JavaScript String Operations

```
let firstName = "HuXn";
let lastName = "WebDev";
let fullName = firstName + lastName;
// console.log(fullName);

1. CONCATENATION
let fullName = firstName + " " + lastName
let fullName = firstName.concat(lastName);

2. APPEND
firstName += " something else"

3. LENGTH
console.log(firstName.length);

4. CASES
console.log(firstName.toUpperCase());
console.log(firstName.toLowerCase());

5. SLICE
console.log(fullName.slice(0, 3));

6. SPLIT & JOIN
console.log(fullName.split("").join("-"));

7. INCLUDES
console.log(fullName.includes("HuXn"));

8. TRIM
console.log(fullName.trim());
```

Type Conversion

```
let amount = 100;
let money = "100";
let floatValue = "99.5";

// - Convert string to number
amount = parseInt(amount);
amount = +amount;
amount = Number(amount)

//  - Convert number to string
money = money.toString();
```

```
money = String(money);

// - Change string to decimal
floatValue = parseFloat(floatValue);
console.log(floatValue);
```

Comparision operator

```
// RELATIONAL OPERATORS
// > Greater Then
// < Less Then
// >= Greater Then or equal to
// <= Less Then or equal to

console.log(10 > 10);
console.log(10 < 10);
console.log(10 >= 10);
console.log(10 <= 10);

// EQUALITY OPERATORS
// === strict equality (Type + Value)
// !== strict non-equality (Type + Value)
// == Lose equality (values)
// != Lose equality (values)

console.log(10 === 10);
console.log(10 !== 10);
console.log(10 == 10);
console.log(10 != 10);
```

Arrays (typeOf --> object)

```
// An array is an object that can store multiple values at once. For
example,
// const words = ["hello", "world", "welcome"];
// ['string', 123, true, []]


// empty array
const myList = [];

// array of numbers
const numberArray = [2, 4, 6, 8];

// array of strings
```

```
const stringArray = ["eat", "work", "sleep"];

// array with mixed data types
const newData = ["work", "exercise", 1, true];

const newData = [
  { task1: "exercise" },
  [1, 2, 3],
  function hello() {
    console.log("hello");
  },
];

// ACCESSING ITEMS
const myArray = ["h", "e", "l", "l", "o"];

// first element
console.log(myArray[0]); // "h"

// second element
console.log(myArray[1]); // "e"
```

Array methods

Method	Description
concat()	joins two or more arrays and returns a result
includes()	checks if an array contains a specified element
push()	aads a new element to the end of an array and returns the new length of an array
unshift()	adds a new element to the beginning of an array and returns the new length of an array
pop()	removes the last element of an array and returns the removed element
shift()	removes the first element of an array and returns the removed element
sort()	sorts the elements alphabetically in strings and in ascending order
slice()	selects the part of an array and returns the new array
splice()	removes or replaces existing elements and/or adds new elements

Array methods implementation

Part 1:-

```
// push() -> Add item at the end of the array
// pop() -> Remove item at the end of the array
```

```
// shift() -> Remove from the start
// unshift() -> Add to start
// concat() -> combining arrays

const fruits = [
  "apples",
  "pomegranate",
  "mango",
  "strawberries",
  "pineapple",
  "grapefruit",
];

console.log(fruits);
fruits.push("banana");
fruits.pop();
fruits.shift();
fruits.unshift("orange");

// CONCAT
// const fruits = ["apples", "pomegranate", "mango"];
// let moreFruits = ["strawberries", "pineapple", "grapefruit"];
// let totalFruits = fruits.concat(moreFruits);

console.log(fruits);
```

Part 2:-

```
// join - creates string from array
// reverse - reverse an array
// slice - copy portion of an array
// sort - sorts an array

let pl = ["JavaScript", "Golang", "Python", "php"];
let numbers = [3, 5, 2, 4, 1];

console.log(pl.includes("Golang"));
console.log(pl.join("-"));
console.log(pl.reverse());
console.log(pl.slice(0, 3));
console.log(pl.sort());
```

