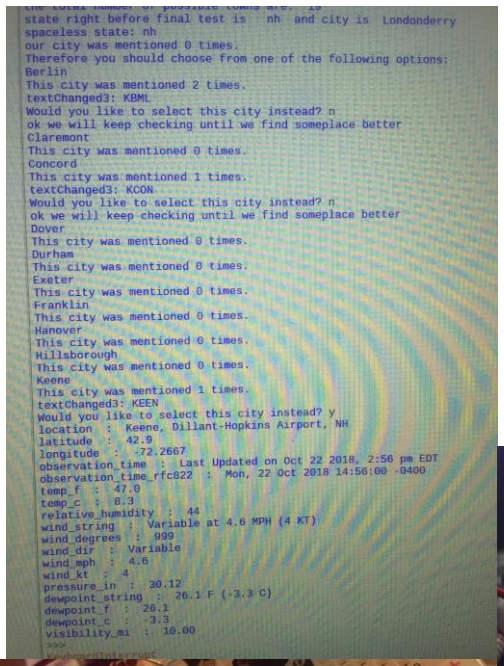
Lab Report 6 – Accessing the Internet

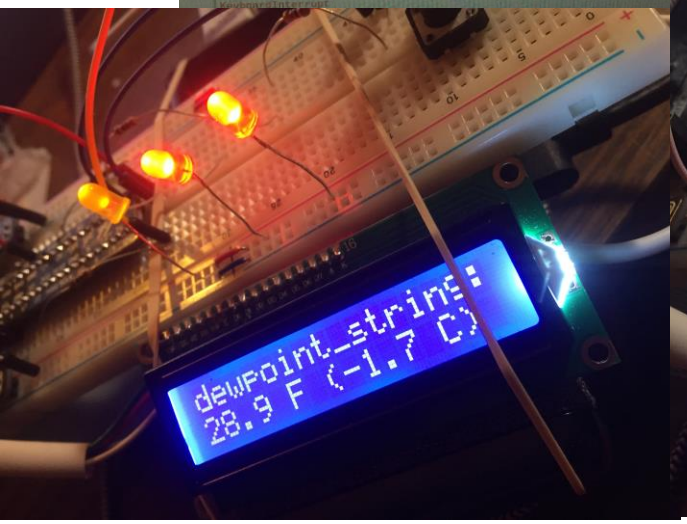
During this lab for some reason I felt compelled to go above and beyond what would be necessary and decided to create a fully functioning program that could potential lead to some sort of useful objective being achieved. Let me say right off the bat, that everything here works and everything here is terrible. Basically, my partner and I have been slowly, little by little combining various bits that we've created in labs with things we've done outside of labs in order to produce something that could potentially be used in our final project. I had been hoping that, by going through all this extra effort in making this lab into a fully-fledged device, that I would be able to really push much farther into unknown territory that I would normally dare in designing my goals for a final project—however, based on things turned out here, I guess the one benefit I might be able to take away from all this, is how *not* to do a final project, and perhaps the best lesson to be taken here is how quickly things can get out of control. As can be seen, there are over a thousand lines of code here, so describing how everything works might be a little bit difficult—but I'll attempt to do so in the most concise manner possible.

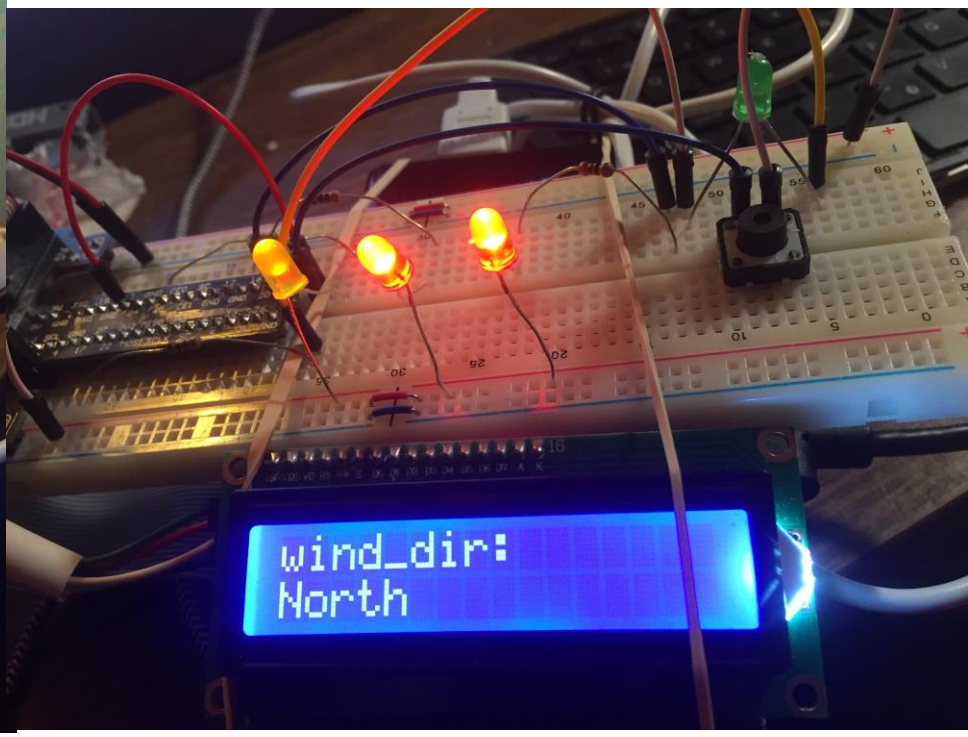
The entire point of this code is to gather info from online sites in order to give us info on the weather for our particular location. However, our code is also able to extrapolate outward in case our particular location is not mentioned on any NOAA website. In other words, this code will give you instant access to all of NOAA's weather info on your current location, regardless of your state, automatically and instantaneously by determining your location, and then plugging it into other websites which will determine the weather in that location at any given time. The only problems crop up when:

A) For whatever reason your particular city (where you are currently located) isn't mentioned on any NOAA websites discussing weather in your state
B) …or when you actually want info on a *different* state then the one you are currently located

These problems are solved by allowing our code to adapt and extrapolate outwards, and ask questions of the user to gain further information on what the user might want. For example, if your particular city (where you are currently located) isn't mentioned, then our code will grab info on all major cities in your state and check them one by one. For each city that is mentioned our code will then ask the user whether this *other* city's weather info will be sufficient for their needs. Our code also allows the user to change which state they might want to be gaining info on (if their current state is not the one they are interested in) by allowing them to enter any other state in Morse code (as well as their city).

