

# DATA VISUALIZATION WITH P5.JS

With the flash cards you can recreate  
your own first prototype step by step.

# THE SURFACE OF EDITOR.P5JS.ORG

On [editor.p5js.org](https://editor.p5js.org) you get to the editor in which you can start programming directly.



Tüftel  
Akademie



GOETHE  
INSTITUT

A preview of your program  
will be shown here when  
you click the play button

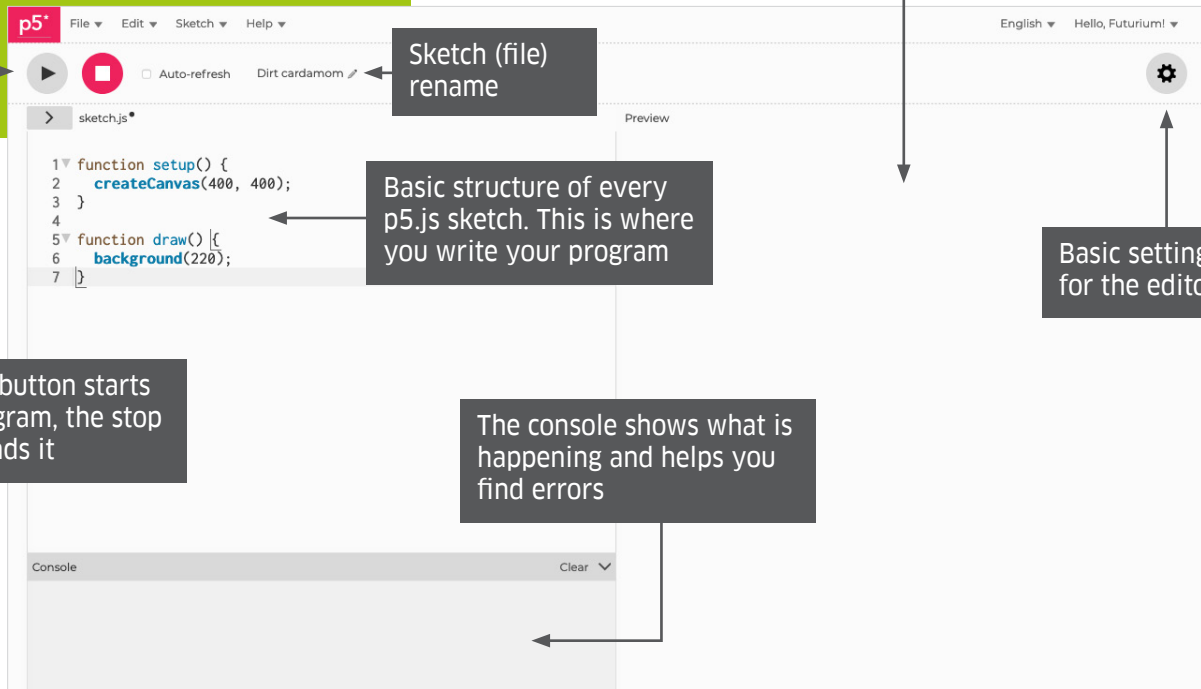
Sketch (file)  
rename

Basic structure of every  
p5.js sketch. This is where  
you write your program

Basic settings  
for the editor

The play button starts  
your program, the stop  
button ends it

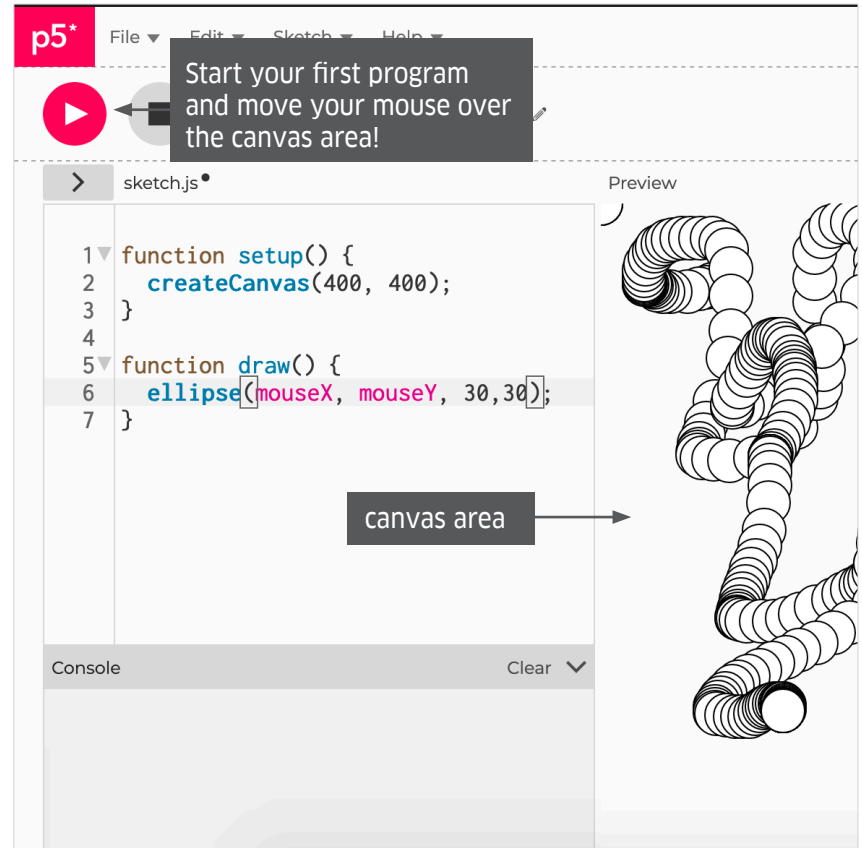
The console shows what is  
happening and helps you  
find errors



# HELLO ELLIPSE!

Following a small programming tradition, we also want to start with a „Hello World“ program, or more precisely with „Hello Ellipse“.

