Carolina Cerda

May 24, 2022

Foundations of Programming: Python

Assignment 06

https://carolinacerda.github.io/IntroToProg-Python-Mod06/

Using Functions and Classes to Create a Script to Manage a To-Do List

Introduction

In this assignment, a script that manages a To-Do List will be created using functions and classes to build upon a starter script provided by Professor Root. Additionally, this work will be posted to a new GitHub repository that includes a GitHub webpage.

Creating the Script

PROCESSING

This section of the script is dedicated to building a class to group functions, or a group of code that performs a specific action, that will process the data within the program.

Adding Data to the List

This function adds data to a list of dictionary rows. From the provided starter script, to complete this function, the dictionary, "row", is appended to the list, "list_of_rows" as written in Line 41. This function thus becomes useful in that it helps to separate the processing portion of the code from the main body and overall makes the code easier to understand and delineate what is processing and what is presentation.

```
31      @staticmethod
32      def add_data_to_list(task, priority, list_of_rows):
33          """ Adds data to a list of dictionary rows
34
35          :param task: (string) with name of task:
36          :param priority: (string) with name of priority:
37          :param list_of_rows: (list) you want filled with file data:
38          :return: (list) of dictionary rows
39          """
```

```
40          row = {"Task": str(task).strip(), "Priority": str(priority).strip()}
41          list_of_rows.append(row)
42          return list_of_rows
```

Removing Data from the List

To create a function that removes data from the list, a for loop and if statement are used to find a row in the list, “list_of_rows”, that match each other, using the lower() method to ensure that the lower case of these values are compared, and if the value for the keyword, “task”, matches the task that is inputted by the user then the row is removed from the list using the remove() method (Lines 52 – 54). The list is then returned using the return statement that ends the functions and returns the result to what calls to it (Line 55).

```
44      @staticmethod
45      def remove_data_from_list(task, list_of_rows):
46          """ Removes data from a list of dictionary rows
47
48          :param task: (string) with name of task:
49          :param list_of_rows: (list) you want filled with file data:
50          :return: (list) of dictionary rows
51          """
52          for row in list_of_rows:
53              if row["Task"].lower() == task.lower():
54                  list_of_rows.remove(row)
55          return list_of_rows
```

Writing Data to the File

This function writes data from a list of dictionary rows to a File. To accomplish this, first, a file must be opened using the open() function and writing in the two parameters which are the file name, “file_name”, and the write mode, “w”, that will write the data to the file (Line 65). A for loop is created to so that the row in the “list_of_rows” is written into the file using the specified format (Lines 66 – 67). Then the file is closed using the close() function (Line 68).

```
57      @staticmethod
58      def write_data_to_file(file_name, list_of_rows):
59          """ Writes data from a list of dictionary rows to a File
60
61          :param file_name: (string) with name of file:
62          :param list_of_rows: (list) you want filled with file data:
63          :return: (list) of dictionary rows
64          """
65          file = open(file_name, "w")
66          for row in list_of_rows:
67              file.write(row["Task"] + ", " + row["Priority"] + "\n")
```

```
68          file.close()
69          return list_of_rows
```

PRESENTATION (INPUT/OUTPUT)

This section of the script is dedicated to building a class of functions that will obtain user input and present data output within the program.

Inputting a New Task and Priority into the To-Do List

To add a new task and its priority to the To-Do List, first input from the user must be obtained. Firstly, instruction is printed that states, “Type in your task and its priority.” (Line 121). Then, two variables are created, “task” and “priority”, that use the input() function to firstly obtain data from the user about what the task is and what is its priority, respectively, that is then returned as a string through the str() function and also uses the strip() method to remove leading and trailing characters, such as extra spaces (Lines 122 – 123). An extra line is added afterwards for looks (Line 124). Lastly, the return statement is used to end the function and return the results of “task” and “priority” and use the title() method to ensure consistency of the data that is inputted by the user by capitalizing each of the words in the string (Line 125).

```
115     @staticmethod
116     def input_new_task_and_priority():
117         """ Gets task and priority values to be added to the list
118
119         :return: (string, string) with task and priority
120         """
121         print("Type in your task and its priority.")
122         task = str(input("  Enter a task: ")).strip()
123         priority = str(input("  Enter a priority [High|Medium|Low]: ")).strip()
124         print()  # Add an extra line for looks
125         return task.title(), priority.title()
```

