Requirements for this module

Offline	Online
Arduino	—
Breadboard	—
Jumper cables	—
Resistors (220 Ω; 330 Ω; 1 kΩ; 10 kΩ)	—
Potentiometer (1 kΩ; 10 kΩ)	—
Diode	—
2x LDR	—
LED (red, yellow, green)	—
RGB LED	—
Servo motor	—
DC Motor	—
5x Push button	—
2N2222 transistor	—
LCD	—
Active piezo element	—

Prerequisite knowledge

Ohm's law	$R = \dfrac{U}{I}$
Components in series	equal current
Components in parrallel	equal voltage
Power	$P = U \cdot I = I^2 \cdot R$
Alternator / electric motor operation	

Introduction

Nice that you are getting started with Arduino! Perhaps you have no idea what an Arduino is and what you can do with it. An Arduino is a kind of microcomputer. You can connect the Arduino to a computer from where you send a program (in Arduino they call it a sketch) to the Arduino. The Arduino will then run your written script and take care of the output. For example, you can keep a refrigerator at a certain temperature, control a self-driving robot, make light sensors, control an LCD screen on your shirt and so on. It is also possible not to use software and only play with the electronics. The Arduino is then the voltage source.

In this module we will work with the Arduino and learn the basic capabilities of an Arduino. When working with an Arduino you have two important parts: the Arduino and a breadboard. The Arduino is the computer, with input and output capabilities. You connect the electronics that are controlled by the Arduino to the breadboard.
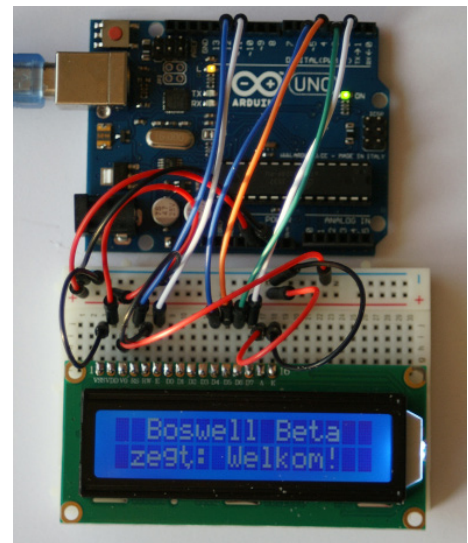
Breadboard

The breadboard has two columns on both sides that are connected to the power supply (+ and -). The + side is connected to the 5 V output or to an output port of the Arduino. The - side is connected to the GND (ground) of the Arduino. Although you do not always use the constant output, it is wise to always connect it.

The breadboard has rows and columns. The figure to the right shows a breadboard, and below it you see how the connections are made internally. The points in a row are connected. But let's not dwell on this too long…let's get started!

NOTE: Often you have to indicate to which USB port your Arduino is connected to your computer. You do this by going in the program to: Tools / Port. There you can click on the correct USB port.
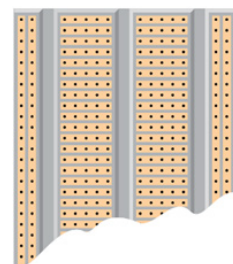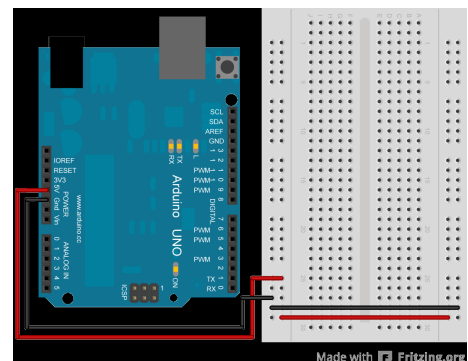
Assignment 1 Connecting an LED

An LED is a diode that emits light when current flows through the LED. The LED only allows current to flow in one direction. The long leg of the LED must always be connected to the + side, the short leg to the - side, see the figure on the side.
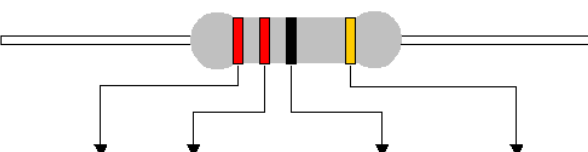
The current through an LED can usually only be 20 mA. The voltage across the LED is then approximately 2.0 V, these values differ a bit per type of LED, see table below. The Arduino supplies a 5.0 V voltage. The LED must therefore be connected in series with a resistor. The resistance must therefore be at least 150 Ω ($R = \frac{U}{I} = \frac{3.0\,V}{0.020\,A} = 150\,\Omega$).

1. There is no resistor with 150 Ω, but there is one of 220 Ω. Look up the resistor with the help of the color code: the first ring must be red, the second ring also and the third ring brown ($22 \cdot 10$). Another possibility is red, red, black, black ($220 \cdot 1$).

2. Now connect the 5 V output of the Arduino to the + column and the GND (ground) output of the Arduino to the - column.

3. Connect the LED and the resistor in series, see the drawing.

4. Connect the Arduino to the computer via the USB. If you did it correctly, the LED will light!
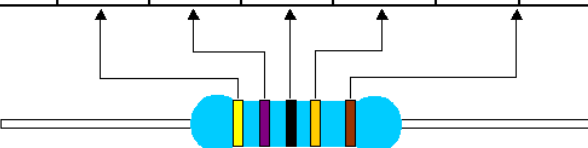
Because the red LED emits light at a lower voltage than, for example, a green LED, you can use a smaller resistance for green and blue LEDs. In this way they burn just as brightly!

