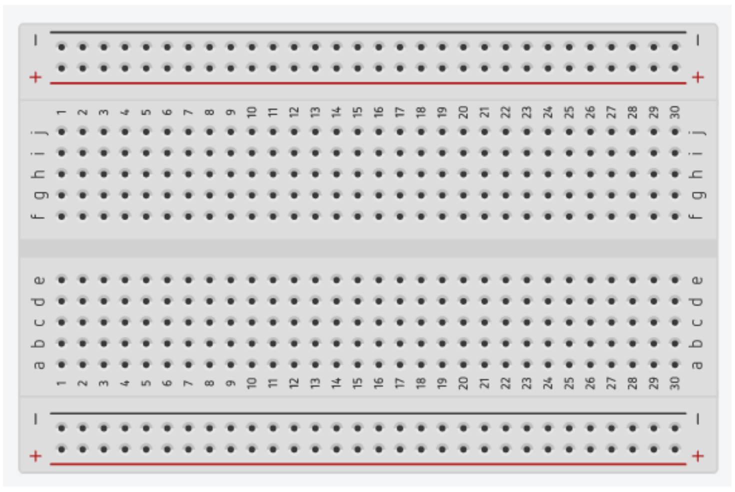
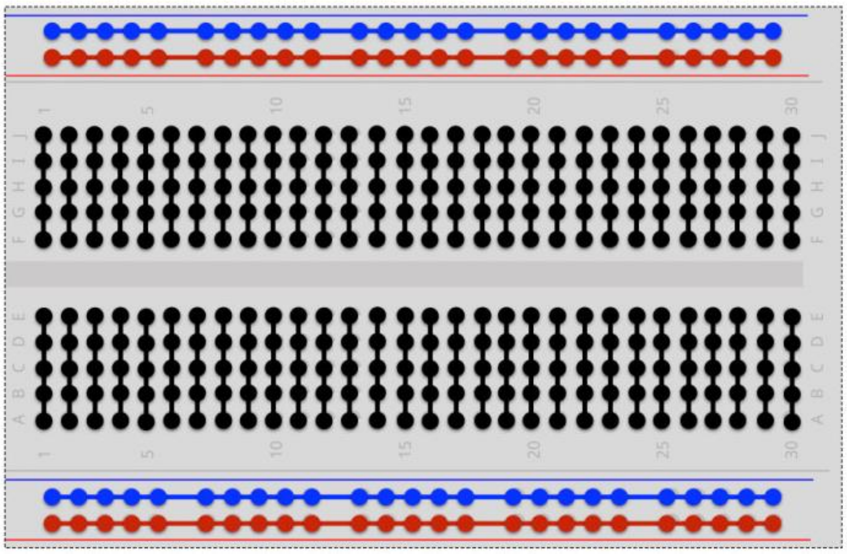
Bread board

