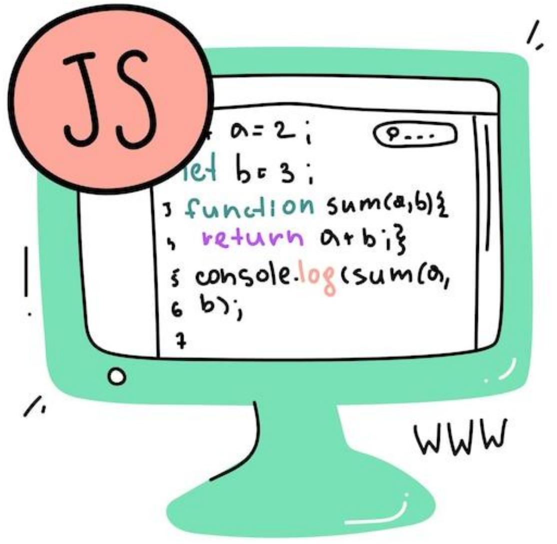
FRONT END DEVELOPMENT

WEEK 1 – JS Revisions + DATA VIEW



```
 a= 2;
let b= 3;
function sum(a,b){
  return a+b;}
console.log(sum(a,
b));
```

CADT
IDT

🥇 What will you learn today? 🥇

✓ Review **TERM-1** **(JS and DOM)**

✓ Differentiate the **DATA** and the **VIEW**

✓ Understand the **CLEAN APPROACH** to update the view with the data

✓ Apply the **CLEAN APPROACH** , by refactoring a smelly code

IN-CLASS **QUIZ**

What do you remember form Term 1 ?

Q1

Match **left items** with **right items**

Write your answer like : A-2 B-4 etc..

A. Get the **child** of given **INDEX** of the current element

B. Get **all elements** that match the **QUERY**

C. Get the **element** that matches with **ID**

D. Get **all elements** that match with the **CLASS**

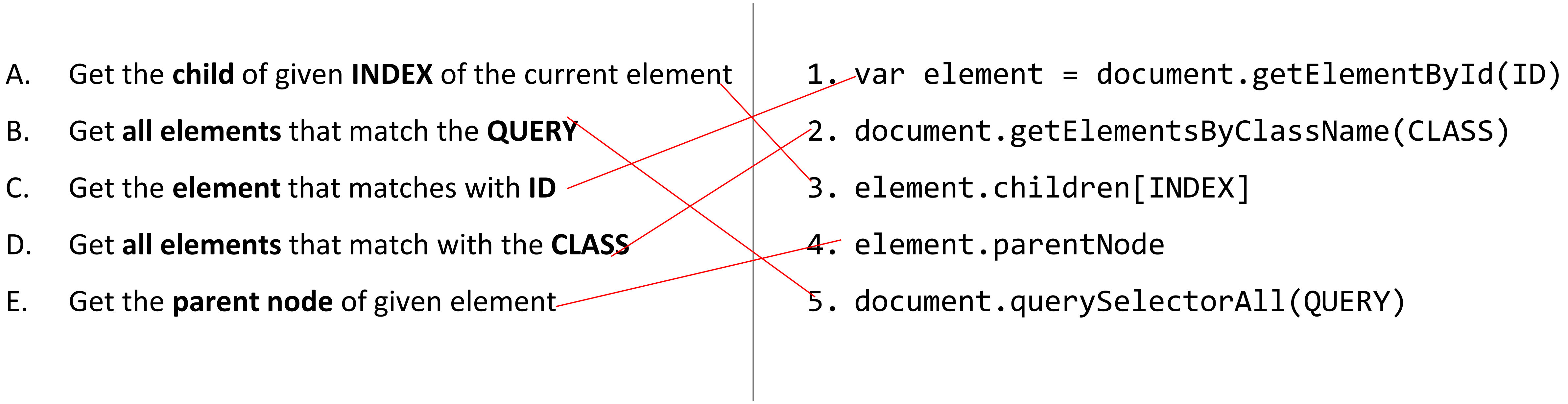
E. Get the **parent node** of given element

1. `var element = document.getElementById(ID)`
2. `document.getElementsByClassName(CLASS)`
3. `element.children[INDEX]`
4. `element.parentNode`
5. `document.querySelectorAll(QUERY)`

ASNWER

Match **left items** with **right items**

Write your answer like : A-2 B-4 etc..



