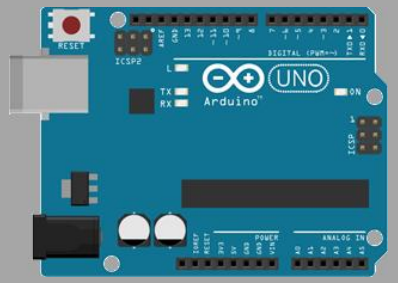
# Introduction to Arduino

## Fundamentals




HAW HAMBURG

Prof. Dr. Marc Hensel

http://www.haw-hamburg.de/marc-hensel

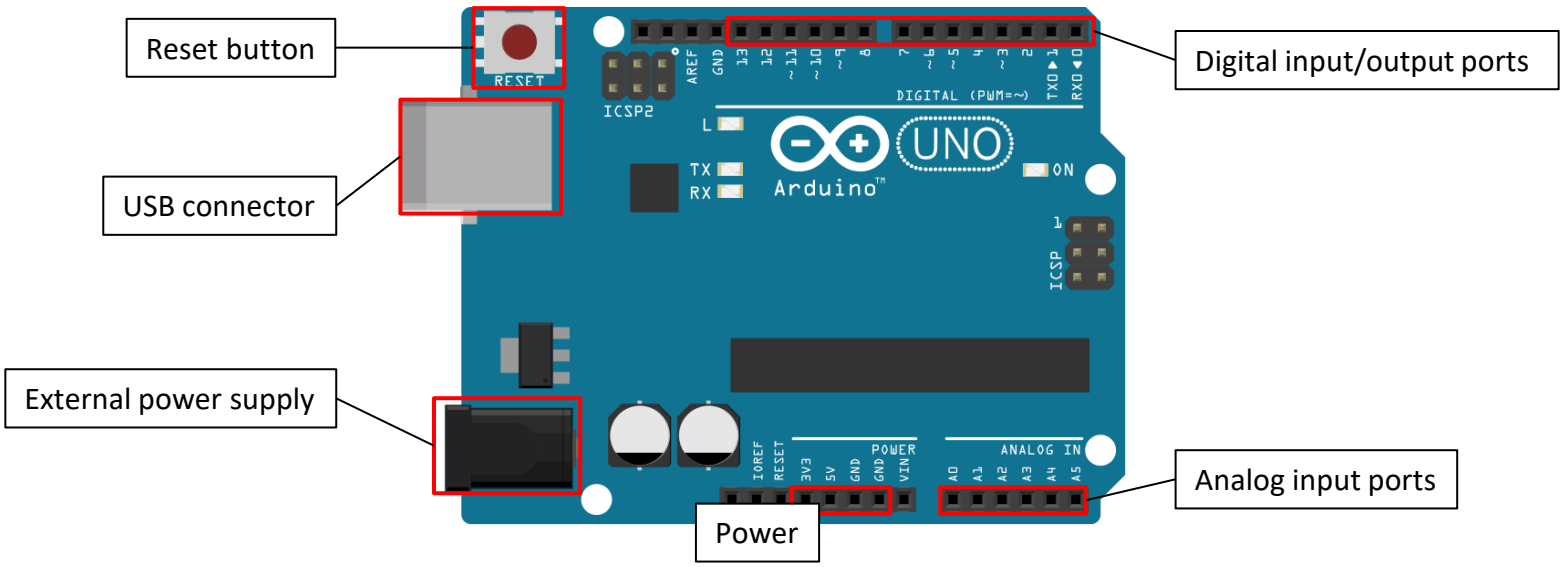
# Contents

# Arduino board and IDE

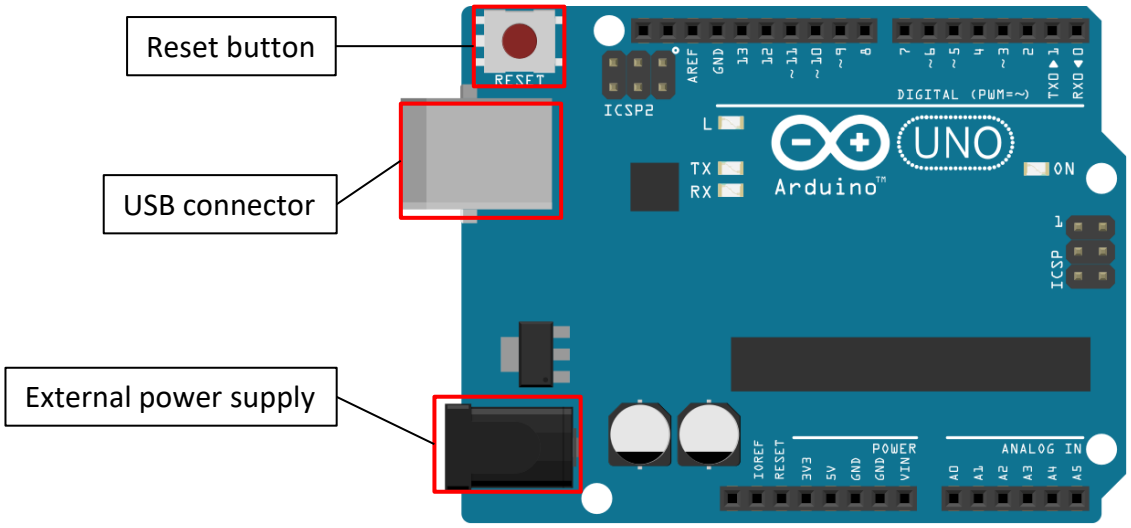
- You identify fundamental pins and interfaces of an Arduino Uno R3.
- You install the software required to program Arduino boards.
- You create a basic sketch (program template).

# Arduino Uno R3

- A variety of boards with different sizes and properties (e.g., interface pins) exist.
- For the beginning, you are fine with Arduino Uno R3:



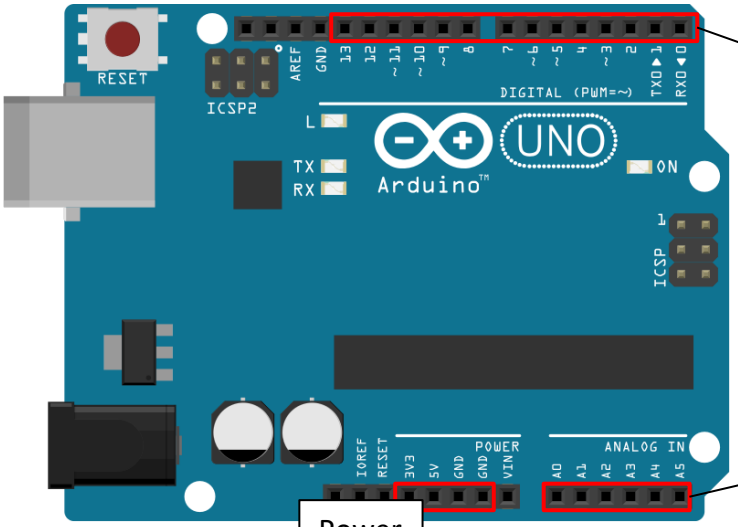
# Arduino Uno R3




- *Reset button:* Re-start execution of program last uploaded to Arduino
- *USB connector:* Connect board to a computer
- *External power:* Supply Arduino with power (up to 12 V) when not connected to PC

USB connection is used to transmit sketches (programs) to the board, send strings to be displayed on the computer (e.g., for debugging), and supply the board with power.

