JavaScript

Billangual Notes

www.linkedin.com/in/omi92229

https://t.me/SolutionsAndTricksIT

@SolutionsAndTricks

Created By: Om Shankar Mishra

INDEX

Basics....3
Setting Up the Development Environment for JavaScript 4
How to Add Script Tag in HTML....5
Writing Your First JavaScript Program....6
Comment....7
JavaScript Output....8
JavaScript Variables....9
Data Types....10
Types of Operators....12
Type Conversion....18
Math Object....20
If Statement....21
Switch Statement....25
While Loop....27
BigInt....30
For Loop....31
JavaScript Function....35
Objects....42
Arrays....53
Strings....70
for...in & for...of Loop....78
Number Methods....79
Random Number....80
Date Objects....81
Sets....85
Maps....87
JavaScript Destructuring....88
Bitwise Operations....91
JavaScript Hoisting....92
Console.log() & Use Strict....94
This Keyword....96
JavaScript Object-Oriented Programming (OOP)....97
Modules....115
Error Handling....116
JavaScript JSON....120
Introduction to DOM (Document Object Model) 122
JavaScript Forms: Introduction and Manipulation....142
Window - The Browser Object Model (BOM)....147
Cookies....157
JavaScript Callbacks....160
DOM Animation using JavaScript....171
Regular Expression (regex)....173

Basics

What is JavaScript?

JavaScript एक high-level, interpreted programming language है जो mainly web development के लिए use होती है। यह client-side और server-side दोनों पे run हो सकती है, और web pages में interactive elements add करने के लिए बहुत ही important है।

Key Features of JavaScript

- **Lightweight and Interpreted:** JavaScript को directly browser में run कर सकते हैं बिना किसी compilation के।
- **Dynamic Typing:** JavaScript variables का type runtime पे change हो सकता है।
- **Prototype-based Inheritance:** JavaScript objects एक दूसरे से inherit कर सकते हैं using prototypes.
- **First-class Functions:** Functions को variables की तरह treat कर सकते हैं और उनको पास या return कर सकते हैं as arguments.
- **Asynchronous Programming:** JavaScript में promises और async/await की मदद से asynchronous operations handle किये जा सकते हैं।
- **Event-driven Programming:** JavaScript events का use करके respond करता है जैसे button click, mouse hover, etc.

Applications of JavaScript

- **Web Development:** Frontend development में JavaScript use होता है for creating interactive web pages.
- **Server-side Development:** Node.js के through backend services भी create कर सकते हैं।
- **Mobile App Development:** Frameworks जैसे React Native और Ionic से mobile apps develop कर सकते हैं।
- **Game Development:** JavaScript और HTML5 को use करके web-based games बना सकते हैं।
- **Desktop Applications:** Electron framework की मदद से cross-platform desktop applications develop की जा सकती हैं।

Why Learn JavaScript

- **High Demand:** Web development industry में JavaScript developers की demand बहुत high है।
- **Versatile:** Frontend और backend दोनों के लिए use होती है।
- **Community Support:** JavaScript की एक strong community है जो continuously tools और libraries develop कर रही है।
- **Easy to Learn:** Beginners के लिए easy to understand और learn करने में आसान है।
- **Rich Ecosystem:** N number of libraries और frameworks available हैं जो development को fast और efficient बनाते हैं।

What Can JavaScript Do?

- **Dynamic Content:** Web pages को dynamically update कर सकता है without reloading.
- **Form Validation:** User inputs को validate करने के लिए use होता है।
- **Interactive Maps:** Maps को create और manage कर सकता है (Google Maps integration).
- **Animations:** HTML elements को animate कर सकता है to make the web page more engaging.
- **Real-time Updates:** WebSockets और AJAX के through real-time updates provide कर सकता है जैसे live chat applications.

Setting Up the Development Environment for JavaScript

JavaScript development environment setup करना relatively simple है। यहां पर main steps और tools explain कर रहे हैं जो आपको एक efficient development environment setup करने में मदद करेंगे।

1. Install a Code Editor: JavaScript development के लिए एक अच्छा code editor choose करना ज़रूरी है। यहां कुछ popular code editors हैं :

- **Visual Studio Code (VS Code):** Microsoft का free and open-source editor जो JavaScript development के लिए बहुत popular है।
- **Sublime Text:** Lightweight और fast editor, लेकिन कुछ advanced features के लिए paid license चाहिए।
- **Atom:** GitHub का open-source editor जो customization और plugins support करता है।

2. Install Browser: JavaScript development के लिए एक modern browser ज़रूरी है। Chrome और Firefox JavaScript development के लिए बहुत popular browsers हैं।

- **Google Chrome:** Visit Google Chrome website to download and install.
- **Mozilla Firefox:** Visit Mozilla Firefox website to download and install.

3. Install Browser Developer Tools: Modern browsers in-built developer tools provide करते हैं जो JavaScript debugging, inspecting, और testing के लिए use होते हैं।

For Google Chrome:

- Press Ctrl+Shift+I (Windows/Linux) or Cmd+Option+I (macOS) to open Developer Tools.
- Use the Console tab to write and test JavaScript code.

For Mozilla Firefox: Press Ctrl+Shift+K (Windows/Linux) or Cmd+Option+K (macOS) to open the Web Console.

4. Setting Up a Local Server: Local development environment में एक local server run करना useful होता है। यहां कुछ options हैं :

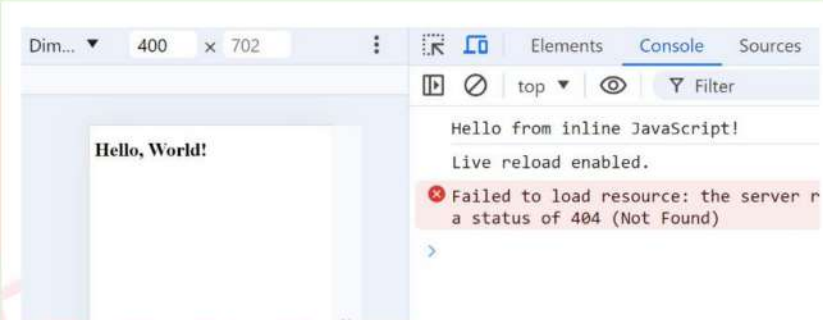
- **Live Server Extension (for VS Code):** यह extension install करने से आपको अपने project को local server पे run करने की facility मिलती है।

How to Add Script Tag in HTML

HTML में JavaScript code include करने के लिए <script> tag का use होता है। इस tag को आप directly HTML file में लिख सकते हैं या फिर external JavaScript file को link कर सकते हैं। दोनों methods को निचे explain किया गया है :

1. Inline JavaScript: Inline JavaScript का use तब होता है जब आप directly HTML file में JavaScript code लिखना चाहते हैं। यह method simple और small scripts के लिए useful है।

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Inline JavaScript Example</title>
</head>
<body>
  <h1>Hello, World!</h1>
  <script>
    // Inline JavaScript code
    console.log('Hello from inline JavaScript!');
  </script>
</body>
</html>
```



2. External JavaScript File: अगर आपका JavaScript code ज़्यादा है या आप अपने code को separate file में organize करना चाहते हैं, तो आप external JavaScript file को link कर सकते हैं।

Create the JavaScript File

- अपने project folder में एक new file create करें और उसका नाम script.js रखें।
- इस file में अपना JavaScript code लिखें :

```
// External JavaScript code
console.log('Hello from external JavaScript file!');
```

Link the JavaScript File in HTML: HTML file में <script> tag का use करके external JavaScript file को link करें। यह tag आप <head> section या <body> section के end में include कर सकते हैं।

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>External JavaScript Example</title>
  <script src="script.js" defer></script>
</head>
<body>
  <h1>Hello, World!</h1>
</body>
</html>
```

यहाँ पे <script src="script.js" defer></script> line से script.js file को link किया गया है। defer attribute ensure करता है की JavaScript file load होने के बाद ही execute हो जब HTML content fully loaded हो चुका हो।

3. Adding Multiple Script Tags: अगर आपको multiple JavaScript files include करनी हैं, तो आप multiple <script> tags का use कर सकते हैं।

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Multiple Scripts Example</title>
  <script src="script1.js" defer></script>
  <script src="script2.js" defer></script>
</head>
<body>
  <h1>Hello, World!</h1>
</body>
</html>
```

4. Using async and defer Attributes

async Attribute: यह attribute use करने से script asynchronously load होता है और load होने के बाद तुरंत execute होता है। इसका use तब useful है जब आपको किसी script के load होने का wait नहीं करना।

```
<script src="script.js" async></script>
```

defer Attribute: यह attribute ensure करता है की script load होने के बाद HTML parsing complete होने पर execute हो। यह attribute ज़्यादा preferred होता है क्यूंकि यह HTML parsing को block नहीं करता।

```
<script src="script.js" defer></script>
```

Writing Your First JavaScript Program

JavaScript का use करते हुए अपना पहला program लिखना शुरू करने के लिए, हम एक simple "Hello, World!" example से start करते हैं। ये example आपको basic HTML और JavaScript integration के concepts समझने में मदद करेगा।

1. Set Up the Project Folder: एक new folder create करें जहाँ आप अपना project store करेंगे। इस folder को, for example, first-js-program नाम दे सकते हैं।

2. Create the HTML File:

- इस project folder के अंदर एक new file create करें और उसका नाम **index.html** रखें।
- इस file में basic HTML structure add करें। HTML file का structure कुछ ऐसा होगा :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
```

```
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My First JavaScript Program</title>
</head>
<body>
  <h1>Hello, World!</h1>
  <script src="script.js"></script>
</body>
</html>
```

3. Create the JavaScript File

- अब एक new file create करें और उसका naam **script.js** रखें।
- इस file में अपना पहला JavaScript code लिखें।

```
// Display a greeting message in the console
console.log('Hello, World!');
```

4. Link the JavaScript File to HTML

- HTML file में <script> tag का use करके script.js file को link करें। हमने यह काम पहले से ही कर दिया है index.html file के <body> tag के end में।
- **<script src="script.js"></script>** यह line ensure करता है की script.js file execute हो तब जब HTML content load हो चुका हो।

5. Run Your Program

- अब अपनी index.html file को browser में open करें। यह आप directly file पे double-click करके या browser में drag and drop करके कर सकते हैं।
- Browser open होने के बाद, Developer Tools को open करें (Chrome में Ctrl+Shift+I या Cmd+Option+I use करें) और Console tab select करें।
- Console tab में आपको "Hello, World!" message display होगा जो हमने script.js file में console.log statement का use करके print किया था।

js comment

JavaScript में comments का use code को explain करने, temporarily disable करने या किसी specific part को note करने के लिए होता है। Comments को JavaScript interpreter ignore करता है, यानि comments execute नहीं होते। JavaScript में दो types के comments होते हैं : single-line comments और multi-line comments.

1. Single-line Comments: Single-line comments double forward slash // से start होते हैं और line के end तक चलेंगे।

```
// This is a single-line comment
console.log('Hello, World!');  // This comment explains the code
```

2. Multi-line Comments: Multi-line comments slash asterisk /* */ के beech लिखे जाते हैं और multiple lines को comment कर सकते हैं।

```
/*
 This is a multi-line comment.
 It can span multiple lines.
```

```
 Useful for explaining complex code or commenting out blocks of code.
*/
console.log('Hello, World!');
```

JavaScript Output

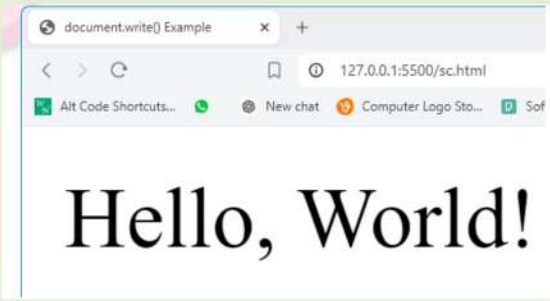
JavaScript में output generate करने के कई तरीके हैं। यह methods आपको user को information दिखाने, debugging करने, और interact करने में मदद करते हैं। यहां हम कुछ commonly used methods discuss करेंगे :

1. console.log(): console.log() का use console में message print करने के लिए होता है। यह method debugging के लिए बहुत useful है।

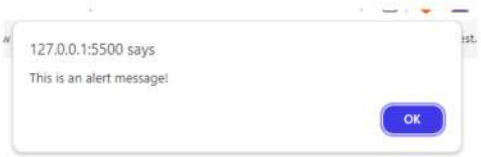
```
var name = 'Om';
console.log('Hello, ' + name); // Output: Hello, Om
```

2. document.write(): document.write() का use HTML document में directly content लिखने के लिए होता है। यह method mainly testing और learning के लिए useful है, लेकिन production code में avoid करना चाहिए क्यूंकि यह पुराने content को overwrite कर सकता है।

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>document.write() Example</title>
</head>
<body>
  <script>
    document.write('Hello, World!');
  </script>
</body>
</html>
```



3. alert(): alert() का use एक popup alert box display करने के लिए होता है। यह method user को information देने के लिए useful है, लेकिन frequent use से user experience ख़राब हो सकता है।



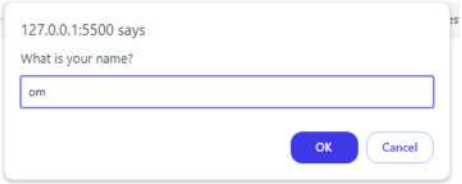

```
alert('This is an alert message!');
```

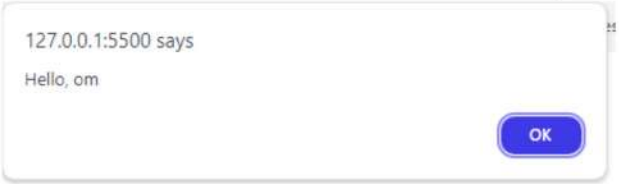
4. innerHTML: innerHTML property का use HTML element के content को set करने के लिए होता है। यह method dynamically web page update करने के लिए useful है।

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>innerHTML Example</title>
</head>
<body>
  <div id="output"></div>
```

```
  <script>
    document.getElementById('output').innerHTML = 'Hello, World!';
  </script>
</body>
</html>
```

5. prompt(): prompt() का use user से input लेने के लिए होता है। यह method एक dialog box display करता है जो user से input लेता है।

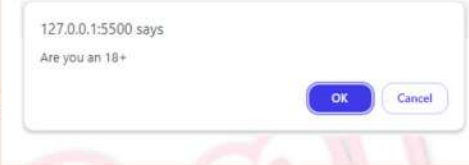





```
var name = prompt('What is your name?');
alert('Hello, ' + name);
```

6. confirm(): confirm() का use confirmation dialog display करने के लिए होता है। यह method एक dialog box display करता है जो user से confirmation लेता है (OK or Cancel).

```
var age = confirm('Are you an 18+');
    if (age) {
      console.log('user is an adult');
    } else {
      console.log('user is not adult');
    }
```



JavaScript Variables

JavaScript में variables का use data को store करने के लिए होता है। Variables को define करने के लिए var, let, या const keywords का use किया जाता है। आईये इसको detail में समझते हैं।

Declaring Variables: Variables को declare करने के लिए var, let, या const keywords का use करते हैं। let और const modern JavaScript (ES6) में introduce किये गए हैं।

var: var keyword का use करते हुए variable declare किया जाता है। यह globally या function scope में होता है।

```
var city = 'Mumbai';
console.log(city);          // Output: Mumbai
// Redeclaring with var is allowed
var city = 'Delhi';
console.log(city);          // Output: Delhi
```

let: let keyword का use block scope variable declare करने के लिए होता है। यह सिर्फ उस block के अंदर accessible होता है जहाँ यह declare किया गया है।

```
let country = 'India';
console.log(country);       // Output: India
// Redeclaring with let in the same scope is not allowed
// let country = 'USA';     // Error: Identifier 'country' has already been declared
country = 'USA'; // Allowed to update the value
console.log(country);       // Output: USA
```

const: const keyword का use constant values declare करने के लिए होता है। यह भी block scope होता है और इसका value reassign नहीं किया जा सकता।

```
const birthYear = 1990;
console.log(birthYear);        // Output: 1990
// Redeclaring and reassigning with const is not allowed
// const birthYear = 2000;     // Error: Identifier 'birthYear' has already been declared
// birthYear = 2000;           // Error: Assignment to constant variable (immutable).
```

Variable Naming Rules

- Variable name सिर्फ letters (a-z, A-Z), digits (0-9), underscores (_), और dollar signs ($) contain कर सकते हैं।
- Variable name number से start नहीं हो सकते।
- Variable names case-sensitive होते हैं, यानि name और Name अलग - अलग variables हैं।

Scope of Variables

- **Global Scope:** Variables जो function के बाहर declare किये गए हैं, वह globally accessible होते हैं।
- **Function Scope:** Variables जो function के अंदर declare किये गए हैं, वह सिर्फ उस function के अंदर accessible होते हैं।
- **Block Scope:** Variables जो block के अंदर (if, for, etc.) declare किये गए हैं, वह सिर्फ उस block के अंदर accessible होते हैं। यह feature सिर्फ let और const के साथ होता है।

Example of Scope

```
// Global scope
var globalVar = 'I am global';
function scopeExample() {
  // Function scope
  var functionVar = 'I am inside a function';
  console.log(globalVar);        // Output: I am global
  console.log(functionVar);      // Output: I am inside a function
}
scopeExample();
// console.log(functionVar);     // Error: functionVar is not defined
if (true) {
  // Block scope
  let blockVar = 'I am inside a block';
  console.log(blockVar);         // Output: I am inside a block
}
// console.log(blockVar);        // Error: blockVar is not defined
```

JavaScript Data Types

JavaScript में data types को दो main categories में divide किया जाता है : Primitive Data Types और Non-Primitive Data Types. हर category में different types के data होते हैं जिनका use आप अपने programs में कर सकते हैं।

Primitive Data Types: Primitive data types वह types होते हैं जो single value को represent करते हैं। इन्हे further categorize किया जा सकता है :

String: Strings text को represent करते हैं। Strings को single ('), double ("), या backticks (`) quotes में लिखा जाता है।

```
let name = "Om";
let greeting = 'Hello';
let message = `Welcome, ${name}`;
```

Number: Numbers को represent करते हैं, चाहे वह integer हो या floating-point.

```
let age = 25;
let price = 99.99;
```

Boolean: Boolean values true या false को represent करते हैं।

```
let isStudent = true;
let hasGraduated = false;
```

Undefined: Undefined tab होता है जब किसी variable को declare किया गया हो लेकिन उसको value assign नहीं की गई हो।

```
let x;
console.log(x);    // Output: undefined
```

Null: Null एक special value होती है जो "कोई value नहीं" को represent करती है।

```
let y = null;
```

Symbol: Symbol एक unique और immutable data type है जो mainly object properties को unique बनाने के लिए use होता है।

```
let sym = Symbol('description');
```

BigInt: BigInt large integers को represent करने के लिए use होता है जो normal numbers से बाहर होते हैं।

```
let bigNumber = BigInt(1234567890123456789012345678901234567890n);
```

Non-Primitive Data Types: Non-primitive data types complex structures को represent करते हैं। इनमें सबसे common data type objects होते हैं।

Object: Objects key-value pairs का collection होते हैं। Objects को curly braces {} के अंदर define किया जाता है।

```
let person = {
  name: "Om",
  age: 30,
  isStudent: true
};
```

Array: Arrays ordered collection of items को represent करते हैं। Arrays को square brackets [] के अंदर define किया जाता है।

```
let numbers = [1, 2, 3, 4, 5];
let mixedArray = [1, "hello", true, null];
```

Types of Operators

1. Arithmetic Operators: JavaScript में arithmetic operators का use basic mathematical calculations perform करने के लिए होता है। इन operators का use integers, floating-point numbers, और even strings के साथ भी किया जा सकता है। आईये इन्हे detail में समझते हैं।

a. Addition (+): Addition operator दो numbers को add करता है। अगर strings को add (concatenate) करने की कोशिश की जाये तो यह strings को concatenate करता है।

```
let a = 10;
let b = 5;
let result = a + b;
console.log(result);    // Output: 15
```

```
let firstName = "Om";
let lastName = "Mishra";
let fullName = firstName + " " + lastName;
console.log(fullName);    // Output: Om Mishra
```

b. Subtraction (-): Subtraction operator दो numbers को subtract करता है।

```
let a = 10;
let b = 5;
let result = a - b;
console.log(result);   // Output: 5
```

c. Multiplication (*): Multiplication operator दो numbers को multiply करता है।

```
let a = 10;
let b = 5;
let result = a * b;
console.log(result);    // Output: 50
```

d. Division (/): Division operator दो numbers को divide करता है।

```
let a = 10;
let b = 5;
let result = a / b;
console.log(result);   // Output: 2
```

e. Modulus (%): Modulus operator division का remainder return करता है।

```
let a = 10;
let b = 3;
let result = a % b;
console.log(result);   // Output: 1
```

f. Exponentiation (**): Exponentiation operator first operand को second operand के power में raise करता है (ES6 में introduce किया गया)।

```
let a = 2;
let b = 3;
let result = a ** b;
console.log(result);    // Output: 8
```

Arithmetic Operator Precedence: Operator precedence determines the order in which operators are evaluated in an expression. Operators with higher precedence are evaluated before operators with lower precedence.

Order	Operators	Associativity
1.	()	In to out
2.	**	Right to Left
3.	* , / , %	Left to Right
4.	+ , -	Left to Right

Example 1:

```
let x= 3+3*4*3/6/4*3-2;
// step1: 3+12*3/6/4*3-2
// step2: 3+36/6/4*3-2
// step3: 3+6/4*3-2
// step4: 3+1.5*3-2
// step5: 3+4.5-2
// step6: 7.5-2
console.log(x)   //5.5
```

Example 2:

```
let x= (13/4*3)%5+1;
// step1: (3.25 * 3) % 5 + 1
// step2: 9.75 % 5 + 1
// step3: 4.75 + 1
console.log(x)   //5.75
```

2. Assignment Operators: JavaScript में assignment operators का use variables को values assign करने के लिए होता है। सबसे common assignment operator = होता है, लेकिन कई और operators भी हैं जो assignment को arithmetic या bitwise operations के साथ combine करते हैं।

a. Basic Assignment Operator (=): Basic assignment operator right operand की value को left operand में assign करता है।

```
let x = 10;
console.log(x); // Output: 10
```

b. Compound Assignment Operators

Compound assignment operators arithmetic operation को assignment के साथ combine करते हैं। ये shorthand हैं एक operation perform करके result को वापस variable में assign करने के लिए।

Addition Assignment (+=)

```
let x = 10;
x += 5; // Equivalent to x = x + 5;
console.log(x); // Output: 15
```

Subtraction Assignment (-=)

```
let x = 10;
x -= 5; // Equivalent to x = x - 5;
console.log(x); // Output: 5
```

Multiplication Assignment (*=)

```
let x = 10;
x *= 5; // Equivalent to x = x * 5;
console.log(x); // Output: 50
```

Division Assignment (/=)

```
let x = 10;
x /= 5; // Equivalent to x = x / 5;
console.log(x); // Output: 2
```

Modulus Assignment (%=)

```
let x = 10;
x %= 3; // Equivalent to x = x % 3;
console.log(x); // Output: 1
```

Exponentiation Assignment (=)**

```
let x = 2;
x **= 3; // Equivalent to x = x ** 3;
console.log(x); // Output: 8
```

Bitwise AND Assignment (&=)

```
let x = 5; // (binary 0101)
x &= 3; // (binary 0011) Equivalent to x = x & 3;
console.log(x); // Output: 1 (binary 0001)
```

Bitwise OR Assignment (|=)

```
let x = 5; // (binary 0101)
x |= 3; // (binary 0011) Equivalent to x = x | 3;
console.log(x); // Output: 7 (binary 0111)
```

Bitwise XOR Assignment (^=)

```
let x = 5; // (binary 0101)
x ^= 3; // (binary 0011) Equivalent to x = x ^ 3;
console.log(x); // Output: 6 (binary 0110)
```

Bitwise Left Shift Assignment (<<=)

```
let x = 5; // (binary 0101)
x <<= 1; // Equivalent to x = x << 1;
console.log(x); // Output: 10 (binary 1010)
```

Bitwise Right Shift Assignment (>>=)

```
let x = 5; // (binary 0101)
x >>= 1; // Equivalent to x = x >> 1;
console.log(x); // Output: 2 (binary 0010)
```

Question: x - = y + 1 is same as: (choose any one)

(a.) x=x-y+1

(b.) x=x-y-1 //Ans: x=x-(y+1)

Bitwise Zero-Fill Right Shift Assignment (>>>=)

```
let x = -5; // (binary representation of negative numbers varies)
x >>>= 1; // Equivalent to x = x >>> 1;
console.log(x); // Output depends on binary representation
```

Right-to-Left Associativity: Assignment operators right-associative होते हैं, मतलब यह expressions को right से left की तरफ evaluate करते हैं।

```
let a, b, c;
a = b = c = 5;      // firstly c = 5, then b = c, and at last a = b.
console.log(a, b, c);     // Output: 5 5 5
```

3. Comparison Operators: JavaScript में comparison operators का use values को compare करने के लिए होता है। यह operators boolean (true या false) result देते हैं। आईये इन्हे detail में समझते हैं ।

a. Equal (==): यह operator दो values को compare करता है और type conversion (coercion) भी करता है।

```
console.log(5 == '5'); // Output: true, because it will consider 5 and '5' equal after type conversion.
```

b. Strict Equal (===): यह operator दो values को compare करता है बिना type conversion के।

```
console.log(5 === '5'); // Output: false, because one is number and another is string.
console.log(5 === 5); // Output: true, because both values and types are same.
```

c. Not Equal (!=): यह operator दो values को compare करता है और type conversion भी करता है। अगर values equal नहीं हैं, तो result true होता है।

```
console.log(5 != '5'); // Output: false, because it will consider 5 and '5' equal after type conversion.
```

d. Strict Not Equal (!==): यह operator दो values को बिना type conversion के compare करता है। अगर values और types दोनों different हैं, तो result true होता है।

```
console.log(5 !== '5'); // Output: true, because one is number and another is string.
console.log(5 !== 5); // Output: false, because both values and types are same.
```

e. Greater Than (>): यह operator check करता है की left operand right operand से बड़ा है या नहीं।

```
console.log(10 > 5); // Output: true
console.log(5 > 10); // Output: false
```

f. Greater Than or Equal (>=): ये operator check करता है की left operand right operand से बड़ा या उसके बराबर है या नहीं।

```
console.log(10 >= 5); // Output: true
console.log(5 >= 10); //Output: false
```

g. Less Than (`<`): यह operator check करता है की left operand right operand से छोटा है या नहीं।

```
console.log(5 < 10); // Output: true
console.log(10 < 5); // Output: false
```

h. Less Than or Equal (<=): यह operator check करता है की left operand right operand से छोटा या उसके बराबर है या नहीं।

```
console.log(5 <= 10); // Output: true
console.log(10 <= 5); // Output: false
```

Questions:

```
var x=5 , y=2 , z;
  z=x>y;
  console.log(z);  //true
```

```
var x=5, y=5,z;
  z=x>=y;
  console.log(z);  //true
```

```
var a=5, b=2, c=1, d;
  d=a>b>c;
  console.log(d);  //false
```

```
var x=5, y=2 ,z;
  z=x<y;
  console.log(z);  //false
```

```
var x=5, y=2,z;
  z=x!=y;
  console.log(z);  //true
```

```
var x;
  x=1<0==0;        //(1<0)==0
  console.log(x);  //true
```

```
var a=5, b=2, c=1, d;
   d=c+a>b;
   console.log(d);  //true
```

```
Arithmetic>Relation
c + a > b
1 + 5 > 2
6 > 2
```

* , / , %	↓	Arithmetic
+ , -		
> , < , >= , <=		Relational
== , !=		
=		Assignment

4. Logical Operators: JavaScript में logical operators का use logical operations perform करने के लिए होता है। यह operators boolean (true या false) values को combine करने या उनके साथ operations perform करने के लिए use होते हैं। आईये इन्हे detail में समझते हैं।

a. Logical AND (&&): यह operator तब true return करता है जब दोनों operands true होते हैं। अगर कोई भी operand false हो, तो यह false return करता है। (Double ampersand-Binary)

```
let a = true;
let b = false;
console.log(a && b);   // Output: false, because one operand is false.
```

```
let x = 5;
console.log(x > 0 && x < 10);   // Output: true, because both conditions are true.
```

b. Logical OR (||): यह operator तब true return करता है जब काम से काम एक operand true हो। अगर दोनों operands false हैं, तब यह false return करता है। (Double piped-Binary)

```
let a = true;
let b = false;
console.log(a || b);   // Output: true, because one operand is true.
```

```
let x = 5;
console.log(x > 0 || x > 10);   // Output: true, because first condition is true.
```

c. Logical NOT (!): यह operator एक operand को negate करता है। अगर operand true है, तो यह false return करता है, और अगर operand false है, तो यह true return करता है। (Unary)

```
let a = true;
console.log(!a);   // Output: false, because a is true.
```

```
let b = false;
console.log(!b);   // Output: true, because b is false.
```

Combining Logical Operators: आप multiple logical operators को combine करके complex logical expressions बना सकते हैं।

```
let x = 5;
let y = 10;
let z = 15;

console.log(x < y && y < z);  // Output: true, because both conditions are true.
console.log(x < y || y > z);  // Output: true, because first condition is true.
console.log(!(x < y));  // Output: false, because x<y is true and because of using NOT operator it became false.
```

Operator Precedence: Logical operators का अपना precedence होता है जो determine करता है की किस order में expressions evaluate होंगे। ! operator की precedence सबसे high होती है, फिर &&, और last में ||. Also→ (unary precedence > binary precedence)

Note: Any non zero value in case, it will be considered as true.

5. Increment Operator (++): Increment operator variable की value को 1 से बढ़ा देता है।

- **Postfix Increment (x++):** पहले current value return करता है, फिर increment करता है।
- **Prefix Increment (++x):** पहले increment करता है, फिर new value return करता है।

Example:

```
let a = 5;
console.log(a++);   // Output: 5, firstly value will be returned, then increment happens.
console.log(a);   // Output: 6, now after increment value became 6.
```

```
let b = 5;
console.log(++b);   // Output: 6, firstly increment will happen then new value is returned.
console.log(b);   // Output: 6, now after increment value became 6.
```

Question:

```
var x=5, y;
 y= x++ * x++;  //5*6
  console.log(x);  //7
  console.log(y);  //30
```

```
var x=5,y;
 y= ++x * x++;  //6*6
  console.log(x);  //7
  console.log(y);  //36
```

```
var x=5,y;
  y= ++x * ++x;  //6*7
   console.log(x);  //7
   console.log(y);  //42
```

```
var x=1,y;
  y= x++ * ++x * x++ * ++x * x++ * x++;  //1*3*3*5*5*6
   console.log(x);  //7
   console.log(y);  //1350
```

```
var x=5, z=6,y;
  y= ++z + x++;  //7+5
   console.log(x);  //6
   console.log(y);  //12
   console.log(z);  //7
```

6. Decrement Operator (--): Decrement operator variable की value को 1 से कम कर देता है।

- **Postfix Decrement (x--):** पहले current value return करता है , फिर decrement करता है।
- **Prefix Decrement (--x):** पहले decrement करता है , फिर new value return करता है।

Example:

```
let a = 5;
console.log(a--);   // Output: 5, firstly value will be returned, then decrement happens.
console.log(a);   // Output: 4, now after decrement value became 4.
```

```
let b = 5;
console.log(--b);   // Output: 4, firstly decrement will happen then new value is returned.
console.log(b);   // Output: 4, now after increment value became 4.
```

Note: Output of Pre and Post decrement/increment is exactly same in next line but whenever we are working in same line then they are different.

