An Introduction to Programming with Romibo and Arduino:

by

-Adam Warburton

-etc.

The following will apply to all programs/sketches found in this tutorial:

// This is free software; you can redistribute it and/or modify

// it under the terms of the GNU General Public License as published by

// the Free Software Foundation; either version 2 of the License, or

// (at your option) any later version.

//

// This is distributed in the hope that it will be useful, but

// WITHOUT ANY WARRANTY; without even the implied warranty of

// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

// GNU General Public License for more details.

//

// You should have received a copy of the GNU General Public License

// along with this file; if not, write to the Free Software Foundation,

// Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

// ##END_LICENSE##

*/**/*

*/**/*

What are Romibo and Arduino anyway?

An Arduino is a single-board microcontroller based on open-source hardware. Essentially an Arduino Board is a small computer that can be programmed to send and receive electrical signals. The Arduino is a computer, just like a laptop or desktop, but instead of being used for word-processing or web-surfing, can be used in the design of robots and other devices. It also does not have anywhere near the processing power of a desktop computer, but is significantly cheaper and can interact with the physical world. An Arduino board can be programmed through a connection by USB to a computer.

Arduino Boards in themselves can only output weak electrical signals (5 Volts at an approximate maximum of 40 mA). Consequently, the board itself can do little more on its own than turn on LEDs. For this reason, Arduino boards are often connected to shields such as the Romibo Shield.

The Romibo Shield is a board that takes input from the Arduino Board and outputs the appropriate output voltages to the motors, servos, and other components on the Romibo Robot. The Romibo Shield also allows the Arduino to access input from numerous sensors including an accelerometer, microphone, IR sensor, two photo resistors and more. Table 1.1 contains the pins of the Arduino board with names for what they correspond to on the Romibo Shield.

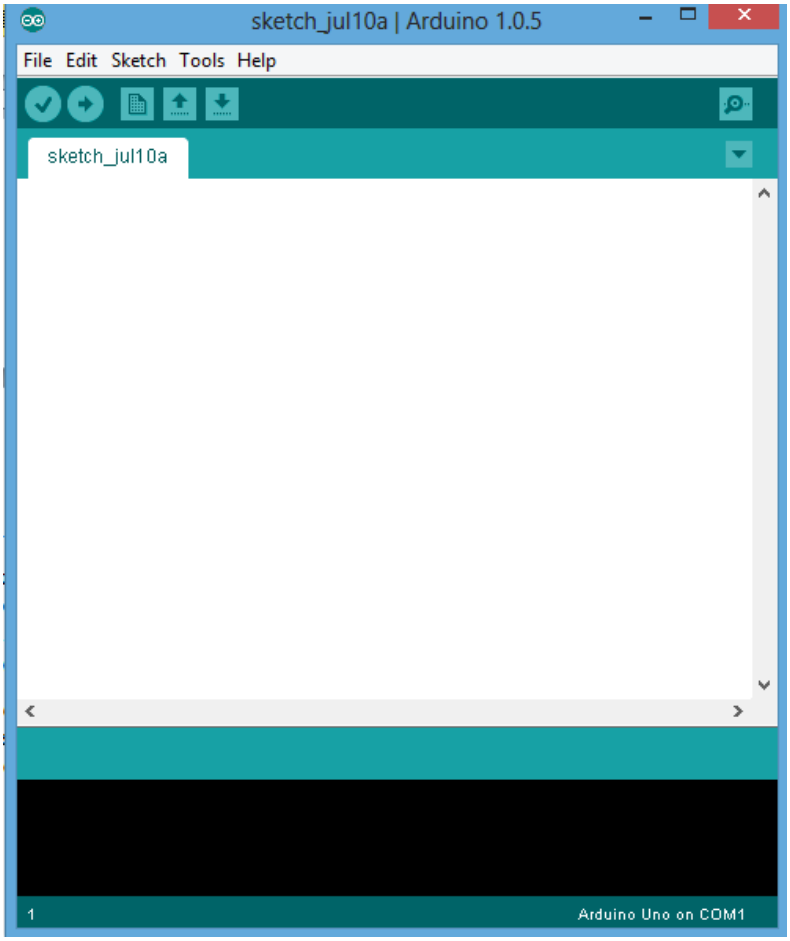
Table 1.1 (modified from Romibo-Arduino-Pin-Assignments.xls by Garthz)

Arduino Pin	Rev3 Signal	Rev3 Function
A1	RANGE	range sensor input
A2	PHOTOLEFT	left photosensor input
A3	PHOTORT	right photosensor input
A4	PHOTOTOP	top photosensor input
A5	MIC	microphone signal input
A6	ENVELOPE	microphone envelope input
A7	PHOTOFACE	face photosensor input
A8	LCURRENT	left drive motor current feedback input
A9	RCURRENT	right drive motor current feedback input
A10	VINSENSE	battery input voltage input
D2	FWDLEFT	left drive wheel IN1
D3	REVLEFT	left drive wheel IN2
D4	RED	red led PWM
D5	MOTSLEEP	drive motor sleep output
D6	MOTFAULT	drive motor fault input
D7	FWDRT	right drive wheel IN1
D8	REVRT	right drive wheel IN2
D9	GRN	green led PWM
D10	BLUE	blue led PWM
D13	AUDIOPWM	experimental class-D audio amplifier PWM output
D18	WIFITX	Serial1 data to WiFi module
D19	WIFIRX	Serial1 data from WiFi module
D20	SDA5V	I2C bus data
D21	SCL5V	I2C bus clock
D28	TOUCHLEFT	touch switch digital input
D29	TOUCHBOT	touch switch digital input
D31	TOUCHTOP	touch switch digital input
D34	!DACS!	audio SPI chip select out
D35	DASCK	bit-banged audio SPI clock out
D36	DASDI	bit-banged audio API data out
D37	!SPKSHDN	active-high enable for speaker amplifier
D40	!SDCD!	SD Card detect
D45	TILTBACK	servo PWM for rear neck servo
D46	EYE	servo PWM for eyelid servo
D48	IRIN	IR remote receiver input
D50	SDDO	hardware SPI for SD Card
D51	SDDI	hardware SPI for SD Card
D52	SDSCLK	hardware SPI for SD Card
D53	SDCS	digital I/O for master-mode SPI

