Smart Objects

SAPIENZA Università di Roma, M.Sc. in Product Design
Fabio Patrizi

What is a Smart Object?

- Essentially, an object that:
  - Senses
  - Thinks
  - Acts

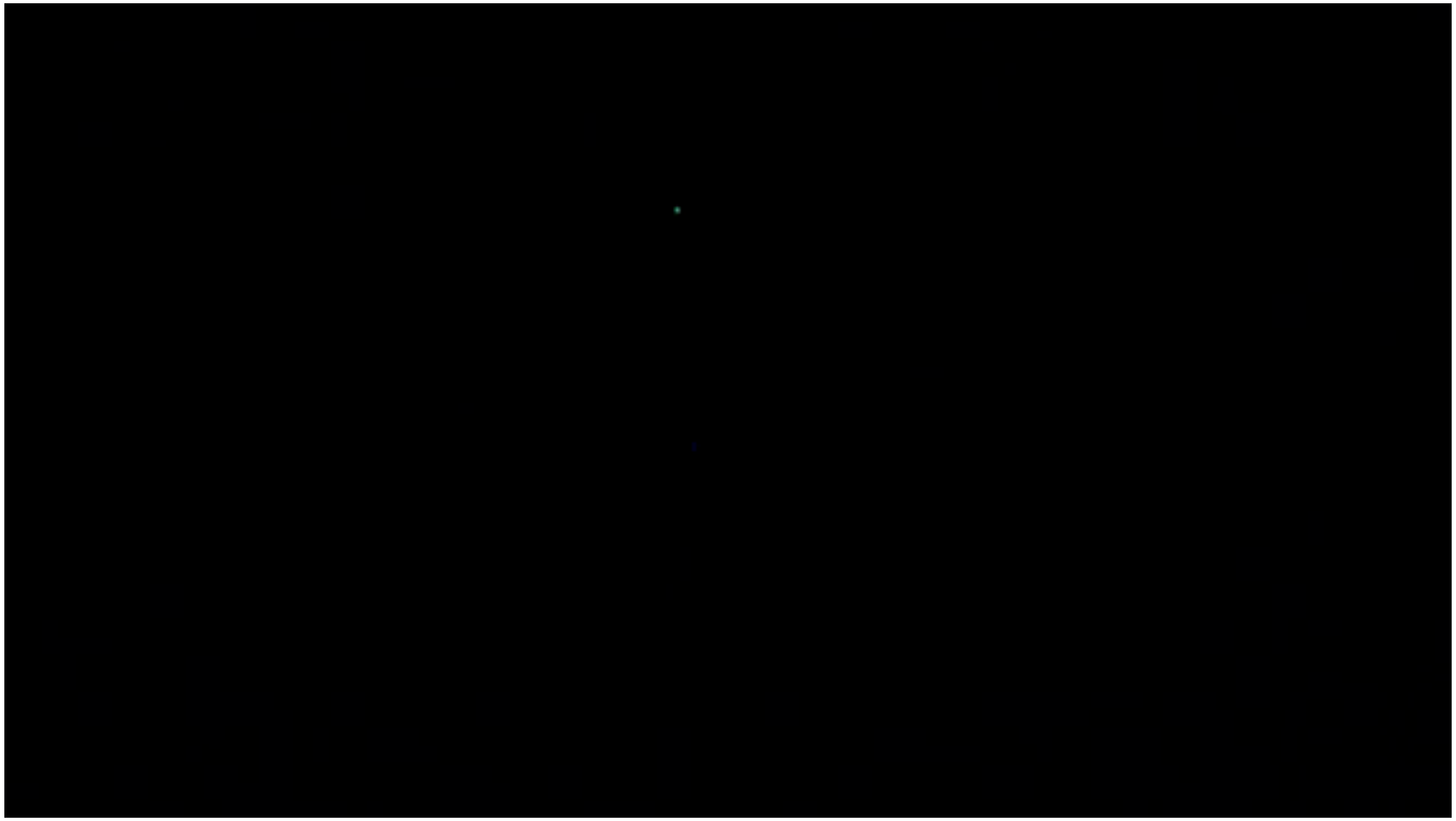
Example 1

https://www.youtube.com/watch?v=6bNcjD8ekE0

Example 2

https://www.youtube.com/watch?v=TWjHFw_eDIY

Example 3



https://www.youtube.com/watch?v=YErWfe0aTiQ

Example 4

https://www.youtube.com/watch?v=OLfF4b49MLs

Example 5

https://www.youtube.com/watch?v=dvAfefBIR4c

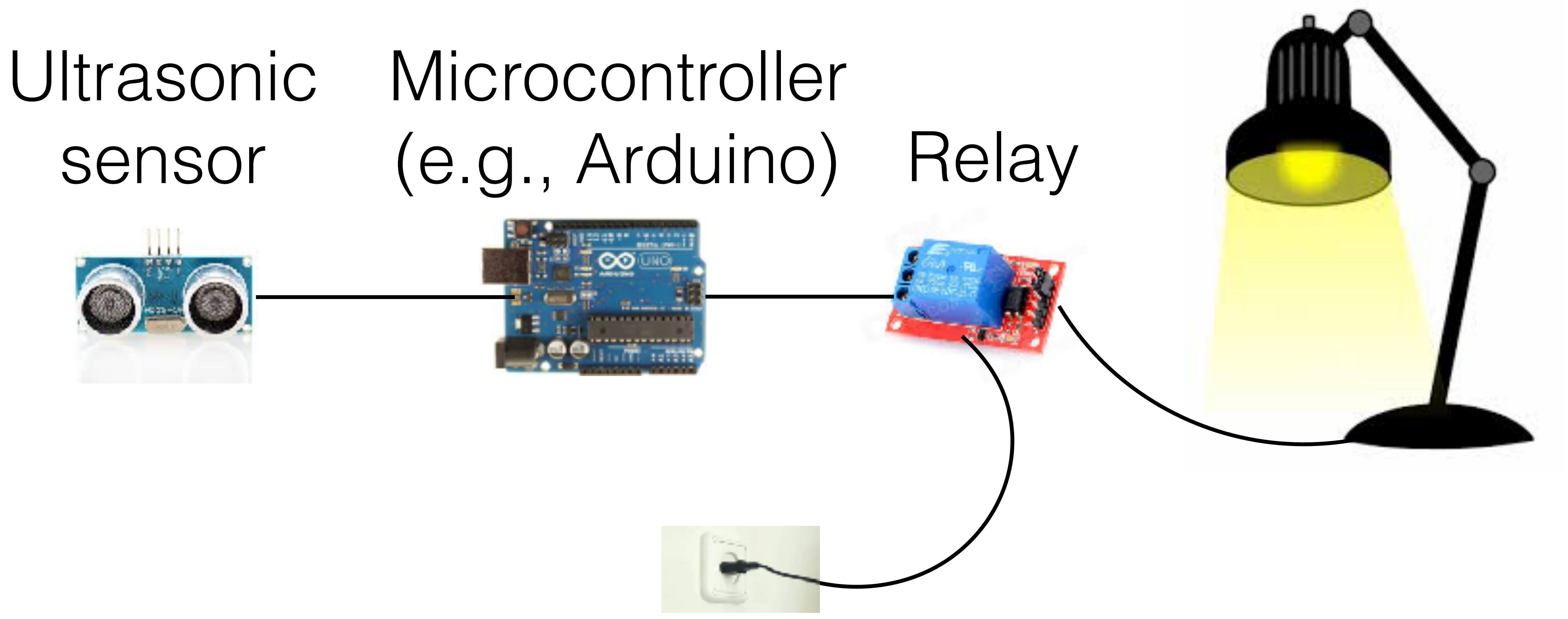
Homework 1

- Browse the web to learn about Arduino projects and make a list of all the capabilities you encounter (e.g., can sense light, can rotate wheel, can switch light on, etc.)
- This will be useful for your prototype, as you'll learn about available features

Smart Objects (A closer look)

Remember Example 1 (motion-controlled lamp)

A simplified schema:



Smart Objects (A closer look)

Ultrasonic sensor

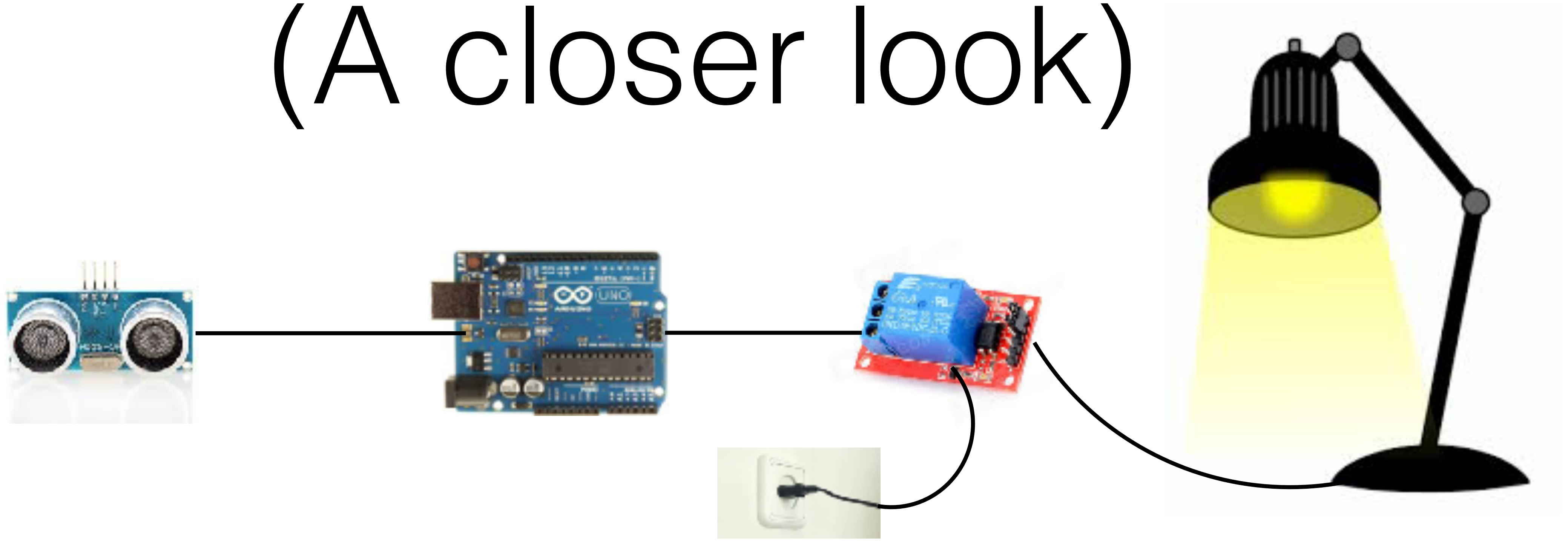
Senses distance to closest object (up to 30 cm)

Relay

An electronically-controlled interruptor

We need to know WHAT these components do, not HOW THEY ARE INTERNALLY BUILT!

Smart Objects (A closer look)



How does it work?

1. The ultrasonic sensor measures the distance to the closest object
2. Arduino reads the measurement
3. If the distance is small enough, Arduino switches the relay on
4. The relay lets the current flow and the lamp switch on
5. If the distance is large, Arduino switches the relay off
6. The relay interrupts the current flow and the lamp switches off

Sense-think-act

Sense-think-act is a popular *interaction paradigm* (and the one we will use in this course)

- Sense: observe the environment (some features)
- Think: based on observation, make a decision
- Act: based on decision, perform some action(s)

Sense-think-act

Who takes care of what?

- Sense: sensors (e.g., ultrasonic sensor)
- Think: micro controller (e.g., Arduino)
- Act: actuators (e.g., relay)

Sense-think-act

We will learn how to make products interactive

We will do so by implementing the sense-think-act paradigm with Arduino

To do so, we need to:

1. Understand a bit of the Arduino structure

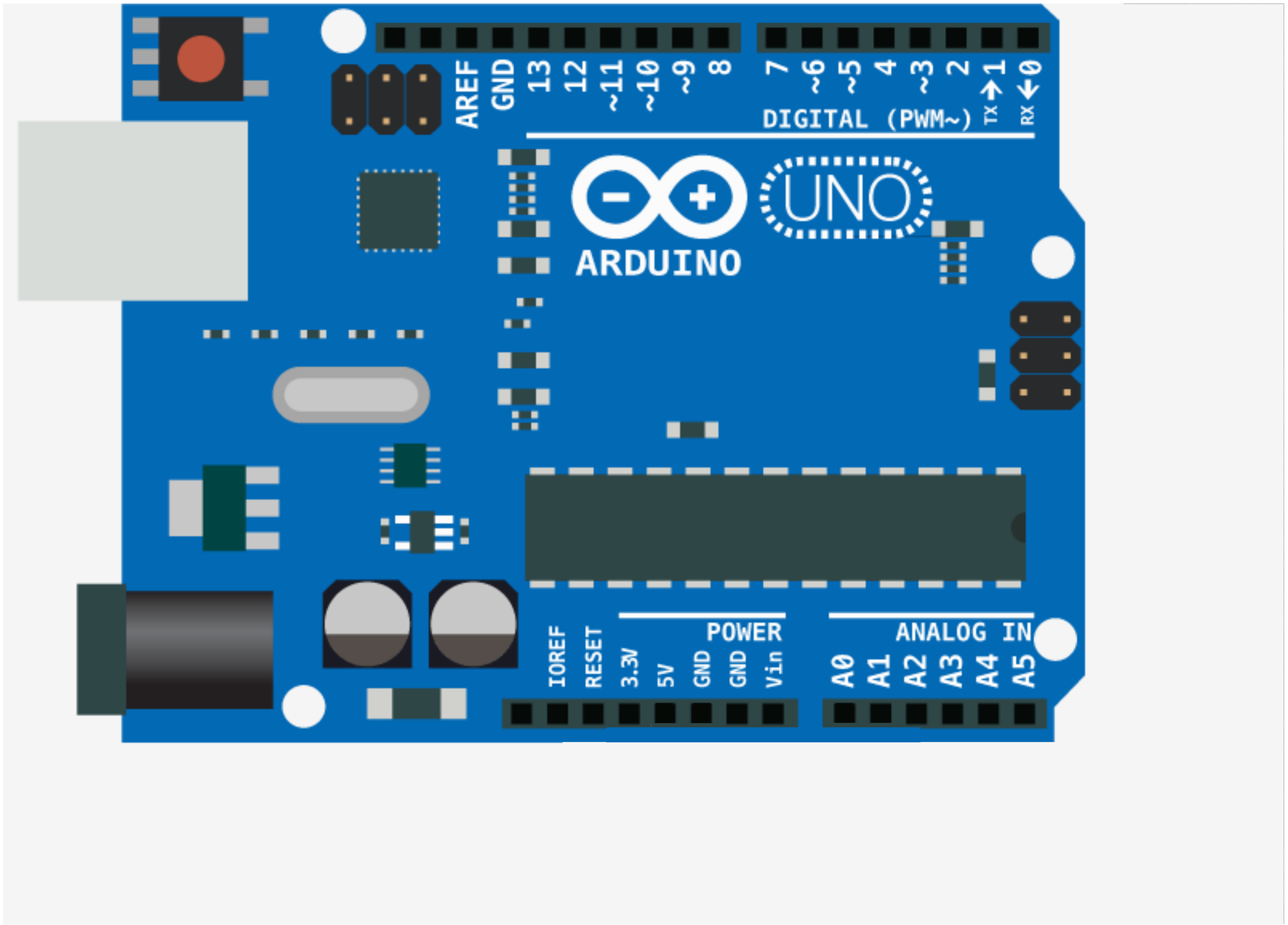
2. Learn how to program Arduino

3. Learn how to physically connect sensors and actuators

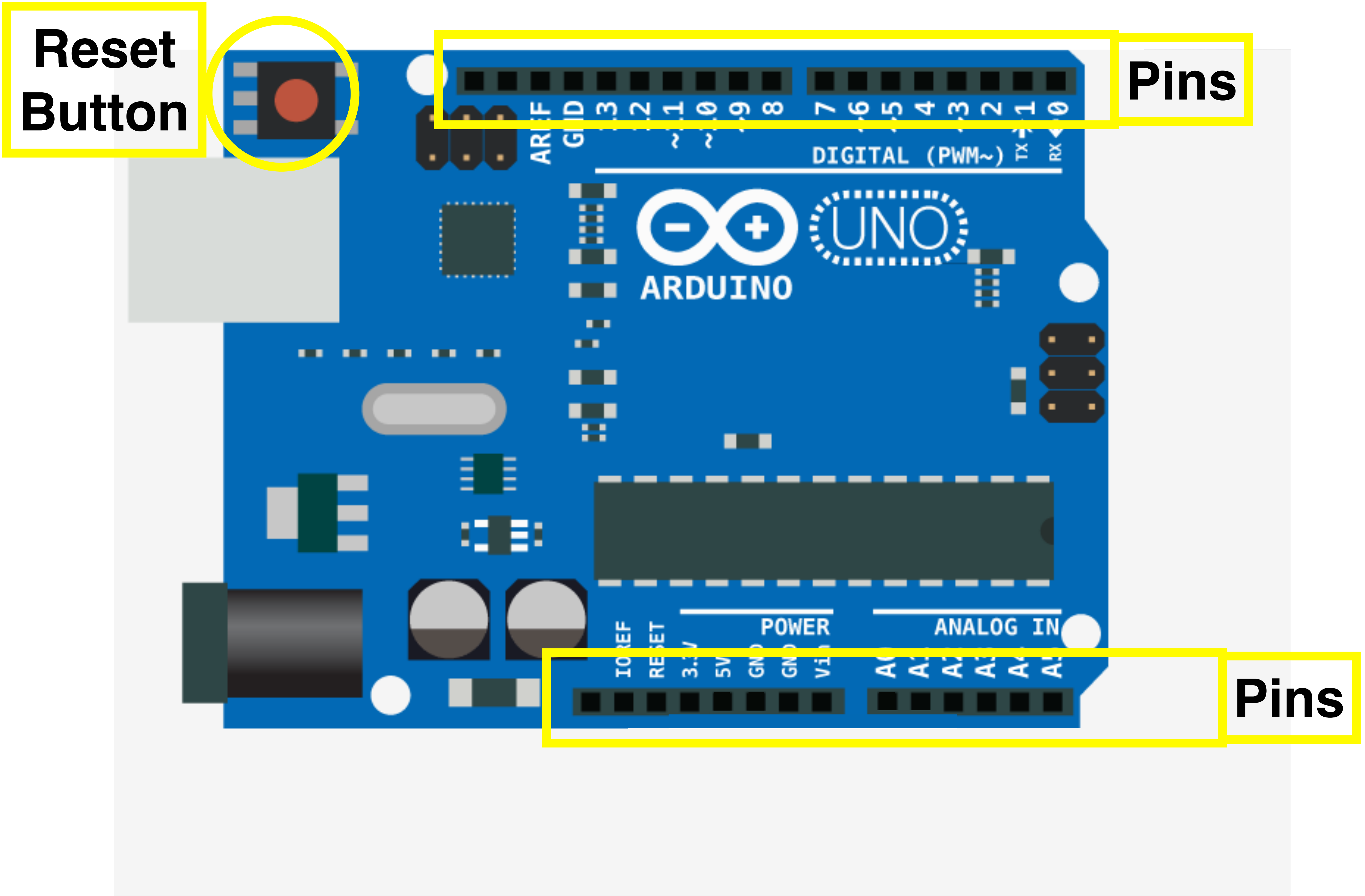
Observations

- We chose Arduino as microcontroller
- Arduino is a (fast) *prototyping tool*, not a component you want to embed into the final product (it costs too much, it is too large, it is not optimized for your product)
- This is just for us to *preview* the final product (and decide whether it is worth to produce it)

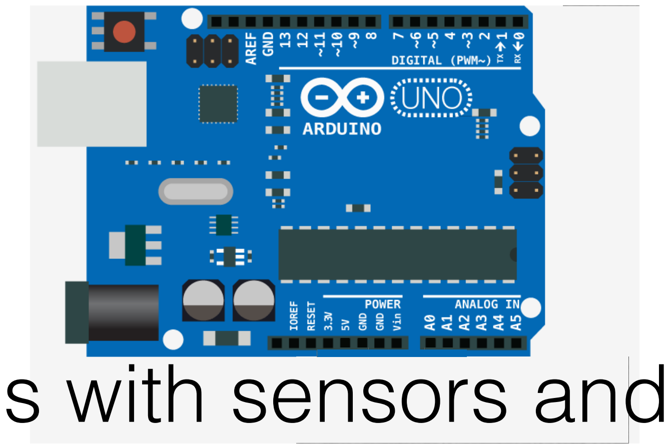
Arduino Overview



Arduino Overview



Arduino Overview



- Arduino communicates with sensors and actuators through **pins**
- You can only **write from** or **read to** pins
- We will connect actuators and sensors to pins
- Programs consist of:
  - *reading* from pins (sensing)
  - elaborating (thinking)
  - *writing* to pins (acting)

Part I: Programming

- Basics of Arduino Programming
  - Overview of Arduino programs
  - Variables, constants, assignments, comparison and logical operators
  - (pin and terminal) Input/output
  - Instructions: sequence, if-then-else, loops

Programming

- In order to work, Arduino needs to be *programmed*
- Programs define the way Arduino behaves, when and how sensors and actuators are used
- Programs tell Arduino *what to do*

Structure of Arduino Programs

- Every Arduino program consists of 2 parts:
  - `setup`: this defines the preliminary actions that Arduino needs to perform before starting the actual work
  - `loop`: this tells Arduino what to do when running

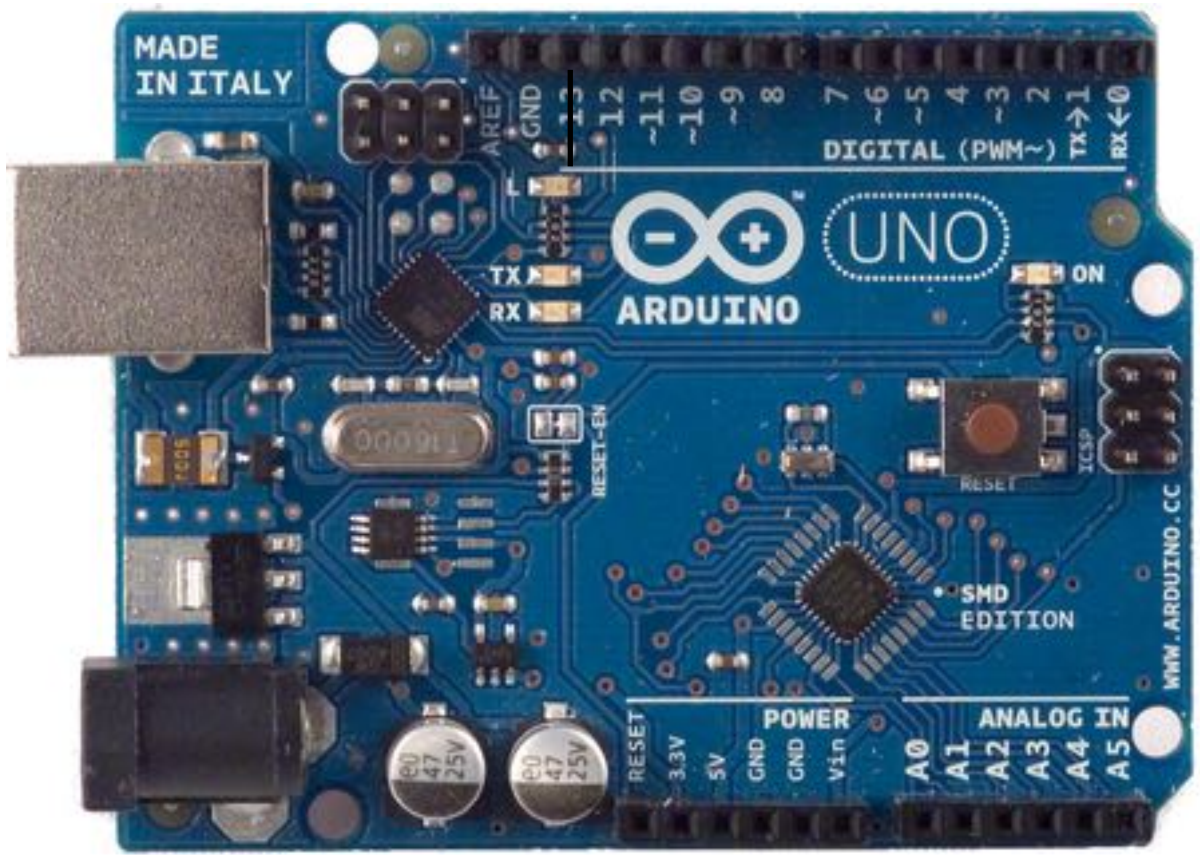
My First Arduino Program

A program that makes
a LED blink

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

void loop(){
 digitalWrite(13,HIGH);
 delay(1000);

 digitalWrite(13,LOW);
 delay(1000);
}
```

My First Arduino Program

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

void loop(){
 digitalWrite(13,HIGH);
 delay(1000);

 digitalWrite(13,LOW);
 delay(1000);
}
```



Sketch

- *Sketch* is the *programming language* of Arduino
- It is very similar to the language C (for those of you who know it)
- Let’s see the basic rules to write Sketch Programs (a.k.a. *sketches*)

loop and setup functions

Sketches consist of two ***functions***: setup and loop

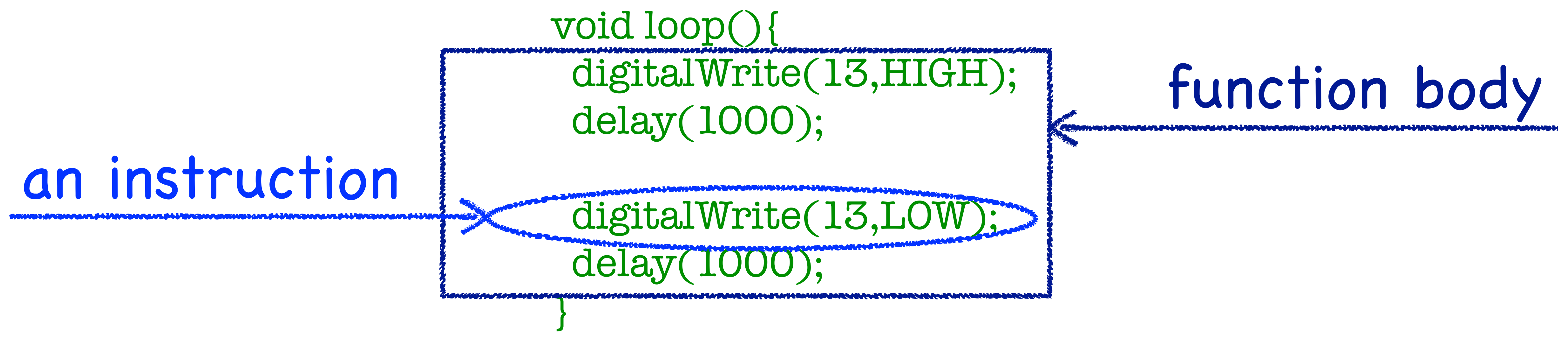
- setup: executed once, when Arduino powered or reset button pressed
- loop: executed over and over, after setup

```
void setup(){
}

void loop(){
}
```

To write your own program you need to put your *code within the curly brackets of each function*, i.e., *write the **function body***

Function body

The **function body** is a ***sequence of instructions,*** each terminating with `**;**`


```
void loop(){
  digitalWrite(13,HIGH);
  delay(1000);

  digitalWrite(13,LOW);
  delay(1000);
}
```

When the function is executed, instructions are executed in the order they occur in the body

Some basic Instructions

pinMode(pin,mode): assigns mode **mode** to pin **pin**
(**mode** can only be **INPUT** or **OUTPUT**)

digitalWrite(pin,val): writes value **val** to pin **pin**
(**pin** mode must have been set to **OUTPUT**,
val can only be **HIGH** or **LOW**)

delay(msec): waits for **msec** milliseconds

As you can see, instructions have ***parameters***

You don't need to memorize all the instructions!
Use these slides any time you need!
(This is what programmers do ;))

My First Arduino Program / 2

Do we understand this program now?

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

void loop(){
 digitalWrite(13,HIGH);
 delay(1000);

 digitalWrite(13,LOW);
 delay(1000);
}
```

Hands-on #1

Now, let’s make Arduino blink:

1. Install the Arduino software on your PC
2. Start the Arduino software
3. File -> New
4. Write the blink program
5. Connect your PC and Arduino through a USB cable
6. Press the Upload Button ➔
7. Enjoy!

Advice

Before setting up a circuit, ALWAYS disconnect the USB cable
(and any other power source)

Unless you REALLY know what you’re doing,
and ``really’’ means REALLY!!!,
don't connect Arduino to a power source other than USB

These are SAFETY advices, to protect Arduino and
YOURSELF!

Hands-on #2

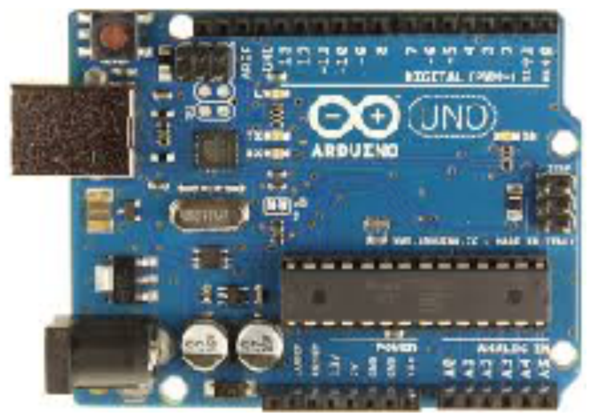
Next, we are going to make other lights blink

We are going to setup a circuit that we will use for a while

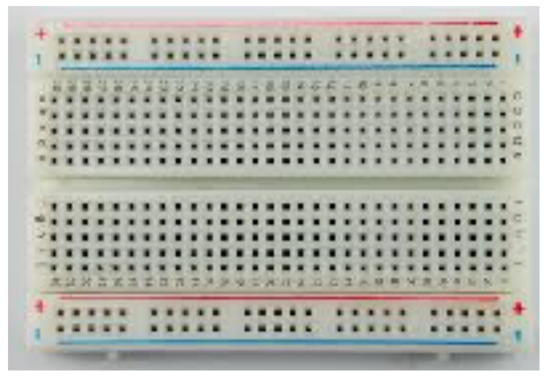
For now, you won’t understand all the circuit details but we will get to that

Hands-on #2

Common to all projects



Arduino



Breadboard

A-B USB cable

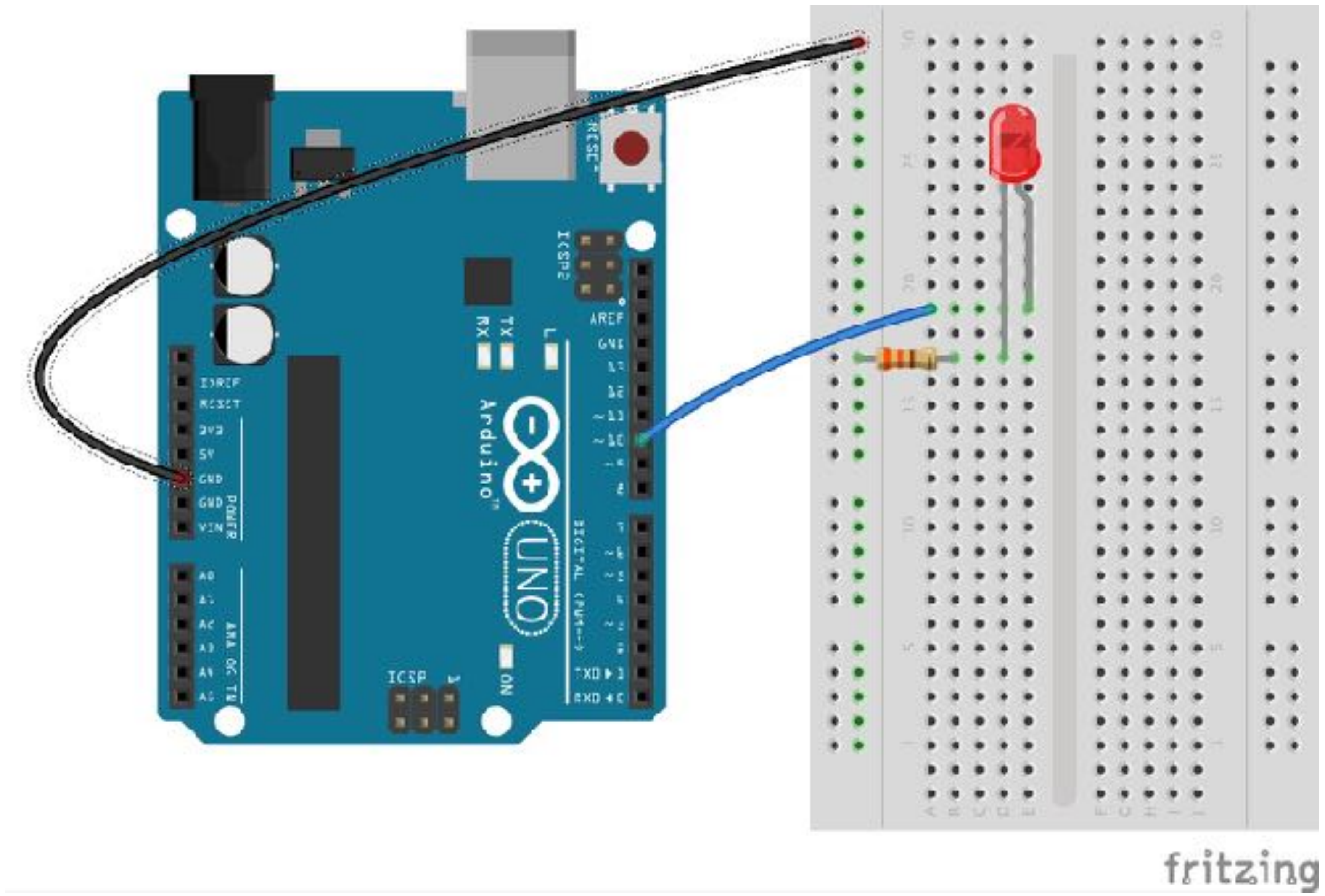
Light Emitting Diode (LED)
(Pin sizes matter)

220 Ohm Resistor
(Color stripes matter)

Jumper Wires

Hands-on #2

(ALWAYS Keep the security advices in mind!)



Hands-on #2

We still don't know much about circuits, but who built the circuit told us that:

If you set pin 10 to `HIGH`*, then current flows through the LED and…*

…well, try it yourself!

Hands-on #2

```
void setup(){
 pinMode(10,OUTPUT);
}

void loop(){
 digitalWrite(10,HIGH);
}
```

How would you modify this program to make the LED blink?

Commenting your Code

Since programs are not always as clear as the ones above (which, btw, is clear just because we have commented on it), it is useful to use **comments** inside the code:

```
void loop(){
  /*
  The sequence of characters above starts a multi-line comment
  This is a multi-line comment. Anything you write until the end
  of the comment is ignored when the program runs
  The sequence of characters below closes the comment
  */

 // Everything until the end of a line is a comment
}
```

Comments have **no impact** on program execution

Comments are for you and for who reads your code!

``Official'' version of the blink program:

```
/*
 Blink
 Turns on an LED on for one second, then off for one second, repeatedly.

