

Saturdays at Stevens

Version 0.1

Gregg Vesonder

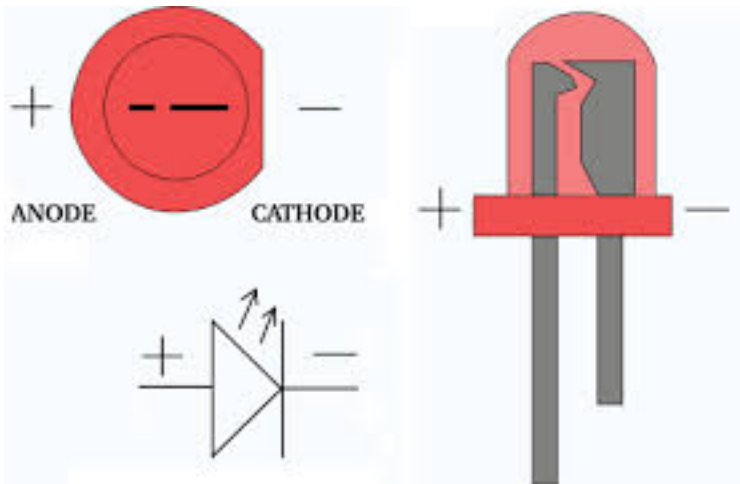
Ye Yang

Welcome!

These Labs use a Raspberry Pi single board computer and a breadboard. There are many kits available, many for under \$100. For a listing of resources, code and this document, visit the web site at <http://aarphacker.com> . I hope you enjoy these Labs and please send comments to vesonder@mac.com.

Lab 0: Throwie!

Let's get a bit familiar with electronics. For some of you this may be the first electronic circuit you make! The kit for a throwie consists of a 10mm LED, a CR2032 disk battery and some tape. Note that the LED has two leads of unequal size. The long lead is called the Anode and the shorter end is the cathode. Place the longer end on the + side of the disk battery and the cathode on the - side. It should look like this:

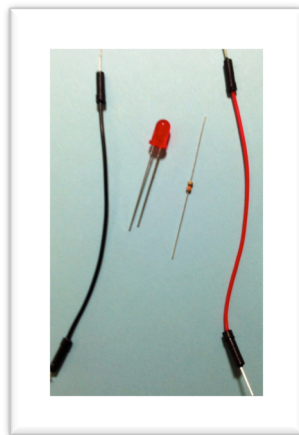


Now take a strip of tape and wrap it around the battery and leads, attaching the leads to the battery. Your LED should now be glowing. Congratulations you have finished your first project.

Why is it called a throwie? Well folks have gone a step further and wrapped a magnet with tape around the battery. Once the magnet is attached you can throw it on a metal surface and it sticks creating some impressive lighting effects on metallic surfaces. On non-metallic surfaces you can attach it with sticky tack or poster tack. The LED should stay lit for over a week, depending on the quality of the battery. I have had them last 2+ weeks.

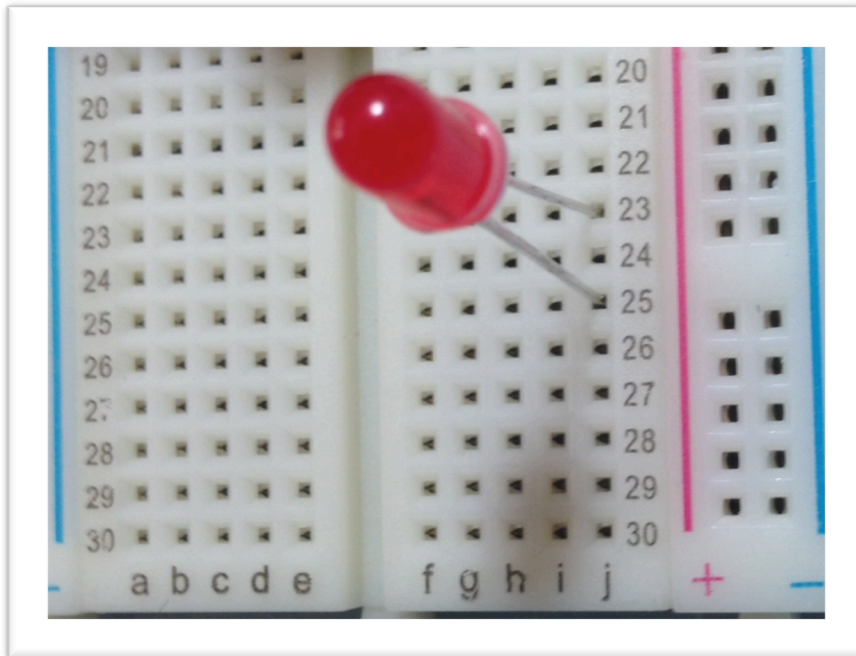
Lab 1: Light!

One of the popular examples to do on a Raspberry Pi or raspi is to make a LED (Light Emitting Diode) blink. This requires wiring a breadboard connected to the raspi with a LED and accompanying resistor, writing the code and running it. However first we are just going to light up a LED to understand the circuit. Although basically the same result this circuit is more complicated than the throwie.

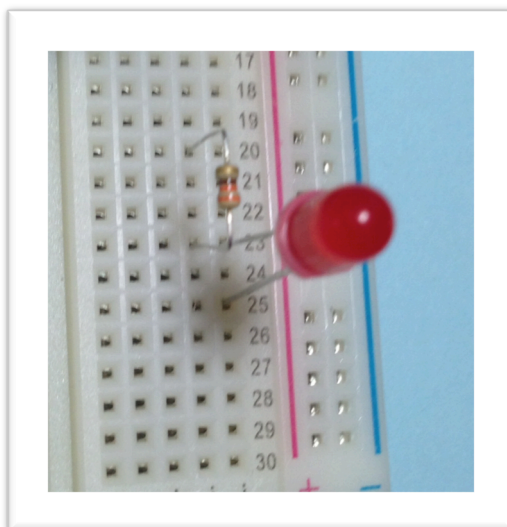


- 1) Parts list:
 - LED
 - Resistor under 1000 ohms
 - 2 jumper wires, 1 black and 1 red
- 2) Make sure the raspi is disconnected from the power.
- 3) In the parts list picture you will notice that the LED has two leads and that one is longer than the other. That is the positive lead and, by elimination, the shorter is the negative lead! If some prankster, snipped both leads to the same size you could look at the bulb and see that one of the leads expands to a squarish area, which is attached to the negative lead. Another sign is that usually the lower ring on the LED bulb is flat on the negative side.
- 4) Take the led, and on the fghij rows, place the positive element in row 23, column j and the negative element in (25, j) – note from now on I will use this

(row, column) notation. It should look like this:



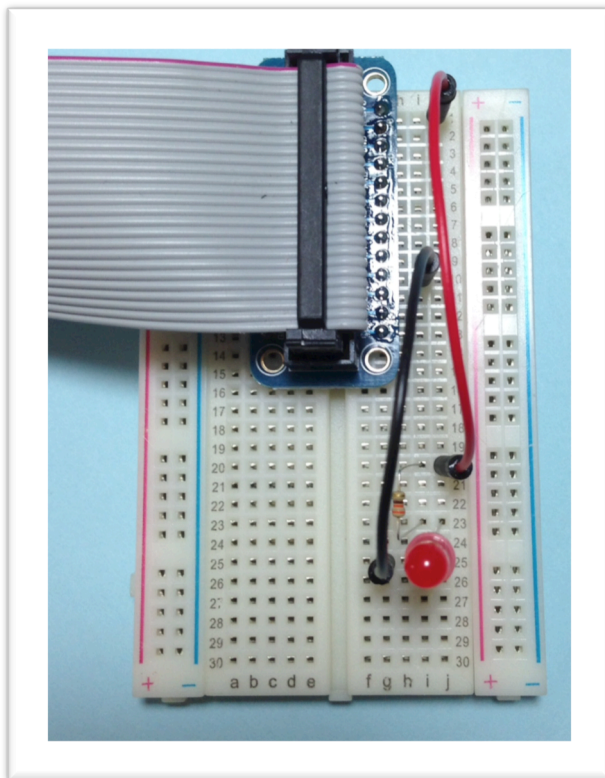
- 5) Now it is time to add the resistor. In the picture above note that each row has five slots. Each of these slots is electrically connected to each other. So if I want to attach something to the positive lead I can use the slots in row 23 f-i to connect them. That is what makes breadboards special allowing experimenters like us to make quick connections. I am taking advantage of this to connect the resistor. First I bent the leads of the resistor to make it easier to insert into the board (yours are already bent). Place one of the leads, either will work, in (23,i). Place the other end of the resistor lead in (20, i). Your board should resemble this:



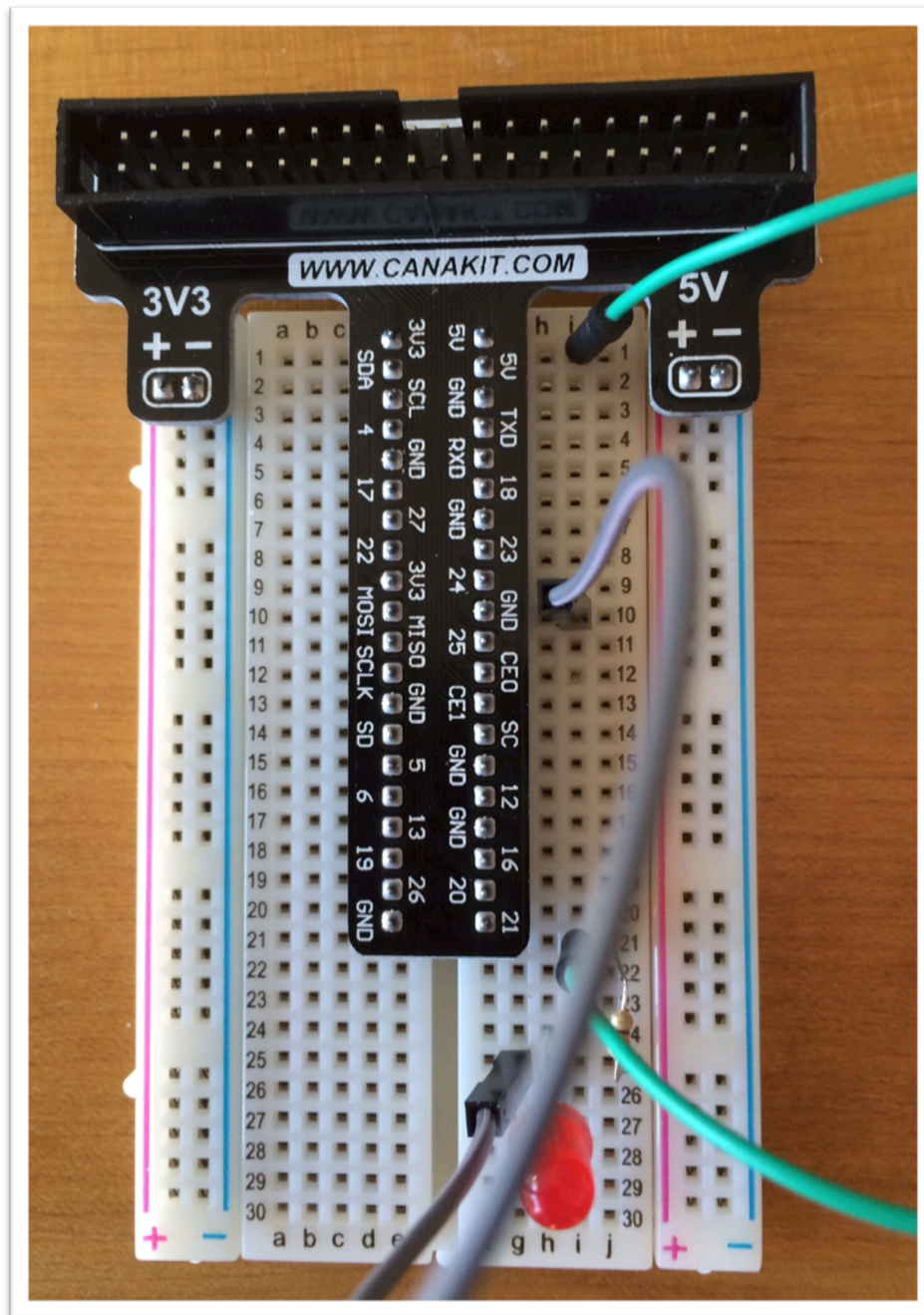
- 6) On resistors. The resistor we are using is 330 ohms. The bands on the resistor represent the value of the resistor. This resistor has bands in order:

orange, orange, brown and gold. Orange stands for 3 and brown, when in the 3rd position, stands for x10. So the value is 33 x 10 or 330 ohms. (the bands are aligned so that the gold or the silver band is at the far end). Why did we use 330 ohms. Because that is a safe value for this LED. The resistor lowers the current flow so we do not burn up the LED. It depends on a few factors beyond the scope of this class but is worth exploring on the web.

- 7) Attach the jumper wires. By convention the black wire always links ground to the negative lead. Place one end of the jumper wire into a slot on row 25, connected to the negative lead of the led. In the picture it is attached to (25,g). Insert the other end into a ground slot of the interface board, row 10. The interface board “extends” the interface pins of the raspi to the breadboard. I selected the ground slot, labeled GND on the right hand side of the board, represented by slot (10,i). If you look carefully you will see GND labeled at that slot on the board. Next attach the red wire to power, in this case 5 volts. Insert the red jumper in row 1, in the picture it is (1,j). or or



