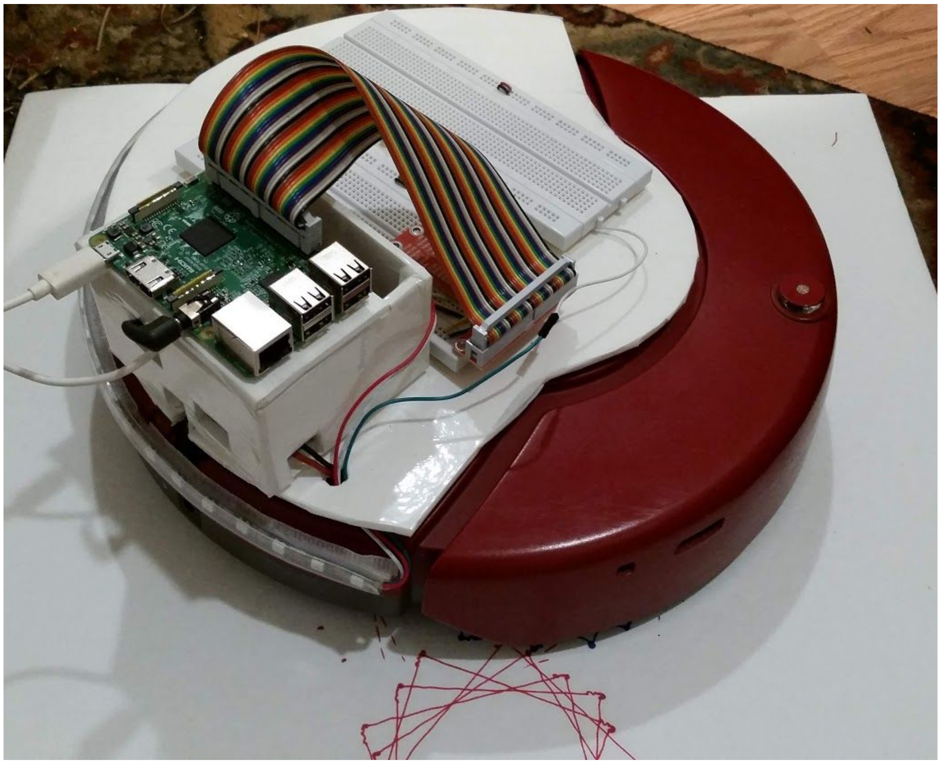
Roomba Logo Bot

Programming Manual



Dedicated To: Talia Rose

Merry Christmas 2018!

Introduction

Growing up in the 80’s I had a keen interest in the roll that robots would play in the future. From Rosie, the robot maid in the Jetsons cartoon, to R.O.B the Nintendo robot that helped you play video games, robots seemed to be where the future was headed. One particular show that sparked my curiosity was called “Start Here! Adventures Into Science”. In this show a robot named Konrad helped students learn science by helping them perform experiments ranging from electronics to magnetics to everything in between. Konrad had these television displays for his eyes, and bright flashing lights everywhere. His head could pivot all the way around, and he could converse with the kids as if he had true intelligence. As a kid, I wanted a robot like Konrad. I wanted my own robot teacher that I could ask questions, and that could respond with all the secrets of the universe.

Fast forward 40 years, and we may not have robot teachers, but we do have some amazing technology. We have super computers that fit in our pockets (ie smartphones). We have a fountain of knowledge at our fingertips (ie Google and the Internet). And, although we don’t seem to be surrounded by the robots that were promoted in my childhood, they are out there. Roomba is an example of one of these robots. Roomba is a robotic vacuum cleaner invented by a company called iRobot in 2002. Roomba uses a plethora of brushes, vacuums, and sensors in order to clean floors. Pretty cool, right? What makes Roomba really neat though in my opinion is a feature that iRobot provided called the SCI, or Serial Command Interface. This interface provides a communication plug on the side of the robot that you can connect to and tell the robot what to do! Using the SCI, you can have a fully programmable robot, limited only by your imagination and programming skills.

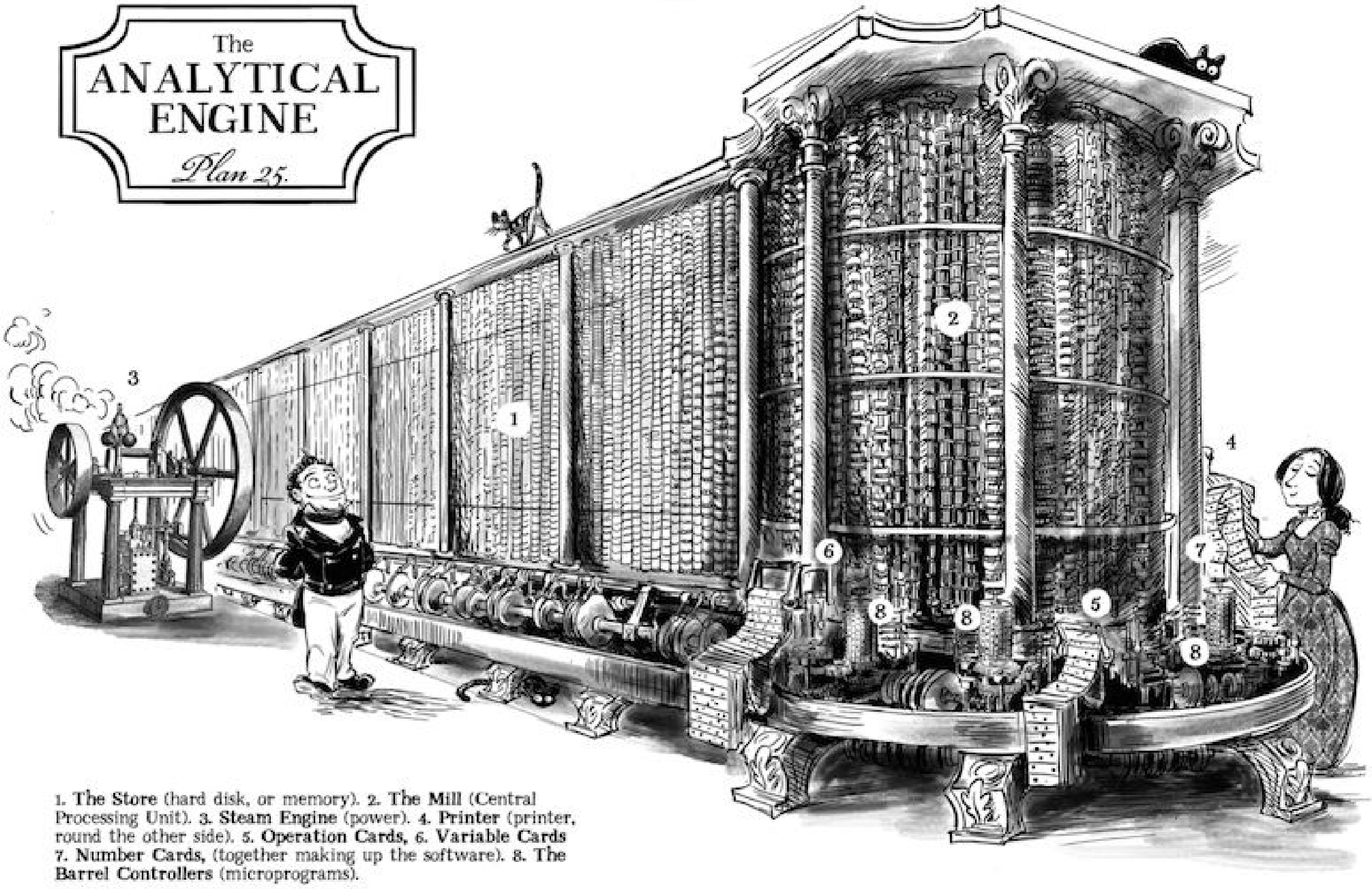
In this book, I hope to expose students to the wonders and possibilities of robotics, by giving them the tools they need to program a Roomba.

