Variables and Types (Python)

myint = 7 # if number does not have decimal point, then it is integer type

myfloat = 7.0 # adding decimal point creates a variable of float type

mystring = 'hi there' # strings are specified by enclosing in quotes ' ' or double quotes " "

Python 2		Python 3	
Shell command	Response	Shell command	Response
print type(myint)	<type 'int'>	print(type(myint))	<class 'int'>
print type(myfloat)	<type 'float'>	print(type(myfloat))	<class 'float'>
print type(mystring)	<type 'str'>	print(type(mystring))	<class 'str'>

Working with files in Python

Files can be manipulated using the **open** function. The code below opens a file, reads the contents, and adds content to the file (Python version 2).

```
# Just testing the file open function. This is built-in function so no need to import a module
# There is a file named testing.txt in same directory as this python file


f = open('testing.txt', 'r')                                    # create file-type object named f connected to file named testing.txt
                                                                # the second argument, 'r', specifies that file is to be opened in read mode
print 'for object named f', type(f)                             # print the object type of f so we can confirm it is a file type object
text = f.read()                                                 # read the contents of object f and assign the contents to object named text
print 'for object named text', type(text)                       # print the object type of text so we can confirm it is a string type object
print '\nnow we will print the contents of object named text\n'  # \n is a new line special character
print text                                                      # print the contents of object named text
f.close()                                                       # we must close the file before we open it in a writing mode
f = open('testing.txt', 'a')                                    # create another object named f and open file in append mode 'a'
textAdd = '\nI have added a line'                               # create a string-type object named textAdd
f.write(textAdd)                                                # write the contents of object named textAdd to the file testing.txt
                                                                # since we have opened in append mode, contents will be added to text in file
f.close()                                                       # now we want to read contents of file, but must first close as we are in append mode
f = open('testing.txt', 'r')                                    # create another object named f and open file in read mode
newContent = f.read()                                           # read contents of file and assign to object named newContent
print '\nnow we will print the contents of object named newContent\n'
print newContent                                                # print the contents of the object named newContent
f.close()                                                       # we are done now, so close the file
```

The file named **testing.txt** contains two lines of text: **I am learning Python**, **I am working with open**. When the program on previous page is run, the output in the Python shell is as copied below:

```
Python 2.7.13
>>>
============= RESTART: /home/pi/LaFavre_Python/test_file_open.py =============
for object named f <type 'file'>
for object named text <type 'str'>

now we will print the contents of object named text

I am learning Python
I am working with open

now we will print the contents of object named newContent

I am learning Python
I am working with open
I have added a line

>>>
```

There is another mode, **write**, that can be used with the open function. If we had used write mode instead of append mode, the original lines of the text file would be erased when the line 'I have added a line' was written to the file. Here is proper syntax for using the write mode: **f = open('testing.txt', 'w')**. Another mode is **binary mode**. We used this in the lesson on FTP with Python where a file must be opened in binary mode before uploading to the GEAR web server. Here is syntax for opening in binary mode for reading: **f = open('testing.txt', 'rb')**

If the file you are trying to open is not in the same directory as the Python file, then you must specify the path to the file. Suppose the file **testing.txt** was located in this directory: **/home/pi/python**. Then the proper syntax would be: **f = open('/home/pi/python/testing.txt', 'r')** You can also specify a file name or path as an object. To do this, first create the object, for example, **filePath = '/home/pi/python/testing.txt'** then do the open statement: **f = open(filePath, 'r')**

You can use the open function in a slightly different way using **with** statements. The code on the next page accomplishes the same tasks as the first code example.

```
with open('testing.txt', 'r') as f:
    print 'for object named f', type(f)
    text = f.read()
    print 'for object named text', type(text)
    print '\nnow we will print the contents of object named text\n'
    print text

with open('testing.txt', 'a') as f:
    textAdd = '\nI have added a line'
    f.write(textAdd)

with open('testing.txt', 'r') as f:
    newContent = f.read()
    print '\nnow we will print the contents of object named newContent\n'
    print newContent
```

You might notice that the above code does not include any close statements. That is one advantage of using **with** statements, no need to close when you are done. Notice that lines following the **with** statement are indented. When Python comes to the end of the indented lines, it automatically does a close procedure without the need of a close statement. It is a way of insuring that all items are closed once the lines are executed.