Question:

```
var x=5,y;
 y = --x * x-- * ++x * --x * x-- * ++x * x++ * x++;  //4*4*4*3*3*3*3*4
  console.log(x);  //5  (4 increment & 4 decrement so it will cancel out)
  console.log(y);  //20736
```

Precedence and Associativity: Increment और decrement operators की precedence और associativity को समझना important है।

- Pre-increment (++x) और Pre-decrement (--x) की precedence high होती है और यह right-to-left associative हैं।
- Post-increment (x++) और Post-decrement (x--) की precedence low होती है और यह left-to-right associative हैं।

7. Ternary Operator: JavaScript में ternary operator (conditional operator) का use एक concise way में conditional expressions लिखने के लिए होता है। यह एक short form है if-else statement का।

Syntax: **condition ? expression1 : expression2**

- **condition:** यह एक expression है जो true या false evaluate होता है।
- **expression1:** यह value return होती है अगर condition true हो।
- **expression2:** यह value return होती है अगर condition false हो।

Example:

```
let age = 18;
 let canVote = (age >= 18) ? "Yes, you can vote" : "No, you cannot vote";
  console.log(canVote); // Output: "Yes, you can vote"
```

- इस example में, अगर age 18 या उससे ज़्यादा है, तो canVote की value "Yes, you can vote" होगी। अगर age 18 से कम है, तो canVote की value "No, you cannot vote" होगी।

Nested Ternary Operator: आप nested ternary operators का use कर सकते हैं complex conditions handle करने के लिए। लेकिन, readability को ध्यान में रखते हुए इसे avoid करना चाहिए।

```
let score = 85;
let grade = (score >= 90) ? 'A' :
        (score >= 80) ? 'B' :
        (score >= 70) ? 'C' :
        (score >= 60) ? 'D' : 'F';
console.log(grade); // Output: "B"
```

Note: Associativity of ternary operator is Right to Left.

Questions:

How to find greatest of 3 numbers using conditional operator?

```
var a=5, b=3, c;
c= a>b ? a:b;
console.log(c);   //5
```

```
var a= 3 ? 4 ? 2 : 4 : 3 : 5;
 //There should be equal no. of ? and :
console.log(a);  //Error
```

How to find greatest of 3 numbers using conditional operator?

```
var a,b,c,d,e, short;
a=prompt('enter vale of a: ');
b=prompt('enter vale of b: ');
c=prompt('enter vale of c: ')
d=a>b ? a:b;
e=d>c ? d:c;
console.log(e);
short = a>b?a>c?a:c:b>c?b:c;
console.log(short);
```

```
var a= 10 ? 0 ? 2 : 3 : 1;
// 10 ? 0 ? 2 : 3 : 1;   (0 will be treated as false)
// 10 ? 3:1  (non zero value will be treated as true)
console.log(a);   //3
```

```
var a=2 ? 3 ? 4 : 5 : 7 ? 6 : 8;
// 2 ? 3 ? 4 : 5 : 6;
// 2 ? 4 : 6;
console.log(a);  //4
```

```
var a= 0 ? 7 ? 2 : 3 : 4;
// 0 ? 2 : 4;
console.log(a);  //4
```

```
var a=3 ? 2 : 1 ? 4 : 6;
// 3 ? 2 : 4;
console.log(a);  //2
```

```
var a= 3 ? 2 ? 5 : 4 ? 6 : 7 : 1;
// 3 ? 2 ? 5 : 6 : 1;
// 3 ? 5 : 1;
console.log(a);  //5
```

```
var a= 3 ? 2 ? 5 ? 4 : 1 : 8 : 9;
// 3 ? 2 ? 4 : 8 : 9;
// 3 ? 4 : 9;
console.log(a);  //4
```

8. Type Operator: JavaScript में typeof operator का use किसी variable या expression की data type check करने के लिए किया जाता है। यह एक unary operator है जो एक string return करता है जो data type को represent करता है।

Syntax: **typeof operand**

Example:

```
let number = 42;
console.log(typeof number);  // Output: "number"
```

```
let text = "Hello";
console.log(typeof text);  // Output: "string"
```

```
let isTrue = true;
console.log(typeof isTrue);  // Output: "boolean"
```

Note: JavaScript में, typeof null का result "object" होता है, जो एक historical bug है लेकिन backward compatibility के लिए ऐसे ही रखा गया है।

Using with Expressions

```
console.log(typeof (42 + " is a number"));    // Output: "string"
console.log(typeof (5 > 2));    // Output: "boolean"
```

Type Conversion

JavaScript में type conversion का मतलब है एक data type को doosre data type में convert करना। Type conversion दो तरीके से होता है : implicit (automatic) और explicit (manual).

Implicit Type Conversion: Implicit type conversion को type coercion भी कहा जाता है। JavaScript automatically data types को convert करता है जब different types के operands use किये जाते हैं।

Example: Implicit Conversion to String

```
let result;

result = '5' + 2;
console.log(result);   // Output: "52"

result = '5' + true;
console.log(result);   // Output: "5true"

result = '5' + null;
console.log(result);   // Output: "5null"
```

Example: Implicit Conversion to Number

```
let result;

result = '5' - 2;
console.log(result);   // Output: 3

result = '5' * 2;
console.log(result);   // Output: 10

result = '5' / 2;
console.log(result);   // Output: 2.5
```

Explicit Type Conversion: Explicit type conversion को typecasting भी कहा जाता है। यह conversion manually किया जाता है using built-in functions.

Converting to Number:

Using Number()

```
let result;

result = Number('5');
console.log(result);  // Output: 5

result = Number(true);
console.log(result);  // Output: 1

result = Number(false);
console.log(result);  // Output: 0

result = Number(null);
console.log(result);  // Output: 0

result = Number(' ');
console.log(result);  // Output: 0

result = Number('hello');
console.log(result);  // Output: NaN (Not a Number)
```

Using parseInt() and parseFloat()

```
let result;

result = parseInt('5');
console.log(result);  // Output: 5

result = parseInt('5.7');
console.log(result);  // Output: 5

result = parseFloat('5.7');
console.log(result);  // Output: 5.7
```

Converting to String

Using String()

```
let result;

result = String(123);
console.log(result); // Output: "123"

result = String(true);
console.log(result); // Output: "true"

result = String(null);
console.log(result); // Output: "null"
```

Using toString()

```
let result;

result = (123).toString();
console.log(result); // Output: "123"

result = (true).toString();
console.log(result); // Output: "true"
```

Converting to Boolean

Using Boolean()

```
let result;

result = Boolean(1);
console.log(result); // Output: true

result = Boolean(0);
console.log(result); // Output: false

result = Boolean('hello');
console.log(result); // Output: true

result = Boolean('');
console.log(result); // Output: false

result = Boolean(null);
console.log(result); // Output: false

result = Boolean(undefined);
console.log(result); // Output: false
```

Math Object

JavaScript में Math object built-in object है जो mathematical constants और functions provide करता है। यह object global scope में available होता है और इसको directly use किया जा सकता है। Math object को constructor के तरीके से use नहीं किया जा सकता, यानि आप Math का instance create नहीं कर सकते।

Common Math Methods

1. Math.PI: Pi का value return करता है (approximately 3.14159).

```
console.log(Math.PI); // Output: 3.141592653589793
```

2. Math.round(): Nearest integer पर round करता है।

```
console.log(Math.round(4.7)); // Output: 5
console.log(Math.round(4.4)); // Output: 4
```

3. Math.ceil(): Nearest higher integer (ऊपर के तरफ) round करता है।

```
console.log(Math.ceil(4.1)); // Output: 5
console.log(Math.ceil(4.9)); // Output: 5
```

4. Math.floor(): Nearest lower integer (नीचे के तरफ) round करता है।

```
console.log(Math.floor(4.9)); // Output: 4
console.log(Math.floor(4.1)); // Output: 4
```

5. Math.trunc(): Decimal part को remove करके integer part return करता है।

```
console.log(Math.trunc(4.9)); // Output: 4
console.log(Math.trunc(-4.9)); // Output: -4
```

6. Math.max(): Arguments में से maximum value return करता है।

```
console.log(Math.max(1, 3, 2)); // Output: 3
console.log(Math.max(-1, -3, -2)); // Output: -1
```

7. Math.min(): Arguments में से minimum value return करता है।

```
console.log(Math.min(1, 3, 2)); // Output: 1
console.log(Math.min(-1, -3, -2)); // Output: -3
```

8. Math.sqrt(): Square root return करता है।

```
console.log(Math.sqrt(16)); // Output: 4
console.log(Math.sqrt(2));  // Output: 1.4142135623730951
```

9. Math.pow(): Exponentiation function, यानि base और exponent return करता है।

```
console.log(Math.pow(2, 3)); // Output: 8 (2^3)
console.log(Math.pow(5, 2)); // Output: 25 (5^2)
```

10. Math.abs(): Absolute value return करता है।

```
console.log(Math.abs(-5)); // Output: 5
console.log(Math.abs(5));  // Output: 5
```

11. Math.random(): 0 (inclusive) और 1 (exclusive) के बीच एक pseudo-random number return करता है।

```
console.log(Math.random()); // Output: Random number between 0 and 1
```

12. Math.sin(), Math.cos(), Math.tan(): Trigonometric functions (radians में input लेते हैं)।

```
console.log(Math.sin(Math.PI / 2)); // Output: 1
console.log(Math.cos(0));           // Output: 1
console.log(Math.tan(Math.PI / 4)); // Output: 1
```

Question: Write a Program to input 4 digit number from keyboard & find sum of its digit.

```
let x;
x= prompt("Enter 4 Digit no.: ");
let d1 = x%10;
x= Math.floor(x/10);
//console.log("d = "+d1+" , x = "+x);
let d2 = x%10;
x=Math.floor(x/10);
//console.log("d = "+d2+" , x = "+x);
let d3 = x%10;
x=Math.floor(x/10);
//console.log("d = "+d3+" , x = "+x);
x= x + d3 + d2 + d1;
console.log(x);
```

x 1234 123 12 1

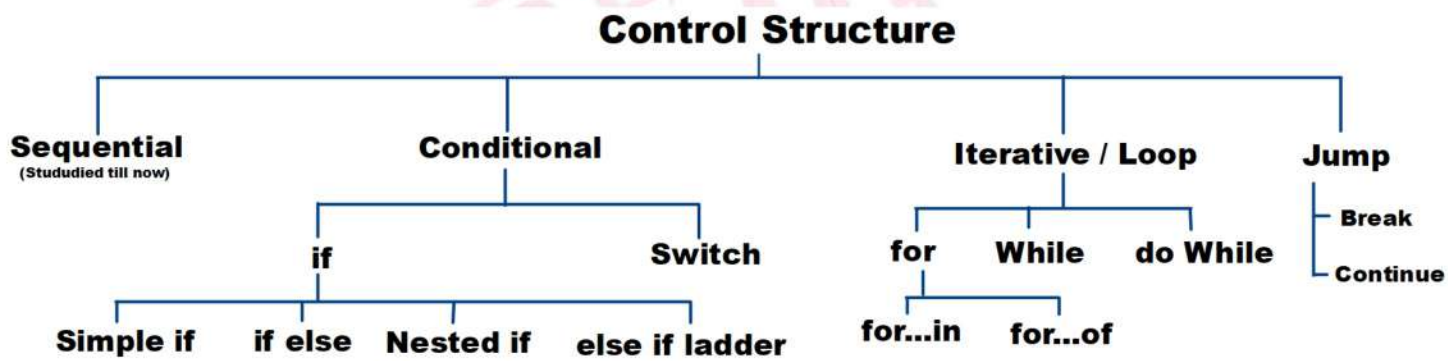
d1 4

d2 3

d3 2

Question: Write a Program to find reverse of 4 digit number.

```
let x;
x= prompt("Enter 4 Digit no.: ");
let d1 = x%10;
x= Math.floor(x/10);
//console.log("d = "+d1+" , x = "+x);
let d2 = x%10;
x=Math.floor(x/10);
//console.log("d = "+d2+" , x = "+x);
let d3 = x%10;
x=Math.floor(x/10);
//console.log("d = "+d3+" , x = "+x);
x= d1*1000 + d2*100 + d3*10 + x;
console.log(x);
```



Simple If Statement

JavaScript में if statement का use conditions check करने के लिए किया जाता है। अगर condition true होती है, तो code block execute होता है ; अगर false होती है, तो code block skip कर दिया जाता है।

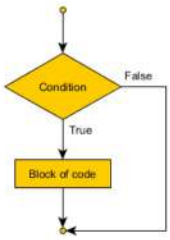
Syntax

```
if (condition) {
  // Code to be executed if the condition is true
}
```

Example: एक simple example देखते हैं जहाँ हम check करते हैं अगर एक number 10 से बड़ा है या नहीं:

```
let number = 15;
if (number > 10) {
  console.log("The number is greater than 10.");
}
// Output: "The number is greater than 10."
```

Om Shankar



Explanation

- if keyword को use करके एक condition check की जाती है।
- Parentheses (()) के अंदर condition लिखी जाती है।
- Curly braces ({}) के अंदर code block होता है जो execute होगा अगर condition true होती है।

Questions:

```
let x=5;
if(x==5){
   console.log("x is equal to 5");
   }
 //output: x is equal to 5
```

```
let x=5;
if(x==4);{  // ; it will not give any error
   console.log("x is equal to 5");
   }
   // output: x is equal to 5 (because check is ignored as ';' present)
```

```
let x=9;
if(x=5){  //non zero value is considered as true
   console.log("x is equal to 5");
}
 //Output: x is equal to 5
```

```
let x=9;
if(x==5)
   console.log("x is equal to 5");   //It will only run
   console.log("x is equal to 9");  // This will be out from if block
```

```
let number = 5;
if (number > 0) {
 console.log("The number is positive.");
 console.log("Now it is in if block);
}
```

```
let x= 0;
if(x=0){  //No output as 0 is false
   console.log("x is equal to 5");
   console.log("x is equal to 9");
}  // Output: "The number is positive."
```

Checking if a String is Empty

```
let str = "";
if (str === "") {
 console.log("The string is empty.");
}    // Output: "The string is empty."
```

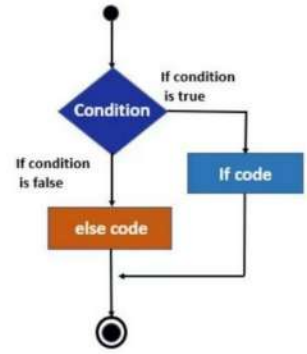
Checking if a Variable is Defined

```
let myVar;
if (typeof myVar !== "undefined") {
 console.log("myVar is defined.");
}     // Output: "myVar is not defined."
```

If-Else Statement: JavaScript में if-else statement का use conditions check करने के लिए किया जाता है। यह statement decision-making के लिए use होता है जहाँ आपको specific conditions के basis पर different code blocks execute करने होते हैं।

Syntax

```
if (condition) {
 // Code to be executed if the condition is true
} else {
 // Code to be executed if the condition is false
}
```



Examples

```
let x = 3;
if(x==5)
console.log("abc");
else
console.log("def");   //def
```

```
let x = 3;
if(x==5)
console.log("abc");
else
console.log("def");  //def
```

```
let x = 3;
if(x>3)
console.log("abc");
console.log("def");  //where if is closed there only
else should be placed
else
console.log("def");
   //output: error
```

```
let x = 3;
if(x>3);{   //scope ended
console.log("abc");
console.log("def");
}
else   //else is misplaced
console.log("def");
   //output: error
```

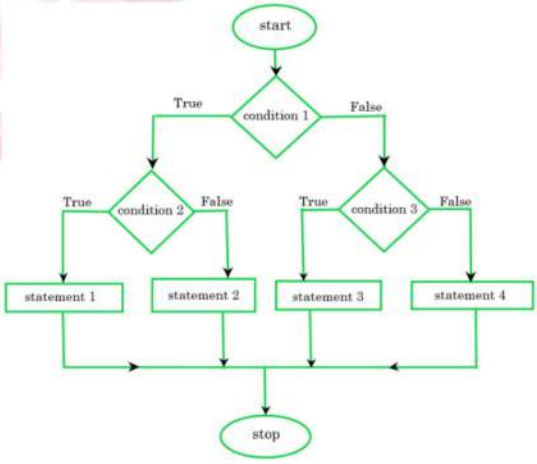
```
let x = 3;
if(x>3){
console.log("abc");
console.log("def");
}
else
console.log("def");   //def
```

```
let x=3;
if(x>3);
else
console.log("abc");
 //output: abc
```

Nested If Statement: JavaScript में nested if statements का use complex conditions check करने के लिए किया जाता है। Nested if statement का मतलब है एक if block के अंदर दूसरा if block लिखना। यह तब useful होता है जब आपको एक condition के true होने पर दूसरी condition भी check करनी होती है।

Syntax

```
if (condition1) {
  // Code to be executed if condition1 is true
  if (condition2) {
    // Code to be executed if condition1 and condition2 are true
  } else {
    // Code to be executed if condition1 is true and condition2 is false
  }
} else {
  // Code to be executed if condition1 is false
}
```



Questions:

Write a program to input marks of a student in 2 subjects, student is pass if marks in both subjects are greater than 50 otherwise fail.

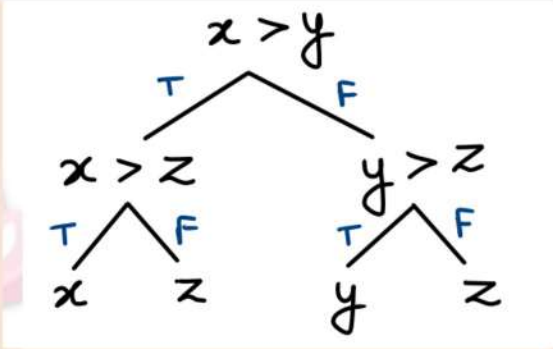
```
let m1,m2;
m1 = prompt("enter mark in first subject: ");
m2 = prompt("enter mark in second subject: ");
if(m1>50)
{
   if(m2>50)
      console.log("Student is Pass");
    else
      console.log("Student is Fail");
}
else
console.log("Student is Fail");
```

Write a program to input marks of a student in 3 subjects, student is pass if marks in all subjects are greater than 50, if any 2 then compartment, otherwise fail.

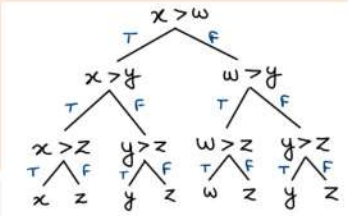
```
let m1, m2, m3;
m1 = prompt("enter mark in first Subject: ");
m2 = prompt("enter mark in second Subject: ");
m3 = prompt("enter mark in Third Subject: ");
if(m1>50){
   if(m2>50){
      if(m3>50)
      console.log("Pass");
      else
      console.log("compartment");
   }else
   if(m3>50)
   console.log("compartment");
   else
   console.log("Fail");
}
else
if(m2>50){
   if(m3>50)
   console.log("compartment");
   else
   console.log("Fail");
}
else
console.log("Fail");
```

Write a program for greatest of 3 number using nested if.

```
let x, y, z;
x = prompt("enter value of x: ");
y = prompt("enter value of y: ");
z = prompt("enter value of z: ");
if(x>y)
   if(x>z)
   console.log("x is greater");
   else
   console.log("z is greater");
else
   if(y>z)
   console.log("y is greater");
   else
   console.log("z is greater");
```



Algorithm for finding greatest of 4



Using Logical Operators with If-Else: आप logical operators (&&, ||, !) का use करके multiple conditions को combine कर सकते हैं।

Question:

Write a program to input marks of a student in 2 subjects, student is pass if marks in both subjects are greater than 50 otherwise fail.

```
if(m1>50 && m2>50)
console.log("Pass");
else
console.log("Fail");
```

Write a program to input marks of a student in 3 subjects, student is pass if marks in all subjects are greater than 50, if any 2 then compartment, otherwise fail.

```
if(m1>50 && m2>50 && m3>50)
console.log("Pass");
else
if(m1>50 && m2>50 || m1>50 && m3>50 || m2>50 && m3>50)
console.log("Compartment");
else
console.log("Fail");
```

Else If Ladder: JavaScript में else if ladder का use multiple conditions को check करने के लिए किया जाता है। else if ladder allow करता है आपको एक के बाद एक conditions check करने की और first true condition के corresponding code block को execute करने की। अगर कोई भी condition true नहीं होती, तो else block execute होता है।

Syntax

```
if(condition1){ → अगर ये expression चल गई तो control नीचे नही जाएगा
  // Code to be executed if condition1 is true
} else if(condition2){ → अगर ऊपर नही चली तब control यहा आएगा।
  // Code to be executed if condition2 is true
} else if(condition3){
  // Code to be executed if condition3 is true
} else {
  // Code to be executed if none of the conditions are true
}
```

and so on

↔

```
if(exp) {
  ≡
}else {
  if(exp) {
    ≡
  }else {
    if(exp) {
      ≡
    }else {
      ≡}
```

जहा एक से ज्यादा conditions लगे वहा else-if ladder का use करते हैं।

Example 1: Checking Grades Based on Score

Write a program to print grade of student if range of marks is given below:

- ***60 - 69 → D***
- ***70 - 79 → C***
- ***80 - 89 → B***
- ***90 - 99 → A***

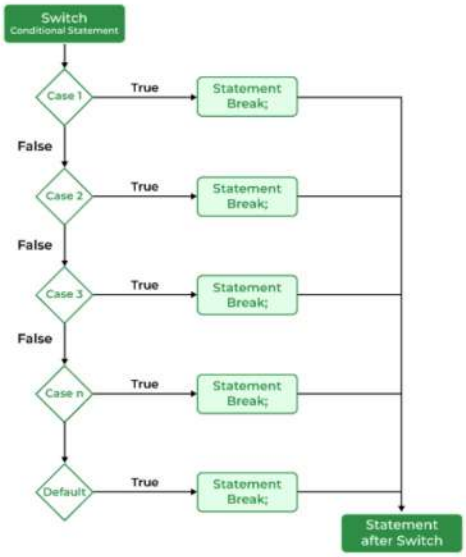
```
let m;
m=prompt("please enter score: ");
if(m>60 && m<=69){
   console.log("Grade: D");
}
else if(m>=70 && m<=79){
   console.log("Grade: C");
}
else if(m>=80 && m<=89){
   console.log("Grade: B");
}
else if(m>=90 && m<=99){
   console.log("Grade: A");
}
else
   console.log("Invalid Score");
```

Switch Statement

JavaScript में switch statement का use multiple conditions को check करने के लिए किया जाता है। यह else if ladder का एक alternative है जो specific values के against एक variable को match करता है। switch statement का use तब किया जाता है जब आपको एक variable के different values के लिए different code blocks execute करने होते हैं।

Syntax:

```
switch(expression) {
  case value1:
    // Code to be executed if expression === value1
    break;
  case value2:
    // Code to be executed if expression === value2
    break;
  // Add more cases as needed
  default:
    // Code to be executed if expression doesn't match any case
}
```



Break Statement: break statement हर case के बाद लिखा जाता है ताकि switch statement exit हो जाए और next cases को check न करे। अगर आप break statement नहीं लिखेंगे तो code execution अगले case block तक continue रहेगी।

Default Case: default case तब execute होती है जब कोई भी case match नहीं करता। यह optional होता है, लेकिन recommended है ताकि unexpected values handle हो सकें।

Example:

```
let x=1;
switch(x){
   case 1:
      console.log("January");
      break;
   case 2:
      console.log("Febraury");
      break;
   case 3:
      console.log("March");
      break;
   default:
   console.log("invalid");
}  Output: January
```

```
let x=3;
switch(x){
   case 1:
      console.log("January");
      break;
   case 2:
      console.log("Febraury");
      break;
      console.log("This will always be ignored");
   case 3:
      console.log("March");
      break;
   default:
   console.log("invalid");
}  Output: March
```

```
let x=3, y=3;
switch(x){
   case 1:
      console.log("January");
      break;
   case 2:
      console.log("Febraury");
      break;
   case y:
      console.log("March");
}  Output: March
//variable is supported
```

```
let x=2;
switch(x){
   case 2.0:
      console.log("abc");
      break;
   default:
      console.log("def");
}  Output: abc
//integer and float are treated same
```

```
let x=2.0;
switch(x){
   case 2:
      console.log("abc");
      break;
   default:
      console.log("def");
}  Output: abc
```

```
let x=4, y=3;
switch(x>y){
   case 1:
      console.log("True");
}  Output:
//wrong approach
```

```
let x='a';
switch(x){
   case 'a' || 'e' || 'i' || 'o' || 'u':
      console.log("vovel");
      break;
   default:
      console.log("Not Vovel");
}   Output: vovel
```

```
let x='a';
switch(x){
   case 'a':
   case 'e':
   case 'i':
   case 'o':
   case 'u':
      console.log("Vovel");
      break;
   default:
      console.log("Not Vovel");
}   // Output: Vovel
```

Questions:

Write a program to print grade of student if range of marks is given below:-

- ***60 - 69 → D***
- ***70 - 79 → C***
- ***80 - 89 → B***
- ***90 - 99 → A***

```
let x=70;
switch(Math.floor(x/10)){
  case 6:
    console.log("D");
    break;
  case 7:
    console.log("C");
    break;
  case 8:
    console.log("B");
    break;
  case 9:
    console.log("A");
    break;
  default:
  console.log("invalid");
}
```

Write a program to print grade of student if range of marks is given below:-

- ***61 - 70 → D***
- ***71 - 80 → C***
- ***81 - 90 → B***
- ***91 - 100 → A***

```
let x=81;
if(x%10==0)
x--;
switch(Math.floor(x/10)){
  case 6:
    console.log("D");
    break;
  case 7:
    console.log("C");
    break;
  case 8:
    console.log("B");
    break;
  case 9:
    console.log("A");
    break;
  default:
  console.log("invalid");
}
```

Write a program to find greatest of 2 no. using switch.

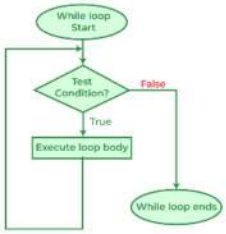
```
let x=50, y=100;
switch(x>y){
  case true:
    console.log("X is greater");
    break;
  case false:
    console.log("Y is greater");
}
```

Write a program to find greatest of 3 no. using switch.

```
let x=50, y=200 , z=150;
switch(x>y){
  case true:
    switch(x>z){
      case true:
        console.log("X is Greater");
        break;
      case false:
        console.log("Z is Greater");
    }
    break;
  default:
    if(y>z)
      console.log("Y is greater");
    else
      console.log("Z is greater");
}
```

While Loop

JavaScript में while loop का use repeatedly code execute करने के लिए किया जाता है जब तक एक specific condition true होती है। while loop को तब use किया जाता है जब हमें पता नहीं होता की loop कितनी बार चलेगा, लेकिन condition true होने तक repeatedly execute करना है।



Syntax

```
while (condition) {
 // Code to be executed as long as condition is true
}
```

Example

```
let i = 0;
while (i < 5) {
 console.log("The number is " + i);
 i++;
}
```

```
// Output:
// The number is 0
// The number is 1
// The number is 2
// The number is 3
// The number is 4
```

```
let i=0;
while (i<10){
   console.log("abc");
   i++;
} //output: 10 times abc
```

```
let i=0;
while (i<=10){
   console.log("abc");
   i++;
} //output: 11 times abc
```

```
let i=0;
while (i<10);{  //scope of control structure is of 1 line, it will loops for infinite times.
   console.log("abc");
   i++;
} //output: No output
```

```
let i=0;
while (i<10)  //so { is important
   console.log("abc");
   i++;
//output: infinite times abc
```

WAP to print counting from 1 to 10.

```
let i=1;
while (i<=10){
   console.log(i);
   i++;
}
```

```
let i=1;
while (i++ <= 10)
   console.log(i);   //2 3 4 5 6 7 8 9 10 11
   console.log(i);   //12
```

```
let i=1;
while (++i < 10)
   console.log(i);  //2 3 4 5 6 7 8 9
   console.log(i);  //10
```

```
let i=0;
while (i++ <= 10)
   console.log(i);   //1 2 3 4 5 6 7 8 9 10 11
   console.log(i);   //12
```

```
let i=1;
while (++i <= 10)
   console.log(i);  //2 3 4 5 6 7 8 9 10
   console.log(i);  //11
```

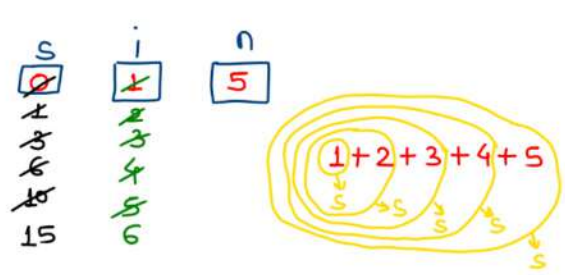
```
let i=1;
while (i++ <= 10);{  //if ++i then it will return i=12
   console.log("abc");  //abc
   i++;
   }
   console.log(i);  //13
```

What if we swap lines in body of loop?

```
let i=1;
while (i <= 10){
   i++;
   console.log(i);  //2 3 4 5 6 7 8 9 10 11
   }
```

WAP to find sum of 1st n natural number.

```
let n, i=1, s=0;
n = prompt("enter value: ");
while (i <= n){
   s=s+i;
   i++;
   }
   console.log(s);
```



Without Loop

```
let n, s=0;
n = parseInt(prompt("enter value: "));
s= n*(n+1)/2;
  console.log(s);
```

WAP to find sum of 1st n even natural no.

```
let i=2, n, s=0;
n=parseInt(prompt("Enter Value: "));
while(i<=2*n){
  s=s+i;
  i+=2;
}
console.log(s);
```

WAP to I/p any integer & find sum of its digits.

```
var n,s=0,d;
n = Number(prompt("Please enter any digit: "));
while (n!=0){
  d=n%10;
  n=Math.floor(n/10);
  s=s+d;
}
console.log(s);
```

Assignment 1: Find sum of 1st n even natural no. without loop.

Assignment 2: WAP to find sum of 1st n odd natural no. by while loop with dry run in paper.

WAP to print table of given number

```
let n, i=1;
n=parseInt(prompt("Enter Value: "));
while(i<=10){
  console.log(n+"*"+i+"="+(n*i));
  i++;
}
```

WAP to print factorial of given number.

```
let n, f=1, i=1;
n=parseInt(prompt("Enter Value: "));
while(i<=n){
  f=f*i;
  i++;
}
console.log(f);
```

Do-While Loop: JavaScript में do-while loop का use repeatedly code execute करने के लिए किया जाता है, लेकिन इस loop में condition को check करने से पहले code काम से काम एक बार execute होता है। do-while loop का use तब किया जाता है जब आपको ensure करना हो की loop body एक बार तो execute हो ही।

Syntax

```
do {
 // Code to be executed
} while (condition);   //Exit of control loop
```

Example:

```
let x=11;
while(x<=10)
console.log("abc");
```

```
let x=11;
do{
  console.log("abc");
} while(x<=10);  //output: abc
```

- do while is exit control loop but while is entry control loop.
- Its guaranteed that body of do while loop will be executed at least once.

```
do{
  console.log("abc");
} while(1);  //infinite loop
```



```
while(1){
  console.log("abc");
}
```

```
let x=1;
while(x++) {
   console.log("abc");
} //it is not a infinite loop but it will act like infinite loop, it will run upto max limit.
```

```
let x=1;
do{
   console.log("abc");
}
while(x++);
```

Note: while and do while loops have same power. Any program which can be made from while loop can also be made by do while loop and vise versa.

