Station 0 – First Steps

Welcome to the Module All about Light!

So, today you want to construct a parking assistance, measure speeds, let a flower wander with the sun or display temperatures via colors? This is great! But before you start working on these projects, it is important that you first get to know your materials. In this worksheet, you will get an overview of the Arduino microcontroller, the components and the programming environment, which you will use!

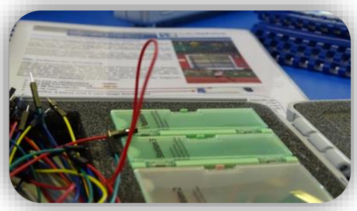
Throughout the module, the worksheets will support you in the implementation process. Just pay attention to the following symbols, which...

- structure your work and point out objectives 🎯,
- offer advice, point out important information or possible difficulties, etc., and ⚠
- include (step by step) instructions to guide you 🖱.

After completing this station, you will know,...

- what to do with the stuff in the grey box.
- how to light up a LED.
- how to make that a LED blinks.
- how to use a push button.

[1]

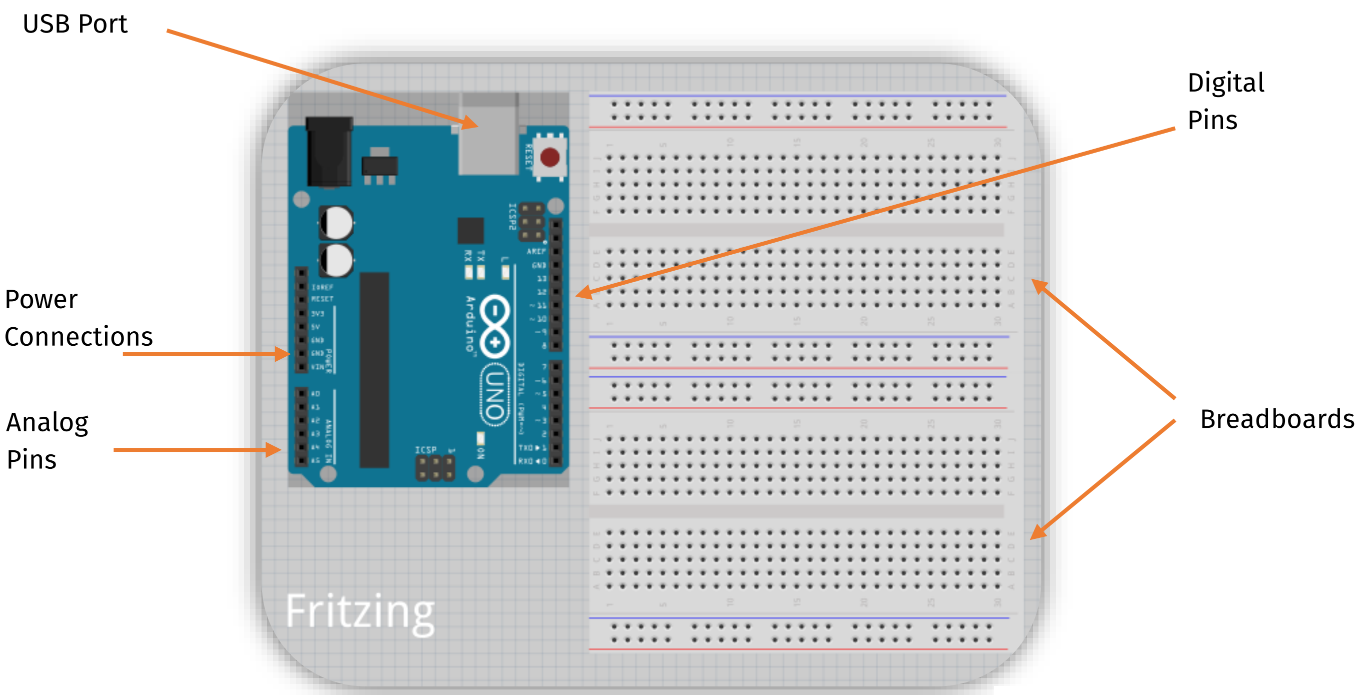


[2]

First of all, you will learn how to connect the different components, to which aspects you have to pay attention to by dealing with LEDs, and how to program the Arduino.

The Arduino

Take a look at the Arduino. It is at the center of every circuit.



[3]

This is only a schematic representation, in which some insignificant parts have been omitted. Even on this schematic, you can see more than you really need. In the following, the important components will be explained.