Removing a Task from the To-Do List

To remove a task from the To-Do List, first input from the user must be obtained in order to specify which task they want to have removed. Thus, Line 133 creates a variable, “task”, that uses the input() function to firstly obtain that information that is then returned as a string through the str() function and also uses the strip() method to remove leading and trailing characters, such as extra spaces. An extra line is added afterwards for looks and then the return statement is used to end the function and return the result of “task” (Lines 134 – 135).

```
127     @staticmethod
128     def input_task_to_remove():
129         """ Gets the task name to be removed from the list
130
```

```
        :return: (string) with task
        """
        task = str(input("  Which task do you want to remove?: ")).strip()
        print()  # Add an extra line for looks
        return task
```

Running the Script

By building upon the provided starter script, the following completed code that manages a To-Do List is displayed below. The highlighted sections emphasize what was altered or added from the original starting script. Lines 86 – 87, Line 163, and Line 167 were altered from the original for aesthetic purposes and thus were not previously discussed in the sections above.

```
# Data ---------------------------------------------------------------------- #
# Declare variables and constants
file_name_str = "ToDoFile.txt"  # The name of the data file
file_obj = None  # An object that represents a file
row_dic = {}  # A row of data separated into elements of a dictionary {Task,Priority}
table_lst = []  # A list that acts as a 'table' of rows
choice_str = ""  # Captures the user option selection


# Processing  --------------------------------------------------------------- #
class Processor:
    """  Performs Processing tasks """

    @staticmethod
    def read_data_from_file(file_name, list_of_rows):
        """ Reads data from a file into a list of dictionary rows

        :param file_name: (string) with name of file:
        :param list_of_rows: (list) you want filled with file data:
        :return: (list) of dictionary rows
        """
        list_of_rows.clear()  # clear current data
        file = open(file_name, "r")
        for line in file:
            task, priority = line.split(",")
            row = {"Task": task.strip(), "Priority": priority.strip()}
            list_of_rows.append(row)
        file.close()
        return list_of_rows

```

```
31      @staticmethod
32      def add_data_to_list(task, priority, list_of_rows):
33          """ Adds data to a list of dictionary rows
34
35          :param task: (string) with name of task:
36          :param priority: (string) with name of priority:
37          :param list_of_rows: (list) you want filled with file data:
38          :return: (list) of dictionary rows
39          """
40          row = {"Task": str(task).strip(), "Priority": str(priority).strip()}
41          list_of_rows.append(row)
42          return list_of_rows
43
44      @staticmethod
45      def remove_data_from_list(task, list_of_rows):
46          """ Removes data from a list of dictionary rows
47
48          :param task: (string) with name of task:
49          :param list_of_rows: (list) you want filled with file data:
50          :return: (list) of dictionary rows
51          """
52          for row in list_of_rows:
53              if row["Task"].lower() == task.lower():
54                  list_of_rows.remove(row)
55          return list_of_rows
56
57      @staticmethod
58      def write_data_to_file(file_name, list_of_rows):
59          """ Writes data from a list of dictionary rows to a File
60
61          :param file_name: (string) with name of file:
62          :param list_of_rows: (list) you want filled with file data:
63          :return: (list) of dictionary rows
64          """
65          file = open(file_name, "w")
66          for row in list_of_rows:
67              file.write(row["Task"] + ", " + row["Priority"] + "\n")
68          file.close()
69          return list_of_rows
70
71
72  # Presentation (Input/Output)  -------------------------------------------- #
73
74
```

```
75  class IO:
76      """ Performs Input and Output tasks """
77
78      @staticmethod
79      def output_menu_tasks():
80          """  Display a menu of choices to the user
81
82          :return: nothing
83          """
84          print('''
85          Menu of Options
86          1) Add a New Task
87          2) Remove an Existing Task
88          3) Save Data to File
89          4) Exit Program
90          ''')
91          print()  # Add an extra line for looks
92
93      @staticmethod
94      def input_menu_choice():
95          """ Gets the menu choice from a user
96
97          :return: string
98          """
99          choice = str(input("Which option would you like to perform? [1 to 4] - ")).strip()
90          print()  # Add an extra line for looks
100         return choice
101
102     @staticmethod
103     def output_current_tasks_in_list(list_of_rows):
104         """ Shows the current Tasks in the list of dictionaries rows
105
106         :param list_of_rows: (list) of rows you want to display
107         :return: nothing
108         """
109         print("******* The current tasks ToDo are: *******")
110         for row in list_of_rows:
111             print(row["Task"] + " (" + row["Priority"] + ")")
112         print("*********************************************")
113         print()  # Add an extra line for looks
114
115     @staticmethod
116     def input_new_task_and_priority():
117         """  Gets task and priority values to be added to the list
```

```
118
119         :return: (string, string) with task and priority
120         """
121         print("Type in your task and its priority.")
122         task = str(input("  Enter a task: ")).strip()
123         priority = str(input("  Enter a priority [High|Medium|Low]: ")).strip()
124         print()  # Add an extra line for looks
125         return task.title(), priority.title()
126
127     @staticmethod
128     def input_task_to_remove():
129         """  Gets the task name to be removed from the list
130
131         :return: (string) with task
132         """
133         task = str(input("  Which task do you want to remove?: ")).strip()
134         print()  # Add an extra line for looks
135         return task
136
137
138 # Main Body of Script  ------------------------------------------------------ #
139
140 # Step 1 - When the program starts, Load data from ToDoFile.txt.
141 Processor.read_data_from_file( file_name=file_name_str, list_of_rows=table_lst)  # read
file data
142
143 # Step 2 - Display a menu of choices to the user
144 while (True):
145     # Step 3 Show current data
146     IO.output_current_tasks_in_list(list_of_rows=table_lst)  # Show current data in the
list/table
147     IO.output_menu_tasks()  # Shows menu
148     choice_str = IO.input_menu_choice()  # Get menu option
149
150     # Step 4 - Process user's menu choice
151     if choice_str.strip() == '1':  # Add a new Task
152         task, priority = IO.input_new_task_and_priority()
153         table_lst = Processor.add_data_to_list(task=task, priority=priority,
list_of_rows=table_lst)
154         continue  # to show the menu
155
156     elif choice_str == '2':  # Remove an existing Task
157         task = IO.input_task_to_remove()
158         table_lst = Processor.remove_data_from_list(task=task, list_of_rows=table_lst)
```

```
159         continue  # to show the menu
160
161     elif choice_str == '3':  # Save Data to File
162         table_lst = Processor .write_data_to_file(file_name=file_name_str,
list_of_rows=table_lst)
163         print(" Data Saved!\n")
164         continue  # to show the menu
165
166     elif choice_str == '4':  # Exit Program
167         print(" Goodbye!")
168         break  # by exiting loop
```