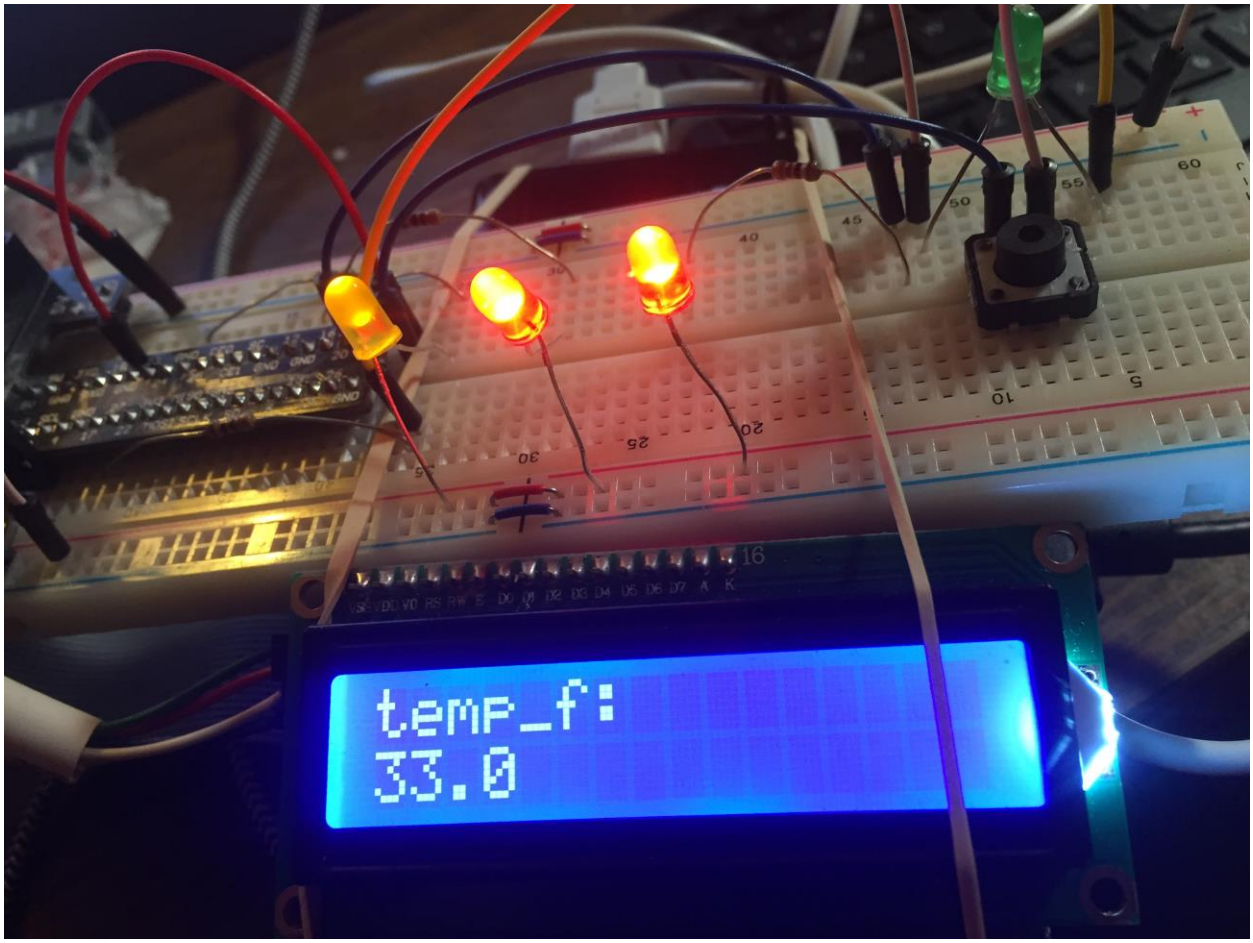

state right before final test is nh and city is Londonderry
spaceless state: nh
our city was mentioned 0 times.
Therefore you should choose from one of the following options:
Berlin
This city was mentioned 2 times.
textChanged3: KBML
Would you like to select this city instead? n
ok we will keep checking until we find someplace better
Claremont
This city was mentioned 0 times.
Concord
This city was mentioned 1 times.
textChanged3: KCON
Would you like to select this city instead? n
ok we will keep checking until we find someplace better
Dover
This city was mentioned 0 times.
Durham
This city was mentioned 0 times.
Exeter
This city was mentioned 0 times.
Franklin
This city was mentioned 0 times.
Hanover
This city was mentioned 0 times.
Hillsborough
This city was mentioned 0 times.
Keene
This city was mentioned 1 times.
textChanged3: KEEN
Would you like to select this city instead? y
location : Keene, Dillant-Hopkins Airport, NH
latitude : 42.9
longitude : -72.2667
observation_time : Last Updated on Oct 22 2018, 2:56 pm EDT
observation_time_rfc822 : Mon, 22 Oct 2018 14:56:00 -0400
temp_f : 47.0
temp_c : 8.3
relative_humidity : 44
wind_string : Variable at 4.6 MPH (4 KT)
wind_degrees : 999
wind_dir : Variable
wind_mph : 4.6
wind_kt : 4
pressure_in : 30.12
dewpoint_string : 26.1 F (-3.3 C)
dewpoint_f : 26.1
dewpoint_c : -3.3
visibility_mi : 10.00


dewpoint_string:
28.9 F (-1.7 C)


wind_dir:
North

The Wiring:

Much of the wiring used here is the same as it was for previous labs (such as the LCD display) — nonetheless we have included this info as well (as can be seen below. The green LED, which lights up based on the pushing of the button, when entering city and state info via Morse code is wired up using a resistor, and is hooked up on one side into GPIO 17 and the other side into ground. Our button is hooked up on one side into GPIO 18 and the other side into ground. Our warning light LED's (which weren't yet applied to our code, and simply light up regardless of weather conditions are hooked up to



GPIO 24, 12, and 16.

Finally, the LCD display:

- This is a bit more complicated, and involves us attaching a special kind of tool which can be calibrated using a screwdriver to allow us to create a very specific amount of resistance, which is soldered into place and can later be re-adjusted to ensure we correctly display the text we desire onto our screen, such that it can be easily seen.
- We then simply connect GND to Ground via a green wire, VCC to our 5 volt power supply (via a gray wire), SDA to SDA1 via a white wire, and SCL to SCL1 via a blue wire.

The wires connected to the button that we see on the board are superfluous (as is the button itself), and is not used anywhere in this lab (and therefore unnecessary).