Installing the Arduino IDE

Now that you've learned a little about the hardware, now would be a good time to start learning about the software. As stated earlier, the Arduino board can be programmed using a USB port. In order to begin programming with Arduino, you must first install the Arduino IDE on your computer. The Arduino IDE is available free of charge from http://arduino.cc/en/main/software

Please install the version that is appropriate for your device (Windows, Mac, Linux) and then restart your computer if necessary. After a successful installation, you will be able to start the program which should appear similar to that of Figure 1 below.



The first thing you will likely need to do is to change your IDE to work with the Arduino Mega. This is a fairly easy step, simply click **Tools** and then click on **board** in the drop down menu and then choose the **Arduino Mega 2560**.

Then be sure to plug the robot into the USB port. Then click **Tools** and then **Serial Port**. Choose the only serial port available (example COM3). After that you should be ready to start programming.

Program 1:

Your First Sketch - Getting the Robot to Move

The Arduino Community refers to programs as Sketches. This section will guide you through the process of writing sketch that causes the robot to move forward, stop, then repeat. Arduino programming is based off of the open source programming language "Processing". "Processing" shares similarities with other programming languages such as C++ and Java.

Every Arduino sketch must have two parts:

1. A **setup** function
2. A **loop** function

The **setup** function is self-explanatory. It sets up the Arduino Board. This function is run once and is important because it is used to tell the Arduino which pins to use for input and which to use for output.

The **loop** function is a function that is run after the setup function. Unlike the setup function however, the loop function will run an infinite number of times. It repeats or *loops* the function over and over again.

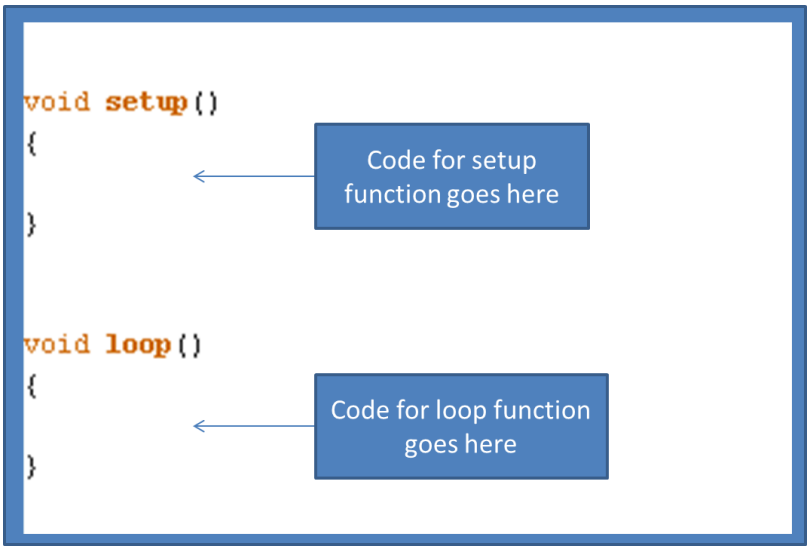
To write the outline of these bits of code, type the following:

```
void setup()

{

}

void loop  ()

{

}
```

You will notice that the text changes colors. This is to help with programming as different types of code will be different colors. It is very important to keep the capitalization and punctuation the same. Brackets {} are not the same as parenthesis () .

Do not worry too much about the word “void” for now, we will discuss that later. The functions are void because there is no value of a number or character returned by them.

If you typed the code in correctly, it will appear like what is seen in Figure 2 below. Notice that the color of the text turned orange. The space in-between the brackets is where the code for each of the functions will go.



As mentioned earlier, the setup function will tell the Arduino which pins to use as input and output. For this particular program we will only be using output to the motors. There are four pins which will be important to us. These are Digital Pins 2, 5, 6, and 7.

Pins 2 and 7 tell the left and right motors to move forward respectively.

Pins 5 and 6 correspond to the motor sleep and motor fault switches.

Motor Sleep and motor fault may sound foreign to you. Comparing them to every-day life, a person does not simply get into a car, put their foot on the gas and start moving. One has to first turn the key to start the car. One also has to shift gears from park to drive. The motor sleep switch will need to be triggered to essentially start the motor (turn the key). The motor fault switch can be compared to putting the car in drive. To enable these pins, the corresponding pins will first need to be declared as output. To do this, type the following code into the setup function (*between the brackets {} after void setup()*).

```
pinMode (2, OUTPUT);

pinMode (5, OUTPUT);

pinMode (6, OUTPUT);
```

```
pinMode (7, OUTPUT);
```

Note that there is a semicolon ";" after each line. This semicolon tells the compiler (the program on your computer in which you typed this) that you have typed a complete statement. The semicolon is similar to punctuation in speech. Once you finish a statement, you end with a period. Otherwise, sentences would be run-on jumbled messes of difficult to understand nonsensical word jumble that is really really really long, confusing, hard to decipher, and did I mention confusing because if I didn't I should have.

All joking aside, look at the statements that were typed. Each one of them has the pinMode function. This function tells Arduino that the pin number that follows should be either an input or an output. The number within parenthesis is followed by a comma and then the word INPUT or OUTPUT in all capital letters. The ending parenthesis is then added followed by a semicolon.

