Welcome to the workshop that is all about light!
So you want to construct a park distance control system, measure speed, have a flower wander with the sun or have colors indicate temperatures?
That’s great, but before you can do all that, it’s important that you get to know all the material.
In this introduction, you will get an overview of the Arduino-Microcontroller, the components and the programming environment!

Goal of this introduction
Once you’ve finished this introduction, you will know

... what you can do with all this stuff.

Also, you will be able to...

... get an LED to shine,

... get it to blink,

... and turn it on and off with the help of a button.

To do all that, you will first learn how to connect everything, what you should pay attention to with the LED and how to program the Arduino!
The Arduino

First you need to take a look at your Arduino, which will play the main part in the upcoming projects.

![Arduino board and patch panels](image)

This is just a schematic presentation, which doesn't display some of the less important parts. Even this scheme includes more stuff than you need. Everything you need for now is this:

**USB port**

With the help of the USB connection you can connect the Arduino with the computer to transmit your program. In addition it will serve as a power supply.

**Electrical ports**

Some of the components need power. Even though it doesn't say it explicitly, just keep in mind:

- **GND** = − (negative pole)
- **5V** = + (positive pole)

**Digital** pins are either ON (in Arduino language: **HIGH**) or OFF (**LOW**). Digital pins can be used as input (e.g. for buttons) or output (e.g. for lamps).

**[Hint]** Only use the pins 2-13, please. The other two have a special role.
Contrary to the digital pins, the analog pins can only be used as input. That’s why we also call them analog input. They don’t just know HIGH and LOW, but 1024 different values in total. If you connect 0V, the value is also 0, at 5V the value is 1023. But more of that at a later point.

If you’ve found everything on the Arduino, you can take a look at the patch panels.

You can find many holes on your breadboard. Those are connected beneath the surface. You can see in this picture how that works:

![Breadboard with marked connections](image)

**Important:** Before you change anything in the wiring, always take the Arduino off the power! Just unplug the USB cable from the computer.

---

**Power on the board**

At the beginning, you should connect the outer input ledges with the Arduino. That way you can energize all components, so these ledges are like multi-outlet power strips.

Connect your Arduino with the patch panel using two cables (red to 5V, blue to GND)!

[Tip: The black cable in the picture is just an example. You can find many different cables in many different colors in your box!]

![Electrical connections](image)

![Cable](image)
Before you can start programming, you will first learn how the wiring works and how to know whether the current flows. Also you will get to know two important devices: LED and resistor.

**LEDs**

LED stands for **Light Emitting Diode**. So, this is a device that transmits light. Diodes are devices that can only let the light pass through in one direction. It’s easy to tell which direction that is: One of the little legs is larger than the other. The larger one has to be connected to the **positive pole**!

Unfortunately, LEDs are incredibly ambitious devices. They always want to produce as much light as possible. They heat up and in the worst case they can get so hot that they break.

Because of that, you always have to limit the power, when you are working with LEDs. That’s why we need **resistors**. You can find two different ones in your box. The ones with the **red-red-brown-golden** rings (220 Ω) are the right ones in this case.

It doesn’t matter whether you connect the transistor before or after the LED.

So this is how it looks like if you want to connect one LED:

Now it’s your turn. Take a **yellow** cable, the transistor and the LED and just connect it on your panel. **As a reminder**: The short leg should be connected to the negative pole!

If you’ve done everything correctly, it should look like this:

If your LED is not shining:

- Is your USB cable plugged in?
- Is your LED connected correctly?
- Do you have the LED, the transistor and the cables in the same row?
- Did you use the correct transistor?
As a start, this was awesome! Now you’re ready for the next exciting things, for example the programming of the microcontroller. On your computer, you can find a program, which is called just like your microcontroller, Arduino.

If you start the program you can see this window:

![Arduino Software Screenshot](image)

Fig. 11: Screenshot of the Arduino-Software

Before you can get started, you should learn about the most important elements:

- These two buttons check your program. Your software can point out some flaws in your code for you. The left button only checks your program and the right button sends it to your Arduino.