The final result of the script in both PyCharm and the OS Command are shown below in Figures 1 and 2, respectively.

Figure 1.Final result of script in PyCharm.

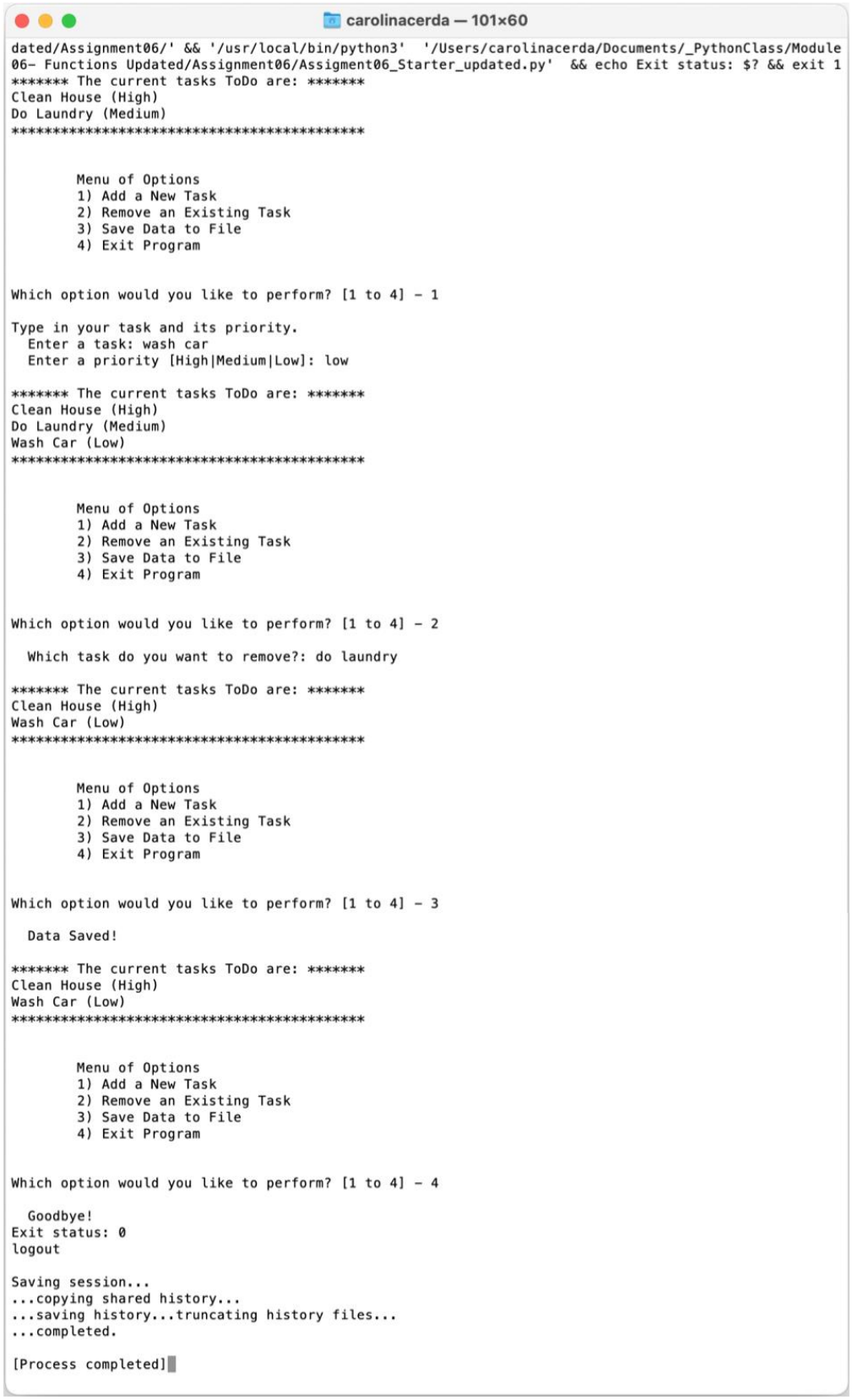

Figure 2. Final result of script in OS Command.

Finally, a verification of the success of the script is done by confirming that the data inputted by the user has successfully been saved within the text file, “ToDoList.txt” (Figure 3).

Figure 3. File used in script for data.

GitHub Pages

In extension from the last assignment’s introduction to the software, GitHub, part of this assignment is to further understand GitHub and create a webpage from a new repository. For this module, a new repository, “IntroToProg-Python-Mod06”, is created (Figure 4).

