Programming Online Experiments

Javascript, HTML and CSS
For Cognitive Sciences

Elyès Tabbane

Scan this QR code and take the experiment to the end.

Alternative Option. Go to:
https://tinyurl.com/stroop33

What do you think are the advantages of doing an online experiment over an offline one?
What about the disadvantages ?

✅ **Advantages**

- Can be done remotely.
- Can be done with (nearly) any device.
- Plug and Play : avoid issues with python environments / long package lists

⚠️ **Limits**

- No control on the environment around participants (*probe trials are necessary*)
- Have to program a code that adapt to different device. (*Define in the welcoming page which browser and device are supported*)

Course Objectives

- **Understand the basics of web programming** and how it applies to online experiments.
- **Explore how JS, HTML, and CSS** work together to create interactive experiments.
- **Set up a functional coding environment** to start building online experiments.
- **Discover useful tools and resources** for self-learning and further practice.

What This Course is NOT About: Mastering online experiment programming in just 2 sessions.

By the End of This Course, You Should:

✓ Understand the general steps in designing an online experiment.
✓ Know how and where to find answers to your coding questions.
✓ Have a realistic sense of the learning journey in programming.

Remember to :

- Have fun !
- Experiment with the concepts. Work at your own rhythm and prioritize understanding over speed.

Tools that we need to be working right now

- VS Code
- VS Code extension : *Live Server*
- Chrome Browser

✅ If you’re done

- Help your neighbours with setting up the tools.
- Take a pen and paper and write pseudo code structuring the *stroop experiment*.

Download the git repository

Instructions:

1. Open Terminal (Mac/Linux) or Git Bash (Windows).

2. Navigate to desired directory. For example:

```
cd Documents/my_courses/2025/programming/online_experiments
```

3. Clone the repository.

```
git clone https://github.com/ElyesT0/webProgramming_cogSci.git
```

Alternative Option. Go to:
https://tinyurl.com/progJS

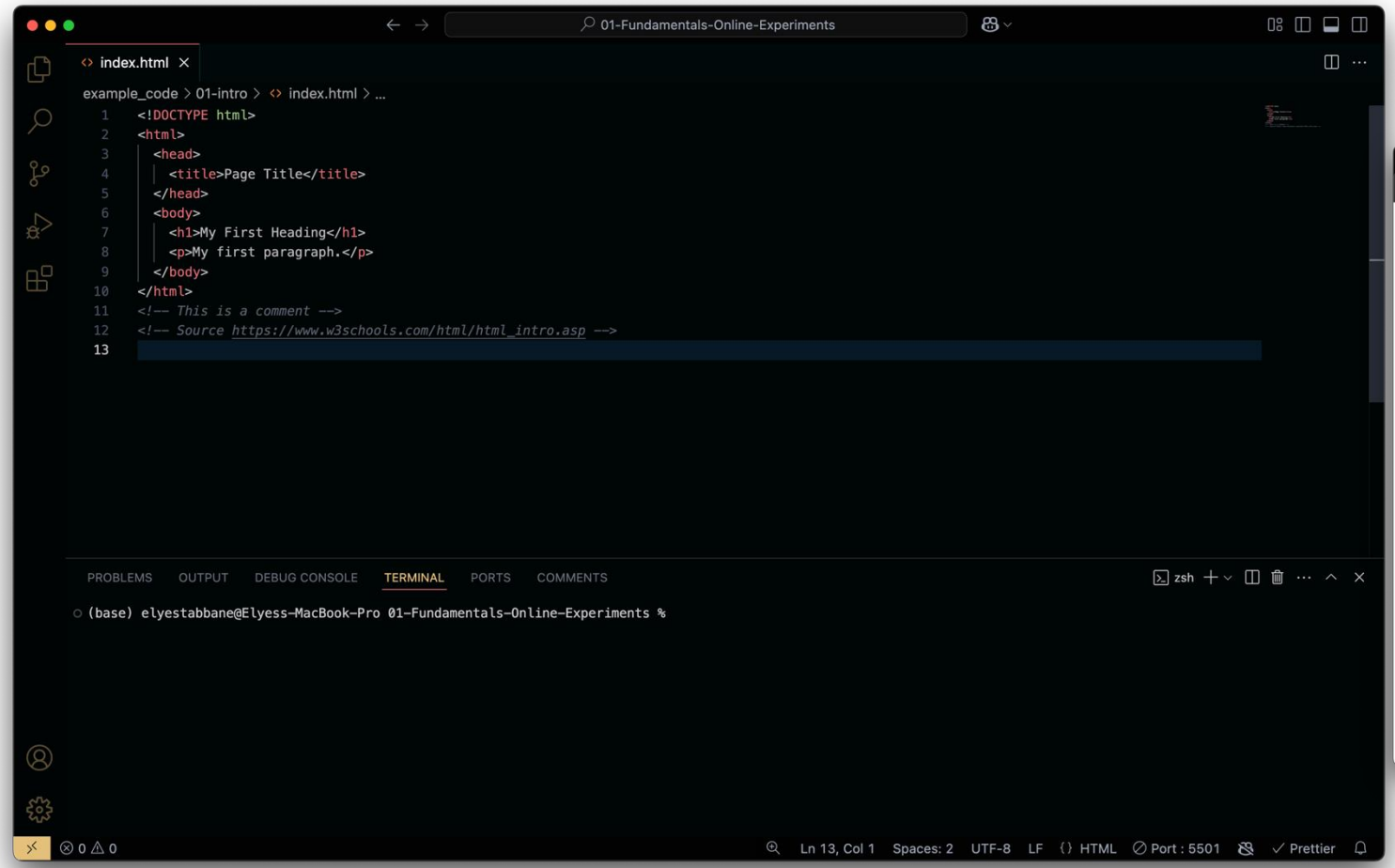
HTML (HyperText Markup Language) – The structure of a webpage (headings, paragraphs, buttons, etc.).

CSS (Cascading Style Sheets) – The design and appearance (colors, fonts, layouts).

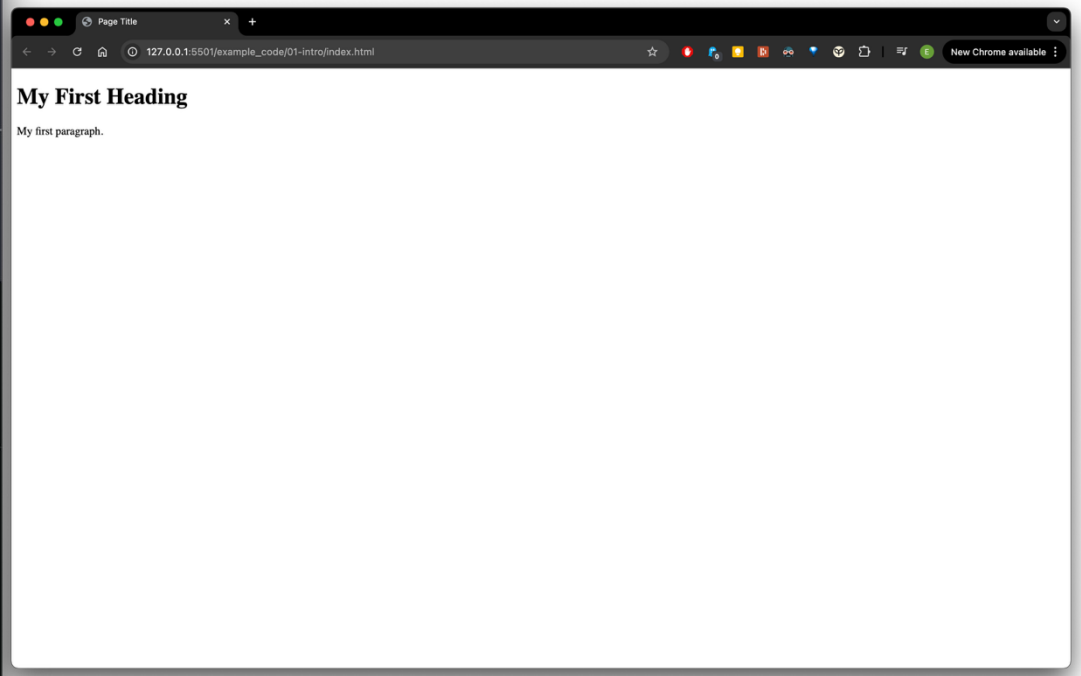
JavaScript – The interactivity and dynamic behavior (animations, pop-ups, forms, computations, data transmission).

This class focuses on JavaScript Programming. Specifically, on function oriented programming (as opposed to Object Oriented Programming).

