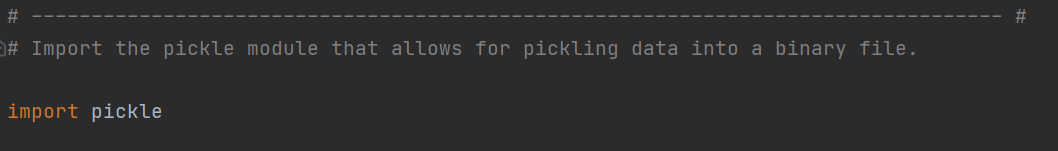
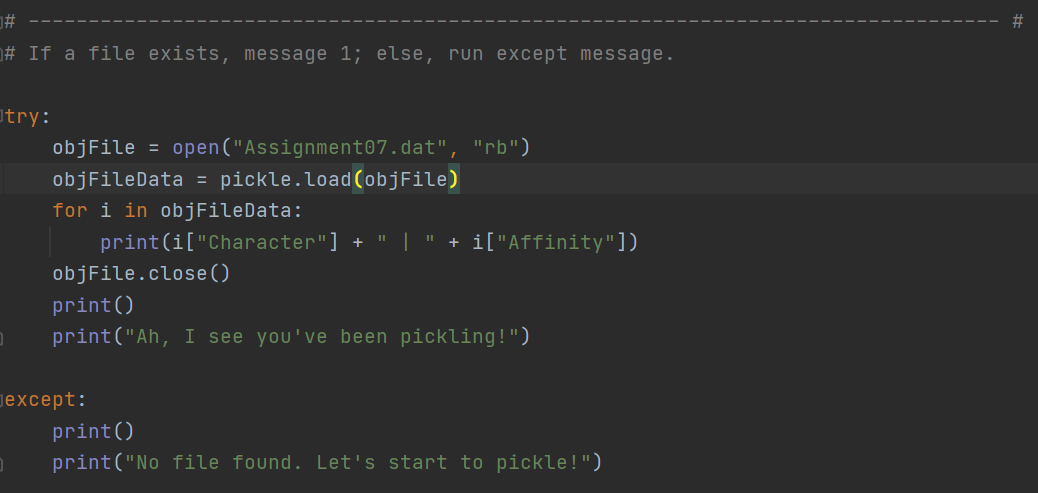
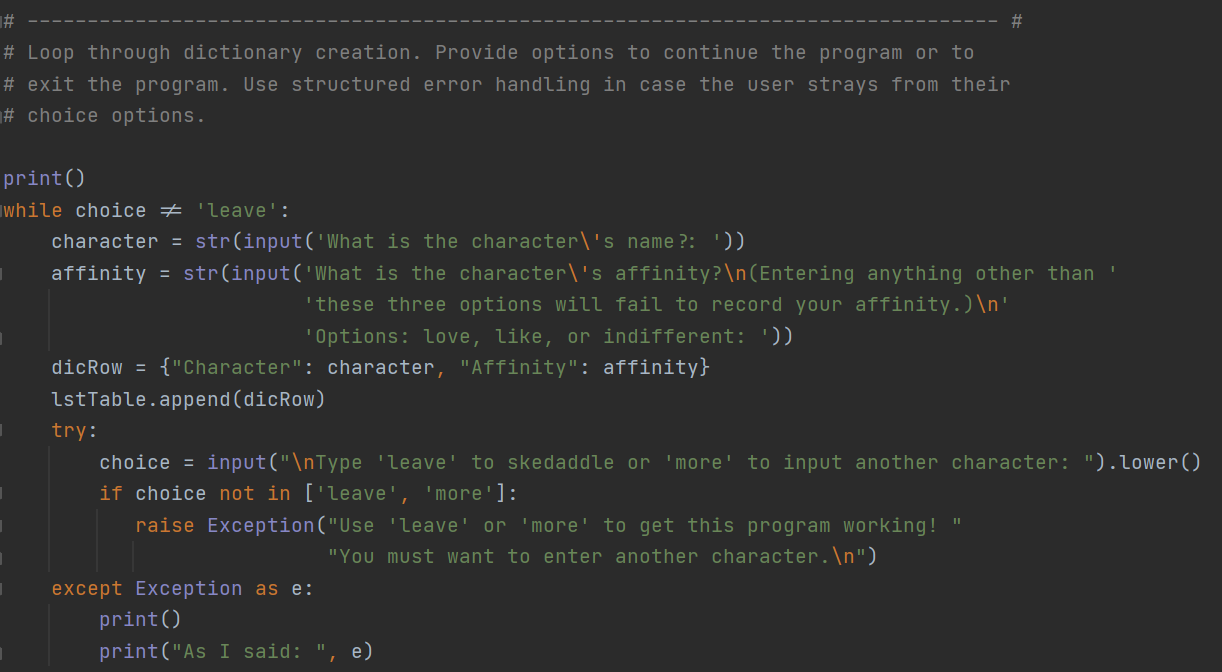
Robert F. Hiatt  
  