Now that we have declared the pins to be outputs, it would be good to set the MOTSLEEP and MOTFAULT pins to output a signal so that we have essentially turned the key to our motors and set them to drive. To do this, type the following code into the setup function (between the brackets) but after the code that set the pins to be outputs.

```
digitalWrite (5, HIGH);

digitalWrite (6, HIGH);
```

Again, don't forget the semicolons! Also, please make sure that the capitalization is correct.

What you've just typed is the digitalWrite function. This function is digital because it is outputting to a digital pin (Arduino also has analog pins which will be discussed later). A digital pin is either on or off. To turn a pin on, it is written to as HIGH for a high signal being sent. To turn a pin off it will be sent a LOW signal. Just like the pinMode function, the number used in-between the parenthesis corresponds to the pin that the data is being sent to.

Now that you have successfully set up your Arduino, it's time to move on to the loop function.

To get the robot to move, you will use the same digitalWrite function that you used to turn the MOTFAULT AND MOTSLEEP switches on. See if you can write these bits of code on your own for pins 2 and 7 in the loop function.

The next function you will need to know about is the delay function. The Arduino board will move through the code you type at an incredibly rapid rate (Thousands of lines per second). To get the board to slow down, you will need to tell it to wait. The delay function has one parameter

(value within parenthesis) that tells it how many milliseconds to delay the program. Consequently to have the board wait one second or 1000 milliseconds you should type the function

delay (1000);

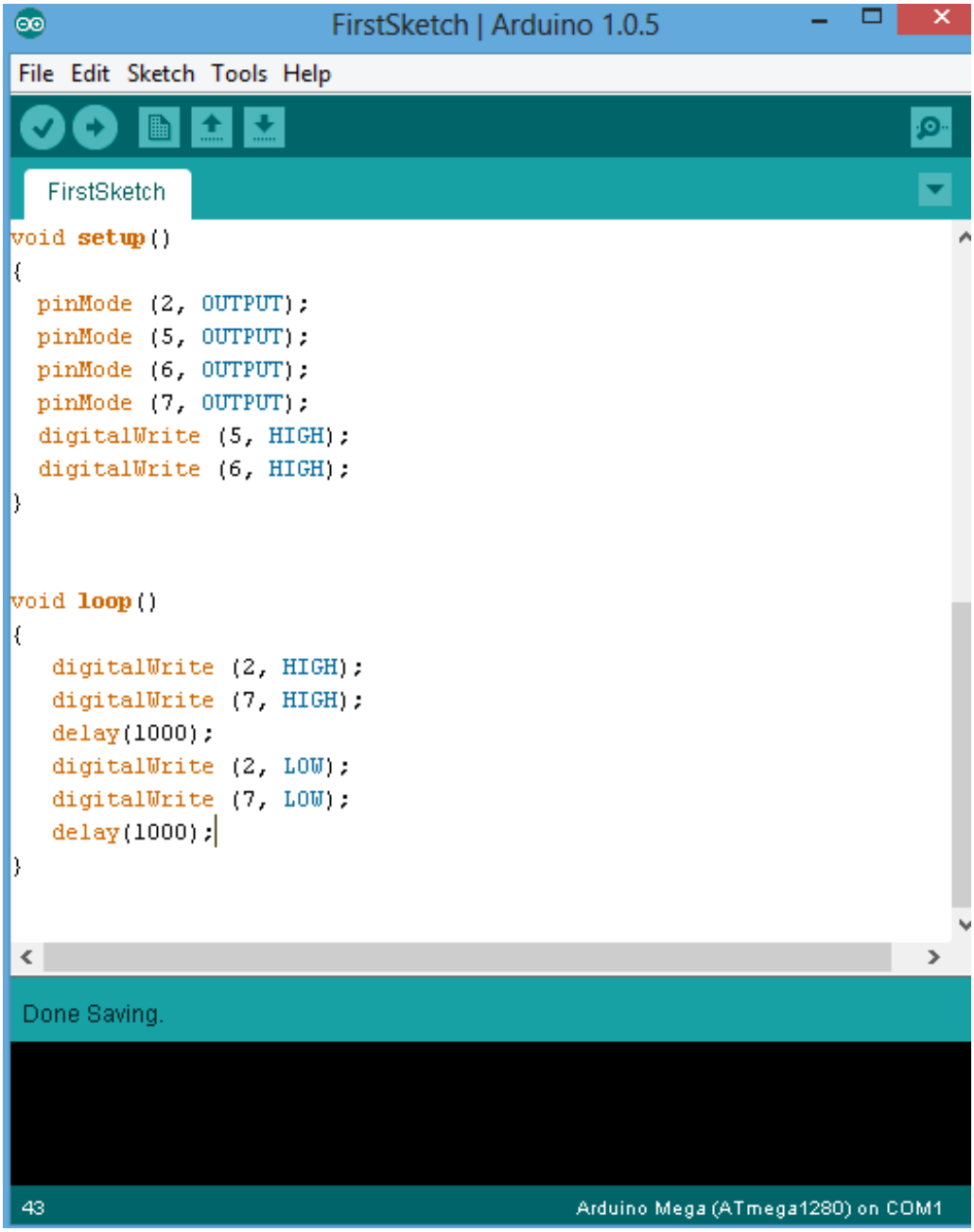
This will cause the Arduino to wait a full second before moving onto any code that follows.

