ECOES: Software Engineering

Version 20190712
Gregg Vesonder

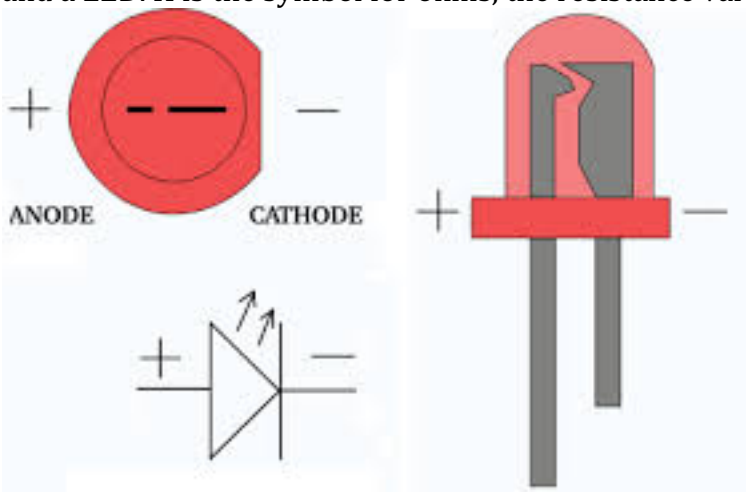
Welcome!

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.

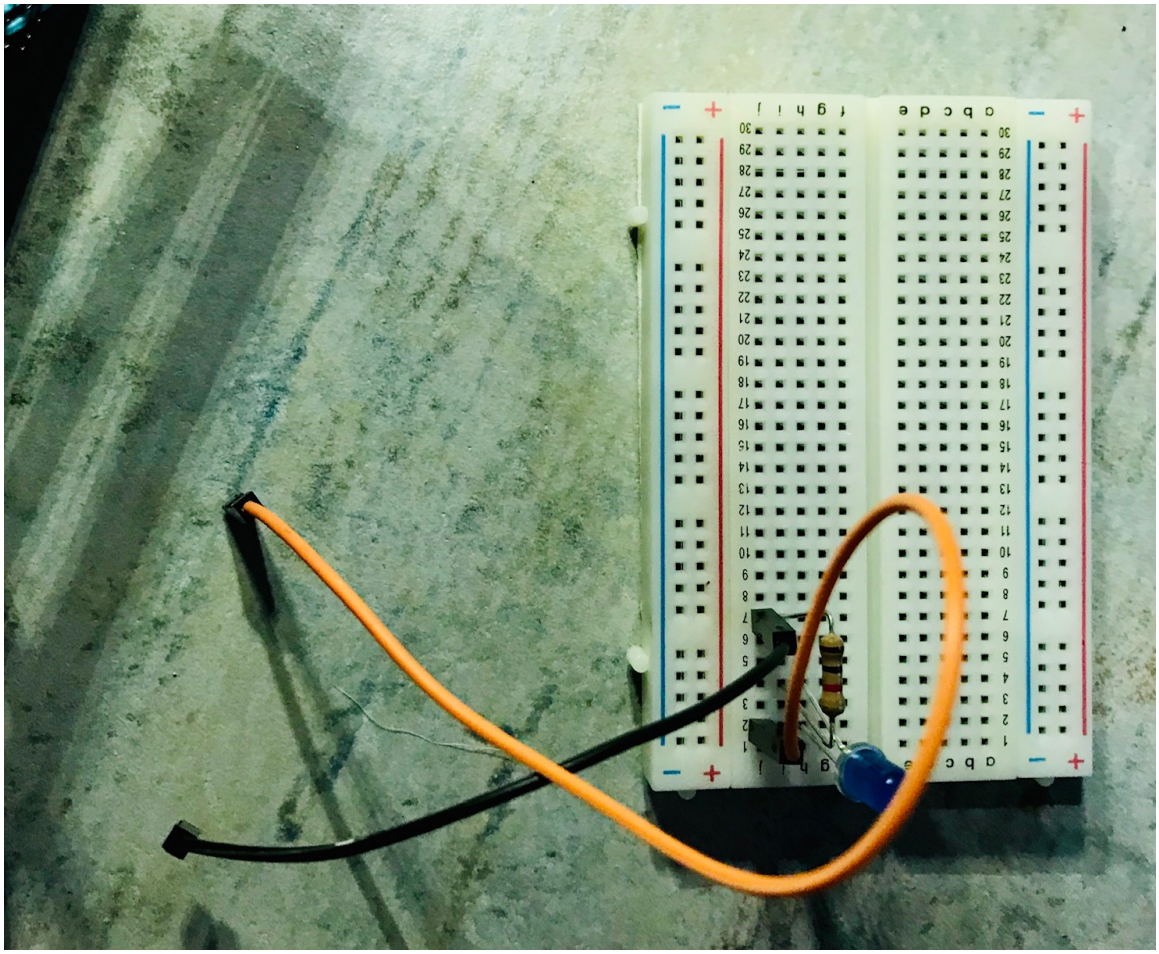
Lab 1: You Blinked!

