DOM basics

In this section, we will see how we can interact with the HTML, how we can change different texts, handle button clicks, and many other things.

DOM

DOM stands for Document Object Model. So, `Document` is basically the HTML and `Object` is the Javascript object. `Model` is how it is structured.

DOM is basically a representation of HTML into a tree like structure.

Browser provide different objects and functions through which we can access different HTML elements inside the JavaScriptlo

Brower provide an object called `document` through which we can access the HTML. This object is not created by us. It is provided by the browser itself.

Selectors

Lets start by selecting some HTML elements.

getElementById

Returns the element object representing the element whose id is equal to the passed string.

```
<h1 id="heading-1">Hello world</h1>
<p>Nice to see you</p>

<h1>Heading again</h1>
```

```
const firstHeading = document.getElementById('heading
console.log(firstHeading); // <h1 id="heading-1">Hell
```

You get the JavaScript object with multiple properties and methods through which you can manipulate that HTML element.

For example, you can change the styles of that element using `style` property on that object.

```
const firstHeading = document.getElementById('heading-1');

firstHeading.style.color = '#ff0000';
```

The color of that element will now change to red.

---

getElementsByTagName

Returns a array of elements with the given tag name.

```
<h1>Hello world</h1>
<p>Nice to see you</p>

<h1>Heading again</h1>
<h2>Heading 2</h2>
```

```
const allHeadings = document.getElementsByTagName('h1
console.log(allHeadings);
// [<h1>Hello world</h1>, <h1>Heading again</h1>]
```

You cant call `push` and `pop` on this array, but you can loop this array to modify elements.

```
const allHeadings = document.getElementsByTagName('h1');

for (const heading of allHeadings) {
  heading.style.color = 'blue';
}
```

This will change the color of all the `h1` elements to blue.

---

getElementsByClassName

Returns array of elements with the same class.

```
<h1 class="heading">Hello world</h1>
<p>Nice to see you</p>
```

```
const arr = document.getElementsByClassName('heading'
console.log(arr);
```

```
<h1>Heading again</h1>
<h2>Heading 2</h2>
<p class="heading">Hello again</p>
```

```
// [<h1 class="heading">Hello world</h1>,
// <p class="heading">Hello again</p>]
```

Similarly, you can loop through this array and do different stuff.

querySelector

Returns the first element that matches the given selector. The string that you pass must be a valid CSS selector.

```
var el = document.querySelector('h1'); // returns first h1 tag
var el = document.querySelector('.heading'); // returns first element having heading as a class name
var el = document.querySelector('input[type="text"]'); // returns first input element having type = text
```

querySelectorAll

`querySelector` returns just the first element that matches the given string. If you want all the elements that matches the given string you can use querySelectorAll.

```
var el = document.querySelectorAll('h1'); // returns a list of all h1 tags
```

Some properties on HTML element

innerText

Returns a string representing the renedered text content of an element.

```
<p>
  Hello <i>World</i>. <span>Nice to meet you.</span>
</p>
```

```
const el = document.querySelector('p');
console.log(el.innerText);
// Hello world. Nice to meet you.
```

textContent

Very similar to `innerText`. It also returns a string representin the text inside an element.

```
<p>
  Hello <i>World</i>. <span>Nice to meet you.</span>
</p>
```

```
const el = document.querySelector('p');
console.log(el.textContent);
// Hello world. Nice to meet you.
```

Difference between `innerText` and `textContent` is that, `innerText` is aware of the styling.

For example, if there is a `span` inside `p` whose display is none. `innerText` will not print that but `textContent` will print that.

```
<p>
    Hello <i>World</i>. <span style="display: none;">Nice to meet you.</span>
</p>
```

```
const el = document.querySelector('p');
console.log(el.innerText); // Hello World.
console.log(el.textContent); // Hello World. Nice t
```

Refer this → https://developer.mozilla.org/en-US/docs/Web/API/Node/textContent#differences_from_innertext , to know more differences between them.

innerHTML

Returns the text content with the HTML markup

```
<p>
    Hello <i>World</i>. <span style="display: none;">Nice to meet you.</span>
  </p>
```

```
const el = document.querySelector('p');
console.log(el.innerHTML);
// Hello <i>World</i>. <span style="display: none;">Nice to meet you.</span>
```

Changing text inside the HTML element

You can use the above properties to change the text inside the selected HTML element.

```
<h1>Hello world</h1>
```

```
const heading = document.querySelector("h1");
heading.innerText = 'Hello India';
```

The `h1` element will now have the changed text.