As of now, you should have a loop that turns the motors on, waits a second, and then turns the motors on again, then waits another second and so on. But this isn't especially useful if you want the robot to take a pause for a while. Therefore, you will need to tell the motors to turn off as well. This can be done with the digitalWrite function. Only this time, you will be using LOW instead of HIGH.

```
digitalWrite (2, LOW);

digitalWrite (7, LOW);
```

This will turn the motors off after a one second delay. However, the motors will immediately be turned back on again when the loop function repeats. Therefore you should add another delay of a time of your choosing after turning the motors off.

Overall, your program should look something like the following in the compiler.



To check if your program was written with proper syntax, click on the check mark at the top of the program. This will compile your program. This means that it will translate the code you typed into something that can be understood by the Arduino board. If the code is written so that the computer can understand it, the compiler will show a "done compiling" message. Otherwise the compiler will inform you of errors.

Just because a program compiles does not mean it will do what you want it to do. To check to see if your sketch works, you will need to upload it to your Romibo. To do this, connect your

Romibo to your computer using a USB cable. Then click on the upload arrow next to the check mark you just used to compile your sketch.

(Depending on your machine, you may have some errors the first time you connect the robot. For help connecting your computer to the Arduino, there are multiple pages on the internet that can be found from your favorite search engine.)

If everything was successful, you will see a message that says "done uploading". You may now unplug your robot and set it on the floor (away from any obstacles). Turn on the robot and see if it behaves as you expected.

Did the robot do what you expected?

Hopefully it did. If not, there are a few possible reasons it did not. If the robot did not move at all, the motors may not be properly connected or the code may need adjusted. The batteries in the robot may also need replaced.

Did your robot move backward? If so, the motors were installed backward. This isn't as huge of a problem as you might expect. One solution is to open the robot and move the motors. The other is to simply alter your program. In the code you can replace the pin numbers you used for the motors to move forward *(2 and 7)* with the ones for the motors to move in the opposite direction *(3 and 8)*. This is a lot easier than actually opening up the robot, but you may want to note this somewhere if you plan on keeping the robot this way.

Program 2:

Responding to Input:

In this section, you will learn how to make the Romibo Robot respond to input from its touch sensors. The example program will have the robot turn in circles but change direction each time it is triggered by a touch sensor.

As can be seen from the previous table of Arduino pins *(table 1.1)*, the pins for the touch sensors are numbered 28, 29, 30, and 31. Since these pins will be inputting data into the Arduino, these pins will need to be declared as INPUT using the pinMode function.

```
pinMode ( pinNumber, INPUT);
```

Make sure to declare all four of these pins in the setup function. To read from a pin you will use the digitalRead() function. The digitalRead function includes one parameter, the pin number that is being read.

digitalRead(28) will return HIGH if the touch sensor/ button is not being pressed and LOW if the button is being pressed. The reason for this is that the button grounds the connection whenever the two pieces of metal come in contact. Simply writing digitalRead(28) as a statement in your code though will do very little. To make the input change something, you will need to instruct your Arduino to make a decision.

One way of allowing your Arduino Board to make a choice is with the "if statement". An "if statement" asks if something is true, and if it is will do something else. A basic setup of the way to write this "if statement" using normal everyday decisions is:

```
if (you have a dog)
        {
        walk it everyday
        }
```

As you can see from the above pseudo-code, the text in the parenthesis is what is being checked to see if it is true. The text in the brackets is what is being done if the checked statement was true. It would not make sense to walk a dog that does not exist.

To check multiple statements you will need to connect the statements with either an || (or) or an && (and). The || or && is text that is understood by the compiler. In pseudo-code

```
if (you have a dog || you have a cat)
        {
        put food out
        }
```

This bit of pseudo-code tells you that if you have a dog or if you have a cat that you should put food out for it. You can also give commands for when something isn't true. For example

```
if (you have a dog || you have a cat)
        {put food out}
else
{
        consider getting a pet
}
```

This "if -else statement" allows the program to choose from two different responses. The "else" is performed whenever the previous if was not performed. From these examples, one "if statement" you will want to form looks something like this:

```
if (digitalRead(28) == LOW || digitalRead(29) == LOW || digitalRead(30) == LOW ||
digitalRead(31) == LOW)
{
}
```

This code will check to see if any of the buttons have been pressed. If so, the code that follows in the brackets will be performed. Notice that there are 2 equal signs to test if something is

equal. This is important. One equal sign will actually make something equal to something else. The something that is referred to is a variable. Since this program will be causing the robot to choose between turning right and turning left, we will need to declare a variable to decide which direction the robot is turning.

To declare a Boolean variable named TurningRight that is either true or false we write

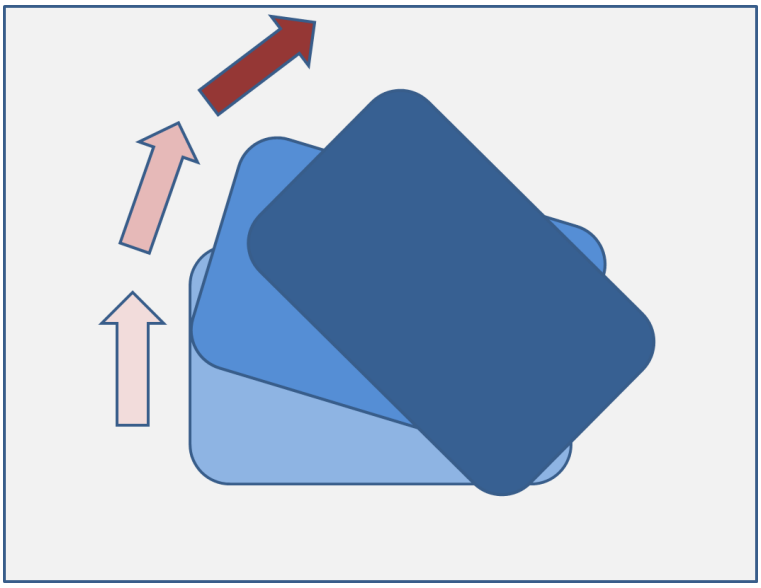
```
boolean TurningRight;
```

Be sure to put this bit of code at the very top of your Sketch. Being before any of the functions will allow this variable to be accessed anywhere in the sketch. We could have named this variable anything we like (provided it starts with a letter and has no spaces or illegal symbols). However, TurningRight is descriptive of what should be true or false.

