DOM and Event Fundamentals | Cheat Sheet

1. JavaScript Variables

1.1 Variable Declaration

Variables are like containers for storing values. We can create a variable using the let keyword.

let message;

1.2 Assigning a Value to the Variable

We can put data into a variable using an assignment operator (=).

Example:

let message = 'Hello Rahul';

(OR)

let message;

message = 'Hello Rahul';

Note: Printing a variable without assigning a value will give the output undefined.

2. Document Object Model (DOM)

The DOM is the structured representation of the HTML document created by the browser. It allows JavaScript to manipulate, structure, and style your website.

2.1 Document Object

It is the entry point of the DOM. For accessing any HTML Element, you should always start with accessing the document object first.

2.2 HTML DOM Tree

The DOM tree represents an HTML document as nodes. Each node is referred to as an Object.

2.3 Methods

2.3.1 getElementById

The getElementById() method helps to select the HTML Element with a specific ID.

Example:

```
console.log(document.getElementById("headingElement"))
```

2.4 Properties

2.4.1 textContent

To manipulate the text within the HTML Element, we use textContent Property.

2.4.2 style

The style property is used to get or set a specific style of an HTML Element using different CSS properties.

Use Camel Case naming convention (starting letter of each word should be in the upper case except for the first word) for naming the Style Object Properties.

For example: color, fontFamily, backgroundColor, etc.

2.5 Events

Events are the actions by which the user or browser interacts with the HTML Elements. Actions can be anything like clicking a button, pressing keyboard keys, scrolling the page, etc.

2.5.1 onclick Event

The onclick event occurs when the user clicks on an HTML Element. We will give the name of the function as a value for the HTML onclick attribute.

Example:

```
<body>
  <h1 id="headingElement">Web Technologies</h1>
  <button onclick="manipulateStyles()">Change Heading</button>
</body>
```

JS:

```
function manipulateStyles() {
  document.getElementById("headingElement").textContent = "4.O Technologies";
  document.getElementById("headingElement").style.color = "blue";
}
```

Primitive Types & Conditionals | Cheat Sheet

1. JavaScript Values

Basically In JavaScript values are of two categories.

- Primitive Types

- Reference Types

1.1 Primitive Types

- Number
- Boolean
- String
- Undefined, etc.

Primitive Type	Description
Number	All the numbers are of Number type.
Boolean	Boolean values are either true or false.
String	String is a stream of characters. The String should be enclosed with Single quotes, Double quotes, or Backticks.
Undefined	If a value is not assigned to the variable, then it takes `undefined` as its value. In JS, `undefined` refers to the value that is not being assigned.

1.2 Operators

1.2.1 typeof()

The **typeof()** operator is used to find the type of value.

```
let a = 900;

let b = 9.2;

console.log(typeof(a));  // number

console.log(typeof(b));  // number
```

2. Converting String to a Number

In JavaScript, when we combine the number and string, it results in a string.

The parseInt() function accepts a string and converts it into an integer.

Example:

```
let a = '20';

console.log(typeof(a));  // string

let b = parseInt(a);

console.log(typeof(b));  // number
```

3. Conditional Statements

The Conditional Statement allows you to execute a block of code only when a specific condition is true.

If...Else Statement:

Syntax:

```
if (conditionA) {
  Block1;
}
else if (conditionB) {
  Block2;
}
else {
  Block3;
}
```

Input Element and Math Functions | Cheat Sheet

1. Math Functions

1.1 Math.random()

The **Math.random()** function returns a random number (float value) in range 0 to less than 1 (**0 <= randomNumber < 1**).

Example:

```
console.log(Math.random());
```

1.2 Math.ceil()

The **Math.ceil()** function always rounds a number up to the next largest integer.

Example: console.log(Math.ceil(95.906698007537561)); // 96

2. HTML Elements

2.1 HTML Input Element

The HTML Input element creates interactive controls to **accept** the data from the user.

There are different types of inputs.

- Text
- Password
- Radio
- Date
- Checkbox

2.1.1 Text Input

Exmple: <input type="text" />

Note: Default type for the HTML input element is text.

2.1.2 Password Input

It provides a way for the user to enter a password securely.

Syntax: <input type="password" />

3. DOM Properties

3.1 Value

We can use the **value** property to get the value of the HTML input Element.

Syntax: document.getElementById("inputElement").value;

4. Comparison Operator

4.1 Loose equal to vs Strict equal to (== vs ===)

Loose equal to (==): Loose equality compares two values for equality but doesn't compare types of values.

Strict equal to (===): Strict equality compares two values for equality including types of values.

Example: console.log(2 == '2'); // true

console.log(2 === '2'); // false

Arrays & More DOM Manipulations | Cheat Sheet

1. Data Structures

Data Structures allow us to store and organize data efficiently. This makes us access and performs operations on the data smoothly.

In JavaScript, we have built-in Data Structures like,

- Arrays
- Objects
- Maps
- Sets

2. Array

An Array holds an ordered sequence of items.

2.1 Creating an Array

```
let myArray = [5, "six", 2, 8.2];

console.log(myArray);  // [5, "six", 2, 8.2]
```

2.2 Accessing an Array Item

Exmple:

```
let myArray = [5, "six", 2, 8.2];

console.log(myArray[0]);  // 5

console.log(myArray[1]);  // six
```

2.3 Modifying an Array Item

Example:

```
let myArray = [5, "six", 2, 8.2];

myArray[1] = 6;

console.log(myArray);  // [5, 6, 2, 8.2]
```

2.4 Finding Array Length

The **array.length** is used to find the number of items in the array.

