Workshop 2:
Robotic Sensory Systems

Nicholas Ho

Objectives:

1. Apply and evaluate robotic sensory systems and their operations

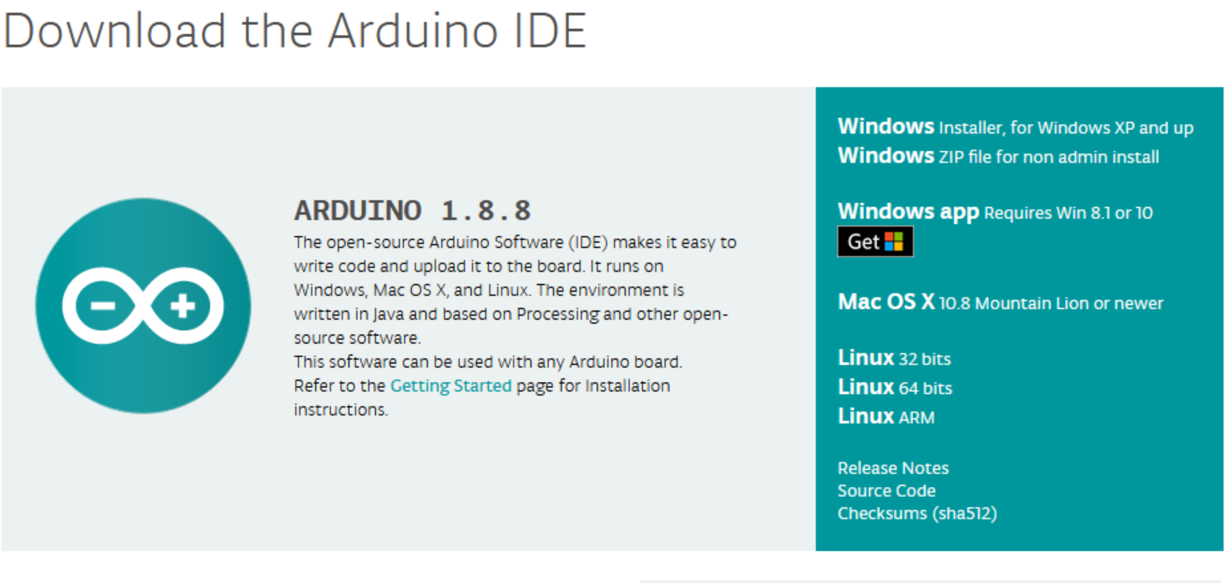
Contents

1. Introduction to Arduino
2. Configuring sensors in Arduino
3. Programming robotic arm to respond differently to various sensors' input
4. Group Work

Chapter 1:
Introduction to Arduino

Installing Arduino Software (IDE)

- Download and install Arduino: https://www.arduino.cc/en/Main/Software
- Done for most of the computers



Download the Arduino IDE

ARDUINO 1.8.8

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.
This software can be used with any Arduino board.
Refer to the Getting Started page for Installation instructions.

Windows Installer, for Windows XP and up
Windows ZIP file for non admin install

Windows app Requires Win 8.1 or 10
Get

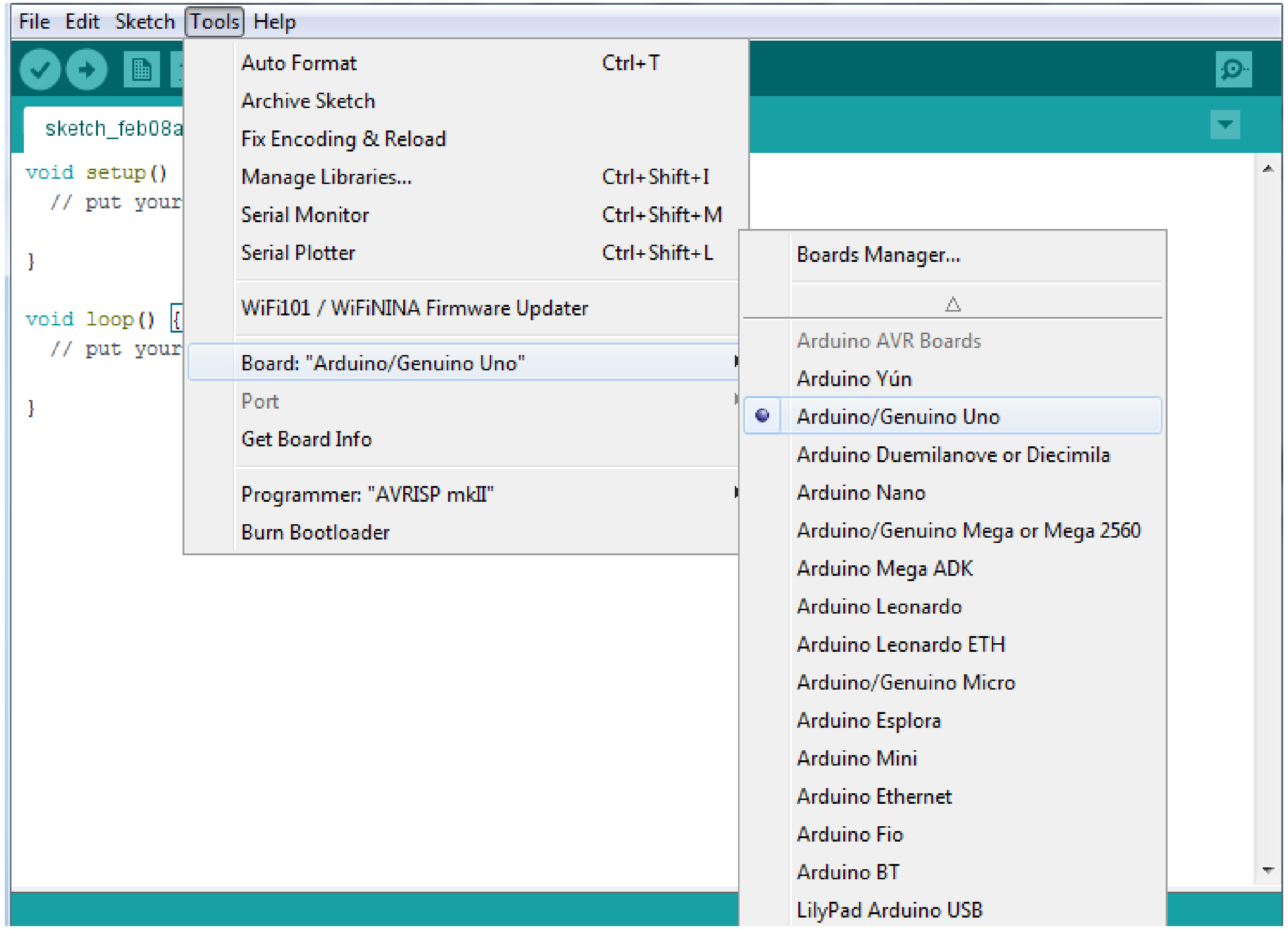
Mac OS X 10.8 Mountain Lion or newer

Linux 32 bits
Linux 64 bits
Linux ARM

Release Notes
Source Code
Checksums (sha512)

Setting up Arduino Software (IDE)

- Open Arduino Software, go to ***Tools* then *Board*, select *Arduino Uno***
- Next, **go to *Port*, and select the correct Port** [i.e. ACM0 (Uno)]
- For this workshop, we will use Arduino Uno for our board

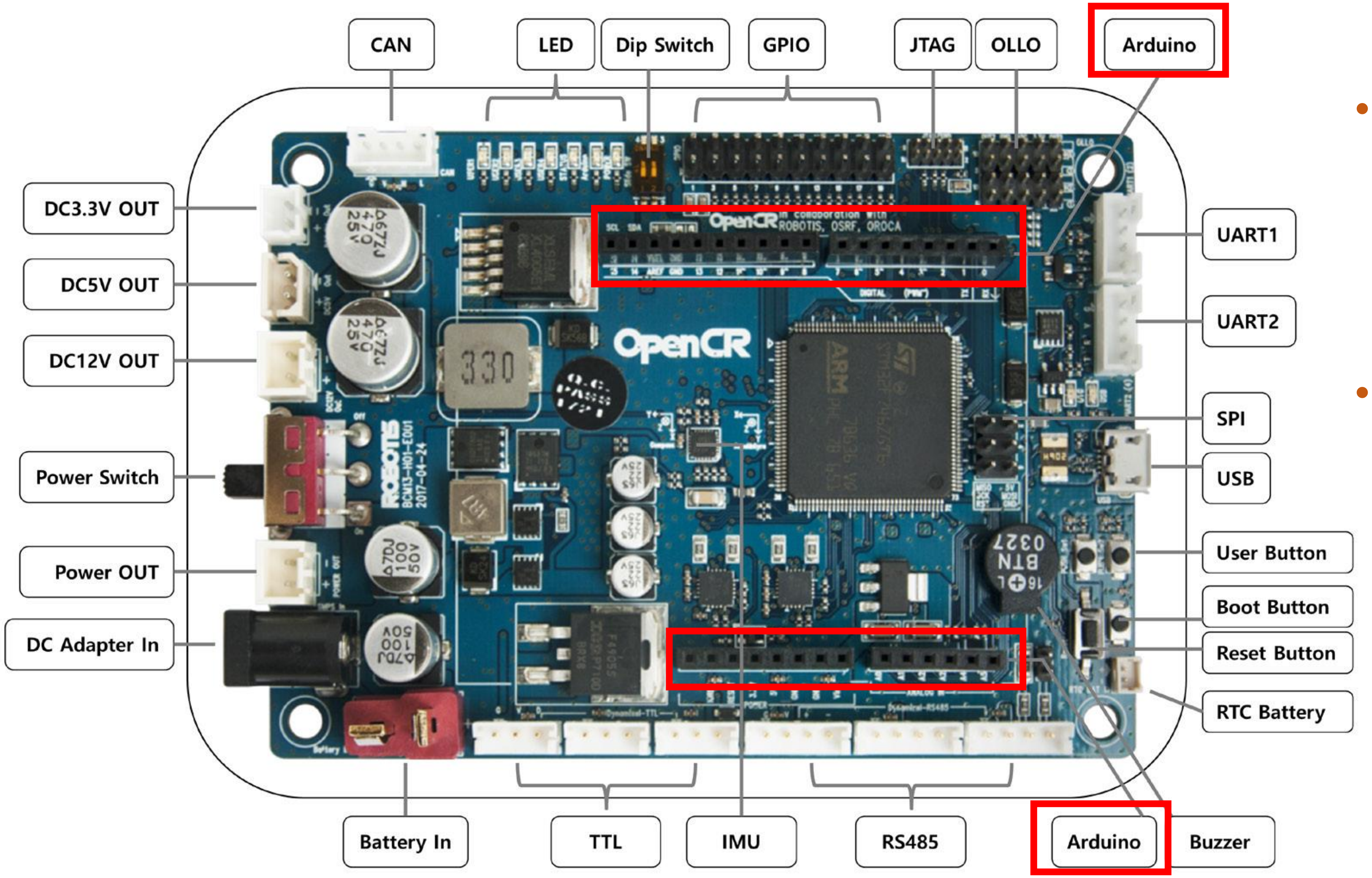


```
void setup()
  // put your

}

void loop() {
  // put your

}
```

Board: "Arduino/Genuino Uno"

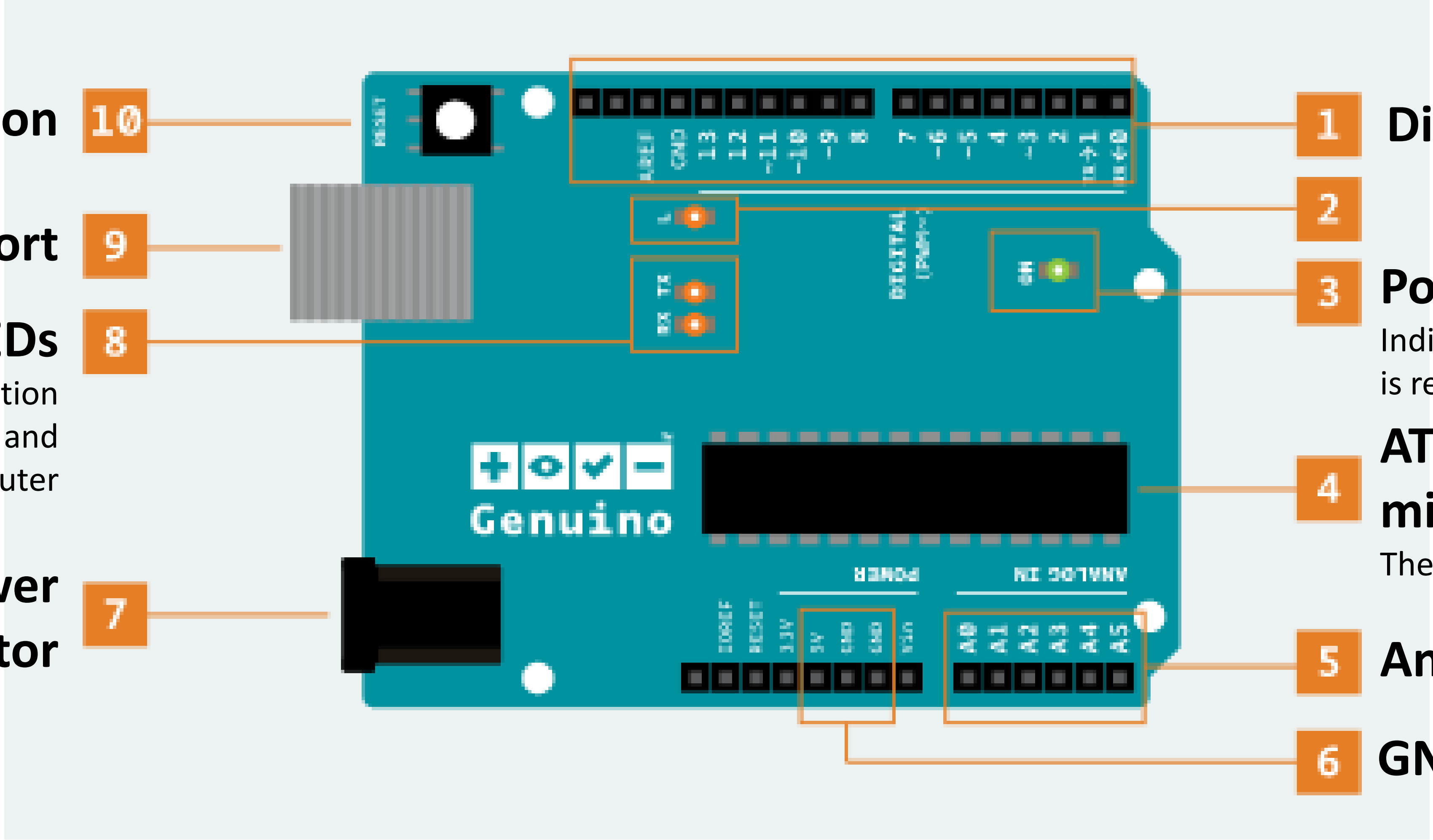
● Arduino/Genuino Uno

OpenCR is also equipped with Arduino



- Note that to equip the robot with sensors, we have to use the Arduino or the GPIO or the OLLO on the OpenCR board
- However, due to current physical constraints (i.e. assembled turtlebot restrict the connection of sensors), we will use a separate Arduino board to simulate this process

Understanding the Arduino Uno Board



10 **Reset Button**

9 **USB port**

8 **TX & RX LEDs**

indicate communication between your board and your computer

7 **Power connector**

1 **Digital pins**

2

3 **Power LED**

Indicates that your board is receiving power

4 **ATmega microcontroller**

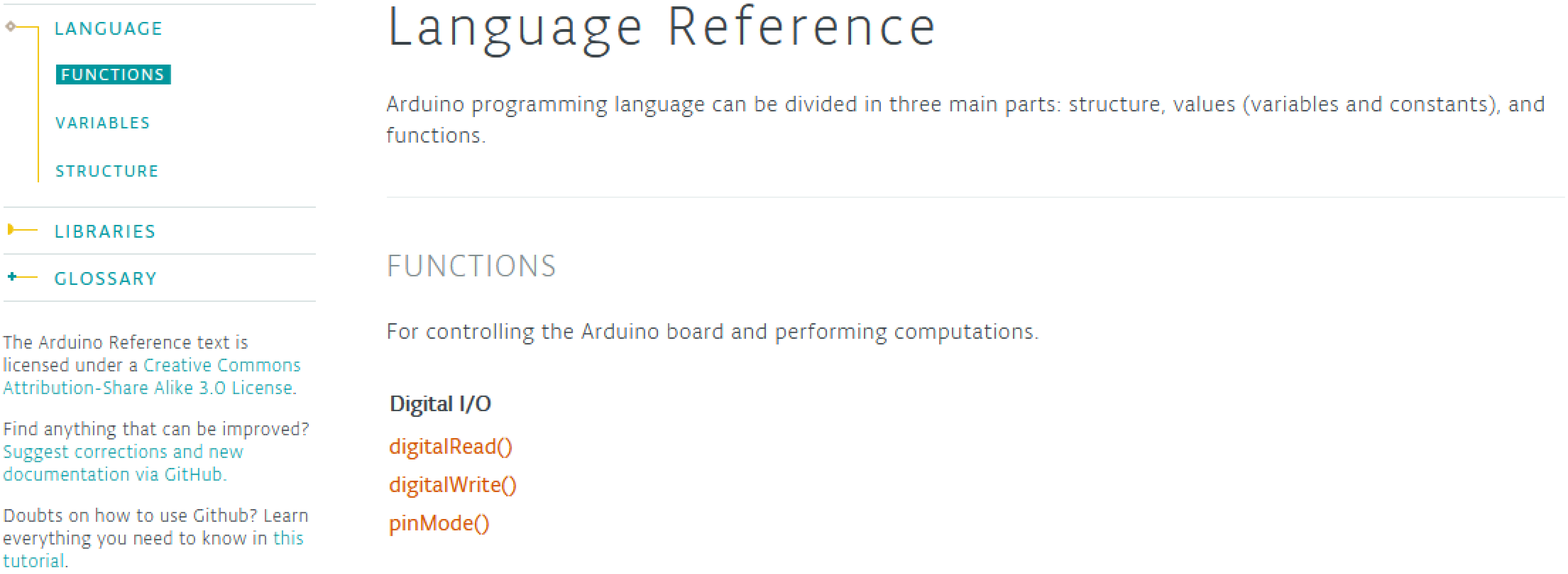
The heart of your board

5 **Analog in**

6 **GND & 5V pins**

Basic Understanding of Codes in Arduino

- https://www.arduino.cc/reference/en/



The Arduino Reference text is licensed under a Creative Commons Attribution-Share Alike 3.0 License.

Find anything that can be improved? Suggest corrections and new documentation via GitHub.

Doubts on how to use Github? Learn everything you need to know in this tutorial.

Language Reference

Arduino programming language can be divided in three main parts: structure, values (variables and constants), and functions.

FUNCTIONS

For controlling the Arduino board and performing computations.

Digital I/O

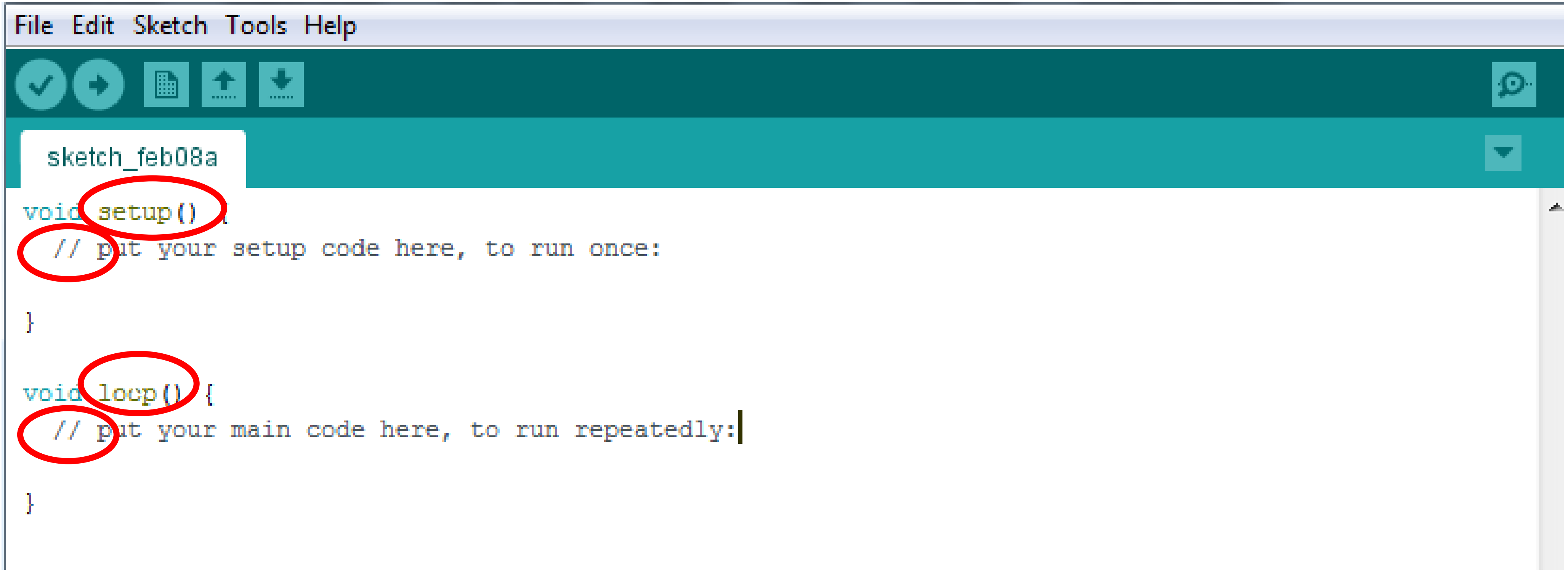
digitalRead()

digitalWrite()

pinMode()

Basic Understanding of Codes in Arduino

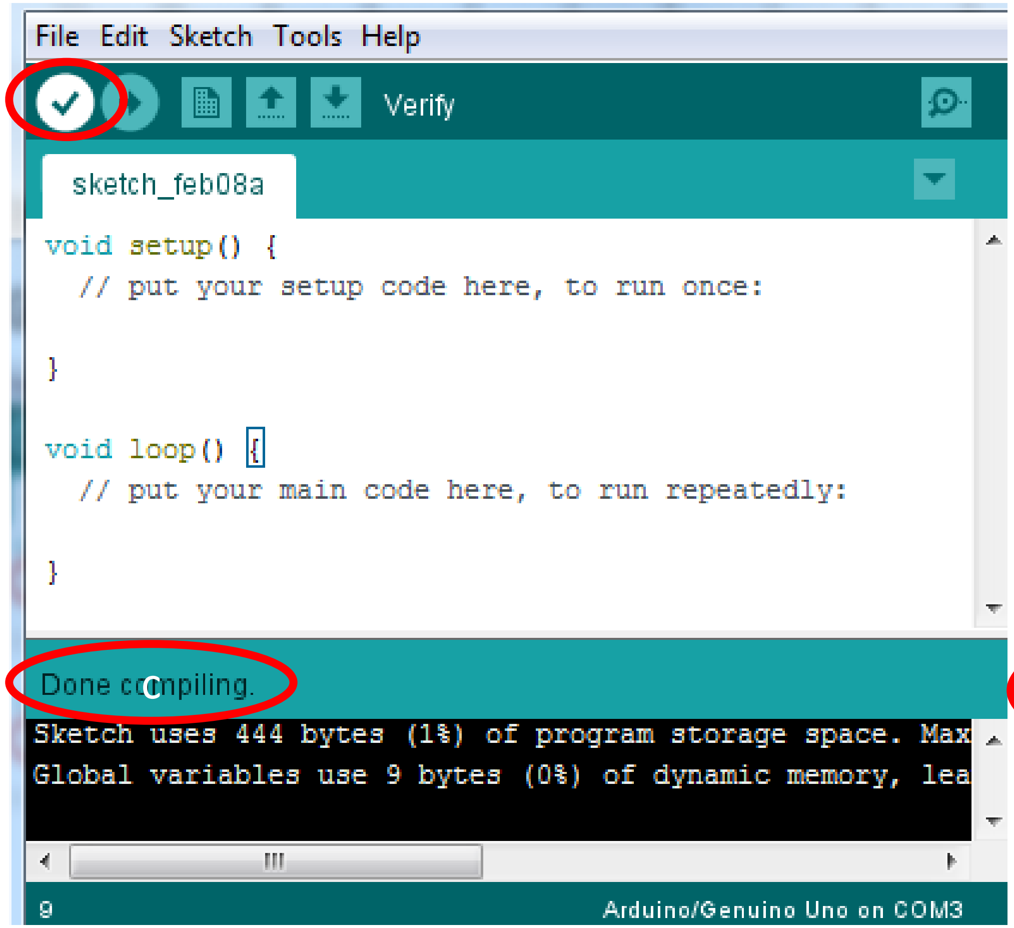
- Go to *file* and click *new*; a sketch will open



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Basic Understanding of Codes in Arduino

