Topic:
2. Arduino hardware

Summary session:

This session will introduce the second lab of the module. The second lab of the module is about the Arduino hardware. In this lab, you will learn the basics of how the Arduino board works, how to connect it to the Arduino software, and how to run your first patch: "Hello, world" with an LED.

TINKERING

Time: 5'

1. **Read this quote from the Exploratorium in San Francisco:**

"Tinkering is what happens when you try something you don't quite know how to do, guided by whim, imagination, and curiosity. When you tinker, there are no instructions—but there are also no failures, no right or wrong ways of doing things. It's about figuring out how things work and reworking them.

Contraptions, machines, wildly mismatched objects working in harmony—this is the stuff of tinkering.

Tinkering is, at its most basic, a process that marries play and inquiry."

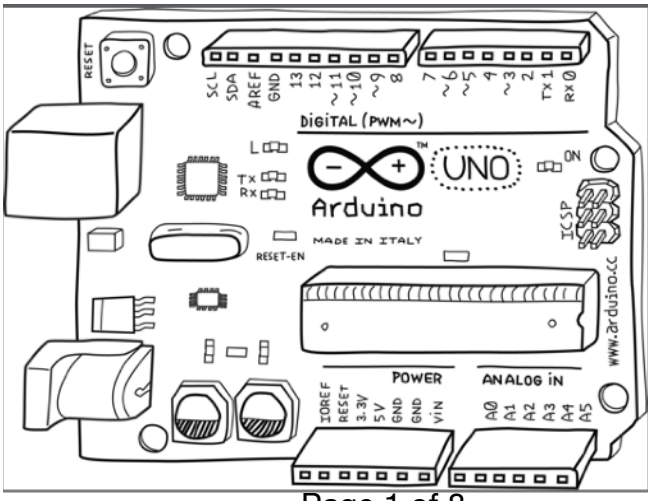
Excerpt From: Exploratorium in San Francisco. Massimo Banzi. "Make: Getting Started with Arduino".

BLOGGING

Time: 5' (setup), 110' (full activity)

1. **Write your second blog post at https://matd3039.our.dmu.ac.uk/**
    1. A good practice is to write a blog post during the whole lab session. This way, you can keep notes of the whole process, which can be useful for later.
    2. Start your second blog post of the module.
    3. Make sure to keep writing from the beginning to end of the lab session with key ideas and findings.

THE ARDUINO HARDWARE



Arduino is composed of two major parts: the Arduino board, which is the piece of hardware you work on when you build your objects; and the Arduino Integrated Development Environment, or IDE, the piece of software you run on your computer.

You use the IDE to create a sketch (a little computer program) that you upload to the Arduino board. The sketch tells the board what to do.

Time: 15'

1. **Look at the board. Can you identify where is the microchip (integrated circuit)? (which is the heart of your board).**
2. **What is the difference between a chip that is directly soldered on the board vs a chip on an IC holder?**
3. **Can you identify where is the USB serial adapter also known as the IC/USB controller?**
4. **Can you identify where are the two oscillators?**
5. **Can you identify where are the headers and soldering holes? Why can you use them for?**
6. **Can you identify the 14 Digital I/O pins (pins 0–13)?** Note that these pins can be either inputs or outputs. Inputs are used to read information from sensors, while outputs are used to control actuators. You will specify the direction (in or out) in the sketch you create in the IDE. Digital inputs can only read one of two values, and digital outputs can only output one of two values (HIGH and LOW).
7. **Can you identify the Analogue In pins (pins 0–5)?** Note that the analogue input pins are used for reading voltage measurements from analogue sensors. In contrast to digital inputs, which can distinguish between only two different levels (HIGH and LOW), analogue inputs can measure 1,024 different levels of voltage.
8. **Can you identify the Analogue Out pins (pins 3, 5, 6, 9, 10, and 11)?** Note that the analogue input pins are used for reading voltage measurements from analogue sensors. In contrast to digital inputs, which can distinguish between only two different levels (HIGH and LOW), analogue inputs can measure 1,024 different levels of voltage.
9. **Can you identify the Analogue Out pins (pins 3, 5, 6, 9, 10, and 11)?** Note that these are actually six of the digital pins that can perform a third function: they can provide analogue output. As with the digital I/O pins, you specify what the pin should do in your sketch. Pin 13 of the Arduino Uno is connected to the built-in LED. Pins 3,5,6,9,10,11 have PWM capability.
10. **USB cable** - when connected to the computer, provides 5 volts at 500mA. The power source you use determines the power you have available for your circuit. For instance, powering the circuit using the USB limits you to 500mA.
11. **Barrel Jack** - The Barrel jack, or DC Power Jack can be used to power your Arduino board. The barrel jack is usually connected to a wall adapter. The board can be powered by 5-20 volts but the manufacturer recommends to keep it between 7-12 volts. Above 12 volts, the regulators might overheat, and below 7 volts, might not suffice. 9 volts recommended, 2.1 mm barrel tip, center positive. In class we will only use USB cable.
12. **VIN Pin** - This pin is used to power the Arduino Uno board using an external power source. The voltage should be within the range mentioned above.
13. **GND** - In the Arduino Uno pinout, you can find 5 GND pins, which are all interconnected. The GND pins are used to close the electrical circuit and provide a common logic reference level throughout your circuit. Always make sure that all GNDs (of the Arduino, peripherals and components) are connected to one another and have a common ground.
14. **RESET** - resets the Arduino.