The Code:

```
#calculating fire risk through image analysis

import datetime
import time
import urllib.request

#!/usr  /bin/env python
# ----------------
# Modified 2018-09-30 to use Matt Hawkins' LCD library for I2C available
# from https://bitbucket.org/MattHawkinsUK/rpispy-misc/raw/master/python/lcd_i2c.py
#
import RPi.GPIO as GPIO
import time

# Additional stuff for LCD
import smbus


ButtonPin = 18
LedPin = 17

yellowLedPin=5
greenLedPin=6

#LCD pin assignments, constants, etc
I2C_ADDR  = 0x27 # I2C device address
LCD_WIDTH = 16   # Maximum characters per line

# Define some device constants
LCD_CHR = 1 # Mode - Sending data
LCD_CMD = 0 # Mode - Sending command

LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line
LCD_LINE_3 = 0x94 # LCD RAM address for the 3rd line
LCD_LINE_4 = 0xD4 # LCD RAM address for the 4th line

LCD_BACKLIGHT  = 0x08 # On
#LCD_BACKLIGHT = 0x00 # Off

ENABLE = 0b00000100 # Enable it

# Timing constants
E_PULSE = 0.0005
E_DELAY = 0.0005


# Ultrasonic pin assignments
```

```
SR04_trigger_pin = 20
SR04_echo_pin = 21

# LCD commands
LCD_CMD_4BIT_MODE = 0x28  # 4 bit mode, 2 lines, 5x8 font
LCD_CMD_CLEAR = 0x01
LCD_CMD_HOME = 0x02  # goes to position 0 in line 0
LCD_CMD_POSITION = 0x80  # Add this to DDRAM address

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)     # Numbers GPIOs by standard marking
GPIO.setup(LedPin, GPIO.OUT)  # Set LedPin's mode is output
GPIO.output(LedPin, GPIO.HIGH) # Set LedPin high(+3.3V) to turn off led

GPIO.setup(yellowLedPin, GPIO.OUT)  # Set LedPin's mode is output
GPIO.output(yellowLedPin, GPIO.HIGH) # Set LedPin high(+3.3V) to turn off led

GPIO.setup(greenLedPin, GPIO.OUT)  # Set LedPin's mode is output
GPIO.output(greenLedPin, GPIO.HIGH) # Set LedPin high(+3.3V) to turn off led


warningLed1=24
warningLed2=12
warningLed3=16

GPIO.setup(warningLed1, GPIO.OUT)  # Set LedPin's mode is output
GPIO.output(warningLed1, GPIO.HIGH) # Set LedPin high(+3.3V) to turn off led

GPIO.setup(warningLed2, GPIO.OUT)  # Set LedPin's mode is output
GPIO.output(warningLed2, GPIO.HIGH) # Set LedPin high(+3.3V) to turn off led


GPIO.setup(warningLed3, GPIO.OUT)  # Set LedPin's mode is output
GPIO.output(warningLed3, GPIO.HIGH) # Set LedPin high(+3.3V) to turn off led




# Set ButtonPin's mode as input with the internal pullup
GPIO.setup(ButtonPin, GPIO.IN, pull_up_down = GPIO.PUD_UP)

# Set up the SR04 pins
GPIO.setup(SR04_trigger_pin, GPIO.OUT)
GPIO.setup(SR04_echo_pin, GPIO.IN)
GPIO.output(SR04_trigger_pin, GPIO.LOW)

#Open I2C interface
#bus = smbus.SMBus(0)  # Rev 1 Pi uses 0
bus = smbus.SMBus(1) # Rev 2 Pi uses 1

GPIO.output(LedPin, GPIO.LOW)  # led off
GPIO.output(yellowLedPin, GPIO.LOW)  # led off
GPIO.output(greenLedPin, GPIO.LOW)  # led off

def distance(metric):
    # set Trigger to HIGH
    GPIO.output(SR04_trigger_pin, GPIO.HIGH)

    # set Trigger after 0.01ms to LOW
```

```
    time.sleep(0.00001)
    GPIO.output(SR04_trigger_pin, GPIO.LOW)

    startTime = time.time()
    stopTime = time.time()

    # Get the returnb pulse start time
    while 0 == GPIO.input(SR04_echo_pin):
        startTime = time.time()

    # Get the pulse length
    while 1 == GPIO.input(SR04_echo_pin):
        stopTime = time.time()

    elapsedTime = stopTime - startTime
    # The speed of sound is 33120 cm/S or 13039.37 inch/sec.
    # Divide by 2 since this is a round trip
    if (1 == metric):
        d = (elapsedTime * 33120.0) / 2   # metric
    else:
        d = (elapsedTime * 13039.37) / 2   # english

    return d

def lcd_init():
  # Initialise display
  lcd_byte(0x33,LCD_CMD) # 110011 Initialise
  lcd_byte(0x32,LCD_CMD) # 110010 Initialise
  lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction
  lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off
  lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font size
  lcd_byte(0x01,LCD_CMD) # 000001 Clear display
  time.sleep(E_DELAY)

def lcd_byte(bits, mode):
  # Send byte to data pins
  # bits = the data
  # mode = 1 for data
  #        0 for command

  bits_high = mode | (bits & 0xF0) | LCD_BACKLIGHT
  bits_low = mode | ((bits<<4) & 0xF0) | LCD_BACKLIGHT

  # High bits
  bus.write_byte(I2C_ADDR, bits_high)
  lcd_toggle_enable(bits_high)

  # Low bits
  bus.write_byte(I2C_ADDR, bits_low)
  lcd_toggle_enable(bits_low)

def lcd_toggle_enable(bits):
  # Toggle enable
  time.sleep(E_DELAY)
  bus.write_byte(I2C_ADDR, (bits | ENABLE))
  time.sleep(E_PULSE)
  bus.write_byte(I2C_ADDR,(bits & ~ENABLE))
  time.sleep(E_DELAY)

def lcd_string(message,line):
  # Send string to display
```

```
  message = message.ljust(LCD_WIDTH," ")

  lcd_byte(line, LCD_CMD)

  for i in range(LCD_WIDTH):
    lcd_byte(ord(message[i]),LCD_CHR)

# functions not in the original library
def lcd_xy(col, row):
      lcd_byte(LCD_CMD_POSITION+col+(64*row), LCD_CMD)

def lcd_msg(msg_string):
      for i in range(0, len(msg_string)):
            lcd_byte(ord(msg_string[i]), LCD_CHR)


lcd_init()
lcd_string("Weather Test", LCD_LINE_1)
time.sleep(1)
lcd_string("Finding weather", LCD_LINE_1)
lcd_string("in your location...", LCD_LINE_2)
time.sleep(2)


def abbrevToFull(myStateInitial, myStateDashed):

    if(len(myStateInitial)>=3):
          myStateInitial=myStateInitial[1:]
    print("debug abbrev state var is: ear"+myStateInitial+"ear")
    if(myStateInitial=="AL"):
          myStateDashed="Alabama"
    elif(myStateInitial=="AK"):
          myStateDashed="Alaska"
    elif(myStateInitial=="AZ"):
          myStateDashed="Arizona"
    elif(myStateInitial=="AR"):
          myStateDashed="Arkansas"
    elif(myStateInitial=="CA"):
          myStateDashed="California"
    elif(myStateInitial=="CO"):
          myStateDashed="Colorado"
    elif(myStateInitial=="CT"):
          myStateDashed="Connecticut"
    elif(myStateInitial=="DE"):
          myStateDashed="Delaware"
    elif(myStateInitial=="FL"):
          myStateDashed="Florida"
    elif(myStateInitial=="GA"):
          myStateDashed="Georgia"
    elif(myStateInitial=="HI"):
          myStateDashed="Hawaii"
    elif(myStateInitial=="ID"):
          myStateDashed="Idaho"
    elif(myStateInitial=="IL"):
          myStateDashed="Illinois"
    elif(myStateInitial=="IN"):
          myStateDashed="Indiana"
    elif(myStateInitial=="IA"):
          myStateDashed="Iowa"
    elif(myStateInitial=="KS"):
```

```
        myStateDashed="Kansas"
    elif(myStateInitial=="KY"):
        myStateDashed="Kentucky"
    elif(myStateInitial=="LA"):
        myStateDashed="Louisiana"
    elif(myStateInitial=="ME"):
        myStateDashed="Maine"
    elif(myStateInitial=="MD"):
        myStateDashed="Maryland"
    elif(myStateInitial=="MA"):
        myStateDashed="Massachusetts"
    elif(myStateInitial=="MI"):
        myStateDashed="Michigan"
    elif(myStateInitial=="MN"):
        myStateDashed="Minnesota"
    elif(myStateInitial=="MS"):
        myStateDashed="Mississippi"
    elif(myStateInitial=="MO"):
        myStateDashed="Missouri"
    elif(myStateInitial=="MT"):
        myStateDashed="Montana"
    elif(myStateInitial=="NE"):
        myStateDashed="Nebraska"
    elif(myStateInitial=="NV"):
        myStateDashed="Nevada"
    elif(myStateInitial=="NH"):
        myStateDashed="New-Hampshire"
    elif(myStateInitial=="NJ"):
        myStateDashed="New-Jersey"
    elif(myStateInitial=="NM"):
        myStateDashed="New-Mexico"
    elif(myStateInitial=="NY"):
        myStateDashed="New-York"