or like this if you have a B+



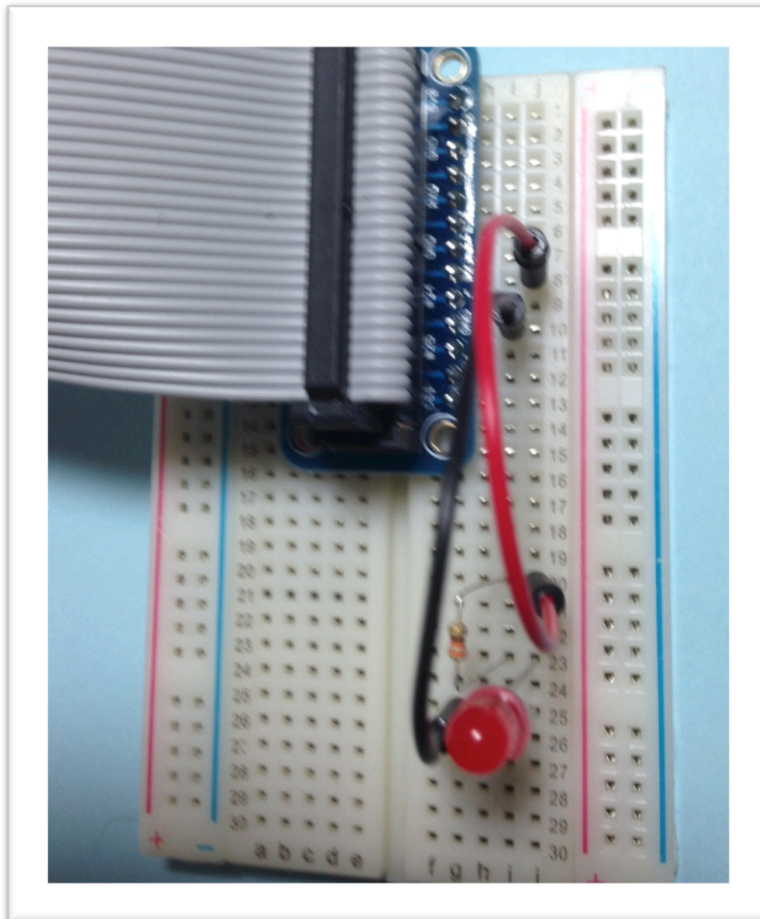
- 8) Time to test it! First, **please have your instructor check your wiring.** After the instructor checks your wiring power up the raspi by inserting the micro usb connector into the raspi. The LED lights! Congratulations you have wired your first electronic circuit! Unplug the raspi and on to your next adventure.

How was this different from the throwie?

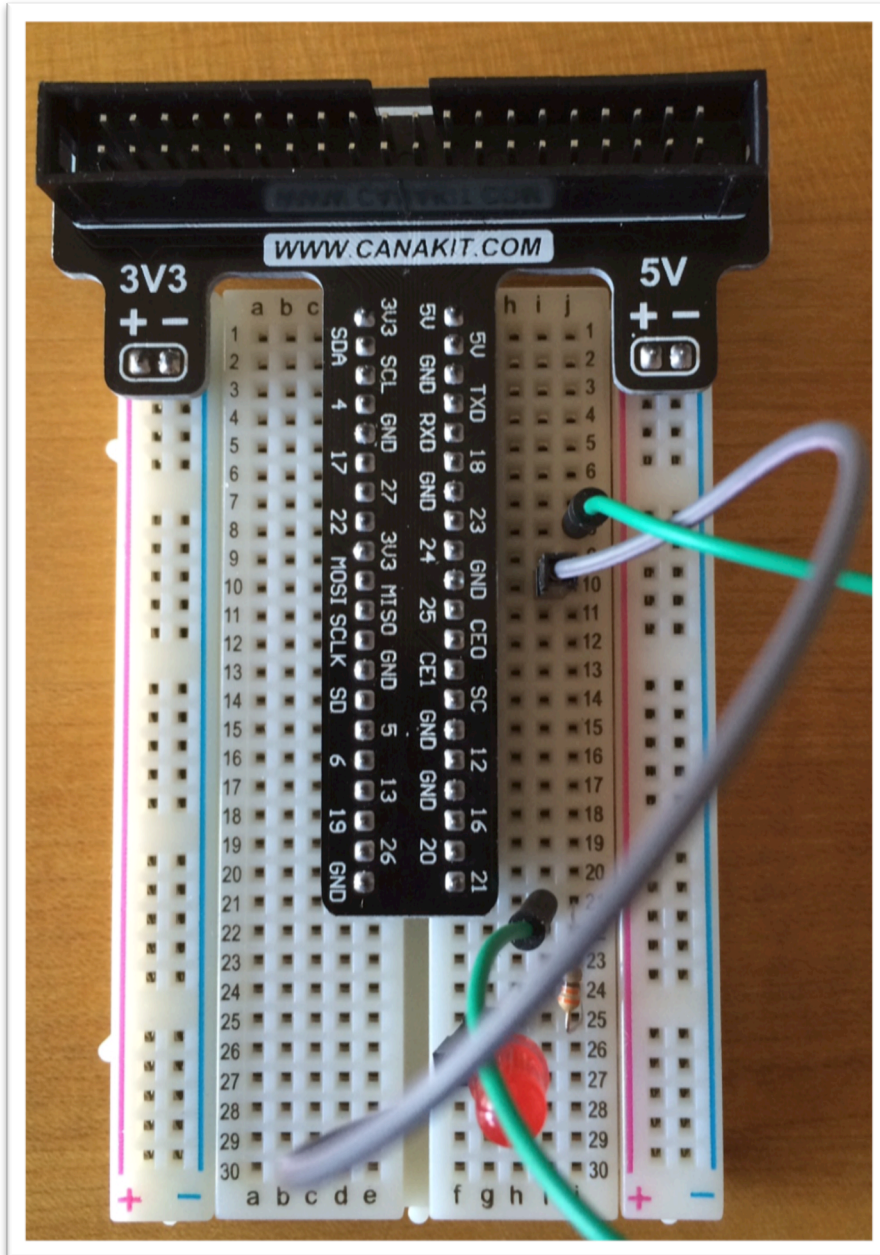
Lab 2: You Blinked!

In Lab 1 the raspi was used merely as a power source. We used none of its ability to actually control circuits. In this Lab you will program the LED to blink. There is a lot going on in this Lab, it provides the foundation for the rest of the Labs. First let's build the circuit we will use. From a coding perspective we will be working with variables and loops.

- 1) The parts are the same as the first Lab: a 330Ω resistor, an LED and a red and a black jumper wire. Ω is the symbol for ohms, the resistance value of a resistor that we discussed in Lab 1.
- 2) Wire the circuit exactly as you wired it in Lab 1 with one exception, instead of placing the red jumper leading from the resistor to (1,j) place the end to (8,j); this connects the power to a pin that can be controlled through the raspi by a program. Your board should look like this:

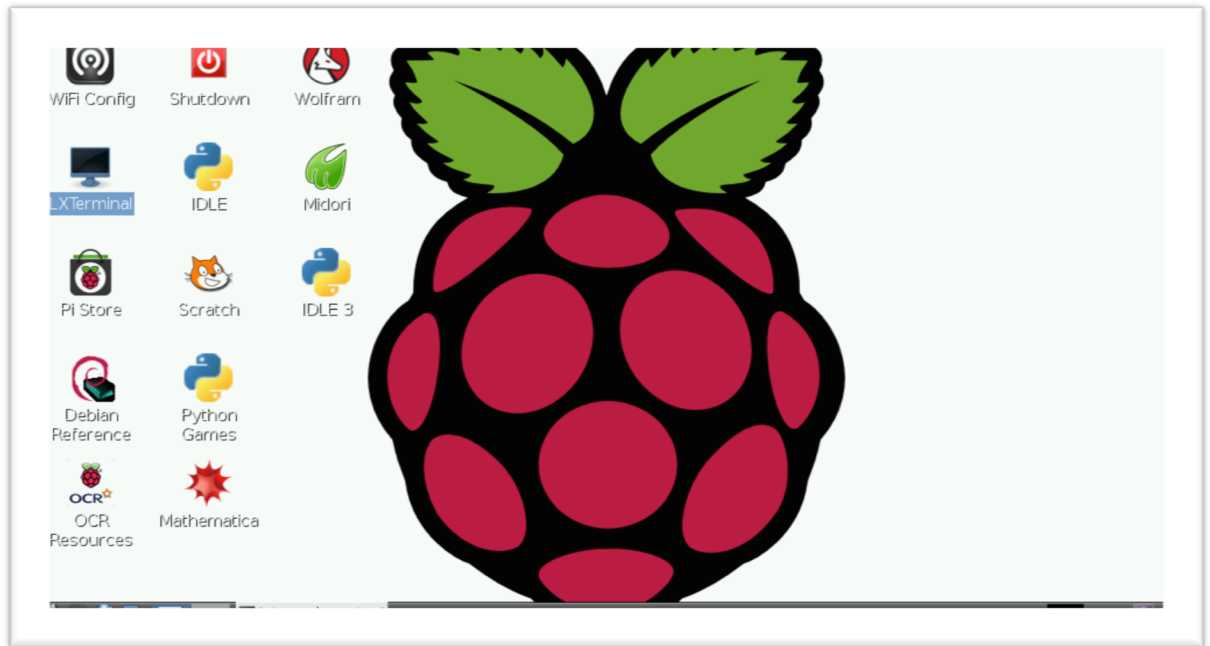


- 3) Or like this if you have a B+!



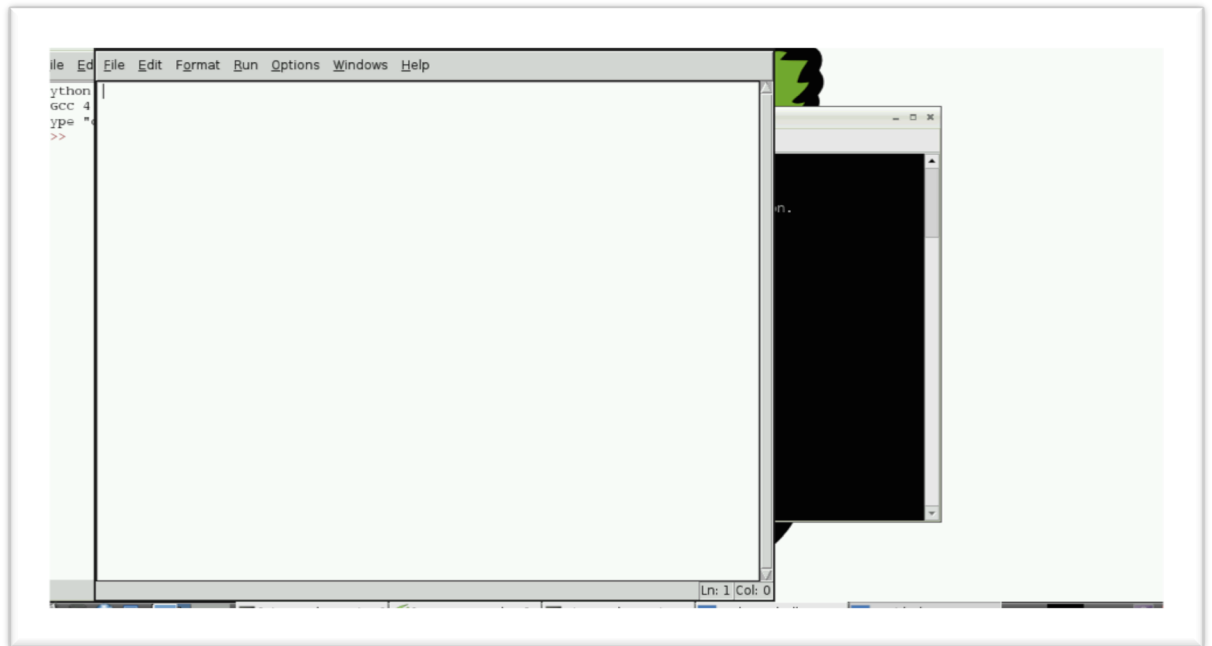
- 4) Have your instructor check your board. After the instructor checks it, plug in the power supply and wait for the computer to start. It will take a minute or so.
- 5) If everything goes fine you will be taken into the raspberry pi window system. However if you get a login prompt, type `pi` – note that whenever there is an entry on the screen I will use `this` font. You will now be prompted for a password. Ask your instructor for it if the instructor has not told you. If successful you should get the prompt: `pi@raspberrypi ~$`. At the prompt type `startx`. This command will take you from the command line interface of linux to the window interface of linux. In either case your

screen should look like this:

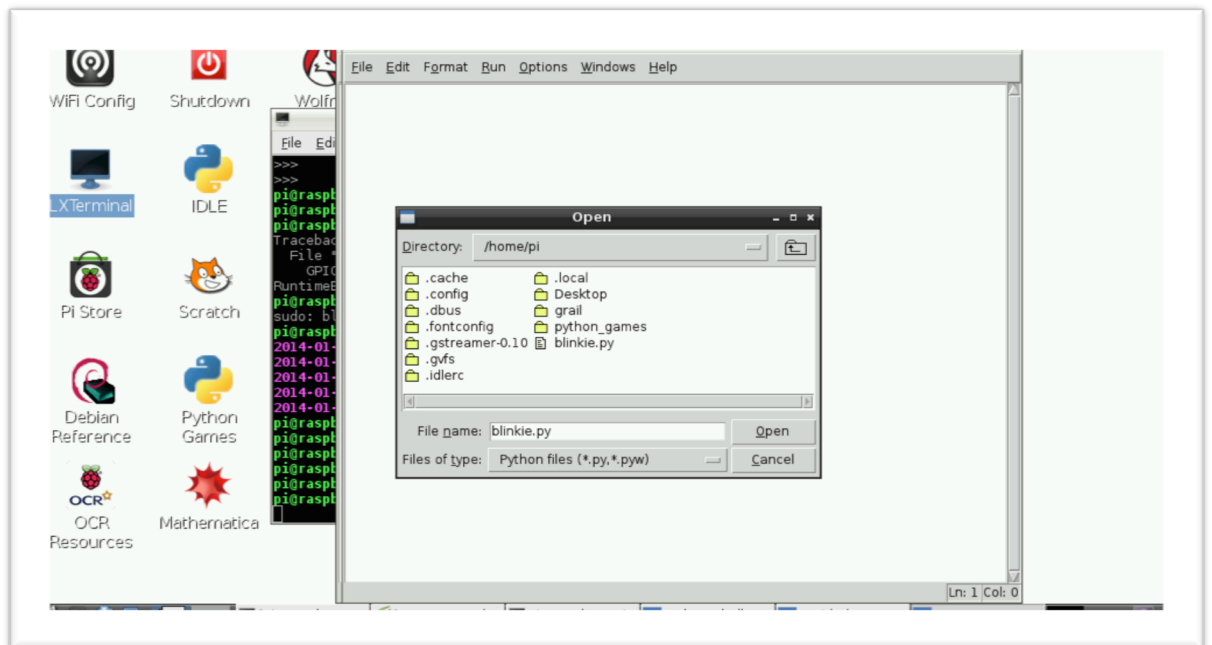


- 6) Time to start the programming environment. For all of our Labs we will be using the Python language and the IDLE programming environment. The Python language is a popular language that is used by students, educators and professional developers. Python was not named after a snake but after a British comedy group, Monty Python. One of their movies, *Monty Python and the Holy Grail* is on my top ten list. Back to programming. Point the mouse to LXTerminal, and double click quickly on it. This takes you into a command line window. Type `sudo idle` at the prompt and this will launch the Python programming environment. Point the mouse at the file menu item at the top of the window, click and then select “New Window.” This will open an editor in the IDLE programming environment. If everything goes right

your screen should now look like this:



- 7) In order to save time, a partial version of the program blinkie.py has been prepared for you. Load it now by moving the mouse to the file menu and selecting “Open”. Choose blinkie.py, and your screen should look like this:



Select open and the program will display in a simple editor. Your task is to code two lines at the end of this program in the for loop. In the first line you must replace the question marks with instructions. You must write the complete second line, replacing the question marks on that line. Please note that Python is white space sensitive. The indentation of lines following the

`for` command indicates that those lines belong within the `for` loop. When you think you have completed the program, ask your instructor to check it and then “Save” the file using the file menu and after the save completes, select the Run menu and choose “Run module”. Watch the blinking light. Congratulations, with this newfound skill you are ready to control the world through the raspi!

A bit on the program

The first part of any code is a description of what the code does. In Python `#` indicates that what follows on that line is text and is not to be interpreted as code. After describing the program, additional modules necessary for the operation of the program are loaded through the `import` command. The module can be renamed with the `as` command. The module `RPi.GPIO` renamed as `GPIO`, provides the program access to the pins on the computer that were wired on the breadboard. The module `time` provides the `sleep` command that permits the code to do nothing for a specified time, so that the LED will stay on or off for that interval.

The next section of the program sets up the environment for the blinking led. First set a variable, `repeat`, that indicates how many times the LED should blink. The next two lines talk about the code’s interface to the breadboard. `GPIO.BOARD`, indicates that we are identifying the pins by the physical location of the pin on the interface board, not the name that the raspi board uses, which would be `#23` and is labeled as such on the interface board. Given that the next line indicates the pin, `16`. It is `16` because, facing the board the numbering starts at the upper left corner of the interface board as `1`, the upper right corner as `2` and continues to alternate. Therefore all odd pin numbers are on the left hand side and even pins on the right hand side. The jumper wire was inserted in `(8,j)` which counting is `16`. An easy way is just to double the row number to get the addressable pin number. Since the pin will be used to light the LED it is an `OUT` port. If the port was used to collect a button press that the software would read it would be an `IN` port, since it would input data to the program.

This brings us to the `for` loop. The `for` loop is a way to repeat a set of commands for a specified number of times. As you learn Python, you will discover that there are many wondrous ways to specify this, but the current code uses a basic count variable which we set as `repeat`. Loops and other Python mechanisms end their description with a `:`. Usually that `:` will be followed by indented lines. The indentation indicates that those lines are part of the body of the `for` loop. The first line in the body of the `for` loop turns on the LED by setting the pin to `HIGH`. `time.sleep` then sleeps, i.e., does nothing for the interval specified within the parentheses which indicates the number of seconds to sleep. The next two lines you contribute. What do you want to do next? Turn off the LED and wait. So if the LED is turned on by setting it to `HIGH` then turn off the LED by setting it to `???`. It should then stay off for a time before it begins the next loop. The last command ensures that the program ends with all pins in a default state. Phew! Here’s the code:


```

#
# blinkie.py, a program to blink an led
# based on a program by Rahul Kar
# http://www.rpiblog.com
#
# add some essential modules
#

import RPi.GPIO as GPIO
import time

#
# GPIO provides access to the board pins
# from the program.
# time provides us with a sleep capability
# so that we can add a delay
#

##
##declare repeat
##

#
# repeat indicates how many times to blink
#

#
#setup board access
#

GPIO.setmode(GPIO.BOARD)
GPIO.setup(16, GPIO.OUT)

##
##do a for loop with repeat
##
    GPIO.output(16, GPIO.HIGH)
    time.sleep(1)
    GPIO.output(16, GPIO.LOW)
    time.sleep(1)

#
#cleanup
#
GPIO.cleanup()

```

Lab 3: Help!

In this Lab we will use all we have learned to this point to send a distress message to the world. An international indicator of distress is SOS, the acronym has been described as an abbreviation for “Save Our Souls”, “Save Our Ship” or even “Send Out Succour”! More on the history of SOS can be found in wikipedia. Our task is to use the LED to send out a constant SOS to the world. The way we will do this is to vary the duration of the blinks and that can be used to send Morse Code.

From a coding perspective we will be introduced to a new looping construct while, the logical value True, functions, strings and printing. That’s a lot but with these tools you are well on your way to learning python!

Morse Code transmits information as a series of on-off lights, tones, clicks of a certain duration. It has been around since the mid 1800s and is still in use today. Again Wikipedia has an excellent article on its history.

Letters in Morse Code are sent as a series of dots (sometimes called dits) and dashes. A dash is 3 times longer than a dot. Letters are a collection of dots and dashes. The morse code for a S is 3 dots, and for an O is 3 dashes. A dash is 3 times longer than a dot. There are also specific pauses between dots and dashes, letters and words to indicate the various separators. This table summarizes the relationship:

Element	Units of duration
dot	1
dash	3
Pause between dot or dash	1
Pause between letters	3
Pause between words	7

So our goal in this program is to have the board, through the led, send out a continuous SOS signal.

- 1) First check the board. If you have not done so wire the board as in Lab 2.
- 2) Now start a new IDLE session by typing `sudo idle` at the command line. When the IDLE screen appears, select “File” with your mouse and then select “New Window”. When the Window appears, select File and then Open with your mouse and load the `sos.py` file.
- 3) In this programming task you must code the S, O and SOS Python functions. Ask your instructor if you are having difficulty.
- 4) When you think you have completed the program, ask your instructor to check it and then “Save” the file using the file menu and after the save completes, select the Run menu and choose “Run module”. Now just wait until help comes to Save Our Ship!

A bit on the code

This code builds on the code of `blinkie.py`. The new thing introduced is functions. A function provides a way to capture code that is used frequently. A function should represent one task. In this case our tasks were building dits, dashes, pauses, letters and a word, SOS. Functions can be built from other functions and this becomes a very powerful tool. In Python we define a function by starting with the special word `def`, then name the function (the name should be a clear indication of what it does) and then provide a list of zero or more arguments, items that you want the function to use. In this case we always passed `unit`, which was the duration of one sleep unit, later set to 0.1 or a tenth of a second. The identifying line of the function ends with a `:` signifying that the indented lines that follow belong with the function. `return` indicates the end of each of these functions. In Python you do not have to use `return` to end a function but I think it is useful, especially in Python, since otherwise the end is indicated by lack of indentation. `return` can also return values to a calling module but that is beyond this Lab!