CONNECT THE ARDUINO SOFTWARE WITH HARDWARE

Time: 10'

1. In your computer, open the Arduino software IDE.
2. Connect your Arduino Uno clone to your computer via a USB cable. Note: Before plugging the Arduino clone to the computer, Windows users need to install a driver for USB connectivity: https://sparks.gogo.co.nz/ch340.html
3. An LED labeled PWR on the board should come on, and an LED labeled L should start blinking.
4. Now you need to select the proper port to communicate with the Arduino Uno.
5. In the Arduino IDE, select Serial Port and then:
    1. If you are on Mac, select the port that begins with `/dev/cu.wchusbserial1410`
    2. If you are on Windows, you will see one or more COM ports with different numbers. Make a note of which numbers are available. Now unplug your Arduino from your computer, look at the list of ports again, and see which COM port vanishes. It might take a moment or two, and you may have to leave the Tools menu and open it again to refresh the list of ports. Once you've figured out the COM port assignment, you can select that port from the Tools→Serial Port menu in the Arduino IDE.
6. Also, check that Arduino is configured for the type of board you're using. From the Tools menu in the Arduino IDE, select Board, and then select Arduino Uno.
7. Upload this sketch to the board through the USB connection and wait a couple of seconds for the board to restart by pressing the button "upload". The code that you have written is translated into the C language (which is generally quite hard for a beginner to use), and is passed to the `avr-gcc` compiler, an important piece of open source software that makes the final translation into the language understood by the microcontroller.
8. The general workflow is that you can upload a sketch and the board executes (performs) it.

BLINKING AN LED

Time: 15'

The LED blinking sketch is the first program that you should run to test whether your Arduino board is working and is configured correctly. Your Arduino board comes with an LED preinstalled. It's marked L on the board. This preinstalled LED is connected to pin number 13.

1. In Arduino IDE, select File→New and you'll be asked to choose a sketch folder name: this is where your Arduino sketch will be stored. Name it Blinking_LED and click OK.
2. Type the following sketch into the Arduino sketch editor (the main window of the Arduino IDE).

```
const int LED = 13; // LED connected to
                    // digital pin 13

void setup() {
  // put your setup code here, to run once:
  pinMode(LED, OUTPUT);    // sets the digital
                           // pin as output
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(LED, HIGH);   // turn the LED on (HIGH is the
voltage level)
  delay(1000);               // wait for a second
```

```
  digitalWrite(LED, LOW);    // turn the LED off by making the
voltage LOW
  delay(1000);               // wait for a second
}
```

3. Now that the code is in your IDE, you need to verify that it is correct. Click the Verify button. If everything is correct, you'll see the message "Done compiling" appear at the bottom of the Arduino IDE. This message means that the Arduino IDE has translated your sketch into an executable program that can be run by the board, a bit like an .exe file in Windows or an .app file on a Mac. If you get an error, most likely you made a mistake typing in the code. Look at each line very carefully and check each and every character.
4. Once your code verifies correctly, you can upload it into the board by clicking the Upload button. This will tell the IDE to start the upload process, which first resets the Arduino board, forcing it to stop what it's doing and listen for instructions coming from the USB port. The Arduino IDE will then send the sketch to the Arduino board, which will store the sketch in its permanent memory. Once the IDE has sent the entire sketch, you will see the message "Done uploading" at the bottom of the Arduino IDE and the Arduino board will start running your sketch.
5. There are two LEDs, marked RX and TX, on the Arduino board; these flash every time a byte is sent or received by the board. During the upload process, they keep flickering. This also happens very quickly, so unless you're looking at your Arduino board at the right time, you might miss it.
6. Once the code is in your Arduino board, it will stay there until you put another sketch on it.
7. What is your program doing? Try to understand it. How does the LED L turn on for a second and then turn off for a second?
8. Alternatively, you can also connect the shorter end of the LED to GND and the longer end to Pin 13.
9. "If you intend to keep the LED lit for a long period of time, you should use a resistor. We will see how to do it next week.