    elif(myStateInitial=="NC"):
        myStateDashed="North-Carolina"
    elif(myStateInitial=="ND"):
        myStateDashed="North-Dakota"
    elif(myStateInitial=="OH"):
        myStateDashed="Ohio"
    elif(myStateInitial=="OK"):
        myStateDashed="Oklahoma"
    elif(myStateInitial=="OR"):
        myStateDashed="Oregon"
    elif(myStateInitial=="PA"):
        myStateDashed="Pennsylvania"
    elif(myStateInitial=="RI"):
        myStateDashed="Rhode-Island"
    elif(myStateInitial=="SC"):
        myStateDashed="South-Carolina"
    elif(myStateInitial=="SD"):
        myStateDashed="South-Dakota"
    elif(myStateInitial=="TN"):
        myStateDashed="Tennessee"
    elif(myStateInitial=="TX"):
        myStateDashed="Texas"
    elif(myStateInitial=="UT"):
        myStateDashed="Utah"
    elif(myStateInitial=="VT"):
        myStateDashed="Vermont"
    elif(myStateInitial=="VA"):
        myStateDashed="Virginia"
```

```
    elif(myStateInitial=="WA"):
        myStateDashed="Washington"
    elif(myStateInitial=="WV"):
        myStateDashed="West-Virginia"
    elif(myStateInitial=="WI"):
        myStateDashed="Wisconsin"
    elif(myStateInitial=="WY"):
        myStateDashed="Wyoming"
    else:
        myStateDashed="NONE"
    return myStateDashed



def quickGrabAllText(inputUrl,outputText):
    grabBytes= urllib.request.urlopen(inputUrl)
    bText = grabBytes.read()
    outputText = bText.decode("utf8")
    grabBytes.close()

    return outputText




#now we define the morse code function:
#This conversion would be about converting dots/dashes to letters)
def dd_to_let(convertedLetter):
    if(convertedLetter=='.-'):
        convertedLetter='a'
    elif(convertedLetter=='-...'):
        convertedLetter='b'
    elif(convertedLetter=='-.-.'):
        convertedLetter='c'
    elif(convertedLetter=='-..'):
        convertedLetter='d'
    elif(convertedLetter=='.'):
        convertedLetter='e'
    elif(convertedLetter=='..-.'):
        convertedLetter='f'
    elif(convertedLetter=='--.'):
        convertedLetter='g'
    elif(convertedLetter=='....'):
        convertedLetter='h'
    elif(convertedLetter=='..'):
        convertedLetter='i'
    elif(convertedLetter=='.---'):
        convertedLetter='j'
    elif(convertedLetter=='-.-'):
        convertedLetter='k'
    elif(convertedLetter=='.-..'):
        convertedLetter='l'
    elif(convertedLetter=='--'):
        convertedLetter='m'
    elif(convertedLetter=='-.'):
        convertedLetter='n'
    elif(convertedLetter=='---'):
        convertedLetter='o'
    elif(convertedLetter=='.--.'):
```

```
            convertedLetter='p'
        elif(convertedLetter=='--.-'):
            convertedLetter='q'
        elif(convertedLetter=='.-.'):
            convertedLetter='r'
        elif(convertedLetter=='...'):
            convertedLetter='s'
        elif(convertedLetter=='-'):
            convertedLetter='t'
        elif(convertedLetter=='..-'):
            convertedLetter='u'
        elif(convertedLetter=='...-'):
            convertedLetter='v'
        elif(convertedLetter=='.--'):
            convertedLetter='w'
        elif(convertedLetter=='-..-'):
            convertedLetter='x'
        elif(convertedLetter=='-.--'):
            convertedLetter='y'
        elif(convertedLetter=='--..'):
            convertedLetter='z'
        elif(convertedLetter=='.----'):
            convertedLetter='1'
        elif(convertedLetter=='..---'):
            convertedLetter='2'
        elif(convertedLetter=='...--'):
            convertedLetter='3'
        elif(convertedLetter=='....-'):
            convertedLetter='4'
        elif(convertedLetter=='.....'):
            convertedLetter='5'
        elif(convertedLetter=='-....'):
            convertedLetter='6'
        elif(convertedLetter=='--...'):
            convertedLetter='7'
        elif(convertedLetter=='---..'):
            convertedLetter='8'
        elif(convertedLetter=='----.'):
            convertedLetter='9'
        elif(convertedLetter=='-----'):
            convertedLetter='0'

        else:
            convertedLetter='?'
        return convertedLetter

#-------------------------------------------------------------
def end_message(convertedMessage):
    convertedMessage=(str(convertedMessage)+'. ---OUT---')
    print('COMPLETE MESSAGE: ', str(convertedMessage))
    return convertedMessage
#-------------------------------------------------------------


def run_new_letter_funcs(ddCombo, convertedLetter):
    convertedLetter=ddCombo
    #print('Debug: our midconversion letter is ', str(convertedLetter))
    convertedLetter=dd_to_let(convertedLetter)
    print('our new letter is ', str(convertedLetter))
    return convertedLetter
```

```
#-------------------------------------------------------------
def add_word_to_message(convertedWord,convertedMessage):
    convertedMessage=(str(convertedMessage)+' '+str(convertedWord))
    print('our current message so far is ', str(convertedMessage))
    return convertedMessage
#-------------------------------------------------------------
def add_letter_to_word(convertedLetter,convertedWord):
    convertedWord=(str(convertedWord)+str(convertedLetter))
    print('our current word so far is ', str(convertedWord))
    return convertedWord

#-------------------------------------------------------------




currentWebsiteString=""

stid=""
fullText=""


#================================================================================================
#================================================================================================


def morseCode(textInOut):
    OFFtimer=0 #this is the amount of time that the LED was OFF for
    ONtimer=0 #this is the amount of time that the LED was ON for

    ONtimerStarted=0 #this checks if our ONtimer has started ticking yet -- if is HAS,
    # then we we set ONtimerStarted=1... when our timer turns off, ONtimerStarted is
    # reset back to ZERO.  OFFtimerStarted follows the same basic principle.
    OFFtimerStarted=0

    startTimeOn=1.111 #records exact time at the moment the LED is turned ON
    startTimeOff=1.111 #records exact time at the moment the LED is turned OFF

    onList = []
    offList = []

    numButtonPushes = 0
    numButtonReleases = 0

    waitTime=0
    timeUnit=0.14


    #==============SECTION  2: DEFINING FUNCTIONS (AND DECLARING VARIABLES THAT USE THEM)========
    #================================================================================

    convertedLetter='c'
    convertedWord=''
    convertedMessage='Message: '
```

```
    ddCombo='-...'#this will get SUBSTITUTED with whatever we use in OTHER code


    #===SECTION 3: THIS WAITING LOBBY RUNS CONTINUALLY UNTIL THE USER STARTS PRESSING BUTTONS===
    #=============================================================================================


    lcd_string("Waiting for       ", LCD_LINE_1)
    lcd_string("morse code input...     ", LCD_LINE_2)


    while(0 != GPIO.input(ButtonPin)):
      if (waitTime==0):
        startWaiting=time.time()
      waitTime=time.time()-startWaiting
      print("waiting for user to push button... current wait time is",waitTime)
      if(waitTime>3):
           sadString="No Morse Code Detected"
           return sadString

    waitTime=0


    lcdButtonDisplay=""
    lcd_string("Input detected:         ", LCD_LINE_1)
    lcd_string(lcdButtonDisplay+"                         ", LCD_LINE_2)