Lesson 1

What is Programming?

Programming is simply the process of teaching a computer how to do something. Programming involves writing a set of instructions, called a program, in a special language that the computer understands. This is very similar to how we as humans learn. Think of how you learned to do something in your own life (ie play soccer, Monopoly, Ham Radio). How did you learn to do that thing? Most likely you had someone teach you how to do that thing. You were taught using a language that you understood. For you, your programming language might be English, Spanish, or whatever your native language is. And the program or set of instructions that taught you how to do something might have been your parent, a coach, Monopoly instructions, etc.

The first computer program was written by Ada Lovelace in the 1830's. Although they didn't have computers in the 1830's in the sense that we think of them today, they did have the concept of a mechanical computer known as the "Analytical Engine" that was thought up by a man named Charles Babbage. This analytical engine was built from pulleys, levers, and mechanical trappings that could store and fetch computer programs. Ada Lovelace created her own algorithms and programs for the analytical engine, although the machine itself was never built.



1. **The Store** (hard disk, or memory). 2. **The Mill** (Central Processing Unit). 3. **Steam Engine** (power). 4. **Printer** (printer, round the other side). 5. **Operation Cards**, 6. **Variable Cards** 7. **Number Cards**, (together making up the software). 8. **The Barrel Controllers** (microprograms).

What Is Python?

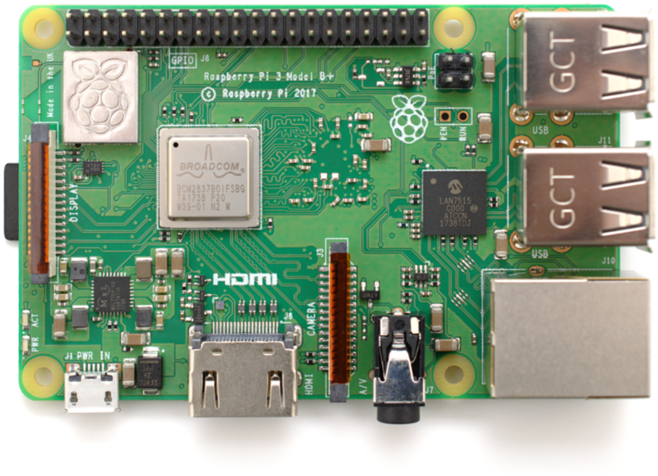
Python, simply put, is a type of programming language. As noted previously, a programming language is the language that the computer understands. This is a language that you can use to write instructions for the computer to follow. The language itself was created by a man named Guido van Rossum in 1991. Guido created the language to be easily readable by humans. Because of this, it's a great first language to learn in the world of programming. This is the primary language we'll be using in this book to write programs for your Roomba Logo Bot. Although the Python logo, as seen above, is of two snakes, the language was not named after Python the snake, but rather after the british comedian and movie writer Monty Python, who Guido was a huge fan of.

Some useful links as we learn Python are :

- https://www.python.org (the Python website)
- https://www.python.org/dev/peps/pep-0008/ (the Python style guide and best practices for writing good Python code)

What Is The Raspberry Pi?

When you hear the words raspberry pi, what do you think of? I'll bet for many of us we think of that yummy summertime treat that fills the air with a heavenly aroma. But since early 2012, the term has also come to be known as a very popular small and affordable computer designed to teach kids how to program. You might ask, why on earth did they name a computer after a dessert? Well, surprisingly, naming computers after fruit is nothing new. Apple, Acorn, Tangerine and Apricot are all computer companies named for various types of fruit. The Pi was added to the name not because it was to be baked inside a crust and left to cool on a windowsill, but because the primary programming language that was to be used on the computer was Python.



What is Roomba?

Roomba is an autonomous robot vacuum cleaner designed by a company called iRobot. iRobot saw a future with robot helpers and in 2002 decided to be a part of it by introducing Roomba to the world. Through a series of bump and distance sensors, Roomba is designed to navigate by itself around an indoor environment to clean the floors. Wanting to encourage the next generation of Robotics engineers and tinkerers, iRobot added a plug on the side of the Roomba which we can connect to in order to control the robot from a computer. The computer can send Roomba a set of commands that it understands (known as the SCI or Serial Command Interface). These commands can control every aspect of Roomba from the vacuum, to the brushes, to driving the wheels, and reading the sensors.

What is Logo?

Logo is a programming language created by Wally Feurzeig, Seymour Papert, and Cynthia Solomon in 1967. Logo got its name from the Greek word *logo*, which means word or thought. The creators of Logo wanted to design a language that was easy to read and simple enough for new programmers to learn. Logo for the Apple II computer was one of the languages I played with as a child. With Logo you could control a small dot (or turtle) on the screen, and make it draw designs by typing in commands. You could tell the turtle to go forward, right, left, etc a certain number of spaces. By combining these movements into a program, the turtle could draw very impressive looking designs. Your Roomba Logo Bot can be controlled with a similar programming structure as Logo. You can write a program that tells your robot to move forward, backward, left and right a certain number of spaces. By placing a marker into the hole in the center of the robot you can watch the programs you create on the computer come to life in the form of amazing designs on paper!

Lesson 2

Writing Your First Program

You’ve gotten to know your shiny new robot toy, you’ve had a primer in the things we’ll be covering with the rest of this book, now it’s time to write your first program! To start with, turn your robot on by flipping the power switch located on the side of the control module. You’ll see some leds illuminate on the power supply and Raspberry Pi to indicate that they’re receiving power. Wait about 30 seconds to give the Raspberry Pi time to start up and then open up the VNC Viewer application on your computer.