 Most Arduinos have an on-board LED you can control. On the Uno and
 Leonardo, it is attached to digital pin 13. If you're unsure what
 pin the on-board LED is connected to on your Arduino model, check
 the documentation at http://arduino.cc

 This example code is in the public domain.

 modified 8 May 2014
 by Scott Fitzgerald
 */

// the setup function runs once when you press reset or power the board
void setup() {
 // initialize digital pin 13 as an output.
 pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
 digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
 delay(1000);              // wait for a second
 digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
 delay(1000);              // wait for a second
}
```

We will be using comments all the time!

Variables

Typically, programs need to *store values*

For instance, suppose after every blink, we want to increase the delay by .5 secs

To do so, we *need to record* how long the LED has been on at previous step

To store and retrieve values, Sketch (like all programming languages) offers a structure called **variable**

A variable can be thought of as ***a box containing a value***

Variables

```
int t;  //Variable declaration

void setup(){
 t = 0;  //Variable initialization: now t contains the value 0
  pinMode(10,OUTPUT);
}

void loop(){
  t = t + 500; // New assignment: the value of t is increased by 500
  digitalWrite(10,HIGH);
  delay(t);  //Value retrieval

  digitalWrite(10,LOW);
  delay(t);
}
```

Observations

- Variables must be *declared* before use:
```
int t;
```

- Sketch variables have *a type*, the type of values they store, e.g.:
  `int`: integer

- To change a variable's value you use the assignment operator =, e.g.,:
```
t = 0;
```

- To access a variable's value, you just use the variable as if it were the value, e.g.,:
```
t = t + 500;
delay(t);
```

Visibility of Variables

- In the example, `t` was declared outside the body of functions
    - In this case, it visible to all functions. It is called a **global variable**

- You can also declare a variable inside a function body (or a block)
    - In this case, it is visible only to the function (or block) it was declared in
    - It is called a ***local variable***

- The rule is that *a variable is visible to the block (and sub-blocks) it is declared in*

Constants

- Constants are similar to variables, but *can be assigned a value only when declared*, e.g.:

  ```
  const int INCREMENT = 500;
  ```

  (`INCREMENT` is an integer constant containing the value `500`)

- Visibility rules for constants are the same as for variables

- It is a (stylistic) convention that constant names be capitalized
  (and we will stick to it!)

Example

```
// Constant declarations and initialization (notice capitalization):
const int PIN = 10;
const int INCREMENT = 500;

int t = 0;  //Variable declaration and initialization

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

void loop(){
  t = t + INCREMENT; // New assignment: the value of t is increased by INCREMENT
  digitalWrite(PIN,HIGH);
  delay(t);  //Value retrieval

  digitalWrite(PIN,LOW);
  delay(t);
}
```

Observations

- We have already encountered some constants (guess which ones)

- Constants are desirable because they give a meaning to numbers:
  - `t = t + 500; // What is 500????`
  - `t = t + INCREMENT; // The value by which you want to increase t!`

- Constants are desirable because they make changes easier:
  - Imagine you want to use pin 14 instead of 10
  - Compare which changes are required in the two program versions (with and without constants)

- Whenever you have the choice, using a constant is better than not!

Program Structure Revisited

The previous observations suggest the following *good practice of* program organization:

```
// Global constant declarations and initialization (OPTIONAL)

// Global variable declarations and initialization (OPTIONAL)

// setup function (MANDATORY):
void setup(){
/* YOUR CODE
HERE */}

// loop function (MANDATORY):
void loop(){
/* YOUR CODE
HERE */}
```

Hands-on #3

Write and execute a program that makes the LED (on pin 10) blink according to the following rules:

- Initially, the LED alternates 1 sec on and 2 secs off
- After every blink, on and off times are swapped
  (i.e., at the second iteration, the LED will be 2 secs on and 1 sec off, then 1 sec on and 2 off, then 2 secs on and 1 off, and so on)

```
// Global constant declarations
const int PIN = 10; // output pin
const int T1 = 1000; // initial on-time
const int T2 = 2000; // initial off-time

// Global variable declarations
int t_on = T1; // on-time variable declaration and initialization
int t_off = T2;// off-time variable declaration and initialization

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

void loop(){
 digitalWrite(PIN,HIGH);
 delay(t_on);

 digitalWrite(PIN,LOW);
 delay(t_off);

 int aux = t_on; // Using local variable aux to swap t_on and t_off
 t_on = t_off;
 t_off = aux;
}
```

digitalRead

- We have already used the instruction `digitalWrite`
- It can be used to write either `HIGH` or `LOW` on a pin

- We can also read values from a pin
- For `HIGH` or `LOW` we can use the instruction `digitalRead`

- `digitalRead(pin)`: *returns* the value read on pin `pin` (`pin` mode must be `INPUT`, return value is either `HIGH` or `LOW`)
- The returned value can be stored in a variable to be used

- Let see how it works…

Hands-on #4

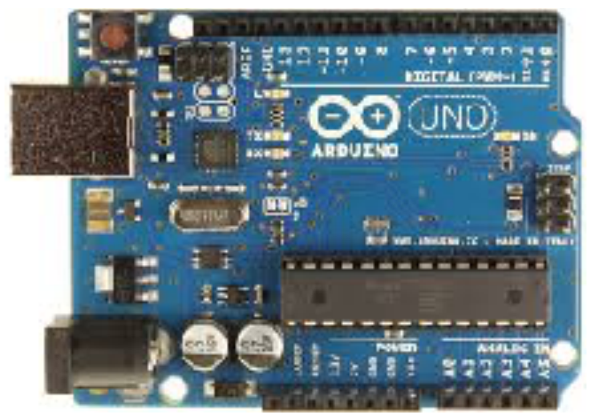
We are now going to build our first *interactive* device!

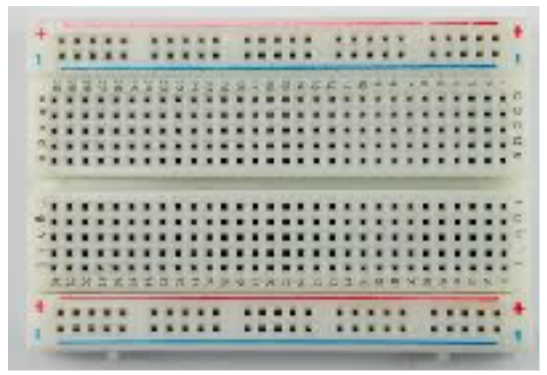
We will switch the LED on whenever a button is pressed

A button is a very simple *contact sensor*

Hands-on #4

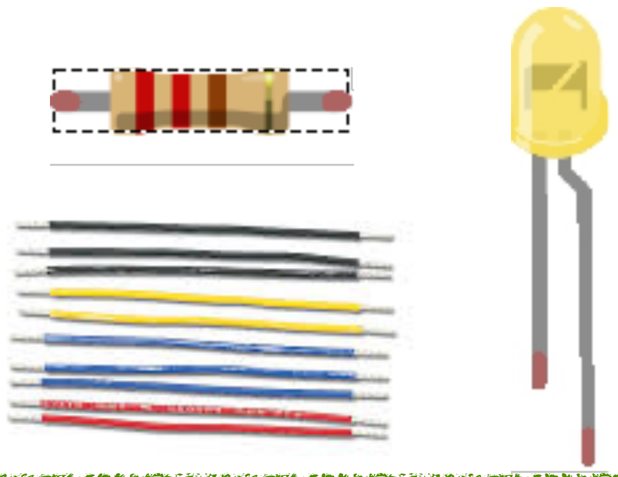
Common to all projects


Arduino

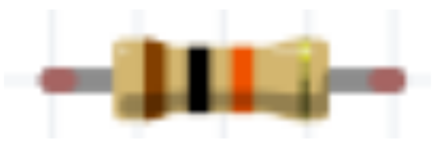
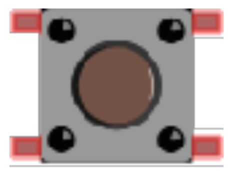

Breadboard

A-B USB cable

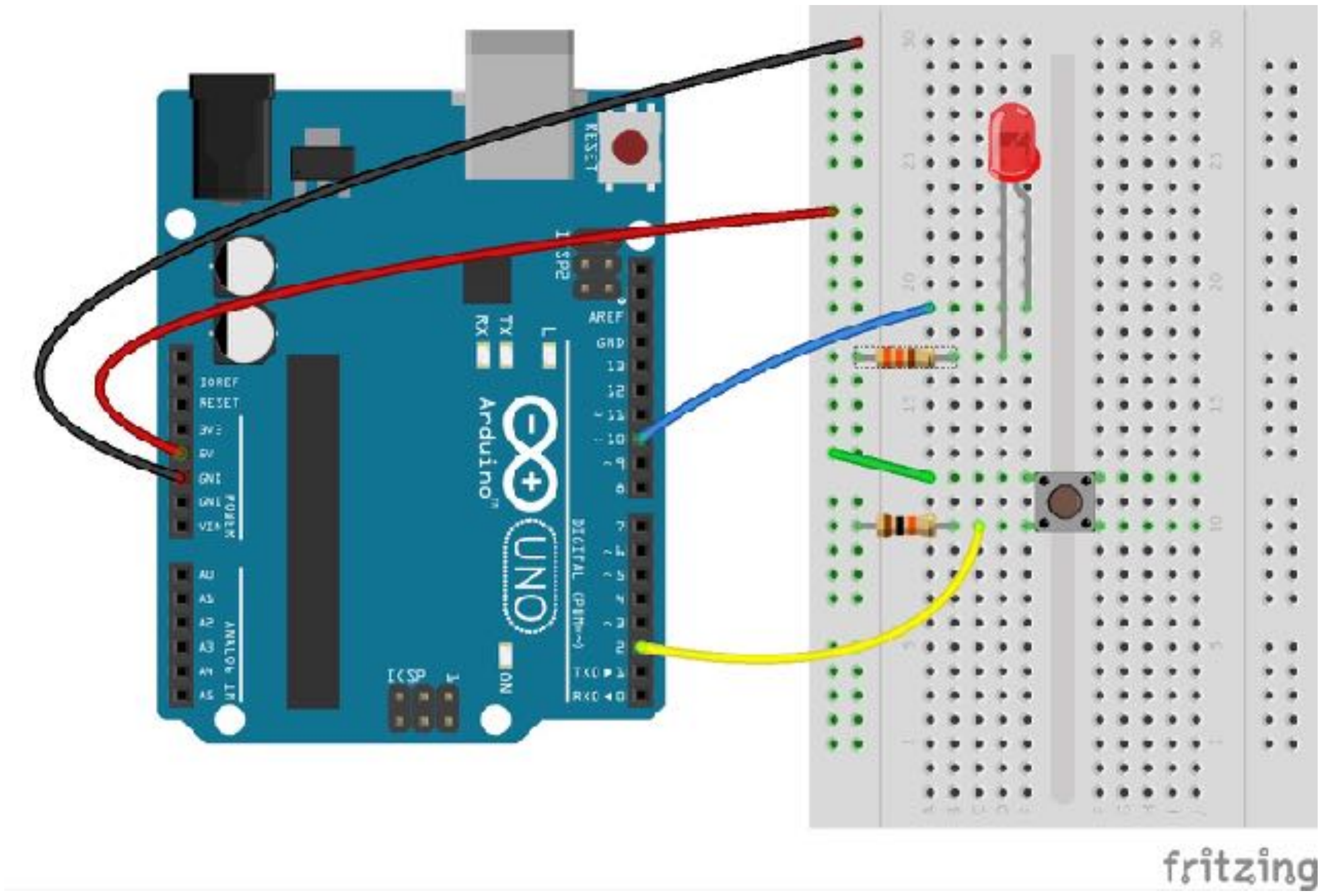
We have these



These are new

Hands-on #4



Hands-on #4

```
// Global constant declarations
const int OUT_PIN = 10; // output pin
const int IN_PIN = 2; // input pin


void setup(){
 pinMode(OUT_PIN,OUTPUT);
 pinMode(IN_PIN,INPUT);
}

void loop(){
 int val = digitalRead(IN_PIN);
 digitalWrite(OUT_PIN,val);
}
```

Observations

```
// Global constant declarations
const int OUT_PIN = 10; // output pin
const int IN_PIN = 2; // input pin


void setup(){
  pinMode(OUT_PIN,OUTPUT);
  pinMode(IN_PIN,INPUT);
}

void loop(){
  int val = digitalRead(IN_PIN);
  digitalWrite(OUT_PIN,val);
}
```

Global constants
(accessed by both functions)

local variable (used only by loop)

digitalRead reads the value on PIN 10
(HIGH if button pushed)

Did we buy an Arduino to push a button and switch a light on????????

Boohoo!!!!

OK, we can do better, but we need further instructions

Example

- Consider a variant of Hands-on #4
- We want the LED to remain on 3 seconds when the button is pressed
- What do we need?

Example/2

Ideally, we need a program like this:

```
void loop(){
   /* if button is pressed
            then switch LED on for 3 seconds
   */
}
```

We already know how to switch the LED on

Unfortunately, we don’t know how to check whether the button is pressed

`if-then-else`

- The `if-then-else` instruction allows us to:
  - test a condition, and
  - if the condition is true, execute some instructions
  - if the condition is false, execute some other instructions

if-then-else / 2

```
if (<condition>){
  /* <if-branch>:
          mandatory, executed if <condition> is true
  */
}
else{
  /*
       <else-branch>
          optional, if present, executed if <condition> is false
*/
}
```

Example

```
// Global constant declarations
const int OUT_PIN = 10; // output pin
const int IN_PIN = 2; // input pin

void setup(){
 pinMode(OUT_PIN,OUTPUT);
 pinMode(IN_PIN,INPUT);
}

void loop(){
 int val = digitalRead(IN_PIN); //Reads value on PIN 2

 if (val == HIGH){
  digitalWrite(OUT_PIN,HIGH); //switches LED on
  delay(3000);
  digitalWrite(OUT_PIN,LOW); //switches LED off
 }
}
```

Observations

Condition

```
if (val == HIGH) {
   digitalWrite(OUT_PIN,HIGH); //switches LED on
   delay(3000);
   digitalWrite(OUT_PIN,LOW); //switches LED off
   delay(3000);
  }
```

If-branch

Else-branch not present in this example

Hands-on #5

Using the same circuit as that of hands-on #4,

write a program that makes the LED blink 3 times, whenever the button is pressed

Hands-on #5

```
// Constant declarations and setup function same as before

void loop(){
 int val = digitalRead(IN_PIN); //Reads value on PIN 10
 if (val == HIGH){
  digitalWrite(OUT_PIN,HIGH);
  delay(500);
  digitalWrite(OUT_PIN,LOW);
  delay(500);
  digitalWrite(OUT_PIN,HIGH);
  delay(500);
  digitalWrite(OUT_PIN,LOW);
  delay(500);
  digitalWrite(OUT_PIN,HIGH);
  delay(500);
  digitalWrite(OUT_PIN,LOW);
  delay(500);
 }
}
```

The else-branch

Imagine you want the LED (on pin 10) blink according to the following rules:

- Initially, the LED alternates .5 sec on and .5 sec off
- After every blink, on- and off-times are decreased by .025 sec
- When .025 is reached, times are reset to .5 sec (after blinking)

How would you write your sketch?