Colour	Threshold voltage
Blue	2.3 V
Green	2.0 V
Red	1.5 V

| COLOR	1ST BAND	2ND BAND	3TH BAND	MULTIPLIER	TOLERANCE	
BLACK	0	0	0	1		
BROWN	1	1	1	10	± 1%	F
RED	2	2	2	100	± 2%	G
ORANGE	3	3	3	1K		
YELLOW	4	4	4	10K		
GREEN	5	5	5	100K	± 0.5%	D
BLUE	6	6	6	1M	± 0.25%	C
VIOLET	7	7	7	10M	± 0.10%	B
GREY	8	8	8		± 0.05%	A
WHITE	9	9	9			
GOLD				0.1	± 5%	J
SILVER				0.01	± 10%	K
PLAIN					± 20%	M
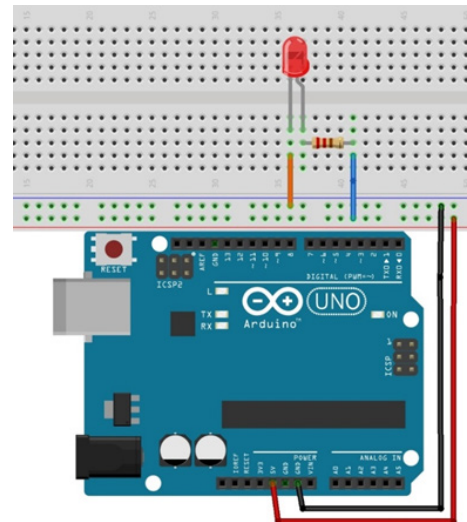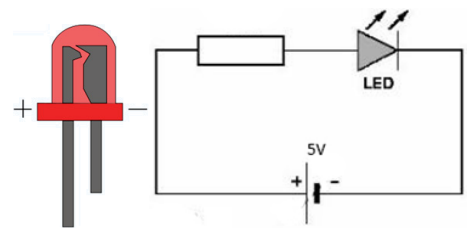
Assignment 2 A blinking LED

Now you can light your LED, but that won't be of much use to you yet…The next step is to control the LED using a piece of code. To control the LED using code you need to connect an output port (for example pin 13) of the Arduino to your LED. You don't need the 5 V output now, pin 13 now supplies the voltage. In the drawing you can see that the 5 V output is still connected. With larger projects you quickly forget the 5 V, that's why it's smart to always connect it!

a) Build the setup you see in the drawing and connect the Arduino to the computer.

We want to control the Arduino, we use the Arduino program for that.

a) Open the program and open the script blink via: file / examples / base / blink.

b) Check the script with the check mark. If the program does not work, it will display an error message at the bottom (you may have to assign the COM port, you do this via tools / port).

c) Upload the script to your Arduino using the button 🔘 (shortcut: ctrl + u)

d) Briefly describe what you see. Try to explain what you see using the code.

e) Change the script so that the LED flashes faster.

f) Connect three different LED lights, change the script so that they turn on sequentially.

g) Change the script and create a traffic light with orange on for a short time.

4
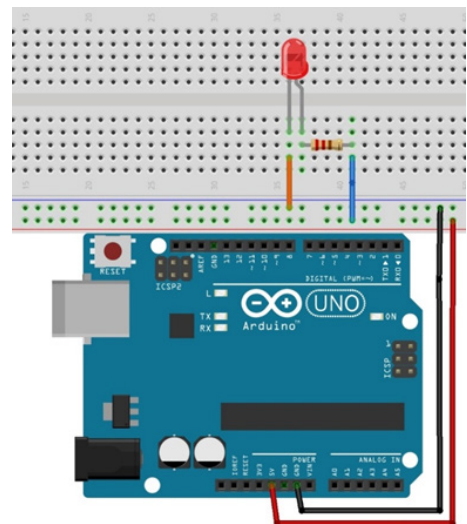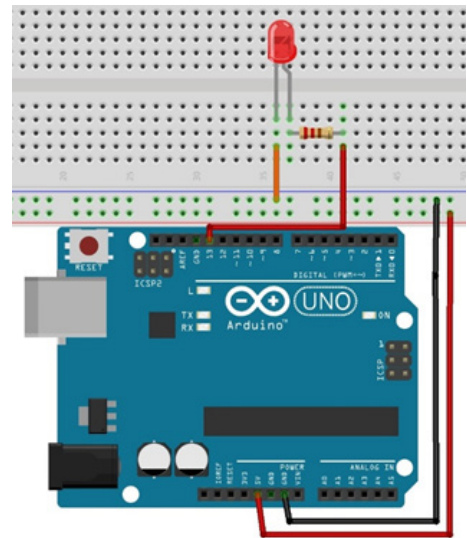
Programming part 1

Arduino has its own programming language based on C++. If you already have experience with programming and/or with C++ (or Java), programming is very easy. And if you have no experience with programming? It's much less difficult than you might think!

At the top of the script Blink the first character is /*, which means that everything after this starting character and before the closing character */ is commented out. The code in between these characters is ignored by the program. Here you put the name of the program, what it does, who made it and when you last changed it.

```
Blink §

1  /*
2    Blink
3    Turns on an LED on for one second, then off for one second, repeatedly.
4    Most Arduinos have an on-board LED you can control. On the Uno and
5    Leonardo, it is attached to digital pin 13. If you're unsure what
6    pin the on-board LED is connected to on your Arduino model, check
7    the documentation at http://www.arduino.cc
8
9    This example code is in the public domain.
10  */
11
12
13 // the setup function runs once when you press reset or power the board
14 void setup() {
15   // initialize digital pin 13 as an output.
16   pinMode(13, OUTPUT);
17 }
18
19 // the loop function runs over and over again forever
20 void loop() {
21   digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
22   delay(1000);              // wait for a second
23   digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
24   delay(1000);              // wait for a second
25 }
```

A second possibility to add comments is with the help of //. Everything on the same line after // is a comment. This way you can keep track of what the code does on that line. With the help of the shortcut ctrl+/ you can quickly comment out an entire line of code.

Before the most important code is executed, you must say what exactly is being controlled. First we tell the Arduino that there is something on pin 13 and that the Arduino should give an output (voltage). It would be a bit neater to first define the name of pin 13 before setup: you give the pin a name (int LEDred = 13;). If you then want to control pin 13, you can do that with the name LEDred.