# Arduino Uno R3




Digital input/output ports

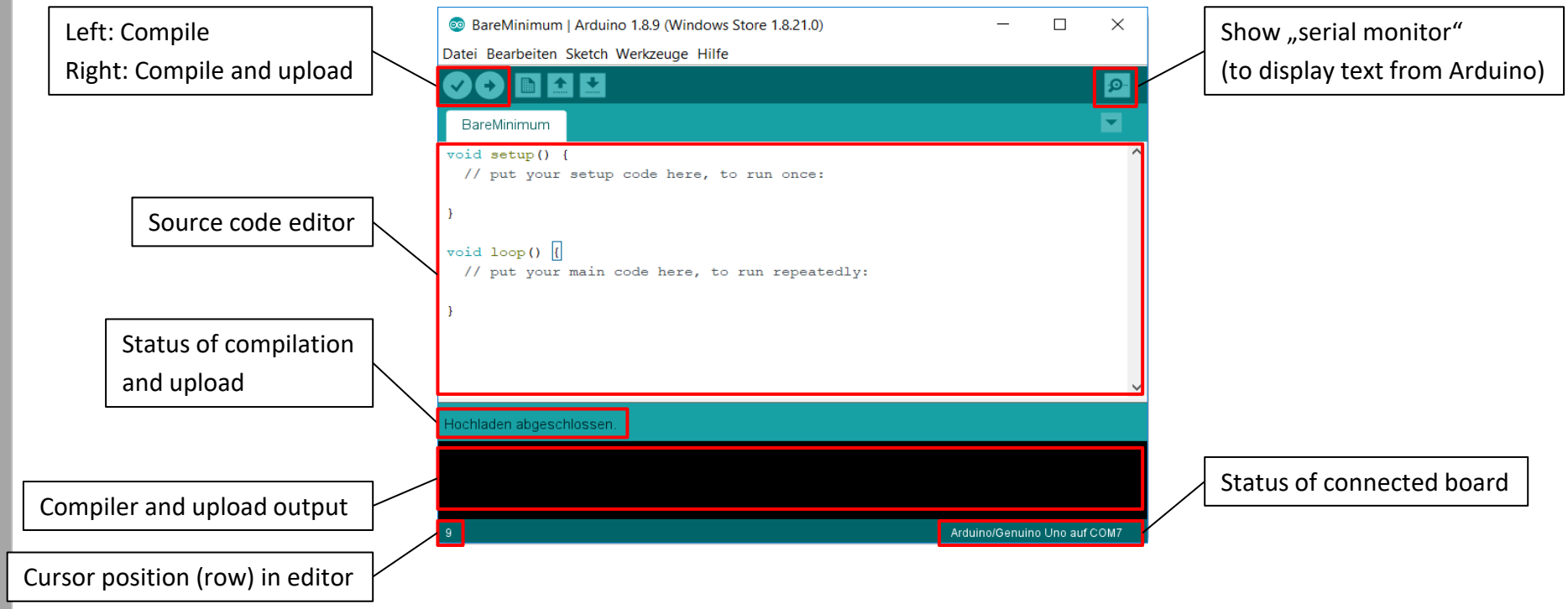
Analog input ports

Power

- *Digital I/O ports:* Set or detect “high” state (5 V) or “low” state (0 V)
- *Analog ports:* Read voltages in 0 – 5 V
- *Power pins:* Provide ground (GND), 3.3 V, and 5 V (e.g., for other components)

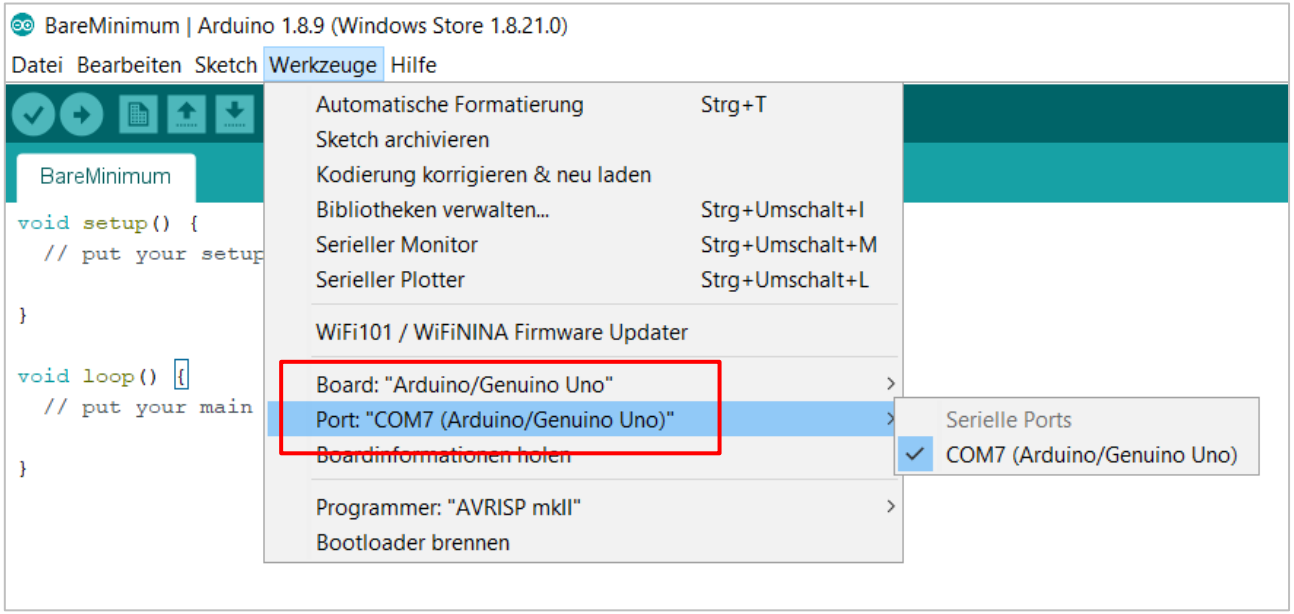
# Arduino IDE

- IDE containing, e.g., source code editor and serial monitor for debugging
- Language used is C++ not C, but C programmers will hardly notice the difference
- Website: https://www.arduino.cc/en/Main/Software