```
// Global constant and variable declarations
const int OUT_PIN = 10; // output pin
const int INIT_DELAY = 500; // initial delay
const int DECREMENT = 25; // time decrement
int t; // current delay

void setup(){
 pinMode(OUT_PIN,OUTPUT); // set pin as output
 t = INIT_DELAY; // initialize current delay
}

void loop(){
 // Make the led blink
 digitalWrite(OUT_PIN,HIGH); // on
 delay(t); // wait
 digitalWrite(OUT_PIN,LOW); // off
 delay(t); // wait

 // Set the delay
 if(t == DECREMENT){
  t = INIT_DELAY; // reset delay
 }
 else{
  t = t - DECREMENT; // decrease wait time
 }
}
```

Conditions

- A condition represents some property of a program in execution
- E.g., `val == HIGH` represents the fact that variable `val` is assigned value `HIGH` (notice the use of `==` instead of `=`)
- Conditions can either be **true** or **false** (this matters, e.g., when the condition occurs in an if-then-else instruction)
- To write conditions, we need to know the ***language of conditions***

- Conditions can be built in various way. We will consider the following:
  - Comparison of a variable against another variable, constant, or value, e.g.:
    - `val == 8` (the value of variable `val` equals `8`)
    - `val != IN_PIN` (the value of variable `val` is different than the value of constant `IN_PIN`)
    - `val > x` (the value of variable `val` is greater than that of variable `x`)
    - `val <= 9` (the value of variable `val` is less than or equal to `9`)
    - also `>=` (greater or equal), `<` (less than) available
  - Combination of above comparisons through logical operators && (and), || (or), ! (not):
    - `(val >= 8) && (val != 9)`
    - `(val != 10) && (val <= 5)`
    - `!((val > 8) || (val == 10))`

Don't worry: you'll learn with practice!

Hands-on #6

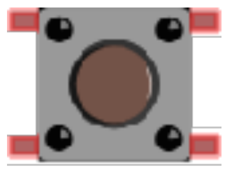
Add one button to the circuit used in hands-on #5 (and #4)

Then, write a sketch such that:

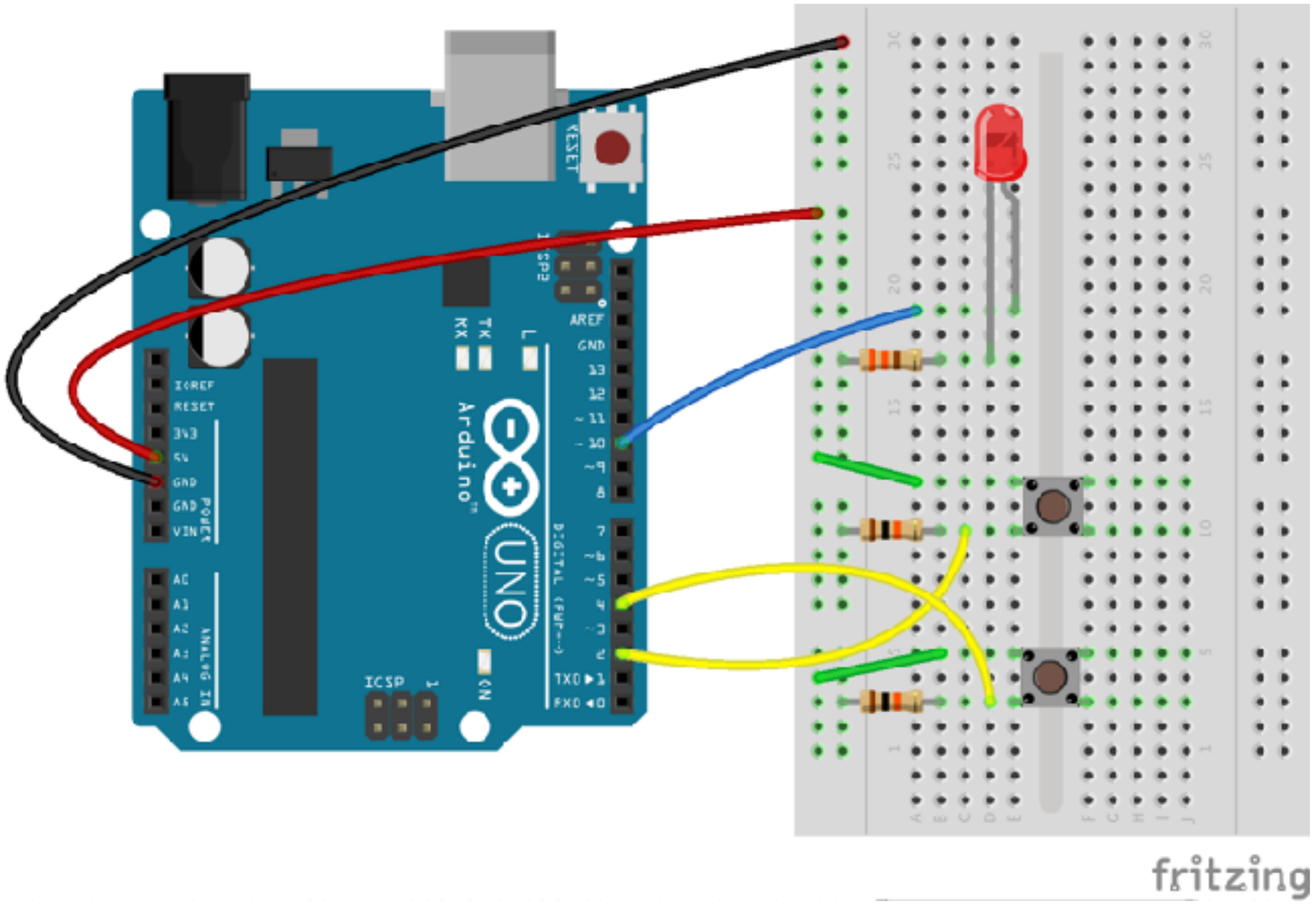
the LED is always on except when both buttons are pressed

Hands-on #6

You need another set of these



Hands-on #6



Hands-on #6

```
// Global constant declarations
const int BUTTON1_PIN = 2;
const int BUTTON2_PIN = 4;
const int LED_PIN = 10;

void setup(){
 pinMode(BUTTON1_PIN,INPUT);
 pinMode(BUTTON2_PIN,INPUT);
 pinMode(LED_PIN,OUTPUT);
}

void loop(){
 int b1 = digitalRead(BUTTON1_PIN);
 int b2 = digitalRead(BUTTON2_PIN);

 if ((b1 == HIGH) && (b2 == HIGH)){
  digitalWrite(LED_PIN,LOW);
 }
 else{
  digitalWrite(LED_PIN,HIGH);
 }
}
```

Loops

- Loops allow for iterating over a code block
- Useful when one needs to execute the same instructions many times (possibly on different variables, pins, etc.)

Example

- Imagine you need to set the mode of all digital pins to OUTPUT
- How would you write the `setup` function?

Example

```
void setup(){
 pinMode(0,OUTPUT);
 pinMode(1,OUTPUT);
 pinMode(2,OUTPUT);
 pinMode(3,OUTPUT);
 pinMode(4,OUTPUT);
 pinMode(5,OUTPUT);
 pinMode(6,OUTPUT);
 pinMode(7,OUTPUT);
 pinMode(8,OUTPUT);
 pinMode(9,OUTPUT);
 pinMode(10,OUTPUT);
 pinMode(11,OUTPUT);
 pinMode(12,OUTPUT);
 pinMode(13,OUTPUT);
}
```

while loop

The previous example can be conveniently written as follows, using the while instruction:

```
int i = 0;
while (i <= 13){
    pinMode(i,OUTPUT);
    i = i +1;
  }
```

while loop

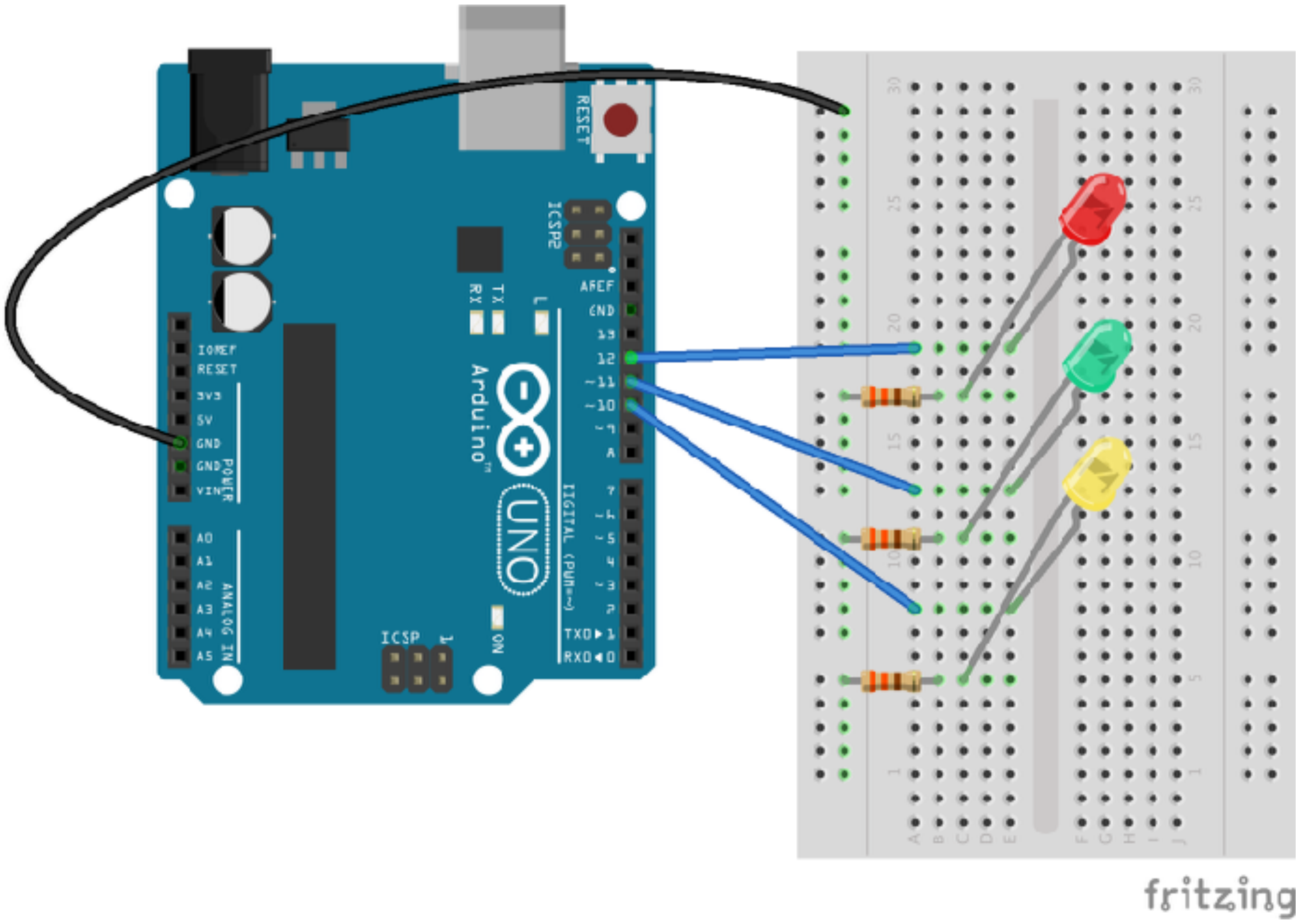
Condition

```
int i = 0;
while (i <= 13){
    pinMode(i,OUTPUT);
    i = i +1;
  }
```

Block

- Condition is evaluated:
  - if **true**:
    - Block is executed
    - loop is repeated
  - if **false**: loop exits

Hands-on #7



RESET
ICSP2
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX▶1
RX◀0
DIGITAL (PWM=~)
TX
RX
L
Arduino
UNO
ON
ICSP
IOREF
RESET
3V3
5V
GND
GND
VIN
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN
fritzing

Hands-on #7

- Write a sketch that:
- switches the led on pin 10 on for .2 secs, then
- switches the led on pin 10 off and switches the led on pin 11 on for .2 secs, then
- switches the led on pin 11 off and switches the led on pin 12 on for .2 secs, then switches the led on pin 12 off and repeats

Hands-on #7

```
void setup(){
 int i = 10;
 while (i <= 12){
  pinMode(i,OUTPUT);
  i = i+1;
 }
}

void loop(){
 int i = 10;
 while (i <= 12){
  digitalWrite(i,HIGH);
  delay(200);
  digitalWrite(i,LOW);
  i = i+1;
 }
}
```

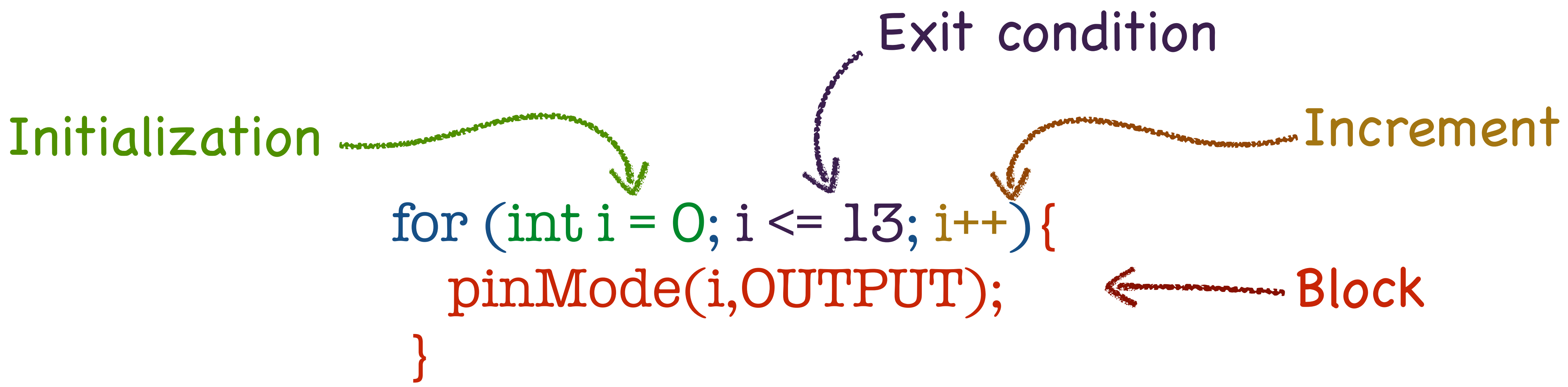
for loop

We can also use the **for** instruction:

```
for (int i = 0; i <= 13; i++){
  pinMode(i,OUTPUT);
}
```

Note: `i++` is used as a shortcut for `i = i + 1`

for loop



```
for (int i = 0; i <= 13; i++){
  pinMode(i,OUTPUT);
 }
```

1. **Initialization** is executed
2. **Exit condition** is evaluated:
   - if **true**:
     - **Block** is executed
     - **Increment** is executed
     - 2. is repeated
   - if **false**: loop exits

Hands-on #8

- Rewrite the sketch of hands-on #7 using instruction `for` instead of `while`

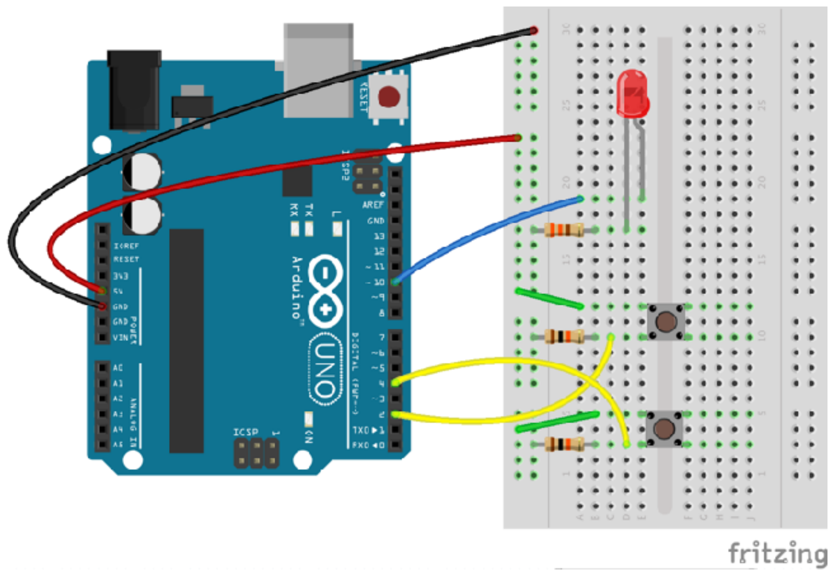
Hands-on #8