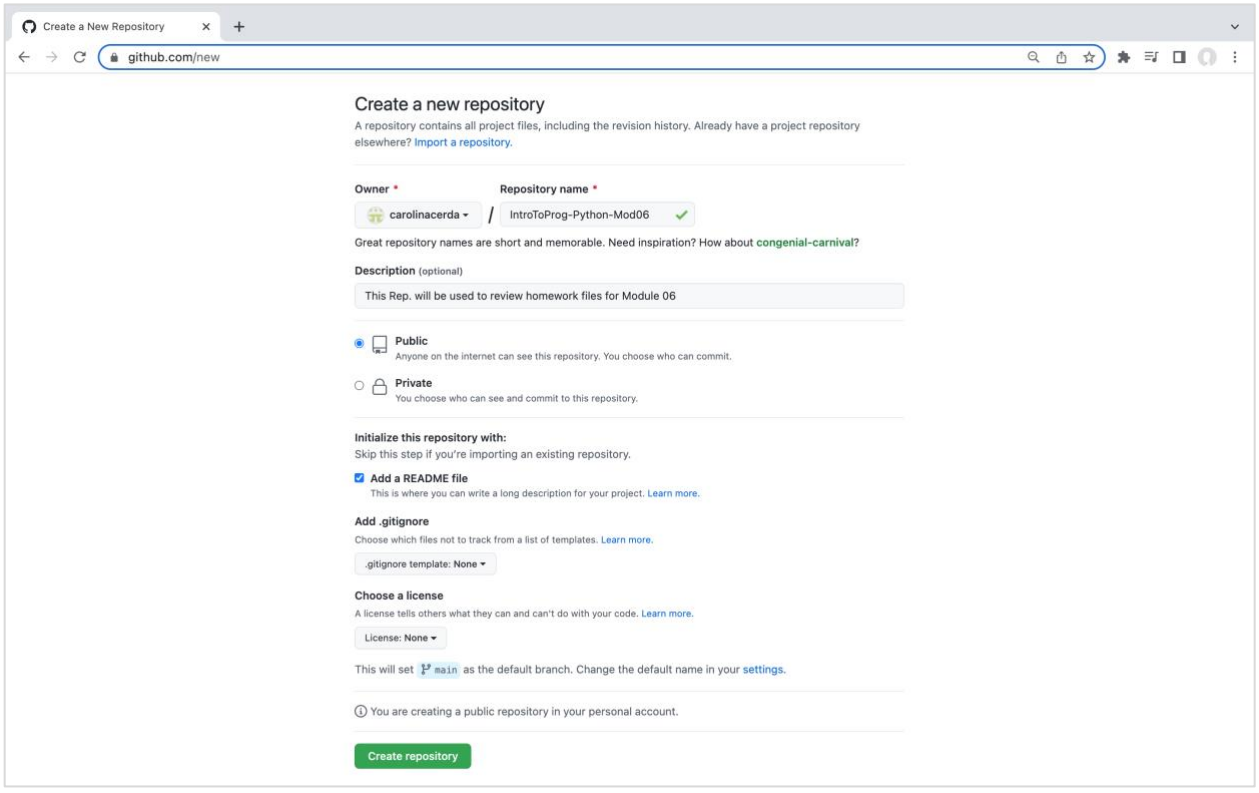

Figure 4. Creating a new repository.

To enhance this repository, a simple webpage will be created using “GitHub Pages.” To create the webpage a new folder must first be created by selecting the “Add file” button in the repository (Figure 5). From there, the name of the file, “docs”, must be written then by following it with a forward slash, “/”, a new folder is automatically added and requires a file to be added in order to create the folder (Figure 6). Thus, the file “index.md” is created using the

starting code provided in the document “Assignment06” provided by Professor Root for Module 06 (Figure 7). To save the folder and the file, “Commit new file” must be selected and then should be ready to for the webpage (Figure 8).

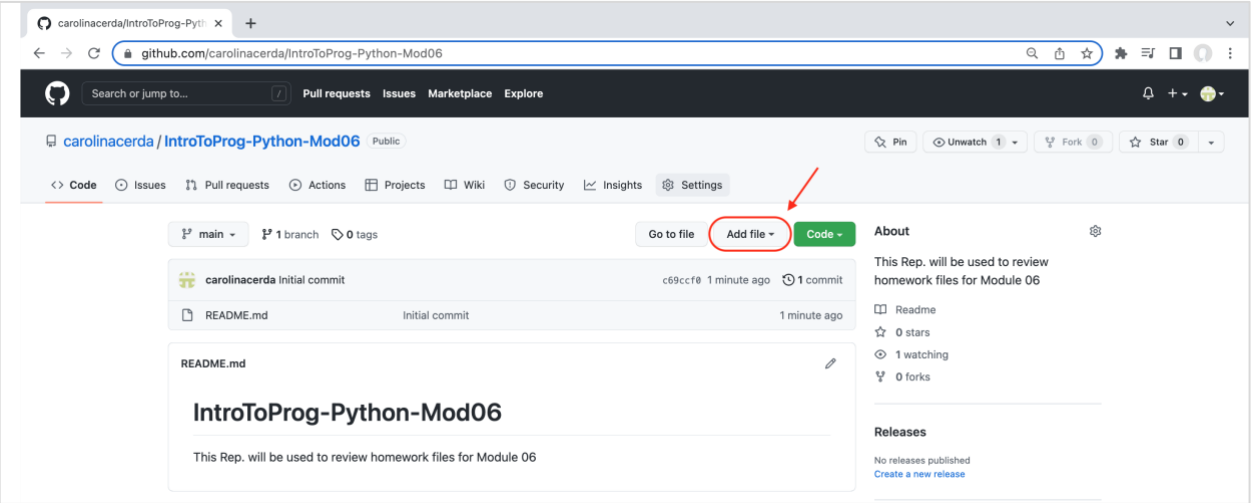

Figure 5. Adding a new folder.