A. Get the **child** of given **INDEX** of the current element

B. Get **all elements** that match the **QUERY**

C. Get the **element** that matches with **ID**

D. Get **all elements** that match with the **CLASS**

E. Get the **parent node** of given element

```
1. var element = document.getElementById(ID)
2. document.getElementsByClassName(CLASS)
3. element.children[INDEX]
4. element.parentNode
5. document.querySelectorAll(QUERY)
```

Which statements are **true**?

Multiple answer possible

```
const box = document.getElementById("box");
box.style.backgroundColor = "red";
```

A. This code changes the element’s background color immediately

B. backgroundColor must be written in camelCase in JavaScript

C. This change modifies the CSS file linked to the page

D. The style change only affects this element, not others using the same CSS class

ASNWER

Which statements are **true**?

Multiple answer possible

```
const box = document.getElementById("box");
box.style.backgroundColor = "red";
```

[✓ A.] This code changes the element’s background color immediately

[✓ B.] backgroundColor must be written in camelCase in JavaScript

C. This change modifies the CSS file linked to the page

[✓ D.] The style change only affects this element, not others using the same CSS class

Q3

Which elements will still exist in the DOM?

Multiple answer possible

INITIAL DOM

```
<div id="parent">
  <p id="a">A</p>
  <p id="b">B</p>
  <span id="c">C</span>
</div>
```

JS

```
const parentElement = document.getElementById("parent");
const a = document.getElementById("a");
const b = document.getElementById("b");
const c = document.getElementById("c");

parentElement.removeChild(b);
parentElement.removeChild(b);
parentElement.remove(c);
```

[✓ A.] <p id="a">

B. <p id="b">

[✓ C.] <span id="c">

[✓ D.] <div id="parent">

Second removeChild(b) **throws an error**

ASNWER

Which elements will still exist in the DOM?

Multiple answer possible

INITIAL DOM

```
<div id="parent">
  <p id="a">A</p>
  <p id="b">B</p>
  <span id="c">C</span>
</div>
```

JS

```
const parentElement = document.getElementById("parent");
const a = document.getElementById("a");
const b = document.getElementById("b");
const c = document.getElementById("c");

parentElement.removeChild(b);
parentElement.removeChild(b);
parentElement.remove(c);
```

A. <p id="a">

B. <p id="b">

C. <span id="c">

D. <div id="parent">

Q4

After clicking the button **once**, what will be displayed inside <p id="count">?

```
<button id="btn">Click</button>
<p id="count">0</p>
```

```
const button = document.getElementById("btn");
const counter = document.getElementById("count");

let value = 0;

function increment() {
  value++;
  counter.textContent = value;
}


button.addEventListener("click", increment);
button.addEventListener("click", increment);

button.removeEventListener("click", increment);
```

[✓ A.] 0

B. 1

C. 2

D. An error occurs

ASNWER

After clicking the button **once**, what will be displayed inside <p id="count">?

```
<button id="btn">Click</button>
<p id="count">0</p>
```

```
const button = document.getElementById("btn");
const counter = document.getElementById("count");

let value = 0;

function increment() {
  value++;
  counter.textContent = value;
}


button.addEventListener("click", increment);
button.addEventListener("click", increment);

button.removeEventListener("click", increment);
```

Adding the same function reference twice does not create two listeners.

[✓ A. 0]

B. 1

C. 2

D. An error occurs

Here No click listeners remain !

So Clicking the button does nothing.

removeEventListener removes the matching listener

evenKey -> function

Q5

Fill out the gaps

When the button is clicked, all list items (<li>) with a score below 50 must turn red.

```
<ul id="scores">
  <li>72</li>
  <li>45</li>
  <li>90</li>
  <li>30</li>
</ul>

<button id="checkBtn">Check </button>
```

```
const button = document.getElementById("checkBtn");
const items = document._____A____("scores").____B_____("li");

button.____C_____("click", () => {
  for (let i = 0; i < items.____D_____; i++) {
    const score = parseInt(items[i]._____E____);

    if (score < 50) {
      items[i].style.___F______ = "red";
    }
  }
});
```