One final aspect to this program is the `while True:` statement. This provides a way for us to loop until the program is interrupted. Stop the program by selecting "Exit" in the file menu. In effect the loop can go on forever since a while body loops if the condition is `True`. Since the condition in this case is `True`, it always loops. Using `while True` is a convenient way for waiting an indeterminate amount of time for input. In this case our rescue ship home! Here's the code for the Lab:

```
#
# sos.py
# program to do morse code
#
# import necessary libraries.
# GPIO accesses the ports of Raspi
# time provides us with sleep
#

import RPi.GPIO as GPIO
import time

#
# morse code info
# sos in morse code is:
# ... --- ...
# dashes are 3 times longer than
# dit. 1 unit space within a letter
# 3 unit space between letters,
# 7 unit space between words
#

repeat = 50
```

```

pin = 16
GPIO.setmode(GPIO.BOARD)
GPIO.setup(pin,GPIO.OUT)

#
# define dit (dot)
#

def dit(unit):
    GPIO.output(pin, GPIO.HIGH)
    time.sleep(unit)
    GPIO.output(pin, GPIO.LOW)
    time.sleep(unit)
    #pause between elements of a letter
    return

def dash(unit):
    GPIO.output(pin, GPIO.HIGH)
    time.sleep(unit * 3)
    #dash is 3 times dit
    GPIO.output(pin, GPIO.LOW)
    time.sleep(unit)
    #pause between elements of a letter
    return

def letter_pause(unit):
    time.sleep(unit * 2)
    # why 2? each unit has a pause
    # at end, just add 2 more
    # to make it 3.
    return

def word_pause(unit):
    time.sleep(unit * 6)
    # again accommodating
    # unit pause
    return

#
#
# Define functions s_letter,
# o_letter and sos
#
#
# for ever
#

while True:
    sos(0.1)

```

```
#  
#time to cleanup  
#
```

```
GPIO.cleanup()
```

Lab 4: Roll Them!

Dungeons and Dragons is a complex role playing game where a small group serves as a party of adventurers taking on roles such as a wizard and one person serves as the dungeon master controlling the flow of the game. The adventures take on challenging monsters and other challenges and fate is often determined by rolling a variety of dice. Your task is to create a program or programs that act as six sided, eight sided, ten sided, 12 sided and 20 sided dice. A picture of some D&D dice from Wikipedia is provided for motivation!



Lab 5: Was it a Dit or Dah?

One challenge with using Morse Code to send messages is that it depends on your potential rescuers differentiating between a dot/dit and a dash/dah. This is further complicated by the fact that if your raspberry pi is doing a lot of other tasks, sleep becomes less accurate and dots may be elongated to look closer to dashes. One way to combat that is to use redundant cues providing multiple ways to determine a dot or a dash. This Lab will do just that.

The wiring for this Lab will be different from the previous labs. The wiring of the first four Labs was simplified to ease the patient reader into electronics. The key simplification was that we were relying on the raspi control pins to provide the

power. Since in this Lab we will have more power needs, we are shifting to a more conventional wiring of our breadboard with a “separate” power source. We are using the power pins from the raspberry pi to provide us with power. The raspberry pi has two sources of power, 3.3 volts and 5 volts. This should be clear from this diagram from http://elinux.org/RPi_Low-level_peripherals:



and for the B+ from

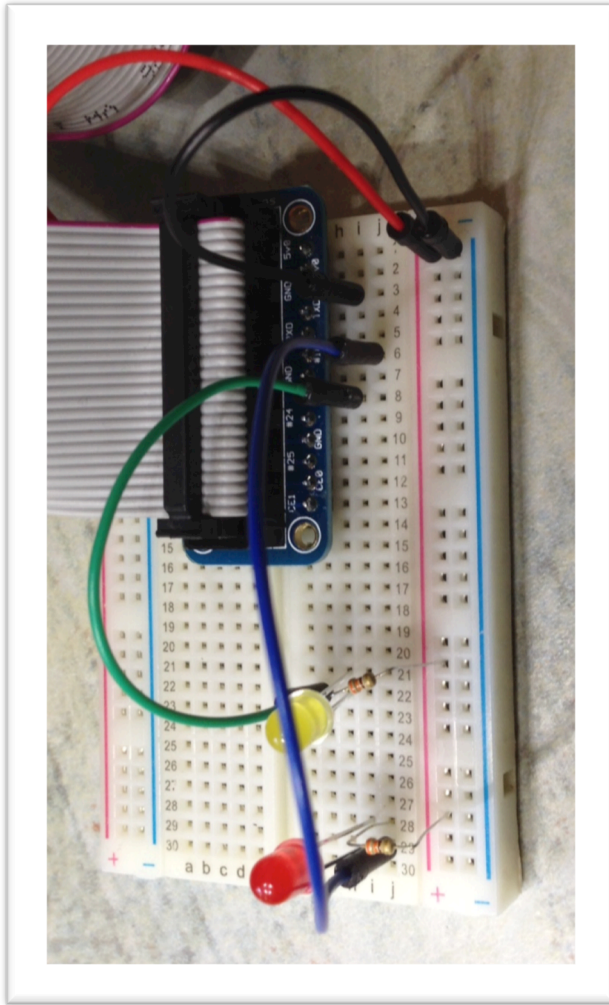
<http://www.raspberrypi.org/forums/viewtopic.php?f=78&t=82397>.

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

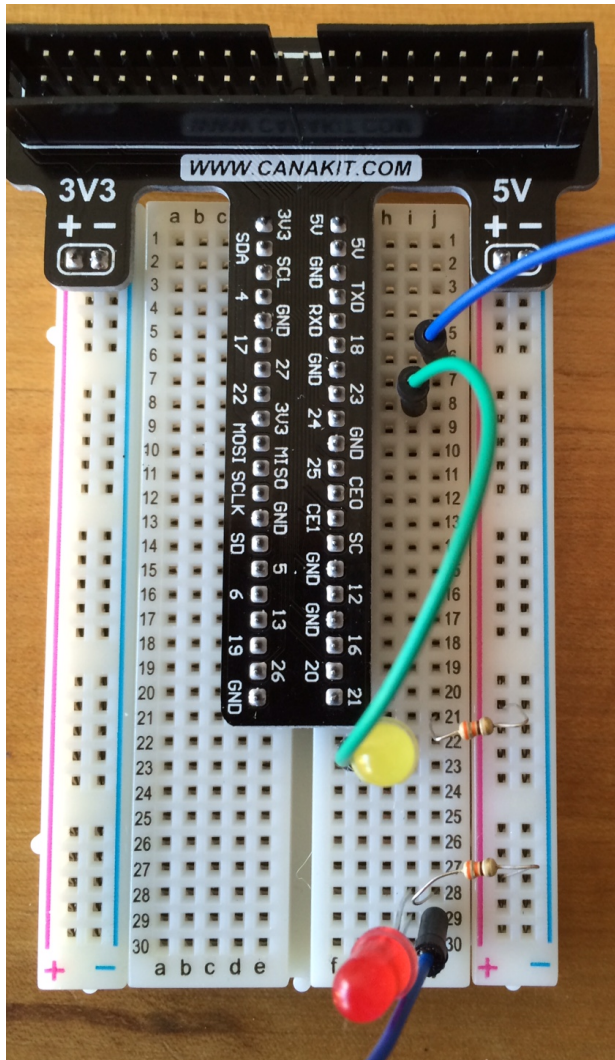
Rev. 1.1
16/07/2014

<http://www.element14.com>

(These pins correspond to the pin placement on the breakout board on the breadboard. So, referring to the numbering scheme describe in Lab 1, 3.3 volts are available from pin 1 and pin 17 and 5 volts is available from pins 2 and 4. This Lab will use 3.3 volts. In order to easily access the 3.3 volts we will use the power and ground columns on the right hand side of the board. The entire circuit photo is provided for reference as the wiring proceeds.