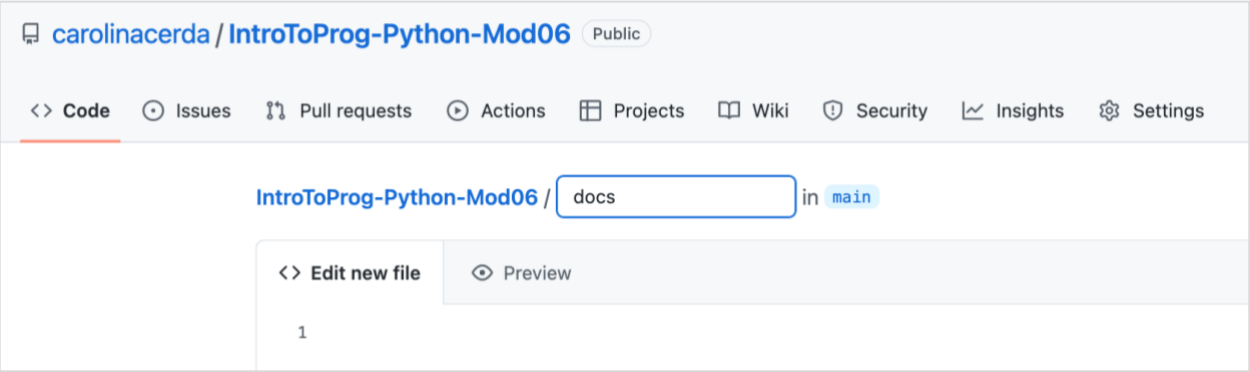

Figure 6. Labelling the folder as "docs".

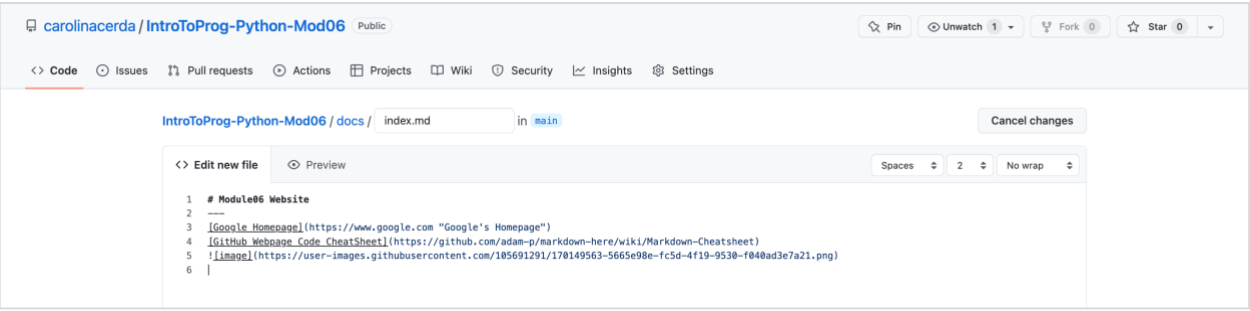

Figure 7. Creating the file "index.md" and adding provided starting code.

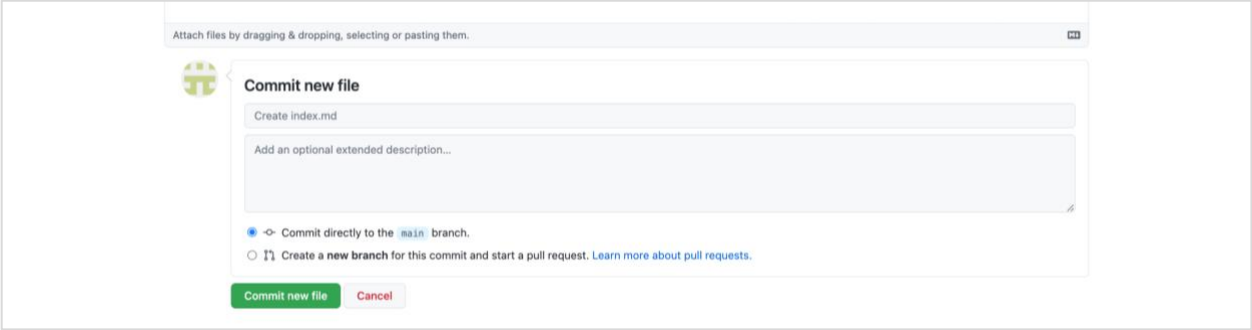

Figure 8. Press "Commit new file" to save file.

The webpage can be created under the “Settings” of the repository as seen in Figure 9. Under “Code and automation” in the “Settings”, select “Pages” (Figure 10).

