MicroPython and Microcontrollers

NetApp YWIT

May 1, 2020

Contents

Chapter 1

Introduction

This workshop will introduce the student to Python coding, electronics, and project design. We will building several projects ranging from simple to complicated. These projects are based on the ESP8266 microcontroller which is running MicroPython and they depend on some other electronics components such as LEDs, buttons, and more.

Chapter 2

Project 1: Blink

Chapter 3

Project 2: Button

Chapter 4

Project 3: LED Party

Chapter 5

Project 4: Sensor

Chapter 6

Project 5: Sound

Chapter 7

Project 6: Game

Chapter 8

Project 7: Chat

Appendix A

Electronics Essentials

Appendix B

Python Primer

If you're new to Python, this section will give you a few things you should know in order to better understand the projects in this guide. This is by no means a complete or comprehensive look at the Python language. For that, we recommend looking at the official Python site and reading through the tutorial there.

> Note: for the projects being used here, we are using an implementation of Python known as MicroPython. This version is meant to run on microcontrollers with limited resources. It also has built into it libraries for dealing with hardware devices that are not part of the standard CPython distribution. Therefore, not all Python examples you find online will run on your microcontroller and not all projects for a microcontroller can be run on your computer. But a lot of the code can be shared so the lessons you learn here can apply to other Python projects.

Here is a sample of a small Python script. We will disect and explain what each section does below:

Listing B.1: An example Python script

```
1  def show(message, repeat=1):
2      """This function prints the given message to the
3      console as smany times as specified in the
4      srepeat parameter.
5      """
6
7      for iteration in range(0, repeat):
8          print(iteration, message)
9
10 name = input("What is your name: ")
11 show(name)
12 show(name, repeat=3)
```

On line 1, we are defining a function named show. This function accepts two parameters, message and repeat. The message parameter is required and the repeat parameter is optional with a default value of 1.

Lines 2 through 5 comprise the docstring for the function. This information is meant for programmers to read and explains what the function does. It does not affect how the function works.

Line 7 starts a loop. The loop will repeat the statements in the loop body until a condition is met. In this case, it will loop until it has performed the operation for each repeat.

Line 8 is the body of the loop. This statement will print the message that the user passed in to the console along with the iteration number of the loop.

Line 10 prompts the user for their name and saves the result in a variable called name.

Line 11 calls our show function which will print the user's name once (the default).

Line 12 calls our show function again, this time saying that we want to repeat the loop of printing the name twice.

Running the program, we will see output like this:

```
$ python program.py
What is your name: Emily
0 Emily
0 Emily
1 Emily
2 Emily
$
```

Another feature that you'll see used often in Python are classes. Classes are a convienent way to model something in your program that holds state and implements functionality. For example, let's say that we are writing a game about racing go- karts. We need to allow each player to have their own kart and keep track of how fast it is going, which way they are turning, and allow the kart to speed up and slow down. Here is a small class that will help us do that:

Listing B.2: An example of a Python class

```
1  class Kart:
2      MAXIMUM_SPEED = 100
3
4      def __init__(self):
5          """The kart starts motionless at the ↘
               beginning"""
6          self._speed = 0
7          self._direction = 0
8          self._acceleration = 0
9
10     def brake(self):
```

```
11          """This is called when the user presses the ↘
                brake button"""
12          self._acceleration = -5
13
14      def accelerate(self):
15          """This is called when the user presses the ↘
                accelerator button"""
16          self._acceleration = 5
17
18      def steer(self, direction):
19          """This is called when the user presses left ↘
                or right"""
20          self._direction = direction
21
22      def update(self, ticks):
23          """Update will be called by our game engine ↘
                and will be
24          provided the number of ticks since it was ↘
                last called.
25          """
26
27          self._speed += self._acceleration * ticks
28
29          # limit our speed so that we don't go faster ↘
                than our
30          # kart is allowed to, or slower than 0
31          if self._speed > Kart.MAXIMUM_SPEED:
32              self._speed = Kart.MAXIMUM_SPEED
33          if self._speed < 0:
34              self._speed = 0
```

Looking at this class, there are 5 methods. The first one (on line 4) is a special method that is called by the Python interpreter whenever a new Kart is created. It will initialize some variables for this particular Kart object.

You may have noticed that the first method takes a parameter called "self". This is the first parameter of all methods in a class in Python. It is automatically passed by the interpreter and is a reference to the current object. It lets us access the variables that belong to the class, like those we defined in the __init__ method.

Speaking of the variables in the __init__ method. Notice how we named them all with an underscore? This is a convention in Python that says they are private to our class and that code written outside of the class shouldn't access them directly. That means that our class should provide ways to modify or read these variables via other methods.

The second method starts on line 10. This is called when the player presses the brake button on their controller and will set our Kart's acceleration to a neg-

ative value so that we start to slow down. It modifies the private _acceleration member of the class.

The third starts on line 14. It is the opposite of braking and will start speeding our Kart up when the user presses the accelerator. It aslo modifies the private _acceleration member of the class.

The fourth method, line 18, is again something to deal with user input. This time we can see that it takes a second parameter, direction. If the user presses left on their controller, then we can expect left to be passed here. The same for right. We will modify the private _direction member here.

Finally, we have a fifth method starting on line 22. This method is called by our game engine and uses the class members to determine what happens to the Kart throughout the game. That is, it is asking the Kart to update itself at a certain moment in time (usually once per frame) so that next time it draws it to the screen, it will be in the updated location.

Notice in the last method, we are accessing not only our own variables, _speed, and _acceleration, but we are also reading a class variable, Kart.MAXIMUM_SPEED. Unlike our member variables, a class variable is the same for all instances of a class. It is useful here to keep the game fair so that all Karts have the same limitation on their speed.