

## **'We are the makers - IoT' Learning Scenario: glass fiber communication using Adafruit CPX Express**

Author: Thomas Jörg, Johannes-Kepler-Gymnasium Weil der Stadt

*The following paper was developed and tested in a school-environment with ca. 16 students of age 13-14 in the schoolyear 2019/2020. It reflects the experience with talented and inquiring students which made their programming scripts by themselves after a network technology teaching unit. This paper is supposed to be a recommendation as a starting point.*

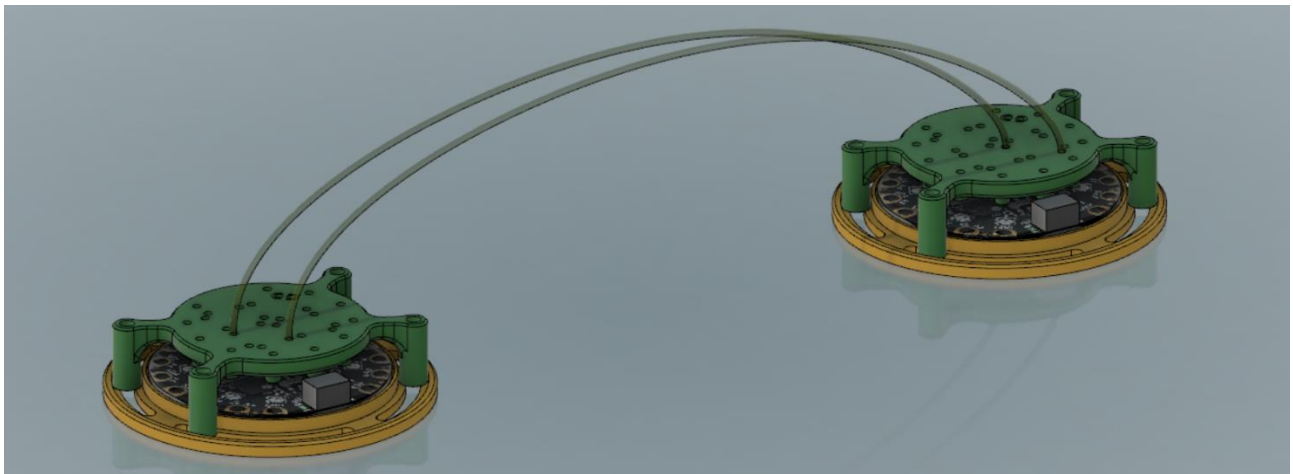
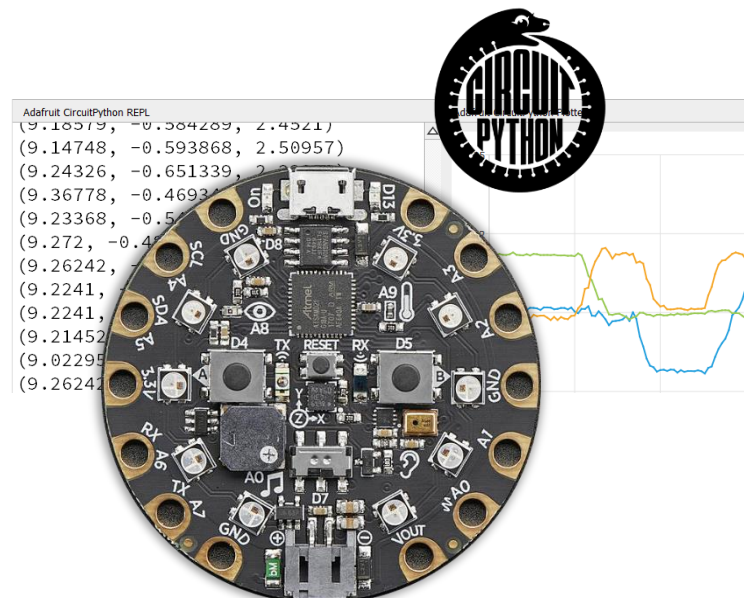


Figure 1: Prototype of a glass fibre communication setup of two microcontrollers

IoT means networking of computer-controlled devices. Network communication often means that the devices exchange their information wirelessly, i.e. via radio. In fact, classic carrier media such as the fiber optic connection are also included. The didactic advantage of using fiber optics for the introduction to communication technology is the visibility of the exchange of information: you can see the light pulses with which information bits are exchanged. A simple physical principle, namely total reflection, is sufficient to understand modern fiber optic communication.