March 2, 2021  
  
IT FDN 110 B  
  
Assignment07  
  
[GitHub URL](https://github.com/roberthiatt/ITFDN110B-Mod07)

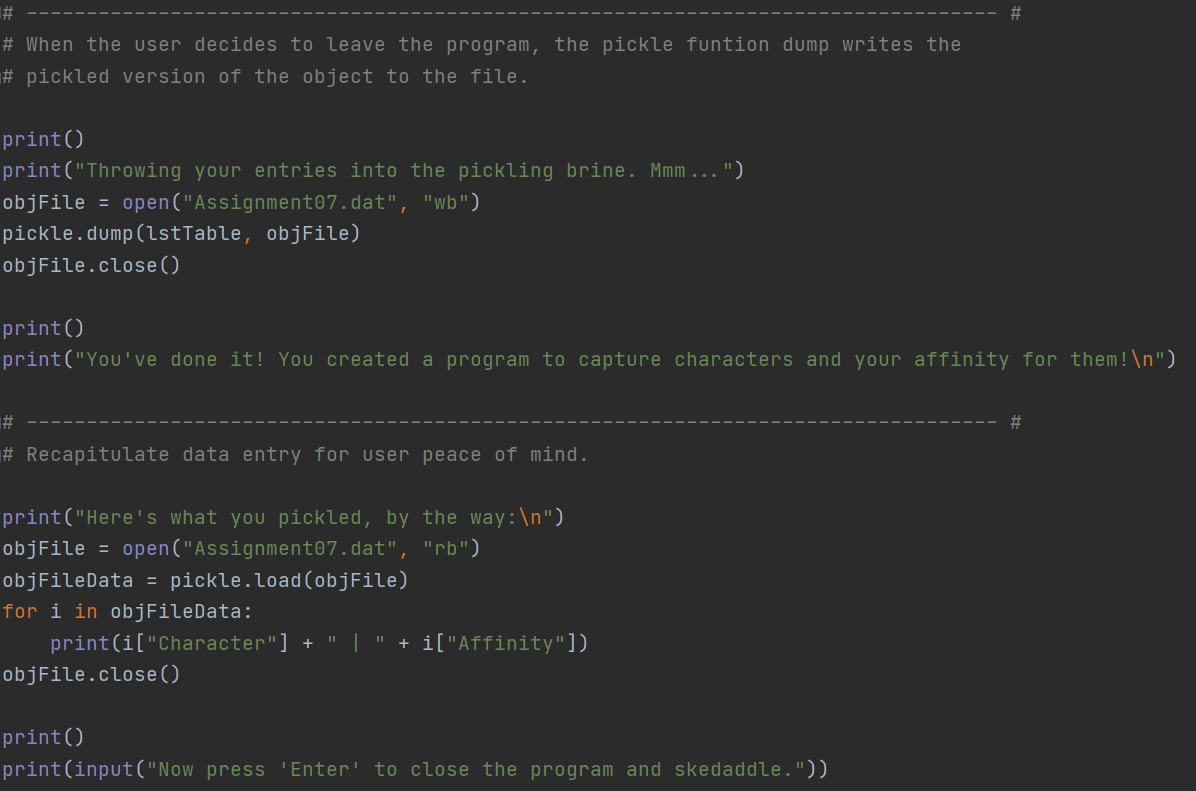
Pickling and Structured Error Handling in Python

