



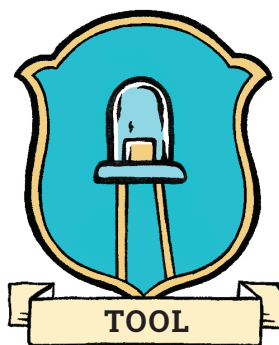
ARDUINO IN 6 CHALLENGES

A collection of sheets to understand how to use Arduino
and to be able to run original projects:
incredible machines, take measurements.
The only limit is your imagination.



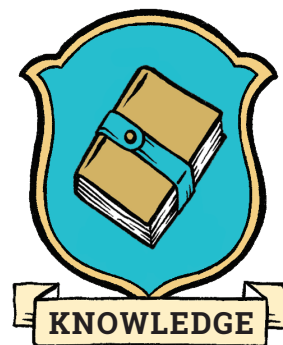
CHALLENGE

Six "Challenge" sheets to discover the
essentials of Arduino boards



TOOL

"Tool" sheets to help you get to
know the equipment



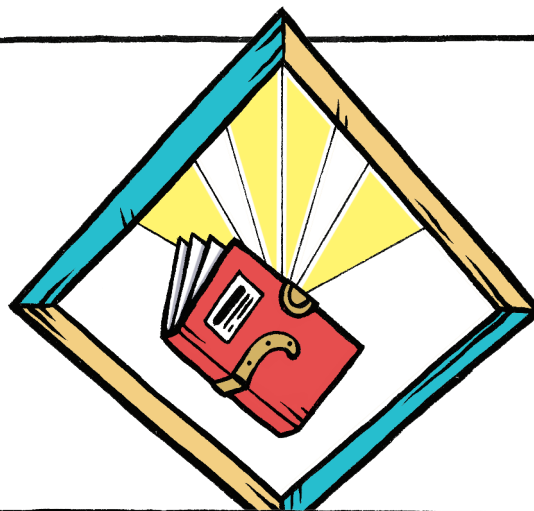
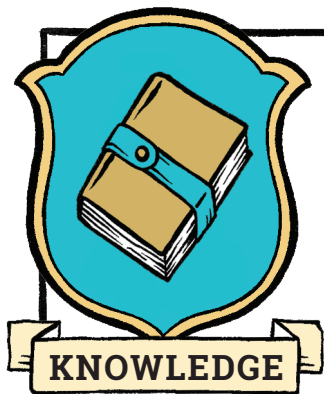
KNOWLEDGE

"Knowledge" sheets to
expand on the basic concepts

Development: **Marine Joumard**
in collaboration with **Frédéric Bouquet** and **Julien Bobroff**,
team **La Physique Autrement**, Université Paris-Sud

— find it online at: www.opentp.fr —

The images of the circuits were created with the open-source software Fritzing



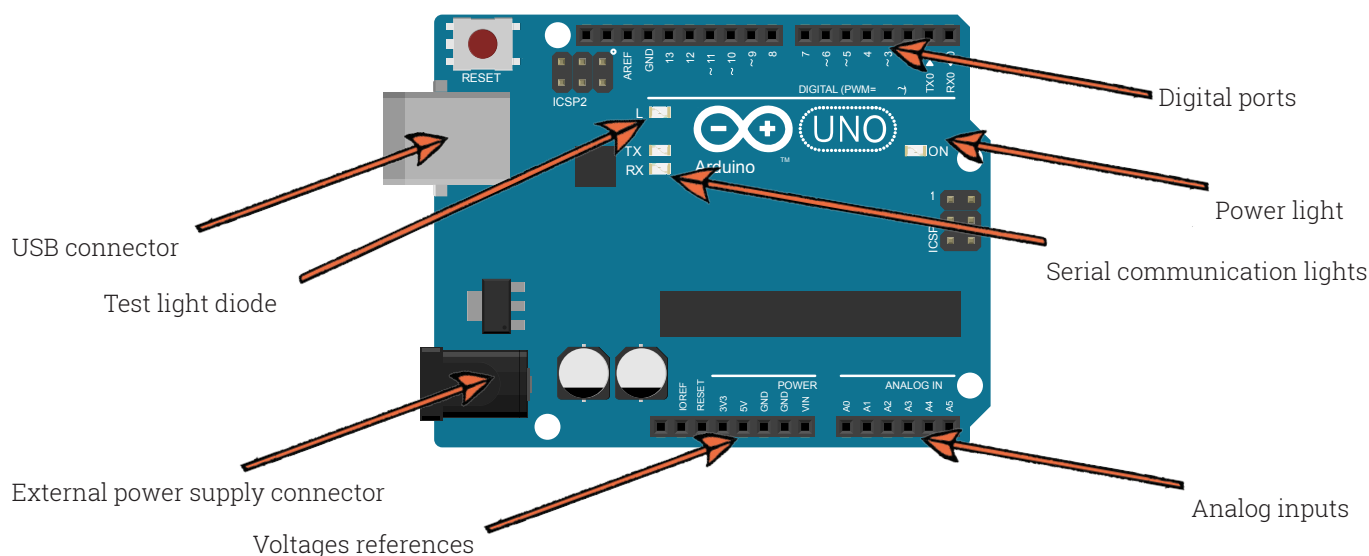
THE ARDUINO BOARD

Understand your board

"Arduino" is a generic term that covers different microcontrollers. Here, we are using the Arduino board Uno which is the most commonly used today although the content presented here is equally valid for the majority of other boards, for example the MEGA Arduino board. The reference site www.arduino.cc is a good starting point to discover the world of Arduino

The Arduino Uno board is not the most powerful microcontroller but its architecture is open-source and its entire philosophy is based on the free model. This board is a microcontroller, which is a system that resembles a computer: it has memory, a microprocessor and interfaces with the outside world. A simple search for the words "Arduino projects" would quickly show the diversity of what can be done using an Arduino board: robots, clocks and even home automation.

The Uno board



External power supply 7 – 12 V: this power supply is useful only if the board is not linked to a computer using a USB cable. It allows nomadic use of the board without a computer.

Digital ports 2 to 13: these ports can be used as input in order to read voltage, or as outputs to produce voltage. They only accept two values, 0 or 5 volts (see sheets “digital inputs” and “digital outputs”).

Ports 0 and 1 are used for serial communication with a computer using a USB cable. It is best to avoid using them in your projects.