<b>1. Title of Scenarios</b>	<b>IoT Device Network Communication: Glass Fiber Programming</b>
<b>2. Target audience</b>	<ul style="list-style-type: none"> <li>14 - 16 years</li> </ul>
<b>3. Duration</b>	<ul style="list-style-type: none"> <li>Minimum 7 weeks with 2-3 lessons per week</li> </ul>
<b>4. Learning items discussed during class</b>	<ul style="list-style-type: none"> <li>Sensors and actuators in the exchange of information between digital devices</li> <li>Principle of protocol- and packet-based network communication</li> <li>Application of total reflection for the transport of light pulses.</li> <li>Programming of python-based microcontrollers in small groups of two students each</li> </ul>
<b>5. Expected learning outcomes</b>	<ul style="list-style-type: none"> <li>How does an IoT system work?</li> <li>How to structure and implement network communication?</li> <li>Why do you need a communication protocol?</li> <li>How does packet-based digital communication work?</li> </ul>
<b>6. Methods</b>	<ul style="list-style-type: none"> <li>In this scenario students will construct, build, and program a serial communication between two microcontroller-devices from scratch by themselves. Students will also use the Serial Monitor and Serial plotter for visualizing and plotting data.</li> </ul>
<b>7. Setting</b>	<ul style="list-style-type: none"> <li>A class set of fiber optic cables to connect the microcontrollers</li> <li>A class set of Adafruit CPX microcontrollers,</li> <li>These CPX controllers are programmed with Circuit python</li> <li>Each student receives a laptop with a pre-installed Mu editor for their CPX microcontroller</li> <li>Each CPX is built into a 3D-printed enclosure, which ensures the precise alignment of the glass fibers.</li> <li>Each student writes a log of their project work</li> </ul>

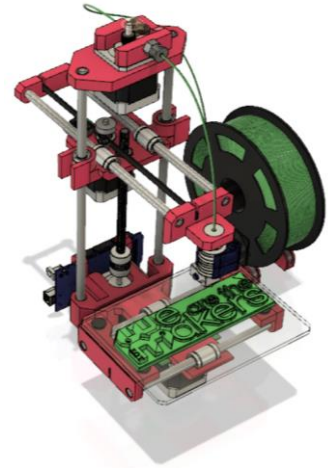
## 8. Tools, Materials and Resources

### 3D-Drucker

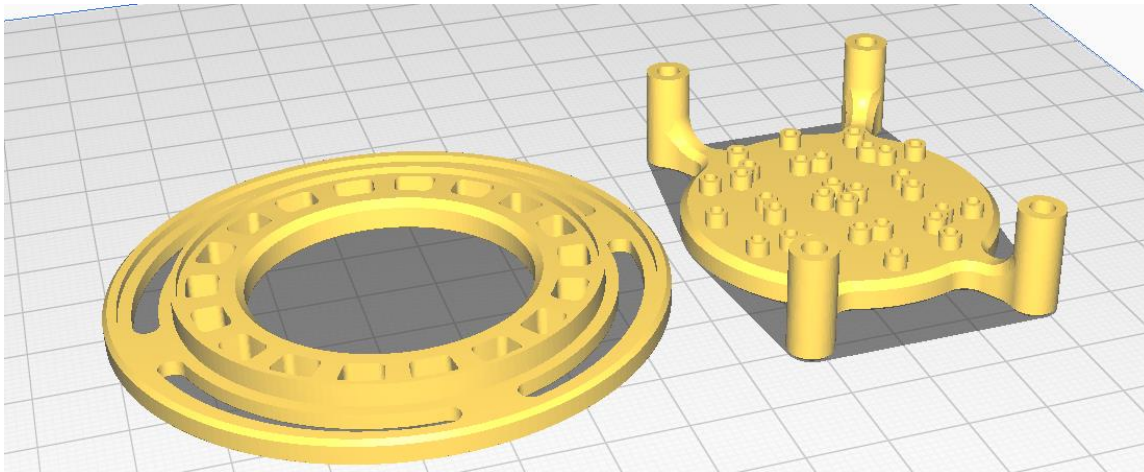
About 2-3 3D-printers are necessary since students will print their CPX-housings. Of course, it's possible for the students to construct machine parts by themselves

### 3d printed components:

As a starting point, all necessary parts are provided in .stl-format and as Autodesk Fusion 360 Files:



*Figure 2: A model of a 3D-Printer*



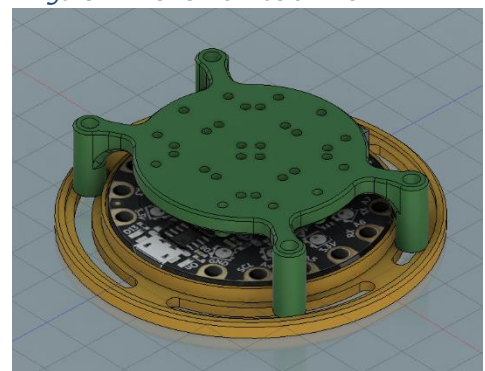
*Figure 3: STL-File. Printing time for both parts about 1 hour*

The two parts are dimensioned in such a way that a CPX with about 0.5mm game can be inserted into the lower part. The upper and lower parts are connected with M3 screws: Recommended length of the screw is 25mm. The parts can be locked with wing nuts, so that the microcontroller can be quickly removed and removed.