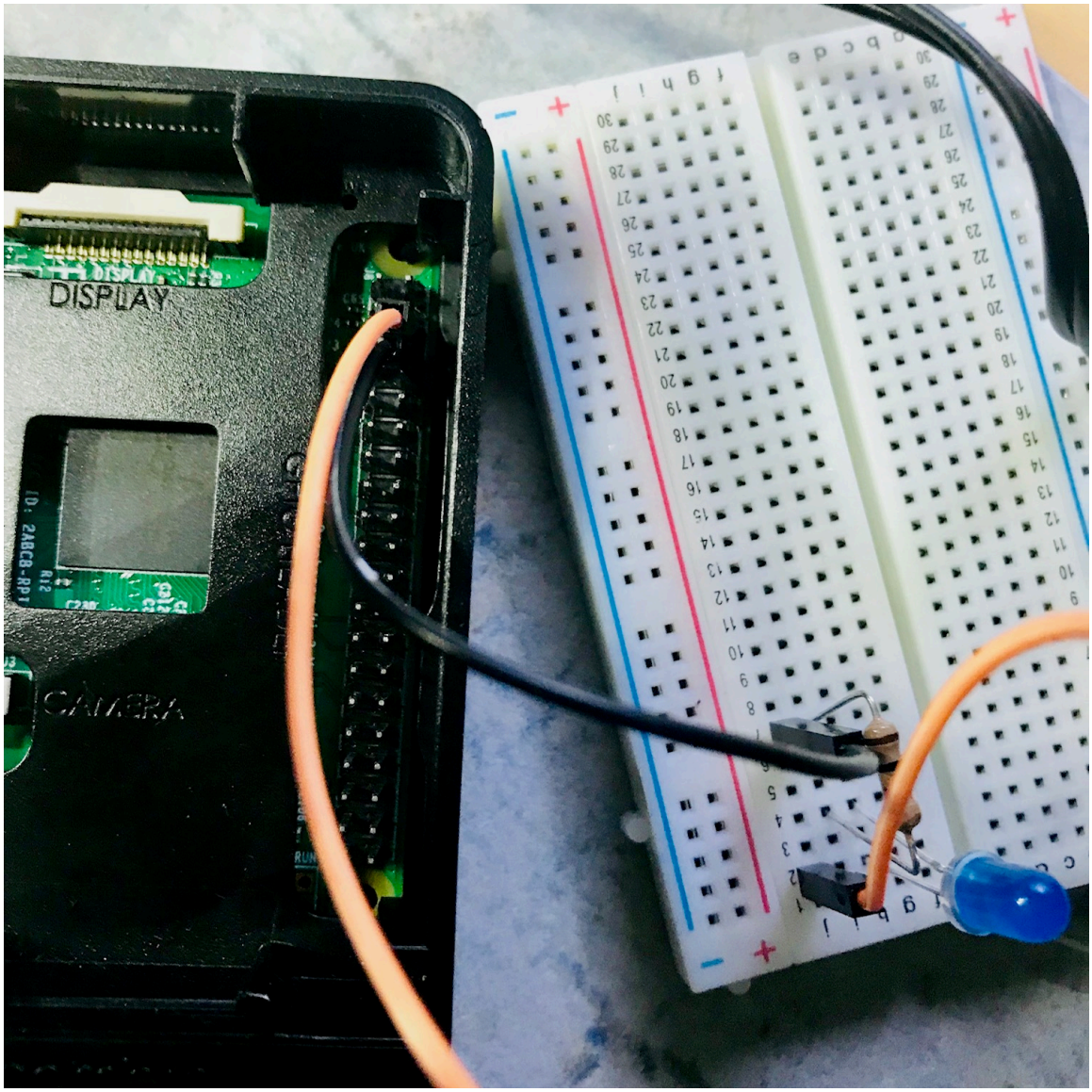
In this Lab you will program the LED to blink. There is a lot going on in this Lab, it provides the foundation for the rest of the Labs. First let's build the circuit we will use. From a coding perspective we will be working with variables and loops.

