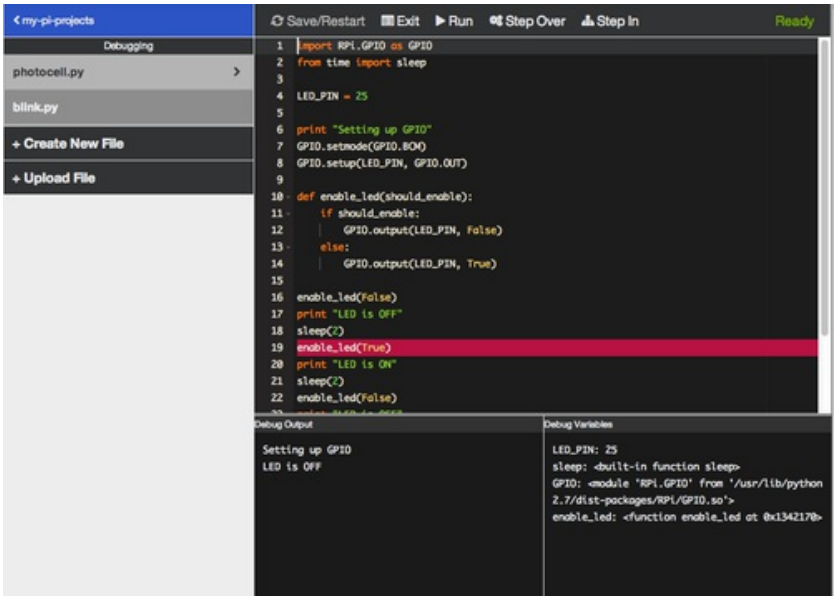
adafruit learning system

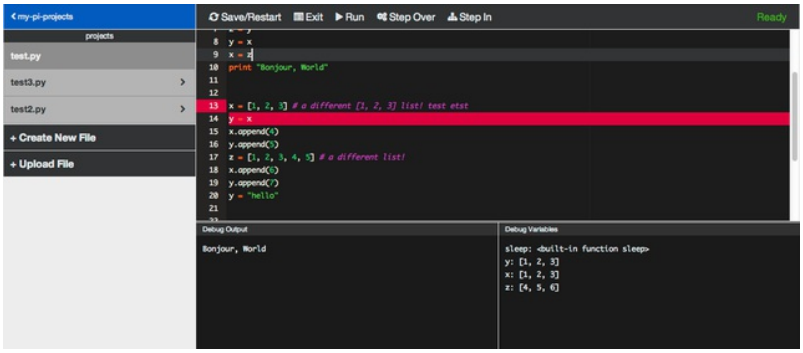
Debugging with the Raspberry Pi WebIDE

Created by Justin Cooper



Guide Contents

Overview



The Raspberry Pi WebIDE includes an advanced, yet easy to use tool, to help you work through code that you've downloaded or written in Python.

If you haven't used a debugger, either on the command line or in an Integrated Development Environment (IDE), hopefully this guide will help you understand why you'd want to do so, and how to effectively debug your code.

Installation and Setup

If you already have your Raspberry Pi setup with an Operating System and the WebIDE you can skip this section, otherwise, read on for how to get up and running.

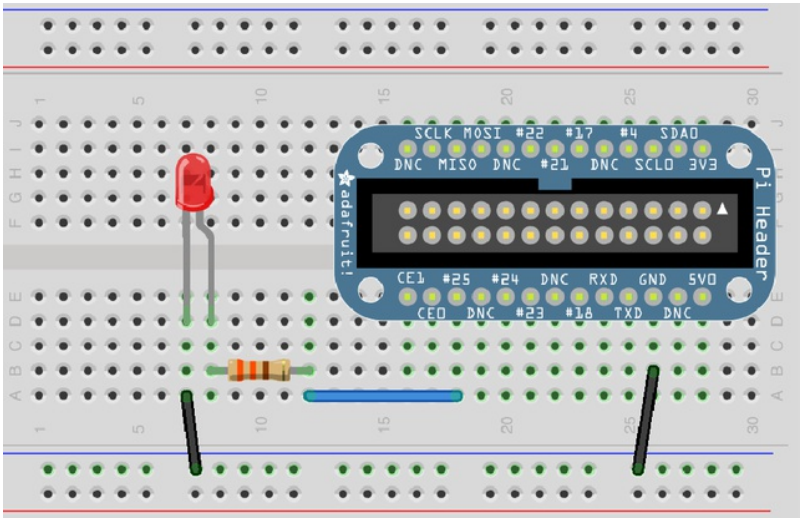
You'll first want to setup your Raspberry Pi with an operating system on an SD card. You can learn how to setup an SD card for you Raspberry Pi (http://adafru.it/aWq), if you haven't already done that. We also have an entire series on how to get started with your Raspberry Pi (http://adafru.it/aWr) that you may find useful if you're new to the the world of Linux and Raspberry Pi.