Manipulating attributes

getAttribute

This method returns the value of the given attribute

```html
<a href="https://google.com">Go to google</a>
```

```javascript
const a = document.querySelector("a");
console.log(a.getAttribute('href'));
// https://google.com
```

setAttribute

Just like `getAttribute`, you can also set the attribute for the selected element.

```html
<a href="https://google.com">Go to google</a>
```

```javascript
const a = document.querySelector("a");
a.setAttribute('href', 'http://amazon.in');
```

The anchor tag will not point to `amazon.in`

Use cases

You can use `getAttribute` and `setAttribute` to change classes and apply different classes for different occasions.

Practice

Make a webapp, where image changes every 2 seconds.

```html
<img src="" />
```

```javascript
const arr = [
  'https://images.unsplash.com/photo-1662330287683-6032da28889c?ixlib=rb-1.2.1&ixid=MnwxMjA3fDB8MHxlZGl0b3JpYWwtZmVlZHwyfHx8ZW58MHx8fH
  'https://images.unsplash.com/photo-1659535907680-0e219b46c01d?ixlib=rb-1.2.1&ixid=MnwxMjA3fDF8MHxlZGl0b3JpYWwtZmVlZHwxfHx8ZW58MHx8fH
  'https://images.unsplash.com/photo-1660899599007-771f97039eee?ixlib=rb-1.2.1&ixid=MnwxMjA3fDB8MHxlZGl0b3JpYWwtZmVlZHwxNHx8fGVufDB8fH
  'https://images.unsplash.com/photo-1662324580989-591a589c5e10?ixlib=rb-1.2.1&ixid=MnwxMjA3fDB8MHxlZGl0b3JpYWwtZmVlZHwyOHx8fGVufDB8fH
]

const imageEl = document.querySelector('img');

let num = 0;

setInterval(function() {
  imageEl.setAttribute('src', arr[num]);
  num = (num + 1) % arr.length;
}, 2000);
```

Adding element to DOM

To add an element to the DOM, we first have to create that element

Creating HTML element

We can use `createElement` to create new element

```
const heading = document.createElement('h1');
```

Adding element to DOM

There are different methods to add elements to DOM

appendChild()

Adds element to the end of the list of children of the selected element.

```
<div id="section">
</div>
```

```
const section = document.getElementById('secti
const heading = document.createElement('h1');
section.appendChild(heading);
```

This will add `h1` to the `div` element. Nothing will show on the viewport because there is no text inside `h1`. We have to first add some text to `h1` before adding to the DOM.

```
const section = document.getElementById('section');
const heading = document.createElement('h1');
heading.innerText = 'Hello world';
section.appendChild(heading);
```

This will add `h1` tag with the `Hello world` text.

---

append()

Very similar to `appendChild`. You can add several elements at once using `append`, but only one element using `appendChild`

```
const section = document.getElementById('section');
const heading1 = document.createElement('h1');
const heading2 = document.createElement('h1');
heading1.innerText = 'Hello world 1';
heading2.innerText = 'Hello world 2';
section.append(heading1, heading2);
```

prepend()

Similar to `append`, but will add elements before the first child.

```
const section = document.getElementById('section');
const heading = document.createElement('h1');
heading.innerText = 'Heading 1';
section.prepend(heading);
```

after() and before()

checkout mdn docs

Removing elements from DOM

There are different methods through which you can remove element from DOM

remove()

This methods removes element from DOM

```
<h1>Heading 1</h1>
<h2>Heading 2</h2>
<h3>Heading 3</h3>
```

```
const heading1 = document.querySelector('h1'
heading1.remove();
```

This will remove `h1` element from the DOM

---

removeChild()

Removes the child element

```
<div>
    <h1>Heading 1</h1>
    <h2>Heading 2</h2>
  </div>
```

```
const section = document.querySelector('div'
const heading1 = document.querySelector('h1'

section.removeChild(heading1);
```

This will remove `h1` tag that is the child of `div` tag.

Events

Events are the actions that happen in the html document, to which you can react to.

Some of the events are, user clicking a button, user resizes a window, user types in the input field, user submits a form, and so on.

Ways to add event listeners

inline

```
<button onclick="doSomething()">Click me</button>
```

```
function doSomething() {
  console.log('Hello world');
}
```

We have added a listener to the button. When the button is clicked, `doSomething` function will run.

event handler properties

```
<button>Click me</button>
```

```
const btn = document.querySelector("button")

btn.onclick = function() {
  console.log('Hello world');
}
```

As we have already discussed, when we select element using `querySelector`, it returns an object that represents that element.