*Figure 5: M3 Screw and wing nut*



*Figure 4: Preview of Fusion-file*





*Figure 6: 100 meters of glass fibre*

The enclosure is designed to easily plug in the fiber optic cables and position them optimally above the LEDs and the brightness sensor.

A 1.5mm PMMA cable is used as fibre glass. You get these fibers in rolls of about 100 meters in length for about 20 euros. Each group of students who want to connect two CPXs requires 2 by 1 meter of cable. A cable as a transmitting line and a cable as a receiving line. With such a role, several classes can be supplied with fiber optics cheaply.

bending or buckling.

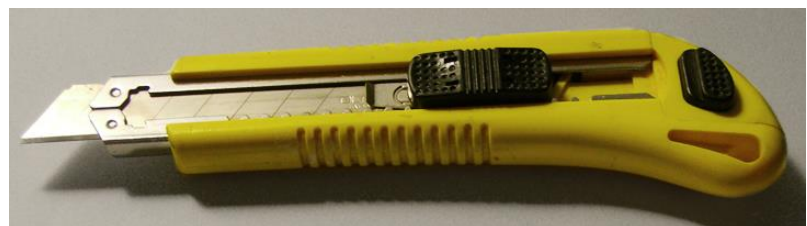
The 1.5mm diameter ensures that students can't get hurt so easily. In addition, fibres of this thickness are robust against

The upper part of the 3D-printed enclosure has many holes, which are intended as a guide for these cables. The pre-constructed bore has a diameter of 1.8mm.



*Figure 7: Hand drill*

If it should occur due to fast and poor quality pressure that the cable cannot be pushed through the openings of the enclosure, then can be reworked with a small hand drill. These hand drills are hand-guided and therefore harmless; they are usually delivered with a good basic equipment on various drills.



*Figure 8: Cutter*

To cut the glass fibers, you should not use pliers or scissors because they squeeze the fiberglass. A smooth cut is required from which the light can step out. Therefore, it is recommended to use a carpet knife.



## The Adafruit CPX

In this work we use the Python-based microcontroller "Circuit Python Express" of the company "Adafruit Industries". This board is a convenient way to use a microcontroller together with many prefabricated and built-in actuators and sensors in the classroom.

The CPX has proven to be robust and reliable in multiple teaching applications. In addition, Adafruit also offers a lot of information about the microcontroller. In addition to numerous sample programs, there is also good documentation.

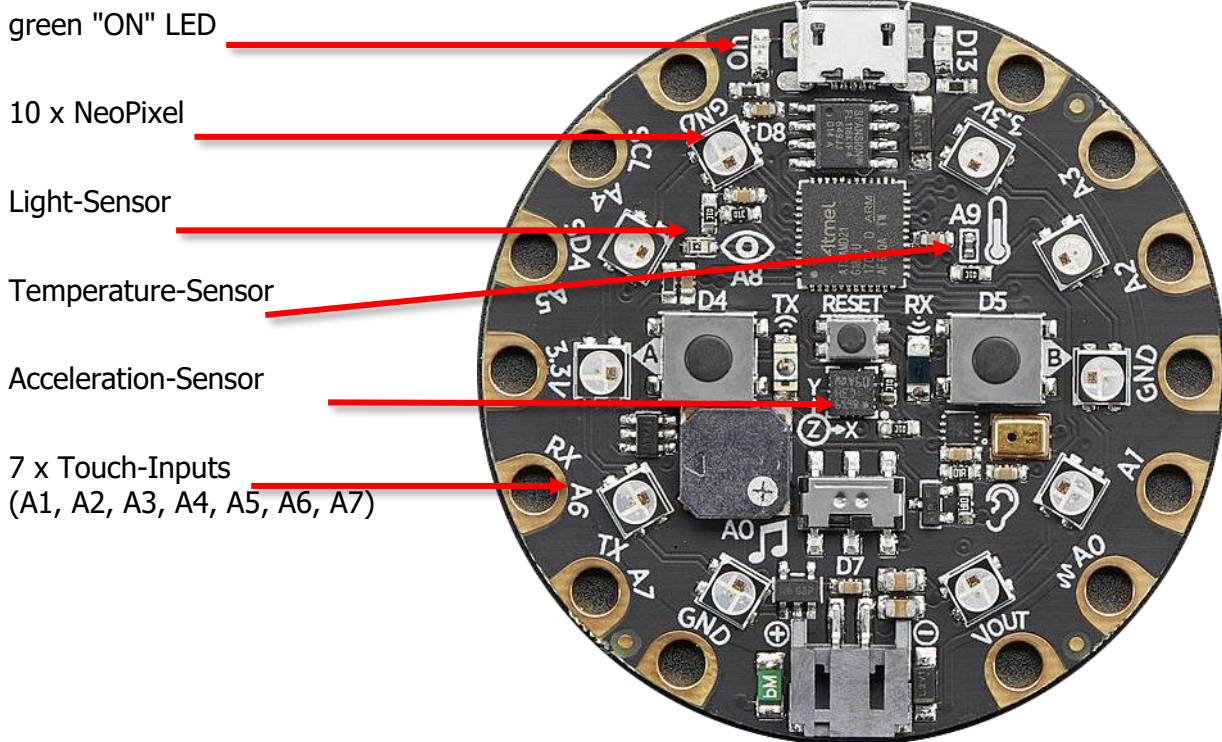


Figure 9: some features of the CPX

A tutorial on how to introduce the programming of the CPX together with many sample programs can be found here:

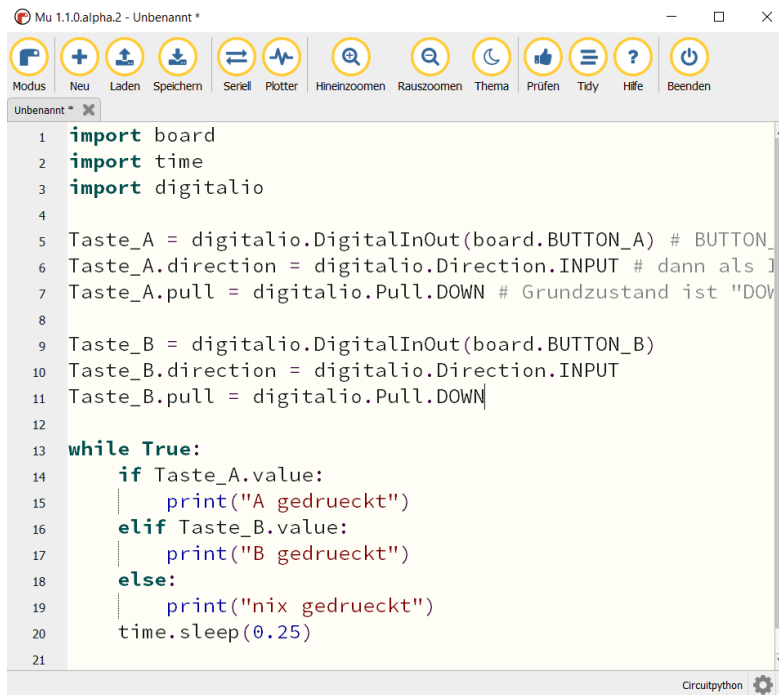
[https://iludis.de/?page\\_id=291](https://iludis.de/?page_id=291)



## Necessary equipment

The computers that students work with should have the following software installed:

- Autodesk Fusion 360 (or any other 3D-modeling-Software, e.g. Wings3D)
- CURA slicing software,
- An internet connection for downloading libraries
- **Mu Editor** (<https://codewith.mu/>)



The screenshot shows the Mu Editor window titled 'Mu 1.1.0.alpha.2 - Unbenannt \*'. The interface includes a toolbar with icons for Modus, Neu, Laden, Speichern, Seriell, Plotter, Hineinzoomen, Rauszoomen, Thema, Prüfen, Tidy, Hilfe, and Beenden. The code editor contains the following Python code:

```

1 import board
2 import time
3 import digitalio
4
5 Taste_A = digitalio.DigitalInOut(board.BUTTON_A) # BUTTON_
6 Taste_A.direction = digitalio.Direction.INPUT # dann als I
7 Taste_A.pull = digitalio.Pull.DOWN # Grundzustand ist "DOV
8
9 Taste_B = digitalio.DigitalInOut(board.BUTTON_B)
10 Taste_B.direction = digitalio.Direction.INPUT
11 Taste_B.pull = digitalio.Pull.DOWN
12
13 while True:
14     if Taste_A.value:
15         print("A gedrueckt")
16     elif Taste_B.value:
17         print("B gedrueckt")
18     else:
19         print("nix gedrueckt")
20     time.sleep(0.25)
21

```

The bottom right corner of the window shows 'Circuitpython' and a gear icon.

