JavaScript

JavaScript is used for client-side scripting.

Use of JavaScript

- **Front-End Interactivity**: Web development is only made better by the increased interactivity and features that JavaScript offers.
- **Web Applications**: Web applications are similar to websites, but instead, they get packaged into a neat little box, which improves control over security and more.
- **Browser Games**: The modern web browser has come a long way; developers even make robust games that function in a browser.
- **Back End Web Development**: Web development has come a long way, and now JavaScript is so robust it can even be used to manage the back end of websites and web applications.

Ways to include JavaScript

1. Between <script> and </script> tags inside head or body :

   Example : <script>
   document.getElementById("demo").innerHTML = "My First JavaScript";
   </script>

2. External JavaScript :

   It is advantageous to add it before the body tag is over.
   Example :
   <script src="myScript.js"></script>

JavaScript Syntax

- The JavaScript syntax defines two types of values:
- Fixed values
- Variable values

Fixed values are called Literals.
Variable values are called Variables.

- JavaScript syntax is the set of rules, how JavaScript programs are constructed: // Variable creation :

```
var x;
let y;

// Variable manipulation:
```

x = 5; y = 6;
let z = x + y;

Events in JavaScript

- Events are **"things"** that happen to HTML elements.
- When JavaScript is used in HTML pages, JavaScript can **"react"** on these events.
- Some Basic events are as follow :

click	An element is clicked on
contextmenu	An element is right-clicked to open a context menu
copy	The content of an element is copied
cut	The content of an element is cutted
dblclick	An element is double-clicked
drag	An element is being dragged
dragend	Dragging of an element has ended
dragenter	A dragged element enters the drop target
dragleave	A dragged element leaves the drop target
dragover	A dragged element is over the drop target
dragstart	Dragging of an element has started
drop	A dragged element is dropped on the target
durationchange	The duration of a media is changed
ended	A media has reach the end ("thanks for listening")
error	An error has occurred while loading a file
focus	An element gets focus

focusin	An element is about to get focus
focusout	An element is about to lose focus
fullscreenchange	An element is displayed in fullscreen mode
fullscreenerror	An element can not be displayed in fullscreen mode
hashchange	There has been changes to the anchor part of a URL
input	An element gets user input
invalid	An element is invalid
keydown	A key is down
keypress	A key is pressed
keyup	A key is released
load	An object has loaded
loadeddata	Media data is loaded
loadedmetadata	Meta data (like dimensions and duration) are loaded
loadstart	The browser starts looking for the specified media
message	A message is received through the event source
mousedown	The mouse button is pressed over an element
mouseenter	The pointer is moved onto an element

Basic validation

- JavaScript has an Object called RegExp to work with Regular Expressions. The syntax is RegExp (pattern, modifier). Where pattern indicates the pattern you are searching for, and the modifier tells how to go about the search/(pattern matching) such as whether will it be in the global scope or will it be case sensitive or not.
- Methods of the RegExp object includes test(), exec(), and toString(). test() returns true or false based on the success or failure of the search. exec() will return the matched text. toString() when applied to a regular expression object will return the pattern in a string.
- Example

  var searchPattern = new RegExp("wor");
  searchPattern.test("hello world"));

Local storage

- Local storage is a storage given to user client with no expiration date which means data is not deleted when browser is closed.
- It provides maximum 10MB.
- Syntax

1. **Save Data to Local Storage**

   localStorage.setItem(*key*, *value*);
2. **Read Data from Local Storage**

   let lastname = localStorage.getItem(*key*);
3. **Remove Data from Local Storage**

   localStorage.removeItem(*key*);
4. **Clear Local Storage**

   localStorage.clear();

Session storage

- Session storage is a storage that is given to user for one session only which means session storage is deleted when tab is closed or tab is changed but session storage duplicates with duplication of tab.
- It also has maximum storage of 10MB.
- Syntax

  1. **Save Data to Session Storage**

     sessionStorage.setItem("key", "value");
  2. **Read Data from Session Storage**

     let lastname = sessionStorage.getItem("key");
  3. **Remove Data from Session Storage**

     sessionStorage.removeItem("key");
  4. **Clear session Storage**

     sessionStorage.clear();

Cookies

- Cookies are data, stored in small text files, on your computer.
- When a user visits a web page, his/her name can be stored in a cookie.
- Next time the user visits the page, the cookie "remembers" his/her name.
- Cookies are saved in name-value pairs like:
  username = John Doe