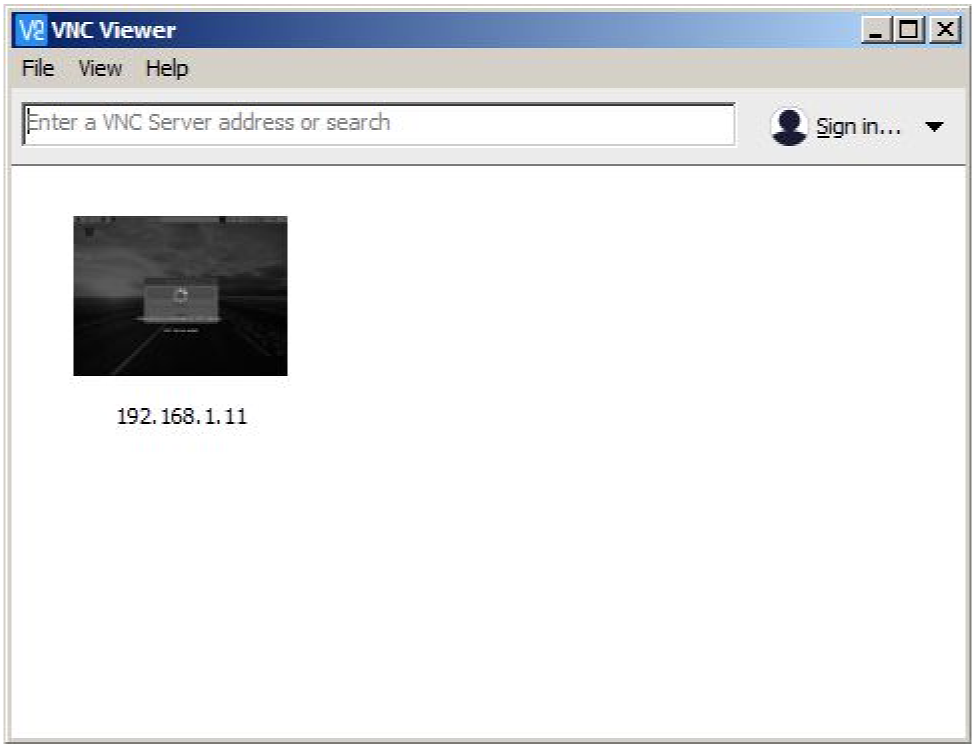


figure 2.a

Double click on the small computer screen icon that corresponds to the IP address of your robot. Note: the IP address of your robot should be printed on a label on the side of the control module. You should see a screen that says it's connecting to your IP. Then, after a moment, you will see the desktop environment on the Raspberry Pi. At the top left hand corner of the screen you'll see the Raspberry Pi logo . Click on it.

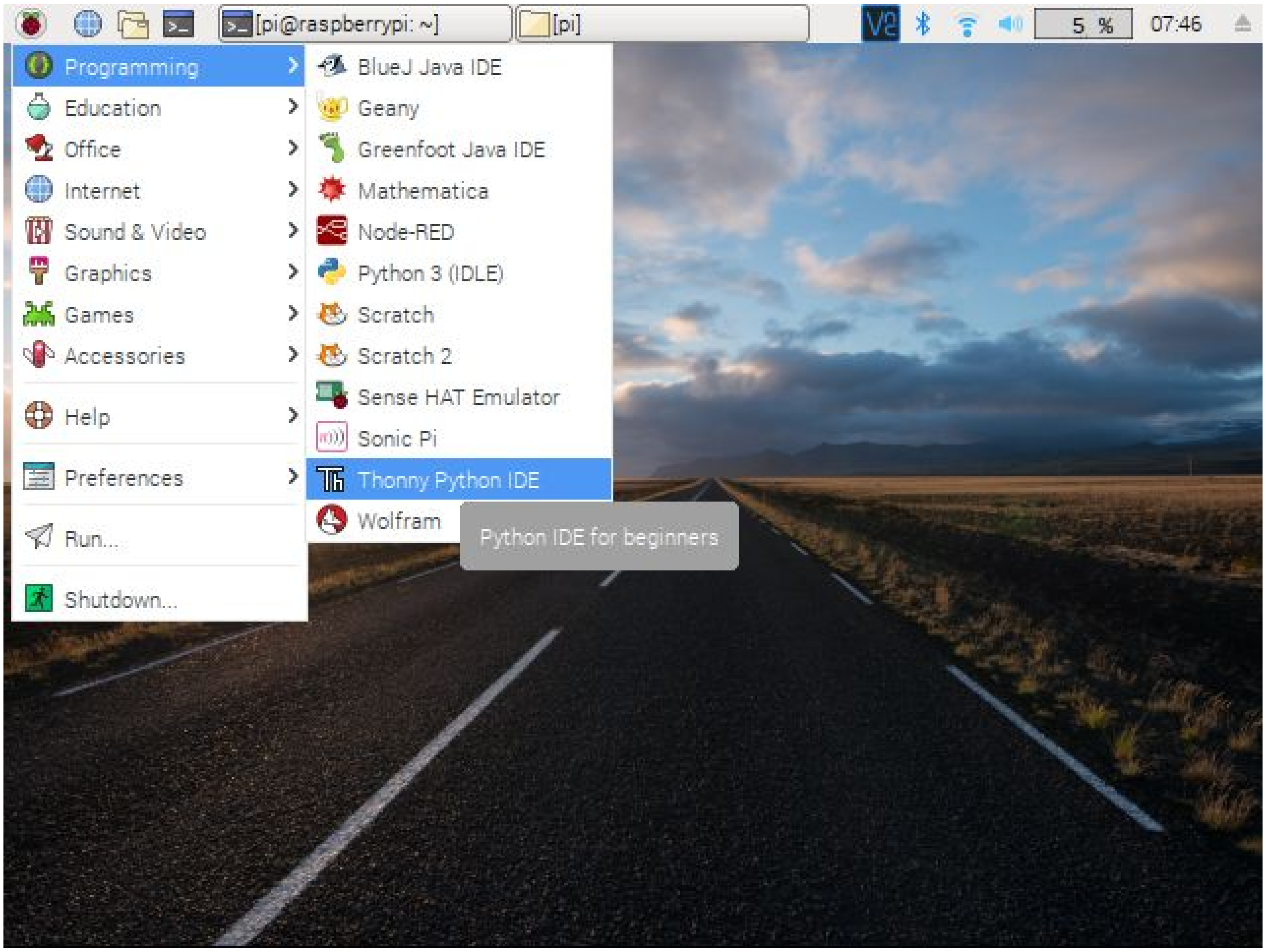


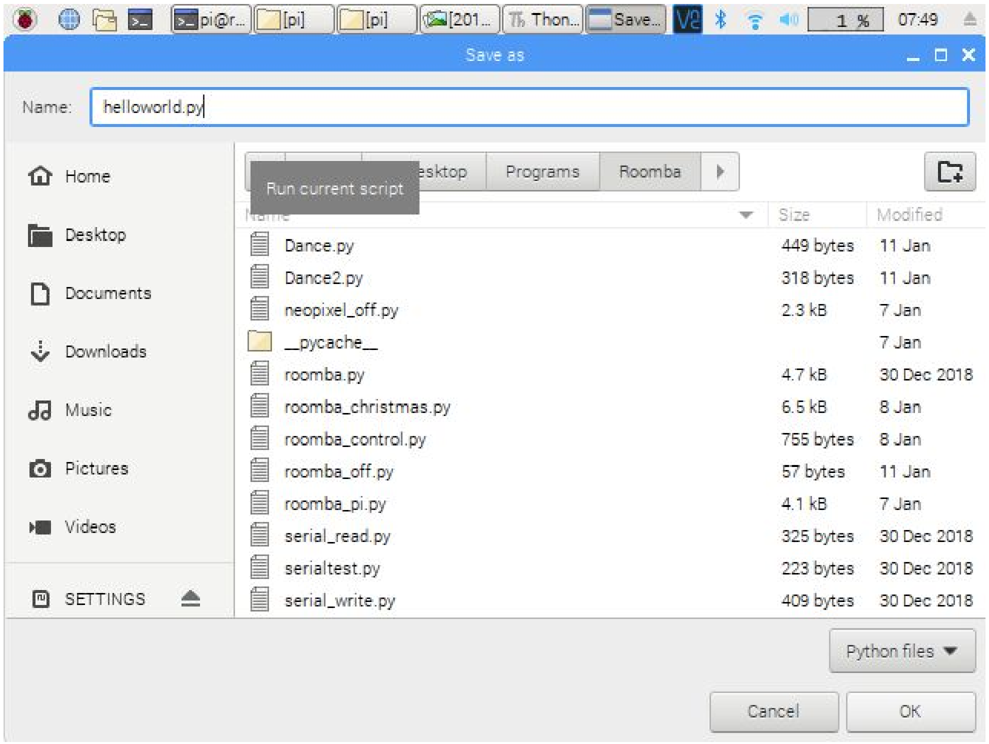
figure 2.b

You should see a pull down menu. Click on Programming. Then scroll down and click on the program named Thonny Python IDE to open it. Thonny Python is the name of the program that we'll use to develop and run our programs we'll build for the Roomba Logo Bot. You may be wondering what the IDE means at the end of the program name. IDE is an acronym that means Integrated Development Environment. An IDE provides all of the things on the software side that we'll need to write and run or execute a Python program. It combines, or integrates, a program editor (where we write our code), a debugger (that helps us identify bugs or problems in our code), and several nifty features that we'll discover later on.