Using Break Statement: अगर आपको loop को prematurely terminate करना है, तो आप break statement का use कर सकते हैं।

```
let i = 0;

do {
 console.log("The number is " + i);
 if (i === 3) {
  break;    // Exiting the loop when i is 3
 }
 i++;
} while (i < 10);

// Output:
// The number is 0
// The number is 1
// The number is 2
// The number is 3
```

BigInt

JavaScript में एक single variable के अंदर maximum integer number जो store किया जा सकता है, उसका range predefined है। यह range JavaScript के number type के specification के according होता है, जो IEEE 754 double-precision floating-point format का use करता है।

Maximum Safe Integer: JavaScript में, Number type का range है :

- **Smallest positive value:** Number.MIN_VALUE (approximately 5e-324)
- **Largest positive value:** Number.MAX_VALUE (approximately 1.7976931348623157e+308)

लेकिन integers के case में, maximum safe integer जो accurately represent किया जा सकता है, वह Number.MAX_SAFE_INTEGER है। यह value है :

```
console.log(Number.MAX_SAFE_INTEGER); // Output: 9007199254740991
```

Similarly, minimum safe integer value जो accurately represent की जा सकती है, वह Number.MIN_SAFE_INTEGER है। यह value है :

```
console.log(Number.MIN_SAFE_INTEGER); // Output: -9007199254740991
```

Example: Storing Maximum Safe Integer

```
let maxInt = Number.MAX_SAFE_INTEGER;
console.log(maxInt); // Output: 9007199254740991
```

Storing a Value Beyond Maximum Safe Integer

```
let beyondMaxInt = Number.MAX_SAFE_INTEGER + 1;
console.log(beyondMaxInt); // Output: 9007199254740992
```

```
let anotherBeyondMaxInt = Number.MAX_SAFE_INTEGER + 2;
console.log(anotherBeyondMaxInt); // Output: 9007199254740992
```

Note: जब आप Number.MAX_SAFE_INTEGER के beyond values store करते हैं, तो आपको precision errors देखने को मिल सकते हैं।

Using BigInt for Larger Integers: अगर आपको Number.MAX_SAFE_INTEGER के beyond larger integers को accurately store करना है, तो आप BigInt का use कर सकते हैं। BigInt arbitrary precision integers को represent करता है और यह Number type के limitations को overcome करता है।

Example:

```
let bigIntValue = BigInt("9007199254740991") + BigInt("1");
console.log(bigIntValue); // Output: 9007199254740992n
```

```
let largeBigIntValue = BigInt("12345678901234567890123456789012345678 90");
console.log(largeBigIntValue); // Output: 12345678901234567890123456789012345678 90n
```

For Loop: JavaScript में for loop का use repeatedly code execute करने के लिए किया जाता है जब तक एक specific condition true होती है। for loop को tab use किया जाता है जब हमें पता होता है की loop कितनी बार चलेगा।

Syntax

```
for (initialization; condition; increment/decrement) {
  // Code to be executed
}
```

- **Initialization:** यह एक बार loop के start होने से पहले execute होता है। यहां आप counter variable को initialize करते हैं।
- **Condition:** हर iteration से पहले यह condition check की जाती है। अगर यह condition true होती है तो loop continue करता है, otherwise loop terminate हो जाता है।
- **Increment/Decrement:** हर iteration के बाद यह statement execute होता है। यहां आप counter variable को increment या decrement करते हैं।

```
for (let i = 0; i < 5; i++) {
  console.log("The number is " + i);
}
```

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
```

Infinite Loop: अगर for loop की condition कभी false नहीं होती, तो यह infinite loop बन जाता है। इससे बचने के लिए ensure करें की condition eventually false हो जाये।

```
for (let i = 0; ; i++) {
  console.log("This is an infinite loop!");
  if (i === 10) {
    break;   // Breaking the infinite loop
  }
}
```

```
This is an infinite loop!
This is an infinite loop!
This is an infinite loop!
This is an infinite loop!
This is an infinite loop!
This is an infinite loop!
This is an infinite loop!
This is an infinite loop!
This is an infinite loop!
This is an infinite loop!
This is an infinite loop!
```

Dynamic Loop Control: आप for loop के control parameters को dynamic values दे सकते हैं।

```
let start = 0;
let end = 5;
let step = 1;

for (let i = start; i < end; i += step) {
  console.log("The number is " + i);
}
```

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
```

Questions:

WAP to print abc 10 times using for loop.

```
let i;
for(i=1;i<=10;i++)
console.log("abc");
```

WAP to print counting from 1 to 10.

```
let i;
for(i=1;i<=10;i++)
console.log(i);
```

Nested For Loop: आप for loops को nest भी कर सकते हैं, यानी एक for loop के अंदर दूसरा for loop लिख सकते हैं।

```
for (let i = 0; i < 3; i++) {
 console.log("Outer loop iteration: " + i);
 for (let j = 0; j < 3; j++) {
  console.log("  Inner loop iteration: " + j);
 }
}
```

```
Outer loop iteration: 0
  Inner loop iteration: 0
  Inner loop iteration: 1
  Inner loop iteration: 2
Outer loop iteration: 1
  Inner loop iteration: 0
  Inner loop iteration: 1
  Inner loop iteration: 2
Outer loop iteration: 2
  Inner loop iteration: 0
  Inner loop iteration: 1
  Inner loop iteration: 2
```

Questions:

How many times abc will be printed?

```
for(let i=1;i<=10;i++){  //11 times executed
   for(let j=1;j<=10;j++){  //10 times executed
      console.log("abc");
   }
}
// Total 100 times abc will be printed.
```

```
//i=1 j--> 1 to 10 --> 10 times abc will printed
//i=2 j--> 1 to 10 --> 10 times abc will printed
.
.
.
.
//i=10 j--> 1 to 10 --> 10 times abc will printed
```

```
for (let i=1; i<=10; i++){
   for(let j=1;j<=10;j++)
   console.log("abc");
   console.log("123");
}
```

WAP to print table of numbers from 1 to 10.

```
for(let n=1;n<=10;n++){
   for(let i=1;i<=10;i++)
   console.log(n+"*"+i+"="+n*i);
   console.log("");
}
```

Tab Space (\t): Tab space का use code output में indentation देने के लिए होता है।

- **Alternate option:** **console.log(“”+“ ”+ “”);**

New Line Space (\n): New line space का use एक statement के बाद नए line में जाने के लिए होता है।

- **Alternate option:** **console.log();**

```
for(n=1;n<=10;n+=3){
   for(i=1;i<=10;i++)
  console.log(n+"*"+i+"="+n*i+"\t"+(n+1)+"*"+i+"="+(n+1)*i+"\t"+(n+2)+"*"+i+"="+(n+2)*i);
   console.log("\n");
}
```

```
1*1=1    2*1=2    3*1=3
1*2=2    2*2=4    3*2=6
1*3=3    2*3=6    3*3=9
1*4=4    2*4=8    3*4=12
1*5=5    2*5=10   3*5=15
1*6=6    2*6=12   3*6=18
1*7=7    2*7=14   3*7=21
1*8=8    2*8=16   3*8=24
1*9=9    2*9=18   3*9=27
1*10=10  2*10=20  3*10=30


4*1=4    5*1=5    6*1=6
4*2=8    5*2=10   6*2=12
```

Using Break Statement: अगर आपको loop को prematurely terminate करना है, तो आप break statement का use कर सकते हैं।

```
for(let i=1; i<=10; i++){
   if (i===5)
   break;
   console.log(i);
}
```

```
1
2
3
4
```

```
for(let i=1; i<=10; i++){
   console.log(i);
   if (i===5)
   break;
}
```

```
1
2
3
4
5
```

```
for(let i=1; i<=10; i++){
   console.log(i);
   break;
}
```

```
1
```

WAP to check given number is prime or not.

```
let n = parseInt(prompt("Please enter any digit: "), 10);   // Convert input to an integer
if (isNaN(n) || n <= 1) {
   console.log("Not Prime");   // Handle cases where input is less than or equal to 1
} else {
   let isPrime = true;   // Assume the number is prime
   for (let i = 2; i <= Math.floor(n / 2); i++) {
      if (n % i === 0) {
         isPrime = false;   // If divisible, not a prime number
         break;
      }
   }
   if (isPrime) {
      console.log("Prime");
   } else {
      console.log("Not Prime");
   }
}
```

Note you can take 3 conditions:
2 to n-1
2 to n/2
2 to Math.sqrt(n)

```
Please enter any digit: 7
Prime
```

Note:

Radix 10: JavaScript में parseInt function के second argument, 10 का मतलब है की आप number को decimal (base-10) system में parse करना चाहते हैं। यह 10 radix या base को specify करता है, जो बताता है की input string को किस base में interpret करना है।

Radix 2 से 36 तक : आप different bases specify कर सकते हैं, जैसे 2 binary के लिए, 8 octal के लिए, और 16 hexadecimal के लिए। Example: parseInt("1010", 2) इस string को binary number के रूप में parse करेगा, जो decimal में 10 के बराबर होता है।

WAP to print prime numbers between 1 & 500.

```
for (let n = 2; n <= 500; n++) {
   let isPrime = true;   // Assume the number is prime
   for (let i = 2; i <= Math.floor(n / 2); i++) {
      if (n % i === 0) {
         isPrime = false;   // If divisible, not a prime number
         break;
      }
   }
   if (isPrime) {
      console.log(n);
   }
}
```

Assignment: Explain dry run upto n<=10.

Using Continue Statement: continue statement का use current iteration को skip करने के लिए होता है और loop की next iteration को continue करता है।

```
for (let i = 0; i < 10; i++) {
  if (i % 2 === 0) {
    continue;   // Skipping the even numbers
  }
  console.log("The number is " + i);
}
```

```
The number is 1
The number is 3
The number is 5
The number is 7
The number is 9
```

```
for(let i=1; i<=10; i++){
   if(i==5)
   continue;
   console.log(i);
}
```

```
1
2
3
4
6
7
8
9
10
```

```
for (let i=1; i<=10; i++){
   console.log(i);
   if(i==5)
   continue;
}
```

```
1
2
3
4
5
6
7
8
9
10
```

```
for(let i=1; i<=10; i++){
   if(i==5)
   continue;
   if(i==5)
   break;
   console.log(i);
}
```

```
1
2
3
4
6
7
8
9
10
```

```
for(let i=1; i<=10; i++){
   if(i==5)
   continue;
   if(i==6)
   break;
   console.log(i);
}
```

```
1
2
3
4
```

WAP to accept item number, item name, rate and quantity. Find price (rate × qty). Calculate Discount (Disc) as follows

- If(pr>=1000 & <2000), Disc is 10% on pr
- If(pr>=2000 & <3000), Disc is 15% on pr
- If(pr>=3000 & <5000), Disc is 20% on pr
- If(pr>=5000), Disc is 25% on pr

❖ Calculate net pr (pr-disc)

❖ Display item number, item name, price (pr.), discount (disc.) and net price.

```
let num = Number(prompt("Please enter Item number: "));
let nam = prompt("Please enter Item name: ");
let rate = Number(prompt("Please enter Item rate: "));
let qua = Number(prompt("Please enter Item quantity: "));
let price = rate*qua;
let disc = 0;
console.log("Net ammount is: "+price);
if(price>=1000 && price<2000)
disc= price*0.1;
else if(price>2000 && price<3000)
disc= price*0.15;
else if(price>3000 && price<5000)
disc= price*0.2;
else if(price>=5000)
disc= price*0.25;
else
disc=0;
let net = price-disc;
console.log("Item number "+num+" "+nam+" is having Net payable amount of (after disc. of "+disc+"): "+net);
```

WAP to accept two number and display the following optional menu.

1. Add 2. Subtract 3. Multiply 4. Divide

```
let x= Number(prompt("Please enter first value: "));
let y= Number(prompt("Please enter second value: "));
let z= Number(prompt("please enter operation\n 1 to add\n 2 to subtract\n 3 to multiply\n 4 to divide\n"));
switch(z){
   case 1:
      console.log(x+"+"+y+"="+(x+y));
      break;
   case 2:
      console.log(x+"-"+y+"="+(x-y));
      break;
   case 3:
      console.log(x+"*"+y+"="+(x*y));
      break;
   case 4:
      console.log(x+"/"+y+"="+(x/y));
      break;
   default:
      console.log("Try again");
}
```

WAP to accept the current balance and display the following options/menu

1. Deposit 2. Withdraw 3. Display Balance 4. Exit

The menu should display repeatedly until the user chooses Exit.

```
let command, current=5000;
do{
command = Number(prompt("Welcome to Omi's Bank, Please select following option to proceed\n 1. Deposit \n 2. Withdraw\n 3. Display Balance \n 4. Exit\n"));
switch(command){
  case 1:
    let deposit=Number(prompt("Enter deposit ammount: "));
    current = current+deposit;
    break;
  case 2:
    let withdraw = Number(prompt("Please Enter withdraw ammount: "));
    if(current < withdraw)
    console.log("Insufficient Balance");
    else
    current=current-withdraw;
    break;
  case 3:
    console.log("Your current balance is "+current);
    break;
  case 4:
    console.log("Thankyou for using banking service");
    break;
  default:
    console.log("Invalid Choice");
  }
}
while(command !== 4);
```

JavaScript Function

JavaScript में functions एक core feature हैं जो code को reusable और modular बनाते हैं। Functions को आप define कर सकते हैं, उनको call कर सकते हैं, और multiple times use कर सकते हैं बिना code को बार-बार लिखे।

Syntax

```
function functionName(parameters) {
  // Function body: Code is written here.
  // code get executed when function is called
}
```

- **function:** यह keyword function को define करता है।
- **functionName:** Function का नाम होता है, जिसे call करके function को execute किया जाता है।
- **parameters:** यह optional है, और function के inputs होते हैं। Multiple parameters को comma से separate कर के दिया जाता है।
- **function body:** इसमें वह code होता है जो function call होने पर execute होता है।

Example

```
function greet(name) {
  console.log("Hello, " + name + "!");
}
// calling to Function
greet("Om");      // Output: Hello, Om!
greet("Shankar");      // Output: Hello, Shankar!
```

- greet() function एक parameter name accept करता है।
- console.log() statement उस नाम को greet करता है।
- Function को call करने पर greet("Om"), यह "Hello, Om!" print करेगा।

Function Return Type: JavaScript में functions एक value को return करने के लिए return statement का use करते हैं। यह value किसी भी type की हो सकती हैं, जैसे की number, string, array, object, या even another function. अगर function में return statement नहीं होता, तो default return type undefined होती हैं।

Case	Argument	Return type	Possibility
1	✓	✓	✓
2	✓	X	✓
3	X	✓	✓
4	X	X	✓

Syntax:

```
function functionName(parameters) {
   // Function body
   return value; // This value is returned by function.
}
```

Function with Default Parameters: JavaScript में आप default parameter values define कर सकते हैं अगर function call करते वक़्त कोई value न दी जाये।

```
function greet(name = "Guest") {
   console.log("Hello, " + name + "!");
}
greet();          // Output: Hello, Guest!
greet("Om");      // Output: Hello, Om!
```

- अगर greet() function को बिना parameter के call किया जाए, तो "Guest" default value use होती है।

Argument vs Parameter

Argument	Parameter
Argument function call करते वक़्त दिया जाता है।	Parameter function definition का हिस्सा होता है।
जब आप function को invoke (call) करते हैं, तो आप actual values provide करते हैं जिन्हे arguments कहते हैं।	जब आप function को define करते हैं, तो आप function के अंदर placeholders specify करते हैं जिसे parameters कहते हैं।
Arguments parameters की values को set करते हैं, और यह values function के अंदर use होती हैं।	Parameters function के inputs के रूप में काम करते हैं, जिन्हे function call करते वक़्त values assign की जाती हैं।
Example: `greet("Om");` • यहाँ "Om" एक argument है।	**Example:** `function greet(name){ }` • यहाँ name एक parameter है।

Return Type: JavaScript में functions को अलग -अलग तरीकों से define किया जा सकता है, और उनमें arguments और return types के combinations को handle करने के कई तरीके हैं। चलिए, इन combinations को समझते हैं :

Case1:

```
let x,y,z;
x=Number(prompt("Please enter first no.: "));
y=Number(prompt("Please enter second no.: "));
z=add(x,y);
console.log(z);
```

```
function add(a,b){
   let c=a+b;
   return c;
}
```

Case 2:

```
let x,y;
x=Number(prompt("Please enter first no.: "));
y=Number(prompt("Please enter second no.: "));
add(x,y);
function add(a, b){
  let c;
  c=a+b;
  console.log(c);
}
```

Case 3:

```
let z=add();
console.log(z);
function add(){
  let a,b,c;
  a=Number(prompt("Please enter first no.: "));
  b=Number(prompt("Please enter second no.: "));
  c=a+b;
  return c;
}
```

Case 4:

```
add();
function add(){
  let a,b,c;
  a=Number(prompt("Please enter first no.: "));
  b=Number(prompt("Please enter second no.: "));
  c=a+b;
  console.log(c);
}
```

Recursion in JavaScript: Recursion एक ऐसा technique है जिसमे एक function अपने आप को ही call करता है। यह programming में complex problems को solve करने का एक powerful tool है, जहाँ problem को smaller sub-problems में divide किया जाता है, जो की similar nature की होती हैं।

How Recursion Works

- **Base Case:** यह condition होती है जो recursion को stop करती है। जब base case मिल जाता है, तो function अपने आप को call करना बंद कर देता है।
- **Recursive Case:** यह condition होती है जिसमे function अपने आप को call करता रहता है। यहाँ पर problem को smaller sub-problems में divide किया जाता है।

Example:

```
function factorial(n) {
  // Base Case: if n is 1, then recursion will stop.
  if (n === 1) {
    return 1;
  }
  // Recursive Case: function call itself
  return n * factorial(n - 1);
}
let result = factorial(5);
console.log(result);      // Output: 120
```

- **Base Case:** जब n === 1 होता है, तो function 1 return करता है और recursion यहीं पर stop हो जाता है।
- **Recursive Case:** factorial(n - 1) function अपने आप को call करता रहता है जब तक base case नहीं मिलता।

Benefits of Recursion

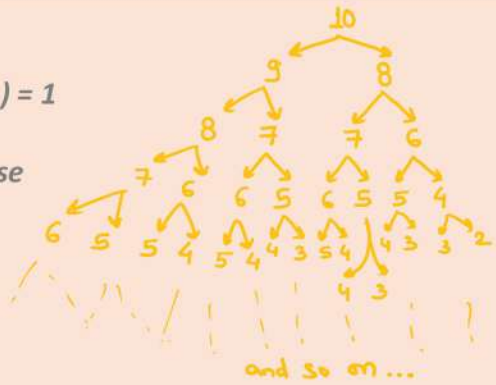
- **Simplifies Complex Problems:** Recursion complex problems को simpler sub-problems में तोड़ कर उन्हें solve करने में help करता है।
- **Elegant and Clean Code:** Recursive solutions अक्सर iterative (loop-based) solutions से ज़्यादा concise और readable होते हैं, especially tree-like structures, graphs, and mathematical computations में।

Risks of Recursion

- **Stack Overflow:** अगर base case define न किया हो या गलत हो, तो function infinite loop में चला जाता है, जो stack overflow error cause कर सकता है।
- **Performance Issues:** Recursion में हर function call के लिए stack memory use होती है, जो बहुत सारे recursive calls के case में performance को impact कर सकती है।

Example: Fibonacci series: Fibonacci series एक ऐसी sequence है जिसमे हर number अपने पिछले दो numbers का sum होता है। Series का first दो numbers हमेशा 0 और 1 होते हैं। इसके बाद हर अगला number previous दो numbers का sum होता है।

```
function fibonacci(n) {
  if (n <= 1) {
    return n;        // Base case: Fibonacci(0) = 0, Fibonacci(1) = 1
  }
  return fibonacci(n - 1) + fibonacci(n - 2);     // Recursive case
}
function printFibonacciSeries(n) {
  for (let i = 0; i < n; i++) {
    console.log(fibonacci(i));
  }
}
let n = 10;        // Change this value to print more numbers
console.log("Fibonacci series upto " + n + " terms:");
printFibonacciSeries(n);     // 0 1 1 2 3 5 8 13 21 34
```



Example: Find out GCD(Greatest Common Divisor) or HCF (Highest Common Factor).

```
function gcd(a, b) {
 // Base case: if b is 0, return a
 if (b === 0) {
  return a;
 }
 // Recursive case: GCD of b and the remainder of a divided by b
 return gcd(b, a % b);
}
console.log("GCD of 48 and 18:", gcd(48, 18));      // Output: GCD of 48 and 18: 6
```

Example: Check if a Number is a Power of Two

```
function isPowerOfTwo(n) {
 // Base case: if n is 1, return true
 if (n === 1) {
  return true;
 }
 // Base case: if n is less than 1 or not divisible by 2, return false
 if (n < 1 || n % 2 !== 0) {
  return false;
 }
 // Recursive case: check if n / 2 is a power of two
 return isPowerOfTwo(n / 2);
```

```
}
console.log("Is 16 a power of two?", isPowerOfTwo(16));     // Output: Is 16 a power of two? true
console.log("Is 18 a power of two?", isPowerOfTwo(18));     // Output: Is 18 a power of two? False
```

Functions as Variables: JavaScript में functions को एक variable में store किया जा सकता है, just like any other value (e.g., numbers, strings, etc.). इसको "function expression" कहते हैं। Functions को variables के रूप में treat करने से आप उन्हें easily pass, return, और manipulate कर सकते हैं। ये एक Anonymous function है जिसका कोई नाम नहीं है।

```
let greet = function(name) {
  console.log("Hello, " + name + "!");
};
// We are calling function through variable.
greet("Om");        // Output: Hello, Om!
```

Passing Functions as Arguments: JavaScript में functions को दूसरे functions में argument के रूप में pass कर सकते हैं। यह concept higher-order functions कहलाता है, और यह functional programming का एक important feature है।

1. Passing Function as an Argument: आप किसी function को दूसरे function में as an argument pass कर सकते हैं, जिसे callback function कहते हैं।

Example:

```
function add(x, y){     //Custom function: A simple operation to add two numbers
  return x+y;
}
function multiply(x, y){
  return x*y;
}
// Higher-order function: Applies an operation (callback) to two values
function calculate(operation, x, y){
  return operation(x,y);
}
let result = calculate(add, 3, 4);   // Passing the custom 'subtract' function as a callback
console.log(result);        //  7
```

2. Using Predefined Functions as Callbacks: आप किसी predefined function को भी as a callback pass कर सकते हैं।

Example:

```
// Even built-in functions can be used, like Math.max
let result3 = calculate(Math.max, 5, 10);
console.log(result3);        // Output: 10
```

The Arguments Object: The Arguments Object JavaScript में एक special object होता है जो functions के अंदर available होता है। इस object का नाम है arguments. यह object function के अंदर pass किये गए सभी arguments को represent करता है, चाहे आपने explicitly function definition में parameters define किये हो या नहीं।

```
function sum() {
  let total = 0;
  for (let i = 0; i < arguments.length; i++) {
    total += arguments[i];
  }
  return total;
}
console.log(sum(1, 2, 3, 4));      // Output: 10
console.log(sum(5, 10, 15));       // Output: 30
```

Explanation:

- **sum function:** यह function कितने भी arguments ले सकता है, और सभी arguments का sum return करता है।
- **arguments.length:** यह arguments object के अंदर कितने elements हैं, उसका count return करता है।
- **Looping through arguments:** हर argument को loop के through access करके total में add कर दिया।

call() Method: call() method किसी function को एक specific this value और individual arguments के साथ immediately invoke करता है।

Syntax: **function.call(thisArg, arg1, arg2, ...)**

```
function greet(greeting, punctuation) {
  console.log(greeting + ', ' + this.name + punctuation);
}
```

```
let person = { name: 'Om' };
greet.call(person, 'Hello', '!');  // Output: Hello, Om!
```

- यहां this value person object है, और greet function को immediately call किया गया है।

apply() Method: apply() method बिलकुल call() की तरह ही होता है, लेकिन यह arguments को एक array या array-like object के रूप में लेता है।

Syntax: **function.apply(thisArg, [argsArray])**

```
function greet(greeting, punctuation) {
  console.log(greeting + ', ' + this.name + punctuation);
}
```

```
let person = { name: 'Om' };
greet.apply(person, ['Hi', '.']);     // Output: Hi, Om.
```

- यहां this value person object है, और arguments एक array के form में pass किये गए हैं।

bind() Method: bind() method एक new function return करता है जिसमे fixed this value और optional arguments pre-defined होते हैं। यह function को immediately call नहीं करता, बल्कि एक नया function return करता है जो बाद में call किया जा सकता है।

Syntax: **function.bind(thisArg, arg1, arg2, ...)**

```
function greet(greeting, punctuation) {
  console.log(greeting + ', ' + this.name + punctuation);
}
```

```
let person = { name: 'Om' };
let greetPerson = greet.bind(person, 'Hey');
greetPerson('!');     // Output: Hey, Om!
```

- यहां bind() method एक new function return करता है जिसमे this value person object set हो चुकी है और एक argument (greeting) भी pre-defined है।

Arrow Functions: Arrow functions, JavaScript ES6 (ECMAScript 2015) का एक feature है, जो functions को concise और readable बनाते है। यह function expressions का short syntax provide करते हैं, और lexical this binding के लिए famous हैं।

Syntax: Arrow functions का syntax concise होता है और यह एक arrow (=>) का use करता है।

(param1, param2, ..., paramN) => expression

`let add = function(a, b) {  // Traditional function expression`
`  return a + b;`
`};`
	`let add = (a, b) => a + b;  // Arrow function`

No return or Curly Braces for Single Expressions: अगर function body में एक ही expression है, तो return और {} की ज़रुरत नहीं होती।

Example:

```
let square = x => x * 2;
console.log(square(4));    // Output: 8
```

- square arrow function है जो एक argument x accept करता है और x * 2 return करता है।
- Single parameter के case में parentheses () भी optional हैं।

Arrow Function with Multiple Lines: अगर function body में multiple statements हैं, तो आपको {} और return keyword का use करना होगा।

Example:

```
let sum = (a, b) => {
  let result = a + b;
  return result;
};
console.log(sum(5, 7));  // Output: 12
```

- यहाँ arrow function sum के body में multiple lines हैं, इसलिए curly braces {} और return keyword का use किया गया है।

Arrow Functions Without Parameters: अगर function कोई parameter accept नहीं करता, तो आपको empty parentheses () का use करना होगा।

Example:

```
let greet = () => console.log("Hello, World!");
greet();  // Output: Hello, World!
```

Lexical this Binding: Arrow functions में this keyword lexically bind होता है, मतलब arrow function का this हमेशा उस context का होगा जिसमे arrow function define किया गया है। यह behavior traditional functions से different है।

Example:

```
// Object with a method using a regular function
const person = {
  name: "Om",
  regularFunction: function() {
    console.log("Regular Function:", this.name);
  },
  arrowFunction: () => {
    console.log("Arrow Function:", this.name);
  }
};
// Calling the methods
person.regularFunction();    // Output: Regular Function: Om
person.arrowFunction();      // Output: Arrow Function: undefined
```

No arguments Object: Arrow functions में arguments object available नहीं होता। अगर आपको arguments object चाहिए, तो आपको traditional function expression का use करना होगा।

```
function add(x, y){
  console.log(arguments);    // [Arguments] { '0': 3, '1': 4 }
  return x+y;
}
add(3,4);
```

```
let showArgs = () => {
  console.log(arguments);
};
showArgs(1, 2, 3);  // Error: arguments is not defined
```

JavaScript Objects

JavaScript में objects एक ऐसे data structure हैं जो key-value pairs को store करते हैं। Objects allow करते हैं की आप related data और functions को एक organized form में store और manage कर सकें।

Creating an Object: JavaScript में objects को multiple ways से create किया जा सकता है :

1. Object Literal Notation: Object literal JavaScript में object बनाने का एक simple और concise method है। Object literals directly object create करते हैं by using curly braces {}.

```
let person = {
  firstName: "Om",
  lastName: "Mishra",
  age: 25,
  greet: function() {
    console.log("Hello, " + this.firstName + " " + this.lastName);
  }
};
```

- **Dot Notation:** यह सबसे common तरीका है किसी object की properties को access करने का। आप बस object का नाम लिखते हैं, उसके बाद एक dot (.) और फिर property का नाम लिखते हैं।

```
console.log(person.firstName);     // Output: Om
```

- **Bracket Notation:** Bracket notation तब useful होता है जब property names dynamic होते हैं या फिर valid identifiers नहीं होते (जैसे की spaces या special characters होते हैं)।

```
console.log(person["lastName"]);     // Output: Mishra
```

```
// Create an Object
const person = {};
// Add Properties
person.firstName = "Om";
person.lastName = "Mishra";
person.age = 27;
person.eyeColor = "black";
```

Adding and Modifying Properties: आप existing object में नए properties को add या modify कर सकते हैं :

```
person.middleName = "Shankar";     // Adding a new property
person.age = 27;          // Modifying an existing property
```

Deleting Properties: आप object properties को delete कर सकते हैं using the delete keyword:

```
delete person.middleName;
console.log(person.middleName);     // Output: undefined
```

Checking Property Existence: किसी property का object में exist करना check करने के लिए in operator या hasOwnProperty() method का use किया जा सकता है :

```
console.log("age" in person);          // Output: true
console.log(person.hasOwnProperty("age"));     // Output: true
```

2. Using the new Object() Constructor: new keyword का use करके आप dynamically objects create कर सकते हैं। यह constructor function को call करता है, जो नया object बनाता है।

```
let car = new Object();
car.company = "Toyota";
car.model = "Fortuner";
car.year = 2020;
```

Object Methods: Methods वो functions होते हैं जो objects के अंदर store होते हैं। Methods को हम object के actions या behaviors के रूप में समझ सकते हैं।

```
let car = {
  company: "Toyota",
  model: " Fortuner",
  year: 2020,
  start: function() {
    console.log("Car started");
  },
  drive: function(speed) {
    console.log("Car is driving at " + speed + " km/h");
  }
};
```

- start और drive यह methods हैं जो car के actions को define करते हैं।
- start method car start होने की action को represent करता है, और drive method speed के साथ drive होने की action को।

Calling Methods: आप methods को call कर सकते हैं by using the object name followed by a dot (.) and then the method name with parentheses ().

```
car.start();          // Output: Car started
car.drive(60);        // Output: Car is driving at 60 km/h
```

Nested Objects: Objects के अंदर दूसरे objects को भी store किया जा सकता है :

```
let student = {
  name: "Om",
  grades: {
    HTML: 90,
    CSS: 85
  }
};
console.log(student.grades.HTML);   // Output: 90
```

3. Using an Object Constructor: JavaScript में objects create करने का एक और तरीक़ा है using an Object Constructor. इस method में आप एक function define करते हैं जो एक blueprint की तरह काम करता है, और उसके through नए objects create किये जाते हैं।

Example:

```
function Person(firstName, lastName, age) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.age = age;
  this.greet = function() {
    console.log("Hello, " + this.firstName + " " + this.lastName);
  };
}
let person1 = new Person("Om", "Mishra", 27);
let person2 = new Person("Bhavya", "Singh", 28);
console.log(person1.firstName);     // Output: Om
person2.greet();          // Output: Hello, Bhavya Singh
```

4. Using Object.assign(): Object.assign() एक JavaScript method है जो एक या ज़्यादा source objects की properties को एक target object में copy करता है। यह method mostly object cloning और merging के लिए use होता है।

Syntax: **Object.assign(target, ...sources)**

- **target:** यह वो object है जिसमे properties को copy किया जाएगा।
- **sources:** एक या ज़्यादा source objects जिनसे properties को copy किया जाएगा।

Example 1: Cloning an Object

```
let person = {
  firstName: "Om",
  lastName: "Mishra",
  age: 27
};
let clone = Object.assign({}, person);
console.log(clone);    // Output: { firstName: "Om", lastName: "Mishra", age: 27 }
```

- इस example में, Object.assign() का use करके हमने person object का एक clone create किया है। clone object में person object की सारी properties copy हो गयी हैं।
- target object एक empty object {} है, और person source object है।

Example 2: Merging Multiple Objects

```
let person = {
  firstName: "Om",
  age: 27
};
let job = {
  title: "Developer",
  company: "Global Logic"
};
let contact = {
  email: "omshankar92229@gmail.com",
  phone: "89629XXXXX"
};
let merged = Object.assign({}, person, job, contact);
console.log(merged);
// Output: { firstName: "Om", age: 27, title: "Developer", company: "Global Logic", email: "omshankar92229@gmail.com", phone: "89629XXXXX" }
```

- इस example में, हमने person, job, और contact objects को एक merged object में combine किया है। सारे source objects की properties को target object {} में copy किया गया है।
- Resulting merged object में सभी source objects की properties available होती हैं।

Example 3: Overwriting Properties

```
let person = {
  firstName: "Om",
  lastName: "Mishra",
  age: 27
};
let update = {
  age: 28,
  lastName: "Betu"
};
let updatedPerson = Object.assign({}, person, update);
console.log(updatedPerson);    // Output: { firstName: "Betu", lastName: "Mishra", age: 28 }
```

- इस example में, update object की properties ने person object की corresponding properties को overwrite कर दिया है।
- अगर कोई property multiple source objects में होती है, तो जो last source object है उसकी property value target object में set होती है।

5. Using Object.create(): Object.create() JavaScript में एक method है जिसका use करके आप एक नया object create कर सकते हैं, जिसमे specified prototype object से properties inherit होती हैं। यह method mainly inheritance के concept को implement करने के लिए use होता है।

Syntax: **Object.create(proto, propertiesObject)**

- **proto:** यह object है जो नए object का prototype होगा। यह null भी हो सकता है अगर आप prototype chain को break करना चाहते हैं।
- **propertiesObject:** यह optional second argument हैं जो नयी properties को define करता हैं जो नए object में होनी चाहिए। यह properties वैसे ही define होती हैं जैसे Object.defineProperties() method में होती हैं।

Example 1: Basic Usage

```
let person = {
  greet: function() {
    console.log("Hello, " + this.name);
  }
};
let om = Object.create(person);
om.name = "Om";
om.greet();    // Output: Hello, Om
```

- person object एक simple object हैं जिसमे एक greet method defined हैं।

- om object को Object.create(person) के ज़रिये create किया गया हैं, जिसमे person object को as a prototype set किया गया हैं।
- om object में name property add की गयी हैं, और जब om.greet() call होता हैं, तो वो person object के greet method को inherit करके call करता हैं।

Example 2: Creating an Object with a Null Prototype

```
let obj = Object.create(null);
obj.name = "This object has no prototype";
console.log(obj);      // Output: { name: 'This object has no prototype' }
console.log(obj.toString);      // Output: undefined
```

- इस example में, obj object create किया गया हैं जिसका prototype null set किया गया हैं। इसका मतलब यह हैं की यह object किसी भी default JavaScript object properties को inherit नहीं करेगा, जैसे toString.
- यह technique कभी -कभी तब useful होती हैं जब आपको एक pure object create करना होता हैं जो prototype chain में नहीं आता।

Example 3: Defining Properties with Object.create()

```
let person = {
  greet: function() {
    console.log("Hello, " + this.name);
  }
};
let om = Object.create(person, {
  name: {
    value: "Om",
    writable: true,
    enumerable: true,
    configurable: true
  },
  age: {
    value: 27,
    writable: false,
    enumerable: true,
    configurable: true
  }
});
console.log(om.name);      // Output: Om
console.log(om.age);      // Output: 27
om.greet();      // Output: Hello, Om
```

- इस example में, Object.create() का second argument use किया गया हैं जिससे properties को define किया गया हैं।
- name और age properties को define करते वक़्त उनके descriptors भी set किये गए हैं जैसे writable, enumerable, और configurable.
- यह properties अब om object में exist करती हैं और वो person object का greet method भी inherit करता हैं।

Example 4: Inheritance Chain

```
let animal = {
  type: "Animal",
  sound: function() {
    console.log(this.type + " makes a sound.");
  }
};
let dog = Object.create(animal);
dog.type = "Dog";
dog.sound();      // Output: Dog makes a sound.
let labrador = Object.create(dog);
labrador.sound();      // Output: Dog makes a sound.
```

- इस example में, animal object create किया गया हैं जिसमे एक type property और एक sound method हैं।
- dog object को Object.create(animal) के ज़रिये create किया गया हैं जिससे वो animal object से properties और methods inherit करता हैं।
- labrador object को Object.create(dog) के ज़रिये create किया गया हैं, जिससे एक multi-level inheritance chain बनती हैं।

6. Using Object.fromEntries(): Object.fromEntries() JavaScript में एक built-in method हैं जो key-value pairs की list को एक object में convert करता हैं। यह method mainly तब useful होता हैं जब आपको किसी iterable (जैसे array या map) से एक object बनाना हो।

Syntax: Object.fromEntries(iterable)

- **iterable:** यह एक iterable object हैं, जैसे array या map, जिसमे हर element एक array हैं जिसमे दो elements होते हैं — first element key होती हैं और second element value होती हैं।

Example 1: Converting an Array of Key-Value Pairs into an Object

```
let entries = [
  ['name', Om],
  ['age', 27],
  ['city', 'Rewa']
];
let obj = Object.fromEntries(entries);
console.log(obj);      // Output: { name: 'Om', age: 27, city: 'Rewa' }
```

- entries एक array हैं जिसमे हर element एक key-value pair को represent करता हैं।
- Object.fromEntries(entries) से यह array एक object में convert हो जाता हैं जहाँ हर key array के first element से होती हैं और value second element से।

Object prototype: JavaScript में Object.prototype एक built-in object हैं जो सारे JavaScript objects का ancestor या parent होता हैं। इसका मतलब हैं की जब भी आप एक object create करते हैं, वो Object.prototype से inherit करता हैं।

What is Object Prototype?

