Week 3 Lab 1: *Getting started with your Embedded System*

Introduction

During your lab session today, you will work through this sheet. Don't worry if you don't get it all done during the session (especially during this first one). There are optional tasks or extensions for those who want to explore further. Optional tasks are not graded and are just for your practice.

Each lab has a small report be written up and submitted. There are questions that are highlighted in red that you will have to answer in your lab report. Your lab report is due two days after your lab session. For example: if your lab is on a Monday, you must submit your soft copy lab report on Canvas before 23:59:59 on the Wednesday. Lab reports should use the Lab Report Template and not be longer than two pages.

You should have received: *a box of 37 Sensors, 1x WeMOS D1 R2, 1x I2C A2D, 1x I2C IMU, 1x USB Cable, 2x breadboards, and some jumper cables*.

Your tasks for today are the following:

T1. Getting Started with Arduino IDE
T2. Driving a Simple Actuator (an LED)
T3. Reading a Simple Sensor (a button)
T4. Microcontroller… Interrupted.
T5. Is it hot in here?
T6. Exploring your box of sensors

T1. Getting Started with Arduino IDE

The Arduino IDE is a free and open-source integrated development environment (IDE). It is one of the possible options for programming our WeMOS device, and it is the one we will use during our lab sessions. It is a simple and straightforward IDE which will let us get going with our hardware quickly.

The programming language we will use is a subset of the language c/c++, which contains an additional set of Arduino specific calls (https://www.arduino.cc/en/Reference/HomePage) and has a limited set of the full c/c++ libraries. Fear not though, as there is a lot of support and libraries for your final project!

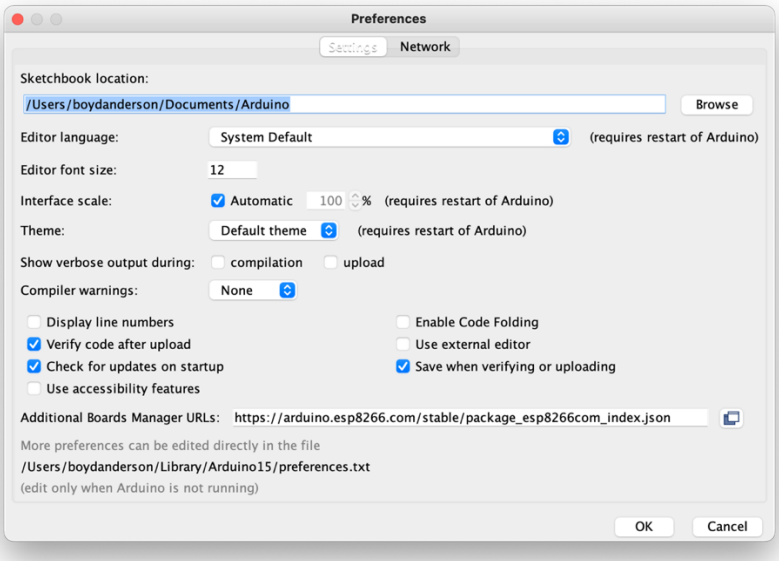
1. First download and install the latest version of Arduino IDE (https://www.arduino.cc/en/software) [For Windows] Do not use the App Store