With Thonny open, our first task will be to create a new file for our program to live inside. Click on the icon (a disk with an up arrow) to create a new program. This will create a new blank tab with the title “untitled” in it. Here is where you will write your code. To begin, click anywhere inside this new program editor and type out this line of code:

```
print("hello world!")
```

Cool, we wrote our first line of code! But what does it do? Well, this is a very special line of code, although it may not look like much. This line of code has been passed down to us by generations of programmers. As you may have guessed, this program will print out the words “hello world!” onto the screen. The hello world program is many times one of the first programs that a programmer will write when learning a new programming language, because it is so simple. In our code, the word “print” is actually something called a *function*. In programming, a function is something that we can utilize to reuse portions of our code. We’ll cover functions in more detail later on, but for now just know that the print function is a function that is built into Python, meaning it has already been written for us. A list of all of the built in functions for Python can be found here: https://docs.python.org/3.7/library/functions.html. The print function takes one argument, or parameter. The parameter is a list of text (also called “strings” in programming) that you want to print out, in our case the text is “hello world!”. Click on the green circle with the white triangle icon to run our program.

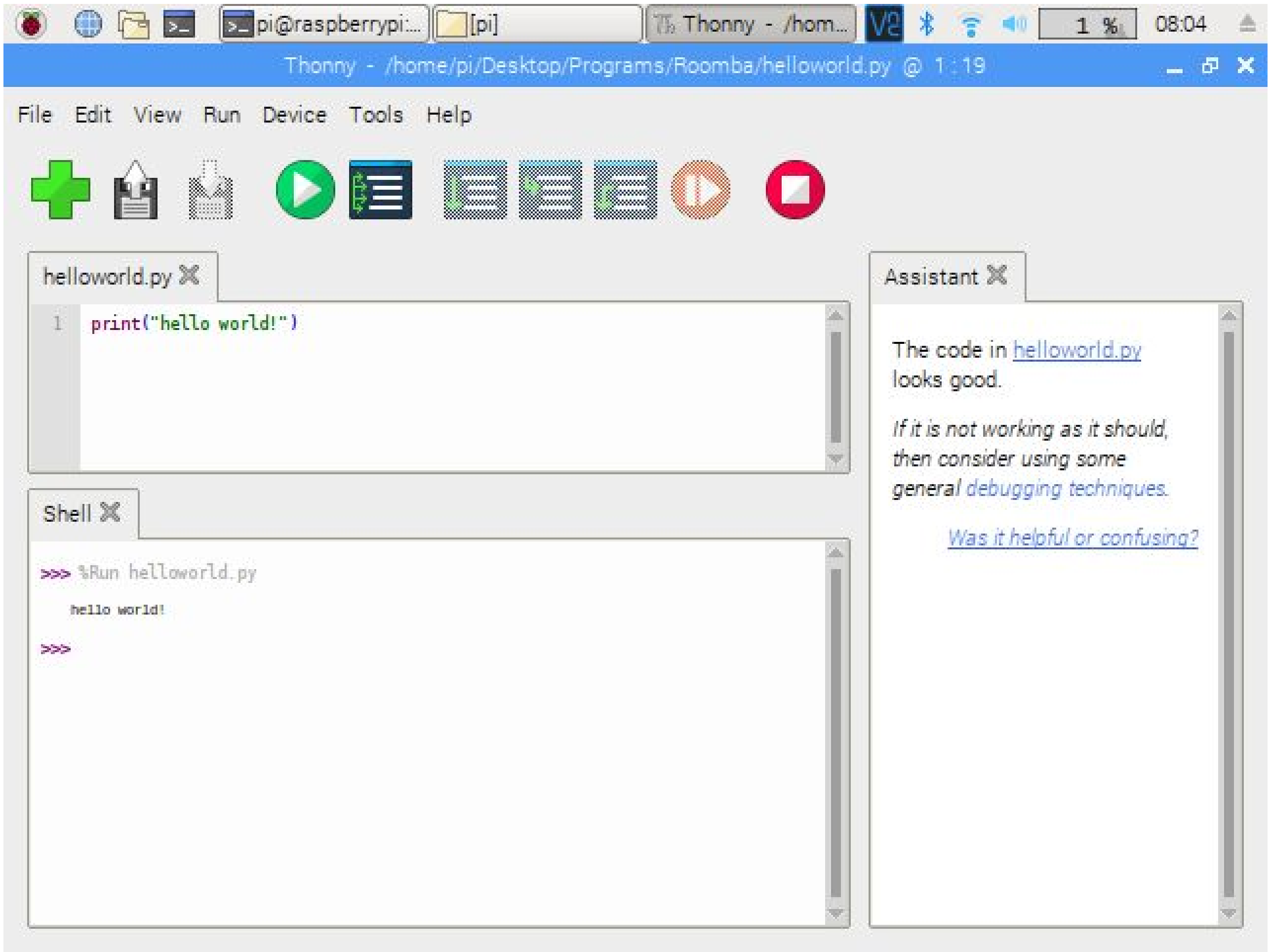


Name: helloworld.py

Name	Size	Modified
Dance.py	449 bytes	11 Jan
Dance2.py	318 bytes	11 Jan
neopixel_off.py	2.3 kB	7 Jan
__pycache__		7 Jan
roomba.py	4.7 kB	30 Dec 2018
roomba_christmas.py	6.5 kB	8 Jan
roomba_control.py	755 bytes	8 Jan
roomba_off.py	57 bytes	11 Jan
roomba_pi.py	4.1 kB	7 Jan
serial_read.py	325 bytes	30 Dec 2018
serialtest.py	223 bytes	30 Dec 2018
serial_write.py	409 bytes	30 Dec 2018

figure 2.c

A little window will open up, asking you where you would like to save the program, as well as what you would like to call it. In the Name text box type in “helloworld.py”(.py is the extension used to let the computer know that this is a Python program) and then click the OK button. Your program should now run and print out the text “hello world!” in the Shell window. If you don’t see the Shell window, click on the View menu and make sure Shell is checked. The Shell is simply the window that displays the output of our program, it is also where we can interact with our program more. We will go into more details about the Shell in a later lesson.



```
print("hello world!")
```

```
>>> %Run helloworld.py
  hello world!
>>>
```

The code in helloworld.py looks good.

If it is not working as it should, then consider using some general debugging techniques.

Was it helpful or confusing?

figure 2.d

Pretty cool right? What else can you make the program print? You can edit the text inside the print function to print out anything you want. For example, I could change the program to tell me hello by name:

```
print("hello Jon!")
```

Practice modifying the code to print out something else, maybe your name, or Shakespeare or whatever else your heart desires.