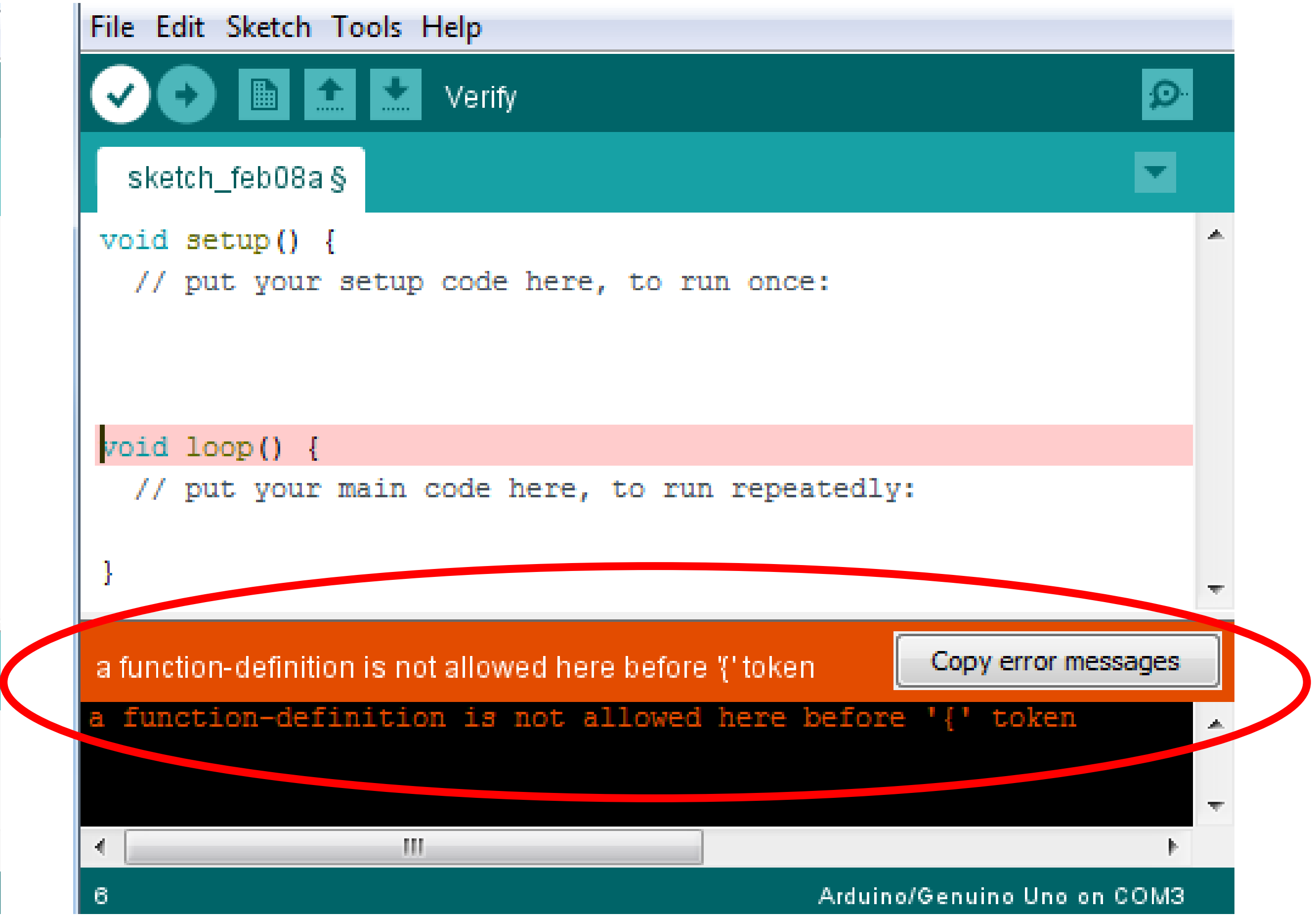
- To check if your codes has no errors, click *verify*



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Done compiling.

```
Sketch uses 444 bytes (1%) of program storage space. Max
Global variables use 9 bytes (0%) of dynamic memory, lea
```



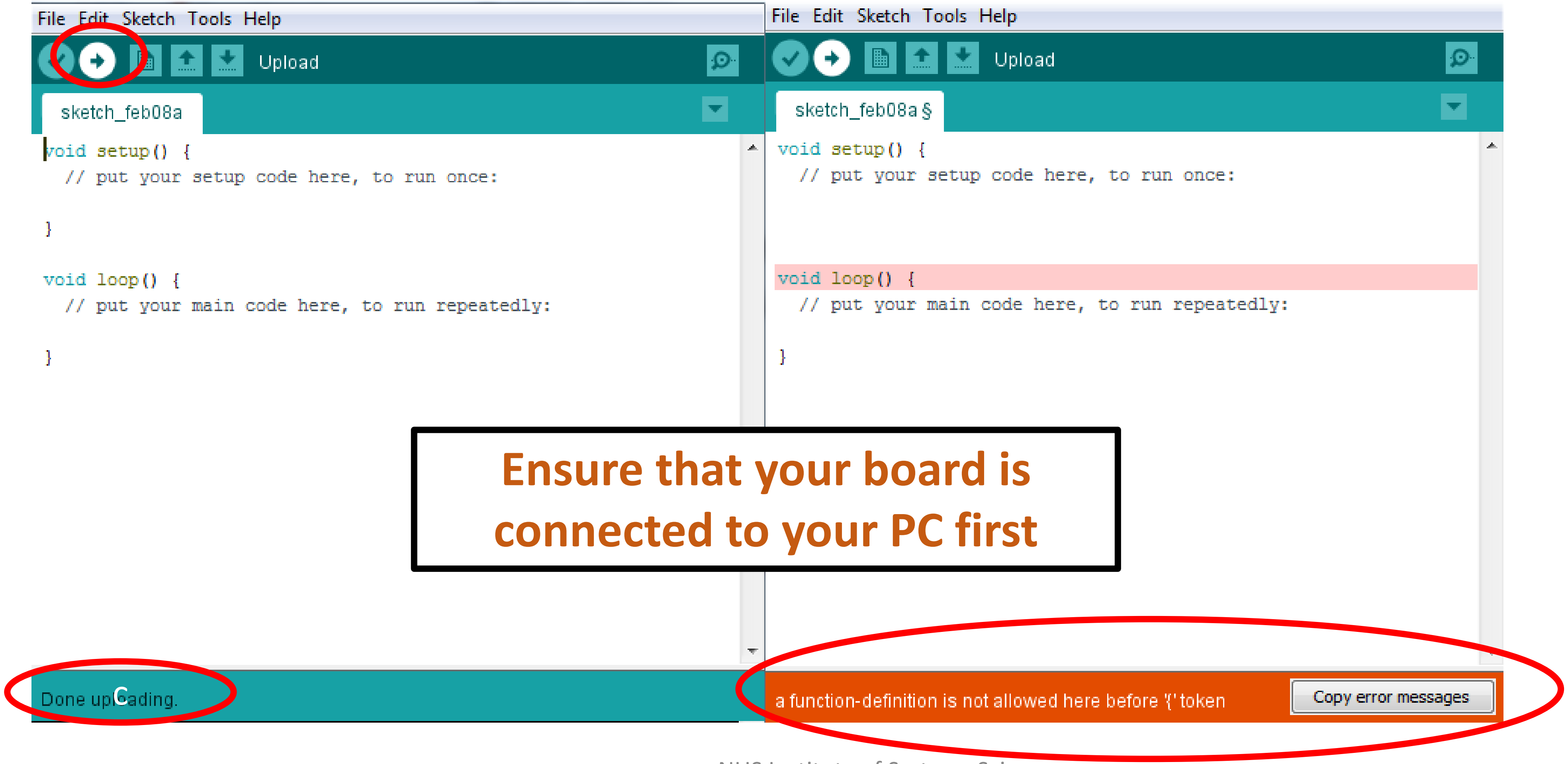
```
void setup() {
  // put your setup code here, to run once:



void loop() {
  // put your main code here, to run repeatedly:

}
```

a function-definition is not allowed here before '{' token

```
a function-definition is not allowed here before '{' token
```

Basic Understanding of Codes in Arduino

- To upload your codes to the board, click *upload*



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Done uploading.

```
void setup() {
  // put your setup code here, to run once:


void loop() {
  // put your main code here, to run repeatedly:

}
```

a function-definition is not allowed here before '{' token

Ensure that your board is connected to your PC first

Basic Understanding of Codes in Arduino

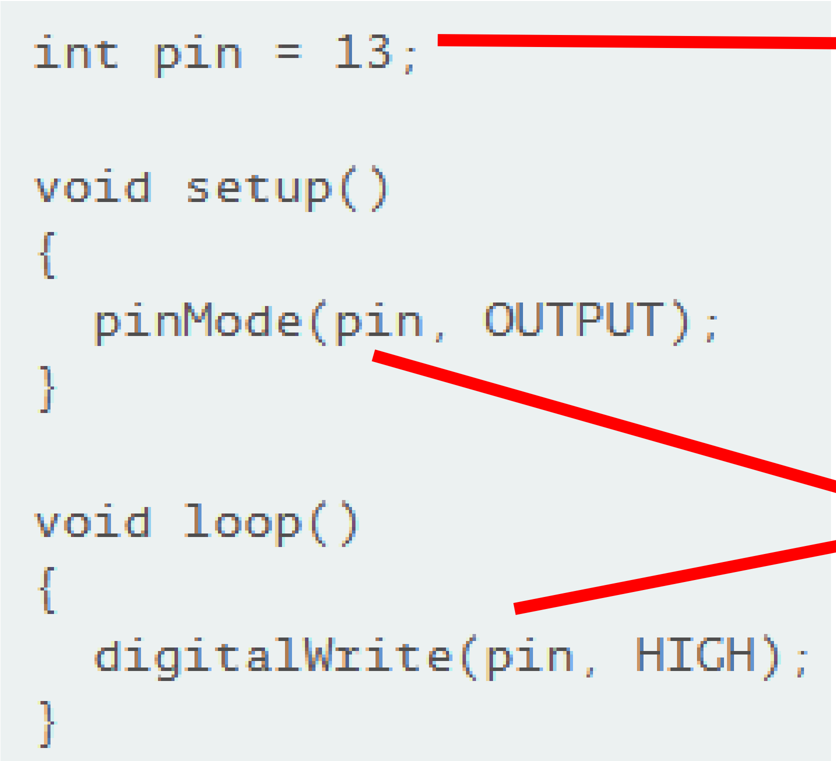
Defining Variables

- `int pin = 13;`
  - Variable name set to `pin`, which value is `13` and type is `int`
- `pinMode(pin, OUTPUT);`
  - Value of `pin` that is passed to the pinMode() function
  - Exactly the same as `pinMode(13, OUTPUT)`
- Advantage of defining the variable pin → **able to quickly specify the actual number of the pin** by just changing the number in the following statement: `int pin = 13;`

Basic Understanding of Codes in Arduino

Global Variables

- Variables that can be used anywhere in your program



```
int pin = 13;

void setup()
{
  pinMode(pin, OUTPUT);
}

void loop()
{
  digitalWrite(pin, HIGH);
}
```

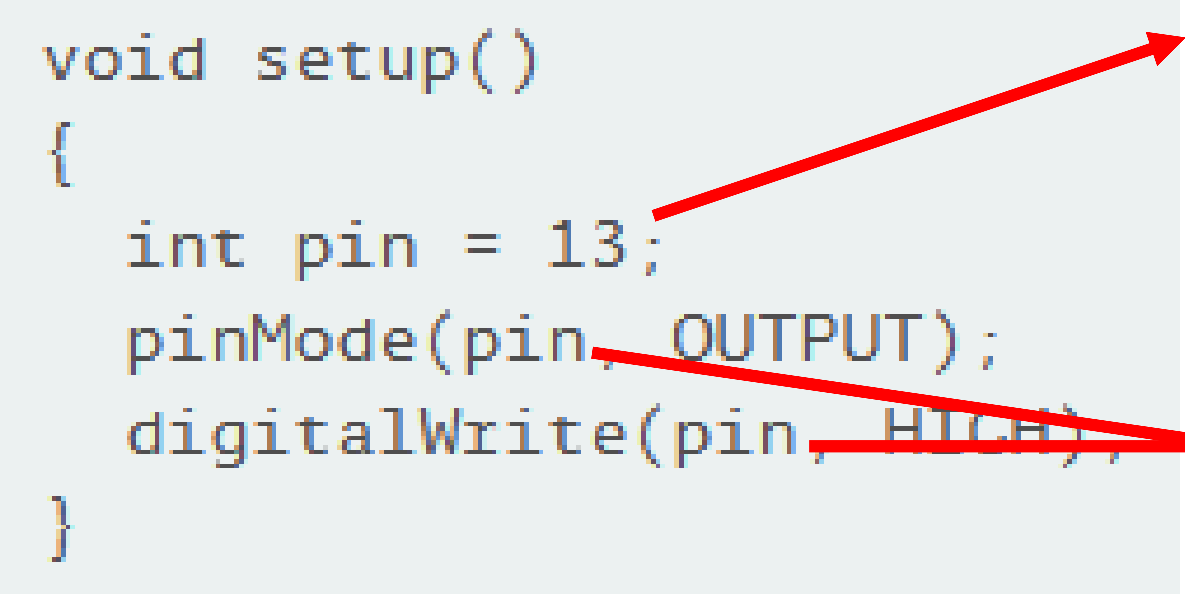
Declared at the top of the code, outside the `setup()` and `loop()` functions, aka Globals Section

`pin` values are all 13 as defined globally

Basic Understanding of Codes in Arduino

Local Variables

- Variables that can be used only in the function



```
void setup()
{
   int pin = 13;
   pinMode(pin, OUTPUT);
   digitalWrite(pin, HIGH);
}
```

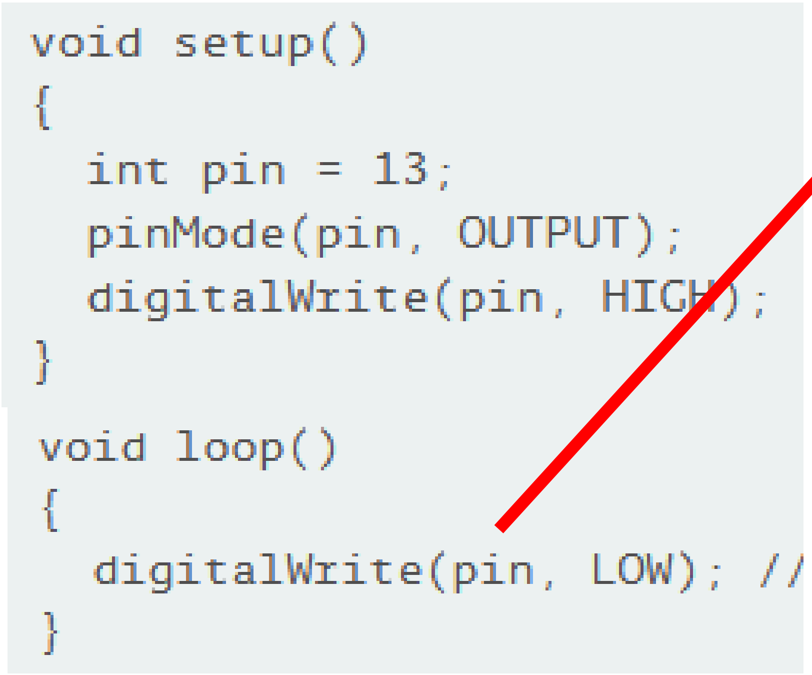
Declared within the function itself; in this example, the `setup()` function

`pin` values are all 13 within the `setup()` function as defined locally

Basic Understanding of Codes in Arduino

Local Variables

- Variables that can be used only in the function



```
void setup()
{
  int pin = 13;
  pinMode(pin, OUTPUT);
  digitalWrite(pin, HIGH);
}
void loop()
{
  digitalWrite(pin, LOW); //
}
```

Undefined variable in the `loop()` function → this will lead to error!

Two ways to solve this:

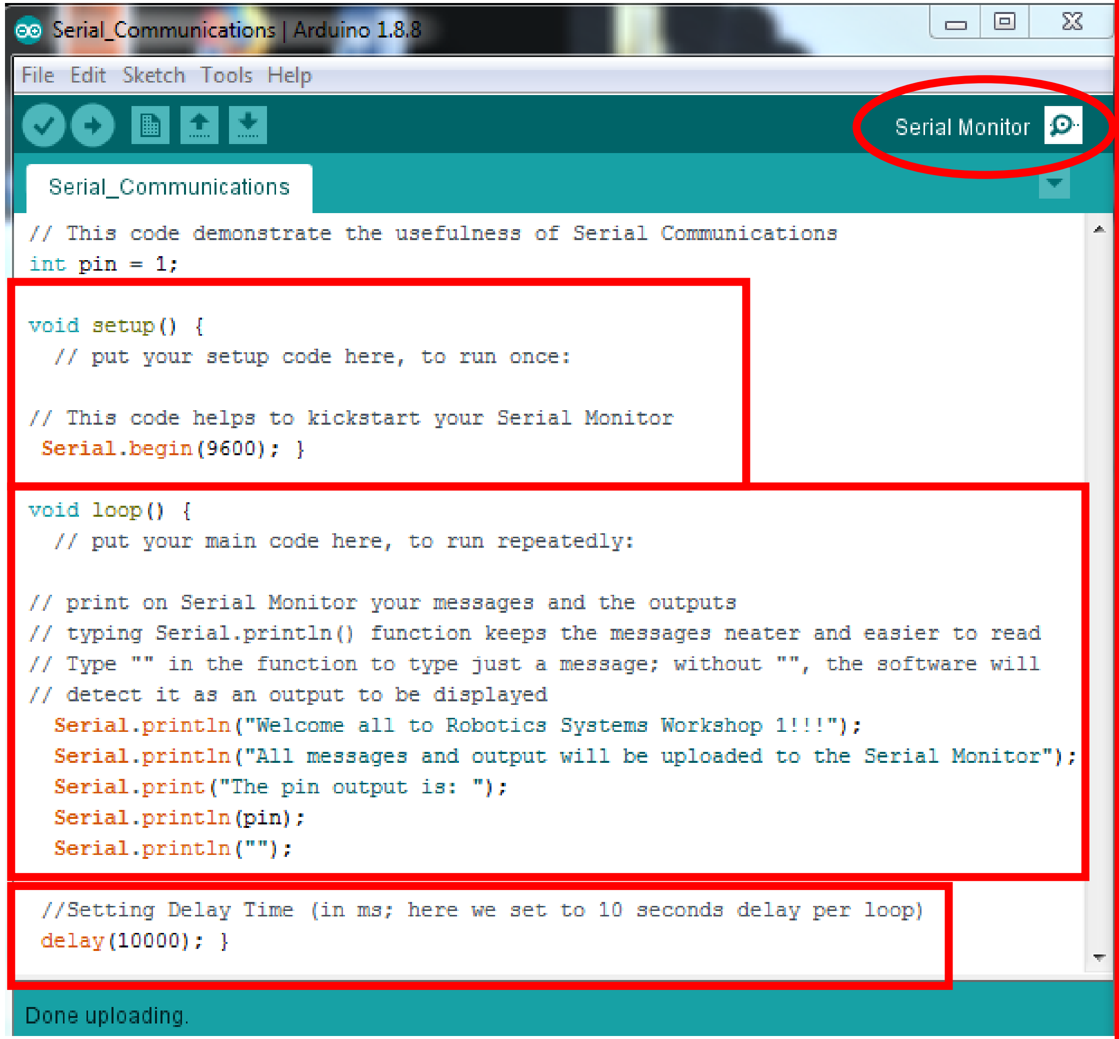
1. Define `pin` as a global variable (i.e. put the statement on top)
2. Define `pinLoop` as a local variable in the loop() function, and change `pin` to `pinLoop`

Basic Understanding of Codes in Arduino

Other important codes to take note

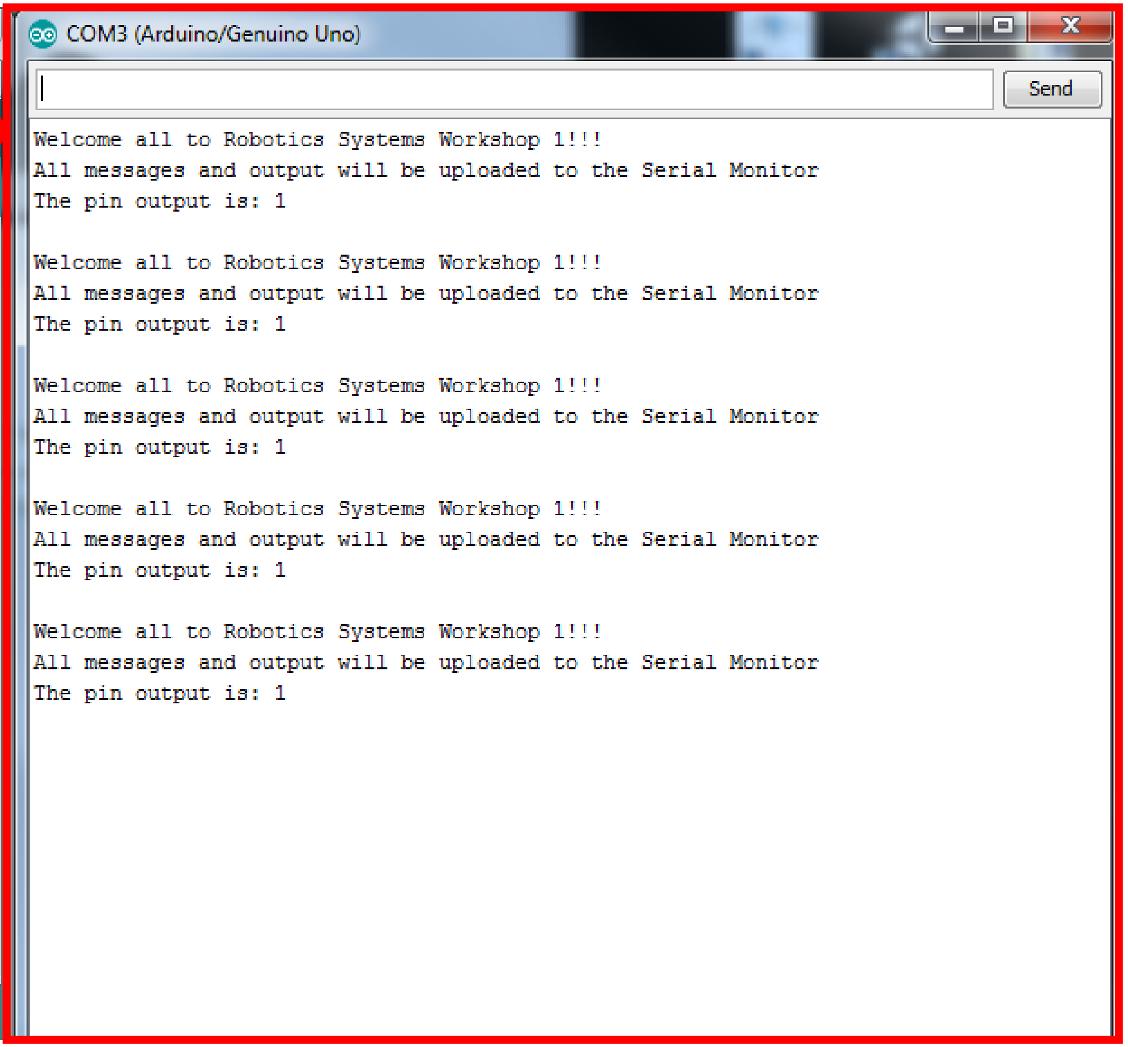
1. `Serial.println()` function
   - print values on the Serial Monitor
   - Important function to monitor input/output values and to help with troubleshooting
2. `analogRead()` function
   - Read the analog values of an input
   - Map input voltages between 0 and 5 into values between 0 and 1023
3. `digitalRead()` function
   - Read the digital values of an input (`HIGH` or `LOW`)
4. `Delay()` function
   - Delay the loop (in ms)

Serial Communication: using `Serial.println()`


Serial_Communications | Arduino 1.8.8
File Edit Sketch Tools Help
Serial Monitor
Serial_Communications

```
// This code demonstrate the usefulness of Serial Communications
int pin = 1;

void setup() {
  // put your setup code here, to run once:

// This code helps to kickstart your Serial Monitor
 Serial.begin(9600); }

void loop() {
  // put your main code here, to run repeatedly:

// print on Serial Monitor your messages and the outputs
// typing Serial.println() function keeps the messages neater and easier to read
// Type "" in the function to type just a message; without "", the software will
// detect it as an output to be displayed
  Serial.println("Welcome all to Robotics Systems Workshop 1!!!");
  Serial.println("All messages and output will be uploaded to the Serial Monitor");
  Serial.print("The pin output is: ");
  Serial.println(pin);
  Serial.println("");

 //Setting Delay Time (in ms; here we set to 10 seconds delay per loop)
 delay(10000); }
```

Done uploading.



```
Welcome all to Robotics Systems Workshop 1!!!
All messages and output will be uploaded to the Serial Monitor
The pin output is: 1

Welcome all to Robotics Systems Workshop 1!!!
All messages and output will be uploaded to the Serial Monitor
The pin output is: 1

Welcome all to Robotics Systems Workshop 1!!!
All messages and output will be uploaded to the Serial Monitor
The pin output is: 1

Welcome all to Robotics Systems Workshop 1!!!
All messages and output will be uploaded to the Serial Monitor
The pin output is: 1

Welcome all to Robotics Systems Workshop 1!!!
All messages and output will be uploaded to the Serial Monitor
The pin output is: 1
```

Example (Serial Communications 2)

Open the given Arduino codes (name: *Serial_Communications_2*; the codes are uploaded on luminus. Examine the codes.