```
void setup(){
 for(int i = 10; i <= 12; i=i+1){
  pinMode(i,OUTPUT);
 }
}

void loop(){
 for(int i = 10; i <= 12; i=i+1){
  digitalWrite(i,HIGH);
  delay(100);
  digitalWrite(i,LOW);
 }
}
```

Textual output

- Arduino can also output text to the PC connected via USB
- Textual output is useful to keep track of program execution
- The text can be read on a terminal (Tools -> Serial Monitor on the Arduino Software)

Example

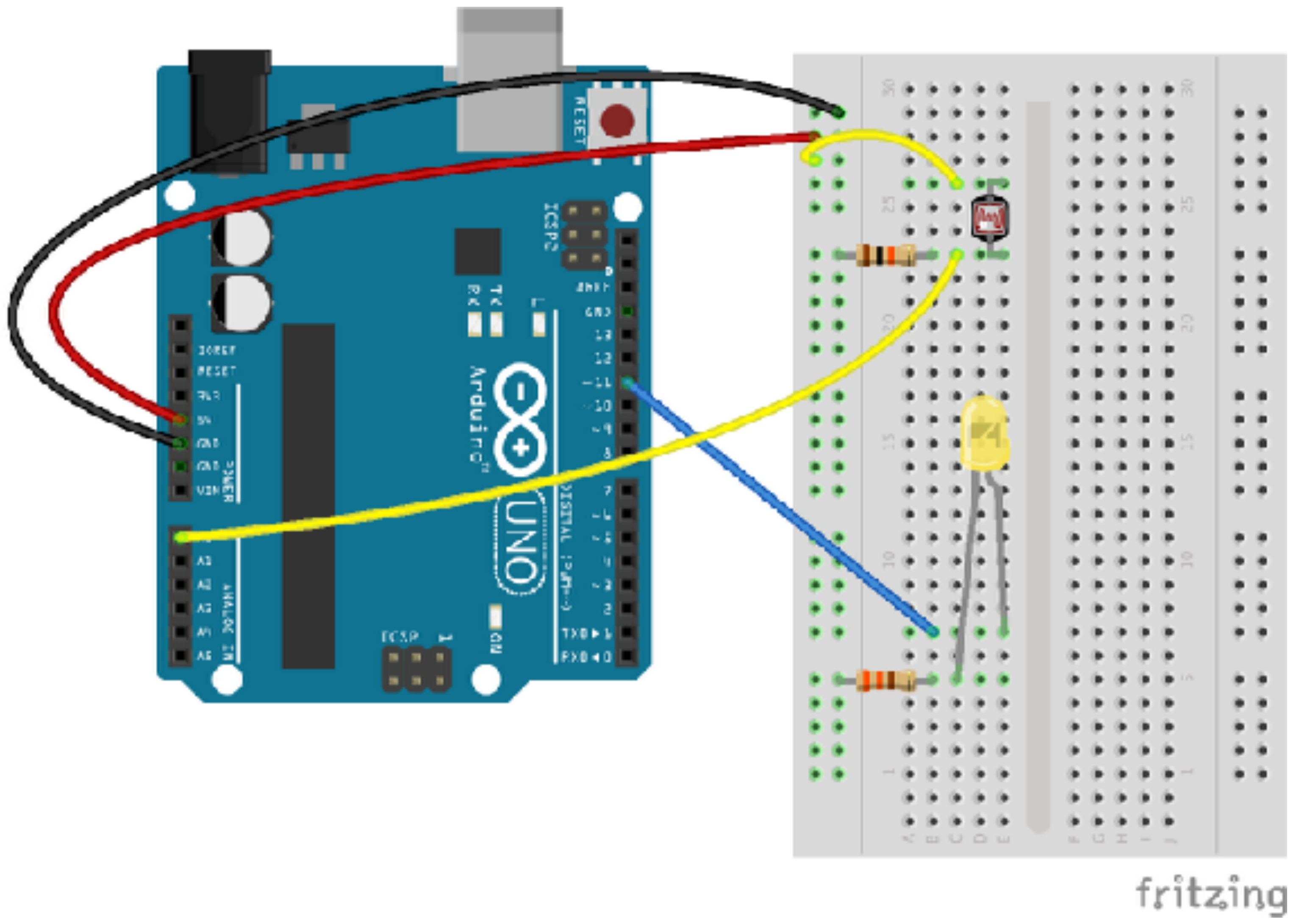


```
void setup(){
 Serial.begin(9600); //set transmission rate
}

void loop(){
 if (digitalRead(4) == HIGH){
  Serial.print("Button pressed!"); // write to terminal
  Serial.println(); // write end of line
  Serial.println("Button pressed!"); // write to terminal + end of line
}
```

`analogRead`, `analogWrite`

- `digitalRead` and `digitalWrite` allow us to read/write digital (`HIGH` or `LOW`) input/output
- Sometimes, we need to read/write values on a scale (e.g., light intensity, noise volume, etc.)
- For this we can use `analogRead`/`analogWrite`

Hands-on #9



Hands-on #9

- We are now going to change the light intensity of an LED, based on the amount of light in the environment
- We will use a light sensor, called *photoresistor* or *light-dependent resistor* (LDR)
- The light intensity of the LED will change based on the light intensity on the LDR

Hands-on #9

```
const int SENSOR = A0;
const int LED = 11;

void setup(){
 pinMode(LED,OUTPUT);
 /* NOTE:
  *     - A0 is only input and doesn't need setup
  */
}

void loop(){
 int input_light = analogRead(SENSOR); // analogRead: 0 - 1023
 analogWrite(LED, input_light / 4); // analogWrite: 0 - 255
}
```

Observations

- analogRead returns on scale 0-1023
- analogWrite writes on a scale 0 - 255
- Need to scale value for LED

```
void loop(){
 int input_light = analogRead(SENSOR); // analogRead: 0 - 1023
 analogWrite(LED, input_light / 4); // analogWrite: 0 - 255
}
```

Hands-on #10

- Program Arduino so that the LED of the circuit of hands-on #9 reduces its intensity as the environment is more illuminated, and viceversa.

Hands-on #10

```
const int SENSOR = A0;
const int LED = 11;
const int MAX_LIGHT = 1023;