Everything in between the curly brackets {} of the loop is repeated continuously. First pin 13 is made high (5.0 V) using the code digitalWrite (digitalWrite can only be on or off). Then the program must wait 1000 ms before executing the next line of code. The next line of code makes pin 13 low again (0.0 V). Then the program must wait 1000 ms again, before it starts the loop again.

The pin is controlled with a high or a low signal. Is there anything else in between? Yes and no ...The output is always 0.0 V or 5.0 V. But you can dim the LED by only turning on the LED for a certain time. If the LED flashes fast enough, you will not see the LED flashing, it just seems that the LED is less bright. If you want to dim an LED, use an output with the symbol ∼. This is a Pulse Width Modulation (PWM). The value of the PWM is between 0 (fully off) and 255 (fully on).

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

```
void loop() {
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```
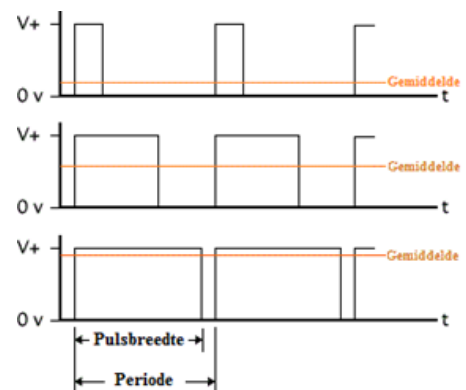
Assignment 3 A dimmable LED

a) Build the setup on the right. Use a PWM pin for the output, for example Pin 9 (note, the LED does not need to be connected to the constant voltage, the voltage is now supplied by pin 9).

b) Open the script Fade in the examples / base and upload the script.

c) Describe what you see and try to explain what is happening with the help of the code.

d) Change the code so that the LED lights up faster and is off faster. Note, there are two ways! Try them both.

e) `analogWrite` is used in the code. Previously we used `digitalWrite`. Explain why this is not possible now.
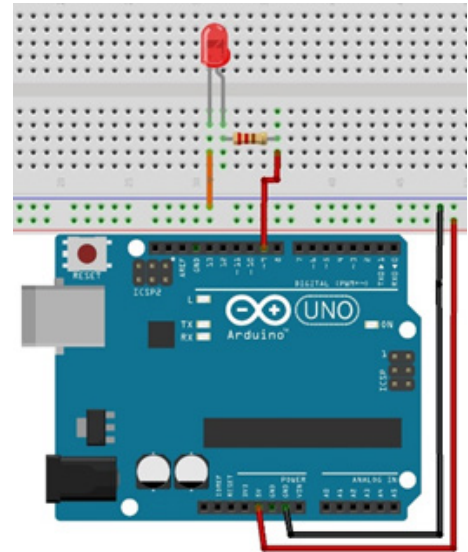
Assignment 4 An on and off button for the LED

We can now turn the LED on and off with the help of the code and even dim it. But often you also want to be able to switch a circuit on and off manually. We need a push button for that.

a) Build the circuit on the right. Make sure you use a large resistor so that the supplied current is limited, because an Arduino is bad at supplying large currents.

The idea is now that we use pin 2 as INPUT. Pin 2 measures the voltage at that point (compares it with 0.0 V). This is done with the code `digitalRead()`. It can now measure a high (5.0 V) or low (0.0 V) signal. With an analog pin such as A0 to A5, intermediate volages can also be measured, with 10-bit resolution. Pin 2 should only measure a voltage when the button is pressed.
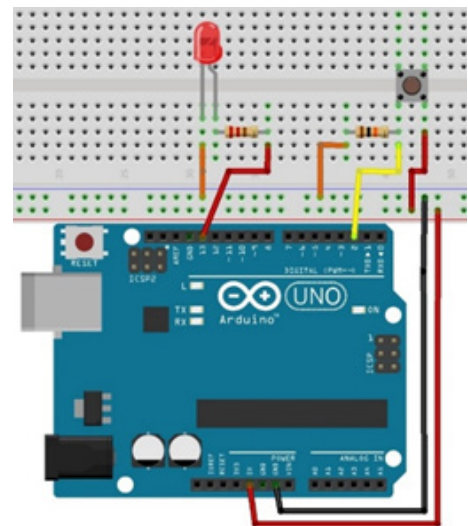
a) Open the Button script via examples / digital and upload the script.

b) Press the button and check when the LED is off and when it is on.

c) Adjust the script so that the function of the button is reversed.

NOTE: There is something special going on with the button. A button posses so-called Bounce. This means that the voltage does not go directly from 0 V to 5 V but goes up and down again. This is because there is a spring in the button that goes up and down. When using and reading the button it is therefore useful to build in a delay...
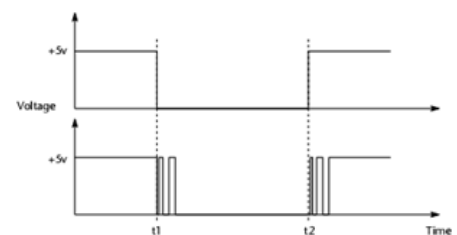
Programming part 2

The Button script is more comprehensive than we've seen before. We walk through the script step by step.

```
const int buttonPin = 2;
const int ledPin = 13;
```

We are dealing with two pins on the Arduino where something has to be done. Pin 2 is an input and pin 13 must be an output. The pin does not change, it is a constant (const). It is an integer value (int). We give pin 2 a recognizable name, pin 2 is now called buttonPin. Pin 13 has been named ledPin.

```
int buttonState = 0;
```

We will soon want to know what the "state" of the button is (pressed or not). It can be 1 or 0. First we have to tell the Arduino that we want to later know the state of the button (create a variable). Before we run the script, the buttonState equals 0.

```
void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT
      );
}
```

As mentioned, we have to tell you that the ledPin is an OUTPUT and the buttonPin is a INPUT. Now the Arduino knows that too...

But something must happen when the button is pressed. At the start of the loop, the button is read (actually the voltage across the resistor is measured to see if it reads 5.0 V). Subsequently an if statement follows. If (if) the button is pressed (buttonState == HIGH) then the LED should light (digitalWrite (ledPin, HIGH);). In all other cases (else), the LED should not be lit (digitalWrite (ledPin, LOW);).

