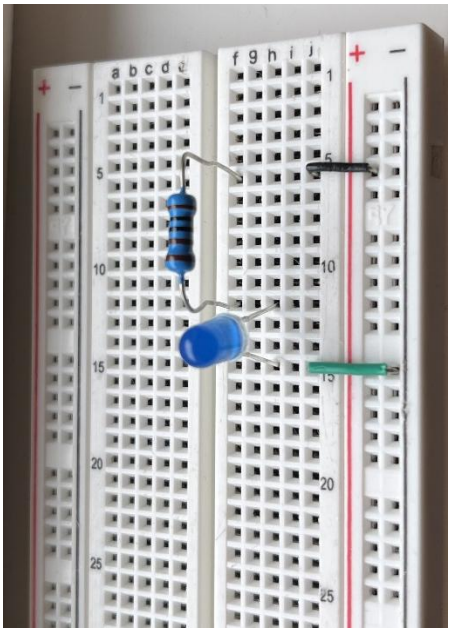
Welcome to the workshop with Akkodis!

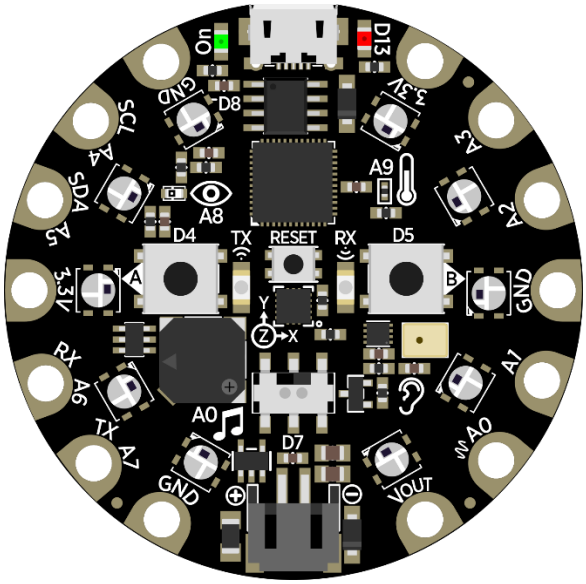
Electronics and coding tasks

Today, you will learn about circuits and transistors! In addition to learning the fundamental theory behind these concepts, you will get hands-on experience by building circuits that demonstrate how they work in real life.

We have also included a programming task! It is suitable for beginners as well as those with some or a lot of programming experience.

If you have any questions along the way, feel free to ask any of the supervisors!





Task 1: A Simple Circuit to Turn on an LED

In this task, you are going to build a simple circuit with a switch, a resistor, and an LED. To do so, you will need the following components:

- 1 breadboard
- 2 resistors: 1 kΩ and 47 kΩ
- 1 LED
- 1 button switch
- Wires

Before getting started, you will learn a little about electric circuits and how some of the components listed above work.

Get Familiar with Electric Circuits

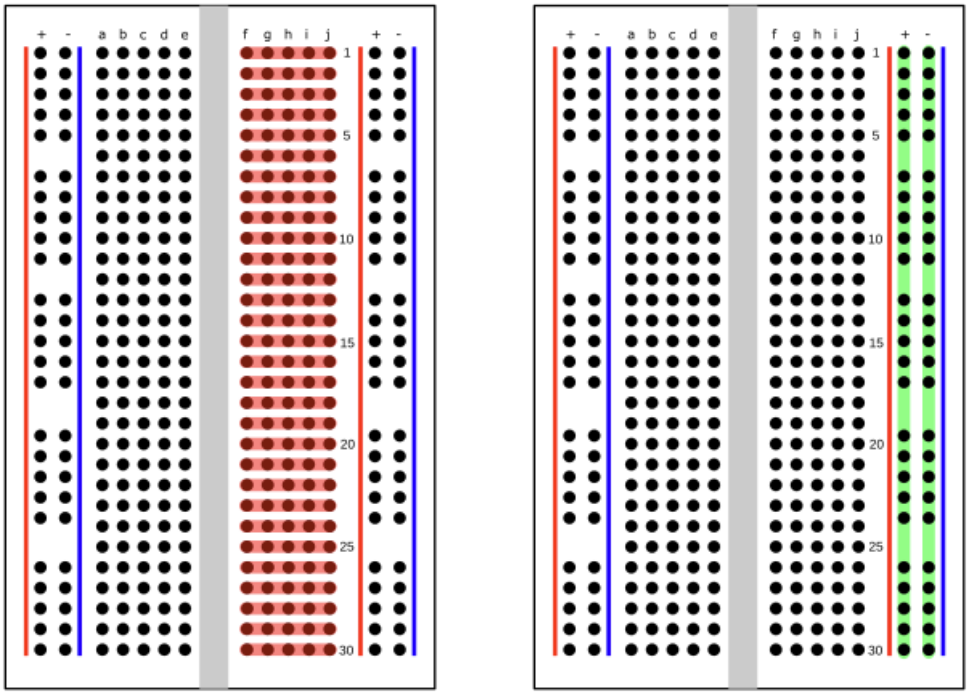
Electric Circuit

An electric circuit is a closed loop that allows electric current to flow. To build a circuit, you connect the positive terminal of a power supply (such as a battery) with its negative terminal through resistors and other components. If the positive and the negative side of the power supply are not connected, the current will not flow. If you want to control the flow of the electric current without connecting and disconnecting the power supply, you can use a button switch. When the button is pressed, the circuit is closed, and it will allow the current to flow. When the button is not pressed, the circuit is open, and no current will flow.

You should **NEVER** connect the positive and negative terminals of a power supply directly to each other without any components in between. This creates a short circuit and can cause dangerously high current, damaging the power supply and other components.

Breadboard

It is common to use a breadboard when building and testing electric circuits. Below, are two images of a breadboard.



The images are from: https://circuitfever.com/how-to-use-a-breadboard

If you look at the image above, you can see red and green lines illustrating the internal connections of a breadboard. The red lines show that each group of five holes is connected internally by metal strips. The same applies to the rows on the other side of the central gap. Any components you insert into these connected holes will allow the current to flow freely between them, just like a wire connecting those points.