Object.prototype एक JavaScript object हैं जो सारे objects में common properties और methods को define करता हैं। यह एक blueprint जैसा होता हैं जिसमे methods और properties define होते हैं जो हर object में accessible होते हैं।

Example:

```
let person = {
  firstName: "Om",
  lastName: "Mishra"
};
// Accessing a method from Object.prototype
console.log(person.toString());    // Output: [object Object]
```

- toString() method Object.prototype में defined होता हैं, और यह सारे JavaScript objects के लिए available होता हैं।
- person object के पास toString() method अपने prototype chain के through आता हैं।

Prototype Chain: जब आप किसी object की property या method को access करते हैं, JavaScript engine पहले उस object में उस property को ढूंढता हैं। अगर वो property वह नहीं मिलती, तो engine

उसके prototype में ढूंढता हैं, और यह process तब तक चलता हैं जब तक वो property मिल जाये या prototype chain end न हो जाये।

Custom Prototype Properties and Methods: आप Object.prototype में custom properties और methods को add कर सकते हैं, जो automatically सारे objects के लिए available हो जाते हैं।

Example:

```
Object.prototype.greet = function() {
  console.log("Hello, " + this.firstName);
};
let person1 = {
  firstName: "Om"
};
let person2 = {
  firstName: "Betu"
};
person1.greet();    // Output: Hello, Om
person2.greet();    // Output: Hello, Betu
```

- greet() method को Object.prototype में add किया गया हैं, इसलिए यह हर object के लिए available हैं।
- person1 और person2 objects greet() method को अपने Object.prototype से inherit करते हैं।

Overriding Prototype Properties: अगर आप किसी object में वही property या method define करते हैं जो Object.prototype में हैं, तो वो object उस method को override कर लेता हैं।

Example:

```
let person = {
  firstName: "Om",
  toString: function() {
    return "Person: " + this.firstName;
  }
};
console.log(person.toString());  // Output: Person: Om
```

- person object अपने toString() method को override कर रहा हैं, जो originally Object.prototype से आता हैं।

Common Methods of Prototype Chain:

- **toString():** यह method एक object को string में convert करता हैं।
- **hasOwnProperty():** यह method check करता हैं की एक object के पास specified property हैं या नहीं, without checking the prototype chain.
- **isPrototypeOf():** यह method check करता हैं की एक object दूसरे object का prototype हैं या नहीं।

Caution:

- Object.prototype को directly modify करना risky होता हैं, क्यूंकि यह सारे objects को affect कर सकता हैं।
- यह तभी करना चाहिए जब आपको अपने code में इसकी ज़रूरत हो और आप इसके side effects को समझ रहे हो।

Object.entries(object): Object.entries(object) JavaScript में एक built-in method हैं जो एक object के key-value pairs को array of arrays के form में return करता हैं। हर एक inner array में दो elements होते हैं : पहला element key होता हैं और दूसरा element उस key का value होता हैं।

Syntax: **Object.entries(object)**

Example:

```
let person = {
  name: "Om",
  age: 27,
  city: "Rewa"
};
let entries = Object.entries(person);
console.log(entries);
Output: [["name", "Om"], ["age", 27], ["city", "Rewa"]]
```

- Object.entries(person) method person object के सारे key-value pairs को array of arrays में convert कर देता हैं।
- इस array के हर element में दो values होती हैं : एक key और एक value.

Converting an Object to a Map: Object.entries() को आप Map object बनाने के लिए भी use कर सकते हैं।

```
let person = {
  name: "Om",
  age: 27,
  city: "Rewa"
};
let personMap = new Map(Object.entries(person));
console.log(personMap);     Output: Map(3) { 'name' => 'Om', 'age' => 27, 'city' => 'Rewa' }
```

1. JavaScript Object.defineProperty(): Object.defineProperty() method JavaScript में एक powerful method हैं जो एक object के property को define या modify करता हैं। इस method से आप एक property के behavior (जिसे "meta data" कहते हैं) को control कर सकते हैं, जैसे की property writable होगी या नहीं, enumerable होगी या नहीं, etc.

Syntax: **Object.defineProperty(obj, propertyName, descriptor)**

- **obj:** Object जिसमे property define करनी हैं।
- **propertyName:** Property का नाम जो आप define या modify करना चाहते हैं।
- **descriptor:** एक object जो property के behavior को define करता हैं।

Example:

```
let person = {};
// Adding a new property "name" with custom metadata
Object.defineProperty(person, "name", {
  value: "Om",
  writable: false,   // Can't be modified
  enumerable: true,   // Can be looped over
  configurable: false   // Can't be deleted or reconfigured
});

console.log(person.name);     // Output: Om
person.name = "Betu";     // Error in strict mode or simply ignored in non-strict mode
console.log(person.name);     // Output: Om (unchanged)
```

2. Changing Meta Data: Object.defineProperty() का primary use होता हैं एक property के metadata को control करना। Metadata कुछ attributes होते हैं जो property के behavior को define करते हैं :

- **writable:** अगर false हैं, तो property का value change नहीं हो सकता।
- **enumerable:** अगर false हैं, तो property loop में appear नहीं होगी (e.g., for...in loop).
- **configurable:** अगर false हैं, तो property को delete या reconfigure नहीं किया जा सकता।

Example:

```
let person = {};
Object.defineProperty(person, "name", {
  value: "Om",
  writable: true,
  enumerable: false,
  configurable: true
});
console.log(person.name);    // Output: Om
for (let key in person) {
  console.log(key);
 // Output: (nothing, because enumerable is false)
}
```

3. JavaScript getOwnPropertyNames(): Object.getOwnPropertyNames() एक object के सारे properties (including non-enumerable) के नाम को एक array में return करता हैं।

Syntax: Object.getOwnPropertyNames(obj)

- **obj:** Object जिसके properties के नाम चाहिए।

Example:

```
let person = {
  name: "Om",
  age: 27
};
Object.defineProperty(person, "gender", {
  value: "Male",
  enumerable: false
});
let propertyNames = Object.getOwnPropertyNames(person);
console.log(propertyNames);    // Output: ["name", "age", "gender"]
```

4. JavaScript Object.keys(): Object.keys() method एक object के सिर्फ enumerable properties के नाम को एक array में return करता हैं। इसमें non-enumerable properties को ignore किया जाता हैं।

Syntax: Object.keys(obj)

- **obj:** Object जिसके enumerable properties के नाम चाहिए।

Example:

```
let person = {
  name: "Om",
  age: 27
};
Object.defineProperty(person, "gender", {
  value: "Male",
  enumerable: false
});
let keys = Object.keys(person);
console.log(keys);  // Output: ["name", "age"]
```

JavaScript Accessors (Getters and Setters): JavaScript में Accessors (Getters and Setters) एक तरीका हैं जिससे आप object properties को get (retrieve) और set (modify) कर सकते हैं बिना directly उन्हें access किये। यह methods encapsulation provide करते हैं, जो object-oriented programming का एक important concept हैं।

Getters (Accessors):	Setters (Mutators):
Getters का use होता हैं एक property का value retrieve करने के लिए। यह एक method होता हैं जो एक property की तरह behave करता हैं, मतलब आप इसे बिना parentheses के call कर सकते हैं।	Getters का use होता हैं एक property का value retrieve करने के लिए। यह एक method होता हैं जो एक property की तरह behave करता हैं, मतलब आप इसे बिना parentheses के call कर सकते हैं।
Syntax: **get propertyName() {** **// return value** **}**	**Syntax:** **set propertyName(value) {** **// set value** **}**
Example: *let person = {* *firstName: "Om",* *lastName: "Mishra",* *get fullName() {* *return this.firstName + " " + this.lastName;* *}* *};* *console.log(person.fullName);* // Output: Om Mishra	**Example:** *let person = {* *firstName: "Om",* *lastName: "Mishra",* *set fullName(name) {* *let parts = name.split(" ");* *this.firstName = parts[0];* *this.lastName = parts[1];* *}* *};* *person.fullName = "Om Mishra";* *// Setter method called* *console.log(person.firstName);* // Output: Om *console.log(person.lastName);* // Output: Mishra

Combining Getters and Setters: आप एक ही object में getters और setters दोनों define कर सकते हैं, जो एक property को encapsulate करते हैं।

```
let person = {
  firstName: "Om",
  lastName: "Mishra",
  get fullName() {
    return this.firstName + " " + this.lastName;
  },
  set fullName(name) {
    let parts = name.split(" ");
    this.firstName = parts[0];
    this.lastName = parts[1];
  }
};
console.log(person.fullName); // Output: Om Mishra
person.fullName = "Shubham Payasi"; // Setter method
console.log(person.fullName); // Output: Shubham Payasi
```

Defining Getters and Setters Using Object.defineProperty(): आप Object.defineProperty() method का use करके भी getters और setters define कर सकते हैं।

```
let person = {
  firstName: "Om",
  lastName: "Mishra"
};
Object.defineProperty(person, "fullName", {
  get: function() {
    return this.firstName + " " + this.lastName;
  },
  set: function(name) {
    let parts = name.split(" ");
    this.firstName = parts[0];
    this.lastName = parts[1];
  }
});
console.log(person.fullName); // Output: Om Mishra
person.fullName = "Shubham Singh";
console.log(person.fullName); // Output: Shubham Singh
```

Free Billangual Notes Available !

HTML CSS JavaScript Angular React Hibernate

SpringBoot JDBC

Git & GitHub Data Analyst

www.linkedin.com/in/omi92229

Basics of HTML

What is HTML?

Definition: HTML का full form HyperText Markup Language है। यह एक standard language है जो web pages बनाने और design करने के लिए use होती है। यह web pages को structure provide करता है अलग -अलग elements और tags का उसे करके जो content और layout define करते हैं।

History: HTML को Tim Berners-Lee ने 1991 में बनाया था। यह समय के साथ evolve होता गया, और कई versions release हुए जो इसकी capabilities को enhance करते गए। अभी जो latest version है, वह HTML5 है, जो नए elements और features introduce करता है जिससे multimedia support, semantics, और applications और robust ho गए हैं।

Key Features:

- **Structure:** HTML headings, paragraphs, lists, links, और दूसरे content types का use करके web pages का structure define करता है।
- **Elements और Tags:** HTML tags का use करता है जो angle brackets में होते हैं (जैसे <tagname>). ज़्यादातर elements के पास opening tag और closing tag होता है (जैसे `<p>content</p>`), पर कुछ self-closing भी होते हैं (जैसे `<img src="image.jpg" />`).
- **Attributes:** Tags के पास attributes होते हैं जो element के बारे में additional information देते हैं (जैसे `<a href="https://example.com">link</a>`). Attributes opening tag के अंदर लिखे जाते हैं।
- **Hypertext:** HTML hyperlinks बनाने की सुविधा देता है, जो users को different web pages या resources के बीच navigate करने में मदद करता है।

Importance: HTML web development का foundation है। HTML के बिना web pages के पास basic

Data Science SQL

Mongo DB JAVA

Om Shankar Mishra

Data Science with R Programming

Lesson 1: Introduction to Business Analytics

Business analytics का मतलब है statistical analysis, data mining, और predictive modeling का use करके data को analyze और interpret करना, ताकि informed business decisions लिए जा सकें। यह modern business strategies का एक essential part है, जो valuable insights provide करता है, जिससे organizations market trends, customer preferences, और operational efficiencies को समझ सकते हैं।

Overview

Business analytics एक process है जो data को actionable insights में convert करता है, ताकि business outcomes को improve किया जा सके। यह various techniques और methodologies को encompass करता है जो historical data को समझने, future trends को predict करने, और data-driven decisions लेने में help करता है। इसका main goal है data को leverage करके business planning और strategic initiatives को drive करना।

Business Decisions and Analytics

Analytics, business decision-making में एक crucial role play करता है , क्यूंकि यह evidence-based insights provide करता है जो strategic planning, operational improvements, और competitive strategies को guide करता है। Analytics-supported business decisions से efficiency बढ़ सकती है, costs कम हो सकती हैं, customer satisfaction improve हो सकती है, और profitability बढ़ सकती है। Decision-making process में problem को identify करना, relevant data collect करना , data analyze करना, insights generate करना, और best course of action implement करना शामिल है।

Types of Business Analytics

Object protection: JavaScript में Object Protection Methods का use object properties को protect करने के लिए किया जाता हैं। यह methods आपको control देते हैं की आपके objects में properties add, modify, या delete हो सकती हैं या नहीं। इन methods से आप objects को more secure बना सकते हैं।

1. Object.preventExtensions(): Object.preventExtensions() method का use करके आप object को non-extensible बना सकते हैं, मतलब उस object में आप new properties add नहीं कर सकते।

```
let car = {
  make: "Toyota"
};
Object.preventExtensions(car);
car.model = "Corolla";    // Error or silently ignored in strict mode
console.log(car.model);    // Output: undefined (property wasn't added)
```

2. Object.isExtensible(): Object.isExtensible() method check करता हैं की एक object extensible हैं या नहीं, मतलब उसमे new properties add की जा सकती हैं या नहीं।

```
let car = {
  make: "Toyota"
};
console.log(Object.isExtensible(car));   // Output: true
Object.preventExtensions(car);
console.log(Object.isExtensible(car));  // Output: false
```

3. Object.seal(): Object.seal() method एक object को seal कर देता हैं, जिससे :

- आप new properties add नहीं कर सकते।
- Existing properties को delete नहीं कर सकते।
- लेकिन आप existing properties के values को modify कर सकते हैं।

```
let car = {
  make: "Toyota"
};
Object.seal(car);
car.model = "Corolla";   // Can't add new property
delete car.make;        // Can't delete property
car.make = "Honda";      // Can modify existing property
console.log(car);       // Output: { make: "Honda" }
```

4. Object.isSealed(): Object.isSealed() method check करता हैं की एक object seal किया गया हैं या नहीं।

```
let car = {
  make: "Toyota"
};
Object.seal(car);
console.log(Object.isSealed(car));    // Output: true
```

5. Object.freeze(): Object.freeze() method एक object को freeze कर देता हैं, जिससे :

- आप new properties add नहीं कर सकते।
- Existing properties को delete नहीं कर सकते।
- Existing properties के values को modify नहीं कर सकते।
- यह object को completely immutable (unchangeable) बना देता हैं।

```
let car = {
  make: "Toyota"
};
Object.freeze(car);
car.model = "Corolla";   // Can't add new property
delete car.make;        // Can't delete property
car.make = "Honda";     // Can't modify existing property
console.log(car);       // Output: { make: "Toyota" }
```

6. Object.isFrozen(): Object.isFrozen() method check करता हैं की एक object freeze किया गया हैं या नहीं।

```
let car = {
  make: "Toyota"
};
```

```
Object.freeze(car);
console.log(Object.isFrozen(car));    // Output: true
```

Arrays in JavaScript

JavaScript में array एक ordered collection होता है जिसमें multiple values को एक ही variable में store किया जा सकता है। यह values किसी भी data type की हो सकती हैं, जैसे numbers, strings, objects, या even दूसरे arrays. Arrays काफी flexible होते हैं और complex data structures बनाने में help करते हैं। कुछ key reasons arrays use करने के लिए :

```
let arr = ["om", 27, true, null, undefined];
```

- **Easy Data Management:** Arrays की help से आप data को efficiently manage कर सकते हैं, जैसे sorting, filtering, या traversing करना।
- **Flexible Data Structure:** Arrays dynamic होते हैं, इसका मतलब आप उनमें elements को easily add या remove कर सकते हैं बिना किसी hassle के।

Array Declaration: आप JavaScript में arrays को दो तरीको से declare कर सकते हैं :

1. Using Square Brackets ([]): यह सबसे common और simple तरीका है।

```
let fruits = ["Apple", "Banana", "Mango"];
```

2. Using the Array Constructor: इस method में Array constructor का use किया जाता है।

```
let fruits = new Array("Apple", "Banana", "Mango");
```

Accessing Array Elements: Array के elements को उनके index के through access किया जाता है। JavaScript arrays zero-based indexing use करते हैं, इसका मतलब first element का index 0 होता है।

```
let fruits = ["Apple", "Banana", "Mango"];
console.log(fruits[0]);    // Output: Apple
console.log(fruits[2]);    // Output: Mango
```

Access the Full Array: आप पूरे array को एक बार में access या print कर सकते हैं using console.log() या किसी और output method से। JavaScript में arrays automatically अपने elements को comma-separated format में print कर देते हैं।

```
let fruits = ["Apple", "Banana", "Mango"];
console.log(fruits);    // Output: ["Apple", "Banana", "Mango"]
```

Modifying Array Elements: आप array के किसी भी element को उसके index का use करके modify कर सकते हैं।

```
let fruits = ["Apple", "Banana", "Mango"];
fruits[1] = "Orange";    // Banana is replaced by Orange
console.log(fruits);    // Output: ["Apple", "Orange", "Mango"]
```

Converting an Array to a String: JavaScript में आप arrays को string में convert कर सकते हैं using different methods. यह काम तब आता है जब आपको array के सारे elements को एक single string format में convert करना हो।

1. toString() Method: toString() method array के सारे elements को comma-separated string में convert कर देता है।

```
let fruits = ["Apple", "Banana", "Mango"];
let fruitString = fruits.toString();
console.log(fruitString); // Output: "Apple,Banana,Mango"
```

2. join() Method: join() method भी array को string में convert करता है, लेकिन आपको custom separator define करने का option देता है।

```
let fruits = ["Apple", "Banana", "Mango"];
let fruitString = fruits.join(" - ");
console.log(fruitString);     // Output: "Apple - Banana - Mango"
```

The length Property: length property array के total number of elements को return करती है। यह array की size को check करने के लिए काम आती है।

```
let fruits = ["Apple", "Banana", "Mango"];
console.log(fruits.length);     // Output: 3
```

- **Note:** आप length property को modify कर के array के size को increase या decrease कर सकते हैं।

```
let fruits = ["Apple", "Banana", "Mango"];
fruits.length = 2;
console.log(fruits);     // Output: ["Apple", "Banana"]
```

Accessing the First Array Element: आप first element को access करने के लिए array का index 0 use करते हैं, क्यूंकि JavaScript arrays zero-based indexing follow करते हैं।

```
let fruits = ["Apple", "Banana", "Mango"];     console.log(fruits[0]);     // Output: Apple
```

Accessing the Last Array Element: Last element को access करने के लिए आप length property का use कर सकते हैं, क्यूंकि last element का index हमेशा length - 1 होता है।

```
let fruits = ["Apple", "Banana", "Mango"];     console.log(fruits[fruits.length - 1]); // Output: Mango
```

Looping Array Elements: JavaScript में आप arrays के elements को loop के ज़रिये easily traverse कर सकते हैं। आप different तरीको से array elements को loop कर सकते हैं।

1. for Loop: Traditional for loop का use करके आप array के elements को index के ज़रिये access कर सकते हैं।

```
let fruits = ["Apple", "Banana", "Mango"];
for (let i = 0; i < fruits.length; i++) {
  console.log(fruits[i]);
}
```

2. for...of Loop: for...of loop का use करके आप directly array के elements को access कर सकते हैं बिना index के।

```
let fruits = ["Apple", "Banana", "Mango"];
for (let fruit of fruits) {
  console.log(fruit);
}
```

3. forEach() Method: forEach() एक JavaScript method है जो array के हर element पर एक provided function को execute करता है। यह method original array को modify नहीं करता और कोई return value भी नहीं देता।

Example 1: with Anonymous Function

```
let numbers = [1, 2, 3, 4, 5];

numbers.forEach(function(number) {
  console.log("Number: " + number);
});
```

Explanation:

- **Array:** numbers नाम का एक array है जिसमे 1 से 5 तक के elements हैं।
- **forEach Method:** forEach() method को call किया गया numbers array पर। इस method में एक anonymous function pass किया गया है जो array के हर element पर execute होगा।
- **Anonymous Function:** यह function array के हर element को number parameter के through access करता है। हर element को console.log() के ज़रिये print किया जाता है with the prefix "Number: ".

Example 2: with Arrow Function

```
let fruits = ["Apple", "Banana", "Mango"];
fruits.forEach(fruit => console.log("Fruit: " + fruit));
```

- इस example में, arrow function fruit => console.log("Fruit: " + fruit) use किया गया है जो array के हर element को access करता है और उसको print करता है।

Questions:

how to initialize every element of array 5 if we have array size of 100.

```
let x = [];      // Initialize x as an empty array
for(let i = 0; i <= 99; i++) {
   x[i] = 5;      // Assign 5 to each element
}
console.log(x);
```

Take input of 10 arrays from user/keyboard:

```
let x=[];
for(let i=0; i<10; i++)
   x[i]=prompt("enter number "+(i+1)+": ");
 console.log(x);
```

write a program to input array and print in reverse order.

```
let x = [];
// Inputting 5 numbers into the array
for (let i = 0; i < 5; i++) {
   x[i] = Number(prompt("Enter number " + (i + 1) + ": "));
}
// Printing the array in reverse order
for (let i = x.length - 1; i >= 0; i--) {
   console.log("Number " + (i + 1) + " is: " + x[i]);
}
```

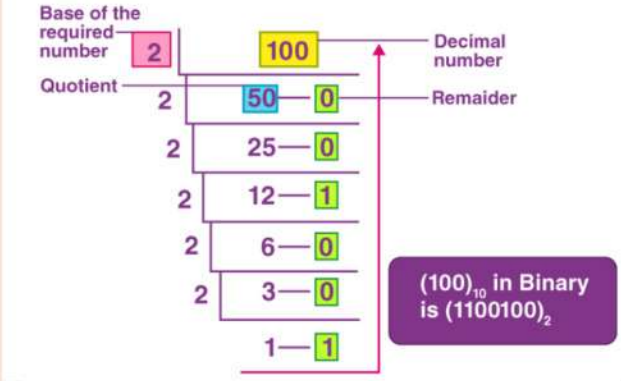
Write a program to find greatest of 5 numbers?

```
let x = [];
// Inputting 5 numbers into the array
for (let i = 0; i < 5; i++) {
   x[i] = Number(prompt("Enter number " + (i + 1) +
": "));
}
```

```
let t = x[0];
for (i=1;i<5;i++){
   if(x[i]>t)
   t=x[i];
}
console.log("The greatest number is "+t);
```

WAP to input the integer & convert it into binary using arrays.

```
let x = [], i=0,j;
let n = Number(prompt("enter any number: "));
while(n!=0){
   x[i]=n%2;
   n=Math.floor(n/2);
   i++;
}
for(let j=i-1; j>=0;j--){
   console.log(x[j]);
}
```



Without array?

```
let i=0,x,s=0;
let n = Number(prompt("enter any number: "));
while(n!=0){
   x=n%2;
   n=Math.floor(n/2);
   s= s+ Math.pow(10,i) *x;
   i++;
}
console.log(s);
```

2D Array: JavaScript में 2D arrays (arrays of arrays) को manage करना काफी useful होता है, specially तब जब आपको grids, matrices, या tables जैसे data structures को represent करना हो ।

2D Array Creation: एक 2D array को initialize करने के लिए, आप arrays के अंदर arrays का use करते हैं।

```
// 2D Array Initialization
let matrix = [
   [1, 2, 3],    // Row 1
   [4, 5, 6],    // Row 2
   [7, 8, 9]     // Row 3
];
```

- यहां matrix ek 3x3 का 2D array है जिसमें 3 rows और 3 columns हैं।

Accessing Elements in 2D Array: 2D array के elements को access करने के लिए आपको दो indices use करने पड़ते हैं :

- पहला index row को represent करता है।
- दूसरा index column को represent करता है।

```
// Accessing element in row 1, column 2
let element = matrix[0][1];
console.log(element);      // Output: 2
```

Modifying Elements in 2D Array: आप elements को modify भी कर सकते हैं similar syntax use करके

```
// Changing element in row 2, column 3
matrix[1][2] = 10;
console.log(matrix[1][2]);      // Output: 10
```

Iterating Over 2D Array: 2D array के elements को access या print करने के लिए आप nested loops का use करते हैं।

```
for (let i = 0; i < matrix.length; i++) {      // Loop over rows
   for (let j = 0; j < matrix[i].length; j++) {      // Loop over columns
      console.log(matrix[i][j]);           // Access each element
   }
}
```

Questions

WAP to find transpose of matrix of size 3×4

```
let x = [];
// Input the 3x4 matrix
for (let i = 0; i <= 2; i++) {
   x[i] = [];      // Initialize the inner array (row)
   for (let j = 0; j <= 3; j++) {
      x[i][j] = Number(prompt("Enter value for " + (i + 1) + " no. row and " + (j + 1) + " no. column: "));
   }
}
console.log("Original Matrix:");
console.log(x);
// Finding the transpose of the matrix
let transpose = [];
for (let i = 0; i < x[0].length; i++) {
   transpose[i] = [];
   for (let j = 0; j < x.length; j++) {
      transpose[i][j] = x[j][i];
   }
}
console.log("Transpose Matrix:");
console.log(transpose);
```

What if we don't know size of matrix?

```
let x = [];
console.log("Enter the size of matrix");
let r = Number(prompt("Enter size of row: "));
let c = Number(prompt("Enter size of column: "));
// Input the 3x4 matrix
for (let i = 0; i <= r-1; i++) {
   x[i] = [];      // Initialize the inner array (row)
   for (let j = 0; j <= c-1; j++) {
      x[i][j] = Number(prompt("Enter value for " + (i + 1) + " no. row and " + (j + 1) + " no. column: "));
   }
}
console.log("Original Matrix:");
console.log(x);
// Finding the transpose of the matrix
let transpose = [];
for (let i = 0; i <= c-1; i++) {
```

```
  transpose[i] = [];
  for (let j = 0; j <= r-1; j++) {
    transpose[i][j] = x[j][i];
  }
}
console.log("Transpose Matrix:");
console.log(transpose);
```

WAP to add 2 matrix

```
let x = [], y = [], z = [];
console.log("Enter Size of First Matrix");
let r1 = Number(prompt("Enter number of rows: "));
let c1 = Number(prompt("Enter number of columns: "));
// Input for first matrix
for(let i = 0; i < r1; i++) {
  x[i] = [];      // Initialize row
  for(let j = 0; j < c1; j++) {
    x[i][j] = Number(prompt("Enter value for " + (i + 1) + " no row and " + (j + 1) + " no column: "));
  }
}
console.log("Enter Size of Second Matrix");
let r2 = Number(prompt("Enter number of rows: "));
let c2 = Number(prompt("Enter number of columns: "));
// Input for second matrix
for(let i = 0; i < r2; i++) {
  y[i] = [];  // Initialize row
  for(let j = 0; j < c2; j++) {
    y[i][j] = Number(prompt("Enter value for " + (i + 1) + " no row and " + (j + 1) + " no column: "));
  }
}
// Display the first and second matrices
console.log("First matrix is:");
console.log(x);
console.log("Second matrix is:");
console.log(y);
```

Matrix Addition of 2*2 Matrices

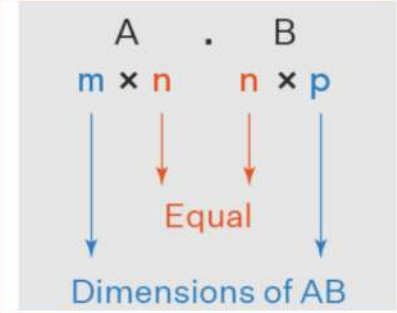
$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \end{bmatrix}$$

```
// Check if addition is possible
if (r1 !== r2 || c1 !== c2) {
  console.log("Matrix addition not possible since dimensions don't match.");
} else {
  // Initialize and calculate the result matrix
  for(let i = 0; i < r1; i++) {
    z[i] = [];      // Initialize row
    for(let j = 0; j < c1; j++) {
      z[i][j] = x[i][j] + y[i][j];
    }
  }
```

```
  // Display the result matrix
  console.log("The result matrix is:");
  console.log(z);
}
```

WAP to multiply two matrices.

```
let x = [], y = [], z = [];

console.log("Enter Size of First Matrix");
let r1 = Number(prompt("Enter number of rows: "));
let c1 = Number(prompt("Enter number of columns: "));

// Input for first matrix
for(let i = 0; i < r1; i++) {
  x[i] = [];     // Initialize row
  for(let j = 0; j < c1; j++) {
    x[i][j] = Number(prompt("Enter value for " + (i + 1) + " no row and " + (j + 1) + " no column: "));
  }
}

console.log("Enter Size of Second Matrix");
let r2 = Number(prompt("Enter number of rows: "));
let c2 = Number(prompt("Enter number of columns: "));

// Input for second matrix
for(let i = 0; i < r2; i++) {
  y[i] = [];     // Initialize row
  for(let j = 0; j < c2; j++) {
    y[i][j] = Number(prompt("Enter value for " + (i + 1) + " no row and " + (j + 1) + " no column: "));
  }
}
```




```
// Display the first and second matrices
console.log("First matrix is:");
console.log(x);
console.log("Second matrix is:");
console.log(y);

// Check if multiplication is possible
if (c1 !== r2) {
  console.log("Matrix multiplication not possible since the number of columns in the first matrix is not equal to the number of rows in the second matrix.");
} else {
  // Initialize and calculate the result matrix
  for(let i = 0; i < r1; i++) {
    z[i] = [];     // Initialize row
    for(let j = 0; j < c2; j++) {
      z[i][j] = 0;     // Initialize the element to 0
      for(let a = 0; a < c1; a++) {  // Iterate over the columns of the first matrix and the rows of the second matrix
        z[i][j] += x[i][a] * y[a][j];
      }
```

```
    }
  }
  // Display the result matrix
  console.log("The result matrix is:");
  console.log(z);
}
```

$$AB = \begin{bmatrix} 1 & 0 \\ -1 & 2 \\ 3 & 4 \end{bmatrix}_{3\times 2} \times \begin{bmatrix} 1 & -2 & 3 \\ 4 & 5 & 7 \end{bmatrix}_{2\times 3}$$

$$AB = \begin{bmatrix} 1\times 1+0\times 4 & 1\times(-2)+0\times 5 & 1\times 3+0\times 7 \\ -1\times 1+2\times 4 & -1\times(-2)+2\times 5 & -1\times 3+2\times 7 \\ 3\times 1+4\times 4 & 3\times(-2)+4\times 5 & 3\times 3+4\times 7 \end{bmatrix}_{3\times 3}$$

$$AB = \begin{bmatrix} 1 & -2 & 3 \\ 7 & 12 & 11 \\ 19 & 14 & 37 \end{bmatrix}_{3\times 3}$$

Adding Array Elements: आप JavaScript arrays में elements को dynamically add कर सकते हैं using different methods.

