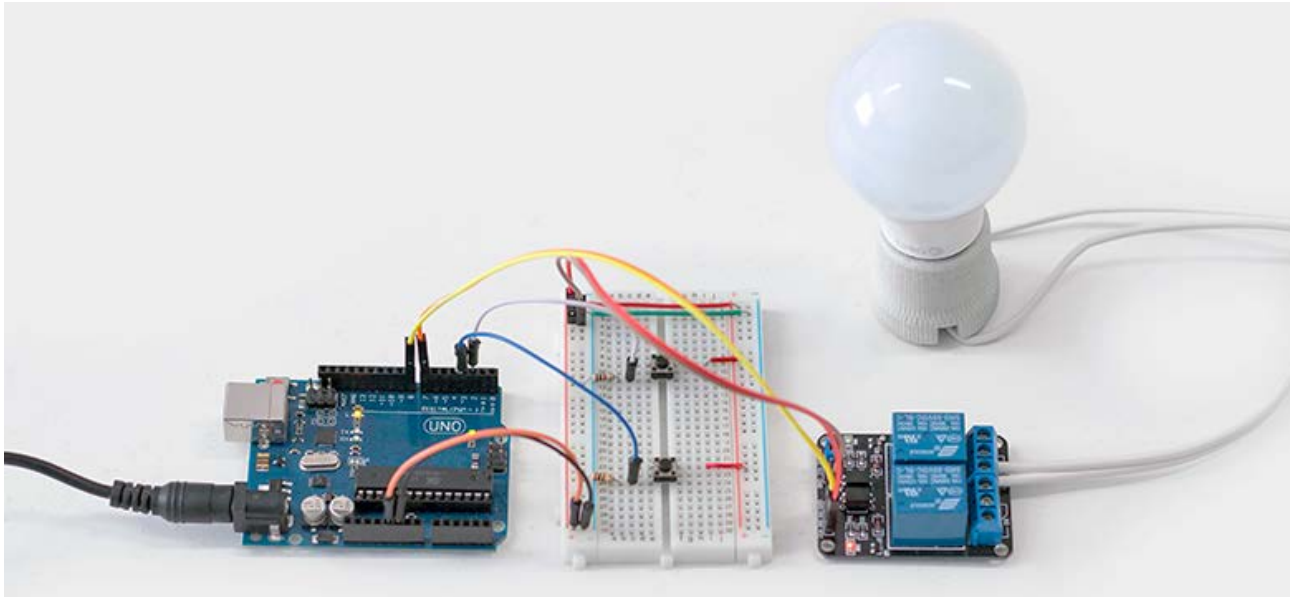
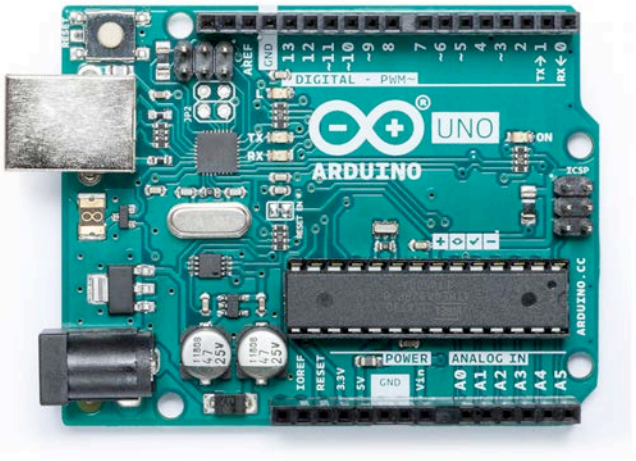
Arduino, a quick introduction

Everything you need to know to make cool things



ARDUINO™

What is the Arduino?



The Arduino is a microcontroller, which means that it is a very simple computer. You can connect sensors to it to sense things in the surroundings, and connect lots of different outputs like motors and lamps.[1]

On the Arduino you see numbers and markings that indicate what you can connect. '3.3V' and '5V' are the pins that supply electricity at those voltages. Think of them as the plus of a battery. The minus of the Arduino board is called 'GND'.

Most other markings refer to the 14 digital pins and 6 analog pins. We can use all of them both as inputs and outputs. That means any of those pins can be used to turn on a motor for instance (output) or to read data coming from sensors (input).

Installing the software

We need the Arduino software to be able to program the Arduino board. To download the software, go to arduino.cc/download.

Choose your operating system (Windows, Linux or Mac) in the dark green column titled 'Download options'. (Are you using Windows? Choose the first option.)

On the next screen, click on 'Just download'.
Once the program has been downloaded, install it and open it.

[1] There are many different types of Arduinos and similar microcontrollers, but the Arduino Uno depicted here is the most well-known.

Your first Arduino program

The pieces of software we write for the Arduino are called sketches or just programs. Sketches are basically text files that contain instructions for the Arduino board. Programmers also call these instructions ‘code’. You are now going to make your first Arduino sketch by opening and changing an existing example.

Open the menu File, then Examples, Basics and then click on Blink. The sketch that now opens in a new window is what we are going to upload to the Arduino.

Connect the Arduino board to your computer using a USB cable. Then click on the second icon from the left:

The software will now write the sketch to the chip on the Arduino board.

The first time you do this, a window may pop up which asks you to tell the computer where it can find the Arduino. Choose an adress that has ‘usb’, ‘serial’ or ‘COM’ in it, and not ‘bluetooth’.