The next step to continue on with the tutorial will be to install the WebIDE. We have instructions for installing the WebIDE (http://adafru.it/aRr) as well. It's a fairly straightforward, and quick process to get the WebIDE installed.

Once you have both of those components setup, and installed, test that your WebIDE works by going to http://raspberrypi.local (http://adafru.it/aWs) on any computer on your local network.

Debug a Blinking LED

Now that we have everything setup, let's start with a simple debugging session. We have a red LED that we've wired up, but for some reason it won't blink correctly when we run our Python code. We'll step through it, and try and figure out what went wrong.



The above schematic is simply using pin #25 with a resistor between 330 to 1000 ohms. If you don't have a Raspberry Pi Cobbler, you can match up to the equivalent pins on the Raspberry Pi.

Before we copy the code, let's setup a project in the WebIDE for this guide.

First, let's create a new "Debugging" project within the "my-pi-projects" folder.

Next, let's copy and paste the below code into a newly created "blink.py" file in that "Debugging" project.

```
import RPi.GPIO as GPIO
from time import sleep

LED_PIN = 25

print "Setting up GPIO"
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED_PIN, GPIO.OUT)

def enable_led(should_enable):
 if should_enable:
  GPIO.output(LED_PIN, False)
 else:
  GPIO.output(LED_PIN, True)

enable_led(False)
print "LED is OFF"
sleep(2)
```

```
enable_led(True)
print "LED is ON"
sleep(2)
enable_led(False)
print "LED is OFF"
sleep(2)
enable_led(True)
print "LED is ON"
sleep(2)
GPIO.cleanup()
```

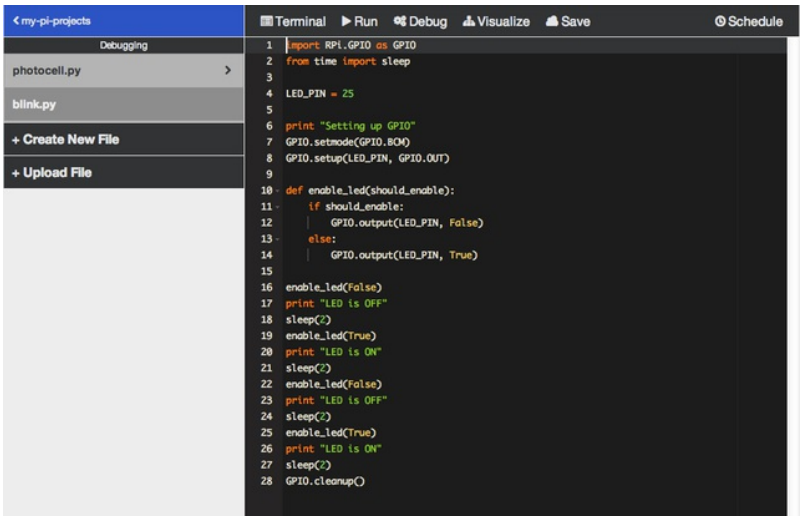
The code is pretty straightforward. We want to turn off the LED, then turn it back on, with a 2 second break in between, and the print statements should match what the LED is doing.

We first import the required libraries. We're using GPIO pins, so we'll need GPIO, and the python sleep library to slow things down a bit at times. Next is setting up the GPIO for the LED. Starting on line 10 is our enable_led method that either turns the LED on or off based on the value of "should_enable". After that, starting at line 16, is just turning the LED on and off, with the print statements, and sleep to slow things down.

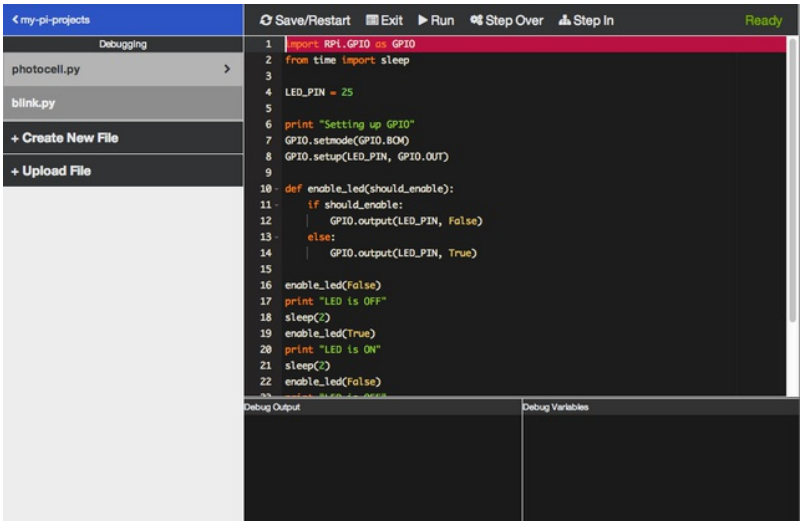
For some reason, the output from our print statements isn't matching what the LED is doing during the sleep statements.