Pseudo-analog outputs: they are marked with the symbol ~ and can be used as PWM output to create intermediary voltage between 0 and 5 volts. See “PWM outputs” sheet).

Analog inputs: the ports A0 to A5 can measure electric voltage from between 0 and 5 volts (they cannot create voltage however).

Fixed voltage outputs: these ports create fixed voltage:

- Port 5V creates a voltage of 5 volts;
- Port 3V3 creates a voltage of 3.3 volts;
- The GND ports are connected to ground (0 volts).

Test light diode: it is piloted by the digital port 13 configured as output and allows one to rapidly test the board (see “test your board” sheet).

USB connector: it is used to connect to your computer using a suitable cable. It allows communication with the board and also provides power for it to function.

Power light: it switches on when the board is powered.

Serial communication lights: they light up when the board and the computer communicate.

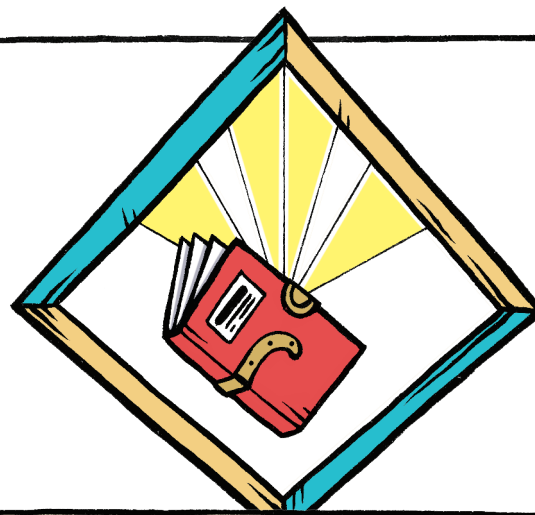
Some precautions

You should avoid connecting two ports that produce different voltages and avoid voltages that are too high:

- Never connect a digital output to ground directly;
- Never connect two digital outputs directly;
- Carefully check the polarity of an external power supply before switching on (or by checking with a multimeter before connecting);
- Never send voltage to fixed output ports (they are marked with 5V, 3V3 and GND);
- Never send more than 5.5 V into an input whether it is analogue or digital;
- Never draw more than 40 mA from a single port and never draw more than 200mA overall.

If you have to connect an output to ground (between port 13 and GND for example), your circuit must contain a resistor of more than 125 Ω , which corresponds to a current of 40 mA. A resistance of 200 Ω is preferable – that is to say a current of 25 mA – to leave a security margin.

All the voltages are measured based on the GND port. “Do not exceed a voltage of 5.5 V on a port” means “do not exceed a difference in voltage of 5.5 V between the port and the port GND”.



ARDUINO IDE SOFTWARE

Indispensable to be able to program your board

The behavior of the microcontroller is determined by its program. In practice, the user writes this program using software (IDE) installed on a computer. This program is then compiled and uploaded into the microcontroller via a USB cable. Many different types of software can be used as IDE although here, we use the Arduino software that can be downloaded on the reference site www.arduino.cc (see the "Test your board" Challenge sheet to install the software).

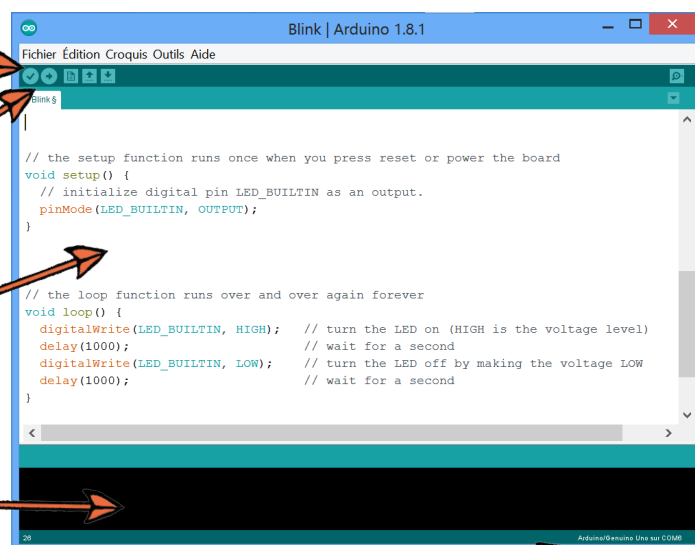
"Compile" icon

"Upload" icon

Programming window

Compiler message window

Indicates which Arduino board and COM port are in use



This software is open and free. The screenshots are out of date, no doubt, as the software is constantly evolving. You will, nonetheless, find these main elements:

Menus to access different settings (File, Edit ...).

Icons allowing shortcuts. The icon "upload" is particularly important as it sends the program to the Arduino board.

A programming window: it is here for you to write the program that will control the Arduino board once it is uploaded.

A message window: when messages in orange appear, it's because there is a problem! In general it is due to a badly written program but sometimes it is due to a faulty connection with the board.

You just have to type the code into the programming window, compile it and upload it to the program on the Arduino board by clicking on the corresponding icons.

Compile: The program is transformed into comprehensible language by the microcontroller: programming errors are indicated at this step.

Upload: The program is sent to the board, which must be connected to the computer using a USB cable (the serial communication lights should blink).

The program is thus installed on the microcontroller and executes (see the "Programming" sheet for information on the different ways of writing a program). If an external power supply is connected to the board, the USB cable can be disconnected.

Useful tools

Here are some functionalities of this software that are useful to know:

Tools menu, Board type: the type of board selected must correspond to your board. If it is not the case, the compilation will not be adapted.

Tools menu, Port: the port selected must correspond to the one to which your board is connected. If it is not the case, uploading will not be possible.

File menu, Examples: many examples are available. Study them and get inspired!

Tools menu, Serial monitor: allows one to receive messages that the Arduino board sends to the serial port. Useful for debugging programs.