2. Connect your WeMos to your laptop via USB cable

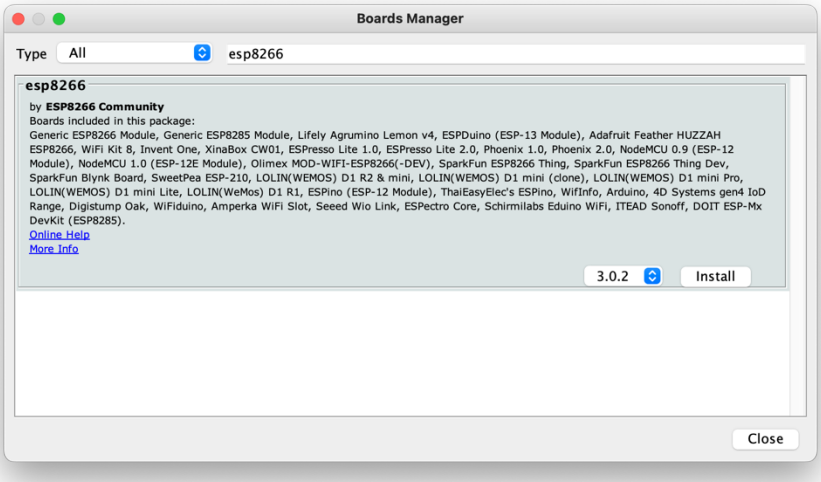
3. Start Arduino IDE on your laptop

We now need to get the board definitions for our WeMOS board.

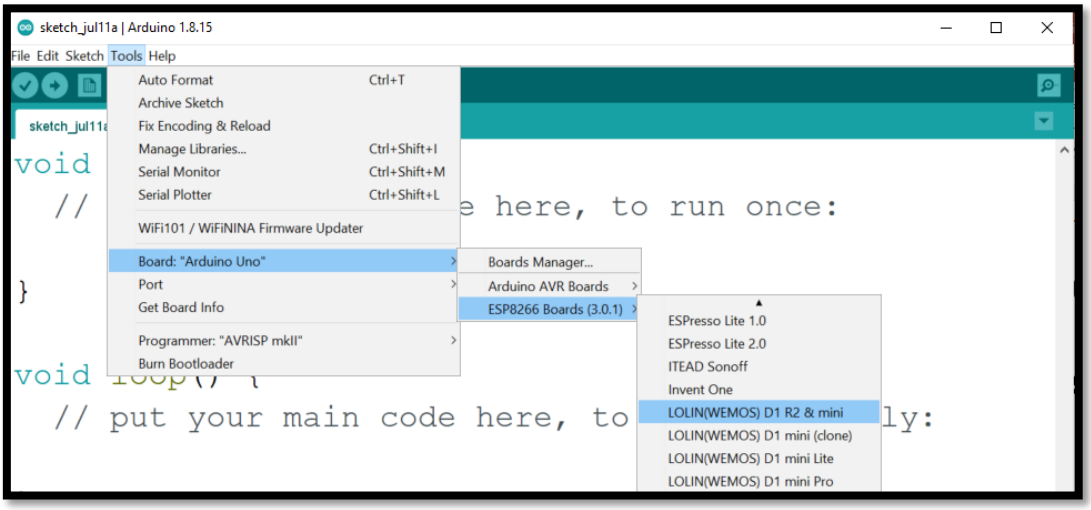
4. Click "File➔Preferences" (on Windows) or "Arduino➔Preferences" (on MacOS X) and paste https://arduino.esp8266.com/stable/package_esp8266com_index.json to Additional Boards Manager URLs and click **Ok** as pictured on the next page.



5. Click "Tools ➔ Board ➔ Boards Manager" and search and install **esp8266** boards



6. Now let's do a simple test to check whether you need to install additional drivers. Click "Tools➔Board➔ESP8266 Boards" and select "LOLIN (WEMOS) D1 R2 & mini"



7. Click “Tools➔Port” and select “COM***x***” (for Windows) or “/dev/cu.usbSerial**x**” (for MacOS X). If you don’t see anything under “Tools➔Port” you might need to install a “USB to Serial” driver. I have included them on Canvas. For Windows, install the given “CH341SER_WIN_3.5.zip”. MacOS X /Linux should have drivers already.