On that object, there is a property called `onclick` which is currently null. You can override that property with a function which will run when you click a button.

event listener

another way to react to events is to add event listeners.

```
<button>Click me</button>
```

```
const btn = document.querySelector("button")

btn.addEventListener('click', function() {
  console.log('Hello world');
});
```

We have added an event listener to `click` event. `addEventListener` will take two arguments. The first is the event you want to listen to and the second argument, a function. This function will be called when click event is fired on that button.

Difference between the above methods

The major difference between `eventListener` and other methods is that you can not add multiple event listeners to the same element.

Lets say there are two functions that you want to run when user clicks a button. You can do that easily with `eventListeners` just by adding two listeners.

```
const btn = document.querySelector("button");

function one() {
  console.log('one');
}

function two() {
  console.log('two');
}

btn.addEventListener('click', one);
btn.addEventListener('click', two);
```

Now, when you click a button, both event listeners will run. and `one` and `two` will get printed on the console.

But when you try to add multiple event listeners with `onClick` property, it will not work.

```
const btn = document.querySelector("button");

function one() {
  console.log('one');
}

function two() {
  console.log('two');
}

btn.onclick = one;
btn.onclick = two;
```

Now, when you click a button, only `two` will print as the later line will override the first `onclick` assignment.

Practice

Add a button. When user clicks a button the background color of `body` will become red.

```html
<body>
  <button>Click me</button>
  <script src="./index.js"></script>
</body>
```

```javascript
const btn = document.querySelector('button')
const bodyEl = document.querySelector('body'

btn.addEventListener('click', function() {
  bodyEl.style.backgroundColor = 'red';
})
```

input event

You can listen to various events on input element. One such event is `input`. This event is fired when the value of an element is changed or we can say when the user types something in the input, this event fires.

```
<input type="text">
```

```
const inputEl = document.querySelector('input'

inputEl.addEventListener('input', function() {
  console.log('hello world');
})
```

Now, whenever user types something in the input field, `hello world` gets printed on the screen.

accessing the text that user types

Till now, we are just printing `hello world` for whatever user type. We can get access to the text typed by user using the following way.

The callback function that you pass to the event listener revceives an argument. You can call it anything, lets call it `event`.

This argument tells you everything about the event.

For example, if you want to know, on which element the event fired, you can use `target` property on event object.

```
const inputEl = document.querySelector('input');

inputEl.addEventListener('input', function(event) {
  console.log(event.target); // <input type="text">
})
```

To access the typed text, we access the `value` property on `event.target`

```
const inputEl = document.querySelector('input');

inputEl.addEventListener('input', function(event) {
  console.log(event.target.value);
})
```

whatever user type, will get printed on the console.

Practice

There is an input element. Whatever user type, will get shown as `h1` tag below the input element.

```html
<input type="text">
<h1></h1>
```

```javascript
const inputEl = document.querySelector('input'
const headingEl = document.querySelector('h1')

inputEl.addEventListener('input', function(eve
  headingEl.innerText = event.target.value;
})
```

submit event

```html
<form>
    <input type="text" name="username" />
    <input type="text" name="password" />
    <button type="submit">Submit</button>
</form>
```

```javascript
const formEl = document.querySelector("form"

formEl.addEventListener('submit', function()
  console.log('form submitted');
})
```

We can listen to `submit` event on form element. This event is fired when a form is submitted.

But when you submit on form, you do not see `form submitted` being printed on console. This is because, when you submit the form, the browser reloads and the console gets cleared.

You can prevent this default behaviour using the event parameter passed to the callback function.

```javascript
const formEl = document.querySelector("form");

formEl.addEventListener('submit', function(event) {
  event.preventDefault();
  console.log('form submitted');
})
```

If you call `event.preventDefault`, this will stop the default behaviour of the browser.

Now, when you submit the form, the browser will not refresh and you can see `form submitted` being printed on the console.

---

accessing elements of the form

There is a property on `form` element called `elements`, through which you can access the elements of form. Basically you can access the input elements and buttons inside the form. Once you get access to the input element, you can also get the text typed in those input elements though `value` property

```javascript
const formEl = document.querySelector("form");

formEl.addEventListener('submit', function(event) {
  event.preventDefault();
  console.log(formEl.elements[0].value);
  console.log(formEl.elements[1].value);
})
```

Note taking app

A note taking app where user can add a note and delete them by clicking on any note

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Note taking app</h1>

  <input type="text" placeholder="Title" id="title">
  <button id="add-button">Add</button>

