WAKE TECH
Biotechnology, Engineering, & Skilled Technologies

Python Lab 3 – Loop Again with Functions

BAT-212: BAT Logic and Programming

This material is based upon work supported by the National Science Foundation Advanced Technical Education grant program, A New Technician Training Program for Advanced Building Technologies, DUE-2000190.

The opinions, findings, and conclusions or recommendations expressed are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Python Lab 3 – Loop Again with Functions

OBJECTIVES

Upon completion of this activity the student will be able to:

- Write a program correctly using the for loop structure
- Write programs correctly using while loop structure.
- Write a simple function.
- Write a function with an argument.

PARTS AND EQUIPMENT

- Circuit Playground Express board.
- Computer with CircuitPython and Mu editor

REFERENCES

The following are immediately relevant to the lab:

- https://www.geeksforgeeks.org/python-while-loop/
- https://www.geeksforgeeks.org/python-for-loops/?ref=lbp
- https://www.geeksforgeeks.org/python-functions/?ref=lbp up to Types of Arguments
- https://www.w3schools.com/python/python_functions.asp

The following are general references:

- https://greenteapress.com/thinkpython/html/thinkpython002.html#toc5
- https://docs.python.org/3/tutorial/

BACKGROUND

Review of Loops

In the previous lab, you wrote a program a *for* loop. A *for* loop will execute a fixed, or determinate, number of times. The executable statements are indented below the *for* statement to the same level in order to mark off the block of code. We have used the *range()* function to determine how many times the *for* loop is executed.

In contrast, a *while* loop is an indeterminant loop, which means it continues until a condition is false. How long it takes for the condition to be false is not determined by the program, but by the environment of the program. It is useful for continuing as long as or until a binary input is asserted, or for waiting until a threshold is reached for an analog input. Note that a *while* loop will block any action outside of the loop for the duration of the loop.

Functions

Purpose

A function is a structure that allows encapsulation of code. There are two reasons to use a function. One is to separate a block of code that will be reused, so that the function can be called repeatedly, without having to write the same code over and over. A program could call the same function at different points in the code, or different programs might use the same function. If different programs use the same function, the function would be included in a library that could be shared between the programs.

A second reason to use functions is to set aside the code needed to perform a particular task, so that the main flow of the program is clearer. For example, the code needed to convert a sensor input to the desired quantity, like converting input voltage to CFM, could be a function called *voltage_to_CFM( )*. Then the idea of what the code does is included in the main flow, without having to sort through all the details of the conversion each time you look at the code. This also allows the code to be tested as a unit and set aside so it is not likely to get altered as the rest of the program code is developed.

Syntax

The syntax of a function is in two parts – the definition of the function and the call of the function.

The definition of the function sets the name of the function, the parameters or data values that are expected to be passed to the function, and the statements that are executed when the function is called. The syntax of the definition is the keyword *def* following by the name of the function, followed by *()*. The parentheses can contain the arguments or data that is passed to the function. The arguments are given as variable names that are used in the executable statements of the function. Following is an example with no data passed. Notice the *def* keyword and the colon after the parentheses.

def func1_name():

    executable statement 1

    executable statement 2

    executable statement …

Next line not indented is not part of the function.

The call of the function is how the function is invoked. The call is a statement placed in code outside of the function definition. Following is the call for the previous example.

executable statement 1

func1_name()

executable statement 2

Notice that the call is just the function name with the parentheses. It is at the same level of indentation as the executable lines before and after it. Statement 1 would execute, then all the statements in the function definition would execute, and then statement 2 would execute.

Following is an example with two values passed.

def func2_name(arg1, arg2):

Use arg1 and arg2 as variables, for example,

if arg1>arg2:

do something…

executable statement 2

executable statement …

Next line not indented is not part of the function.

Notice that the indentation of the if maintains the indentation for the function and the statements in the if block are indented further. Notice that the variable name used in the parentheses is the same as that used in the body of the function.

Following is the call for the previous example.

sensor1 = 5

sensor2 = 10

func2_name (sensor1, sensor2)

executable statement …

Notice that the variables in the parentheses are used and have value in the calling program. Data can be passed in the call without being a variable, for instance, *func2_name(5, 10)* would work in the above example.

This is a very limited description of functions, but all that you should need to do this lab. For further examples and more information about the variety of ways functions can interact with data, please see the referenced websites.

PROCEDURES

While Loop Program 1

- This program will have to following action:
  - While button a is pressed, neo pixel 1 and only neo pixel 1 is lit.

    - While button b is pressed neo pixel 8 and only neo pixel 8 is lit.
- Look at previous programs you have done and try to determine what you need for setup, that is, what imports you need, etc. You will still need the *while True:* loop even though we are using whiles inside it, to be the run forever loop.
- Test the operation of your code and demonstrate to the instructor.
- **Comment your code well and submit.** Using the commenting guidelines given in the introduction lab.

While Loop Program 2

For this program, you will combine ideas from previous programs to roll the neo pixels either clockwise or counter clockwise based on a threshold value of the light sensor.

- Look at your previous programs to determine how to set up the light sensor and to find the code for rolling the neo pixels in both directions. Choose a value for the threshold that is easy to test and print the light sensor value to the serial monitor as a debugging aid.
- Test the operation of your code and demonstrate to the instructor. Save a copy of this code for use later.
- **Comment your code well and submit.** Using the commenting guidelines given in the introduction.

Function Program 1:

Start with the program you saved above. Look at https://www.geeksforgeeks.org/python-functions/?ref=lbp for the syntax of a function.

- Create two functions, *rollCW( )* and *rollCCW( )* that encapsulate the code that rolls the lights in either direction. Call the functions from the while loop to implement the lights rolling.
- Test the operation of your code and demonstrate to the instructor.
- **Comment your code well and submit.** Using the commenting guidelines given in the introduction.

Function Program 2:

- Change Function Program 1 to pass the time value for the roll to the functions as an argument.
- Test the operation of your code and demonstrate to the instructor.
- **Comment your code well and submit.** Using the commenting guidelines given in the introduction.