As you have already discovered, the Raspberry Pi is a very capable digital device. Nevertheless, it does have some weaknesses. For example, it does not produce a clean pulse width modulation output (unless you use the hardware PWM mode, available only on one pin). If you have completed the lesson on connecting a servo to the RPi, then you may have noticed that the servo "jitters." This is due to a slight variation in the pulse width the RPi produces. Another shortcoming is the lack of built-in support for input of analog signals (RPi GPIO only supports digital inputs).

We may be able to improve on the performance of the Raspberry Pi by connecting it to an Arduino. With the combination, tasks best handled by a computer (or that can only be handled by a computer) are assigned to the Raspberry Pi. The Arduino can then be used as a **microcontroller** for various connected devices (servos, etc.).

The language used to program the Arduino is a version of C language. After downloading the IDE software for Arduino to your RPi, you can use the RPi to write programs for the Arduino. The two devices are connected by a USB cord (type A on one end type B on the other). The USB cable provides a conduit for uploading programs from the RPi to Arduino and for two-way communication between the two devices during operation.

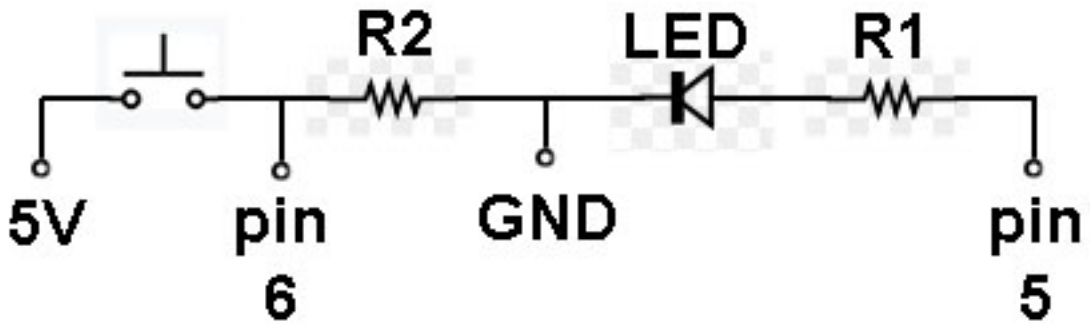
Install Arduino IDE on the RPi

Open the terminal in your Raspberry Pi and enter these commands.

```
sudo apt-get update
sudo apt-get dist-upgrade
sudo apt-get install arduino
```

Wiring Arduino with breadboard

Use the schematic below to wire the Arduino for this lesson



Resistor R1 should be between 500 and 1000 ohms, R2 should be 10K ohms.

Writing your first Arduino program

After you have finished wiring the Arduino, you are ready to write your first program for the device.

1. Open the Arduino IDE on your RPi (it will be in the Programming menu).
2. Before you start writing the code, save the file with the name LED_ON_OFF
3. Enter the code provided below (text following // on a line is a comment)

```
int led1 = 5;        //make variable named led1 and store a value of 5
int buttonPush = 6;   //variable named buttonPush, store a value of 6
int wait = 500;       //variable named wait with value of 500
void setup() {
  pinMode(led1, OUTPUT);  // set pin 5 as output type
  pinMode(buttonPush, INPUT);   // set pin 6 as input type
}
void loop() {
  if (digitalRead(buttonPush) == HIGH) {   //if button is pushed
    digitalWrite(led1, HIGH);  //make pin 5 output 5 volts (HIGH)
    delay(wait);              //wait for 500 milliseconds
    digitalWrite(led1, LOW);  //make pin 5 output 0 volts (LOW)
    delay(wait);
  }
}
```

Synopsis of above program

While the momentary switch is pressed on the breadboard, the LED will flash on and off (0.5 seconds on, 0.5 seconds off). When the switch is not pressed (open), pin 6 is at ground potential (0 volts). In this condition pin 6 is LOW. When the button is pressed, the switch is closed and pin 6 is HIGH because it is fed by the 5 volt source of the Arduino. Take a look at the if statement. buttonPush is the name applied to pin 6 and when pin 6 is HIGH, then the code under the if statement runs, making the LED blink on and off.

Preparing to run the code on Arduino

1. After you are finished entering the code lines, save the file
2. Connect a USB cable between the RPi and Arduino
3. Verify and compile your code by selecting **Verify/Compile** in the **Sketch** menu of the Arduino IDE software. If any errors occur, you must fix them and do step 3 until verification is successful.
4. Upload the program to the Arduino (click the button with arrow pointing to the right)

Running your program on the Arduino

After you have uploaded the program to the Arduino, it will start to run. The Arduino is powered by the RPi through the USB cable (notice the green LED on the Arduino, which should be on). Press the momentary switch on the breadboard and the LED should begin to flash. It will continue flashing on and off as long at the button is pressed.

Congratulations! You have finished your first Arduino program. Now you will learn how to write Arduino and Python programs that work together to pass information between Arduino and RPi.