Example:

```
let myArray = [5, "six", 2, 8.2];

let lengthOfArray = myArray.length;

console.log(lengthOfArray);  // 4
```

2.5 Array Methods

2.5.1 push()

The **push()** method adds new items to the end of the array.

Example:

```
let myArray = [5, "six", 2, 8.2];

myArray.push(true);

console.log(myArray);  // [5, "six", 2, 8.2, true]
```

2.5.2 pop()

The **pop()** method removes the last item of an array and returns that item.

Example:

```
let myArray = [5, "six", 2, 8.2];

let lastItem = myArray.pop();

console.log(myArray);  // [5, "six", 2]

console.log(lastItem);  // 8.2
```

3. Functions

3.1 Function Declaration

```
function showMessage() {
  console.log("Hello");
}

showMessage();
```

3.2 Function Expression

There is another syntax for creating a function which is called Function Expression.

Example:

```
let showMessage = function() {
  console.log("Hello");
};
showMessage();
```

4. More DOM Manipulations

4.1 Creating an HTML Element - createElement()

Example:

```
let h1Element = document.createElement("h1");
h1Element.textContent = "Web Technologies";
console.log(h1Element);  // <h1>Web Technologies</h1>
```

4.2 Appending to an HTML Element - appendChild()

Appending to Document Body Object:

```
document.body.appendChild(h1Element);
```

Appending to Existing Container Element:

```
let containerElement = document.getElementById("myContainer");
containerElement.appendChild(h1Element);
```

4.3 Adding Event Listeners Dynamically

```
let btnElement = document.createElement("button");
btnElement.textContent = "Change Heading";
document.getElementById("myContainer").appendChild(btnElement);
btnElement.onclick = function(){
  console.log("click event triggered");
};
```

4.4 Providing Class Names Dynamically - classList.add()

```
btnElement.onclick = function(){
```

```
  h1Element.textContent = "4.0 Technologies";
  h1Element.classList.add("heading");
  console.log(h1Element);
};
```

//CSS

```
.heading {
  color: blue;
  font-family: "Caveat";
  font-size: 40px;
  text-decoration: underline;
}
```

4.5 Removing Class Names Dynamically - classList.remove()

Example:

```
let removeStylesBtnElement = document.createElement("button");
removeStylesBtnElement.textContent = "Remove Styles";
document.getElementById("myContainer").appendChild(removeStylesBtnElement);
removeStylesBtnElement.onclick = function(){
  h1Element.classList.remove("heading");
};
```

Objects | Cheat Sheet

Object

An Object is a collection of properties.

A property is an association between a name (or key) and a value.

For example, a person has a name, age, city, etc. These are the properties of the person.

Key	Value
firstName	Rahul
lastName	Attuluri
age	28
city	Delhi

1. Creating an Object

We can add properties into {} as key: value pairs.

Example:

```
let person = {
  firstName: "Rahul",
  lastName: "Attuluri",
  age: 28,
};
console.log(person);   // Object {firstName: "Rahul", lastName: "Attuluri", age: 28}
```

1.1 Identifiers

A valid Identifier should follow the below rules:

It can contain alphanumeric characters, _ and $.

It cannot start with a number.

Valid Identifiers:

```
firstName;
$firstName;
_firstName;
firstName12;
```

Invalid Identifiers:

```
12firstName;
```

firstName 12;

To use an Invalid identifier as a key, we have to specify it in quotes.

Example:

```
let person = {
  firstName: "Rahul",
  lastName: "Attuluri",
  age: 28,
  "1": "value1",
  "my choice": "value2",
};
```

console.log(person); // **Object {1: "value1", firstName: "Rahul", lastName: "Attuluri", age: 28, my choice: "value2"}**

2. Accessing Object Properties

2.1 Dot Notation

```
let person = {
  firstName: "Rahul",
  lastName: "Attuluri",
  age: 28,
  "1": "value1",
  "my choice": "value2",
};
```

console.log(person.firstName); // **Rahul**

Use Dot notation when the key is a valid Identifier.

2.2 Bracket Notation

```
let person = {
  firstName: "Rahul",
  lastName: "Attuluri",
  age: 28,
```

```
  "1": "value1",
  "my choice": "value2",
};
console.log(person["firstName"]);  // Rahul
```

2.3 Accessing Non-existent Properties

Dot Notation:

```
let person = {
  firstName: "Rahul",
  lastName: "Attuluri",
  age: 28,
  "1": "value1",
  "my choice": "value2",
};
console.log(person.gender);  // undefined
```

Bracket Notation:

```
let person = {
  firstName: "Rahul",
  lastName: "Attuluri",
  age: 28,
  "1": "value1",
  "my choice": "value2",
};
console.log(person["gender"]);  // undefined
```

2.4 Variable as a Key

```
let person = {
  firstName: "Rahul",
  lastName: "Attuluri",
```

```
  age: 28,
};
let a = "firstName";
console.log(person[a]);  // Rahul
console.log(person.a);  // undefined
```

2.5 Object Destructuring

To unpack properties from Objects, we use Object Destructuring. The variable name should match with the key of an object.