The objectives for this program is to:

1. Obtain pin output as 13 and
2. Configure pin2 output to start from 1 and be increasing by 1 per second

Example (Serial Communications 2)

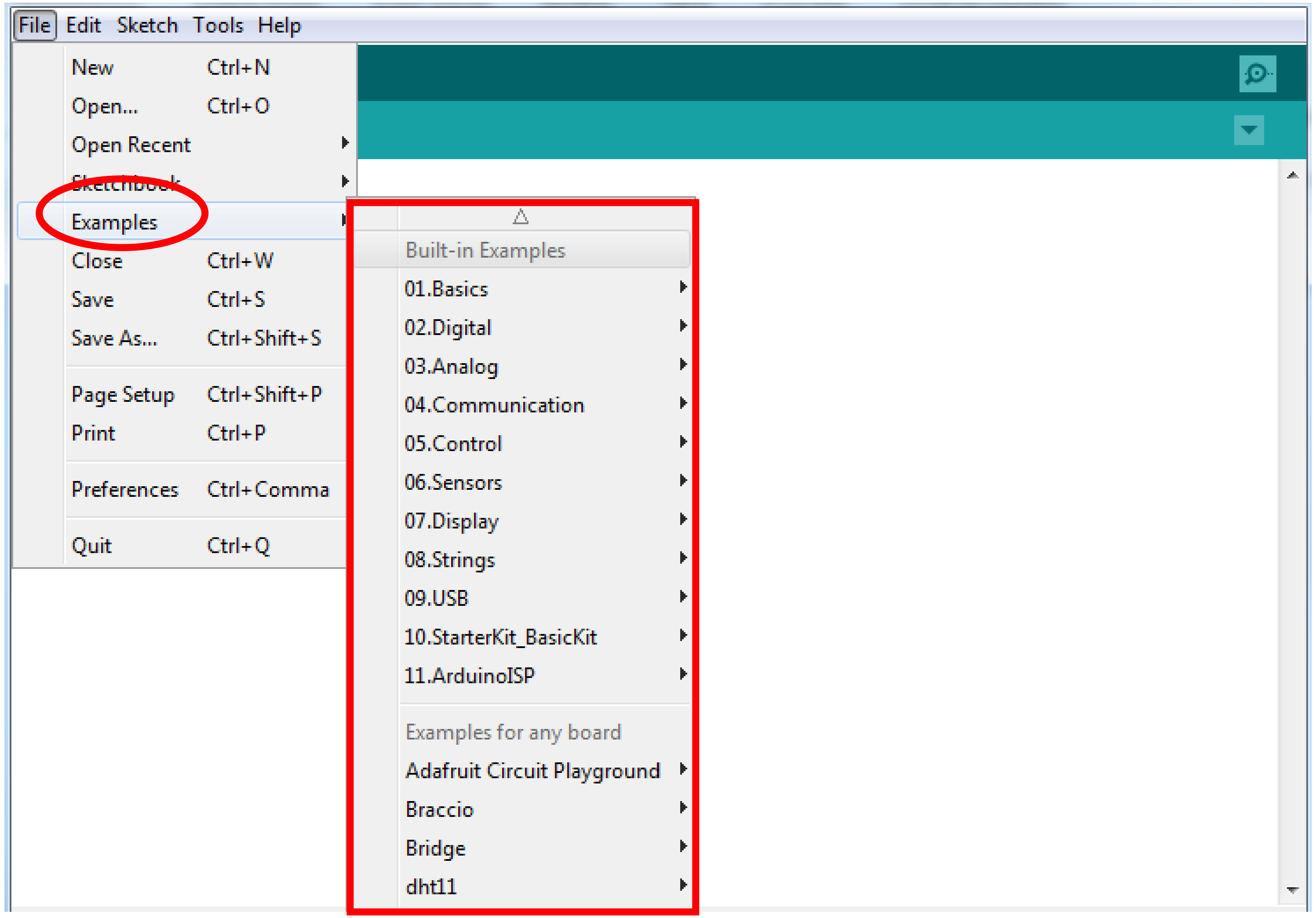
Codes:

```
// The aims for this program is to:
// (a) obtain pin output as 13 and
// (b) configure pin2 output to start from 1
// and be increasing by 1 per second

//Defining Global Variables pin and pin2
int pin = 13;
int pin2 = 0;

void setup() {
// put your setup code here, to run once:
  // This code helps to kickstart your
  // Serial Monitor
 Serial.begin(9600); }
```

```
void loop() {
// put your main code here, to run
// repeatedly:

  //To increase pin2 output by 1 per loop
  pin2 = pin2 + 1;

 // print on Serial Monitor the outputs

Serial.println("pin: ");
  Serial.println(pin);
   Serial.println("pin2: ");
  Serial.println(pin2);

// Delay the loop by 1 second
  delay(1000);  }
```

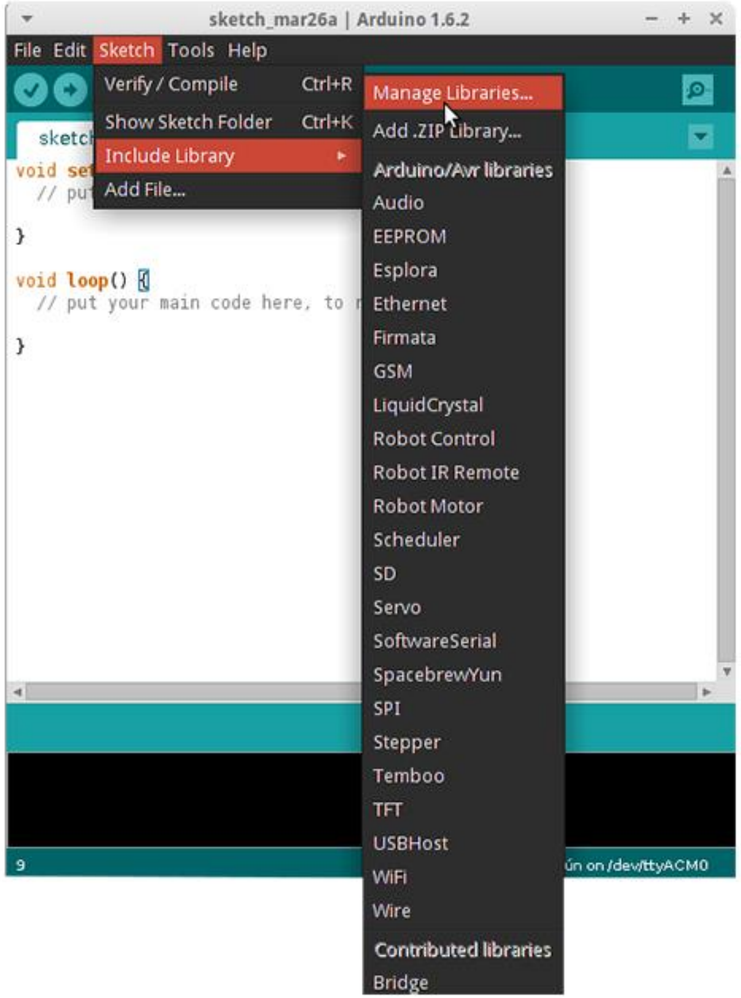
Arduino Libraries

- Go to *file*, then *examples*
- Here, there is a list of templates that you can refer to, to start building your codes



Installing Arduino Libraries

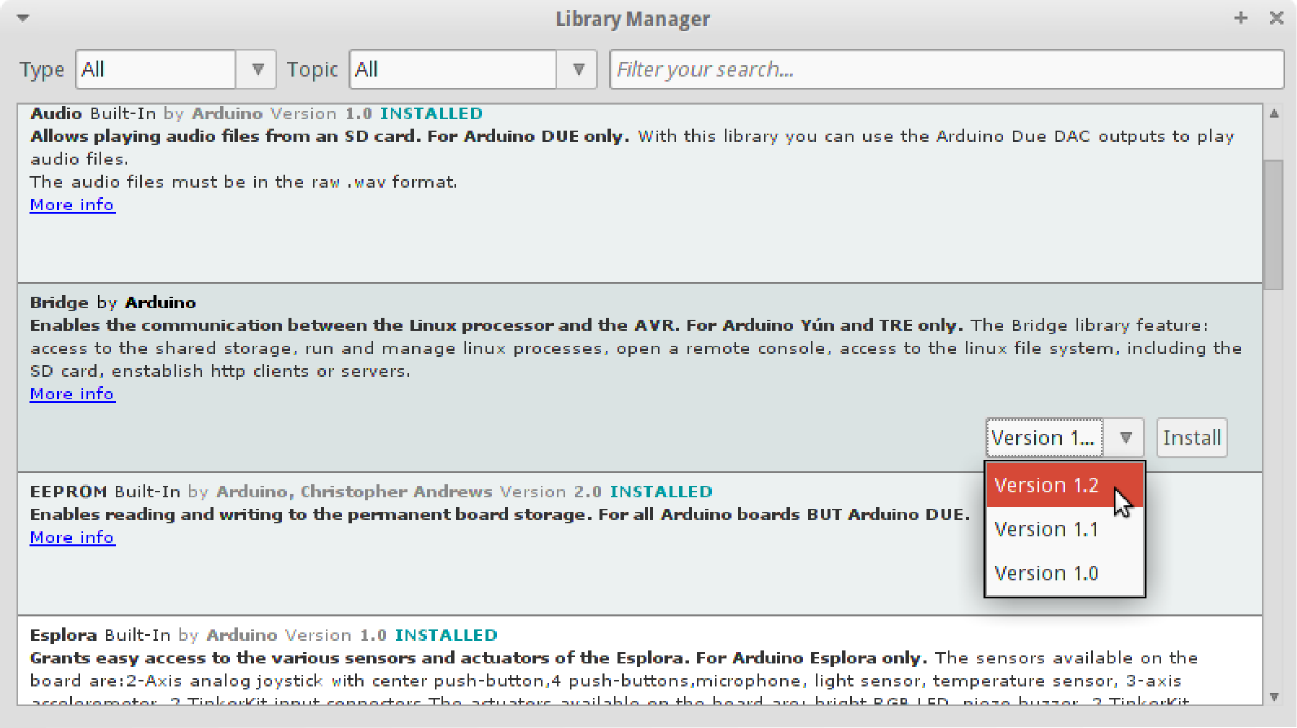
- To install a new library into your Arduino IDE, you can use the library Manager
- Click on the "Sketch" and then *Include Library > Manage Libraries*



```
void se
  // pu
}

void loop() {
  // put your main code here, to r
}
```

Installing Arduino Libraries

- The Library Manager will open and you can search the library that you want to install
- Click on install at the right side of the targeted library and wait for the IDE to install the new library
- An “INSTALLED” status will be reflected once successfully installed



Type All Topic All Filter your search...

Audio Built-In by **Arduino** Version **1.0** **INSTALLED**
Allows playing audio files from an SD card. For Arduino DUE only. With this library you can use the Arduino Due DAC outputs to play audio files.
The audio files must be in the raw .wav format.
More info

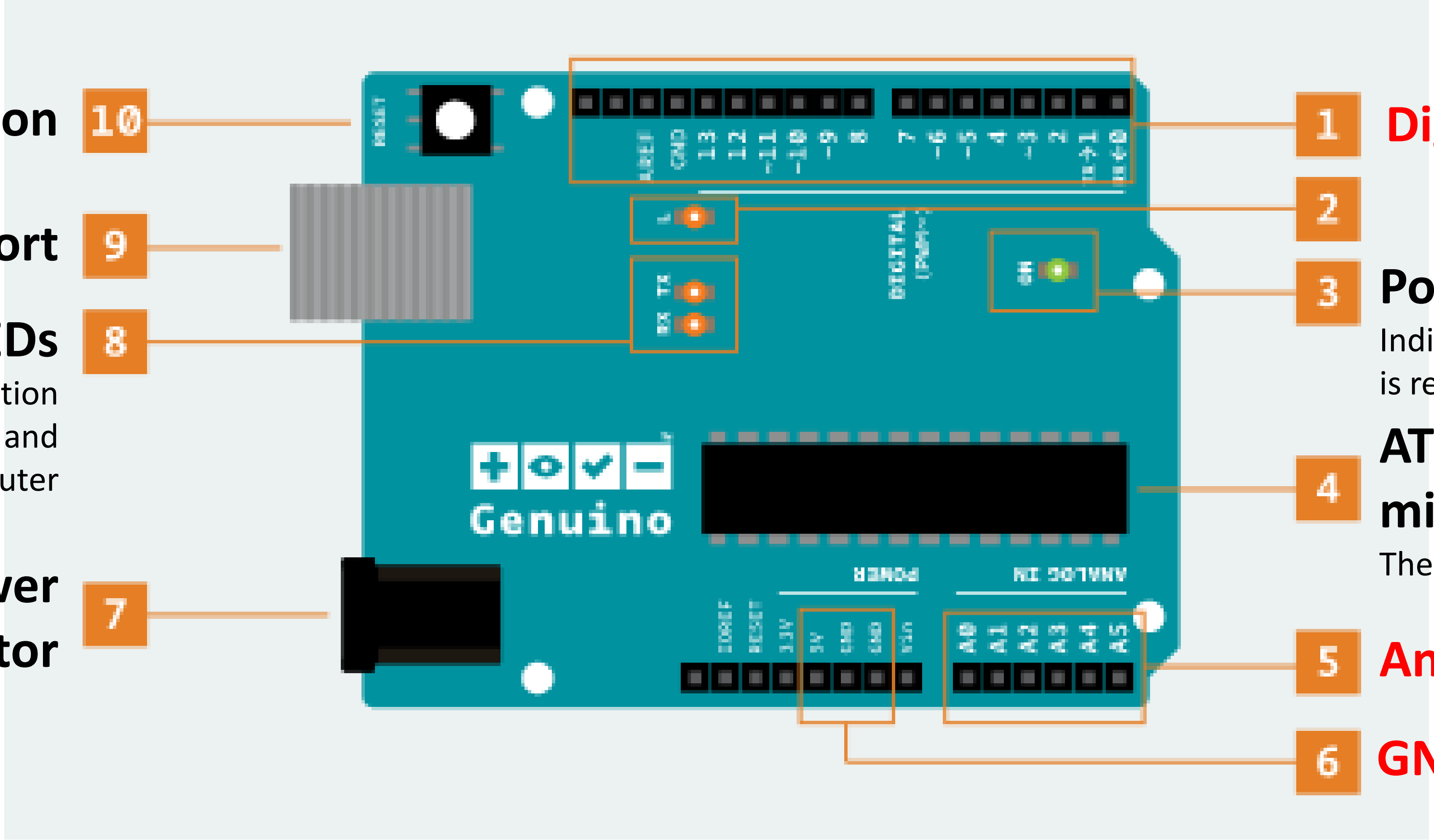
Bridge by **Arduino**
Enables the communication between the Linux processor and the AVR. For Arduino Yún and TRE only. The Bridge library feature: access to the shared storage, run and manage linux processes, open a remote console, access to the linux file system, including the SD card, enstablish http clients or servers.
More info

EEPROM Built-In by **Arduino, Christopher Andrews** Version **2.0** **INSTALLED**
Enables reading and writing to the permanent board storage. For all Arduino boards BUT Arduino DUE.
More info

Esplora Built-In by **Arduino** Version **1.0** **INSTALLED**
Grants easy access to the various sensors and actuators of the Esplora. For Arduino Esplora only. The sensors available on the board are:2-Axis analog joystick with center push-button,4 push-buttons,microphone, light sensor, temperature sensor, 3-axis accelerometer, 2 TinkerKit input connectors.The actuators available on the board are: bright RGB LED, piezo buzzer, 2 TinkerKit

Chapter 2: Configuring sensors in Arduino

Understanding the Arduino Uno Board (Recap)



Reset Button

USB port

TX & RX LEDs
indicate communication between your board and your computer

Power connector

Digital pins

Power LED
Indicates that your board is receiving power

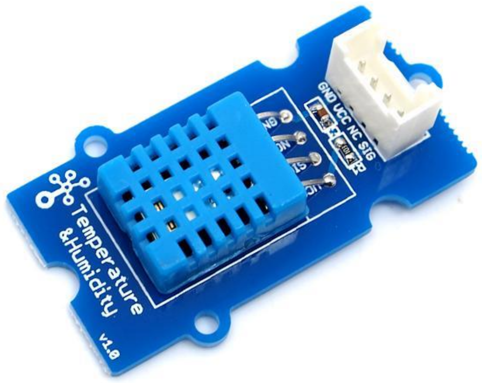
ATmega microcontroller
The heart of your board

Analog in

GND & 5V pins

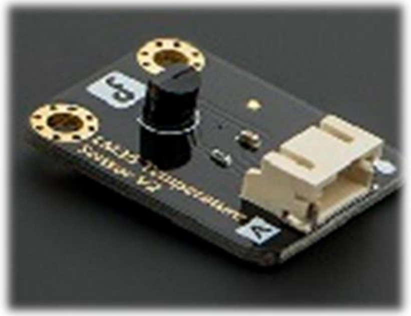
Understanding your sensors

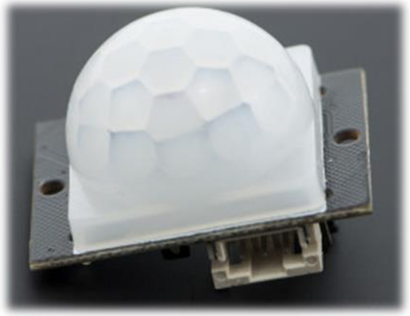
- **Light sensor – Analog**
- **Sound sensor – Analog**
- **Temperature & Humidity Sensor – Digital**



Understanding your sensors

- Linear Temperature sensor - Analog
- Infrared Motion sensor – Digital



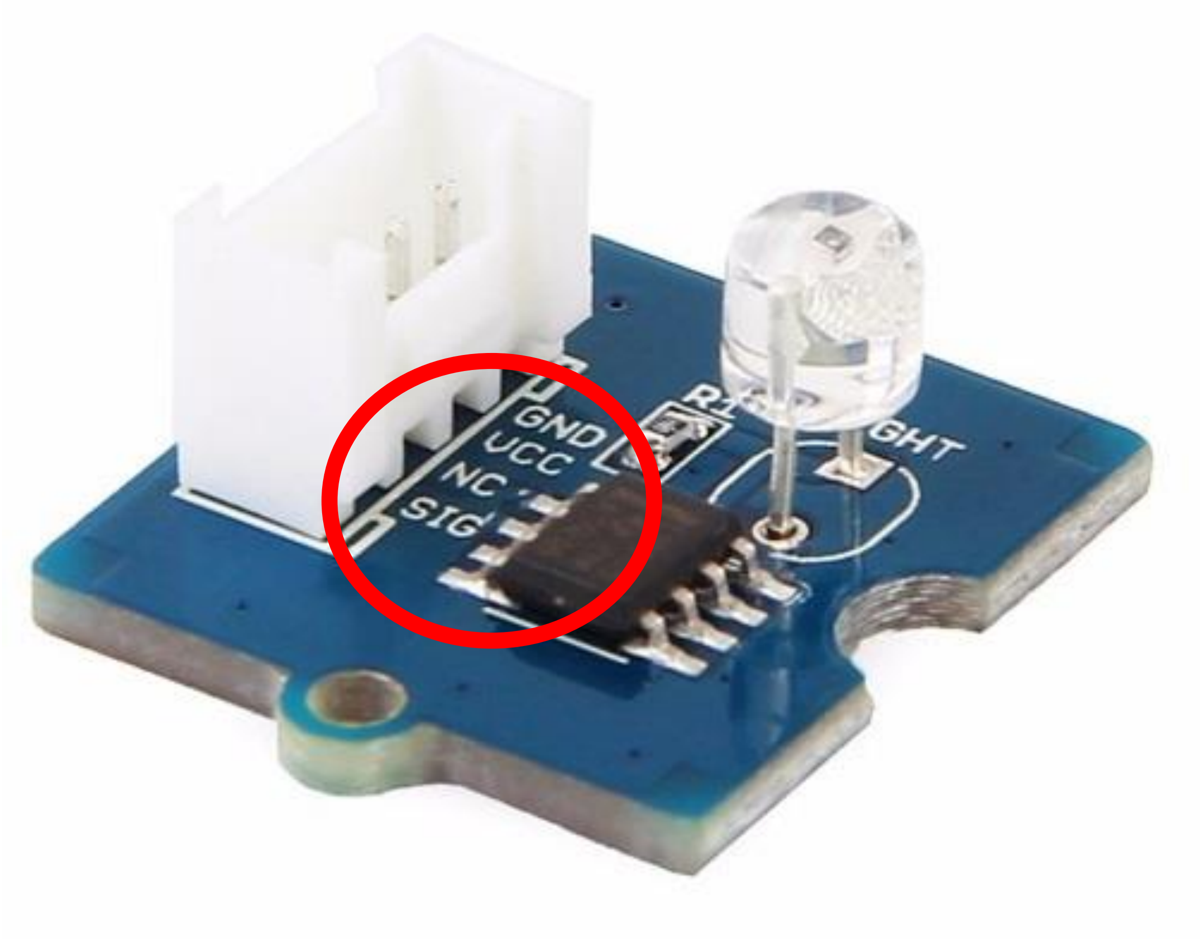


Important to know to guide you in plugging the respective sensor into the correct pin types; if you are not sure, you can easily find out by searching Google

NOTE: Industrial Grade Sensors are Different!

How different???

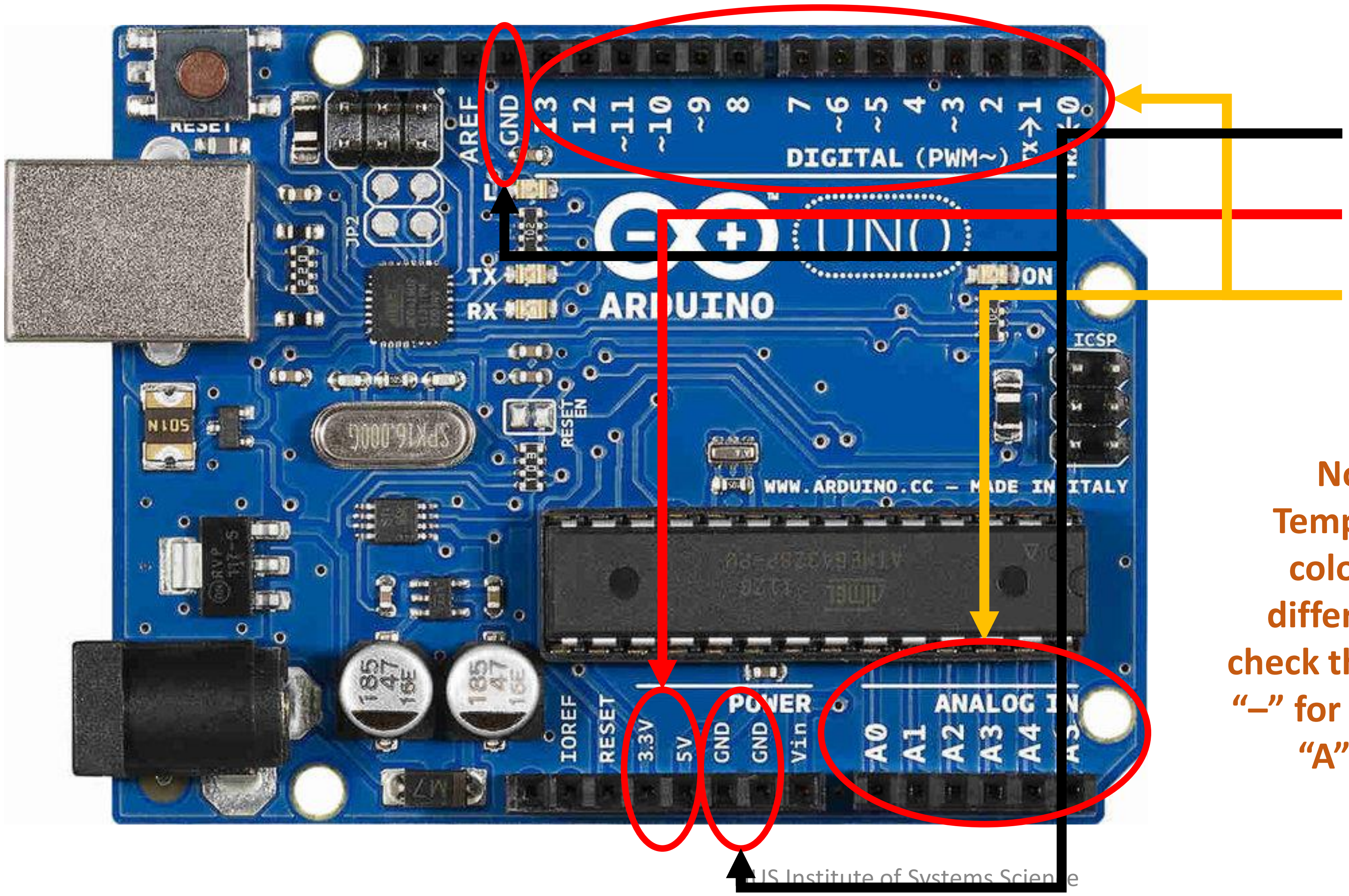
Understanding standard logic circuits



GND – Ground (connects to '-ve' pole)
VCC – Input Voltage (connects to '+ve' pole of power supply)
NC – Not Connected (not important)
SIG – Output Signal (to board)

For Grove Sensors (mostly used for this workshop), each wire colour is represented by each circuit connection:
GND – Black
VCC – Red
NC – White (not important)
SIG – Yellow