# Setting up an Arduino for development

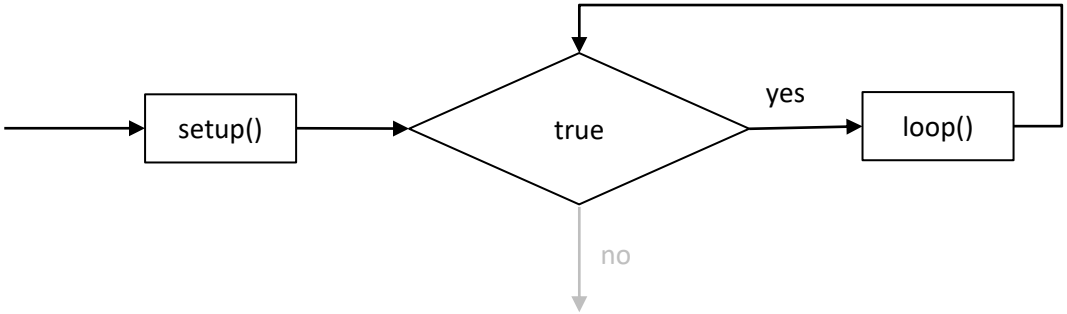
1. Start Arduino IDE
2. Connect the Arduino and computer by USB
3. Select the board type in menu *Tools / Board …*
4. Select the connected port in menu *Tools / Port …*



# Basic Arduino sketch (source code)

You cannot see *main()*, but it is there and …

- calls *setup()* once
- calls *loop()* repeatedly




Minimum sketch:

```
void setup() {
  // Code to run once for initialization
}

void loop() {
  // Code to run repeatedly in a loop
}
```

# Basic digital output

- You create digital output signals with low (0 V) and high (5 V) values.
- You make LEDs (light emitting diodes) blink in specific patterns.

# Connecting a LED to the Arduino

Think about it:

- We want to use LEDs with current $I_L$ = 20 mA and voltage $U_L$ = 2 V.
- But digital pins have either state low ($U_0$ = 0 V) or high ($U_0$ = 5 V).
- How to connect a LED to the Arduino?

Protect the LED with a resistor:

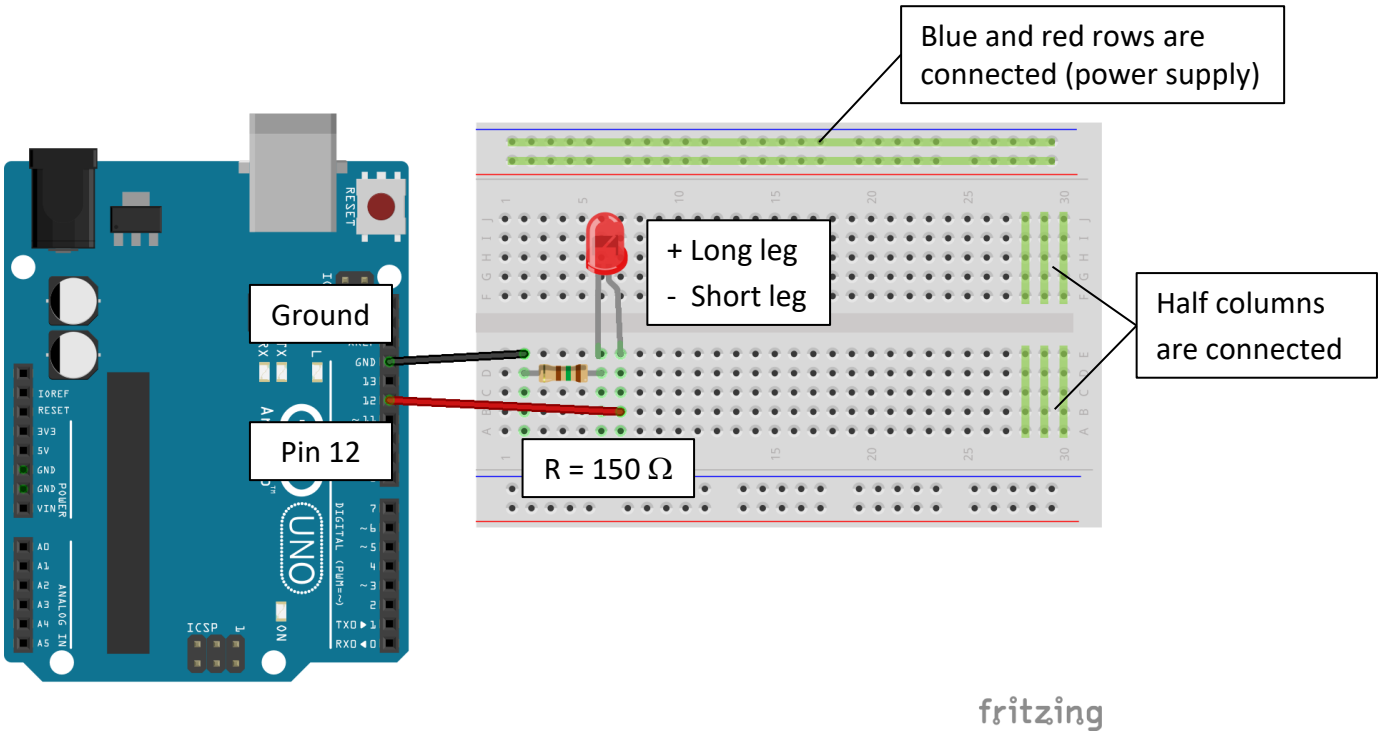
- Use a resistor *R* in series to the LED as voltage divider (i.e., absorb part of the voltage).
- The current $I_R = I_L$ = 20 mA is given by the LED.
- The target voltage is $U_R = U_0 - U_L$ = 3 V.

$$\Rightarrow R = \frac{U_R}{I_R} = \frac{3\ V}{0.02\ A} = 150\ \Omega$$

# Connecting a LED to the Arduino

Convenient way to build the circuit is using a *breadboard*:

- Blue (ground) and red (positive voltage) rows are connected.
- Columns are connected in the upper and the lower half.
- We use pin 12, but it could be any other digital pin. (Note that state high on pin 13 turns on a LED on the board.)



# Making the LED blink

Required methods (“functions”):

- Set digital pin to be an output pin → *pinMode(pin, mode)*
- Set state of pin to *high* or *low* → *digitalWrite(pin, state)*
- Wait for a specific time (in ms) → *delay(time)*

Sketch:

- Both programs behave equivalently
- Recommended to use defines to make code more readable

```
void setup() {
  pinMode(12, OUTPUT);
}

void loop() {
  digitalWrite(12, HIGH);
  delay(2000);
  digitalWrite(12, LOW);
  delay(1000);
}
```

```
#define PIN_LED 12

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

void loop() {
  digitalWrite(PIN_LED, HIGH);
  delay(2000);
  digitalWrite(PIN_LED, LOW);
  delay(1000);
}
```

# Exercise: Wind turbine lights

Implement the blinking pattern for wind turbines to warn low-flying aircraft. The light is turned on for 1s, ½ s off, 1 s on, 1.5 s off, and repeating.

