Carolina Cerda

June 7, 2022

Foundations of Programming: Python

Assignment 08

https://carolinacerda.github.io/IntroToProg-Python-Mod08/

Creating a Script Using Object-Oriented Programming Techniques

Introduction

In this assignment, classes and their components will be used to create a script that stores data on a product's name and its price. Classes organize functions and data by collecting together instance variables and methods that can define an object type. The following script will be created from provided pseudo-code and will require the use of code seen previously throughout the course with the addition of using classes and understanding their components. Furthermore, GitHub desktop will be downloaded, installed, and used for this investigation.

Creating the Script

Classes, Objects, and Object-Oriented Programming (OOP)

Object-Oriented Programming (OOP) is a pattern within programming that combine attributes, or characteristics, and methods, or behaviors, to create software objects from a definition, called a "class", that defines these attributes and methods. Classes are the design for an object. Thus, there can be multiple objects that are of the same class (M. Dawson, *Python® Programming for the Absolute Beginner, Third Edition*). To define a class, the keyword class is used followed by the name for the class, as can be seen below (Line 6). A docstring is also added to document the class and describe the kind of objects that can be created from the class (Lines 7 – 18). The following sections will describe the class pattern and detail these components that build up classes.

```
1    # Data ------------------------------------------------------------ #
2    strFileName = 'products.txt'
3    lstOfProductObjects = []
4
5
6    class Product:
7        """Stores data about a product:
8
9        properties:
```

Creating a Constructor and Attributes

Constructors, or the __init__ method, are special methods that run when an object is created from the class. To create an object instance from a class, the class's name is used directly as if it were a function (R. Root, "_Mod8PythonProgrammingNotes"). Using the keyword def to define the __init__ method and two attributes, "product_name" and "product_price", are initialized (Line 21). The keyword "self" connects these attributes to the arguments. Attributes are "virtual" fields that hold internal data and are created when a value is assigned to a new variable outside of a method. The attributes are made private to limit direct access to them by the user and is created using two underscore characters that begin the attribute name. The two private attributes for this class are constructed in this manner as seen below (Lines 25 – 26).

A try-except block is created to catch for errors when setting the initial values. The program will first try to run through the attributes but in the case of an error, an except block is created to capture any exception as "e" and, if so, raises the exception and prints to the user a message with the information of the error using the keyword raise (Lines 27 – 28)).

```
20        # -- Constructor –
21        def __init__(self, product_name: str, product_price: float):
22           """ Set name and price of a new object """
23           # -- Attributes –
24           try:
25              self.__product_name = str(product_name)
26              self.__product_price = float(product_price)
27           except Exception as e:
28              raise Exception("Error setting initial values: \n" + str(e))
```

Creating Properties

Properties are functions that manage attribute data. Properties allow restrictions to be imposed in a way in which attributes cannot when the user attempts to input or change the value that the attribute refers to (M. Dawson, *Python® Programming for the Absolute Beginner, Third Edition*). The @property decorator must precede in order to create a property and is defined using the name that is shared with the method, in this case "product_name" and "product_price" (Lines 37 and 45). Then the value is returned with indirect access using a return statement (Lines 34 and 46). The setter attribute of the property provides write access with limits to prevent the user from inputting data that is outside of these limits (Lines 36 – 41, 48 – 53). The decorator @name.setter precedes the method in which the name is equal to the name

of the method (replace "name" with method name). Then the method is then defined and limits can be set using if and else statements to define conditions that must be met or else to raise an exception and return a message regarding the input using the raise keyword (Lines 38-41, 50 – 53).

```
30      # -- Properties –
31      # product_name
32      @property
33      def product_name(self):  # (getter or accessor)
34         return str(self.__product_name)
35
36      @product_name.setter
37      def product_name(self, value: str):  # (setter or mutator)
38         if not str(value).isnumeric():
39            self.__product_name = value
40         else:
41            raise Exception("Product names cannot be numbers.")
42
43      # product_price
44      @property
45      def product_price(self):  # (getter or accessor)
46         return float(self.__product_price)  # cast to float
47
48      @product_price.setter
49      def product_price(self, value: float):  # (setter or mutator)
50         if isinstance(value, float):
51            self.__product_price = float(value)  # cast to float
52         else:
53            raise Exception("Product prices must be in numbers.")
```