```
void loop() {
  buttonState =
      digitalRead(buttonPin
      );
  if (buttonState == HIGH)
      {
    digitalWrite(ledPin,
      HIGH);
  }
  else {
    digitalWrite(ledPin,
      LOW);
  }
}
```

It's an easy if statement. You can also set multiple conditions. If you have to press two buttons at the same time before the LED turns on, put && between the characters
(if buttonState1 == HIGH && buttonState 2 == HIGH)).

You can also use the character ||. Then either one condition must apply or the other. Unfortunately, the light does not yet stay on when you have pressed the button. But you can change the code to make sure that the light can be switched on and off with one button.

```
void loop() {
  buttonState =
      digitalRead(buttonPin
      );
  if (buttonState == HIGH
      && state == LOW) {
    state2 = HIGH;
  }
  if (buttonState == HIGH
      && state == HIGH) {
    state2 = LOW;
  }
  state = state2;
  digitalWrite(ledPin,
      state);
  delay (100);
}
```

Don't forget to define the new variable (int) at the beginning! There are even simpler ways, these are only shown as script. Explain how they work!

```
if(buttonState == HIGH){i
    ++;}
digitalWrite(ledPin,i%2);
```

```
digitalWrite(13,!
    digitalRead(13));
```

Assignment 5 Securing a safe

The bank's safe is secured. The safe does not open (the LED will only light up) when two keys are turned (buttons pressed). One keyhole is in the warden's room, the other is next to the safe.

Build the corresponding circuit and program the script so that the safe does not open until both keys are turned at the same time.

Assignment 6 A two-way switch

When you are at the bottom of the stairs you want to turn on the light at the top of the stairs. There is another light switch at the top of the stairs. With that you can turn the light off or on again. Build the circuit with two buttons and write the program so that you have a working two-way circuit. Does this not work immediately? Then try it first with a single button.

Assignment 7 A pedestrian crossing

A special transition area has been created for pedestrians. This place is on a busy road. If there are no pedestrians, the traffic light in front of the cars is green. The moment a pedestrian wants to cross, he presses the button. The traffic light in front of the cars then changes from green to yellow to red. Pedestrians then have 15 seconds to cross. The traffic light in front of them is green! Then the green light will blink five times and change to red. A second later, the traffic light in front of the cars turns green again.

Make this system.

Electronics part 1

With only a button and an LED you can quickly become bored, and we have hardly scratched the surface of what an Arduino can do…We want to control robots, prevent collisions of racing cars with walls, or to automatically control greenhouses, and so on. In order to achieve that we need sensors. Almost all sensors work the same, the resistance value changes with a changing input (for example light or force). We are going to start with a light sensor. For a light sensor we need an LDR, see the photo. LDR stands for Light Dependent Resistor. The resistance value changes with changing light intensity, see the graph.

We connect the LDR in series with an ohmic resistor (a resistor with a constant resistance value). This way we can create a voltage divider. Part of the voltage is across the LDR and part of the voltage is across the resistor. All we have to do is read the voltage across the LDR and we know how how much light shines on the LDR!

It is important to know a little more about how a voltage divider works. The LDR and the ohmic resistor are in series so that: $R_{\text{total}} = R_{\text{LDR}} + R_{\Omega}$. The source voltage is 5.0 V, and because the resistors are connected in series, the current is the same everywhere in the circuit.

You calculate the current with: $I = \frac{U_{\text{source}}}{R_{\text{total}}}$.

The voltage is distributed so that the largest resistance receives the largest voltage. If you combine this information you will see that the ratio between the voltages is equal to the ratio between the resistors:
$\frac{U_{\text{LDR}}}{U_{\Omega}} = \frac{R_{\text{LDR}}}{R_{\Omega}}$.

When the LDR picks up more light, the resistance value of the LDR decreases: there will be a lower voltage across the LDR but a higher voltage across the ohmic resistor. The voltage across the LDR ($U_{\text{LDR}} = 5.0\,\text{V} \cdot \frac{R_{\text{LDR}}}{(R_{\text{LDR}} + R_{\Omega})}$ can be measured using the ANALOG IN of the Arduino, so that the voltage is then a measure of the measured light intensity.

Depending over which range your sensor should be sensitive determines the choice of an ohmic resistor. See the second graph. The LDR has a value between $10\,\text{k}\Omega$ and $20\,\text{k}\Omega$: so choose a large resistor!
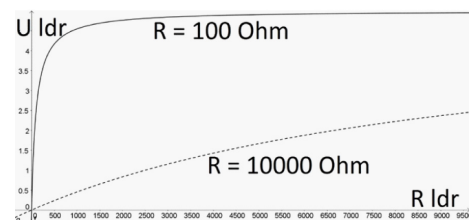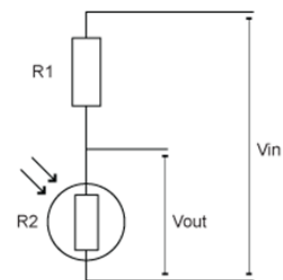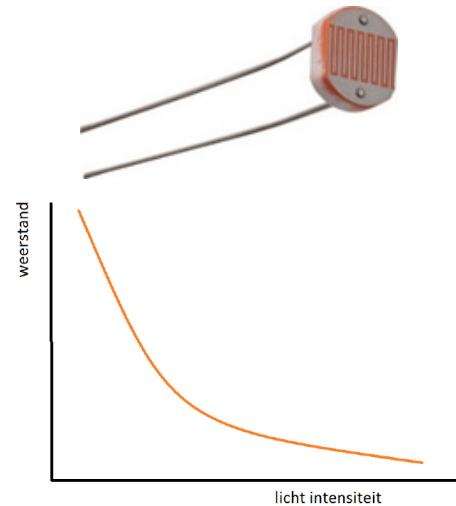
Assignment 8 A light sensor

The circuit is very similar to Assignment 4. Only now we use an LDR and an ANALOG IN.

a) Build the circuit.

b) Open the analogReadSerial script: examples / base / Analogreadserial.

The most important code can be found below.