Okay, that one was still easy:

```
#define PIN_LED 12

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

void loop() {
  digitalWrite(PIN_LED, HIGH);
  delay(1000);
  digitalWrite(PIN_LED, LOW);
  delay(500);

  digitalWrite(PIN_LED, HIGH);
  delay(1000);
  digitalWrite(PIN_LED, LOW);
  delay(1500);
}
```

# Exercise: Ambulance light

Implement a pattern for German ambulances consisting of two flashes within 200 ms followed by a pause of 300 ms and repeating.

Sample solution:

- Let's introduce a method to blink for a specific time.
- Additionally, we calculate the pause so that the flashes last 200 ms.

```
#define PIN_LED 12
#define TIME_FLASH_MS 65

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

void blink(int pin, int timeMs) {
  digitalWrite(pin, HIGH);
  delay(timeMs);
  digitalWrite(pin, LOW);
}
```

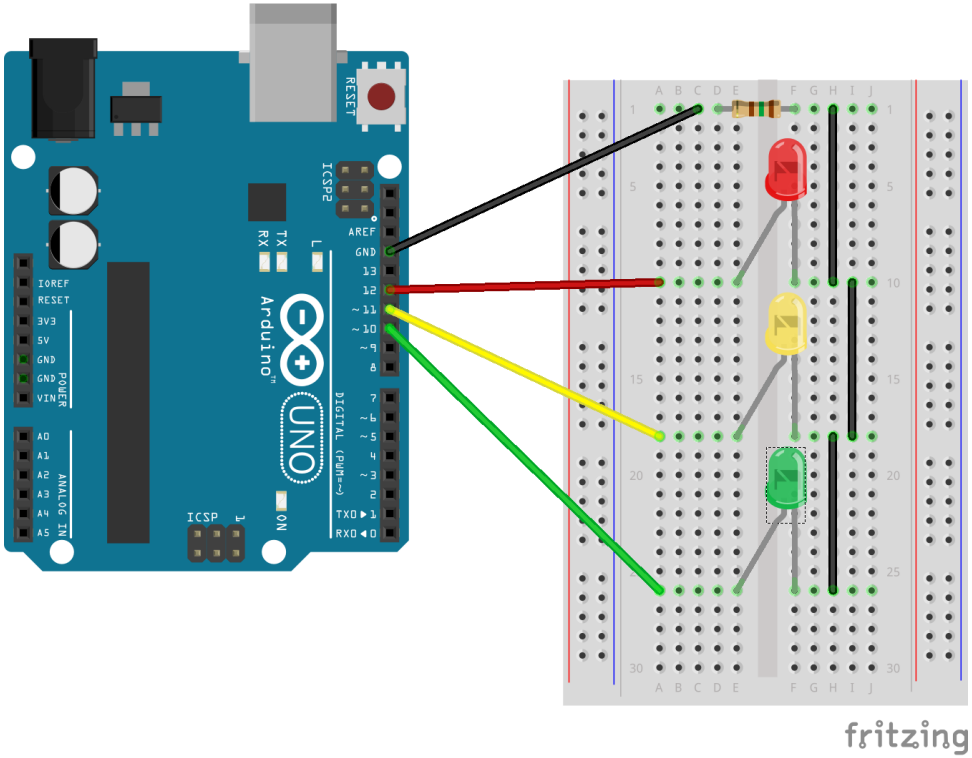
```
void loop() {
  // Blink twice in 200 ms
  blink(PIN_LED, TIME_FLASH_MS);
  delay(200 - 2 * TIME_FLASH_MS);
  blink(PIN_LED, TIME_FLASH_MS);

  // Lights off for 300 ms
  delay(300);
}
```

# Exercise: Traffic light

Implement a traffic light repeating the pattern red, red & yellow, green, and yellow.

Sample solution:

- Let’s add two more LEDs with all LEDs sharing a resistor for protection.



# Exercise: Traffic light

- Sample sketch:

```
#define PIN_RED 12
#define PIN_YELLOW 11
#define PIN_GREEN 10
#define TIME_LONG_MS 7000       // Time for green and red lights
#define TIME_SWITCH_MS 2000     // Time when switching between greem and red

void setup() {
  pinMode(PIN_RED, OUTPUT);
  pinMode(PIN_YELLOW, OUTPUT);
  pinMode(PIN_GREEN, OUTPUT);
}

void loop() {
  setLights(HIGH, LOW, LOW, TIME_LONG_MS);      // Red
  setLights(HIGH, HIGH, LOW, TIME_SWITCH_MS);   // Red & yellow
  setLights(LOW, LOW, HIGH, TIME_LONG_MS);      // Green
  setLights(LOW, HIGH, LOW, TIME_SWITCH_MS);    // Yellow
}

void setLights(int stateRed, int stateYellow, int stateGreen, int timeMs) {
  digitalWrite(PIN_RED, stateRed);
  digitalWrite(PIN_YELLOW, stateYellow);
  digitalWrite(PIN_GREEN, stateGreen);
  delay(timeMs);
}
```

# Digital output using PWM

- You control the power of digital output pins by generating PWM signals.
- You dim LEDs using the Arduino's PWM pins.

# Pulse width modulation (PWM)

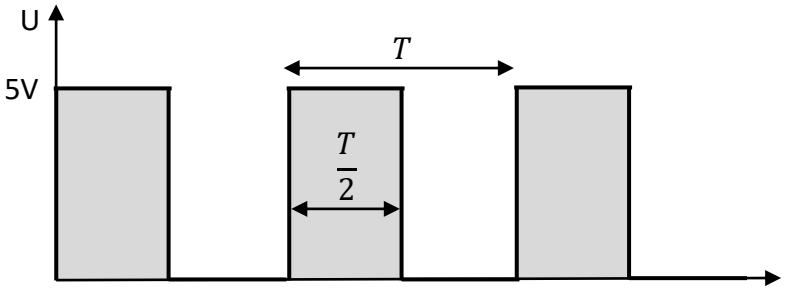
We have a principal issue:

- Arduino's digital output pins have either 0 V (low) or 5 V (high).
- But we might want to have voltages in-between 0 and 5 V (e.g., to dim a LED).