The power supply is connected along the green lines: the **positive terminal** connects to the column marked +, and the **negative terminal** connects to the column marked -. All the holes in the + column are internally connected, as are all the holes in the - column. This means that applying voltage to one hole in the + column distributes the same voltage to all other holes in that column.

Resistors

If too much current flows through a circuit, components can overheat or become damaged! **Resistors** help prevent this by limiting the amount of current that can pass through. By adding resistors to a circuit, you can control the flow of electricity and protect sensitive components from excessive current. The higher the resistance, the less current can pass through.

The image illustrates the relationship between voltage, current and resistance. Voltage is the force that pushes electric current through a circuit. A battery is a common voltage source.

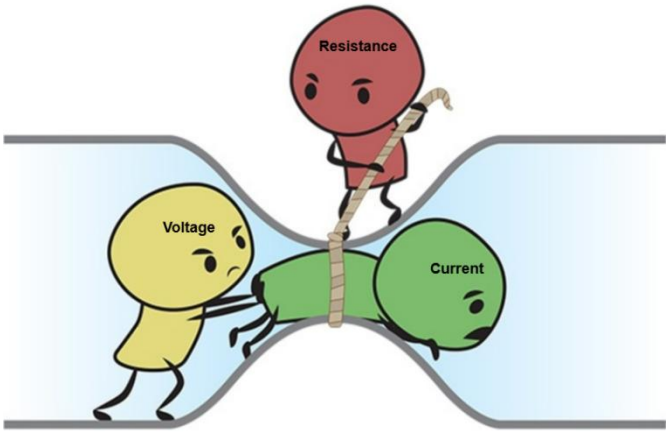
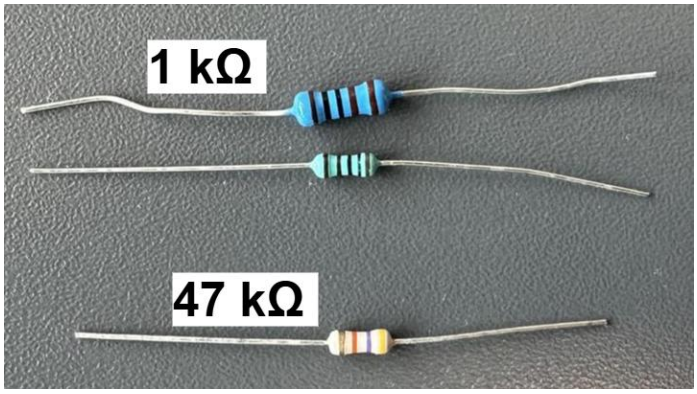


Image from: https://www.hpacmag.com/features/sizing-piping-controlled-device/attachment/ohms-law-cartoon/

Current is the flow of electric charge and resistance slows it down. In the image above, the red man illustrates how resistance affects the flow of the current. When he pulls hard on the rope, resistance is high, making it harder for the current to pass through. When he holds the rope loosely, resistance is low, allowing more current to pass through.

Here is a photo of the resistors you will be using today:



The blue (or turquoise) resistor is 1 kΩ, and the beige one is 47 kΩ. Ω is pronounced as Ohm and is the unit we use to measure the resistance.

Diode

An LED (Light Emitting Diode) is an electronic component that emits light when an electric current flows through it. It only allows current to flow in **one direction**. When connected correctly, the current flows through the LED and it lights up. However, if it is connected in the opposite direction, the current is blocked, and the LED remains off.

As shown in the image to the right, an LED has two legs. The long leg is the positive side, and the short leg is the negative side. This means that the long leg must be connected to the positive terminal of the circuit and the short leg to the negative terminal.

In addition, a resistor must be connected in series with the LED to limit the amount of current flowing through it. This is important because LEDs are sensitive components and cannot handle too much current, and allowing too much current could damage or destroy them.

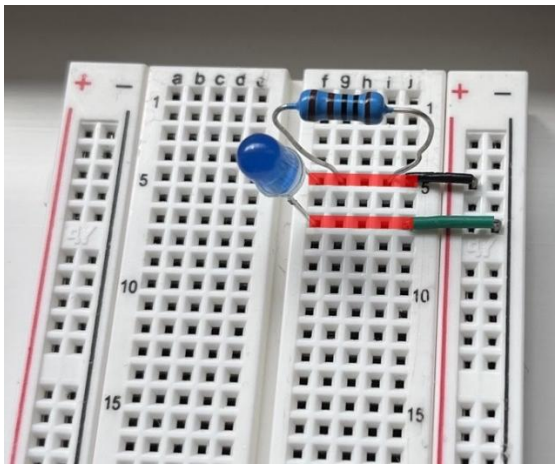
Current

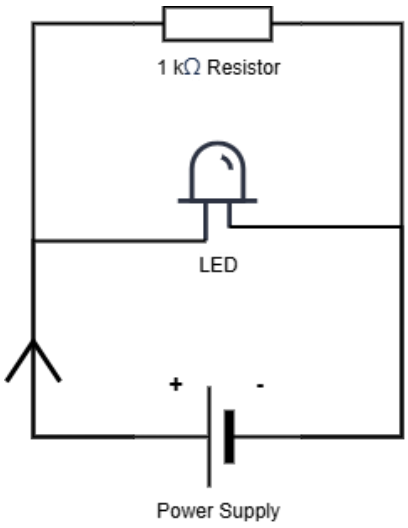
As previously mentioned, electric current flows freely in a closed circuit because there is a continuous connection between the positive and negative terminals of a power supply. In the following task, **a supervisor will connect the power supply to the circuit** you build. Your role, therefore, is to connect the components and wires in a way that creates a path for the current to flow from the + column to the - column.

Keep in mind that electric current naturally follows *the path of least resistance*. Therefore, when assembling the circuit, ensure that your components are arranged in a way that guides the current through the intended path, avoiding any shortcuts or low-resistance paths that could bypass important components.