Minimum programming requirements

The programs of the Arduino boards must always contain two particular procedures:

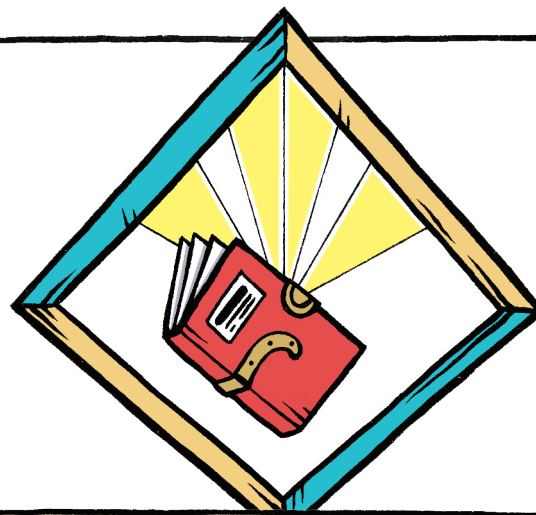
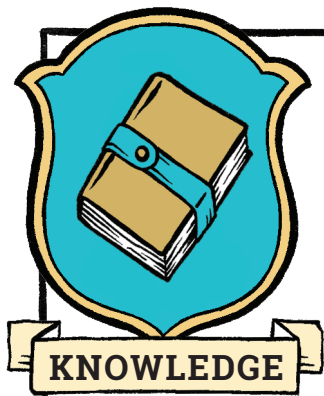
- The `setup()` procedure, which is executed once at the beginning of the program;
- The `loop()` procedure which is executed continuously afterwards.

Here is the minimum program that the Arduino board can accept::

```
void setup() {    // beginning of setup. The text after // is ignored
                  // This procedure is executed only once
                  // The user can insert commands
}                 // end of setup

void loop() {     // start of loop
                  // This loop will execute for ever
                  // after the execution of setup
}                 // end of loop
```

This program does not give any instructions to the microcontroller. The programming sheet provides the most useful instructions but the site www.arduino.cc provides all the instructions that the board can understand as well as numerous examples ("Reference" tab on the site). We advise you to consult this site in case of need.



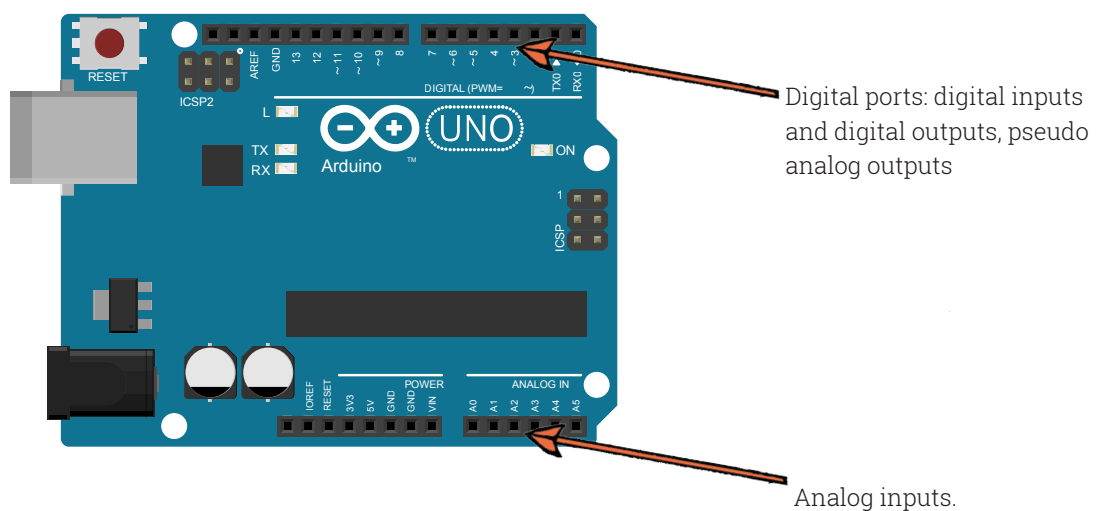
INPUTS & OUTPUTS

Understand how the board interacts with the outside world

The input/output (I/O – input/output) represents the way in which the Arduino board interacts with the outside world. The outputs are controlled by the board allowing the microcontroller program to trigger actions (switch on or switch off a LED, a ventilator or a motor). The inputs are read by the microcontroller which allows it to become aware of the state of the system that it is linked to.

Outputs are voltage sources and are controlled by the board.
Inputs are voltmeters and their measurements are read by the board.

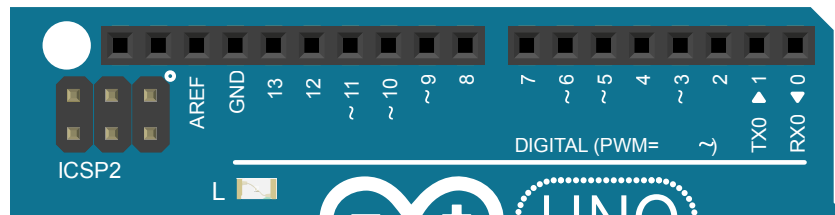
There are two types of I/O: digital I/Os, and analog I/Os.



KNOWLEDGE – INPUTS & OUTPUTS

INPUTS / DIGITAL OUTPUTS

The digital inputs / digital outputs can only take two values, the value LOW (that is to say GND, zero volts), and the value HIGH (that is to say 5 volts). The value of a digital port can therefore be coded on one bit, 0 or 1, true or false.



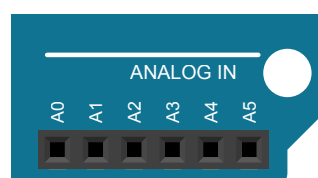
The Arduino board is comprised of 14 digital ports (named DIGITAL on the board), numbered 0 to 13. Each of the ports can be declared as being inputs or outputs in the microcontroller program (see “digital outputs” and digital inputs” sheets).

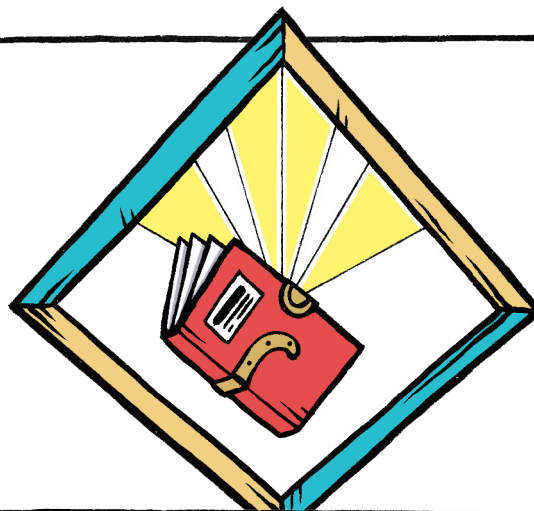
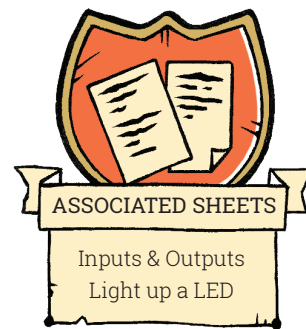
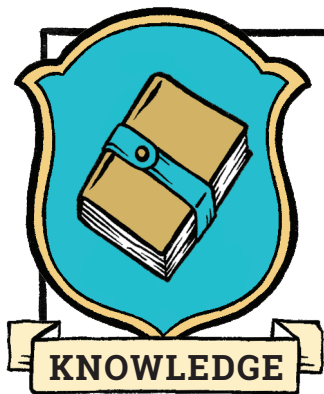
PSEUDO-ANALOG OUTPUTS