```
void loop() {
  int sensorValue = analogRead(A0);
  Serial.println(sensorValue);
  delay(10);
}
```

The Arduino is instructed to read out the analog port (`analogRead (A0)`). This value is linked to the variable `sensorValue`.

Next we want to know this value. The value is then also printed for us (`Serial.println (sensorValue);`), which can be read using the serial monitor (the magnifying glass at the top right). For stability it is good to build in a `delay`.

a) Upload the script to the Arduino and read the values using the Serial Monitor.

b) Cover the LDR with your hand. Does the given value change?

c) Combine assignment 3 and this assignment. Make sure the LED lights up brighter when it gets darker.

d) Expand the circuit further so that you can switch the entire circuit on and off with a button.

e) Using graph 2, explain that a large value of R_Ω makes the LDR almost equally sensitive throughout the range.

f) Think of a way to calibrate the sensor yourself. So let the sensor itself determine the minimum and maximum value.
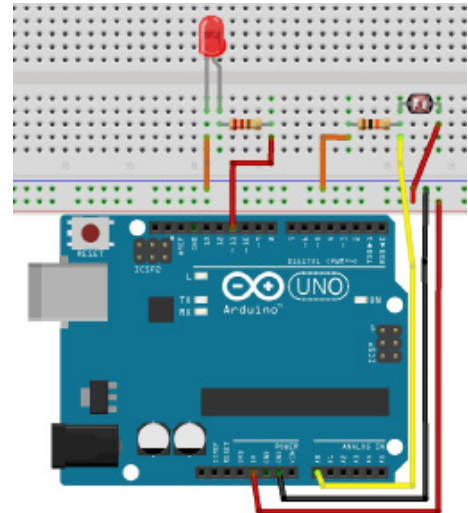
Assignment 9 Measure response time

Build a response timer that requires someone to press the button as soon as possible when a red LED goes out and a green LED comes on. To make this more difficult for the person who has to press the button, you can use function `random (a, b)`. In which a and b are numbers, so using this function you can build in a random delay. The function `millis()` defines how many milliseconds have passed.

Programming part 3

The Arduino cannot read all values. The ANALOG IN has a 10 bit chip. This means that the resolution is limited to $2^{10} = 1024$ values. It's like dividing the 5.0 V into 1024 small blocks. The value that you read out at task 8 is therefore not the voltage itself, but the number that belongs to a voltage. So the number 223 belongs to: $\dfrac{223}{1023} \cdot 5.0\,\text{V} = 1.09\,\text{V}$. The PWM ($\sim$) is an 8 bit system and can therefore give $2^8 = 256$ values.

Oops …do you see the problem? We can read with 1023 different values, but writing is only possible with 255 values …. Fortunately, there is a code that provides automatic scaling (map).

The map function requires 5 numbers for information. The first number is the value that the analog port reads. The second number is the lowest number that can be read, the third number is the largest value that can be read. This does not necessarily have to be 0 and 1023, you get the value when calibrating your sensor. The sensitivity of your sensor can therefore also be between 500 and 900, see again graph 2 from programming part 3. The fourth and fifth numbers indicate the range of the output. In the example, an input value of 500 should become an output value of 255. An input value of 900 should become an output value of 0.

Another useful feature is the ++ function. Each time the loop enters the next cycle, the value is incremented by 1. For example, you can easily keep track of how many loops (iterations) there have already been. It is also possible in this way to control a different pin each time or to change a frequency later.

If ++ exists, then -- will also exist. And that's right. -- causes the value to be decreased by 1. Both codes can be placed in a for loop. This is a loop within the loop. Take a look at the example where there is some kind of countdown mechanism in it before the program itself starts…(note, you still have to declare the variable i for the setup!

Assignment 10 Faster programming

Rewrite assignment 2h with the function ++.

Assignment 11 Alarm switch

Under the cash register of a shop there is a switch (push button). Once pressed, a pulsating alarm bell should ring. Mimic the function of the alarm bell by pulsing an LED. Make this system.
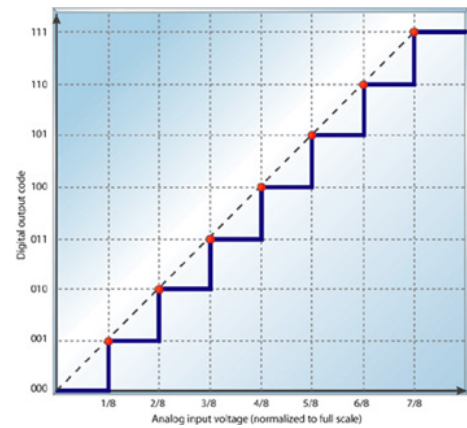

```
void loop() {
  sensorValue = analogRead
    (sensorPin);
  brightness = map(
    sensorValue
    ,500,900,255,0);
  Serial.println(
    sensorValue);
  delay(10);
  analogWrite(ledPin,
    brightness);
}
```