A =

B =

C =

D =

E =

F =

ASNWER

Fill out the gaps

When the button is clicked, all list items (<li>) with a score below 50 must turn red.

```
<ul id="scores">
  <li>72</li>
  <li>45</li>
  <li>90</li>
  <li>30</li>
</ul>

<button id="checkBtn">Check </button>
```

```
const button = document.getElementById("checkBtn");
const items = document._____A____("scores").____B_____("li");

button.____C_____("click", () => {
  for (let i = 0; i < items.____D_____; i++) {
    const score = parseInt(items[i]._____E____);

    if (score < 50) {
      items[i].style.___F______ = "red";
    }
  }
});
```

A. getElementById
B. querySelectorAll
C. addEventListener
D. length
E. textContent
F. color

Q6

Fill out the gaps

When the button is clicked, all list items (<li>) with a score below 50 must turn red.

```
<label for="numA">Number A:</label>
<input type="number" id="numA">

<label for="numB">Number B:</label>
<input type="number" id="numB">

<button id="validateBtn">Validate</button>
<p id="errorMsg" style="color:red;"></p>
```

```
const numA = document.getElementById("numB");
const numB = document.getElementById("numB");
const button = document.getElementById("validateBtn");
const errorMsg = document.getElementById("errorMsg");

// Add click event listener
button.__A__("click", function() {

    // Get values
    const a = parseInt(numA.__B__);
    const b = parseInt(numA.__B__);

    // Check if B is greater than A
    if (b <= a) {
        errorMsg.textContent = "B must be greater than A";
    } else {
        errorMsg.textContent = "__C__"; // Clear
    }
});
```

A =

B =

C =

ASNWER

Fill out the gaps

When the button is clicked, all list items (<li>) with a score below 50 must turn red.

```
<label for="numA">Number A:</label>
<input type="number" id="numA">

<label for="numB">Number B:</label>
<input type="number" id="numB">

<button id="validateBtn">Validate</button>
<p id="errorMsg" style="color:red;"></p>
```

```
const numA = document.getElementById("numB");
const numB = document.getElementById("numB");
const button = document.getElementById("validateBtn");
const errorMsg = document.getElementById("errorMsg");

// Add click event listener
button.__A__("click", function() {

    // Get values
    const a = parseInt(numA.__B__);
    const b = parseInt(numA.__B__);

    // Check if B is greater than A
    if (b <= a) {
        errorMsg.textContent = "B must be greater than A";
    } else {
        errorMsg.textContent = "__C__"; // Clear
    }
});
```

A = addEventListener

B = value

C = " "

EXPLAIN

DATA vs VIEW

- ✓ Data is the **actual information** your application works with.
- ✓ It is stored in **JavaScript variables**, objects, or arrays.
- ✓ Data **does not depend on the DOM**—it’s independent and can exist even without a UI.

```
const student = { name:"namKea", thebest:true};
```

