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

**Student Name: Jonathan Adrianto Saleh (CCCID :** CAU9032)

**Instructions**: Use Python IDLE to write and 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 separate files 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

The following snippet of code doesn't work! Figure out what is wrong and fix the bugs.

|  |
| --- |
| **def** **both\_positive**(x, y):  """Returns True if both x and y are positive.  >>> both\_positive(-1, 1)  False  >>> both\_positive(1, 1)  True  """  **return** x > 0 **and** y > 0 # You can replace this line! |

### Q2: A Plus Abs B Fill in the blanks in the following function for adding a to the absolute value of b, without calling abs. You may **not** modify any of the provided code other than the two blanks. **Hint: You can use add and sub functions**

|  |
| --- |
| **from** operator **import** add, sub  **def** **a\_plus\_abs\_b**(a, b):  """Return a+abs(b), but without calling abs.  >>> a\_plus\_abs\_b(2, 3)  5  >>> a\_plus\_abs\_b(2, -3)  5  """  **if** b < 0:  f = add(a,-b)  **else**:  f = add(a,b)  **return** f |

### Q3: Two of Three Write a function that takes three *positive* numbers and returns the sum of the squares of the two largest numbers. **Use only a single line for the body of the function.**

|  |
| --- |
| **def** **two\_of\_three**(a, b, c):  """Return x\*x + y\*y, where x and y are the two largest members of the  positive numbers a, b, and c.  >>> two\_of\_three(1, 2, 3)  13  >>> two\_of\_three(5, 3, 1)  34  >>> two\_of\_three(10, 2, 8)  164  >>> two\_of\_three(5, 5, 5)  50  """  **throw = min(a,b,c)**  **num = (a\*a) + (b\*b) + (c\*c) – (throw\*throw)**  **return num** |

**Hint:** Consider using the max or min function:  
 >>> max(1, 2, 3)  
 3  
 >>> min(-1, -2, -3)  
 -3

### 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.

|  |
| --- |
| **def** **largest\_factor**(n):  """Return the largest factor of n that is smaller than n.  >>> largest\_factor(15) # factors are 1, 3, 5  5  >>> largest\_factor(80) # factors are 1,2,4,5,8,10,16,20,40  40  >>> largest\_factor(13) # factor is 1 since 13 is prime  1  """  Divisor = 1  Answer = 0  while n > 1:  while divisor < n:  if n % divisor == 0:  answer = divisor  divisor = divisor + 1  else  divisor = divisor + 1  return answer |
|  |

### 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!)

|  |
| --- |
| **def** **sum\_digits**(n):  """Sum all the digits of n.  >>> sum\_digits(10) # 1 + 0 = 1  1  >>> sum\_digits(4224) # 4 + 2 + 2 + 4 = 12  12  >>> sum\_digits(1234567890)  45  >>> x = sum\_digits(123) # make sure that you are using return rather than print  >>> x  6  """  total = 0  While n > 0:  N2 = n%10  n = n // 10  total = total + n2  Return total |
|  |

### Q6: Hailstone

Douglas Hofstadter's Pulitzer-prize-winning book, *Gödel, Escher, Bach*, poses the following mathematical puzzle.

1. Pick a positive integer n as the start.
2. If n is even, divide it by 2.
3. If n is odd, multiply it by 3 and add 1.
4. Continue this process until n is 1.

The number n will travel up and down but eventually end at 1 (at least for all numbers that have ever been tried -- nobody has ever proved that the sequence will terminate). Analogously, a hailstone travels up and down in the atmosphere before eventually landing on earth.

This sequence of values of n is often called a Hailstone sequence. Write a function that takes a single argument with formal parameter name n, prints out the hailstone sequence starting at n, and returns the number of steps in the sequence:

|  |
| --- |
| **def** **hailstone**(n):  """Print the hailstone sequence starting at n and return its  length.  >>> a = hailstone(10)  10  5  16  8  4  2  1  >>> a  7  ""”  Print(n)  while n != 1:  if n % 2 = 0:  n = n / 2  print(n)  else :  n \* 3 + 1  print(n) |
|  |

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.**

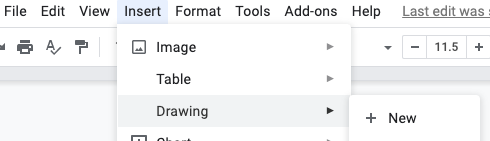
|  |
| --- |
| **def** **fibonacciN**(n):  """Return the nth Fibonacci number.  Fibonacci Numbers is a series of numbers in which each number is the sum of the two preceding numbers  >>> **fibonacciN**(5) # 1, 1, 2, 3, 5  5  >>> **fibonacciN(**7**)**  13  """  old = 0  current = 1  ct = 1  while ct < n:  series = old + current  old = current  current = series  ct = ct + 1  return current |
|  |

Q8: Is Prime?

Write a function that returns True if a positive integer n is a prime number and False otherwise.  
Hint: use the % operator: x % y returns the remainder of x when divided by y.

|  |
| --- |
| def is\_prime(n):  """  >>> is\_prime(10)  False  >>> is\_prime(7)  True  """  If n > 1 :  If n % 2 == 0 or n % 3 == 0 or n % 5 == 0 or (n % 7 == 0 and n != 7):  Return False  Else :  Return True |

### **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.

|  |  |
| --- | --- |
| x = 10 % 4  y = x  x \*\*= 2 |  |

**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, labelling 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 double(x):  return x \* 2  hmmm = double  wow = double(3)  hmmm(wow) |  |