DOM Manipulation

One of the most fundamental use-case of JS is to be able to manipulate and control DOM (document object model).

What is the meaning of the word DOM manipulation ?

When we say, manipulating the DOM, we mean that using JS we should be able to control what elements are being added / removed to the DOM, how they behave in terms of user interaction, how their look and feel is and everything else related to these HTML elements.

How to achieve DOM manipulation ?

So our browsers provide us a lot of functions and objects using which we can control the whole DOM. Important thing to note is that these functions are not native to JS, instead they are provided by the browser (which is the runtime environment).

Some of the most important functions are:

- document.getElementById
- document.getElementsByClassName
- document.querySelector
- document.getElementsByTagName
- and more ....

In order to understand DOM manipulation we will learn by building a small TierMaker app, similar to this one

What features do we expect ?

- One should be able to create a tier bucket and name it.
- Name of the bucket should be edittable
- One should be able to add elements which can be dragged and dropped in these buckets.
- We should be able to move an element from one bucket to another bucket.
- May be double clicking on the element should bring them out of any bucket.
- Every element will be having an image, so to add an element we need to provide URL of the image directly in a form may be.

How to add JS in your HTML pages ?

There are mainly two ways to insert JS in our HTML pages

- To use the `script` tag at the end of the body and write all the JS there.

```
<body>
        <h1>
                Your Own Tier List App...
        </h1>
        <form>
                <label for="tier">Tier:</label>
                <input type="text" id="tier" name="tier" placeholder="Give
the tier name ...">
                        <br>
                <input type="submit" value="Submit">
        </form>
```

```
        <script>
                console.log("hello");
                console.log("world");
        </script>

</body>
```

Adding JS directly in the HTML page, can lead to less readability of the code, hence this is not always preferred in production codebases.

- Have a separate JS file and then link it to the HTML page using `script` tag.

```
<body>
        <h1>
                Your Own Tier List App...
        </h1>
        <form>
                <label for="tier">Tier:</label>
                <input type="text" id="tier" name="tier" placeholder="Give
the tier name ...">
                <br>
                <input type="submit" value="Submit">
        </form>

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

</body>
```

So the `script` tag has got a `src` attribute, that can help us to link a JS file to our HTML page.

How to select or target elements using JS ?

We know that using CSS if we want to target / select an element we can use class selector, id selector, tag selector etc.

```
#tier {
        padding: 5px;
}
```

To target an element in JS also, we can take help of these tag names, id's and classes.
And that's where the first function of DOM manipulation comes

A very important global object - document

Browsers provide us a very important object which is **globally present every where** in the browser's JS, called as **document**.
This object helps us to access the DOM. This object has access to the HTML, and contains a lot of methods and properties using which we can manipulate the DOM.

Event driven programming

To understand event driven programming, we need to understand concept of events first.
Events are some action or interaction done by an external agent (may be the user) on our application. For example: clicking a button is an event, scrolling is an event, moving the mouse is an event, pressing a button on keyboard is an event and so on.
So when we click a button an event is triggered.

So react on these actions, we can write logics which will only execute when these actions are triggered. And the programming paradigm where the logic / code runs as a reaction / response to these events is called as **event driven programming**.

document.getElementById

So, as the name suggests, `getElementById` helps you to target a particular individual HTML element using the `id` attribute of the HTML element.

```
document.getElementById("tier");
```

The above piece of code will help you to select the element whose id is `tier`.

If we save this element in a variable and then on that variable try to access the `value` property we will get whatever has been written inside the corresponding element.

```
const tierInput = document.getElementById("tier");
console.log(tierInput.value); // this will print the value inside the input box
```

Sometime we might have to fetch the text written inside a div, paragraph, button etc in those cases `value` doesn't help us because `value` is generally used to fetch the value of different input tags. If we want to fetch the text written inside an element we can use the `textContent` property to get and set the text of that HTML element.

```
<button id="btn">Click me</button>
```

```
const btn = document.getElementById("btn");
console.log(btn.textContent); // get existing text from the button -> Click
me
btn.textContent = "Howdy doody"; // set a new text in the button
```

htmlElement.addEventListener

This method `addEventListener` is present on almost all of the HTML elements. We can select an HTML element first and then call this `addEventListener` function on that element. It takes two arguments:

- Name of the event - example : `click`, `dblclick`, `mouseover`, `scroll` etc.
- Callback function which will be executed once the event is triggered.

```
const submitBtn = document.getElementById('submit');

submitBtn.addEventListener('click', () => {

        console.log("button is clicked");

});
```