The Arduino board does not have a true analog output capable of producing voltage of an arbitrary value chosen by the user. Certain digital ports however can simulate an analog output by using the PWM (Pulse Width Modulation) technique: it concerns ports 3, 5, 6, 9, 10 and 11 (indicated using a ~ on the board). These ports can simulate a voltage between 0 and 5 volts by switching rapidly between LOW (0 V) and HIGH (5 V). The average voltage value then depends on the time spent in the LOW state compared to the time spent in the HIGH state, all controlled by the board (see “PWM output” sheet).

ANALOG INPUTS

An analog input is a kind of voltmeter: the board reads the voltage that is applied to the port. However, the microcontroller only works with numbers: you must therefore transform the applied voltage into a numerical value. It is the work of the analog/digital converter, called “ADC” (see “Analog inputs” sheet.)

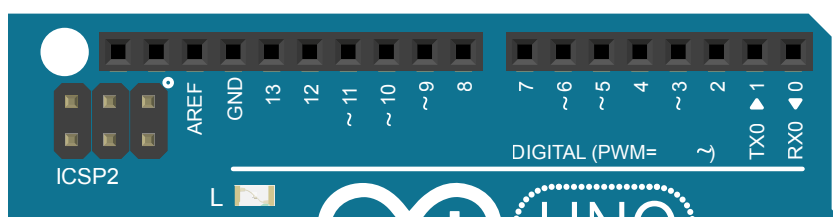




DIGITAL OUTPUTS

How the Arduino board can control a voltage of 5 volts

A digital output is a pilotable source of voltage that can only deliver two values LOW (\sim GND, 0 V), and HIGH value (\sim 5 V).



The Arduino board UNO contains 14 digital ports (called DIGITAL on the board), numbered from 0 to 13. Each of these ports can be declared as being an output in the microcontroller program.

The ports 0 and 1 are reserved for serial communication. You must not use them for anything else.

The port GND is the ground of the board (0 V).

The port 13 is connected to the test LED "L" on the board: when this output is in the HIGH state, the test LED is switched on, in the LOW state, it is switched off.

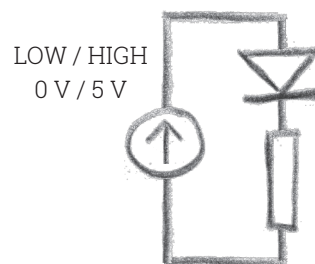
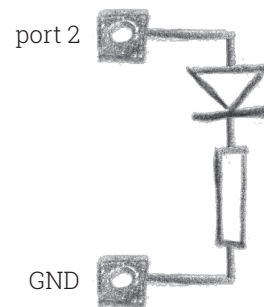
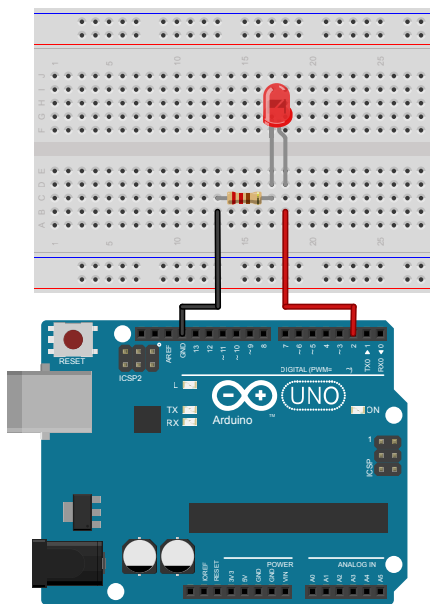
The digital ports 2 à 13 of the board can be used as outputs. If a port is declared as output, it then behaves as a provider of voltage the value of which is controlled by the microcontroller, either 0 or 5 volts.



Be careful, the current that a digital port can deliver as output is limited to 40 mA: drawing more can damage the board! This type of situation can happen if a port, declared as output, is directly linked to ground (port GND) with a weak resistance (a wire), and the program switches the output to HIGH (5 V). The opposite is also dangerous (a digital output linked to a 5 V port toggled onto the LOW value). Check the value of the current that flows in your circuits!

Usage examples

Here is a simple setup to control the state of a LED as well as two equivalent electrical diagrams: port 2 is equivalent to a controlled voltage source that can provide either 0 volts or 5 volts.



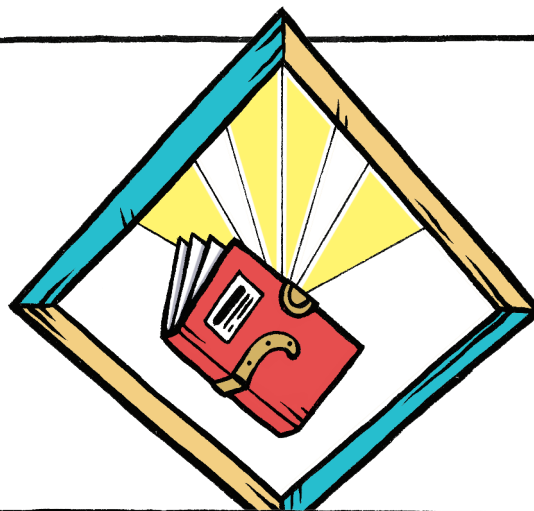
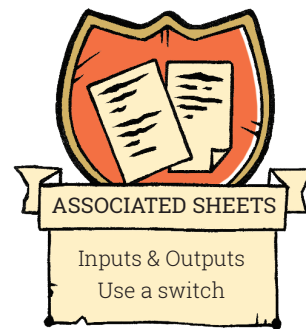
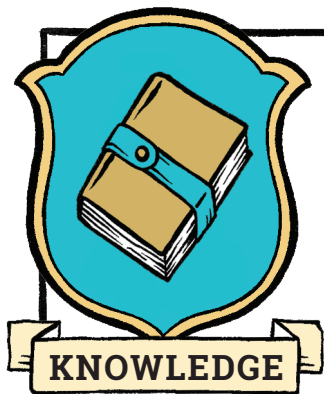
To control a LED, the uploaded program on the microcontroller must define the digital port 2 as output (see Challenge sheet "Light up a LED").
When the program flips the value of port 2 to HIGH, the LED switches on.
When the program switches the value of this port to LOW, the LED switches off.