```
void loop(){
  for(i<5;i++;) {
    digitalWrite(LEDPin,
      HIGH);
    delay(100-i);
    digitalWrite(LEDPin,
      LOW);
    delay(100);
  }
  [...]
}
```

Assignment 12 An RGB LED

An RGB LED has four legs. The LED actually contains three LEDs. By controlling the three LEDs with the help of a PWM ($\sim$) you can mix colors and get intermediate colors. Now we can determine the strength of each color through the software, but it would be nice if we could also manually set the colors. We use variable resistors also called potentiometers for this.

The potentiometer has three legs. One for the 5.0 V in, one for the earth (0.0 V) and one leg for reading the meter using an analog pin. see also the figure on the right .

When you turn the rotary knob of the potentiometer, the resistance changes, which also changes the value of the voltage that you read out. A potentiometer is actually a voltage divider.

a) Build the circuit.

b) Write the corresponding code, first reading out the value of the potentiometer and then controlling the RGB LED.

c) Try the different positions of the RGB LED and check if the picture below corresponds well with the colors of the LED.
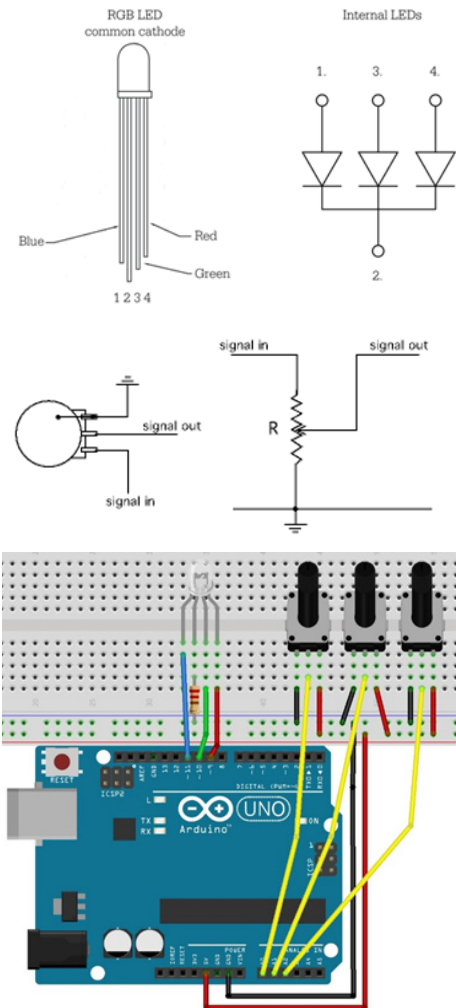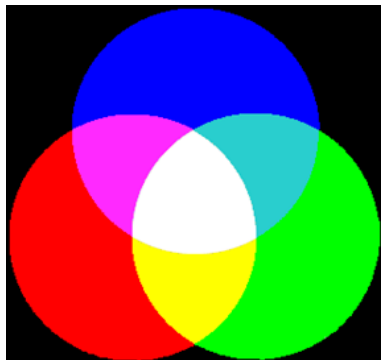
Question 13 The speed of an electric motor

An electric motor converts electrical energy into kinetic energy. You will find an electric motor in all devices that move and work on electricity. The speed of an electric motor depends on the voltage across the motor. In this lesson we will work with the PWM pin ($\sim$). We can of course directly enter the speed of the electric motor in our script, but it would be more convenient if we could adjust the speed of the electric motor manually, for example using a rotary knob. We are going to use a variable resistor, also called a potentiometer.

a) Build the right side of the circuit with the potentiometer. You don't have to connect the electric motor yet.

b) Explain why the signal out of the potentiometer should go to ANALOG IN.

c) Explain why the diode is included in the circuit.

d) Program the accompanying script so that the speed of rotation of the electric motor can be set using the potentiometer. Use the map function for this.
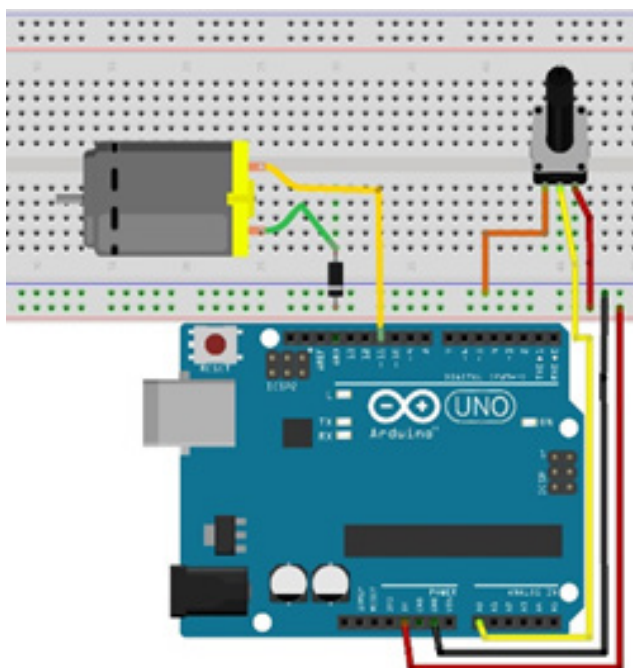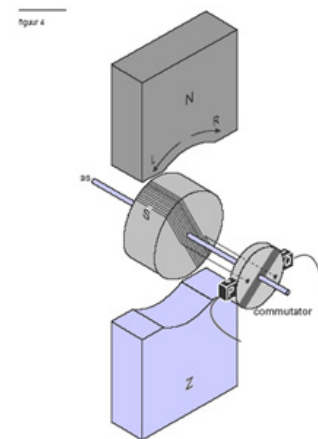
Electronics part 2

An electric motor can consume a lot of power. The Arduino is bad at providing a lot of power. To provide large power consumers with the necessary power, you can use an external power source such as a battery. Then you still need a transistor so that you can control the magnitude of the current and adjust the speed of rotation of the electric motor.

The transistor (here we only use a so-called NPN transistor) has three legs, see the photo. The emitter is on the left, the base in the middle and the collector on the right. (ATTENTION! With several transistors the pin order is different, we assume you use the 2N222 transistor). The external power supply (+ side) is connected to the collector at all times. Voltage across the base determines whether current will flow to the emitter. This makes a transistor a kind of tap: the amount of voltage at the base determines the amount of current that is passed. The base must therefore be controlled with a PWM pin if you want to control the speed.

With very high currents you have to use a larger transistor or one with a cooling element so that the transistor can lose its heat.

Question 14 The speed of an electric motor part 2

The circuit of assignment 13 is too simple and the Arduino may well be supplying too little current. Especially when you want to run the DC motor quickly.

a) Build the circuit as shown below.

b) Use the script from assignment 12 and upload the program. Can the electric motor actually turn faster?
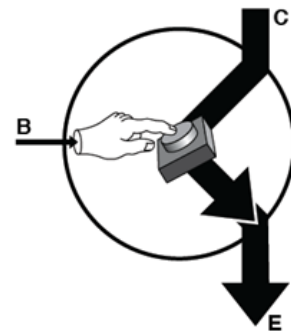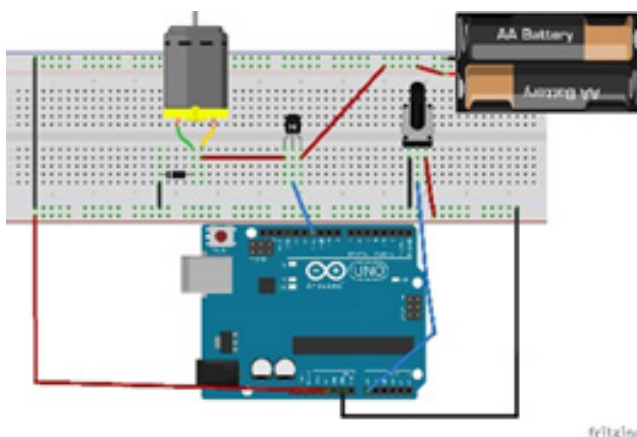
Programming part 4

Scripts have already been written for controlling some devices, making controlling the devices easier. One of those devices is a servo motor. A servo motor is a type of electric motor that can rotate 180°. The rotation angle can be set precisely because it works with a built-in potentiometer. To use a servo we need the following script.