Creating Methods

Other functions, besides properties, inside of a class are called methods and allow for the organization of processing statements into named groups (R. Root, "_Mod8PythonProgrammingNotes"). The first method within this class returns the data found in an object instance, from the keyword self, as a string through the __str__ method (Lines 56 – 58). To override this method, a second method defines __str__(self) and converts it to become more useful as the contents of the class' attributes then returns this using a return statement (Lines 60 – 62).

```
55      # -- Methods –
56      def to_string(self):
57         """ alias of __str__(), converts product data to string """
58         return self.__str__()
59
60      def __str__(self):
61         """ Converts product data to string """
62         return self.product_name + "," + str(self.product_price)
63
64      # Data ----------------------------------------------------------------- #
```

Processing Code for Saving and Reading Data from a File Using OOP

        This section creates a class, "FileProcessor", dedicated to processing data to and from the file and list of product objects.

```
67    # Processing  ------------------------------------------------------- #
68    class FileProcessor:
69        """Processes data to and from a file and a list of product objects:
70
71        methods:
72            save_data_to_file(file_name,list_of_product_objects):
73
74            read_data_from_file(file_name): -> (a list of product objects)
75
76        changelog: (When,Who,What)
77            RRoot,1.1.2030,Created Class
78            CCerda,06.07.2022,Modified code to complete assignment 8
79        """
```

Saving Data to the File

        To save data to the file, first a Boolean flag is created so that once the data is successfully saved to the file, the success_status is returned as true (Lines 89, 95, and 99). A try-except block is created to catch any errors that could possibly appear when running the code (Lines 90 and 96). The program will first try to create a variable, "file", that opens a file in write, "w", mode using the open () function (Lines 90 - 91). Then, a for loop is created so that for every product in the list of product objects, by using the write () function, the product will be written to the file and uses the __str__ method to return the class objects as a string representation and is then followed by the creation of a new line (Lines 92 – 93). Lastly, the file is closed use the close() function and success_status is set to True. In the case of an error, an except block is added to capture any exception as "e" meaning that for any exception it is bound to variable "e" (Line 96). Then a message will be printed to the user expressing that there was an error and will also print "e", documentation of the object "e", and class type of the argument(object) (Lines 97 – 98). The success status is then returned using a return statement (Line 99)

```
81        @staticmethod
82        def save_data_to_file(file_name: str, list_of_product_objects: list):
83            """ Write data to a file from a list of product rows
84
85            :param file_name: (string) with name of file
86            :param list_of_product_objects: (list) of product objects data saved to file
87            :return: (bool) with status of success status
88            """
89            success_status = False
90            try:
91                file = open(file_name, "w")
92                for product in list_of_product_objects:
93                    file.write(product.__str__() + "\n")
```

```
94              file.close()
95              success_status = True
96          except Exception as e:
97              print("There was an error!")
98              print(e, e.__doc__, type(e), sep='\n')
99          return success_status
```

Reading Data from the File

To read data from the file into a list of product rows, first, a list, "list_of_product_rows", is created using empty brackets (Line 108). Then, a try-except block is created to catch any error that may arise. The program will first try to create a variable, "file", that opens a file in read, "r", mode using the open () function (Lines 109 – 110). A for loop is then added to find line in file to then split the line by "name" and "value" to create a variable, "row", to instantiate a new Product object using the variables "name" and "value" that were just obtained and the strip() method is used for "value" to remove any excess leading and trailing characters (Lines 111 – 113). The new row is then appended to the list of product rows using the append() function and the file is then closed using the close() function (Lines 114 – 115). In the case of an error, an except block is added to capture any exception as "e" then a message will be printed to the user expressing that there was an error and will also print "e", documentation of the object "e", and class type of the argument(object) (Lines 116 – 118). Finally, the list, "list_of_product_rows "will be returned using a return statement (Line 119).

```
101         @staticmethod
102         def read_data_from_file(file_name: str):
103             """ Reads data from a file into a list of product rows
104
105             :param file_name: (string) with name of file
106             :return: (list) of product rows
107             """
108             list_of_product_rows = []
109             try:
110                 file = open(file_name, "r")
111                 for line in file:
112                     name, value = line.split(",")
113                     row = Product(name, value.strip())
114                     list_of_product_rows.append(row)
115                 file.close()
116             except Exception as e:
117                 print("There was an error!")
118                 print(e, e.__doc__, type(e), sep='\n')
119             return list_of_product_rows
120
121     # Processing ------------------------------------------------------ #
```