Source : example_code/01-intro/index.html



Code



Browser rendering

index.html

example_code > 01-intro > index.html > ...

```
1   <!DOCTYPE html>
2   <html>
3     <head>
4       <title>Page Title</title>
5     </head>
6     <body>
7       <h1>My First Heading</h1>
8       <p>My first paragraph.</p>
9     </body>
10  </html>
11  <!-- This is a comment -->
12  <!-- Source https://www.w3schools.com/html/html_intro.asp -->
13
```

HTML uses **tags** to define a nested structure.
Tags must be **opened** (<tag>)
and **closed** (</tag>). Tags define **elements**.

<html>: Defines the entire document.
<head>: Contains metadata (not visible on the page).

- <title>: Sets the page title (shown in the browser tab).

<body>: Contains all visible content.

- <h1>: Defines the largest heading.
- <p>: Represents a paragraph of text.

<!-- text --> is the comment format.
On VS Code: Press ctrl + / or cmd + / to automatically make the line into a comment.

Instructions:

1. Open: *stroop_project/Starter/index.html*
2. Recognize the different tags opening and closing mentioned in the previous slide.
3. What new tags have appeared ?

Instructions:

1. Open: *stroop_project/Starter/index.html*
2. Recognize the different tags opening and closing mentioned in the previous slide.
3. What new tags have appeared ?

```
<div> </div>
```

- Base container used to group elements and apply styles.
- No default visual appearance.
- Used with CSS for layout and design.

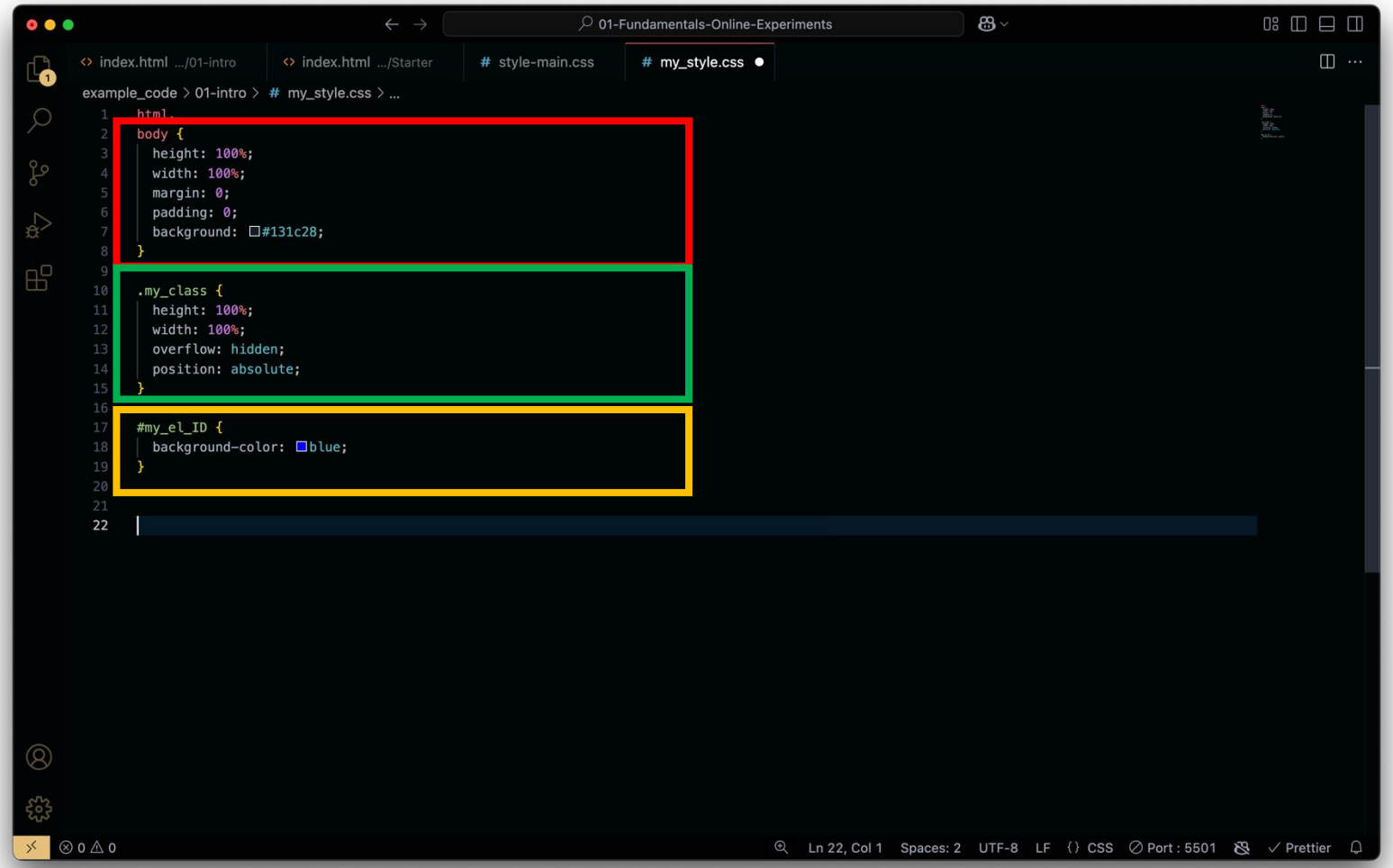
```
<script src="my_script.js">
</script>
```

- Used to **integrate Javascript** in the HTML page.
- Usually placed in the body. Order matters as HTML is executed top to bottom.
- Has a src attribute that points to .js file.

```
<head>
    <link href="styles.css"
    rel="stylesheet" />

 </head>
```

- Used to **integrate CSS** in the HTML page.
- Usually placed in the head.
- Has a href attribute that points to .css file.
- Has a rel attribute that specifies the file is used for styling.

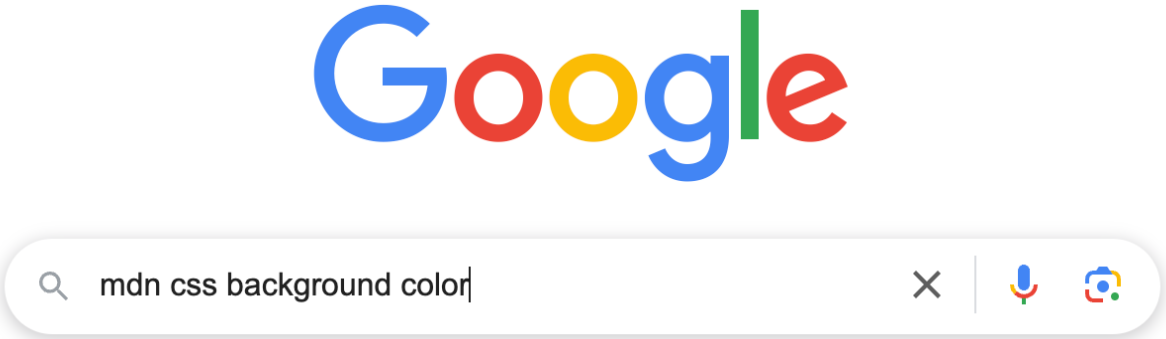


Your HTML file links to a **CSS file** to handle styling separately. The CSS file defines styles using **selectors** inside {}.

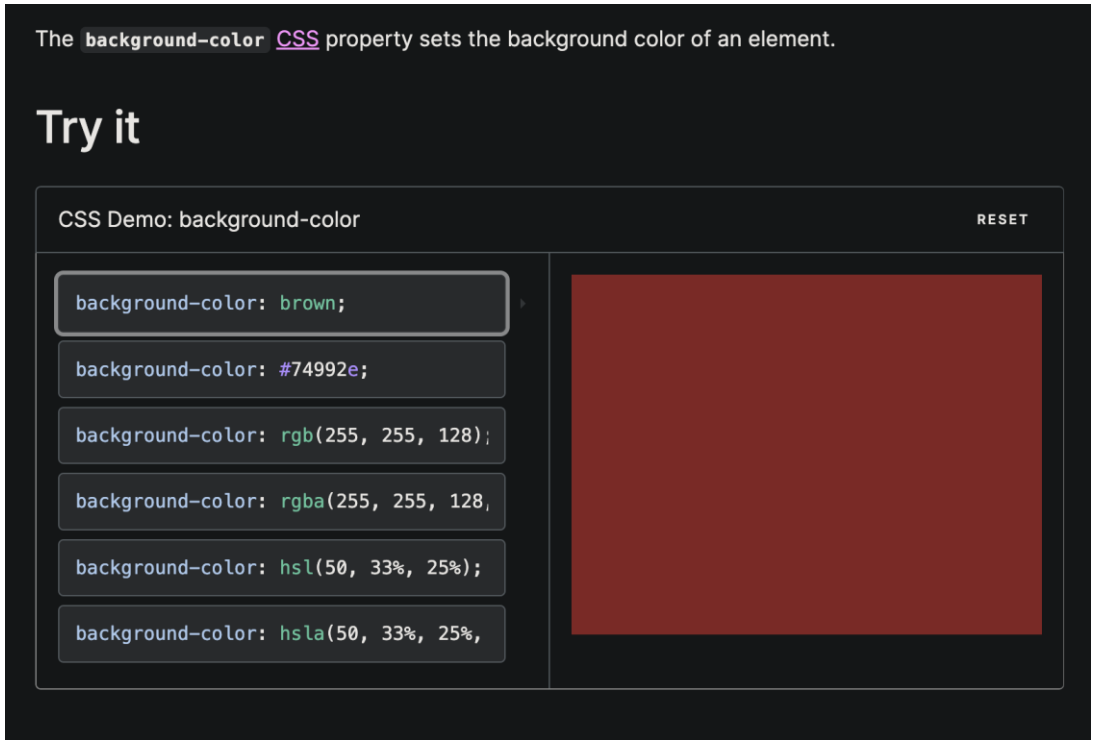
Example:

- body {} → Styles the whole page.
- .my_class {} → Styles all elements with this **class.**
- #my_element_ID {} → Styles a **specific element** with this ID.
- /* comment syntax */

Best resource to learn about CSS: https://developer.mozilla.org/en-US/docs/Web/CSS



MDN Website



- ✓ Gives the possible syntax and values.
- ✓ Provides a lot of examples.
- ✓ Interactive interface.