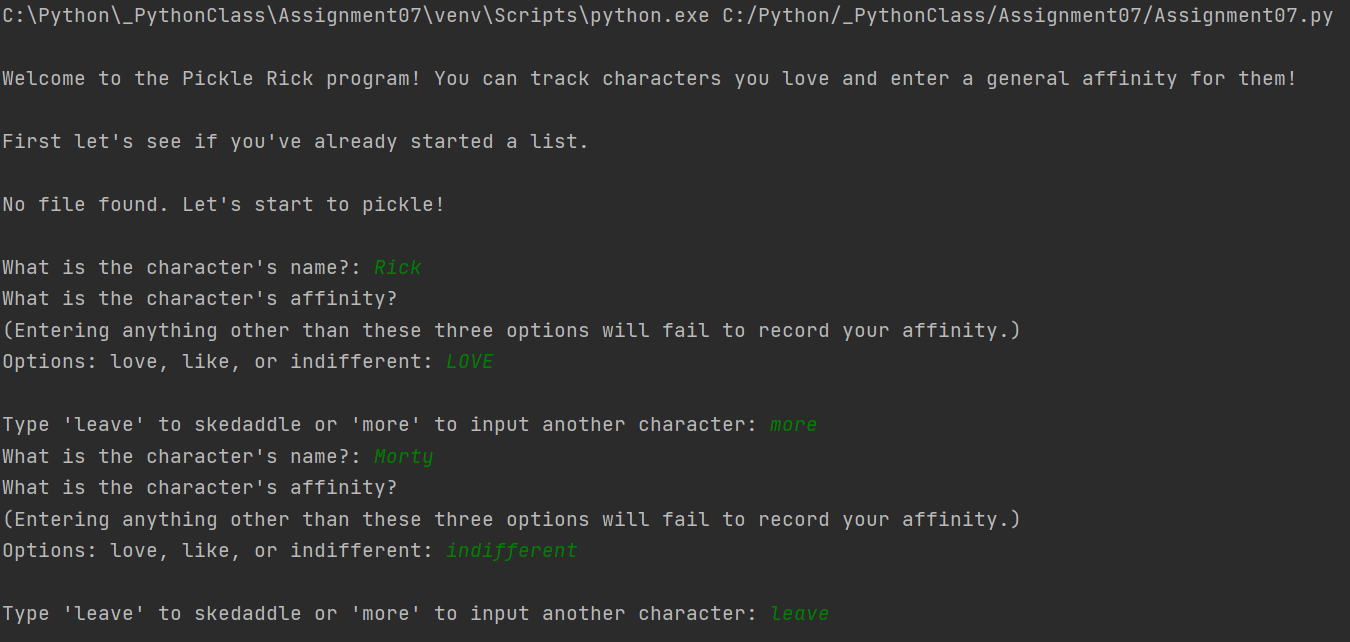
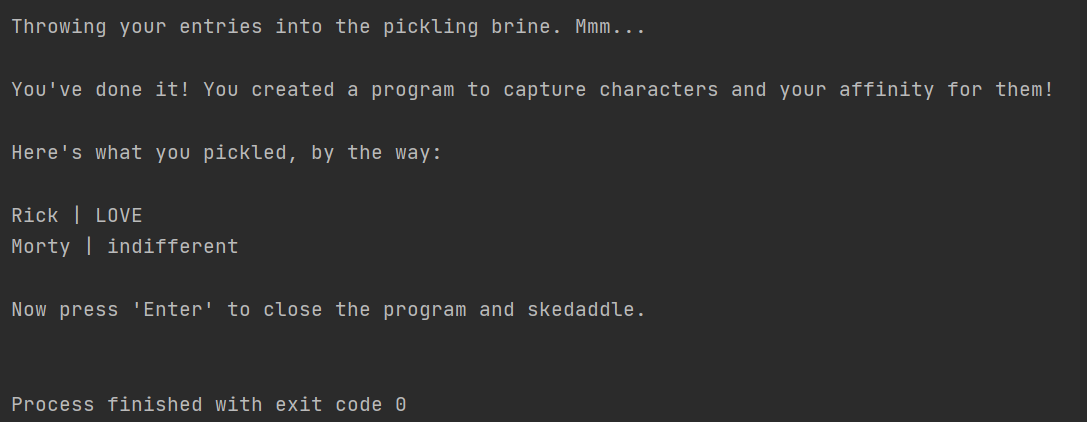
1. Introduction  
This program is a use case for importing the pickle module, employing structured error handling, and creating/managing binary files. Separately, there is a corresponding GitHub page that explains the concept behind pickling and structured error handling and also reviews this program.   
  