There are additional methods to retrieve part of a text file after you open it. These are covered in the lesson "*Temperature Data Logger with Google Charts*" (http://lafavre.us/robotics/IoT_temperature_data_logger.pdf). Open this lesson and find the function definition: **def get_dallas_data():** Find the lines that use the **.split** method. For example, this code line: **secondline = text.split('\n')[1]** splits the text in the object named **text** at each new line character (\n) and then retrieves the second line of text [1] to store in the object named **secondline**. If we wanted to store the first line of text, then change the code to [0]. Remember, in many programming languages, including Python, the first item is specified as a zero [0] and then the second item as [1], etc.

Loops - while

```
# runs continuously
while True:
   some code here
```

```
# Prints out 0,1,2,3,4
count = 0
while count < 5:
   print(count)
   count = count + 1
```

An example from the lesson *"Temperature Data Logger with Google Charts"*

```
ReadingsToTake = int(input("Enter the number of readings to take: "))    # user inputs the number of temperature readings to take
numReadings = 0
while numReadings < ReadingsToTake:
   numReadings = numReadings + 1                                        # counter to stop loop at specified number
   put code here to take temp reading here
```

Loops – for

For loops iterate over a given sequence. Here is an example:

```
primes = [2, 3, 5, 7]            # this object is known as a list
for prime in primes:
   print(prime)                  # prints out 2, 3, 5, 7
```

```
# Prints out the numbers 0,1,2,3,4
for x in range(5):
   print(x)
```

```
# Prints out 3,4,5

for x in range(3, 6):
   print(x)
```

Using try for error handling (exiting a program)

If your code runs in a while loop until the user presses some key or key combination on the keyboard, then there may be a problem with proper program termination. For example, if you are using a GPIO module to control pins on the RPi and you don't close it properly, you will get a warning message next time you start a program using GPIO pins. What is needed is a way to run some lines of code to close things up when the user closes the program by pressing keys. The code below will do the trick.

Code from lesson *"Controlling Servos with Raspberry Pi"*

"For this exercise we will add a button to the breadboard and use the button to control the position of the servo. When the button is in the up position the servo will rotate to the zero degree position and when the button is pressed, it will rotate to the 90 degree position."

```
try:
   while 1:                #this loop runs continually until user presses control key and c key together
      if GPIO.input(12):      # if button is not pressed then input will be high on pin 12 because we set pull_up_down=GPIO.PUD_UP
         pwm.ChangeDutyCycle(5)       #the duty cycle of the pwm is set to 5%, which at 50 Hz results in 1.0 mS positive pulse
         time.sleep(1) #wait one second then repeat loop

      else:                # button is pressed:
         pwm.ChangeDutyCycle(10)       #the duty cycle of the pwm is set to 10%, which at 50 Hz results in 2.0 mS positive pulse
         time.sleep(1)  #wait one second then repeat loop

except KeyboardInterrupt:           # If CTRL+C is pressed on keyboard, exit cleanly:
   pwm.stop()                       # stop PWM
   GPIO.cleanup()                   # cleanup all GPIO
```

Importing Modules

```
import time
time.sleep(5)        # sleep function is part of time module
sleep(5)             # NameError: name 'sleep' is not defined

from time import*  # * is wild card, import all functions contained in time module
sleep(5)             # sleep for 5 seconds, now works because function was imported

from time import sleep
sleep(5)             # this also works because we specifically imported the sleep function
```

Discovering Functions Contained in a Module (use Python shell)

```
>>> import time

>>> dir(time)

['__doc__', '__name__', '__package__', 'accept2dyear', 'altzone', 'asctime', 'clock', 'ctime', 'daylight', 'gmtime', 'localtime', 'mktime',
'sleep', 'strftime', 'strptime', 'struct_time', 'time', 'timezone', 'tzname', 'tzset']
```

Learn more about a Function in a Module

```
>>> help(time.sleep)

    Help on built-in function sleep in module time:
    sleep(...)
      sleep(seconds)

       Delay execution for a given number of seconds.  The argument may be a floating point number for subsecond precision.
```