    #SECTION 4: "SIMON SAYS" WHILE LOOP RECORDS BUTTON PUSHES UNTIL 14 TU PASS WITHOUT A
PRESS=========================================================
    #=============================================================================================

    dotDashMess='Message: '#these are the first 9 characters. dot/dashes start on



    while(OFFtimer<(16*timeUnit)):#if user fails to blink light for 10 or more time units, our code
    #assumes he is done recording and ends the while loop, thereby proceeding on to the
    #next part where our code will plug the ON/OFF timer arrays into a PARROT array,
    #where the LED will parrot our own button pushes exactly.

    #=============================================================================================
      if (0 == GPIO.input(ButtonPin)):#When button is pushed this if loop runs
        GPIO.output(LedPin, GPIO.HIGH) #turns LED on

        if (OFFtimerStarted != 0):#all info gathered from the previously run elseloop goes here
          offList.append(OFFtimer)
          numButtonReleases=numButtonReleases+1
          if (OFFtimer>(3*timeUnit)):
            print (dotDashMess + '-')
            lcdButtonDisplay=lcdButtonDisplay+" "
            lcd_string("Input detected:         ", LCD_LINE_1)
            lcd_string(lcdButtonDisplay+"", LCD_LINE_2)

        OFFtimer=0
        OFFtimerStarted=0#the off timer values are all reset to 0


        if (ONtimerStarted==0):#recording the time at the moment of the button push
          startTimeOn=time.time()
          ONtimerStarted=1
```

```
        ONtimer= time.time() - startTimeOn
        print("current time elapsed on button push ",numButtonPushes+1," is ",(ONtimer/timeUnit),"time units")

#=============================================================================================
    else:#when button is NOT pushed
        GPIO.output(LedPin, GPIO.LOW) #turns LED off
        if (ONtimerStarted != 0):
            onList.append(ONtimer)#add our latest ONtimer recorded to our on array
            currentTimeOff=0
            #I want to print out dash every time I get a dash and add that to the messageSoFar
            if (ONtimer>(3*timeUnit)):
                print (dotDashMess + '-')
                lcdButtonDisplay=lcdButtonDisplay+"-"
                lcd_string("Input detected:          ", LCD_LINE_1)
                lcd_string(lcdButtonDisplay+"", LCD_LINE_2)
            else:
                print (dotDashMess+ '.')
                lcdButtonDisplay=lcdButtonDisplay+"."
                lcd_string("Input detected:          ", LCD_LINE_1)
                lcd_string(lcdButtonDisplay+"", LCD_LINE_2)
            numButtonPushes = numButtonPushes +1
            print("final time duration recorded for button push",numButtonPushes," is ",(ONtimer/timeUnit)," time units")
        ONtimer=0
        ONtimerStarted=0#the on timer values are all reset to 0

        if (OFFtimerStarted==0):#recording the time at the moment of the button RELEASE
            startTimeOff=time.time()
            OFFtimerStarted=1

        OFFtimer= time.time() - startTimeOff
        print("current time elapsed SINCE button press number ",numButtonPushes+1," is ",OFFtimer/timeUnit," time units")

#=============================================================================================
#=============================================================================================
#=============================================================================================
#SECTION 5: FOR LOOP USED TO PARROT USER BUTTON INPUT (AND TRANSFER VALUES FROM ARRAYS INTO=====
# STRINGS)-- it also performs 2 successive conversions with the aid of functions: first,========
#       1) from button input times into dots/dashes, and then ==================================
#       2) from dots/dashes into letters=========================================================
if (OFFtimerStarted != 0):
    offList.append(OFFtimer)#add one last recorded Off time (since we can't go back the other
#loop to record it once its completed)
    numButtonReleases=numButtonReleases+1

#We now have TWO arrays (lists): onList and offList

dotDashList=[]#This (dotDashList) probably doesn't do anything--from older iteration of code

count=0
letterBreak=99#this (letterBreak) probably doesn't do anything--from older iteration of code

dd1="#this is where our dots and dashes are stored-- it is illegal to have more than 5 in an english morse code char
dd2="
dd3="
dd4="
dd5="
dd6="
ddCombo="

lenOff=0
lenOn=0
```

```
lenOff=len(offList)
lenOn=len(onList)

lastVal=0
lastVal=offList[-1]
if(lenOff>lenOn):
    onList.append(lastVal)
print("length of offlist: "+str(lenOff))
print("length of onList: "+str(lenOn))


for q in range(0, lenOff):

  print("onList value",q," is ", onList[q])
  GPIO.output(LedPin, GPIO.HIGH) #turns LED on
  time.sleep(onList[q])

  count=count+1


  if(onList[q]>=(3*timeUnit)):#this means you've got a dash
    if (dd1 != '.' and dd1 != '-'):
      dd1='-'
    elif (dd2 != '.' and dd2 != '-'):
      dd2='-'
    elif (dd3 != '.' and dd3 != '-'):
      dd3='-'
    elif (dd4 != '.' and dd4 != '-'):
      dd4='-'
    elif (dd5 != '.' and dd5 != '-'):
      dd5='-'
    elif (dd6 != '.' and dd6 != '-'):
      dd6='-'
  else:
    if (dd1 != '.' and dd1 != '-'):
      dd1='.'
    elif (dd2 != '.' and dd2 != '-'):
      dd2='.'
    elif (dd3 != '.' and dd3 != '-'):
      dd3='.'
    elif (dd4 != '.' and dd4 != '-'):
      dd4='.'
    elif (dd5 != '.' and dd5 != '-'):
      dd5='.'
    elif (dd6 != '.' and dd6 != '-'):
      dd6='.'
  if (onList[q]>(3*timeUnit)):
    dotDashList.append(3)
  else:
    dotDashList.append(1)


  ddCombo=dd1+dd2+dd3+dd4+dd5+dd6
  print (ddCombo+' is our letter so far in dashes and dots')


  print("offList value",q," is ", offList[q])
  GPIO.output(LedPin, GPIO.LOW)#turns LED off
  time.sleep(offList[q])

  if (count>5):
```

```
            someText = ddCombo+ ' is not a valid character in morse code.'
            dd1=''
            dd2=''
            dd3=''
            dd4=''
            dd5=''
            dd6=''
            ddCombo=''
            count=0
        if (offList[q]>=(3*timeUnit)):
            convertedLetter=run_new_letter_funcs(ddCombo, convertedLetter)
            convertedWord=add_letter_to_word(convertedLetter,convertedWord)
            dd1=''
            dd2=''
            dd3=''
            dd4=''
            dd5=''
            dd6=''
            ddCombo=''
            count=0
            if (offList[q]>=(7*timeUnit)):
                convertedMessage=add_word_to_message(convertedWord,convertedMessage)
                convertedWord=''
                if (offList[q]>=(14*timeUnit)):
                    convertedMessage=end_message(convertedMessage)
                    convertedWord=''
                    convertedMessage=convertedMessage[9:]
                    convertedMessage=convertedMessage[:-11]
                    return convertedMessage
```

```
#=============================================================================================================
#=============================================================================================================
```

```
#=================================================================================================================
#=================================================================================================================