1. push() Method: push() method array के end में एक या ज़्यादा elements add करता है।

```
let fruits = ["Apple", "Banana"];
fruits.push("Mango");
console.log(fruits);    // Output: ["Apple", "Banana", "Mango"]
```

2. unshift() Method: unshift() method array के start में एक या ज़्यादा elements add करता है।

```
let fruits = ["Apple", "Banana"];
fruits.unshift("Grapes");
console.log(fruits);    // Output: ["Grapes", "Apple", "Banana"]
```

3. Direct Indexing: आप directly index specify करके भी array में element add कर सकते हैं।

```
let fruits = ["Apple", "Banana"];
fruits[2] = "Mango";
console.log(fruits);    // Output: ["Apple", "Banana", "Mango"]
```

Arrays are Objects: JavaScript में arrays actually objects होते हैं। यह इसलिए है क्यूंकि arrays special types of objects होते हैं जिसमें key-value pairs की जगह indexed elements होते हैं। इसलिए, आप arrays के साथ object-like behavior expect कर सकते हैं :

- Arrays के properties और methods होते हैं, जैसे length, push(), pop(), etc.
- Arrays dynamic होते हैं, इसका मतलब आप उनमें elements को add या remove कर सकते हैं बिना array को recreate किये।

```
let fruits = ["Apple", "Banana", "Mango"];
console.log(typeof fruits);    // Output: "object"
```

Array Elements Can Be Objects: Array के elements में आप किसी भी data type को store कर सकते हैं, including objects. इससे आप complex data structures बना सकते हैं जिसमें each element एक object हो सकता है।

```
let students = [
  { name: "Om", marks: [90, 85, 88] },
  { name: "Khushboo", marks: [92, 81, 79] },
  { name: "Priyanka", marks: [88, 77, 95] }
];
console.log(students[0].marks[1]);    // Output: 85
console.log(students[2].marks[2]);    // Output: 95
```

- यहाँ students array के हर element में एक object है जिसमें marks property एक array है। आप इस nested structure को dot और bracket notation के ज़रिये access कर सकते हैं।

Associative Arrays: JavaScript में arrays usually numerical indices (indexes) use करते हैं, लेकिन आप associative arrays भी बना सकते हैं जो keys को use करते हैं। However, JavaScript associative arrays को true arrays नहीं मानता, उनको objects के रूप में treat किया जाता है।

```
let person = [];
person["name"] = "Om";
person["age"] = 27;
console.log(person["name"]);    // Output: Om
```

- **Note:** Associative arrays को objects की तरह treat किया जाता है, और उनके साथ normal array operations (जैसे push, pop) काम नहीं करते।

When to use Arrays	When to use Objects
You need to store ordered data.	You need to store data in key-value pairs.
You are dealing with a list of items that will be accessed by index.	You are dealing with complex data structures where properties are important.
You want to use array methods like push, pop, map, filter, etc.	You need to easily look up values using keys.

How to Recognize an Array: JavaScript में check करने के लिए की कोई variable array है या नहीं, आप Array.isArray() method का use कर सकते हैं। यह method true return करता है अगर variable array हो, और false अगर नहीं हो।

```
let fruits = ["Apple", "Banana", "Mango"];
console.log(Array.isArray(fruits));    // Output: true
```

```
let person = { name: "Om", age: 30 };
console.log(Array.isArray(person));  // Output: false
```

Basic Array Methods: JavaScript arrays के साथ काम करने के लिए बहुत सारे useful methods available हैं। यह methods आपको arrays को modify, access, और manage करने में help करते हैं।

1. Array at(): at() method array के किसी specific index के element को return करता है। यह method negative index को भी support करता है, जिससे आप array के end से elements access कर सकते हैं।

```
let fruits = ["Apple", "Banana", "Mango"];
console.log(fruits.at(1));    // Output: "Banana"
console.log(fruits.at(-1));    // Output: "Mango"
```

2. Array pop(): pop() method array के last element को remove करता है और उस element को return करता है।

```
let fruits = ["Apple", "Banana", "Mango"];
console.log(lastFruit);    // Output: "Mango"
let lastFruit = fruits.pop();
console.log(fruits);  // Output: ["Apple", "Banana"]
```

3. Array delete: delete operator array के किसी specific element को delete करता है, लेकिन array की length को affect नहीं करता। यह method undefined छोड़ता है deleted element के जगह पर।

```
let fruits = ["Apple", "Banana", "Mango"];
delete fruits[1];
console.log(fruits);  // Output: ["Apple", undefined, "Mango"]
console.log(fruits.length);  // Output: 3
```

4. Array concat(): concat() method दो या ज़्यादा arrays को combine करता है और एक new array return करता है बिना original arrays को modify किये।

```
let fruits = ["Apple", "Banana"];
let moreFruits = ["Mango", "Grapes"];
let allFruits = fruits.concat(moreFruits);
console.log(allFruits);    // Output: ["Apple", "Banana", "Mango", "Grapes"]
```

10. Array copyWithin(): copyWithin() method array के एक part को उसके दूसरे part पर copy करता है बिना length को change किये। यह shallow copy create करता है।

Syntax: array.copyWithin(target, start, end);

- target: The index at which to copy the sequence to. If negative, it's counted from the end of the array.
- start (optional): The index from which to start copying elements. Default is 0.
- end (optional): The index at which to stop copying elements (exclusive). Default is the array's length.

Example 1: Basic Usage

```
let arr = [1, 2, 3, 4, 5];
arr.copyWithin(1, 3);  // Copy elements from index 3 to the end, to index 1
console.log(arr);     // Output: [1, 4, 5, 4, 5]
```

Example 2: Using Negative Indices

```
let arr = [1, 2, 3, 4, 5];
arr.copyWithin(-2, 0, 2);  // Copy elements from index 0 to index 2 (exclusive), to the last two positions
console.log(arr);    // [1, 2, 3, 1, 2]
```

11. Array flat(): flat() method nested arrays को flatten करता है, यानि उनके elements को एक single level array में merge करता है। आप depth specify कर सकते हैं।

```
let nestedArray = [1, [2, 3], [4, [5, 6]]];
let flatArray = nestedArray.flat(2);      //Depth is 2
console.log(flatArray);    // Output: [1, 2, 3, 4, 5, 6]
```

12. Array splice(): splice() method array से elements को add या remove करता है। यह method directly original array को modify करता है।

Syntax: array.splice(start, deleteCount, item1, item2, ..., itemN);

- start: The index at which to start changing the array. If negative, it indicates an offset from the end of the array.
- deleteCount: The number of elements to remove starting from the start index. If set to 0, no elements are removed.
- item1, item2, ..., itemN: Elements to add to the array, starting at the start index. This parameter is optional.

1. Removing elements:

```
let arr = ['a', 'b', 'c', 'd'];
arr.splice(1, 2);  // Removes 2 elements starting from index 1
console.log(arr);    // ['a', 'd']
```

2. Adding elements:

```
let arr = ['a', 'b', 'c', 'd'];
arr.splice(2, 0, 'x', 'y');  // Adds 'x' and 'y' at index 2
console.log(arr);     // ['a', 'b', 'x', 'y', 'c', 'd']
```

3. Replacing elements:

```
let arr = ['a', 'b', 'c', 'd'];
arr.splice(1, 2, 'x', 'y');    // Replaces 2 elements starting from index 1
console.log(arr);     // ['a', 'x', 'y', 'd']
```

13. Array toSpliced(): toSpliced() method एक new array return करता है जिसमें specified elements को add या remove किया गया हो, बिना original array को modify किये।

```
let fruits = ["Apple", "Banana", "Mango"];
let newFruits = fruits.toSpliced(1, 1, "Grapes");
console.log(newFruits);      // Output: ["Apple", "Grapes", "Mango"]
console.log(fruits);   // Output: ["Apple", "Banana", "Mango"]
```

14. Array slice(): slice() method array के एक part को return करता है एक new array के रूप में बिना original array को modify किये।

```
let fruits = ["Apple", "Banana", "Mango", "Grapes"];
let citrus = fruits.slice(1, 3);
console.log(citrus);      // Output: ["Banana", "Mango"]
```

Array Sort Methods: JavaScript में arrays को sort करने के लिए काफी सारे methods available हैं। यह methods arrays के elements को alphabetical या numerical order में arrange करते हैं।

Alphabetic Sort

1. Array sort(): sort() method array के elements को alphabetically sort करता है। By default, यह elements को string के रूप में compare करता है, चाहे array में numbers हो या strings ।

```
let fruits = ["Banana", "Apple", "Mango", "Grapes"];
fruits.sort();
console.log(fruits);      // Output: ["Apple", "Banana", "Grapes", "Mango"]
```

- **Note:** Numeric values को sort करते वक़्त यह method expected result नहीं दे सकता जब तक आप custom compare function provide न करें।

2. Array reverse(): reverse() method array के elements को reverse order में arrange करता है। यह array को पहले sort करके, फिर उसका order reverse करना useful होता है।

```
let fruits = ["Banana", "Apple", "Mango", "Grapes"];
fruits.reverse();
console.log(fruits);      // Output: ["Grapes", "Mango", "Apple", "Banana"]
```

3. Array toSorted(): toSorted() method एक new array return करता है जिसमें elements sorted होते हैं बिना original array को modify किये। यह method sort() का safer variant है।

```
let fruits = ["Banana", "Apple", "Mango", "Grapes"];
let sortedFruits = fruits.toSorted();
console.log(sortedFruits);      // Output: ["Apple", "Banana", "Grapes", "Mango"]
console.log(fruits);      // Output: ["Banana", "Apple", "Mango", "Grapes"]
```

4. Array toReversed(): toReversed() method एक new array return करता है जिसमें elements reverse order में होते हैं बिना original array को modify किये।

```
let fruits = ["Banana", "Apple", "Mango", "Grapes"];
let reversedFruits = fruits.toReversed();
console.log(reversedFruits);      // Output: ["Grapes", "Mango", "Apple", "Banana"]
console.log(fruits);      // Output: ["Banana", "Apple", "Mango", "Grapes"]
```

5. Sorting Objects: Objects के arrays को sort करते वक़्त आपको एक custom compare function देना पड़ता है। यह function दो objects को compare करता है और sorting order decide करता है।

```
let students = [
  { name: "Om", age: 27 },
  { name: "Priyanka", age: 29 },
  { name: "Khushboo", age: 22 }
];
students.sort((a, b) => a.age - b.age);
console.log(students);
```

```
[
  { name: 'Khushboo', age: 22 },
  { name: 'Om', age: 27 },
  { name: 'Priyanka', age: 29 }
]
```

Explanation:

sort() Method:

- यहाँ students.sort((a, b) => a.age - b.age); एक custom compare function pass किया गया है। यह function a और b के age properties को compare करता है।

Compare Function: (a, b) => a.age - b.age

- यह compare function array के हर दो adjacent elements को compare करता है।

- अगर a.age less than b.age होता है, तो function negative value return करता है, जिससे a element पहले आता है।
- अगर a.age greater than b.age होता है, तो function positive value return करता है, जिससे b element पहले आता है।
- अगर दोनों equal होते हैं, तो function 0 return करता है, और उनकी positions same रहती है।

Sorting Process:

- पहले Om और Priyanka compare होते हैं : 27 - 29 = -2, यह negative है, तो Om पहले रहता है।
- फिर Om और Khushboo compare होते हैं : 27 - 22 = 5, यह positive है, तो Khushboo को Om से पहले रखा जाता है।
- Finally, Priyanka और Khushboo compare होते हैं: 29 - 22 = 7, यह positive है, तो Khushboo पहले रहती है।

Numeric Sort

1. Numeric Sort: sort() method में जब numbers को sort करना हो, तो आपको custom compare function provide करना होता है जो numeric sorting ensure करे।

```
let numbers = [40, 100, 1, 5, 25, 10];
numbers.sort((a, b) => a - b);   // for ascending
console.log(numbers);      // Output: [1, 5, 10, 25, 40, 100]
numbers.sort((a, b) => b - a);  // for descending (we can also apply reverse after sorting)
console.log(numbers);      // Output: [ 100, 40, 25, 10, 5, 1 ]
```

2. Random Sort: अगर आप elements को random order में arrange करना चाहते हैं, तो sort() method के compare function में random logic use कर सकते हैं।

```
let numbers = [1, 2, 3, 4, 5];
numbers.sort(() => 0.5 - Math.random());
console.log(numbers);      // Output: Random order, e.g., [3, 5, 1, 4, 2]
```

3. Math.min(): Math.min() function array में से minimum value को return करता है, लेकिन इसके लिए array के elements को apply() के साथ pass करना पड़ता है।

```
let numbers = [40, 100, 1, 5, 25, 10];
let min = Math.min.apply(null, numbers);
console.log(min);      // Output: 1
```

4. Math.max(): Math.max() function array में से maximum value को return करता है, इसे भी apply() के साथ use करना पड़ता है।

```
let numbers = [40, 100, 1, 5, 25, 10];
let max = Math.max.apply(null, numbers);
console.log(max);      // Output: 100
```

5. Homemade Min(): आप अपना खुद का min() function बना सकते हैं जो array में से minimum value को find करे।

```
let numbers = [40, 100, 1, 5, 25, 10];
let min = function(arr) {
  return arr.reduce((a, b) => Math.min(a, b));
};
console.log(min(numbers));      // Output: 1
```

6. Homemade Max(): इसी तरह से, आप अपना खुद का max() function बना सकते हैं जो array में से maximum value को find करे।

```
let numbers = [40, 100, 1, 5, 25, 10];
let max = function(arr) {
  return arr.reduce((a, b) => Math.max(a, b));
};
console.log(max(numbers));      // Output: 100
```

Array Iteration Methods: JavaScript में array iteration methods का use हर array element पर operations perform करने के लिए होता है। यह methods array के हर item को process करते हैं और किसी specific logic को apply करने में help करते हैं।

1. Array forEach(): forEach() method array के हर element पर एक function execute करता है। यह method कुछ return नहीं करता, बस operation perform करता है।

```
let numbers = [1, 2, 3, 4, 5];
numbers.forEach(function(num) {
  console.log(num * 2);      // Output: 2, 4, 6, 8, 10
});
```

2. Array map(): map() method array के हर element पर एक function apply करता है और result को एक new array में return करता है। Original array modify नहीं होती।

```
let numbers = [1, 2, 3, 4, 5];
let doubled = numbers.map(num => num * 2);
console.log(doubled);      // Output: [2, 4, 6, 8, 10]
```

3. Array flatMap(): flatMap() method पहले array के हर element पर function apply करता है, फिर result को flatten करता है एक single-level array में।

```
let arr = [1, 2, 3];
let flatMapped = arr.flatMap(x => [x, x * 2]);
console.log(flatMapped); // Output: [1, 2, 2, 4, 3, 6]
```

4. Array filter(): filter() method array के elements को filter करता है based on a condition और एक new array return करता है जिसमें सिर्फ वह elements होते हैं जो condition को satisfy करते हैं।

```
let numbers = [1, 2, 3, 4, 5];
let evenNumbers = numbers.filter(num => num % 2 === 0);
console.log(evenNumbers);      // Output: [2, 4]
```

5. Array reduce(): reduce() method array के elements को reduce करता है एक single value में by applying a function to each element. यह method left से right process करता है।

Syntax: array.reduce(callback(accumulator, currentValue, currentIndex, array), initialValue);

- callback: एक function जो array के हर element पर execute होता है। इसमें ये arguments होते हैं :
- accumulator: पहले iteration से accumulated value.
- currentValue: Current element जो process हो रहा है array से।
- currentIndex (optional): Current element का index.
- array (optional): Array जिसपर reduce() call हो रहा है।
- initialValue (optional): एक value जो first argument (accumulator) के लिए use होती है first iteration में। अगर नहीं देते हो, तो पहला element accumulator बन जाता है, और iteration दूसरे element से start होता है।

```
let numbers = [1, 2, 3, 4, 5];
let sum = numbers.reduce((total, num) => total + num, 0);
console.log(sum);      // Output: 15
```

6. Array reduceRight(): reduceRight() method reduce() जैसा ही है, लेकिन यह right से left array के elements को process करता है।

```
let numbers = [1, 2, 3, 4, 5];
let product = numbers.reduceRight((total, num) => total * num, 1);
console.log(product);      // Output: 120
```

7. Array every(): every() method check करता है की array के सारे elements एक given condition को satisfy करते हैं या नहीं। यह method true return करता है अगर सारे elements condition satisfy करते हैं, वरना false.

```
let numbers = [2, 4, 6, 8];
let allEven = numbers.every(num => num % 2 === 0);
console.log(allEven);      // Output: true
```

8. Array some(): some() method check करता है की array में से कोई भी एक element given condition को satisfy करता है या नहीं। यह method true return करता है अगर एक भी element condition satisfy करे।

```
let numbers = [1, 3, 5, 8];
let hasEven = numbers.some(num => num % 2 === 0);
console.log(hasEven);      // Output: true
```

9. Array from(): Array.from() method किसी iterable या array-like object से एक new array create करता है।

```
let str = "hello";
let arr = Array.from(str);
console.log(arr);      // Output: ["h", "e", "l", "l", "o"]
```

10. Array keys(): keys() method एक array iterator return करता है जो array के indexes के keys provide करता है।

```
let fruits = ["Apple", "Banana", "Mango"];
let keys = fruits.keys();

for (let key of keys) {
   console.log(key);
}      // Output: 0, 1, 2
```

11. Array entries(): entries() method एक array iterator return करता है जो array के indexes के key-value pairs provide करता है।

```
let fruits = ["Apple", "Banana", "Mango"];
let entries = fruits.entries();

for (let entry of entries) {
   console.log(entry);
} // Output: [0, "Apple"], [1, "Banana"], [2, "Mango"]
```

12. Array with(): with() method एक नया array बनाता है जिसमें एक specified position पर नया element replace कर दिया जाता है बिना original array को modify किये।

```
let numbers = [1, 2, 3, 4, 5];
let newNumbers = numbers.with(2, 10);
console.log(newNumbers);   // Output: [1, 2, 10, 4, 5]
console.log(numbers);      // Output: [1, 2, 3, 4, 5]
```

13. Array Spread (...): Spread operator का use करके आप array के elements को separate कर सकते हैं। यह method new arrays बनाने या array elements को functions में pass करने के लिए useful है।

```
let numbers = [1, 2, 3];
let moreNumbers = [...numbers, 4, 5];
console.log(moreNumbers);  // Output: [1, 2, 3, 4, 5]

function sum(x, y, z) {
   return x + y + z;
}
console.log(sum(...numbers));      // Output: 6
```

Question 1: JavaScript Program to Find the Maximum and Minimum Element in an Array

```
function findMaxMin(arr) {
   let max = Math.max(...arr);    // Step 1: Find the maximum element
   let min = Math.min(...arr);      // Step 2: Find the minimum element
   return { max, min };          // Step 3: Return the result
}

let array = [5, 3, 8, 1, 9];       // Example input
let { max, min } = findMaxMin(array);
console.log(`The maximum element is: ${max}`);
console.log(`The minimum element is: ${min}`);
```

Question 2: JavaScript Program to Find the Sum of Elements in an Array

```
function sumOfArray(arr) {
  // Step 1: Calculate the sum using the reduce() method
  let sum = arr.reduce((acc, curr) => acc + curr, 0);
  return sum;      // Step 2: Return the sum
}
let array = [1, 2, 3, 4, 5];      // Example input
let totalSum = sumOfArray(array);
console.log(`The sum of elements in the array is: ${totalSum}`);
```

Question 3: JavaScript Program to Count Occurrences of a Specific Element in an Array

```
function countOccurrences(arr, element) {
  // Step 1: Use filter to count occurrences of the specific element
  const count = arr.filter(value => value === element).length;
  console.log(`${element} appears ${count} times`);      // Step 2: Display the result
}
const numbers = [1, 2, 2, 3, 4, 4, 5];      // Example usage
countOccurrences(numbers, 2);      // Output: 2 appears 2 times
```

Question 4: JavaScript Program to Count Occurrences of All Elements in an Array

```
function countAllOccurrences(arr) {
    const occurrences = arr.reduce((acc, value) => { // Step 1: Use reduce to count occurrences of each element
    acc[value] = (acc[value] || 0) + 1;
    return acc;
  }, {});
    for (const [element, count] of Object.entries(occurrences)) {    // Step 2: Display the result
    console.log(`${element} -> ${count}`);
  }
}
const fruits = ['apple', 'banana', 'apple', 'orange', 'banana'];      // Example usage
countAllOccurrences(fruits);
```

```
apple -> 2
banana -> 2
orange -> 1
```

Question 5: JavaScript Program to Count Occurrences of All Elements in an Array using forEach()

```
function countOccurrencesForEach(arr) {
  const countMap = {};
  arr.forEach(value => {    // Step 1: Use forEach to count occurrences of each element
    if (countMap[value]) {
      countMap[value]++;
    } else {
      countMap[value] = 1;
    }
  });
  for (let key in countMap) {    // Step 2: Display the result
    console.log(`${key} -> ${countMap[key]}`);
  }
}
const names = ['Om', 'Priyansi', 'Om', 'Khusboo', 'Priyansi'];      // Example usage
countOccurrencesForEach(names);
```

```
Om -> 2
Priyansi -> 2
Khusboo -> 1
```

JavaScript Array const: JavaScript में const keyword का use करके arrays को declare किया जा सकता है। const से declared array के reference को change नहीं किया जा सकता, लेकिन उसके elements को modify करना, add करना, या remove करना allowed है।

1. Declaring an Array with const: const keyword से array declare करते वक़्त उसकी reference fixed होती है। इसका मतलब यह है की आप उस array की memory location को किसी दूसरी value या array से overwrite नहीं कर सकते।

```
const fruits = ["Apple", "Banana", "Mango"];
```

2. Modifying Array Elements: आप const से declared array के elements को freely modify कर सकते हैं, elements add या remove कर सकते हैं। यह operation array की reference को change नहीं करता।

```
const fruits = ["Apple", "Banana", "Mango"];
fruits[0] = "Grapes";      // Apple will replace by Grapes
console.log(fruits);      // Output: ["Grapes", "Banana", "Mango"]
```

3. Adding and Removing Elements: const से declared array में आप नए elements push कर सकते हैं या existing elements को pop कर सकते हैं।

```
const fruits = ["Apple", "Banana"];
fruits.push("Mango");      // Mango is added to array
console.log(fruits);      // Output: ["Apple", "Banana", "Mango"]

fruits.pop();          // Last element Mango is removed
console.log(fruits);      // Output: ["Apple", "Banana"]
```

4. Reassigning the Array: आप const से declared array को दोबारा assign नहीं कर सकते। अगर आप ऐसा करते हैं, तो error throw होता है।

```
const fruits = ["Apple", "Banana"];
fruits = ["Grapes", "Orange"];      // Error: Assignment to constant variable
```

5. Why Use const for Arrays?: const keyword का use arrays को declare करने के लिए इसलिए किया जाता है ताकि accidentally array को पूरी तरह से reassign करने से बचाया जा सके। इससे यह ensure होता है की array की reference intact रहेगी, और आप उसके elements को safely modify कर सकते हैं।

```
const numbers = [1, 2, 3, 4, 5];
// valid
numbers[0] = 10;
numbers.push(6);

// invalid and will throw error
numbers = [10, 20, 30];
```

Other Arrays Methods:

1. fill(): Array के हर element को एक specific value से fill कर देता है।

```
let arr = [1, 2, 3, 4];
arr.fill(0);      // All elements will become 0.
console.log(arr);      // Output: [0, 0, 0, 0]
```

2. find(): Array में पहला element return करता है जो condition को satisfy करता हो।

```
let arr = [5, 12, 8, 130, 44];
let found = arr.find(element => element > 10);
console.log(found);      // Output: 12
```

3. findIndex(): Array में पहले element का index return करता है जो condition को satisfy करता हो।

```
let arr = [5, 12, 8, 130, 44];
let index = arr.findIndex(element => element > 10);
console.log(index);          // Output: 1
```

4. findLast(): Array में last element return करता है जो condition को satisfy करता हो (find method का reverse).

```
let arr = [5, 12, 8, 130, 44];
let found = arr.findLast(element => element > 10);
console.log(found);          // Output: 44
```

5. findLastIndex(): Description: Array में last element का index return करता है जो condition को satisfy करता हो।

```
let arr = [5, 12, 8, 130, 44];
let index = arr.findLastIndex(element => element > 10);
console.log(index);          // Output: 4
```

6. includes(): Check करता है की कोई specific element array में exist करता है या नहीं।

```
let arr = [1, 2, 3];
console.log(arr.includes(2));        // Output: true
console.log(arr.includes(4));        // Output: false
```

7. indexOf(): Array में पहला occurrence का index return करता है।

```
let arr = [2, 9, 9];
console.log(arr.indexOf(9));         // Output: 1
```

8. lastIndexOf(): Array में last occurrence का index return करता है।

```
let arr = [2, 9, 9];
console.log(arr.lastIndexOf(9));     // Output: 2
```

9. of(): एक new array create करता है specified elements के साथ।

```
let arr = Array.of(7, 3, 5);
console.log(arr);            // Output: [7, 3, 5]
```

10. prototype: JavaScript objects के properties और methods को extend करने के लिए use होती है।

```
Array.prototype.myCustomMethod = function() {
  console.log("This is a custom method!");
};
let arr = [1, 2, 3];
arr.myCustomMethod();        // Output: This is a custom method!
```

11. shift(): Array के first element को remove करता है और return करता है।

```
let arr = [1, 2, 3];
let firstElement = arr.shift();
console.log(firstElement);           // Output: 1
console.log(arr);            // Output: [2, 3]
```

12. values(): Array के iterator को return करता है जो हर element को iterate करता है।

```
let arr = ['a', 'b', 'c'];
let iterator = arr.values();
for (let value of iterator) {
  console.log(value);        //Output: a  b  c
}
```

13. valueOf(): Array का primitive value return करता है, जो actual array ही होती है।

```
let arr = [1, 2, 3];
console.log(arr.valueOf());          // Output: [1, 2, 3]
```

Question: Search for an Element in an Array

```
let fruits = ["apple", "banana", "cherry", "date"];     // Our array of fruits
let searchFruit = "banana";     // The fruit we want to search for

// Using the includes() method to check if the fruit exists in the array
let fruitExists = fruits.includes(searchFruit);
```

```
if(fruitExists) {   // Displaying the result
  console.log(`The fruit ${searchFruit} exists in the array.`);
} else {
  console.log(`The fruit ${searchFruit} does not exist in the array.`);
}
```

JavaScript Strings

JavaScript में strings एक sequence of characters होते हैं जो text को represent करते हैं। Strings को single quotes (' ') या double quotes (" ") के बीच लिखा जाता है। JavaScript में strings immutable होते हैं, यानि once create होने के बाद उनकी values change नहीं की जा सकती, लेकिन नए strings create किये जा सकते हैं।

Quotes Inside Quotes: अगर आपको string के अंदर quotes का use करना हो, तो आप opposite types का use कर सकते हैं, या escape character का।

Example 1: Different Quotes

```
let text1 = "He said, 'JavaScript is fun!'";
let text2 = 'He said, "JavaScript is fun!"';
```

Example 2: Escape Character: अगर same quotes का use करना है, तो escape character (\) का use करना होगा।

```
let text = "He said, \"JavaScript is fun!\"";
```

Template Strings (Template Literals): Template strings backticks (`) का use करके लिखे जाते हैं और string के अंदर variables को inject करने का एक easy way provide करते हैं using ${}.

Interpolation: Interpolation एक technique है जो string के अंदर variables या expressions को directly insert करने का method provide करती है, बिना concatenation (+) का use किये। JavaScript में string interpolation को template literals या template strings के ज़रिये achieve किया जाता है।

Variable Substitutions: Variable substitution का मतलब है की आप string के अंदर variables को directly insert कर सकते हो। इससे पहले, आपको variables को concatenate करना पड़ता था, पर template literals के साथ, आप easily variables को embed कर सकते हो।

Example 1: Without Template Literals (Concatenation)

```
let name = "Om";
let message = "Hello, " + name + "! Welcome to JavaScript.";
console.log(message);    // Output: Hello, Om! Welcome to JavaScript.
```

Example 2: With Template Literals (Variable Substitution)

```
let name = "Om";
let message = `Hello, ${name}! Welcome to JavaScript.`;
console.log(message);    // Output: Hello, Om! Welcome to JavaScript.
```

Expression Substitution: Expression substitution का मतलब है की आप string के अंदर directly mathematical या logical expressions लिख सकते हो, और उनका result string के अंदर insert हो जाता है। आप complex expressions को भी ${} के अंदर लिख कर evaluate कर सकते हैं।

Example 1: Simple Expression

```
let a = 5;
let b = 10;
console.log(`The sum of ${a} and ${b} is ${a + b}.`);    // Output: The sum of 5 and 10 is 15.
```

Example 2: Function Call Inside Template Literal

```
function greet(name) {
  return `Hello, ${name}!`;
}

let userName = "Om";
console.log(`${greet(userName)} Welcome to our website.`);
```

```
Hello, Om! Welcome to our website.
```

Multi-line Strings Example:

```
let message = `This is line one.
This is line two.`;
console.log(message);
```

```
This is line one.
This is line two.
```

String Length: String की length को length property के ज़रिये access किया जा सकता है। यह property string के characters को count करती है, including spaces.