Getting user input

Do not enter text in response to an input method without enclosing in quotes unless it is the name of a variable

```
>>> input('enter your input   ')
enter your input   Hi              # I entered the word Hi without enclosing it in quotes

Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    input('enter your input ')
  File "<string>", line 1, in <module>
NameError: name 'Hi' is not defined
```

From the above NameError we can see that Python is complaining that we have not defined the name **Hi**. Python thinks we are trying to use a variable named **Hi** without defining it. So let's try this again:

```
>>> input('enter your input ')
enter your input 'Hi'
'Hi'
```

This time we did not get an error because the response was enclosed in quotes. But suppose we do want to put the name of a variable as the response to an input method. Then we must first define the variable:

```
Hi = 'hi there'
>>> input('enter your input ')
enter your input  Hi
>>>  'hi there'
```

In the case of integers or floating point numbers, you enter your response without quotes:

```
>>> input('enter your input  ')
enter your input  5
5
>>> input('enter your input  ')
enter your input  5.0
5.0
```

Uploading Files to Server

FileZilla – should be installed on your RPi and laptop and configured to connect to GEAR web site

Python script

```
from ftplib import FTP_TLS                    # import function named FTP_TLS from ftplib module
s = FTP_TLS('host name goes here')                    # argument for FTP_TLS function is host name,
                                              # object created is named s
s.login('user name here', 'password here')   # user name and password
s.prot_p()                                    # encrypt all data
s.dir()                                       # list contents of directory on server
s.cwd(put directory path here)                # change directory on server
filename = "test2.html"                       # create a string type variable
file = '/home/pi/LaFavre_web_pages/test2.html'  # create another string type variable
f = open(file, 'rb')                          # open file for reading in binary format
s.storlines('STOR ' + filename, f)            #  upload contents of file to server and apply
                                               # the file name contained in filename to it
                                              # use storlines only for text files, use storbinary for
                                              # other file types (photos, media files, etc.)
f.close()
s.quit()
```

SSH to RPi with FileZilla

Do you know how to connect remotely to your RPi on a network using SSH? Using FileZilla makes it easy.

VNC Remote Desktop for RPi

did you create an account? https://manage.realvnc.com/en/pricing

Controlling a Robot with Raspberry Pi and Arduino

program for Arduino in C language

```
#include <Servo.h>            // include Servo library


Servo myservoRight;          // create servo object to control right motor
Servo myservoLeft;           // create servo object to control left motor

int value;                   // variable to temporarily store data sent by RPi
int varCode;                 // r = right motor l = left motor
int getRight = 0;            // 1  = yes, apply value to right motor
int getLeft = 0;             // 1 = yes, apply value to left motor
int loopLap = 0;             // loop control variable


void setup() {
  myservoRight.attach(9);           // attach right servo to pin 9 of Arduino
  myservoLeft.attach(10);           // attach left servo to pin 10 of Arduino
  Serial.begin(9600);               // start serial communications with RPi at 9600 bits per second
}
void loop() {
  while (Serial.available()) {     // while there is serial data available
   if (loopLap == 0){             // the first character in serial data packet will be code for data identity, the letter r for
                                  // right motor, letter l for left motor, this is read only once, first time through loop
     varCode = Serial.read();     // read first character and assign to variable varCode
   }
   if (varCode == 114 && loopLap == 0) {       // the letter r has the ASCII code value of 114
     getRight = 1;                             // sets loop to get value for right motor
     loopLap = 1;                            // now second and subsequent passes through loop will skip varCode assignment
   }
```

```
  else if (varCode == 108 && loopLap == 0) {            // the letter l has the ASCII code value of 108
    getLeft = 1;                                        // sets loop to get value for left motor
    loopLap = 1;                            // now second and subsequent passes through loop will skip varCode assignment
  }

  if (getRight == 1) {                          // here we collect the number value for pulse length of right motor
    char ch = Serial.read();                    // we already read first character and assigned it to varCode,
                                                // the second and subsequent characters of the serial buffer are
                                                 // read here and accumulate in the variable named value
    if (ch >= '0' && ch <= '9') {           // if character is not between 0 and 9, it is not a number and we will just ignore it
        value = (value * 10) + (ch - '0'); // What we are doing is subtracting the ASCII value of zero (which is 48), from
                                           // the ASCII character value.  In essence we are converting the string number
                                           // character to an  integer. For example, if character was a zero, its  ASCII value
                                           // would be 48 and the expression (ch –'0') would evaluate to  the number zero.
   }

    else if (ch == 10) {     // the ASCII value for line return (\n} is 10. When this character is read, then we have reached the
                             // end of the data packet (we add the  \n in the Python program when sending the motor command).
        setServoRight(value);                       // here we apply the speed value to the motor
        Serial.write("right motor value is ");      // we print out speed value to use as check
        Serial.println(value);
        value = 0;                                  // we must reset this to zero so that it can be used for next serial data
        getRight = 0;                               // reset
        loopLap = 0;                                // reset
      }
}

  else if (getLeft == 1) {                          // here we collect the number value for pulse length of left motor
                                                    // same code below as for the right motor
    char ch = Serial.read();
```

```
    if (ch >= '0' && ch <= '9') {
        value = (value * 10) + (ch - '0');
      }

    else if (ch == 10) {
        setServoLeft(value);
        Serial.write("left motor value is ");
        Serial.println(value);
        value = 0;                          // reset for next serial communication
        getLeft = 0;                        // reset
        loopLap = 0;                        // reset
      }
   }
  }                                         // end while (Serial.available()) loop
}                                           // end void loop ()


void setServoRight(int pulse) {             // pulse = pulse width in microseconds
  myservoRight.writeMicroseconds(pulse);    // sets the servo position
  delay(15);                                // waits for the servo to get there
}

void setServoLeft(int pulse) {              // pulse = pulse width in microseconds
  myservoLeft.writeMicroseconds(pulse);     // sets the servo position
  delay(15); // waits for the servo to get there
}
```

