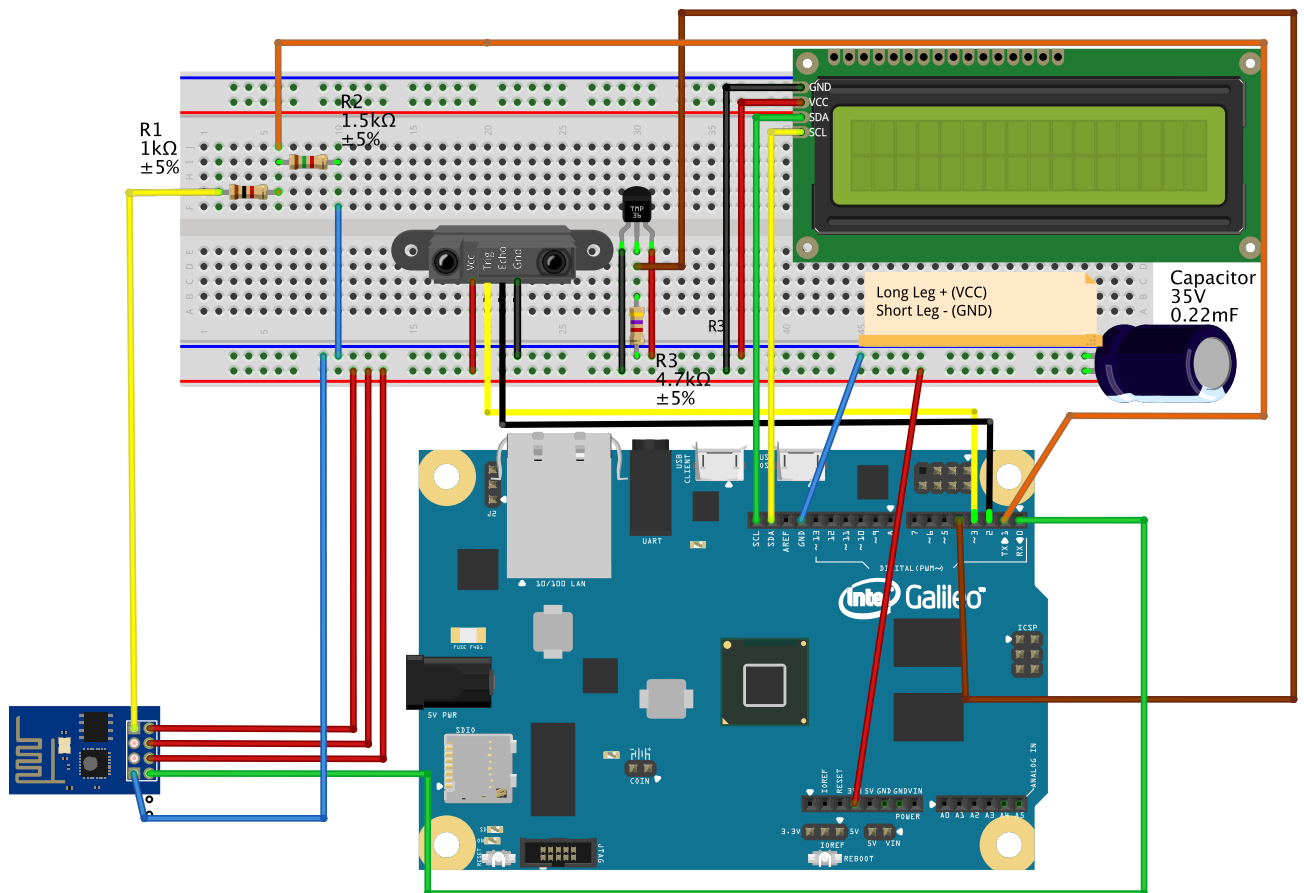


<b>Deadline</b>	<b>19th of January 12 p.m.</b>
	Send as email to your tutor
<b>Mail Subject</b>	<code>[PFE15][Assignment6]\$matriculationnumber</code>
<b>Accepted Formats</b>	PDF, txt
<b>File Name Convention</b>	<code>PFE15_Assignment6_\$matriculationnumber.\$format</code>
<b>Remark</b>	You are supposed to include all code examples into your document

## Circuit [0 Points]

On this exercise sheet, you are asked to implement a circuit for a remote controllable sensor platform. Here, the sensor readings can be viewed using an integrated web server over a wireless network. First of all, assemble the circuit as depicted in the figure below.

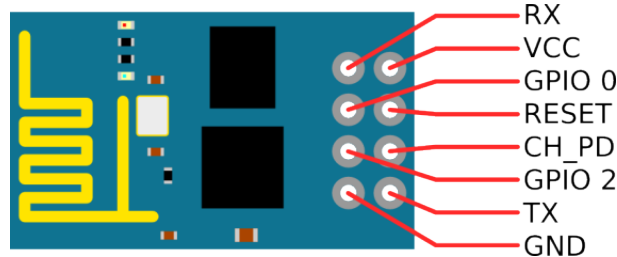


fritzing

The  $1k\Omega$  and  $1.5k\Omega$  resistors in this circuit serve as a *voltage divider*<sup>1</sup>. You can replace them with  $220\Omega$  and  $330\Omega$  resistors. An overview on the resistors color codes is at the end of the assignment sheet.

<sup>1</sup>[https://en.wikipedia.org/wiki/Voltage\\_divider](https://en.wikipedia.org/wiki/Voltage_divider)

The connectors for the ESP8266 wireless module are shown in more detail here.