  [Note: You need to be patient when sending your codes to the Arduino, because the upload may take a while! Just watch the progress bar, which you can see on the lower right!]

- This button opens a new sketch.

- These buttons open or save a sketch.
Programming: First steps

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

This is how the **default structure**, which you will need for every program, looks like.

![Sketch structure](image)

**Fig. 12: Structure of an Arduino-Sketch**

1. **Commands:**
You will tell your Arduino what he’s supposed to do. Always add a semicolon “;” after each command, so the Arduino knows where a command ends.

2. **Variables und Types:**
Variables are small **containers**, where you can put information. If you calculate something, you can put the result in a container. Every time you need the result, you don’t have to calculate it again, you just have to look at your container.

But the Arduino has to know if you want to save numbers or words for instance. Whole numbers are saved using variables called “int” (short form for integers).

If you want to create (also called declare in computer science) a new variable, you always have to specify the type (in this case “int”), then type in the name and then end the command with a semicolon:

```c
<Type> <Name>;  
e.g.: int result;
```

This line creates variables for **whole numbers** and the name is called “result”.

You can also save a certain value:

```c
<Type> <Name> = <value>;  
e.g.: int number = 13;
```
The Arduino distinguishes between upper and lower case! So “Result” is not the same as “result”!

[Hint: Programmers can leave notes in the code. If you write something after two slashes // it is just a note and the program will ignore it.]

This is enough theory, now let’s start programing!
First, change your wiring a little bit, after all you want to control the LED with your Arduino. To do so, unplug the USB cable first.

Now connect the top of your yellow cable, which was in the positive ledge, to one of the digital pins of the Arduino.
Write down, which digital pin you used:

Your ToDo’s

1. Save your sketch. Give it a reasonable name and put it in an appropriate folder (ask your teacher or an instructor). You’ve already set up the default structure on page 5.
2. Now implement a variable, which saves which pin you used for the LED. As a reminder: A whole number is put in a variable called int. So:

   ```
   int ledPin = ____;
   ```

   You have to copy this line to Settings (see page 5).
   In the gap you add the number of the pin, where you plugged in the yellow cable.
3. Now do setup():
   a) The setup() is used, to set up some default settings. For example, you can specify if a digital pin is an input or an output.
   b) First, you have to define the type of pin using this command

   ```
   pinMode(<Pin-Name>, <Pin-Type>);
   ```

   Write down, what this line has to look like in your case:

   ```
   pinMode(______________, ____________);
   ```

4. Now the centerpiece: loop()
Your LED is connected to a digital output. The command that turns the LED on and off is called

   ```
   digitalWrite(<Pin-Name>, <condition>);
   ```

   What does this line have to look like in your case?

   ```
   digitalWrite(______________, ____________);
   ```

5. Save your code. To run the code, you have to plug in the USB cable and click on the buttons that check and transmit your code.

If you’ve done everything correctly, your LED should light up again!
It didn’t work? Make sure you have finished every command with a semicolon, that you’ve included all the brackets [ ( ) & { } ] and that your variable uses the number of the correct pin.
If it is still not working, compare your sketch with this sample solution!

```cpp
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
}
```

**Fig. 13: Programming code to switch on the LED**

Very good, the LED is glowing again. Let’s continue with something new: Let’s make the LED blink!

The command is really simple: Since `loop()` commands are repeated over and over, it should be sufficient to first turn the LED on and then off again.

**[As a reminder: The opposite of HIGH is LOW!]**

**Your ToDo's**
- Change your sketch and test how your program works.

That’s weird, it just keeps on glowing? Not really, it is actually blinking. But `loop()` is run so quickly that our eyes don’t notice that the LED is switched off.

It is time, to slow down the Arduino a bit. There is a simple command for that:

```
  delay(<time>);
```

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

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

**Awesome**, so you can make your LED blink with the help of your microcontroller!
This should make the LED blink:

```c
void loop() {  
  // put your main code here, to run repeatedly:
  digitalWrite(ledPin, HIGH); // turn on
  delay(1000);
  digitalWrite(ledPin, HIGH); // turn off
  delay(1000);
}
```

Fig. 14: Code to make the LED blink

Light at the push of a button!