  <ul id="list"></ul>

  <script src="./index.js"></script>
</body>
</html>
```

```
// Select input element
const titleEl = document.getElementById('title');

// select list element
const listEl = document.getElementById('list');

// select button element
const addButtonEl = document.getElementById('add-button');

// adding "click" event listener on button element
addButtonEl.addEventListener('click', function() {

  // getting the value of the input field
  const titleText = titleEl.value;

  // creating and 'li' element and adding inner text
  const li = document.createElement('li');
  li.innerText = titleText;

  // add the newly created li element to the list
  listEl.appendChild(li);

  // clear the input field after adding li to the list
  titleEl.value = '';
  // remove li when it is clicked
  li.addEventListener('click', function() {
    li.remove();
  })

})
```

Calculator app

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="./index.css">
  <title>Document</title>
</head>
<body>

  <div id="container">
    <input />

    <div class="button-grid">

      <button class="gray-button">C</button>
      <button class="gray-button">**</button>
      <button class="gray-button">%</button>
      <button class="yellow-btn">/</button>

      <button>7</button>
      <button>8</button>
      <button>9</button>
      <button class="yellow-btn">*</button>

      <button>4</button>
      <button>5</button>
      <button>6</button>
      <button class="yellow-btn"> - </button>


      <button>1</button>
      <button>2</button>
      <button>3</button>
      <button class="yellow-btn">+</button>

      <button class="zero">0</button>
      <button>.</button>
      <button class="yellow-btn"> = </button>
    </div>

  </div>

  <script src="./index.js"></script>
</body>
</html>
```

```css
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

body {
  width: 100vw;
  height: 100vh;
  background-color: #1c1c1c;
}

#container {
  width: 400px;
  margin: auto;
  height: 100vh;

  display: flex;
  flex-direction: column;
}

input {
  flex-grow: 1;
  background-color: #1c1c1c;
  border: 0;
  outline: none;
  font-size: 64px;
  color: white;
  text-align: right;
  padding: 0px 32px;
}

.button-grid {
  flex-grow: 2;
  display: grid;
  grid-template-columns: 1fr 1fr 1fr 1fr;
  grid-template-rows: 1fr 1fr 1fr 1fr 1fr;
  gap: 5px;
}

button {
  width: 80%;
  height: 80%;
  place-self: center;
  border-radius: 50%;
  border: 0;
  font-size: 24px;
  font-weight: 500;
  cursor: pointer;

  background-color: #505050;
  color: white;

  transition-property: transform;
  transition-duration: 200ms;
}

button:active {
  transform: scale(0.85);
}

.gray-button {
  background-color: #d4d4d2;
  color: #1c1c1c;
}

.yellow-btn {
  background-color: #ff9500;
  color: white;
}

.zero {
  grid-column: span 2;
  border-radius: 35%;
}
```

```
let buttons = document.querySelectorAll('button');
let input = document.querySelector('input');

for (let button of buttons) {
  button.addEventListener('click', function(event) {
    let btnText = event.target.innerText;
    if (btnText === 'C') {
      input.value = '';
    } else if (btnText === '=') {
      try {
        input.value = eval(input.value);
      } catch(e) {
        input.value = 'Invalid'
      }
    } else {
      input.value += btnText;
    }
  })
}
```

Canvas

Canvas is an HTML element. It helps in displaying graphics on the web page like lines, rectangles, circles, text and more complex designs.

You can give height and width attribute to this canvas element. It will be transparent. You can give border to the canvas in css.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>

  <style>
    canvas {
      border: 20px solid salmon;
    }
  </style>

</head>
<body>
  <canvas height="500px" width="500px"></canvas>
</body>
</html>
```

Inside this canvas, you can draw different shapes using javascript.

First, you have to create a `context`. It is an object which will give you access to all the methods that are needed to draw 2D shapes.

```
let canvas = document.querySelector('canvas');
let ctx = canvas.getContext('2d');
```

Consider `ctx` as a paint brush and with its help you are going to paint the canvas.

Lets say you want to draw a rectangle. The method to draw a rectangle is `fillRect`

```
ctx.fillRect(100, 120, 50, 60);

// ctx.fillRect(x, y, width, height)
```

This is going to draw a black rectangle of size 50px * 60px and at position 100px and 120px from top right.

If you want to draw a red rectangle, you have to dip the brush in red color before drawing. The same you have to do with `ctx`.

```
let canvas = document.querySelector('canvas');
let ctx = canvas.getContext('2d');

ctx.fillStyle = 'red';
ctx.fillRect(100, 120, 50, 60);
```

Instead of filling the rectnagle, you can just add border to the rectangle or we can say stroke the rectangle.