The resistance is used to limit the current that protects both the microcontroller and the LED (which also has a critical current beyond which it smokes). If the LED is mounted in the opposite sense, it will never switch on (it is a diode).

Programming

```
pinMode(2, OUTPUT) ;    // defines digital port 2 as output
                        // put this instruction into the setup()

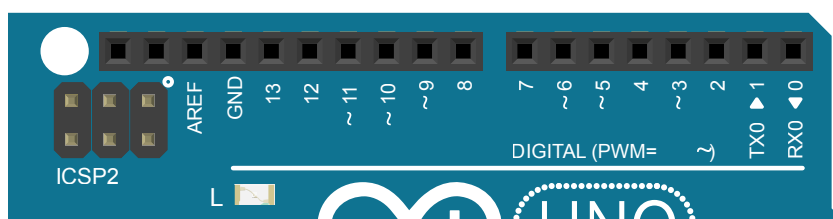
digitalWrite(2, HIGH) ;  // switches the state of digital port 2 to HIGH (5 V)
digitalWrite(2, LOW) ;   // switches the state of digital port 2 to LOW (0 V)
```



DIGITAL INPUTS

How the Arduino board detects a 5 V signal

A digital input is a voltmeter that can only read two values LOW (\sim GND, 0 V), and the value HIGH (\sim 5 V).



The Arduino board UNO has 14 digital ports (called DIGITAL on the board), numbered from 0 to 13. Each of the ports can be declared as being an input in the microcontroller program. The ports 0 and 1 are reserved for serial communication, you should not use them. The GND port is the ground of the board (0 V).

When a digital port is declared as input, the state of the port can be read by the Arduino board (HIGH or LOW) and that value can be used in the program to launch this or that action.

To determine the state of a digital input (HIGH or LOW), the board measures the value of the voltage applied to that port. The voltages that are inferior to around 3 V will be read as LOW, the voltages that are superior will be read as HIGH.



You must avoid intermediary voltages, which risk giving random results.

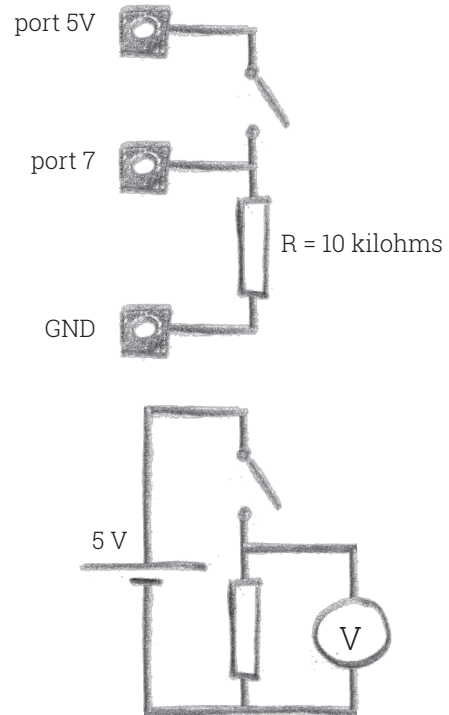
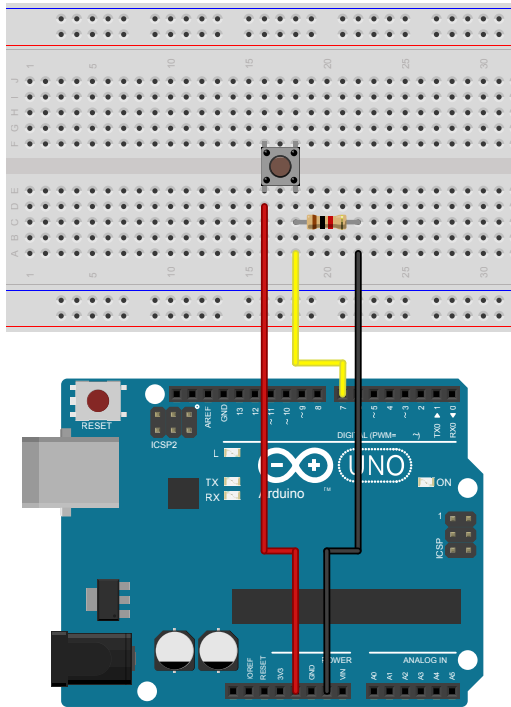
A non-connected digital entry will float (with no affected voltage its state will remain undetermined). The reading of this port can give almost any result.

A voltage superior to 5.5 V can destroy the Arduino board.

KNOWLEDGE – DIGITAL INPUTS

Usage examples

Here is a simple setup to read the state of a switch (open or closed) as well as two equivalent electrical diagrams: the port 7 is equivalent to a voltmeter that can only measure 0 volts or 5 volts.



When the switch is open, port 7 and the ground are connected and are at the same electrical potential because no current is circulating: the board measures 0 V (LOW). When the switch is closed, the potential of the input 7 rises to 5 V (HIGH) and a current of 0.5 mA is circulating between the 5V port and the GND port, which poses no problem.

The resistor ensures that the measured voltage on port 7 is indeed that of the GND port when the switch is open. Without this resistance (called pull-down), the port 7 would be floating and its measurement would be indeterminate.