Our program is pretty cool already, but how could we make it a bit better? Maybe we could have it ask for our name and then print our name out instead of having to modify the code every time we want to print something new. Let’s take our program a bit further to accomplish this. Modify your program with the following lines of code:

```
name = input("I'm Roomba Logo Bot, what's your name?")
print("Nice to meet you," name, "!")
```

What does this code do? Let’s examine the first line of code :

```
name = input("I'm Roomba Logo Bot, what's your name?")
```

We can see that we’re setting something called “name” equal to a function called “input”. The word “name” here is what is called a *variable*. A variable simply stores something you want to use later in your program. We could have named the variable anything we wanted, but we used the word “name” because that was describing what the variable would contain. Whenever possible, it’s a good idea to give your variable descriptive names to make the code more readable. So we’re setting the variable “name” equal to a function called “input”. What does this input function do? Well, as you may have guessed, the input function is another built in Python function that allows you to collect input from the user running the program. The input function takes a single text (or string) parameter and returns whatever the user types in response to the input prompt. In our case, the input function will first print out the question “I’m Roomba Logo Bot, what’s your name?”. Then the input function will return the text that the user types and stick it into the “name” variable. Pretty simple right? Let’s check out the next line of code:

```
print("Nice to meet you," name, "!")
```

This line should look familiar to you, at least somewhat. We see our old friend the print function again. Remember before when we learned that the print function can take a list of text to print to the screen? The text we want to print out can be separated by commas that the print function will print out one right after another. So you can see here that the print function will first print “Nice to meet you, “ followed by the text in the name variable, and then lastly a “!”. That’s it! Now run the program to see it working.

You can modify the code to as different questions. Play around with the code and then continue onto the next lesson!

Lesson 3

Getting To Blinky

In the last lesson we learned how to write a basic program that takes input from a user and prints that input onto the screen. Now we'll go a bit further and show how we can write a program that will flash an LED (light emitting diode) on the breadboard of the Roomba Logo Bot! What I believe makes robotics so exciting is having the ability to program a real tangible thing, and to watch it interact with its environment. By blinking an LED, we're showing in a small way how you can write a program that can change the physical world around us.

To begin, let's briefly talk about what an LED is, and some fundamental principles of electronics. An LED is short for Light Emitting Diode. To emit light means to shine light. A diode is a component in electronics that only allows *electrical current* to flow in one direction. So an LED is a diode that can be used to shine a light. For our use we will use the LED's light emitting capability to flash the light on and off.

I mentioned something called electrical current in the preceding paragraph. What is electrical current? Well, current describes the amount of electrons that flows past a point in an electrical circuit at a given time. What is an electron? An electron is a negatively charged particle that makes electrical circuits work. Have you ever had the experience of walking across a carpet with bare feet and when you reach out to touch a metal door knob, you see a tiny spark and feel a small zap on the tips of your finger? That tiny spark is actually electrons hopping off your finger and onto the knob! This phenomenon, called static electricity, happens because the negatively charged electrons on your finger build up when you shuffle your feet on the floor, and jump over to the positively charged protons on the knob. This same thing happens, in a more controlled fashion, when you hook up an LED to a battery. The negatively charged electrons will want to travel through the LED and to the positively charged terminal of the battery because opposite charges attract each other.

So electrical current describes how many electrons are flowing past a point in a circuit at a given time. But is there a way for us to actually know and describe how many electrons are flowing at a point in a circuit? Yes, there is! We measure electrical current in something called *amperes* or amps for short. If we were to measure the amount of electrons that pass one point in a circuit that has a current of one ampere, we would count approximately 6.24 quintillion electrons in just one second! That's 6,240,000,000,000,000,000! That is a lot of electrons, and if your circuit isn't setup correctly, bad things can happen. This is why if you were to just hook up your LED directly to a battery it would most likely burn up, because it wasn't designed to handle the amount of current that the battery was able to deliver. For this reason, we use something else in our circuit, called a *resistor*, which as its name suggests, resists electrical current flow. You can think of a resistor as slowing down the current in a circuit. By putting a resistor in series with our LED, it's only allowing a certain amount of current to flow through the LED.

To begin, let's draw a schematic of our circuit. A schematic is just a fancy word we use for a drawing of an electrical circuit. Here's a schematic showing the first circuit that we will build:

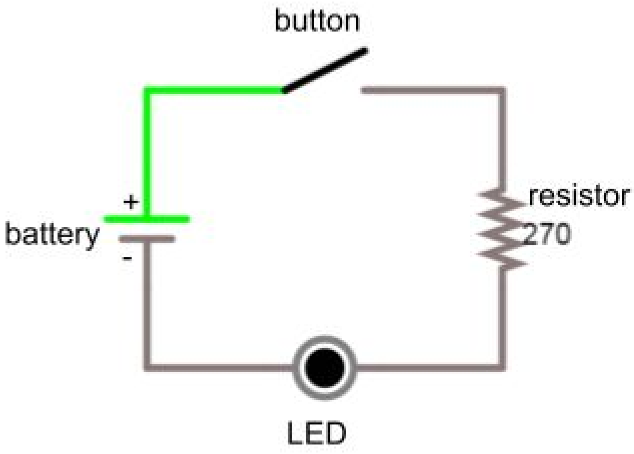
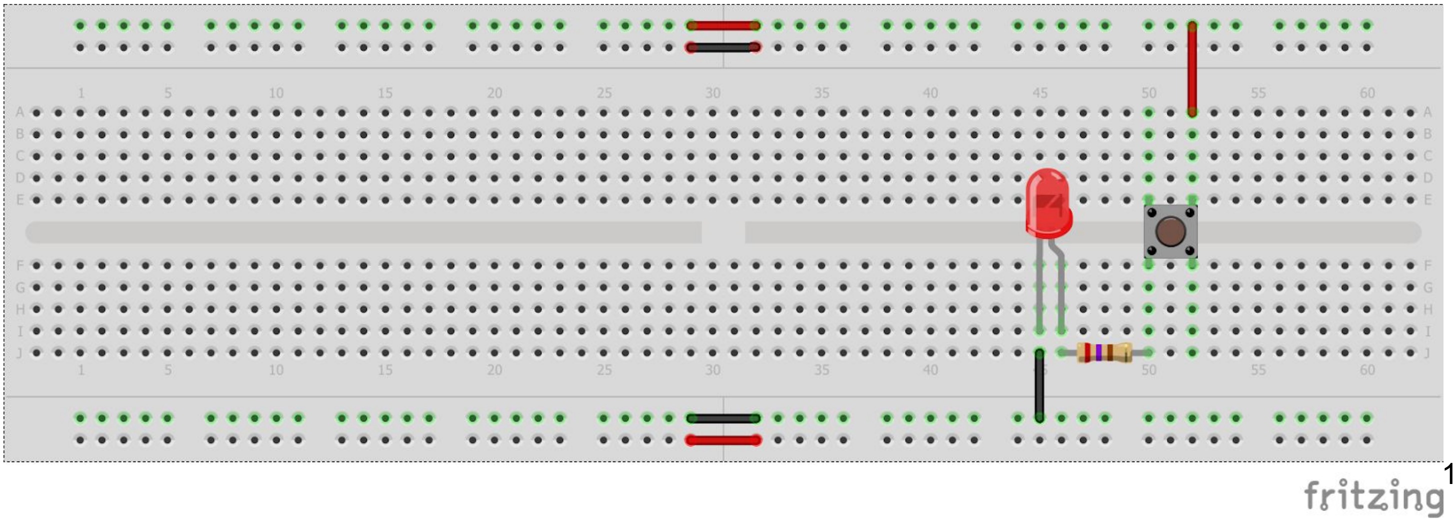


