Working with the DOM and Web APIs

- **API**: An API (Application Programming Interface) is a set of rules and protocols that allow software applications to communicate with each other and exchange data efficiently.
- **Web API**: Web APIs are specifically designed for web applications. These types of APIs are often divided into two main categories: browser APIs and third-party APIs.
- **Browser APIs**: These APIs expose data from the browser. As a web developer, you can access and manipulate this data using JavaScript.
- **Third-Party APIs**: These are not built into the browser by default. You have to retrieve their code in some way. Usually, they will have detailed documentation explaining how to use their services. An example is the Google Maps API, which you can use to display interactive maps on your website.
- **DOM**: The DOM stands for Document Object Model. It's a programming interface that lets you interact with HTML documents. With the DOM, you can add, modify, or delete elements on a webpage. The root of the DOM tree is the html element. It's the top-level container for all the content of an HTML document. All other nodes are descendants of this root node. Then, below the root node, we find other nodes in the hierarchy. A parent node is an element that contains other elements. A child node is an element that is contained within another element.
- **navigator Interface**: This provides information about the browser environment, such as the user agent string, the platform, and the version of the browser. A user agent string is a text string that identifies the browser and operating system being used.
- **window Interface**: This represents the browser window that contains the DOM document. It provides methods and properties for interacting with the browser window, such as resizing the window, opening new windows, and navigating to different URLs.

Working with the querySelector(), querySelectorAll() and getElementById() Methods

- **getElementById() Method**: This method is used to get an object that represents the HTML element with the specified id. Remember that ids must be unique in every HTML document, so this method will only return one Element object.

Example Code

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

Example Code

```
const container = document.getElementById("container");
```

- **querySelector() Method**: This method is used to get the first element in the HTML document that matches the CSS selector passed as an argument.

Example Code

```
<section class="section"></section>
```

Example Code

```
const section = document.querySelector(".section");
```

- **querySelectorAll() Method**: You can use this method to get a list of all the DOM elements that match a specific CSS selector.

Example Code

```
<ul class="ingredients">
  <li>Sugar</li>
  <li>Milk</li>
  <li>Eggs</li>
</ul>
```

Example Code

```
const ingredients = document.querySelectorAll('ul.ingredients li');
```

Working with the innerText(), innerHTML(), createElement() and textContent() Methods

- **innerHTML Property**: This is a property of the Element that is used to set or update parts of the HTML markup.

Example Code

```
<div id="container">
  <!-- Add new elements here -->
</div>
```

Example Code

```
const container = document.getElementById("container");
```

container.innerHTML = '
Cheese
Tomato
';

- **createElement Method**: This is used to create an HTML element.

Example Code

```
const img = document.createElement("img");
```

- **innerText**: This represents the visible text content of the HTML element and its descendants.

Example Code

```
<div id="container">
  <p>Hello, World!</p>
  <p>I'm learning JavaScript</p>
</div>
```

Example Code

```
const container = document.getElementById("container");
console.log(container.innerText);
```

- **textContent**: This returns the plain text content of an element, including all the text within its descendants.

Example Code

```
<div id="container">
  <p>Hello, World!</p>
  <p>I'm learning JavaScript</p>
</div>
```

Example Code

```
const container = document.getElementById("container");
console.log(container.textContent);
```

Working with the appendChild() and removeChild() Methods

- **appendChild() Method**: This method is used to add a node to the end of the list of children of a specified parent node.

Example Code

```
<ul id="desserts">
  <li>Cake</li>
  <li>Pie</li>
</ul>
```

Example Code

```
const dessertsList = document.getElementById("desserts");
const listItem = document.createElement("li");

listItem.textContent = "Cookies";
dessertsList.appendChild(listItem);
```

- **removeChild() Method**: This method is used to remove a node from the DOM.

Example Code

```
<section id="example-section">
  <h2>Example sub heading</h2>
  <p>first paragraph</p>
  <p>second paragraph</p>
</section>
```

Example Code

```
const sectionEl = document.getElementById("example-section");
const lastParagraph = document.querySelector("#example-section p:last-of-type");

sectionEl.removeChild(lastParagraph);
```

Work with the setAttribute() Method

- **Definition**: This method is used to set the attribute for a given element. If the attribute already exists, then the value is updated. Otherwise, a new attribute is added with a value.