Serial Communication between Arduino and RPi

You have already used a USB cable to upload a program to Arduino. Now you will use the same cable as a means of communication between the two devices while they run programs. This particular type of communication is called serial and Python and Arduino programming languages have methods for establishing serial communications.

To begin, we need to identify the port designation for the USB connection:

1. Disconnect the USB cable between RPi and Arduino if connected
2. In the **Terminal** of RPi run this command: **`ls /dev/tty*`**
3. In the terminal will be a list of serial connections, which WILL NOT include the connection to Arduino
4. Connect the USB cable between RPi and Arduino
5. In the **Terminal** of RPi run this command again: **`ls /dev/tty*`**
6. Now you should notice one additional item. This is the name of the port connecting to the Arduino. The name will likely be: `ttyACM0` . **Write down the port name as you will need it later in the lesson when writing code.**

Create an Arduino program for serial communication with RPi

Using the Arduino IDE, open the file named LED_ON_OFF (if it is not already open). Using SAVE AS, save the file with the name **serial_LED_ON_OFF**. Then enter the code on the next page. When you are done, save and verify your code.

```
int led1 = 5;
int buttonPush = 6;
int wait = 500;

void setup() {

  pinMode(led1, OUTPUT);
  pinMode(buttonPush, INPUT);
  Serial.begin(9600);  //start serial communications with RPi

}


void loop() {

  if (digitalRead(buttonPush) == HIGH) {

    Serial.println("Button is pressed!");  //send message to RPi

    digitalWrite(led1,HIGH);
    delay(wait);
    digitalWrite(led1,LOW);
    delay(wait);

    }

  }
```

Synopsis of above Arduino program

The above code is very similar to the first Arduino program. In **void setup()** there is the new line: **`Serial.begin(9600)`**. This initiates a serial connection at a baud rate of 9600 (9600 bits per second). The other new line is found in the **void loop()** section: **`Serial.println("Button is pressed!")`**. This sends out over the serial line the text characters enclosed in quotes. Now we need a Python program running on the RPi to capture the text message.

Create a Python program for serial communication with Arduino

Using Python 2 IDE, create a new file named **serial_LED_ON_OFF-pi_side**. Enter the code on the next page and save.

```
import serial

#if your port name is not ttyAMC0, then edit line below
serialMsg = serial.Serial("/dev/ttyACM0", 9600, timeout=1)

while True:

    rawMsg = serialMsg.readline()

    message = (rawMsg.decode().strip())

    if (message == "Button is pressed!"):

        print"Arduino says: Button is pressed!"
```

Synopsis of above Python program

First, we must import the Python serial library. In the second line we use the **.Serial** method (make sure the S is uppercase). This method establishes a serial connection to the Arduino so that RPi can listen for messages. The port for serial communication is **`/dev/ttyACM0`** (unless you have determined otherwise). The baud rate is specified as 9600, which matches the Arduino program. The timeout value is set to one second.

When the RPi receives text over the serial line it stores it in the object named **`rawMsg`** using the **`.readline()`** `method.` The contents of **`rawMsg`** contain some extra bits added by the serial protocol, which we don't want to appear in our message. The line containing **`(rawMsg.decode().strip())`** strips away the unwanted characters and delivers the text message. If the text message matches "Button is pressed!," then that is printed out in the Python console.

Running the Serial programs

Upload the Arduino serial program to Arduino. On the RPi, run its serial program. Then press the momentary switch on the breadboard. The LED should flash and the "Button is pressed!" message should appear in the Python console on RPi.

Congratulations! Your Arduino sent a message to your RPi.

Now you will create a pair of programs to send a message the other way, RPi to Arduino.

Second pair of serial communication programs

Create an Arduino program with the code on the next page. Save it and verify it. You can just edit the previous Arduino program and save with a new name.

```
int led1 = 5;
int buttonPush = 6;
int wait = 500;


void setup() {
  pinMode(led1, OUTPUT);
  pinMode(buttonPush,INPUT);
  Serial.begin(9600);
}


void loop() {
  if (digitalRead(buttonPush) == HIGH) {
    if (Serial.available()) {
      flash(Serial.read() - '0');
    }
  }
  delay(5000);
}


void flash(int repetitions) {
  for (int i = 0; i < repetitions; i ++){
    digitalWrite(led1,HIGH);
    delay(wait);
    digitalWrite(led1,LOW);
    delay(wait);
  }
}
```

Synopsis of above Arduino program

Note that a function named **flash(int repetitions)** has been added to this program. Also note that this function is called in the **void loop()** section: **`flash(Serial.read() - '0').`** The Python program running on RPi (which you will create next) sends a number to the Arduino. This number is applied to the argument named repetitions of the flash() function. The value contained in repetitions determines the number of times the for loop inside flash runs, which in turn determines the number of times the LED will flash when the button is pressed.