If you see ‘Done uploading’ in the bottom left of the screen, then it worked and now an LED light is blinking on your Arduino!

If you get an error message, have a look at the page ‘Troubleshooting’ near the end of this guide.

Exercise

1. Can you make the LED light blink faster?
   Note: if you change something in the sketch, then you need to upload it again for it to work on the Arduino.

2. Can you make the LED blink different patterns? Can you make it blink ‘SOS’?

How does an Arduino program work?

The important thing to remember is that Arduino programs consist of three sections.

The first part, at the beginning of the sketch, is where variables are first named. (What variables are and how they work you can read further in this guide.)

In the case of Blink there are only a few **comments** there. All lines that begin with two forward slashes are explanation. Such lines are there to help you understand what the sketch is doing. All lines between /* and */ are comments as well. (You can also recognize comments by their grey color.)

The second part is a function called '**setup**'. All instructions between the curly brackets are part of the setup and will be executed only once.
In the Blink sketch for instance, the built-in LED is defined as an output:

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

The third part is the function called '**loop**'. Everything inside that section will be repeated over and over again forever:

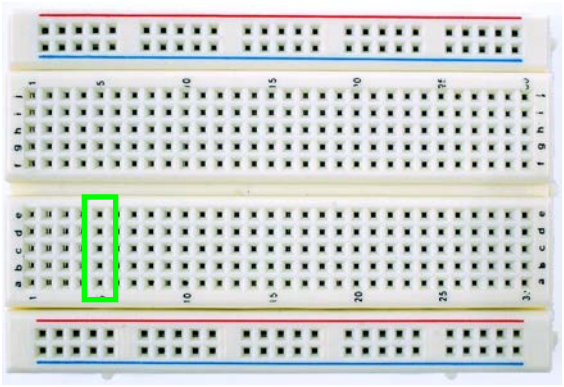
```
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second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

As you can see above, in the loop() of Blink the LED light on the Arduino board[2] is turned on ('HIGH'), then it waits for a second (one-thousand milliseconds), then the LED is turned off ('LOW'), then another one-second wait, and then loop() starts over.

[2] The built-in LED of the Arduino Uno is connected to pin 13. 'LED_BUILTIN' is a word Arduino recognizes, but you could also write '13' there instead.

How a breadboard works

To connect sensors and other components to the Arduino (or to eachother) it's often the easiest to use a breadboard.



The holes on the breadboard are connected to eachother by metal strips on the inside of the plastic board. The holes in the middle are connected in columns of five (such as the one in the green rectangle in the photo).

We don't usually mind the numbers and letters on the breadboard.

Those five holes are only connected to eachother, so not to holes in the columns right next to them or to holes on the other side of the groove in the middle.

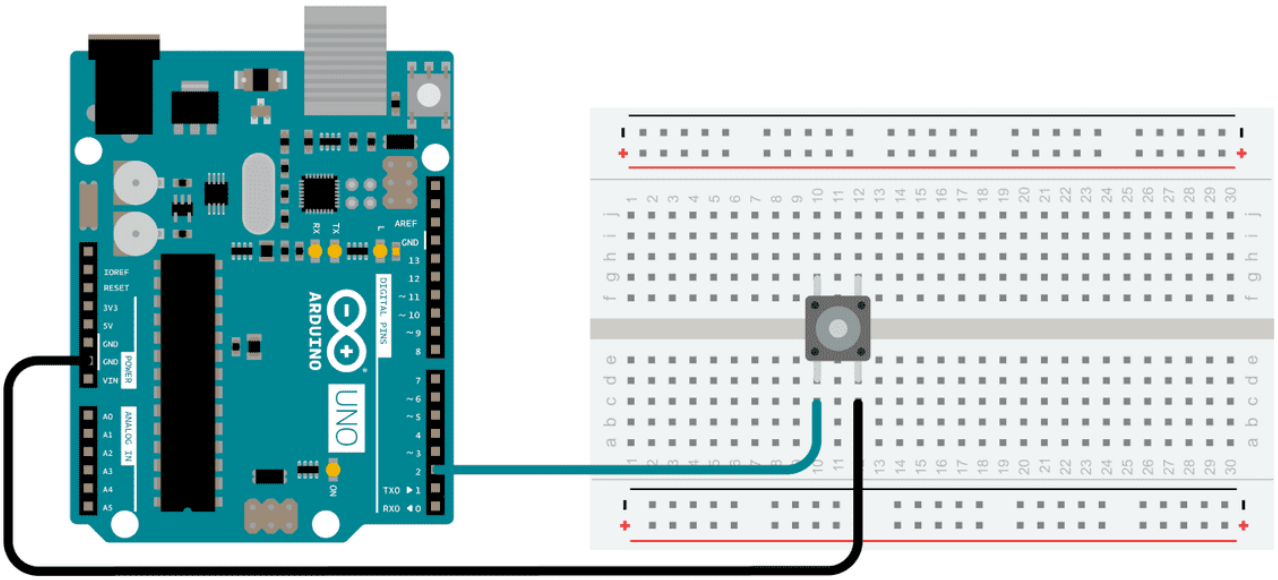
The horizontal rows at the top and bottom of the board, with the red and blue lines next to them, are used to connect multiple components to a power supply. Red is plus and blue (or black) is minus, just like batteries have a plus and a minus. Those holes are connected to all the holes in the same horizontal row, so not to any holes below or above them.

It's pretty hard to break the Arduino, or any other piece of electronics, but if you don't keep plus and minus separated it <u>will</u> break.

It is therefore smart to always stick to the right colors: the red row on the breadboard for plus and the blue row for minus. Also, as much as possible, use red wires to connect components to the positive side, and black or blue wires to make connections with the negative side.

Digital sensors: a switch

Programming an Arduino to do things (like blinking lights) is fun, but it's even more interesting to make it react to its environment. Let's try that now.



First, connect a pushbutton with the Arduino like in the diagram:

- Put the button in the breadboard with two legs on either side of the groove in the middle (when it doesn't work later on, you might need to rotate the button a quarter turn).

- Connect one leg of the button with one of the holes of the Arduino that have 'GND' printed next to them.

- Connect the other side of the button (like in the diagram) with the hole where it says '2' next to it (not 'A2').

Now open the sketch for this circuit, via the menu File > Examples > Digital > DigitalInputPullup.

Upload the sketch to the Arduino.

Try the button. You should now have your first working digital sensor! When you push the button, the LED on the Arduino turns on. So this already is a combination of an input and an output.

How does the script for the digital sensor work?

In the sketch below you see that in the function setup() pin number 2 is ‘opened’ as an input:

```
pinMode(2, INPUT_PULLUP);
```

That pin is where you connected the button, and thanks to this line the Arduino knows it too.

Next up is defining pin number 13 as an output:

```
pinMode(13, OUTPUT);
```

The built-in LED on the Arduino is connected to pin 13, and thanks to this line the Arduino knows we want to use that LED.

Now we will check if the button is being pressed. We can using the function digitalRead():

```
int sensorVal = digitalRead(2);
```

When you push the button that is connected to pin 2 of the Arduino, digitalRead here will return 0. When you let go of the button, that changes to 1.[3] That value is then saved to a variable called ‘sensorVal’ (more explanation of variables is coming up).

All of this is happening in the function loop(), which means that digitalRead() is executed over and over again and thus the status of the button is checked all the time. ‘sensorVal’ always contains the current state of the button: 1 or 0, on or off.

To use the information the sensor is giving us (in this case, whether it is being pressed), the sketch then uses an if-statement. How that works, we will look at next.

---

[3] That is why a button is an example of a digital sensor: there are only two possible states. Other examples of digital sensors are motion sensors (like PIR sensors that automatically turn on the light in bathrooms) and ‘electric eyes’ that make sure elevator doors do not close when someone is between them.

If-statements

To have the Arduino do something when something changes, we need to teach it to ask questions. This is what a question in Arduino looks like:

```
if (sensorValue > 30) {
  // do something here
}
```

This is called an if-statement. It lets Arduino ask: is the button being pressed? Or, like in the example above: is the value which a sensor returns larger than 30? When the answer to such a question is 'yes', then Arduino will execute the code between the curly brackets.

In the DigitalInputPullup sketch we just saw, it looks like this:

```
if (sensorVal == HIGH) {
  digitalWrite(13, LOW);
} else {
  digitalWrite(13, HIGH);
}
```

So here the question being asked is: is the button connected to pin 13 being pressed?[4] When it isn't, we turn off the LED, and turn it on otherwise. This is an addition to the if-statement: 'else' allows us to add instructions in case the answer to the question is 'no'.

Punctuation

Note that every opening '(' needs a closing ')', every '{' needs a closing '}' and every command such as digitalWrite() needs to end with a semicolon (;). Programming languages are very sensitive when it comes to these things and will not cooperate when you get these things wrong. When you do get errors, see 'Problems with code' in the section Troubleshooting at the end of this guide.

Exercise

The LED currently turns on when you press the button. Can you change the code so that the LED *turns off* when you press the button?

---

[4] Because of how we connected the button here,'HIGH' in this case means the button is *not* being pressed.

See what is happening: Arduino's serial monitor

Imagine something goes wrong, for instance the LED does not turn on when you expect it to. It would be practical if the Arduino could tell us if it at least registers the button being pressed, because that would help us find out what the problem is. Fortunately, the Arduino can do that.

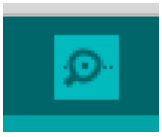
We can set up a communication line between the computer and the Arduino board. We do that by adding this line to setup() (in the DigitalInputPullup sketch you just used this is already there):

```
Serial.begin(9600);
```

In loop() we add these lines directly below the lines where we read the sensor:

```
Serial.print("sensor = ");
Serial.println(sensorValue);
```

Then open the serial monitor by clicking on the magnifying glass icon in the top right of the window.

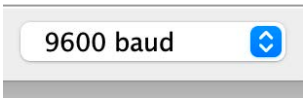


If the sketch DigitalInputPullup is still on the chip of the Arduino, then you should now be able to see it in the serial monitor when the button is pressed. If you don't see it changing, then you know that the button is either not working or not connected correctly to the right pin.

Serial communication often is incredibly useful to find out if the Arduino is doing what we were expecting it to, and if not, where the problem is.

We can also use the serial connection between the computer and the Arduino to use sensor data from the Arduino in other software on the computer, to make interactive animations or music for instance.

That number 9600 is the baud rate, the speed at which the computer and the Arduino talk to each other. Some sketches use different values, such as 115200. The baud rate in the serial monitor needs to be the same as the one in the sketch, otherwise you will see weird characters—or nothing at all.

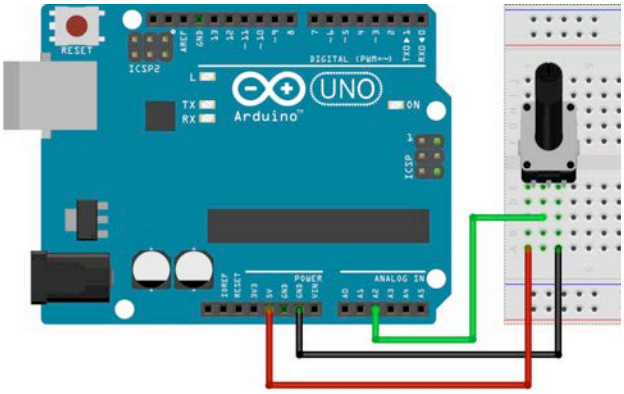


Analog sensors

The button we used so far is a digital sensor. Digital sensors only know 1 and 0, in other words on or off, or LOW and HIGH.

Another very common type are analog sensors. Those sensors can give in-between values. An example is the light sensor, which can measure if it's dark or actually very light, but which can also measure all gradient values in between.

Another example of an analog sensor is the potentiometer. This is a knob which you probably know best as the volume knob on speakers and amplifiers. In Arduino you can see the setting of the knob very precisely.



Connect the potentiometer like in the diagram: the middle pin to analog pin A0 on the Arduino, and the outer pins to 5V en GND.

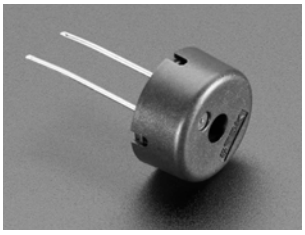
Then open this sketch: File > Examples > 03. Analog > AnalogInput.

Upload that sketch to the Arduino. You should now be able to make the LED blink faster or slower by turning the knob!

Also fun: looking at the values coming from the potentiometer using the serial plotter. How to do that is revealed on the next page.

Exercise

Connect a small speaker or 'buzzer' to GND and pin 13 (via the breadboard). Those are the same pins the built-in LED is connected to internally. Can you hear a difference when you turn the knob?

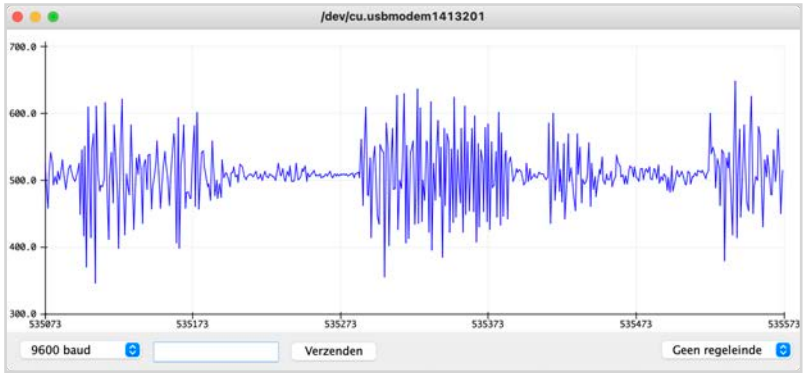


Serial plotter

To show analog values in a more accurate way, the Arduino software has a great feature built into it.

Open the Tools menu and click on Serial plotter (first close the serial monitor if that window is still open).[5]

Now open the sketch AnalogReadSerial via the menu File > Examples > 01. Basics > AnalogReadSerial, and upload that to the Arduino.



In the window of the serial plotter you should now see a graph of the changes in the position of the potentiometer! (Or of the values of another analog sensor, such as the light sensor.)

Multiple sensors

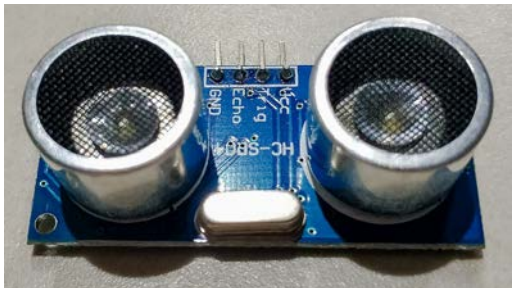
If you want to see data from multiple sensors in the serial plotter, then you have to make sure their values are sent to the computer separated by tabs:

```
Serial.print(sensor1);    // value from the first sensor
Serial.print(",\t");      // separate values with a comma and tab
Serial.print(sensor2);    // value from the second sensor
Serial.println();         // new line
```

If you have any print()'s in your sketch intended for the serial monitor (so with text in them instead of numbers coming from your sensor), then turn them off temporarily by putting // in front of those lines to turn them into comments.

[5] The shortcut for the serial plotter is shift+command+L.

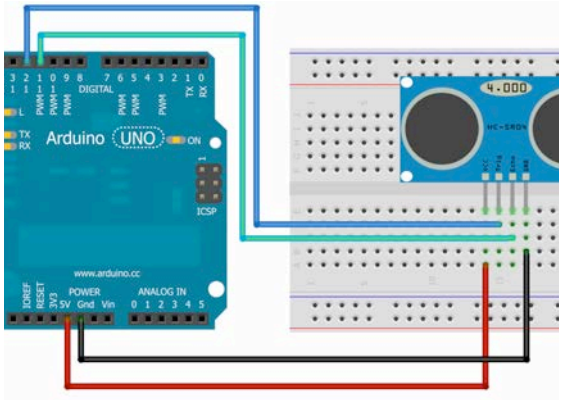
Distance sensors



If you want to know how far away something or someone is (from your robot or artwork for instance), using an ultrasonic distance sensor is great. It has two round things, a speaker and a microphone. The sensor works by sending an ultrasonic sound with its speaker, and then using the microphone to measure how long it takes for the echo to come back. By dividing that time by the speed of sound,[6] the Arduino can calculate how far away an object is.

Connect the sensor to the Arduino, possibly via a breadboard, like so:

- the pin on the sensor where it says VCC to 5V on the Arduino,
- GND on the sensor to GND on the Arduino,
- Trigger to digital pin 12 on the Arduino,
- Echo to pin 11 on the Arduino.



Code for this type of sensor can be found online everywhere, but the NewPing library is very easy to use. You do need to install that—if you don't know how to yet, check out the page 'Installing Libraries' further on in this guide.

In the example sketch NewPingExample (see File > Examples > NewPing) you can set the maximum distance you want to measure.[7] About four or five meters is the maximum you can get with this sensor.

Upload the sketch to the Arduino and open the serial monitor to see the distances the sensor is measuring.

Ultrasonic distance sensors measure the distance to objects in a narrow beam of 15 degrees. If you want to scan a wider area, you can combine multiple ultrasonic sensors next to eachother, or have one sensor rotate on a little motor to make a sweeping radar. A more complete solution would be image recognition, but that is also quite a bit more complicated.
If you only want to know *if* there is a person moving nearer (and not necessarily how far away they are), then use a PIR sensor.

[6] 343 meters per second; the measured time is also divided by two, because the sound has travelled twice the distance once it comes back as an echo.

[7] More info on the NewPing library can be found here: https://bitbucket.org/teckel12/arduino-new-ping

Variables

Variables are words or letters that we use to make Arduino remember things. That's often important, for instance when we want Arduino to keep track of how often something has happened.

A variable is defined like so:

```
int count = 100;
```

By putting 'int' in front of it, we say: the variable is a whole number, so not a decimal.[8] Next, we give the variable its name. In this case we name it 'count', but it could just as well be 'x', or 'sensorValue'. Usually we choose a name that makes really clear what information we want to store in that variable.
Then we tell the variable what its initial value is going to be—in this case 100.
Lastly the statement is closed with a semicolon, like all statements in Arduino.

Once a variable is defined like that, you can start calculating using that variable.
If we do this…