Example:

```
let text = "Hello, World!";
console.log(text.length);      // Output: 13
```

Escape Characters: Escape characters special characters होते हैं जो string के अंदर use किये जा सकते हैं, जैसे new lines, tabs, या quotes को escape करना।

Common Escape Characters:

- **\'** - Single quote
- **\"** - Double quote
- **** - Backslash
- **\n** - New line
- **\t** – Horizontal Tab
- **\v** – Vertical Tab
- **\f** – Form Feed
- **\r** - Carriage return

Example:

```
let text = "This is a new line\nAnd this is another line.";
console.log(text);
```

```
This is a new line
And this is another line.
```

```
let text = 'It\'s a great day!';
console.log(text);    // Output: It's a great day!
```

Question 1: JavaScript Program to Reverse a String using a For Loop

```
function reverseStringForLoop(str) {
  let reversed = '';
  for (let i = str.length - 1; i >= 0; i--) {
    reversed += str[i];
  }
  return reversed;
}

// Example usage
const inputString = "hello";
console.log("Original String:", inputString);
console.log("Reversed String (For Loop):", reverseStringForLoop(inputString));
```

Question 2: JavaScript Program to Reverse the Words in a Sentence without Built-in Functions

```
function reverseWords(sentence) {
  let words = [];      // Array to store the words
  let word = "";       // String to temporarily store each word

  // Step 1: Traverse the sentence and extract words manually
```

```
  for (let i = 0; i < sentence.length; i++) {
    if (sentence[i] === ' ') {
      // When encountering a space, push the word to the array and reset the word
      if (word.length > 0) {
        words.push(word);
        word = "";      // Reset the word variable for the next word
      }
    } else {
      // Add the character to the current word
      word += sentence[i];
    }
  }
  // Push the last word to the array (if any)
  if (word.length > 0) {
    words.push(word);
  }
  // Step 2: Reverse the order of the words
  let reversedSentence = "";      // String to store the reversed sentence
  for (let i = words.length - 1; i >= 0; i--) {
    reversedSentence += words[i];      // Add each word to the result
    if (i !== 0) {
      reversedSentence += " ";       // Add a space between words
    }
  }
  return reversedSentence;
}
// Example usage
const sentence = "Hello World from JavaScript";
console.log("Original Sentence:", sentence);
console.log("Reversed Words:", reverseWords(sentence));
```

JavaScript String Methods

1. String length: यह method string की total length (कितने characters हैं, including spaces) return करता है।

```
let str = "JavaScript";
console.log(str.length);     // Output: 10
```

2. String charAt(): इस method से आप किसी specific index पर जो character है, use return कर सकते हो।

```
let str = "JavaScript";
console.log(str.charAt(3));     // Output: "a"
```

3. String charCodeAt(): यह method string के किसी index पर जो character है, उसका Unicode value return करता है।

```
let str = "JavaScript";
console.log(str.charCodeAt(3));  // Output: 97  (Unicode of 'a')
```

4. String at(): इस method से आप किसी specific index पर जो character है, उसको access कर सकते हो। Negative index भी use कर सकते हैं (negative values से end से count होता है)।

```
let str = "JavaScript";
console.log(str.at(-1)); // Output: "t" (last character)
```

5. String [ ] (Bracket Notation): आप directly square brackets के अंदर index लिख कर किसी specific character को access कर सकते हो।

```
let str = "JavaScript";
console.log(str[3]);     // Output: "a"
```

6. String slice(): इस method से आप string का एक portion निकाल सकते हो। यह starting index से लेकर ending index तक का portion return करता है (ending index को exclude करता है)।

```
let str = "JavaScript";
console.log(str.slice(0, 4));     // Output: "Java"
console.log(str.slice(-6));     // Output: "Script"
```

Om Shankar Mishra

7. String substring(): substring() भी slice() की तरह ही काम करता है, लेकिन यह negative indexes को handle नहीं करता। इसमें आप smaller index पहले देंगे, चाहे आप order गलत भी देंगे तो भी यह swap कर लेगा।

```
let str = "JavaScript";
console.log(str.substring(4, 0));     // Output: "Java"
```

8. String substr() (Deprecated but still used): substr() method से आप एक portion निकाल सकते हो। पहले argument में starting index और दूसरे में कितने characters निकालने हैं, यह specify करते हैं।

```
let str = "JavaScript";
console.log(str.substr(0, 4));     // Output: "Java"
console.log(str.substr(-6, 6));     // Output: "Script"
```

9. String toUpperCase(): यह method string के सभी lowercase letters को uppercase में convert करता है।

```
let str = "JavaScript";
console.log(str.toUpperCase());//Output: JAVASCRIPT
```

10. String toLowerCase(): इस method से string के सभी uppercase letters को lowercase में convert किया जा सकता है।

```
let str = "JavaScript";
console.log(str.toLowerCase()); // Output: javascript
```

11. String concat(): यह method दो या ज़्यादा strings को concatenate (जोड़) करने के लिए use होता है।

```
let str1 = "Java";
let str2 = "Script";
console.log(str1.concat(str2));     // Output: "JavaScript"
```

12. String trim(): इस method से string के दोनों तरफ से (start और end) extra spaces remove किये जा सकते हैं।

```
let str = "   JavaScript   ";
console.log(str.trim());     // Output: JavaScript
```

13. String trimStart(): यह method सिर्फ string के starting से extra spaces को remove करता है।

```
let str = "   JavaScript   ";
console.log(str.trimStart()); // Output: JavaScript
```

14. String trimEnd(): यह method सिर्फ string के end से extra spaces को remove करता है।

```
let str = "   JavaScript   ";
console.log(str.trimEnd());     // Output: "   JavaScript"
```

15. String padStart(): यह method string के start में specified characters या spaces को add करता है, जब तक string की total length desired length तक न हो जाये।

```
let str = "5";
console.log(str.padStart(3, "0"));     // Output: "005"
```

16. String padEnd(): यह method string के end में specified characters या spaces को add करता है, जब तक string की total length desired length तक न हो जाये।

```
let str = "5";
console.log(str.padEnd(3, "0"));     // Output: "500"
```

17. String repeat(): इस method से आप string को एक specified number of times repeat कर सकते हैं।

```
let str = "Hello";
console.log(str.repeat(3));          // Output: "HelloHelloHello"
```

18. String replace(): यह method string में किसी specified value को replace करता है, लेकिन यह सिर्फ पहली बार मिलने वाली value को ही replace करता है।

```
let str = "JavaScript is awesome";
console.log(str.replace("awesome", "great"));     // Output: "JavaScript is great"
```

19. String replaceAll(): यह method string में specified value को हर जगह replace करता है, जितनी बार वो value मिले।

```
let str = "JavaScript is awesome, and JavaScript is fun";
console.log(str.replaceAll("JavaScript", "JS"));     // Output: "JS is awesome, and JS is fun"
```

20. String split(): इस method से string को specified delimiter के basis पर parts में तोड़ा जा सकता है, और result एक array के form में मिलता है।

```
let str = "JavaScript is awesome";
console.log(str.split(" "));     // Output: ["JavaScript", "is", "awesome"]
```

Question 1: JavaScript Program to Reverse a String

```
function reverseString(str) {
  // Step 1: Split, reverse, and join the string
  let reversedStr = str.split('').reverse().join('');
  return reversedStr;
}

// Example input
let inputString = "hello";
let result = reverseString(inputString);
console.log(`The reversed string is: ${result}`);
```

Question 2: JavaScript Program to Check if a String is a Palindrome

```
function isPalindrome(str) {
  // Step 1: Normalize the string (convert to lowercase and remove non-alphanumeric characters)
  let normalizedStr = str.toLowerCase().replace(/[^a-z0-9]/g, '');
  let reversedStr = normalizedStr.split('').reverse().join('');   // Step 2: Reverse the string
  return normalizedStr === reversedStr; // Step 3: Check if the original string and reversed string are the same
}
// Example input
let inputString = "madam";
if (isPalindrome(inputString)) {
  console.log(`${inputString} is a palindrome`);
} else {
  console.log(`${inputString} is not a palindrome`);
}
```

Question 3: JavaScript Program to Split a String into an Array of Words

```
function splitStringIntoWords(str) {
  // Step 1: Split the string by spaces
  const wordsArray = str.split(' ');
  // Step 2: Display the array of words
  console.log("Array of words:", wordsArray);
}
// Example usage
const sentence = "Hello world, welcome to JavaScript!";
splitStringIntoWords(sentence);
```

Question 4: JavaScript Program to Join an Array of Words into a String

```
function joinArrayIntoString(arr) {
  // Step 1: Join the array into a string with space as a separator
  let result = arr.join(' ');

  // Step 2: Return the resulting string
  return result;
}

// Example input
let wordsArray = ["Hello", "world", "this", "is", "JavaScript"];
let resultString = joinArrayIntoString(wordsArray);
console.log(`The joined string is: "${resultString}"`);
```

Question 5: JavaScript Program to Check if Two Strings are Anagrams

```
function areAnagrams(str1, str2) {
  // Step 1: Remove non-alphabetic characters, convert to lowercase, and sort
  const normalizeString = (str) => str.toLowerCase().replace(/[^a-z]/g, '').split('').sort().join('');

  // Step 2: Compare the normalized versions of the strings
  return normalizeString(str1) === normalizeString(str2);
}

// Example usage
const string1 = "listen";
const string2 = "silent";

if (areAnagrams(string1, string2)) {
  console.log("The strings are anagrams.");
} else {
  console.log("The strings are not anagrams.");
}
```

Question 6: JavaScript Program to Count the Number of Words in a String

```
function countWords(str) {
  // Step 1: Trim any leading/trailing spaces and split the string by spaces
  let wordsArray = str.trim().split(/\s+/);

  // Step 2: Return the number of words
  return wordsArray.length;
}

// Example input
let inputString = "Hello world this is JavaScript";
let wordCount = countWords(inputString);
console.log(`The number of words in the string is: ${wordCount}`);
```

Question 7: JavaScript Program to Convert a String to Title Case

```
function toTitleCase(str) {
  // Step 1: Convert the string to an array of words
  let wordsArray = str.toLowerCase().split(' ');

  // Step 2: Capitalize the first letter of each word
  let titleCasedArray = wordsArray.map(word => {
    return word.charAt(0).toUpperCase() + word.slice(1);
```

```
  });
  // Step 3: Join the words back into a single string
  return titleCasedArray.join(' ');
}
// Example input
let inputString = "hello world";
let result = toTitleCase(inputString);
console.log(`Title Case: "${result}"`);
```

JavaScript String Search

1. String indexOf(): यह method पहली बार किसी specified substring या character का index return करता है जो string में मिलता है। अगर substring नहीं मिलती, तो यह -1 return करता है।

```
let str = "JavaScript is awesome";
console.log(str.indexOf("is"));      // Output: 11
console.log(str.indexOf("python"));      // Output: -1
```

- Example में "is" का पहला index 11 है, और "python" string में नहीं है, इसलिए -1 return होता है।

आप starting point भी specify कर सकते हैं:

```
console.log(str.indexOf("a", 5));        // Output: 9
```

- यह search index 5 से start करता है।

2. String lastIndexOf(): यह method string में किसी specified substring का last occurrence (आखरी बार मिलने वाला index) return करता है। अगर substring नहीं मिलती, तो यह भी -1 return करता है।

```
let str = "JavaScript is awesome, and JavaScript is fun";
console.log(str.lastIndexOf("JavaScript"));     // Output: 26
```

- यह method आखरी बार "JavaScript" कहा मिला, उसका index return करेगा।

आप search को specific index तक limit कर सकते हैं :

```
console.log(str.lastIndexOf("JavaScript", 25));       // Output: 0
```

- यह search पहला "JavaScript" return करेगा, क्यूंकि search index 25 तक limit है।

3. String search(): यह method भी substring को search करता है, लेकिन यह regular expressions को support करता है। अगर match मिलता है, तो यह पहला index return करता है, वरना -1.

```
let str = "JavaScript is awesome";
console.log(str.search("awesome"));   // Output: 15
```

- Search method simple string या regular expression use करता है।

Regular expression के साथ use कर सकते हैं:

```
let str = "JavaScript is awesome!";
console.log(str.search(/is/i));     // Output: 11
```

- यहाँ /is/i एक case-insensitive search है।

4. String match(): match() method एक string के अंदर एक regular expression के match को search करता है और उसके matches को return करता है एक array में। अगर कोई match नहीं मिलता, तो null return होता है।

```
let str = "JavaScript is awesome";
let result = str.match(/is/g);     // Regular expression with global flag
console.log(result);     // Output: ["is"]
```

- यह "is" को match करता है और उसको array में return करता है।

अगर कोई global flag (g) use नहीं किया गया, तो यह सिर्फ पहले match को return करता है :

```
let result = str.match(/a/);
console.log(result);      // Output: ["a", index: 1, input: "JavaScript is awesome", groups: undefined]
```

5. String matchAll(): matchAll() भी regular expression के matches को search करता है, लेकिन यह सभी matches return करता है एक iterator के form में, including their detailed information. इसको use करते time global flag (g) ज़रूर होना चाहिए।

```
let str = "JavaScript is amazing and awesome";
let result = str.matchAll(/a\w+/g);    // Match words starting with 'a'
for (let match of result) {
   console.log(match);
}
```

```
[
  'avaScript',
  index: 1,
  input: 'JavaScript is amazing and awesome',
  groups: undefined
]
[
  'amazing',
  index: 14,
  input: 'JavaScript is amazing and awesome',
  groups: undefined
]
[
  'and',
  index: 22,
  input: 'JavaScript is amazing and awesome',
  groups: undefined
]
[
  'awesome',
  index: 26,
  input: 'JavaScript is amazing and awesome',
  groups: undefined
]
```

- यह iterator object देता है जो सभी matches के साथ detailed info provide करता है।

6. String includes(): includes() method check करता है की एक string में कोई specified substring present है या नहीं। यह true या false return करता है।

```
let str = "JavaScript is awesome";
console.log(str.includes("awesome")); // Output: true    console.log(str.includes("python"));    // Output: false
```

- यह method case-sensitive होता है।

आप search को specific position से start भी कर सकते हैं :

```
console.log(str.includes("is", 12));      // Output: false
```

- Search index 12 से start होता है, इसलिए "is" नहीं मिलता।

7. String startsWith(): startsWith() method check करता है की एक string किसी specified substring से start होती है या नहीं। यह भी true या false return करता है।

```
let str = "JavaScript is awesome";
console.log(str.startsWith("Java"));      // Output: true
console.log(str.startsWith("awesome"));   // Output: false
```

- यह भी case-sensitive होता है।

आपको specify करना है की string किस position से check करे, तो आप एक optional second argument दे सकते हैं :

```
console.log(str.startsWith("is", 10));      // Output: true
```

8. String endsWith(): endsWith() method check करता है की एक string किसी specified substring से end होती है या नहीं।

```
let str = "JavaScript is awesome";
console.log(str.endsWith("awesome"));     // Output: true
console.log(str.endsWith("Java"));      // Output: false
```

इसमें भी optional position argument है जो specify करता है की string कौनसा portion check करे :

```
console.log(str.endsWith("is", 13));      // Output: true
```

- यहाँ यह check करेगा अगर 13th character तक string "is" पे end होती है।

Program 1: JavaScript Program to Check if a String Contains Another String

```
function containsSubstring(mainString, substring) {
  // Step 1: Use the includes() method to check for the substring
  return mainString.includes(substring);
}

// Example input
let mainString = "Hello world";
let substring = "world";
let result = containsSubstring(mainString, substring);
console.log(`Does the string "${mainString}" contain "${substring}"? ${result}`);
```

for...in Loop

- for...in loop object properties को iterate करता है। इसका use तब होता है जब हमे object के keys या array के indices को loop करना हो।
- यह loop enumerable properties (object के properties या array के indices) को traverse करता है।

Syntax:

```
for (let key in object) {
  // code
}
```

Example:

```
let student = { name: "Om", age: 21, city: "Mumbai" };
for (let key in student) {
  console.log(key + ": " + student[key]);
}
```

```
name: Om
age: 21
city: Mumbai
```

- इस example में for...in loop student object के properties को access करता है और key-value pairs को print करता है।

Arrays के case में भी indices को traverse करता है:

```
let arr = [10, 20, 30];
for (let index in arr) {
  console.log(index + ": " + arr[index]);
}
```

```
0: 10
1: 20
2: 30
```

- इस example में array के indices (0, 1, 2) को traverse करता है।

for...of Loop

- for...of loop iterable objects को iterate करता है। यह loop array के elements को directly traverse करता है, बिना indices को involve किये।
- इसका use mainly arrays, strings, maps, sets या किसी भी iterable object के elements को loop करने के लिए होता है।

Syntax:

```
for (let element of iterable) {
  // code
}
```

Example:

```
let arr = [10, 20, 30];
for (let value of arr) {
  console.log(value);
}
```

```
10
20
30
```

- यहाँ for...of loop directly array के values को iterate करता है।

Strings के case में हर एक character को loop करता है :

```
let str = "JavaScript";
for (let char of str) {
  console.log(char);
}
```

J
a
v
a
S
c
r
i
p
t

JavaScript Number Methods

JavaScript में कुछ built-in methods होते हैं जो numbers के साथ काम करने में help करते हैं। यह methods हमे numerical values को manipulate या format करने के लिए काम आते हैं।

1. toString(): इस method से एक number को string में convert किया जा सकता है।

```
let num = 123;
console.log(num.toString());  // Output: "123"
```

2. toFixed(): इस method का use number को fixed number of decimals तक round करने के लिए होता है।

```
let num = 9.876;
console.log(num.toFixed(2));  // Output: "9.88"
```

3. toExponential(): Number को exponential notation में convert करता है।

```
let num = 123456;
console.log(num.toExponential(2));// Output: "1.23e+5"
```

4. toPrecision(): Number को specified length तक format करता है।

```
let num = 12.34567;
console.log(num.toPrecision(4));// Output: "12.35"
```

5. valueOf(): इस method का use number के primitive value को return करने के लिए होता है।

```
let num = 100;
console.log(num.valueOf());  // Output: 100
```

6. parseInt(): String को integer number में convert करता है।

```
console.log(parseInt("50"));  // Output: 50
```

7. parseFloat(): String को floating-point number में convert करता है।

```
console.log(parseFloat("99.99"));  // Output: 99.99
```

8. isFinite(): Check करता है की value finite है या नहीं।

```
console.log(Number.isFinite(100));  // Output: true
console.log(Number.isFinite(Infinity)); // Output: false
```

9. isNaN(): Check करता है की value "Not-a-Number" (NaN) है या नहीं।

```
console.log(Number.isNaN(100));  // Output: false
console.log(isNaN("hello"));  // Output: true
```

JavaScript Number Properties

JS में कुछ predefined Number properties भी होती हैं जो constant values को represent करती हैं।

1. Number.MAX_VALUE: Maximum number जो JavaScript represent कर सकता है।

```
console.log(Number.MAX_VALUE);  // Output: 1.7976931348623157e+308
```

2. Number.MIN_VALUE: Smallest positive number जो JavaScript represent कर सकता है।

```
console.log(Number.MIN_VALUE);  // Output: 5e-324
```

3. Number.POSITIVE_INFINITY: Positive infinity को represent करता है।

```
console.log(Number.POSITIVE_INFINITY);  // Output: Infinity
```

4. Number.NEGATIVE_INFINITY: Negative infinity को represent करता है।

```
console.log(Number.NEGATIVE_INFINITY);  // Output: -Infinity
```

5. Number.NaN: Not-a-Number को represent करता है।

```
console.log(Number.NaN);  // Output: NaN
```

Random Number in JavaScript

JavaScript में randomness को Math.random() से generate किया जा सकता है। ये method एक random number generate करता है जो 0 और 1 के बीच होता है (1 exclusive, मतलब कभी भी 1 नहीं होता)।

1. Basic Random Number Between 0 and 1

```
let randomNumber = Math.random();
console.log(randomNumber);
```

Output: जैसे कोई random number 0.453 या 0.789 जो 0 और 1 के बीच होता है।

2. Random Number Between 0 and any Maximum Value: अगर आपको 0 से कोई maximum value तक random number चाहिए, तो Math.random() को max से multiply कर दो।

```
let max = 10;
let randomNumber = Math.random() * max;
console.log(randomNumber);
```

Output: एक random number जो 0 और 10 के बीच होता है।

3. Random Integer Between 0 and Maximum: अगर आपको whole number चाहिए, तो Math.floor() का use करो जो number को round down करता है:

```
let max = 10;
let randomInt = Math.floor(Math.random() * max);
console.log(randomInt);
```

Output: एक random integer जो 0 और 9 के बीच होगा (क्यूंकि Math.floor() number को round down करता है)।

4. Random Integer Between min and max: अगर आपको किसी specific range में random integer चाहिए, तो ऐसे करेंगे :

```
let min = 5;
let max = 15;
let randomInt = Math.floor(Math.random() * (max - min + 1)) + min;
console.log(randomInt);
```

Output: एक random integer जो 5 और 15 के बीच हो सकता है।

5. Random Decimal Between min and max: अगर आपको random decimal number चाहिए specific range में :

```
let min = 1.5;
let max = 5.5;
let randomDecimal = Math.random() * (max - min) + min;
console.log(randomDecimal);    //Output: एक random decimal number जो 1.5 और 5.5 के बीच होगा।
```

Random Element from an Array: Array में से random element pick करने के लिए Math.random() और Math.floor() का combination use करते हैं :

```
let colors = ["red", "green", "blue", "yellow", "purple"];
let randomIndex = Math.floor(Math.random() * colors.length);
let randomColor = colors[randomIndex];
console.log(randomColor);    //Output: Array में से कोई random color, जैसे "green" या "yellow".
```

Random Boolean: Random true या false generate करना :

```
let randomBool = Math.random() >= 0.5;
console.log(randomBool);
```

Output: true या false randomly return करेगा।

JavaScript Date Objects

JavaScript में Date Object का use date और time के साथ काम करने के लिए होता है। इस object से आप current date, specific date, और time values को easily access और manipulate कर सकते हैं।

Creating a Date Object: Date object को create करने के multiple ways होते हैं :

Current Date and Time:

```
let currentDate = new Date();
console.log(currentDate);     // Output: Current date and time
```

```
Sat Nov 23 2024 10:12:44 GMT+0530
(India Standard Time)
```

Specific Date: (Year, Month, Date)

```
let customDate = new Date(1997, 3, 10);
console.log(customDate);   // Note: Month is 0-indexed (0 = January)
```

```
Thu Apr 10 1997 05:30:00 GMT+0530
(India Standard Time)
```

Year, Month, Day, Hour, Minute, Second, Millisecond:

```
let fullDate = new Date(1997, 3, 10, 4, 54, 45, 500);
console.log(fullDate);
```

```
Thu Apr 10 1997 04:54:45 GMT+0530
(India Standard Time)
```

JavaScript Date Formats: JavaScript में date formats काफी flexible हैं, और dates को multiple ways में format किया जा सकता है। जब आप एक date को JavaScript में create या display करते हैं, तो वो different formats में हो सकती है। चलिए, कुछ common date formats को समझते हैं।

1. ISO Date (International Standard): ISO format YYYY-MM-DD (year-month-day) है, और ये widely accepted और recommended format है।

```
let isoDate = new Date("2023-08-15");
console.log(isoDate);
```

```
Thu Apr 10 1997 05:30:00 GMT+0530
(India Standard Time)
```

```
let isoDateTime = new Date("1997-04-10T10:30:00Z");  // (Z indicates UTC time)
console.log(isoDateTime);
```

```
Thu Apr 10 1997 16:00:00
GMT+0530 (India Standard Time)
```

```
let isoDateTime = new Date("1997-04-10T10:30:00");
console.log(isoDateTime);
```

```
Thu Apr 10 1997 10:30:00
GMT+0530 (India Standard Time)
```

2. Short Date Format: Short date format country-specific होती है और mostly MM/DD/YYYY या DD/MM/YYYY के रूप में होती है।

Example (MM/DD/YYYY format in the U.S.):

```
let shortDate = new Date("04/10/1997");
console.log(shortDate);
```

```
Thu Apr 10 1997 00:00:00
GMT+0530 (India Standard Time)
```

3. Long Date Format: Long date format usually month name के साथ होता है, और ये भी country-specific होता है। जैसे, "Month Day, Year" या "Day Month Year".

Example (Month Day, Year):

```
let longDate = new Date("April 10, 1997");
console.log(longDate);
```

```
Thu Apr 10 1997 00:00:00
GMT+0530 (India Standard Time)
```

Example (Day Month Year):

```
let longDate = new Date("10 April 1997");
console.log(longDate);
```

Date Methods: JavaScript Date Object provides multiple methods to get and set the date and time values.

a. Getting Date & Time Information:

getFullYear(): Current year को return करता है।

```
let currentYear = new Date().getFullYear();
console.log(currentYear);     // Output:  2024
```

getMonth(): Month ko return karta hai (0 = January, 11 = December).

```
let currentMonth = new Date().getMonth();    // Note: Month is 0-indexed (0 = January)
console.log(currentMonth);    // Output: 10 (November)
```

getDate(): Current day of the month को return करता है (1-31).

```
let currentDay = new Date().getDate();
console.log(currentDay);    // Output: 23
```

getDay(): Day of the week को return करता है (0 = Sunday, 6 = Saturday).

```
let currentWeekday = new Date().getDay();
console.log(currentWeekday);  // Output: 6 (Saturday)
```

getHours(): Current hour को return करता है (0-23).

```
let currentHour = new Date().getHours();
console.log(currentHour);    // Output: 11
```

getMinutes(): Current minutes को return करता है (0-59).

```
let currentMinute = new Date().getMinutes();
console.log(currentMinute);    // Output: 19
```

getSeconds(): Current seconds को return करता है (0-59).

```
let currentSecond = new Date().getSeconds();
console.log(currentSecond);    // Output: 45
```

getMilliseconds(): Milliseconds को return करता है (0-999).

```
let currentMillisecond = new Date().getMilliseconds();
console.log(currentMillisecond);    // Output: 500
```

getTime(): Timestamp return करता है, जो milliseconds since January 1, 1970 (UTC) (unix or epoch (i.e. computing time)) को represent करता है।

```
let currentDate = new Date();    // Date.now()
console.log(currentDate.getTime());    // Output: 1692238245000 (milliseconds since 1970)
```

getTimezoneOffset(): Local time और UTC (Coordinated Universal Time) के बीच का difference minutes में return करता है।

```
let currentDate = new Date();
console.log(currentDate.getTimezoneOffset());
// Output: -240 (for example, for UTC-4)
```

getUTCFullYear(): UTC time के हिसाब से year को retrieve करता है।

```
let currentDate = new Date();
console.log(currentDate.getUTCFullYear());
 // Output: 2023
```

getUTCMonth(): UTC time के हिसाब से month को retrieve करता है (0 to 11).

```
let currentDate = new Date();
console.log(currentDate.getUTCMonth());
 // Output: 7 (for August)
```

getUTCDate(): UTC time के हिसाब से day of the month को retrieve करता है (1 to 31).

```
let currentDate = new Date();
console.log(currentDate.getUTCDate());
// Output: 15 (if it's the 15th of the month)
```

getUTCDay(): UTC time के हिसाब से day of the week को retrieve करता है (0 to 6).

```
let currentDate = new Date();
console.log(currentDate.getUTCDay());
// Output: 2 (for Tuesday)
```

getUTCHours(): UTC time के हिसाब से current hour को retrieve करता है (0 to 23).

```
let currentTime = new Date();
console.log(currentTime.getUTCHours());
// Output: 14 (for 2 PM in UTC)
```

b. Setting Date & Time Information:

setFullYear(year): Date के year को set करता है।

```
let date = new Date();
date.setFullYear(2025);
console.log(date);  // Output: Year will be set to 2025
```

setHours(hours): Hours को set करता है (0-23).

```
let date = new Date();
date.setHours(15);    // Set to 3 PM
console.log(date);
```

setDate(day): Date के day of the month को set करता है।

```
let date = new Date();
date.setDate(10);  // Set to the 10th of the month
console.log(date);
```

setMonth(month): Date के month को set करता है (0-11).

```
let date = new Date();
date.setMonth(5);     // Set to June
console.log(date);
```

setMinutes(): Minutes को set करता है (0 to 59).

```
let date = new Date();
date.setMinutes(30); //
dateObject.setMinutes(minutes, [seconds], [milliseconds])
console.log(date);
```

setSeconds(): Seconds को set करता है (0 to 59).

```
let date = new Date();
date.setSeconds(45);   //
dateObject.setSeconds(seconds, [milliseconds])
console.log(date);
```

setMilliseconds(): Milliseconds को set करता है (0 to 999).

```
let date = new Date();
date.setMilliseconds(123);  //
dateObject.setMilliseconds(milliseconds)
console.log(date);
```

setTime(): Timestamp को set करता है, जो milliseconds since January 1, 1970 (UTC) represent करता है।

```
let date = new Date();
date.setTime(1692238245000);  //
dateObject.setTime(milliseconds)
console.log(date);
```

setUTCFullYear(): UTC time के according year को set करता है (4-digit format).

```
let date = new Date();
date.setUTCFullYear(2025);   //
dateObject.setUTCFullYear(year, [month], [day])
console.log(date);  // Sets the UTC year to 2025
```

setUTCDate(): UTC time के according day of the month को set करता है (1 to 31).

```
let date = new Date();
date.setUTCDate(10);  //
dateObject.setUTCDate(day)
console.log(date);
```

setUTCMonth(): UTC time के according month को set करता है (0 to 11).

```
let date = new Date();
date.setUTCMonth(5);  //
dateObject.setUTCMonth(month, [day])
console.log(date);
```

setUTCHours(): UTC time के according hours को set करता है (0 to 23).

```
let date = new Date();
date.setUTCHours(15);  //
dateObject.setUTCHours(hours, [minutes], [seconds], [milliseconds])
console.log(date);
```

Date Formatting Methods

toString(): JavaScript में एक Date object को toString() method के through default text format में convert किया जाता है। जब आपको Date और Time दोनों की पूरी details चाहिए।

```
let currentDate = new Date();
console.log(currentDate.toString());  // Sat Nov 23 2024 05:38:43 GMT+0000 (Coordinated Universal Time)
```

toDateString(): Date को human-readable string format में return करता है (without time).

```
let date = new Date();
console.log(date.toDateString());     // Output: e.g., "Thu Aug 15 2024"
```

toLocaleDateString(): Local format के according date को return करता है।

```
let date = new Date();
console.log(date.toLocaleDateString());     // Output: "8/15/2024" (depends on locale)
```

Custom Date Formatting: JavaScript में dates को custom format में display करने के लिए toLocaleDateString() का use किया जाता है। इसमें आप custom options pass कर सकते हैं।

Example (Locale-Specific Format):

```
let currentDate = new Date();
console.log(currentDate.toLocaleDateString("en-US"));     // Output: 8/15/2023 (U.S. format)
console.log(currentDate.toLocaleDateString("en-GB"));     // Output: 15/08/2023 (British format)
```

Example (Custom Options):

```
let currentDate = new Date();
let options = { year: 'numeric', month: 'long', day: 'numeric' };
console.log(currentDate.toLocaleDateString('en-US', options));     // Output: August 15, 2023
```

toTimeString(): Time को human-readable string format में return करता है।

```
let time = new Date();
console.log(time.toTimeString());     // Output: e.g., "10:30:45 GMT+0000"
```

toLocaleTimeString(): Local format के according time को return करता है।

```
let time = new Date();
console.log(time.toLocaleTimeString());     // Output: "10:30:45 AM"
```

Unix Timestamp Format: Unix timestamp एक number है जो seconds या milliseconds को represent करता है since January 1, 1970 (UTC).

Get current timestamp in milliseconds:

```
let timestamp = Date.now();
console.log(timestamp);     // Output: 1692238245000 (example value)
```

Convert timestamp to date:

```
let dateFromTimestamp = new Date(1692238245000);
console.log(dateFromTimestamp);     // Output: Wed Aug 16 2023
```

Date Parsing: अगर date string non-standard format में है, तो Date.parse() method use किया जा सकता है। ये method date string को parse करता है और एक timestamp return करता है।

```
let timestamp = Date.parse("Aug 15, 2023");
console.log(timestamp);     // Output: 1692144000000 (Milliseconds since 1970)
```

Comparing Dates: JavaScript में dates को compare किया जा सकता है :

Equality Comparison:

```
let date1 = new Date('2023-08-15');
let date2 = new Date('2023-08-15');

console.log(date1.getTime() === date2.getTime());    // true
```

Before and After Comparison:

```
let date1 = new Date('2023-08-15');
let date2 = new Date('2024-08-15');

console.log(date1 < date2);  // Output: true
```

Program 1: Countdown Timer

```
function countdown(endDate) {
  let now = new Date();
  let timeRemaining = endDate - now;

  let days = Math.floor(timeRemaining / (1000 * 60 * 60 * 24));
  let hours = Math.floor((timeRemaining % (1000 * 60 * 60 * 24)) / (1000 * 60 * 60));
  let minutes = Math.floor((timeRemaining % (1000 * 60 * 60)) / (1000 * 60));
  let seconds = Math.floor((timeRemaining % (1000 * 60)) / 1000);

  return `${days}d ${hours}h ${minutes}m ${seconds}s`;
```

```
}
let eventDate = new Date("2026-12-31T23:59:59");
console.log("Countdown to New Year:", countdown(eventDate));     // 768d 11h 37m 30s
```

Program 2: **Age Calculator**

```
function calculateAge(birthDate) {
  let now = new Date();
  let age = now.getFullYear() - birthDate.getFullYear();
  let monthDifference = now.getMonth() - birthDate.getMonth();
  if (monthDifference < 0 || (monthDifference === 0 && now.getDate() < birthDate.getDate())) {
    age--;
  }
  return age;
}
let birthDate = new Date("1997-04-10");
console.log("Age:", calculateAge(birthDate));        // Age: 27
```

Sets

JavaScript में Set एक special type का object है जो unique values को store करता है। इसका मतलब ये है की Set में duplicate values allowed नहीं होते।

How to create Set?