Figure 10: Screenshot Mu Editor

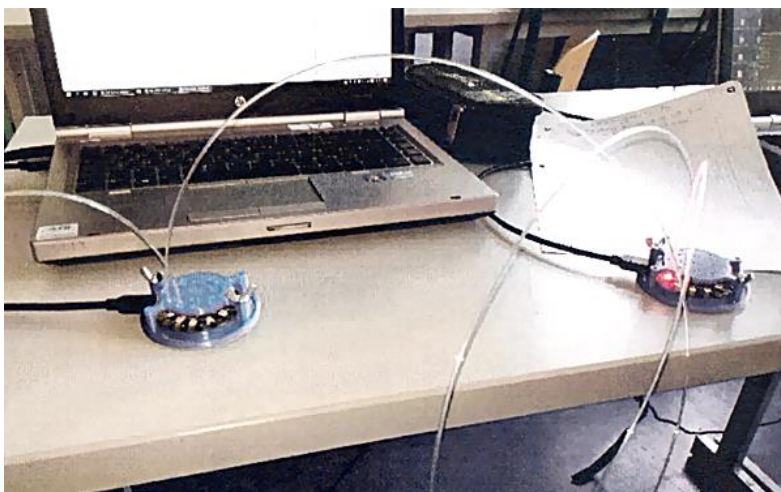


Figure 11: Photograph of students setup

## 9. Lesson plan: Step by step description of the activity/ content

### Lesson 1 & 2 (90min): Introduction to IoT

Students will be introduced IoT by examples: Vacuum robots with app remotes, internet-based weather stations, smart farming and last but not least health applications. Students should examine how those devices work and which components are needed: a microcontroller based system controls and coordinates attached sensors and actors. Furthermore it communicates and coordinates with other systems of similar type often via wireless communication networks. Parts needed: Sensors, Actors, Communication devices. Possibilities and threats need to be discussed and also limitations: where does IoT make sense and where not?

### Lesson 2 & 3 (90min): Introduction to the theory of communication protocols

The students of the class play a role-playing game: the class watches and develops ideas. The following rules apply:

- Two students "transmitter" and "receiver" sit together on chairs, which stand back to back so that both can not see each other between the two stands a third, empty chair.
- The transmitter is supposed to send the receiver two completely different words, which are spoken backwards a completely different meaning, such as the pairs of words:  
(4 letters: "LIVE"/"EVIL" and "STAR"/ "RATS" | 3 letters: "GOD"/ "DOG" and "RAW"/"WAR")
- Like spoken language, word transmission allows only one sequence of individual characters in succession, so all letters must be transmitted individually. Therefore, the letters of the two words are written on individual notes.
- These notes can only be transferred one by one from the sender to the receiver by placing a note on the chair and the receiver picking up the note there – without the two parties being able to see each other.
- The only direct communication allowed between the two parties is a single tone (e.g. "beep") that everyone is allowed to give.
- If a communication goes wrong, the message transmission is interrupted, a new rule is established, and students start again.

It makes sense for students to realise that communication must be made through jointly agreed rules, namely a protocol:

Students must agree on a start signal. There must be a pause in time after each letter for the characters to arrive in the correct order; this can be agreed by clocking or by "beeps". Furthermore, the order in which the letters are exchanged must be agreed. Otherwise, the words will have a different, unintended meaning. And finally, the communication must be stopped.

## Lesson 4 & 5 & 6 (120min): Principles of serial communication

Principles of multiplexing and demultiplexing: Data bits are transmitted one by one, starting from the MSB ('most significant bit') to the LSB ('least significant bit') in a specified order via a single data cable.

A distinction is made between synchronous and asynchronous data transmission: In the simpler synchronous case, the receiver – which functions as a master – specifies the common clock frequency with which it triggers individual bit transmission at the transmitter (slave). After a pre-agreed number of transmitted bits, the data transfer is finished.

In the asynchronous case, a cycle time must be agreed for both devices involved in the communication beforehand, although these times may differ only truly little from each other.



### Lesson 7 (45min):

#### History of data transmission: Émile Baudot

Using the 5-bit baudot code, the basics of data transmission are developed using the practical example. If you only want to transfer uppercase letters, you need 26 different letter characters and possibly 2-3 punctuation marks, such as the space or the question mark. To encode these less than 32 different symbols with bits, you need 5 bits; possible coding would be, for example,:

Figure 12: Émile Baudot

Symbol	A	B	C	D	E	F	G	H	I	J	K
Bitcode	00001	00010	00011	00101	00110	00111	01000	01001	01010	01011	01100

Symbol	L	M	N	O	P	Q	R	S	T	U	V
Bitcode	01101	01110	01111	10000	10001	10010	10011	10100	10101	10110	10111

Symbol	W	X	Y	Z	LEER	START	STOP	.	
Bitcode	11000	11001	11010	11011	11100	11101	11110	11111	00000

Students should consider a system that can be used to encode letters bitwise—ideally they come up with ideas similar to those in the table above.

Since bits are transmitted at a certain speed, it is called a so-called baud rate, which was named after the French engineer Emile Baudot. Here you can go into the biography of Baudot.



