For this lab, you will write a simple pocket calculator program. The program will be able to display the current value on the screen of the calculator. You will store the current value of the calculator in a variable. The initial value of the calculator (i.e., the initial current value) is 0. Do not write out the entire lab assignment and only then try to debug it: this almost never works. If you're new to programming, you shouldn't, as a rule, write more than five lines of code between trying the new code out to see if it does something sensible. You will get credit for the lab if you make reasonable progress toward completing it. Credit may be given for programs that accomplish only some of the tasks assigned. The TAs are here to help you. If you are stuck, ask for help!

Question 1. Warmup

Start with the following code in Pyzo:

```
def my_sqrt(x):
    sqr = x**.5
    return sqr

if __name__ == "__main__":
    res = my_sqrt(25)
```

Part (a)

When you run this code, nothing is output to the screen. Explain why. Now, add a line of code inside the `main` block so that the result of the computation in `my_sqrt` is printed.

Part (b)

If you add the line `print(sqr)` inside the `main` block, running the program produces an error. Explain why.

How can you modify the function `my_sqrt` so that `print(sqr)` doesn't produce an error?

Part (c)

Write a function with the signature `my_print_square(x)` which **prints** (rather than returns) the square of the argument `x`. Call this function from the `main` block, and explain the difference between the effect of calling `my_print_square(x)` and calling `my_sqrt(x)`.

What is the output for the following code, if it is run from within the main block? Explain.

```
res = my_print_square(25)
print(res)
```

Question 2. Tracing

Trace your function line-by-line using Pyzo. Demonstrate to your TA that you can step to the line where `sqr` is set, and can then access its value. Show that outside of the function `my_sqrt`, you cannot access `res` when tracing.

Videos showing how to trace are on the course website under Week 2.

Note: tracing is an important skill that you need to learn now. **You cannot skip this part**.

Question 3. Welcome Message

In the `if __name__ == "__main__"` block, write code that displays the following:

```
Welcome to the calculator program.
Current value: 0
```

Question 4. Displaying the Current Value

Write a function whose signature is `display_current_value()`, and which displays the current value in the calculator. In the `if __name__ == "__main__"` block, test this function by calling it and observing the output.

Question 5. Addition

Write a function whose signature is `add(to_add)`, and which adds `to_add` to the current value in the calculator, and modifies the current value accordingly. In the `if __name__ == "__main__'"` block, test the function `add` by calling it, as well as by calling `display_current_value()`. **Hint**: when modifying global variables from within functions, remember to declare them as `global`.

Question 6. Multiplication

Write a function whose signature is `mult(to_mult)`, and which multiplies the current value in the calculator by `to_mult`, and modifies the current value accordingly. In the `if __name__ == "__main__"` block, test the function.

Question 7. Division

Write a function whose signature is `div(to_div)`, and which divides the current value in the calculator by `to_div`, and modifies the current value accordingly. In the `if __name__ == "__main__"` block, test the function. What values of `to_div` might cause problems? Try them to see what happens.

Question 8. Memory and Recall

Pocket calculators usually have a memory and a recall button. The memory button saves the current value and the recall button restores the saved value. Implement this functionality.

Question 9. Undo

Implement a function that simulates the Undo button: the function restores the previous value that appeared on the screen before the current one.

Pressing the Undo twice restores the original value:

```
# current value: 25
add(5) # current value: 30
mult(2) # current value: 60
```

```
undo()  # current value: 30
undo()  # current value: 60
undo()  # current value: 30
```