#When python went from 2 to 3, the decision was made to use the above method for acquiring data online: when a user
#attempts to grab data in this way, what they are actually getting (initially) is going to be saved as an array of
#bytes (rather than as a normal string).

#We want to put this into this method/function (quickGrabAll) which accepts one parameter (the url) and returns a value of fullText
def quickGrabAll(inputUrl,outputText):
    grabBytes= urllib.request.urlopen(inputUrl)
    bText = grabBytes.read()
    outputText = bText.decode("utf8")
    grabBytes.close()
    return outputText



#=================================================================================================================
#=================================================================================================================

#I then want to create a series of functions I call quickGrabStage1, 2 and 3 which are designed specifically to grab info that is
#   1) unique (usually only about one instance on a given webpage)-- and which is also
#   2) NOT specifically designed to be picked up and easily utilyzed and used by a bit computer (like the kind that we might create in
# python)—therefore, this is the work-around.
#Again, the solution here is neat and simple:
#First:

def quickGrabStage1(whatWeWant,textToChange):
    startOfList=""
    locOfWanted=""
    startOfList=textToChange.find(""+whatWeWant)
    return (textToChange[startOfList:])


def quickGrabStage2(whatWeWant2,textToChange2):
    return textToChange2[:45]



def quickGrabStage3(whatWeWant3,textToChange3,textChanged2):
    bookend1=textChanged2.find("(")
    bookend2=textChanged2.find(")")
    return (textChanged2[(bookend1+1):bookend2])

#=================================================================================================================
#=================================================================================================================




def weatherWebsiteStidFinder(City,State,allTheText):
    currentWebsiteString=""+"https://w1.weather.gov/xml/current_obs/seek.php?state="+State+"&Find=Find"
    allTheText=quickGrabAll(currentWebsiteString, allTheText)
```

```
    textCutOffPreText=""
    textCutOffPreText=quickGrabStage1(City,allTheText)

    textCutOffPostText=""
    textCutOffPostText=quickGrabStage2(textCutOffPostText,textCutOffPreText)

    textCutToWithinBookends=""
    textCutToWithinBookends=quickGrabStage3(textCutToWithinBookends,textCutOffPostText,textCutOffPostText)

    print("textChanged3: "+textCutToWithinBookends)

    return textCutToWithinBookends



def quickBackwardsGrabStage1(whatWeWant,textToChange):
    startOfList=""
    locOfWanted=""
    startOfList=textToChange.find(""+whatWeWant)
    return (textToChange[:startOfList])


def quickBackwardsGrabStage2(whatWeWant2,textToChange2):
    return textToChange2[11:]




def simpleXmlDataGrabber(whatWeWant,textToChange):
    startOfList="<"+whatWeWant+">"
    locOfWanted="</"+whatWeWant+">"
    startOfList=textToChange.find(""+whatWeWant)
    return (textToChange[startOfList:])




def givenBookendsFindText(givenText, glimmer, bookend1, bookend2, searchRadius):

    gCount=givenText.count(glimmer)
    print("total gleams left: "+str(gCount))
    if(gCount==0):
        return "No Objects Remaining"

    midPointRainbow=givenText.find(glimmer)

    trailOfGold=givenText[midPointRainbow:]

    bp2=trailOfGold.find(bookend2)

    #print("bp2 is",bp2)
    trailOfGold=trailOfGold[:(bp2)]
    #print("trail of gold is: ", trailOfGold)

    bp1=trailOfGold.find(bookend1)
```

```
    trailOfGold=trailOfGold[(bp1+1):]
    #print("trail of gold is: ", trailOfGold)

    return trailOfGold


print("enter your state abbreviation:")
print("Use a button plugged into GPIO pin 18 to type Morse Code")
print("(if no button is available, please wait a moment)")
print("(conventional input via keyboard will be made available momentarily)")

time.sleep(1)
lcd_string("Enter your state", LCD_LINE_1)
lcd_string("abbreviation in", LCD_LINE_2)
time.sleep(1)
lcd_string("morse code if", LCD_LINE_1)
lcd_string("available.", LCD_LINE_2)
time.sleep(1)







morseReply=""
morseReply=morseCode(morseReply)
print("You entered: ",morseReply)
if (morseReply=="No Morse Code Detected"):
    print("",morseReply)
    allText=""
    currentWebsiteString="https://wtfismyip.com/xml"
    allText= quickGrabAllText(currentWebsiteString,allText)


    myState=givenBookendsFindText(allText,"<your-fucking-location" , ", " , ", United States</your-fucking-location>",45)
    print("my state is: ",myState)

    myStatePlaceholder=","+myState+", "
    #print("state placeholder is: "+myStatePlaceholder)


    myCity=givenBookendsFindText(allText,"<your-fucking-location" , ">" ,str(myStatePlaceholder),45)
    print("my city is: ",myCity)

else:
    morseStateUppercase=""
    morseStateUppercase=morseReply.upper()

    print(""+str(morseStateUppercase))


    morseStateUppercase=morseStateUppercase.replace(" ","")
    morseStateUppercase=morseStateUppercase[:3]
    morseReply=abbrevToFull(morseStateUppercase, morseReply)
    if(morseReply=="NONE"):
        print("You entered: ear",morseStateUppercase,"ear.  This is not a valid choice.")
        allText=""
        currentWebsiteString="https://wtfismyip.com/xml"
        allText= quickGrabAllText(currentWebsiteString,allText)
```

```
        myState=givenBookendsFindText(allText,"<your-fucking-location" , ", " , ", United States</your-fucking-location>",45)
        print("my state is:",myState)

        myStatePlaceholder=","+myState+", "
        #print("state placeholder is: "+myStatePlaceholder)


        myCity=givenBookendsFindText(allText,"<your-fucking-location" , ">" ,str(myStatePlaceholder),45)
        print("my city is: ",myCity)

    else:
        myState=morseStateUppercase
        print("You entered: ",morseStateUppercase,".  This is a valid choice.")

if(morseReply != "No Morse Code Detected" and morseReply != "NONE"):
    morseReply=""

    print("Enter City: ",morseReply)
    lcd_string("Now enter your ", LCD_LINE_1)
    lcd_string("city location: ", LCD_LINE_2)
    time.sleep(2)
    morseReply=""

    morseReply=morseCode(morseReply)
    correctCapitalization=""
    correctCapitalization=morseReply[0]
    correctCapitalization=correctCapitalization.upper()
    correctCapitalization2=""
    correctCapitalization2=correctCapitalization2[0:]
    myCity=str(correctCapitalization)+str(correctCapitalization2)
    if(len(myCity)<=7):
        myCity="London"
    print("You entered: ",morseReply)




lcd_string("Finding location...   ", LCD_LINE_1)
time.sleep(1)
lcd_string("your location is:    ", LCD_LINE_1)
lcd_string(""+str(myCity)+str(myState), LCD_LINE_2)
time.sleep(2)


fullState=""
fullState=abbrevToFull(myState, fullState)
print("full state with dashes: ", fullState)

stateWithoutDashes=fullState.replace("-"," ")

# start flag:New-Hampshire" class="md-crosslink">
#
encycloGrabCities="https://www.britannica.com/topic/list-of-cities-and-towns-in-the-United-States-2023068"
currentWebsiteString="https://www.britannica.com/topic/list-of-cities-and-towns-in-the-United-States-2023068"
```

```
allText=""
allText= quickGrabAllText(currentWebsiteString,allText)

gleamForAllCities=str("-"+fullState+"\" class=\"md-crosslink\"")
print("gleamCheck debug: "+str(gleamForAllCities))
currentCityCount=allText.count(gleamForAllCities)