Example:

```
let person = {
  firstName: "Rahul",
  lastName: "Attuluri",
  age: 28,
};
let { gender, age } = person;
console.log(gender);  // undefined
console.log(age);  // 28
```

3. Modifying Objects

3.1 Modifying Object Property

Dot Notation:

```
let person = {
  firstName: "Rahul",
  lastName: "Attuluri",
  age: 28,
};
person.firstName = "Abhi";
console.log(person.firstName);  // Abhi
```

Bracket Notation:

```
let person = {
  firstName: "Rahul",
  lastName: "Attuluri",
  age: 28,
};
person["firstName"] = "Abhi";
console.log(person["firstName"]);  // Abhi
```

3.2 Adding Object Property

Dot Notation:

```
let person = {
  firstName: "Rahul",
  lastName: "Attuluri",
  age: 28,
};
person.gender = "Male";
console.log(person);  // Object {firstName: "Rahul", lastName: "Attuluri", age: 28, gender: "Male"}
```

Bracket Notation:

```
let person = {
  firstName: "Rahul",
  lastName: "Attuluri",
  age: 28,
};
person["gender"] = "Male";
console.log(person);  // Object {firstName: "Rahul", lastName: "Attuluri", age: 28, gender: "Male"}
```

4. Property Value

The Value of Object Property can be

- Function

- Array
- Object

4.1 Function as a Value

```
let person = {
  firstName: "Rahul",
  lastName: "Attuluri",
  age: 28,
  run: function () {
    console.log("Start Running.");
  },
};
person.run();  // Start Running.
```

Methods:

A JavaScript method is a property containing a function definition.

For example, in **document.createElement();,** the document is an Object, **createElement** is a key and **createElement()** is a Method.

4.2 Array as a Value

```
let person = {
  firstName: "Rahul",
  lastName: "Attuluri",
  age: 28,
  habits: ["Playing Chess", "Singing"],
};
console.log(person.habits);  // ["Playing Chess", "Singing"]
console.log(person.habits[0]);  // Playing Chess
console.log(person["habits"][1]);  // Singing
```

4.3 Object as a Value

```
let person = {
  firstName: "Rahul",
  lastName: "Attuluri",
  age: 28,
  habits: ["Playing Chess", "Singing", "Dancing"],
  car: {
    name: "Audi",
    model: "A6",
    color: "White",
  },
};
console.log(person.car.name);  // Audi
console.log(person.car["model"]);  // A6
```

Todos Application | Cheat Sheet

1. HTML Input Element

1.1 Checkbox

The HTML input element can be used to create a Checkbox. To define a Checkbox, you need to specify the HTML type attribute with the value checkbox for an HTML input element.

Syntax: <input type="checkbox" />

1.2 The HTML Label Element

The HTML label element defines a Label.

Syntax: <label for="myCheckbox">Graduated</label>

1.2.1 The HTML for Attribute

The HTML **for** attribute associates the HTML **label** element with an HTML **input** element.

Syntax: <input type="checkbox" id="myCheckbox" />

<label for="myCheckbox">Graduated</label>

2. DOM Manipulations

2.1 The htmlFor Property

We can use **htmlFor** property to add HTML **for** attribute to the HTML **label** element.

Syntax: let labelElement = document.createElement("label");

labelElement.htmlFor = "myCheckbox";

2.2 The setAttribute() Method

We can use **setAttribute()** method to set any HTML attribute name and its corresponding value. If the attribute already exists, the value is updated. Otherwise, a new attribute is added with the specified name and value.

Syntax: Element.setAttribute(name, value);

Example: let labelElement = document.createElement("label");

labelElement.setAttribute("for", "myCheckbox");

3. Loops

Loops allow us to execute a block of code several times.

- for...of Loop

- for...in Loop
- for Loop
- while Loop and many more.

3.1 The for...of Loop

```
let myArray = [1, 2, 3, 4];

for (let eachItem of myArray) {
  console.log(eachItem);
}
```

The HTML Code for creating a Todo Item:

```
<li class="todo-item-container d-flex flex-row">
 <input type="checkbox" id="checkboxInput" class="checkbox-input" />
 <div class="d-flex flex-row label-container">
  <label for="checkboxInput" class="checkbox-label">
   Learn HTML
  </label>
  <div class="delete-icon-container">
   <i class="far fa-trash-alt delete-icon"></i>
  </div>
 </div>
</li>
```

4. CSS Box Properties

4.1 Border

The CSS border property is a shorthand property for:

- border-width
- border-style (required)
- border-color

For example,

```
.button {
  border-style: dashed;
  border-width: 2px;
  border-color: #e4e7eb;
}
```

Instead of writing the CSS properties **border-style**, **border-width** and **border-color** individually, we can apply these properties at once with a single CSS property called **border**.

Syntax: border: border-width border-style border-color

Example:

```
.button {
        border: 2px dashed #e4e7eb;
}
```

To specify the border on one of the four sides of an HTML element, you can use the below CSS properties.

- border-top
- border-bottom
- border-right
- border-left

```
.button {
  border-top: 1px dotted #e4e7eb;
}
```

If the border is not required, we can apply the none as value to the CSS border property.

```
.button {
  border: none;
}
```

For example, if the border property is not required on the top side of an HTML element. You can use,

```
.button {
  border-top: none;
```

}

On-Demand Session | Cheat Sheet

1. Most Commonly Made Mistakes

1.1 Most of the JS properties and methods should be in the Camel case.

Most of the JS properties and methods are in the Camel case (the starting letter of each word should be in uppercase except for the first word).

Code	Mistake	Correct Syntax
document.CreateElement()	C in Uppercase	document.createElement()
document.getElementbyId()	b in Lowercase	document.getElementById()
element.textcontent	c in Lowercase	element.textContent
element.classlist.add()	l in Lowercase	element.classList.add()

1.2 The ID should be the same in both the HTML and JS.

1.2.1 Mistake:

Html

```
<h1 id="heading">Shopping List</h1>
```

JS

```
let headingEl = document.getElementById("listHeading");

headingEl.textContent = "Items Needed";
```

In the above Code Snippets, the HTML element's text content doesn't change because the ID used in HTML and JS are different.