UNDERSTANDING YOUR CODE

Time: 20'

A programming language reference of Arduino can be found here: https://www.arduino.cc/reference/en/

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

The Arduino executes the code sequentially from top to bottom, so the first line at the top is the first one read; then it moves down. Let's analyse the code.

1. Notice the presence of curly braces, which are used to group lines of code together. These are particularly useful when you want to give a name to a group of instructions. To group together a number of instructions, you stick a `{` before the block of code and a `}` after.
2. You can see that there are two blocks of code defined in this way here. Before each one of them are some strange words, `void setup()` and `void loop()`:

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

}

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

```
}
```

3. `void setup()` gives a name to a block of code and `void loop()` gives a name to another block of code. These blocks are called functions. You can always call a function by its name, and Arduino will execute the instructions within it.
4. Arduino can do only one thing at a time, one instruction at a time. As Arduino runs your program, line by line, it's executing, or running, only that one line. When it jumps to a function, it executes the function, line by line, before returning to where it was. Arduino can't run two sets of instructions at the same time.
5. Arduino always expects that you've created two functions: one called `setup()` and one called `loop()`.
   1. `setup()` is where you put all the code that you want to execute once at the beginning of your program. See: https://www.arduino.cc/reference/en/language/structure/sketch/setup/
   2. `loop()` contains the core of your program, which is executed over and over again. This is done because Arduino is not like your regular computer—it cannot run multiple programs at the same time, and programs can't quit. When you power up the board, the code runs; when you want to stop, you just turn it off. See: https://www.arduino.cc/reference/en/language/structure/sketch/loop/
6. Any text beginning with // is ignored by Arduino. These lines are comments, which are notes that you leave in the program for yourself, so that you can remember what you did when you wrote it, or for somebody else, so they can understand your code.
7. Let's analyse our blinking an LED code:

```
const int LED = 13; // LED connected to
                    // digital pin 13

void setup() {
  // put your setup code here, to run once:
  pinMode(LED, OUTPUT);    // sets the digital
                           // pin as output
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(LED, HIGH);   // turn the LED on (HIGH is the
voltage level)
  delay(1000);               // wait for a second
  digitalWrite(LED, LOW);    // turn the LED off by making the
voltage LOW
  delay(1000);               // wait for a second
}
```

   1. `const int` means that LED is the name of an integer number that can't be changed (i.e., a constant) whose value is set to 13. It's like an automatic search-and-replace for your code; in this case, it's telling Arduino to write the number 13 every time the word LED appears. The reason we need the number 13 is that the preinstalled LED we mentioned earlier is attached to Arduino pin 13. A common convention is to use uppercase letters for constants.
   2. `void setup()` This line tells Arduino that the next block of code will be a function named `setup()`.
   3. `{` With this opening curly brace, a block of code begins.
   4. `pinMode(LED, OUTPUT)`: `pinMode()` tells Arduino how to configure a certain pin. See: https://www.arduino.cc/reference/en/language/functions/digital-io/pinmode/ Digital pins can be used either as input or output, but we need to tell Arduino how we intend to use the pin. In this case, we need an output pin to control our LED.

`pinMode()` is a function, and the words (or numbers) specified inside the parentheses are called its arguments. Arguments are whatever information a function needs in order to do its job. The `pinMode()` function needs two arguments. The first argument tells `pinMode()` which pin we're talking about, and the second argument tells `pinMode()` whether we want to use that pin as an input or output. INPUT and OUTPUT are predefined constants in the Arduino language.

1. Remember that the word `LED` is the name of the constant which was set to the number 13, which is the pin number to which the LED is attached. So, the first argument is LED, the name of the constant.
2. The second argument is `OUTPUT`, because when Arduino talks to an actuator, it's sending information out.