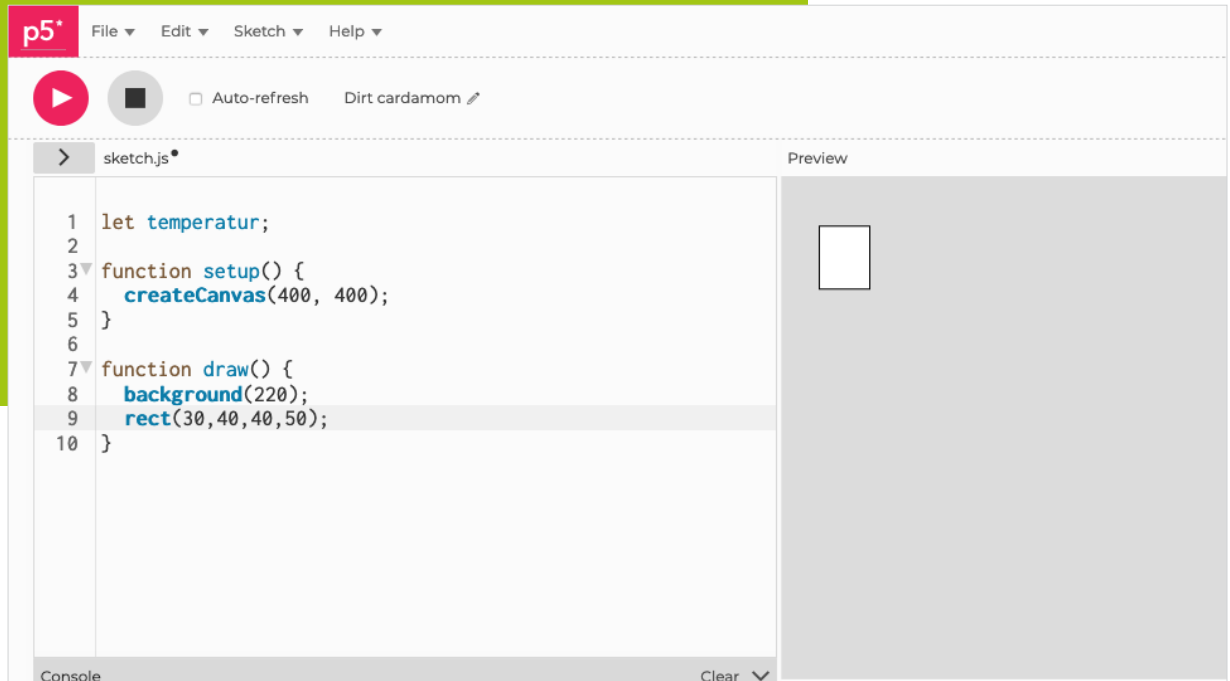
The idea behind the JavaScript library p5 is based on the open source programming language Processing. It's a kind of software sketchbook that makes creative design with programming easy for artists, designers and other people with ideas. You can use it to create graphic and interactive applications for the web directly in the browser! P5 is open source and was brought to the world by Lauren McCarthy.



# DRAW A RECTANGLE

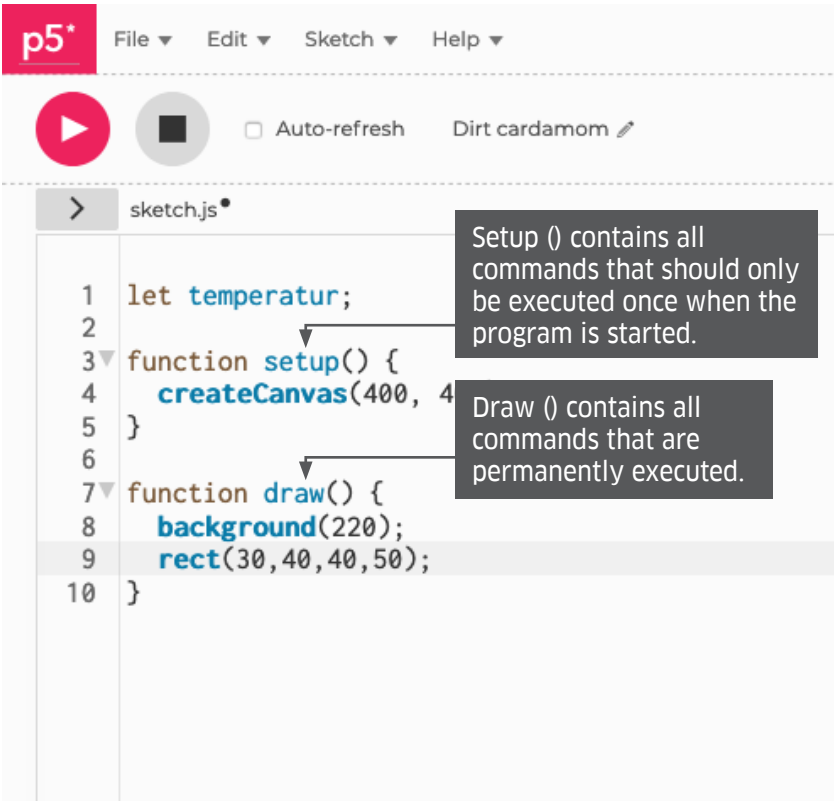
For the data visualization we want to color a rectangle depending on the temperature, for this we draw a rectangle on our drawing surface at the beginning.

## STEP 1



# SETUP & DRAW FUNCTION

p5.js provides various functions that are called automatically. The two most important are setup and draw. They can contain various commands that are processed step by step. The setup () function is executed once when the program starts, for example createCanvas () only needs to be called once. Then the function draw () repeats itself infinitely often.



The screenshot shows the p5.js web editor interface. At the top, there's a menu bar with 'File', 'Edit', 'Sketch', and 'Help'. Below the menu bar, there are controls for running the sketch: a red play button, a black square button, and a checkbox for 'Auto-refresh'. The main area displays a file named 'sketch.js' with the following code:

```
1 let temperatur;  
2  
3 function setup() {  
4   createCanvas(400, 400);  
5 }  
6  
7 function draw() {  
8   background(220);  
9   rect(30, 40, 40, 50);  
10 }
```

Two callout boxes provide additional information:

- A box pointing to the `setup()` function: "Setup () contains all commands that should only be executed once when the program is started."
- A box pointing to the `draw()` function: "Draw () contains all commands that are permanently executed."