Instructions:

1. Open: course_practice_code*/01-intro/index.html*
2. Go in the <head> and link the css file *my_style.css*
3. Observe any changes?
4. Which part of the CSS was applied?

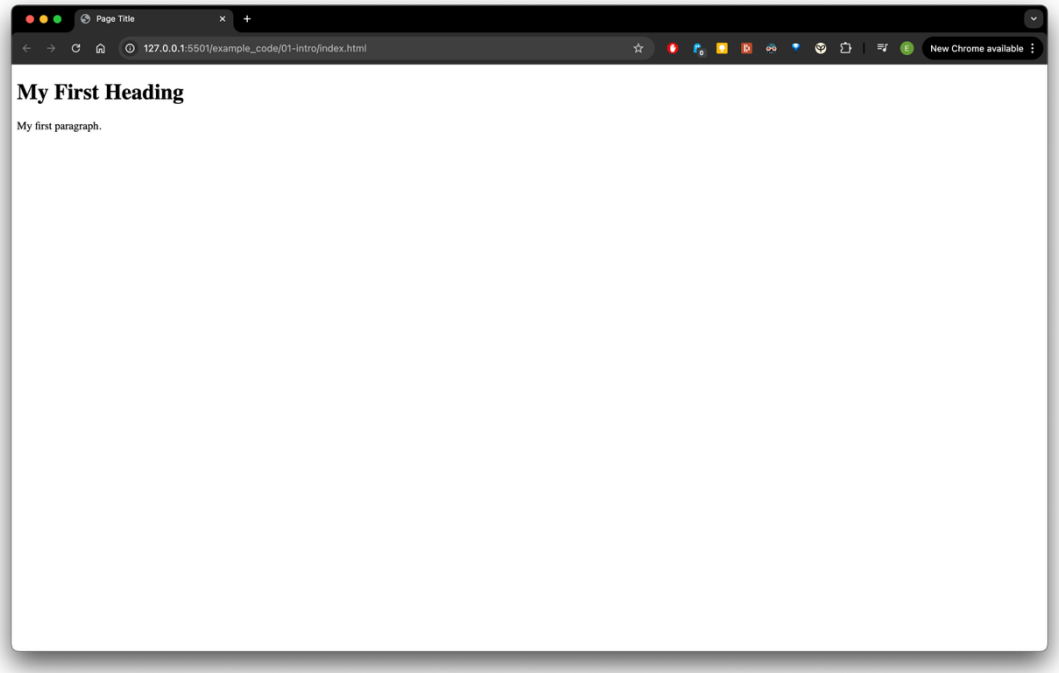
Instructions:

1. Open: course_practice_code */01-intro/index.html*
2. Go in the <head> and link the css file *my_style.css*
3. Observe any changes?
4. Which part of the CSS was applied?

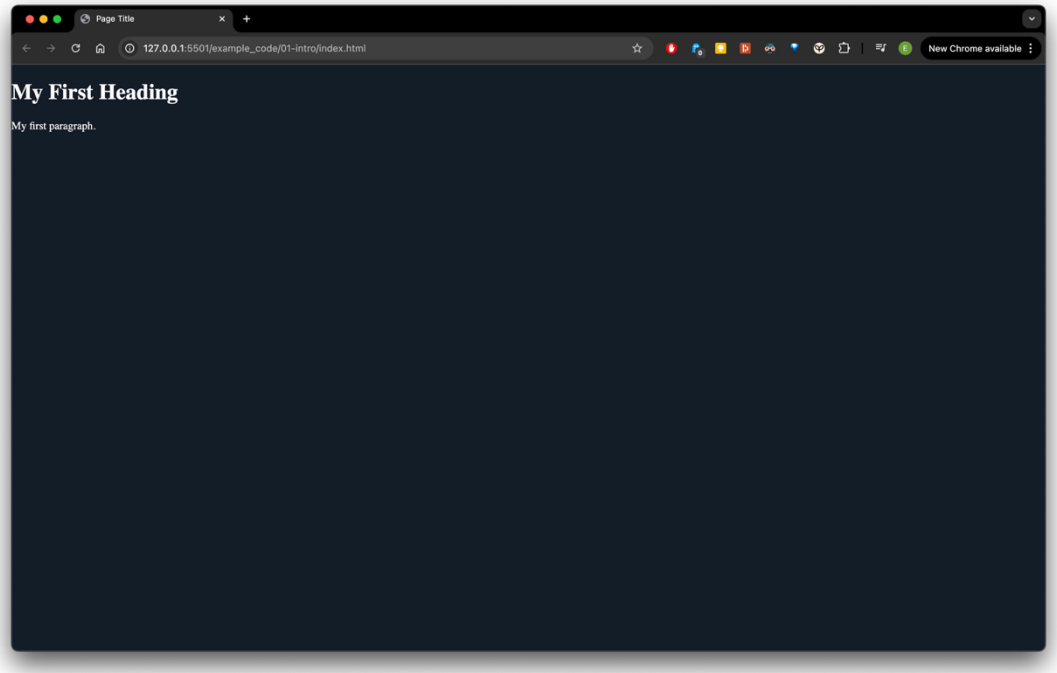
example_code > 01-intro > index.html > ...

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
    <link href="my_style.css" rel="stylesheet" />
  </head>
```

```css
body {
  height: 100%;
  width: 100%;
  margin: 0;
  padding: 0;
  background: #131c28;
  color: white;
}
```



Before linking CSS



After linking CSS

Styling content related to:

Text.

- Core properties. https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Text_styling/Fundamentals
- Fonts. https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Text_styling/Web_fonts

Box model (container dimensions). https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Box_model

For a comprehensive guide. https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics

02 – Basics of Javascript

Learn all the essential Javascript you need for Cognitive Science online experiments !

What is JavaScript?

- JavaScript (JS) is a programming language used to perform computations in the browser.
- It is the "alive" part of web programming, enabling dynamic interactions.
- We will primarily use JavaScript to program online experiments.

Syntax rules

- **Statements** (line of code) **must end with ;**
- Single line comments starts with //
- Multiple lines comments are contained inside /* this is a multiple line comment */

'use strict' mode

- Every .js file should start with 'use strict';
- 'use strict' statement activate a stricter set of rules on errors being raised. Without it, some errors will create bugs but never appear as errors in the console.
- For example:

```
x=10 // undeclared variable will raise with strict mode but not with standard mode
```

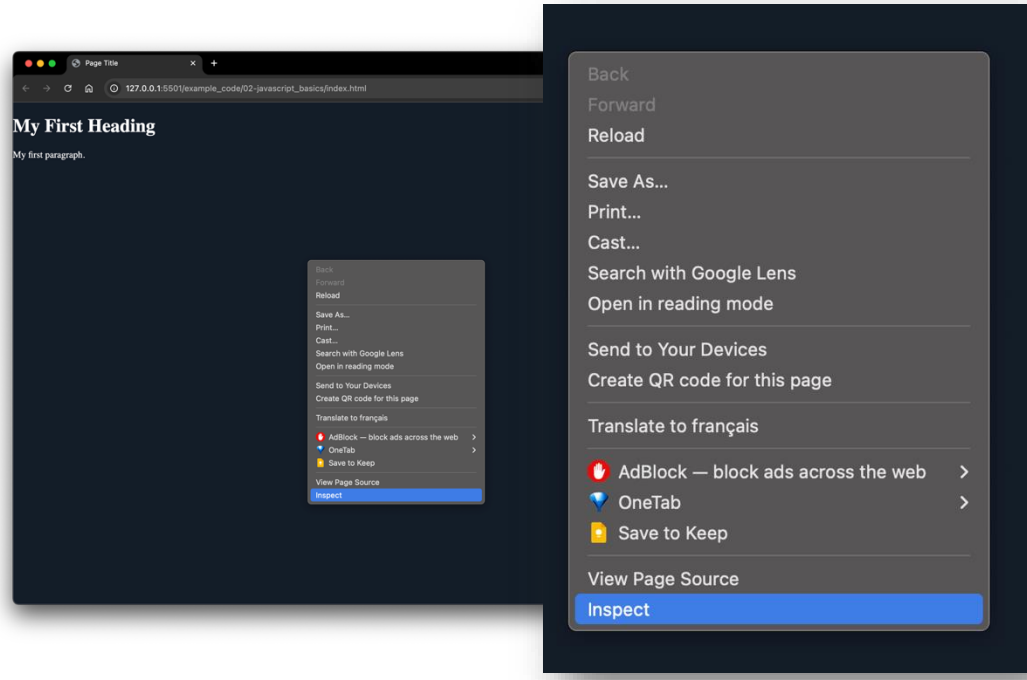
Linking .js file with HTML

This is done with the <script></script> tag. Although JS code can be written in between script tags, it is best to separate it in another file.
JavaScript files have the **.js suffix**.
To link a script.js file to HTML, in the body element:

```
<script src='script.js'></script>
```

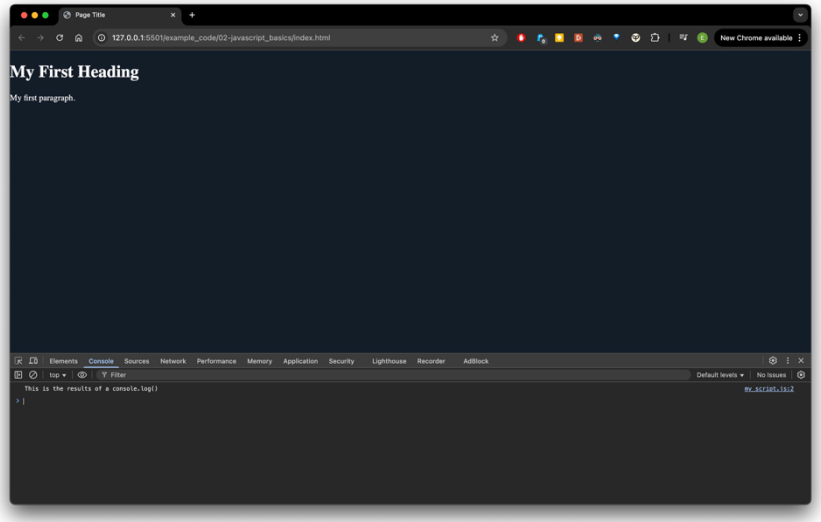
Base command for debugging– console.log('text')

- Displays the content of the function inside the browser's console.
- Content can be : object, variable, text, function's output, etc.
- Extremely useful for debugging.



Developer Tool

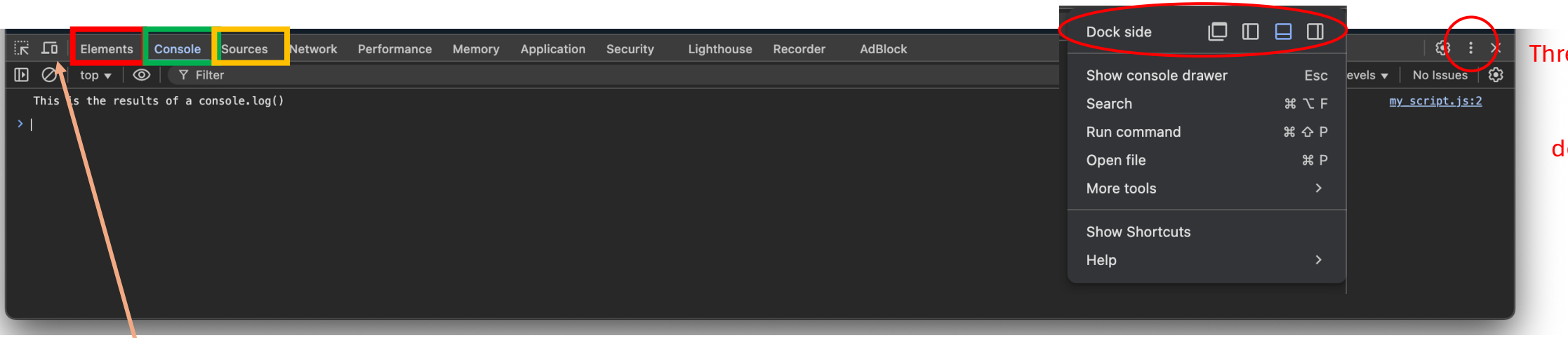
- Contains all important information and interface for development.
- Right click anywhere on the page and click 'inspect'.



Developer Tool Key Tabs:

- Elements → Elements and Styles present on the page
- Console → Error outputs, console messages, execution environment for javascript code.
- Sources → Files loaded by the browser to display the page (HTML, CSS, Javascript).

Reloading the page clears the console and memory of the browser.



Three dots> Dock side => Change the position of the development tool.

Chrome: This icon switches the view to **smartphone/tablet rendering**. Very useful for development.

Instructions:

1. Open: course_practice_code*/02-javascript_basics*
2. Respecting the JS guidelines (strict mode), write a script that outputs “Link established” in the console.
3. In the HTML: link the javascript file.
4. Start the live server > open the developer tool
5. Check that the console has your “Link established” message.
6. Change the message to “hello world” (or anything) in the javascript file.

For a variable to exist, it should be **declared** with one of three keywords : **let, var, const.**

Let

- Can be reassigned, i.e. value can change,
- Defined in a local scope, i.e. inside the most inner { } block
- Use when declaring a variable that will be updated (ex: counter).

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let

Var

- Can be reassigned, i.e. value can change,
- Defined in all scopes, i.e. will be defined out of function/if statement/for loop.
- Do not use unless very specific cases where you need higher scope.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var

Const

- Cannot be reassigned, it is a constant.
- Defined in local scope, i.e. inside the most inner { } block
- Use for values that won't change (ex: experimental parameters such as SOA, trial number, etc.)

It is good practice to use **let** for variables that change and **const** for constants. Avoid var.

Functions in Javascript can be defined in several ways.

Function Expression

- Can be called before it is defined, i.e. used higher up in the code from where it is defined.

```
function expression_func(text) {
  console.log(text);
}
```

Function Declaration

- Cannot be called before definition,
- Can be generated dynamically by another function / statement.