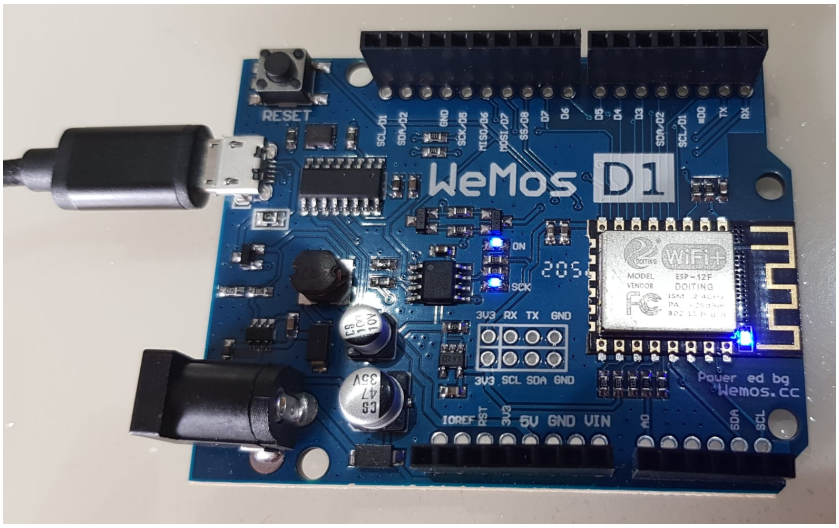
8. After the driver installation, replug the WeMos and try step 7 again

9. We can now run a sample program. Select "File→Examples→01.Basics→Blink". It will open a new “Sketch” (the Arduino terminology for a program). Have a scroll through and see what it is doing.

   Every Arduino Sketch minimally needs to have two main functions, the setup() and loop() functions. The setup function runs once when the device is powered on, and then the loop function runs over and over again. Additional functions and variables can be declared.

   We can set the mode of the GPIO pins using the function pinMode(pin, mode), where the pin is the pin number and mode is either INPUT or OUTPUT . We can write a digital state to the pin using digitalWrite(pin, value) where the value is either 0 or 1.

10. Let’s run the program now. You can compile the sketch using the tick button in the top left-hand corner and upload the program using the arrow button (it also compiles). Try uploading the program to the WeMOS.
    Behold: The LED on WeMos should start blinking!



Ok great! You have run your first program on the WeMOS (possibly?). Lets

Lab Report Task 1: Change the Blink program to blink with ever increasing frequency. Start with four second blinks, then two second blinks, and so on, until it is permanently on. Describe your changes briefly in your lab report.

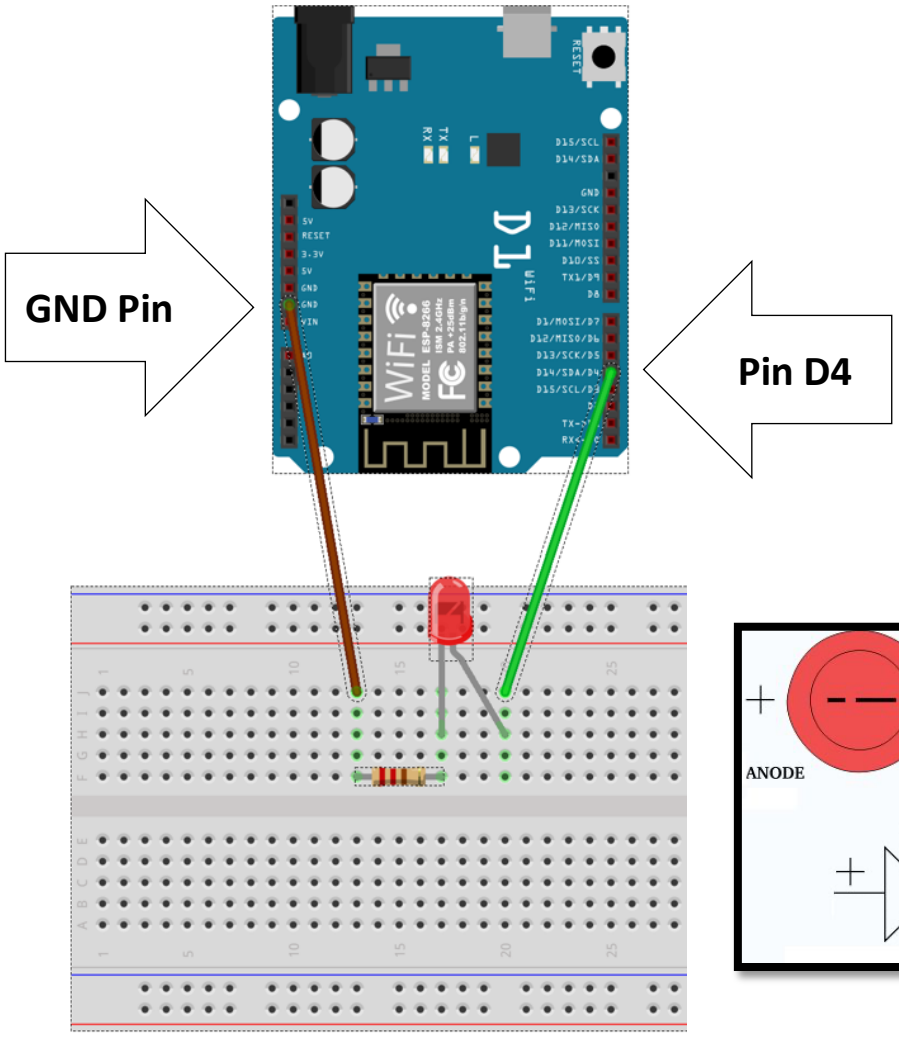
Optional Task: Can you make your WeMOS blink **SOS (** ■■■ ▬▬ ▬▬ ▬▬ ■■■**)**?