2. Program Review   
The core ask is to demonstrate pickling and structured error handling. The chosen path to do so is to create a list, which recalls prior lessens. With this list, though, we do not want to store the information in a text file but rather a generic data file that has the extension .DAT. Further, we need to understand how to implement pickling and the purpose of pickling. But, as always, let’s start with the script header:  
  
  
***Script header with updated ChangeLog.***So, what’s pickling? It’s a concept in Python, and the name comes from the cooking world where you want to preserve food. In like manner, the pickle module “allows you to pickle and store more complex data in a file…Instead of writing characters to a file, you can write a pickled object to a file” (Dawson 201). Pickled objects have to be stored in a binary file, hence the .DAT extension. Lastly, note that *pickle* is a built-in Python module.   
  
  
***Importing the* pickle *module.***  
  
The program proceeds to initialize variables and greet the user, which is old hat at this point in the course.

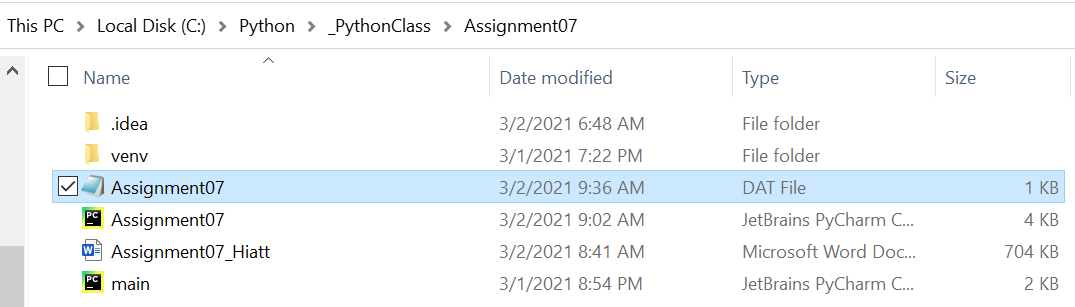
***Initialize your variables, greet the user, and tell the user your program’s purpose.***  
  
Now I would like to focus on the *new*, which here refers to structured error handling.   
  
In the image below, you see the first implementation of structured error handling. You’ll note the usage of the *try* statement and the *except* clause.  
  
As Dawson puts it, the *try* statement allows you to “section off some code that could potentially raise an exception” (206). Specific to the script below, at the program’s onset we want to open a file (if a file does exist) and we want to display the hypothetical file’s contents, substantiated with the binary file access mode of *rb*—*rb* means to read from a binary file.   
  
But what if the file doesn’t exist? That’s exactly where the *except* clause comes into view. The *except* clause executes statements when exceptions are raised. Germane to the example in discussion, if there is no pre-existing file then the program will tell the user as much. We could potentially include *exception types*, such as the *FileNotFoundError*, but for the scope of this use case we’ll stick to the basic premise of *try* and *except*.