figure 3.b

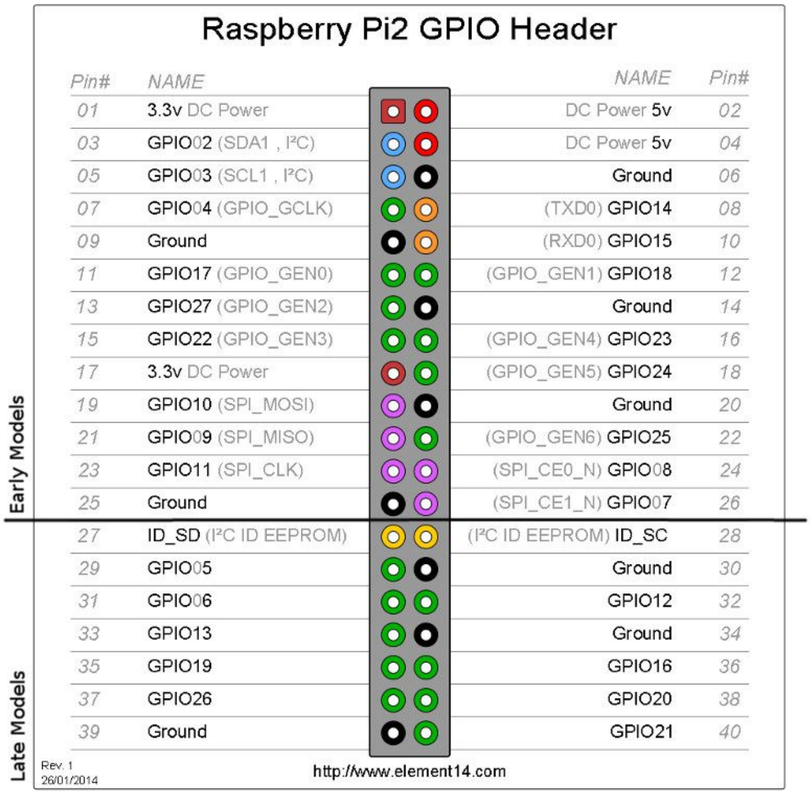
To view a real working schematic of this circuit visit this link: http://tinyurl.com/y87qbtwg . Use your mouse to hover your cursor over the button and simulate pushing the button in. You can see that the button completes the circuit and allows the electrons to flow from the negative side of the battery, through the LED (turning the LED on), through the resistor, and into the positive side of the battery.

Now it's time to wire this circuit up on our breadboard! Using the LED, resistor, and wires that came with your Roomba Logo Bot, wire up the circuit just like the breadboard pictured below. Notice that the LED has one end that is flat, and one end that is round. The flat end is called the cathode, and should connect to the ground or negative portion of the circuit as shown below:



If the circuit is built correctly, when you push the button, the LED should light. When the button is released, the LED turns off. Pretty neat, eh? This is cool and all, but what we really wanted to see was the robot blinking the LED, right? That's coming! But sometimes it's a good idea to build up a project in a series of small steps, called iterations, instead of just connecting everything all at once. Just as we started off our helloworld.py program with a basic print statement and slowly added more functionality, we want to do the same thing here. By doing this, we can avoid a lot of common mistakes along the way.

With the first iteration of our circuit done, let's now hook the LED up to the Raspberry Pi and write a little program to make it blink! The Raspberry Pi is a special kind of computer. Instead of keeping its internals sealed up and inaccessible, it is designed to spur your imagination and curiosity by giving you access to things a normal computer wouldn't. Take a look at the Raspberry Pi sitting on top of the control module. What do you notice that is different than a typical laptop or desktop computer? One thing that is obvious is it doesn't have a case. You can see all the electronic components laid out there on the PCB (printed circuit board). And the other thing that is strange is it has all these pins on the side where the ribbon cable is connected. These pins are "broken out" to locations on the breadboard. These pins give you access to what is referred to as the GPIO, or General Purpose Input Output of the Raspberry Pi. As their name suggests, these pins can be controlled by your code, to provide both input and output to the Raspberry Pi. Here is a closeup view of the pins on the Raspberry Pi, showing the names of each of the GPIO:

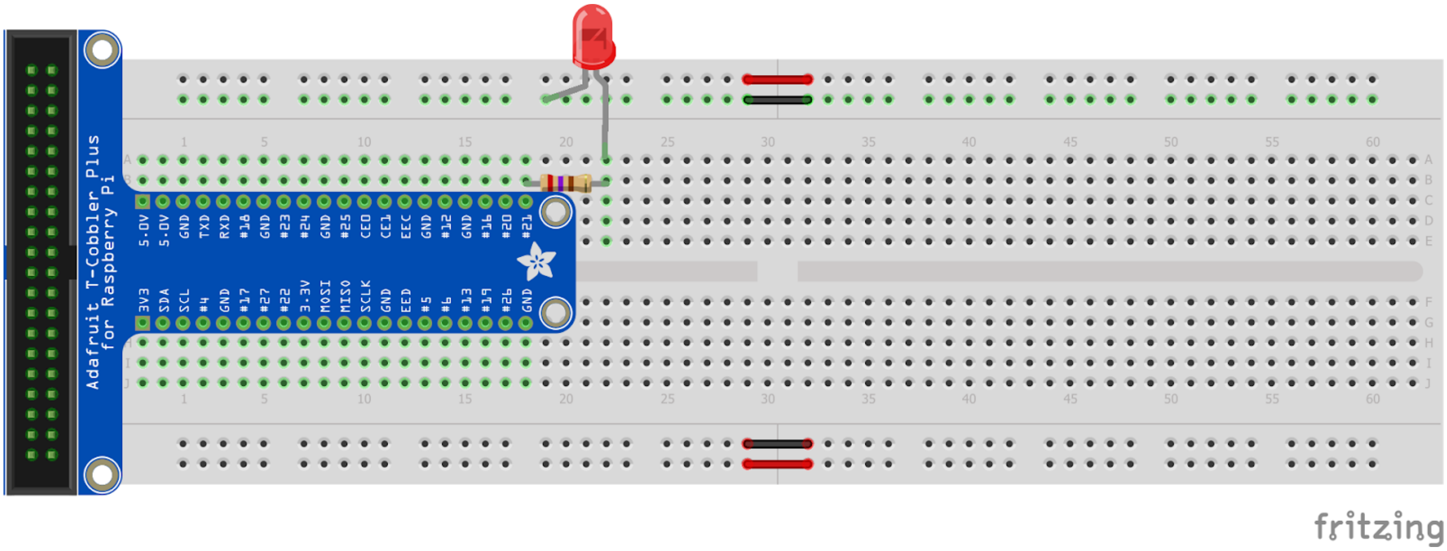


Raspberry Pi2 GPIO Header

Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I²C)	DC Power 5v	04
05	GPIO03 (SCL1 , I²C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)	(I²C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

figure 3.d

Wire up the breadboard as shown below. You'll be connecting the anode (round) portion of the LED to GPIO 21 of the Raspberry Pi. The cathode (flat) side of the LED will be connected to ground. Don't forget to add the resistor as well to limit the current drawn by the LED.



Now in the Thonny editor, create a new program file by clicking on the ➕ icon. In the program file add the following lines of code:

```
import RPi.GPIO as GPIO
import time

LedPin = 21    # GPIO21

GPIO.setmode(GPIO.BCM)       # Numbers GPIOs by name
GPIO.setup(LedPin, GPIO.OUT)   # Set LedPin's mode to output

while True:
    GPIO.output(LedPin, GPIO.HIGH)  # led on
    time.sleep(1) # sleep for 1 second
    GPIO.output(LedPin, GPIO.LOW) # led off
    time.sleep(1) # sleep for 1 second
```

Click on the ▶ button to run your program. Name the file something descriptive, like blink.py and click OK. If everything went according to plan, the LED on the breadboard should begin to blink once per second! Let's dive into the code to see what's happening.