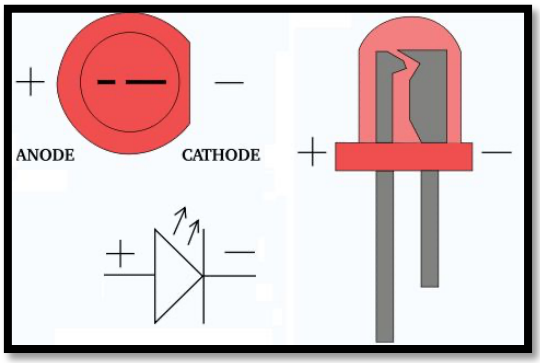
T2. Driving a Simple Actuator (an LED)

Have a look at the figure of the WeMOS on Lecture 2 Slide 15. Notice that the pin **D4** also says “builtin LED” right next to it. Some pins have multiple purposes, and this pin D4 drives both the LED and the output pin.

Let’s test this, try replacing **LED_BUILTIN** in your program with **D4**, and uploading it again. Does it still blink?

Now let’s wire up our first circuit. We are going to drive an external LED using D4. Follow the wire diagram below. Make sure that the LED is the correct way around!





Now plug your WeMOS back in. Do both LEDs blink?

Lab Report Task 2: Change the Blink program and wiring to make your LEDs blink on and off independently. Hint: you might need to use different pins. Describe your changes briefly in your lab report.

In Lecture 2, we introduced the idea of Pulse Width Modulation (PWM) which would allow us to simulate an "analog" signal using our digital pins by changing the duty cycle (turning them on and off rapidly). Refer to the WeMOS figure on the Lecture 2 Slide 15 and note that all the pins say PWM next to them. All your pins on the WeMOS support PWM.

We can use an Arduino function analogWrite(pin, value) to set the output state of the pin, where value is a number between 0 and 255. Where 0 is low and 255 is high. Create a new sketch and try out the following program.

```
#define LED D4

int Brightness = 20;

void setup() {
  pinMode(LED, OUTPUT);
  analogWrite(LED, Brightness);
}

void loop() {
  //Empty for this example
}
```

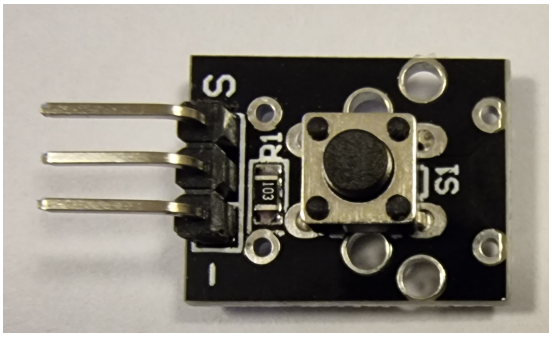
Try modifying the **Brightness** variable value to see the change in LED brightness.