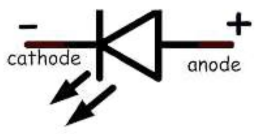


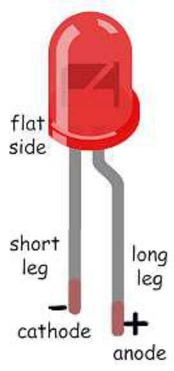
1. Connections

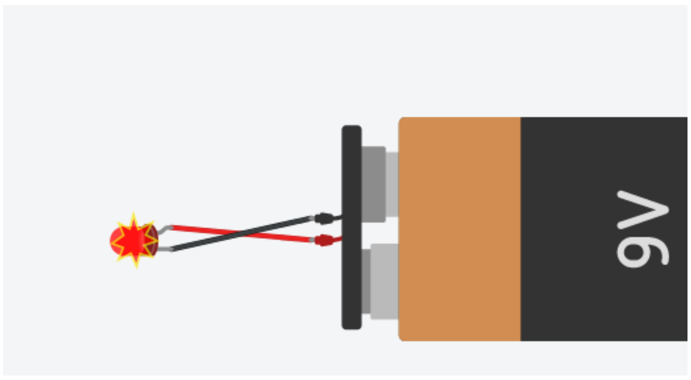


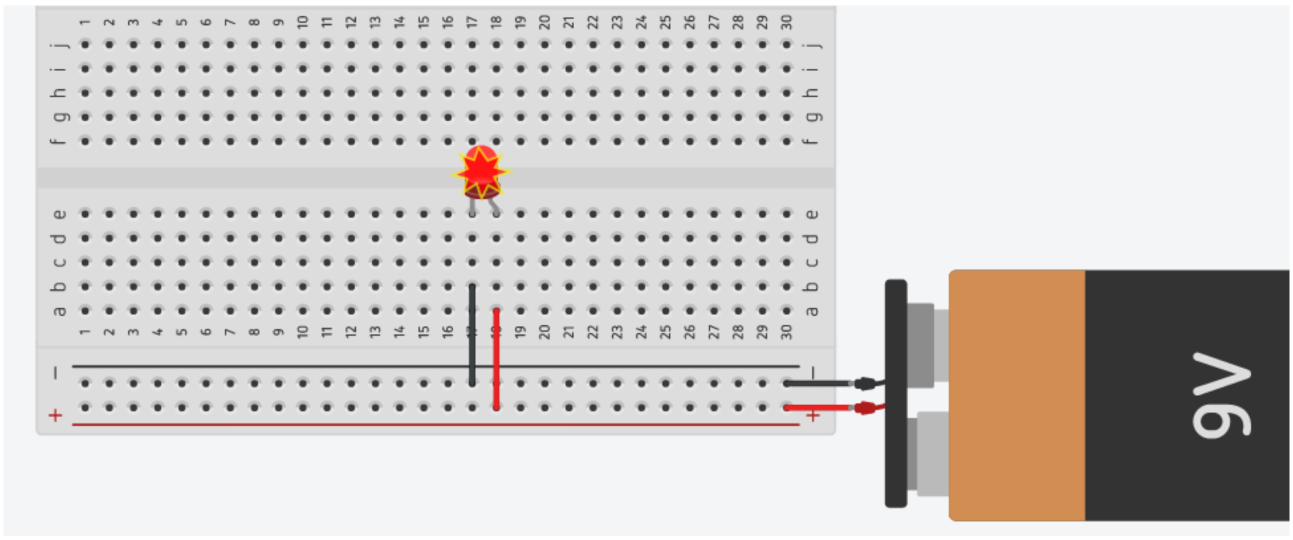
2. Powering the breadboard up to light an LED

Intentionally, we will start by doing this the **wrong** way
Connect
 the **long** leg to the **positive** terminal
 the **short** leg to the **negative** terminal



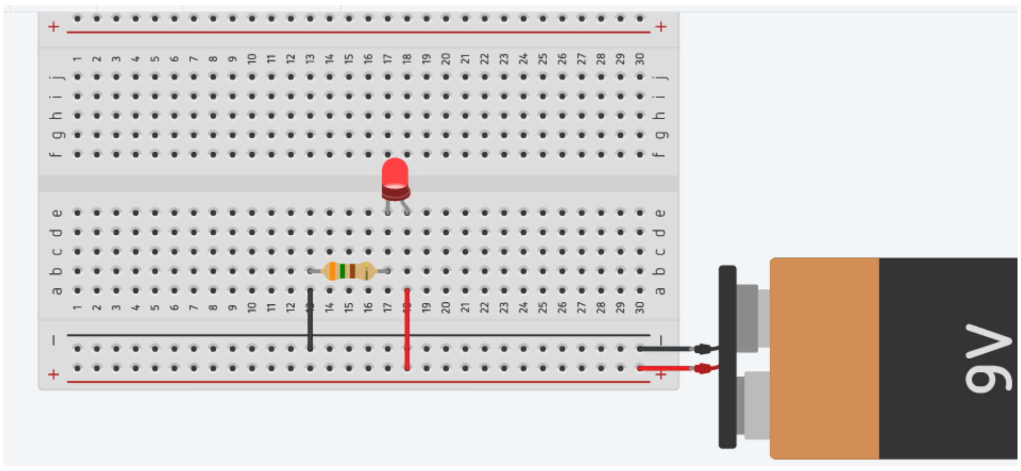






Again, this is the **WRONG** way of connecting the LED, it will lead the LED to burn.

The right way would be to include a resistor that controls how much CURRENT flows through the LED



The question now is, which **VALUE** should the resistor have?

To find out about this, we need to know some information about the LED

- The maximum current that goes through the RED led is **20 mA** … which is **.02 A**
- The voltage across the RED led terminals is ~ **2.2 V**

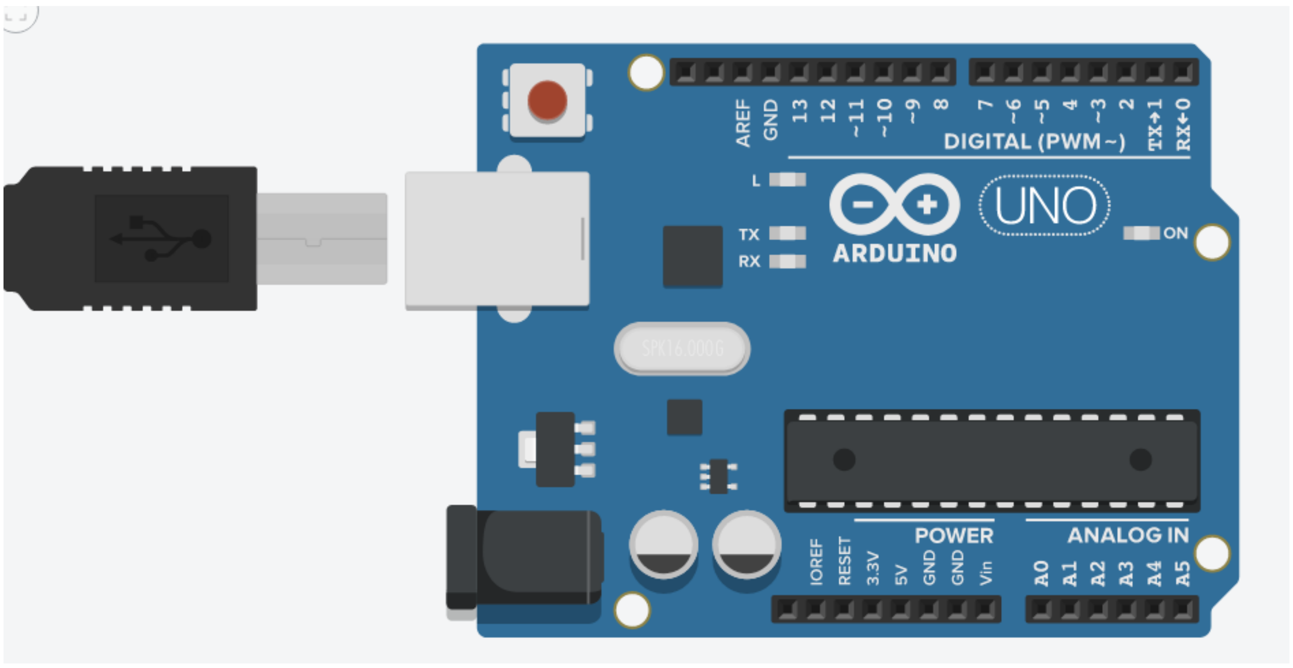
To calculate the value of the resistor, we use **Ohm's law**