So, While accessing an HTML element with the **ID** using JS, **the ID used in the HTML element and the document.getElementById method must be the same.**

Syntax:

Html

```
<h1 id="heading">Shopping List</h1>
```

JS

```
let headingEl = document.getElementById("heading");

headingEl.textContent = "Items Needed";
```

1.2.2 Mistake:

Html

```
<h1 id="listHeading ">Shopping List</h1>
```

JS

```
let headingEl = document.getElementById("listHeading");

headingEl.textContent = "Items Needed";
```

The HTML element's text content doesn't change because there is an extra space at the end of the ID in the HTML code.

So, there shouldn't be any extra spaces in the IDs used in both the HTML and JS.

Html

```
<h1 id="listHeading">Shopping List</h1>
```

JS

```
let headingEl = document.getElementById("listHeading");

headingEl.textContent = "Items Needed";
```

1.3. The Function name must be the same in both the Function declaration and the Function call.

1.3.1 Mistake:

```
function greeting() {
  let message = "Hello Rahul";
  console.log(message);
}

greet();
```

As there is no function called **greet**, we will get an error in the above Code Snippet.

So, while calling a function, you must use the same function name used in the function declaration.

Example: function greeting() {

```
  let message = "Hello Rahul";
```

```
  console.log(message);
}
greeting();
```

Todos Application | Part 2 | Cheat Sheet

1. HTML Input Element

1.1 Placeholder

Placeholder is the text that appears in the HTML input element when no value is set. We can specify it using the HTML attribute **placeholder.**

Syntax: <input type="text" placeholder="Enter your name" />

2. JavaScript Built-in Functions

2.1 alert()

The alert() function displays an alert box with a specified message and an OK button.

Syntax: alert("Enter Valid Text");

3. DOM Properties

3.1 Checked

The **checked** property sets or returns the checked status of an HTML checkbox input element as a boolean value.

Syntax:

```
let checkboxElement = document.getElementById(checkboxId);
    checkboxElement.checked = true;
```

4. DOM Manipulations

4.1 The removeChild() Method

The **removeChild()** method removes an HTML child element of the specified HTML parent element from the DOM and returns the removed HTML child element.

Example:

```
function onDeleteTodo(todoId) {
  let todoElement = document.getElementById(todoId);
  todoItemsContainer.removeChild(todoElement);
}
```

4.2 The classList.toggle() Method

The **classList.toggle()** method is used to toggle between adding and removing a class name from an HTML element.

Example:

```
function onTodoStatusChange(checkboxId, labelId) {

  let checkboxElement = document.getElementById(checkboxId);

  let labelElement = document.getElementById(labelId);

  labelElement.classList.toggle('checked');

}
```

We can replace **classList.add()** and **classList.remove()** methods with **classList.toggle()** method.

Todos Application | Part 3 | Cheat Sheet

1. Execution Context

- The environment in which JavaScript Code runs is called Execution Context.
- Execution context contains all the variables, objects, and functions.
- Execution Context is destroyed and recreated whenever we reload an Application.

2. Storage Mechanisms

2.1 Client-Side Data Storage

Client-Side Data Storage is storing the data on the client (user's machine).

- Local Storage
- Session Storage
- Cookies
- IndexedDB and many more.

2.2 Server-Side Data Storage

Server-Side Data Storage is storing the data on the server.

3. Local Storage

- It allows web applications to store data locally within the user's browser.
- It is a Storage Object. Data can be stored in the form of key-value pairs.
- Please note that both key and value must be strings. If their type is other than a string, they get converted to strings automatically.

Key	Value
fullName	Rahul Attuluri
gender	Male
City	Delhi

To access and work with Local Storage we have below methods:

- **setItem()**
- **getItem()**
- clear()

- removeItem()

3.1 The setItem() Method

The **setItem()** method can be used for storing data in the Local Storage.

Syntax:

```
localStorage.setItem("Key", "Value");
```

3.2 The getItem() Method

The **getItem()** method can be used for getting data from the Local Storage.

Syntax:

```
localStorage.getItem("Key");
```

4. Values

4.1 null

We use null in a situation where we intentionally want a variable but don't need a value to it.

Example:

```
let selectedColor = null;

console.log(selectedColor);  // null

console.log(typeof(selectedColor));  // object
```

5. HTML Elements

5.1 The textarea Element

The HTML **textarea** element can be used to write the multiline text as an input.

Syntax: <textarea rows="8" cols="55"></textarea>

The HTML rows attribute specifies the number of lines.

The HTML cols attribute specifies the number of characters per each line.

Todos Application | Part 4 | Cheat Sheet

1. JavaScript Object Notation (JSON)

JSON is a data representation format used for:

- Storing data (Client/Server)
- Exchanging data between Client and Server

1.1 Supported Types

- Number
- String
- Boolean
- Array
- Object
- Null

1.2 JS Object vs JSON Object

In JSON, all keys in an object must be enclosed with double-quotes. While in JS, this is not necessary.

JS:

```
let profile = {
  name: "Rahul",
  age: 29,
  designation: "Web Developer"
};
```

JSON:

```
let profile = {
  "name": "Rahul",
  "age": 29,
  "designation": "Web Developer"
};
```

1.3 JSON Methods

1.3.1 JSON.stringify()

It converts the given value into JSON string.

Syntax:

```
JSON.stringify( value )
```

1.3.2 JSON.parse()

It parses a JSON string and returns a JS object.