Lab Report Task 3: Modify this program to make your LEDs pulse on and off. Increasing in brightness and then deceasing. Describe your changes briefly in your lab report.

Optional Task: Can you use a timer to achieve this pulsing behaviour?

T3. Reading a Simple Sensor (a switch)

We will be using the switch from our sensor box:



Create a new sketch and copy in the program below. Notice that we are using a library called “Serial”. This is for serial communication, and we will go more in depth in the week 3 lecture. For now, we will treat it like a black box. You can use the serial monitor to send messages over USB to your laptop. You can view these messages by clicking on the serial monitor button on the right-hand side of the Arduino IDE (the magnifying glass).

```
#define SWITCH D5

void setup() {
  Serial.begin(9600);
  pinMode(SWITCH, INPUT_PULLUP);
}

void loop() {
  if (digitalRead(SWITCH) == HIGH) {
    Serial.println("Switch Open");
  } else {
    Serial.println("Switch Closed");
  }
}
```

We will now wire up our circuit. Only two of the three pins on the switch module need to be connected, the **–** pin can be connected to the **GND,** and the **S** pin can be connected to **D5** on the **WeMOS.** Upload the program and try it out. Press the switch and see the messages. Neat!

For those interested, the pull-up means the button's logic is inverted. It goes HIGH when it's open, and LOW when it's closed.

Let's modify this program so that button behaves like a toggle. If we press the button, it should "toggle" on and when we press it again it should "toggle" off.

```
#define SWITCH D5

byte state = LOW;

void setup() {
  Serial.begin(9600);
  pinMode(SWITCH, INPUT_PULLUP);
}

void loop() {
  if (digitalRead(SWITCH) == LOW) {
    state = !state;
  }
  if (state) {
    Serial.println("Toggle On");
  } else {
    Serial.println("Toggle Off");
  }
}
```

Upload and try out this program. You might notice that sometimes clicking the button does not cause it to toggle. This is because the button needs to be debounced (have a look a Lecture 2 Slide 34). Let's fix this through software, by keeping track of when the button was last pushed, and ignoring any input that occurs in a short window after that time.

Lab Report Task 4: Modify this program to make your WeMOS debounce the switch. You might need to use a larger delay than in the lecture notes. Describe your changes briefly in your lab report.

But this is a very inefficient way of sensing a button push, and notice that our main loop() is continuously running. Our WeMOS is doing a lot of "busy waiting". Let's improve this!

T4. Microcontroller… Interrupted.

We are going to use interrupts to detect our button pushes and toggle the state as before. Recall that interrupts let us handle events without need to keep checking the state of our sensors. Have a look over the slides from Lecture 2 Slide 27 to 30.
Create a new sketch and copy the program below.

```
#define SWITCH D5

volatile byte state = HIGH;

IRAM_ATTR void toggle() {
    state = !state;
}

void setup() {
  Serial.begin(9600);
  pinMode(SWITCH, INPUT_PULLUP);
  attachInterrupt(
          digitalPinToInterrupt(SWITCH),
toggle, RISING);
}

void loop() {
  Serial.print("Working hard...");
  delay(1000);
  Serial.print("done. State: ");
  if (state) {
    Serial.println("Toggle On");
  } else {
    Serial.println("Toggle Off");
  }
}
```

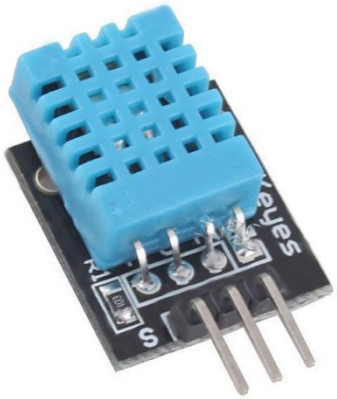
We can observe that in the setup() function, we are now attaching an ISR to the SWITCH pin. Also notice that the program has a new function called toggle(). This function is our Interrupt Service Routine or ISR.