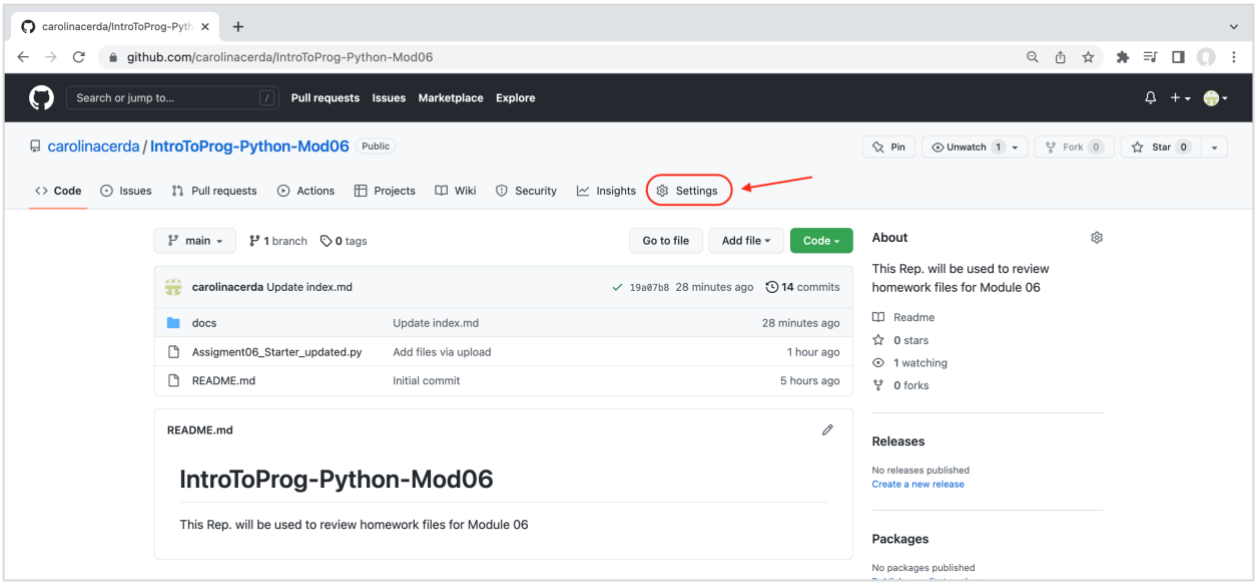

Figure 9. Select the repository "Settings".

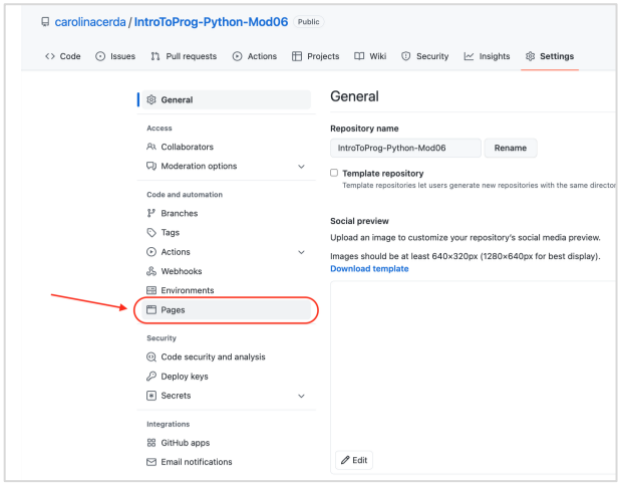

Figure 10. "Pages" tab in the repository "Settings".

In “Pages”, the folder that was created previously can be selected as the source to build up the GitHub webpage. A theme for the webpage can also be selected in “Pages”. The site URL is also found here and also tells you if the site has been published (Figure 11).