# CANVAS, RECT & BACKGROUND

The screenshot shows the p5.js IDE interface. The top bar includes the p5.js logo, a play button, a square icon, and menu items: File, Edit, Sketch, and Help. Below the bar are controls for Auto-refresh and a text input field containing 'Dirt cardamom'. The main area is divided into a code editor on the left and a preview window on the right.

The code editor shows the following code in `sketch.js`:

```
1 let temperatur;  
2  
3 function setup() {  
4   createCanvas(400, 400);  
5 }  
6  
7 function draw() {  
8   background(220);  
9   rect(30, 40, 40, 50);  
10 }
```

Three callout boxes explain the code:

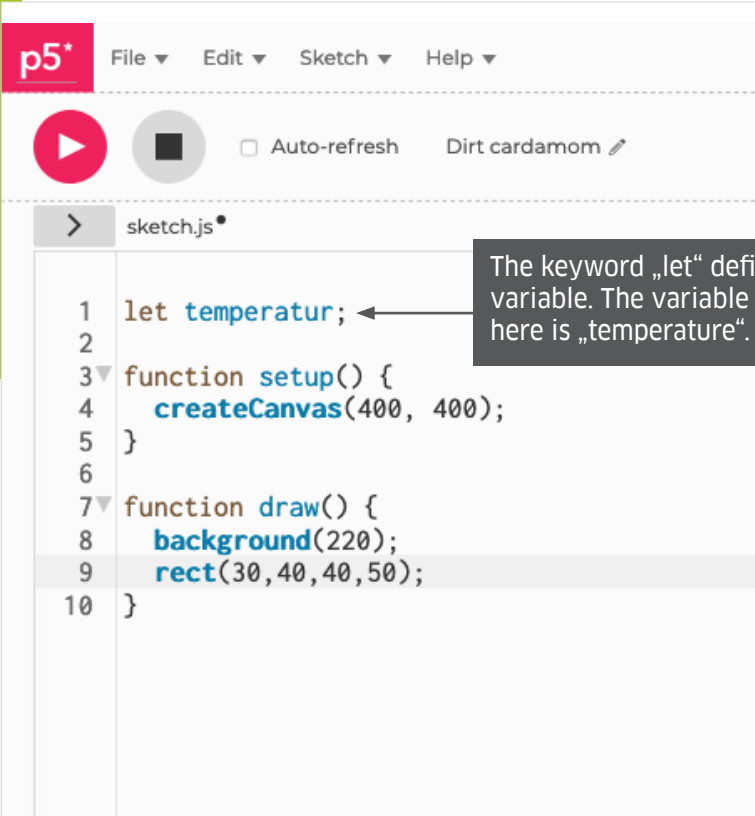
- Generates a drawing area for your program by specifying the pixel dimensions (x-axis, y-axis)**: Points to line 4, `createCanvas(400, 400);`.
- Gives the drawing area a color in RGB values (red, green, blue). If there is only one value in brackets, it applies to all 3 colors**: Points to line 8, `background(220);`.
- Draws at the point x: 30, y: 40 a rectangle with a width of 40 and a height of 50 pixels**: Points to line 9, `rect(30, 40, 40, 50);`.

The preview window on the right shows a white rectangle on a gray background. Dimensions are indicated: x: 30px (width) and y: 40px (height).

# VARIABLES

A variable can store information that can be used anywhere in the code.

You can visualize a variable like a box. The box is the variable, the content the value. The value can change at will. In our example we want to save a temperature value (e.g. 14) of a senseBox in the „temperature“ box.



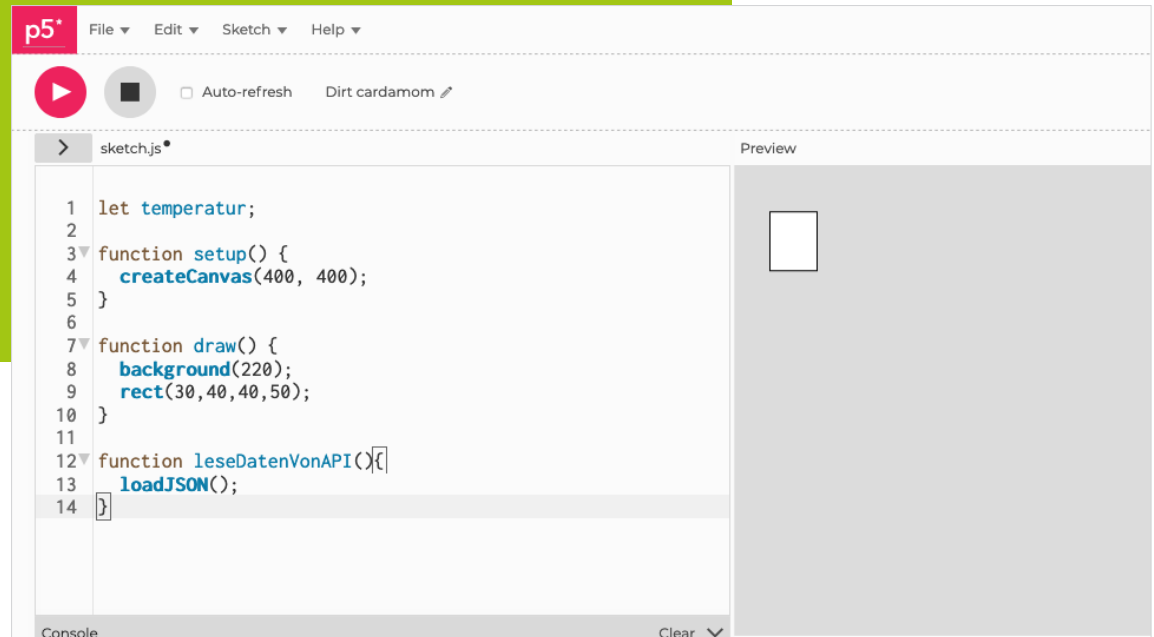
```
p5* File Edit Sketch Help
▶ Auto-refresh Dirt cardamom
> sketch.js
1 let temperatur;
2
3 function setup() {
4   createCanvas(400, 400);
5 }
6
7 function draw() {
8   background(220);
9   rect(30, 40, 40, 50);
10 }
```

The keyword „let“ defines a variable. The variable name here is „temperature“.