```
const declaration_func = function (text) {
  console.log(text);
};
```

Arrow Function

- Does not create its own arguments scope (ignore that for the moment),
- Useful for functions that take other functions as arguments (we'll see examples later).

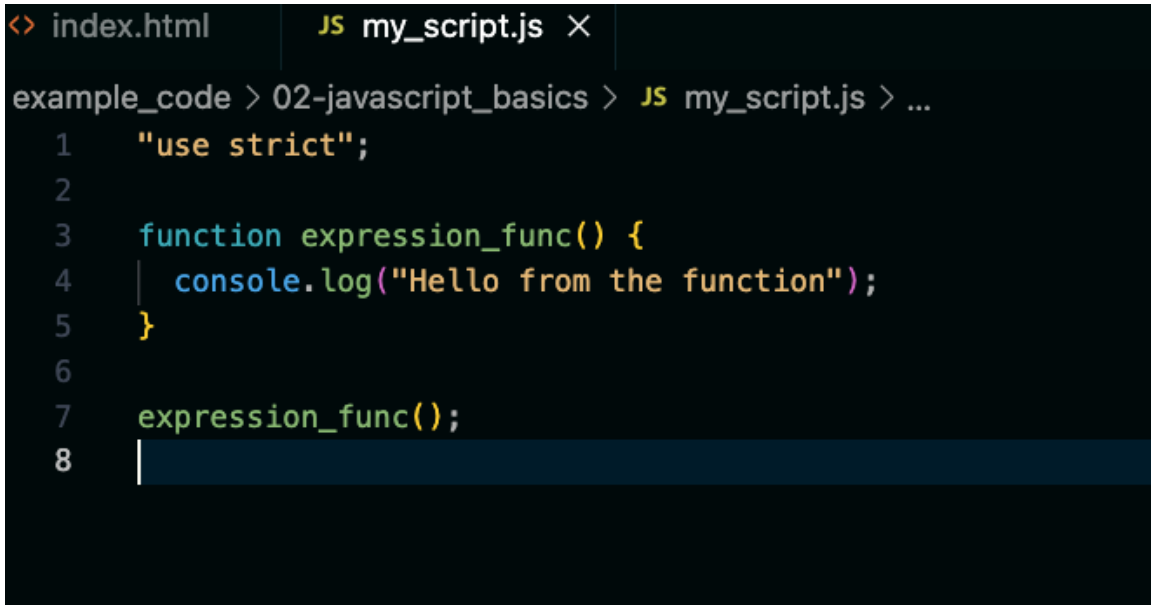
```
// Arrow function
(text) => {
  console.log(text);
};
```

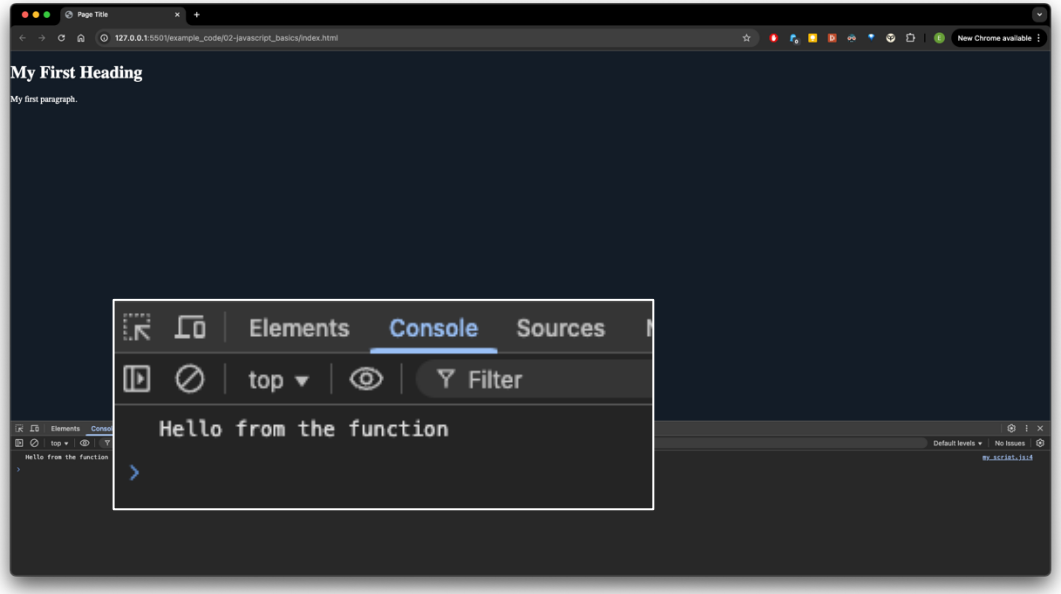
Instructions:

1. Open: course_practice_code*/02-javascript_basics/my_script.js*
2. Declare a function with function expression that logs into the console “Hello from the function”
3. Call your function in the browser’s console,
4. Call your function in your script and check its execution in the browser’s console.

Instructions:

1. Open: course_practice_code*/02-javascript_basics/my_script.js*
2. Declare a function with function expression that logs into the console “Hello from the function”
3. Call your function in the browser’s console,
4. Call your function in your script and check its execution in the browser’s console.





For examples see : *example_code/02-javascript_basics/final_code/my_script.js*

Arrays

Useful for grouping values together. For example, to hold stimuli (as array of arrays usually) or record participants responses.

Useful Methods:

```
const array_stimuli = ["red", "blue", "yellow"];
```

- my_array.**push**(item) : *adds item at the end of my_array (similar to append() in Python).*
- my_array.split(‘,’) : converts the array into a string by joining elements with whatever is entered as argument (must be string).
- my_array.length : *returns the number of elements in the array (similar to len(array) in Python).*

Objects

Useful for storing and structuring data. For example, participant’s data.

```
const my_object = {
  favorite_fruit: "apple",
  home_city: "Paris",
};
```

Object interface and useful methods:

- my_object.home_city : *allows accessing the value associated to the “home_city” key, i.e. “Paris”.*
- my_object[‘home_city’] : *allows accessing the value associated to the “home_city” key, i.e. “Paris”.*
- Object.keys(my_object): *returns an array with keys of the object as value, i.e. [’favorite_fruit’, ‘home_city’]*
- Object.values(my_object): *returns an array with values of the object as value, i.e. [‘apple’, ‘Paris’]*
- my_object.new_property = 5 : *Adds the “new_property” and 5, key-value pair to the object.*

Instructions:

1. Open: course_practice_code*/02-javascript_basics/my_script.js*

2. Create an object with the following key-value pairs:
   - Key : participant_id // Value: "insert_random_id”
   - Key: participant_starting_time // Value: Date.now()
   - Key: participant_response // Value: Empty array.

3. Log to the console the participant_id

4. Add a new key-value pair to the object : participant_RT with empty array as value.

5. Add to the participant_response property, one after the other, the values: 1, 2, 3.

6. Log to the console the length of the participant_response array.

`Date.now()` : returns the time at the ms resolution.

Tip: Use a holder for last click and subtract the value to a new execution of Date.now() to get response times in miliseconds.

Instructions:

1. Open: course_practice_code*/02-javascript_basics/my_script.js*
2. Create an object with the following key-value pairs:
   - Key : participant_id // Value: "insert_random_id"
   - Key: participant_starting_time // Value: Date.now()
   - Key: participant_response // Value: Empty array.
3. Log to the console the participant_id
4. Add a new key-value pair to the object : participant_RT with empty array as value.
5. Add to the participant_response property, one after the other, the values: 1, 2, 3.
6. Log to the console the length of the participant_response array.

```
const my_participant_data = {
  participant_id: "insert_random_id",
  participant_starting_time: Date.now(),
  participant_responses: [],
};

// Accessing values of the object
console.log(my_participant_data.participant_id); // output: insert_random_id
console.log(my_participant_data["participant_id"]); // output: insert_random_id

// Adding a property to the Object
my_participant_data.participant_RT = [];

// Adding individual values to responses
my_participant_data.participant_responses.push(1);
my_participant_data.participant_responses.push(2);
my_participant_data.participant_responses.push(3);
console.log(
  "my_participant_data.participant_responses.length : ",
  my_participant_data.participant_responses.length
);
```

For loops – are the bread and butter of online experiments. They will allow you to loop through blocks / trials / stimuli (when presenting sequences).

```
let counter = 0;

for (let i = 0; i < 5; i++) {
  counter += 1;
}

console.log("counter : ", counter); //output: "counter : 5"
```

Common mistakes, be careful !

First parenthesis:
- *Defines the counter with* ***let,***
- *Statements are separated with* ***;***
- *End statement is the increment.*

```
(let i = 0; i < 5; i++)
```

If/else – Are useful in general. For example, they help you decide if you should push a *success* or *fail* in the participant_performance array. Or, if you should take into account a double click or not (avoid accidental registering of double taps).

```
if (
  participant_responses[counter_trial] ==
  participant_correct_stimuli[counter_trial]
) {
  participant_performance.push("success");
} else {
  participant_performance.push("fail");
}
```

```
let score = 75;

if (score >= 90) {
    console.log("Grade: A");
} else if (score >= 80) {
    console.log("Grade: B");
} else if (score >= 70) {
    console.log("Grade: C");
} else {
    console.log("Grade: F");
}
```

Instructions: Practice 1

1. Open: course_practice_code*/02-javascript_basics/my_script.js*

2. Use a for loop to increment block numbers (counter_block) every 10 trial (counter_trial) increments. There are 60 trials in the experiment.

3. Every time you increment the block number, log into the console its current value.

Tips: define a total_nb_trials_per_block constant + total_nb_trials.

Instructions: Practice 2

1. Open: *example_code/02-javascript_basics/my_script.js*

2. Add a property : experiment_stimuli to the my_participant_data object, with the array [‘blue’, ‘red’, ‘yellow’] as values

3. Add a property: participant_responses to the my_participant_data object, with array [‘red’, ‘red’, ‘blue’] as values.

4. Add a property: participant_performance to my_participant_data object, with empty array as value.

5. Loop over the participant_responses and experiment_stimuli to compare them. When they match, add “success” to the participant_performance array, otherwise, add “fail”.

6. Log participant performance to the console.

Instructions: Practice 1

1. Open: *course_practice_code/02-javascript_basics/my_script.js*
2. Use a for loop to increment block numbers (counter_block) every 10 trial (counter_trial) increments. There are 60 trials in the experiment.
3. Every time you increment the block number, log into the console its current value.

Instructions: Practice 2

1. Open: *example_code/02-javascript_basics/my_script.js*
2. Add a property : experiment_stimuli to the my_participant_data object, with the array [‘blue’, ‘red’, ‘yellow’] as values
3. Add a property: participant_responses to the my_participant_data object, with array [‘red’, ‘red’, ‘blue’] as values
4. Add a property: participant_performance to my_participant_data object, with empty array as value.
5. Loop over the participant_responses and experiment_stimuli to compare them. When they match, add “success” to the participant_performance array, otherwise, add “fail”.
6. Log participant performance to the console.