Example Code

```
<p id="para">I am a paragraph</p>
```

Example Code

```
const para = document.getElementById("para");

para.setAttribute("class", "my-class");
```

Event Object

- **Definition**: The Event object is a payload that triggers when a user interacts with your web page in some way. These interactions can be anything from clicking on a button or focusing an input to shaking their mobile device. All Event objects will have the type property. This property reveals the type of event that triggered the payload, such as keydown or click. These values will correspond to the same values you might pass to addEventListener(), where you can capture and utilize the Event object.

addEventListener() and removeEventListener() Methods

- **addEventListener Method**: This method is used to listen for events. It takes two arguments: the event you want to listen for and a function that will be called when the event occurs. Some common examples of events would be click events, input events, and change events.

Example Code

```
const btn = document.getElementById("btn");


btn.addEventListener("click", () => alert("You clicked the button"));
```

- **removeEventListener() Method**: This method is used to remove an event listener that was previously added to an element using the addEventListener() method. This is useful when you want to stop listening for a particular event on an element.

Example Code

```
const bodyEl = document.querySelector("body");

const para = document.getElementById("para");

const btn = document.getElementById("btn");
```

```
let isBgColorGrey = true;


function toggleBgColor() {
  bodyEl.style.backgroundColor = isBgColorGrey ? "blue" : "grey";
  isBgColorGrey = !isBgColorGrey;
}


btn.addEventListener("click", toggleBgColor);


para.addEventListener("mouseover", () => {
  btn.removeEventListener("click", toggleBgColor);
});
```

- **Inline Event Handlers**: Inline event handlers are special attributes on an HTML element that are used to execute JavaScript code when an event occurs. In modern JavaScript, inline event handlers are not considered best practice. It is preferred to use the addEventListener method instead.

Example Code

```
<button onclick="alert('Hello World!')">Show alert</button>
```

The Change Event

- **Definition**: The change event is a special event which is fired when the user modifies the value of certain input elements. Examples would include when a checkbox or a radio button is ticked. Or when the user makes a selection from something like a date picker or dropdown menu.

Example Code

```
<label>
  Choose a programming language:
  <select class="language" name="language">
```

```
  <option value="">---Select One---</option>
  <option value="JavaScript">JavaScript</option>
  <option value="Python">Python</option>
  <option value="C++">C++</option>
 </select>
</label>

<p class="result"></p>
```

Example Code

```
const selectEl = document.querySelector(".language");
const result = document.querySelector(".result");

selectEl.addEventListener("change", (e) => {
 result.textContent = `You enjoy programming in ${e.target.value}.`;
});
```

Event Bubbling

- **Definition**: Event bubbling, or propagation, refers to how an event "bubbles up" to parent objects when triggered.
- **stopPropagation() Method**: This method prevents further propagation for an event.

Event Delegation

- **Definition**: Event delegation is the process of listening to events that have bubbled up to a parent, rather than handling them directly on the element that triggered them.

DOMContentLoaded

- **Definition**: The DOMContentLoaded event is fired when everything in the HTML document has been loaded and parsed. If you have external stylesheets, or images, the DOMContentLoaded event will not wait for those to be loaded. It will only wait for the HTML to be loaded.

Working with style and classList

- **Element.style Property**: This property is a read-only property that represents the inline style of an element. You can use this property to get or set the style of an element.

Example Code

```
const paraEl = document.getElementById("para");

paraEl.style.color = "red";
```

- **Element.classList Property**: This property is a read-only property that can be used to add, remove, or toggle classes on an element.

Example Code

```
// Example adding a class

const paraEl = document.getElementById("para");

paraEl.classList.add("highlight");


// Example removing a class

paraEl.classList.remove("blue-background");


// Example toggling a class

const menu = document.getElementById("menu");

const toggleBtn = document.getElementById("toggle-btn");


toggleBtn.addEventListener("click", () => menu.classList.toggle("show"));
```

Working with the setTimeout() and setInterval() Methods

- **setTimeout() Method**: This method lets you delay an action for a specified time.