5. `}` This closing curly brace signifies the end of the setup() function.
6. `void loop() {` => `loop()` is where you specify the main behaviour of your interactive device. It will be repeated over and over again until you remove power from the board.
7. `digitalWrite(LED, HIGH);` => `digitalWrite()` is able to turn on (or off) any pin that has been configured as an output. See: https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/ `digitalWrite()` expects two arguments: the first argument tells digitalWrite() what pin we're talking about (we use the constant name LED to refer to pin number 13), which is where the preinstalled LED is attached. The second argument is different: in this case, the second argument tells `digitalWrite()` whether to set the voltage level to `0 (LOW)` or to `5 V (HIGH)`. On the Arduino, HIGH means that the pin will be set to 5 V, while LOW means the pin will be set to 0 V. For most cases you can just pretend that HIGH means ON and LOW means OFF.
8. `delay()` basically makes the microcontroller sit there and do nothing for the amount of milliseconds that you pass as an argument. See: https://www.arduino.cc/reference/en/language/functions/time/delay/ We need to keep the LED on for a while so that we can see it, and the way to do that is to tell Arduino to wait for a while before going to the next step. We tell the LED to stay on for 1 second here.
9. `digitalWrite(LED, LOW);` => This instruction now turns off the LED that we previously turned on.
10. `delay(1000);` => Here, we delay for another second. The LED will be off for 1 second.
11. `}` This closing curly brace marks the end of the `loop()` function. When Arduino gets to this, it starts over again at the beginning of `loop()`.
12. To sum up, this program does this:
    1. Turns pin 13 into an output (just once at the beginning)
    2. Enters a loop
    3. Switches on the LED connected to pin 13
    4. Waits for a second
    5. Switches off the LED connected to pin 13
    6. Waits for a second
    7. Goes back to beginning of the loop
13. Feel free to play with the code:
    1. Reduce the amount of delay, using different numbers for the on and off pulses so that you can see different blinking patterns.
    2. What happens when you make the delays very small?
14. Remember that Arduino doesn't really understand what you connect to the output pins. Arduino just turns the pin HIGH or LOW, which could be controlling a light, or an electric motor, or your car engine.

UNDERSTANDING SERIAL COMMUNICATION

Serial communication is used to exchange data between the Arduino board and another serial device such as computers, displays, sensors and more. Each Arduino board has at least one serial port. Serial communication occurs on digital pins 0 (RX) and 1 (TX) as well as via USB.

Arduino supports serial communication through digital pins with the SoftwareSerial Library as well. This allows the user to connect multiple serial-enabled devices and leave the main serial port available for the USB.

After a sketch is uploaded and is running on Arduino, the sketch can use this same connection to send messages to or receive them from your computer. The way we do this from a sketch is to use the serial object.

Serial means the data is sent 1 by 1 several times per second depending on the configuration of the rate. The serial port is connected to the USB port on the board so we can use the built in serial function to send data to the serial monitor in the Arduino IDE.

Time: 20'

1. Using the same code as before, you will add the following code to print out the output values to the serial monitor.

```
const int LED = 13; // LED connected to
                    // digital pin 13

void setup() {
  // put your setup code here, to run once:
  pinMode(LED, OUTPUT);    // sets the digital
                           // pin as output
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(LED, HIGH);   // turn the LED on (HIGH is the
voltage level)
  delay(1000);               // wait for a second
  if (digitalRead(LED)==HIGH) {
     Serial.write('H');
  } else {
     Serial.write('L');
  };
  digitalWrite(LED, LOW);    // turn the LED off by making the
voltage LOW
  delay(1000);               // wait for a second
}
```

2. This new code is a conditional: if the LED is HIGH, it should print the value 'H' on the terminal. Else, if the LED is LOW, it should print the value 'L' on the terminal.
3. After you've uploaded the code to your Arduino, you might think that nothing interesting happens. What you need is the serial monitor, and it's built in to the Arduino IDE.
4. The Serial Monitor button is near the top-right corner of the Arduino IDE. It looks a bit like a magnifying glass. Click the Serial Monitor button to open the monitor, and you'll see the values rolling past in the bottom part of the window.
5. One problem with this code is that, because Arduino reads line by line, we never see the value 'L' printed on the Serial Monitor.
6. A solution is to create a function checkstatus() and call it after each of the two delays. Change the code to:

```
const int LED = 13; // LED connected to
                    // digital pin 13
```

```
void setup() {
  // put your setup code here, to run once:
  pinMode(LED, OUTPUT);     // sets the digital
                            // pin as output
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(LED, HIGH);   // turn the LED on (HIGH is the
voltage level)
  delay(1000);               // wait for a second
  checkstatus();
  digitalWrite(LED, LOW);    // turn the LED off by making the
voltage LOW
  delay(1000);               // wait for a second
  checkstatus();
}

void checkstatus() {

  if (digitalRead(LED)==HIGH) {
     Serial.write('H');
  } else {
     Serial.write('L');
  };

}
```

BLOGGING

1. Complete your blog post.