CCO1100-01 Computer Programming

Spring 2023

Lab 9

Bernd Burgstaller

Yonsei University

Overview

- Programming Problems

- Deliverables, Due Date and Submission

- Notes:

  1) Please consider using our **Python Tutor** when debugging your code.
     - http://pythontutor.elc.cs.yonsei.ac.kr
  2) Some lab problems can be tested with Hyeongjae Python.
     - Please refer to the last page of this assignment for details.

Programming Problems

Problem 1: Use the stack module from the textbook (Chapter 7.6.3, pp. 264—266, **provided** with the lab spec on LearnUs) and write a program that asks the user to enter a series of parentheses and/or braces, then indicates whether or not they are properly nested.

Example 1:

```
    Enter parentheses and/or braces: (){}{()})
    Nested properly.
```

Example 2:

```
    Enter parentheses and/or braces: ((()}
    Not properly nested.
```

Hint 1: As the program processes characters, have it push each left parenthesis or left brace. When it reads a right parenthesis or right brace, have it pop the stack and check that the popped item is a matching parenthesis or brace. (If not, the parentheses/braces are **not** nested properly.) After processing the last character from the user input, check whether the stack is empty; if so, the parentheses/braces are matched. If the stack is not empty (or if a stack underflow occurs), the parentheses/braces are not matched.

Hint 2: you can assume that the user enters nothing except parentheses and braces.

Hint 3: You are required to use the stack module as given in the text book (not using the stack module is forbidden and will be detected during automated grading). Please pay utmost attention to type in the textbook's stack module correctly. If your code relies on a stack-module with mis-spelled function names or other deviations, it will fail during automated grading. **Please do not hand in the stack module.**

Problem 2: Some calculators use a system of writing mathematical expressions in so-called reverse Polish notation (RPN). In this notation, operators are placed after their operands instead of between their operands. For example, 1 + 2 would be written 1 2 + in RPN, and 1 + 2 * 3 would be written as 1 2 3 * +. RPN expressions can be easily evaluated using a stack (**please use the one provided on LearnUs**). The algorithm involves reading the operators and operands in an expression from left to right, performing the following actions:

- ❑ When an **operand** is encountered, push it onto the stack.
- ❑ When an **operator** is encountered, pop its operands from the stack, perform the operation on those operands, and then push the result onto the stack.

Write a program that evaluates RPN expressions. The input operands will be integers. You can assume at least one blank between each pair of operators and/or operands. The operators are +, -, *, /, and =. The = operator pops the top-most stack item and displays it. Afterwards, the user is prompted to enter another expression. The process continues until the user enters a character that is not an operator and not an operand:

```
Enter an RPN expression: 1 2 3 * + =
Value of expression: 7
Enter an RPN expression: 5 8 * 4 9 - / =
Value of expression: -8
Enter an RPN expression: 3 12 - 9 10 / * =
Value of expression: -8.10
Enter an RPN expression: q
```

Hints:

You are allowed to use the `split()` method to process the user input.

In case of malformed expressions, your calculator should output ``**`Evaluation error`**" and terminate the program. Stack underflow because of an insufficient number of operands makes an expression malformed. A non-empty stack after having popped the operand of the "=" operation indicates a malformed expression. An expression must contain exactly one "=" operator, which must occur in the last (right-most) position.

In the following you find several examples of malformed expressions.

```
1 2 + * =      # stack underflow
+ =            # stack underflow
=              # stack underflow
1 5            # "=" not in right-most position
1 = 2 2 +      # "=" not in right-most position
1 2 3 + =      # stack not empty after "="
```

How to detect malformed expressions: it is a good strategy to check errors (1) before and/or (2) after performing an operation.

For example, if your calculator encounters the "*" operator, it needs to pop 2 operands from the stack. If popping fails because of an empty stack, the calculator has encountered a stack underflow.

If the stack is not empty after popping the element that should be displayed for the "=" operator, the expression is malformed, too. See the above examples for more checks that your calculator must conduct to catch all malformed expressions.

Expressions may contain division operators and hence result in floating-point numbers. Therefore, convert all numbers from the user-input to float and have your calculator calculate with those floating-point numbers.

An expression that contains a division by zero counts as a malformed expression. Your calculator must not attempt the division by zero (and crash), but rather check the division operands in advance and announce a malformed expression.

Floating-point results shall be displayed with two digits after the fractional point.

Integer results should be displayed without a fractional point. For example: ``5.10" (float) and ``5" (integer).