# LOAD DATA FROM A SENSEBOX

With the LoadJSON function we can load data from a specific senseBox. This function here calls another function called loadJSON. The "loadJSON" function is already provided by p5.js to call up data of the "JSON" file type. You can think of it as a text file with a certain structure in order to access specific elements in the file during programming.

## STEP 2





# OWN FUNCTIONS IN JAVASCRIPT

Functions are a block of instructions with a name. The function block is defined once. The JavaScript commands of the function can then be called several times in the program using the name. These functions can then be called and executed in `setup()` and `draw()`, for example.

```
3 function setup() {  
4   createCanvas(400, 400);  
5 }  
6  
7 function draw() {  
8   background(220);  
9   rect(30, 40, 40, 50);  
10 }  
11  
12 function leseDatenVonAPI() {  
13   loadJSON();  
14 }
```

The keyword „function“ creates and names a separate function in JavaScript. Everything in the curly brackets {...} belongs to this function. The function `readDataVonAPI()`, for example, contains the command to retrieve the data of a certain type via `loadJSON()`.

# OUTPUT DATA FROM A SENSEBOX TO THE CONSOLE

## STEP 3



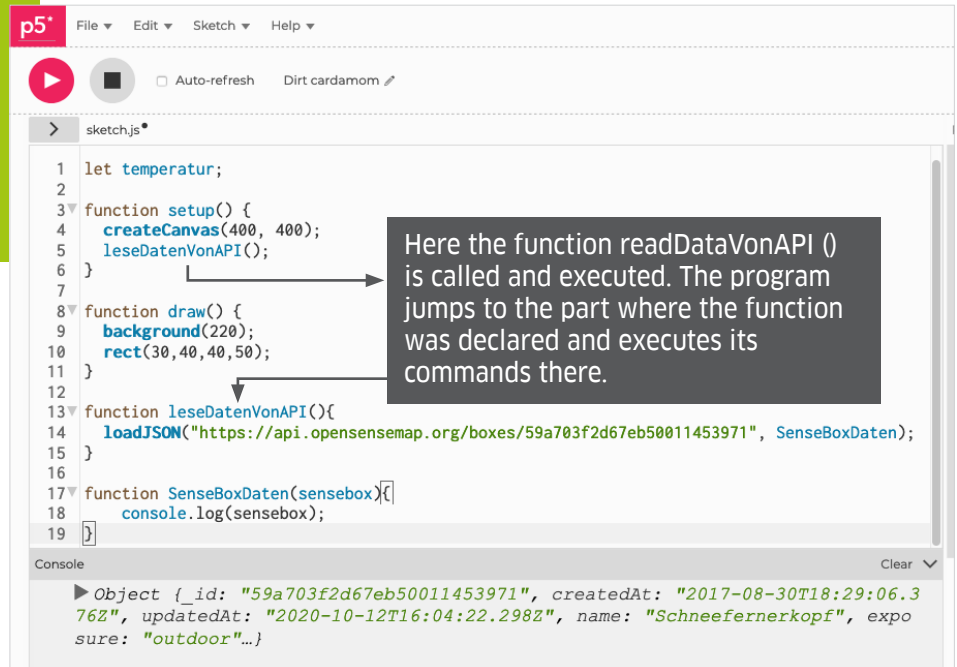
The screenshot shows a web-based code editor interface. At the top, there are controls: a red play button, a black square button, and a checkbox labeled "Auto-refresh" which is currently unchecked. To the right of the checkbox is the text "Dirt cardamom" with a small edit icon. Below these controls is a tab labeled "sketch.js". The main area contains the following JavaScript code:

```
1 let temperatur;  
2  
3 function setup() {  
4   createCanvas(400, 400);  
5   leseDatenVonAPI();  
6 }  
7  
8 function draw() {  
9   background(220);  
10  rect(30, 40, 40, 50);  
11 }  
12  
13 function leseDatenVonAPI(){  
14   loadJSON("https://api.opensensemap.org/boxes/59a703f2d67eb50011453971", SenseBoxDaten);  
15 }  
16  
17 function SenseBoxDaten(sensebox){  
18   console.log(sensebox);  
19 }
```

On the right side of the code editor is a "Preview" pane showing a small white rectangle on a gray background. Below the code editor is a "Console" pane. It contains a log entry: `Object {_id: "59a703f2d67eb50011453971", createdAt: "2017-08-30T18:29:06.376Z", updatedAt: "2020-10-12T16:04:22.298Z", name: "Schneefernerkopf", exposure: "outdoor"}...`. There is a "Clear" button with a dropdown arrow in the top right of the console.

# CALL YOUR OWN FUNCTION

The function `readDataVonAPI ()` was only declared before. It is only executed the first time when the function is called in `setup ()`. The program jumps to, so to speak where the function is written, executes the commands and then jumps back to `set-up ()`. This was the last command in `setup ()`, so `draw ()` is executed afterwards.



```
p5* File Edit Sketch Help
[Run] [Stop] [Auto-refresh] [Dirt cardamom]

> sketch.js
1 let temperatur;
2
3 function setup() {
4   createCanvas(400, 400);
5   leseDatenVonAPI();
6 }
7
8 function draw() {
9   background(220);
10  rect(30, 40, 40, 50);
11 }
12
13 function leseDatenVonAPI(){
14   loadJSON("https://api.openseemap.org/boxes/59a703f2d67eb50011453971", SenseBoxDaten);
15 }
16
17 function SenseBoxDaten(sensebox){
18   console.log(sensebox);
19 }
```

Here the function `readDataVonAPI ()` is called and executed. The program jumps to the part where the function was declared and executes its commands there.

Console

```
▶ Object {_id: "59a703f2d67eb50011453971", createdAt: "2017-08-30T18:29:06.376Z", updatedAt: "2020-10-12T16:04:22.298Z", name: "Schneefernerkopf", exposure: "outdoor"...
```

# LOADJSON (PATH,CALLBACK)

The loadJSON (path, callback) function expects at least two parameters. The path parameter is the address to the JSON file, the callback parameter expects a function that is called when loadJSON has loaded values. The received values are used by the callback function (in our case the SenseBoxDaten () function).