Let's look at an example to understand how current flows in a circuit. In the first circuit, both the resistor and the LED is connected to the same row, which is also connected directly to the + column on the breadboard. Because the resistor and the LED share the same row, the current doesn't have to flow through the resistor. Instead, it takes the path of least resistance and bypasses the resistor entirely. As a result, too much current can flow through the LED, potentially damaging it.

1)

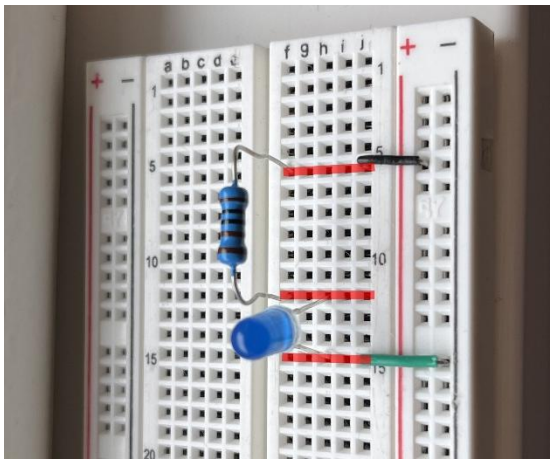


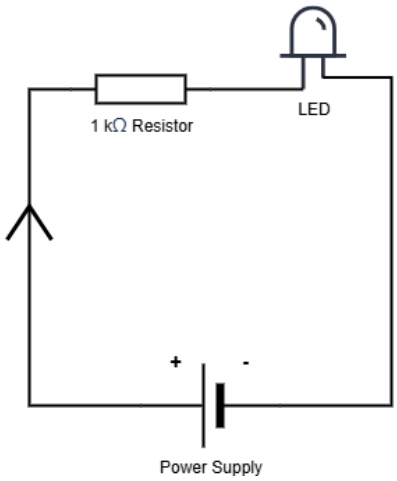


Look at the second circuit shown below. Here, one leg of the resistor is connected to the same row as one leg of the LED. The other leg of the resistor is connected to a different row that leads to the + column. This

arrangement forces the current to pass through the resistor before reaching the LED, ensuring the current is properly limited and protecting the components.

2)





This example showcases the difference between series and parallel circuits. The first circuit is a parallel circuit, as the current has more than one pathway to choose from. In the second circuit, the components are placed one after the other, so the current only has one pathway to follow. This setup makes it a series circuit. As you will see in the following tasks, both parallel and series circuit can be very useful, however, they are used in different applications.

!

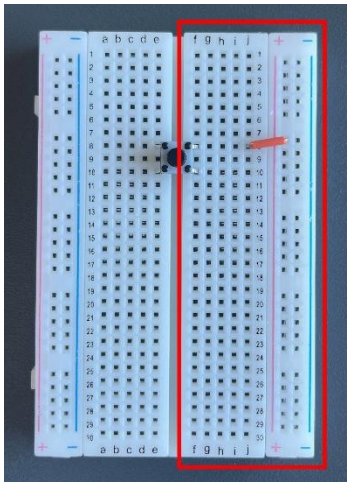
You will be using a Circuit Python Express (CPX) as a power supply for the circuit. The CPX gets its power from your computer via a USB cable. The first time the CPX is connected to the breadboard a supervisor will do it. After that, you must always unplug the USB cable from your computer before making any changes to the circuit or the breadboard. This is very important because unplugging the cable ensures that no current is flowing through the circuit while you're working on it. Since your body can conduct electricity, there's a risk of accidentally becoming part of the circuit if it's still powered — and that can be dangerous.

Task Description

In this task, you are going to build a circuit using a button, a resistor and an LED (any color). The current will flow from the positive (+) rail on your breadboard, through the button, the resistor and the LED (using a wire if

needed), and finally return to the negative (-) rail. Down below, you see a breadboard with a button already connected to the + column.

This is what you will be working on:



a) The first step is to complete the circuit using this setup. You will be using a 1 kΩ resistor (the blue/turquoise one), bend the legs to make the resistor fit where you want to place it, but be aware that the legs are a bit delicate. **All the components and wires should be connected on the right side of the breadboard**.

Once you have built the circuit, call over a supervisor and let them connect the power supply to your breadboard.

When the power is connected, the LED should light up when you press the button. This means the circuit is closed while the button is pressed. When the button is not pressed, the circuit is open, and no current flows through the LED.

b) Unplug the USB cable from your computer. Replace the 1 kΩ resistor with a 47 kΩ resistor (the beige one). Plug the USB cable back in.

?

What happens when you press the button? Can you explain why this happens?

c) Unplug the USB cable from your computer. Keep the 47 kΩ resistor in the circuit but turn the LED around so it's connected the wrong way. Plug the USB cable back in.

?

What happens when you press the button this time? Can you explain why this happens?

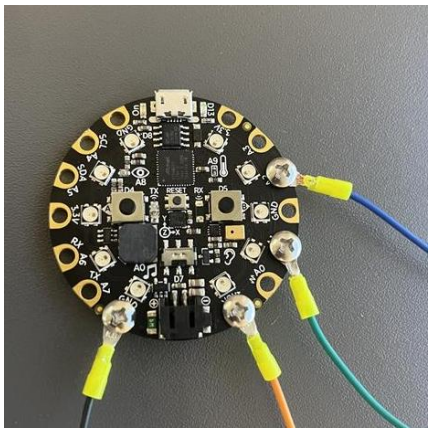
Task 2: Logic Gates

In this task, you will be building logic gates illustrating AND and OR gates using transistors.