1) The parts are: a 1000Ω resistor and 2 M-F jumper wires of different colors and a LED. Ω is the symbol for ohms, the resistance value of a resistor.



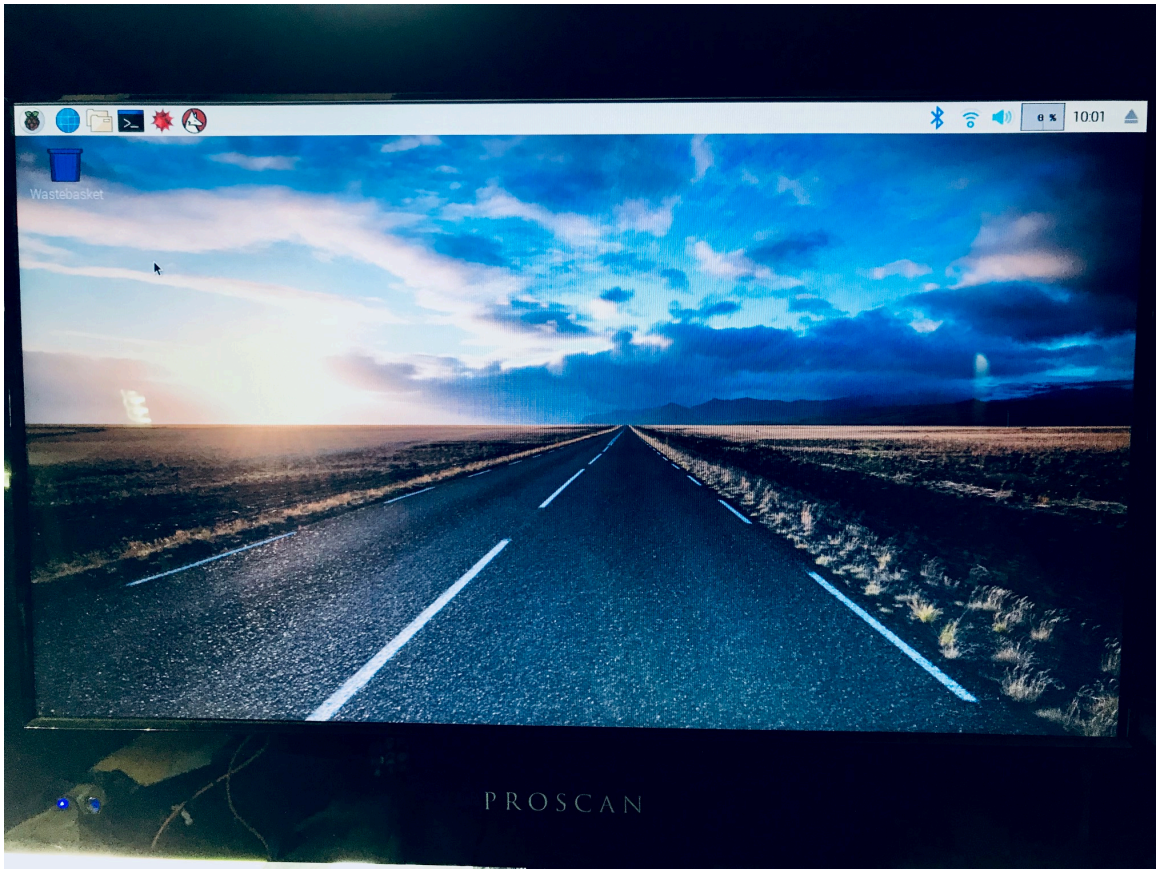
2) Wire the circuit on the breadboard as pictured: Your board should look this:



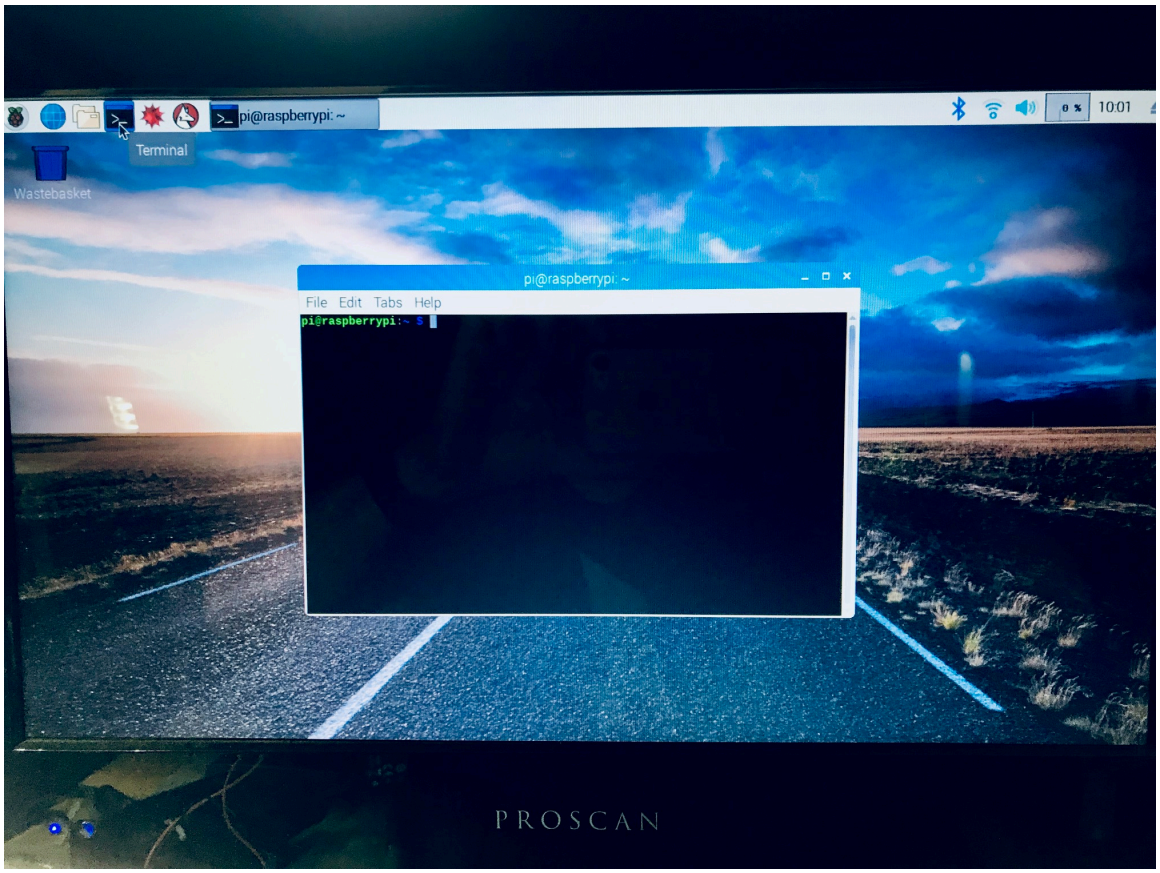


3) Have your instructor check your board. After the instructor checks it, plug in the power supply and wait for the computer to start. It will take a minute or so.
4) If everything goes fine you will be taken into the raspberry pi window system. However if you get a login prompt, type `pi` – note that 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