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

void loop(){
 int input_light = analogRead(SENSOR);
 analogWrite(LED, (MAX_LIGHT-input_light)/4);
}
```

Powering Arduino

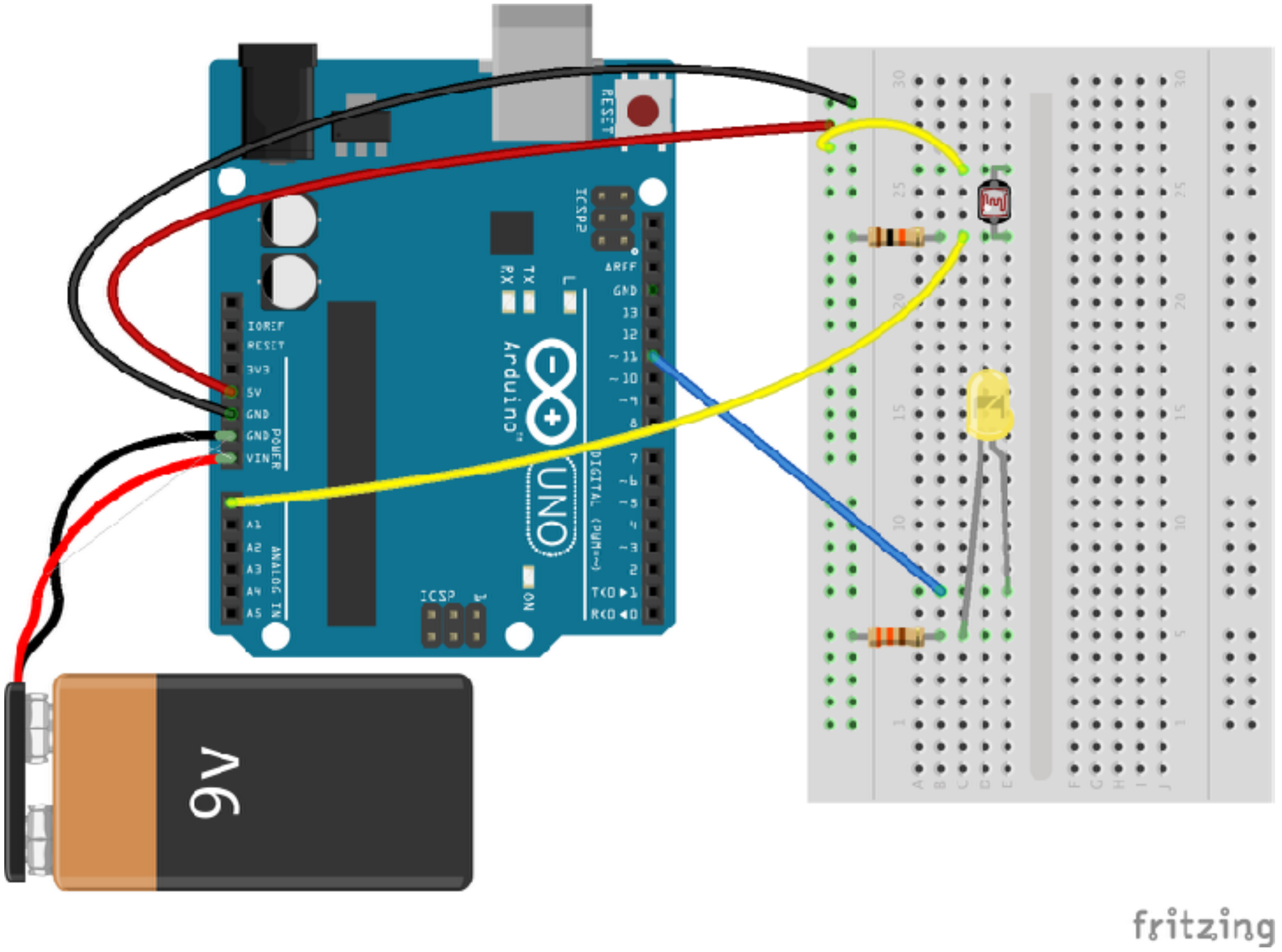
- In most cases, you will need to use Arduino without connecting to a PC
- Once the sketch you want to execute is uploaded on Arduino, you can unplug the cable and run Arduino
- However, you need to *attach Arduino to a power source*

Powering Arduino

Two ways:

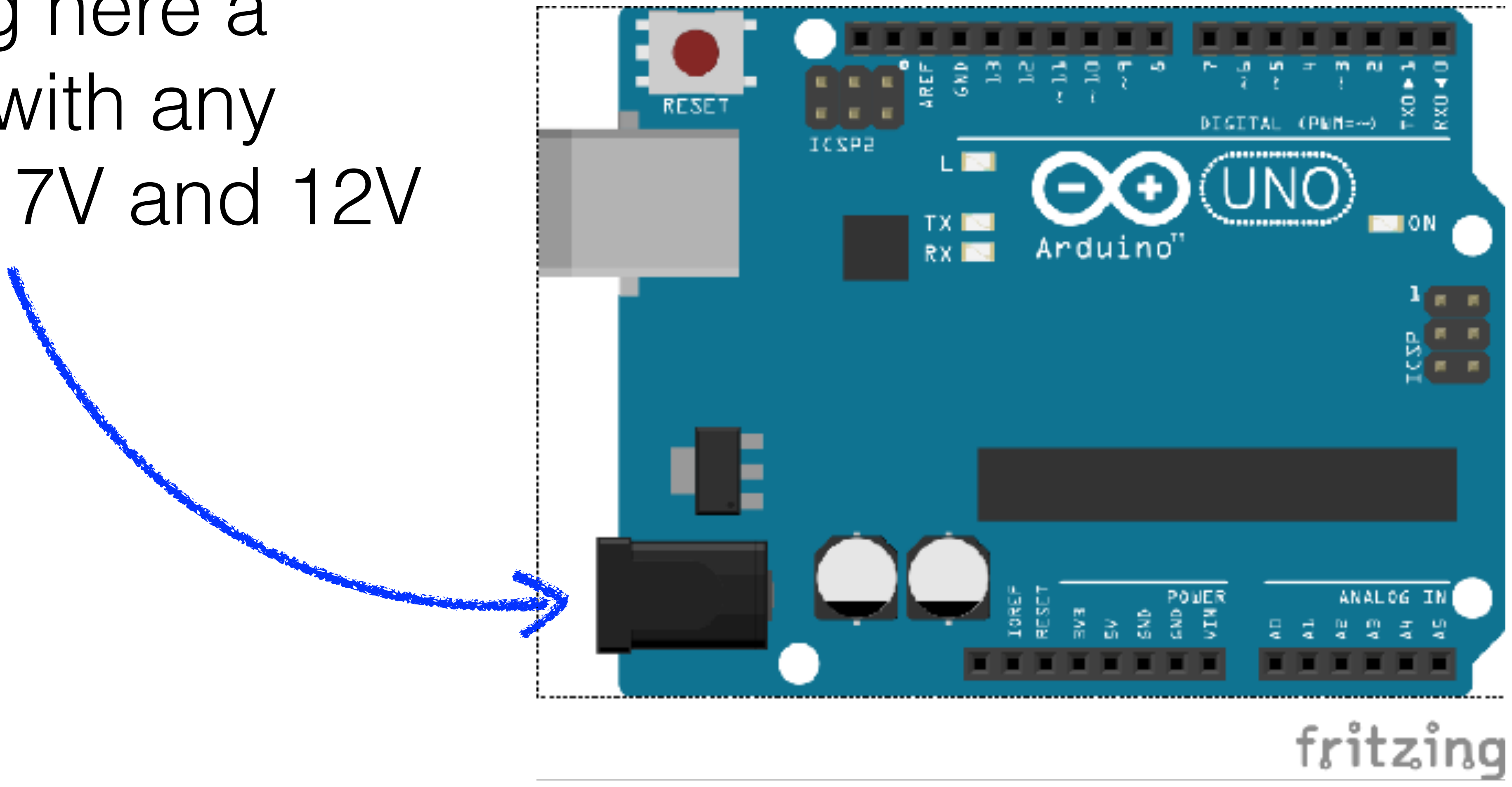
- Battery
- Adapter (not portable, but useful in some cases)

Powering Arduino: battery



Powering Arduino: adapter

You can plug here a
DC adapter with any
voltage between 7V and 12V

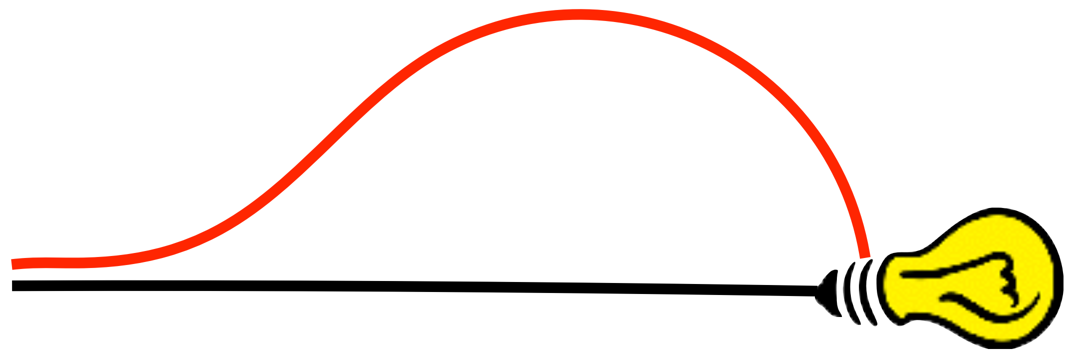


Electric Circuits

- Electric circuits are networks of electric components

- To work, electric circuits need electric current flowing through them

Example



Electric Current

- Electric current: *flow of electric charges*
- Think of electric current as particles flowing in a conductor (metal) wire
- Sometime easier to think about water in a pipe (not accurate but helpful)

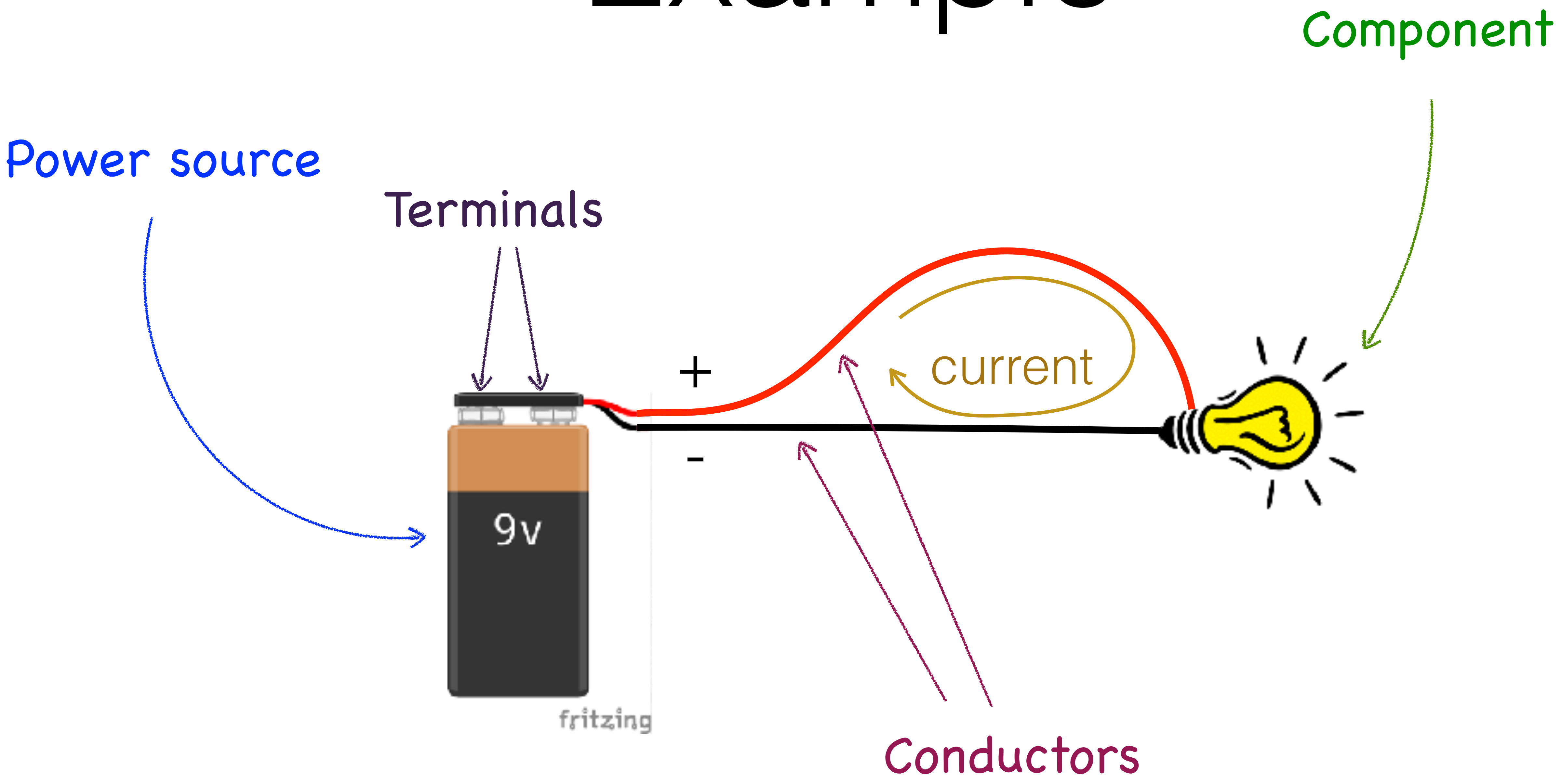
Electric Current

- We don't need to know what it is, but:
  - ***what we can do with it***
  - ***how we can deal with it***
- For this, we need **some basics**

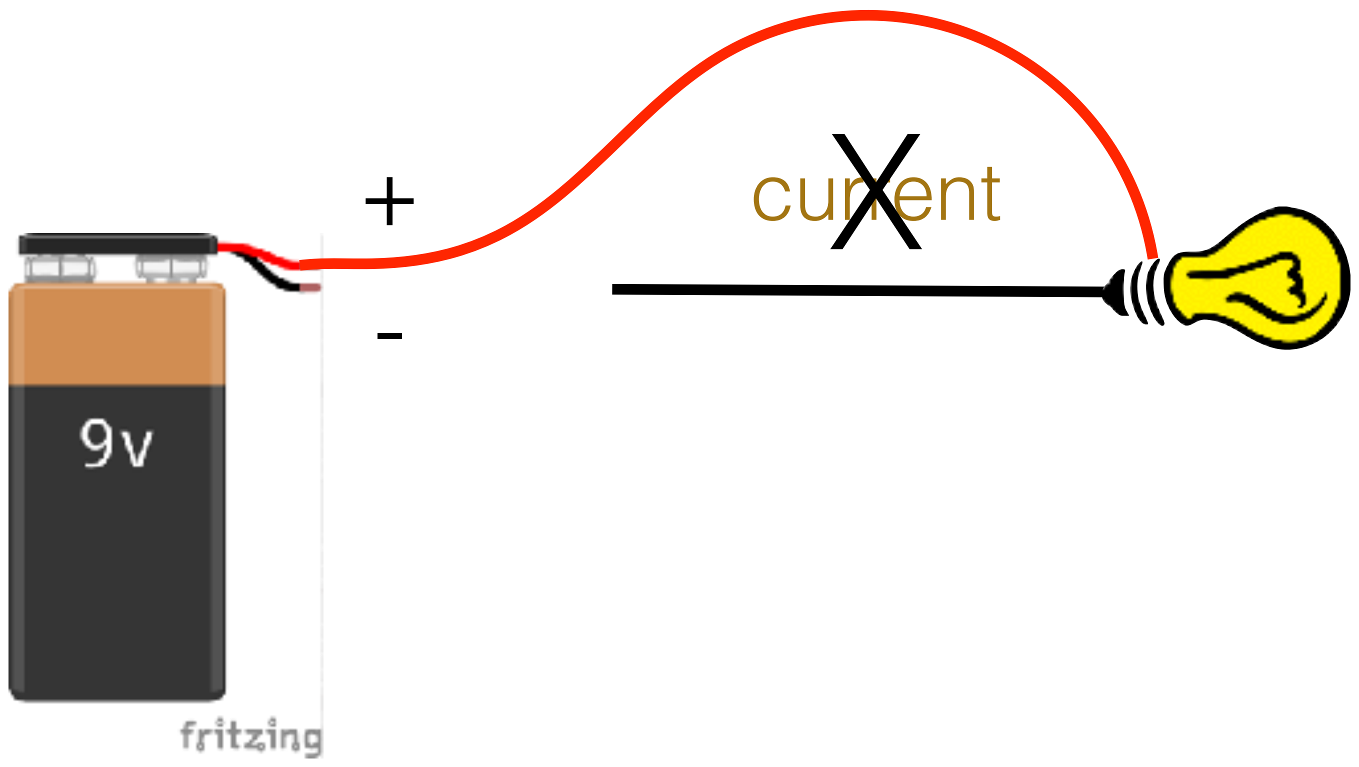
Electric Current

- *Power source* (battery, adapter, etc.) provides *two terminals*:
  - Positive (+)
  - Negative (-, *ground*)
- When the circuit is *closed*, current flows from + to -
- When *current flows* through a component (e.g., a light bulb), the component is *activated*

Example


Component
Power source
Terminals
+
-
current
9v
fritzing
Conductors

Example



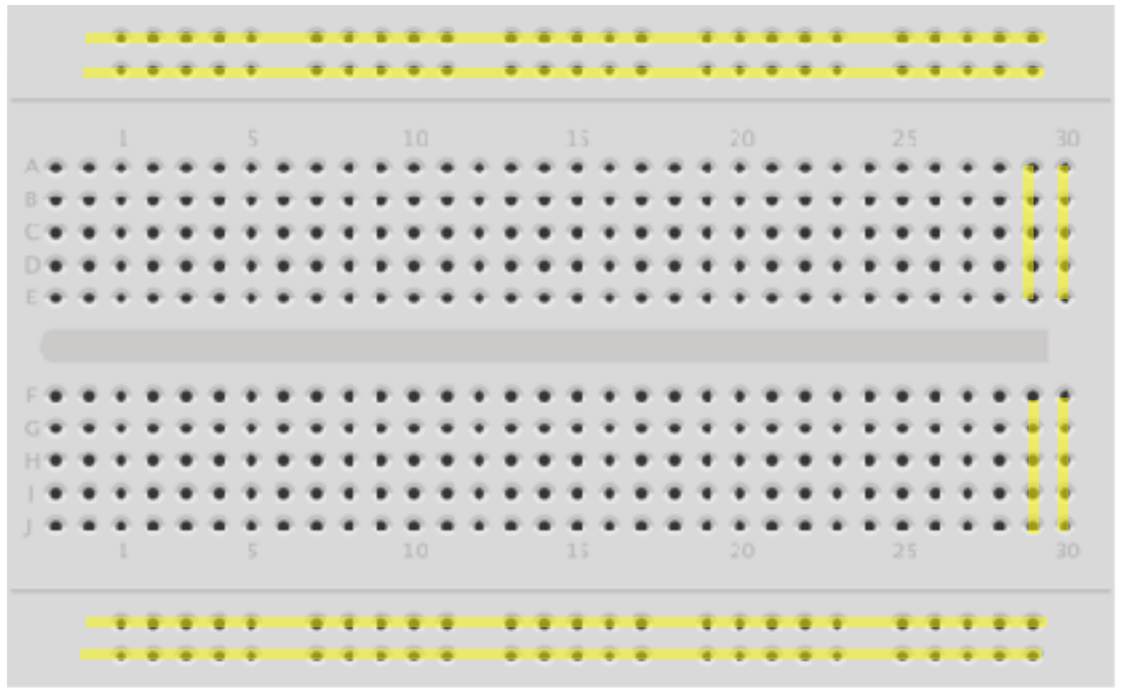
Breadboard

- Circuits are typically built by soldering components and cannot be reconfigured
- Breadboards allow for quick&dirty circuit realization and are *reconfigurable* (this is why we use them!)

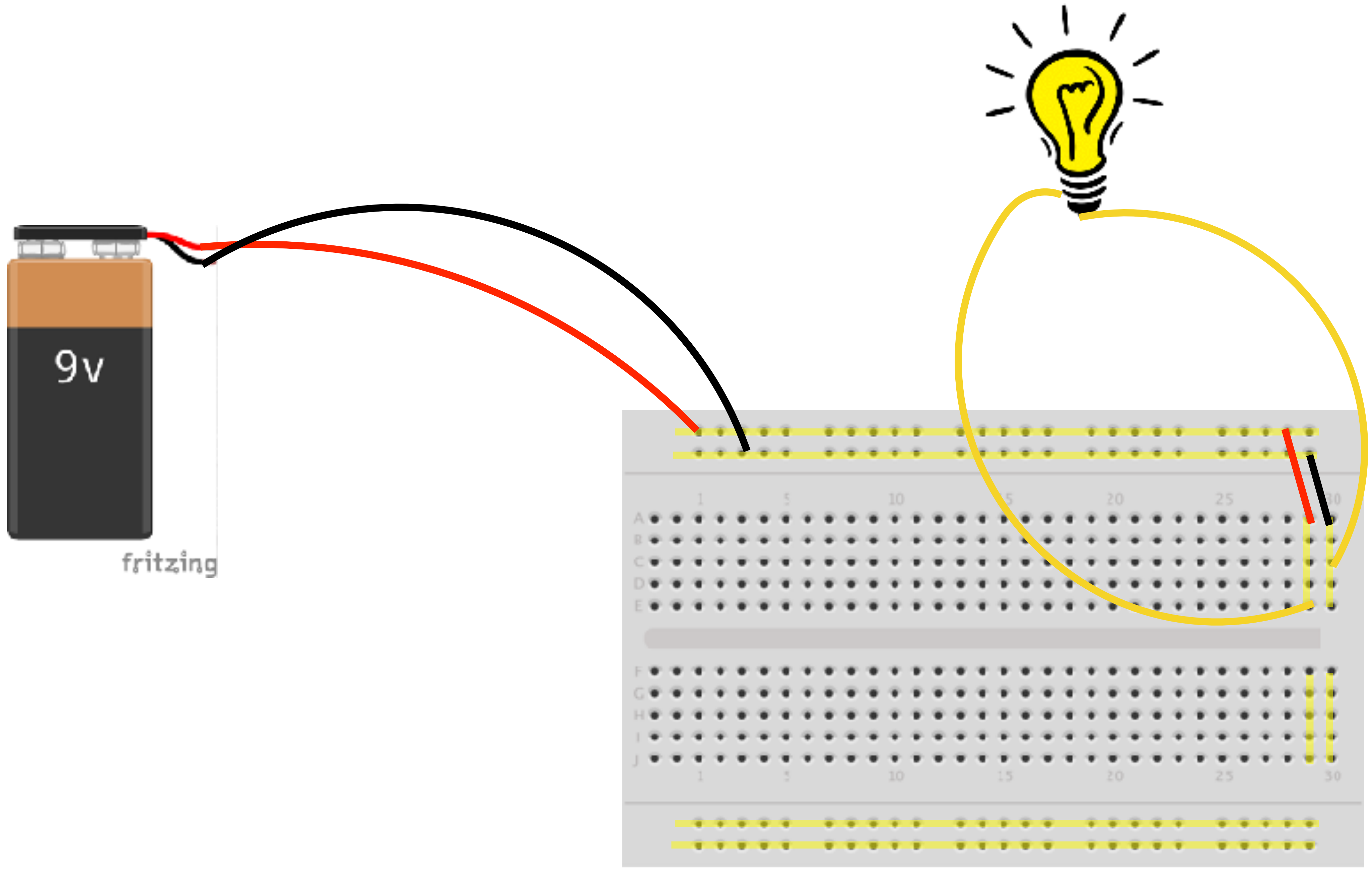
Part II: Basics on Electronics

- Electric circuits
- Current, voltage
- Breadboard
- Short circuits
- Basic components: resistors, LEDs, diodes, buttons, LDRs
- Voltage reading

Breadboard



Example



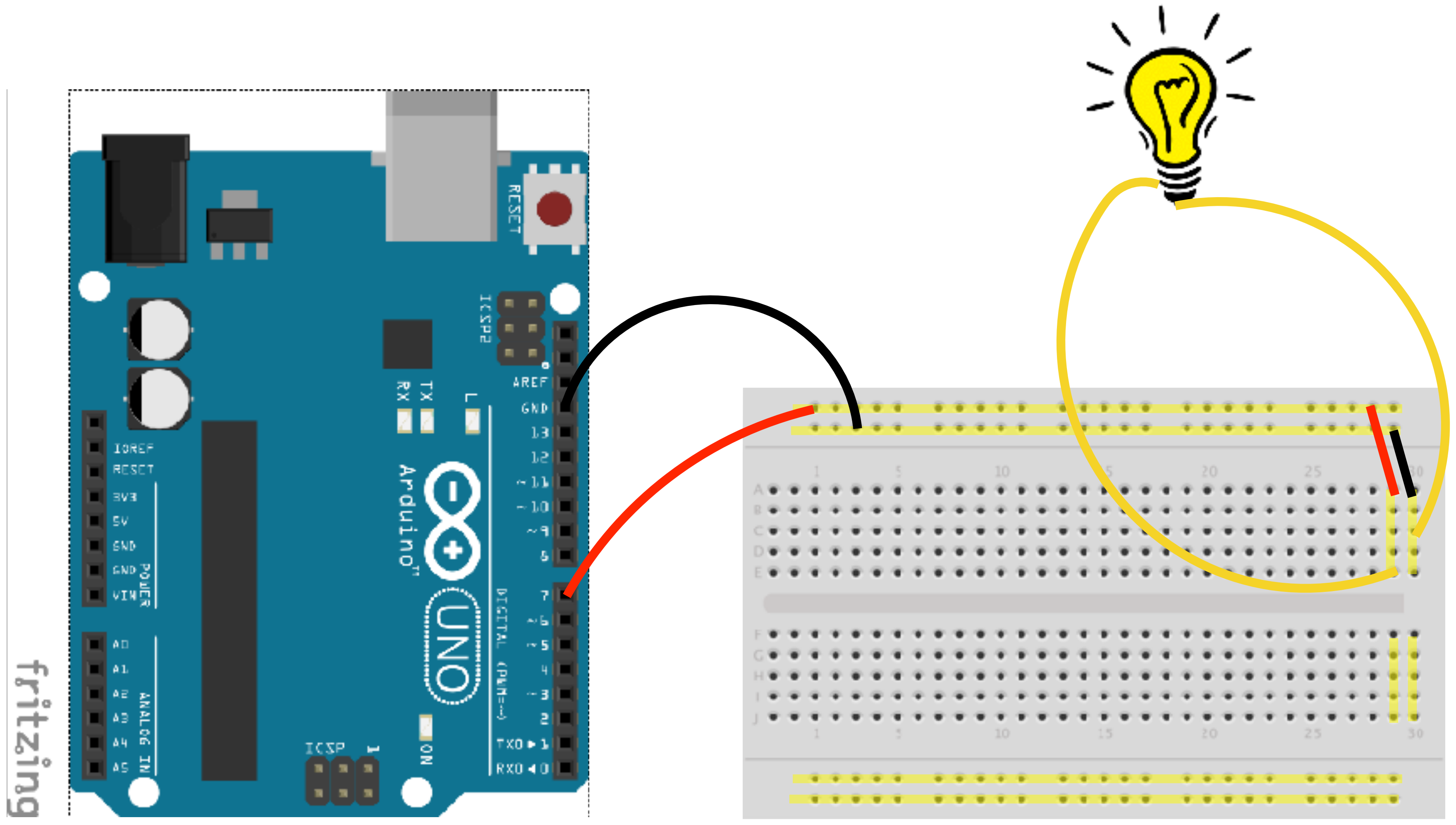
Voltage

- Power source terminals provide a v*oltage*, measured in *Volts* (V)
- Negative (-) terminal provides (conventionally) 0V
- Positive (+) terminal provides higher voltage
- The higher the positive voltage, the higher the current in the circuit! (That is, if you increase the voltage on +, the light bulb emits more light)

Arduino Output Pins

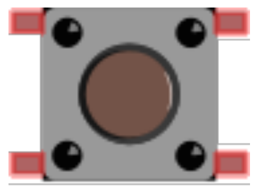
- When Arduino is powered:
- Any pin labelled with `GND` (ground) provides 0V
- Pins labelled with `5V` and `3.3V` provide 5V and 3.3V
- When used in `OUTPUT` mode, Arduino pins:
  - if set to `LOW`, provide 0V
  - if set to `HIGH`, provide 5V

Example



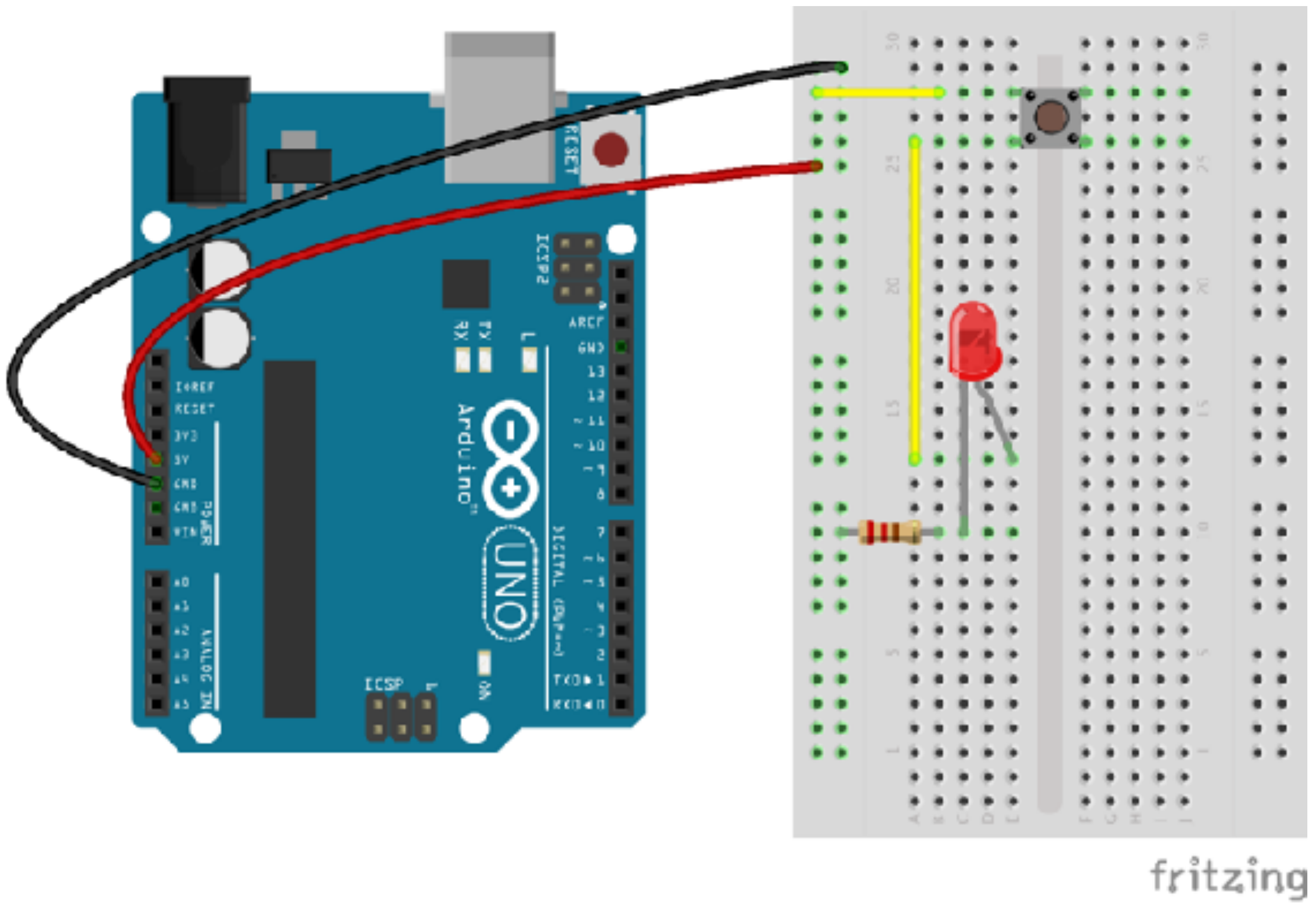
```
…
digitalWrite(7,HIGH)
…
```

Buttons



- Possibly the easiest kind of components
- Allow to open/close a circuit
- Pressed: closed
- Released: open
- Buttons we use:
  - Horizontally always connected
  - Vertically connected when pressed

Example

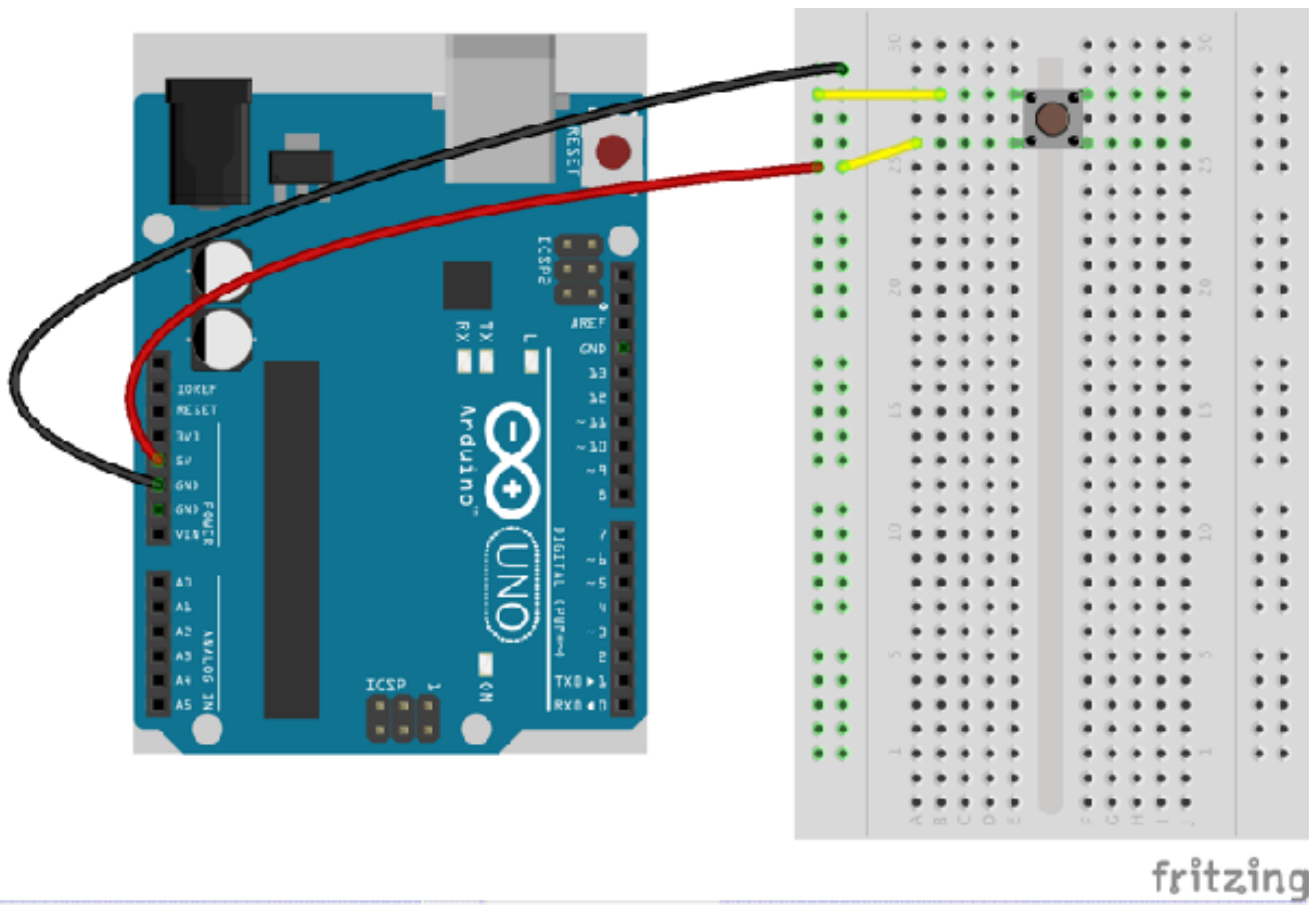

fritzing
UNO
Arduino
DIGITAL (PWM~)
ANALOG IN
POWER
ICSP
ICSP2
RESET
AREF
GND
TX
RX
L
ON

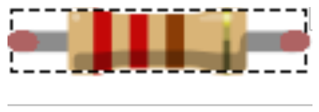
Short-Circuit

- If you connect the positive and the negative terminals, you create a *short-circuit*
- In a short-circuit the current flow is extremely high and this can:
  - create sparks
  - induce battery explosions
  - lead to conductor heating and even melting
- **So… NEVER create a short circuit!!!**

Example

Can this circuit create a short-circuit?

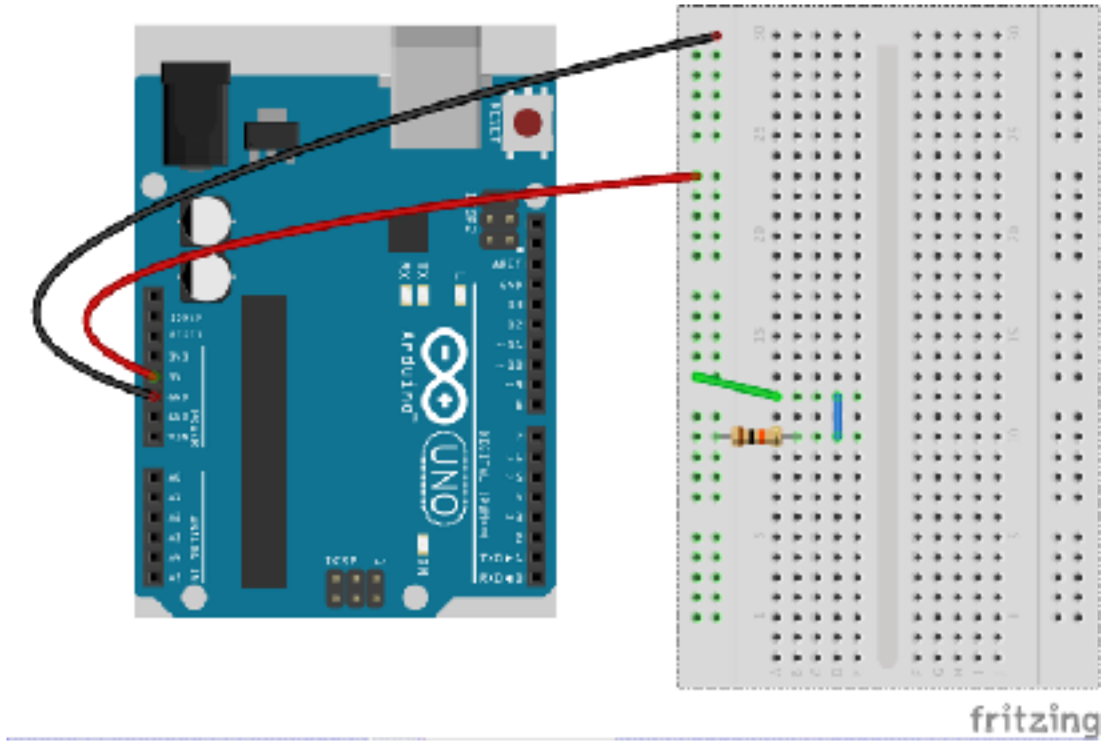




Resistors

- We have used resistors previously
- Resistors offer resistance to current flow
- The higher the resistance (measured in Ohm), the lower the current flow
- Resistors are *not directional*: they let current flow in either way
- Many components (e.g., light bulbs) show a resistor-like behavior

Example



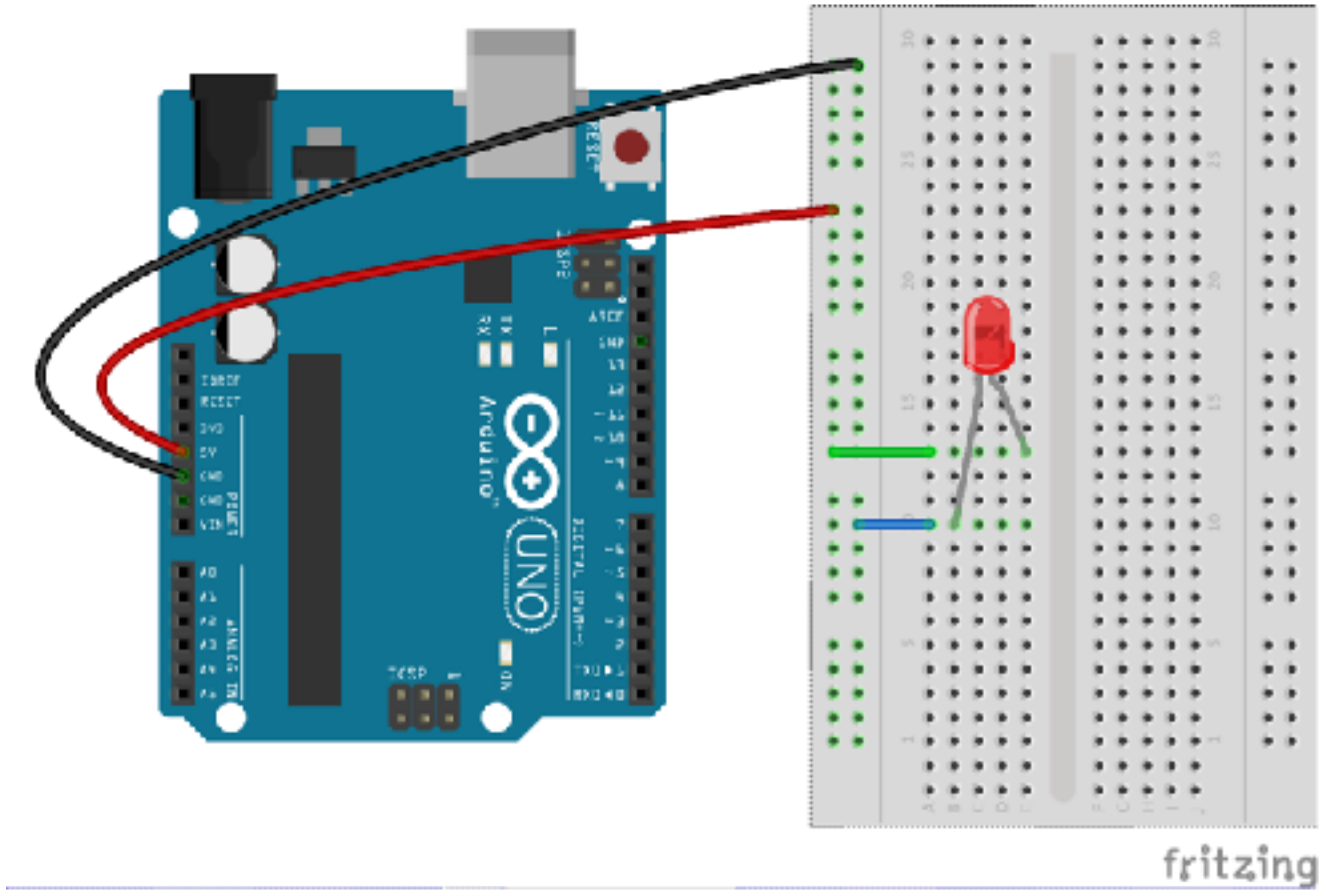
Preventing Short-Circuits

- To prevent a short-circuit, make sure that a component of *suitable resistance* is put between + and -
- This reduces current flow

Diodes and LEDs

- Diodes are components that let current flow in one direction only (anode to cathode)
- LEDs (Light-emitting diodes) work the same as diodes but in addition emit light when current flows
- Diodes and LEDs have essentially *no resistance*

Example

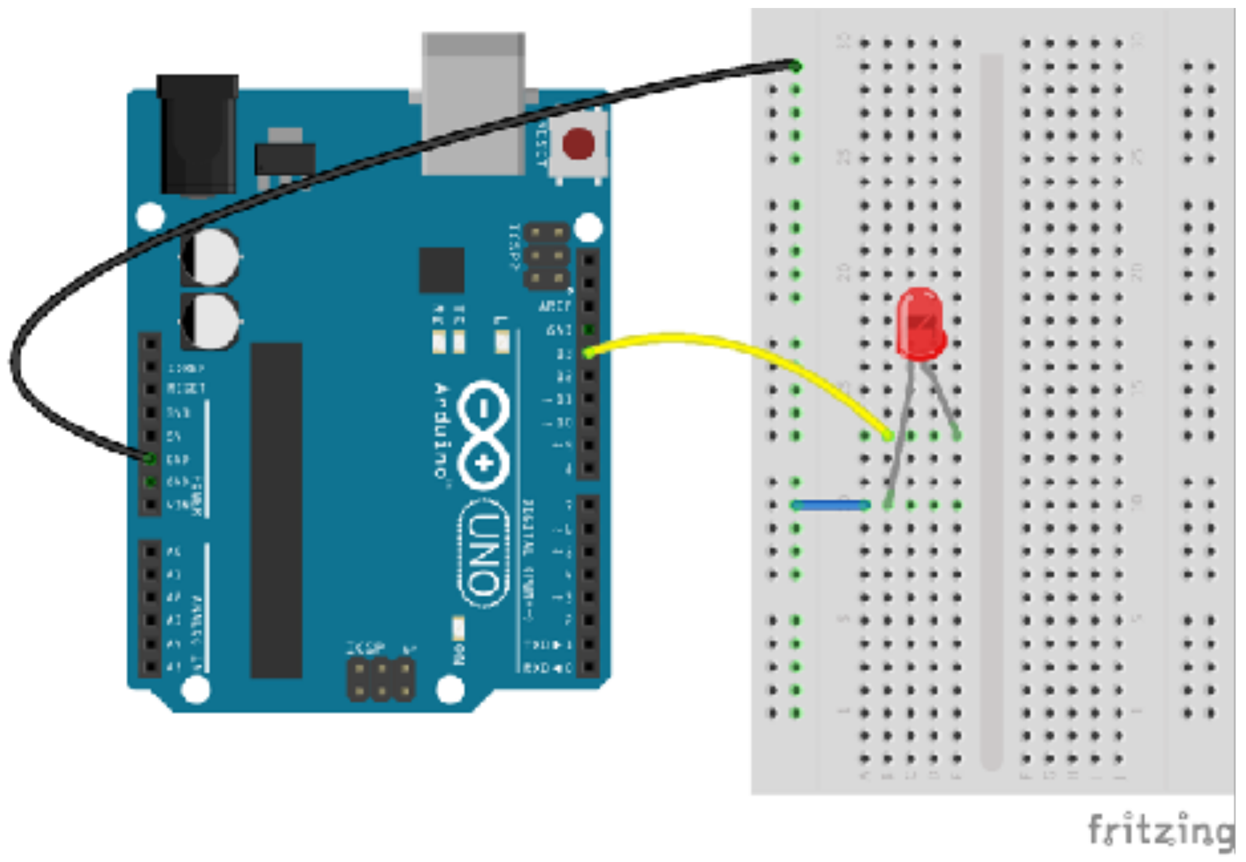


Is there a short-circuit?

(Remember: diodes and LEDs have essentially *no resistance)*

Hands-on #11

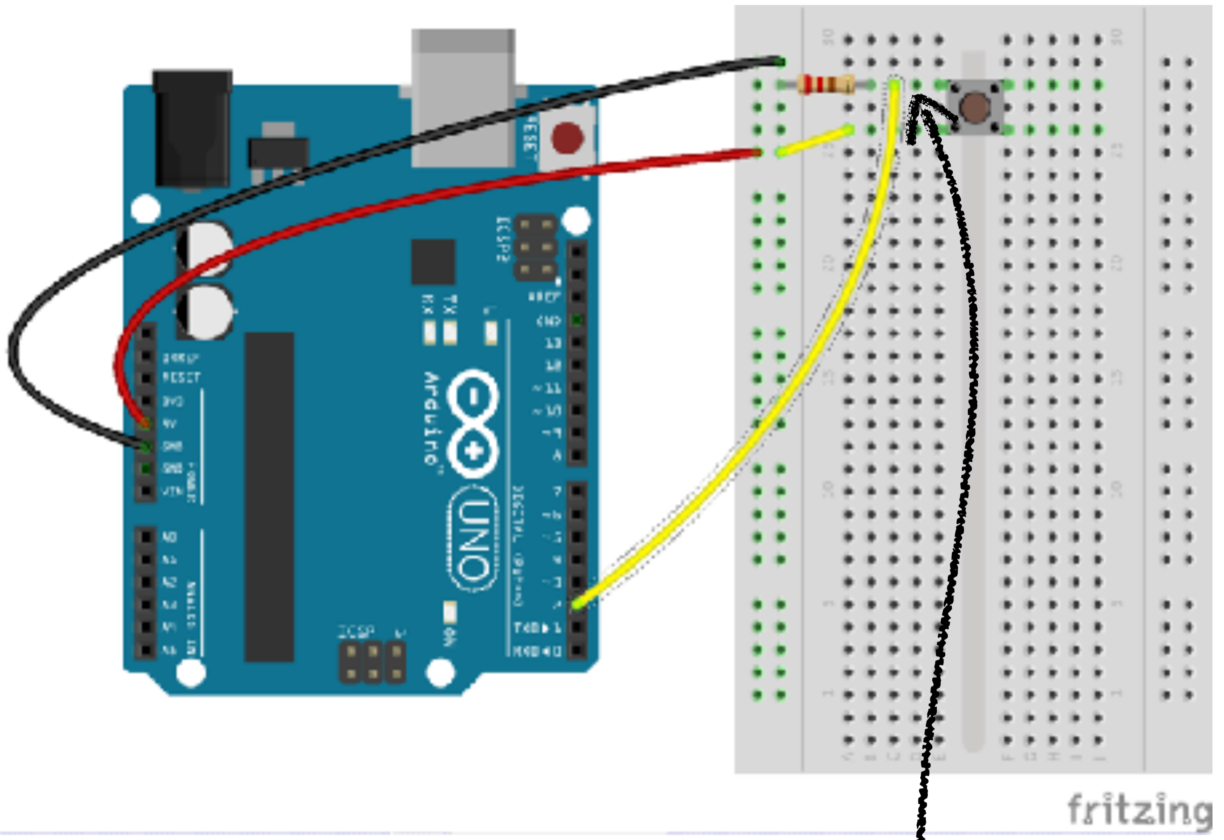
- Can this circuit generate a short-circuit?
- If so, can it be prevented?



Voltage/2

- Each point of the circuit has a voltage
- The voltage at each point depends on how the components are connected and how they are working
- Voltage at the end of (constant) resisting components drops if current increases
- If current does not flow, voltage is the same on both ends
- If you connect a wire to a point of the circuit and plug it into an Arduino pin, you can read the voltage at that point
- Calculating voltage (and current) at each point is complicated, and we will not get into it in details!

Example 1