- When a browser requests a web page from a server, cookies belonging to the page are added to the request. This way the server gets the necessary data to "remember" information about users.
- Example
    1. Create cookie
        document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";
            - where expires specifies expiration time in UTC by default cookie is deletd when browser is closed and path specifies where to store cookie by default it is stored in current page
    2. Read cookie
        let x=document.cookie;
    3. Delete cookie
        document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";

Cache

- Caching involves storing frequently accessed data in a fast-access storage medium, such as memory, so that subsequent requests for the same data can be served quickly without having to go through the full process of retrieving it from the source.
- Syntax
    1. Open cache
        ```
        caches.open('my-cache').then(function(cache) {
        // Cache opened successfully
        }).catch(function(error) {
        // Failed to open cache
        });
        ```
    2. Add to cache
        ```
        caches.open("my-cache").then((cache) => {
          cache
            .add("/api/data")
            .then(() => console.log("Data added to cache."))
            .catch((error) => console.error("Error adding data to cache:", error));
        });
        ```
    3. Add all to cache
        ```
        const urls = ["/api/data1", "/api/data2", "/api/data3"];
        caches.open("my-cache").then((cache) => {
          cache
        ```

```
        .addAll(urls)
        .then(() => console.log("Data added to cache."))
        .catch((error) => console.error("Error adding data to
    cache:", error));
    });
```

4. Remove from cache

```
caches
  .open("my-cache")
  .then(function (cache) {
    cache.delete("/path/to/resource").then(function
     (isDeleted) {
      if (isDeleted) {
        // Resource deleted successfully
      } else {
        // Resource not found in cache
      }
  .catch(function (error) {
    // Failed to delete resource from cache
  });
```

Classes in JavaScript

- ECMAScript 2015, also known as ES6, introduced JavaScript Classes.
- JavaScript Classes are templates for JavaScript Objects.
- Syntax

```
class ClassName {
 constructor() {
        ...
 }
}
```

- Example

```
class CreateUSer {
  constructor(firstName, lastName, age) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
  }
  about() {
        return `${this.firstName} is ${this.age} years old`;
  }
```

```
  is18() {
        return this.age >= 18;
  }
}
const user1 = new CreateUSer('abc', 'cde', 10);
const user2 = new CreateUSer('qwdbnb', 'scjhjkd', 33);
console.log(user1.about());
```

Possible ways to implement class

- To create a class inheritance, use the extends keyword.
- A class created with a class inheritance inherits all the methods from parent class.
- Example

```
class Animal {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
  eat() {
    return `${this.name} is eating`;
  }
  isSuperCute() {
    return this.age <= 1;
  }
}
class Dog extends Animal {
  constructor(name, age, speed) {
    super(name, age);
    this.speed = speed;
  }
  run() {
    return `${this.name}'s speed is ${this.speed} kmph`;
  }
}
const doggo = new Dog('doggo', 1, 34);
```

Static class and properties

- Static properties and methods of a class can be used without instance of the class directly.
- Can not modify static properties or methods.

- Example

```
class Cars{
   constructor(name,speed,color){
      this.name=name;
      this.speed=speed;
      this.color=color;
   }
   static info(){
      return "This is cars class";
   }
   static desc="Desc static property"
   isRed(){
      return color=="red";
   }
}
console.log(Cars.info());
console.log(Cars.desc);
```

Arrow functions

- Arrow functions were introduced in ES6.
- Arrow functions allow us to write shorter function syntax.
- Example

```
hello = () => {
  return "Hello World!";
}
```

or

```
hello = () => "Hello World!";
```

Import

- JavaScript modules allow you to break up your code into separate files.
- This makes it easier to maintain a code-base.
- Modules are imported from external files with the import statement.
- Modules also rely on type="module" in the <script> tag.
- Example

```
import { name, age } from "./person.js";
```

Export

- Modules with functions or variables can be stored in any external file.
- There are two types of exports: Named Exports and Default Exports.

1. **Named Exports**
   - We can create named exports two ways. In-line individually, or all at once at the bottom.
   - In-line individually:

```
export const name = "Jesse";
export const age = 40;
```

   - All at once at the bottom:

```
const name = "Jesse";
const age = 40;
export {name, age};
```

2. **Default Exports**
   - We can only have one default export in a file.
   - Example

```
const message = () => {
const name = "Jesse";
const age = 40;
return name + ' is ' + age + 'years old.';
};
export default message;
```

async and await

- *async and await make promises easier to write.*
- **async** makes a function return a Promise.
- **await** makes a function wait for a Promise.
- Example

```
async function myDisplay() {
  let myPromise = new Promise(function(resolve) {
    setTimeout(function() {resolve("I love You !!");}, 3000);
  });
  document.getElementById("demo").innerHTML = await my
Promise;
}
myDisplay();
```