It’s kind of boring to just have things turn on and off, and not to be able to control it manually. To do that you need a new 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 away your finger, the current cannot flow anymore.

You can find a push-button in your box, it looks like this:

![Fig. 15: Button](image)

There are some minor things that you have to pay attention to. One side of the push-button is connected to a **positive** pol, the other side is connected to a digital pin of the Arduino. This side also has to be connected to the **negative** pol in combination with a strong transistor (100 kΩ, **brown-black-yellow-gold**). If you’re interested why, have a look at the extra sheet Electrical Engineering under Pull-down.

Implement the push-button!

You wiring should look like the picture on the right:

![Fig. 16: wiring including the push-button an the LED](image)

Write down which digital pin you’ve connected to your push-button:

[Tip: Try to retrace the current flow from the Arduino through the wiring and back again! You can paint in the picture, if you like!]
Before you can start the programming, you have to set up a new sketch first. Again, you need the default structure from page 5, so `loop()` and `setup()`. Have a look at your last sketch and set up the variables of the LED. You have to follow two steps: You have to declare the variable (under Settings) and define the pin as output (in `setup()`).

You will get to know a new construction now, which you will soon know like the back of your hand: the `if`-command.

```plaintext
if (<condition>) {
  <command>
}
```

The `if`-command checks whether the preset condition is true. If it is, the commands in the curly brackets are executed. If not, they will be ignored and the program will continue with the next line of code.

But how do you formulate such a condition? For instance you could check whether the push-button is pressed, so whether the input value is `HIGH`! A possible condition could be:

```plaintext
if (digitalRead(buttonPin) == HIGH) {...}
```

Just like with `digitalWrite()`, which you used for outputs, you can now utilize `digitalRead(<Pin-Name>)` for digital inputs.

Some things should be noted:

1. What is `buttonPin`? Yes, well observed! You have to add a variable for `buttonPin`, just like you did for `ledPin`.
2. In `setup()` you have to tell the Arduino whether the pin is an input or an output. Do you still know what you added to the `ledPin` in `setup()`? What could the command for it be called? _____________________________
3. Before we only used a simple “=”.
   All of a sudden, we need a double “==”.
   That’s not a typo.
   A “=” tells the Arduino “assign a value”, but “==” tells it “compare both values”.

Your ToDo’s

- Enough of the talking, now it’s your turn! Try to switch on the LED every time the push-button is pressed!

[Small Hint: If you want to restart the Arduino, you don’t always have to unplug the USB cable. There is a small button on the Arduino, labeled “Reset”, which can be used too!]

What else?

Step one is done, but you’ve probably noticed something: The LED is turned on if the push-button is pressed, but it doesn’t turn off again, if the button is released!
Your code is missing a command which tells the Arduino what to do when the button is not pressed! You need to expand the `if`-command like this:

```cpp
if (<condition>) {
  <command>
}
else {
  <second command>
}
```

You already know the upper part. The "else" command tells the program what to do, if the first condition was false.

Are you able to change your sketch, so that the LED is turned off if you release the button?
Arduino for programmers!

Switching LEDs on and off is not the only way the Arduino can communicate with its programmer. Particularly for future projects, it is important to have the Arduino display sensor values and mathematical results.
For that, the developers 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 this line of code to `setup()`:

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

The 9600 means how quickly the data is transmitted through the USB.
If you want to display this data, you can send them to the computer by adding

```
Serial.println(<Your Data>);
```

to the `loop()`.

<your data> can be any variable, but also text that you put in inverted commas, e.g. “Turn on LED”.

After you've 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.

Change your sketch, so that after each switching operation a text is displayed!

Congratulations!
You've mastered all the basics. If you still have questions, just have a look at the supplementary sheets. Have fun with the other projects!

List of references:
**Figure 1, 2, 6, 10, 15, 17** – Source: InfoSphere
**Figure 3, 4, 5, 7, 8, 9, 16** – Source: Screenshots of the Fritzing-Software
**Figure 12, 13, 14, 18** – Source: Screenshots of the Arduino-Software

Any other graphics/icons – Source: InfoSphere