EXPLAIN

DATA vs VIEW

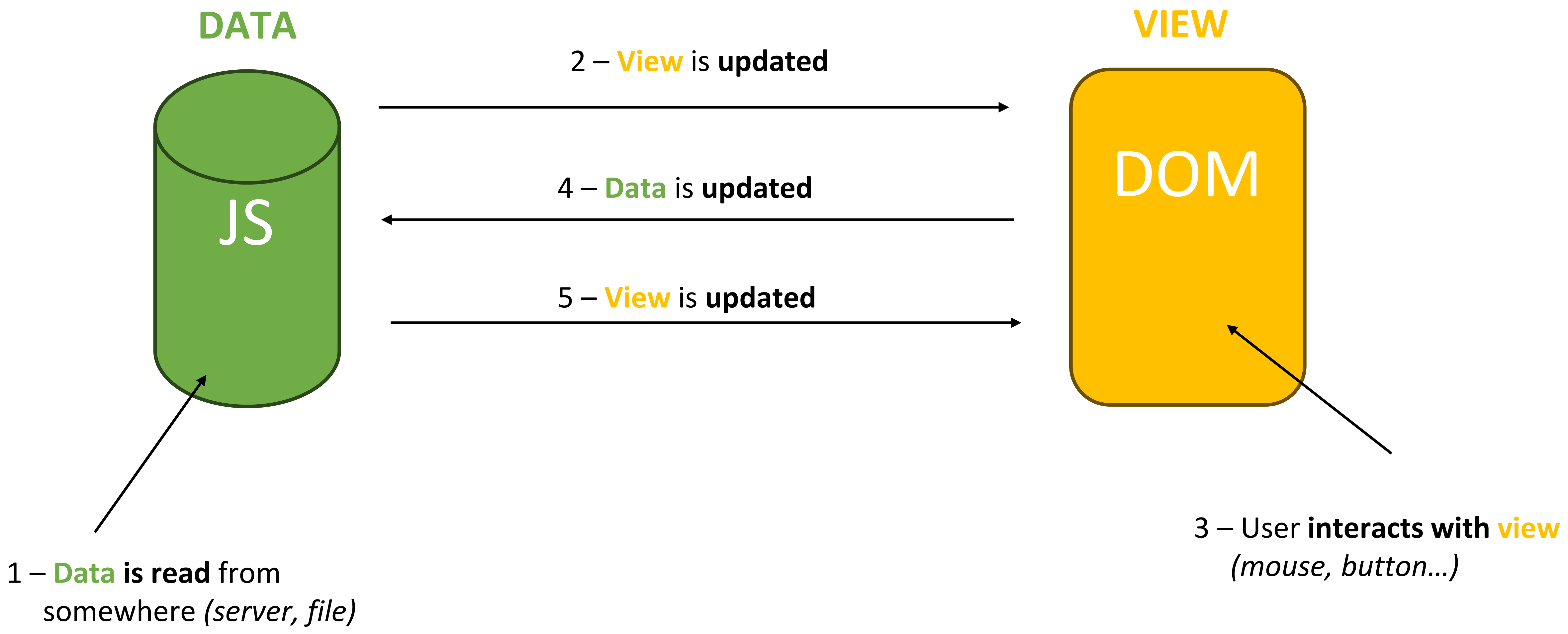
- ✓ View is the **visual representation** of the data in **the browser**.
- ✓ It is created using **HTML elements** (<p>, <ul>, <div>).
- ✓ The view is **derived from the data** and should be refreshed whenever data changes.

```
<p id="name"></p>
<p id="status"></p>
```

The content of <p> tags depends on student data.

EXPLAIN

DATA <> VIEW flow


DATA
VIEW
JS
DOM
2 – View is updated
4 – Data is updated
5 – View is updated
1 – Data is read from somewhere (server, file)
3 – User interacts with view (mouse, button…)

EXPLAIN

DATA vs VIEW – Bad approach ❌

DATA

```
const student = { name: "namKea", thebest: true };
```

VIEW

```
<div id="studentContainer">
  <p id="name"></p>
  <p id="status"></p>
  <button id="toggleBtn">Toggle Status</button>
</div>
```

CONTROL

```
const nameEl = document.getElementById("name");
const statusEl = document.getElementById("status");
const button = document.getElementById("toggleBtn");

// Initially display data directly in DOM
nameEl.textContent = student.name;
statusEl.textContent = student.thebest ? "The Best!" : "Not the best";

// Event directly modifies DOM
button.addEventListener("click", function() {
    if (statusEl.textContent === "The Best!") {
        statusEl.textContent = "Not the best";
    } else {
        statusEl.textContent = "The Best!";
    }
});
```

- We no longer have a real JS data source.
- All logic depends on reading the DOM, not the actual data.

EXPLAIN

DATA vs VIEW – Good approach ✅

DATA

```
const student = { name: "namKea", thebest: true };
```

VIEW

```
<div id="studentContainer">
  <p id="name"></p>
  <p id="status"></p>
  <button id="toggleBtn">Toggle Status</button>
</div>
```

CONTROL

```
const nameEl = document.getElementById("name");
const statusEl = document.getElementById("status");
const button = document.getElementById("toggleBtn");

// Function to render the student info from data
function renderStudent() {
    nameEl.textContent = student.name;
    statusEl.textContent = student.thebest ? "The Best!" : "Not the
best";
}

// Event modifies data, then refreshes view
button.addEventListener("click", function() {
    student.thebest = !student.thebest; // update data
    renderStudent(); // refresh UI
});

// Initial render
renderStudent();
```

- Data is the **single source of truth** (student object).
- UI is **always derived** from data.