The USB Port

You will use this port to connect the Arduino and the computer to transfer your program. This port also serves as a power supply for the Arduino.

Power Connections

Some components require power. Just try to remember:

GND = – (Negative Pole) 5V = + (Positive Pole)

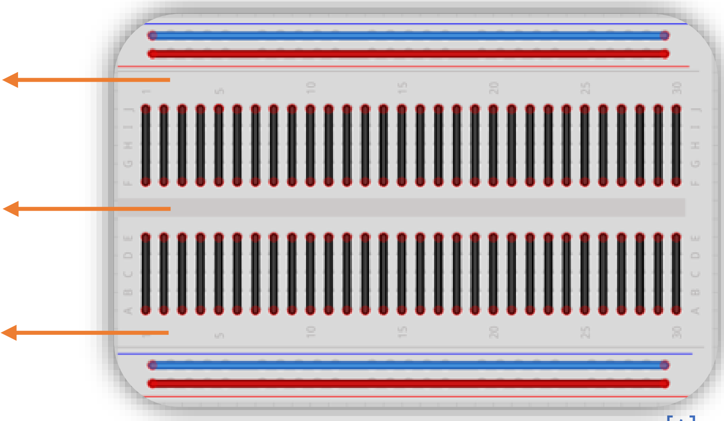
Just use Pin 2 to 13. The rest has special functions.

The Pins

In contrast to the **digital pins**, the **analog pins** are only used as inputs. Therefore, they are also called analog inputs. They do not only know **high** and **low**, but 1024 different values. If 0V are connected, the value is 0, at 5V it is 1023. But more about this later.

The Breadboards

On the breadboards, you can see a lot of holes. They are connected under the surface. In the following figure, you can see how exactly they are connected:



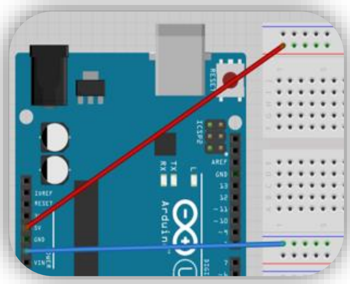
Gaps that you will have to bridge with wires

[4]

Always disconnect the Arduino from the power source, before you change the wiring! Simply unplug the USB cable from your computer.

Getting Power to the Breadboard

A good start is to connect the outer rows of the breadboard to the Arduino. After doing that, you can easily supply all components on the breadboard with power since the outer rows function as multiple sockets.



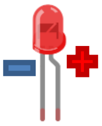
[5]

Now, connect your Arduino to the breadboard (red to 5V, blue to GND) using two plug-in wires.

[6]

There Is Light

Before you start programming, you will learn, how the wiring works and how to check, if your circuit has any power. You will also get to know two important components: **LED** and **resistor**.



[7]

LEDs

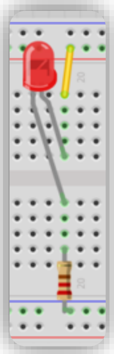
LED is the acronym for **L**ight **E**mitting **D**iode. So, it is a component that emits light. Diodes are components, that allow current to pass in only one direction. Therefore, it is very important to install them the right way around. For the LED's installation, you can roughly orientate yourself by the length of the legs because: **one is longer than the other**. The long one has to be connected to the **positive pole**.

Unfortunately, LEDs are overly ambitious components. They always want to produce as much light as possible. But in doing so, they also heat up and, in the worst case, get so hot that they break.

Therefore, you have to limit the current, when you work with LEDs. To do this, you use resistors. In your box you will find two types. The ones with red-red-brown-golden rings (**220 Ω**) are the ones you need right now.

[8]

It does not matter, if you install the resistor before or after the LED. The direction is also irrelevant.

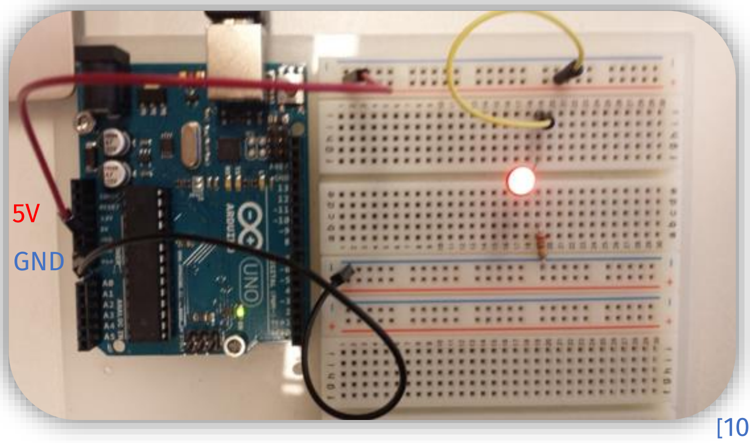


[9]

This is the schematic for connecting one LED.

