WebAPI

DOM, Client-side storage APIs

VIVIFY
ACADEMY

DOM

VIVIFY
ACADEMY

DOM

- It represents the page so that programs can change the document structure, style and content.
- The DOM represents the document as nodes and objects. That way, programming languages can connect to the page.
- DOM is represented as a tree of Objects, and when a web page is loaded, the browser creates a Document Object Model of the page.

Core Interfaces in the DOM

This section lists some of the most commonly-used interfaces in the DOM. The idea is not to describe what these APIs do here but to give you an idea of the sorts of methods and properties you will see very often as you use the DOM.

```
document.getElementById(id)
document.getElementsByTagName(name)
document.createElement(name)
document.body.appendChild(node)
element.innerHTML
element.style.left
element.setAttribute()
element.getAttribute()
element.addEventListener()
window.onload
window.scrollTo()
```

Window - The Browser Object Model

- The Browser Object Model (BOM) allows JavaScript to "talk to" the browser.
- The `window` object is supported by all browsers. It represents the browser's window.
- All global JavaScript objects, functions, and variables automatically become members of the `window` object.

DOM Events

Events

The Event interface represents any event which takes place in the DOM; some are user-generated (such as mouse or keyboard events), while others are generated by APIs (such as events that indicate an animation has finished running, a video has been paused, and so forth).

A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.

Registering event listeners

```
element.addEventListener(event, callback);
```

The first parameter is the type of the event (like "click" or "mousedown").

The second parameter is the function we want to call when the event occurs.

Removing event listeners

The removeEventListener() method removes event handlers that have been attached with the addEventListener() method:

```
element.removeEventListener("click", callback);
```

Example

HTML Part

```
<button id="myBtn">Hey</button>
```

JavaScript Part

```
var myBtn = document.querySelector('#myBtn');

myBtn.addEventListener('click', function() {

    alert('Hey');

});
```

Exercise

1. Create a div with id ‘myDiv’ inside HTML and then using `querySelector` select that div and change text inside div into your full name using `innerText` property
2. Create a button with id ‘myBtn’ inside HTML, using `document.getElementById` select ‘myBtn’ and add event listener on that button which will show alert with your full name

Client-Side Storage

Client-Side Storage

- Client-Side Storage allows you to store data in user browser. Storage is **sandboxed.**
- **Sandboxed** means that when you store data on url www.example.com that only www.example.com can access to those data again.

LocalStorage

- LocalStorage allows you to store data in user browser and it won’t expire unless you or user removes it.
- With LocalStorage you can store only strings but if you need to store an object you need:
  - to use **JSON.stringify** when you are saving the object and
  - **JSON.parse** when you are retrieving object

SessionStorage

- SessionStorage is almost same as the LocalStorage only difference is that data in SessionStorage are cleared when a page session ends. **If you open a new tab or a new window, you will have a new session.**

Example

```
var user = {
  name: 'Jake'
};
localStorage.setItem('user', JSON.stringify(user));

...

var storageUser = JSON.parse(localStorage.getItem('user'));
console.log(storageUser);// { name: 'Jake' }
```

Exercise

1. Create object **user** and set your first and last name as properties and then save that user inside of a **localStorage**, then read it and do a console.log of that user.