The screenshot shows a code editor with the following code:

```
8 function draw() {  
9   background(220);  
10  rect(30,40,40,50);  
11 }  
12  
13 function leseDatenVonAPI(){  
14   loadJSON("https://api.opensensemap.org/boxes/59a703f2d67eb50011453971", SenseBoxDaten);  
15 }  
16  
17 function SenseBoxDaten(sensebox){  
18   console.log(sensebox);  
19 }
```

Annotations with arrows pointing to the code:

- Parameter path** points to the URL in the loadJSON function call.
- The ID of the Sensebox** points to the ID `59a703f2d67eb50011453971` in the URL.
- Parameter callback** points to the `SenseBoxDaten` function name in the loadJSON function call.

A text box explains the callback function:

The callback function SenseBoxDaten is called as soon as the function loadJSON () has loaded data. The JSON data is in the senseBox variable.

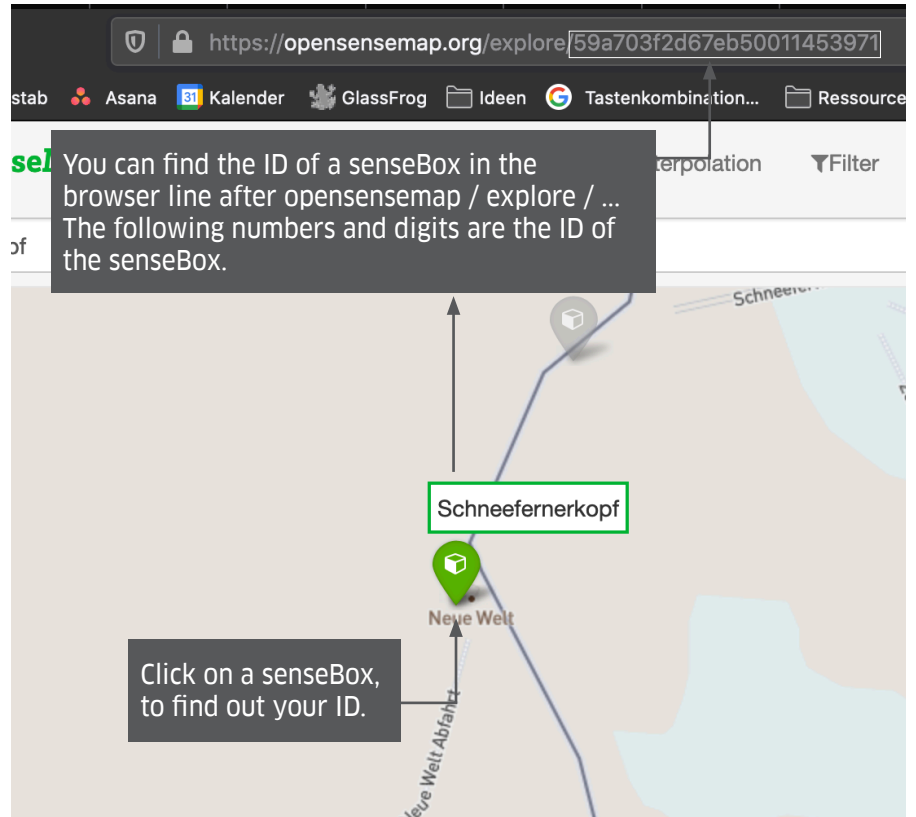
The console output shows the loaded JSON data:

```
Object {_id: "59a703f2d67eb50011453971", name: "Schneeföhnkopie", exposure: "outdoor", ...}
```

# ID OF A SENSEBOX

You can find many senseBoxes on the openSenseMap.

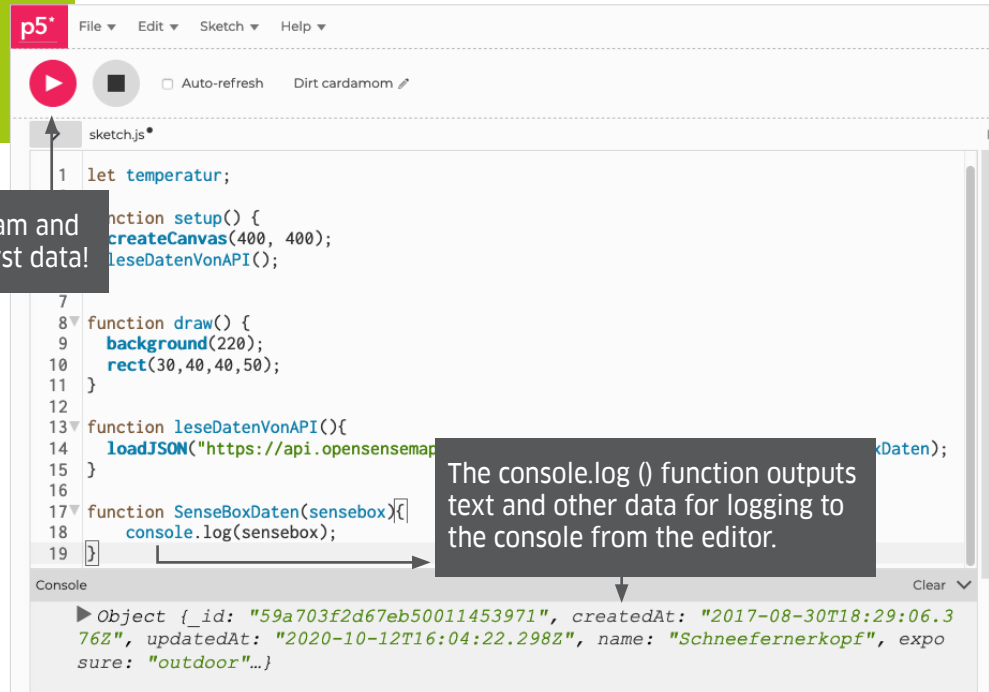
So that the different data of each senseBox can be sorted and evaluated, each senseBox needs a number or sequence of numbers that occurs only once or a combination of both. This combination of digits is then the ID (identifier) of a senseBox, which can thus be clearly identified.



# OUTPUT DATA FROM A SENSEBOX TO THE CONSOLE

So that we can see whether we have really received data from a SenseBox, we can output this to the console. Depending on your internet connection, this may take a while.