Now, it is your turn. Take a yellow wire, the correct resistor and one LED, and built the circuit.
Remember: The short leg points to the negative pole.

Your circuit should look like this, when you followed the schematic:



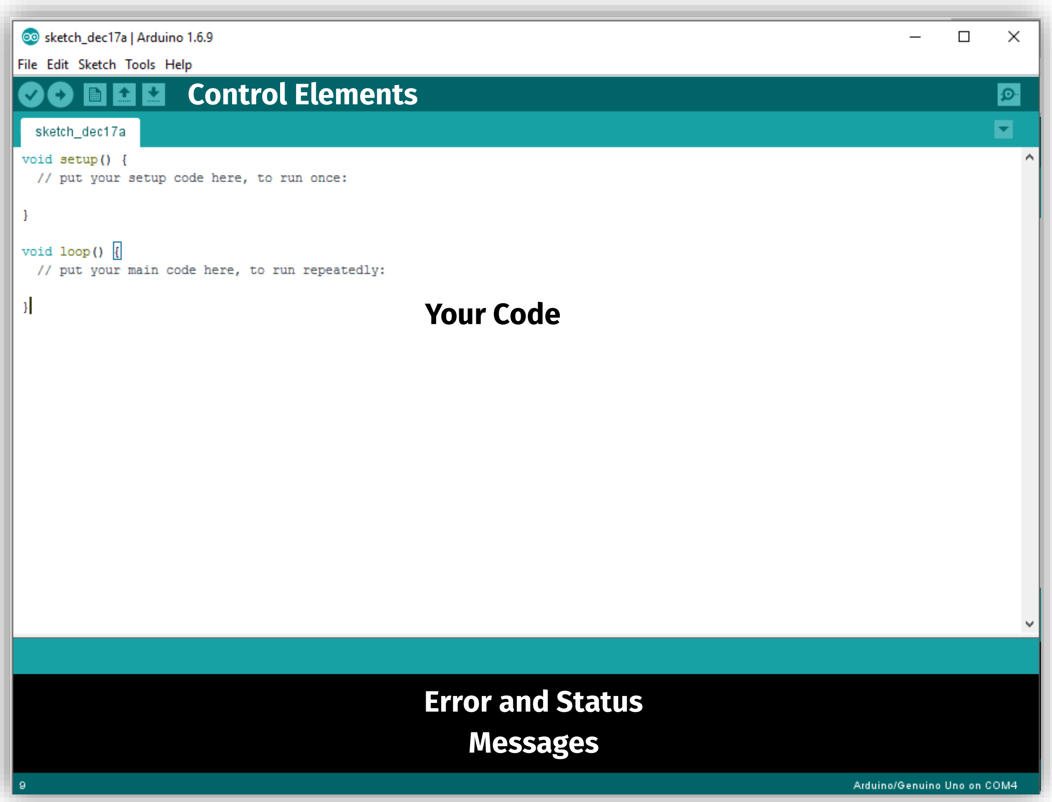
[10]

If your LED does not light up:

- Did you plug in the USB cable?
- Is the LED plugged in the right way?
- Are LED, resistor and wire plugged in the same row?
- Did you use the correct resistor?

That was great to get started! Now, you are ready for the more exciting things like programming the microcontroller. On the computer, you will find a software with the same name as the microcontroller: Arduino.

After starting the program, you will see the following window:


sketch_dec17a | Arduino 1.6.9
File Edit Sketch Tools Help
Control Elements
sketch_dec17a
Your Code
Error and Status Messages
9
Arduino/Genuino Uno on COM4

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

[11]

In the following, you find a short description of the most important control elements:

[12] You can use these buttons to check your program. The software shows possible errors. The left button just checks the program; the right one checks the program, and, after successful testing, transfers it to the microcontroller.

⚠ When sending the code to the microcontroller, you need to be patient. The upload may take a while. Just watch the progress bar, which you can see on the lower right.

[13] This button opens a new sketch.

