# **CIS 61 :: Lab 02 - Functions and Control**

#### **Student Name:**

**Instructions**: Use Sublime Text to write and use Command Line execute below programs. Attach Snipping photos of your source code and executions of the code in Python shell (or run a doc test). You can create a separate Python file for each exercise. Make sure to submit this template as a single file with all of your solutions. Do not zip your file.

[DO NOT USE THE **FOR** **LOOP, LISTS** or **RECURSION** IN ANY OF YOUR SOLUTIONS]

### Part 1 - Functions and Control

### Q1: Fix the Bug

|  |
| --- |

### Q2: A Plus Abs B

|  |
| --- |

### Q3: Two of Three

|  |
| --- |

### Q4: Largest Factor

Write a function that takes an integer n that is **greater than 1** and returns the largest integer that is smaller than n and evenly divides n.

|  |
| --- |

### Q5: Sum Digits Write a function that takes in a nonnegative integer and sums its digits. (Using floor division and modulo might be helpful here!)

|  |
| --- |

### Q6: Hailstone

| The largest hailstone sequence starting number I found was 73 |
| --- |

Hailstone sequences can get quite long! Try 27. What's the longest you can find?

Q7: Fibonacci Number

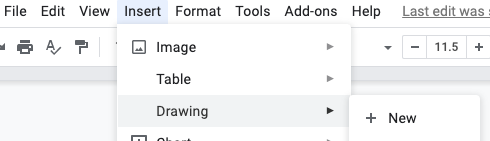
Write a function to compute the nth Fibonacci number. **DO NOT USE RECURSION.**

|  |
| --- |

Q8: Is Prime?

|  |
| --- |

### **Part 2 - Environment Diagram**

For the below questions you can use the drawing tool or write your solution on a piece of paper and then take a picture and add the picture under the question.

**Q9:** Assignment statements, such as x = 3, define variables in programs. To execute

one in an environment diagram, record the variable name and the value:

1. Evaluate the expression on the right side of the = sign
2. Write the variable name and the expression's value in the current frame.

Use these rules to draw a simple diagram for the assignment statements below.

| **a = 3**  **b = 4**  **def square(x):**  **return x \* x**  **def sum\_of\_squares(x, y):**  **return square(x) + square(y)**    **c = sum\_of\_squares(a, b)** |
| --- |
|  |

**Q10:** Call expressions, such as square(2), apply functions to arguments. When executing call expressions, we create a new frame in our diagram to keep track of local variables:

1. Evaluate the operator, which should evaluate to a function.
2. Evaluate the operands from left to right.
3. Draw a new frame, labeling it with the following:
   * A unique index (f1, f2, f3, ...)
   * The intrinsic name of the function, which is the name of the function object itself.  
     For example, if the function object is func square(x)  
     [parent=Global], the intrinsic name is square.
   * The parent frame ([parent=Global])
4. Bind the formal parameters to the argument values obtained in step 2 (e.g. bind x to 3).
5. Evaluate the body of the function in this new frame until a return value is obtained. Write down the return value in the frame.

If a function does not have a return value, it implicitly returns None. In that case, the “Return value" box should contain None.

Let's put it all together! Draw an environment diagram for the following code.

| **def square(x):**  **return x \* x**  **def a\_plus\_bc(a, b, c):**  **bc = b \* c**  **return a + bc**  **a\_plus\_bc(square(2), 3, square(square(3)))** |
| --- |
|  |