or like this if you have a B+



1. Check that the raspberry pi is not connected to power. Start with a bare breakout board, breadboard.
2. Unlike the 5 row columns on either side of the divide of the breadboard, the 2 separate columns on each end are linked electrically not by row, but by column. Therefore when power is provided to the "+", red row (1,+) in our notation, (2,+) to (30,+) can provide power. Similarly if (1,-) is grounded, (2,-) to (30,-) provides ground. Therefore we establish power and ground b, on the left hand side using breadboard pin 1, left hand side(1, a) and connecting it to right hand side (1,+). It is best to use a red or orange jumper to do this. Note the remainder of the connections will be on the right hand side. Ground will be established by connecting (3,i) to (1,-).
3. Place the yellow LED with the positive (longer) leg in (21,h) and the negative in (22,h). Place a 330Ω resistor leg in (21, j) and the other end of the leg in (21,+). Place one end of a green jumper in (22,g) and the other end of the jumper to (8,i), connecting the yellow LED to pin 12.
4. Place the red LED with the positive leg in (28,i) and the negative leg in (29,i). Place a 330Ω resistor leg in (28,j) and the other leg in (28,+). Place one end

of a blue jumper wire in (29,j) and the other end in (6,j) connecting the red LED to pin 16.

Coding Task

Have your instructor check the circuit. Power up the raspberry pi, startx and then launch a shell window and `sudo idle`. Your coding task is to add the code to operate the second LED and code the loop for it. Note the duration of sleep provided by the loop may need to be altered a bit.

A bit on the code

The code for this task adds no new coding constructs but it does add some logic twists in controlling items on the breadboard. Since two LEDs are now powered, the trickle of power the logic pins could provide was inadequate. We had to add power to the circuit and rely on the pin to complete the circuit when commanded by linking ground and closing the circuit. Therefore to close the circuit the pin needed to be pulled LOW and to open the circuit and turn off the light, it was pulled HIGH.

```
#
# sosry.py
# program to do morse code
# and provide redundant cues
# for dot and dash using yellow
# and red LEDs
#
# import necessary libraries.
# GPIO accesses the ports of Raspi
# time provides us with sleep
#

import RPi.GPIO as GPIO
import time

#
# morse code info
# sos in morse code is:
# ... --- ...
# dashes are 3 times longer than
# dit. 1 unit space within a letter
# 3 unit space between letters,
# 7 unit space between words
#
repeat = 50
pin = 12
#
# setup pin 2
#
# establish 2 pins
# to control LEDs
#
```

```

GPIO.setmode(GPIO.BOARD)
GPIO.setup(pin,GPIO.OUT)
GPIO.setup(pin2,GPIO.OUT)
#
# define dit (dot)
# note that LOW and HIGH
# have reversed, pulling
# pin LOW provides power
#
def dit(unit):
    GPIO.output(pin,GPIO.LOW)
    time.sleep(unit)
    GPIO.output(pin,GPIO.HIGH)
    time.sleep(unit)
    #pause between elements of a letter
    return

def dash(unit):

#
#define dash
#

def letter_pause(unit):
    time.sleep(unit * 2)
    # why 2? each unit has a pause
    # at end, just add 2 more
    # to make it 3.
    return

def word_pause(unit):
    time.sleep(unit * 6)
    # again accommodating
    # unit pause
    return

def s_letter(unit):
    dit(unit)
    dit(unit)
    dit(unit)
    #code S
    return

def o_letter(unit):
    dash(unit)
    dash(unit)
    dash(unit)
    #code O
    return

```

```

def sos(unit):
    s_letter(unit)
    letter_pause(unit)
    o_letter(unit)
    letter_pause(unit)
    s_letter(unit)
    word_pause(unit)
    #code SOS
    return

#
# for ever
#

#
# define loop
#

#
# time to cleanup
#

```

```
GPIO.cleanup()
```

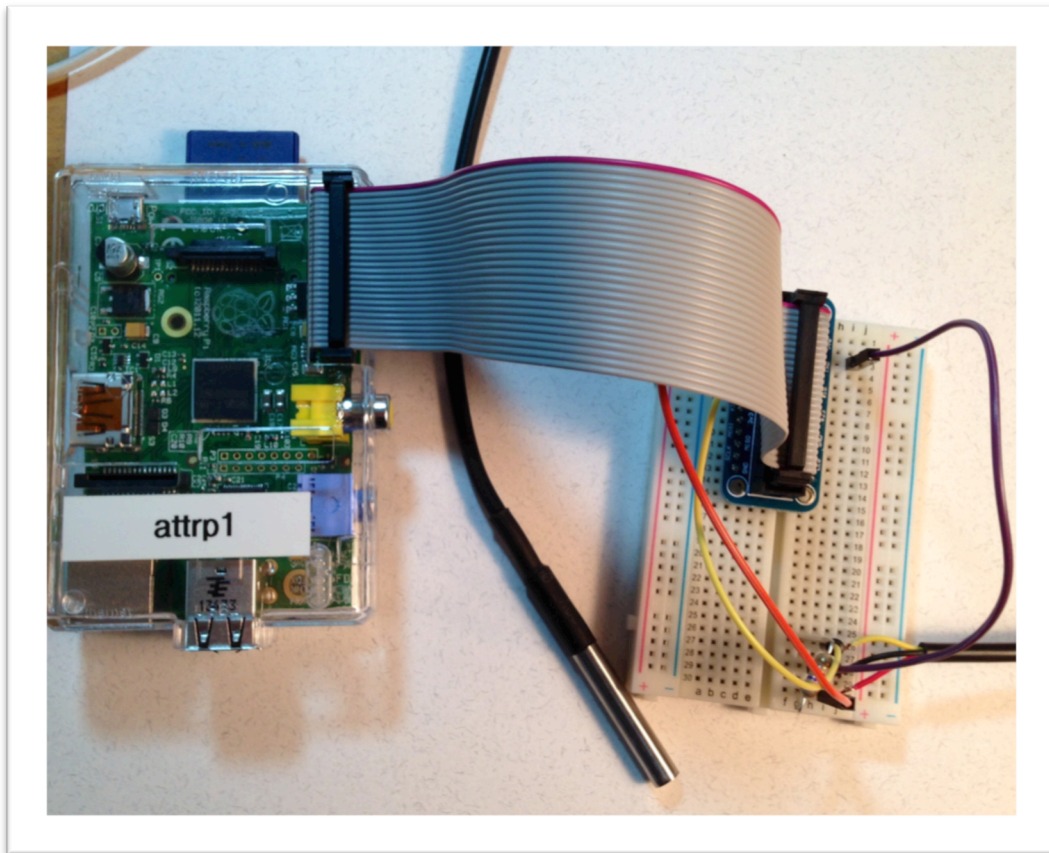
Lab 6: Sensored!

This lab explores reading sensors. It requires a bit more sophistication including comfort with command line computing, intricate, delicate wiring and some familiarity with python. The objective of this lab is to connect and read a digital temperature sensor.

Let's wire the breadboard first. The parts list:

- 3 jumper wires, red, yellow and black, if possible
- DS18B20 waterproof digital temperature sensor
- 4.7K Ω resistor (yellow, purple, red)

Follow these wiring directions carefully and make sure that your raspberry pi breadboard combination is disconnected from power.