This callback method will not be automatically called. It will be only called when `click` event happens on the `submitBtn` altogether. This callback method is sometimes also referred as `eventListener`.
The eventListener callback has a parameter called as the `event` object. This event object has a lot of details about the event triggered, like on which element this was triggered, what event got triggered and so on.
To get access of the element on which the event was fired, we can use the `event.target` property.

preventDefault

Every event object passed in the event listener callback has a preventDefault method, which stops the default execution behaviour of the event.
For example, a click event on a submit button inside a form, submits the forms and refreshes the page. This is the default behaviour. We can avoid it by using `event.preventDefault()`.

```
const submitBtn = document.getElementById('submit');

submitBtn.addEventListener('click', (event) => {
        event.preventDefault();
        console.log("button is clicked");
});
```

document.createElement

This function can help you to create brand new HTML elements using JS while running the program. This function takes one argument i.e. name of the tag of the element to be created.

The attributes of the element created can be be modified as well.
For example, if we want to add a class name to the newly created element we use the `classList` property.

Note:

The `classList` property is kind of like an array of class names attached to an html element. This property can be accessed on newly created elements and already existing elements as well.

```
const newTierList = document.createElement('div');
newTierList.classList.add('tier-list');

const heading = document.createElement('h1');
heading.textContent = tierInput;

const newTierListItems = document.createElement('div');
newTierListItems.classList.add('tier-list-items');
```