ACTIVITY

ACTIVITY1 – **Render** tasks

Write the code to render the list of tasks

CONTROL

```
function renderTasks(array) {

  // 1  - Create the tasks container
  let container=document.createElement('taskContainer');
  container.className='container';

  // 2  - Create the elements for the task
  for(let task of tasks){
     // YOUR CODE
  }

  // 3 – Add the container to the body
  let body=document.querySelector('body');
  body.appendChild(container);
}
```

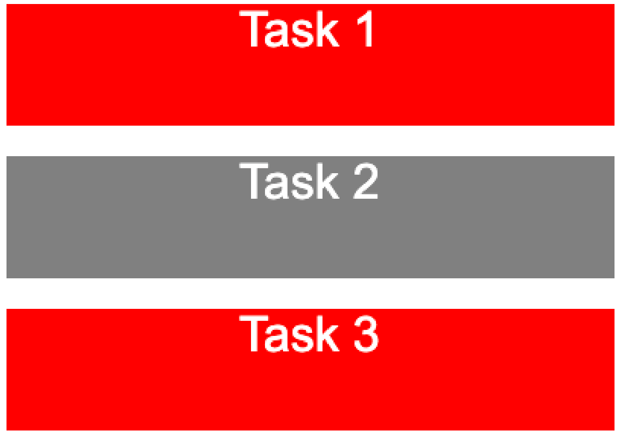
DATA

```
let tasks = [
{ description:"Task 1", priority:1},
{ description:"Task 2", priority:0},
{ description:"Task 3", priority:1}
];
```

VIEW

```
<div id="taskContainer">
 <div class="item" style="background-color: red;">Task 1</div>
 <div class="item" style="background-color: grey;">Task 2</div>
 <div class="item" style="background-color: red;">Task 2</div>

</div>
```

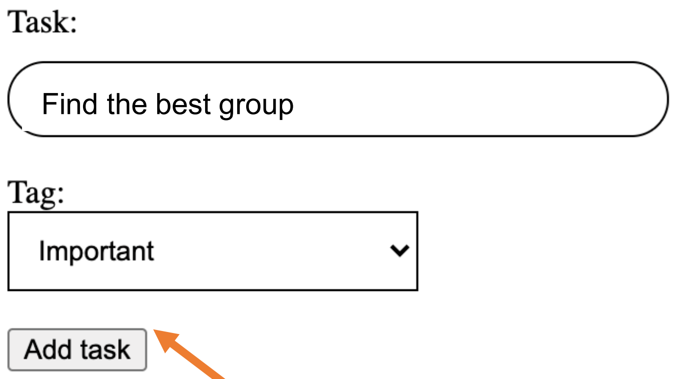


If priority 1 => RED tasks

If priority 0 => GREY tasks

ACTIVITY

ACTIVITY2 – **Update the data + re-render**



When the button is clicked …

1 - Create the new task from inputs

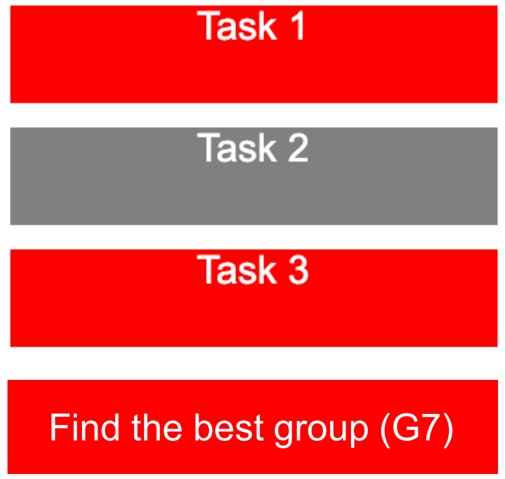
```
let tasks = [
{ description:"Task 1", priority:1},
{ description:"Task 2", priority:0},
{ description:"Task 3", priority:1}

{ description:“Find the best group", priority:1}

];
```

2 – Add it to the list

3 – Re-render



NEXT SESSION = QUIZ !

⚠ Review this session + **last term JS DOM** concepts to be ready for the **practice quiz**!!

- Understand **what is the DOM**
- **Select DOM** elements

- Change DOM **element properties**
- Change DOM **element styles**
- Handle **DOM events**
- Get **DOM input value**
- Create a **new DOM element**

- Differentiate the **DATA** and the **VIEW**
- Understand the **CLEAN APPROACH**
- Apply the **CLEAN APPROACH** ,