Syntax:

```
JSON.parse( string )
```

Todos Application | Part 5 | Cheat Sheet

1. Array Methods

Method	Functionality
includes, indexOf, lastIndexOf, find, findIndex()	Finding Elements
push, unshift, splice	Adding Elements
pop, shift, splice	Removing Elements
concat, slice	Combining & Slicing Arrays
Join	Joining Array Elements
Sort	Sorting Array Elements

1.1 splice()

The splice() method changes the contents of an array. Using splice() method, we can

- Remove existing items
- Replace existing items
- Add new items

1.1.1 Removing existing items

Syntax:

```
arr.splice(Start, Delete Count)
```

Start: Starting Index

Delete Count: Number of items to be removed, starting from the given index

Example:

```
let myArray = [5, "six", 2, 8.2];

myArray.splice(2, 2);
```

```
console.log(myArray);  // [5, "six"]

let deletedItems = myArray.splice(2, 2);

console.log(deletedItems);  // [2, 8.2]
```

The **splice()** method returns an array containing the deleted items.

1.1.2 Adding new items

Syntax:

```
arr.splice(Start, Delete Count, Item1, Item2 ... )
```

Here the **Item1, Item2 ...** are the items to be added, starting from the given index.

Example:

```
let myArray = [5, "six", 2, 8.2];

myArray.splice(2, 0, "one", false);

console.log(myArray);  // [5, "six", "one", false, 2, 8.2]
```

1.1.3 Replacing existing items

Syntax:

```
arr.splice(Start, Delete Count, Item1, Item2 ... )
```

Example:

```
let myArray = [5, "six", 2, 8.2];

myArray.splice(2, 1, true);

console.log(myArray);  // [5, "six", true, 8.2]
```

1.2 findIndex()

The **findIndex()** method returns the first item's index that satisfies the provided testing function. If no item is found, it returns -1.

Syntax:

```
arr.findIndex(Testing Function)
```

Here Testing Function is a function to execute on each value in the array.

Example:

```
let myArray = [5, 12, 8, 130, 44];

let itemIndex = myArray.findIndex(function(eachItem) {

  console.log(eachItem);

});
```

In the above code snippet, the below function is a Testing Function.

```
function(eachItem) {

  console.log(eachItem);

}
```

Try out the **splice()** and **findIndex()** methods in the below Code Playground.

1.3 includes()

The **includes()** method returns true if the provided item exists in the array. If no item is found, it returns false.

Syntax:

```
arr.includes(item)
```

1.4 indexOf()

The **indexOf()** method returns the first index at which a given item can be found in the array. If no item is found, it returns -1.

Syntax:

```
arr.indexOf(item)
```

1.5 lastIndexOf()

The **lastIndexOf()** method returns the last index at which a given item can be found in the array. If no item is found, it returns -1.

Syntax:

```
arr.lastIndexOf(item)
```

1.6 find()

The **find()** method returns the first item's value that satisfies the provided testing function. If no item is found, it returns undefined.

Syntax:

```
arr.find(Testing Function)
```

1.7 unshift()

The **unshift()** method adds one or more items to the beginning of an array and returns the new array length.

Syntax:

```
arr.unshift(item1,item2, ..., itemN)
```

1.8 shift()

The **shift()** method removes the first item from an array and returns that removed item.

Syntax:

```
arr.shift()
```

1.9 concat()

The **concat()** method can be used to merge two or more arrays.

This method does not change the existing arrays but instead returns a new array.

Syntax: let newArray = arr1.concat(arr2);

1.10 slice()

The **slice()** method returns a portion between the specified start index and end index(end index not included) of an array into a new array.

Syntax:

```
arr.slice(startIndex, endIndex)
```

1.11 join()

The join() method creates and returns a new string by concatenating all of the items in an array, separated by commas or a specified separator string.

If the array has only one item, then it will be returned without using the specified separator.

Syntax:

```
arr.join(separator)
```

Here **separator** is a string used to separate each item of the array. If omitted, the array items are separated with a comma.

1.12 sort()

The **sort()** method sorts the items of an array and returns the sorted array. The default sort order is ascending.

Syntax:

```
arr.sort()
```

1. Local Storage

1.1 The removeItem() Method

The **removeItem()** method removes the specified storage object item based on the key.

Syntax:

```
localStorage.removeItem(key)
```

Key - Name of the key to be removed

```
localStorage.removeItem("todoList");
```

HTTP Requests using JS | Cheat Sheet

1. Fetch

The **fetch()** method is used to fetch resources across the Internet.

Syntax: fetch(URL, OPTIONS)

URL - URL of the resource

OPTIONS - Request Configuration

1.1 Request Configuration

We can configure the fetch request with many options like,

- Request Method
- Headers
- Body
- Credentials
- Cache, etc.

We can configure a request by passing an options object with required properties and their values.

For example,

```
let options = {
  method: "GET",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify(data)
};
```

2. Making HTTP Requests using Fetch

The method property value in the options object can be GET, POST, PUT, DELETE, etc. The default method is GET.

2.1 GET

The **GET** method can be used to retrieve (get) data from a specified resource.

For example,

```
let options = {
  method: "GET"
};
fetch("https://gorest.co.in/public-api/users", options);
```

2.2 POST

The **POST** method can be used to send data to the server.