Example Code

```
setTimeout(() => {
 console.log('This runs after 3 seconds');
```

}, 3000);

- **setInterval() Method**: This method keeps runs a piece of code repeatedly at a set interval. Since setInterval() keeps executing the provided function at the specified interval, you might want to stop it. For this, you have to use the clearInterval() method.

Example Code

```
setInterval(() => {
 console.log('This runs every 2 seconds');
}, 2000);


// Example using clearInterval
const intervalID = setInterval(() => {
 console.log('This will stop after 5 seconds');
}, 1000);


setTimeout(() => {
 clearInterval(intervalID);
}, 5000);
```

The requestAnimationFrame() Method

- **Definition**: This method allows you to schedule the next step of your animation before the next screen repaint, resulting in a fluid and visually appealing experience. The next screen repaint refers to the moment when the browser refreshes the visual display of the web page. This happens multiple times per second, typically around 60 times (or 60 frames per second) on most displays.

Example Code

```
function animate() {
 // Update the animation...
 // for example, move an element, change a style, and more.
```

```
 update();
 // Request the next frame
 requestAnimationFrame(animate);
}
```

Web Animations API

- **Definition**: The Web Animations API lets you create and control animations directly inside JavaScript.

Example Code

```
const square = document.querySelector('#square');


const animation = square.animate(
 [{ transform: 'translateX(0px)' }, { transform: 'translateX(100px)' }],
 {
  duration: 2000, // makes animation lasts 2 seconds
  iterations: Infinity, // loops indefinitely
  direction: 'alternate', // moves back and forth
  easing: 'ease-in-out', // smooth easing
 }
);
```

The Canvas API

- **Definition**: The Canvas API is a powerful tool that lets you manipulate graphics right inside your JavaScript file. To work with the Canvas API, you first need to provide a canvas element in HTML. This element acts as a drawing surface you can manipulate with the instance methods and properties of the interfaces in the Canvas API. This API has interfaces like HTMLCanvasElement, CanvasRenderingContext2D, CanvasGradient, CanvasPattern, and TextMetrics which contain methods and properties you can use to create graphics in your JavaScript file.

Example Code

```
<canvas id="my-canvas" width="400" height="400"></canvas>
```

Example Code

```
const canvas = document.getElementById('my-canvas');


// Access the drawing context of the canvas.
// "2d" allows you to draw in two dimensions
const ctx = canvas.getContext('2d');


// Set the background color
ctx.fillStyle = 'crimson';


// Draw a rectangle
ctx.fillRect(1, 1, 150, 100);
```

Opening and Closing Dialogs and Modals with JavaScript

- **Modal and Dialog Definitions**: Dialogs let you display important information or actions to users. With the HTML built-in dialog element, you can easily create these dialogs (both modal and non-modal dialogs) in your web apps. A modal dialog is a type of dialog that forces the user to interact with it before they can access the rest of the application or webpage. In contrast, a non-modal dialog allows the user to continue interacting with other parts of the page or application even when the dialog is open. It doesn't prevent access to the rest of the content.
- **showModal() Method**: This method is used to open a modal.

Example Code

```
<dialog id="my-modal">
  <p>This is a modal dialog.</p>
</dialog>
<button id="open-modal">Open Modal Dialog</button>
```

Example Code

```
const dialog = document.getElementById('my-modal');
const openButton = document.getElementById('open-modal');

openButton.addEventListener('click', () => {
  dialog.showModal();
});
```

- **close() Method**: This method is used to close the modal.

Example Code

```
<dialog id="my-modal">
  <p>This is a modal dialog.</p>
  <button id="close-modal">Close Modal</button>
</dialog>
<button id="open-modal">Open Modal Dialog</button>
```

Example Code

```
const dialog = document.getElementById('my-modal');
const openButton = document.getElementById('open-modal');
const closeButton = document.getElementById('close-modal');

openButton.addEventListener('click', () => {
  dialog.show();
});

closeButton.addEventListener('click', () => {
  dialog.close();
});
```