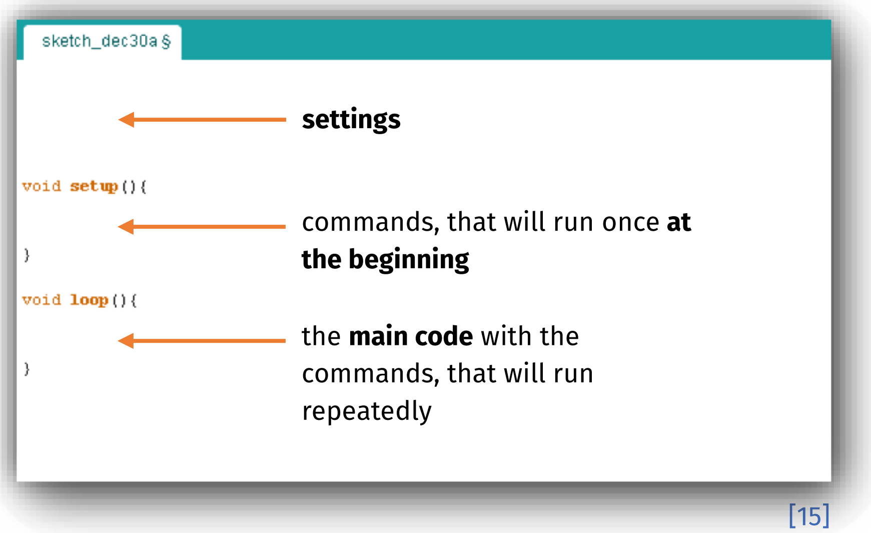
Sketch, that is the name for a program in Arduino language.

[14] These buttons open or save a sketch.

Programming: First Steps

Programming your Arduino is really simple. There are some rules, which you have to keep in mind. The rest is a piece of pie.

The following figure shows the default structure, which you will need for every program.



```

void setup(){

}
void loop(){

}
```

settings

commands, that will run once **at the beginning**

the **main code** with the commands, that will run repeatedly

[15]

If this default structure is – against expectations – not in your sketch, copy it.

Commands

You will tell your Arduino, what it is supposed to do. Add a semicolon (;) after each command to let the Arduino know, that a command ends.

Variables and Types

Variables are small containers, which store information. If you calculate something, you can store the result in a container. Every time you need the result, you do not have to calculate again; you just have to take a look at the container. The Arduino needs to know, which kind of information you want to store, e. g. numbers or words. For whole number storage, you use so called **int** variables (short for integer). If you want to declare (create) a new variable, you first have to specify the type (in this case `int`). Then you type in the name. Finally, you end the command with a semicolon.

```
       type name;
e. g.: int result;
```

This line declares a variable for whole numbers with the name “result”. You can directly store a value:

```
       type name = value;
e. g.: int result = 13;
```

Types, which are written in a correct way, will be colored orange.

Arduino distinguishes between upper and lower case. That means, that “result” is not the same as “Result”!

Programmers can take notes in the code. If you write something after two slashes //, it is just a note, and the program ignores it.

Enough of theory! Let’s start programming!

First, you have to change the circuit a bit, because now you want to control the LED with your Arduino.

1. To do so, first, unplug the USB cable.
2. Now, connect the upper yellow wiring, which was in the positive pole, with one of the digital pins of the Arduino.
   Note the **number of the digital pin**: _________
3. Reconnect the USB cable to the computer.

After this preparation of your Arduino, you can now start programming.

1. Save your sketch under a meaningful name in the Arduino folder. (If you cannot find the folder, please ask your teacher or a tutor.) You have already set up the default structure on page 6.
2. Declare a variable, which storages the pin, that you used for your LED.
   Reminder: A whole number is stored in a variable called **`int`**.

   ```
   int ledPin = ________;
   ```

   Copy this line to the **settings** (page 6). In the gap you add the number of the pin, in which you have plugged in the yellow cable.
3. Now do the **`setup()`**:
   a. The `setup()` is used to do some default settings. You can, for example, specify, if a digital pin is an out- or input.
   b. So, first you have to define the type of pin. Therefore, you can use the following command:

   ```
   pinMode (pin-name, pin-type);
   ```

   Note, how this line has to look like in your program:

   ```
   pinMode (_____________, _____________);
   ```

`pin-name` is your variable; `pin-type` is `OUTPUT`.

4. Now, do the main part: `loop()`:
   Your LED is connected to a digital output. This is the command, which turns a LED on and off:
   `digitalWrite (pin-name, condition);`
   Note, how this line has to look like in your program:

   `digitalWrite (______________, ______________);`
5. Save your code. To run the code, you have to plug in the USB cable, and click on the buttons, which check and transmit your code.