$$R = \frac{\Delta V}{I}$$

$$R = \frac{SourceVoltage - LEDvoltage}{Current}$$

$$R = \frac{9 - 2.2}{0.02} = \frac{6.8}{0.02} = 340\Omega$$

Arduino (Uno)



Pins

Digital: These are **13 digital pins** that you can use as **inputs** or **outputs**. They're only meant to work with digital signals, which have 2 different levels:

- **Binary 0:** represented by the voltage 0V
- **Binary 1:** represented by the voltage 5V

Analog (bottom-right): These are **6 analog pins** that you can use as analog inputs. They're meant to work with an arbitrary voltage between 0V and 5V.

Power: These are **5 power pins**. They're mainly used for powering external components (Like the **breadboard** mentioned above) .

Microcontroller

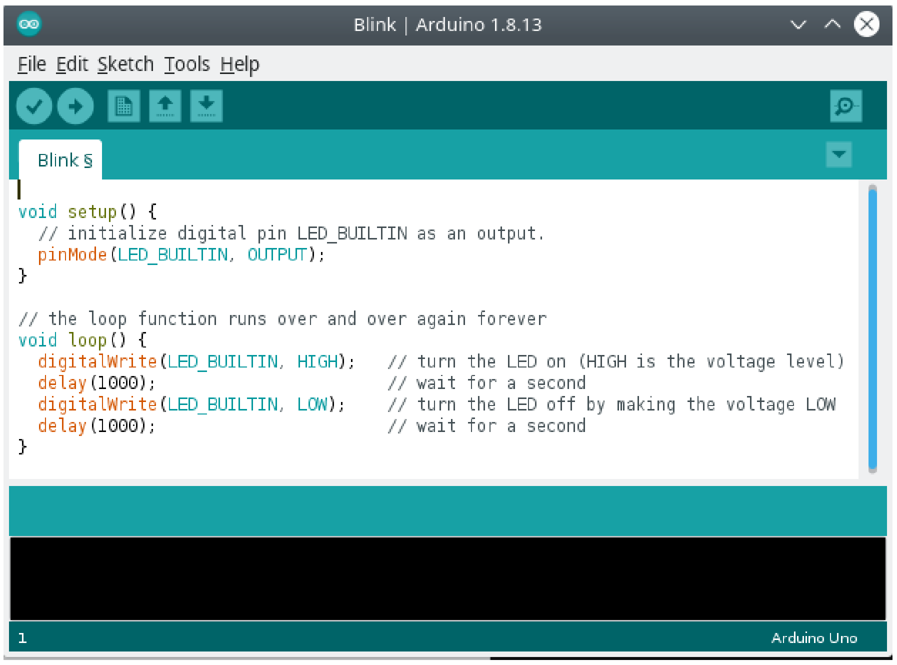
- The black rectangular device with 28 pins (**ATmega328**).
- can be easily replaced.
- Inside of this microcontoller, you can find
  - CPU
  - RAM (Volatile – its data gets deleted in case of power loss)– Working memory
  - Flash memory (non-volatile) - Holds program instructions
  - EEPROM (non-volatile) - acts like a tiny hard drive