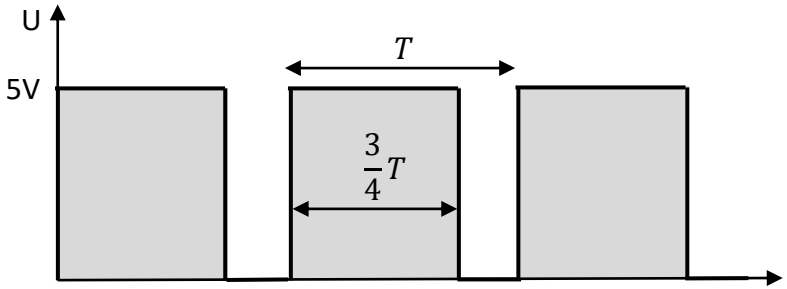
Solution:

- Switch very quickly periodically between 0 V and 5 V.
- The ratio of high and low periods adapts the power $P = U \cdot I$ transmitted by the output.
- Note that this impacts power, but does not generate voltages between 0 V and 5 V.

Examples:




50% of the maximum power

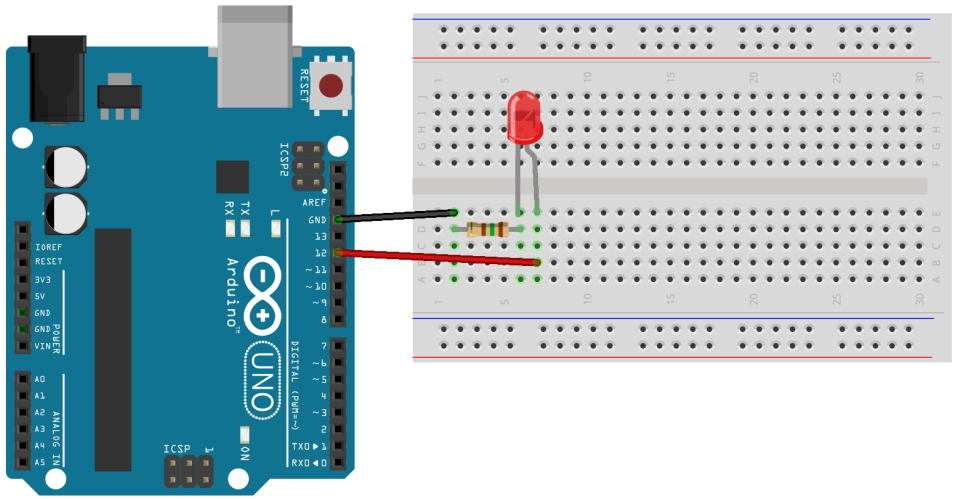



75% of the maximum power

# Exercise: Smoothly turn off LED

Turn off a LED visually smoothly:

- Slowly decrease the power provided by the appropriate digital output pin.
- Experiment with different periods *T*. Is it possible to dim smoothly without visual flicker?



# Exercise: Smoothly turn off LED

Sample solution:

No good parameters exist for smooth turning off without visible flickering

```
#define PIN_LED 12
#define PWM_PERIOD_MS 20              // Period T in ms of PWM signal

double percentPower = 100.0;          // Perentage of "high" in period
double deltaPercent = 0.25;           // Decresing percentage in each loop

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

void loop() {
  int pulseLengthMs = (int)(percentPower/100.0 * PWM_PERIOD_MS);
  digitalWrite(PIN_LED, HIGH);
  delay(pulseLengthMs);
  digitalWrite(PIN_LED, LOW);
  delay(PWM_PERIOD_MS - pulseLengthMs);
  percentPower = max(percentPower - deltaPercent, 0.0);
}
```

Returns the larger of its two arguments
⇒ Prevent negative values

# Exercise: Smoothly turn off LED

Improved solution:

- Uses *delayMicroseconds()* for smoother transition
- Parameters adapt accordingly (period and percentage divided by 5)

```
#define PIN_LED 12
#define PWM_PERIOD_MICROS 4000        // Period T in µs of PWM signal

double percentPower = 100.0;          // Perentage of "high" in period
double deltaPercent = 0.05;           // Decresing percentage in each loop

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

void loop() {
  unsigned int pulseLengthMicros = (unsigned int)(percentPower/100.0 * PWM_PERIOD_MICROS);
  digitalWrite(PIN_LED, HIGH);
  delayMicroseconds(pulseLengthMicros);
  digitalWrite(PIN_LED, LOW);
  delayMicroseconds(PWM_PERIOD_MICROS - pulseLengthMicros);
  percentPower = max(percentPower - deltaPercent, 0.0);
}
```

# Using dedicated PWM ports

Okay, we can generate a PWM signal, but we still have an issue:

- Generating the periodical signal keeps the Arduino busy.
- How to generate more than one PWM signal with different period?
- How to do other tasks while generating a PWM signal?

Solution:

- Digital ports marked by a tilde (~) can generate PWM signals.
- Arduino Uno R3: ports 3, 5, 6, 9, 10, 11
- Use *analogWrite(pin, value)* with a value in 0 (always off) to 255 (always on).
- Ports need not be set as output by calling *pinMode()*.

# Exercise: Smoothly turn off LED (continued)

Let's use this handy feature:

- Adapt your solution to the last exercise accordingly.

Sample solution:

- Remember to connect the LED to a PWM pin (here: 11 instead of 12).
- Much better than the previous solution … isn't it?

```
#define PIN_LED 11      // Must be a PWM port (i.e., marked by a tilde ~)

int pwmValue = 255;

void setup() {
}

void loop() {
  analogWrite(PIN_LED, pwmValue);
  delay(20);
  pwmValue = max(pwmValue - 1 , 0);
}
```

# Reading digital input

- You read the logical voltage level at digital input pins.
- You react to push button presses.
- You use a serial interface to print to the IDE’s serial monitor.
- You generate random numbers and measure elapsed time.
- You measure pulse lengths.

# Connecting push buttons

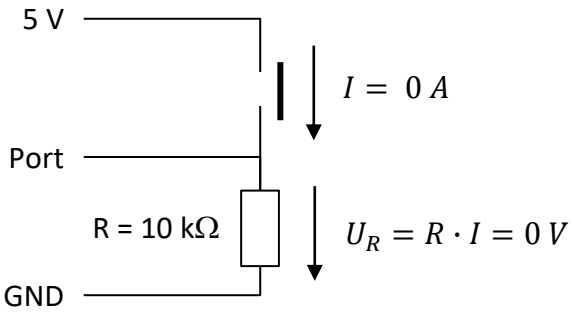
Digital pins can act as input pins:

- Set the pin mode → *setMode(pin, INPUT)*
- Read the logical level at the pin → *digitalRead(pin)*