Presentation Code for Input and 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 by using a class, "IO".

```
124    # Presentation (Input/Output) ------------------------------------------- #
125    class IO:
126        """ A class for performing Input and Output
127
128       methods:
129          print_menu_items():
130
131          print_current_list_items(list_of_rows):
132
133          input_product_data():
134
135       changelog: (When,Who,What)
136          RRoot,1.1.2030,Created Class:
137          CCerda,06.07.2022,Modified code to complete assignment 8
138       """
```

Outputting Menu Items

Similar to Module 06's assignment, a menu of options is displayed to the user to select from using a print statement with an additional print statement for looks directly following (Lines 143 – 149).

```
140        @staticmethod
141        def print_menu_items():
142            """ Print a menu of choices to the user  """
143            print('''
144            Menu of Options
145            1) Show Current Items
146            2) Add a New Item
147            3) Save Data to File
148            4) Exit Program''')
149            print()  # Add an extra line for looks
```

Inputting Menu Choice

This section of code is identical to the starting code provided by Professor Root for the assignment for Module 06. The only differences are the addition of an extra print() function statement for looks and a colon is used instead of a dash in the print message when asking for user input (Lines 157 – 158).

```
151        @staticmethod
152        def input_menu_choice():
153            """ Gets the menu choice from a user
154
155            :return: string
156            """
157            print()  # Add an extra line for looks
158            choice = str(input("Which option would you like to perform? [1 to 4]: ")).strip()
159            print()  # Add an extra line for looks
160            return choice
```

Outputting Current List Items

        The first choice in the user menu of options is to display to the user the current products and their prices. Thus, an if statement is written for the condition that if the integer length of number of items in "list_of_rows" is greater than 0 (i.e., there is data in the file), then it will first print a "table" with the current names and prices of the products in the formatted order by using a for loop to print out each row in "list_of_rows" (Lines 168 – 174). An else statement follows to cover for if this condition is not met (i.e., there is no data in the file) and will print a message to the user expressing that there are no current items in the list and to choose another option to add data (Lines 175 – 176).

```
162        @staticmethod
163        def print_current_list_items(list_of_rows: list):
164            """ Print the current items in the list of rows
165
166            :param list_of_rows: (list) of rows you want to display
167            """
168            if int(len(list_of_rows)) > 0:
169                print("---- YOUR CURRENT ITEMS: ----")
170                print("PRODUCT  |  PRICE")
171                for row in list_of_rows:
172                    print(row.product_name + "  |  $%.2f" % row.product_price)
173                print("----------------------------")
174                print()  # Add an extra line for looks
175            else:
176                print("  There are no current items in the list. Please choose 'Option 2' to add new products.\n")
```

Inputting New Product Data

        The second choice in the user menu of options is to obtain user input to add data for a new product object. A try-except block is first created to handle any unforeseen errors that may arise from the user input. For the try block, instructions to enter a product and its price are presented to the user followed by an empty print() statement to add an extra line for looks (Lines 183 – 185). Then, two variables are created, "name" and "price", that use the input() function to firstly obtain data from the user about the name and price of the product, respectively, that are then returned as a string through the str() function for "name" and as a float using the float() function for the "price" and also use the strip() method to remove leading and trailing characters, such as extra spaces which is finally followed by another print statement to add an extra line for looks (Lines 186 – 188). The title() method is also used for the input for the name to ensure consistency of the data that is inputted by the user by capitalizing each of the words in the string (Line 186). Another variable "p" is created to instantiate a new Product object using the previous variables for "name" and "price" of the product (Line 189).