totalCityCount=currentCityCount
textToChange=allText

stringToFind=gleamForAllCities

stringLoc=textToChange.find(stringToFind)


townsListWithWeatherInfo = ["location","latitude","longitude", "observation_time", "observation_time_rfc822", "temp_f",
"temp_c","relative_humidity", "wind_string","wind_degrees","wind_dir", "wind_mph", "wind_kt","pressure_in","dewpoint_string",
"dewpoint_f", "dewpoint_c", "visibility_mi"]


downSizedText=allText
while(currentCityCount>0):
   downSizedText=textToChange
   print("Current number of cities remaining: ", currentCityCount)
   altCity=givenBookendsFindText(downSizedText,gleamForAllCities , ">" ,"</a>",35)
   print("alt city is", altCity)
   townsListWithWeatherInfo.append(altCity)
   currentCityCount=currentCityCount-1

   stringLoc=textToChange.find(stringToFind)
   timesThroughCycle=totalCityCount-currentCityCount
   print("cycle number: ",timesThroughCycle)

   textToChange=textToChange[(stringLoc+len(stringToFind)):]
   #print("text to change is: ",textToChange)



#print("My full state is: ", fullState)


#print("Full list of towns and weather parameters is: ", townsListWithWeatherInfo)
#print("weather parameters alone: ", townsListWithWeatherInfo[:18])
weatherPara=townsListWithWeatherInfo[:18]

print("weather parameters alone: ", weatherPara)


#print("possible town names alone: ", townsListWithWeatherInfo[18:])
randomTownsList=townsListWithWeatherInfo[18:]
print("possible town names alone: ", randomTownsList)

totalPossibleTowns=0
totalPossibleTowns= len(randomTownsList)
print("the total number of possible towns are: ", totalPossibleTowns, )


myState=myState.lower()
#myState=myState[3:]
#myState=="GB"
#print("my state is ear", myState,"ear")
```

```
print("state right before final test is ", myState, " and city is ", myCity)

#quickBackwardsGrabStage1(whatWeWant,textToChange)
myState=myState.replace(" ","")
print("spaceless state:",myState)

currentWebsiteString=""+"https://w1.weather.gov/xml/current_obs/seek.php?state="+myState+"&Find=Find"


allText= quickGrabAllText(currentWebsiteString,allText)
numTimesOurCityIsMentioned=allText.count(myCity)


cityStidInfo=randomTownsList
townsListInfo19=randomTownsList
sllWeatherStats=randomTownsList
if(numTimesOurCityIsMentioned<1):
   print("our city was mentioned "+str(numTimesOurCityIsMentioned)+" times.")
   print("Therefore you should choose from one of the following options:")

   lcd_string("Results:        ", LCD_LINE_1)
   lcd_string("Site not found       ", LCD_LINE_2)
   time.sleep(1)
   lcd_string("info on site    ", LCD_LINE_1)
   lcd_string("is unavailable  "+str(myCity)+str(myState), LCD_LINE_2)
   time.sleep(2)



   lcd_string("Searching for   ", LCD_LINE_1)
   lcd_string("alternatives:     ", LCD_LINE_2)
   time.sleep(1)
   lcd_string("please choose    ", LCD_LINE_1)
   lcd_string("from this list    "+str(myCity)+str(myState), LCD_LINE_2)
   time.sleep(1)
   lcd_string("of possible      ", LCD_LINE_1)
   lcd_string("alternatives    "+str(myCity)+str(myState), LCD_LINE_2)
   time.sleep(1)
   lcd_string("near your       ", LCD_LINE_1)
   lcd_string("location.      "+str(myCity)+str(myState), LCD_LINE_2)
   time.sleep(1)



   lcd_string(">>Searching...<<", LCD_LINE_1)
   lcd_string("enter y to         "+str(myCity)+str(myState), LCD_LINE_2)
   time.sleep(1)
   lcd_string(">>Searching...<<", LCD_LINE_1)
   lcd_string("select a city       "+str(myCity)+str(myState), LCD_LINE_2)
   time.sleep(1)
   lcd_string(">>Searching...<<", LCD_LINE_1)
   lcd_string("or n to continue       "+str(myCity)+str(myState), LCD_LINE_2)
   time.sleep(1)
   lcd_string(">>Searching...<<", LCD_LINE_1)
   lcd_string("listing options     "+str(myCity)+str(myState), LCD_LINE_2)
   time.sleep(1)

   #lcd_string("enter y to     ", LCD_LINE_1)
   #lcd_string("select a city    "+str(myCity)+str(myState), LCD_LINE_2)
   #time.sleep(2)
```

```
#lcd_string("or n to continue    ", LCD_LINE_1)
#lcd_string("listing options    "+str(myCity)+str(myState), LCD_LINE_2)
#time.sleep(2)


testedCities=0
totalCities=len(randomTownsList)
numValidCitiesCounted=0
while(testedCities<totalCities):

  myCity=randomTownsList[testedCities]
  print(myCity)

  time.sleep(2)
  numTimesOurCityIsMentioned=allText.count(myCity)

  time.sleep(1)
  if(numTimesOurCityIsMentioned>=1):
      numValidCitiesCounted=numValidCitiesCounted+1
      lcd_string("Option "+str(numValidCitiesCounted)+":        ", LCD_LINE_1)
      lcd_string("----------------------------",   LCD_LINE_2)
      lcd_string(""+str(myCity)+", "+str(myState)+"?", LCD_LINE_1)
      lcd_string("(anwer with y/n)", LCD_LINE_2)
  else:
      lcd_string(">>Searching...<<", LCD_LINE_1)
      lcd_string("Please wait...   ", LCD_LINE_2)

  print("This city was mentioned "+str(numTimesOurCityIsMentioned)+" times.")
  townsListInfo19.append(numTimesOurCityIsMentioned)

  if(numTimesOurCityIsMentioned>=1):
    stid=weatherWebsiteStidFinder(myCity,myState, fullText)

    userReply = input("Would you like to select this city instead? ")
    if(userReply=="n" or userReply=="N" or userReply=="No" or userReply=="NO" or userReply=="no"):
      print("ok we will keep checking until we find someplace better")
    else:
      testedCities=totalCities

  else:
    stid="none"

  if (testedCities<totalCities):
    cityStidInfo.append(stid)
    testedCities=testedCities+1



currentWebsiteString="https://w1.weather.gov/xml/current_obs/display.php?stid="+stid
allText= quickGrabAllText(currentWebsiteString,allText)
#KMHT

recordedWeatherStats=0

totalWeatherStatsToRecord=len(weatherPara)

while(recordedWeatherStats<totalWeatherStatsToRecord):
  weatherObs=allText

  startBookend=("<"+weatherPara[recordedWeatherStats]+">")
  endBookend=("</"+weatherPara[recordedWeatherStats]+">")
```

```
        startObs=weatherObs.find(""+startBookend)

        weatherObs=weatherObs[(startObs+len(startBookend)):]

        endObs=weatherObs.find(""+endBookend)

        weatherObs=weatherObs[:endObs]

        #weatherObs=simpleXmlDataGrabber((""+weatherPara[recordedWeatherStats]),allText)


        lcd_string(""+str(weatherPara[recordedWeatherStats])+": ", LCD_LINE_1)
        lcd_string(""+str(weatherObs), LCD_LINE_2)
        time.sleep(2)

        print(weatherPara[recordedWeatherStats], " : ", weatherObs)
        recordedWeatherStats=recordedWeatherStats+1

else:
    print("our city was mentioned "+str(numTimesOurCityIsMentioned)+" times.")
    print("Therefore weather information on this location is available:")

    lcd_string("", LCD_LINE_1)
    time.sleep(1)
    lcd_string("", LCD_LINE_1)
    lcd_string(""+str(myCity)+str(myState), LCD_LINE_2)
    time.sleep(2)
    while(recordedWeatherStats<totalWeatherStatsToRecord):
        weatherObs=allText

        startBookend=("<"+weatherPara[recordedWeatherStats]+">")
        endBookend=("</"+weatherPara[recordedWeatherStats]+">")

        startObs=weatherObs.find(""+startBookend)

        weatherObs=weatherObs[(startObs+len(startBookend)):]

        endObs=weatherObs.find(""+endBookend)

        weatherObs=weatherObs[:endObs]

        #weatherObs=simpleXmlDataGrabber((""+weatherPara[recordedWeatherStats]),allText)


        lcd_string(""+str(weatherPara[recordedWeatherStats])+": ", LCD_LINE_1)
        lcd_string(""+str(weatherObs), LCD_LINE_2)
        time.sleep(2)

        print(weatherPara[recordedWeatherStats], " : ", weatherObs)
        recordedWeatherStats=recordedWeatherStats+1
```