Plugging in the sensors to the board



From Grove Sensor

GND (Ground) – Black

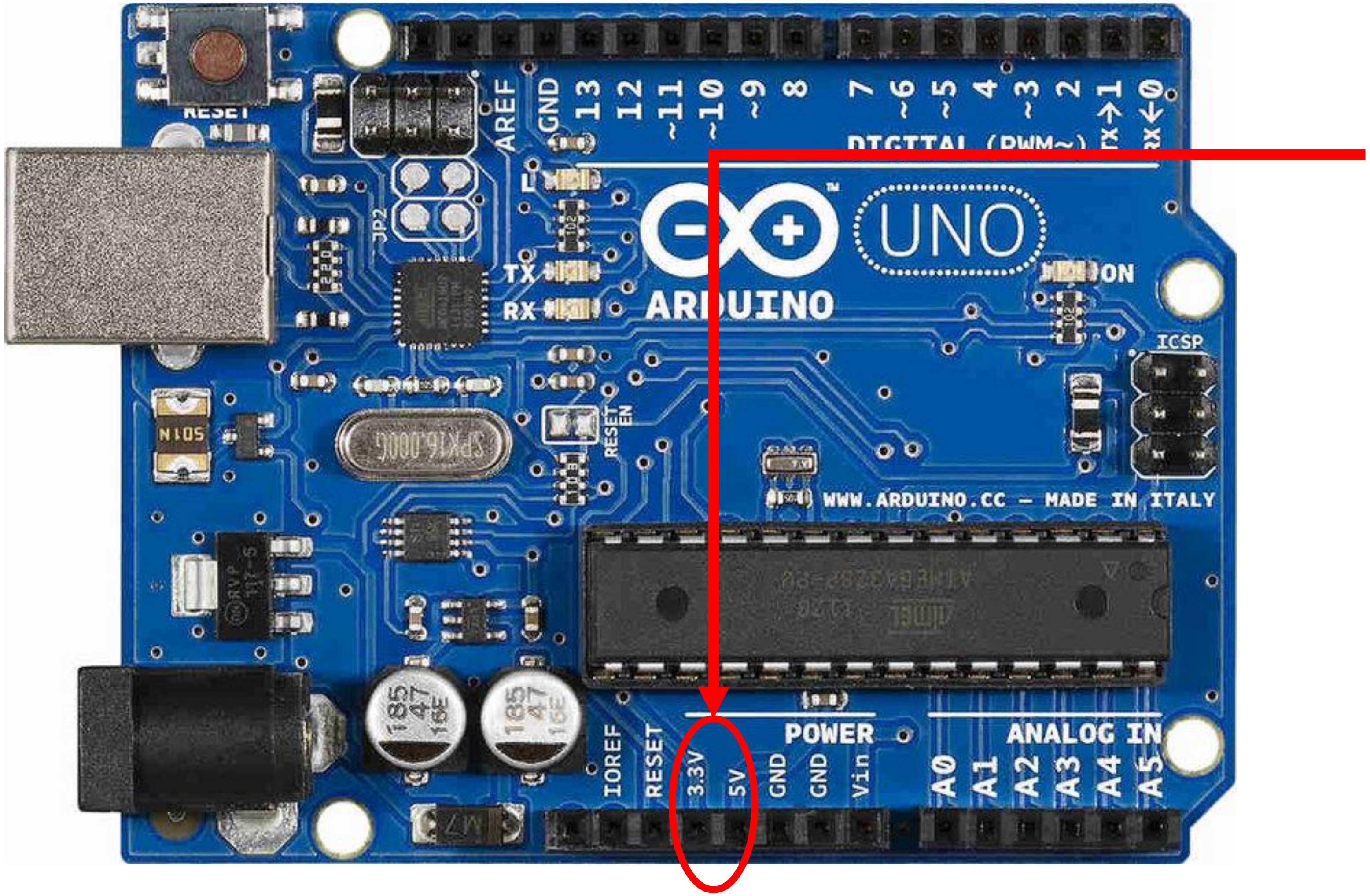
VCC (Input Voltage) – Red

SIG (Output Signal) – Yellow

NC – White (ignore)

Note that for Gravity LM35 Temperature Sensor, the various colours of each wire represent differently from this; you have to check the signs on the mini board (i.e. "–" for Ground, "+" for Input Voltage, "A" for Analog Output Signal)

Plugging in the sensors to the board



<u>From Grove Sensor</u>

VCC (Input Voltage) – Red

Note that each sensor has their own operating voltage range:

Grove Light Sensor → 3~5 V

Grove Sound Sensor → 4~12 V

Grove DHT Sensor → 3.3~5 V

Gravity LM35 Temperature Sensor
→ 3.3~5 V

Gravity Infrared Motion Sensor
→ 3~5V

Testing the Sensors

- We will now test the sensors using only the Arduino board
- **Prepare 3 jumper wires** to connect to GND, VCC and SIG
- **Open up the Arduino software**
- On new terminal, **type the following to give the port permission first:**

```
$ sudo chmod a+rw /dev/ttyACM0
```

Testing the Grove Light Sensor

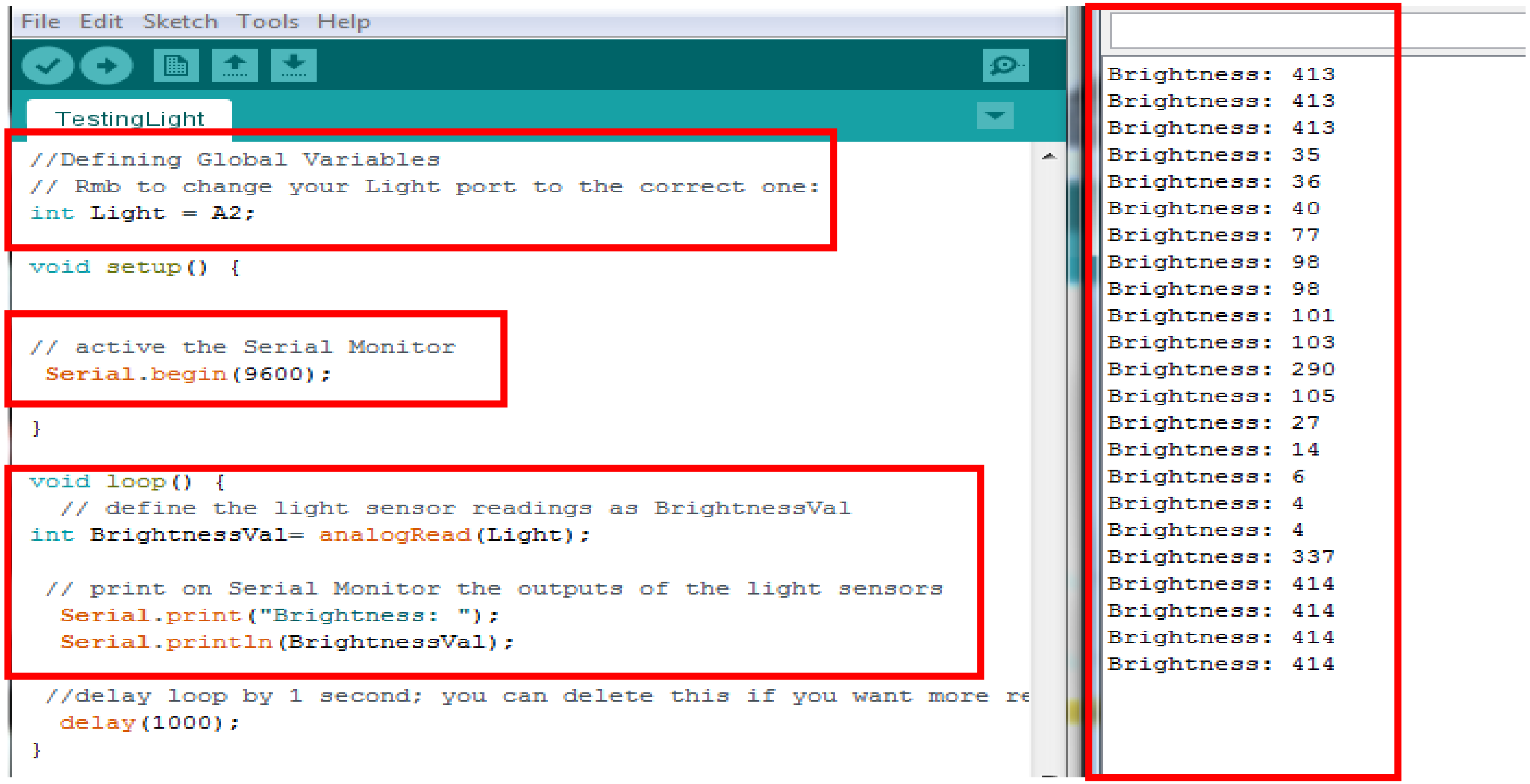
- **Connect the jumper wires correctly to wires of the Grove Light Sensor;**
- From GND port to the black wire of the sensor, 3.3/5V port to the red wire of the sensor, any of the analog ports (i.e. A0 to A5) to the yellow wire of the sensor; connect the USB when you are ready
- ***Note that you must connect the GND wire first**

Testing the Grove Light Sensor

- **Connect the board to your laptop** via the USB
- **Open up your Arduino Software** and **open up the sketch:** "*TestingLight*" (will be sent to you)
- **Edit the necessary codes** (i.e. port no.)
- **Upload to your board**
- **Open up the Serial Monitor** to monitor the light readings; you can **test it** by covering the light sensor to see if it responds

Testing the Grove Light Sensor

“TestingLight” Sketch



```
//Defining Global Variables
// Rmb to change your Light port to the correct one:
int Light = A2;

void setup() {

// active the Serial Monitor
 Serial.begin(9600);

}

void loop() {
  // define the light sensor readings as BrightnessVal
int BrightnessVal= analogRead(Light);

 // print on Serial Monitor the outputs of the light sensors
  Serial.print("Brightness: ");
  Serial.println(BrightnessVal);

 //delay loop by 1 second; you can delete this if you want more re
  delay(1000);
}
```

```
Brightness: 413
Brightness: 413
Brightness: 413
Brightness: 35
Brightness: 36
Brightness: 40
Brightness: 77
Brightness: 98
Brightness: 98
Brightness: 101
Brightness: 103
Brightness: 290
Brightness: 105
Brightness: 27
Brightness: 14
Brightness: 6
Brightness: 4
Brightness: 4
Brightness: 337
Brightness: 414
Brightness: 414
Brightness: 414
Brightness: 414
```

Testing the Grove Sound Sensor

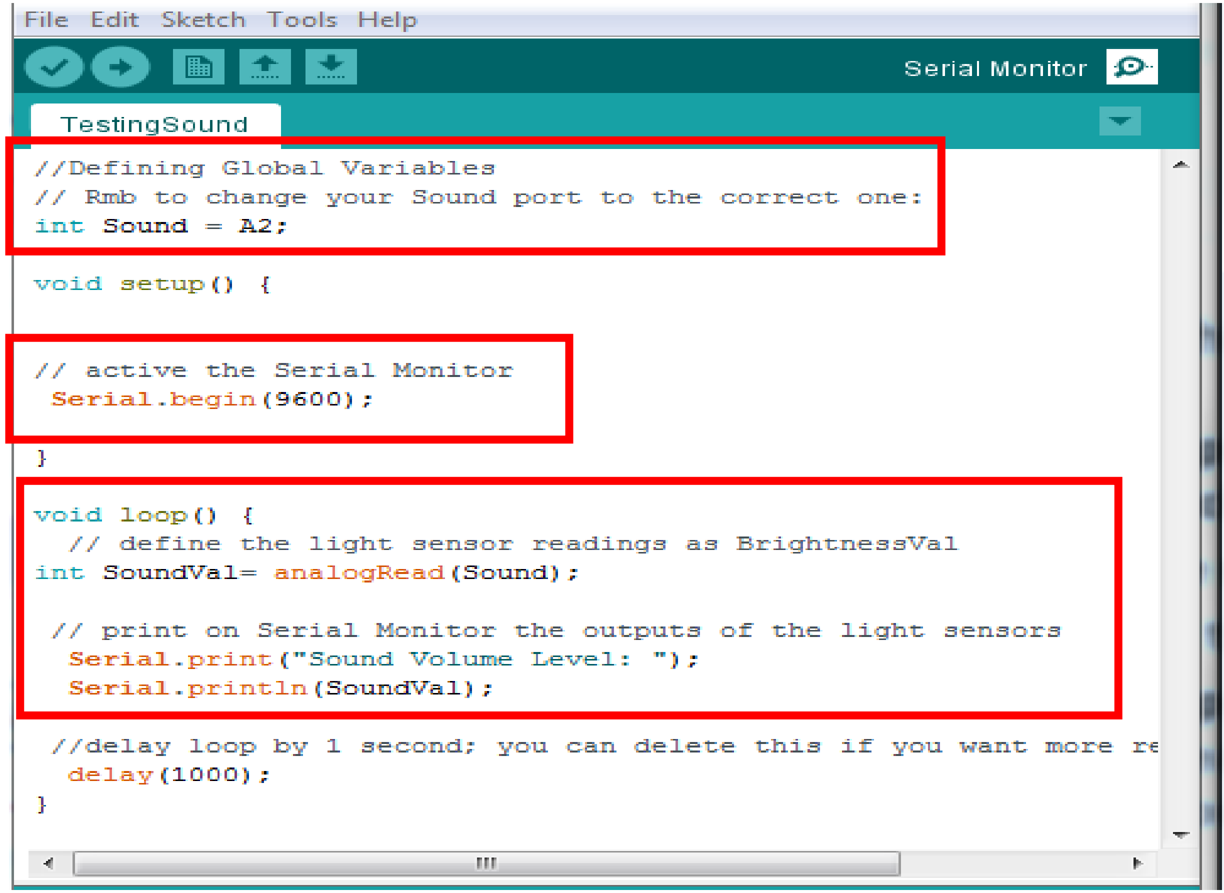
- **Remove the USB to ensure power is off**
- **Connect the jumper wires correctly to wires of the Grove Sound Sensor;**
- From GND port to the black wire of the sensor, **5V port to the red wire of the sensor**, any of the analog ports (i.e. A0 to A5) to the yellow wire of the sensor; connect back the USB when you are ready
- ***Note that you must connect the GND wire first**

Testing the Grove Sound Sensor

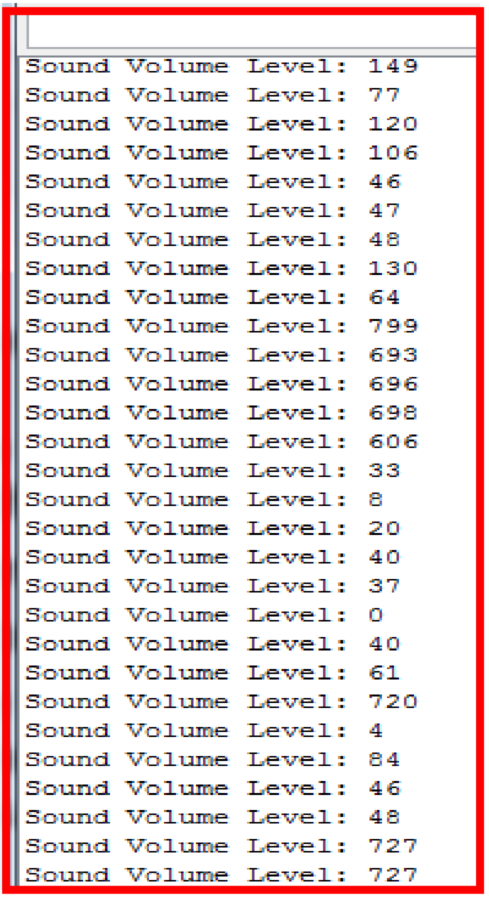
- **Open up your Arduino Software** and **open up the sketch:** “*TestingSound*” (will be sent to you)
- **Edit the necessary codes** (i.e. port no.)
- **Upload to your board**
- **Open up the Serial Monitor** to monitor the sound readings; you can **test it** by making noise to the sound sensor to see if it responds

Testing the Grove Sound Sensor

"TestingSound" Sketch



```
//Defining Global Variables
// Rmb to change your Sound port to the correct one:
int Sound = A2;

void setup() {

// active the Serial Monitor
 Serial.begin(9600);

}

void loop() {
  // define the light sensor readings as BrightnessVal
int SoundVal= analogRead(Sound);

 // print on Serial Monitor the outputs of the light sensors
  Serial.print("Sound Volume Level: ");
  Serial.println(SoundVal);

 //delay loop by 1 second; you can delete this if you want more re
  delay(1000);
}
```



```
Sound Volume Level: 149
Sound Volume Level: 77
Sound Volume Level: 120
Sound Volume Level: 106
Sound Volume Level: 46
Sound Volume Level: 47
Sound Volume Level: 48
Sound Volume Level: 130
Sound Volume Level: 64
Sound Volume Level: 799
Sound Volume Level: 693
Sound Volume Level: 696
Sound Volume Level: 698
Sound Volume Level: 606
Sound Volume Level: 33
Sound Volume Level: 8
Sound Volume Level: 20
Sound Volume Level: 40
Sound Volume Level: 37
Sound Volume Level: 0
Sound Volume Level: 40
Sound Volume Level: 61
Sound Volume Level: 720
Sound Volume Level: 4
Sound Volume Level: 84
Sound Volume Level: 46
Sound Volume Level: 48
Sound Volume Level: 727
Sound Volume Level: 727
```

Testing the Grove DHT Sensor

- **Remove the USB to ensure power is off**
- **Connect the jumper wires correctly to wires of the Grove DHT Sensor;**
- From GND port to the black wire of the sensor, 3.3/5V port to the red wire of the sensor, any of the **digital ports (i.e. 2 to 13)** to the yellow wire of the sensor; connect back the USB when you are ready
- ***Note that you must connect the GND wire first**

Testing the Grove DHT Sensor

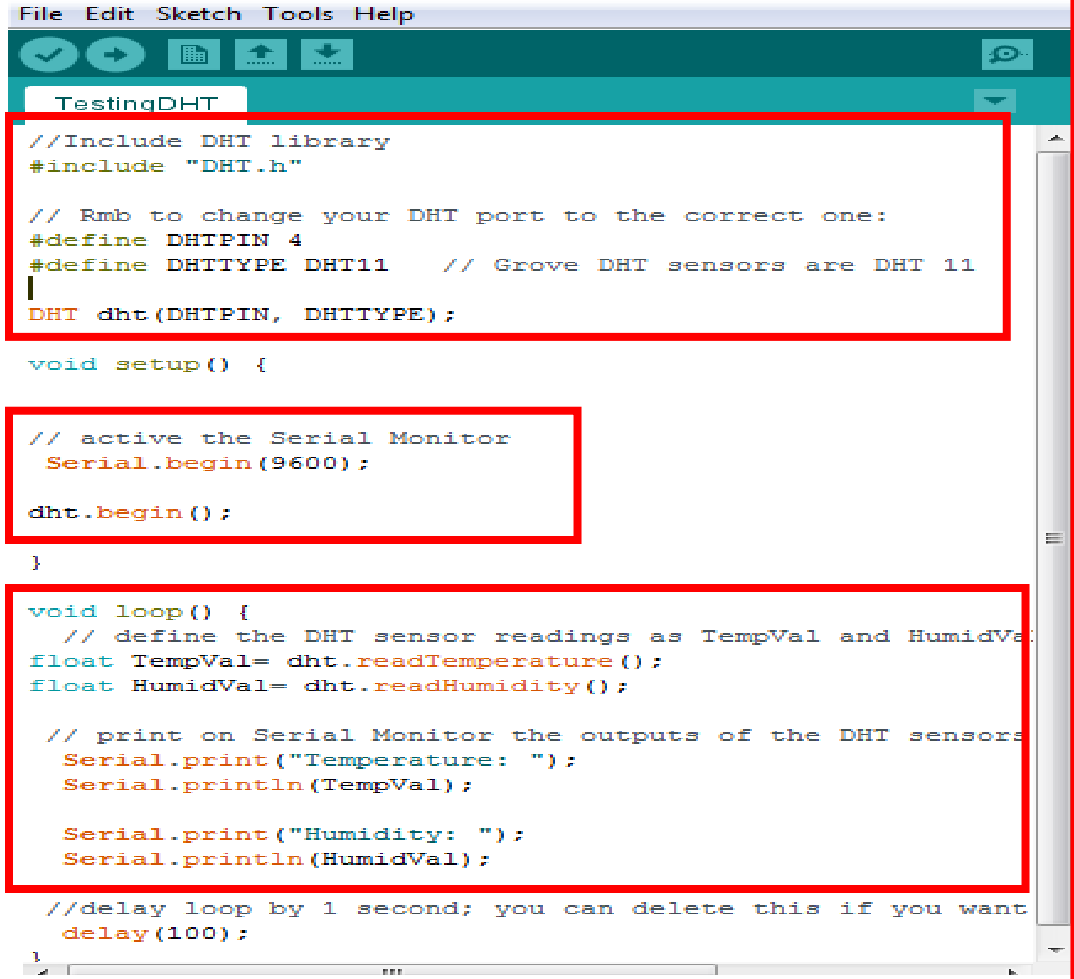
- DHT Sensor is a **special digital sensor** that are able to read two analog data (i.e. temp, humid)
- To extract the analog data from the digital data, you will **need a DHT library**
- Go to **“Manage Libraries”**, **search** *“Grove Temperature And Humidity Sensor”*
- **Install the library:** *“Grove Temperature And Humidity Sensor Built-In by Seeed Studio”*
- **Restart the Arduino Software**

Testing the Grove DHT Sensor

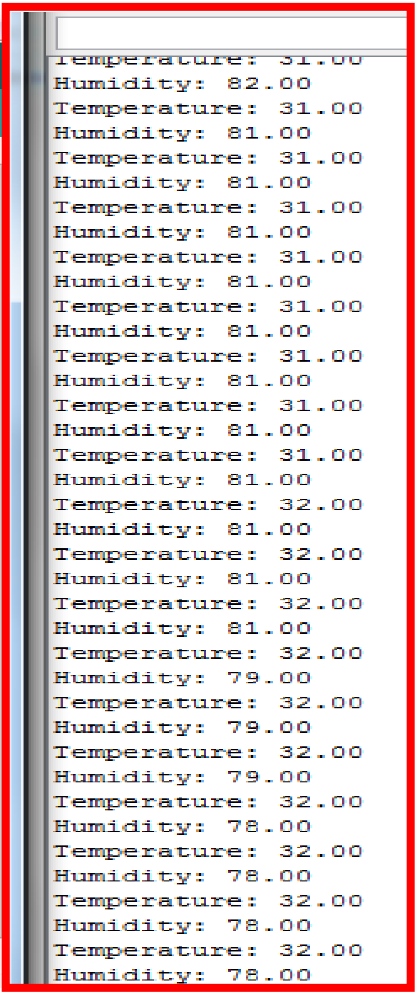
- **Open up the sketch:** "*TestingDHT*" (will be sent to you)
- **Edit the necessary codes** (i.e. port no.)
- **Upload to your board**
- **Open up the Serial Monitor** to monitor the temperature/humidity readings; you can **test it** by pressing on the DHT sensor to see if it responds

Testing the Grove DHT Sensor

“TestingDHT” Sketch



```
//Include DHT library
#include "DHT.h"

// Rmb to change your DHT port to the correct one:
#define DHTPIN 4
#define DHTTYPE DHT11   // Grove DHT sensors are DHT 11

DHT dht(DHTPIN, DHTTYPE);

void setup() {

// active the Serial Monitor
 Serial.begin(9600);

dht.begin();

}

void loop() {
  // define the DHT sensor readings as TempVal and HumidVa
float TempVal= dht.readTemperature();
float HumidVal= dht.readHumidity();

 // print on Serial Monitor the outputs of the DHT sensors
  Serial.print("Temperature: ");
  Serial.println(TempVal);

  Serial.print("Humidity: ");
  Serial.println(HumidVal);

 //delay loop by 1 second; you can delete this if you want
  delay(100);
}
```



```
Temperature: 31.00
Humidity: 82.00
Temperature: 31.00
Humidity: 81.00
Temperature: 31.00
Humidity: 81.00
Temperature: 31.00
Humidity: 81.00
Temperature: 31.00
Humidity: 81.00
Temperature: 31.00
Humidity: 81.00
Temperature: 31.00
Humidity: 81.00
Temperature: 31.00
Humidity: 81.00
Temperature: 31.00
Humidity: 81.00
Temperature: 32.00
Humidity: 81.00
Temperature: 32.00
Humidity: 81.00
Temperature: 32.00
Humidity: 81.00
Temperature: 32.00
Humidity: 79.00
Temperature: 32.00
Humidity: 79.00
Temperature: 32.00
Humidity: 79.00
Temperature: 32.00
Humidity: 78.00
Temperature: 32.00
Humidity: 78.00
Temperature: 32.00
Humidity: 78.00
Temperature: 32.00
Humidity: 78.00
```

Configuring Two Sensors on the same board

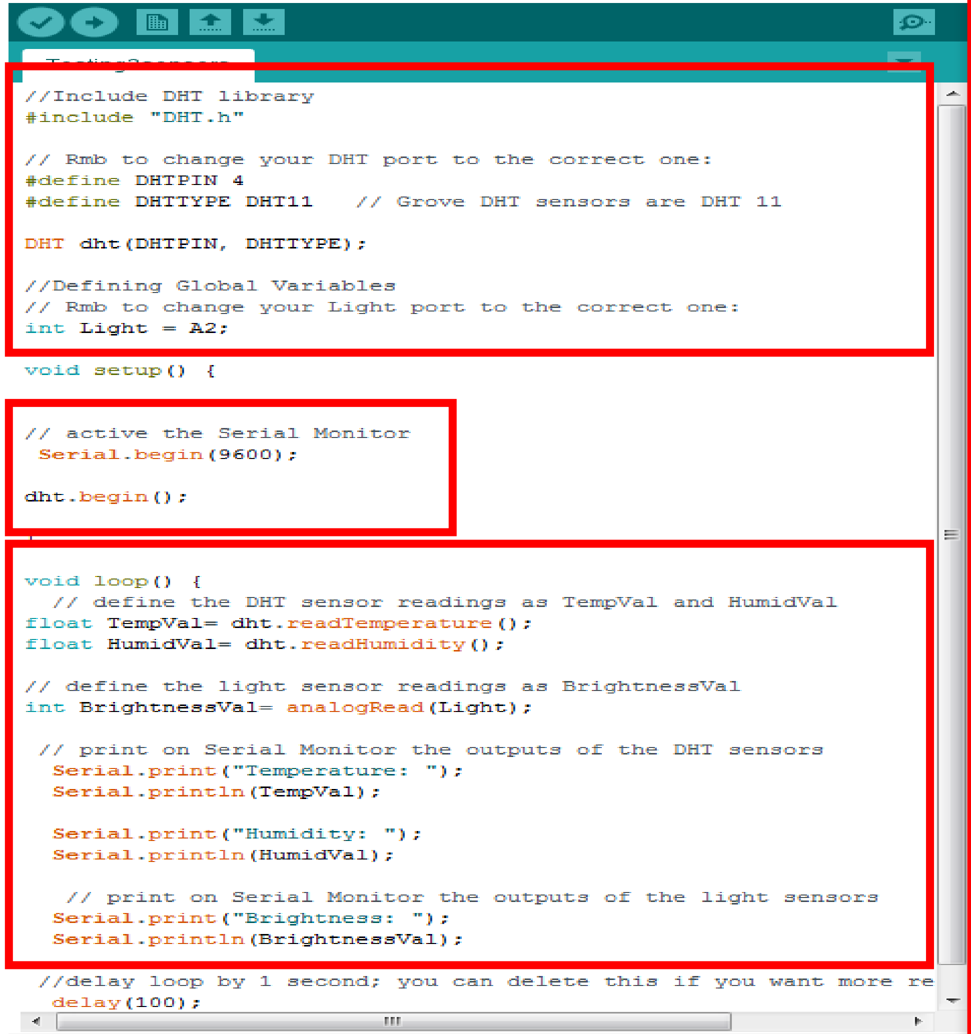
- **Remove the USB to ensure power is off**
- **Leaving the DHT sensor connected, connect the jumper wires correctly to wires of the Grove Light Sensor**;
- From GND port to the black wire of the sensor, 3.3/5V port to the red wire of the sensor, any of the analog ports (i.e. A0 to A5) to the yellow wire of the sensor; connect back the USB when you are ready
- ***Note that you must connect the GND wire first**