To set a variable to some value, you use one equal sign. If you want the robot to turn right at the start of the program, put this bit of code into the setup loop.

```
TurningRight = true;
```

This will set the variable to true. Simply doing this though will not cause your robot to turn right. You will have to cause the motors to turn after testing the TurningRight variable. To cause the robot to turn right, you will actually have to tell the left motor to move forward. This is because the right wheel will effectively remain stationary. If you turn to the right yourself, you will notice that you pivot on your right foot and move your left foot around. This is how the robot will have to move as well. This can be visualized with the following image.



Given this information about variables, if statements, and input, you should be able to write a complete working sketch that will make the robot change direction when a button is pressed. Give it a try on your own and upload it to the robot. Remember that the left motor forward is pin 2 and the right motor forward is pin 7. If you need help, try searching the internet. For a complete program see the next page.

Hint: “if-else statements” can go within other “if-else statements”

```
//SecondSketch_Input_

boolean TurningRight;

void setup()
{
  TurningRight = true;
  pinMode (2, OUTPUT);
  pinMode (5, OUTPUT);
  pinMode (6, OUTPUT);
  pinMode (7, OUTPUT);
  digitalWrite (5, HIGH);
  digitalWrite (6, HIGH);
  pinMode (28, INPUT);
  pinMode (29, INPUT);
  pinMode (30, INPUT);
  pinMode (31, INPUT);
}

void loop()
{
if (digitalRead(28) == LOW || digitalRead(29) == LOW || digitalRead(30) == LOW || digitalRead(31) ==
LOW)
  {
    if (TurningRight == true)
    {
      TurningRight = false;
      digitalWrite (2, LOW);
      digitalWrite (7, HIGH);
      delay (250);                          //This delay will keep the button from being checked to quickly
    }
     else
    {
      TurningRight = true;
      digitalWrite (2, HIGH);
      digitalWrite (7, LOW);
      delay (250);                          //This delay will keep the button from being checked to quickly
     }
  }
}
```

Program 3:
Analog Input, Variable Types, and the Serial Monitor

Up until now, you've been using entirely digital input and output. These digital inputs are either completely on or off. However, there is another type of input/output called analog. Use of analog allows there to be a range of values between on and off.

A sensor that is ideal for showing analog input is the photo resistor. If you are unfamiliar with electricity, resistors exist in electrical circuits in order to impede the flow of electrons. Resistors can be thought of as small tunnels that electrons have to pass through. The electrons have to move slower in the same way that cars move slower when entering a tunnel. A photo resistor is a resistor whose resistance (essentially the size of the tunnel) changes whenever a light is shining onto the resistor. A photo resistor's resistance is less whenever more light is being shined on it. This is because the light essentially loosens the electrons from the material, allowing them to flow easier. (The actual explanation is slightly more complicated. This idea will give you a general idea of the concept however).

The two photo resistors on the front of the Romibo are connected to Analog Pins 2 and 3 for left and right respectively. You may be wondering, "isn't pin 2 taken already". The answer is no, digital pin 2 is taken for a motor but analog pin 2 is used for the left photo resistor.

To receive data from the photo resistor, you will need to store it somewhere. Just as you declared a Boolean variable in the last programming exercise, you will also need to declare a variable to store this data. However, this data will not be true/false. The data you will be receiving will be a number. There are multiple ways of storing numbers using Arduino.

Abbreviation	Name	Meaning
int	Integer	Stores integer variables from -32,768 to 32,767
long	Long	Stores larger integer values from -2,147,483,648 to 2,147,483,647
double	Double	Stores values with a decimal point.

The number we will be storing for each photo resistor reading will be an integer. To declare these values you can type:

```
int leftPhotoReading;

int rightPhotoReading;
```

Integer values are helpful in storing readings from analog input. They can also be used to save time in remembering values you'd rather not memorize. For example, you've been writing to a digital pin 7 to trigger one of the motors for a while now. Instead of writing digitalWrite(7, HIGH), you could define a variable such as FWDRT to 7 and instead type digitalWrite(FWDRT, HIGH). To declare a constant, just type const before the name of the variable and put an equal sign after it followed by the value you want it declared to. For example:

```
const int FWDRT = 7;
```

While it is not required, constants are typically typed in all capital letters. To change a non-constant variable, you just type the name of the variable, an equal sign, and then the value you want the variable to change to. The great part about this, is that you can make a variable equal to the value returned from a function or a mathematical statement. For example,

```
int PHOTOLEFT = 2*1-0;          //  * means multiplication
int PHOTORIGHT = 1 + 4/2;
```

The compiler will follow order of operations; just like in math class. Therefore the above code will set PHOTOLEFT to 2 and PHOTORIGHT to a value of 3. Instead of assigning values to our reading variables, it will be more useful to allow them to actually read in data from an analog input. To read analog input, use the function analogRead(pin_number). For our code, we will use

```
leftPhotoReading = analogRead(PHOTOLEFT);
rightPhotoReading = analogRead(PHOTORIGHT);
```

The analogRead( ) function will return a value from 0 to 1023. For photo resistors, the brighter the light shining on the photo resistor, the lower the number returned.