```
count = count + 100;
```

… then 'count' now has a value of 200.

Exercise
Variables allow you to keep track of things, for instance how often a button was pressed. You can use the sketch DigitalInputPullup to do that (see the page on digital sensors). First try to do it yourself, and then read the solution in the footnote.[9]

[8] Other types of variables are for instance *float* (that is a decimal numeral) and *bool* (can only be 'true' or 'false'). Find all types at https://www.arduino.cc/reference/en/ under 'Data Types'.

[9] Add a variable at the top of the sketch (so even above setup()). Add 1 to the variable, right where the sketch does something when the button is pressed. Move the 'Serial.println' statement to that same spot to write the value of your variable to the serial monitor instead of 'sensorVal'. When you also add 'delay(500);' under that line, then the Arduino will wait half a second before it continues. That way it only counts just one press of the button instead of keep on counting for as long as the button is being pressed :-)

Combining inputs and outputs

Often we want to use different inputs and outputs at the same time, and have them work together. You could use a potentiometer to change the speed of a motor, or a light sensor to change the pitch of a buzzer. Here we explain how to do that.

Step 1

If you want two components to work together, always first make them work separately. Then if something doesn't work later on, at least you will know the problem is not with the individual components and that makes looking for a solution a lot easier.

So for instance:

- Connect the potentiometer to the Arduino
- Upload the sketch for the potentiometer to the Arduino, and make sure you see values coming from the potentiometer in the serial monitor.
- Connect the buzzer (leave the potentiometer connected!).
- Put the code for the buzzer on the Arduino (leave the windows with the sketch for the potentiometer opened), and only proceed after you hear the buzzer.

Step 2

Now you have two working sketches, which you can start combining into one sketch.
Put them right next to eachother on your screen.

In Arduino programs setup() and loop() can only appear once. So we cannot just combine the sketch by simply pasting one below the other.

That means you have to copy the contents of one setup() into the other setup(). Make sure you don't copy any curly brackets with the code. Also make sure not to copy any code above or below the curly brackets of setup()—it has to go between { and } or it won't run.

This what that looks like for the depicted programs:

Next, do the same for loop() and move any lines above setup() to the other sketch.

When you see the exact same instruction in both sketches (like `Serial.begin(9600);`) then you don't need to copy it.

Sometimes you will want to combine two sketches that both use the same input pin. When you combine those sketches, then you will have to find another pin for one of those inputs (so for instance digital 4 instead of digital 13) and also connect the component to that other pin on the Arduino.

Step 3

Have Arduino check your new sketch for errors. Use the first icon in the top left of the window to do that. When it says 'Done compiling' then you did the combination correctly! If you get an error instead, fix the problem. You can use the Troubleshooting section near the end of this guide to help you.

Save your combined sketch to your computer, via File > 'Save as…'.

Step 4

Now you have one sketch that lets two components work side-by-side. When you upload this sketch to the Arduino, both should work simultaneously.

But what you want of course is to have them work together! To make that happen, it's often easiest to use an if-statement, as covered earlier in this guide. Another option is to stick sensor data directly into instructions for an ouput; an example could be using light values as the degrees for a servo motor.

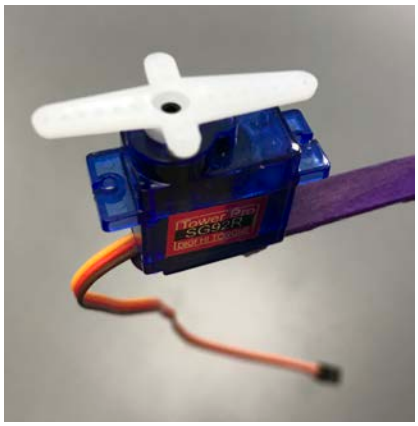
Arduino sketches often use the delay() function. That sometimes leads to problems when you want Arduino to do two or more things at the same time. The solution to this is to set a timer; how to do that is explained further on the page about timers and millis().

Using existing code

What you can also do, is use existing code created by others. Let's say you want to use a color sensor together with an LED strip, then just type into a search engine something like 'Arduino color sensor LEDs'. No doubt you will find a lot of tutorials that share the components they used, including the necessary code.

When you copy such code, be sure to read that code thoroughly and try to understand what each part does. When the code doesn't work, use the checklist at the end of this guide to solve the problem.

Making things move with servo motors



The simplest way to have something move using the Arduino (things like robots, pen plotters, flower pots moving towards the light) is a servo. There are two flavours: servos that turn between 0 and 180 degrees, and servos that can keep on turning.[10]

Most small servo motors have three wires:

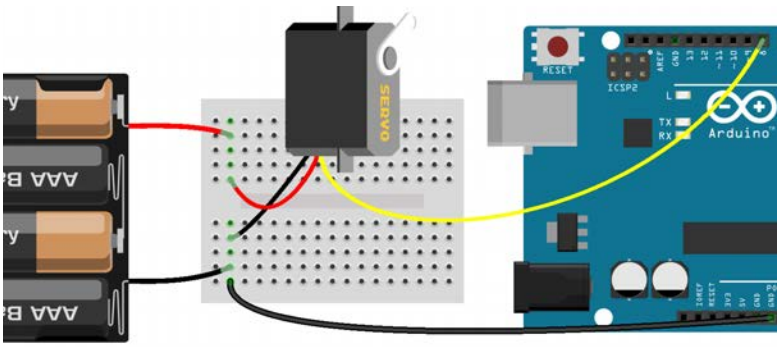
- brown is ground, so minus/GND
- red is voltage, so plus/5V
- yellow is the wire through which the Arduino is telling the servo where to turn.

Connect the servo, with the yellow wire going to pin 9 on the Arduino. When you upload this sketch, the motor wil start moving: File > Examples > Servo > Sweep

As you can see in the sketch, the servo is sent to a certain angle with the function myservo.write(). In the sketch that happens in something called a **for-loop** to make it turn back and forth in small steps. You could also send the servo to a certain position, for instance to the middle position using the command `myservo.write(90);`

Larger servos

Servos like in the photo are small and not very strong. There are many servos that are (much) stronger. You can use those with Arduino as well, but to give them enough power you need an external battery:



Exercise

Move the servo to different positions a couple of times, and have it wait a second every time. Hint: you don't need the for-loops for this—a couple of myservo.write's and delay()'s should be enough.

---

[10] 'Continuous' servos like that translate the position you tell them into a direction to turn to and a speed. '0' for instance is full speed to the left, '90' is standing still and '110' means 'turn slowly to the right'. If you want to know how far a continuous servo has turned, you have to use a 'rotary encoder' or 'encoder wheel'.

Conclusion

There are an infinite number of input and output combinations you can make with the Arduino. You can, for instance, let an MP3-player play sounds when a motion sensor detects motion, or let LED strips change color based on a heart rate you pick up with a heart rate sensor.