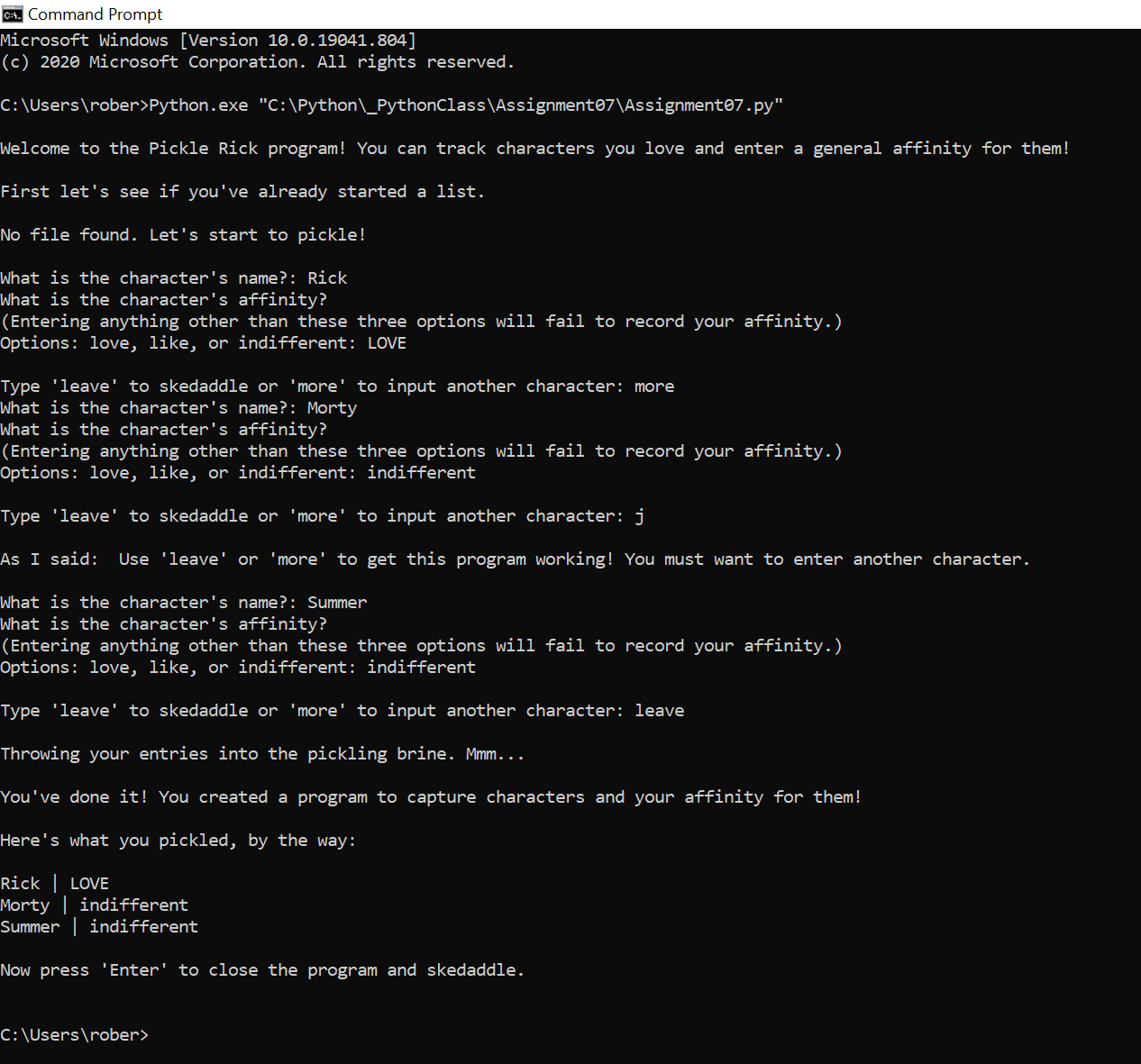
  
***Example of* try *statement and* except *clause*.**   
  
Now let’s look at the list/dictionary creation portion of our program (image below). We start with the familiar conditional *while* loop that states if the user doesn’t want to “leave” the program then keep on going with character and affinity cataloging. We append the entries to the table via the *append* method and then we come to an additional example of structured error handling.