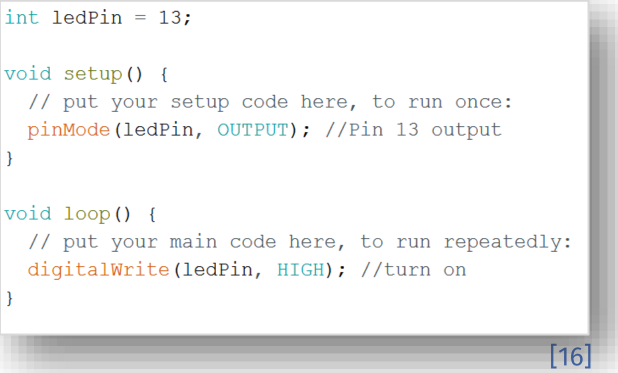
The `condition` is `HIGH`, if power is on, or `LOW`, if power is off.

If you have done everything correctly, your LED should light up again.

It does not work? Take another look at your code:

- Did you finish every command with a semicolon?
- Did you set every bracket [() and {}]?
- Does your variable use the correct pin number?
- Is everything written in the same and correct way?

If your code is still not working, you should compare it to the following sample solution:



```
int ledPin = 13;

void setup() {
  // put your setup code here, to run once:
  pinMode(ledPin, OUTPUT); //Pin 13 output
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(ledPin, HIGH); //turn on
}
```

[16]

Your LED is lightening? Great! You can continue with something new!

Now, you will let your LED blink! The command is really simple. As all commands in `loop()` are repeated over and over again, it should be sufficient to first turn the LED on and then off again.
Reminder: The opposite of `HIGH` is `LOW`.

Change your sketch, and test it.

This is weird! It seems, as if your LED is constantly lightening, but, indeed, it is blinking. The problem is, that `loop` runs too quickly; the human eye cannot notice, that the LED is permanently switched on and off. It is time to slow down a bit. For that, you can use a simple command:

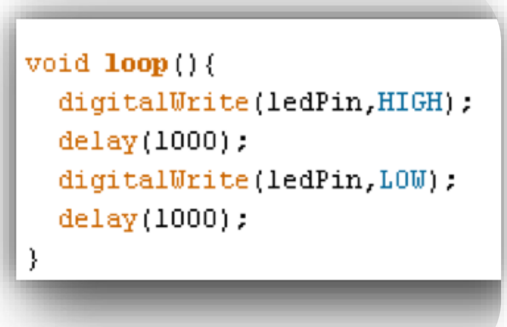
```
delay(time);
```

This command will pause the program for exactly `time` milliseconds.

`delay(1000);` will pause the program for one second.

Just add `delays` after each switching operation, and test your sketch again.

With this code the LED should blink:



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

[17]

Awesome, you did it! Your LED blinks.

Light at the Push of a Button

It is boring, when things turn on and off, and you do not have any influence on it. You will now learn to control the light manually. For this purpose, you need another constituent: the push button.

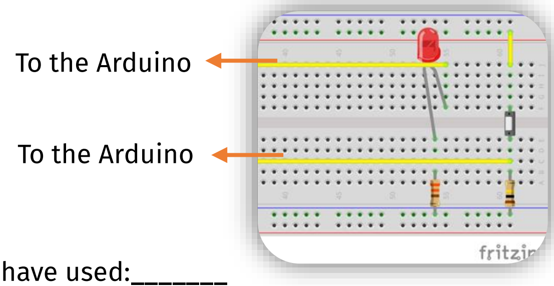
The Push Button

A push button is an element, which conducts electricity as long as the button is pushed down. If you take your finger away, the current cannot flow anymore. You can find a push button in your box.

This is, what it looks like: [18]

When you use the push button in your circuit, you need to pay attention to the following things. One side of the push button has to be connected to the positive pole, the other side to a **digital pin** of the Arduino. This side also has to be connected to the negative pole in combination with a strong resistor (**100 kΩ**, brown-black-yellow-gold).

Install the push button in your circuit. This is, what your circuit should look like.



[19]

Note the number of the digital pin, which you have used:_______

Try to retrace the current flow of the circuit.

Before you start programming, you have to set up a new sketch. Again, you need the default structure from page 6 (`loop()` and `setup()`). Take a look at your last sketch, and declare the variable for the LED. Two steps are important: to declare the **variable** (in the settings) and to define the pin as an **output** (in setup()).