The Code (explanation/Description):

First:

The first hundred lines of code involve me setting up my initial variables for my GPIO pins, and their initial outputs at start. It should be noted that the one part of the code that doesn't work exactly as it would seem to be intended is that the warning lights just go off automatically —they're not actually tied to any of the data. The reason I didn't implement this is simply that I have sunk far too much time into this lab already, and somehow every single time I go into this code try to correct one thing and be done and then submit this lab, somehow something which I think should take 2 minutes ends up with me still sitting there, in the same spot two hours later. As can be seen from the earlier version of the lab that Xavier submitted, this is relatively easy to achieve, and have done it before as part of this lab —I just get sucked back into working on this code. I so desperately wanted this code to be amazi ng, and the fact that it is terrible instead, sucks me back in as well, since I sunk so many hours into it and I don't want them to have been wasted. The fact that every single bit of code (except for the warning lights, which I'm not really bothered by) in here is fully functioning makes it so much worse that I realize now that the only smart thing to do now is to throw it all away.

Second:

The second hundred lines of code just involves grabbing all of the code from the earlier LCD lab (the modified Matt Hawkins' LCD library) so that we can use our LCD's for this lab. Ultimately, I really enjoyed working with the LCD's and having decided to incorporate them into our final project, I'll admit, that I actually have a somewhat difficult time actually justifying why I would do that. For instance, I was actually able to simply *purchase* a fully-functioning 3.5 inch monitor for under $20 that can essentially produce anything that a larger monitor could produce (which is of course *infinitely* higher quality than anything this little 2x16 LCD monitor could do —and that's not *that* much more money than what an LCD screen would cost (~$7-$23 dollars a piece, for a small LCD screen like the one we use). The best, reasonable justification that I could come up with bother here and there, is that the LCD screens we use are far more durable than a miniature monitor, they take up *far* less space (in terms of clogging up the GPIO pins, and if you're trying to design something that small enough to be portable, I seems like it would make whatever you're transporting, durable enough to survive the process. I intended for everything I'm working on here to function like a cheap Casio watch with internet capability— essentially, my thought was, the only thought where something like th is would be useful would be someplace where easy access to expensive, fragile electronics like smart phones and other such devices might not really be reasonable—an example of this being wildland firefighters —or really anyone who works in incredibly dirty jobs, where anything you use will get wrecked no matter what, so it might as well be something cheap and durable.

Third:

Here in lines 200-300 I create a function that converts all state abbreviations into a dashed version of their own name. This will be useful later, as I will show. As seen, it converts anything that isn't a state abbreviation to NONE.

Fourth:

300-700: Here we see a series of functions we created which converts dashes and dots into letters and numbers, and performs another series of functions connected with creating Morse Code. It's all fairly simple and straight-forward stuff.

Fifth:

Between lines 700 and 835 we see the remaining functions defined. These functions allow us to grab data from websites using various methods, depending on the context, and our particular needs.

From lines 835 onward we see how the info we need will be gathered up, and then used by the functions defined earlier. First our code will check to see where your current location is by pulling this info from a website and then saving it. If you wanted to change this, you can also enter a different state abbreviation using Morse code. If no Morse Code is detected, then the code we've created will simply plug the location it gathered from this website into a NOAA website in order to gather weather info on that site, and display it onto the screen (as well as displaying it onto an LCD display). This is done by simply plugging our abbreviation for our state into a URL and then searching through the webpage that we've pulled up *using* the URL with our state abbreviation plugged into it for our specific city. However, if the NOAA website we've pulled up for our state makes no mention of our particular city, then we'll obviously need to extrapolate. This is done by pulling info from *another* website which contains all the major cities for each state within the united states, and searching for each (within our state) one by one. If the user wants to choose one of these sites instead, they can then do so.

Basically, this code should work properly for any state inside the United States (including Alaska and Hawaii). However, not all cities are available, for various complicated reasons —only those listed. Basically, once you've entered the correct state (using Morse Code) —or simply plugged that data into the code itself for testing purposes—our code will take the abbreviation for the state which you entered, and plug it into a URL, search that URL for whichever city might be most useful to us, and then use the info gathered and plug that into a *third* URL which will be created to match up with the specific city within the specific state that we want weather info on. Then our code will display all weather info on this site, as desired, both on our monitor, and on our LCD screen.