आप Set object को new Set() keyword के साथ बना सकते हैं :

let mySet = new Set();

Add values in Set: Set में values को add() method से add करते हैं :

```
let mySet = new Set();
mySet.add(10);
mySet.add(20);
mySet.add(10);     // Duplicate value
console.log(mySet);     // Output: Set { 10, 20 }
```

- यहाँ, 10 को दो बार add करने पर भी वो एक ही बार store होता है क्यूंकि Set duplicate values को ignore करता है।

Providing values to set during initialization: आप Set को create करते time ही values pass कर सकते हैं :

```
let mySet = new Set([1, 2, 3, 4, 5]);
console.log(mySet);     // Output: Set { 1, 2, 3, 4, 5 }
```

Set Methods: JavaScript में Set एक special object है जो unique values store करता है। Sets के कुछ methods होते हैं जो हम Set के elements को manipulate करने के लिए use करते हैं।

1. add(): Set में एक new value add करता है। अगर value पहले से मौजूद है, तो वो duplicate value को ignore करेगा।

```
let mySet = new Set();
mySet.add(10);
mySet.add(20);
mySet.add(10);     // Duplicate value
console.log(mySet);     // Output: Set { 10, 20 }
```

2. delete(): Set में से किसी specific value को remove करता है। अगर वो value Set में नहीं है तो कोई effect नहीं होगा।

```
let mySet = new Set([1, 2, 3, 4]);
mySet.delete(3);
console.log(mySet);     // Output: Set { 1, 2, 4 }
```

3. has(): Check करता है की Set में कोई specific value है या नहीं। ये method boolean (true/false) value return करता है।

```
let mySet = new Set([5, 6, 7]);
console.log(mySet.has(6));     // Output: true
console.log(mySet.has(10));     // Output: false
```

4. clear(): Set के सारे elements को remove कर देता है, Set को empty बना देता है।

```
let mySet = new Set([1, 2, 3]);
mySet.clear();
console.log(mySet);     // Output: Set {}
```

5. size: Set में कितने elements हैं वो return करता है।

```
let mySet = new Set([10, 20, 30]);
console.log(mySet.size);     // Output: 3
```

6. values(): Set के सारे values को एक iterator के form में return करता है।

```
let mySet = new Set(["apple", "banana", "mango"]);
let values = mySet.values();
for (let value of values) {
  console.log(value);
}
```

```
apple
banana
mango
```

7. keys(): Keys को return करता है। Note: Set में keys और values दोनों same होते हैं, to keys() भी values() जैसे ही काम करता है ।

```
let mySet = new Set([100, 200, 300]);
let keys = mySet.keys();
for (let key of keys) {
  console.log(key);
}
```

```
100
200
300
```

8. entries(): Set के हर element का एक array pair बनता है जिसमे दोनों [value, value] होता है (since Set में key और value same होते हैं)।

```
let mySet = new Set(["a", "b", "c"]);
let entries = mySet.entries();
for (let entry of entries) {
  console.log(entry);
}
```

```
[ 'a', 'a' ]
[ 'b', 'b' ]
[ 'c', 'c' ]
```

Loop through a Set: आप Set के elements को for..of loop या forEach() से loop करके access कर सकते हैं :

```
let numbers = new Set([1, 2, 3, 4]);
```

```
// Using for..of loop
for (let num of numbers) {
  console.log(num);
}
```

```
1
2
3
4
```

```
// Using forEach
numbers.forEach(function(value) {
  console.log(value);
});
```

```
1
2
3
4
```

Convert Set into Array: कभी कभी Set को Array में convert करना useful होता है। इसके लिए आप Array.from() या spread operator (...) का use कर सकते हैं :

```
let set = new Set([10, 20, 30]);
let array = Array.from(set);
console.log(array);     // Output: [10, 20, 30]
```

```
//or
let array = [...set];
console.log(array);     // Output: [10, 20, 30]
```

Practical Use Case of Set: Sets को तब use किया जाता है जब आपको unique values store करनी हो। जैसे, अगर आपके पास एक array है जिसमे duplicate values हैं और आपको unique values चाहिए :

```
let arr = [1, 2, 2, 3, 4, 4, 5];
let uniqueValues = [...new Set(arr)];
console.log(uniqueValues);     // Output: [1, 2, 3, 4, 5]
```

Example: इस example में हम एक array create करेंगे जिसमे कुछ mobile numbers होते हैं। फिर हम एक Set का use करके unique mobile numbers को store करेंगे, और duplicate numbers का count भी करेंगे जो ये show करेगा की किसी number से कितनी बार call receive हुई है।

```
let mobileNumbers = [
   "9876543XXX", "9876543XXX", "9123456XXX",
   "9123456XXX", "9123456XXX", "9876543XXX",
   "9998887XXX", "9998887XXX", "8765432XXX"
];
let uniqueNumbers = new Set();    // Set for Unique numbers
let callCount = {};    // object for Call count
// Loop through each mobile number
mobileNumbers.forEach(function(number) {
   uniqueNumbers.add(number);       // Adding unique number in Set
   // logic for increament of Count
   if (callCount[number]) {
      callCount[number] += 1;    // if number is existed then increase count
   } else {
      callCount[number] = 1;      // if number occurs first time then set count to 1
   }
});
// Display unique numbers
console.log("Unique Mobile Numbers:");
console.log([...uniqueNumbers]);      // convert set into array and then display
// Display call count for each number
console.log("\nCall Count:");
for (let number in callCount) {
   console.log(`Mobile Number: ${number}, Received Calls: ${callCount[number]}`);
}
```

```
Unique Mobile Numbers:
[ '9876543XXX', '9123456XXX', '9998887XXX', '8765432XXX'
    ]

Call Count:
Mobile Number: 9876543XXX, Received Calls: 3
Mobile Number: 9123456XXX, Received Calls: 3
Mobile Number: 9998887XXX, Received Calls: 2
Mobile Number: 8765432XXX, Received Calls: 1
```

Maps

Map JavaScript का एक built-in object है जो key-value pairs को store करता है। Map के keys और values किसी भी type के हो सकते हैं, चाहे वो primitive हो या object. Maps में insertion order को maintain किया जाता है, और ये एक efficient data structure है जब हमे large amount of data को key-value format में store करना होता है।

Creating a Map: Map को create करने के लिए new Map() constructor का use किया जाता है।

```
let myMap = new Map();
```

Map Methods: JavaScript में Map object key-value pairs को efficiently handle करता है। Map के साथ काफी useful methods होते हैं जो आपको data को store, access, और manipulate करने में help करते हैं।

1. set(key, value): Map में एक key-value pair add करता है। अगर वो key पहले से exist करती है, तो वो उस key की value को overwrite कर देता है।

```
let myMap = new Map();
myMap.set("name", "Om");
myMap.set("age", 27);
console.log(myMap);
// Output: Map { 'name' => 'Om', 'age' => 27 }
```

2. get(key): दी गयी key के corresponding value को return करता है। अगर key नहीं मिलती, तो undefined return करता है।

```
console.log(myMap.get("name"));    // Output: Om
console.log(myMap.get("gender"));    // Output: undefined
```

3. has(key): Check करता है की दी गयी key map में exist करती है या नहीं। अगर key exist करती है तो true, वरना false return करता है।

```
console.log(myMap.has("name"));   // Output: true
console.log(myMap.has("gender"));    // Output: false
```

4. delete(key): दी गई key को map से remove करता है। अगर key successfully remove हो जाती है तो true, वरना false return करता है।

```
myMap.delete("age");
console.log(myMap);    // Output: Map { 'name' => 'Om' }
```

5. size: Map में total कितने key-value pairs हैं, वो return करता है।

```
console.log(myMap.size);    // Output: 1
```

6. clear(): Map के अंदर जितने भी key-value pairs हैं, सबको remove करता है। Map खाली हो जाता है।

```
myMap.clear();
console.log(myMap.size);    // Output: 0
```

7. keys(): Map में जितनी भी keys हैं, उन सबको एक iterator के रूप में return करता है।

```
let map = new Map([
  ["firstName", "Om"],
  ["lastName", "Mishra"]
]);
```

```
for (let key of map.keys()) {
  console.log(key);
}
```

```
firstName
lastName
```

8. values(): Map की सारी values को एक iterator के रूप में return करता है।

```
for (let value of map.values()) {
  console.log(value);
}
```

```
Om
Mishra
```

9. entries(): Map के key-value pairs को [key, value] format में एक iterator के रूप में return करता है।

```
for (let [key, value] of map.entries()) {
  console.log(`${key}: ${value}`);
}
```

```
firstName: Om
lastName: Mishra
```

10. forEach(): हर एक key-value pair पर एक function apply करता है। Callback function को 3 arguments मिलते हैं : value, key, और map.

```
map.forEach((value, key) => {
  console.log(`${key}: ${value}`);
});
```

```
firstName: Om
lastName: Mishra
```

11. Object.entries(): Object.entries() का use कर के आप किसी object के key-value pairs को map में convert कर सकते हैं।

```
let obj = { a: 1, b: 2, c: 3 };
let mapFromObj = new Map(Object.entries(obj));
console.log(mapFromObj);   // Output: Map { 'a' => 1, 'b' => 2, 'c' => 3 }
```

Example: Creating and Using a Map

```
let contacts = new Map();   // Map creation

// Adding entries
contacts.set("Om", "9876543XXX");
contacts.set("Priyanka", "9123456XXX");
contacts.set("Khushboo", "9998887XXX");

// Accessing values using keys
console.log(contacts.get("Om"));      // Output: 9876543XXX
console.log(contacts.get("Priyanka"));  // Output: 9123456XXX

// Checking if key exists
console.log(contacts.has("Khushboo"));  // Output: true
console.log(contacts.has("Priyansi"));    // Output: false

// Map size
console.log(contacts.size);          // Output: 3

// Iterating over keys and values
for (let [name, phone] of contacts.entries()) {
  console.log(`Name: ${name}, Phone: ${phone}`);
}
```

```
9876543XXX
9123456XXX
true
false
3
Name: Om, Phone: 9876543XXX
Name: Priyanka, Phone: 9123456XXX
Name: Khushboo, Phone: 9998887XXX
```

Example: Map with Object as Key

```
let person1 = { name: "Om" };
let person2 = { name: "Priyanka" };

let userMap = new Map();
userMap.set(person1, "Developer");
userMap.set(person2, "Designer");

console.log(userMap.get(person1));   // Output: Developer
console.log(userMap.get(person2));   // Output: Designer
```

Map vs Object

- Objects में keys सिर्फ strings या symbols हो सकते हैं।
- Maps में किसी भी type की key हो सकती है (string, number, object, function, etc.).
- Maps का size (number of key-value pairs) size property से directly accessible होता है।

```
let myMap = new Map();   // Creating Map object

// Different types of keys are set
myMap.set('stringKey', 'This is a string key');
myMap.set(123, 'This is a number key');
myMap.set(true, 'This is a boolean key');
myMap.set(function() {}, 'This is a function key');
myMap.set({}, 'This is an object key');

console.log(myMap);
```

```
console.log("Map ka size: " + myMap.size);     // Output: Size of Map: 5
console.log(myMap.get('stringKey'));     // Output: This is a string key
console.log(myMap.get(123));           // Output: This is a number key
console.log(myMap.get(true));          // Output: This is a boolean key
console.log(myMap.get(function() {}));    // Output: undefined (this is different function)
console.log(myMap.get({}));            // Output: undefined (this is different object)
```

Program 1: **Counting Word Frequency**

```
function countWordFrequency(text) {
  let words = text.split(" ");
  let wordCount = new Map();
  words.forEach(word => {
    let count = wordCount.get(word) || 0;
    wordCount.set(word, count + 1);
  });
  return wordCount;
}
let text = "hello world hello everyone";
let wordFrequency = countWordFrequency(text);
wordFrequency.forEach((count, word) => {
  console.log(`${word}: ${count}`);
});
```

```
hello: 2
world: 1
everyone: 1
```

- "JavaScript में forEach का behavior अलग data structures के लिए अलग होता है : arrays elements और indices पर iterate करता है, objects में Object.entries() से key-value pairs मिलते हैं, maps पहले value और फिर key देते हैं, और sets सिर्फ values पे iterate करते हैं।"

JavaScript Destructuring

JavaScript destructuring एक powerful feature है जो आपको arrays और objects के elements को easily extract करने की facility देता है, बिना manually हर एक value को access किये। इस feature से आप एक array या object के parts को easily variables में assign कर सकते हो।

1. Array Destructuring: Array destructuring से आप array के values को एक साथ multiple variables में assign कर सकते हैं।

```
let numbers = [10, 20, 30];
// Array destructuring
let [a, b, c] = numbers;
```

```
console.log(a);    // Output: 10
console.log(b);    // Output: 20
console.log(c);    // Output: 30
```

- इसमें numbers array के elements को direct a, b, c variables में assign कर दिया गया।

Default Value: अगर destructuring के time कोई value available नहीं होती, तो आप default value भी assign कर सकते हो।

```
let numbers = [10];
// Default value is set
let [a, b = 20] = numbers;
```

```
console.log(a);    // Output: 10
console.log(b);    // Output: 20
```

2. Object Destructuring: Object destructuring में आप objects के properties को easily extract कर सकते हो।

```
let person = {
  name: "Om",
  age: 27,
  city: "New York"
};
```

```
// Object destructuring
let { name, age, city } = person;
console.log(name);    // Output: Om
console.log(age);     // Output: 27
console.log(city);    // Output: New York
```

- यहाँ person object के properties name, age, और city को variables के साथ directly map कर दिया गया।

Rename Variable: आप destructuring के time भी variable का नाम change कर सकते हो।

```
let person = {
  name: "Om",
  age: 27,
  city: "New York"
};
```

```
// Variable renaming
let { name: fullName, age: years } = person;
console.log(fullName);    // Output: Om
console.log(years);    // Output: 27
```

Default Value in Objects: अगर object property नहीं हो तो आप default value भी दे सकते हो।

```
let person = {
  name: "Om",
  age: 27
};
```

```
// Default value for property 'city'
let { name, age, city = "Unknown" } = person;
console.log(city);    // Output: Unknown
```

3. Nested Destructuring: Destructuring के अंदर भी destructuring हो सकती है, जैसे arrays या objects के अंदर arrays या objects हो।

Example (Object with Nested Object):

```
let student = {
  name: "Alice",
  marks: {
    math: 85,
    science: 90
  }
};
```

```
// Nested destructuring
let { name, marks: { math, science } } = student;
console.log(math);    // Output: 85
console.log(science);    // Output: 90
```

Example (Array of Arrays):

```
let colors = ["red", ["green", "lightgreen"], "blue"];
// Nested array destructuring
let [primary, [secondary, light]] = colors;
```

```
console.log(primary);    // Output: red
console.log(secondary);    // Output: green
console.log(light);    // Output: lightgreen
```

Bitwise Operations

JavaScript Bitwise Operations binary (bits) level par काम करती हैं। ये operations numbers के binary form (0s and 1s) पर operate करते हैं। Bitwise operations JavaScript के अंदर integer values को treat करते हैं and 32-bit signed integers में convert करके काम करते हैं।

Common Bitwise Operators:

AND (&): इस operation में दो bits के corresponding positions को compare किया जाता है, और अगर दोनों 1 हैं, तो result भी 1 होता है, otherwise 0 होता है।

```
let a = 5;      // Binary: 0101
let b = 3;      // Binary: 0011
let result = a & b;      // Binary: 0001 (Decimal: 1)
console.log(result);     // Output: 1
```

OR (|): इस operation में अगर किसी एक position पर भी 1 होता है तो result 1 होता है, otherwise 0 होता है।

```
let a = 5;      // Binary: 0101
let b = 3;      // Binary: 0011
let result = a | b;      // Binary: 0111 (Decimal: 7)
console.log(result);     // Output: 7
```

XOR (^): XOR (exclusive OR) operator तब 1 return करता है जब दोनों bits different होते हैं। अगर दोनों same होते हैं (दोनों 0 या दोनों 1) तो result 0 होता है।

```
let a = 5;      // Binary: 0101
let b = 3;      // Binary: 0011
let result = a ^ b;      // Binary: 0110 (Decimal: 6)
console.log(result);     // Output: 6
```

NOT (~): NOT operator एक single number पर operate करता है। यह operator हर एक bit को invert कर देता है, 1 को 0 और 0 को 1 बना देता है। NOT operation का result signed integers में होता है, जिसमे negative value बनती है।

```
let a = 5;      // Binary: 00000000000000000000000000000101
let result = ~a;     // Binary: 11111111111111111111111111111010 (Decimal: -6)
console.log(result);     // Output: -6
```

Left Shift (<<): Left shift operation bits को left में shift करता है। जो rightmost bits खाली होती हैं, उन्हें zero से fill कर दिया जाता है।

```
let a = 5;      // Binary: 0101
let result = a << 1;     // Binary: 1010 (Decimal: 10)
console.log(result);     // Output: 10
```

Right Shift (>>): Right shift operation bits को right में shift करता है। जो leftmost bits खाली होती हैं, उन्हें sign bit (1 for negative, 0 for positive) से fill किया जाता है।

```
let a = 5;      // Binary: 0101
let result = a >> 1;     // Binary: 0010 (Decimal: 2)
console.log(result);     // Output: 2
```

Unsigned Right Shift (>>>): Unsigned right shift sign bit को ignore करता है और leftmost bits को हमेशा 0 से fill करता है। यह operator हमेशा positive result देता है।

```
let a = -5;      // Binary: 11111111111111111111111111111011 (negative number)
let result = a >>> 1;     // Binary: 01111111111111111111111111111101 (Decimal: 2147483645)
console.log(result);     // Output: 2147483645
```

JavaScript Hoisting

JavaScript Hoisting एक ऐसा behavior है जिसमे variables और functions को उनके actual declaration से पहले ही memory allocate हो जाती है। Hoisting का मतलब है की JavaScript code execution के दौरान variables और functions "top" पे move हो जाते हैं, चाहे उन्हें code में कहीं भी declare किया गया हो।

How Hoisting works?

JavaScript के interpreter के दो phases होते हैं :

Memory Creation Phase (Hoisting): Variables और functions के लिए memory allocate हो जाती है। Variables को undefined से initialize किया जाता है, और functions को उनके पूरे body के साथ memory में store किया जाता है।

Execution Phase: Code top to bottom execute होता है।

Variable Hoisting: Variables को hoist करते वक़्त, JavaScript उन्हें undefined से initialize करती है, चाहे आप उन्हें बाद में value assign करें।

```
console.log(x);     // Output: undefined
var x = 5;
console.log(x);     // Output: 5
```

- JavaScript पहले variable x को hoist करके undefined set कर देती है, इसलिए पहला console.log(x) undefined print करता है।
- फिर variable x को 5 की value assign होती है, और दूसरा console.log(x) 5 print करता है।

यह equivalent है :

```
var x;
console.log(x);     // undefined
x = 5;
console.log(x);     // 5
```

Function Hoisting: Functions को उनके definition के साथ hoist किया जाता है। आप function को पहले call कर सकते हैं और बाद में define कर सकते हैं।

```
greet();     // Output: Hello!
function greet() {
  console.log("Hello!");
}
```

- Function greet() को hoisting की वजह से पहले से memory में store कर दिया गया था, इसलिए हम पहले इससे call कर सकते हैं और बाद में define.

Function Expression and Hoisting: Function expressions में, variable declaration तो hoist होती है, लेकिन उसकी value नहीं hoist होती।

```
console.log(add);     // Output: undefined
var add = function() {
  console.log("Add function");
};
```

- Variable add तो hoist हो गया undefined के साथ, लेकिन function body memory में तब तक नहीं store होती जब तक वो define ना हो।

Question:

```
function output() {
  x = 33;
  console.log(x);
  var x;
}
output();     //33
```

Hoisting with Let and Const: let और const भी hoist किये जाते हैं, लेकिन उनकी value उनकी actual declaration से पहले use नहीं की जा सकती। इन्हे "temporal dead zone" (TDZ) के अंदर रखा जाता है जब तक उन्हें declare नहीं किया जाता।

```
console.log(a);     // Error: Cannot access 'a' before initialization
let a = 10;
```

Console.log()

console.log() JavaScript का एक built-in method है जो browser के console में output print करता है। यह debugging के लिए बहुत useful होता है, जिसमे आप variables, expressions, objects, arrays, etc. को print करके उन्हें easily देख सकते हैं।

Example:

```
console.log("Hello, World!");      // Hello, World
```

Formatting Output: console.log() में आप अलग-अलग data types के values को एक साथ print कर सकते हैं। आप एक ही console.log() में multiple values को comma से separate करके print कर सकते हैं।

```
let name = "Om";
let age = 27;
console.log("Name:", name, "Age:", age);     // Name: Om Age: 27
```

आप format specifiers का use करके भी output को style कर सकते हैं, जैसे %s for strings, %d for numbers.

```
console.log("Name: %s, Age: %d", name, age);     // Name: Om, Age: 27
```

Console Methods: console.log() के अलावा, console के और भी useful methods होते हैं :

- **console.error():** Error messages log करने के लिए।
- **console.warn():** Warnings show करने के लिए।
- **console.table():** Arrays या objects को table format में दिखाने के लिए।
- **console.time() and console.timeEnd():** Time calculations के लिए।

```
console.error("This is an error!");
console.warn("This is a warning!");
let students = [
  { name: "Om", age: 21 },
  { name: "Priya", age: 22 }
];
console.table(students);
console.time("myTimer");
for (let i = 0; i < 1000000; i++) {
  // Time-consuming task (like a large loop)
}
console.timeEnd("myTimer");      // myTimer: 125972.2470703125 ms
```

```
▶ This is an error
▶ This is a warn
```

(index)	name	age
0	'Om'	21
1	'Priya'	22

JavaScript Use Strict

"use strict" का use JavaScript में ज़्यादा safe और error-free code लिखने के लिए किया जाता है। जब हम "use strict" लिखते हैं, तो JavaScript कुछ strict rules follow करता है और कुछ गलत practices को रोक देता है, जिससे हमारे code में errors काम होते हैं और debugging easy हो जाती है।

How to Enable Strict Mode: Strict mode को दो तरीकों से enable किया जा सकता है :

- **Globally:** पूरे script file के लिए "use strict"; लिख कर।
- **Locally:** किसी specific function के अंदर "use strict"; लिख कर।

1. Global Strict Mode: जब "use strict"; को JavaScript file के top में लिखते हैं, तो strict mode पूरा script पे apply हो जाता है।

```
"use strict";
x = 5;      // Error: x is not defined
```

2. Local Strict Mode: आप strict mode को सिर्फ किसी specific function के अंदर भी apply कर सकते हैं।

```
function myFunction() {
  "use strict";
  y = 10;      // Error: y is not defined
}
myFunction();
```

Benefits of Strict Mode: Strict mode का use काफी सरे common errors और dangerous actions को रोकता है :

1. Prevents the use of undeclared variables:

```
// Without strict mode:
z = 10;      // No error, global variable created
```

```
// With strict mode:
"use strict";
z = 10;      // Error: z is not defined
```

2. Eliminates this keyword for global scope:

```
// Without strict mode:
function demo() {
  console.log(this);     // Output: Window object (in browsers)
}
demo();
```

```
// With strict mode:
"use strict";
function demo() {
  console.log(this);     // Output: undefined
}
demo();
```

3. Prevents duplicate parameter names: Strict mode में duplicate parameter names allowed नहीं होते।

```
SyntaxError: Duplicate parameter name not allowed in
    this context
```

```
"use strict";
function myFunction(a, a) {
  return a;
}
```

4. Throws errors on assignment to read-only properties: अगर किसी property को write-protect किया गया है और आप उसमे value change करते हैं, तो strict mode में error मिलेगा।

```
"use strict";
const obj = {};
Object.defineProperty(obj, "prop", { value: 10, writable: false });
obj.prop = 20;
```

```
TypeError: Cannot assign to read only property 'prop'
    of object '#<Object>'
```

5. Disallows deleting variables and functions: Strict mode में variables या functions को delete करना allowed नहीं होता।

```
"use strict";
var myVar = 10;
delete myVar;      // Error: Cannot delete variable
```

6. Prevents usage of with statement: with statement को strict mode में allowed नहीं किया जाता क्यूंकि यह code को ambiguous बना देता है।

```
"use strict";
with (Math) {      // Error: Strict mode code may not include a with statement
  console.log(PI);
}
```

This Keyword

JavaScript में this keyword एक special keyword होता है, जो किसी function, method, या object के current execution context को refer करता है । यानी, this का reference context के हिसाब से change होता है।

1. this in Global Context: जब this global scope में use किया जाता है (functions के बाहर), तो यह global object को refer करता है। Browser में, यह window object होता है।

```
console.log(this);     // Output: Window (in browser)
```

2. this in Function Context: जब this keyword को किसी function के अंदर use किया जाता है, तो यह global object (window) को refer करता है अगर function को directly call किया जाये।

```
function show() {
  console.log(this);
}
show();     // Output: Window (browser)
```

लेकिन अगर strict mode ("use strict") on हो, तो this undefined return करेगा।

```
"use strict";
function show() {
  console.log(this);
}
show();  // Output: undefined
```

3. this in Method: अगर this को किसी object के method में use किया जाता है, तो यह उस object को refer करता है।

```
let person = {
  firstName: "Om",
  lastName: "Mishra",
  fullName: function() {
    console.log(this.firstName + " " + this.lastName);
  }
};
person.fullName();     // Output: Om Mishra
```

- इस example में, this object person को refer करता है, क्यूंकि method person object का part है।

4. this in Constructor Functions: जब this को किसी constructor function के अंदर use करते हैं, तो यह नया object refer करता है जो बनाया गया है।

```
function Car(make, model) {
  this.make = make;
  this.model = model;
}
let myCar = new Car("Toyota", "Corolla");
console.log(myCar.make);     // Output: Toyota
```

- यहाँ this new object myCar को refer करता है।

5. this in Event Handlers: अगर this keyword को event handler के अंदर use करते हैं, तो यह उस HTML element को refer करता है जिस पर event trigger हुआ।

```
<button onclick="showThis()">Click me</button>
<script>
function showThis() {
  console.log(this);     // Output: <button> element
}
</script>
```

- यहाँ, this button element को refer करता है।

6. this in Arrow Functions: Arrow functions में this keyword का behavior थोड़ा अलग होता है। यह अपने surrounding (lexical) context से this को inherit करता है, यानि this कभी भी arrow function के अंदर नहीं बदलता।

```
let person = {
  name: "Om",
  greet: () => {
    console.log(this.name);
  }
};
person.greet();          // Output: undefined (arrow function window object से 'this' को inherit करता है )
```

अगर आपको object का this चाहिए, तो normal function का use करना पड़ेगा :

```
let person = {
  name: "Om",
  greet: function() {
    console.log(this.name);
  }
};
person.greet();     // Output: Om
```

7. Class Method: Classes में this class के instance को refer करता है। यह instance के properties और methods को access करने के लिए use होता है।

```
class Person {
 constructor(name) {
  this.name = name;
 }
 greet() {
  console.log('Hello, ' + this.name);     // 'this' refers to the Person instance
 }
}
const om = new Person('Om');
om.greet();     // Output: Hello, Om
```

JavaScript Object-Oriented Programming (OOP)

OOP (Object-Oriented Programming) एक programming paradigm है जो objects को use करके data और methods को organize करता है। JavaScript में भी OOP को support किया जाता है, और इसमें आप classes और objects का use करके अपने code को well-structured और reusable बना सकते हैं।

What is OOP?

OOP एक approach है जिसमे real-world entities को objects के form में represent किया जाता है। इसमें objects का use होता है जो अपने properties (data) और methods (functions) को encapsulate करते हैं।

JavaScript Classes: JavaScript में classes एक template के जैसे काम करती हैं जो objects बनाने के लिए use होती हैं। JavaScript ES6 में classes को introduce किया गया था, जो object-oriented programming के concepts को easy और readable way में implement करने के लिए काम आता है।

Classes, JavaScript में function का syntactical improvement हैं। Classes के through हम objects को easily define कर सकते हैं, methods को group कर सकते हैं, और inheritance को efficiently handle कर सकते हैं।

Creating Classes: Class बनाने के लिए class keyword का use होता है। इसके अंदर constructor और methods define किये जा सकते हैं।

Syntax:

```
class ClassName {
  constructor(parameters) {
    // Initialize properties
  }

  // Methods
}
```

Example:

```
class Person {
  constructor(firstName, lastName, age) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.age = age;
  }
}
```

Constructor Method: constructor() method एक special method होता है जो object बनाते वक़्त automatically call होता है। इसमें initial values set की जाती हैं।

Creating Objects: एक class से object create करने के लिए new keyword का use होता है।

```
let person1 = new Person("Om", "Mishra", 27);
let person2 = new Person("Aman", "Tiwari", 23);

console.log(person1);     // Person { firstName: 'Om', lastName: 'Mishra', age: 27 }
console.log(person2);     //Person { firstName: 'Aman', lastName: 'Tiwari', age: 23}
```

Adding method to a Class: Class के अंदर आप methods define कर सकते हो जो आपके object पे actions perform करेंगे।

Syntax:

```
methodName(){
    //code
  }
```

Example:

```
class Person {
  constructor(firstName, lastName, age) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.age = age;
  }
  getFullName(){
    return `Full Name is ${this.firstName} ${this.lastName}.`
  }
}
let person1 = new Person("Om", "Mishra", 27);
```

Accessing Object properties and methods: आप object की properties को dot (.) notation से access कर सकते हो, जैसे :

```
console.log(person1.age);       //27
console.log(person1.getFullName());     // Full Name is Om Mishra.
```

Modifying Object properties: आप object की properties को update या modify भी कर सकते हो :

```
person1.name = 'Rahul';
console.log(person1.name);     // Output: Rahul
```

Static Methods: Static methods वो methods होते हैं जो directly class पे call किये जाते हैं, object पे नहीं।

```
class Calculator {
  static add(a, b) {
    return a + b;
  }
}

console.log(Calculator.add(5, 3));     // Output: 8
```

Simple Program using Class and Object:

Program 1: Bank Account

```
class BankAccount {
 constructor(accountNumber, accountHolder,
balance) {
   this.accountNumber = accountNumber;
   this.accountHolder = accountHolder;
   this.balance = balance;
 }
 deposit(amount) {
   this.balance += amount;
 }
 withdraw(amount) {
   if (amount > this.balance) {
     console.log("Insufficient funds");
   } else {
     this.balance -= amount;
   }
 }
 getBalance() {
   return this.balance;
 }
}
let account = new BankAccount("12345", "Om
Mishra", 15000);
account.deposit(5000);
account.withdraw(10000);
console.log("Balance:", account.getBalance());
// The balance is 10000
```

Program 2: Library Management

```
class Book{
   constructor(title, author, isbn){
      this.title = title;
      this.author = author;
      this.isbn = isbn;
   }
   getDetails(){
      return `${this.title} by ${this.author} with isbn:
${this.isbn}`;
   }
}
class Library{
   constructor(){
      this.books = [];
   }
   addBook(book){
      this.books.push(book);
   }
   removeBook(isbn){
      this.books = this.books.filter(book => book.isbn
!= isbn);
   }
   getBooks(){
      return this.books.map(book =>
book.getDetails());
   }
}
let library = new Library();
let book1 = new Book ("War and Peace", "Leo
Tolstoy", "12345");
let book2 = new Book("Great Expectations",
"Charles Dickens", "54321");
library.addBook(book1);
library.addBook(book2);
library.removeBook("12345");
console.log(library.getBooks());     // [ 'Great
Expectations by Charles Dickens with isbn: 54321' ]
```

JavaScript Getters and Setters: Getters and Setters JavaScript में properties को access और modify करने का एक structured तरीका provide करते हैं। यह methods की तरह behave करते हैं, लेकिन जब आप object की properties को access या update करते हो, तो यह as a property काम करते हैं, बिना method call की तरह () use करे।

What are Getters and Setters?

- Getters एक method होता है जो object की property को read (get) करता है। इसका use private या protected properties को securely access करने के लिए किया जाता है।

- Setters एक method होता है जो object की property को update (set) करता है। यह values को validate करने के लिए helpful होता है। It always have 1 parameter.

How to create Getters and Setters?

JavaScript में, आप get और set keywords का use करके getters और setters define कर सकते हो।

Syntax:

```
class ClassName {
  constructor(parameters) {
    // Initialize properties
  }
  // Getter
  get propertyName() {
    // Return a value
  }
  // Setter
  set propertyName(value) {
    // Modify a value
  }
}
```

Example:

```
class Person {
  constructor(firstName, lastName, age) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.age = age;
    this.middleName = null;     // Initializing middleName
  }
  get FullName() {
    if (this.middleName) {
      return `Full Name is ${this.firstName} ${this.middleName}
${this.lastName}.`;
    } else {
      return `Full Name is ${this.firstName} ${this.lastName}.`;
    }
  }
  set FullName(middleName) {
    this.middleName = middleName;    // Setting middleName via setter
  }
}
let person1 = new Person("Om", "Mishra", 27);
```

3. How to use Getters and Setters?

जब आप getters और setters use करते हो, तो आपको उन्हें as properties treat करना होता है, methods की तरह नहीं :

```
// Setting the middle name using the setter
person1.FullName = "Shankar";
// Now get the full name by Invoking getter directly
console.log(person1.FullName);      // Full Name is Om Shankar Mishra.
```

Advantages of Getters and Setters

Data Encapsulation: Getters और setters आपके object properties को direct access से protect करते हैं। आप specific rules define कर सकते हो की property कब और कैसे modify हो सकती है।

Validation: Setters की help से आप input values को validate कर सकते हो। अगर कोई गलत value provide करता है, तो आप validation के through error handle कर सकते हो।

Computed Properties: Getters आपको dynamically computed values provide करने का option देते हैं बिना property को explicitly store किये हुए।

Free Billangual Notes Available !