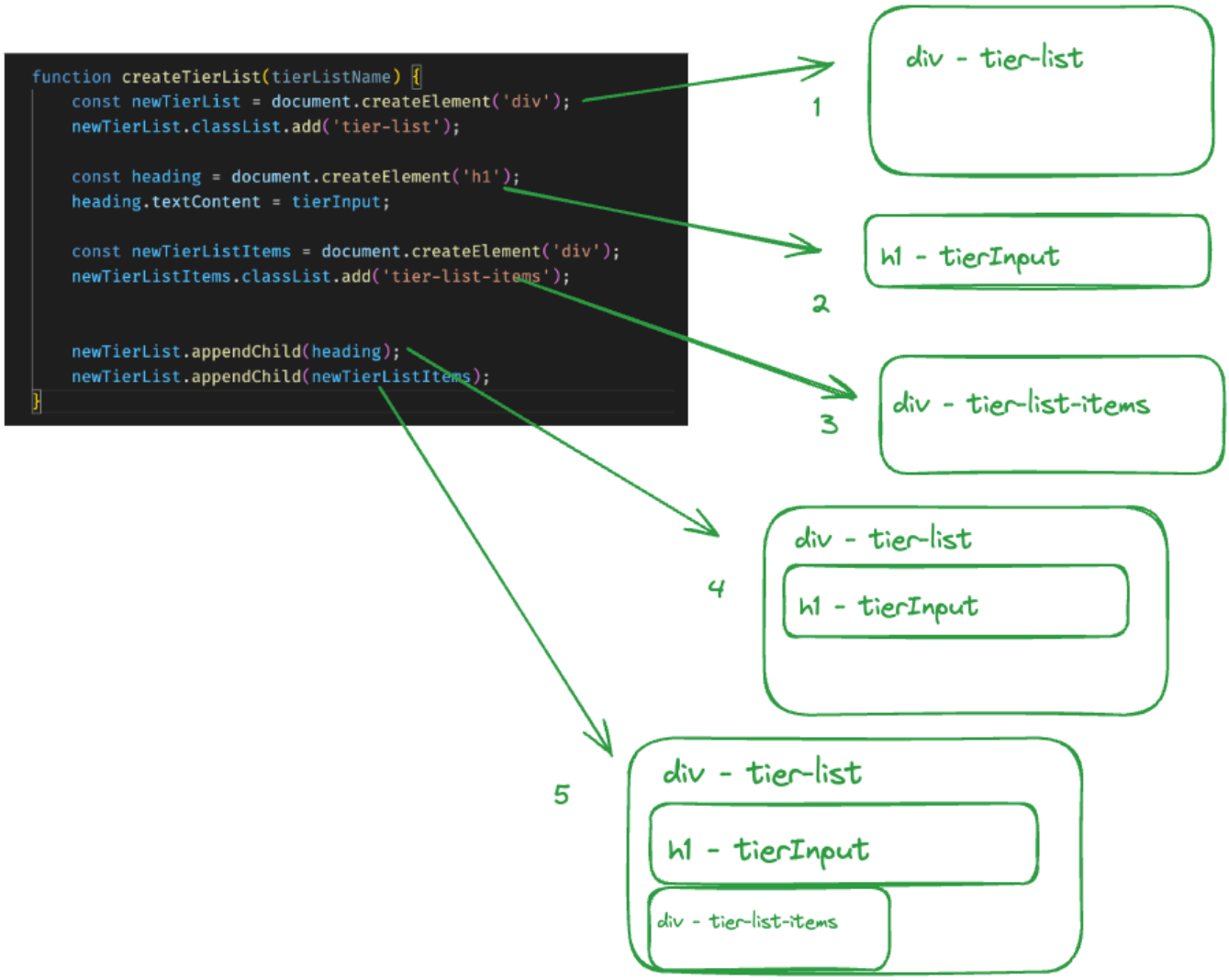
htmlElement.appendChild

We can append a given HTML element as a child of another element, by using the `appendChild` method.

```
newTierList.appendChild(heading);

newTierList.appendChild(newTierListItems);
```

Now combining all of it:



1. We first create an empty div `newTierList` and attach class `tier-list` to it.
2. We create a heading and make the textContent of the heading equal to the `tierListName` parameter passed in the function.
3. We create another div `newTierListItems` and attach class `tier-list-items` to it.
4. But none of this will be visible as of now. Why ? Because these elements exist in the memory, and not a part of DOM tree as of now.
5. To attach them to the DOM, we need to append them as a child of some parent.
6. First we add heading and newTierListItems as children of newTierList

```
newTierList.appendChild(heading);
newTierList.appendChild(newTierListItems);
```

7. We attach this `newTierList` div as a child of `tier-list-section`.

```
const tierSection = document.getElementById('tier-list-section');
tierSection.appendChild(newTierList);
```

8. All of this logic is attached on the eventListener of click to the submit button

```
submitBtn.addEventListener('click', (event) => {
        event.preventDefault(); // stops the default behaviour of the event
        // To get access of the element on which this event was fired
        const target = event.target;
        console.log(tierInput.value);
        if(tierInput.value === '') {
                alert('Please enter a tier name');
                return;
        }
        createTierList(tierInput.value);
        tierInput.value = '';

});
```

Note:

The `alert` method creates a popup for us with the message we want to show to the user.

Now let's say we want to have a separate form which can add initial set of non-tier elements which will be later dragged and dropped.

Now we can repeat the whole process just like the previous form or we can try a new approach also.

Form submission event

Any form when submitted, emits the `submit` event which we can listen to and attach and event listener.

```
<form id="image-form">
        <label for="item">Item:</label>
        <input type="text" id="item" name="item" placeholder="Give the item
image url ..." >
        <button type="submit" id="submitImage">Click me</button>
</form>
```

```
const imageForm = document.getElementById('image-form');
imageForm.addEventListener('submit', (event) => {
        event.preventDefault();
        console.log("form submitted");
});
```

Now to create the tier list items

```
function createTierListItem(imageUrl) {
        const imageDiv = document.createElement('div');
        imageDiv.classList.add('item-container');

        const img = document.createElement('img');
        img.src = imageUrl;
        imageDiv.appendChild(img);
        const nonTierSection = document.getElementById('non-tier-section');
        nonTierSection.appendChild(imageDiv);
}
```

Now the above function created the image item container and appends it to the non tier section. We should call this function on form submission

```
  imageForm.addEventListener('submit', (event) => {
        event.preventDefault();
        console.log("form submitted");
        const imageItemInput = document.getElementById('image-item');
        if(tierInput.value === '') {
                alert('Please enter a valid image url');
                return;
        }
        const imageUrl = imageItemInput.value;
        createTierListItem(imageUrl);
        imageItemInput.value = '';
});
```

Understanding more events like drag and drop

So drag and drop will be an event in the DOM, because it will be based on user actions.
To make an element capable of being dragged from one part to other we have to use the `draggable` attribute. This attribute defines whether the element is draggable or not ?
It's an enumerated attribute meaning, in whatever HTML tag we are defining this attribute we have to manually give it a `"true"` or `"false"` value as a string.

According to MDN:
Warning: This attribute is enumerated and not Boolean. A value of true or false is mandatory, and shorthand like `<img draggable>` is forbidden. The correct usage is `<img draggable="false">`

Where can we add this draggable in our code ?

```
<section id="non-tier-section">

        <div class="item-container">

                <img

                        src="https://cdn.britannica.com/30/182830-050-
96F2ED76/Chris-Evans-title-character-Joe-Johnston-Captain.jpg"

                />

        </div>

</section>
```

Inside the `non-tier-section` we are having a lot of divs with class `item-container`. These divs are going to contain images of our tier-items which needs to be made draggable. So ideally these divs should be given the `draggable = "true"` attribute.

```
<section id="non-tier-section">
        <div class="item-container" draggable="true">
                ...
        </div>
        <div class="item-container" draggable="true">
                ...
        </div>
<section>
```

But these `item-container` divs we are creating on-the-go using JS, whenever the user fills the form with an image url.
That means the newly created divs (which are made using JS) should also contain these attributes.

setAttribute function

This function is available on HTML elements, to set a particular attribute with a value. It takes two arguments :

- attribute name
- attribute value

```
imageDiv.setAttribute('draggable', 'true');
```

So we can use this `setAttribute` function to set these draggable values.