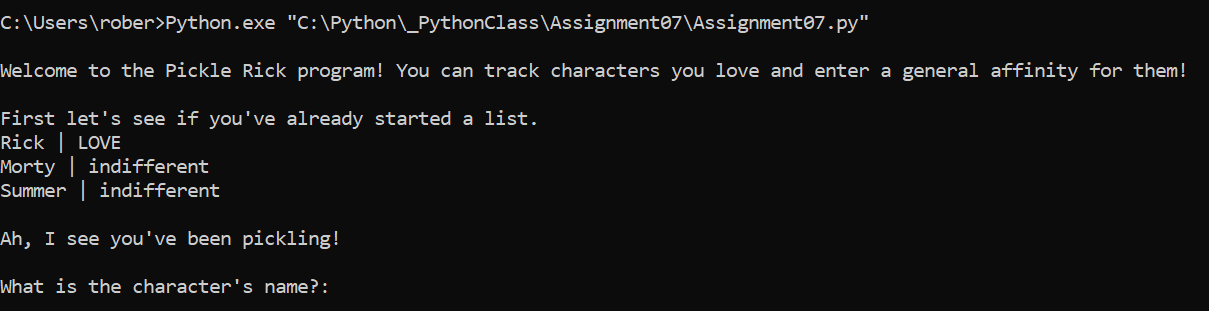
With the latest *try* statement and *except* clause, we want to manage if the user does not enter in one of the two required commands: “leave” or “more.” If the user enters anything otherwise, the *except* clause kicks in and snidely tells the user the *Exception* that is *raised* below:   
  
“As I said: Use ‘leave’ or ‘more’ to get this program working!”  
  
Note that there’s also an if statement employed in this *try* statement, to limit the user’s options. The use of *raise* also allows us to create an *Exception* variable, which we may then alias in our *except* clause and integrate into our playful reminder to the user to stick to the script.   
  
  
***A more complex example of structured error handling.***  
  
Once the user decides to leave the program, we then employ a *pickle function* known as *dump*. As the name suggests, the *dump* function simply takes the data entry and saves aka dumps it into the file. Note that there is an additional *binary file access mode* used—whereas previously we used “rb” to read the file at the program’s onset, this time we’re using “wb” to write to the binary file.   
  
Lastly, we want the user to feel good about their productivity, so before we leave the program fully, we take the pickled object and retrieve the stored data, printing it to the screen so that the user sees one more time what they’ve entered.



***Example of* pickle functions *and accessing sequential pickled data.***3. Running the Program With the new lessons explained for Assignment 07, let’s run the script in both PyCharm and the Command Line to ensure that the program is working properly.   
  
  
***PyCharm: Welcoming the user, explaining the program, moving through data entry and choices.***  
  
  
***PyCharm: Choosing to leave the program, show the user what they entered, and skedaddle!***

  
***Confirmation of Assignment07 DAT file created.***

Now let’s have a go on the Command Line:  
  
  
***Command Line: The program ran successfully on the Command Line. Note that this screenshot uses an entry separate from ‘leave’ or ‘more’ to demonstrate the exception clause.***And here is an example of the program running on the Command Line once we’ve already created a file:



4. GitHub Page   
  
The assignment’s final ask was to create a GitHub Webpage, utilizing the markdown language known as Jekyll.

5. Summary  
By this exercise’s completion you’ve demonstrated an understanding of pickling, how to write structured error handling, and you’ve also created a GitHub page that more fully delves into using the markdown language known as Jekyll. Regarding the GitHub page, you embedded images and also used hyperlinks to cite your research sources. Nice and nicely done!

Works Cited

Dawson, Michael. *Python Programming for the Absolute Beginner, Third Edition*. Course Technology, 2010.