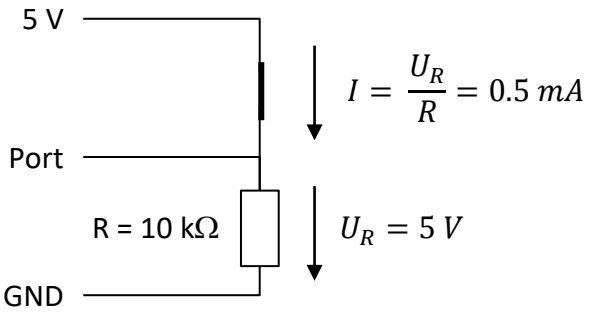
Think about how to connect a push button so that:

- Released button corresponds to *low* ⇒ 0 V at digital pin
- Pressed button corresponds to *high* ⇒ 5 V at digital pin

Solution using a pull-up resistor *R*:




a) Push button released

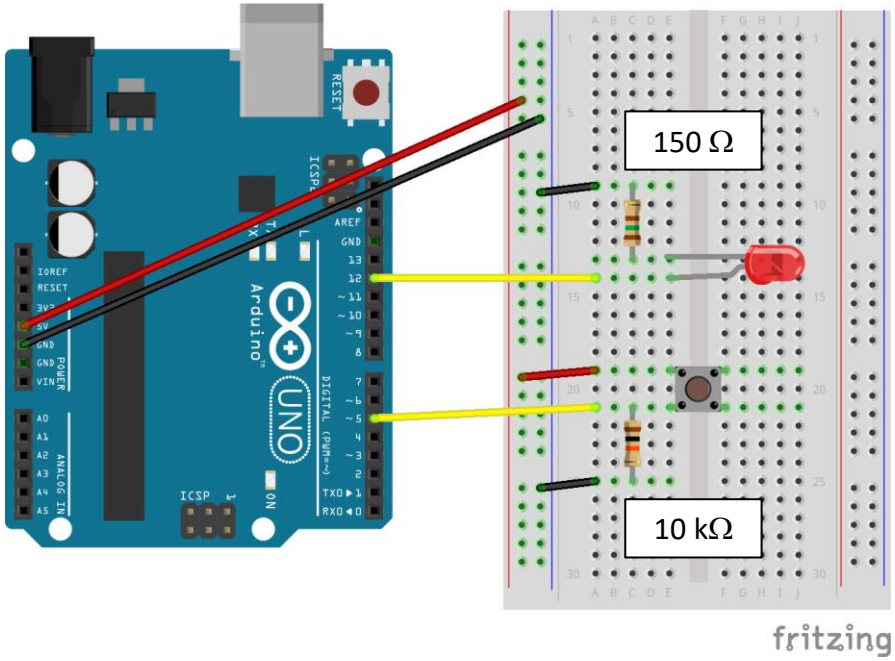



b) Push button pressed

# Exercise: Push button to switch on LED

Let's put that into life:

- Create a solution that switches a LED on when a push button gets pressed.
- React on the first button press, only.



# Exercise: Push button to switch on LED

Sample solution:

```
#define PIN_LED 12
#define PIN_BUTTON 5

boolean isLightOff = true;

void setup() {
  pinMode(PIN_LED, OUTPUT);
  pinMode(PIN_BUTTON, INPUT);
}

void loop() {
  if (isLightOff) {
    int inputState = digitalRead(PIN_BUTTON);
    boolean isButtonPressed = (inputState == HIGH);

    if (isButtonPressed) {
      digitalWrite(PIN_LED, HIGH);
      isLightOff = false;
    }
  }
}
```

# Serial monitor

So far, the Arduino is much alike a married man … it doesn't talk much!

Serial monitor:

- Arduino can send data to computer using the USB connection
- IDE's serial monitor window displays received strings
- Open the serial monitor either in the menu or the symbol in the upper right.

Example:

```
void setup() {
  Serial.begin(9600);                          // Establish serial connection

  int answer = 42;
  Serial.println("Printing with line break.");  // Print line
  Serial.print("The answer is: ");              // Print without line break
  Serial.println(answer);                      // Print value of variable
}

void loop() {
}
```

# Exercise: Measuring your reaction time

Let's find out your reaction time:

- Switch on the LED after a random period (e.g., in 3 to 10 s).
- Measure the time taken until the button is pressed and display the result.
- Switch off the LED and start the next round after 5 s.

Circuit:

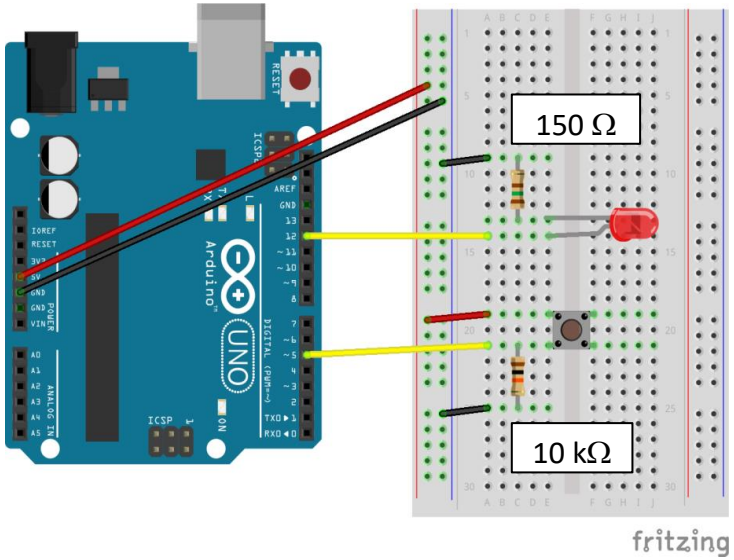
- Same as in prior exercise

Random generator:

- Initialize: *randomSeed(analogRead(0))*
- Next number: *random(min, max)*

Measure time in milliseconds:

- Since start: *millis()*



# Exercise: Measuring your reaction time

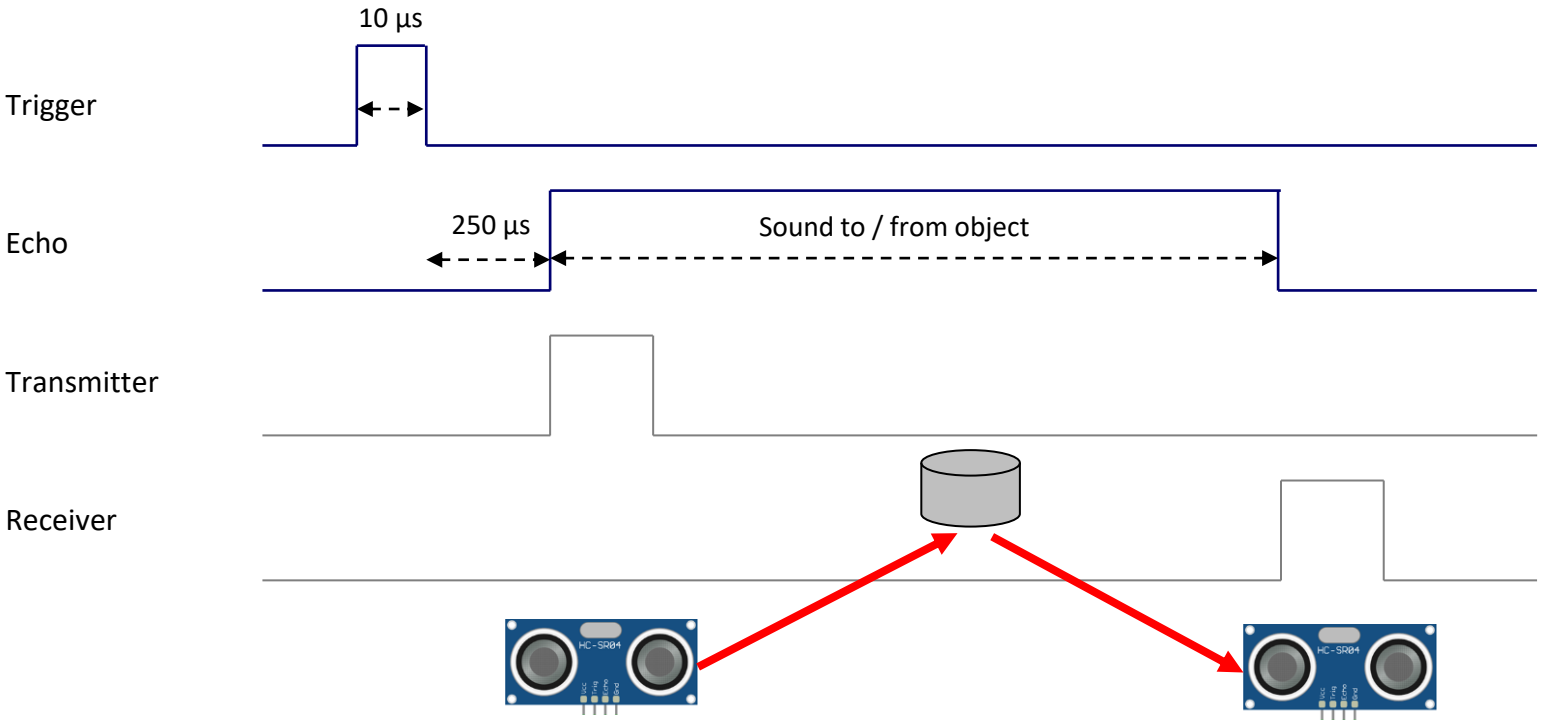
Sample solution:

```
#define PIN_LED 12
#define PIN_BUTTON 5

void setup() {
  Serial.begin(9600);
  randomSeed(analogRead(0));    // Init random generator
  pinMode(PIN_LED, OUTPUT);
  pinMode(PIN_BUTTON, INPUT);
}

void loop() {
  // Wait a random period before switching LED on
  delay(random(3000, 10000));
  digitalWrite(PIN_LED, HIGH);

  // Wait for button pressed
  unsigned long startTime = millis();
  while (digitalRead(PIN_BUTTON) != HIGH) {
  }
  unsigned long stopTime = millis();

  // Display measured time
  Serial.print("Time taken: ");
  Serial.print((stopTime - startTime) / 1000.0);
  Serial.println(" s");

  // Next game after a couple of seconds
  delay(5000);
  digitalWrite(PIN_LED, LOW);
}
```

# Ultrasonic distance sensor HC-SR04

Principle:

1. Trigger pulse (set *trigger* pin to *HIGH* for at least 10 µs)
2. Waits for 250 µs after trigger went back to *LOW*
3. Sends 40 kHz burst and sets *echo* to *HIGH*
4. Sets *echo* to *LOW* when receiving echo



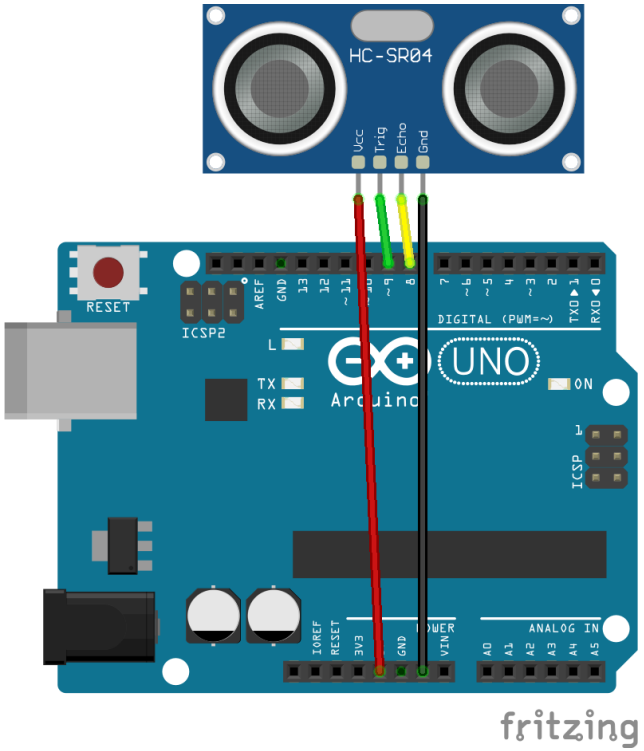
# Ultrasonic distance sensor HC-SR04

Distance calculation:

- Sound travels to and from object during echo pulse *T*
- Speed of sound: $v \approx 343 \frac{m}{s}$
- Distance: $d = v \cdot \frac{T}{2} \approx \frac{T}{58\ \mu s} cm$

Required method (“function”):

- Read pulse length: *pulseIn (pin, mode)*
- Mode: *HIGH* or *LOW*



# Ultrasonic distance sensor HC-SR04

Sample code:

```
#define TRIGGER_PIN 9
#define ECHO_PIN 8
#define MIN_RANGE_CM 2
#define MAX_RANGE_CM 300
#define MICROSEC_TO_CM 29     // Approx. time to cm at 20 C (343 m/s)

void setup() {
  Serial.begin(9600);
  pinMode(TRIGGER_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
}

void loop() {
  // Send trigger pulse
  digitalWrite(TRIGGER_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGGER_PIN, LOW);

  // Read echo pulse and determine distance
  // (Sound travels to object and back => Divide by 2)
  long distanceMicrosec = pulseIn(ECHO_PIN, HIGH) / 2;
  int distanceCm = distanceMicrosec / MICROSEC_TO_CM;

  if ((distanceCm >= MIN_RANGE_CM) && (distanceCm <= MAX_RANGE_CM)) {
    Serial.print(distanceCm);
    Serial.println(" cm");
  } else {
    Serial.println("Distance not within range (2 .. 300 cm)");
  }
  delay(500);
}
```

# Interrupts on digital input changes

- You react on voltage level changes on digital pins.
- You define on which changes to interrupt the current program flow and which methods to call for an interrupt handling.