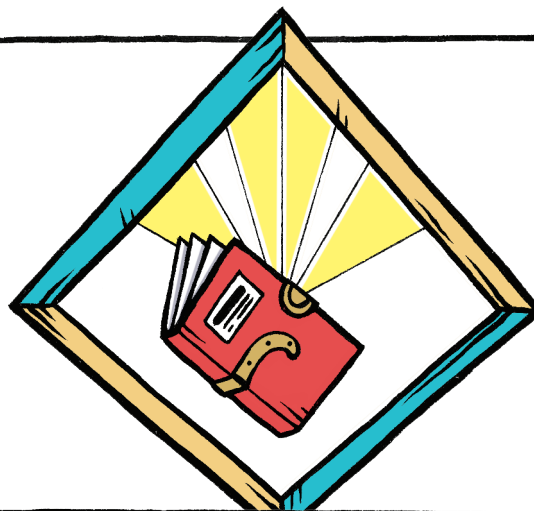
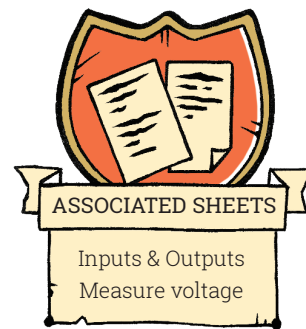
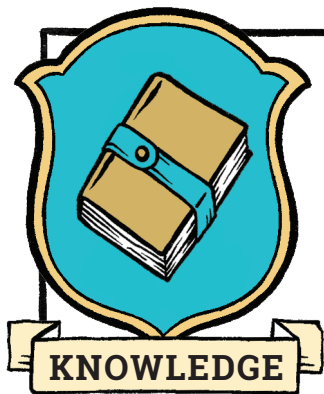


Be careful: if port 7 is declared by error to be the output instead of input in the program, there is a risk of a short circuit when the switch is closed

Programming

```
pinMode(7, INPUT) ;           // defines digital port 7 as input
                                // put this instruction into the setup()

buttonstate = digitalRead(7) ; // returns the state of input 7 (true or false,
                                // 0 or 1, HIGH or LOW) and attributes it to the buttonState variable
                                // (used here as an example).
```

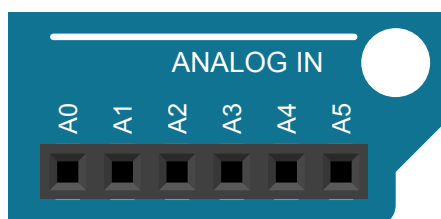


ANALOG INPUTS

How the Arduino board measures
electrical voltage

The Arduino board Uno has 6 analog inputs, A0, A1, ..., A5.

An analog input is a voltmeter: the board measures the difference in potential between the wire that connects the analog input and the potential of the GND port.



That being said, the microcontroller can only work with numbers; it will therefore convert the measured voltage into a number. It is the work of the analog/digital converter called "ADC". The ADC of the Arduino board works on 10 bits: It accepts as input a voltage comprised of between 0 V and V_{ref} (a reference voltage) and supplies the microcontroller with a number comprised of between 0 and 1023 (that is to say $2^{10} - 1$).

- For a voltage of 0V (or less), the ADC returns the value 0.
- For a voltage of V_{ref} (or more), the ADC returns the value 1023.
- For an intermediary voltage, the ADC returns an integer of between 0 and 1023 following a linear scaling law.

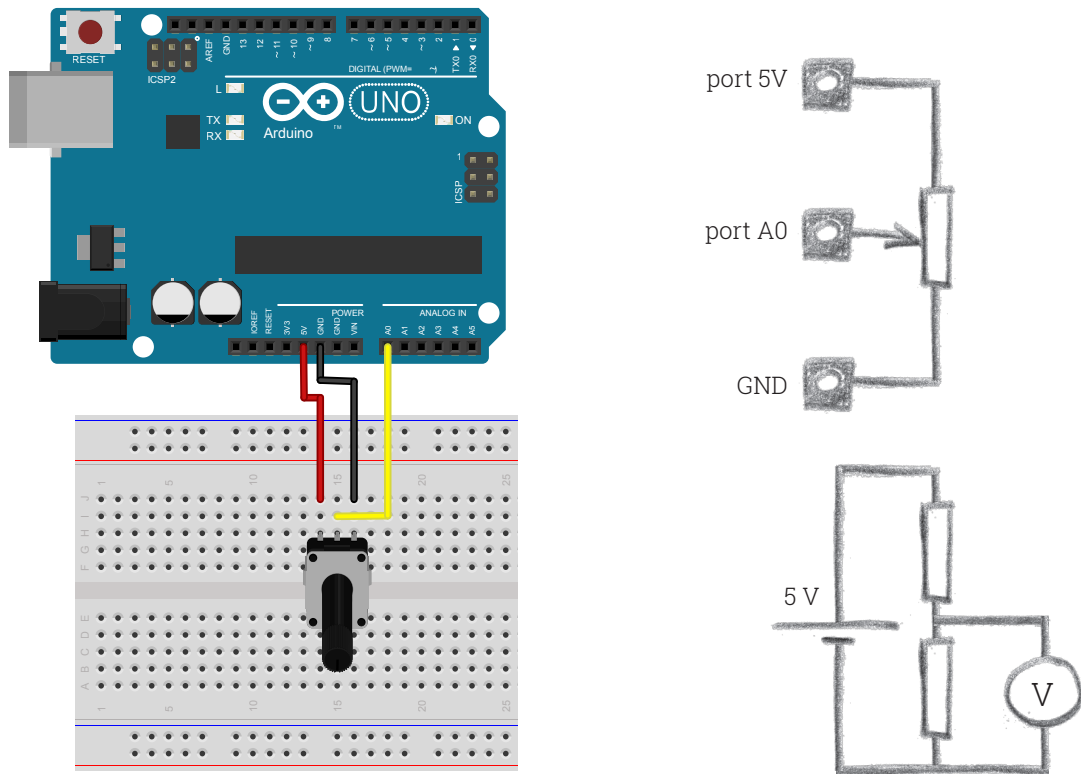
The V_{ref} is 5 V by default but this value can be changed in the program (see the instruction "analogReference" on the arduino.cc site for more details).



Be careful, a voltage superior to 5.5 V can destroy the Arduino board.