To see what numbers are being returned by your robot you can use a tool called the Serial Monitor. To have your robot return data back to your computer, you will have to include a few lines of code. In your setup loop please include

```
Serial.begin(9600);
```

For your loop() please include the following. Be sure that you declare all of the necessary variable and set the correct pinMode for all pins. For a working program see Program3 SerialPrint Example.

```
leftPhotoReading = analogRead(PHOTOLEFT);
rightPhotoReading = analogRead(PHOTORIGHT);
Serial.print("\nleftPhotoReading");
Serial.print(leftPhotoReading);
Serial.print("\nrightPhotoReading");
Serial.print(rightPhotoReading);
delay(300);
```

After successfully compiling your program, upload it to your robot. Do not unplug the USB port. Instead, click on Tools at the top of your Arduino IDE. Then click on Serial Monitor. Be sure that AutoScroll is checked, Both NL & CR is chosen from the first drop down list. And 9600 baud is selected. Cover and shine lights on the photo resistors to see how the data responds. Look back at the code you entered, to see how the Serial.print and Serial.begin functions work.

Notice that parenthesis output the exact text and that variable names output the variable value. The \n or newline character within parenthesis will create a new line in a similar way to typing enter on your keyboard.

```
//Program3 SerialPrint Example

int leftPhotoReading;

int rightPhotoReading;

int PHOTOLEFT = 2;

int PHOTORIGHT = 3;


void setup()
{
  TurningRight = true;
  pinMode (2, OUTPUT);
  pinMode (5, OUTPUT);
  pinMode (6, OUTPUT);
  pinMode (7, OUTPUT);
  digitalWrite (5, HIGH);
  digitalWrite (6, HIGH);
  pinMode (28, INPUT);
  pinMode (29, INPUT);
  pinMode (30, INPUT);
  pinMode (31, INPUT);
  Serial.begin(9600);
}

void loop()
{
leftPhotoReading = analogRead(PHOTOLEFT);
rightPhotoReading = analogRead(PHOTORIGHT);
Serial.print("\nleftPhotoReading");
Serial.print(leftPhotoReading);
Serial.print("\nrightPhotoReading");
Serial.print(rightPhotoReading);
delay(300);
}
```

Now, that you've seen how the analogRead function works with photo resistors, you are ready to start making more complex programs. Instead of the program you made in the last section that has the robot change direction by a button press, try making one that has the robot follow a light. Review the section of if-else statements if need be. The following are available to produce true or false values. You have seen == before, the other should be easy to infer.

>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
==	equal to

Between, these comparisons and the ability to use +,-,*,and / to add, subtract, multiply, and divide, you should be able to make your robot do more complex motions. Example code of a robot following a light can be found on the following page. Try to create your own or edit the one provided to fine tune your program.

```
//Program3 ChaseLight Example

int leftPhotoReading;
int rightPhotoReading;
const int PHOTOLEFT = 2;
const int PHOTORIGHT = 3;
const int FWDRT = 7;
const int FWDLEFT = 2;

void setup()
{
  pinMode (2, OUTPUT);
  pinMode (5, OUTPUT);
  pinMode (6, OUTPUT);
  pinMode (7, OUTPUT);
  digitalWrite (5, HIGH);
  digitalWrite (6, HIGH);
}

void loop()
{


  leftPhotoReading = analogRead(PHOTOLEFT);
  rightPhotoReading = analogRead(PHOTORIGHT);

  if(leftPhotoReading < rightPhotoReading - 30)
  {
    digitalWrite(FWDRT, HIGH);
    digitalWrite(FWDLEFT, LOW);
  }
  else if(rightPhotoReading < leftPhotoReading - 30)
  {
    digitalWrite(FWDRT, LOW);
    digitalWrite(FWDLEFT, HIGH);
  }
  else
  {
    digitalWrite(FWDRT, HIGH);
    digitalWrite(FWDLEFT, HIGH);
  }
}
```