```
// *** Practice - 1
let counter_trial = 0;
let counter_block = 1;
const total_nb_trials_per_block = 10;
const total_nb_trials = 60;

for (let i = 0; i < total_nb_trials; i++) {
  if ((counter_trial % total_nb_trials_per_block == 0) &
  (counter_trial != 0)) {
    counter_block += 1;
    console.log("Moving to Block : ", counter_block);
  }
  counter_trial += 1;
}
```

```
// *** Practice - 2
// Define all the needed properties
my_participant_data.experiment_stimuli = ["blue", "red",
"yellow"];
my_participant_data.participant_responses = ["red", "red",
"blue"];
my_participant_data.participant_performance = [];

// Reset counter for trials
counter_trial = 0;

for (let i = 0; i < my_participant_data.experiment_stimuli.
length; i++) {
  if (
    my_participant_data.participant_responses[counter_trial] ==
    my_participant_data.experiment_stimuli[counter_trial]
  ) {
    my_participant_data.participant_performance.push("success");
  } else {
    my_participant_data.participant_performance.push("fail");
  }
  counter_trial += 1;
}

console.log(
  "participant_performance",
  my_participant_data.participant_performance
);
```

03 – DOM Manipulation (JS)

Manipulate HTML and CSS with javascript

DOM – Document Object Model

The DOM represents the **HTML structure** as a **tree** that JavaScript can manipulate.

- It allows interaction with **HTML elements** using **methods** on the document and window objects.
- Enables **dynamic updates** (changing content, styles, events, etc.).

Get **Screen** measurements: Constant

```
let screen_height = window.screen.height;
let screen_width = window.screen.width;
```

Get inner **browser**'s window measurements: Varies

```
let viewport_height = window.innerHeight;
let viewport_width = window.innerWidth;
```

Select **HTML element** using class (querySelector) or id (getElementById)

```
let container = document.querySelector('.container');
let blue_btn = document.getElementById('blue_btn');
let red_btn = document.getElementById('red_btn');
let green_btn = document.getElementById('green_btn');
```

page.html > ...

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
  </head>
  <body>
    <div class="container">
      <div class="big-button" id="blue_btn"></div>
      <div class="big-button" id="red_btn"></div>
      <div class="big-button" id="green_btn"></div>
    </div>
  </body>
</html>
```

Get **measures of HTML elements**

```
let container_height = container.offsetHeight;
let container_width = container.offsetWidth;
```

Select **HTML elements' properties values**

```
let container_styles = window.getComputedStyles(container);
let container_margin = container_styles.margin;
```

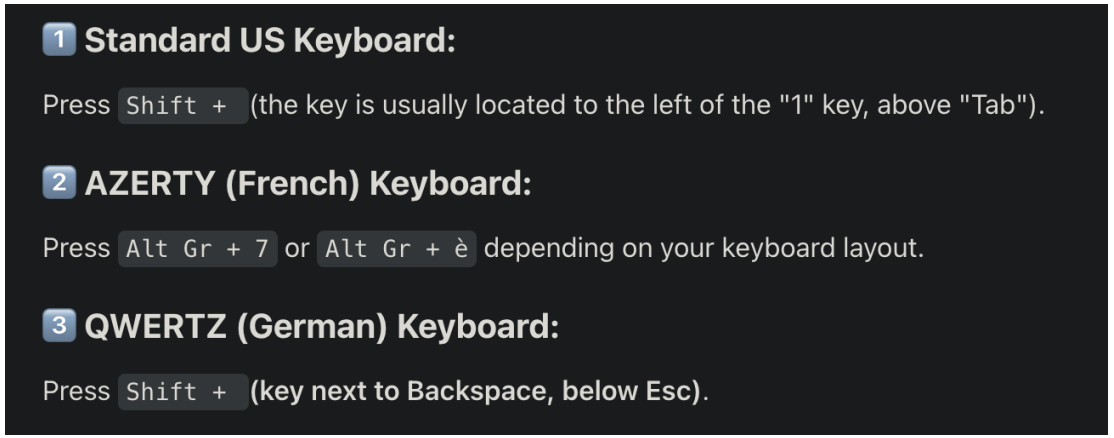
Instructions: Practice 1

1. Open : *course_practice_code/03-DOM_manipulation* go in the script-main.js file
2. Store into variables : screen width and browser window's width.
3. Log those two variables to the console and variate browser window's size. See how it affects the values.
4. [In the index.html] Have a look at the div elements
5. [In the JS file] use HTML element's class names to assign them to constants in script-main.js
6. Retrieve screen_height and screen_width values and store them into variables.
7. Retrieve container_word width and height. Store the values in two variables.
8. Log the two values in the console such that the output will read "container_word width = xyz px, container_word height = xyz px" ➞ Use Template Literal.

Template Literal `${}`: Allows to embed expressions inside a string, similar to f'Hi this is {name}' in python. The backticks ``are on £ key on Mac Keyboard. See on right for other keyboards.

Try:

```
let name = "whatever";
console.log(`Hi, my name is ${name}`);
```



Instructions: Practice 1

1. Open : *course_practice_code/03-DOM_manipulation* go in the script-main.js file
2. Store into variables : screen width and browser window's width.
3. Log those two variables to the console and variate browser window's size. See how it affects the values.
4. [In the index.html] Have a look at the div elements
5. [In the JS file] use HTML element's class names to assign them to constants in script-main.js
6. Retrieve screen_height and screen_width values and store them into variables.
7. Retrieve container_word width and height. Store the values in two variables.
8. Log the two values in the console such that the output will read "container_word width = xyz px, container_word height = xyz px".

```
// ---- Practice 1

const canvas = document.querySelector(".canvas");
const btn_start = document.querySelector(".
btn-start");
const container_word = document.querySelector(".
container-word");

let screen_height = window.innerHeight;
let screen_width = window.innerWidth;

let canvas_height = screen_height - 2 *
canvas_margin_y;
let canvas_width = screen_width - 2 *
canvas_margin_x;

const container_word_width = container_word.
offsetWidth;
const container_word_height = container_word.
offsetHeight;

console.log(
  `Container word width is ${container_word_width}
  px, and height is ${container_word_height}`
);
```

Set **Element** measurements

```
canvas.style.height = ‘100px’;
canvas.style.width = `${screen_width}px`;
```

Add **Element’s** id

```
canvas.id = “my-id”;
```

Add or Remove **Element’s** class

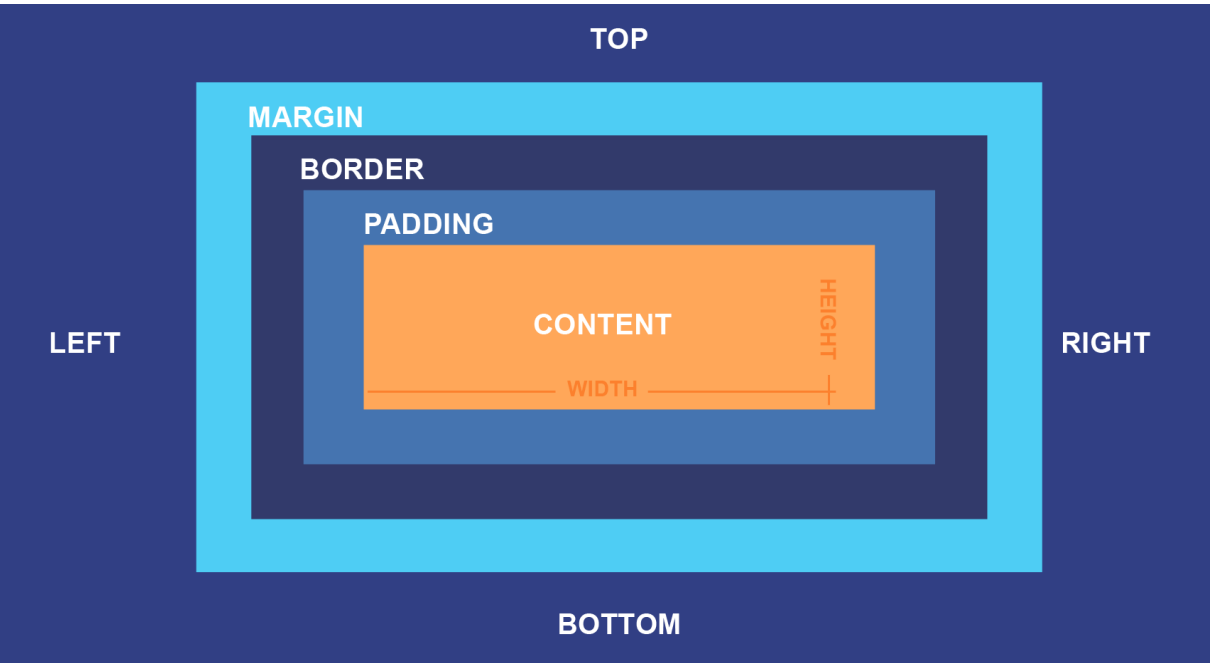
```
canvas.classList.add(‘my_class’);
canvas.classList.remove(‘my_class’);
```

Key properties useful to modify

```
container.textContent = "Hello World";

// Centers the container on the screen vertically.
// Top creates a distance from top.
container.style.top = `${
  (document.body.offsetHeight - container.offsetHeight) / 2
}px`;

// Creates a distance from the left side of the window.
// This code centers the container horizontally.
container.style.left = `${
  (document.body.offsetWidth - container.offsetWidth) / 2
}px`;
```



Box Model - The **CSS Box Model** defines how elements are structured and spaced.