Start the program and receive your first data!



The screenshot shows the p5.js web editor. The top bar includes the p5.js logo, a play button, a stop button, and an 'Auto-refresh' checkbox. Below the toolbar is a file explorer showing 'sketch.js'. The main editor area contains the following code:

```
1 let temperatur;  
  
7  
8 function draw() {  
9   background(220);  
10  rect(30,40,40,50);  
11 }  
12  
13 function leseDatenVonAPI(){  
14   loadJSON("https://api.opensensemap.org/v1/stations/nearest?lat=50.0&lon=10.0&radius=1000000");  
15 }  
16  
17 function SenseBoxDaten(sensebox){  
18   console.log(sensebox);  
19 }
```

A callout box points to the play button with the text: "Start the program and receive your first data!". Another callout box points to the console output with the text: "The console.log () function outputs text and other data for logging to the console from the editor." The console at the bottom shows the following output:

```
► Object {_id: "59a703f2d67eb50011453971", createdAt: "2017-08-30T18:29:06.376Z", updatedAt: "2020-10-12T16:04:22.298Z", name: "Schneefernerkopf", exposure: "outdoor"...
```

# READ THE TEMPERATURE OF THE SENSEBOX

## STEP 4



Auto-refresh   Dirt cardamom

```
> sketch.js
3 function setup() {
4   createCanvas(400, 400);
5   leseDatenVonAPI();
6 }
7
8 function draw() {
9   background(220);
10  if(temperatur) {
11    rect(30,40,40,50);
12  }
13 }
14
15 function leseDatenVonAPI(){
16   loadJSON("https://api.opensensemap.org/boxes/59a703f2d67eb50011453971", SenseBoxDaten);
17 }
18
19 function SenseBoxDaten(sensebox){
20   console.log(sensebox);
21   temperatur = sensebox.sensors[0].lastMeasurement.value;
22 }
23
```

Console   Clear

```
Object {_id: "59a703f2d67eb50011453971", createdAt: "2017-06-30T16:29:06.376Z",
  updatedAt: "2020-10-12T16:46:42.052Z", name: "Schneefernerkopf", exposure: "outdoor"...
```

## TRANSFER JSON FILES IN A VARIABLE

So that we can work with the temperature value in the draw () function, we have to transfer the value to a variable that can also be used by other functions.

```
7  
8 function draw() {  
9   background(220);  
10  if(temperatur) {  
11    rect(30,40,40,50);  
12  }  
13 }  
14  
15 function leseDatenVonAPI(){  
16   loadJSON("https://api.opensea.io/api/v1/asset-owners/  
17 }  
18  
19 function SenseBoxDaten(sensebox){  
20   console.log(sensebox);  
21   temperatur = sensebox.sensors[0].lastMeasurement.value;  
22 }  
23
```

But how do we know that we are really querying the temperature sensor from the file? With senseBox. sensors [0]. we call up the first sensor of the senseBox. On the senseMap you can see the order of the sensors in your selected box. Alternatively, this information is also in the JSON file. Here we can see that the temperature sensor comes first. In computer science, counting always starts from zero, which is why the sensor is in the file at the [0] position.

Here a certain value of the JSON file is transferred to the "temperature" variable

Console  
Object {  
 updatedAt: "2020-10-12T10:40:42.032Z", name: "Schneefernerkopf", exposure: "outdoor" ... }



## FIND A SPECIFIC SENSOR USING THE SENSEMAP

If you are on senseMap select a senseBox, their sensors are shown in a list. The entries in the list can be known to count starting with 0. In this way you get the numbers of the individual sensors correlating to the structure in the JSON file.

The screenshot shows a map on the left with a location marker. Two text boxes with arrows point to specific sensor entries in the list on the right. The first box points to the 'Temperatur' entry, and the second box points to the 'rel. Luftfeuchte' entry.

**Map Labels:** Schneefernerkopf, Neue Welt Abfahrt

**Sensor List:**

- Temperatur**  
-1.02 °C vor 2
- rel. Luftfeuchte**  
71.82 % vor 2
- Luftdruck**  
1257.92 hPa vor 2
- Beleuchtungsstärke**  
5108.00 lx vor
- Beleuchtungsstärke 2**

**Informational Text:** Hier stehen weitere Informationen zu dieser Station. Der Besitzer dieser Station hat jedoch nichts hinterlassen.

## FIND A SPECIFIC SENSOR VIA THE JSON FILE

A JSON file can contain different data from a senseBox. So that we can see the data of a sensor, we navigate through the different tabs of the file. Click on the small triangles to see the lower levels.

p5\* File Edit Sketch Help

Auto-refresh Dirt cardamom

sketch.js

```
1 let temperatur;
```

Console Clear

```
▼Object {_id: "59a703f2d67eb50011453971", createdAt: "2017-08-30T18:29:06.376Z", updatedAt: "2020-10-12T16:04:22.298Z", name: "Schneefernerkopf", exposure: "outdoor"...}
  _id: "59a703f2d67eb50011453971"
  createdAt: "2017-08-30T18:29:06.376Z"
  updatedAt: "2020-10-12T16:04:22.298Z"
  name: "Schneefernerkopf"
  exposure: "outdoor"
  ▼sensors: Array[8]
    ▼0: Object
      title: "Temperatur"
      unit: "°C"
      sensorType: "SHT10"
      icon: "osem-thermometer"
      _id: "59a703f2d67eb50011453979"
      ▼lastMeasurement: Object
        value: "-8.04"
        createdAt: "2020-10-12T16:04:22.298Z"
    ►1: Object
    ►2: Object
    ►3: Object
    ►4: Object
```

Opens the lower ones levels

In the [0] object of „sensors“ the value of that Temperature sensor

The value for the temperature can be found under: sensebox.sensors [0].lastMeasurement. value and is just -8.04 °C

# IF-CONDITIONS

It is often necessary to execute certain lines of code only in certain cases. If statements are often used for this.

```
4   {
5   createCanvas(400, 400);
6   leseDatenVonAPI();
7   }
8   function draw() {
9   background(220);
10  if(temperatur) {
11    rect(30, 40, 40, 50);
12  }
13  }
14
15  function leseDatenVonAPI(){
16    loadJSON("https://api.openweathermap.org/boxes/59a703f2d67eb50011453971", SenseBoxDaten);
17  }
18
19  function SenseBoxDaten(sensebox){
20    console.log(sensebox);
21    temperatur = sensebox.sensors[0].lastMeasurement.value;
22  }
23
```

if (temperature) {...} checks whether the variable has been set. Because only when the variable "temperature" contains a value, the rectangle can be drawn with a corresponding color.