Writing our first program

Assuming, you have an Arduino board already, connected to a computer USB port

1. Download and install the Arduino IDE from https://www.arduino.cc/en/software/

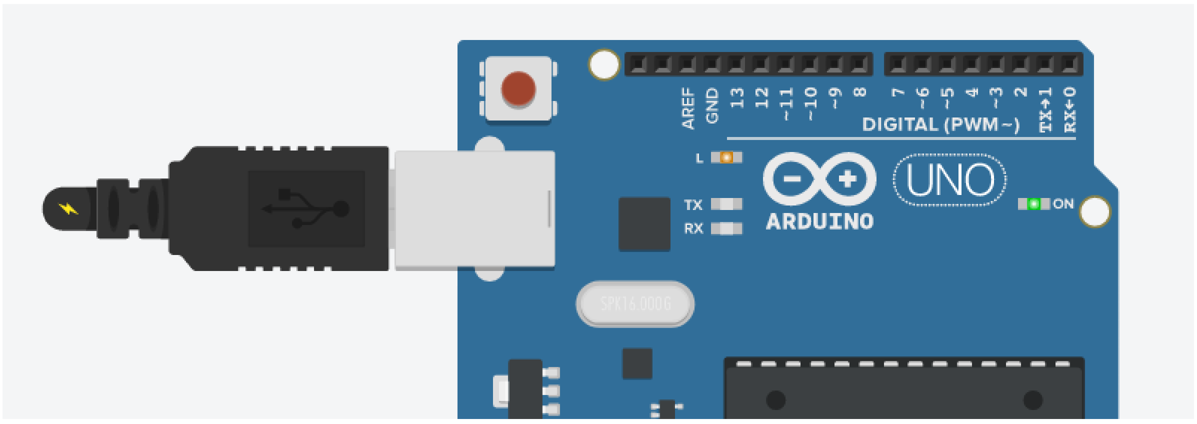
2. Start the program, you'll be presented with this screen



```
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

3. Just keep it as it is and press the arrow button (You'll find it on the top left)

4. A built-in LED on the Arduino board (**marked with L**) will keep blinking



Try this online on

https://www.tinkercad.com/

Serial monitor

What if we want to do some **debugging** ?

Debugging is extremely useful when you want to know what went wrong with your code … you might need to know why is that LED not blinking … why is that **sensor** not sensing stuff any more.

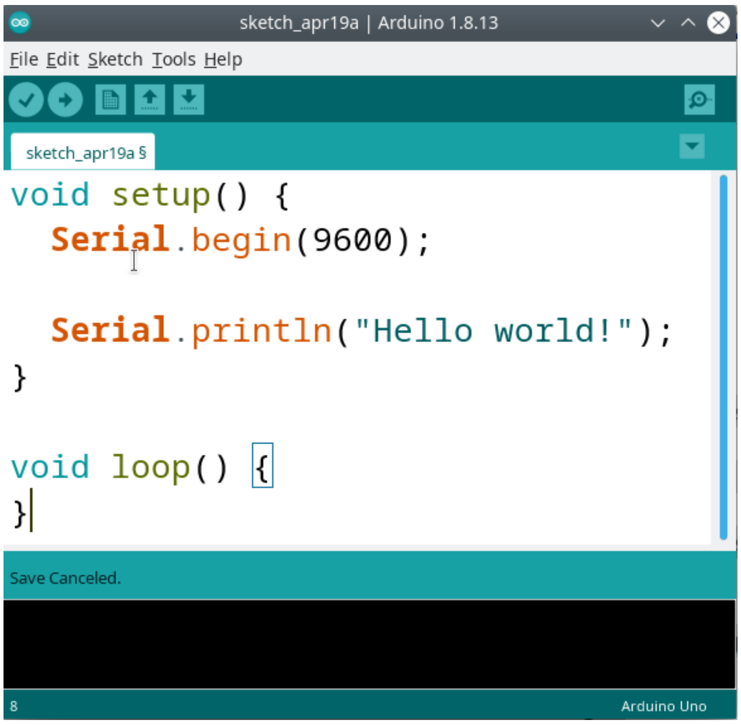
We need then to be able to **print** to a screen some message informing us what kind of signal (**HIGH** or **LOW**) the LED is getting.
Well, here comes into play what's called the **Serial monitor**

The serial monitor is the **link between the computer and your Arduino**. With it you can

- Send information **to** your computer
- Receive information (signals) **from** your computer

Using the serial monitor

Below is a very basic example of using the serial monitor



```
void setup() {
  Serial.begin(9600);