Source : https://blog.hubspot.com/website/css-box-model

Instructions: Practice 2

1. Open : *course_practice_code/03-DOM_manipulation*
2. [in the CSS] Define a class at the bottom “hidden” that hides the element.
3. [in Javascript] Inspect the “measurements for styling” section in the script.
4. [in Javascript] use previously inspected variables to define canvas : height, width, margin. Using: canvas_height, canvas_width, canvas_margin_y and canvas_margin_x.
5. [in Javascript] Use previously defined values to center container_word element. (use left and top properties).
6. [in Javascript] Set fontsize of container_word to the given constant font_size.
7. Hide the start button using the *.hidden* class.
8. Define a function present_word. It has one parameter *word*. The function changes the text in the container word.
9. Execute this function using the *words_set* array to choose a word. Display the word “yellow”.
10. Add a parameter *word_color* to your function present_word. Implement the colour change to your function.
11. Change the color of the displayed word using colors_set (this is chosen colours’ HEX code using *https://coolors.co/*).

Instructions: Practice 2

1. Open : *course_practice_code/03-DOM_manipulation*
2. [in the CSS] Define a class at the bottom “hidden” that hides the element.
3. [in Javascript] Inspect the “measurements for styling” section in the script.
4. [in Javascript] use previously inspected variables to define canvas : height, width, margin. Using: canvas_height, canvas_width, canvas_margin_y and canvas_margin_x.
5. [in Javascript] Use previously defined values to center container_word element. (use left and top properties).
6. [in Javascript] Set fontsize of container_word to the given constant font_size.
7. Hide the start button using the *.hidden* class.
8. Define a function present_word. It has one parameter *word*. The function changes the text in the container word.
9. Execute this function using the *words_set* array to choose a word. Display the word “yellow”.
10. Add a parameter *word_color* to your function present_word. Implement the colour change to your function.
11. Change the color of the displayed word using colors_set (this is chosen colours’ HEX code using *https://coolors.co/*).

CSS

```css
/* Practice -- 2 */
.hidden {
  display: none;
}
```

Javascript

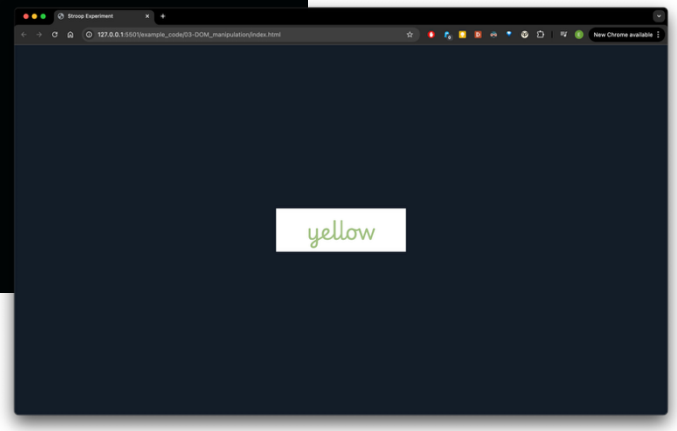
```javascript
// ---- Practice 2
// Apply canvas dimensions and margins
canvas.style.height = `${canvas_height}px`;
canvas.style.width = `${canvas_width}px`;
canvas.style.margin = `${canvas_margin_y}px ${canvas_margin_x}px`;

container_word.style.left = `${(canvas_width - container_word_width) / 2}px`;
container_word.style.top = `${(canvas_height - container_word_height) / 2}px`;
container_word.style.fontSize = `${font_size}px`; // Set font size for text stimuli
```

```javascript
// ---- Practice 2
btn_start.classList.add("hidden");
function present_word(word, word_color) {
  container_word.textContent = word; // Set word text
  container_word.style.color = `#${word_color}`; // Set word color
}

/* ********************************
---------- Code Execution ---------
******************************** */

present_word(words_set[3], colors_set[1]);
```



Three steps to create a HTML elements in javascript.

1. Create the element with **document.createElement**
2. Style the new element,
3. Insert the element in a container using **container_element.appendChild(new_element)**

```
// 1. Create a new element
const newDiv = document.createElement("div"); // Creates a <div> element

// 2. Set attributes and properties
newDiv.id = "myDiv"; // Set an ID
newDiv.className = "container"; // Set a class
newDiv.textContent = "Hello, World!"; // Add text content

// 3. Append to the DOM
document.body.appendChild(newDiv); // Adds <div> to <body>
```

Instructions: Practice 3

1. Open : *course_practice_code/03-DOM_manipulation*
2. Create a div with the class name “container-btns” within the canvas element
3. Create 4 div elements with the class ”btn” within the “container-btns”. Each button should have a class relative to its color “red”, “blue”, “yellow”, “green”.

Instructions: Practice 3

1. Open : *course_practice_code/03-DOM_manipulation*
2. Create a div with the class name “container-btns” within the canvas element
3. Create 4 div elements with the class ”btn” within the “container-btns”. Each button should have a class relative to its color “red”, “blue”, “yellow”, “green”.

```
// ---- Practice 3
const container_btns = document.createElement("div");
container_btns.classList.add("container-btns");

for (let color of words_set) {
  let div = document.createElement("div");
  div.classList.add("btn");
  div.classList.add(`btn-${color}`);
  container_btns.appendChild(div);
}

canvas.appendChild(container_btns);
```

03 – b – CSS digression

Drawing Basic Shapes

Square and **Rectangle** are the most basic shapes in CSS.

```
.square {
width: 100px;
height: 100px;
background-color: rgb(255, 166, 0);
}
```

```
.rectangle {
width: 150px;
height: 100px;
background-color: rgba(255, 0, 0, 0.549);
}
```

For the **circle**, the key is setting ***border-radius*** to ***50%***.

```
.circle {

width: 100px;

height: 100px;

background-color: rgb(0, 191, 255);

border-radius: 50%;

}
```

For the **triangle**, the key are the 3 properties:

- Border-left
- Border-right
- Border-bottom

```
.triangle {

width: 0;

height: 0;

border-left: 50px solid transparent;

border-right: 50px solid transparent;

border-bottom: 100px solid rgb(0, 128, 58);

}
```

Experiment with the different values of these 3 properties. You can draw all sorts of triangles. Use the code in: *course_practice_code/03-b-CSS_basic_shapes*

04 – Handling Events

Event Listeners in JavaScript

Event listeners allow you to execute code when a specific event occurs, such as a button click or a key press. We use them all the time in Cognitive Science experiments to retrieve answers, Response Times, and trigger new events.

```
my_element.addEventListener(type,listener)
```

Parameters of the event listener

Type

- "Keydown" : for keyboard presses
- "click" for mouse clicks (computer)
- "touch" for smartphones/tablets

Listener

Usually, a predefined function or arrow function

🛑 Common undesirable behaviors. These will appear as you test your app and should be addressed/disabled with event listeners.

Double tap – 2 x "touch" events in rapid successions default to zoom. Most of the time, you'll choose to not take into consideration touch events that are faster than 200ms, as they are accidental.

Key presses – Usually, Key presses have a default function on browsers. Space bar for example has a scrolling function.

Pinch – 2 fingers pinches on smartphones automatically zoom in, which can result in several trials/stimuli missed.