Console

Clear

```
Object {_id: "59a70312d67eb50011453971", createdAt: "2017-06-30T16:29:06.370Z",
updatedAt: "2020-10-12T16:46:42.052Z", name: "Schneefernerkopf", exposure: "outdoor"...
```


# COLOR THE RECTANGLE

## STEP 5

▶ Auto-refresh Dirt cardamom

```
sketch.js
3 function setup() {
4   createCanvas(400, 400);
5   leseDatenVonAPI();
6 }
7
8 function draw() {
9   background(220);
10  if(temperatur) {
11    let farbwert = map(temperatur, -20, 50, 0, 255);
12    fill(farbwert, 0, 128);
13    rect(30, 40, 40, 50);
14  }
15 }
16
17 function leseDatenVonAPI(){
18   loadJSON("https://api.openseemap.org/boxes/59a703f2d67eb50011453971",
19     SenseBoxDaten);
20 }
21 function SenseBoxDaten(sensebox){
22   console.log(sensebox);
23 }
```

Preview

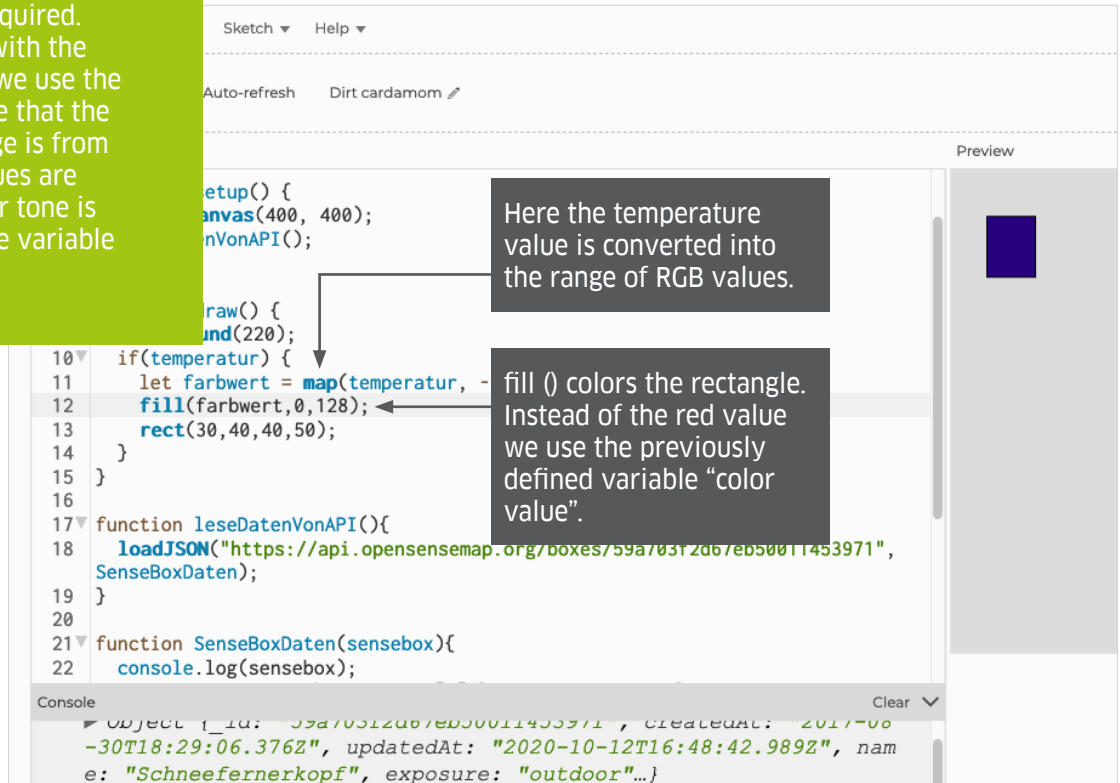


Console

```
Object {_id: "59a703f2d67eb50011453971", createdAt: "2017-06-30T18:29:06.376Z", updatedAt: "2020-10-12T16:48:42.989Z", name: "Schneefernerkopf", exposure: "outdoor"...
```

# VARIABLE FOR THE COLOR VALUE

To color the rectangle, RGB values in a range from 0 to 255 are required. So that the color changes with the temperature, do the math we use the `map()` function. We assume that the temperature value in Range is from -20 and 50 count. RGB values are between 0 and 255. A color tone is calculated and stored in the variable 'color value'.



The screenshot shows a p5.js IDE interface. The code in the editor is as follows:

```
10 setup() {  
11   canvas(400, 400);  
12   loadJSON('https://api.opensensemap.org/boxes/59a70312d67eb50011453971',  
13     SenseBoxDaten);  
14 }  
15 }  
16  
17 function leseDatenVonAPI(){  
18   loadJSON('https://api.opensensemap.org/boxes/59a70312d67eb50011453971',  
19     SenseBoxDaten);  
20 }  
21 function SenseBoxDaten(sensebox){  
22   console.log(sensebox);  
23 }
```

Annotations in the image:

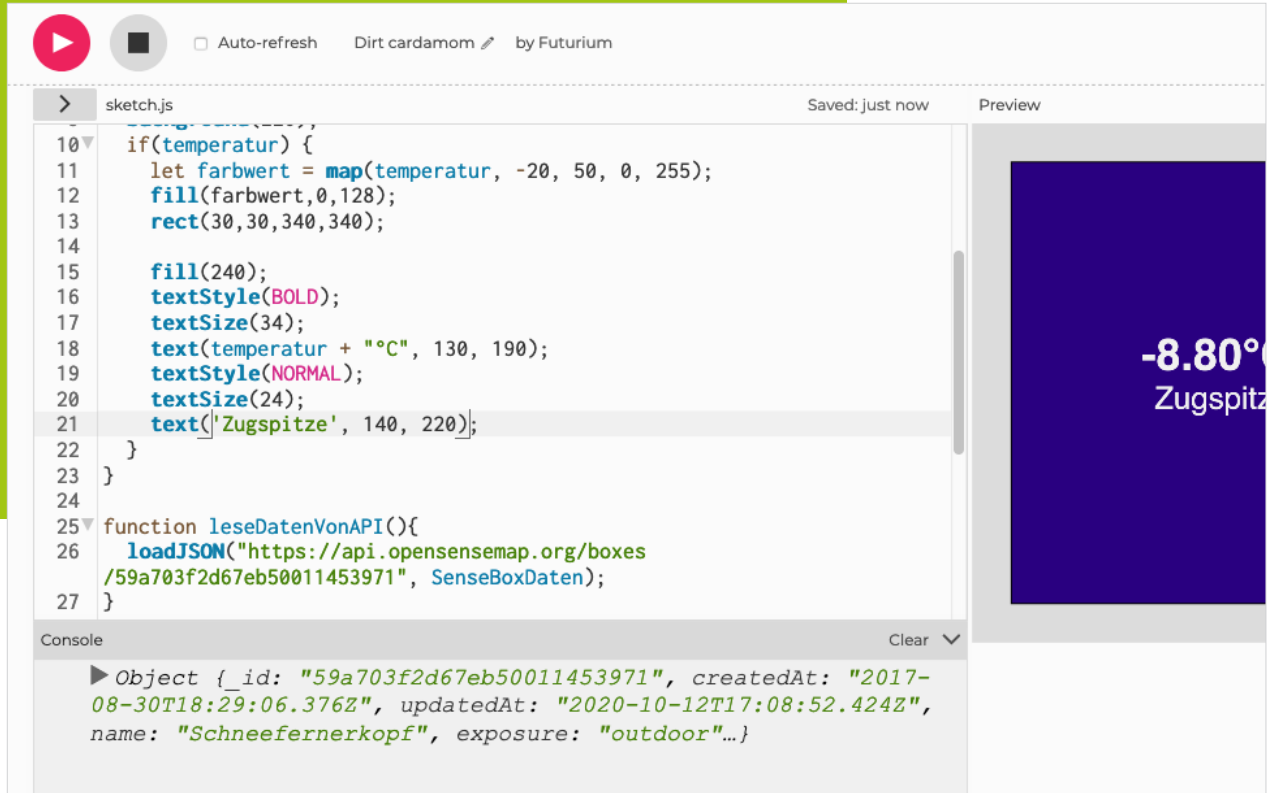
- An arrow points from the `map()` function call in line 12 to a text box: "Here the temperature value is converted into the range of RGB values."
- Another arrow points from the `fill()` function call in line 12 to a text box: "fill () colors the rectangle. Instead of the red value we use the previously defined variable 'color value'."

The right side of the IDE shows a preview window with a small blue rectangle. The bottom console shows the following log output:

```
Object {_id: "59a70312d67eb50011453971", createdAt: "2017-06-30T18:29:06.376Z", updatedAt: "2020-10-12T16:48:42.989Z", name: "Schneefernerkopf", exposure: "outdoor"...
```

## ADJUST THE RECTANGLE AND SET THE TEXT

### STEP 6



The screenshot shows a p5.js sketch editor interface. At the top, there's a toolbar with a play button, a square button, and a checkbox labeled 'Auto-refresh'. The text 'Dirt cardamom by Futurium' is visible. Below the toolbar, the sketch is named 'sketch.js' and is 'Saved: just now'. The main editor area contains the following code:

```
10 if(temperatur) {
11   let farbwert = map(temperatur, -20, 50, 0, 255);
12   fill(farbwert,0,128);
13   rect(30,30,340,340);
14
15   fill(240);
16   textStyle(BOLD);
17   textSize(34);
18   text(temperatur + "°C", 130, 190);
19   textStyle(NORMAL);
20   textSize(24);
21   text('Zugspitze', 140, 220);
22 }
23 }
24
25 function leseDatenVonAPI(){
26   loadJSON("https://api.opensensemap.org/boxes
27   /59a703f2d67eb50011453971", SenseBoxDaten);
28 }
```

On the right side, there's a 'Preview' window showing a dark blue rectangle with the text '-8.80°C' and 'Zugspitze' in white. Below the editor, there's a 'Console' window with the following output:

```
► Object { _id: "59a703f2d67eb50011453971", createdAt: "2017-08-30T18:29:06.376Z", updatedAt: "2020-10-12T17:08:52.424Z", name: "Schneefernerkopf", exposure: "outdoor"...}
```

# DISPLAY THE TEMPERATURE

The function `text` („Text“, `x`, `y`) writes text at the position (`x`, `y`) on your drawing surface. Variables can also be output as text. In our example we output the temperature (`temperature`) together (+) with the unit of measurement `° C` („° C“).

The screenshot shows a P5.js code editor with the following code:

```
13 }  
14  
15 fill(240);  
16 textStyle(BOLD);  
17 textSize(34);  
18 text(temperatur + "°C", 130, 190);  
19 textStyle(NORMAL);  
20 textSize(24);  
21 text('Zugspitze', 140, 220);  
22 }  
23 }  
24  
25 function leseDatenVonAPI(){  
26   loadJSON("https://api.opensensemap.org/boxes  
27     /59a703f2d67eb50011453971", SenseBoxDaten);  
28 }
```

Annotations on the code:

- Line 15: `fill(240);` → fill (240) colors the following text white
- Line 16: `textStyle(BOLD);` → textStyle (BOLD) makes the text bold
- Line 20: `textSize(24);` → textSize (32) sets the text size to 32

The console output shows:

```
► Object {_id: "59a703f2d67eb50011453971", createdAt: "2017-08-30T18:29:06.376Z", updatedAt: "2020-10-12T17:08:52.424Z", name: "Schneefernerkopf", exposure: "outdoor"...}
```

The preview window shows a dark blue rectangle with the text `-8.8 Zug` in white.

# CONGRATULATIONS

You wrote your first p5.js program.