  Serial.println("Hello world!");
}

void loop() {
}
```

```
Save Canceled.
```

First thing to note, we have nothing at all inside of the **loop** function
This does **NOT** mean you can remove it … it needs to be there even if it has nothing inside of it.
Same thing applies to the **setup** function.

Ok, next thing to mention is “**Serial**”
Serial is just a library that includes a couple of functions that helps us doing stuff like debugging

Serial.begin(9600);

That first line inside **setup**, is the transfer rate … you don’t really have to worry about it for now .. but if you’re interested in knowing what it means, it just means how many bits it can send every second … so in this example .. it can send 9600 bits every second

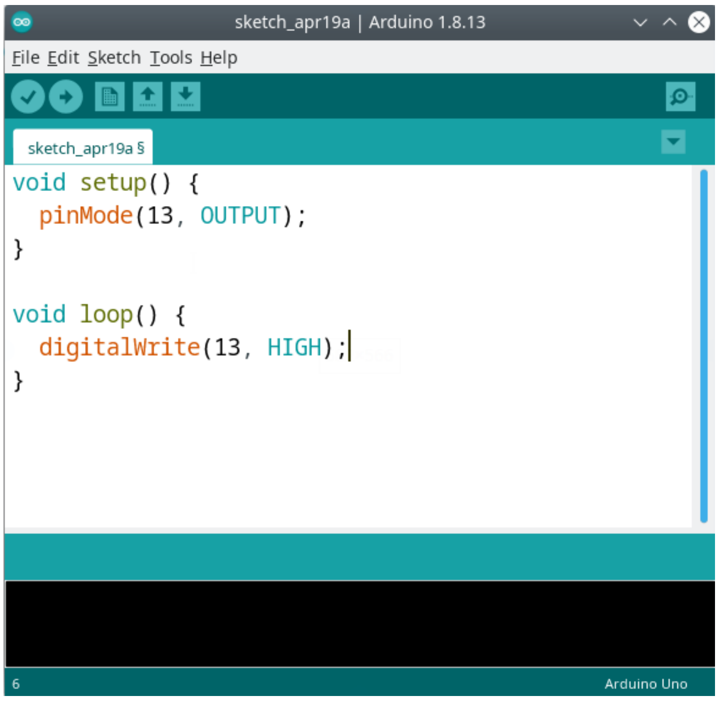
Serial.println(“Hello world”);

The second line sends the text “Hello world” from the arduino kit to your PC, this is really the part we’re interested in.
This uses a function called “println” inside the library “Serial”

“**println**” is short for “**Print Line**”

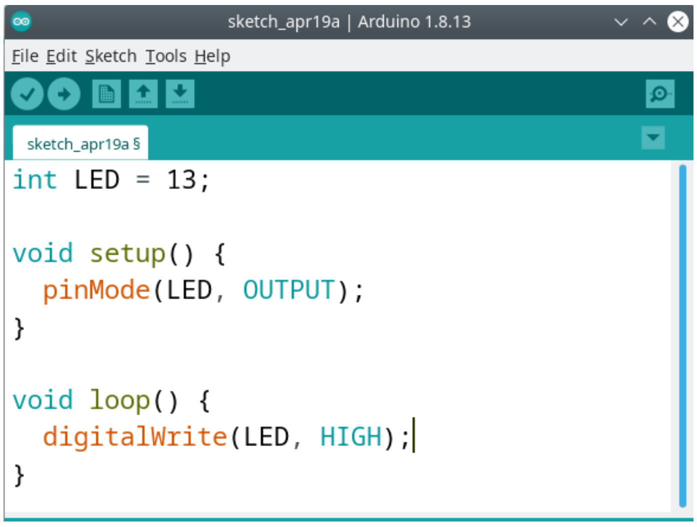
Let's try to make use of the serial monitor the other way around … let's get some input from the PC and pass it to the arduino kit

We will be controlling an LED using input from the keyboard
if we enter a **1** at the serial monitor input field the LED must **light up**
if we enter a **0** at the serial monitor input field the LED must be **turned off**



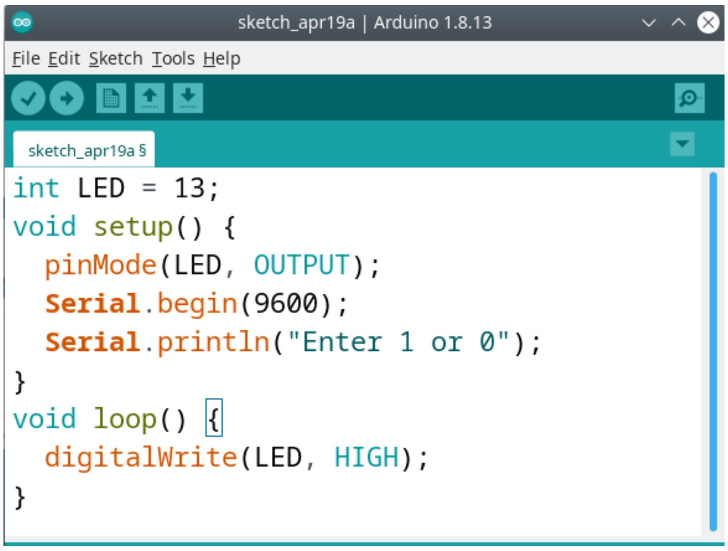
```
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);
}
```

If you recall from the previous class, we connected an LED to pin #13 and used the code above to make it light up. Let's just improve it a little bit by using some variables



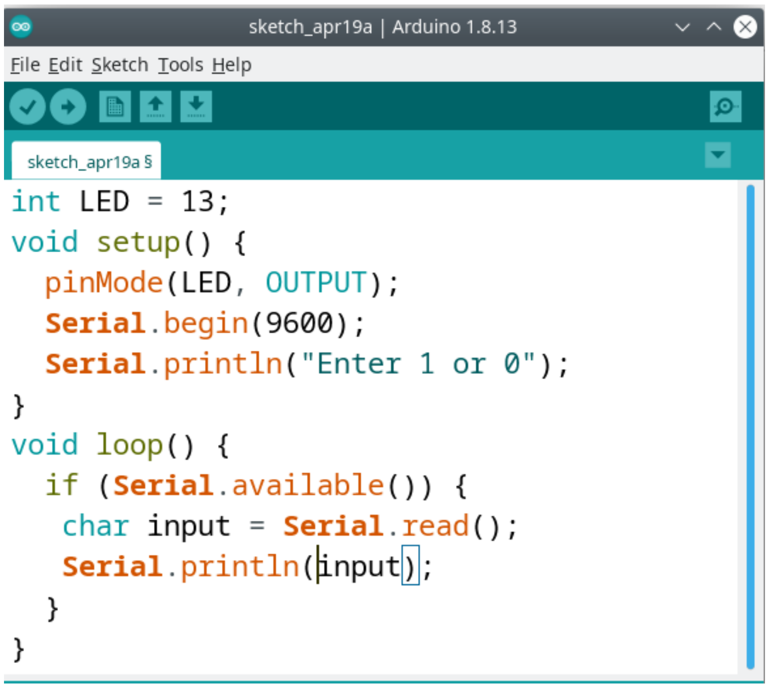
```
int LED = 13;