In the first two lines of code we're introduced to the python import statement.

```
import RPi.GPIO as GPIO
import time
```

The *import* statement is a way for us to include other pieces of code, called modules, in our program. Each of these modules has its own set of functions which we can use in our program to make our code not only easier to read, but also much easier to maintain. You can think of each imported module as a toolbox, with tools designed to help you build different parts of your program. After the import keyword, the next thing we see is the name of the module you want to import. For instance, in line one we see that we want to use the module named Rpi.GPIO and we want to refer to that module in our code using GPIO. This module will help us setup and use the Raspberry Pi's GPIO pins. In the next line, line two, we are importing a module called time. The time module will give us access to really useful functions like timers that we can use to flash the LED at a specific time interval.

The next line of code, you already should be familiar with, at least a little bit:

```
LedPin = 21    # GPIO21
```

Remember in Lesson 2 we learned about variables, which are ways that we can store values in a program. LedPin is just a variable that we're using to store the value of the GPIO pin where our LED is connected, in this case GPIO21. By storing the GPIO pin value in a variable, we can use the variable name throughout our program. If we ever decide to connect the LED to a different pin, all we need to do is change the variable to the new pin number. This not only helps to make the code much easier to read, but as you'll see, much easier to change and update later on. One other thing you might notice is the "#" sign, called a hashtag, followed by GPIO21. In Python, the hashtag means that you want to write a comment in your code to make it easier to understand. Any text after the hashtag is considered a comment. In this case, we wanted to make it clear that we were setting the LedPin variable to GPIO 21, and not pin 21. Using comments throughout your code is considered good practice as it not only makes you able to understand what your code does if you look at it a year from now, but it helps anyone else who may read it understand how your code works.

Let's examine the next section of code:

```
GPIO.setmode(GPIO.BCM)       # Numbers GPIOs by name
GPIO.setup(LedPin, GPIO.OUT)   # Set LedPin's mode to output
```

Remember when we imported the RPi.GPIO module and that we wanted to refer to the module in our code with the name GPIO? Well, this bit of code is where we actually start using that module. To use a function in a module, you first type the module name, in this case GPIO, and then you type a "." and then the name of the function. In the first line of code we see `GPIO.setmode`. That means we want to call the setmode function that is inside the GPIO module. The setmode function just tells the program whether we're going to refer to the GPIO pins (GPIO.BOARD) by the pin number, or by the GPIO number (GPIO.BCM). Since we're using the GPIO number and not the pin number, we're passing into setmode GPIO.BCM. If this sounds confusing, refer back to figure 3.d above, which shows the pinout of the Raspbery Pi GPIO pins. Remember when you plugged in the LED into GPIO 21 on the Raspberry Pi? Now look at figure 3.d and see if you can figure out which pin number GPIO 21 refers to. If you came up with pin 40, you got it right! If instead of GPIO.BCM, we passed GPIO.BOARD into the setmode function, we could set our LedPin variable equal to 40 and it would do the exact same thing. Hopefully that makes sense now.

The next line of code, `GPIO.setup(LedPin, GPIO.OUT)`, sets the pin that will control our LED to an output pin. Remember that GPIO stands for General Purpose Input Output. Setting a GPIO as an input means that we want to read the state or voltage that's applied to that pin in our program. This could be useful if you want to hook up a sensor of some sort to your robot later on. Say for instance, if we wanted to hook up a motion sensor, to detect when someone walks by your robot. We would setup the GPIO pin that the motion sensor is connected to as an input using GPIO.IN. Then when motion is detected, the motion sensor would make the GPIO pin go high(3.3 volts), and we could then read that inside our program to tell whether motion has been detected. But for blinking our LED, we set the GPIO pin to an output, since we want to control the LED by turning on and off the voltage on the pin.

Lastly, let's look at the final segment of code in our blinky program:

```
while True:
    GPIO.output(LedPin, GPIO.HIGH)  # led on
    time.sleep(1) # sleep for 1 second
    GPIO.output(LedPin, GPIO.LOW) # led off
    time.sleep(1) # sleep for 1 second
```

We've come to the section of code that actually makes our LED blink! The first line `while True:` creates something called a *while loop*. Loops will be covered in their own lesson, but for now just know that they can be useful in programming because they allow you to have your program repeat something over and over. This while loop is basically saying "while true, repeat the below section of code". Since this statement can't be false, the loop will run forever. It's almost like saying, "while the earth is round, do this".

Now let's look at the code block inside the while loop. The first thing to notice is that the code under the while True statement is indented, meaning it has spaces before each line. These spaces are required in Python to let the program know that the code is part of the while loop. In the next line `GPIO.output(LedPin, GPIO.HIGH)` we see that we're calling another function from the GPIO module. You can probably guess what this code does, just by looking at it. The GPIO.output function will set a pin to either high (3.3 volts) or low (0 volts). The first argument to the output function is the pin that you want to set. The second argument is the state that you want to set the pin (high or low). So for this line of code we're setting our LedPin to high or 3.3 volts. This is equivalent to when we pushed the button in our previous breadboard blink example. We are completing the circuit through the LED and allowing the LED to turn on. The next line of code `time.sleep(1)` tells the program to sleep, or wait for one second. Remember time was the other module that we imported at the top of the program. The next two lines are nearly identical, except you'll notice that we're setting the LedPin to GPIO.LOW. This means that we're turninG off the LED. And then lastly we're telling the program to sleep another second before continuing on with the loop. So to summarize the code in our loop, it's saying: turn the LED on, wait one second, turn the LED off, wait one second. That code repeats forever, making our LED blink until we stop our program. Pretty cool! Feel free to play around with the code. See if you can make the LED blink at different time intervals. How would you make the LED blink for shorter than one second? Hint: time.sleep can take decimal numbers. Can you get the LED to blink with any other GPIO pins?

.

Lesson 4

Drawing With Roomba

We've had some fun learning the basics of programming and blinking an LED, now let's step up our game and teach our robot how to draw! Your robot comes equipped with a special port where we can attach a marker. The marker is placed precisely between the two drive wheels, allowing us to control the marker by programming the robot to drive in specific patterns.

The roomba_pi.py module contains 4 primary drive functions that we can use to help us steer our robot around the sheet of paper. These functions are forward, backward, left, and right. The forward and backward functions take a single parameter, distance in inches, that you want to robot to travel. The left and right functions also take a single parameter, degree of rotation, that you want the robot to turn. Figure 4.a shows a chart that you can use to gauge how far roomba will turn given a certain degree. For instance, if you wanted roomba to turn around so that it was pointing in the opposite direction you would pass 180 as a parameter to either the right or left function. To have roomba turn completely around one time, pass 360 degrees, etc.