You will be needing the following components:

- 1 breadboard
- 4 transistors
- 2 resistors: 1 kΩ
- 2 LED
- Wires

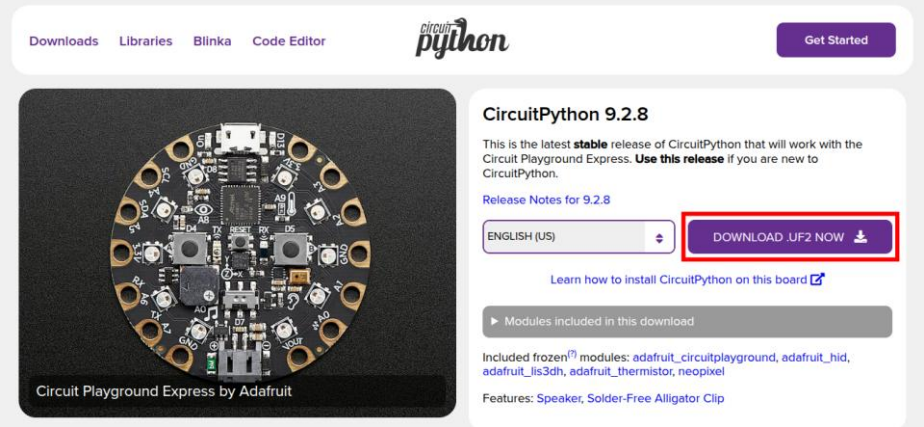
In the following tasks, you will be using the buttons on the CPX as input signals to control the transistors. Down below you see a photo of the CPX. It has two buttons labeled A and B, and a tiny button in the middle labeled as RESET. In the following tasks, you will be using buttons A (the green/yellow wire from A1) and B (the blue/purple wire from A2).



Before moving on to the tasks, check whether there is a file named "CIRCUITPY" in the file explorer when you connect the CPX to your computer using the USB cable. If it is there, you can skip ahead to the section called "**Task Description - OR gate**". If you find a file named "CPLAYBOOT", you need to download CircuitPython, which allows you to use the buttons on the CPX as inputs. To do so, you must follow these steps:

1. Go to this website: https://circuitpython.org/board/circuitplayground_express/

2. Click on "Download .UF2 NOW"



3. Drag the downloaded .uf2 file into the "CPLAYBOOT" folder.

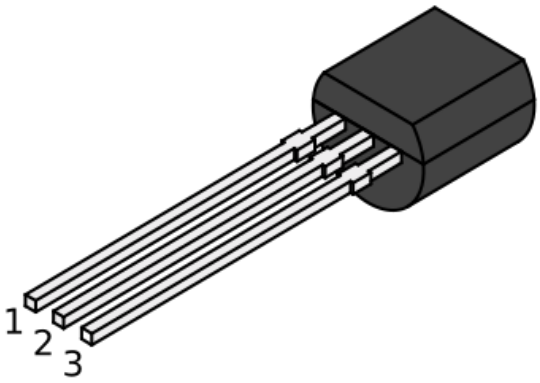
After a few moments, the "CPLAYBOOT" folder will disappear, and a new folder named "CIRCUITPY" should appear.
Inside the "CIRCUITPY" folder, there should be a file named code.py. This file contains the code that sets up the button inputs.
If you don't see this file or need help, please ask one of the supervisors for assistance.

Background info: Transistors and Logic Gates

What is a transistor?

A transistor is an electronic component that acts as a **switch** or an amplifier for electric current. Transistors are widely used in modern electronics, such as computers and mobile phones.

1. Collector
2. Base
3. Emitter



As shown in the image above, a transistor has three terminals: the **collector**, the **base** and the **emitter**. The base acts like a control input. When a small current is applied to the base, it allows a larger current to flow between the collector to the emitter. If no current is applied to the

base, no current flows between the collector and emitter. In this way, a transistor functions like an electronic switch.

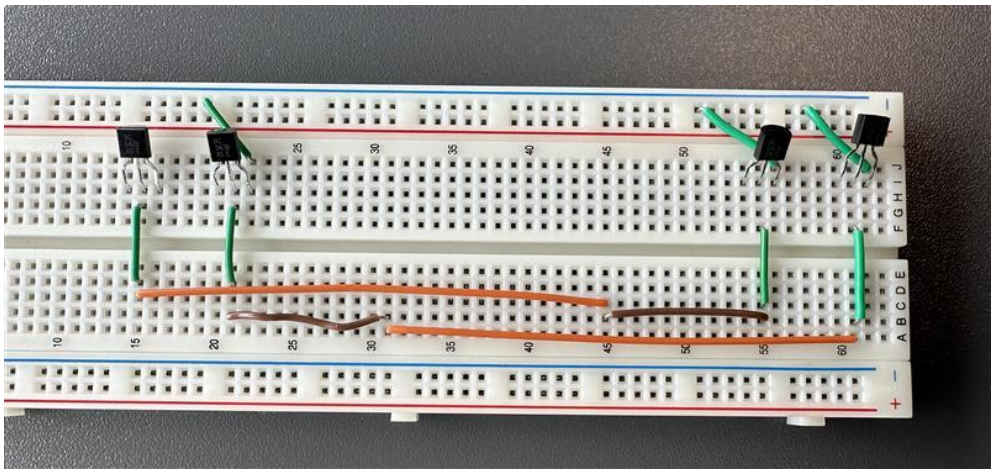
There are several types of transistors. The transistor you will be using today allows a **current to flow from the collector to the emitter** when the base is activated (this is called an NPN transistor).

Logic Gates

Logic gates are fundamental building blocks of digital electronics. A logic gate performs a basic logic operation on one or more inputs to produce an output. A logical operation is a rule that decides what the output should be based on the input value(s). The inputs and outputs can each have one of two values: 0 or 1. These values have different representations depending on the system.

Task description

In the following tasks, you will get familiar with OR and AND gates, and you will build logic gates using transistors. The breadboard will look like this when you begin:



OR gate

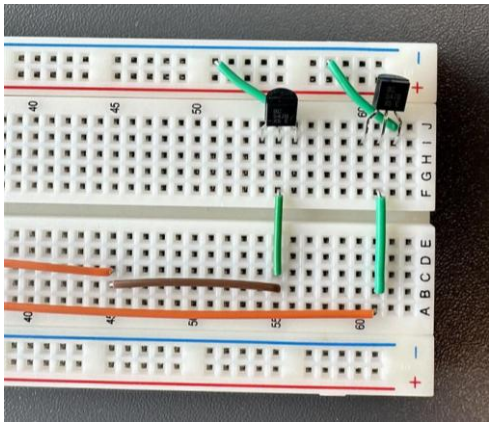
What is an OR gate?