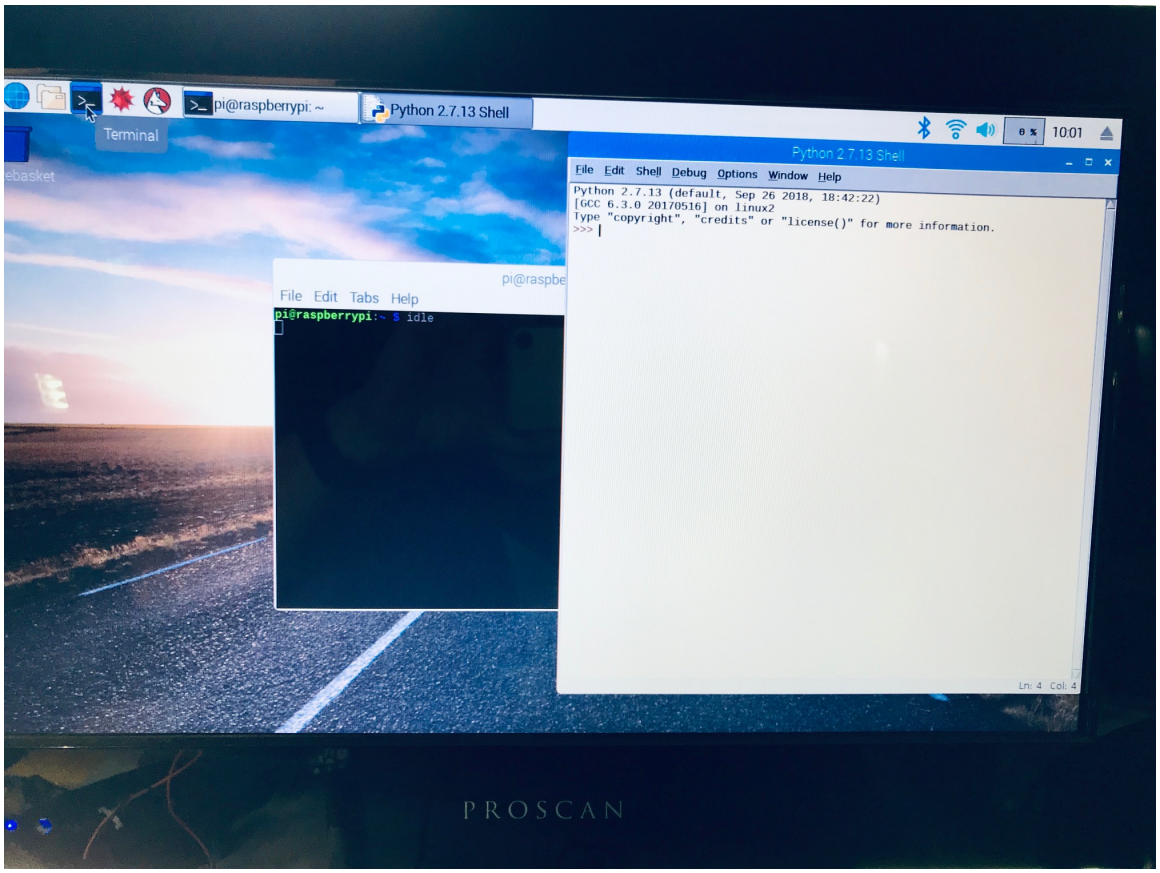
screen should look like this:



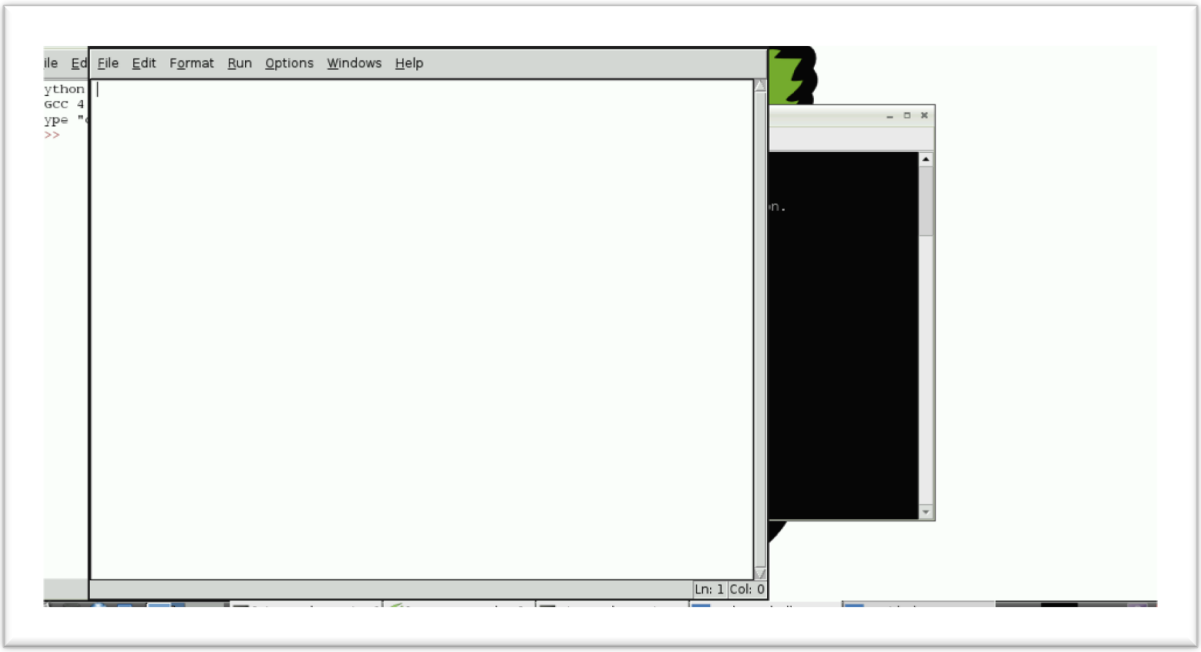
5) Time to start the programming environment. For all of our Labs we will be using the Python language and the IDLE programming environment. The Python language is a popular language that is used by students, educators and professional developers. Python was not named after a snake but after a British 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 the terminal icon, and double click quickly on it. This takes you into a command line window.