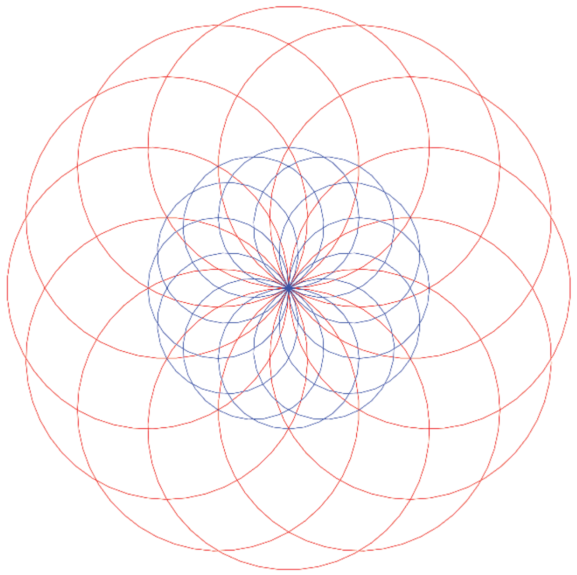
Use `float.is_integer(some_number)` to find out whether `some_number` is an integer.

To be continued on the following page.

Problem 3: Consider the palindrome checker program from Section 7.3.7 (page 267 and following). Rewrite this program to remove the stack. Strictly speaking, no stack nor any other auxiliary data-structure is needed to solve this problem. However, it is fully up to you how you solve this problem (that means, additional data-structures other than stacks will be accepted as a solution, too).

Problem 4: Use the below function template and complete it such that it draws a flower composed of circles as shown.

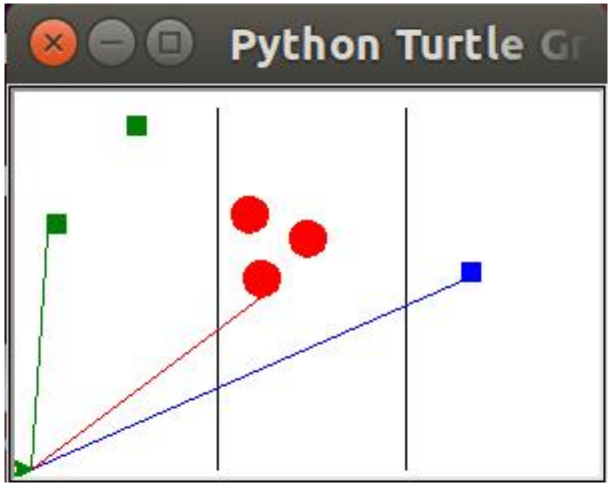
```
def drawFlower(myturtle, r):
  """ Draws a flower composed of 24 circles
      on the screen: 12 red circles and 12
      blue circles. The radius of the red
      circles is ``r‘’. The radius of the
      blue circles is half of ‘’r’’.
      The turtle ``myturtle'' is already
      positioned in the center of the flower.
  """

  # Your code here:
  ...
# Nothing else.
```



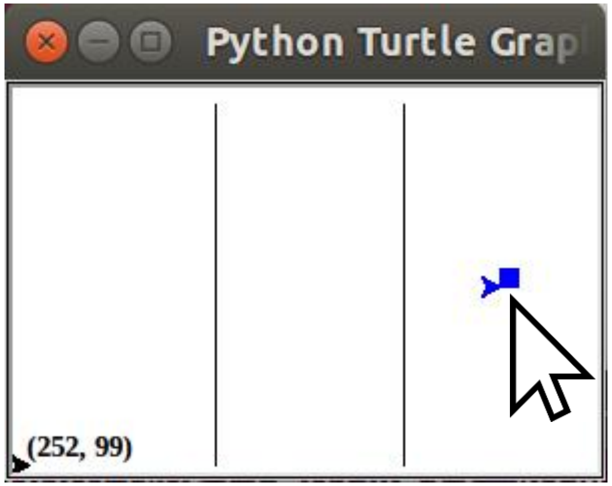
Inside your function, please hide the turtle before starting to draw. Prior to returning from your function, you do not have to unhide the turtle.

Problem 5: Download and study program CallbackFunction.py from the Lab9 folder on LearnUs. The program sets up three sections on the screen, divided by 2 vertical bars ``|". Two handlers for user key-presses (functions **`writeText`** and **`closeDown`**) and a handler for mouse-clicks (function **`drawShape`**) are installed. Click on different sections on the window to see what happens in the mouse-click handler. Press the relevant keys to see what happens in the handlers for user key-presses.

Your task is to modify its keyboard handler such that at any time the user presses the "Up" key, the coordinate of the turtle will be displayed at coordinate (0,0) at the lower-left corner of the screen (see the below example, which also includes the user's mouse pointer). Coordinates must be displayed as integers. See below-right for how your program should work.