    For any other general error, an except block is created to capture any exception as "e" meaning that for any exception it is bound to variable "e" (Line 190). Then a message will be printed from "e" back to the user which will briefly describe the error (Line 191). Lastly, outside of the try-except block, the new Product object, "p", is returned using a return statement (Line 192).

```
177        @staticmethod
178        def input_product_data():
179            """ Gets data for a product object
180
181            :return: (Product) object with input data
182            """
183            try:
184                print("Type in your product and its price.")
185                print() # Add an extra line for looks
186                name = str(input(" Enter the product name: ").title().strip())
187                price = float(input(" Enter the product price: ").strip())
188                print() # Add an extra line for looks
189                p = Product(product_name=name, product_price=price)
190            except Exception as e:
191                print(e)
192            return p
193
194     # Presentation (Input/Output) ------------------------------------------- #
```

Code for the Main Body of the Script

Again, similarly to the assignment for Module 06, the section of code for the main body of the script for this assignment is adapted from the original starter script of Assignment 06. A try-except block is added to ensure that the program will be able to display to the user a readable message for any error that may occur. Thus, the program will first try to load data from the file into a list of product objects by adding elements to the list, "lstOfProductObjects", using the "read_data_from_file" attribute under the "FileProcessor" module through the dot notation to access the data of the file, "strFileName" (Lines 200 – 201). For any general error, an except block is added to capture any exception as "e" meaning that for any exception it is bound to variable "e" (Line 226). Then, a message will be printed to the user expressing that there was an error and will also print "e", documentation of the object "e", and class type of the argument(object) (Lines 227 – 228).

In the absence of an error, the menu of options should be presented to the user followed by the addition of a while loop that, while true, will allow the user to input an option to access a particular option of their choice (Lines 204 – 206). Unlike Assignment 06, outputting the current items in the list is made an option instead of displaying it every time and the code for the option to let the user add data to the list of product objects is different and there is no option to remove data (Lines 207 – 214). Since this script uses object-oriented programming, to add data to the list of product objects, the data obtained from user input from the attribute, "input_product_data()", under the module, "IO", is appended to the list, "lstOfProductObjects" (Line 213). The rest remains similar to the original code from Assignment 06. Lastly, an else statement is added to cover for if the user inputs something that is not a number from 1 to 4 based on choices from the menu of options (Lines 223 – 225).

```
197     # Main Body of Script ---------------------------------------------- #
198
199     # Load data from file into a list of product objects when script starts
```

```
200     try:
201         lstOfProductObjects = FileProcessor.read_data_from_file(strFileName)
202
203         # Show user a menu of options
204         IO.print_menu_items()
205         while True:
206            strChoice = IO.input_menu_choice()  # Get user's  menu option choice
207            if strChoice.strip() == '1':
208               # Show user current data in the list of product objects
209               IO.print_current_list_items(lstOfProductObjects)
210               continue  # to show the menu
211            elif strChoice.strip() == '2':
212               # Let user add data to the list of product objects
213               lstOfProductObjects.append(IO.input_product_data())
214               continue  # to show the menu
215            elif strChoice.strip() == '3':
216               # let user save current data to file and exit program
217               FileProcessor.save_data_to_file(strFileName, lstOfProductObjects)
218               print("  Data saved to file!\n")
219               continue  # to show the menu
220            elif strChoice.strip() == '4':
221               print("  Goodbye!")
222               break  # by exiting loop
223            else:
224               print("  Input is not a number from 1 to 4! Please try again.\n")
225               Continue
226     except Exception as e:
227         print("There was an error! Check file permissions.")
228         print(e, e.__doc__, type(e), sep='\n')
229     # Main Body of Script  -------------------------------------------------- #
```