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

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

Next thing to do is to get some input from the PC and acting upon it.



```
int LED = 13;
void setup() {
  pinMode(LED, OUTPUT);
  Serial.begin(9600);
  Serial.println("Enter 1 or 0");
}
void loop() {
  digitalWrite(LED, HIGH);
}
```

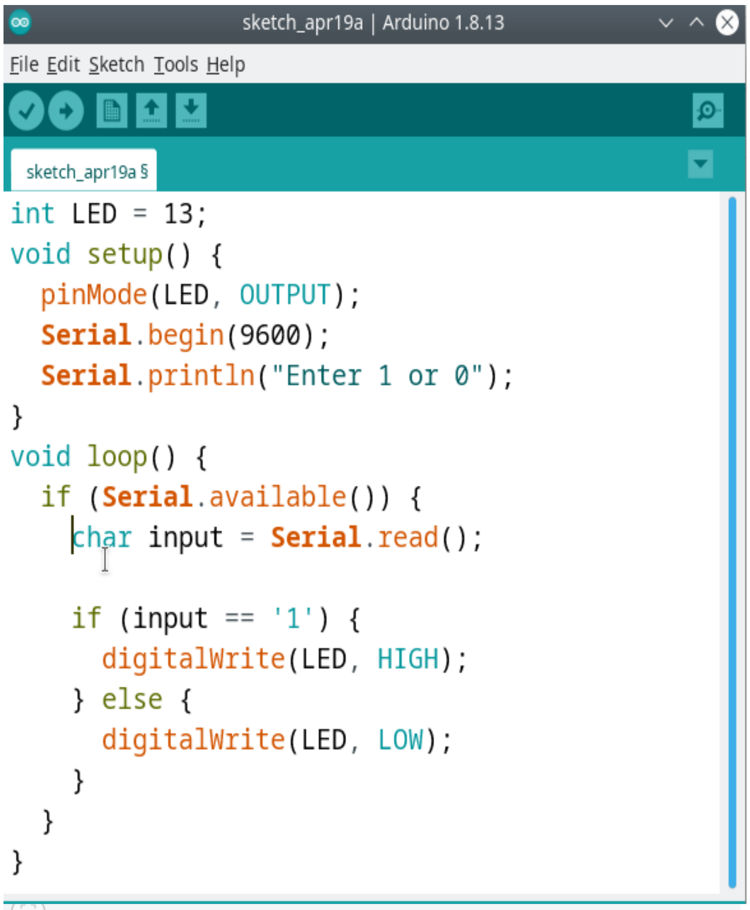
Now the **Serial monitor** is ready, let’s listen to inputs from the PC side



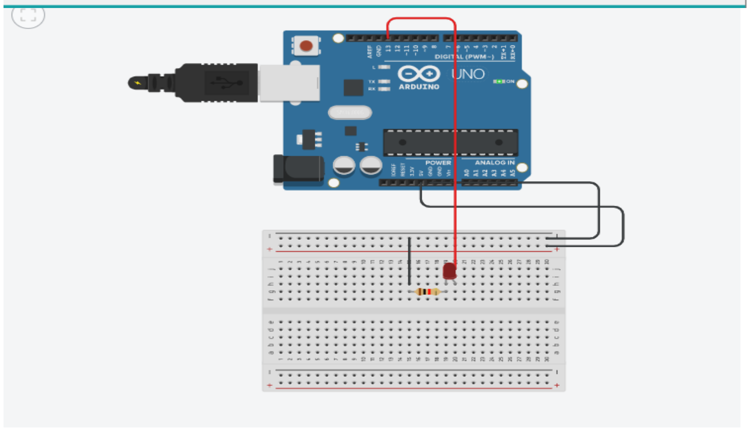
```
int LED = 13;
void setup() {
  pinMode(LED, OUTPUT);
  Serial.begin(9600);
  Serial.println("Enter 1 or 0");
}
void loop() {
  if (Serial.available()) {
   char input = Serial.read();
   Serial.println(input);
  }
}
```

Serial.available() is a function that evaluates to **true** when there’s an input .. and to **false** when there’s no input

So here all we're doing is just reading input and printing it back.