Version from LearnUs



(252, 99)

Your modified version

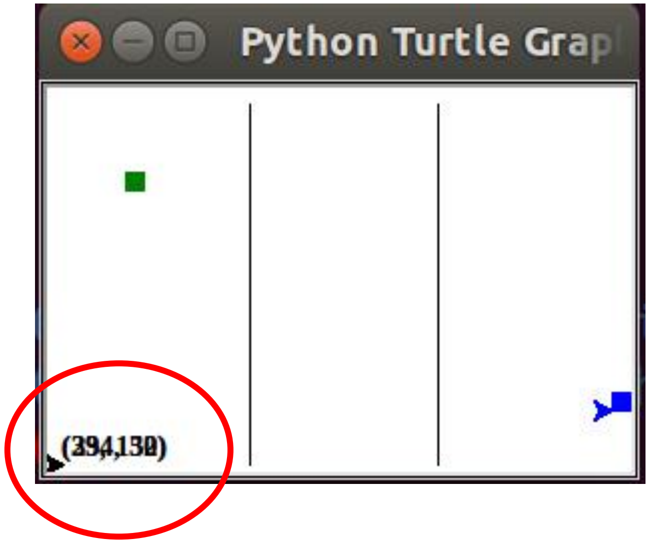
Hint: Writing something on the screen can be accomplished with a turtle's `write()` method.

For example:

```
myturtle.write('Hello!', font = ('Times New Roman', 8, 'bold'))
```

Therein, the named "font" parameter specifies the font, followed by the font-size (8) and bold-face ('bold').

As shown in the below example, subsequent write commands mix with previous text. Before issuing a write-command, you must therefore erase the previous text. But you are not allowed to delete other screen content. One way to achieve this is to create **a second turtle**, which is used exclusively for text-output. Use your ``text'' turtle's `clear()` method to remove the previous text before outputting new text.



Erase previous text before new text output!

Problem 6: Textbook page 245, Problem M5.

Please download and unzip the zip-archive with the source-code from the Lab9 folder on LearnUs. Please follow the description of the horse-racing program in the **provided slides** and the **textbook excerpt for Problem 6 (excerpt_lab9_p6.pdf)**. You can find the different versions of the program on LearnUs (please consider and run them to see how the program evolves.)

Note: you must import the provided source-file (**HorseRace_Final.py**) and the folder with the image files (**img**) into your PyCharm project.

Your program should ask the user whether to run another race. After the user presses 'q', your program shall display the cumulative wins of all horses, close the graphics window and terminate. Use ``**`turtle.bye()`**" to close the graphics window.

```
<race simulation 1>
Press 'q' to quit: a
<race simulation 2>
Press 'q' to quit: q
Horse 1 won 0 times
Horse 2 won 1 times
...
Horse 10 won 1 times
```

- Please do not close the Turtle window between simulations.
- Instead, you must clear the contents of the Turtle window before starting the next simulation.
- For an illustration, please refer to our instruction video titled ``lab9_p6_example.mp4" provided with the video material of week 9.

Continued on the next page...

- Please hand in only the modified Python code
  - File lab9_p6.py
- Do not hand in any gif files, etc.
- Do not change the names of gif files in your program, otherwise your program will fail automated grading.

Marking Criteria

- Score is only given to programs that compile and produce the correct output with Python version 3.
- Points are deducted for programs that are named wrongly. See the list of deliverables for the required file names*.
- Points are deducted for programs that produce warnings.
- Points are deducted for insufficient comments*.
- Points are deducted for missing docstrings*.
- Please pay particular attention to the **requested output format** of your programs. Deviating from the requested output format results in points deductions.

*****

*For the details of **those** marking criteria (superscripted by *****) please refer to the **most recent** lab specification that elaborated on that criterion (some criteria evolve over the course of the semester).*

Plagiarism

- Plagiarism (Cheating)
  - This is an individual assignment. All submissions are checked for plagiarism.
  - Once detected, measures will be taken for **all** students involved in the plagiarism incident (including the ``source'' of the plagiarized code).

Deliverables, Due Date and Submission

- Please prepare the files for the programming problems. The names of the files, their due dates and the archive file-name is given in the below table.
  - Please upload your archive file by the stated due date on LearnUs.
- Lab problems marked as '✓' can be tested on our Hyeongjae Python site http://hyeongjaepython.elc.cs.yonsei.ac.kr/

Problem	File name	Due	Archive name	Hyeongjae Python
1	lab9_p1.py	Wednesday May 10, 2023 23:00	lab9_<student id>.zip	—
2	lab9_p2.py			—
3	lab9_p3.py			—
4	lab9_p4.py			—
5	lab9_p5.py			—
6	lab9_p6.py			—