Configuring Two Sensors on the same board

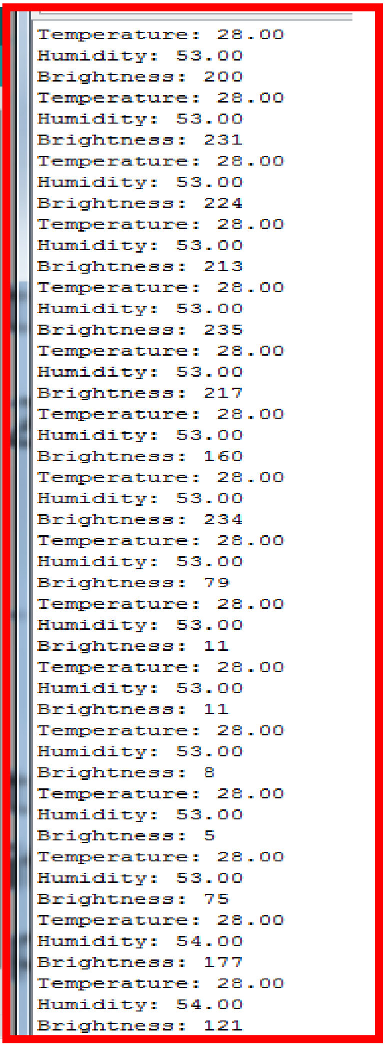
- **Open up the sketch:** “*Testing2sensors*” (will be sent to you)
- **Edit the necessary codes** (i.e. port no.)
- **Upload to your board**
- **Open up the Serial Monitor** to monitor the sound readings; you can **test it** by pressing on the DHT sensor to see if it responds

Configuring Two Sensors on the same board

“Testing2sensors” Sketch



```
//Include DHT library
#include "DHT.h"

// Rmb to change your DHT port to the correct one:
#define DHTPIN 4
#define DHTTYPE DHT11   // Grove DHT sensors are DHT 11

DHT dht(DHTPIN, DHTTYPE);

//Defining Global Variables
// Rmb to change your Light port to the correct one:
int Light = A2;

void setup() {

// active the Serial Monitor
 Serial.begin(9600);

dht.begin();
}

void loop() {
  // define the DHT sensor readings as TempVal and HumidVal
float TempVal= dht.readTemperature();
float HumidVal= dht.readHumidity();

// define the light sensor readings as BrightnessVal
int BrightnessVal= analogRead(Light);

 // print on Serial Monitor the outputs of the DHT sensors
  Serial.print("Temperature: ");
  Serial.println(TempVal);

  Serial.print("Humidity: ");
  Serial.println(HumidVal);

   // print on Serial Monitor the outputs of the light sensors
  Serial.print("Brightness: ");
  Serial.println(BrightnessVal);

 //delay loop by 1 second; you can delete this if you want more re
  delay(100);
```



```
Temperature: 28.00
Humidity: 53.00
Brightness: 200
Temperature: 28.00
Humidity: 53.00
Brightness: 231
Temperature: 28.00
Humidity: 53.00
Brightness: 224
Temperature: 28.00
Humidity: 53.00
Brightness: 213
Temperature: 28.00
Humidity: 53.00
Brightness: 235
Temperature: 28.00
Humidity: 53.00
Brightness: 217
Temperature: 28.00
Humidity: 53.00
Brightness: 160
Temperature: 28.00
Humidity: 53.00
Brightness: 234
Temperature: 28.00
Humidity: 53.00
Brightness: 79
Temperature: 28.00
Humidity: 53.00
Brightness: 11
Temperature: 28.00
Humidity: 53.00
Brightness: 11
Temperature: 28.00
Humidity: 53.00
Brightness: 8
Temperature: 28.00
Humidity: 53.00
Brightness: 5
Temperature: 28.00
Humidity: 53.00
Brightness: 75
Temperature: 28.00
Humidity: 54.00
Brightness: 177
Temperature: 28.00
Humidity: 54.00
Brightness: 121
```

Testing the Grove Ultrasonic Sensor

- **Remove the USB to ensure power is off**
- **Connect the jumper wires correctly to wires of the Grove Ultrasonic Sensor;**
- From GND port to the black wire of the sensor, 3.3/5V port to the red wire of the sensor, **digital port 7** to the yellow wire of the sensor; connect back the USB when you are ready
- ***Note that you must connect the GND wire first**

Testing the Grove Ultrasonic Sensor

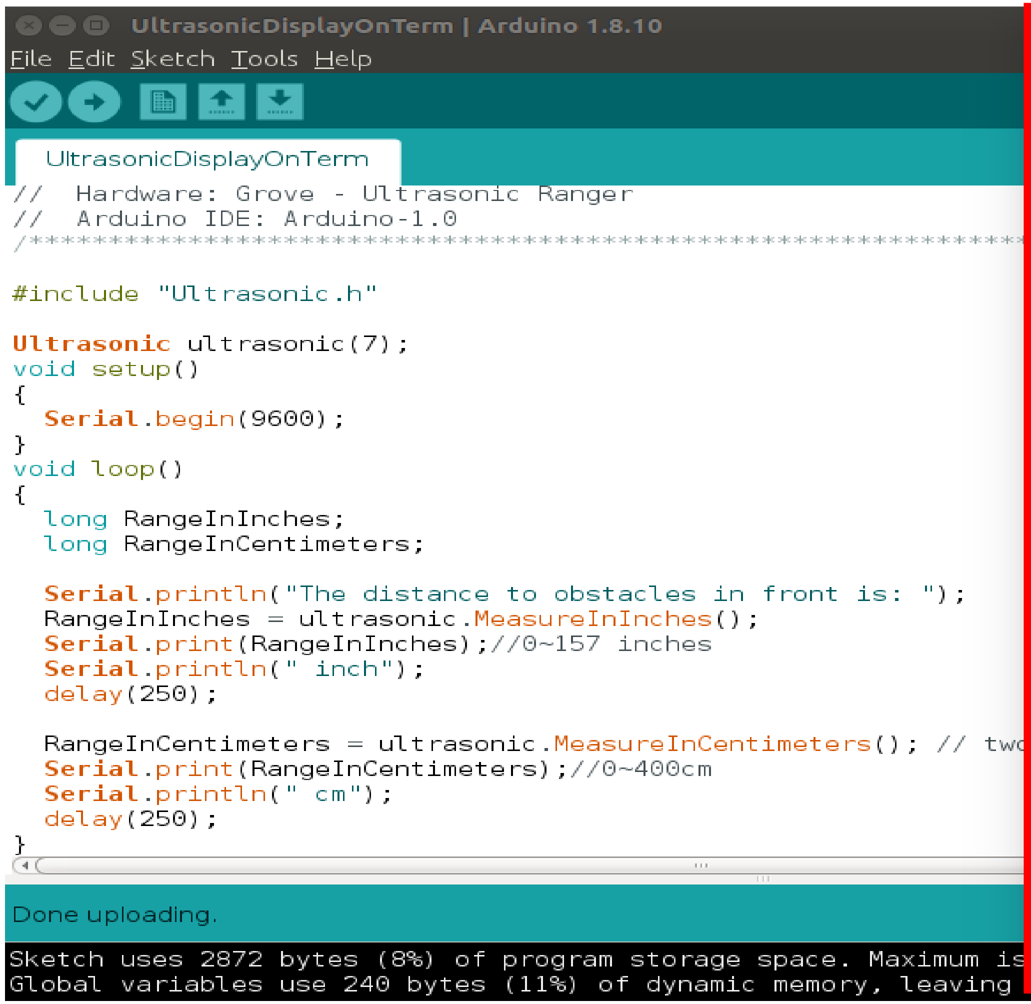
- Grove Ultrasonic Sensor is a **special digital sensor** that are able to measure distances via “True” and “False” readings; **HOW???**
- Go to **“Tools” → “Manage Libraries” in Arduino**, **search** *“Grove Ultrasonic Ranger”*
- **Install the library:** *“Grove Ultrasonic Ranger Built-In by Seeed Studio”*
- **Restart the Arduino Software**

Testing the Grove Ultrasonic Sensor

- **In Arduino software, go to "Files" → "Examples" → "Grove Ultrasonic Ranger"**
- **Open up the sketch:** "*UltrasonicDisplayOnTerm*"
- **Upload to your board**
- **Open up the Serial Monitor** to monitor the distance readings; you can **test it** by blocking the sonar sensor with your hand to see if it responds

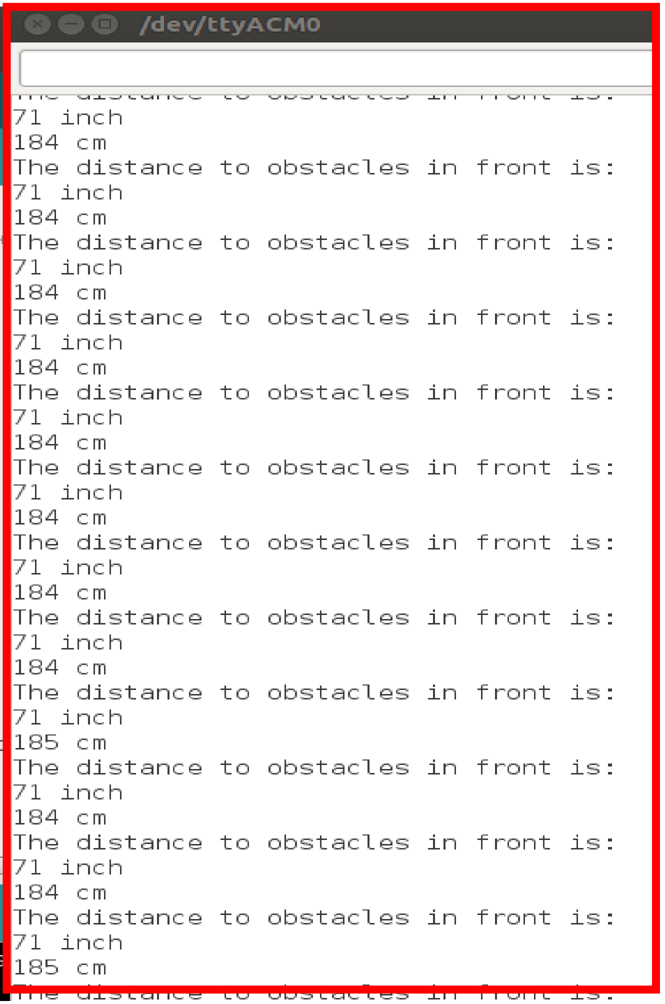
Testing the Grove Ultrasonic Sensor

"UltrasonicDisplayOnTerm" Sketch


UltrasonicDisplayOnTerm | Arduino 1.8.10
File Edit Sketch Tools Help
UltrasonicDisplayOnTerm

```
//  Hardware: Grove - Ultrasonic Ranger
//  Arduino IDE: Arduino-1.0
/*************************************************************

#include "Ultrasonic.h"

Ultrasonic ultrasonic(7);
void setup()
{
  Serial.begin(9600);
}
void loop()
{
  long RangeInInches;
  long RangeInCentimeters;

  Serial.println("The distance to obstacles in front is: ");
  RangeInInches = ultrasonic.MeasureInInches();
  Serial.print(RangeInInches);//0~157 inches
  Serial.println(" inch");
  delay(250);

  RangeInCentimeters = ultrasonic.MeasureInCentimeters(); // two
  Serial.print(RangeInCentimeters);//0~400cm
  Serial.println(" cm");
  delay(250);
}
```

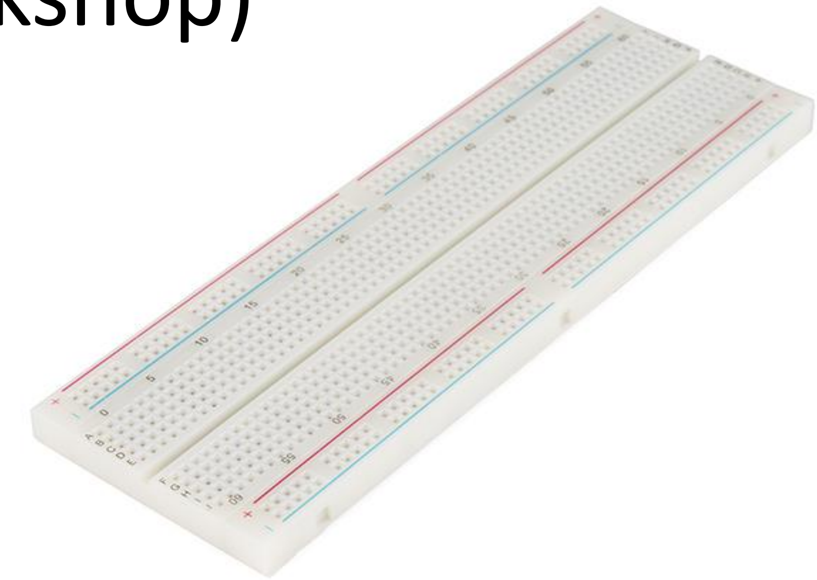
```
Done uploading.
Sketch uses 2872 bytes (8%) of program storage space. Maximum is
Global variables use 240 bytes (11%) of dynamic memory, leaving
```

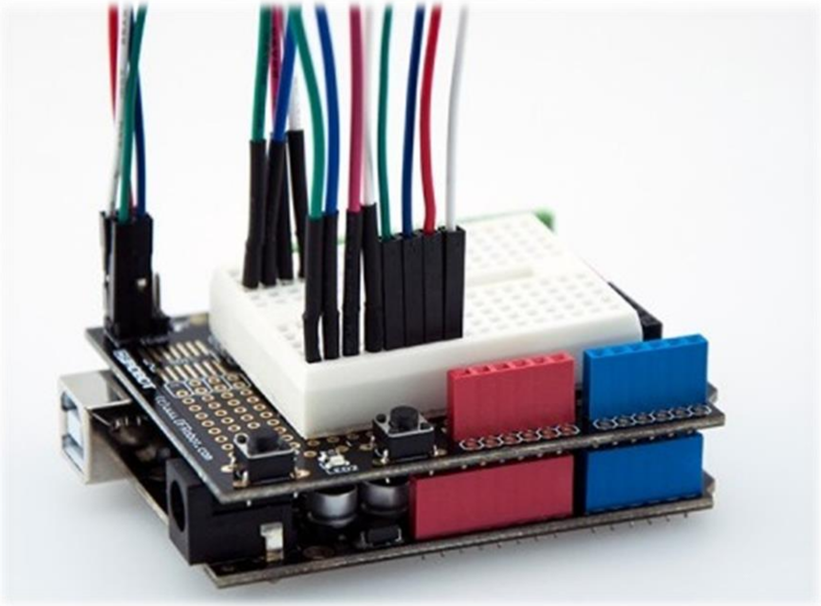


```
The distance to obstacles in front is:
71 inch
184 cm
The distance to obstacles in front is:
71 inch
184 cm
The distance to obstacles in front is:
71 inch
184 cm
The distance to obstacles in front is:
71 inch
184 cm
The distance to obstacles in front is:
71 inch
184 cm
The distance to obstacles in front is:
71 inch
184 cm
The distance to obstacles in front is:
71 inch
184 cm
The distance to obstacles in front is:
71 inch
185 cm
The distance to obstacles in front is:
71 inch
184 cm
The distance to obstacles in front is:
71 inch
184 cm
The distance to obstacles in front is:
71 inch
185 cm
The distance to obstacles in front is:
```

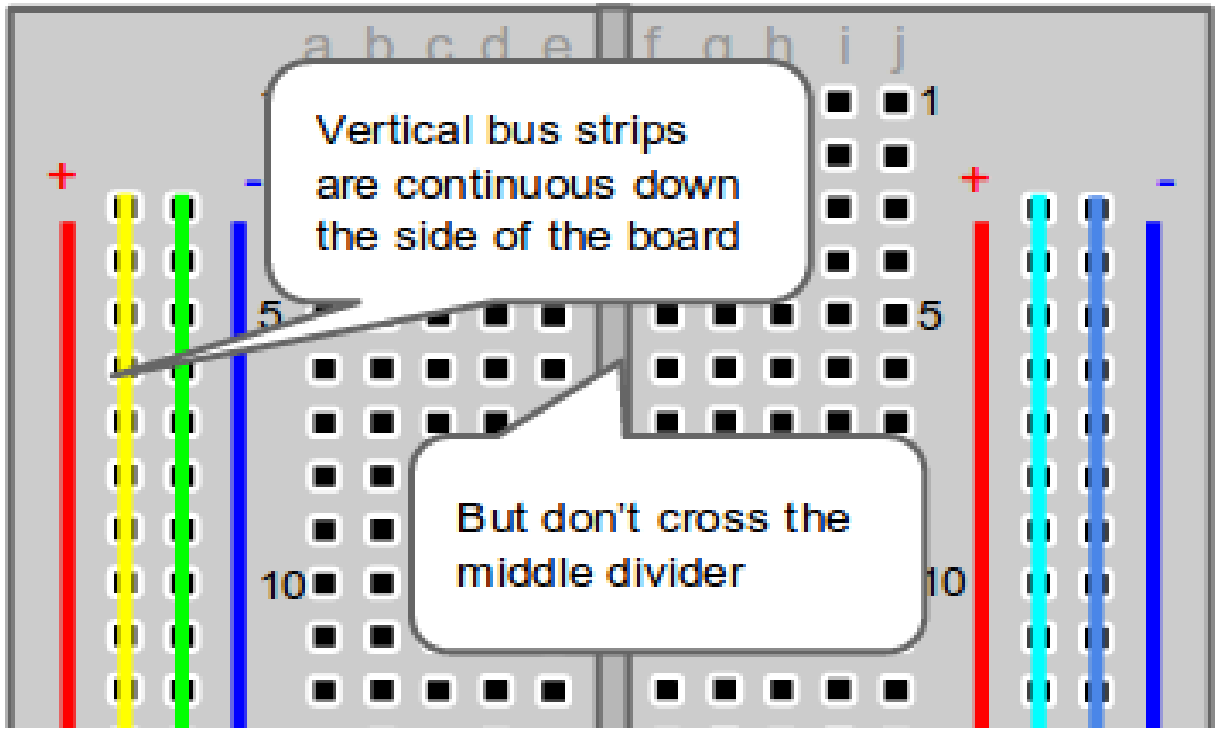
How to Configure Multiple Sensors (more than 2) on the Same Arduino Board

- **Breadboards (will be covered)**
- Prototyping Shield (won’t be covered in today’s workshop)





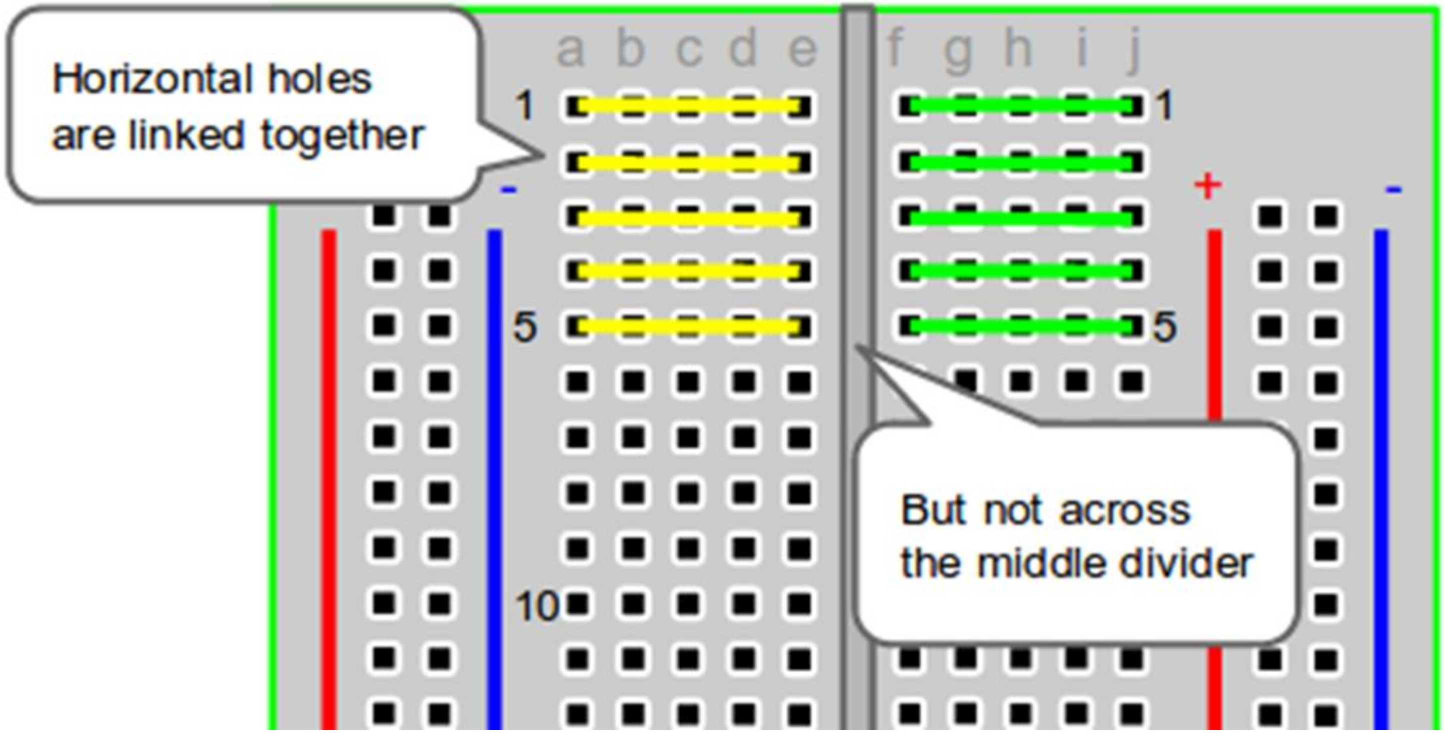
Breadboard Crash Course



Vertical bus strips are continuous down the side of the board

But don't cross the middle divider

Breadboard Crash Course



Horizontal holes are linked together

But not across the middle divider

Chapter 3: Programming robotic arm to respond differently to various sensors' input

Common Question

How can we utilize the sensors' data collected from Arduino so that we can control the OM via ROS to respond differently to these data?

pyFirmata

- Firmata = a protocol for communicating with microcontrollers from software on a computer. The protocol can be implemented in firmware on any microcontroller architecture as well as software on any computer software package
- **pyFirmata = a Python interface for the Firmata protocol**
- That is: a prebuilt library package of python program that allows serial communication between a python script on any computer and an Arduino; it can give access to read and write any pin on the Arduino
- **For this workshop, we want to use pyFirmata to read the sensors' values from the Arduino board to control the actuators (i.e. LED, Buzzer) and/or control the OM via rospy**

Setup to use pyFirmata

Step 1: Installing pyFirmata in Ubuntu

- Connect Internet first!
- Update packages, install pip and verify installation of pip

```
sudo apt update
sudo apt install python-pip
pip --version
```

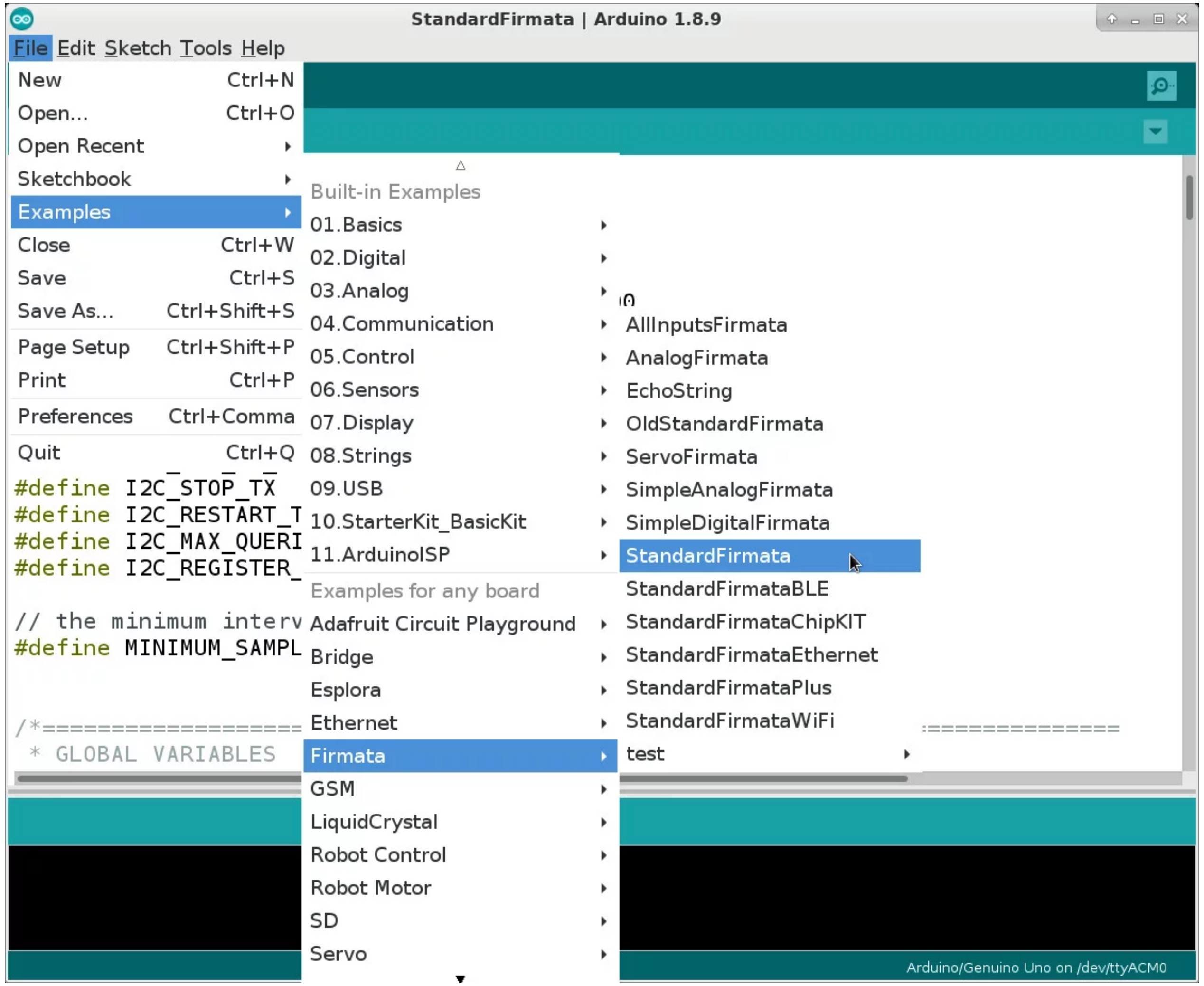
- Install pyFirmata

```
pip install pyfirmata
```