Objects in JS

```
// JavaScript object is a non-primitive data-type that allows you to store
multiple collections of data.
```

```
// {key: value}


// Creating Object
let person = {
  firstName: "HuXn",
  lastName: "WebDev",
  age: 18,
  location: ["Planet", "Earth"],
  isProgrammer: true,
};

// Accessing Properties
console.log(typeof person);
console.log(person.location[1]);
console.log(person["isProgrammer"]);
// console.log(person[isProgrammer]); // ERROR -> without quotes

// Updating Properties
console.log(person.firstName);
console.log((person.firstName = "Sam"));

// Add new properties
console.log((person.isProgrammer = false));
console.log(person);
```

📘 JavaScript Object Methods

```
const user = {
  name: "Vinayak",
  age: 19
};
```

Method	Description	Example	Output
`Object.keys(obj)`	Returns array of keys	`Object.keys(user)`	`["name", "age"]`
`Object.values(obj)`	Returns array of values	`Object.values(user)`	`["Vinayak", 19]`
`Object.entries(obj)`	Returns array of key-value pairs	`Object.entries(user)`	`[["name", "Vinayak"], ["age",19]]`
`obj.hasOwnProperty("key")`	Checks if key exists in object	`user.hasOwnProperty("name")`	`true`

Method	Description	Example	Output
`Object.assign(target, source)`	Copies props from source to target	`Object.assign({}, user, {age: 20})`	`{name: "Vinayak", age: 20}`
`Object.freeze(obj)`	Freezes object (no add/edit/delete)	`Object.freeze(user)`	—
`Object.seal(obj)`	No add/delete, but can update	`Object.seal(user)`	—

Functions in JS

- A function is a block of code that performs a specific task.
- Function makes the code reusable. You can declare it once and use it multiple times.
- Function makes the program easier as each small task is divided into a function.
- Function increases readability.
- DRY - Don't Repeat Yourself
- function name(parameterIfAny) {...}
- Arguments --> whatever we pass to another function
- Parameters --> whatever we receive when that function is called

Functions Declaration

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

Calling a function

```
greet();
```

Return statement

```
// The return statement can be used to return the value to a function call.

function add(numberOne, numberTwo) {
  return numberOne + numberTwo;
  // console.log("Hello World"); NOTHING
}

const result = add(10, 20);
console.log(result);
```

Function expression

```
// Function Expressions
const greetings = function (user) {
  console.log(`Hello ${user}`);
};

greetings("Doe");
```

Callback Functions

When we provide function as an (argument) to other function/argument that function is known as callback function.

```
// function
function greet(name, cb) {
  console.log(`Hello ${name}`);
  cb();
}

// callback function
function callMe() {
  console.log("I am callback function");
}

// passing function as an argument (callback)
greet("Peter", callMe);
```

```
function showCallFunc(fn) {
  const value = 10;
  fn(value);
}

showCallFunc(function (value) {
  console.log(value);
});
```

Scope

- Scope in JavaScript refers to the current context of code, which determines the accessibility of variables to JavaScript.
- The two types of scope are local and global

- Global variables are those declared outside of a block.
- Local variables are those declared inside of a block.

```
// Global Scope
// Local Scope

let textMessage = "hi"; // Global Scope

function showMessage() {
  let textMessage = "hi"; // Local Scope
  console.log(textMessage); // Access Local Scope
}

for (let i = 0; i < 5; i++) {
  console.log(i); // Local Scope
}

console.log(i); // Global Scope
console.log(textMessage); // Access Global Scope
```

Methods

A method is a function that is associated with an object. It allows objects to perform actions or provide functionalities.

```
// Defining a method outside the object

function greet() {
  return `Hello, my name is ${person.name} and I am ${person.age} years
old.`;
}

const person = {
  name: "John",
  age: 30,
  greet,
};

console.log(person.greet());

// Defining a method inside the object (PREFERABLE)

const person = {
  name: "HuXn",
  age: 19,
  greet: function greet() {
    return `Hello, my name is ${person.name} and I am ${person.age} years
old.`;
  },
```

```
};

console.log(person.greet());
```

JSON

JSON stands for JavaScript Object Notation. It is a lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate. JSON is often used for transmitting data between a server and a web application, as well as for storing configuration settings and data.

In JavaScript, JSON is represented as a string, and it closely resembles JavaScript object literal notation. It consists of key-value pairs, where keys must be strings and values can be strings, numbers, booleans, arrays, or nested objects. The key-value pairs are separated by commas, and the entire JSON object is enclosed in curly braces {}.