Let's fire this up in the debugger, and see if we can find out what's going wrong.

When the file is open in the WebIDE, click the "Debug" link to activate the Debugger.



Once the debugger initializes, and is ready to go, it should look like this:



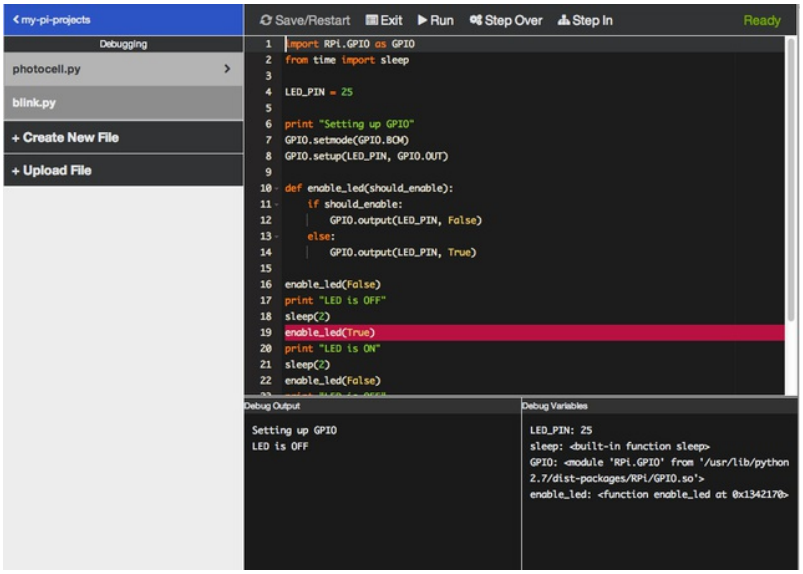
With the red debug line active, and "Ready" in the upper right of the screen, we're ready to start debugging.

This is a rather simple program, so what we can do is just click "Step Over" for each line in our program. As we click "Step Over", the code that has the red line over it will execute, and the red line will continue on to the next line, waiting for your next instruction.

In the below screenshot you can see that we're now on line 19, and when we click "Step Over", line 19 will execute, and the red line will continue on to line 20.

An interesting thing to note here is that if you were to click "Step In" on line 19, it would then jump up into the enable_led function, and allow you to step through that function until it returned.

So, if you have a bug in one of your functions, you may need to occasionally "Step In" to one of your functions to see why it's behaving a certain way.



Ok, now you can step through the entire program.

You should see Debug Output "print" statements as you step through, letting you know what the LED should be doing at any given moment. You can also see the Debug Variables output in the lower right. You can see that the LED_PIN is set to 25, and then you can also see we're using the sleep and GPIO imports.

Now that we've gotten this far, we can start narrowing down where we're having issues with our particular code.

When you stepped over line 16, and then line 17, you were expecting the LED to be off. We can see that we are intending on having the enable_led(False), so that line looks good. This is where "Step In" would be useful, as it looks like the issue is in our enable_led function.

It looks like we've mixed up some True and False statements in the enable_led function, so if we switch those around, and then click "Save/Restart", we should be ready to test this out again. You can either step through the program again, or just click "Run", to see if your program does what you'd expect it to.

> Any time you edit the code while debugging, you need to Save/Restart so the debugger resets with your changes.

Here is the fixed, and working code based on our debugging efforts:

```
import RPi.GPIO as GPIO
from time import sleep

LED_PIN = 25

print "Setting up GPIO"
GPIO.setmode(GPIO.BCM)
GPIO.setup(LED_PIN, GPIO.OUT)

def enable_led(should_enable):
 if should_enable:
  GPIO.output(LED_PIN, True)
 else:
  GPIO.output(LED_PIN, False)

enable_led(False)
print "LED is OFF"
sleep(2)
enable_led(True)
print "LED is ON"
sleep(2)
enable_led(False)
print "LED is OFF"
sleep(2)
enable_led(True)
print "LED is ON"
sleep(2)
GPIO.cleanup()
```

This may seem like a pretty basic example, but when your code gets more complex, it can get really tricky to find out why the LED isn't turning on, or why you're not able to output the correct

values to an LCD at any given time.

Stepping through your program, and viewing the output, and variables in real time is a pretty powerful tool that you can add in to your code/test cycle.