```
...
pinMode(2,INPUT);
...
int val = digitalRead(2);
...
```

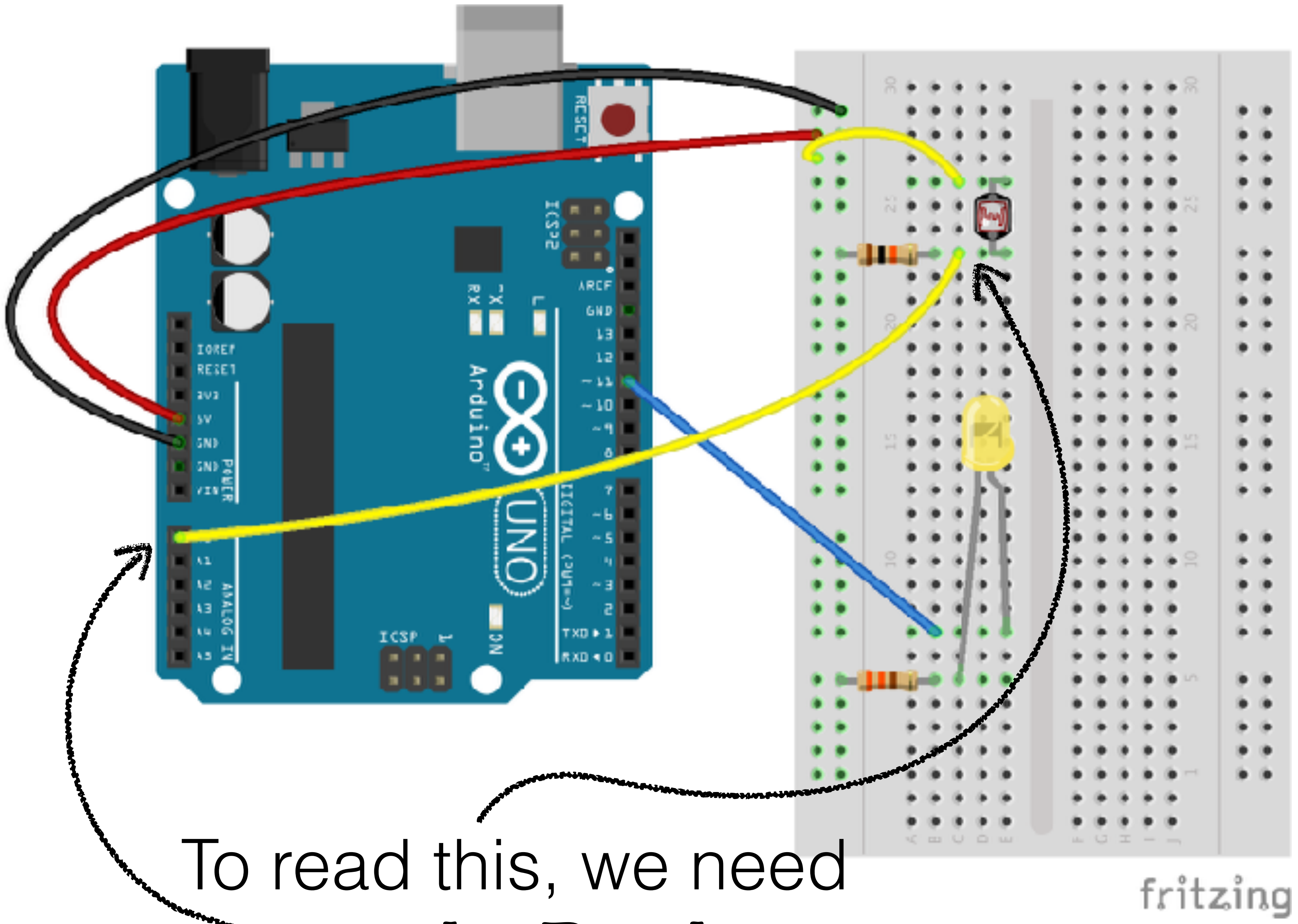
Pin 2 is used to read voltage at this point of circuit

Light-dependent Resistors (LDRs)

- Also called *photoresistors*

- Reduce resistance as light increases
- By reading the voltage at its ends, one gets an approximate measure of the environment light

Example 2



To read this, we need `analogRead`

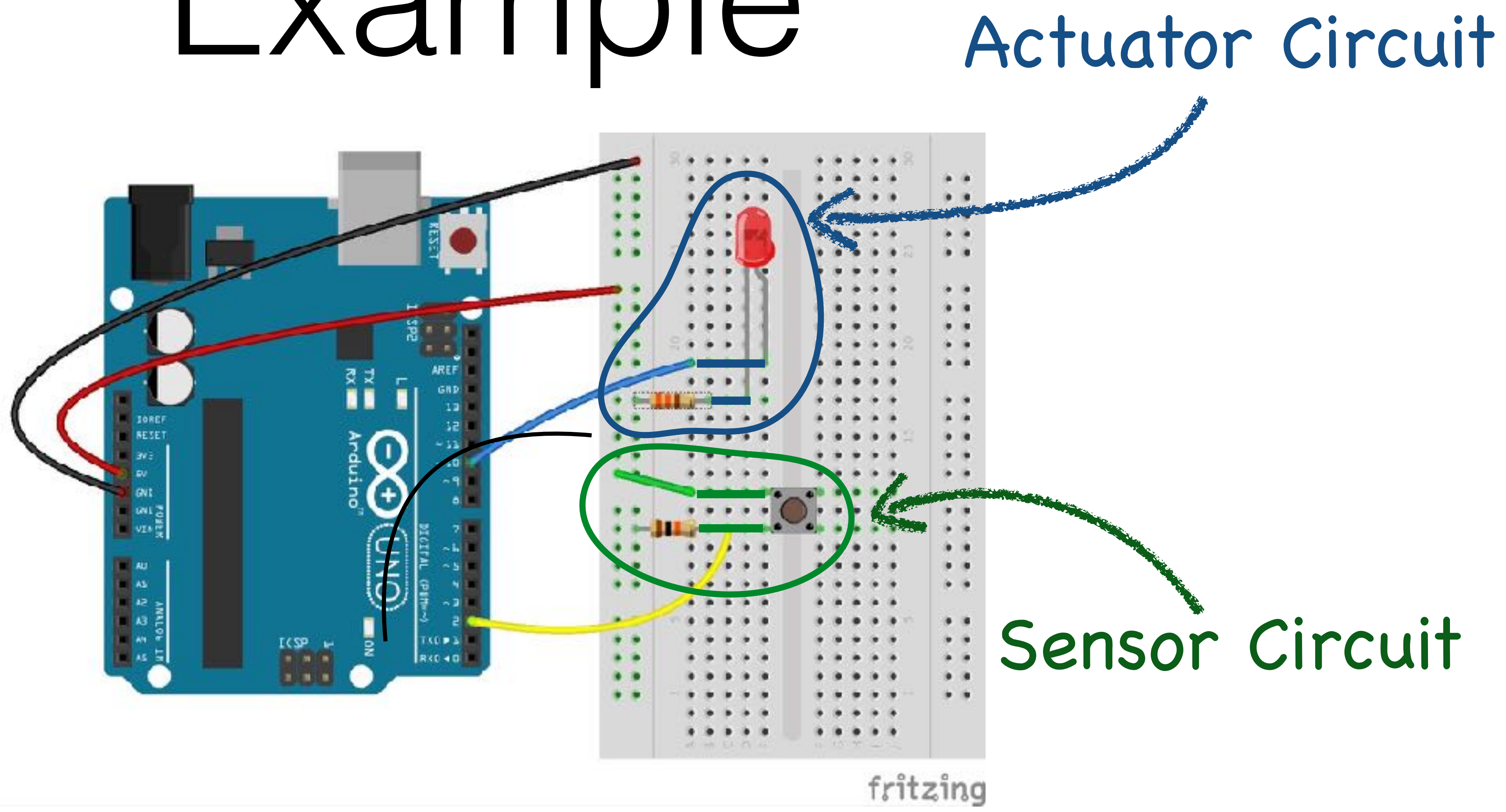
Circuit Assembly

- A basic (yet powerful) approach to assemble circuits
- Basic circuit for actuators
- Basic circuit for sensors

Approach

- Arduino:
    1. Senses: gathers information from sensors
    2. Thinks: elaborates the information
    3. Acts: instructs actuators
- No need to have sensor/actuator direct communication
- Basic idea: have a separate circuit per sensor/actuator

Example



The breadboard hosts two separate circuits
(except for powering)

Sensor and actuator do not interact directly!!!

Approach/2

- To assemble a circuit:
  - Focus on each sub-circuit separately
  - Consider only the interaction of sub-circuit with Arduino, not with other circuits (no interaction)
  - Most sub-circuits are analogous to those we have seen (LEDs, buttons), but they use different components, and might need different resistors (I will help in choosing the right ones ;))
  - More complex circuits (e.g., to drive a motor) will be addressed on demand, depending on the needs of your projects