```
#include <Servo.h>
Servo mijnServo;
int pos = 0;
void setup() {
   mijnServo.attach(9)
}
```

First, the script is retrieved from the library. Then we call the servo myServo and it gets the starting position 0.

With `myServo.attach (9)` we say that the servo is connected to pin 9. Of course this is another PWM port.

Now it is possible to control the servo. Where pos stands for the position, it can be between 0 and 180. The code for controlling this servo will then be: `myServo.write (pos);` A servo motor cannot rotate completely like a normal electric motor, but we can aim very well with a servo motor. This is because there is a potentiometer in the Servo, so the control is done by reading the voltage!

Assignment 15 A light measuring device

Use assignment 8 as a basis. Replace the LED with the servo motor. You can stick a pointer on the servo and make a scale so that the pointer indicates how light / dark it is.

## Question 16 As much sunlight as possible

Solar panels are installed to convert radiant energy from the sun into electrical energy. The panels receive the most light when they are perpendicular to the sun. The earth rotates and we should let the panels rotate to capture as much energy as possible. We are going to do that in this assignment.

All solar panels are set up on a large field, such as solar panel 1. Solar panel 2 and 3 are both slightly rotated in relation to panel 1. You can now clearly see that the solar panels would produce the most electrical energy if solar panels 2 and 3 would produce the same amount of electrical energy. However, currently panel 2 now captures more light rays than panel 3. So the whole system should be turned counterclockwise.
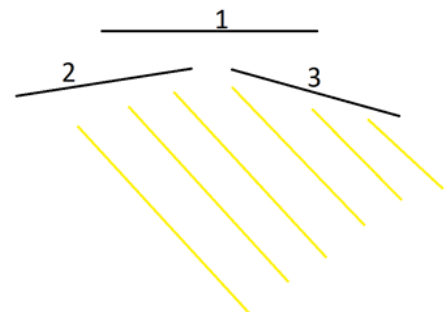
Instead of a solar cell, we start with an LDR.

a) Build the circuit and program the first piece where you can read the values of the LDR.

b) Turn on the flash of your phone and read using `Serial.println(LDR1);` the values of LDR 1 and LDR2. Move your phone from left to right and watch the values vary.

c) Now write the piece of code for the servo. Turn the servo to the left if LDR1 receives more light than LDR2 and vice versa. Keep in mind that the servo uses a potentiometer and you can also use the map function.

d) Explain that it is already quite close to rotating the solar panels.
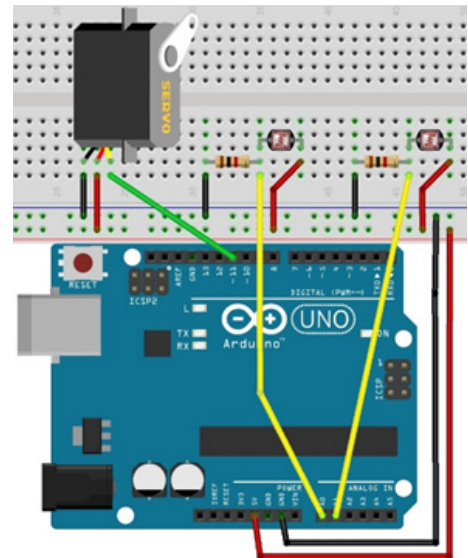
Assignment 17 LCD

With an LCD screen (liquid crystal display) it is possible to print information on a small screen. This eliminates the need to use the serial monitor (and computer) to read data.

Made with Fritzing.org
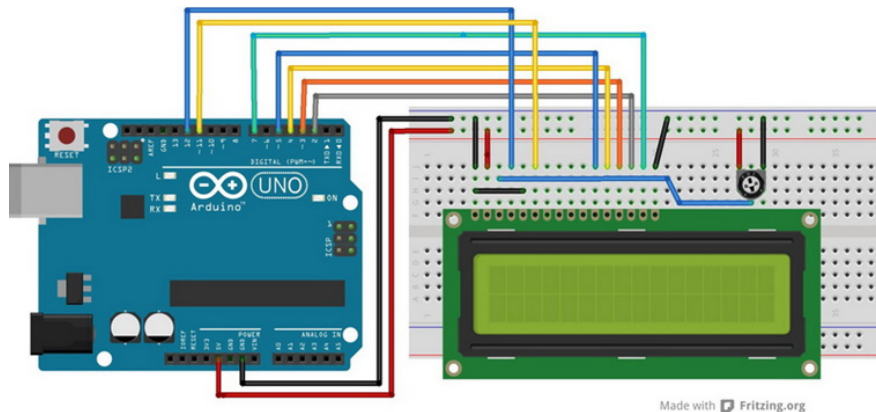
a) Create the setup shown here.

b) Open the program HelloWorld, it can be found at File / examples / LiquidCrystal and upload the program. Note! With HelloWorld your backlight is not yet on, you can control this using pin 7 (and insert the piece of code). You can also always have the backlight on by not connecting to pin 7 but to the +.

c) Try to read the script and see if you understand what the LCD shows.

The script uses a library. A library is a piece of code written by others, which makes programming a lot easier for you. Libraries have already been made for many different sensors, so it is only a matter of finding them. This often works through, for example: instructables.com or arduino.cc. All possible functions for this specific library can be found at: https://www.arduino.cc/en/reference/LiquidCrystal

Let's take a look at the script: after reading the library, we should specify the pins that will be used. The setup specifies that the lcd has 16 columns and two rows (`lcd.begin(16,2);`). The text hello, world is also immediately printed (everything in the setup is done 1x!).

Over time, the time the Arduino is on (`millis ();`) is printed to the LCD, to line 2.

a) Expand your circuit with a push button and change the code so that the LCD screen shows how many times you pressed the button.

Functions

There is a huge list of features. What follows are the most common functions that are used in Arduino.

Function	Explanation
`int`	value between -32768 en 32767 (215)
`long`	value between -2.147.483.648 en 2.147.483.647 (231)
`unsigned long` /	only positive values
`unsigned int`	
`char`	characters stored via the ASCII system
`float`	are decimal values, but are avoided as much as possible in Arduino
`digitalWrite(pin, waarde);`	does not write voltage (`LOW`) or a voltage of 5.0 V to the pin.
`analogWrite(pin, waarde)`	Writes with an 8 bit output values between 0 and 5.0 V. The indicated value must be between 0 and 255.
`digitalRead(pin)`	reads if there is a voltage on the pin, returns a 0 or 1.
`analogRead(pin)`	reads how much voltage is on the pin, returns a value between 0 and 1023.
`map(W,Min1,Max1, Min2,Max2)`	Is a scaling function. Converts a value (`W`) between `Min1` and `Max1` to a value between `Min2` and `Max2`.
`if(voorwaarde){}`	If the set condition is met, the code between the curly brackets is executed.
`while(voorwaarde){}`	As long as the condition is met, the code between the braces will be executed.
`for(init;cond;incr) {}`	A for loop executes the code between the braces a number of times. `init` is the initial value, `cond` is the condition and `incr` is the increment. Ex: `for(int i = 0; i<20; i++){delay(100);}`
`switch case`	switch case is a kind of menu where a piece of code can be executed. If the variable has value one, it executes the code associated with case 1. See code block below.
`&&`	Both conditions must apply
`||`	One of the two conditions must apply
`=!`	Is not equal to.
`Tone(pin,f,t);`	Can produce specific frequency. The tone must indicate on which pin the speaker is located, which frequency must be made and possibly how long the tone must be held. It is not necessary to specify how long the tone should be held.
`noTone(pin);`	Stops the tone that was being produced.
`millis();`	This can serve as a timer. `millis();` indicates the number of milliseconds that the Arduino is on.
`micros();`	Displays the number of microseconds that the Arduino is on. The resolution is 4 µs.
`delay();`	Wait a few milliseconds for the next piece of code to run.
`random(min,max)`	Random returns a number between the lowest value (`min`) and highest value (`max`). A disadvantage of random is that the same sequence is played when the Arduino is restarted. Random is not random…. This problem can be solved by first using the `randomSeed(analogRead(0));` function. This reads pin A0. If nothing is connected there, noise is measured here. randomSeed shakes up Random's fixed sequence, as it were.