Example:

```
let data = {
  name: "Rahul",
  gender: "Male",
  email: "rahul@gmail.com",
  status: "Active"
};
let options = {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    Accept: "application/json",
    Authorization: "Bearer ACCESS-TOKEN"
  },
  body: JSON.stringify(data)
};
```

```
fetch("https://gorest.co.in/public-api/users", options)
  .then(function(response) {
    return response.json();
  })
  .then(function(jsonData) {
    console.log(jsonData);
  });
```

2.3 PUT

The **PUT** method can be used to update the existing resource.

Example:

```
let data = {
  name: "Rahul Attuluri"
};
let options = {
  method: "PUT",
  headers: {
    "Content-Type": "application/json",
    Accept: "application/json",
    Authorization: "Bearer ACCESS-TOKEN"
  },
  body: JSON.stringify(data)
};
fetch("https://gorest.co.in/public-api/users/1359", options)
  .then(function(response) {
    return response.json();
  })
```

```
  .then(function(jsonData) {
    console.log(jsonData);
  });
```

2.4 DELETE

The **DELETE** method can be used to delete the specified resource.

Example:

```
let options = {
  method: "DELETE",
  headers: {
    "Content-Type": "application/json",
    Accept: "application/json",
    Authorization: "Bearer ACCESS-TOKEN"
  }
};
fetch("https://gorest.co.in/public-api/users/1359", options)
  .then(function(response) {
    return response.json();
  })
  .then(function(jsonData) {
    console.log(jsonData);
  });
```

3. HTTP Response Object Properties and Methods

Response Object provides multiple properties to give more information about the HTTP Response.

- status (number) - HTTP status code
- statusText (string) - Status text as reported by the server, e.g. "Unauthorized"

- headers
- url
- text() - Getting text from response
- json() - Parses the response as JSON

For example,

```
let options = {
  method: "GET"
};
fetch("https://gorest.co.in/public-api/users", options)
  .then(function(response) {
    return response.status;
  })
  .then(function(status) {
    console.log(status);  // 200
  });
```

In the above example, we can get the response status as **200** when the request is success.

1. HTML Input Element

1.1 Search

The HTML input element with the type search is designed for the user to enter the search queries.

Syntax: <input type="search" />

2. Bootstrap Components

2.1 Spinner

The Spinners can be used to show the loading state of the page.

Syntax:

```
<div class="spinner-border" role="status">
  <span class="sr-only">Loading...</span>
</div>
```

Forms | Cheat Sheet

1. HTML Forms

The HTML Forms can be used to collect data from the user.

Forms are of different kinds:

- Login/Sign in Form
- Registration Form
- Contact Us Form, etc.

1.1 HTML Form Element

The HTML form element can be used to create HTML forms. It is a container that can contain different types of Input elements like Text Fields, Checkboxes, etc.

1.1 HTML Form Element

The HTML form element can be used to create HTML forms. It is a container that can contain different types of Input elements like Text Fields, Checkboxes, etc.

Syntax: `<form></form>`

Note: Whenever we click a button or press Enter key while editing any input field in the form, the submit event will be triggered.

2. Event Object Methods

2.1 preventDefault

The **preventDefault()** method prevents the occurrence of default action.

Here in the form, it prevents the default behaviour of the **submit** event.

Example:

```
let myFormEl = document.getElementById("myForm");

myFormEl.addEventListener("submit", function(event) {

  event.preventDefault();

});
```

3. Event Types

There are different types of events.

- Keyboard Events
- Mouse Events
- Touch Events
- Form Events, etc.

3.1 Form Events

A Form Event is an event that can occur within a form.

Some of the form events are:

- blur
- focus
- change, etc.

3.1.1 Blur Event

The **blur** event happens when an HTML element has lost focus.

Example:

```
let nameEl = document.getElementById("name");

nameEl.addEventListener("blur", function(event) {

  console.log("blur event triggered");

});
```

Forms | Part-2 | Cheat Sheet

1. HTML Select Element

The HTML select element is used to create a drop-down list.

Syntax: <select></select>

1.1 HTML Option Element

The HTML **option** element is used to create the menu option of a drop-down list.

The text content of the HTML **option** element is used as a label.

Syntax:

```
<select>
  <option>Active</option>
</select>
```

1.1.1 The value Attribute

Every HTML option element should contain the HTML value attribute.

Syntax: <option value="Active">Active</option>

2. HTML Input Element

2.1 Radio

The HTML **input** radio element is used to select one option among a list of given options.

Syntax:
```
<input type="radio" id="genderMale" value="Male" />
<input type="radio" id="genderFemale" value="Female" />
```

2.1.1 HTML name attribute

The HTML name Attribute specifies the name for an HTML Element.

Syntax: <input type="radio" value="Male" name="gender" />

2.1.2 Radio Group

All the radio buttons with same name collectively called as a radio group.

We can select only one radio button within a radio group.

Syntax: <input type="radio" value="Male" name="gender" />

<input type="radio" value="Female" name="gender" />

3. Boolean Attributes

For the HTML Boolean attributes, we only specify the name of the HTML attribute.

The presence of a boolean attribute represents the **true** value, and the absence represents the **false** value.

3.1 HTML selected attribute

The **selected** attribute specifies that an option should be pre-selected when the page loads.

Syntax: <option value="Active" selected>Active</option>

3.2 HTML checked attribute

The **checked** attribute specifies that an input element should be pre-selected (checked) when the page loads.

Syntax: <input type="radio" id="genderMale" value="Male" name="gender" checked />

Callbacks & Schedulers | Cheat Sheet

1. Callback function

A Callback is a function that is passed as an argument to another function.