Program for RPi in Python language

```
import serial

#make sure you have the correct serial port specified, it can vary (ttyACM0 in example)
serialMsg = serial.Serial("/dev/ttyACM0", 9600)

while True:

    #user must input first a letter (r for right motor l for left
    #then a number between 600 and 2400
    #all must be in quotes or an error message will appear and program halts

    position = input('input position, for example "r600" ')
    serialMsg.write(position)
    serialMsg.write('\n')
```

End of programming for robot control

Custom Functions (Python)

Functions are pieces of code designed to carry out specific tasks. There are built-in functions (for example the **open** function) and functions contained in modules you might import into your program (for example, the **sleep** function contained in the **time** module). In addition to these functions, you can create your own functions as part of your code. This can be an efficient way to develop your code if you need to repeatedly run a set of code lines.

To create your own function, you use the keyword **def**, as in define. Suppose we want to create a function named **add2numbers**. We could do that with this line:

```
def add2numbers(a, b):
```

Immediately following the name you have selected for the function you must have an open (and close) parentheses and a colon : In some cases you will want to pass on some variables when the function is called. If not, then leave the space between parentheses blank. In the example above, we have specified two arguments in parentheses, a and b. This means that when the function is called, we need to provide some values for the two arguments. Now let's take a look at a simple function designed to add two numbers.

```
def add2numbers(a, b):
   answer = a + b
   return answer

myAnswer = add2numbers(3, 6)
print  myAnswer
```

The program execution starts with the line beginning with myAnswer. Here is where the function named **add2numbers** is called. Notice that two numbers are specified in the parentheses: (3,6). These are the two variables that are passed to the function. Inside the function, the value 3 will be applied to variable named a and the value of 6 will be applied to the variable named b. As the function executes, the values contained in a and b are added together and the result is stored in the variable named answer. Then the function returns the value contained in the variable answer to the line calling the function, where the value is stored in the variable named myAnswer. Lastly, the program prints out the value stored in myAnswer.

Depending on the function, you may or may not need to return a value when the function has completed execution. However, the last line of the function must contain the keyword return. If nothing is to be returned, then just use the word return by itself. Notice that the lines of the function are indented. After the line containing def, all subsequent lines of the function must be indented.