The OR gate outputs 1 when **at least one** input is 1. This means that if you have two input signals, A and B, then either signal A **OR** signal B must be 1 for the output to be 1. You can think of the OR gate like an addition - if either input adds some value (1), the result is not 0: $1 + 0 = 1$

The table below shows all the possible output values the gate can produce for every combination of two input signals.

Input signal A	Input signal B	Output signal
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

Task: Build the circuit



You are going to build a circuit which illustrates how an OR gate works by using two transistors, a resistor and an LED. You are to use the circuit shown in the image on the right as a starting point (the circuit on the right on the breadboard in front of you).

As you already know, a transistor acts as a switch. When a current is sent to the base, the transistor closes the circuit, allowing current to flow from the collector to the emitter. In this task, you will use the buttons on the CPX (A and B) as input signals to the base of the transistors. We have connected these for you. When you push one of the buttons, current will flow through the wire from the CPX to the base of the connected transistor. Imagine that each of the buttons represents an input signal, which is equal to 1 when the button is pressed and 0 otherwise. In addition, the emitters of both transistors are connected to the negative rail on the breadboard. Your task is to **close the circuit** by connecting wires to the collectors, along with an LED, so that the circuit behaves like an OR gate. When connected correctly, pressing either one of the buttons should cause the LED to light up.

Once you have connected all the components, ask one of the supervisors to help you connect the buttons and the power from the CPX.

Think about what you have learned in the previous task:

- Which direction does the current flow?
- Which way should the LED be placed in the circuit?

An LED can't handle too much current. You need to limit the current flowing through it to avoid damage. Do you remember which component you need to use to limit the current?

Does your circuit behave like an OR gate? Have you built a series or a parallel circuit?

Hint (try it by yourself before reading)
To make the circuit behave like an OR gate, you need to connect a wire from the collector of each of the transistors to the same row on the breadboard. This way, the output from either transistor will reach the same row, allowing the current to flow through it when at least one of the transistors is activated. You can then connect the LED to this shared row to complete the circuit.

AND gate

What is an AND gate?

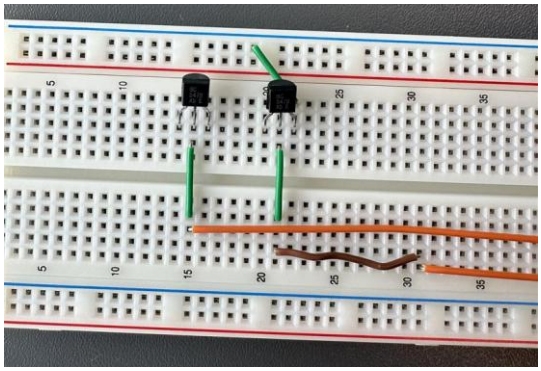
The AND gate only outputs 1 when **every input** is 1. For example, if you have two input signals, both signal A **AND** signal B must be 1 for the output to be 1. You could also think of the AND gate as multiplication, if any input is 0, the result of the multiplication will be 0: $1 \cdot 1 = 1$

The table shows all the possible output values the gate can produce for every combination of two input signals.

Input signal A	Input signal B	Output signal
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

Task Description: Build the circuit

You are going to build a circuit which illustrates how an AND gate works by using two transistors, a resistor and an LED. This time, you will start with the circuit in the image on the right (the circuit on the left on the breadboard in front of you).



Start by unplugging the USB cable from your computer.

This setup is similar to the one used in the previous task, but with a key difference: in this case, **only one of the emitters** is connected to the negative rail of the breadboard. Your task is to connect the collectors of both transistors so that the LED you connect to your circuit only lights up when both buttons are pressed. Thus, the collectors need to be connected differently than in the previous task.

Once you have built the circuit, you can plug the USB cable back into your computer.

?

Does the circuit function like an AND gate when you press the buttons? Have you built a series or a parallel circuit?

Hint 1

In the previous task, the transistors operated independently, since the LED would turn on when at **least one of the buttons** was pressed. However, in this task, the transistors are dependent of each other, since the LED will only light up when **both transistors are open**.

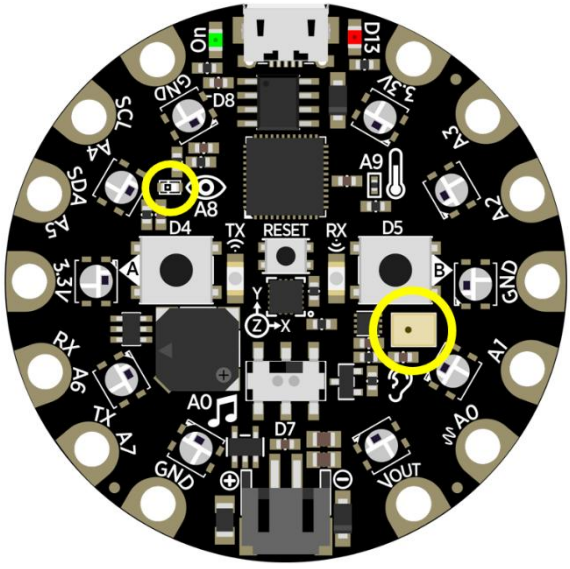
Hint 2

You need to connect the collector of one of the transistors to the emitter of the other transistor. One you have made this connection; you can wire the LED and the resistor just like you did in the previous task. However, this time, connect the LED, with its resistor, to the collector of only one of the transistors. This ensures that the LED will light up only when the current pass through both transistors.

Extra Task: Monitoring the Work Environment

So far, you have worked with two generic input signals, A and B, which were outputs from the buttons on the CPX. You will now explore how these signals, along with the AND or OR gates from the previous task, can be used in real-world situations.

Imagine that you are going to create a system that monitors the work environment in an office. A good working environment involves, among other things, **good lighting** and **acceptable noise levels**. In this task, you will measure both **light** and **sound**, and use the logic circuits you built in the previous task to trigger alerts if any of the levels become too high.



The CPX has built-in sensors for measuring sound and light. You can identify the light sensor by the small "eye" icon and the sound sensor by the "ear" icon on the CPX. We have also highlighted these sensors on the picture to the right.