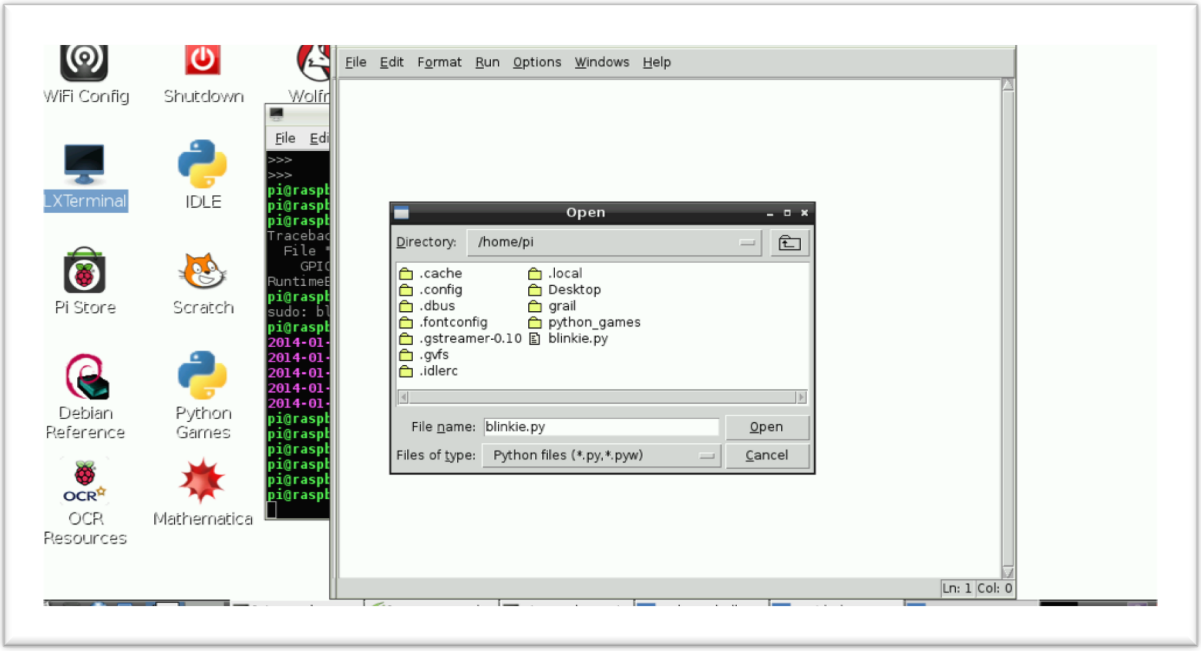
Type `idle` at the prompt and this will launch the Python programming environment.



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:



6) 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 logical name that the raspi board.

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(8, GPIO.OUT)

##
##do a for loop with repeat
##
    GPIO.output(8, GPIO.HIGH)
    time.sleep(1)
    GPIO.output(8, GPIO.LOW)
    time.sleep(1)