# Interrupts

Our so far approach to react to pressed buttons has drawbacks:

- Need to repeatedly read the state of the respective input pin (“busy waiting”)
- Difficult to do other tasks while monitoring the pin

Approach using interrupts:

- Define on what type of change of a pin to react
- System calls a specific method when change occurs (⇒ Notification instead of monitoring)

Implementation:

- Implement a method (e.g., *methodName()*) to call on an interrupt
- Global variables used in that method must have modifier *volatile*

```
void methodName() { ... }
```

- Attach interrupt to a pin (only pins 2 and 3 for Uno R3)

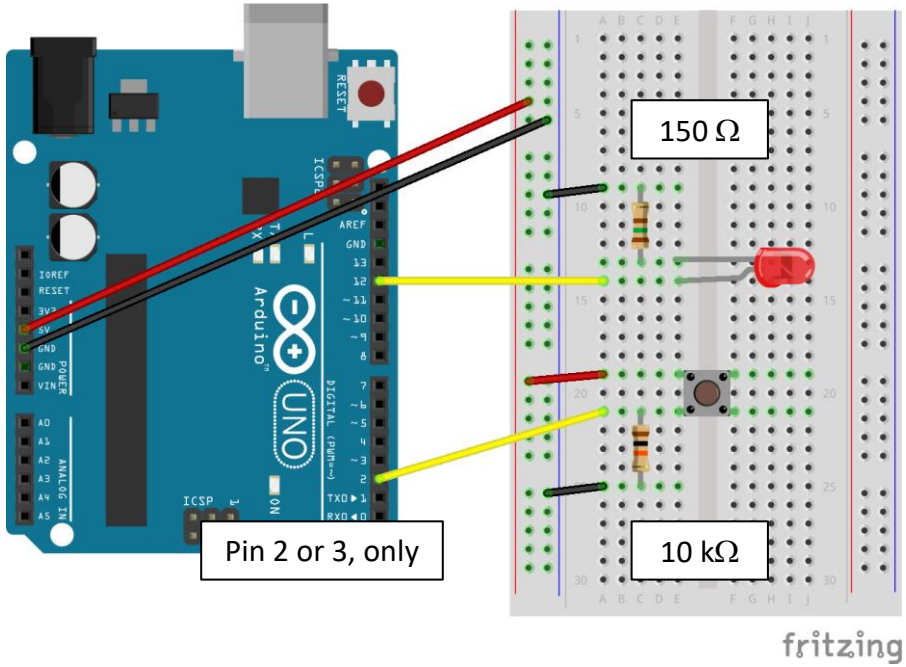
```
attachInterrupt( digitalPinToInterrupt( pin ), methodName, change )
```

LOW, CHANGE, RISING, or FALLING

# Exercise: Toggle LED using interrupt

Let's simplify and improve a prior exercise:

- Toggle a LED on/off whenever a push button gets pressed.



# Exercise: Toggle LED using interrupt

Sample solution:

```
#define PIN_LED 12
#define PIN_BUTTON 2     // Uno: Interrupts only for pins 2 and 3

volatile boolean isLightOff = true;

void setup() {
  pinMode(PIN_LED, OUTPUT);
  pinMode(PIN_BUTTON, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(PIN_BUTTON), toggleLight, RISING);
}

void loop() {
  // Free to do other things here ...
}

void toggleLight() {
  if (isLightOff)
    digitalWrite(PIN_LED, HIGH);
  else
    digitalWrite(PIN_LED, LOW);
  isLightOff = !isLightOff;
}
```

# Measuring analog input

- You measure analog voltages in 0 to 5 V.

# Analog input ports

It is quite straight-forward … honestly:

- Method *analogRead(pin)* reads voltages from the analog ports (A0 – A5).
- Ports need not be set as output by calling *pinMode()*.

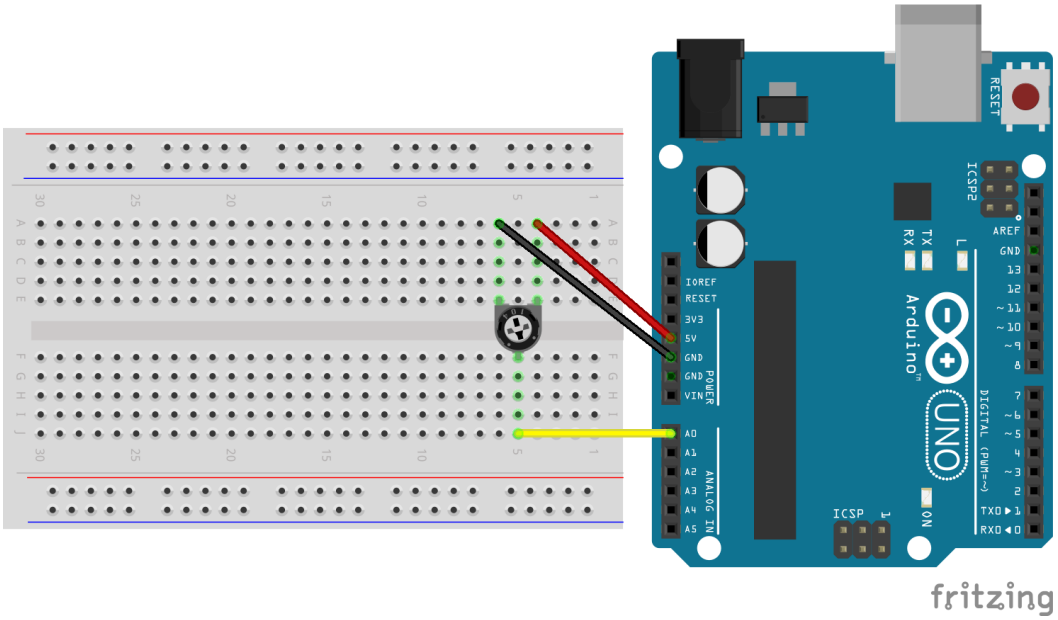
Data ranges:

- Physical input: 0 to 5 V
- Digital values: 0 to 1023 (i. e., 10-bit integer)
- Precision: $\Delta U = \frac{5\ V}{1023} \approx 4.9\ mV$

# Exercise: Potentiometer

Verify the ranges:

- Measure the center connection of a potentiometer.
- Display measured voltages while varying the resistance.
- Repeat the experiment for connecting to 3.3 V instead of 5 V.
- Is the maximum resistance of the potentiometer of relevance?



# Exercise: Potentiometer

Sample solution:

```
#define PIN_INPUT A0

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

void loop() {
  int measured = analogRead(PIN_INPUT);
  double voltage = measured / 1023.0 * 5.0;
  Serial.print(voltage);
  Serial.println(" V");
  delay(200);
}
```