We can now see that the main loop is spent idling (using delay) rather than checking the status of the button over and over. The state of the toggle is printed out in the main loop and not in the ISR. Note that this version is not debounced yet.

Lab Report Task 5: Improve the ISR verson of our code so that it also debounces the button push. Remember that ISRs should be very short functions. Also remember that any global variable used in an ISR needs to be volatile. Copy your entire ISR, any relevant code, and describe your changes briefly in your lab report.

T5. Is it hot in here?

Now we are going to try out another sensor, namely the DHT11, the Humidity & Temperature sensor. It is a simple and low-cost temperature and humidity sensor
It uses only 1 wire to communicate with the microcontroller.



Read the label of the pin carefully. There should be three pins on the DHT11, **DATA**, **VCC**, and **GND**. Connect **GND** and **VCC** on the DHT11, to **GND** and **5V** on the WeMOS. Connect the **DATA** pin to **D6** on your WeMOS. We will now take advantage of the libraries available for Arduino. Go to **Sketch > include Library > Manage Library.**

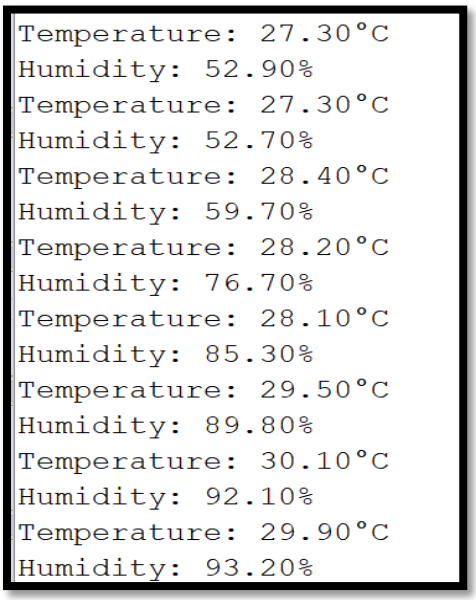
Search “DHT Sensor library” by AdaFruit. You will be asked to install the “AdaFruit Unified Sensor”, click "Install all“. Now let’s open some example code and test it out.

Open File➔Example➔DHT Sensor Library➔DHT_Unified_Sensor

You will need to make some small changes:

1. DHTPIN should be defined as D6
2. DHTTYPE should be defined as DHT11

Try running the program on your WeMOS. In the serial monitor, you will be able to see the temperature and humidity as shown below:



Try blowing on the sensor to increase the humidity or covering it with your hands to change the temperature. Ok, now let's combine all that we have done today together.

Lab Report Task 6: Modify the example program to make your builtin LED turn on if the humidity or temperature goes beyond a threshold. Describe your changes briefly in your lab report.

Optional Task: Can you make the brightness of the light proportional to the humidity or temperature?

T6. Exploring your box of sensors and actuators

By now, you can see something of a recipe for how we can prototype and try out sensors/actuators using our WeMOS board.

1. Find and then connect the electronic components to the correct pin(s). Usually we can find a wiring diagram
2. Write code to interact with the component. Sometimes, libraries are provided together with the components

This is a good general strategy for prototyping, so let's try it out.

Lab Report Task 7: You are to select one of the sensors/actuators in the box you have and try it out. Take a photo of your wired up system, and give any relevant code excerpts. Include these and a brief description in your lab report.

Let us know if you have any questions. Your tutor and lecturer will be walking around the lab.