Earlier you downloaded a program to the CPX. It outputs a '1' on the wire connected to A1 when button 'A' is pressed, and a '0' when it is released. Same with the wire connected to A2 but with button B. However, for this task, you need to create a *new program* with this behavior instead: A1 should output **1 when the sound level is high**, and A2 should output **1 when the light level is too low**. The tables below provide an overview of how this program should function:

Output signal A (the green or yellow wire):

	Sound level above 130	Sound level below 130
Output from A1	1 / high voltage	0 / low voltage

Output signal B (the blue or purple wire):

	Light level above 70	Light level below 70
Output from A2	0 / low voltage	1 / high voltage

Connecting and Pairing the CPX to Your PC

Go to the following website: https://makecode.adafruit.com/#editor

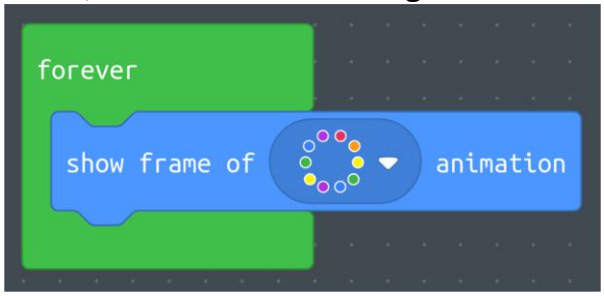
The first step is to connect and pair your CPX. This step ensures that you will be able to download and run your code directly from the website.

1. **Connecting the CPX:**
   - Plug the USB cable connected to your CPX into your computer.
   - Check your computer's file explorer to see what your CPX shows up as:
     - If it shows up as "CPLAYBOOT" - you can skip to the next section
     - If it shows up as "CIRCUITPY" - Double-click on the "RESET" button on your CPX and make sure a folder named "CPLAYBOOT" appears in your file explorer.
2. **Write test code:**

On the website, you will see a green "forever" block. This block defines actions that will repeat continuously while your program runs on the CPX. Currently, the "forever" block is empty, and nothing will happen when you download this code to your CPX. Go to the "LIGHT" category to the left of the code block and select a block that lights up one or more LEDs on your CPX. Then, drag and drop that block into the "forever" block.

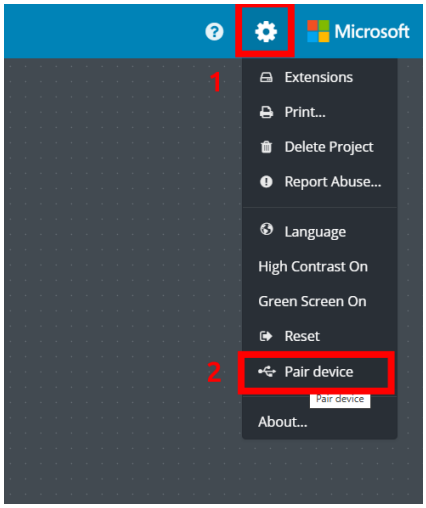
Tips!
To learn more about what a specific block does, simply hover your mouse over it.

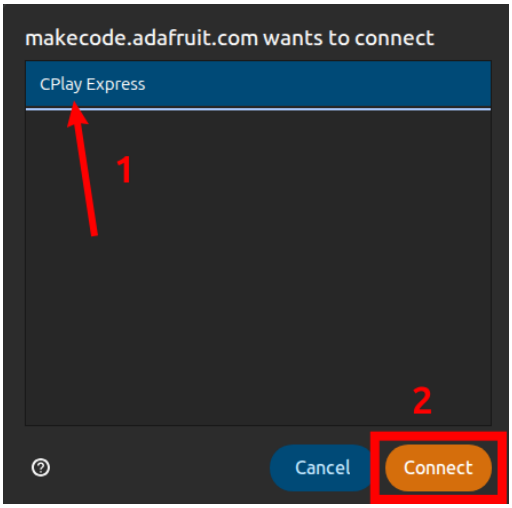
Make sure the block you choose fits properly inside the "forever" block, as shown in the image below:



3. **Pair your CPX:**

Click on the gear icon in the top right corner, then select "Pair device". Then, click on the device that pops up and click on "'Connect". (If pairing fails go to "**If you are unable to pair your device**")





4. **Test if pairing was successful:**

Now you can press on the pink button called "Download" to download the code onto your CPX. If the lights turn on in the same way shown in the simulation on the website, you have successfully downloaded the code to your CPX!

If you are unable to pair your device

Some computers may have difficulty detecting/pairing with the device. In that case, you will need to **manually** download and transfer the code. Here is how:

1. If there is no folder named "CPLAYBOOT" in your file explorer: Press "RESET" button on your CPX until the LED ring lights green.
2. On the website, click the pink "Download" button to download a file with the .uf2 extension.
3. Open your computer's file explorer (Finder on Mac, File Explorer on Windows or Files on Chromebook).
4. Locate the "CPLAYBOOT" drive that appears
5. Drag and drop the downloaded .uf2 file into the CPLAYBOOT folder

You will need to repeat these steps *each time* you want to upload a new program to your CPX.

If you don't have much experience with coding, it's a good idea to use **code blocks** when solving this task. This method provides a simple and intuitive way to create your code by dragging and dropping blocks. If you are familiar with **text coding** or would like to try it out, you can skip ahead to "**Task Description – Coding with Text**".

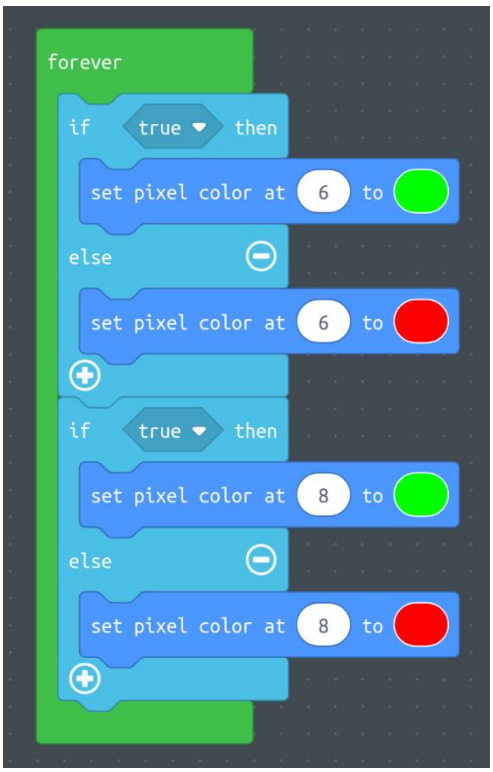
Task Description - Coding with Blocks

a) Add "if-else" code blocks

Now that you have successfully downloaded code to your CPX, the next step is to program the actual *logic* for the program. Start with 2 "if-else" blocks, found in the "LOGIC" category on the left.