1.1 Passing a function as an argument

Example:

```
function displayGreeting(displayName) {
  console.log("Hello");
  displayName();
  console.log("Good Morning!");
}
displayGreeting(function() {
  console.log("Rahul");
});
```

1.2 Passing a function name as an argument

Example:

```
function displayGreeting(displayName) {
  console.log("Hello");
  displayName();
  console.log("Good Morning!");
}
function displayRahul() {
  console.log("Rahul");
}
displayGreeting(displayRahul);
```

1.3 Passing a function expression as an argument

Example:

```
function displayGreeting(displayName) {
```

```
  console.log("Hello");

  displayName();

  console.log("Good Morning!");

}

let displayRam = function() {

  console.log("Ram");

};

displayGreeting(displayRam);
```

2. Schedulers

The Schedulers are used to schedule the execution of a callback function.

There are different scheduler methods.

- setInterval()
- clearInterval()
- setTimeout()
- clearTimeout(), etc.

2.1 setInterval()

The setInterval() method allows us to run a function at the specified interval of time repeatedly.

Syntax:

```
setInterval(function, delay);
```

function - a callback function that is called repeatedly at the specified interval of time (delay).

delay - time in milliseconds. (1 second = 1000 milliseconds)

Example:

```
let counter = 0;

setInterval(function() {

  console.log(counter);

  counter = counter+1;

}, 1000);
```

In the **setInterval()** method, the **callback** function repeatedly executes until the browser tab is closed or the scheduler is cancelled.

When we call the **setInterval()** method, it returns a unique id. This unique Id is used to cancel the **callback** function execution.

2.2 clearInterval()

The **clearInterval()** method cancels a schedule previously set up by calling **setInterval().**

To execute **clearInterval()** method, we need to pass the **uniqueId** returned by **setInterval()** as an argument.

Syntax:

```
clearInterval(uniqueId);
```

Example:

```
let counter = 0;

let uniqueId = setInterval(function() {

  console.log(counter);

  counter = counter+1;

}, 1000);

clearInterval(uniqueId);
```

2.3 setTimeout()

The **setTimeout()** method executes a function after the specified time.

Syntax:

```
setTimeout(function, delay);
```

function - a callback function that is called after the specified time (**delay**).

delay - time in milliseconds.

Example:

```
let counter = 0;

setTimeout(function() {

  console.log(counter);

  counter = counter + 1;

}, 1000);
```

2.4 clearTimeout()

We can cancel the **setTimeout()** before it executes the **callback** function using the **clearTimeout()** method.

To execute **clearTimeout(),** we need to pass the **uniqueId** returned by **setTimeout()** as an argument.

Syntax:

```
clearTimeout(uniqueId);
```

Example:

```
let counter = 0;

let uniqueId = setTimeout(function() {
  console.log(counter);
  counter = counter+1;
}, 1000);

clearTimeout(uniqueId);
```

Event Listeners and More Events | Cheat Sheet

1. Event Listeners

JavaScript offers three ways to add an Event Listener to a DOM element.

- Inline event listeners
- onevent listeners
- addEventListener()

1.1 Inline event listeners

Syntax:

HTML: <button onclick="greeting()">Greet</button>

JS:

```
function greeting() {
  console.log("Hi Rahul");
}
```

1.2 onevent listeners

Syntax:

HTML: <button id="greetBtn">Greet</button>

JS:

```
let greetBtnEl = document.getElementById("greetBtn");

greetBtnEl.onclick = function() {
  console.log("Hi Rahul");
};
```

1.3 addEventListener()

It is a modern approach to add an event listener.

Syntax: element.addEventListener(event, function);

element - HTML element

event - event name

function - Callback function

Example:

HTML <button id="greetBtn">Greet</button>

JS:

```
let greetBtn = document.getElementById("greetBtn");

greetBtn.addEventListener("click", function() {
  console.log("Hi Rahul");
});
```

2. Operators

2.1 Comparison Operators

Operator	Usage	Description
Equal (==)	a == b	returns true if both *a* and *b* values are equal.
Not equal (!=)	a != b	returns true if both *a* and *b* values are not equal.
Strict equal (===)	a === b	returns true if both *a* and *b* values are equal and of the same type.
Strict not equal (!==)	a !== b	returns true if either *a* and *b* values are not equal or of the different type.
Greater than (>)	a > b	returns true if *a* value is greater than *b* value.
Greater than or equal (>=)	a >= b	returns true if *a* value is greater than or equal to *b* value.
Less than (<)	a < b	returns true if *a* value is less than *b* value.
Less than or equal (<=)	a <= b	returns true if *a* value is less than or equal to *b* value.

2.2 Logical Operators

Operator	Usage	Description
AND (&&)	a && b	returns true if both *a* and *b* values are true.
OR (| |)	a | | b	returns true if either *a* or *b* value is true.
NOT (!)	!a	returns true if *a* value is not true.

3. More Events

Events are the actions by which the user or browser interact with HTML elements.

There are different types of events.

- Keyboard Events
- Mouse Events
- Touch Events, and many more.

3.1 Keyboard Events

Keyboard Event is the user interaction with the keyboard.

The keyboard events are

- keydown
- keyup

3.1.1 Keydown event

The **keydown** event occurs when a key on the keyboard is pressed.

Syntax:

```
element.addEventListener("keydown", function);
```

Example:

```
let inputEl = document.createElement("input");

function printKeydown() {

  console.log("key pressed");

}

inputEl.addEventListener("keydown", printKeydown);

document.body.appendChild(inputEl);
```

3.1.2 Keyup event

The keyup event occurs when a key on the keyboard is released.

Syntax:

```
element.addEventListener("keyup", function);
```

3.2 Event Object

- Whenever an event happens, the browser creates an event object.
- It consists of information about the event that has happened.