The div with class `item-container` will be dragged, so we need to add an event listener to this div. If we want to add an event listener to these divs, we need to select them as well. As all of them have got a common class, and we need to attach an event listener to all of these divs, we can use `getElementsByClassName` function.

document.getElementsByClassName

This function takes the name of class we want to select and then return us an array of all those elements which have got the same class.

This method returns us an HTMLCollection list, so normal array functions are not gonna work on it. And We need to iterate on all the elements of this list and setup eventListeners. So we can use the `for of` loop.

```
const itemContainers = document.getElementsByClassName('item-container');
for(const itemContainer of itemContainers) {
        setUpItemContainerForDrag(itemContainer);
}

function setUpItemContainerForDrag(itemContainer) {
        itemContainer.addEventListener('dragstart', () => {
                console.log("Started dragging");
        });
}
```

In Html elements we have an event called as `dragStart` which is fired when we start dragging an element.

What to do when someone starts dragging an item ?

We can store the item being dragged in a variable so that we can keep a track of it.

Because when we start dragging we will be putting the cursor on an image, we need to also get parent of the image, and to get parent of any HTML element we can use the `parentNode` property.

```
let currentDraggedItem;
// more code
function setUpItemContainerForDrag(itemContainer) {
        itemContainer.addEventListener('dragstart', (event) => {
                currentDraggedItem = event.target.parentNode;
        });
}
```

To target any element by id or class or something else, we don't just have option of `getElementById` or `getElementsByClassName`. We can use the method `querySelectorAll`.

document.querySelectorAll

It is kind of a generic method. In this method we can give css like selectors and it will select those elements.

Example 1:

```
document.querySelectorAll("#some_id");
```

This code will select an element with id `some_id`.

Example 2:

```
document.querySelectorAll(".some_class");
```

This code will select elements with class `some_class`.

Handling drops

So in the browser, we have a `drop` event, The drop event is fired when an element or text selection is dropped on a valid drop target. To ensure that the drop event always fires as expected, you should always include a `preventDefault` call in the part of your code which handles the `dragover` event.

The `dragover` event is fired when an element or text selection is being dragged over a valid drop target (every few hundred milliseconds).

```
function setUpDropZoneInTierList(tierList) {

        tierList.addEventListener('drop', (event) => {

                event.preventDefault(); // stops the default behaviour of
the event which is to not allow drop

        });

        tierList.addEventListener('dragover', (event) => {

                console.log("dragged over a drop zone");

                event.target.appendChild(currentDraggedItem);

        });

}
```

There is one problem here that the event object in the event listener of dragover is the Drag Event of the Image Item Container, so current we are trying to insert `currentDraggedItem` which is the parent of Image Item inside the itself, and that is not possible, we cannot add parent of a child as its child.

So here we need to access the tier list in which `dragover` is happening.
To get it we can change the arrow function to normal function expression and use the `this` keyword to get access to the tier list as it is the call site.

```
function setUpDropZoneInTierListItem(tierListItem) {

                tierListItem.addEventListener('drop', (event) => {

                event.preventDefault(); // stops the default behaviour of
the event which is to not allow drop

        });
```

```
        tierListItem.addEventListener('dragover', function (event) {
                // console.log(event.target);
                // event.target.appendChild(currentDraggedItem);
                console.log("event coming up", event);
                if(this !== currentDraggedItem.parentNode) {
                this.appendChild(currentDraggedItem);
        }
  });
}
```

Apart from this there is one last change expected that, when we add a new Image we should set it up for dragStart event

```
function createTierListItem(imageUrl) {
        const imageDiv = document.createElement('div');
        imageDiv.setAttribute('draggable', 'true');
        imageDiv.classList.add('item-container');

        setUpItemContainerForDrag(imageDiv); // this call should be added

        const img = document.createElement('img');
        img.src = imageUrl;

        imageDiv.appendChild(img);
```

```
        const nonTierSection = document.getElementById('non-tier-section');

        nonTierSection.appendChild(imageDiv);



}
```