Program 4:
Analog Output, Lighting, and Changing Speeds

Example Programs:

```
//Antenna Color Shift

int timer = 0;
const int RED = 4;
const int BLUE = 10;
const int GRN = 9;

void setup()
{
  pinMode (RED, OUTPUT);
  pinMode (GRN, OUTPUT);
  pinMode (BLUE, OUTPUT);
}

void loop()
{
  timer = 0;
  while (timer < 254)
  {
    analogWrite(RED, timer);
    timer++;
    delay (4);
  }
  for (int i = 0; i < 254; i++)
  {
    analogWrite(RED, timer - i);
    analogWrite(BLUE, i);
    delay (4);
  }
  timer = 0;
  do
  {
   analogWrite(BLUE, 254 - timer);
   analogWrite(GRN, timer);
   timer++;
   delay(4);
  }while (timer < 253);
  while (timer > 0)
  {
    analogWrite(GRN, timer);
    timer--;
    delay(3);
  }
}
```

```
//Read IR Values

const int IRIN = 48;
const int RED = 4;
const int BLUE = 10;
const int GRN = 9;

int highPulse;
int lowPulse;
const int RESOLUTION = 20;
int pulses[60][2];
long useconds;
int currentPulse;

void setup(void)
{
  pinMode (RED, OUTPUT);
  pinMode (GRN, OUTPUT);
  pinMode (BLUE, OUTPUT);
  pinMode (IRIN, INPUT);
  Serial.begin(9600);
  Serial.println("Printing IR delays");

}
void loop(void)
{
  currentPulse = 0;
  while (digitalRead(IRIN))
  {

    while (digitalRead(IRIN))
    {
      useconds = useconds + RESOLUTION;
      delayMicroseconds(RESOLUTION);
    }
    pulses[currentPulse][0] = useconds;
    useconds = 0;
    while (!digitalRead(IRIN))
    {
      useconds = useconds + RESOLUTION;
      delayMicroseconds(RESOLUTION);
      if (useconds > 4800)
        break;
    }
```

```
  pulses[currentPulse][1] = useconds;
  currentPulse++;
}

  Serial.println("New Data Set");
  for (int i = 0; i < currentPulse; i++)
  {
    Serial.print(pulses[i][0]);
    Serial.print("    ");
    Serial.print(pulses[i][1]);
    Serial.println(" ");
  }
```

```
//Respond to IR Values
//Works with Sanyo Remote to trigger lights

const int IRIN = 48;
const int RED = 4;
const int BLUE = 10;
const int GRN = 9;

int highPulse;
int lowPulse;
const int RESOLUTION = 20;
int pulses[60][2];
long useconds;
int currentPulse;
boolean lighton = false;

void setup(void)
{
  pinMode (RED, OUTPUT);
  pinMode (GRN, OUTPUT);
  pinMode (BLUE, OUTPUT);
  pinMode (IRIN, INPUT);
  Serial.begin(9600);
  Serial.println("Printing IR delays");

}
void loop(void)
{
  currentPulse = 0;
  while (digitalRead(IRIN))
  {

    while (digitalRead(IRIN))
    {
      useconds = useconds + RESOLUTION;
      delayMicroseconds(RESOLUTION);
    }
    pulses[currentPulse][0] = useconds;
    useconds = 0;
    while (!digitalRead(IRIN))
    {
      useconds = useconds + RESOLUTION;
      delayMicroseconds(RESOLUTION);
      if (useconds > 4800)
        break;
    }
    pulses[currentPulse][1] = useconds;
    currentPulse++;
  }

  Serial.println("New Data Set");
  for (int i = 0; i < currentPulse; i++)
  {
    Serial.print(pulses[i][0]);
    Serial.print("    ");
    Serial.print(pulses[i][1]);
```

```
    Serial.println(" ");
  }

  for (int i = 0; i < 90; i++)
  {
    if (pulses[i][0] == 840 && pulses[i+1][0] == 840 && pulses[i+2][0] == 840 && pulses[i+3][0] == 840 &&
pulses[i+4][0] == 1740 && pulses[i+5][0] == 840);
    {
      lighton = !lighton;
      if (lighton)
      {
        digitalWrite(GRN, HIGH);
        digitalWrite(RED, LOW);
        digitalWrite(BLUE, LOW);
      }
      else
      {
        digitalWrite(GRN, LOW);
        digitalWrite(RED, LOW);
        digitalWrite(BLUE, LOW);
      }
      break;
    }
    if (pulses[i][0] == 840 && pulses[i+1][0] == 840 && pulses[i+2][0] == 840 && pulses[i+3][0] == 860 &&
pulses[i+4][0] == 840 && pulses[i+5][0] == 1720);
    {
      lighton = !lighton;
      if (lighton)
      {
        digitalWrite(GRN, LOW);
        digitalWrite(RED, HIGH);
        digitalWrite(BLUE, LOW);
      }
      else
      {
        digitalWrite(GRN, LOW);
        digitalWrite(RED, LOW);
        digitalWrite(BLUE, LOW);
      }
      break;
    }
    if (pulses[i][0] == 1720 && pulses[i+1][0] == 1740 && pulses[i+2][0] == 1720 && pulses[i+3][0] == 1740);
    {
      lighton = !lighton;
      if (lighton)
      {
        digitalWrite(GRN, LOW);
        digitalWrite(RED, LOW);
        digitalWrite(BLUE, HIGH);
      }
      else
      {
        digitalWrite(GRN, LOW);
        digitalWrite(RED, LOW);
        digitalWrite(BLUE, LOW);
      }
```

```
      break;
    }
  }
}
```