There are no real barriers anymore to what you can do with electronics. Parts have become incredibly cheap, and information on how to use them is always available online. Search for something like ‘Arduino plant moisture sensor’ and you will find thousands of tutorials that not only show you how to connect such a sensor but that also share the code.

A couple of websites are particularly useful for inspiration and practical information:

- Instructables: https://www.instructables.com/circuits/arduino/projects/
- Arduino: https://blog.arduino.cc/ and https://www.arduino.cc/reference/en/
- Hackster.io: https://www.hackster.io/arduino

The most important thing is that you understand that everyone can do programming and make things with electronics. Nobody is an expert right from the start at anything, but practicing and not giving up will get you very far. There is an immense amount of possibilities, and you are one of the people who can make cool things.

Troubleshooting

Problems with connecting

This is a list of possible solutions when you are unable to upload to the Arduino. If the first solution does not correct the problem, move on to the next one, in the order in which they are listed.

1. Select the right port.

   An error when trying to upload is usually caused by the computer not knowing to which of its ports the Arduino is connected:

   You can fix that by selecting the right option, in the Tools menu under ‘Port’. Sometimes the Arduino is already listed, but you may need to guess. It should be an adress that has ‘usb’, ‘serial’ or ‘COM’ in it.

   Make sure that under ‘Board’ the right Arduino is selected (usually Arduino Uno).
2. Is the USB cable connected properly? (If it needed adjusting, go to 1 again.)
3. Try connecting the cable to another USB port on your computer (and then step 1 again).
4. Restart the Arduino software (and check step 1 again).
5. If you have things connected to the Arduino, then try unplugging everything. (If you can indeed upload to the Arduino now, then there is a short circuit somewhere *or* something is connected to a wrong pin.)
6. Restart the computer (and then step 1 again).
7. If your computer still can’t find the Arduino under ‘Port’ then you probably need to install a driver. Follow the instructions on this page: https://www.arduino.cc/en/Guide/DriverInstallation (and then step 1 again).
8. Try another Arduino (and then go to step 1 again).

Problems with code

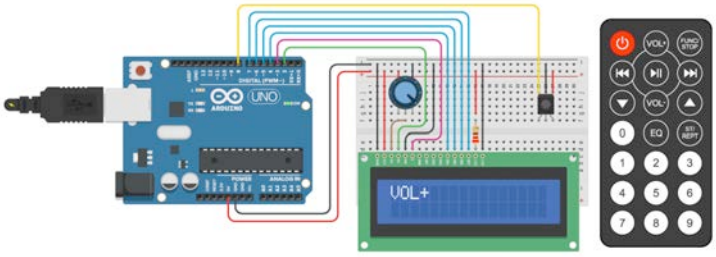
Often, the error messages the Arduino gives in the black window at the bottom of the program are really quite helpful. Try to understand what they mean.

1. Where is the error happening? Find the row that is highlighted red. The error can also be just above that line.
2. The most common error is a function or command that is not closed properly. Is there a } too many or is one missing? Remember that every opening '(' needs a closing ')', every '{' needs a closing '}' and every command such as digitalWrite() needs to end with a semicolon (;).
3. Is all of your code really *inside* setup() of loop()? So in between the curly brackets of either function?
4. If you include a library that cannot be found, you will see the error 'No such file or directory'. In that case, install the corresponding library; see the page 'Installing Libraries' in this guide.
5. Search online for solutions to the particular error messages you are seeing. Search with terms between quotes, so for instance 'Arduino "exact error message here"'.
6. Use another piece of code. Try an example sketch and then adapt that, or try to find code that does what you want online.

No error, but also not working the way you expected?

- Check if the components are really connected to the right pins and that they correspond to the pins mentioned in the code.
- When using a breadboard: are all components really, really in the right row?
- Use Serial.println()'s to find out what Arduino is doing and what *is* working (see the exaplanation page on the Serial Plotter in this guide).
- Try another one of the component you are using, to find out if the first one might be the problem. After that, try different wires or even a different breadboard.

Installing Libraries

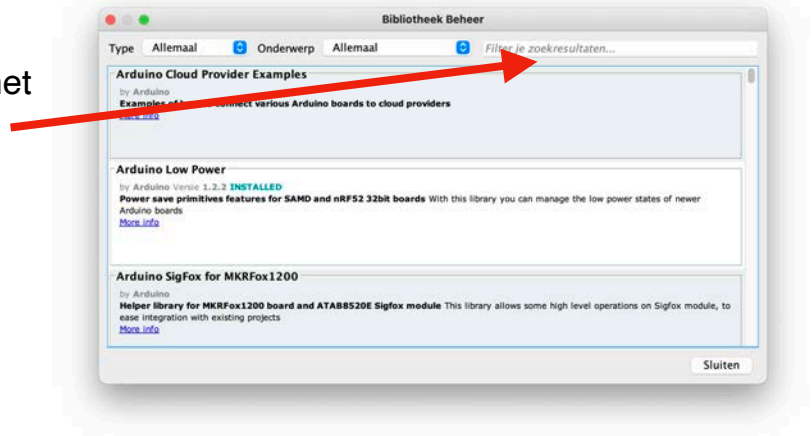


You really do not need to program everything yourself. A *lot* of software has already been written by people who love to share their work with you.[11]

If such software is more comprehensive than a single sketch, such a package of software is called a library. The Arduino program has a number of libraries pre-installed, but sometimes you will need to install them yourself.

Imagine you want to try out an infrared sensor to be able to control your Arduino project with a remote control. That is made easier by a library called IRremote.

Go to the menu Tools > Manage libraries. Search for ‘IRremote’ in het search field at the top right, and then it will probably be the second item in the results list.



Click on ‘Install’ and then ‘Close’.

Once you’ve installed a library, you can open the example sketches of that library via File > Examples. Those example programs will quickly show you how to use the library in your own project.[12]

Not *all* libraries can be installed in this manner. Sometimes you need to download a folder with files and manually add them to the folder where Arduino stores the libraries. You can find out what the location of that folder is via the menu Arduino > Preferences.

[11] It is called open-source software and it’s awesome.

[12] In the case of IRremote: File > Examples > IRremote > IR receiveDemo. More explanation on the use of remote controls and how to use the library can be found here: https://www.circuitbasics.com/arduino-ir-remote-receiver-tutorial/

A timer for Arduino programs: millis()

If you want Arduino to do two things at the same time, like blinking an LED and meanwhile keep an eye on if a button is pressed, then it is important *not* to use the delay() function. delay() makes Arduino wait for a while, and in the meantime everythings halts. So, when it holds everything while the LED turns on or off again, it cannot also keep track of whether the button is being pressed.

To enable Arduino to do two things at the same time, we let Arduino use a kind of stopwatch. The stopwatch is called millis(). That function keeps track of how long our sketch has been running, and help Arduino do something at regular intervals.

An example sketch is shown below. It turns an LED off or on every three seconds without blocking everything in the meantime with delay()'s.[13]

