Programming Abstractions – Homework IV

Dr. Ritwik Banerjee
Computer Science, Stony Brook University

This homework document consists of 4 pages, and involves simple object-oriented programming in Python, with the use of optional and mandatory parameters.

Development Environment: This is a Python programming assignment, and it is highly recommended that you use PyCharm for your code. You can, if you really want, use different IDEs, but if things go wrong there, you may be on your own. Plus, IDEA and PyCharm have very similar user interfaces, so using one of them makes it very intuitive to use the other.

Programming Language: Your Python installation must be version 3.6 or above. Make sure this is reflected in your PyCharm project settings as well. In particular, DO NOT use Python 2.x (this is "legacy", and will never be used in any current or future software).

In this assignment, you will write a few Python functions, which require user interaction and processing strings and lists. The functions are described next.

1. **The `get_number` function.** (20)

   Parameters: `string` — a string to be shown to the user to "prompt" them.

   Returns: a float.

   Description: This function will accept one string argument and show it to the user. It will then get input from the user and convert it into a float, and return the float value. If the user does not enter a valid number, this function will tell them that an invalid number was entered by printing "Invalid input. Try to provide a valid number.", and show the prompt again. In this way, the function will keep allowing the user to provide a valid number repeatedly until the user enters a value that can be successfully converted to a float.

   Example test cases:

   - `get_number("Please enter a number:")` ⟶

     ```
     Please enter a number: 10
     ```

     – Since the user entered a valid number, the function returns the float value provided by the user. In this case, it thus returns 10.0.

   - `get_number("Provide a number:")` ⟶

     ```
     Provide a number: hi
     Invalid input. Try to provide a valid number.
     Provide a number: 1e-5
     Invalid input. Try to provide a valid number.
     Provide a number: 15
     ```

     – Now, the function returns 15.0.

2. **The `get_operator` function.** (20)

   Parameters: `string` – a string to be shown to the user to "prompt" them.

   Returns: a string with one of four values: "+", "-", "*", or "/".

   Description: This function will accept one string argument to use as a prompt and show it to the user. It will then get input from the user. If the user enters a single valid symbol from the set $\{+, -, *, /\}$, the function will return it. If the user enters anything else, it will issue: “You may only enter one of the following operators: + − ∗ /”. The function repeats this behavior until the user enters one of the four acceptable operator symbols.

Example test cases:

- `get_operator("Enter an orithmetic operator:")` ⟶

  ```
  Enter an orithmetic operator: *
  ```

  – Since the user entered a valid operator, the function returns the string representing this operator. In this case, it thus returns “*”.
- `get_operator("Enter an orithmetic operator:")` ⟶

  ```
  Enter an orithmetic operator: add
  You may only enter one of the following operators: + - * /
  Enter an orithmetic operator: plus
  You may only enter one of the following operators: + - * /
  Enter an orithmetic operator: +
  ```

  – Now, the function returns “+”.

3. **The `halt` function.** (20)

   Parameters: `string` — a string to be shown to the user as a prompt.

   Returns: `boolean` — `True` if the user enters Y, y, YES, Yes, or yes; `False` if the user enters N, n, NO, No, or no.

   Description: This function will accept one string as its argument to prompt the user, and display that string to get an input from the user. If the user enter “Y”, “y”, “YES”, “Yes”, or “yes”, then the function returns `True`. If the user enters “N”, “n”, “NO”, “No”, or “no”, the function returns `False`. If the user enters anything else, the function issues “Invalid response. Please enter [Y|N].” This behavior is repeated until the user provides a valid response.

   Example test cases:

   - `halt("Would you like to continue?"`

     ```
     Would you like to continue? Not sure.
     Invalid response. Please enter [Y|N].
     Would you like to continue? Hold on!
     Invalid response. Please enter [Y|N].
     Would you like to continue? N
     ```

     – Now, the function returns `False`.

4. **The `calculate` function.** (20)

   Parameters: none

   Returns: `float` -– the result of a binary operator calculation.

   Description: This function will get an integer, an operation, and a second integer from the user. Once it obtains these three items (it may call other functions you have written in this assignment, but no external functions), it will perform the required operation and return the calculated value. This function will properly deal with invalid inputs from the user, which can clearly be done utilizing your earlier functions. The final result this function returns will be a float, even if it is an integral value.

   Example test case:

   - `calculate()`

     ```
     Enter the first number: 13
     Enter the operator: -
     Enter the second number: 2
     ```

     – Returns 11.0

Other example test cases are not separately provided for this function, since they can be inferred easily from your earlier functions.

5. **The `Calculator` class and its `run` function.** (20)

   Parameters: none

   Returns: none

   Description: The `run` function will use your other functions to let the user perform as many calculations as they wish. This function must be able to handle invalid inputs (which can be easily done by using your earlier functions). After each calculation, `run` will ask the user if they want to continue. It will also save the results of all the calculations done by the user in a single run of this program. When the user decides to no longer continue running this program, it will print a final message stating the total number of calculations performed, and list their results (shown in the example test case below). Then, it will issue a goodbye message and quit.

   Note that the three prompts in the `calculate` method were not provided as arguments to the method. These prompts must be provided as arguments to the `__init__` of the `Calculator` class. Moreover, these prompts must be private instance attributes. Finally, the goodbye message at the end of `run` must be an optional private instance attribute. That is, the user may or may not provide this message as an argument to `__init__`. If no such message is provided to the `Calculator` instance, the run method quits without any goodbye message.

   You may choose to have other additional optional parameters for your `__init__`, if you feel the need for it. But remember that the user should not be forced to provide any parameter to create an instance of a calculator!

   Example test case:

   - `run()`

```
Enter the first number: 3
Enter the operator: -
Enter the second number: 5
3 - 5 = -2.0
Would you like to continue? yes
Enter the first number: 100
Enter the operator: *
Enter the second number: 2
100 * 2 = 200.0
Would you like to continue? sure
Invalid response. Please enter [Y|N].
Would you like to continue? N
You carried out 2 calculations. The results were: -2.0; 200.0
Bye!
```

Rubric

`get_number`

- Handles negative numbers (e.g., "-4") → 2
- Handles leading and/or trailing spaces (e.g., " 3.25" or " 3.25 ") → 3
- Displays the proper prompt → 5
- Returns the proper value → 5
- Handles invalid inputs → 5

`get_operator`

- Displays the proper prompt → 4
- Handles leading and/or trailing spaces → 6
- Handles invalid inputs → 10

`halt`

- Displays the proper prompt → 3
- Handles leading and/or trailing spaces → 6
- Handles all the valid variations of yes/no inputs → 6
- Handles invalid inputs → 5

`calculate`

- Performs correct calculations → 10
- Accepts three valid inputs for each calculation → 5
- Handles invalid inputs for operators and/or numbers → 4
- Returns the correct value as float → 1

`run()` and the `Calcuator` class

- Allows for multiple calculations → 5
- Properly allows the user to quit the program's run → 5
- Prints correct messages along with the correct stored results when quitting → 5
- Allows the user to instantiate a calculator using optional parameters → 5

- You can import modules in your code as long as they satisfy both these conditions: (i) they are a part of core Python, and (ii) they do not provide an implementation of any kind of user-interaction or prompt-interaction beyond reading/writing single line inputs and outputs.
- **What to submit?**
  Your solution in a single file named `calculator.py` (you may not submit any additional files in this assignment).

Submission Deadline: May 8 (Monday), 11:59 pm