Setup to use pyFirmata

Step 2: Open Arduino IDE to open the *StandardFirmata* sketch

- Access the *File* menu, then *Examples*, followed by *Firmata*, and finally click on *StandardFirmata*



StandardFirmata | Arduino 1.8.9

```
#define I2C_STOP_TX
#define I2C_RESTART_T
#define I2C_MAX_QUERI
#define I2C_REGISTER_

// the minimum interv
#define MINIMUM_SAMPL


/*===================                    ===============
 * GLOBAL VARIABLES
```

Arduino/Genuino Uno on /dev/ttyACM0

Setup to use pyFirmata

Step 3: Upload the Firmata sketch into the Arduino board so that you can use pyfirmata to control the board

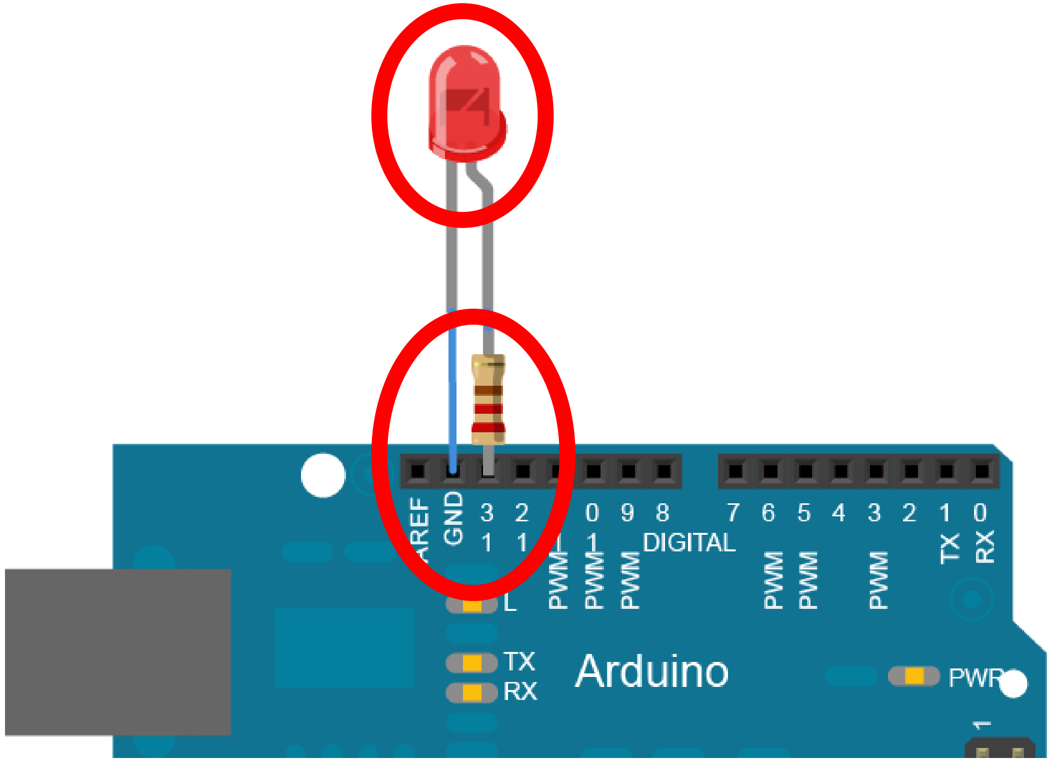
- Connect the PC and the Arduino board via the USB cable
- Select the appropriate board (i.e. Uno) & port (e.g. ACM0) on the IDE
- Upload the *StandardFirmata* sketch to the board
- After the upload is finished, you won’t notice any activity on the Arduino (even when you click on Serial Comms)
- Now, you can only control the board via the pyFirmata package in Python

Note: If your port is denied permission, on new terminal type:

```
$ sudo chmod a+rw /dev/ttyACM0
```

TestLED using pyFirmata

- First, ensure your board is disconnected from your PC. Prepare a LED bulb (any colour will do). Connect it to the Arduino board as follows:



TestLED using pyFirmata

- **Connect back the board to the PC** using the USB cable (at this point you do not require to open the Arduino IDE; you may close it)
- **Open up the python file:** "*testLED.py*" (uploaded in Luminus)
- cd to the correct directory where you put your python files:

```
cd ~/XXXXXXXXX
```

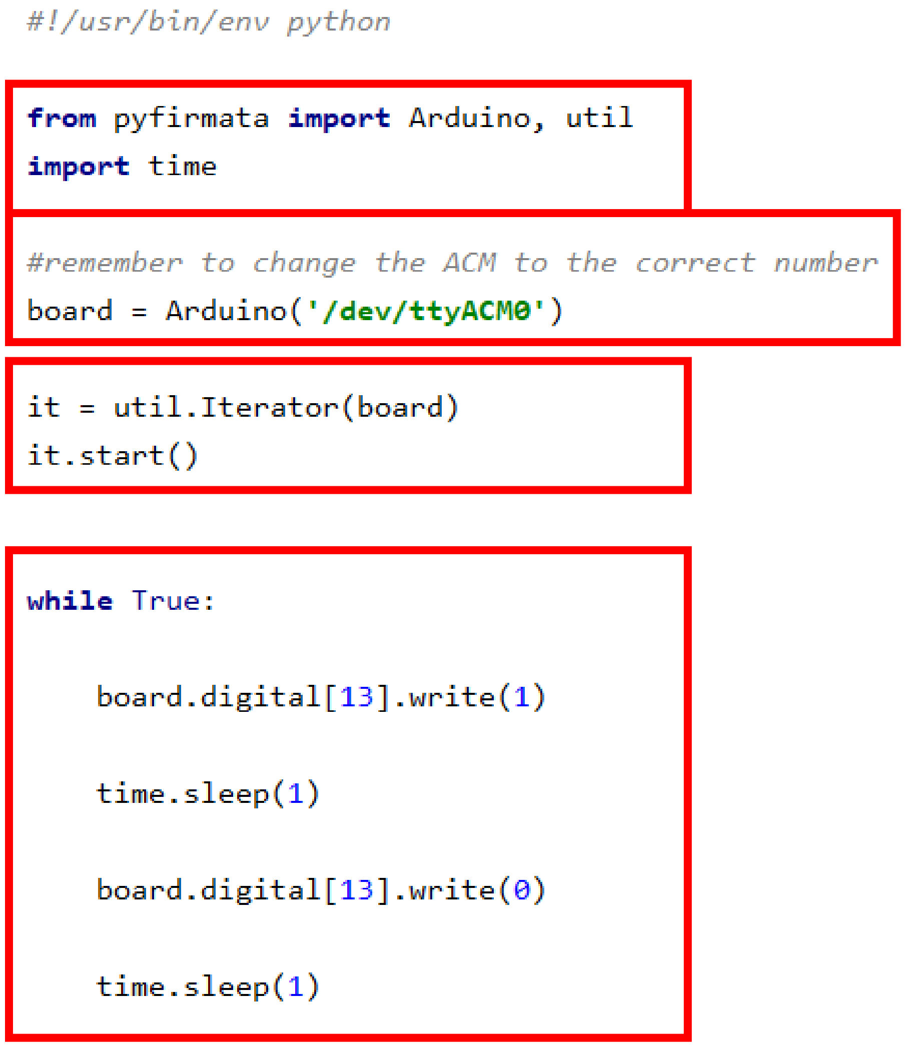
- Run the python file:

```
python testLED.py
```

- You will realize that it is a simple program to blink the LED

TestLED using pyFirmata

testLED.py:



```
#!/usr/bin/env python

from pyfirmata import Arduino, util
import time

#remember to change the ACM to the correct number
board = Arduino('/dev/ttyACM0')

it = util.Iterator(board)
it.start()

while True:

    board.digital[13].write(1)

    time.sleep(1)

    board.digital[13].write(0)

    time.sleep(1)
```

TestLight using pyFirmata

- Plug out the USB cable and **connect the Light sensor to the board**
- **Connect back the board to the PC** using the USB cable (at this point you do not require to open the Arduino IDE; you may close it)
- **Open up the python file:** "*testLight.py*" (uploaded in Luminus)
- **Edit the necessary codes** (i.e. pin no.)
- cd to the correct directory where you put your python files:

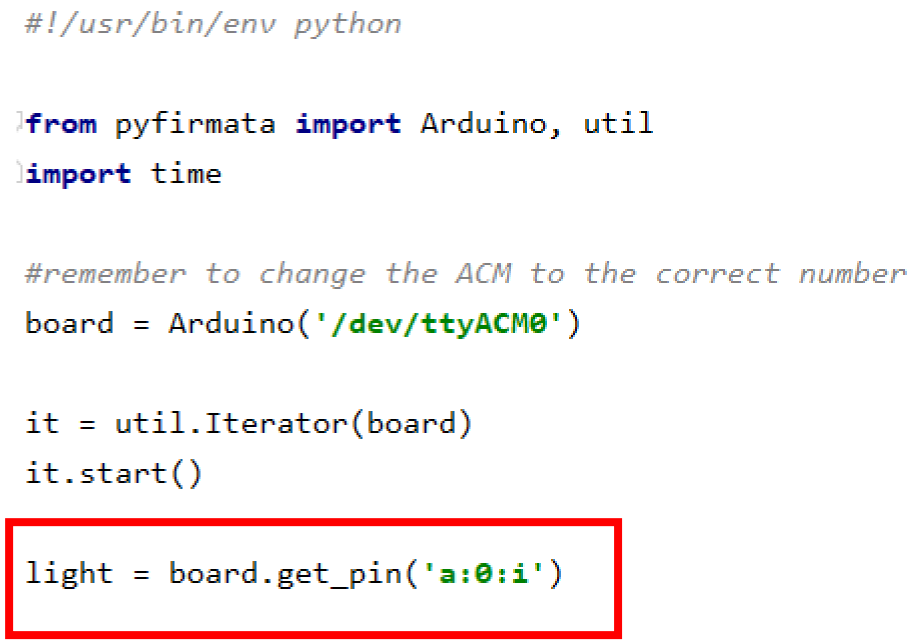
```
cd ~/XXXXXXXXX
```

- Run the python file:

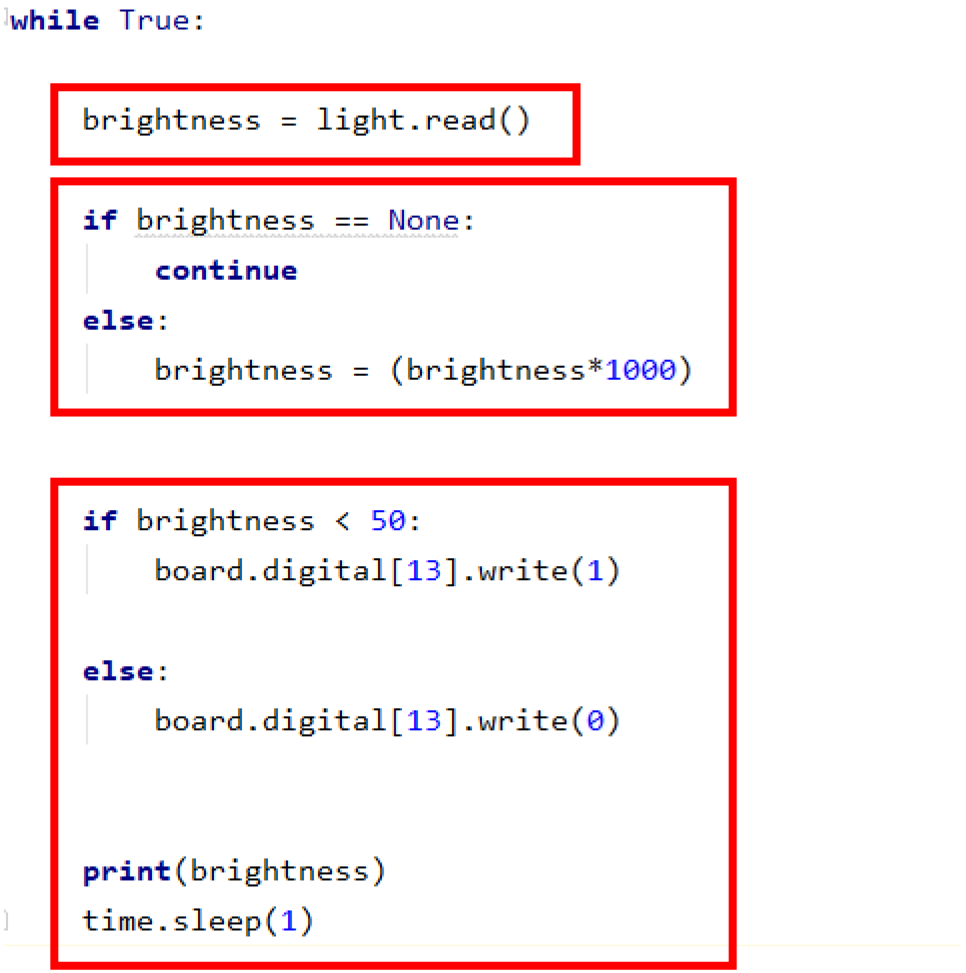
```
python testLight.py
```

TestLight using pyFirmata

testLight.py:



```python
#!/usr/bin/env python

from pyfirmata import Arduino, util
import time

#remember to change the ACM to the correct number
board = Arduino('/dev/ttyACM0')

it = util.Iterator(board)
it.start()

light = board.get_pin('a:0:i')
```



```python
while True:

    brightness = light.read()

    if brightness == None:
        continue
    else:
        brightness = (brightness*1000)


    if brightness < 50:
        board.digital[13].write(1)

    else:
        board.digital[13].write(0)


    print(brightness)
    time.sleep(1)
```

TestTemp using pyFirmata

- Plug out the USB cable and **connect the Gravity LM35 Temperature sensor to the board** (note that this is an Analog sensor)
- **Connect back the board to the PC** using the USB cable (at this point you do not require to open the Arduino IDE; you may close it)
- **Open up the python file:** "*testTemp.py*" (uploaded in Luminus)
- **Edit the necessary codes** (i.e. pin no.)
- cd to the correct directory where you put your python files:

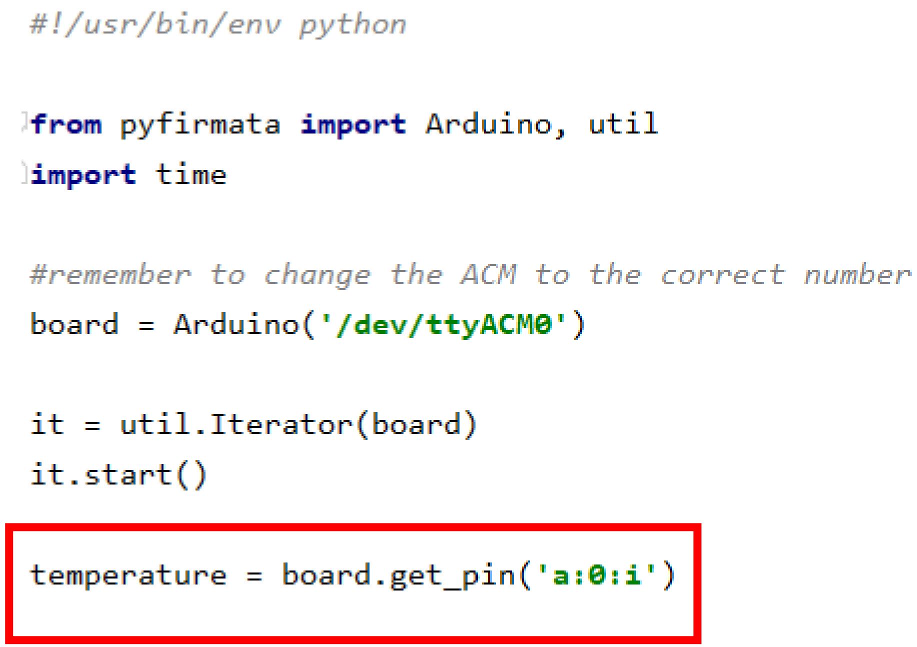
```
cd ~/XXXXXXXXX
```

- Run the python file:

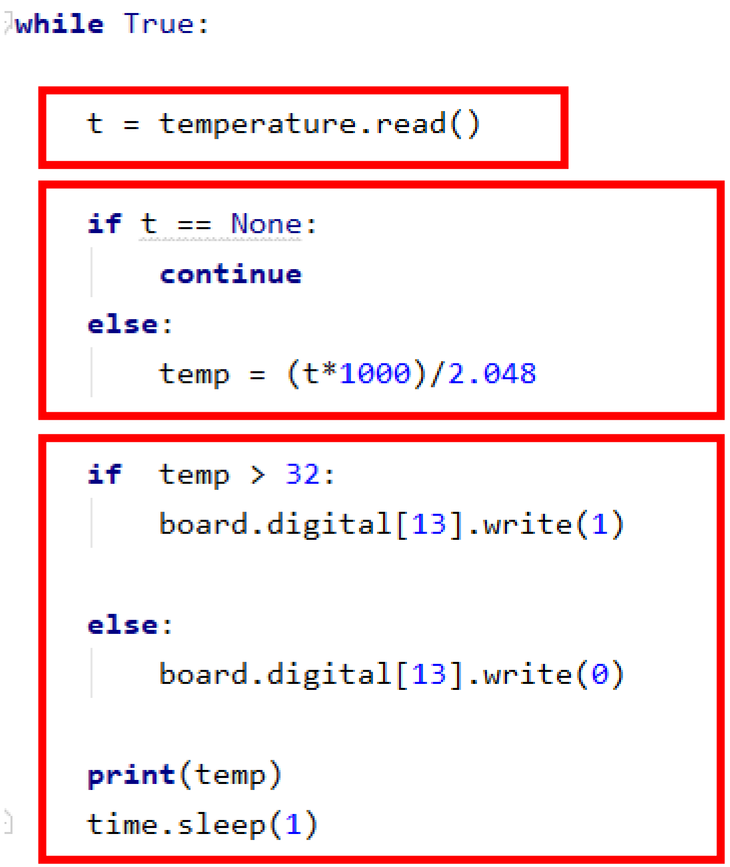
```
python testTemp.py
```

TestTemp using pyFirmata

testTemp.py:



```
#!/usr/bin/env python

from pyfirmata import Arduino, util
import time

#remember to change the ACM to the correct number
board = Arduino('/dev/ttyACM0')

it = util.Iterator(board)
it.start()

temperature = board.get_pin('a:0:i')
```



```
while True:

    t = temperature.read()

    if t == None:
        continue
    else:
        temp = (t*1000)/2.048

    if  temp > 32:
        board.digital[13].write(1)

    else:
        board.digital[13].write(0)

    print(temp)
    time.sleep(1)
```

Practice 1: Integrating Sensors using pyFirmata

Using the previous examples as references, **develop a new python file: “*IntegratingSensors.py*”.**

For this practice, **you will learn how to control desirable outputs based on the sensors’ input.** Connect an LED bulb to the Arduino board as demonstrated in the TestLED example; **this LED bulb will be simulating the output for this practice.**

Your inputs will be utilizing the light, analog temperature and button (Strictly 3.3 Input Voltage for button!) sensors. The required conditions are as follows:

Dark **(brightness level < 25)**	**Hot** **(temperature > 32)**	**Button** **(True/False)**	**LED Status**
Yes AND	Yes AND	True	**Blink**
Yes OR	Yes OR	True	**On**
No	No	False	**Off**

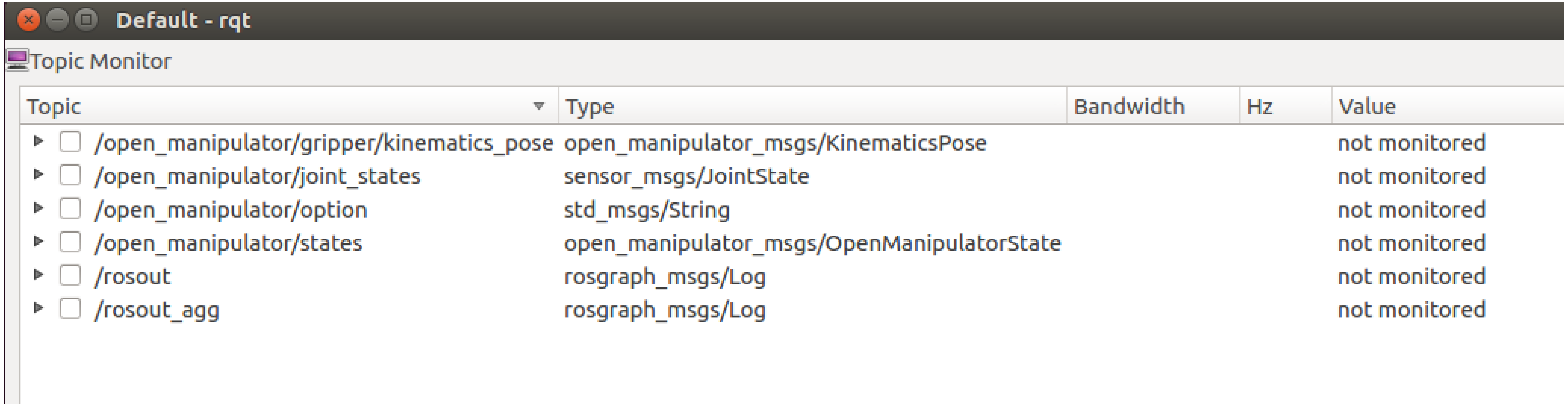
The OM has its own internal sensors

What kind???

How to see/extract these data?

