Programming in the CircuitPython language: A Python language basic reference.

Paul Mirel 20200927

PROGRAMS

A Program is a sequence of instructions, written in a coded language.

VARIABLES

A Variable is a named container for holding data. We put data into a variable like this:

```
x = 200
```

This puts the value 200 into the variable named x. The direction of data flow is right to left. The program evaluates the information on the right side of the equals sign, and assigns the information to the variable on the left side of the equals sign.

Some data types:

number	a number can be a whole number like 8, or a decimal number like 3.4
string	a string is a group of characters grouped by “” marks, like “Hello”, or “89a345@”

FUNCTIONS

A Function is a group of instructions, which may take incoming data.
Within a program, we may make a group of some instructions, and give it a name. The command for defining a function is def, and we use it like this:

```
def functionname (optionalvariablename):
    first instruction
    second instruction
```

To use an existing function, make a function call to it like this:

```
functionname (data)
```

After the last instruction in a function is completed, the program returns to run the next instruction after the function call.

Other people have written functions for us to use. An example of this is print(), which prints to the screen so that we may read what is printed.

```
print(8)
```

prints the number 8 to the screen. The print() function is built in to the CircuitPython programming language itself.

ARGUMENTS

Each piece of data accepted by a function is an Argument. A function may take one argument, more than one argument, or no arguments at all.

functionname ()	takes no arguments
functionname (argument)	takes one argument
functionname (argument1, argument2)	takes two arguments

LIBRARIES

A Library is a collection of functions written by other people for our optional use.

You can download a library file from the internet, choose which libraries from the file you need, and copy those libraries to the lib folder on your CIRCUITPY drive. Library file names are of the form:

```
libraryname.mpy
```

To use a library in your program, import it like this:

```
import libraryname
```

After that import, you can use any of the functions in that library as instructions in your program.

OBJECTS and METHODS

A library may also include instructions for creating a program Object.

A program Object is collection of data. The data in an object can include Methods, which are functions that can act on the object itself.

The method for creating an object in your program is determined by the people who wrote the library. You may name the object whatever you wish to name it.

These are the instructions for creating a Neopixel object for controlling neopixel RGB led lights:

```
num_leds = 10
# Declare a NeoPixel object on pin D8 with num_leds pixels, no auto-write.
pix = neopixel.NeoPixel(board.D8, num_leds, brightness=0.1, auto_write=False)
```

I chose to name the object pix because that is a name that reminds me of what it is, and is easy to type.

When a program sees a #, it ignores the # and anything after it on that line. This allows you to make a comment on the program without interfering with the program's operation.

Some of the methods available in the NeoPixel object are:

pix[address] = (red, green, blue)	which sets the color and brightness data for the RGB led light at that address, but does not send the data.
pix.show ()	which sends the color and brightness data to the RGB led lights
pix.fill ((red, green, blue))	which lights all the RGB led lights with the color specified in red, green, and blue

CONDITIONALS

A conditional is a specific type of function that creates a choice based on the evaluation of a condition:

Two types of conditionals are if, and while.

```
if (condition):
        instruction1
        instruction2
```

"If condition is true, do these things once"
If the condition evaluates to True, the program will do instruction1 and instruction2.
If the condition evaluates to False, the program will skip instruction1 and instruction2.

```
while (condition):
        instruction1
        instruction2
```

"While condition is true, do these things repeatedly."
If the condition evaluates to True, the program will do instruction1 and instruction2, over and over until the condition no longer evaluates to True.
If the condition evaluates to False, the program will skip instruction1 and instruction2.

A useful addition to the if conditional is else.

```
if (condition):
        instruction1
        instruction2
else:
        instruction3
        instruction4
```

COMPARATORS

In conditionals, we can make conditions like these:

True	Always evaluates to True. Use the capital T.
False	Always evaluates to False. Use the capital F.
a == b	True if a and b are equal. False if they are not equal. Important Note: Use the two equal signs, ==, to avoid assigning the value of b to the variable a, as in a = b.
a != b	True if a and b are not equal. False if a and b are equal.
a < b	True if a is less than b. False if a is equal to or greater than b.
a > b	True if a is greater than b. False if a is equal to or less than b.
a >= b	True if a is equal to or greater than b. False if a is less than b.
a <= b	True if a is less than or equal to b. False if a is greater than b.

INCREMENTAL LOOP

Using the while conditional, we can make a loop that runs for a certain number of iterations, and then exits.

```
placeholder = 0
while (placeholder < 5):
    instruction1
    instruction2
    placeholder = placeholder + 1
instruction 3
```

Initially, placeholder is 0, because we set it to be. So the condition in the while statement, placeholder < 5, evaluates to True, because 0 really is less than 5.

The program runs instruction1, and then instruction2, and then adds the value 1 to whatever value is already in placeholder. In this case, 0 + 1 = 1.

instruction3 does not run at this time, because it is not part of the while code block. Code blocks in Python are set by the indentation level of the text.

The program loops back around to the while statement, re-evaluating the condition. The condition now is that placeholder is equal to 1. 1 is still less than 5, so the condition is still true, so the program runs instruction1 and instruction2, and loops back to the while statement.

This goes on until placeholder is equal to 5, at which point the condition (placeholder < 5) evaluates to False. At that point, the program skips instruction1 and instruction2. The program goes directly to run instruction3.

Note: The conditional for is a compact version of while.