Make sure to remove the block you added inside the "forever" block in the previous task. Also, remember to set the correct pixel numbers. Choose the LEDs that are closest to the signal pins connected to your breadboard. This will make is much easier to see and understand what your code is doing in the following tasks.

To understand how an "if-else" block works, start by adding blocks that set two of the pixels (LEDs) to either green or red. Your code should look like this:



Download your code and observe the LEDs.

Are the LEDs green or red?

An "if-else" block can perform one of two actions, depending on whether the condition inside the diamond-shaped input (after "if) is true or false. In this

case, the "if-else" block decides whether to set an LED to red or green. The first "if-else" block controls LED number 6 (next to A1), and the second "if-else" block controls LED number 8 (next to A2). As shown in the image above, both "if-else" blocks are currently set to "true", so both LEDs are green instead of red.

Now, change the condition of the first "if-else" block from "true" to "false", then download the updated code.

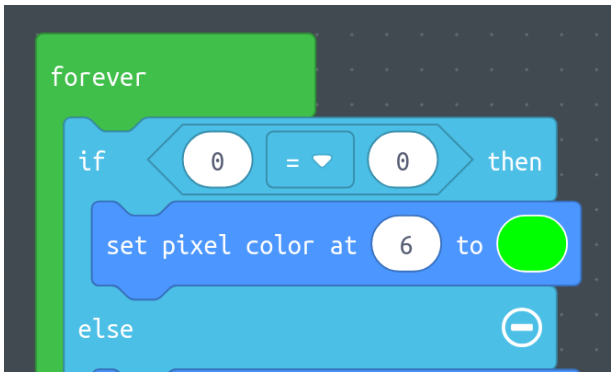
?

What colors are the LEDs now?

b) Add comparison code blocks

The diamond-shaped inputs can also contain values other than "true" or "false", as long as the value is a logical expression that evaluates to either true or false. For example, "0 < 3" is true, while "0 = 3" is false.

Find a "comparison" block from the "LOGIC" category and drag it into the first "if-else" block, like this



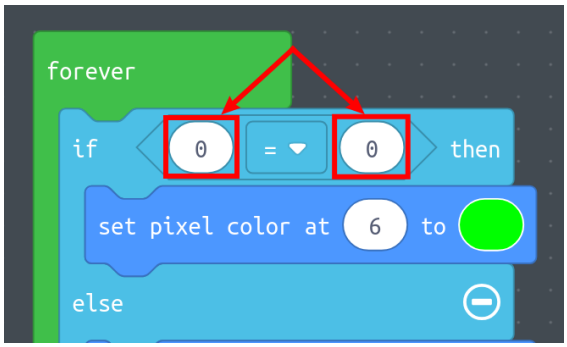
Try experimenting with different values and mathematical operators to better understand how it works.

?

What color is LED number 6 when the condition is "10 < 5"?

c) Measure sound and light levels

The inputs (the slightly long circles, obrounds) inside the comparison block (highlighted in the image below) can also represent the numerical values from sensors. In the "INPUT" category, find the sensor blocks that measure sound and light levels. These blocks have rounded corners as well, so they fit into the rounded inputs of the comparison block. The sensor values update continuously, always reflecting the most recent measurements.

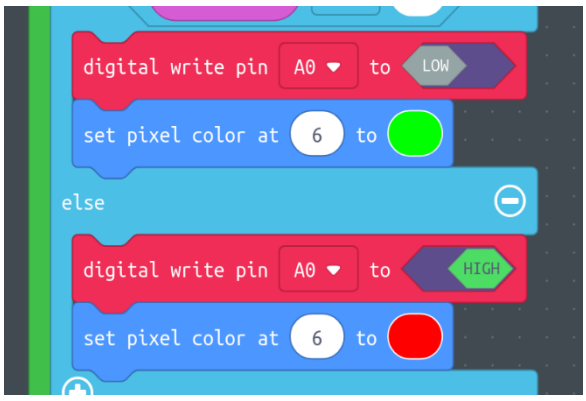


Place the block that measures the *sound level* as **one** of the inputs in the comparison block inside the first "if-else" block. Modify the comparison so that the LED turns **green** when the sound is **less than 130**, and **red** when the sound level is **above 130**. Download the code and test it by blowing on the sound sensor (creating a loud sound). You should see LED number 6 change from red to green.

Place the block that measures the *light level* as **one** of the inputs in the comparison block inside the second "if-else" block. Modify the comparison so that the LED turns **green** when the light level is **above 70**, and **red** when the light level is **less than 70**. Download the code and test it by placing your finger over the light sensor to block the light. You should see LED number 8 go from green to red. When you remove your finger, the LED should go back to green.

d) Supply power to the wires

The purpose of this task is to create a simple system that monitors the work environment. It should trigger an alert if the sound level is too high or the light level is too low. The alert will be shown using one of the LEDs on your breadboard. To make this work, you will need to send signals from your CPX to the breadboard using the connected wires. To do that, locate the block that writes a digital signal to one of the output pins on your CPX. You can find this block under "ADVANCED" → "PINS". It looks like this →



Place a digital write block inside each of the "if-else" blocks, just above the block that sets the LED color (shown in the picture above).

The outputs (or pins) are the small holes along the edge of the CPX, labeled from A0 to A7. We want the first "if-else" block (for the sound level) to write to the output pin connected to the green/yellow wire. Look at your CPX to identify which pin this wire is connected to. Then, update the block inside the first "if-else" block so that it writes to that specific pin

instead of the default A0. Set the pin to "low" when the sound level is acceptable (less than 130), and to "high" when the sound level is too high.