On the following page, you will learn a new construction, which you will soon know like the back of your hand.

The If Construction

```
if(condition) {
  expression 1;
  expression 2;
  …;
}
//next line
```

The if statement checks for a condition, and executes the following expressions (in the curly brackets), if the condition is true. If the condition is not true, these expressions will be ignored, and the program will continue with the next line of code.

How can you formulate such an if statement? For instance, you can check, if the push button is pressed. In this case, the input value is high. So, a possible condition could be:

```
if (digitalRead(buttonPin) == HIGH) { expressions }
```

Parallel to **`digitalWrite()`**, which you used for outputs, you can now use **`digitalRead(pinName`**`)` for digital inputs.

A few things should strike you here:

1. What is `buttonPin`? Well observed! You have to add a variable for `buttonPin`, just like you already did for `ledPin`.
2. In `setup()`, you have to tell the Arduino whether the pin is an input or an output. Do you remember, what you have added to the `ledPin` in `setup()`? What is the corresponding command for an input?
   `pinMode(buttonPin, ______________);`
3. Until now, you have always used a simple "=". Suddenly, there is a double "==". That is not a typo. A simple "=" tells the Arduino to assign a value. Two "==" tell it to compare two values.

Try to switch on the LED every time you push the button.

To restart the Arduino, you do not always have to unplug the USB cable. On the Arduino, you find a small button with the label "Reset", which you can use to restart your program.

[20]

What else?

Step one has been done, but you have probably noticed: The LED is turned on, if the push button is pressed, but it does not turn off again, if the button is released.

You have to add a command, which tells the Arduino what to do, if the button is not pressed. For that, you need to expand the if statement like this:

```
if(condition){
      expression;
}
else {
      other expression;
}
```

You already know the upper part. The **else** tells the program, what to do, if the first condition is false.

Modify your sketch: The LED should be turned off, if you release the button.

Arduino to Programmers

Switching LEDs on and off is not the only way, in which the Arduino can communicate with its programmers. Particularly for future programs, it is important to get sensor values and mathematic results. For this purpose, the developers of Arduino have included a great tool.

The Serial Monitor

The serial monitor sends data from the Arduino to the computer. You can start this transmission by adding the following line of code to **`setup()`**:

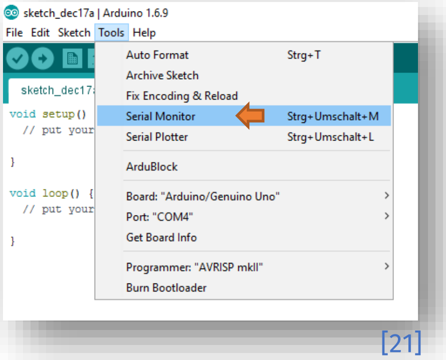
```
Serial.begin(9600);
```

The 9600 is the speed, at which data flows.
If you want to display this data, you can send it to the computer by using the following line of code in **`loop()`**:

```
Serial.println(data);
```

`data` can be any variable, but also text, which you have to put in inverted commas, e.g. “Turn on LED”.

When you have transmitted your program to the Arduino, you can look for **Tools → Serial Monitor** (or the appropriate key combination) in your tool bar to open the console in a new window.


sketch_dec17a | Arduino 1.6.9
File
Edit
Sketch
Tools
Help
sketch_dec17a
Auto Format Strg+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Strg+Umschalt+M
Serial Plotter Strg+Umschalt+L
ArduBlock
Board: "Arduino/Genuino Uno"
Port: "COM4"
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader

```
void setup()
  // put your

}

void loop() {
  // put your

}
```

[21]

Modify your sketch: After each switching operation a text should be displayed.

Congratulation, you have mastered the basics. Have fun with the other projects!

List of references:

Fig. 1, 2, 6, 10, 18, 20 – *Source: InfoSphere*

Fig. 3 to 5, 7 to 9, 19 – *Source: Screenshots of the Fritzing Software (https://fritzing.org/), CC BY-SA 3.0 Attribution-ShareAlike 3.0 Unported (https://creativecommons.org/licenses/by-sa/3.0/), 2022-07-20.*

Fig. 11 to 17, 21 - *Source: Screenshots of the Arduino Software (https://www.arduino.cc/en/software), GNU Lesser General Public License (https://www.gnu.org/licenses/lgpl-3.0.de.html), 2022-07-20.*

🎯, ⚠, 🖱, 🏁 – *Source: InfoSphere*