For some, programming can easily be learned from simply reading over someone else's work. The following program was written with the QoLT young scholars program. Try uploading the code to your robot to see how it behaves. Refer back to the code to see why it does each behavior. Be sure to touch each of the touch sensors.

```
// From_Scratch_Demo
// Anna + Jason + Adam

//These includes include other files for programming.
//These files tell the arduino how to handle functions that involve a timer or servos.
#include <Timer.h>
#include <Servo.h>


//The following are variables.

//int stands for integer.  The name that follows is the name of the variable.
//These variables can hold integer values.
int inversedistance;
int beat1time;
int beat2time;
int beat3time;
int beat4time;
int beataveragetime;
//The const before the int means that the variable is a constant
//These values will not change while the program is running.
const int FWDLEFT = 2;  //These Numbers are the numbers of the pins on the arduino board that connect to the motors.
const int REVLEFT = 3;  //For Example, sending a signal to REVLEFT (pin 3) will cause the Left motor to move in reverse.
const int FWDRT = 7;    //It is not necessary to declare these constants if you remember the pins.  But since that would be difficult the variables
can stand in place for the pin numbers later.
const int REVRT = 8;
const int RANGEPIN = 1;
//A boolean variable is either true or false;  The value can be declared initially with =
//The robot will not start dancing until code instructs it to do so.
boolean IsDancing = false;
int BeatsDanced = 0;
boolean lastButton = true;
boolean motorOn = false;
//A long is also an integer but it is for larger numbers.
//Since time is measured in milliseconds, these values can become quite large.  Therefore long is more appropriate than int.
long startTime;
//Servo is for use with Servo Motors.  These will move to a particular position.
Servo ForwardServo;
Servo BackwardServo;
//This is a function declaration.  void means that the variable has no parameters
//It is neither true, false, a number, or a letter.  It is nothing really.
void DanceFunction (void);

//This setup function is required for use with arduino.
//Anything that follows in the squiggly brackets {} will be performed when the robot is first turned on.
void setup(void)
```

```
{
 Serial.begin(9600);
 ForwardServo.attach(44);
 BackwardServo.attach(45);

 //lets motors work
 pinMode (5, OUTPUT);
 digitalWrite (5, HIGH);
 pinMode (6, OUTPUT);
 digitalWrite (6, HIGH);
 delay (1000);
}

//This loop function is also required for use with arduino
//After setup is performed, loop will be performed.
//The loop function will repeat over and over again until the robot is turned off.

void loop()
{
 if (digitalRead (28) == LOW)
 {
ForwardServo.writeMicroseconds(2000); //tilts head forward
BackwardServo.writeMicroseconds(2000);
 delay(1000);
ForwardServo.writeMicroseconds(1000); //tilts head backward
BackwardServo.writeMicroseconds(1000);
 delay(1000);
ForwardServo.writeMicroseconds(1500); //tilts head to center
BackwardServo.writeMicroseconds(1500);
 delay(500);
 }
 if (digitalRead (29) == LOW)
 {
  digitalWrite (FWDLEFT, HIGH);
  delay (1000);
  digitalWrite (FWDLEFT, LOW);
  digitalWrite (REVLEFT, HIGH);
  delay (1000);
  digitalWrite (REVLEFT, LOW);
 }
 if (digitalRead (31) == LOW)
 {
  digitalWrite (FWDRT, HIGH);
  delay(1000);
  digitalWrite (FWDRT, LOW);
  digitalWrite (REVRT, HIGH);
  delay(1000);
  digitalWrite (REVRT, LOW);
 }
 //DanceFunction();

 if (digitalRead(30) == LOW && lastButton == true)
  {
   startTime = millis();
   digitalWrite (FWDLEFT, HIGH);
   digitalWrite (FWDRT, HIGH);
   ForwardServo.writeMicroseconds(2000); //tilts head forward
   BackwardServo.writeMicroseconds(2000);
   lastButton = false;
   delay(150);
   motorOn = true;
```

```
  }
 if ((digitalRead(30) == LOW && lastButton == false) || (millis() - startTime >= 30000))

  {
   digitalWrite (FWDLEFT, LOW);
   digitalWrite(FWDRT, LOW);
  lastButton = true;
  delay(150);
  motorOn = false;
  }

 if (motorOn == true && analogRead(RANGEPIN) >= 260)
  {
    digitalWrite(FWDLEFT, LOW);
    digitalWrite(FWDRT, LOW);
    if (millis() % 2)
    { digitalWrite(REVLEFT, HIGH);
     digitalWrite(REVRT, HIGH);
     delay(200);
     digitalWrite(REVLEFT, LOW);
     delay(1000);
     digitalWrite(REVRT, LOW);
    }
    else
    {
     digitalWrite(REVLEFT, HIGH);
     digitalWrite(REVRT, HIGH);
     delay(200);
     digitalWrite(REVRT, LOW);
     delay(1000);
     digitalWrite(REVLEFT, LOW);
    }
  }

 if (motorOn == true && analogRead(RANGEPIN) <=230)
  {
   digitalWrite(FWDLEFT, HIGH);
   digitalWrite(FWDRT, HIGH);
  }
 Serial.println(analogRead(RANGEPIN));
 inversedistance = analogRead(RANGEPIN);
if (inversedistance >= 230)
{
  ForwardServo.writeMicroseconds(1000); //tilts head backward
  BackwardServo.writeMicroseconds(1000);
}
if (inversedistance <=210)
{
   ForwardServo.writeMicroseconds(1500); //tilts head to center
   BackwardServo.writeMicroseconds(1500);
}
}

void DanceFunction (void)
{
   if (digitalRead (30) == LOW)
  {
  beat1time = millis();
  delay (180);
   for (int i=0; i < 1000; i++)
   {
```

```
    if (digitalRead(30) == LOW)
    {
      beat2time = millis();
      i=1000;
    }
    delay (2);
  }

   delay (180);
   for (int i=0; i < 1000; i++)
  {
    if (digitalRead(30) == LOW)
    {
      beat3time = millis();
      i=1000;
    }
    delay (2);
  }
  delay (180);
     for (int i=0; i < 1000; i++)
  {
    if (digitalRead(30) == LOW)
    {
      beat4time = millis();
      i=1000;
      delay(180);

    }
    delay (2);
  }

  beataveragetime = (((beat2time - beat1time) + (beat3time - beat2time) + (beat4time - beat3time))/3);
  IsDancing = true;
 }
 if (IsDancing)
 {
  ForwardServo.writeMicroseconds(1800); //tilts head forward
 BackwardServo.writeMicroseconds(1800);
 digitalWrite (FWDLEFT, HIGH);
 digitalWrite (REVRT, HIGH);
 delay(beataveragetime/6);
 digitalWrite (FWDLEFT, LOW);
 digitalWrite (REVRT, LOW);
 delay(beataveragetime/6);
 delay(beataveragetime/3);
 ForwardServo.writeMicroseconds(1200); //tilts head backward
 BackwardServo.writeMicroseconds(1200);
 digitalWrite (FWDRT, HIGH);
 digitalWrite (REVLEFT, HIGH);
 delay(beataveragetime/6);
 digitalWrite (FWDRT, LOW);
 digitalWrite (REVLEFT, LOW);
 delay(beataveragetime/6);
 BeatsDanced++;
 if (BeatsDanced >= 16)
 {
  BeatsDanced = 0;
  IsDancing = false;
 }
 }
}
```

I hope that you've enjoyed this introduction to programming with Romibo and Arduino. There is always more to learn. For other code, check out Romibo's github page. For information on programming with Arduino, there are numerous resources on Arduino.cc/en