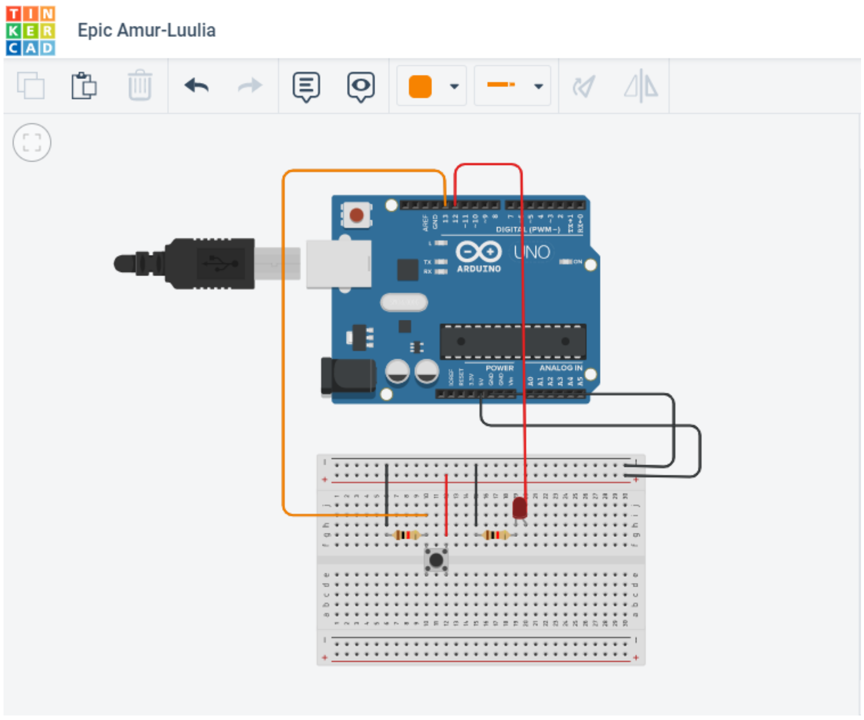


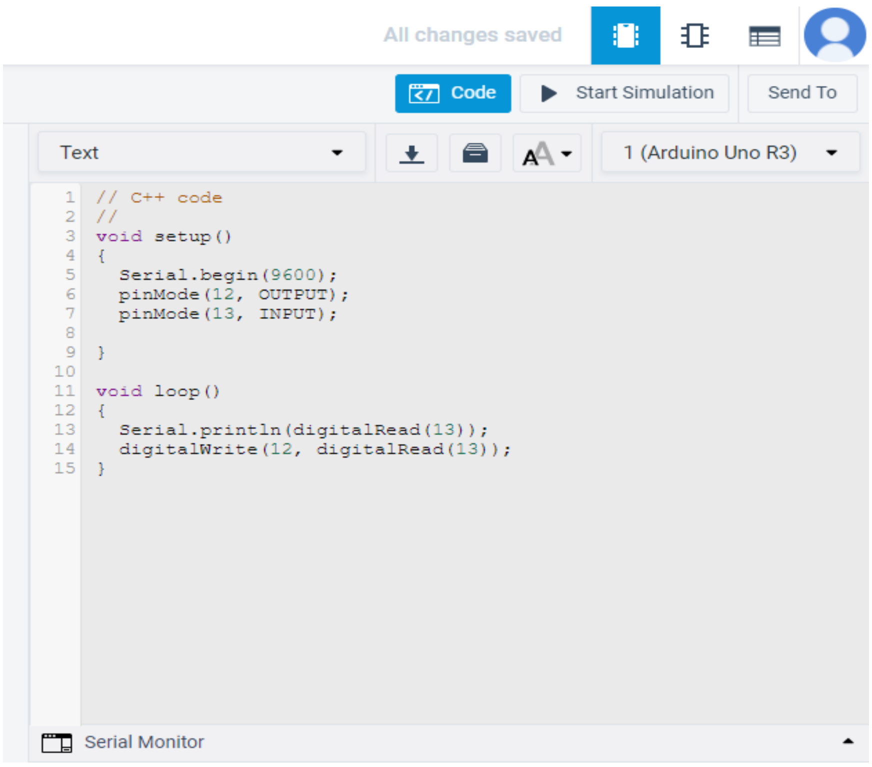
```
int LED = 13;
void setup() {
  pinMode(LED, OUTPUT);
  Serial.begin(9600);
  Serial.println("Enter 1 or 0");
}
void loop() {
  if (Serial.available()) {
    char input = Serial.read();

    if (input == '1') {
      digitalWrite(LED, HIGH);
    } else {
      digitalWrite(LED, LOW);
    }
  }
}
```



Finally we can use that input in an **if statement** and light or turn off the LED based on that