While the program on the previous page is useful in understanding how to create a function, the function created is not very useful. It would be tedious to have to edit the program every time we want to add two different numbers together. Let's see if we can improve the utility of the program.

```
def add2numbers(a, b):
   answer = a + b
   return answer

print 'Welcome to the addition program\n'
print 'This program adds two numbers\n'

while True:
   c = input('enter the first number   ')
   d = input('enter the second number   ')
   myAnswer = add2numbers(c, d)
   print c, ' + ', d, ' =  ', myAnswer
```

You will notice that the definition of the function has not changed in this second version of the program. It is the code outside the function that makes it more useful. Take a look at the line beginning with myAnswer. Notice that now, when the function add2numbers is called, two variables are specified (c, d) rather than two numbers. The numbers contained in those two variables are numbers input by the user. Also notice the while loop, which provides for repeated additions of two numbers. As the program executes, two lines are printed: 'Welcome to the addition program' and 'This program adds two numbers'. Then the program enters the while loop and the user is asked to input a number. After the user inputs a number, he or she is asked to input a second number. Then the program calls the add2numbers function, passing to the function the values contained in the variables c and d. Inside the function the value in c is applied to the variable named a and the value in d is applied to the variable named b. Then a and b are added together and the result applied to the variable named answer. The value contained in answer is returned to the line calling the function where the value is applied to the variable named myAnswer. The last line of code results in the printing of the values contained in variables c and d and the value in myAnswer in this form: 2 + 3 = 5. After printing, execution of the program returns to top of the while loop where the user has the opportunity to enter more numbers to be added. Program execution is halted by pressing Ctrl + C on keyboard.

Here is a function named **get_dallas_data**, from the *Temperature Data Logger with Google Charts* lesson. We can start to appreciate the utility of grouping a set of lines into a custom function. There are 13 lines of code that accomplish these tasks: **1)** instruct RPi to take a temperature reading **2)** open the file containing the temperature data **3)** extract from that file only the temperature (there is other data in the file) **4)** format the temperature correctly (divide by 1000) **5)** convert temperature from Centigrade to Fahrenheit **6)** round temperature to one place past the decimal point.

```
def get_dallas_data():                                          # function retrieves a temperature reading from sensor
  subprocess.call(['modprobe', 'w1-gpio'])                      # Raspberry Pi takes a temperature reading
  subprocess.call(['modprobe', 'w1-therm'])
  filenameD = "/sys/bus/w1/devices/28-0000093b782d/w1_slave"    # w1_slave is the file containing the temperature value
  tfile = open(filenameD)                                       # open the file that contains the temperature
  text = tfile.read()                                           # add the contents of the file to an object named text
  tfile.close()                                                 # now that the file has been read, close it
  secondline = text.split("\n")[1]                              # spilt the text at the line break (\n), contents of second line
                                                                # assigned to object named secondline
  temperaturedata = secondline.split(" ")[9]                    # split the line at each space, add contents of tenth
                                                                #segment to object named temperaturedata
  temperature = float(temperaturedata[2:])                      # add all but first two characters of temperaturedata to
                                                                # object named temperature, which will be float number
  temperature = temperature / 1000                              # temperature needs to be divided by 1000 to place
                                                                # decimal point
  temp = float(temperature)                                     # make temperature a float number assigned to object named temp
  temp = temp*1.8+32                                            # convert temperature from Centigrade to Fahrenheit
  tempDallas = str(round(temp,1))                               # the object tempDallas will contain the temperature as a string,
                                                                #rounded to one place past the decimal point

  return tempDallas                                             # return the temperature to the line calling the function
```

(go to next page for more code)

The **get_dallas_data** function is called within another function named **get_sense_data**.

```
def get_sense_data():                                   # function to collect the time and temperature
  sense_data=[]                                         # list type object used to store data
  dateNow = str(datetime.now())                         # get day and time and store in dateNow
  sense_data.append("['" + dateNow[11:16] + "'")        # add contents of dateNow to the sense_data object,
                                                        # contains day and time, so we need to trim
                                                        # to get just the time [11:16]
  sense_data.append(get_dallas_data() +"],")            # now add the temperature to the object sense_data
                                                        # we need to end the data line with the characters ],  to
                                                        # keep the graphing code correct.
  return sense_data                                     # return the time and temperature to the line calling the function
```