```
// Define variable to keep track of time.
unsigned long lapTime = 0;

void setup() {
  pinMode(LED_BUILTIN, OUTPUT); // Activating the built-in LED on the Arduino.
}

void loop() {

  // Have three seconds passed? Find out by subtracting the passed time since
  // the last lap from from the current time.
  if ( (millis() - lapTime) > 3000 ) {

    // If so, then do this:

    // Change the LED to on or off (the exclamation mark means 'turn it into what it isn't now').
    digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN) );

    // Reset the stopwatch: change lapTime to the current time since the sketch started.
    lapTime = millis();

  }

  // Now we can starting doing other things here, like turning motors, or keeping track of
  // any buttons that may be pushed *without* waiting for delays.

}
```

[13] This solution is actually called 'non-blocking code'. There is a useful library for it called arduino-timer. It lets you set multiple 'stopwatches' in one sketch. When you install it (see the 'Installing Libraries' page), the possibilities are explained in the example sketches that come with it.

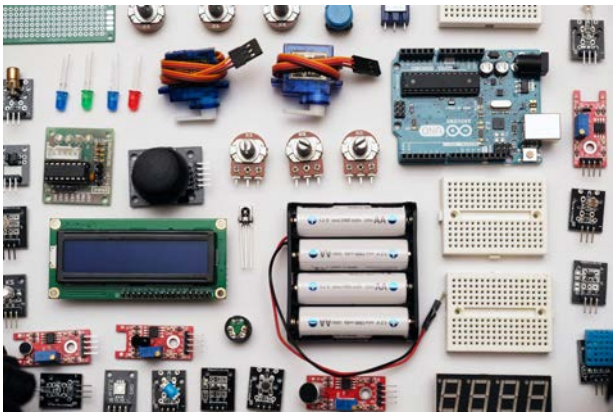
What you could measure and do with Arduino

Inputs

Distance sensor
Accelerometer
Gyroscope
Compass
Temperature
Relative humidity
Moisture (water the plants!)
Barometer
Light sensor
Color sensor (sort candy to color :-)
Control knobs and linear potentiometers (like the ones on audio mixers)
Microphones (or other audio signals)
Touch
Switches
Keyboard and mouse
Fingerprint
Cameras
Motion sensor
Infrared receiver (controlling your project with a remote control!)
Heartbeat[14]
Smoke
Gas
Radiation
Flame detector
Magnetic fields
Vibration
Bend or stretch sensors
Water current
Brain activity

Outputs

Light: lamps, lasers, LEDs, EL wire, glass fibre
Sound: speakers, transducers, amplifiers, MP3 players[15]
Motors: servos, very precise stepper motors, very fast DC motors, very strong gear motors, vibration motors, linear actuators, ventilators.
Pump (for instance to make it rain or let a statue 'cry')
Solenoids
Elektromagnets, ferrofluids
Video (animations, videos, virtual reality/augmented reality)
LCD display (like in vending machines)
Metal that changes shape
Heating and cooling elements
Smoke machine
Website or social media (automatically post info based on sensor data)



This is by no means a complete list, but as you can see the possibilities are endless. What combinations of inputs and outputs are you going to try?

[14] Some heartrate sensors barely work. This one is a bit more expensive, but it is pretty reliable: https://www.kiwi-electronics.nl/pim-438

[15] There are many different ones, but this one is cheap and easy to connect: https://wiki.dfrobot.com/DFPlayer_Mini_SKU_DFR0299

This manual was written by Jaap Meijers (instructor and manager of the Hacklab at the Royal Academy of Art in The Hague). It was published in May 2021 under the Creative Commons Attribution 4.0 International-licence.[16] That means you are free to disseminate and adapt it, as long as you give appropriate credit and provide a link to the licence.

Find out more about Jaap's projects at
http://www.eerlijkemedia.nl
https://www.instructables.com/member/tjaap/

Photo credits
Arduino with relay and lamp: Adilson Thomsen/FilipeFlop/Wikimedia
Breadboard: Victoria.nunez2/Wikimedia
LDR: Arnau 944/Wikimedia
Potentiometer: Iainf/Wikimedia
Piezo buzzer: Adafruit/Flickr
Ultrasonic distance sensor: Nowforever/Wikimedia
Microservo: own photo
Arduino with flowers: madshobye/Instructables.com
Diagram IR remote on 'Installing Libraries': Benne de Bakker/makerguides.com
Stopwatch: Matthew/Flickr
Parts: Robin Glauser/Unsplash
Diagrams with potentiometer, ultrasonic distance sensor and servo: Fritzing.org
Diagram on 'Digital sensors' and photo of the Arduino board on 'What is the Arduino': Arduino.cc

[16] https://creativecommons.org/licenses/by/4.0/