## Lesson 8 & 9 (90min): Repetition Total Reflection

### Total reflection

We use the simulations „Light Refraction“: [https://javalab.org/en/light\\_refraction\\_en/](https://javalab.org/en/light_refraction_en/)

And „Total Internal Reflection“: [https://javalab.org/en/total\\_internal\\_reflection\\_en/](https://javalab.org/en/total_internal_reflection_en/)

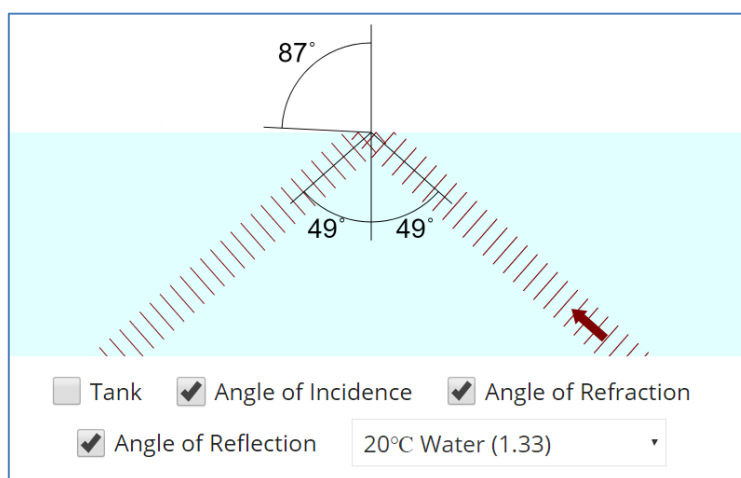
For the transition from water to air,

"The light beam is broken off the solder during the transition from the optically denser medium (water) to the optically less dense medium (air)"

If the angle of incidence of the light beam in the water exceeds a so-called "**critical angle**", then the refractive angle in the medium air would have to be greater than  $90^\circ$  – and that is impossible! Therefore, the beam of light has no choice but to remain in the water.

1) critical angles when moving from different media to air

[https://javalab.org/en/light\\_refraction\\_en/](https://javalab.org/en/light_refraction_en/)



Determine the critical angles – i.e. those angles of incursion at which the light beam does NOT make it out of the medium:

Medium	Critical angle, above is total reflection
Water	$49^\circ$
Diamond	
Sapphire	
Glass	

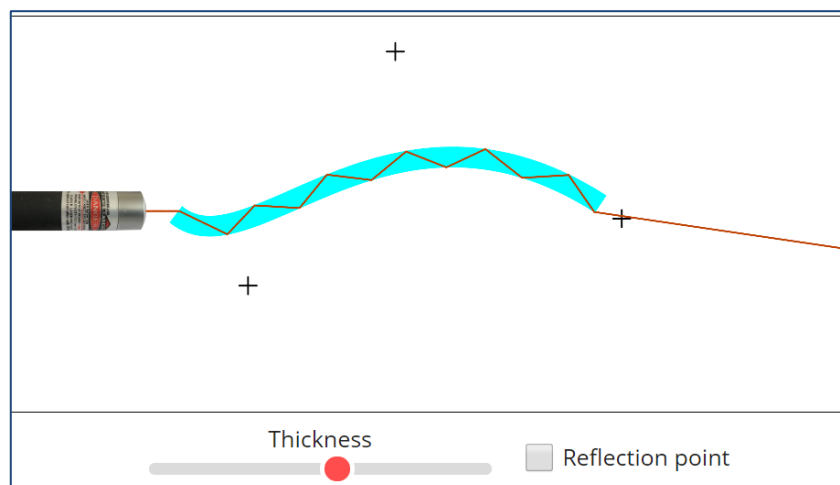
2) Interpret the following two images:



Interpret the selected areas in the two images in their own words. How do these reflections come about? And what exactly do you see in Area 3?

3) Applications of Total Reflection

[https://javalab.org/en/total\\_internal\\_reflection\\_en/](https://javalab.org/en/total_internal_reflection_en/)



On the left side is a laser that feeds its beam of light into a so-called glass fiber. Fiber optics are used to transport information with light. On the one side of the fiber optic computer signals is irradiated by means of laser beam and on the other hand by means of a light sensor the light signals are received and passed on to the target computer. Explain the operating principle of the glass fibre, answering the following questions:

1. Why does the light beam remain in the fiber?
2. Are there situations where the light beam leaks out of the fiber too early?
3. How does the light line within the fiber depend on the thickness of the fiber?

## Lesson 10 & 11 (90min), Introduction to programming with Python:

the CPX is connected to the computer via USB cable and programmed with the programming environment "mu-Editor". Here are some scripts to get used to the CPX. Quelle:

[https://iludis.de/?page\\_id=291](https://iludis.de/?page_id=291)

### Script 1, "Blink", LED switch on and off:

```
import board
import digitalio
import time

meinPin = digitalio.DigitalInOut(board.D13)
meinPin.direction = digitalio.Direction.OUTPUT

while True:
    meinPin.value = True
    time.sleep(1)
    meinPin.value = False
    time.sleep(1)
```

### Script 2, "beep", play Tone:

```
import time
from adafruit_circuitplayground.express import cpx

for i in range(1, 5, 1):
    cpx.start_tone(262)
    time.sleep(0.2)
    cpx.stop_tone()
    time.sleep(0.2)
    print(i)
```