We want the second "if-else" block (for the light level) to write to the output pin connected to the blue/purple wire. Look at your CPX to identify which pin this wire is connected to. Then, update the block inside the second "if-else" block so that it writes to that specific pin instead of the default A0. Set the pin to "low" when the light level is acceptable (above 70) or "high" otherwise.

Verify that your code works correctly by performing the following tests:

1) Cover the light sensor. Only one of the LEDs should turn on
2) Blow on the sound sensor. Only one of the LEDs should turn on
3) Now, try covering the light sensor and blowing on the sound sensor at the same time. What happens?

?

To trigger an alert when at least one of the sensor values is outside the acceptable range, which logic gate would you use? Which gate would you use if you want to alert only when both values are outside the acceptable range?

Congratulations – you have completed all the tasks! 🤩🥳

If there is time, you can ask your supervisors for more tasks

Task Description - Coding with Text

To code with text, click the box that says "**{} Javascript**". If you've never coded with text before, there are a few things you should know. Skip the explanations if you're already familiar with this.

Function

A function is a block of code designed to perform a specific task. A function can take **inputs**, which are values passed into the function. It can also return **outputs**, which are values the function gives back after it runs. In this task, you will be using predefined functions. You can recognize a function by the parentheses () that appear after its name. If the parentheses are empty, the function does not require any input. Here are two examples of how functions look and how they are used:

```
light.setPixelColor(6, Colors.Red)
let temperature = input.temperature(TemperatureUnit.Celsius)
```

The first function sets the LED to a specific color. It takes two inputs: the first one is "6", which specifies which LED should be set, and the second one is "Colors.Red", which indicates the color to display. In other words, this function sets the LED number 6 to red.

The second function is connected to a temperature sensor on the CPX. It takes one input: the unit of measurement for the temperature, in this case, Celsius. The function returns a value - the current temperature. Since we want to use this value later, we store it in a *variable* called "temperature".

Variable

A variable is a container that stores information in a program. You give a variable a name, and then you can store a value in it, like a number or text. You can use that value later in your code.

If-else Statement

An "if-else" statement is used to add logic to your code. It begins with an 'if' condition that checks whether something is true or false:

```
if (lightLevel > 10) {
    music.playTone(440, 500)
}
```

What this does it is: the CPX plays music when the light level is above 10. An 'if' condition can be combined with an 'else' condition, but it is not required. The 'else' condition specifies what should happen if the 'if' condition is false.

```
if (lightLevel > 10) {
    music.playTone(440, 500)
} else {
    music.stopAllSounds()
}
```

The code above will also play a sound if the light level is above 10, but if the light level drops below or is equal to 10, it will stop playing. The first code snippet, which only used an "if", would have kept playing the sound even if the light level went back down.

a) Measure sound and light levels

1) Find out how to read the sound level, and store that value in the variable called "soundLevel". You can use the code below as a starting point:

```
forever(function () {
    let soundLevel = //*WRITE CODE HERE FOR TASK A*
    let lightLevel = //*WRITE CODE HERE FOR TASK A*

    if (soundLevel > 130) {
        light.setPixelColor(6, Colors.Red)
        //*WRITE CODE HERE FOR TASK B*
    } else {
        light.setPixelColor(6, Colors.Green)
        //*WRITE CODE HERE FOR TASK B*
    }


    if (lightLevel < 70) {
        light.setPixelColor(8, Colors.Red)
        //*WRITE CODE HERE FOR TASK B*
    } else {
        light.setPixelColor(8, Colors.Green)
        //*WRITE CODE HERE FOR TASK B*
    }
})
```

Code suggestion

2) Then find out how to read the light level and store the value in the variable called "lightLevel".
3) Download and run your code to test it. If you cover the light sensor (see earlier illustration to locate it), the pixel by A1 should change

from green to red. If you blow on the chip, the pixel by A2 wire should change from green to red.

b) Supply power to the wires

The purpose of this task is to create a simple system that monitors the work environment. It should trigger an alert if the sound level is too high or the light level is too low. The alert will be shown using one of the LEDs on your breadboard. To make this work, you will need to send signals from your CPX to the breadboard using the connected wires. To do that, you can use this function:

```
pins.AX.digitalWrite(true)
```

The input to this function can either be "true" or "false". If it's "true", the function sets the output to 1 / high voltage. If it's "false", the function sets the output to 0 / low voltage.

The outputs (or pins) are the small holes along the edge of the CPX, labeled from A0 to A7. You need to add four of these functions to your code, two in each "if-else" statement. Look at the code we have provided to see where they should be placed.

We want the first "if-else" statement (logic for the sound level) to write to the output pin connected to the green/yellow wire. Look at your CPX to identify which pin this wire is connected to. Then, update the first "if-else" statement so that it writes to that specific pin (change "AX" with the correct pin). Remember to change the input of the function that we provided so that the "if-else" statement sets the pin value to **1 / high voltage** when the sound level is unacceptable (more than 130), and to **0 / low voltage** otherwise.

We want the second "if-else" statement (logic for the light level) to write to the output pin connected to the blue/purple wire. Look at your CPX to identify which pin this wire is connected to. Then, update the "if-else" statement so that it writes to that specific pin (change "AX" with the correct pin). Remember to change the input of the function that we provided so that the "if-else" statement sets the pin value to **1 / high voltage** when the light level is unacceptable (under 70), and to **0 / low voltage** otherwise.

Verify that your code works correctly by performing the following tests:

1) Cover the light sensor. Only one of the LEDs should turn on
2) Blow on the sound sensor. Only one of the LEDs should turn on

3) Now, try covering the light sensor and blowing on the sound sensor at the same time. What happens?

?

To trigger an alert when at least one of the sensor values is outside the acceptable range, which logic gate would you use? Which gate would you use if you want to alert only when both values are outside the acceptable range?

Congratulations – you have completed all the tasks! 🤩 🥳

If there is time, you can ask your supervisors for more tasks