Running the Script

By building upon the starter script for the assignment for Module 06, as well as the various listing examples provided by Professor Root and from the textbook, *Python® Programming for the Absolute Beginner, Third Edition*, by Michael Dawson, the following completed script is displayed below.

```
1     # Data --------------------------------------------------------------- #
2     strFileName = 'products.txt'
3     lstOfProductObjects = []
4
5
6     class Product:
7        """Stores data about a product:
8
9        properties:
10          product_name: (string) with the product's name
11
12          product_price: (float) with the product's standard price
13        methods:
```

```python
            to_string() returns comma separated product data (alias for __str__())
        changelog: (When,Who,What)
            RRoot,1.1.2030,Created Class
            CCerda,06.07.2022,Modified code to complete assignment 8
        """

        # -- Constructor --
        def __init__(self, product_name: str, product_price: float):
            """ Set name and price of a new object """
            # -- Attributes --
            try:
                self.__product_name = str(product_name)
                self.__product_price = float(product_price)
            except Exception as e:
                raise Exception("Error setting initial values: \n" + str(e))

        # -- Properties --
        # product_name
        @property
        def product_name(self):   # (getter or accessor)
            return str(self.__product_name)

        @product_name.setter
        def product_name(self, value: str):   # (setter or mutator)
            if not str(value).isnumeric():
                self.__product_name = value
            else:
                raise Exception("Product names cannot be numbers.")

        # product_price
        @property
        def product_price(self):  # (getter or accessor)
            return float(self.__product_price)  # cast to float

        @product_price.setter
        def product_price(self, value: float):  # (setter or mutator)
            if isinstance(value, float):
                self.__product_price = float(value)  # cast to float
            else:
                raise Exception("Product prices must be in numbers.")

        # -- Methods --
        def to_string(self):
            """ alias of __str__(), converts product data to string """
            return self.__str__()

        def __str__(self):
            """ Converts product data to string """
            return self.product_name + "," + str(self.product_price)
    # Data ------------------------------------------------------------- #


    # Processing  ------------------------------------------------------ #
```

```python
 67    class FileProcessor:
 68        """Processes data to and from a file and a list of product objects:
 69
 70        methods:
 71            save_data_to_file(file_name,list_of_product_objects):
 72
 73            read_data_from_file(file_name): -> (a list of product objects)
 74
 75        changelog: (When,Who,What)
 76            RRoot,1.1.2030,Created Class
 77            CCerda,06.07.2022,Modified code to complete assignment 8
 78        """
 79
 80        @staticmethod
 81        def save_data_to_file(file_name: str, list_of_product_objects: list):
 82            """ Write data to a file from a list of product rows
 83
 84            :param file_name: (string) with name of file
 85            :param list_of_product_objects: (list) of product objects data saved to file
 86            :return: (bool) with status of success status
 87            """
 88            success_status = False
 89            try:
 90                file = open(file_name, "w")
 91                for product in list_of_product_objects:
 92                    file.write(product.__str__() + "\n")
 93                file.close()
 94                success_status = True
 95            except Exception as e:
 96                print("There was an error!")
 97                print(e, e.__doc__, type(e), sep='\n')
 98            return success_status
 99
100        @staticmethod
101        def read_data_from_file(file_name: str):
102            """ Reads data from a file into a list of product rows
103
104            :param file_name: (string) with name of file
105            :return: (list) of product rows
106            """
107            list_of_product_rows = []
108            try:
109                file = open(file_name, "r")
110                for line in file:
111                    name, value = line.split(",")
112                    row = Product(name, value.strip())
113                    list_of_product_rows.append(row)
114                file.close()
115            except Exception as e:
116                print("There was an error!")
117                print(e, e.__doc__, type(e), sep='\n')
118            return list_of_product_rows
119
```

```python
120      # Processing  ----------------------------------------------------------- #
121
122
123      # Presentation (Input/Output)  -------------------------------------------- #
124      class IO:
125          """  A class for performing Input and Output
126
127          methods:
128              print_menu_items():
129
130              print_current_list_items(list_of_rows):
131
132              input_product_data():
133
134          changelog: (When,Who,What)
135              RRoot,1.1.2030,Created Class:
136              CCerda,06.07.2022,Modified code to complete assignment 8
137          """
138
139          @staticmethod
140          def print_menu_items():
141              """ Print a menu of choices to the user  """
142              print('''
143          Menu of Options
144          1) Show Current Items
145          2) Add a New Item
146          3) Save Data to File
147          4) Exit Program''')
148              print()  # Add an extra line for looks
149
150          @staticmethod
151          def input_menu_choice():
152              """ Gets the menu choice from a user
153
154              :return: string
155              """
156              print()  # Add an extra line for looks
157              choice = str(input("Which option would you like to perform? [1 to 4]: ")).strip()
158              print()  # Add an extra line for looks
159              return choice
160
161          @staticmethod
162          def print_current_list_items(list_of_rows: list):
163              """ Print the current items in the list of rows
164
165              :param list_of_rows: (list) of rows you want to display
166              """
167              if int(len(list_of_rows)) > 0:
168                  print("---- YOUR CURRENT ITEMS: ----")
169                  print("PRODUCT  |  PRICE")
170                  for row in list_of_rows:
171                      print(row.product_name + "  |  $%.2f" % row.product_price)
172                  print("----------------------------")
```

```
173              print()  # Add an extra line for looks
174          else:
175              print("  There are no current items in the list. Please choose 'Option 2' to add new products.\n")
176
177      @staticmethod
178      def input_product_data():
179          """ Gets data for a product object
180
181          :return: (Product) object with input data
182          """
183          try:
184              print("Type in your product and its price.")
185              print()  # Add an extra line for looks
186              name = str(input("  Enter the product name: ").title().strip())
187              price = float(input("  Enter the product price: ").strip())
188              print()  # Add an extra line for looks
189              p = Product(product_name=name, product_price=price)
190          except Exception as e:
191              print(e)
192          return p
193
194      # Presentation (Input/Output)  ------------------------------------------- #
195
196
197  # Main Body of Script  --------------------------------------------------- #
198
199  # Load data from file into a list of product objects when script starts
200  try:
201      lstOfProductObjects = FileProcessor.read_data_from_file(strFileName)
202
203      # Show user a menu of options
204      IO.print_menu_items()
205      while True:
206          strChoice = IO.input_menu_choice()  # Get user's  menu option choice
207          if strChoice.strip() == '1':
208              # Show user current data in the list of product objects
209              IO.print_current_list_items(lstOfProductObjects)
210              continue  # to show the menu
211          elif strChoice.strip() == '2':
212              # Let user add data to the list of product objects
213              lstOfProductObjects.append(IO.input_product_data())
214              continue  # to show the menu
215          elif strChoice.strip() == '3':
216              # let user save current data to file and exit program
217              FileProcessor.save_data_to_file(strFileName, lstOfProductObjects)
218              print("  Data saved to file!\n")
219              continue  # to show the menu
220          elif strChoice.strip() == '4':
221              print("  Goodbye!")
222              break  # by exiting loop
223          else:
224              print("  Input is not a number from 1 to 4! Please try again.\n")
225              Continue
```

```
226      except Exception as e:
227          print("There was an error! Check file permissions.")
228          print(e, e.__doc__, type(e), sep='\n')
229      # Main Body of Script  -------------------------------------------- #
230
```