Create the Python program to work with Arduino program on page 6

```
import serial
serialMsg = serial.Serial("/dev/ttyACM0", 9600)
while True:
    number = bytes(input("Enter number of times to flash ..."))
    serialMsg.write(number)
```

Synopsis of above program

In this program the **input** method is used so that you can enter a number for the number of times you wish the LED to flash . The **.write** method of **serial.Serial** specifies that the attribute inside the parentheses must be of the **bytes** type. Therefore, the number entered by the user in the input method is converted to the bytes type by enclosing the input in parentheses and adding the word bytes to the front of it. The value is stored in the object named **number**. That value is sent over the serial line to the Arduino.

Run the second serial program

1. Upload the Arduino program
2. Run the Python program on the RPi
3. Enter a number in the Python console when asked and then press Enter to send that number to the Arduino
4. Press the button on the breadboard and the LED should flash the number of times you entered on the RPi.

Congratulations! You have now learned how to send data both ways between an Arduino and RPi.. Now you will work on a program to control a servo connected to the Arduino.

Connecting a servo to the Arduino and controlling it with RPi

In my testing, it appears that the Arduino produces a cleaner PWM signal than the RPi. The servo does not jitter. In this exercise you will confirm this observation (hopefully).

Wiring the servo

You will be using a Hitech brand servo, model HS-485HB. GEAR has a number of these in its inventory. The Arduino is powered through the USB cord connected to the RPi. If we expect the RPi to also power a servo, there might be some power problems (two servos would definitely be a problem). To prevent power problems, you will use an additional power supply to power the servos. I have made up some power supplies for you to use. They hold 4 AA batteries and if you use rechargeable ones at 1.3 volts each, the supply will deliver 5.2 volts DC with a fresh charge. The HS-485HB servo can be powered with DC in the range of 4.8 to 6.0 volts, so this power supply should work well.

The servo has three wires in its cable: yellow is the signal wire, which should connect to digital pin number 9 on the Arduino. The red wire is connected to the positive side of the 5.2 volt battery supply

and the black wire to the negative side of the battery supply. Connect a wire from the negative side of the battery supply to a ground (GND) pin on the Arduino. It is important for all power supplies to share the same ground to eliminate electrical noise problems. This completes your wiring. I suggest you do not insert all the batteries in the power supply until you are ready to test your programs.

Write the Arduino program to control servo

```
#include <Servo.h>  //include Servo library`
Servo myservo;  // create servo object to control a servo
int value;  //variable to store number sent by RPi

void setup() {
  myservo.attach(9);  // attach myservo object to pin 9
  Serial.begin(9600);   //start serial communications with RPi
}

void loop() {
  if (Serial.available()) {  //if there is a serial connection
     char ch = Serial.read(); //read character and assign to ch
     if (ch >= '0' && ch <= '9') { // is this a digit between 0 and 9?
        value = (value * 10) + (ch - '0');  //yes, add it to value
     }
     else if (ch == 10) {  //is the character the newline character?
        setServo(value); //run setServo function with value
        value = 0;  // reset value to 0 ready for next sequence
     }
   }
}
void setServo(int pulse) {  // pulse = pulse width in microseconds

  myservo.writeMicroseconds(pulse);   // sets the servo position
  delay(15);                          // waits for the servo to get there
}
```

Synopsis of above program

You should read the comments I added to the program lines (text following //). The code above allows the Arduino to receive a string of number characters from the RPi Python program (which you will write next). That number string determines the rotational position of the servo. To control the servo, we use the method named **.writeMicroseconds(),** which requires its argument in microseconds. The servo can accept pulse widths from 553 to 2425 microseconds (553 is fully counterclockwise and 2425 is fully clockwise). The range of rotation between the two extremes is 190.5 degrees. The program above controls the servo by sending pulses between 600 and 2400 microseconds, which gives approximately 180 degrees of rotation on the servo. By entering a value of 600 in the RPi Python console, the servo will rotate nearly to the limit of its rotation in a counterclockwise direction. By entering a value of 2400, the servo will rotate nearly to the limit of its rotation in the clockwise direction. For numbers between 600 and 2400, the servo will rotate to a position proportional to the number value.

Writing the Python program to control the servo

Using Python version 2, write the code below.

```
import serial
serialMsg = serial.Serial("/dev/ttyACM0", 9600)
while True:
    number = str(input("Enter number between 600 and 2400 ..."))
    number = number + '\n'  // the \n is end of line character
    serialMsg.write(number)
```

Synopsis of above program

After importing the serial library on the first line, a serial object named **serialMsg** is created, which represents a serial connection to the Arduino. The **input** method is used to retrieve input from the user. We use **str** in front of the input to convert the user input to the string type. Then we add the **end of line character** to the end of the string (\n). The end of line character is used in the Arduino program to detect the end of the string. The Serial **.write** method is used to send the string of number characters, including the end of line character, to the Arduino.

Testing the servo control

Place the four batteries in the power supply. Upload the Arduino program to the Arduino. Start the Python program on the RPi and enter a number between 600 and 2400. Enter various numbers between 600 and 2400 and note how the servo responds.