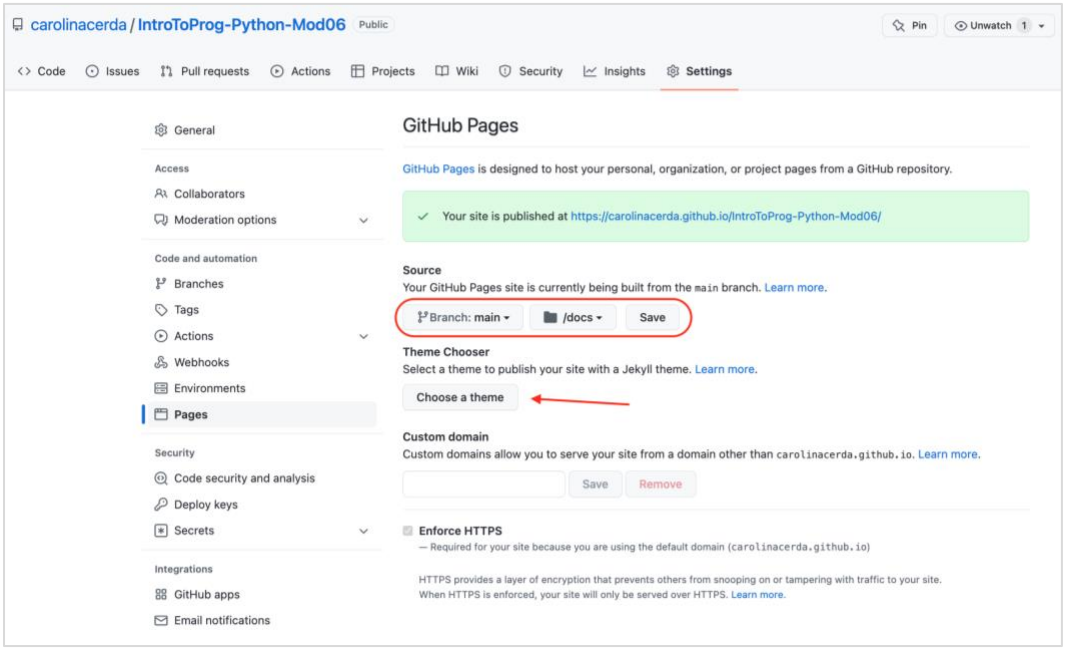

Figure 11. "GitHub Pages" in repository "Settings".

It usually takes a few minutes for the site to publish. Additionally, if any changes made to the “index.md” file or “docs” folder are made it will take a few minutes for the change to be made in effect. The final webpage created for this assignment is shown below in Figure 12.

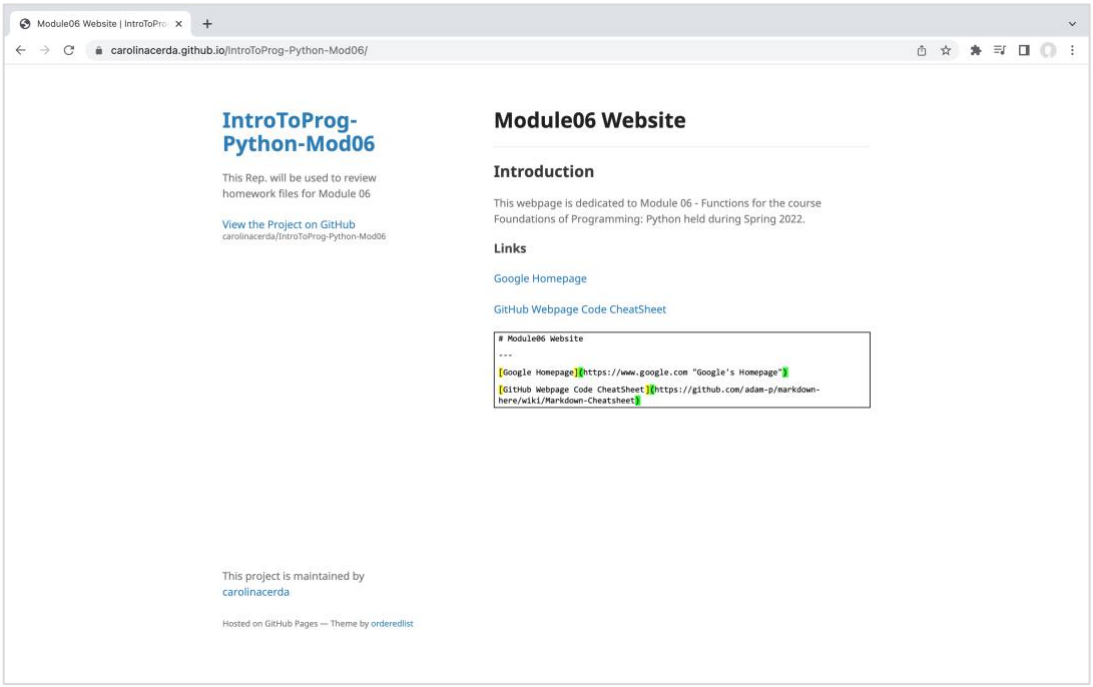

Figure 12. Final GitHub Webpage.

Summary

In this exercise, functions and classes were used in order to create a script that manages an interactive To-Do List. Additionally, GitHub webpages were introduced in order to provide a different way in which files can be presented for professional use as well as to our peers. The purpose of this investigation was to practice using functions in order to better organize and further improve our scripts as well as to become more familiar with GitHub as a tool for file sharing and storage.