⚠ **Never** connect the wireless module to the **5V** power output or the **VCC** of the board. Otherwise the wireless chip will be **damaged permanently**.

⚠ Uploading code to the board is performed using the serial console. Hence, the wireless modules TX and RX **must** be disconnected before any upload can be performed. Otherwise, the programming operation will fail. Make sure, that the variable `WifiSerial` is set to 0.

⚠ In case the temperature sensor has been incorrectly connected to the power source, it can become very hot. Always check the polarity while assembling the circuit.

## 1 Sensor platform [20 Points]

Download the file `webserver.ino` from our course webpage and install the required libraries in your Arduino IDE. Employ the following changes to the web server available on the course web page:

1. Measure the current temperature.


The provided code template already initializes the temperature sensor. However, you need to import the libraries (as shown in assignment 2 on exercise sheet 4) *Arduino-Temperature-Control-Library* and *OnWire* in order to successfully compile the code. Both are provided on the course web page. Afterwards, the temperature can be obtained in a similar fashion as shown below:

```
int temperature;  
tempSensor.requestTemperatures();  
temperature = tempSensor.getTempCByIndex(0);
```

⚠ Don't forget to adjust the constant `TEMPARTURE_PIN` with the digital pin which the temperature sensor is connected to in your assembly.

2. Use the ultrasonic sensor to determine the current distance of your sensor platform from a potential object nearby.
3. Display the temperature and the current distance using the web server. (The degree symbol "°" can be modeled in HTML with `&deg;`.)

## 2 Remote control [20 Points]

 Please note that the already provided LCD library has a bug that prevents writing strings consisting of more than one character to the LCD using `lcd.print()`. This bug only affects newer versions of the Arduino SDK. However, an updated version is available now and should be installed first.

In this exercise, you are asked to extend the already provided web server with parameter support. It should be possible to supply arbitrary strings that can be displayed on the connected LCD. The parameter `path` of the function `void display_page(char *path)` contains the path supplied in the request of a web browser.

The implementation of `display_page()` should satisfy the following requirements:

1. In case the path `/on` is requested, the LCD background light should be turned on.
2. In case the path `/off` is requested, the LCD background light should be turned off.
3. Otherwise the path should be displayed on the LCD screen.

As the wireless module is connected to the serial port of the board, it is not possible to use the serial console for debugging. However, debug information can be displayed on the LCD display as well (e.g. the result of a string comparison operation).

## 3 Percent encoding [10 Points]

In order to avoid encoding or ambiguity issues, a path supplied as part of a URL must never contain white spaces or special characters. Hence, the string `Hallo, Welt!` will be encoded using percent encoding<sup>2</sup> to `Hallo,+Welt%21`.

Extend the last exercise with support for decoding percent encoded paths. In more detail, your code should transform the path using the following algorithm:

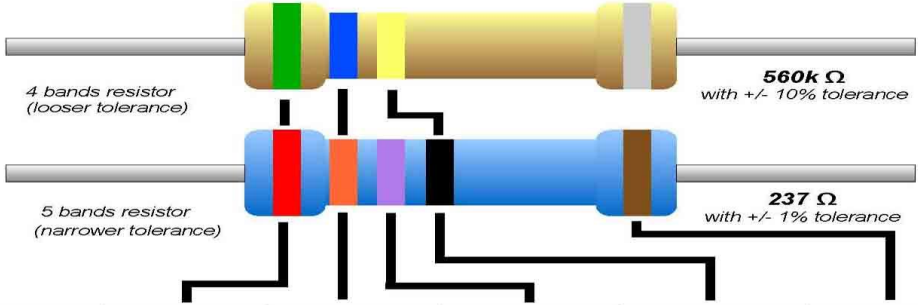
1. The char sequence is traversed character by character.
2. The first character is always a `/`. It should be ignored.
3. Whenever a `+` is read, a white space is printed on the LCD.
4. Whenever a `%` is read, the following 2 characters have to be read in and treated as an hexadecimal digit. The represented character in the ASCII table can be printed using `void lcd.write(char c)`.
5. Every other character should be printed unmodified.
6. Character sequences that exceed 16 characters should be displayed wrapped (on character, not word boundaries).

**Example:** If the path `/Hallo,+Welt%21` is requested, the LCD should contain the text `Hallo Welt!`.

<sup>2</sup><https://en.wikipedia.org/wiki/Percent-encoding>

## Resistor Color codes

**Resistor Color Code**



Color	1 <sup>st</sup> Band	2 <sup>nd</sup> Band	3 <sup>rd</sup> Band	Multiplier	Tolerance
Black	0	0	0	x 1 Ω	
Brown	1	1	1	x 10 Ω	+/- 1%
Red	2	2	2	x 100 Ω	+/- 2%
Orange	3	3	3	x 1K Ω	
Yellow	4	4	4	x 10K Ω	
Green	5	5	5	x 100K Ω	+/- 5%
Blue	6	6	6	x 1M Ω	+/- .25%
Violet	7	7	7	x 10M Ω	+/- .1%
Grey	8	8	8		+/- .05%
White	9	9	9		
Gold				x .1 Ω	+/- 5%
Silver				x .01 Ω	+/- 10%

[http://nearbus.net/wiki/images/7/7d/Resistor\\_color\\_codes.jpg](http://nearbus.net/wiki/images/7/7d/Resistor_color_codes.jpg)