KNOWLEDGE – ANALOG INPUTS

Usage examples



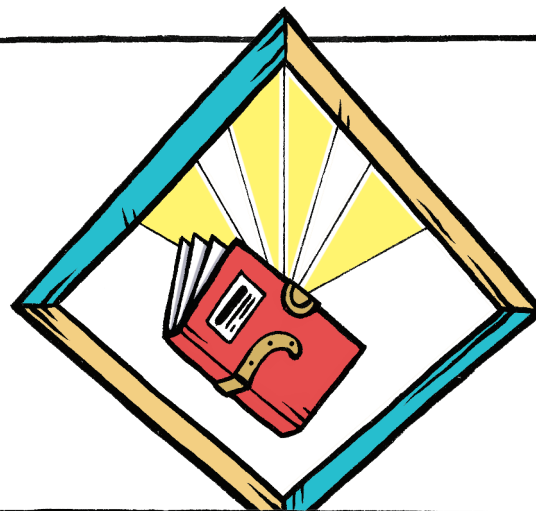
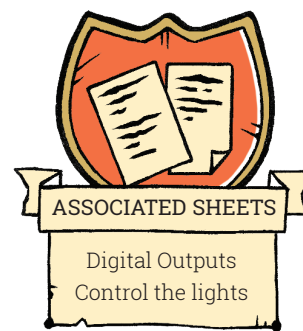
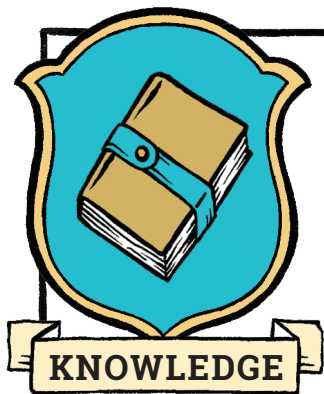
The setup above allows you to measure a voltage modified by a potentiometer comprised of between zero and five volts. This setup is equivalent to a voltage divider (see the equivalent diagram).

Programming

```
int sensorValue ;           // integer type variable
float VoltageInVolts ;      // decimal type variable

sensorValue = analogRead(A0); // analogRead(A0) is the function that
// returns an integer of between 0 and 1023, corresponding
// to the voltage applied to the port A0 ;
// sensorValue is an integer comprised between 0 and 1023.

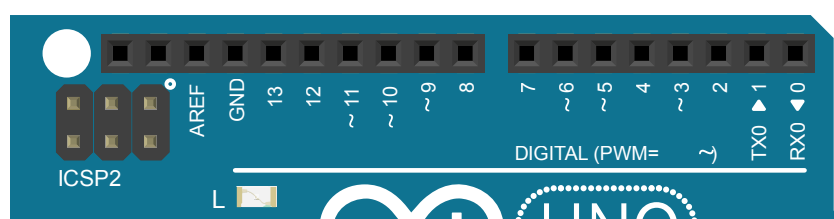
VoltageInVolts = analogRead(A0) * 5.0 / 1023.0 ;
// converts the value read by the ADC in volts,
// supposing that Vref is 5 V (the default value).
```



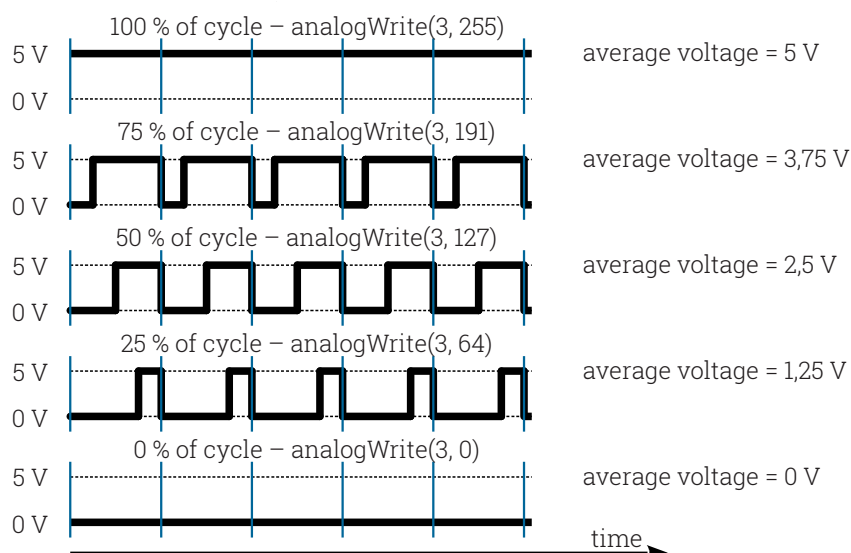
PWM OUTPUTS

Understand "pseudo analog" outputs

An analog output is a controllable voltage source. The Arduino board does not have a real analog output capable of producing a voltage value chosen by the user. Certain digital ports can, however, simulate a digital output by using the PWM (Pulse Width Modulation) technique: it concerns ports 3, 5, 6, 9, 10, and 11 (marked with a ~ on the board).



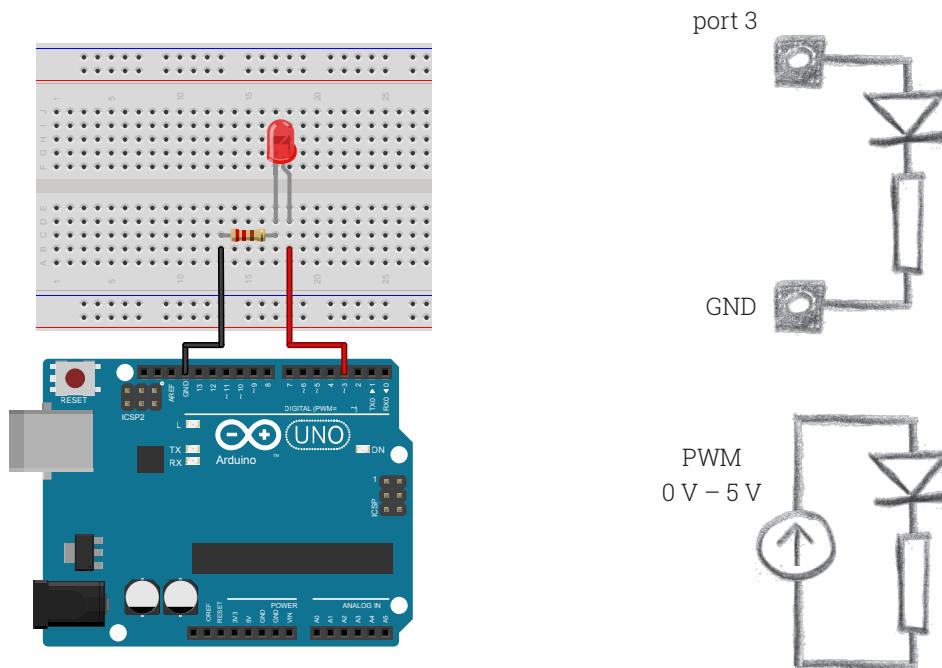
These ports can switch rapidly between their LOW (0 V) state and HIGH (5 V) states. By controlling the relationship between the time spent in the HIGH state compared to the time spent in the LOW state, the Arduino board can vary the average voltage value on the ports between 0 and 5 volts. The diagram below illustrates the principal representing the variations over time of the voltage of port 3 using the PWM output for different required voltages.