#
#cleanup
#
GPIO.cleanup()
```

Lab 2: 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
```

```python
# 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 = 8
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 3: 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!

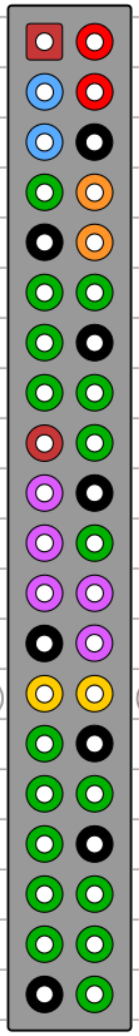
Lab 4: Was it a Dit or Dah?

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:

Raspberry Pi 3 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



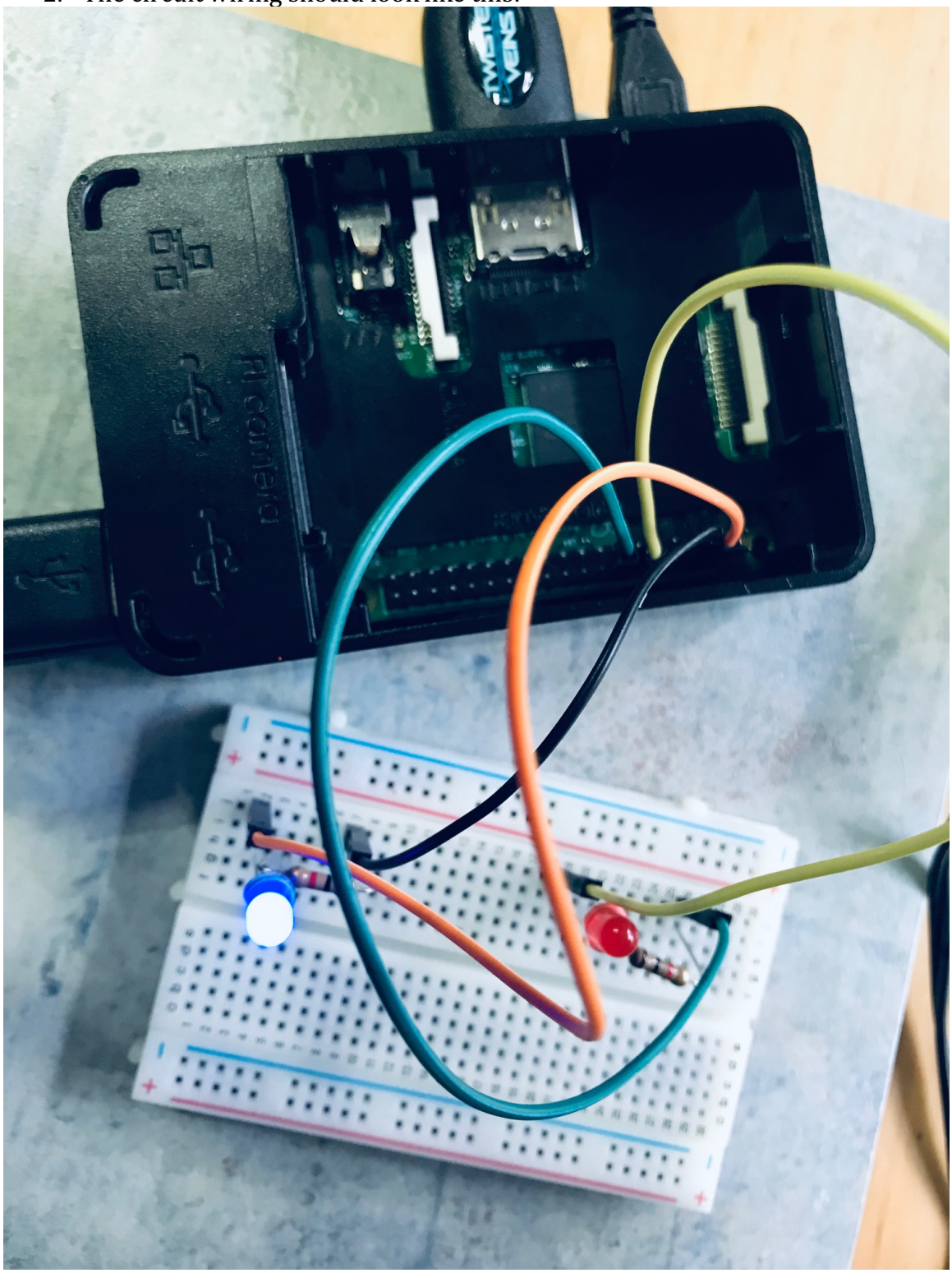
Rev. 2
29/02/2016

www.element14.com/RaspberryPi

(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.

1. Check that the raspberry pi is not connected to power. Start with the circuit you made in Lab 1. Essentially you are replicating the circuit from Lab 1 with a different color LED and using pins 14 and 16 on the raspberry pi (see above). Use two different colored M-F jumpers. Note 16 is the pin number you should use for pin2 in the code.

2. The circuit wiring should look like this:



Coding Task

Have your instructor check the circuit. Power up the raspberry pi, startx and then launch a shell window and `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 = 8
#
# setup pin 2
# pin number on pi is 16
#
# 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 5 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 6 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/
http://sparkfun.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.