It consists of many properties and methods.

- type
- target

- key
- timeStamp
- stopPropagation, and many more.

3.2.1 Properties & Methods

For any event, event-specific properties and methods will be present.

For Example,

The **keydown** event has key property, whereas the onclick event doesn't have it.

event.type

The **event.type** property contains the type of event occurred like click, keydown, etc.

Example:

```
let inputEl = document.createElement("input");

function printKeydown(event) {

  console.log(event.type);  // keydown

}

inputEl.addEventListener("keydown", printKeydown);

document.body.appendChild(inputEl);
```

event.target

The **event.target** property contains the HTML element that triggered the event.

Example:

```
let inputElement = document.createElement("input");

function printKeydown(event) {

  console.log(event.target);  // <input></input>

}

inputElement.addEventListener("keydown", printKeydown);

document.body.appendChild(inputElement);
```

event.key

The **event.key** property contains the value of the key pressed by the user.

Example:

```
let inputElement = document.createElement("input");

function printKeydown(event) {
  console.log(event.key);
}

inputElement.addEventListener("keydown", printKeydown);

document.body.appendChild(inputElement);
```

Keyboard key	event.key value
Enter	Enter
A	a
A	A
1	1
*	*
<	<

Hypertext Transfer Protocol (HTTP) | Cheat Sheet

1. Web Resources

A Web Resource is any data that can be obtained via internet.

A resource can be

- HTML document
- CSS document
- JSON Data or Plain text
- Image, Video, etc.

2. Uniform Resource Locator (URL)

URL is a text string that specifies where a resource can be found on the internet.

We can access web resources using the URL.

Syntax: protocol://domainName/path?query-parameters

In the URL http://www.flipkart.com/watches?type=digital&rating=4,

- **http** is a Protocol
- **www.flipkart.com** is a Domain Name
- **/watches** is a Path
- **type=digital&rating=4** is the Query Parameters

2.1 Protocol

A protocol is a standard set of rules that allow electronic devices to communicate with each other.

There are different types of protocols.

- Hypertext Transfer Protocol (HTTP)
- Hypertext Transfer Protocol Secure (HTTPS)
- Web Sockets, etc.

2.1.1 HTTP

The Hypertext Transfer Protocol (HTTP), is a protocol used to transfer resources over the web.

Examples: Internet forums, Educational sites, etc.

HTTP Request: Message sent by the client

HTTP Response: Message sent by the server

2.1.2 HTTPS

In Hypertext Transfer Protocol Secure (HTTPS), information is transferred in an encrypted format and provides secure communication.

Examples: Banking Websites, Payment gateway, Login Pages, Emails and Corporate Sector Websites, etc.

2.2 Domain Name

It indicates which Web server is being requested.

2.3 Path

The path is to identify the resources on the server.

Examples:

- **/watches** in

http://www.flipkart.com/watches

- **/electronics/laptops/gaming** in

http://www.flipkart.com/electronics/laptops/gaming

2.4 Query Parameters

- Query parameters add some criteria to the request for the resource.
- Multiple query parameters can be added by using an & (ampersand) symbol.

For example,

http://www.flipkart.com/watches?type=digital&rating=4

3. HTTP

3.1 HTTP Requests

HTTP requests are messages sent by the client to initiate an action on the server.

HTTP request includes

- Start Line
- Headers
- Body

3.1.1 Start Line

A Start Line specifies a

- URL
- HTTP Method
- HTTP Version

HTTP Methods

The HTTP Request methods indicate the desired action to be performed for a given resource.

Methods	Description
GET (Read)	Request for a resource(s) from the server
POST (Create)	Submit data to the server
PUT (Update)	The data within the request must be stored at the URL supplied, replacing any existing data
DELETE (Delete)	Delete a resource(s)

HTTP Version

Year	Version
1991	HTTP/0.9
1994	HTTPS
1996	HTTP/1.0
1997	HTTP/1.1
2015	HTTP/2
2019	HTTP/3

3.1.2 Headers

HTTP Headers let the client and the server to pass additional information with an HTTP request or response.

3.1.3 Body

We place the data in the Request body when we want to **send data** to the server.

For example, form details filled by the user.

HTTP Requests

- Start Line
    - URL
        - Protocol
            - HTTP
            - HTTPS
        - Domain Name
        - Path
        - Query Parameters
    - HTTP Method
        - GET (Read)
        - POST (Create)
        - PUT (Update)
        - DELETE (Delete)
    - HTTP Version
- Headers
- Body

3.2 HTTP Responses

HTTP responses are **messages** sent by the server as an answer to the **clients** request.

HTTP Response includes

- Status Line
- Headers
- Body

3.2.1 Status Line

A Status line specifies a

- HTTP version

- Status code
- Status text

Status code

Status codes Indicate whether an HTTP request has been successfully completed or not.

Status Code Series	Indicates
1XX	Information
2XX	Success
3XX	Redirection
4XX	Client Error
5XX	Server Error

- 200 (Success) - Indicates that the request has succeeded
- 201 (Created) - The request has succeeded and a new resource has been created as a result

Status text

Status Code	Status Text
200	OK
204	No Response
301	Moved Permanently
401	Unauthorized
403	Forbidden
404	Not Found

3.2.2 Body

Response Body contains the resource data that was requested by the client.

HTTP Responses

- Status Line
    - HTTP version
    - Status code
        - 1XX
        - 2XX
        - 3XX
        - 4XX
        - 5XX
    - Status text
- Headers
- Body