KNOWLEDGE – PWM OUTPUTS

Usage examples

Be careful: Unlike a real digital output, the PWM does not deliver a constant voltage but a voltage that is constantly oscillating. For certain applications, a PWM output is perfectly adequate; for example, to power a motor or to vary the intensity of a LED. The following circuit allows one to vary the power of a LED based on the value required at port 3 defined by the program as analog output.



If one needs a real continuous analog voltage, one must install a low-pass filter that will eliminate the high frequencies and only keep the average value (the frequency of oscillation of the PWM output is around 500 to 1000 Hz depending on the ports).

Programming

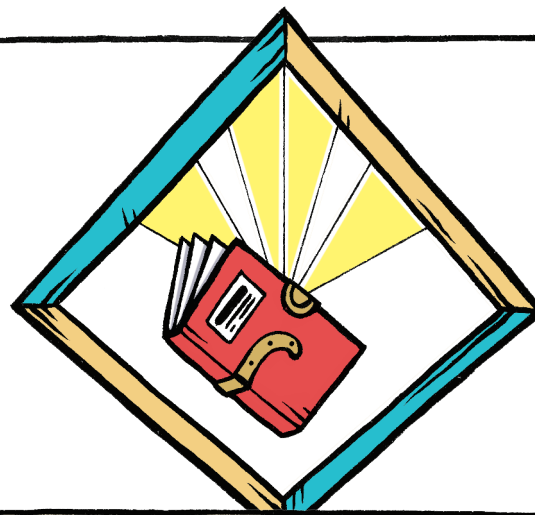
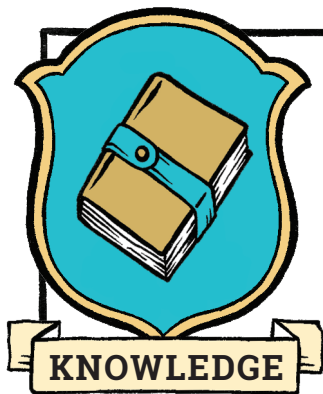
The board can control the ratio of the PWM cycle with a sensitivity of 8 bits: the command in the "analogWrite" instruction is an integer of between 0 and 255 ($= 2^8 - 1$), corresponding to an average voltage of between 0 and 5 volts.

```
int pwmValue ;           // integer type variable
float VoltageInVolts = 3.1 ; // decimal type variable
pinMode(3, OUTPUT) ;     // defines digital port3 as output
                           // put this instruction into the setup()

analogWrite(3, 0) ;       // Creates an average voltage of 0 V on port 3
analogWrite(3, 127) ;     // Creates an average voltage of 2.5 V on port 3
analogWrite(3, 255) ;     // Creates an average voltage of 5 V on port 3

pwmValue = VoltageInVolts * 255 / 5 ;

analogWrite(3, pwmValue) ; // Creates an average voltage on port 3
                           // of value VoltageEnVolts
```



PROGRAMMING

Useful instructions

The `arduino.cc` includes the instructions that the board can understand (<https://www.arduino.cc/en/Reference/HomePage>). We advise you to go to the site if you need to; this sheet is by no means exhaustive. Do not hesitate to consult the available examples in the File menu of the Arduino IDE software either

VARIABLES

The variables are given a name. They must be declared by specifying the type of data that they will store. During the declaration, the variables can be initialized but it is not obligatory. Some examples:

```
int MyVariable1 ;      // declares the variable "MyVariable1" as type "int"
int MyVariable2 = 0 ;  // declares the variable "MyVariable2" as type "int"
                        // and initializes the value as 0

int integer1 ;         // integer, from -32768 to 32767
unsigned int toto ;    // positive integer, from 0 to 65535
float measure1 = 0.0 ; // real number
boolean flag = true ;  // boolean, true or false
int tableauInt[6] ;    // integer array with 6 elements, numbered 0 to 5
```

The variables can only be used in the procedure in which they are declared. For a variable to be defined in a global manner in the whole program, you must declare it at the beginning of the program before the `setup()` procedure.

INPUTS AND OUTPUTS

The digital ports used in the program must be declared as inputs or as outputs in the `setup()` procedure:

```
pinMode(3, OUTPUT);    // declares digital port 3 as output
pinMode(4, INPUT);     // declares digital port 4 as input
```

To define the state of a digital output:

```
digitalWrite(3, HIGH); // imposes the HIGH value(5 volts) to port 3
digitalWrite(3, LOW);  // imposes the LOW value(0 volts) to port 3
```

KNOWLEDGE – PROGRAMMING

For a pseudo-analog output (digital in PWM mode):

```
analogWrite(6,0) ; // imposes the value 0 to port 6 (0 volts)
analogWrite(6,255) ; // imposes the value 255 to port 6 (5 volts)
analogWrite(6,100) ; // imposes the value 100 to port 6 (average of 2 volts)
```

To read a digital input:

```
flag = digitalRead(4) ; // reads the HIGH or LOW value of digital port
                        // and attributes the variable flag
```

To read an analog input:

```
Measure1 = analogRead(A5) ; // measures the voltage on port A5
                        // an integer between 0 and 1023
```

USB CABLE COMMUNICATION

You must define the transfer speed in the setup() function

```
Serial.begin(9600); // initialises the serial port, speed of 9600 bauds
```

The board can then send information to the computer:

```
Serial.print("Measure: "); // the board sends the text
Serial.print(variable) ; // the board sends the value of the variable
Serial.println("toto"); // the board sends the text
                        // and adds a carriage return character
```

USEFUL INSTRUCTIONS

```
delay(100) ; // wait in milliseconds
unsigned long time1 = millis() ; // measures the time in milliseconds
unsigned long time2 = micros() ; // measures the time in microseconds
```

A for loop (this loop executes 100 times; the index variable varies between 0 and 99, increasing by one unit for each iteration):

```
for (int index=0; index < 100; index++){
    // instructions for the loop to execute 100 times
}
```

A while loop (in the example, the loop executes as long as the index variable is less than 3):

```
while (index < 3) {
    // instructions for the loop to execute as long as index < 3}
}
```

An if condition (here on the value of the buttonState variable of boolean type):

```
if (buttonState == HIGH) {
    // instructions to write here if buttonState equals HIGH;
}
else {
    // instructions to write here if buttonState equals LOW;
}
```