1. Connect the digital temperature sensor to the breadboard. Do this carefully since the wires are fragile and stranded. Place the red wire in (30,j), the black wire in (28,j) and the yellow wire in (26,j).
2. Place one end of the red jumper in (30,i) and the other end in (1,a). This connects the temperature sensor to 3.3 volts.
3. Place one end of the black jumper in (28,i) and the other end in (3,i). This connects the temperature sensor to ground.
4. Place one end of the yellow jumper in (26, i) and the other end in (4,a). This connects the temperature sensor to pin 4 on the GPIO bus. Note it is very important that it is attached to pin 4.
5. Finally place one end of the resistor in (26,g) and the other end of the resistor in (30,g).
6. Make sure that subsequent steps did not loosen any connections especially the connections of the temperature sensor to the breadboard.

Command Line Work

This lab requires some command line work. Connect the raspberry pi to power and once logged in, `startx` to the window interface and start an LXTerminal session. Follow these steps at the command line

1. Type `sudo modprobe w1-gpio` and return

2. Type `sudo modprobe w1-therm` and return
3. Type `cd /sys/bus/w1/devices` and return
4. Type `ls` and return
5. Note the numeric file name, this is the serial number of the sensor and we will use it to communicate with it.
6. Cd to the serial number file name, e.g., `cd 28-000006087458` and return
7. Type `cat w1_slave` and return
8. The last item on the second line returned `t=28250` provides the temperature in centigrade, once you divide the number by 1000.

The previous steps were required to obtain the sensors serial number that you will use in the code. Attached is the code provided in `temper.py` in your home directory. This python code provides access to reading the probe. You must modify and add to this code. First replace the serial number with the correct serial number. Second place the temperature reading mechanism in a loop, so that it can constantly report the temperature at 10 sec intervals. Finally provide the code that converts the centigrade temperature to Fahrenheit. If you forget (I did) the formula is:

$$Temp-centigrade * (9/5) + 32$$

The code provided is:

```
#
# temper.py - python temperature sensor
# program adopted from University of Cambridge
# Computer Laboratory.
#
import time
#
# enables sleep
#
tfile = open ("/sys/bus/w1/devices/28-000006087458/w1_slave")
#
# read the file
# note that the number 28-...
# may be different and you acquire
# that from your command line
# investigations
#

#####
##### you will need a loop
#####

text = tfile.read()
tfile.close()
```

```

#
# discard the first line
#

secondline = text.split("\n")[1]

#
# dissect the second line
#

temperaturedata = secondline.split(" ")[9]

temperature = float(temperaturedata[2:])

####
#### prints out as centigrade/celsius, convert
#### to fahrenheit. I know, but it is traditional!
####
print temperature/1000.0

time.sleep(10)

```

A bit on the code

This program illustrates file i/o and string manipulation. Python is particularly adept at string manipulation which makes it a wonderful choice for many applications, including social media analysis, big data, web apps and with the addition of other data structures (e.g., lists, tuples, sequences, ...), artificial intelligence.

The command line manipulations were made possible by a hack to the linux kernel which bypassed some of the GPIO maneuvers we used in previous labs.

Lab 7 The capital of South Dakota is?

Exploring the state capital program. Sometimes it is best to read code. Can we generalize this?

```

#
#
# https://github.com/pythonkc/beginners-python-workshop/blob/master/\_static/dependancies/state\_capitals.py

```


#

```
capitals_dict = {  
'Alabama' : 'Montgomery',  
'Alaska' : 'Juneau',  
'Arizona' : 'Phoenix',  
'Arkansas' : 'Little Rock',  
'California' : 'Sacramento',  
'Colorado' : 'Denver',  
'Connecticut' : 'Hartford',  
'Delaware' : 'Dover',  
'Florida' : 'Tallahassee',  
'Georgia' : 'Atlanta',  
'Hawaii' : 'Honolulu',  
'Idaho' : 'Boise',  
'Illinois' : 'Springfield',  
'Indiana' : 'Indianapolis',  
'Iowa' : 'Des Moines',  
'Kansas' : 'Topeka',  
'Kentucky' : 'Frankfort',  
'Louisiana' : 'Baton Rouge',  
'Maine' : 'Augusta',  
'Maryland' : 'Annapolis',  
'Massachusetts' : 'Boston',  
'Michigan' : 'Lansing',  
'Minnesota' : 'Saint Paul',  
'Mississippi' : 'Jackson',  
'Missouri' : 'Jefferson City',  
'Montana' : 'Helena',  
'Nebraska' : 'Lincoln',  
'Nevada' : 'Carson City',  
'New Hampshire' : 'Concord',  
'New Jersey' : 'Trenton',  
'New Mexico' : 'Santa Fe',  
'New York' : 'Albany',  
'North Carolina' : 'Raleigh',  
'North Dakota' : 'Bismarck',  
'Ohio' : 'Columbus',  
'Oklahoma' : 'Oklahoma City',  
'Oregon' : 'Salem',  
'Pennsylvania' : 'Harrisburg',  
'Rhode Island' : 'Providence',  
'South Carolina' : 'Columbia',  
'South Dakota' : 'Pierre',  
'Tennessee' : 'Nashville',  
'Texas' : 'Austin',  
'Utah' : 'Salt Lake City',
```

```

'Vermont' : 'Montpelier',
'Virginia' : 'Richmond',
'Washington' : 'Olympia',
'West Virginia' : 'Charleston',
'Wisconsin' : 'Madison',
'Wyoming' : 'Cheyenne',
}

import random

while True:
    state = random.choice(capitals_dict.keys())
    capital = capitals_dict[state]
    capital_guess = raw_input("What is the capital of " +
state + "? ")
    if capital_guess == "Exit":
        print "Goodbye"
        break

    if capital_guess == capital:
        print "Correct! Nice job."
    else:
        print "Incorrect. The capital of " + state + " is "
+ capital + "."

```

LAB 8 Shall we play a game?

In the early 80s a movie called War Games made the title of this Lab famous. Here's the IMDb description of the game:

A young man finds a back door into a military central computer in which reality is confused with game-playing, possibly starting World War III.

Although we will not be starting World War III today, let's create a game called nim using all that you learned today. The game is simple:

- Start with 21 sticks (standard game)
- Players selects 1, 2 or 3 sticks
- Player selecting last stick loses

You can program the game having alternating players or if you would like to really get creative, you could pit yourself against the computer and have the computer learn how to win at the game.

Summing Up

i hope you enjoyed these Labs and that they encourage you to explore Python, electronics and the raspi. There are many more single board computers and electronic components to explore. The website <http://aarphacker.com> will provide all the written materials and code for this course and also be a source for other resources to continue your exploration.

Resources

Buying hardware:

<https://www.adafruit.com/>
<http://www.evilmadscientist.com/>
<http://makezine.com/>

Raspberry pi info:

<http://www.raspberrypi.org/>

Check the web site at aarphacker.com!

Thanks

I would like to thank all the experimenters on the web who provided insight into the marvels of python and the raspi. This includes:

- <http://www.rpiblog.com>
- <https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/>
- <https://learn.adafruit.com/category/raspberry-pi>
- <https://projects.drogon.net/raspberry-pi/gpio-examples/tux-crossing/gpio-examples-1-a-single-led/>

I would like to thank Leah and Kathy Vesonder for carefully reviewing this document for clarity, consistency, grammar and common sense.