```
let canvas = document.querySelector('canvas');
let ctx = canvas.getContext('2d');
ctx.strokeRect(100, 120, 50, 60);
```

This will create a stroked rectangle. You can change the color of stroke too.

```
let canvas = document.querySelector('canvas');
let ctx = canvas.getContext('2d');

ctx.strokeStyle = 'red';
ctx.strokeRect(100, 120, 50, 60);
```

Drawing path

You can also draw paths

```
let canvas = document.querySelector('canvas');
let ctx = canvas.getContext('2d');

ctx.beginPath();
ctx.moveTo(10, 10);
ctx.lineTo(50, 10);
ctx.stroke();
```

To draw a path you begin a path. You move your brush to position 10px * 10px. Then you draw a line to 50px * 10px and stroke that line.

Question - Draw a triangle.

```
let canvas = document.querySelector('canvas');
let ctx = canvas.getContext('2d');

ctx.beginPath();
ctx.moveTo(100, 100);
ctx.lineTo(150, 150);
ctx.lineTo(50, 150);
ctx.lineTo(100, 100);
ctx.fill();
ctx.closePath();
```

Drawing text

You can also draw text.

```
let canvas = document.querySelector('canvas');
let ctx = canvas.getContext('2d');

ctx.font = '28px sans-serif';
ctx.fillStyle = 'red';
ctx.fillText('Hello world', 10, 50);
```

First you select the font, then fill some style and then add text.

Snake game

We are going to build a snake game inside a HTML canvas element.

Canavs is going to be of size 1000px * 600px. It could be of any size.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

    <style>
        canvas{
            border:20px solid lightgreen;
            background-color: black;
        }
    </style>

</head>
<body>

    <canvas id="canvas" height="600px" width="1000px"></canvas>
    <script src="./app.js"></script>
</body>
</html>
```

```
let canvas = document.querySelector('canvas');
let ctx = canvas.getContext('2d');

let boardHeight = 600;
let boardWidth = 1000;
let square = 50;

let snakeCells = [[0,0]];

let direction = 'right';

let gameOver = false;

let foodCell = generateRandomCell();

let score = 0;

document.addEventListener('keydown', function(event) {
  if (event.key === 'ArrowLeft') {
    direction = 'left';
  } else if (event.key === 'ArrowDown') {
    direction = 'down';
  } else if (event.key === 'ArrowRight') {
    direction = 'right';
  } else {
    direction = 'up';
  }
})

let intervalId = setInterval(function() {
  update();
  draw();
}, 100);

function update() {
  let headX = snakeCells[snakeCells.length - 1][0];
  let headY = snakeCells[snakeCells.length - 1][1];

  let newX;
  let newY;

  if (direction === 'right') {
    newX = headX + square;
    newY = headY;

    if (newX === boardWidth) {
      gameOver = true;
    }

  } else if (direction === 'left') {
    newX = headX - square;
    newY = headY;

    if (newX < 0) {
      gameOver = true;
    }

  } else if (direction === 'up') {
    newX = headX;
    newY = headY - square;

    if (newY < 0) {
      gameOver = true;
    }

  } else {
    newX = headX;
    newY = headY + square;

    if (newY === boardHeight) {
      gameOver = true;
    }
  }

  snakeCells.push([newX, newY]);

  if (foodCell[0] === headX && foodCell[1] === headY) {
    foodCell = generateRandomCell();
    score += 1;
  } else {
    snakeCells.shift();
  }
}

function draw() {

  if (gameOver === true) {
    clearInterval(intervalId);
    ctx.fillStyle = 'red';
    ctx.font = '40px sans-serif';
    ctx.fillText('Game over', 50, 150);
    return;
  }

  ctx.clearRect(0, 0, boardWidth, boardHeight);
  for (let cell of snakeCells) {
    ctx.fillStyle = 'yellow';
    ctx.fillRect(cell[0], cell[1], square, square);
    ctx.fillStyle = 'red';
    ctx.strokeRect(cell[0], cell[1], square, square);
  }

  ctx.fillStyle = 'red';
  ctx.fillRect(foodCell[0], foodCell[1], square, square);
  ctx.fillStyle = 'yellow';

  // draw score
  ctx.fillStyle = 'white';
  ctx.font = '20px sans-serif';
  ctx.fillText(`Score: ${score}`, 50, 50);
}

function generateRandomCell() {
  return [
    Math.round((Math.random()*(boardWidth - square))/square)*square,
    Math.round((Math.random()*(boardHeight - square))/square)*square
  ]
}
```