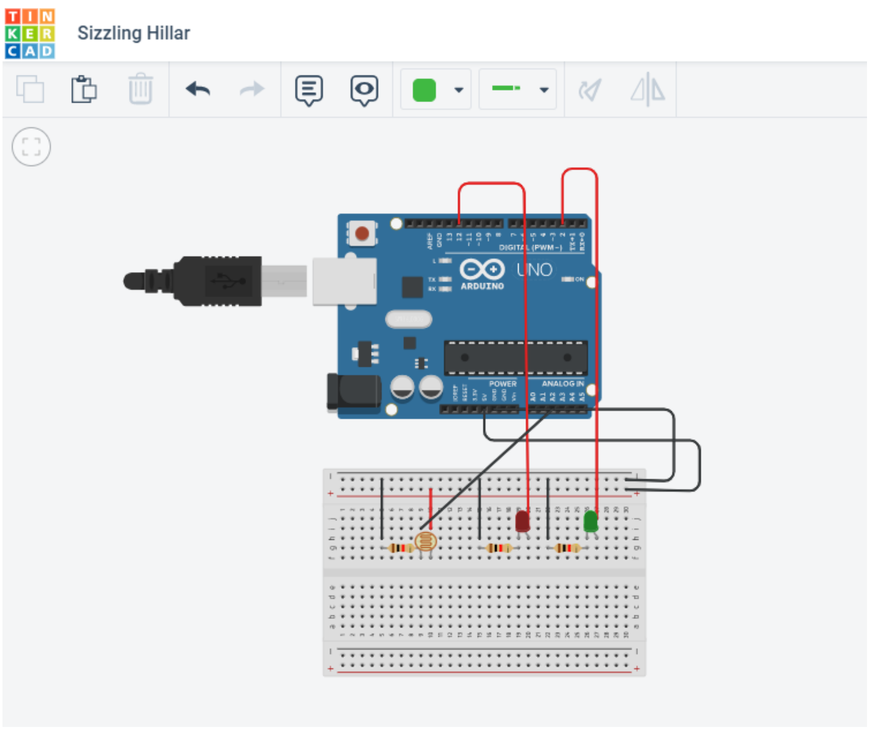
Using switch button to turn a lamp on and off:

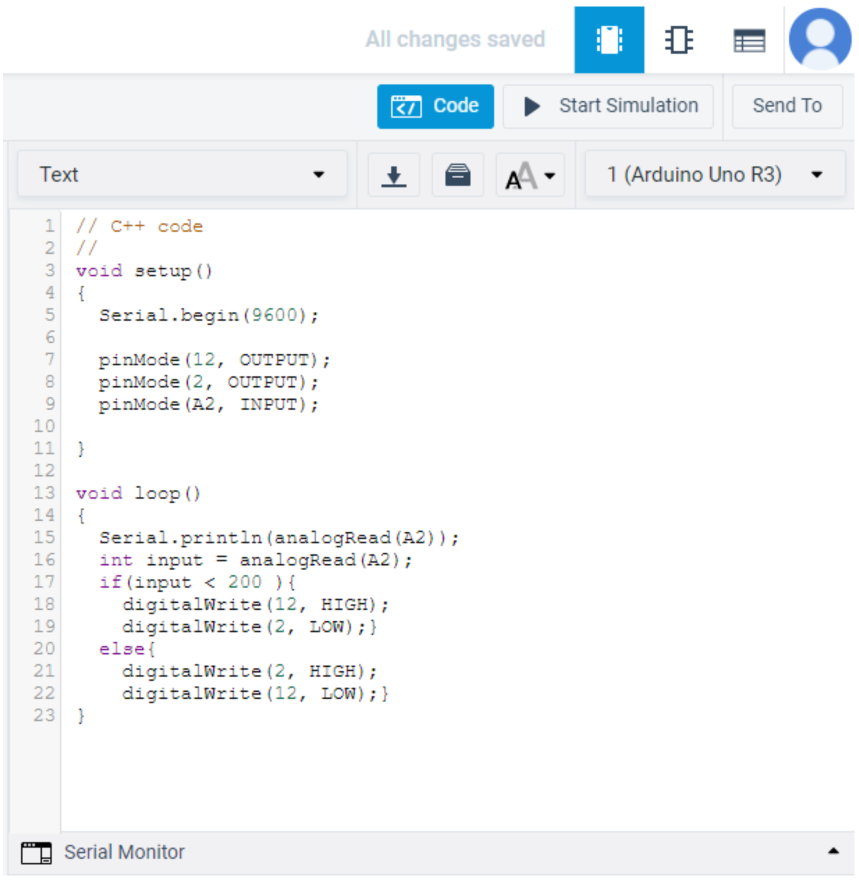




```
// C++ code
//
void setup()
{
  Serial.begin(9600);
  pinMode(12, OUTPUT);
  pinMode(13, INPUT);

}

void loop()
{
  Serial.println(digitalRead(13));
  digitalWrite(12, digitalRead(13));
}
```

Using photoresistor to turn a lamp on and off:





```
// C++ code
//
void setup()
{
  Serial.begin(9600);

  pinMode(12, OUTPUT);
  pinMode(2, OUTPUT);
  pinMode(A2, INPUT);

}

void loop()
{
  Serial.println(analogRead(A2));
  int input = analogRead(A2);
  if(input < 200 ){
    digitalWrite(12, HIGH);
    digitalWrite(2, LOW);}
  else{
    digitalWrite(2, HIGH);
    digitalWrite(12, LOW);}
}
```

#Assignment:

Use the temperature sensor instead of the photoresistor to make the first lamp turn on and the second turn off.