### Script 3, "Touch", Play sound when touch is pressed:

```
import board
import time
import touchio
from adafruit_circuitplayground.express import cpx

B_A1 = touchio.TouchIn(board.A1)
B_A2 = touchio.TouchIn(board.A2)

while True:
    if B_A1.value == True:
        cpx.start_tone(262)
    else:
        cpx.stop_tone()
    if B_A2.value == True:
        cpx.red_led = True
    else:
        cpx.red_led = False

    print(B_A1.value, B_A2.value)
    time.sleep(0.1)
```

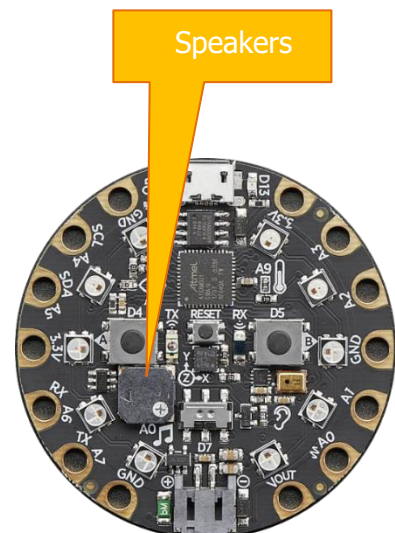


Figure 13: Location of the loudspeaker

## Script 4, Neopixel switching

```
import board, time, neopixel
NeopixelListe = neopixel.NeoPixel(board.NEOPIXEL,
10)

for i in range (10):
    NeopixelListe[i] = (0,64,0)
    print(NeopixelListe[i])
    time.sleep(0.1)
print(NeopixelListe)
```

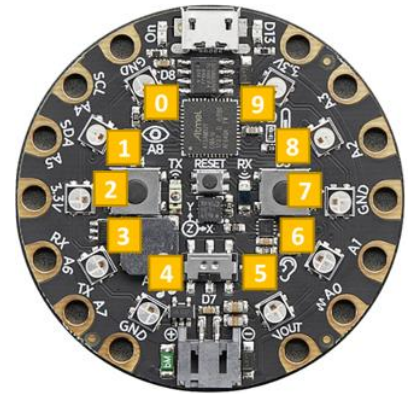


Figure 14: The numbering of the neopixels

## Skript 5, light sensor reading

```
import board
import time
import analogio

licht = analogio.AnalogIn(board.LIGHT)

while True:
    print((licht.value,))
    time.sleep(0.1)
```