- **JavaScript provides methods to work with JSON data:**
    - **JSON.stringify():** Converts a JavaScript object into a JSON string.
    - **JSON.parse():** Parses a JSON string and returns a JavaScript object.

Using JSON.stringify()

```
// Example
const person = {
  name: "John Doe",
  age: 30,
  email: "john@example.com",
  isSubscribed: true,
  hobbies: ["Reading", "Running", "Cooking"],
  address: {
    city: "New York",
    zipCode: "10001",
  },
};

// Convert JavaScript object to JSON string
const jsonString = JSON.stringify(person);

console.log(jsonString);

/* Output:
  {
    "name":"John Doe",
    "age":30,
    "email":"john@example.com",
    "isSubscribed":true,
    "hobbies":["Reading","Running","Cooking"],
    "address":{"city":"New York","zipCode":"10001"}
```

```
  }
*/
```

Using JSON.parse()

```
// Parse JSON string back to JavaScript object
const parsedObject = JSON.parse(jsonString);

console.log(parsedObject.name); // Output: "John Doe"
console.log(parsedObject.hobbies); // Output: ["Reading", "Running",
"Cooking"]
```

setInterval()

The setInterval function is used to repeatedly execute a function or a block of code at a specified interval. It takes two arguments: the function or code to be executed and the time interval (in milliseconds) between each execution.

```
// Example: Execute a function every 2 seconds

setInterval(function () {
  console.log("This function will be executed every 2 seconds.");
}, 2000);

// The setInterval function will continue to execute the specified code at
the specified interval until it is stopped using the clearInterval function.

// Example: Stop the interval after 10 seconds
const intervalId = setInterval(function () {
  console.log("This function is being executed at the interval.");
}, 1000);

// Stop the interval after 10 seconds
setTimeout(function () {
  clearInterval(intervalId);
  console.log("Interval stopped.");
}, 10000);
```

setTimeout()

The setTimeout function is used to execute a function or a block of code after a specified delay. It takes two arguments: the function or code to be executed and the time delay (in milliseconds) before the execution.

```
// Example: Execute a function after 3 seconds

setTimeout(function () {
  console.log("This function will be executed after 3 seconds.");
}, 3000);
```

Date & Time

In JavaScript, you can work with dates using the built-in Date object. The Date object allows you to create, manipulate, and format dates and times.

Creating a Date Object:

You can create a new Date object by calling the Date constructor with or without arguments. If no arguments are provided, it will create a Date object representing the current date and time.

```
// Create a Date object representing the current date and time
const currentDate = new Date();
console.log(currentDate);

// Create a Date object for a specific date and time (year, month, day,
hours, minutes, seconds, milliseconds)
const specificDate = new Date(2023, 6, 25, 12, 30, 0, 0);
console.log(specificDate);
```

Getting Different Parts of the Date:

You can extract various parts of a Date object, such as the year, month, day, hours, minutes, seconds, etc.

```
const date = new Date();

const year = date.getFullYear();
const month = date.getMonth(); // Month is 0-based (0: January, 1: February,
..., 11: December)
const day = date.getDate();
const hours = date.getHours();
const minutes = date.getMinutes();
const seconds = date.getSeconds();
const milliseconds = date.getMilliseconds();

console.log(`year:  ${year}`);
console.log(`month:  ${month}`);
console.log(`day:  ${day}`);
console.log(`hours:  ${hours}`);
console.log(`minutes:  ${minutes}`);
```

```
console.log(`seconds:  ${seconds}`);
console.log(`milliseconds:  ${milliseconds}`);
```

Formatting Dates:

You can format dates to display them in a more readable format using various methods.

```
const date = new Date();

// Convert date to a string representation in different formats

console.log(date.toDateString()); // Output: Mon Jul 25 2023
console.log(date.toISOString()); // Output: 2023-07-25T00:00:00.000Z
console.log(date.toLocaleString()); // Output: 7/25/2023, 12:00:00 AM (based
on the user's local timezone)
```

```
// Working with Time:
// You can perform operations on dates, such as adding or subtracting time.

const date = new Date();

// Add 1 day to the current date
date.setDate(date.getDate() + 1);

// Subtract 2 hours from the current time
date.setHours(date.getHours() - 2);

console.log(date);
```