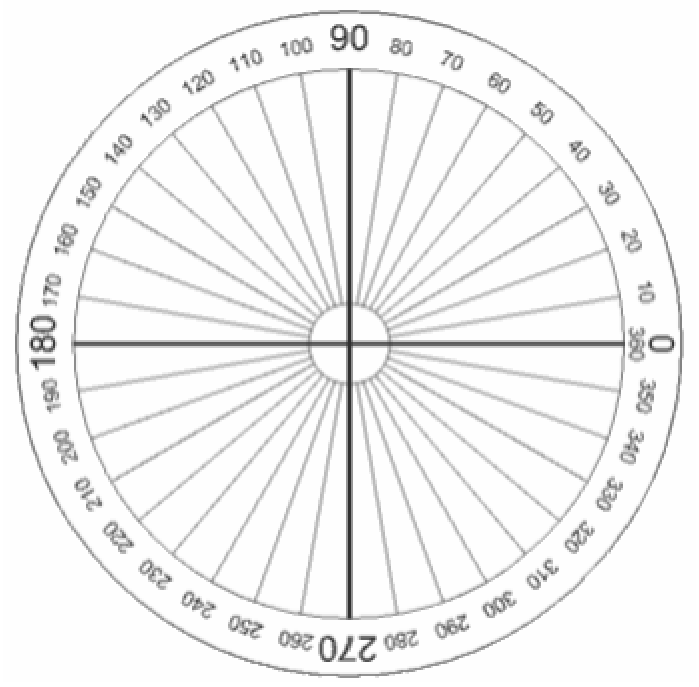


figure 4.a

Let's experiment a bit with these drive commands to see how they work. Remember from an earlier lesson we talked about the Python Shell. The Shell is an area where you can write and interact with code without needing to create a new program file first. It's helpful to use the Shell when you want to test a new code idea or play with a new module you're unfamiliar with.

To start with, make sure the Shell is open in the Thonny IDE by clicking on The View menu and then selecting the Shell option from the drop down. You should see the Shell window appear in the IDE. Click inside of the Shell window and type the following line of code and click enter:

```
import roomba_pi
```

This will import our roomba_pi control module, which will help us control roomba by typing in simple drive commands. Next type in the following line and press enter:

```
bot = roomba_pi.Robot()
```

This line will startup our robot and get it ready to start accepting commands. We can now use our bot variable to call the rest of the commands from the roomba_pi module. Next is the fun part, make sure the Roomba is away from obstacles and type the next line of code and press enter:

```
bot.forward(10)
```

You should see your robot move forward 10 inches. Now try running the next line of code:

```
bot.right(360)
```

Roomba should spin completely around. Now that you know how to use the Shell and run some basic commands, practice controlling Roomba to move around the room. Remember there are also the left and backward commands to experiment with. If you want to experiment with other functions in the module, or if you forget what commands are available, you can type `bot.` and then hit the Tab key on your keyboard. This will bring up a list of all the possible functions that belong to that module. Simply select the one you want and click enter to use it. **Note: some of these functions may have unintended consequences so proceed at your own risk!**

Now that we have some basic drive routines, let's see if we can make roomba drive in a square. What are the steps we need to follow in order to drive the roomba in a perfect square? I find it helpful sometimes to draw out the shape I want to program and write down the functions I'll need to call. If you are more of a visual learner, this might help you as well. Refer to figure 4.b as an example of what I mean. Imagine that Roomba starts out at one of the corners of the square. What is the first thing we want Roomba to do? From our drawing, we see we want Roomba to go forward, then turn right 90 degrees, go forward again, turn right 90 degrees, go forward, turn right 90 degrees and finally go forward, arriving at our original location. If you want to further experiment, you can practice typing commands into the Shell to see what commands you'll need to call in order to create a square.

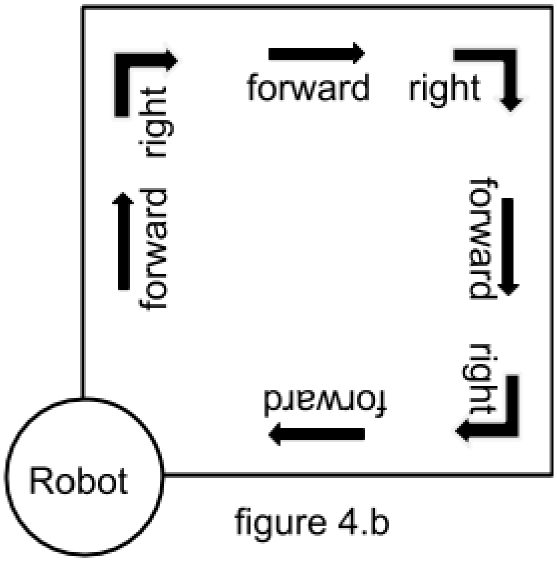


figure 4.b

Great, we have our routine written out, now it’s time to write some code to teach the Roomba how to draw a square. First, as we’ve done before, let’s create a new program and call it square.py. Import the roomba_pi module, which contains the control functions for roomba. Following the drawing we’ve made, let’s program our robot to move in a square. Below is what your program might look like:

```
import roomba_pi #the roomba_pi module is what we use to control the roomba

bot = roomba_pi.Robot() #startup/initialize roomba robot

bot.forward(5) #go forward 5 inches
bot.right(90)  #turn right 90 degrees
bot.forward(5) #go forward 5 inches
bot.right(90)  #turn right 90 degrees
bot.forward(5) #go forward 5 inches
bot.right(90)  #turn right 90 degrees
bot.forward(5) #go forward 5 inches
bot.right(90)  #turn right 90 degrees
```

Most of this code is probably pretty self explanatory. In line 1 we use the import statement to include the roomba_pi module, which we use to control our Roomba. In the next line `bot = roomba_pi.Robot()` we declare a variable named bot that we set equal to roomba_pi.Robot(). Robot is a special Python object called a *class.* We will talk more about classes later on, but for now you can think of a class as containing a set of functions. Our Robot class, when it’s created, will run some code that starts up Roomba and sets it up so that we can start controlling it. We will look closer at this code in another chapter, but for now you can just think of Robot as holding all of the functions (like forward, right, left, backward etc) that we will use to control our robot. By setting our variable named bot equal to roomba_pi.Robot(), we can use that variable to call the functions we need to control the robot.

In the next few lines we use our variable bot to call the functions we need to draw our square. According to our drawing, we first need to tell Roomba to go forward. As you recall from earlier in the chapter, the forward function takes a single parameter, which is the distance in inches you want to go forward. In this program, I chose to go forward 5 inches. Next we want to turn right 90 degrees to make the first corner of our square. To do this we call the right function, and pass in 90 as the degrees that we want to turn right. We just copy and paste these lines a few more times to complete our square. Try copying or typing this code into your square.py program. Run the program and see what happens! If everything went according to plan, you should see Roomba travel in a square pattern.

Now that we’ve successfully tested our program and know that it does what we expect, we can insert a marker into the marker port on the robot. Make sure the robot is placed in the middle of the paper you want to draw on, and run the program again. This time as the robot drives in a square shape, it will drag the marker along underneath it and trace out the square as it goes! Once the program finishes, remove the marker from the marker port and move the robot off of the paper. You should see a glorious square, the shape you taught the robot to draw, in the center of the page! Pretty neat, right? Now that you know the basics of how to teach your robot to draw, practice drawing other shapes besides squares. See what happens when you pass in different parameters into the forward and right functions. Experiment with the left function as well to make shapes that zig zag. Most of all have fun!

References

1. https://sydneypadua.com/2dgoggles/the-marvellous-analytical-engine-how-it-works
2. https://www.techspot.com/article/531-eben-upton-interview/
3. https://en.wikipedia.org/wiki/Logo_(programming_language)
4. http://www.newsdownload.co.uk/pages/RPiLM386AudioAmp.html
5. https://en.wikipedia.org/wiki/Light-emitting_diode
6.