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

```
● ● ●                        Run - Assignment08
Run:    🌿 Assignment08 ×                                              ▯ ✿ —
 ▶  ↑    "/Users/carolinacerda/Documents/_PythonClass/Module08 - Classes and Objects Updated/Assignment08/venv/bin/python"
 🔧  ↓       "/Users/carolinacerda/Documents/_PythonClass/Module08 - Classes and Objects Updated/Assignment08/Assignment08.py"
 ■  ⇥
 ▬  ⊵            Menu of Options
 ▬  🖨            1) Show Current Items
 📌  🗑            2) Add a New Item
                 3) Save Data to File
                 4) Exit Program


        Which option would you like to perform? [1 to 4]: 1

          There are no current items in the list. Please choose 'Option 2' to add new products.


        Which option would you like to perform? [1 to 4]: 2

        Type in your product and its price.

          Enter the product name: cup
          Enter the product price: 1.99


        Which option would you like to perform? [1 to 4]: 2

        Type in your product and its price.

          Enter the product name: plate
          Enter the product price: 4.99


        Which option would you like to perform? [1 to 4]: 1

        ---- YOUR CURRENT ITEMS: ----
        PRODUCT | PRICE
        Cup | $1.99
        Plate | $4.99
        ----------------------------


        Which option would you like to perform? [1 to 4]: 3

          Data saved to file!


        Which option would you like to perform? [1 to 4]: 4

          Goodbye!

        Process finished with exit code 0
```

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, "products.txt" (Figure 3).
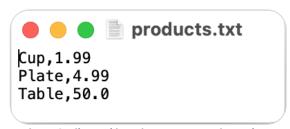

Figure 3. File used in script to save user input data.

Summary

For this exercise, classes and their components were used to create a script that builds upon code seen previously throughout this course to manage a program that displays to the user a menu of options, outputs the current items in a list from a text file, allows the user to input new data, provides the option to save the data to a file, and then exit the program. Finally, GitHub Desktop was used to introduce a way to use GitHub directly from the computer as an application instead of from a browser or command line.