# Section 3: Variable Mechanics

## Debugging and tracing

In lecture, we discussed how to use variables maintain and update information in our program, and how to combine that ability with loops to write sophisticated programs. Using variables appropriately in loops can be tricky, and it's worth making sure you can trace through how they work and how you can resolve bugs in your program.

For example, take a look at the program below, which is supposed to compute the sum of all the numbers from 0 to 99:

```python

def main():
    total = 0
    for i in range(100):
        new_total = total + i
    print("The sum of the first 100 numbers is " + str(new_total))

if __name__ == "__main__":
    main()
    
```

Unfortunately, there is a bug in the program that results in the wrong sum being printed out due to an error in variable usage. Trace through the execution of the program to determine what might have gone wrong, and how you might fix it.

## Running Total

Write a program that asks a user to continuously enter numbers and print out the running total, the sum of all the numbers so far. Once you get the program working, see if you can modify it so that the program stops when the user enters a 0.

```
Enter a value: 7
Running total is 7

Enter a value: 3
Running total is 10

Enter a value: 5
Running total is 15

Enter a value: 12
Running total is 27

Enter a value: 0
```

Tip: If your program is looping without stopping, you can press Control + C to quit it.

### Extension

If you solve this problem quickly, think about how you might extend the program to also keep track of the minimum and maximum numbers entered so far as well. A sample run of this extended program might look like this:

```
Enter a value: 42
Running total is 42
Maximum so far is 42
Minimum so far is 42

Enter a value: 3
Running total is 45
Maximum so far is 42
Minimum so far is 3

Enter a value: -6
Running total is 39
```

```
Maximum so far is 42
Minimum so far is -6

Enter a value: 0
```

## FizzBuzz

In the game Fizz Buzz, players take turns counting up from one. If a player’s turn lands on a number that’s divisible by 3, she should say `fizz` instead of the number, and if it lands on a number that’s divisible by 5, she should say `buzz` instead of the number. If the number is both a multiple of 3 and of 5, she should say `fizzbuzz` instead of the number.

It is an interesting problem in control flow and parameter usage. Write a program asks the user for an integer. The program should count up until and including `n`, fizzing and buzzing the correct numbers along the way. Once it's done, the program should print how many numbers were fizzed, buzzed, or fizzbuzzed along the way.

Here's a sample run of the program (user input is in ***bold italics***):

```
Number to count to: 17
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
Fizzbuzz
16
17

Num fizzed: 4
Num buzzed: 2
Num fizzbuzzed: 1
```

## Collaborative Workspace

*This workspace slide does not have a description.*