Rqt for ROS

- **Allows users to check the topics of the OM controller**
- That is: it allows users to easily see topic status by displaying all topics on a topic list; can see topic name, type, bandwidth, Hz and value on this platform



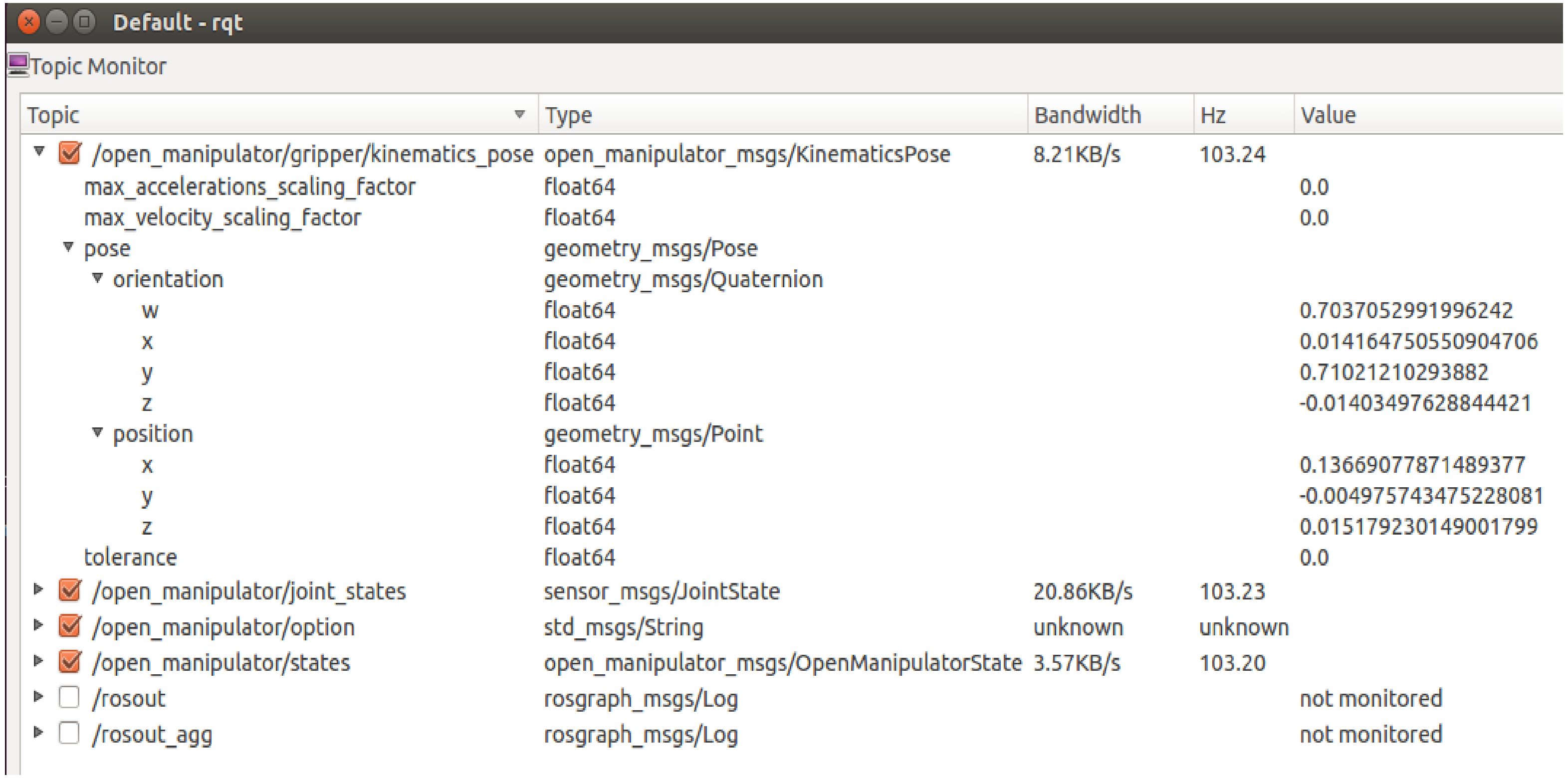
Default - rqt

Topic Monitor

Topic	Type	Bandwidth	Hz	Value
▸ □ /open_manipulator/gripper/kinematics_pose	open_manipulator_msgs/KinematicsPose			not monitored
▸ □ /open_manipulator/joint_states	sensor_msgs/JointState			not monitored
▸ □ /open_manipulator/option	std_msgs/String			not monitored
▸ □ /open_manipulator/states	open_manipulator_msgs/OpenManipulatorState			not monitored
▸ □ /rosout	rosgraph_msgs/Log			not monitored
▸ □ /rosout_agg	rosgraph_msgs/Log			not monitored

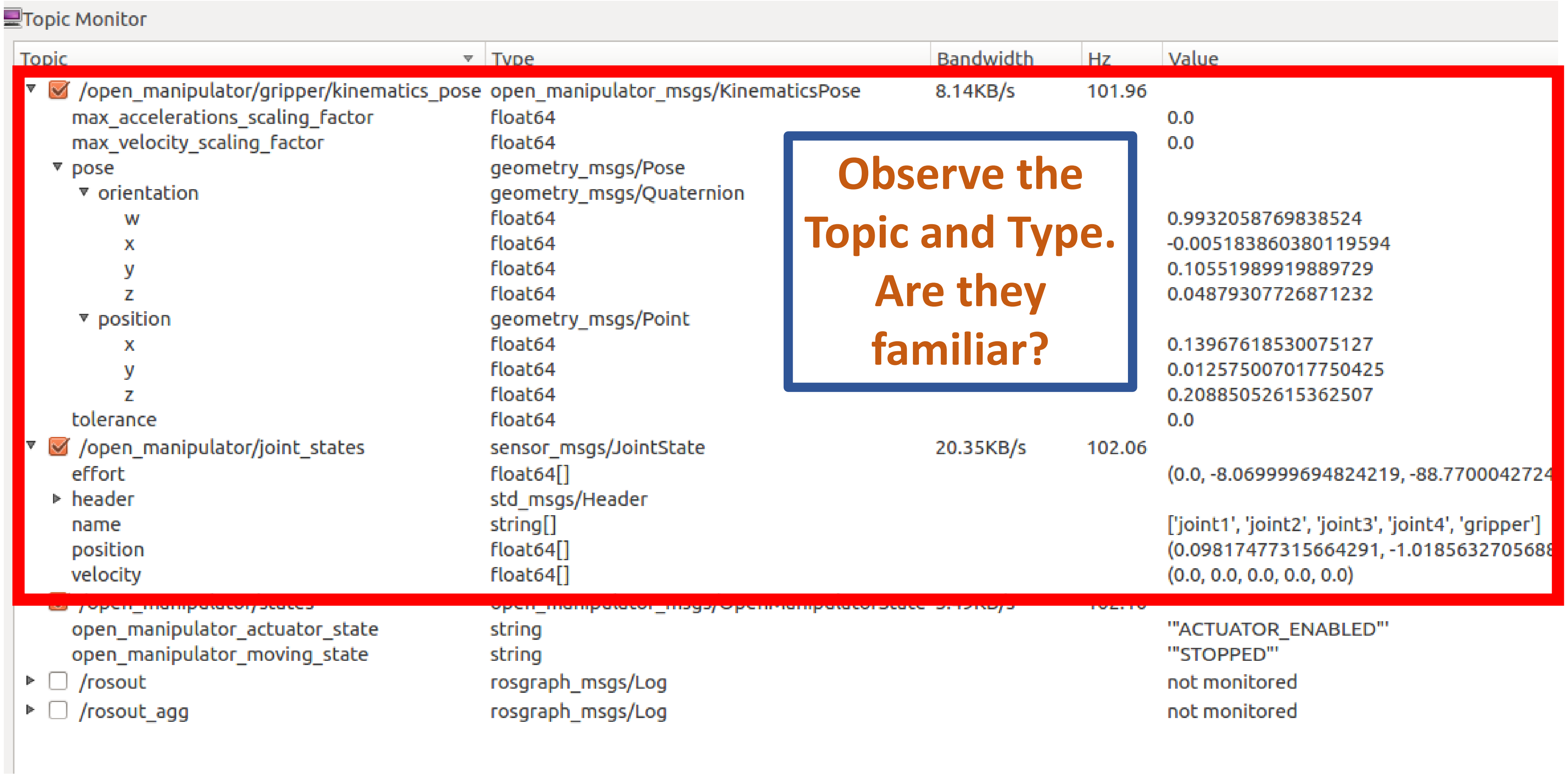
Rqt for ROS

- Topics without a check mark will not be monitored. **To monitor topics, click the checkbox**
- If you would like to **see more details about topic message, click the ▶ button** next to each checkbox



Topic	Type	Bandwidth	Hz	Value
▼ ☑ /open_manipulator/gripper/kinematics_pose	open_manipulator_msgs/KinematicsPose	8.21KB/s	103.24	
max_accelerations_scaling_factor	float64			0.0
max_velocity_scaling_factor	float64			0.0
▼ pose	geometry_msgs/Pose			
▼ orientation	geometry_msgs/Quaternion			
w	float64			0.7037052991996242
x	float64			0.014164750550904706
y	float64			0.71021210293882
z	float64			-0.01403497628844421
▼ position	geometry_msgs/Point			
x	float64			0.13669077871489377
y	float64			-0.004975743475228081
z	float64			0.015179230149001799
tolerance	float64			0.0
▶ ☑ /open_manipulator/joint_states	sensor_msgs/JointState	20.86KB/s	103.23	
▶ ☑ /open_manipulator/option	std_msgs/String	unknown	unknown	
▶ ☑ /open_manipulator/states	open_manipulator_msgs/OpenManipulatorState	3.57KB/s	103.20	
▶ □ /rosout	rosgraph_msgs/Log			not monitored
▶ □ /rosout_agg	rosgraph_msgs/Log			not monitored

Rqt for ROS



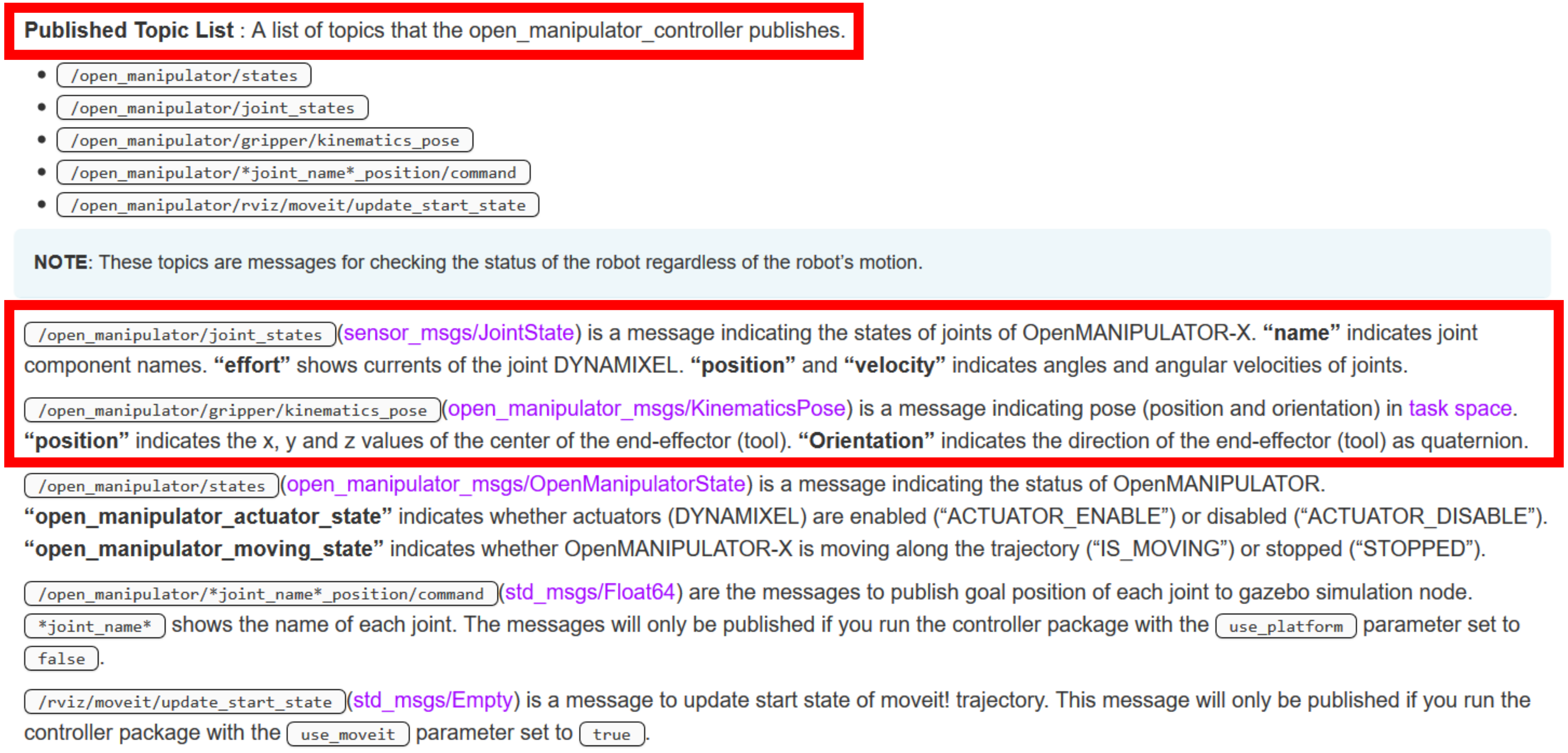
Topic Monitor

Topic	Type	Bandwidth	Hz	Value
▾ ☑ /open_manipulator/gripper/kinematics_pose	open_manipulator_msgs/KinematicsPose	8.14KB/s	101.96	
max_accelerations_scaling_factor	float64			0.0
max_velocity_scaling_factor	float64			0.0
▾ pose	geometry_msgs/Pose			
▾ orientation	geometry_msgs/Quaternion			
w	float64			0.9932058769838524
x	float64			-0.005183860380119594
y	float64			0.10551989919889729
z	float64			0.04879307726871232
▾ position	geometry_msgs/Point			
x	float64			0.13967618530075127
y	float64			0.012575007017750425
z	float64			0.20885052615362507
tolerance	float64			0.0
▾ ☑ /open_manipulator/joint_states	sensor_msgs/JointState	20.35KB/s	102.06	
effort	float64[]			(0.0, -8.069999694824219, -88.7700042724
▸ header	std_msgs/Header			
name	string[]			['joint1', 'joint2', 'joint3', 'joint4', 'gripper']
position	float64[]			(0.09817477315664291, -1.0185632705688
velocity	float64[]			(0.0, 0.0, 0.0, 0.0, 0.0)
☑ /open_manipulator/states	open_manipulator_msgs/OpenManipulatorState	3.19KB/s	102.16	
open_manipulator_actuator_state	string			'"ACTUATOR_ENABLED"'
open_manipulator_moving_state	string			'"STOPPED"'
▸ ☐ /rosout	rosgraph_msgs/Log			not monitored
▸ ☐ /rosout_agg	rosgraph_msgs/Log			not monitored

Observe the Topic and Type. Are they familiar?

Control OM using ROS Software (RECAP)

Control via Python (rospy)



Published Topic List : A list of topics that the open_manipulator_controller publishes.

- `/open_manipulator/states`
- `/open_manipulator/joint_states`
- `/open_manipulator/gripper/kinematics_pose`
- `/open_manipulator/*joint_name*_position/command`
- `/open_manipulator/rviz/moveit/update_start_state`

NOTE: These topics are messages for checking the status of the robot regardless of the robot's motion.

`/open_manipulator/joint_states`(sensor_msgs/JointState) is a message indicating the states of joints of OpenMANIPULATOR-X. **"name"** indicates joint component names. **"effort"** shows currents of the joint DYNAMIXEL. **"position"** and **"velocity"** indicates angles and angular velocities of joints.

`/open_manipulator/gripper/kinematics_pose`(open_manipulator_msgs/KinematicsPose) is a message indicating pose (position and orientation) in task space. **"position"** indicates the x, y and z values of the center of the end-effector (tool). **"Orientation"** indicates the direction of the end-effector (tool) as quaternion.

`/open_manipulator/states`(open_manipulator_msgs/OpenManipulatorState) is a message indicating the status of OpenMANIPULATOR. **"open_manipulator_actuator_state"** indicates whether actuators (DYNAMIXEL) are enabled ("ACTUATOR_ENABLE") or disabled ("ACTUATOR_DISABLE"). **"open_manipulator_moving_state"** indicates whether OpenMANIPULATOR-X is moving along the trajectory ("IS_MOVING") or stopped ("STOPPED").

`/open_manipulator/*joint_name*_position/command`(std_msgs/Float64) are the messages to publish goal position of each joint to gazebo simulation node. `*joint_name*` shows the name of each joint. The messages will only be published if you run the controller package with the `use_platform` parameter set to `false`.

`/rviz/moveit/update_start_state`(std_msgs/Empty) is a message to update start state of moveit! trajectory. This message will only be published if you run the controller package with the `use_moveit` parameter set to `true`.

Controlling OM motor outputs to respond to the various sensors' input

How to configure this???

Recap: Using the python file: "*IntegratingSensors.py*", we now replace the LED output with the motor outputs as learnt in Workshop 1

Reminder to configure the correct port in the OM controller launch node

Note that if you can’t launch your controller successfully, double check if the USB port in the launch file is defined correctly

- From Home, search and open `open_manipulator_controller.launch`
- Ensure that the following line is as such: `<arg name="dynamixel_usb_port" default="/dev/ttyACM1"/>`
- Note that when you plug the Arduino board and the OpenCR1.0 board into the PC at the same time, the port numbers may change (ACM0/ACM1/ACM2); you must reconfigure them if this happens

Example: Motors Response to Various Sensors

- Leave the sensors connected to the board from the previous practice (i.e. light, temperature, button)
- Open python file: “`robotwithsensors1.py`” and ensure that the pin numbers are correct
- Connect OM and PC with the USB extension cable
- Connect OM with the power supply cable
- Activate `roscore` and run the controller launch
- cd to the correct directory where you put your python files:

```
cd ~/XXXXXXXXX
```

Example: Motors Response to Various Sensors

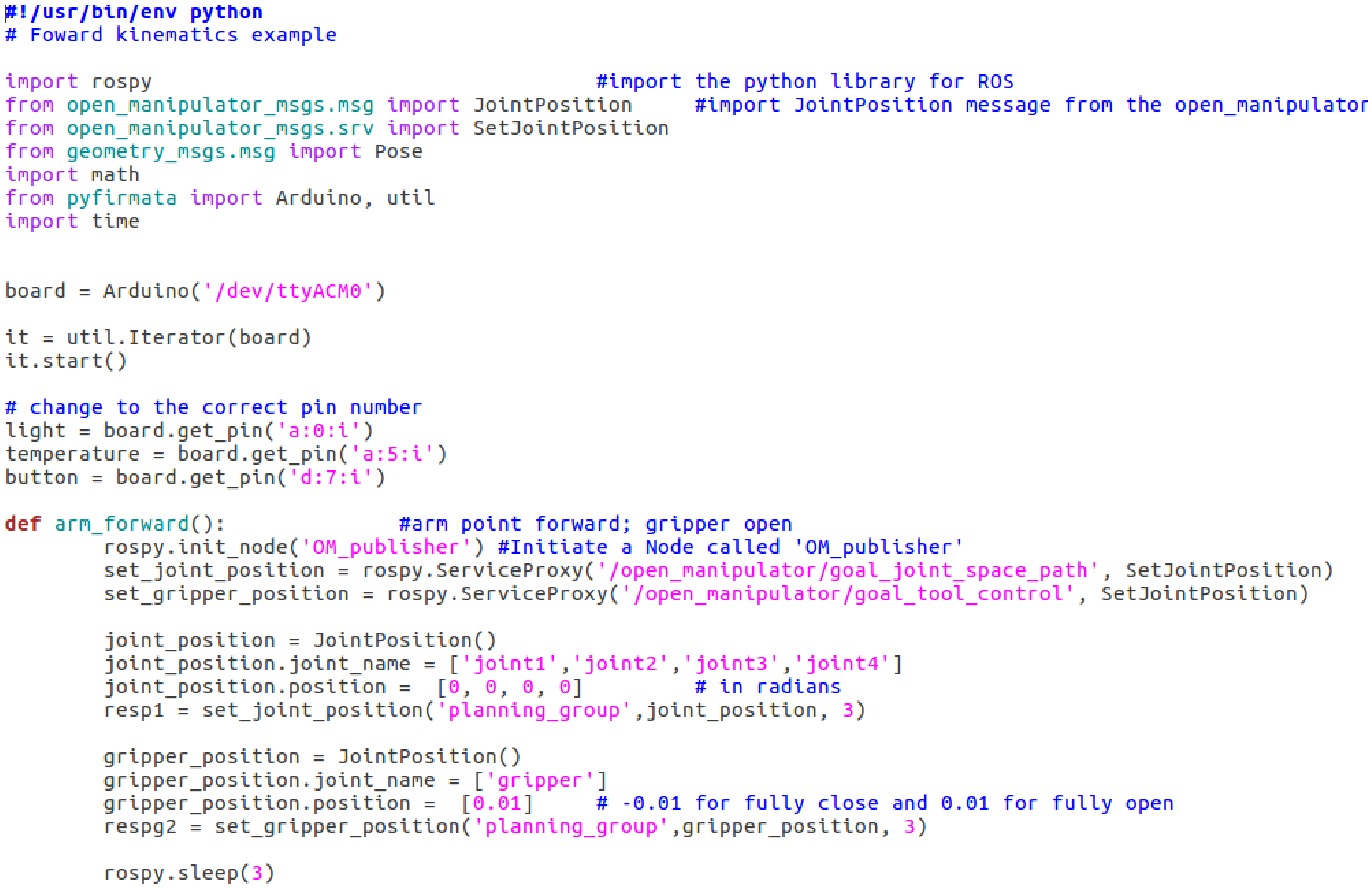
- run the given python file:

```
python robotwithsensors1.py
```

- **Test and observe the robot movements** as you:
  1. Do nothing
  2. Cover the light sensor (dark environment)
  3. Press down the temperature sensor (hot environment)
  4. Press down the button
  5. Do actions No. 2 to 4 together
- Do the same for the given python file:

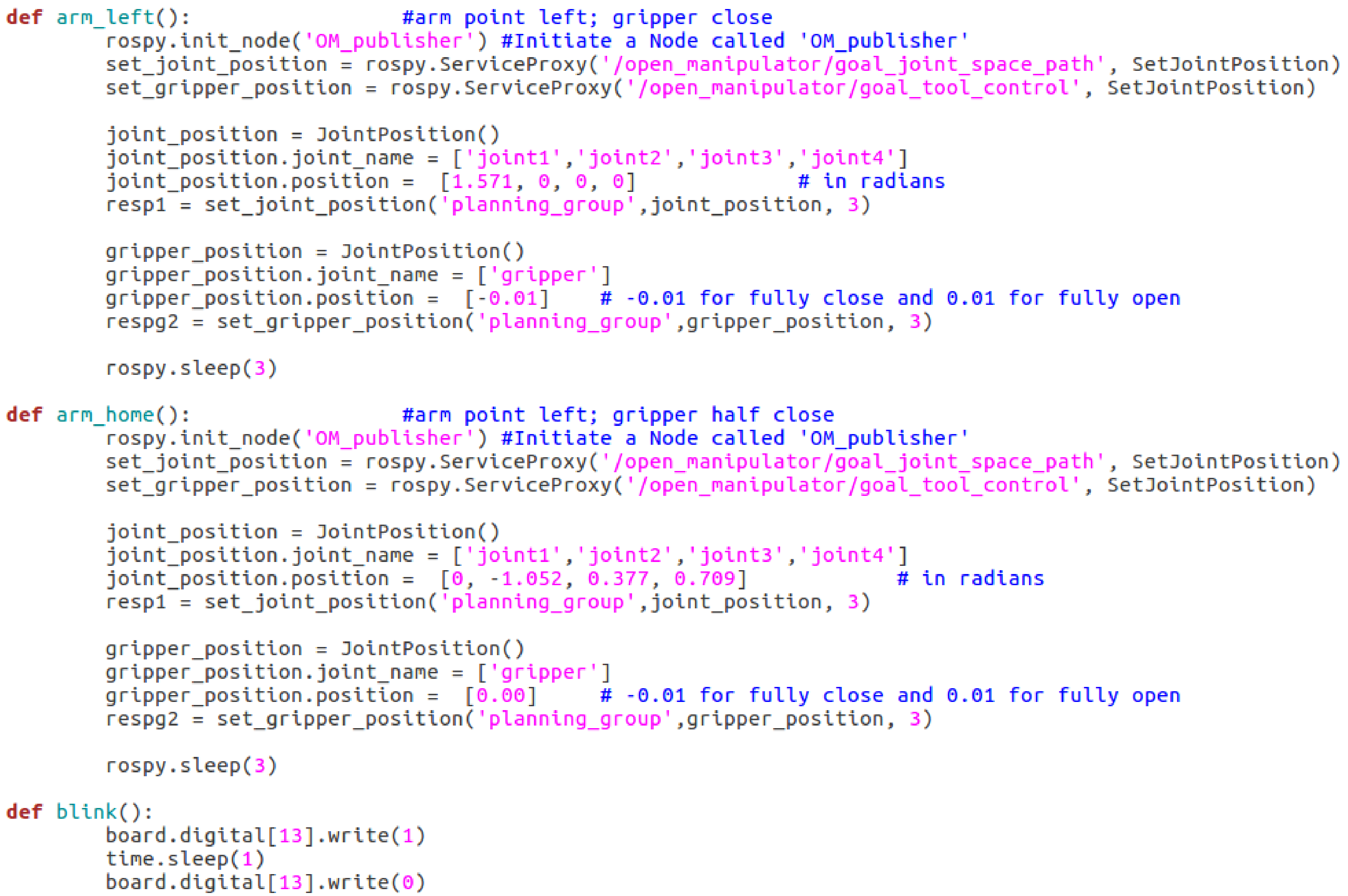
```
python robotwithsensors2.py
```

robotwithsensors1.py:



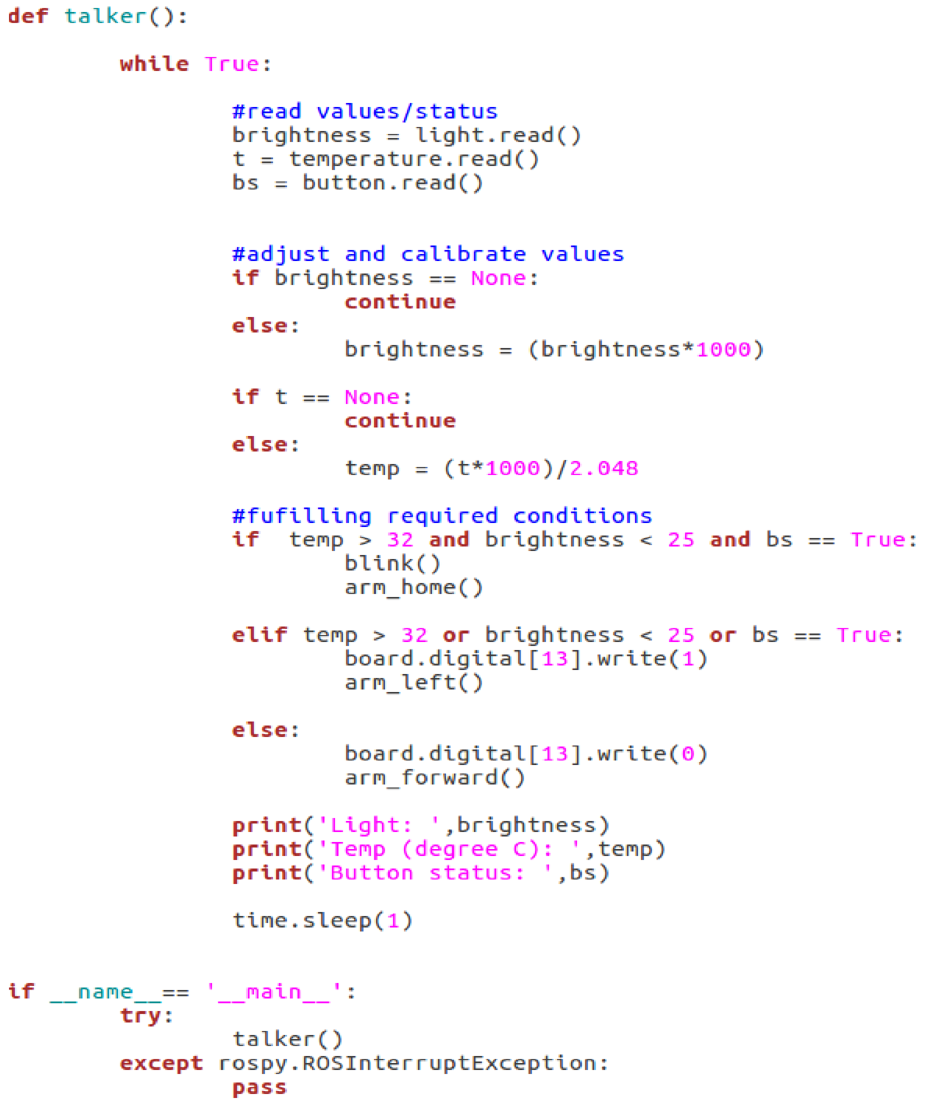
```
#!/usr/bin/env python
# Foward kinematics example

import rospy                                    #import the python library for ROS
from open_manipulator_msgs.msg import JointPosition     #import JointPosition message from the open_manipulator
from open_manipulator_msgs.srv import SetJointPosition
from geometry_msgs.msg import Pose
import math
from pyfirmata import Arduino, util
import time


board = Arduino('/dev/ttyACM0')

it = util.Iterator(board)
it.start()

# change to the correct pin number
light = board.get_pin('a:0:i')
temperature = board.get_pin('a:5:i')
button = board.get_pin('d:7:i')

def arm_forward():              #arm point forward; gripper open
        rospy.init_node('OM_publisher') #Initiate a Node called 'OM_publisher'
        set_joint_position = rospy.ServiceProxy('/open_manipulator/goal_joint_space_path', SetJointPosition)
        set_gripper_position = rospy.ServiceProxy('/open_manipulator/goal_tool_control', SetJointPosition)

        joint_position = JointPosition()
        joint_position.joint_name = ['joint1','joint2','joint3','joint4']
        joint_position.position =  [0, 0, 0, 0]         # in radians
        resp1 = set_joint_position('planning_group',joint_position, 3)

        gripper_position = JointPosition()
        gripper_position.joint_name = ['gripper']
        gripper_position.position =  [0.01]     # -0.01 for fully close and 0.01 for fully open
        respg2 = set_gripper_position('planning_group',gripper_position, 3)

        rospy.sleep(3)
```

robotwithsensors1.py (cont):