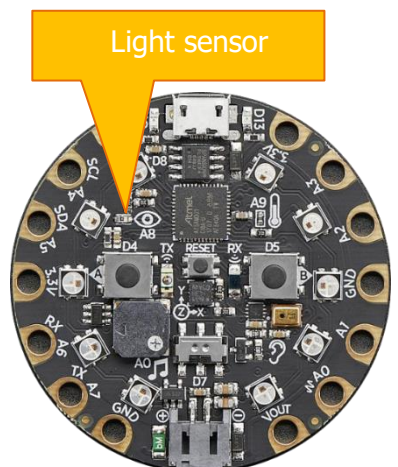


Figure 15: Location of the light sensor

## Lesson 12 & 13 (90 min): Example of synchronous communication

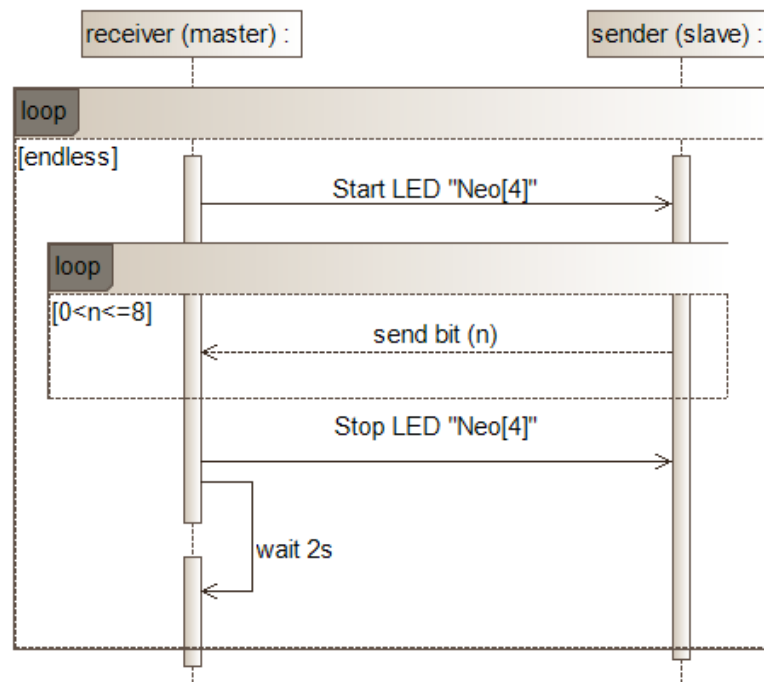


Figure 16: UML-Sequence diagram of synchronous communication

receiver source code	sender source code
<pre> import board import time import neopixel import analogio  light = analogio.AnalogIn(board.LIGHT) Neo=neopixel.NeoPixel(board.NEOPIXEL,1 0) pulseDuration = 0.05  while True:     listenBits = [2, 2, 2, 2, 2, 2, 2, 2]     Neo[4] = (0, 255, 0)     for i in range(len(listenBits)):         time.sleep(pulseDuration)         if light.value &lt; 40000:             listenBits [i] = 0         if light.value &gt; 40000:             listenBits [i] = 1     Neo[4] = (0, 0, 0)     print(listenBits)     time.sleep(2) </pre>	<pre> import board import time import neopixel import analogio  light = analogio.AnalogIn(board.LIGHT) Neo=neopixel.NeoPixel(board.NEOPIXEL,1 0) pulseDuration = 0.05  while True:     sendBits = [1, 0, 0, 1, 1, 0, 1, 0]     if light.value &gt; 40000:         for i in range(len(sendBits)):             if sendBits[i] == 1:                 Neo[6] = (255, 0, 0)             if sendBits[i] == 0:                 Neo[6] = (0, 0, 0)             time.sleep(pulseDuration) </pre>



## Lessons 14 & 15 (90 min): Example of asynchronous communication

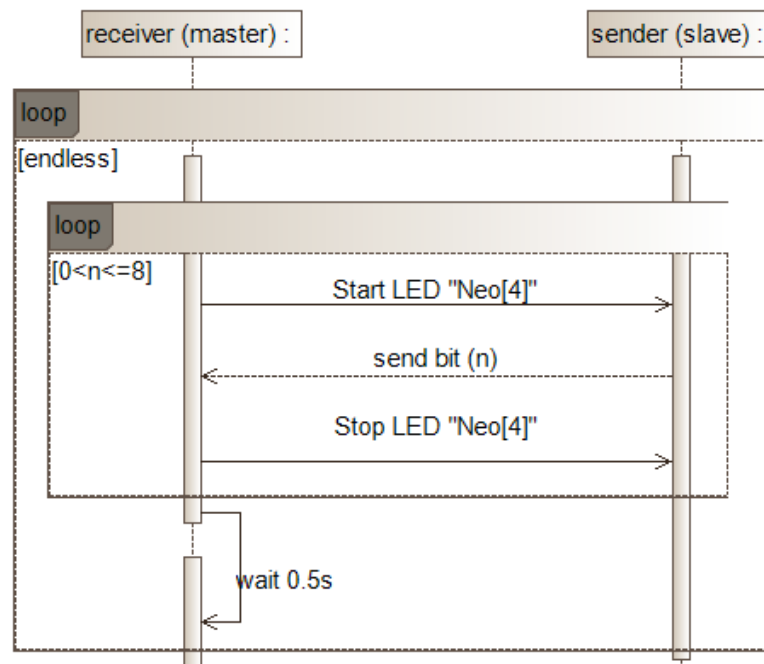


Figure 17: UML-Sequence diagram of asynchronous communication

receiver source code	sender source code
<pre> import board import time import neopixel import analogio  light = analogio.AnalogIn(board.LIGHT) Neo=neopixel.NeoPixel(board.NEOPIXEL, 10) pulseDuration = 0.02  while True:     listenBits = [2, 2, 2, 2, 2, 2, 2, 2]     for i in range(len(listenBits)):         Neo[4] = (0, 255, 0)         time.sleep(pulseDuration)         if light.value &lt; 40000:             listenBits[i] = 0         if light.value &gt; 40000:             listenBits[i] = 1         time.sleep(pulseDuration)         Neo[4] = (0, 0, 0)         time.sleep(pulseDuration)     print(listenBits)     time.sleep(0.5) </pre>	<pre> import board  import neopixel import analogio  light = analogio.AnalogIn(board.LIGHT) Neo=neopixel.NeoPixel(board.NEOPIXEL, 10) sendBits = [1, 0, 0, 1, 1, 0, 1, 0] i = 0 transmit = True  while True:     print(i)     if light.value &gt; 40000 and transmit:         if sendBits[i] == 1:             Neo[6] = (255, 0, 0)         if sendBits[i] == 0:             Neo[6] = (0, 0, 0)         transmit = False     if light.value &lt; 40000 and not transmit:         i = i+1         i = I % 8         transmit = True </pre>

<b>10. Feedback</b>	<p>At the end of the module, students should have developed a deeper understanding of how serial communication works and what computer principles are necessary for implementing this technology. During the lessons, important aspects of electronics, optics and protocol construction are discussed.</p>
<b>11. Assessment &amp; Evaluation</b>	<p>Students keep their labor journal, which can be reviewed by the teacher. Students can also present the results of their experiments. In addition, a standard in-class-test must be conducted at the end of the lessons.</p>