```
switch (var) {
  case 1: \\ do something if var is 1
  break;
  case 2: \\ do something if var is 2
  break;
  default: \\ if neither of the above is true,
    then do the following (optional)
  break;
}
```

Attachment: code Blink.ino

```
/*
  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://www.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
}
```

Attachment: code Fade.ino

```c
/*
 Fade

 This example shows how to fade an LED on pin 9
 using the analogWrite() function.

 The analogWrite() function uses PWM, so if
 you want to change the pin you're using, be
 sure to use another PWM capable pin. On most
 Arduino, the PWM pins are identified with
 a "~" sign, like ~3, ~5, ~6, ~9, ~10 and ~11.

 This example code is in the public domain.
 */

int led = 9;           // the PWM pin the LED is attached to
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by

// the setup routine runs once when you press reset:
void setup() {
  // declare pin 9 to be an output:
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // set the brightness of pin 9:
  analogWrite(led, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading at the ends of the fade:
  if (brightness == 0 || brightness == 255) {
    fadeAmount = -fadeAmount ;
  }
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

Attachment: code Button.ino

```
/*
  Button

 Turns on and off a light emitting diode(LED) connected to digital
 pin 13, when pressing a pushbutton attached to pin 2.


 The circuit:
 * LED attached from pin 13 to ground
 * pushbutton attached to pin 2 from +5V
 * 10K resistor attached to pin 2 from ground

 * Note: on most Arduinos there is already an LED on the board
 attached to pin 13.


 created 2005
 by DojoDave <http://www.0j0.org>
 modified 30 Aug 2011
 by Tom Igoe

 This example code is in the public domain.

 http://www.arduino.cc/en/Tutorial/Button
 */

// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2;      // the number of the pushbutton pin
const int ledPin =  13;       // the number of the LED pin

// variables will change:
int buttonState = 0;          // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

Attachment: code AnalogReadSerial.ino

```
/*
  AnalogReadSerial
  Reads an analog input on pin 0, prints the result to the serial monitor.
  Graphical representation is available using serial plotter (Tools > Serial
      Plotter menu)
  Attach the center pin of a potentiometer to pin A0, and the outside pins to
      +5V and ground.

  This example code is in the public domain.
*/

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);
  // print out the value you read:
  Serial.println(sensorValue);
  delay(1);          // delay in between reads for stability
}
```