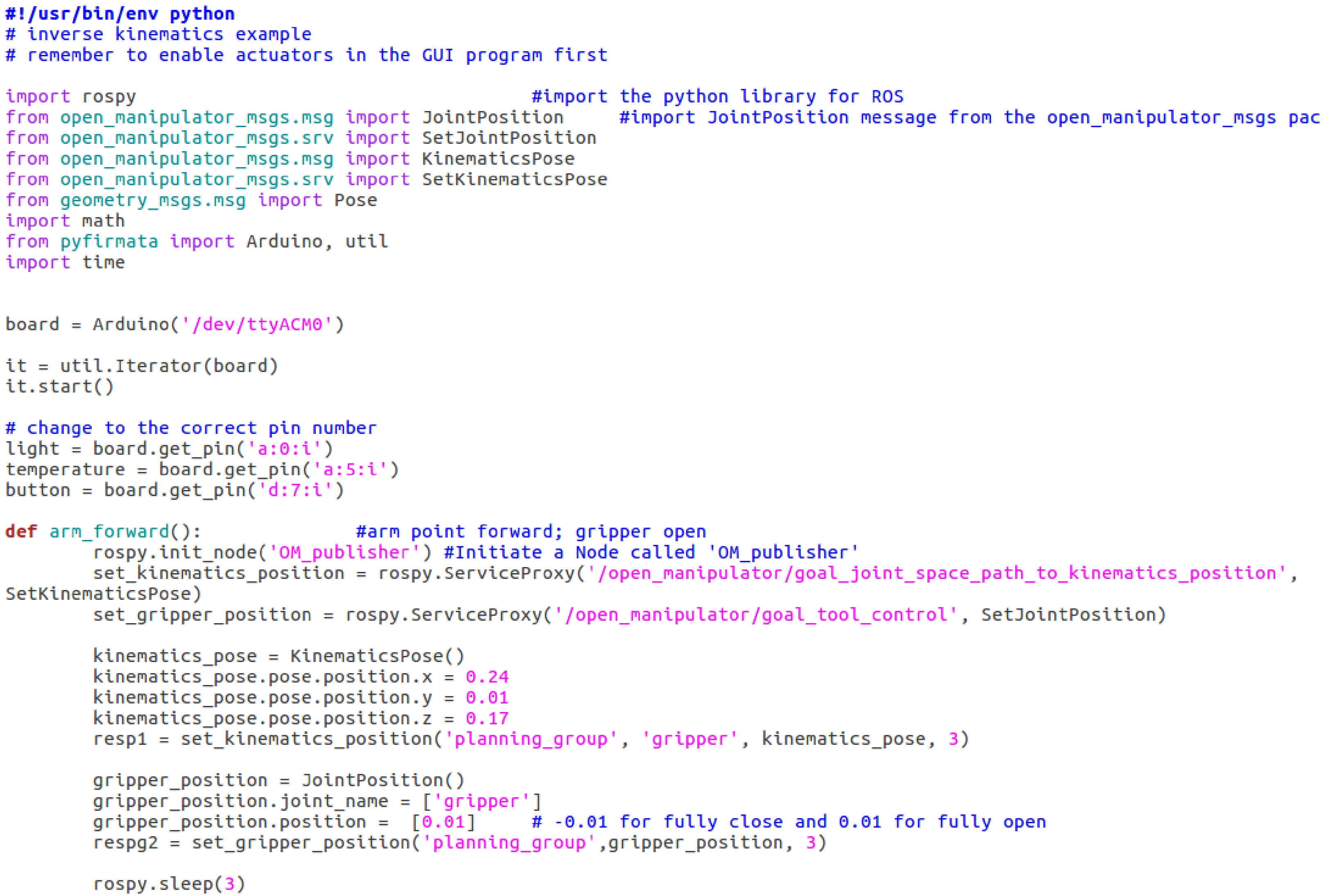
```python
def arm_left():                 #arm point left; gripper close
        rospy.init_node('OM_publisher') #Initiate a Node called 'OM_publisher'
        set_joint_position = rospy.ServiceProxy('/open_manipulator/goal_joint_space_path', SetJointPosition)
        set_gripper_position = rospy.ServiceProxy('/open_manipulator/goal_tool_control', SetJointPosition)

        joint_position = JointPosition()
        joint_position.joint_name = ['joint1','joint2','joint3','joint4']
        joint_position.position =  [1.571, 0, 0, 0]             # in radians
        resp1 = set_joint_position('planning_group',joint_position, 3)

        gripper_position = JointPosition()
        gripper_position.joint_name = ['gripper']
        gripper_position.position =  [-0.01]    # -0.01 for fully close and 0.01 for fully open
        respg2 = set_gripper_position('planning_group',gripper_position, 3)

        rospy.sleep(3)

def arm_home():                 #arm point left; gripper half close
        rospy.init_node('OM_publisher') #Initiate a Node called 'OM_publisher'
        set_joint_position = rospy.ServiceProxy('/open_manipulator/goal_joint_space_path', SetJointPosition)
        set_gripper_position = rospy.ServiceProxy('/open_manipulator/goal_tool_control', SetJointPosition)

        joint_position = JointPosition()
        joint_position.joint_name = ['joint1','joint2','joint3','joint4']
        joint_position.position =  [0, -1.052, 0.377, 0.709]            # in radians
        resp1 = set_joint_position('planning_group',joint_position, 3)

        gripper_position = JointPosition()
        gripper_position.joint_name = ['gripper']
        gripper_position.position =  [0.00]     # -0.01 for fully close and 0.01 for fully open
        respg2 = set_gripper_position('planning_group',gripper_position, 3)

        rospy.sleep(3)

def blink():
        board.digital[13].write(1)
        time.sleep(1)
        board.digital[13].write(0)
```

robotwithsensors1.py (cont):



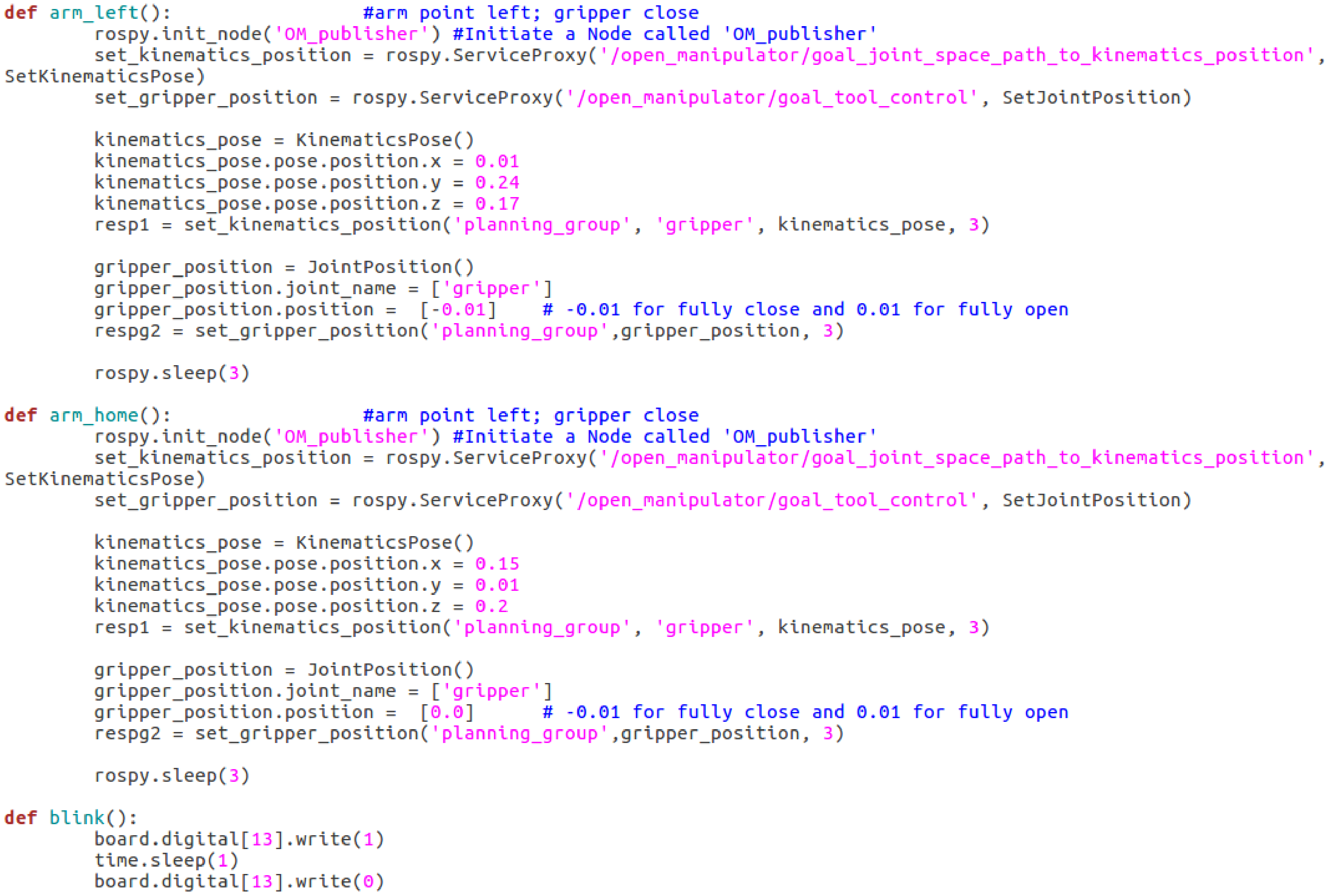
```
def talker():

        while True:

                #read values/status
                brightness = light.read()
                t = temperature.read()
                bs = button.read()


                #adjust and calibrate values
                if brightness == None:
                        continue
                else:
                        brightness = (brightness*1000)

                if t == None:
                        continue
                else:
                        temp = (t*1000)/2.048

                #fufilling required conditions
                if  temp > 32 and brightness < 25 and bs == True:
                        blink()
                        arm_home()

                elif temp > 32 or brightness < 25 or bs == True:
                        board.digital[13].write(1)
                        arm_left()

                else:
                        board.digital[13].write(0)
                        arm_forward()

                print('Light: ',brightness)
                print('Temp (degree C): ',temp)
                print('Button status: ',bs)

                time.sleep(1)


if __name__== '__main__':
        try:
                talker()
        except rospy.ROSInterruptException:
                pass
```

robotwithsensors2.py:



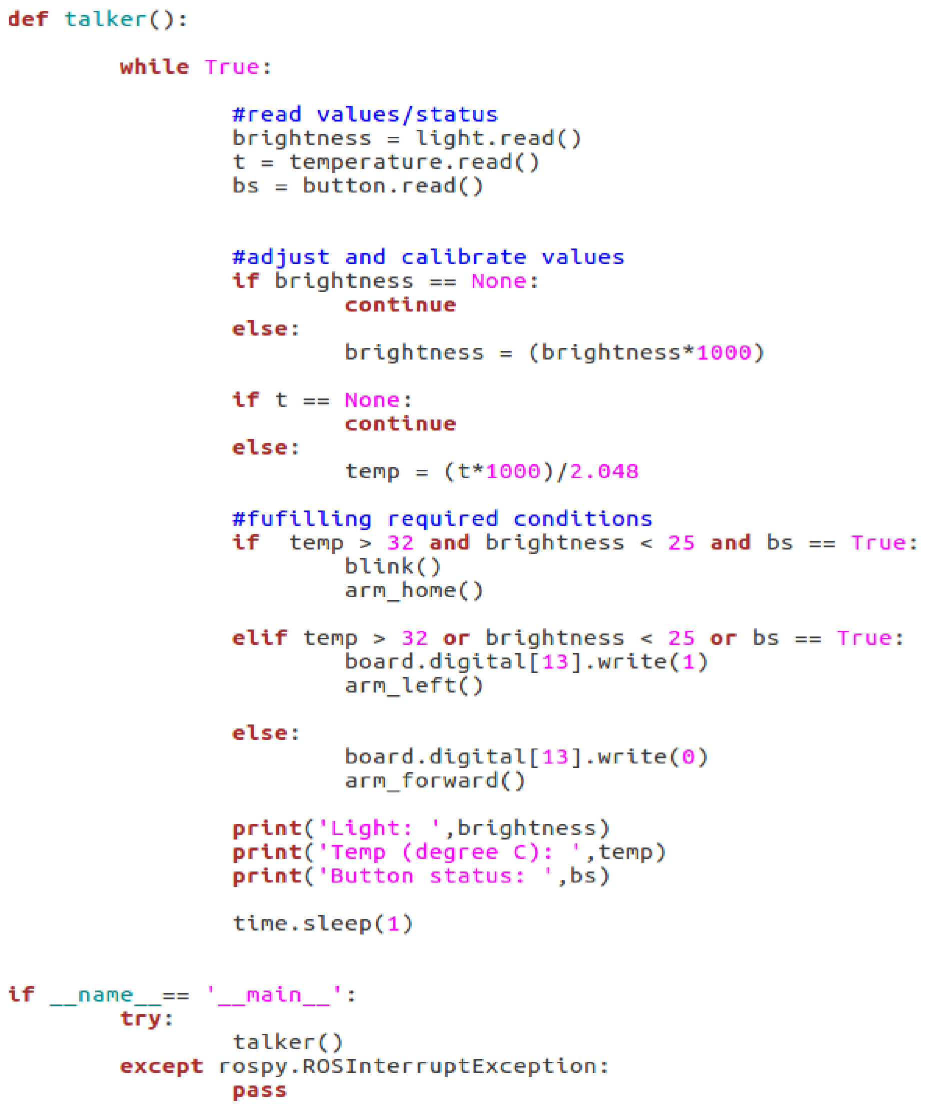
```python
#!/usr/bin/env python
# inverse kinematics example
# remember to enable actuators in the GUI program first

import rospy                                    #import the python library for ROS
from open_manipulator_msgs.msg import JointPosition     #import JointPosition message from the open_manipulator_msgs pac
from open_manipulator_msgs.srv import SetJointPosition
from open_manipulator_msgs.msg import KinematicsPose
from open_manipulator_msgs.srv import SetKinematicsPose
from geometry_msgs.msg import Pose
import math
from pyfirmata import Arduino, util
import time


board = Arduino('/dev/ttyACM0')

it = util.Iterator(board)
it.start()

# change to the correct pin number
light = board.get_pin('a:0:i')
temperature = board.get_pin('a:5:i')
button = board.get_pin('d:7:i')

def arm_forward():              #arm point forward; gripper open
        rospy.init_node('OM_publisher') #Initiate a Node called 'OM_publisher'
        set_kinematics_position = rospy.ServiceProxy('/open_manipulator/goal_joint_space_path_to_kinematics_position',
SetKinematicsPose)
        set_gripper_position = rospy.ServiceProxy('/open_manipulator/goal_tool_control', SetJointPosition)

        kinematics_pose = KinematicsPose()
        kinematics_pose.pose.position.x = 0.24
        kinematics_pose.pose.position.y = 0.01
        kinematics_pose.pose.position.z = 0.17
        resp1 = set_kinematics_position('planning_group', 'gripper', kinematics_pose, 3)

        gripper_position = JointPosition()
        gripper_position.joint_name = ['gripper']
        gripper_position.position =  [0.01]     # -0.01 for fully close and 0.01 for fully open
        respg2 = set_gripper_position('planning_group',gripper_position, 3)

        rospy.sleep(3)
```

robotwithsensors2.py (cont):



```
def arm_left():                     #arm point left; gripper close
        rospy.init_node('OM_publisher') #Initiate a Node called 'OM_publisher'
        set_kinematics_position = rospy.ServiceProxy('/open_manipulator/goal_joint_space_path_to_kinematics_position',
SetKinematicsPose)
        set_gripper_position = rospy.ServiceProxy('/open_manipulator/goal_tool_control', SetJointPosition)

        kinematics_pose = KinematicsPose()
        kinematics_pose.pose.position.x = 0.01
        kinematics_pose.pose.position.y = 0.24
        kinematics_pose.pose.position.z = 0.17
        resp1 = set_kinematics_position('planning_group', 'gripper', kinematics_pose, 3)

        gripper_position = JointPosition()
        gripper_position.joint_name = ['gripper']
        gripper_position.position =  [-0.01]    # -0.01 for fully close and 0.01 for fully open
        respg2 = set_gripper_position('planning_group',gripper_position, 3)

        rospy.sleep(3)

def arm_home():                     #arm point left; gripper close
        rospy.init_node('OM_publisher') #Initiate a Node called 'OM_publisher'
        set_kinematics_position = rospy.ServiceProxy('/open_manipulator/goal_joint_space_path_to_kinematics_position',
SetKinematicsPose)
        set_gripper_position = rospy.ServiceProxy('/open_manipulator/goal_tool_control', SetJointPosition)

        kinematics_pose = KinematicsPose()
        kinematics_pose.pose.position.x = 0.15
        kinematics_pose.pose.position.y = 0.01
        kinematics_pose.pose.position.z = 0.2
        resp1 = set_kinematics_position('planning_group', 'gripper', kinematics_pose, 3)

        gripper_position = JointPosition()
        gripper_position.joint_name = ['gripper']
        gripper_position.position =  [0.0]      # -0.01 for fully close and 0.01 for fully open
        respg2 = set_gripper_position('planning_group',gripper_position, 3)

        rospy.sleep(3)

def blink():
        board.digital[13].write(1)
        time.sleep(1)
        board.digital[13].write(0)
```

robotwithsensors2.py (cont):



```python
def talker():

    while True:

        #read values/status
        brightness = light.read()
        t = temperature.read()
        bs = button.read()


        #adjust and calibrate values
        if brightness == None:
                continue
        else:
                brightness = (brightness*1000)

        if t == None:
                continue
        else:
                temp = (t*1000)/2.048

        #fufilling required conditions
        if  temp > 32 and brightness < 25 and bs == True:
                blink()
                arm_home()

        elif temp > 32 or brightness < 25 or bs == True:
                board.digital[13].write(1)
                arm_left()

        else:
                board.digital[13].write(0)
                arm_forward()

        print('Light: ',brightness)
        print('Temp (degree C): ',temp)
        print('Button status: ',bs)

        time.sleep(1)


if __name__== '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

Chapter 4: Group Work

Group Project 2: Robotic Sensory System

Remember Group Project 1: "*Pick up and Place the item*"? Now for this project, you have **to modify Project's 1 ROS codes so that it can control the robotic arm's motor outputs based on specific requirements**. **The initial and final positions of the item remains the same.**

The objective is to configure the robotic arm and 2 other actuators (i.e. LED, Buzzer) to respond to 3 sensors (i.e. Light, Analog temperature, Button; Strictly 3.3 Input Voltage for button!). You are **only given the following information:**

1. Imagine the OM is operating in a factory and it is **required to repeatedly move items of the same type from the initial to the final position** as depicted in Project 1
2. **The OM cannot be interrupted between each step**; it has to complete the performing step first before it can respond to the sensors' data
3. The OM's actions **have to fulfil the required conditions** (summarized on next few slides):

Required Conditions:

A. When it gets too dark (Brightness < 25), the OM will stop operations and the LED bulb will light up

B. When it gets too hot (Temperature > 32), the OM will stop operations and the buzzer will sound out

C. When the item is detected as defective (assume that this happens when button = True AND the OM is currently holding on an item = True), the OM will throw the item that it is holding away. **Assume that the garbage location to throw the item is at 45 degrees angle to the right side of the OM (refer to illustration).** After throwing the item away, the OM will restart its operations. Hint: Extract the real-time values of the gripper from the internal sensor of the OM to know whether it is holding on an item

D. When Conditions A, B and C are ALL fulfilled, the OM will stop operations while the LED bulb will blink and the buzzer will be on at the same time. After which the OM will throw the item and restart its operations

E. When Conditions A and B are fulfilled, the OM will stop operations while the LED bulb will blink and the buzzer will be on at the same time

F. When Conditions A and C are fulfilled, the OM will stop operations while the LED bulb will light up. After which the OM will throw the item and restart its operations

G. When Conditions B and C are fulfilled, the OM will stop operations while the buzzer will sound out. After which the OM will throw the item and restart its operations

H. When all conditions are NOT fulfilled, the LED bulb and the buzzer will be off, and the OM will continue its operations as per normal

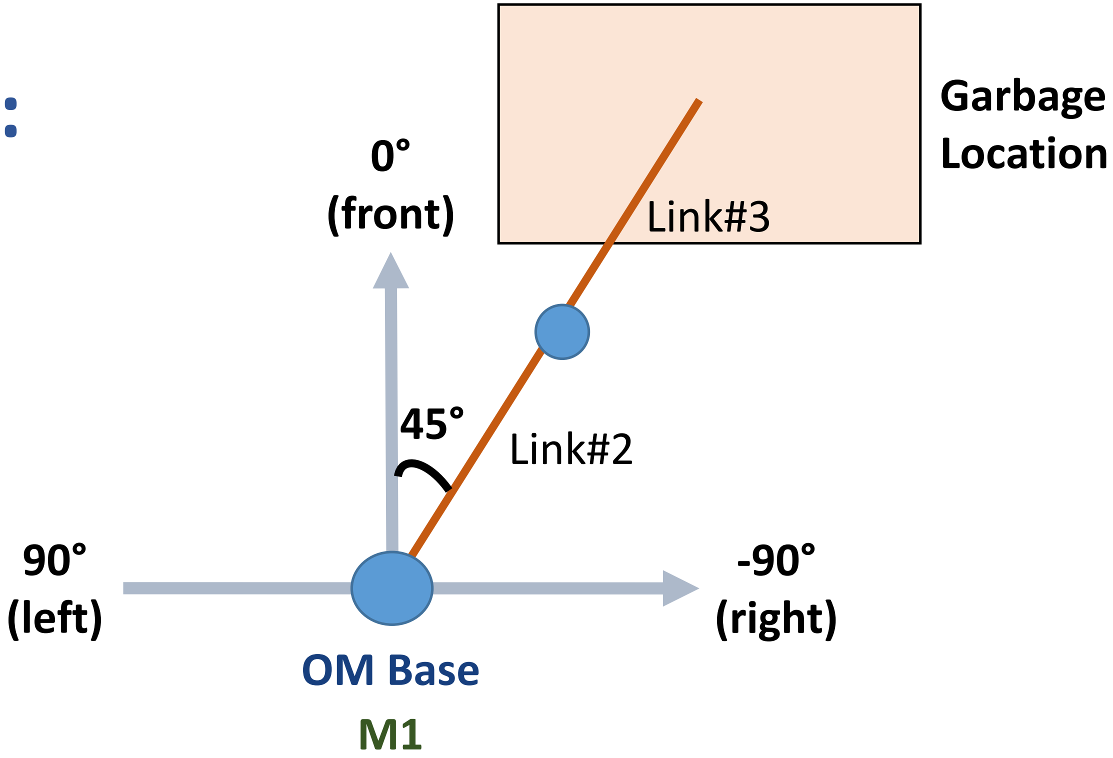
Summary of Required Conditions:

Sensors			Actuators		
Dark (brightness < 25)	Hot (temp > 32)	Defective (True/False)	LED Status	Buzzer	OM
Yes AND	Yes AND	True	**Blink & Buzz same time**		**STOP operations while blinkbuzz, throw garbage and RESTART operations**
Yes AND	Yes	False	**Blink & Buzz same time**		**STOP operations**
No	Yes AND	True	**OFF**	**ON**	**STOP operations while buzz, throw garbage and RESTART operations**
Yes AND	No	True	**ON**	**OFF**	**STOP operations while LED ON, throw garbage and RESTART operations**
Yes	No	False	**ON**	**OFF**	**STOP operations**
No	Yes	False	**OFF**	**ON**	**STOP operations**
No	No	True	**OFF**	**OFF**	**Throw garbage and RESTART operations**
No	No	False	**OFF**	**OFF**	**CONTINUE operations**

Required Conditions:

Garbage location to throw the item:

Top View:



Please upload all your codes with your names as part of the title in Luminus (i.e. Group Project 1 & 2)

Chapter 6: Summary & Conclusion

Summary & Conclusion

- **Integrating additional sensors/actuators** into your robotic system **enables more automatic, intelligent responses** based on various environment or other conditions
- **Understanding and testing your sensors is important** (e.g. the sensor types; analog or digital, how to connect your sensors to the board, quick check to test if the sensor is faulty)
- **pyFirmata allows you to control the Arduino board using python codes;** this allows you to utilize Arduino functions when using ROS to control a robot
- It gets **more challenging to fuse sensors' data as the number of sensors increases**

Thank you!

Questions?

Reflections Time!