Event Listeners in JavaScript

Event listeners allow you to execute code when a specific event occurs, such as a button click or a key press. We use them all the time in Cognitive Science experiments to retrieve answers, Response Times, and trigger new events.

```
/*
========== Supplementary code ==========
*/
/* Handling Keyboard presses.
The following example uses the space bar. event.code for space bar is "Space"
To figure out the code of the key of your choice, look at the console. event codes are logged there
by the event listener.
*/
let spacebarCount = 0;
let last_key_pressed = "";
const space_bar_count = document.getElementById("spaceBarCount");

document.addEventListener("keydown", function (event) {
  last_key_pressed = event.code;

  if (last_key_pressed === "Space") {
    event.preventDefault(); // Prevents default scrolling behavior
    spacebarCount++;
    space_bar_count.textContent = `${spacebarCount} times`;
  } else {
    console.log(`${last_key_pressed} was pressed.`);
  }
});
```

Source : short_coding_exercises/02-handling_events/final/script.js

Steps to retrieve inputs

1. Select relevant HTML elements in the JS.
2. Declare variables that store the inputs.
3. Attach the event listener to the relevant HTML element (for key presses it is document).

```
my_element.addEventListener(type,listener)
```

4. Define parameters of the event listener
   a. Type :
      - "Keydown" : for keyboard
      - "click" for mouse clicks
      - "touch" for smartphones/tablets
   b. Listener : usually a predefined function or arrow function

Event Listeners in JavaScript

Event listeners allow you to execute code when a specific event occurs, such as a button click or a key press. We use them all the time in Cognitive Science experiments to retrieve answers, Response Times, and trigger new events.

```
/*
========= Supplementary code =========
*/
/* Handling Keyboard presses.
The following example uses the space bar. event.code for space bar is "Space"
To figure out the code of the key of your choice, look at the console. event codes are logged there
by the event listener.
*/
let spacebarCount = 0;
let last_key_pressed = "";
const space_bar_count = document.getElementById("spaceBarCount");

document.addEventListener("keydown", function (event) {
  last_key_pressed = event.code;

  if (last_key_pressed === "Space") {
    event.preventDefault(); // Prevents default scrolling behavior
    spacebarCount++;
    space_bar_count.textContent = `${spacebarCount} times`;
  } else {
    console.log(`${last_key_pressed} was pressed.`);
  }
});
```

Source : short_coding_exercises/02-handling_events/final/script.js

Steps to retrieve inputs

1. Select relevant HTML elements in the JS.
2. Declare variables that store the inputs.
3. Attach the event listener to the relevant HTML element (for key presses it is document).

```
my_element.addEventListener(type,listener)
```

4. Define parameters of the event listener
   a. Type :
      - "Keydown" : for keyboard
      - "click" for mouse clicks
      - "touch" for smartphones/tablets
   b. Listener : usually a predefined function or arrow function

Instructions – Practice 1

1. Open : *short_coding_exercises/02-handling_events/starter*
2. Assign the **HTML button** to a const in the **JS file**. Same for the **count** element.
3. Attach an event listener to the button with an arrow function.
4. The arrow function should increment the counter and update the text with counter of clicks on the button.

Instructions – Practice 2

1. Add a **Reset** button in the **HTML**.
2. Assign this new reset button to a const in the JS file.
3. Attach an event listener to the reset button. The button should reset the counter on click and update the displayed text.

Instructions – Practice 1

1. Open : *short_coding_exercises/02-handling_events/starter*
2. Assign the **HTML button** to a const in the **JS file**. Same for the **count** element.
3. Attach an event listener to the button with an arrow function.
4. The arrow function should increment the counter and update the text with counter of clicks on the button.

Instructions – Practice 2

1. Add a **Reset** button in the **HTML**.
2. Assign this new reset button to a const in the JS file.
3. Attach an event listener to the reset button. The button should reset the counter on click and update the displayed text.

```
/*
========= Practice 1 =========
*/

// Select elements
const button = document.getElementById("incrementBtn");
const countDisplay = document.getElementById("count");

let count = 0; // Initial count value

// Add event listener to button
button.addEventListener("click", () => {
  count++; // Increase counter
  countDisplay.textContent = count; // Update displayed count
});
```

```
/*
========= Practice 2 =========
*/
const resetBtn = document.getElementById("resetBtn");
// Reset button event listener
resetBtn.addEventListener("click", function () {
  count = 0; // Reset counter
  countDisplay.textContent = count; // Update display
});
```

```
  <body>
    <button id="incrementBtn">Click Me</button>
    <p>Count: <span id="count">0</span></p>
    <button id="resetBtn">Reset</button>
    <p>Space Bar pressed : <span id="spaceBarCount">0</span></p>
    <script src="script.js"></script>
  </body>
</html>
```

Triggering events with a delay – SetTimeout()

In experiments, we want to have small blocks of events that happen one after the other.

Usually, a response of the participant will result in fixation cross for a set delay, before the next stimulus is displayed. Additionally, it is very useful when presenting several stimuli in sequence with a set delay (SOA) in between each presentation.

In Javascript, this is done using **SetTimeout(function, delay)**. The delay parameter should be specified in milliseconds.

```
const SOA = 400;

setTimeout(() => {
  console.log(`This message appears after ${SOA} ms.`);
}, SOA);
```

Instructions – Practice 1

1. Open: *course_practice_code/04-Events/starter*
2. Examine the HTML and Javascript document files
3. Create a function fixation_screen(duration). That should:
    a. Hide: container_word and container_btns,
    b. Reveal : fixation,
    c. Revert the displayed / hidden elements after set *duration*

4. Attach this function to the red button with an event Listener. Fixation_screen() should be triggered on click of the red button.
5. Try it with 400ms of duration.

Instructions – Practice 2

1. Define a last_click variable. Set its value to Date.now() – current time in milliseconds.
2. Define empty holder arrays : holder_RT, holder_response, holder_timestamp.
3. Create a function log_data(response, timestamp). That should:
    a. Add time in ms since last click to **holder_RT**,
    b. Add response (str) to **holder_response**,
    c. Add current time (timestamp) in ms to **holder_timestamp,**
    d. Update last_click with new current time,
    e. Print all the holders to the console.
4. Attach this function to the btn-blue and experiment with live server. The holder_response should be full of [‘blue’] and holder_RT should track your clicking speed.

Instructions – Practice 1

1. Open: *course_practice_code/04-Events/starter*
2. Examine the HTML and Javascript document files
3. Create a function fixation_screen(duration). That should:
   a. Hide: container_word and container_btns,
   b. Reveal : fixation,
   c. Revert the displayed / hidden elements after set *duration*
4. Attach this function to the red button with an event Listener. Fixation_screen() should be triggered on click of the red button.
5. Try it with 400ms of duration.

Instructions – Practice 2

1. Define a last_click variable. Set its value to Date.now() – current time in milliseconds.
2. Define empty holder arrays : holder_RT, holder_response, holder_timestamp.
3. Create a function log_data(response, timestamp). That should:
   a. Add time in ms since last click to **holder_RT**,
   b. Add response (str) to **holder_response**,
   c. Add current time (timestamp) in ms to **holder_timestamp,**
   d. Update last_click with new current time,
   e. Print all the holders to the console.
4. Attach this function to the btn-blue and experiment with live server. The holder_response should be full of ['blue'] and holder_RT should track your clicking speed.

```
// -- Practice 1
const break_duration = 400; // Duration between two stimuli

function fixation_screen(duration) {
  // duration (int): duration in miliseconds for the fixation display
  fixation.classList.remove("hidden");
  container_word.classList.add("hidden");
  container_btns.classList.add("hidden");

  setTimeout(() => {
    fixation.classList.add("hidden");
    container_word.classList.remove("hidden");
    container_btns.classList.remove("hidden");
  }, duration);
}

/* ********************************
---------- Code Execution ---------
******************************** */

btn_red.addEventListener("click", () => fixation_screen(break_duration));

// -- Practice 2
let holder_RT = [],
  holder_response = [],
  holder_timestamp = [];

let last_click = Date.now();

function log_data(response, timestamp) {
  holder_response.push(response);
  holder_RT.push(timestamp - last_click); // duration between word shown and click
  holder_timestamp.push(timestamp); // timestamp of click
  last_click = Date.now();

  //Logging to the console
  console.log("holder_response : ", holder_response);
  console.log("holder_RT : ", holder_RT);
  console.log("holder_timestamp : ", holder_timestamp);
}

btn_blue.addEventListener("click", () => log_data("blue", Date.now()));
```

Concepts not covered during the course

Concepts that are important in general but that we can do without for Cognitive Science programming.

Javascript

- The **this** keyword. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this
- Hoisting and scope.
- Prototypes and prototypal inheritance.

CSS

- Display properties: Flex, Flexbox, grid.

Apply those rules at all time for better coding.

1. Give self explanatory names to your variables and functions.
2. Don’t Repeat Yourself : if you need one value several times, it should be a constant at the top of your script.
3. Make sure all the values of your participant_data objects are arrays of the same size, to avoid parsing errors in your dataset.

Resources

When doing web programming, you don't have to reinvent the wheel every time. Especially for CSS and HTML code.

1. Go on Codepen.io to find HTML and CSS of an element you like. For example : a button (look at this responsive button!). Don't forget to credit the author in your code. And take some time to read and understand what the code doe. Avoid copying javascript, but CSS of specific elements is fine.

2. Use https://coolors.co/ for color palets.

3. Complete Stroop Project code is in /stroop_project/Final