HTML | CSS | JavaScript | Angular | React | Hibernate

SpringBoot | JDBC | Git & GitHub | Data Analyst

www.linkedin.com/in/omi92229

Basics of HTML

What is HTML?

Definition: HTML का full form HyperText Markup Language है। यह एक standard language है जो web pages बनाने और design करने के लिए use होती है। यह web pages को structure provide करता है अलग -अलग elements और tags का उसे करके जो content और layout define करते हैं।

History: HTML को Tim Berners-Lee ने 1991 में बनाया था। यह समय के साथ evolve होता गया, और कई versions release हुए जो इसकी capabilities को enhance करते गए। अभी जो latest version है, वह HTML5 है, जो नए elements और features introduce करता है जिससे multimedia support, semantics, और applications और robust ho गए हैं।

Key Features:

- **Structure:** HTML headings, paragraphs, lists, links, और दूसरे content types का use करके web pages का structure define करता है।
- **Elements और Tags:** HTML tags का use करता है जो angle brackets में होते हैं (जैसे <tagname>). ज़्यादातर elements के पास opening tag और closing tag होता है (जैसे *** content ***), पर कुछ self-closing भी होते हैं (जैसे ***<img src="image.jpg" />***).
- **Attributes:** Tags के पास attributes होते हैं जो element के बारे में additional information देते हैं (जैसे ***<a href="https://example.com">link</a>***). Attributes opening tag के अंदर लिखे जाते हैं।
- **Hypertext:** HTML hyperlinks बनाने की सुविधा देता है, जो users को different web pages या resources के बीच navigate करने में मदद करता है।

Importance: HTML web development का foundation है। HTML के बिना web pages के पास basic

Data Science | SQL | Mongo DB | JAVA

Om Shankar Mishra

Data Science with R Programming

Lesson 1: Introduction to Business Analytics

Business analytics का मतलब है statistical analysis, data mining, और predictive modeling का use करके data को analyze और interpret करना, ताकि informed business decisions लिए जा सकें। यह modern business strategies का एक essential part है, जो valuable insights provide करता है, जिससे organizations market trends, customer preferences, और operational efficiencies को समझ सकते हैं।

Overview

Business analytics एक process है जो data को actionable insights में convert करता है, ताकि business outcomes को improve किया जा सके। यह various techniques और methodologies को encompass करता है जो historical data को समझने, future trends को predict करने, और data-driven decisions लेने में help करता है। इसका main goal है data को leverage करके business planning और strategic initiatives को drive करना।

Business Decisions and Analytics

Analytics, business decision-making में एक crucial role play करता है , क्यूंकि यह evidence-based insights provide करता है जो strategic planning, operational improvements, और competitive strategies को guide करता है। Analytics-supported business decisions से efficiency बढ़ सकती है, costs कम हो सकती हैं, customer satisfaction improve हो सकती है, और profitability बढ़ सकती है। Decision-making process में problem को identify करना, relevant data collect करना , data analyze करना, insights generate करना, और best course of action implement करना शामिल है।

Types of Business Analytics

Simple Programs using Getters and Setters

Example 1: Temperature Conversion

```
class Temperature {
 constructor(celsius) {
  this.celsius = celsius;
 }
 // Getter for Fahrenheit
 get fahrenheit() {
  return this.celsius * 9/5 + 32;
 }
 // Setter for Fahrenheit
 set fahrenheit(value) {
  this.celsius = (value - 32) * 5/9;
 }
}
let temp = new Temperature(25);
console.log("Celsius:", temp.celsius);     // Celsius: 25
console.log("Fahrenheit:", temp.fahrenheit);     // Fahrenheit: 77
temp.fahrenheit = 100;
console.log("Celsius after setting Fahrenheit:", temp.celsius);     // Celsius .... Fahrenheit: 37.778
```

Example 2: Rectangle Area and Perimeter

```
class Rectangle {
 constructor(width, height) {
  this.width = width;
  this.height = height;
 }
 // Getter for area
 get area() {
  return this.width * this.height;
 }
 // Getter for perimeter
 get perimeter() {
  return 2 * (this.width + this.height);
 }
}
let rect = new Rectangle(10, 5);
console.log("Width:", rect.width);       // Width: 10
console.log("Height:", rect.height);      // Height: 5
console.log("Area:", rect.area);        // Area: 50
console.log("Perimeter:", rect.perimeter);      // Perimeter: 30
```

JavaScript Encapsulation: Encapsulation object-oriented programming (OOP) का एक important concept है जिसमे data (properties) और methods को एक object के अंदर bundle किया जाता है। इसका basic idea यह है की object की internals को direct access ना दिया जाए, और सिर्फ defined methods (getters/setters) के through interact किया जाए। Encapsulation से data को protect किया जा सकता है और accidental modification से रोका जा सकता है।

Benefits of Encapsulation:

Data Protection: Encapsulation की help से आप object के data को hide कर सकते हैं और सिर्फ defined methods के ज़रिये ही access allow करते हैं।

Control: आप data को modify करने के तरीके को control कर सकते हैं, जैसे की setters में validation implement कर सकते हैं।

Maintainability: Encapsulation object के internal implementation को change करने की flexibility provide करता है बिना external code को affect किये।

Security: Encapsulation से आप private fields create कर सकते हैं जिनका external access directly possible नहीं होता।

Implementing Encapsulation: Encapsulation को implement करने के दो main तरीके हैं :

1. Using Private Fields: JavaScript में, encapsulation implement करने के लिए private fields का use किया जाता है। Private fields को underscore _ या # symbol के साथ declare किया जाता है।

```
class Person {
 // Private fields
 #firstName;
 #lastName;

 constructor(firstName, middleName, lastName) {
  this.#firstName = firstName;
  this.middleName = middleName;
  this.#lastName = lastName;
 }

 // Public method to get full name
 getFullName() {
  return `${this.#firstName} ${this.middleName} ${this.#lastName}`;
 }
}

let person = new Person("Om", "Shankar", "Mishra");
console.log(person.getFullName());      // Om Shankar Mishra
console.log(person.middleName);      // Shankar
console.log(person.#firstName);      // This will cause an error
```

2. Using Getters and Setters: Getters और setters encapsulation implement करने का एक और तरीका हैं। यह आपको data को access और modify करने के लिए control provide करते हैं।

```
class Person {
 #firstName;
 #lastName;

 constructor(firstName, lastName) {
  this.#firstName = firstName;
  this.#lastName = lastName;
 }

 // Getter for full name
 get fullName() {
  return `${this.#firstName} ${this.#lastName}`;
 }

 // Setter for full name
 set fullName(name) {
  let parts = name.split(' ');
  this.#firstName = parts[0];
  this.#lastName = parts[1];
 }
}

let person = new Person("Om", "Mishra");
console.log(person.fullName);      // Om Mishra

person.fullName = "Priyansi Tamrakar";
console.log(person.fullName);      // Priyansi Tamrakar
```

Simple Programs using Encapsulation

Example 1: Employee Management with Encapsulation

```
class Employee {
 #pament;

 constructor(name, position, pament) {
  this.name = name;
  this.position = position;
  this.#pament = pament;
 }

 // Getter for salary
```

```
 get salary() {
  return this.#pament;
 }
 // Setter for salary with validation
 set salary(amount) {
  if (amount > 0) {
   this.#pament = amount;
  } else {
   console.log("Salary must be a positive number");
  }
 }
 // Method to get employee details
 getDetails() {
  return `${this.name}, ${this.position}, Salary: ${this.#pament}`;
 }
}
let emp = new Employee("Om Mishra", "Manager", 80000);
console.log(emp.getDetails());      // Om Mishra, Manager, Salary: 80000
emp.salary = 90000;      // Valid update
console.log(emp.getDetails());      // Om Mishra, Manager, Salary: 90000
emp.salary = -5000;      // Salary must be a positive number
```

Example 2: **Student Grades with Encapsulation**

```
class Student {
 _grades;
 constructor(name, grades) {
  this.name = name;
  this._grades = grades;
 }
 // Getter for grades
 get grades() {
  return this._grades;
 }
 // Setter for grades with validation
 set grades(newGrades) {
  if (Array.isArray(newGrades) && newGrades.every(grade => grade >= 0 && grade <= 100)) {
   this._grades = newGrades;
  } else {
   console.log("All grades should be between 0 and 100");
  }
 }
 // Getter for average grade
 get averageGrade() {
  let sum = this._grades.reduce((acc, grade) => acc + grade, 0);
  return sum / this._grades.length;
```

```
 }
}
let student = new Student("Priya", [85, 90, 78]);
console.log("Initial Grades:", student.grades);      // Initial Grades: [ 85, 90, 78 ]
console.log("Average Grade:", student.averageGrade);      // Average Grade: 84.33
student.grades = [88, 92, 80];      // Valid update
console.log("Updated Grades:", student.grades);      // Updated Grades: [ 88, 92, 80 ]
console.log("New Average Grade:", student.averageGrade);      // New Average Grade: 86.67
student.grades = [105, 90, 78];      // All grades should be between 0 and 100
```

JavaScript Inheritance: Inheritance JavaScript में एक object-oriented programming (OOP) का concept है जिसमे एक class दूसरी class की properties और methods inherit करती है। इससे एक class (child class) दूसरी class (parent class) के सारे features ले लेती है, बिना बार-बार लिखने की ज़रूरत के।

Benefits of Inheritance:

- **Code Reusability:** Parent class के methods और properties को child class directly use कर सकती है, जिससे code reuse होता है और redundancy कम होती है।
- **Extensibility:** Child classes में parent class के behavior को customize या extend किया जा सकता है।
- **Maintainability:** Code को maintain करना आसान हो जाता है, क्यूंकि एक base class में modification से सब child classes में automatically changes हो जाते हैं।
- **Organized Structure:** Objects और classes को hierarchical way में organize किया जा सकता है, जिससे code logically arranged और readable होता है।

Creating a Base Class: Base class (या parent class) वह class होती है जिसमे common properties और methods defined होते हैं जो child classes inherit करती हैं।

```
class Person {
 constructor(firstName, lastName, age) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.age = age;
 }
 getFullName() {
  return `${this.firstName} ${this.lastName}`;
 }
}
let person = new Person("Om", "Mishra", 27);
console.log(person.getFullName());    // Om Mishra
```

Extending a Class: Child class को create करने के लिए extends keyword का use किया जाता है। Child class parent class के सारे methods और properties inherit करती है।

```
class Student extends Person {
 constructor(firstName, lastName, age, course)
{
  super(firstName, lastName, age);
  this.course = course;
 }
 getCourse() {
  return this.course;
 }
}
let student = new Student("Khushboo", "Makhija", 23,
"MCA");
console.log(student.getFullName());   // Khushboo Makhija
console.log(student.getCourse());      // MCA
```

Using the super Keyword: super keyword का use parent class के constructor या methods को call करने के लिए किया जाता है। अगर आपको parent class का constructor use करना हो, तो super() का call child class के constructor के अंदर करना ज़रूरी होता है।

```
class Teacher extends Person {
  constructor(firstName, lastName, age, subject) {
    super(firstName, lastName, age);
    this.subject = subject;
  }
  getSubject() {
    return this.subject;
  }
}
let teacher = new Teacher("Priyanka", "Pandey", 28, "Data Mining");
console.log(teacher.getFullName());      // Priyanka Pandey
console.log(teacher.getSubject());       // Data Mining
```

Method Overriding: Method overriding तब होती है जब child class अपने parent class के method को overwrite (replace) कर देती है। Child class अपने method में parent class के method को भी call कर सकती है using super.

```
class Employee extends Person {
  constructor(firstName, lastName, age, jobTitle) {
    super(firstName, lastName, age);
    this.jobTitle = jobTitle;
  }
  getFullName() {
    return `${this.firstName} ${this.lastName}, ${this.jobTitle}`;
  }
}
let employee = new Employee("Priyansi", "Tamrakar", 23, "Principal");
console.log(employee.getFullName());      // Priyansi Tamrakar, Principal
```

Inheritance with Constructors: JavaScript में जब एक child class का constructor define होता है, तो parent class का constructor automatically call नहीं होता। इसलिए हमेशा super() का call constructor के अंदर करना ज़रूरी होता है, वरना error आता है।

```
class Vehicle {
  constructor(make, model) {
    this.make = make;
    this.model = model;
  }
  getDetails() {
    return `${this.make} ${this.model}`;
  }
}
class Car extends Vehicle {
  constructor(make, model, year) {
    super(make, model);
    this.year = year;
  }
  getDetails() {
    return `${super.getDetails()} (${this.year})`;
  }
}
let car = new Car("Toyota", "Camry", 2021);
console.log(car.getDetails());      // Toyota Camry (2021)
```

Simple Programs using Inheritance: Example 1: Library Management with Inheritance

```
class Book {
 constructor(title, author, isbn) {
  this.title = title;
  this.author = author;
  this.isbn = isbn;
 }
 getDetails() {
  return `${this.title} by ${this.author}, ISBN: ${this.isbn}`;
 }
}
class EBook extends Book {
 constructor(title, author, isbn, fileSize) {
  super(title, author, isbn);
  this.fileSize = fileSize;
 }
 getDetails() {
  return `${super.getDetails()}, File Size: ${this.fileSize} MB`;
 }
}
let ebook = new EBook("Godaan", "PremChand", "123456", 2);
console.log(ebook.getDetails());      // The Godaan by PremChand, ISBN: 123456, File Size: 2 MB
```

Example 2: Basic Inheritance

```
class Person {
 constructor(name, age) {
  this.name = name;
  this.age = age;
 }
 introduce() {
  console.log(`Hi, I am ${this.name} and I am ${this.age} years old.`);
 }
}
class Student extends Person {
 constructor(name, age, grade) {
  super(name, age);        // Call the parent class constructor
  this.grade = grade;
 }
 study() {
  console.log(`${this.name} is studying in grade ${this.grade}.`);
 }
}
let student = new Student("Omi", 15, "10th");
student.introduce();        // Output: Hi, I am Omi and I am 15 years old.
student.study();        // Output: Aman is studying in grade 10th.
```

Example 3: **Overriding Parent Method**

```
class Employee {
 constructor(name, salary) {
  this.name = name;
  this.salary = salary;
 }
 details() {
  console.log(`${this.name} earns ${this.salary} per year.`);
 }
}
class Manager extends Employee {
 constructor(name, salary, department) {
  super(name, salary);      // Call the parent class constructor
  this.department = department;
 }
 details() {
  // Overriding the parent method
  console.log(`${this.name} is the manager of ${this.department} department and earns ${this.salary}.`);
 }
}
let manager = new Manager("Om", 75000, "Sales");
manager.details();       // Output: Om is the manager of Sales department and earns 75000.
```

JS Polymorphism: Polymorphism एक object-oriented programming (OOP) का concept है जिसमे एक ही method या function अलग -अलग तरीकों से behave कर सकता है, depending on the object या class जिसपे वो apply किया गया हो। JS में polymorphism का main example method overriding होता है, जिसमे एक child class parent class के method को overwrite या customize करती है।

- Poly का मतलब "many" और morphism का मतलब "forms"। तो polymorphism का simple मतलब है "many forms"।

Why Use Polymorphism?

- **Code Reusability:** Polymorphism के through हम एक method को अलग classes में reuse कर सकते हैं, अलग behavior के साथ।
- **Flexibility:** Different objects के साथ same method का अलग -अलग behavior होता है, जो code को flexible और dynamic बनाता है।
- **Maintainability:** Changes को easily implement कर सकते हैं, क्यूंकि parent-child classes के through specific methods को override करना आसान होता है।
- **Extensibility:** Classes के behavior को easily extend कर सकते हैं, जिससे नए functionalities add किया जा सकता है बिना base code को break किये।

Method Overriding (Polymorphism in JavaScript): Method overriding polymorphism का एक important aspect होता है। जब child class अपने parent class के method को overwrite करती है, तो

यह method overriding कहलाता है। इससे same method name के साथ different behavior create किया जा सकता है।

```
class Animal {
 makeSound() {
  return "Some generic sound";
 }
}

class Dog extends Animal {
 makeSound() {
  return "Bark";
 }
}
```

```
class Cat extends Animal {
 makeSound() {
  return "Meow";
 }
}

let animals = [new Dog(), new Cat()];

for (let animal of animals) {
 console.log(animal.makeSound());
}     // Bark      //Meow
```

Polymorphism in Action: Polymorphism एक ऐसा concept है जो आपको अलग -अलग classes के objects को एक common interface के through treat करने की ability देता है। आप एक code लिख सकते हो जो अलग -अलग classes के objects के साथ काम कर सके बिना इस बात की चिंता किये की object किस class का है।

```
class Shape {
 getArea() {
  return 0;
 }
}

class Circle extends Shape {
 constructor(radius) {
  super();
  this.radius = radius;
 }

 getArea() {
  return Math.PI * Math.pow(this.radius, 2);
 }
}
```

```
class Rectangle extends Shape {
 constructor(width, height) {
  super();
  this.width = width;
  this.height = height;
 }

 getArea() {
  return this.width * this.height;
 }
}

let shapes = [new Circle(5), new Rectangle(4, 6)];

for (let shape of shapes) {
 console.log(shape.getArea());
}     // 78.5398      // 24
```

Simple Program using Polymorphism

Program 1: Payment System with Polymorphism

```
class Payment {
 processPayment() {
  return "Processing generic payment";
 }
}

class CreditCardPayment extends Payment {
 processPayment() {
  return "Processing credit card payment";
 }
}
```

```
class PayPalPayment extends Payment {
 processPayment() {
  return "Processing PayPal payment";
 }
}

let payments = [new CreditCardPayment(), new
PayPalPayment()];

for (let payment of payments) {
 console.log(payment.processPayment());
}
```

```
Processing credit card payment
Processing PayPal payment
```

Program 2: **Transportation System with Polymorphism**

```
class Vehicle {
 move() {
  return "Moving";
 }
}

class Car extends Vehicle {
 move() {
  return "Driving a car";
 }
}

class Bike extends Vehicle {
 move() {
  return "Riding a bike";
 }
}

let vehicles = [new Car(), new Bike()];

for (let vehicle of vehicles) {
 console.log(vehicle.move());
}
```

```
Driving a car
Riding a bike
```

Program 3: **Appliance Control with Polymorphism**

```
class Appliance {
 turnOn() {
  return "Turning on appliance";
 }
}

class WashingMachine extends Appliance {
 turnOn() {
  return "Turning on washing machine";
 }
}

class Refrigerator extends Appliance {
 turnOn() {
  return "Turning on refrigerator";
 }
}

let appliances = [new WashingMachine(), new Refrigerator()];

for (let appliance of appliances) {
 console.log(appliance.turnOn());
}
```

```
Turning on washing machine
Turning on refrigerator
```

Program 4: **Polymorphism in Function Arguments (Document Handling)**

```
class Printer {
 print(document) {
  console.log("Printing document...");
 }
}

class PDFPrinter extends Printer {
 print(document) {
  console.log("Printing PDF document...");
 }
}

class WordPrinter extends Printer {
 print(document) {
  console.log("Printing Word document...");
 }
}

function printDocument(printer, document) {
 printer.print(document);
}

let pdfPrinter = new PDFPrinter();
let wordPrinter = new WordPrinter();

printDocument(pdfPrinter, "File.pdf");
printDocument(wordPrinter, "File.docx");
```

```
Printing PDF document...
Printing Word document...
```

- इस example में printDocument() function polymorphic है क्यूंकि यह अलग-अलग printers (PDFPrinter और WordPrinter) को handle करता है, और सबके print methods अलग behavior दिखाते हैं।

JavaScript super Keyword: super JavaScript का एक keyword है जो inheritance के context में use होता है। जब हम एक class को inherit करते हैं (एक class को दूसरी class से extend करते हैं), तो super keyword का use parent class के constructor या methods को access करने के लिए किया जाता है।

Key Uses of super:

- Parent class के constructor को call करने के लिए।
- Parent class के methods को access करने के लिए।

Using super in Constructors: जब एक child class का constructor parent class के constructor के कुछ properties या methods use करना चाहता है, तब हम super() का use करते हैं। super() को child class के constructor में सबसे पहले call करना पड़ता है, क्यूंकि यह parent class का constructor invoke करता है।

```
class Person {
 constructor(name) {
  this.name = name;
 }
}

class Employee extends Person {
 constructor(name, jobTitle) {
  super(name);     // Calling parent class constructor
  this.jobTitle = jobTitle;
 }

 displayInfo() {
  console.log(`Name: ${this.name}, Job Title: ${this.jobTitle}`);
 }
}

const emp = new Employee("Om Mishra", "Software Developer");
emp.displayInfo();     // Output: Name: Om Mishra, Job Title: Software Developer
```

- Employee class ने Person class को extend किया है।
- super(name) का use किया गया है ताकि Person class का constructor call हो सके और name set हो जाए।
- इसके बाद, jobTitle को Employee class में initialize किया गया है।

Using super to Call Parent Methods: जब child class parent class के methods को access करना चाहती है या उन्हें call करना चाहती है, तब भी हम super का use करते हैं। यह help करता है parent methods को invoke करने में without re-implementing them.

```
class Animal {
 speak() {
  console.log("The animal makes a sound.");
 }
}

class Dog extends Animal {
 speak() {
  super.speak();     // Calling parent class method
  console.log("The dog barks.");
 }
}

let myDog = new Dog();
myDog.speak();
```

- Dog class का speak() method parent class Animal के speak() method को super.speak() से call कर रहा है।
- इसके बाद अपना custom message add कर रहा है : "The dog barks."

Simple Program using super

Example 1: Using super in Constructors

```
class Vehicle {
 constructor(brand) {
```

```
  this.brand = brand;
 }
 displayBrand() {
  console.log(`Brand: ${this.brand}`);
 }
}
class Car extends Vehicle {
 constructor(brand, model) {
  super(brand);   // Calling parent class constructor
  this.model = model;
 }
 displayCarInfo() {
  super.displayBrand();  // Calling parent class method
  console.log(`Model: ${this.model}`);
 }
}
let myCar = new Car("Toyota", "Corolla");
myCar.displayCarInfo();
```

```
Brand: Toyota
Model: Corolla
```

- Car class parent class Vehicle को extend करती है।
- super(brand) को use किया गया है parent class के constructor को call करने के लिए।
- Parent class के displayBrand() method को भी super.displayBrand() से access किया गया है।

Example 2: Using super in Method Overriding

```
class Animal {
 sound() {
  console.log("Animals make different sounds.");
 }
}
class Cat extends Animal {
 sound() {
  super.sound();     // Calling parent class method
  console.log("Cats meow.");
 }
}
let myCat = new Cat();
myCat.sound();
```

```
Animals make different sounds.
Cats meow.
```

- Cat class के sound() method में parent class का sound() method call होता है super.sound() से।
- इसके बाद child class का अपना custom message print होता है। "Cats meow."

JavaScript Prototype Object: JavaScript में prototype एक mechanism है जो objects और functions को properties और methods inherit करने की ability देता है। जब भी एक object या function create होता है, उसके साथ एक hidden property attach होती है, जिसे prototype कहते हैं।

- हर JavaScript object के पास एक internal reference होता है जो उसके prototype को point करता है।
- Prototypes code reusability को promote करते हैं, जिसमे objects एक दूसरे से properties और methods share कर सकते हैं।

The prototype Property

- Functions और objects JavaScript में default से एक property के साथ आते हैं जिसे prototype कहते हैं।
- जब आप किसी function या object को new keyword के through create करते हो, तो उसका prototype automatically set हो जाता है।

```
function Person(name, age) {
 this.name = name;
 this.age = age;
}
Person.prototype.greet = function() {
 console.log("Hello, my name is " + this.name);
};
let person1 = new Person("Om", 27);
person1.greet();
```

```
Hello, my name is Om
```

- Person एक constructor function है।

- हमने Person.prototype.greet method को define किया, जो की हर Person object के लिए available हो गया।
- person1 के object ने greet() method को inherit किया via prototype.

Prototype Chain

- JS में objects एक prototype chain follow करते हैं। अगर किसी object के अंदर कोई property या method नहीं मिलता, तो JavaScript automatically उसके prototype में search करता है।
- यह process तब तक चलता है जब तक prototype chain के end तक नहीं पहुँच जाते या property मिल नहीं जाती।

```
let obj = {};
console.log(obj.toString());    // toString method is from Object prototype
```

- obj एक simple object है, लेकिन हम उसपर toString() method call कर सकते हैं क्यूंकि obj ने Object prototype से method inherit किया है।

Inheriting Properties and Methods: Objects अपने parent prototypes से properties और methods inherit करते हैं। यह inheritance prototype chain के through होता है, जो allow करता है की objects अपने base prototypes से functionality share कर सकें।

```
function Animal(name) {
  this.name = name;
}
Animal.prototype.speak = function() {
  console.log(this.name + " makes a noise.");
};
function Dog(name) {
  Animal.call(this, name);    // Calling parent constructor
}
Dog.prototype = Object.create(Animal.prototype);    // Inheriting properties
Dog.prototype.constructor = Dog;
let dog = new Dog("Rex");
dog.speak();    // Output: Rex makes a noise.
```

- Dog ने Animal से properties और methods inherit किया है using Object.create().
- speak() method Dog के लिए inherit हो गया parent prototype से।

Adding Methods to a Prototype: आप objects के prototypes में directly methods या properties add कर सकते हैं, और वो सब objects जो उस prototype को inherit कर रहे हैं, उन methods को access कर सकते हैं।

```
function Car(brand, model) {
  this.brand = brand;
  this.model = model;
}
Car.prototype.displayInfo = function() {
  console.log("Car: " + this.brand + " " + this.model);
};
let car1 = new Car("Toyota", "Corolla");
let car2 = new Car("Honda", "Civic");
car1.displayInfo();    // Output: Car: Toyota Corolla
car2.displayInfo();    // Output: Car: Honda Civic
```

- Car.prototype.displayInfo method को prototype पे add किया गया, जो हर Car instance के लिए available हो गया।

Prototype Chain in Action: जब JavaScript किसी property या method को search करता है, तो पहले वो object में देखता है, अगर नहीं मिलता, तो वो prototype chain में आगे search करता है।

```
let arr = [1, 2, 3];
console.log(arr.toString());    // Calls the Array prototype method toString()
// If not found in Array, it searches in Object.prototype
```

Simple Program using Prototypes

Example: Adding Methods to a Prototype

```
function Laptop(brand, model) {
  this.brand = brand;
  this.model = model;
}
// Adding method to prototype
Laptop.prototype.getDetails = function() {
  return `Brand: ${this.brand}, Model: ${this.model}`;
};
let laptop1 = new Laptop("Dell", "XPS 13");
let laptop2 = new Laptop("Apple", "MacBook Air");
console.log(laptop1.getDetails());
console.log(laptop2.getDetails());
```

```
Brand: Dell, Model: XPS 13
Brand: Apple, Model: MacBook Air
```

Example: Inheritance with Prototype

```
function Animal(name) {
  this.name = name;
}
Animal.prototype.eat = function() {
  console.log(this.name + " is eating.");
};
function Cat(name) {
  Animal.call(this, name);      // Inheriting properties
}
Cat.prototype = Object.create(Animal.prototype);      // Inheriting methods
Cat.prototype.constructor = Cat;
let myCat = new Cat("Whiskers");
myCat.eat();      // Output: Whiskers is eating.
```

JavaScript Static Methods

- Static methods वह methods होते हैं जो class के ऊपर call किये जाते हैं, ना की class के instances के ऊपर। मतलब, इन्हे class के instance की ज़रुरत नहीं होती। यह methods generally utility functions या helper functions के लिए use किये जाते हैं।
- Static methods instance properties को access या modify नहीं कर सकते क्यूंकि यह specific objects के लिए नहीं, बल्कि poori class के लिए होते हैं।
- इन्हे directly class पर call किया जाता है, instances पर नहीं।

Creating Static Methods: Static method बनाने के लिए static keyword का use किया जाता है class के अंदर।

```
class Calculator {
  static add(a, b) {
    return a + b;
  }
}
```

- यहाँ add एक static method है जो class Calculator के ऊपर call किया जा सकता है, ना की किसी instance के ऊपर।

Using Static Methods: Static methods को directly class के naam के साथ call किया जाता है, बिना instance बनाये।

```
console.log(Calculator.add(5, 3));      // Output: 8
```

- Calculator.add(5, 3) call से 5 और 3 को add किया गया और result 8 आया।

Examples of Static Methods

Utility Function Example:

```
class MathUtils {
  static square(x) {
    return x * x;
  }
}
console.log(MathUtils.square(4));      // Output: 16
```

- यहाँ square method class MathUtils के ऊपर call किया गया है और 4 का square 16 return हुआ।

Counter Example:

```
class Counter {
 static count = 0;

 static increment() {
  this.count += 1;
  return this.count;
 }
}

console.log(Counter.increment());    // Output: 1
console.log(Counter.increment());    // Output: 2
```

- Counter.increment() call से class के static property count को increment किया जाता है और incremented value return होती है।

Advantages of Static Methods

- **No Instance Needed:** Static methods को call करने के लिए किसी object instance की ज़रुरत नहीं होती।
- **Utility Functions:** यह methods general utility functions के लिए useful हैं, जो की किसी particular object state के बिना काम करती हैं।
- **Class-Level Operations:** इन्हे class-level operations के लिए use किया जाता है, जैसे configuration या general-purpose calculations.

JavaScript Modules

JavaScript modules एक ऐसे structure हैं जो code को अलग -अलग parts में divide करने का method देते हैं, ताकि हर part अपने काम के लिए specific हो। ये आपको reusable code लिखने, maintain करने, और organize करने में help करते हैं। Modules के ज़रिये आप अपने code को छोटे pieces में तोड़ कर अलग files में रख सकते हो, और जहाँ ज़रूरत हो वहां उन files को import कर के use कर सकते हो।

- JavaScript में modules का concept ES6 (ECMAScript 2015) के बाद standard बना है। इसमें **import** और **export** keywords का use होता है।

Exporting a Module: अगर आप किसी file या function को बाहर use करना चाहते हो, तो आपको उसे "export" करना पड़ेगा। दो तरीके के exports होते हैं : Named Exports और Default Exports.

Named Exports: Named exports के ज़रिये आप एक या एक से ज़्यादा चीज़ों को export कर सकते हो।

```
// mathUtils.js
export const add = (a, b) => a + b;
export const subtract = (a, b) => a - b;
```

Default Export: Default export का use तब होता है जब आप एक ही default चीज़ को export करना चाहते हो।

```
// calculator.js
export default function multiply(a, b) {
   return a * b;
}
```

Importing a Module: एक बार आप module को export कर देते हो, उसे किसी दूसरे file में import कर के use कर सकते हो।

Named Import: Named exports को import करने के लिए, आपको उनका exact नाम use करना पड़ता है जो export file में define किया गया हो।

```
// app.js
import { add, subtract } from './mathUtils.js';

console.log(add(5, 3));     // Output: 8
console.log(subtract(5, 3));     // Output: 2
```

Default Import: Default exports को import करते वक़्त आप कोई भी नाम दे सकते हो।

```
// app.js
import multiply from './calculator.js';

console.log(multiply(5, 3));     // Output: 15
```

Combining Named and Default Imports: आप एक ही file से named और default दोनों को import कर सकते हो।

```
// app.js
import multiply, { add, subtract } from './mathUtils.js';
console.log(multiply(5, 3));      // Output: 15
console.log(add(5, 3));     // Output: 8
```

Renaming Imports and Exports: कभी-कभी आपको named export या import को rename करने की ज़रूरत होती है। इसका syntax **as** keyword के साथ होता है।

Renaming Export

```
// mathUtils.js
const add = (a, b) => a + b;
const subtract = (a, b) => a - b;
export { add as addition, subtract as subtraction };
```

Renaming Import

```
// app.js
import { addition as add, subtraction as sub } from
'./mathUtils.js';
console.log(add(5, 3)); // Output: 8
console.log(sub(5, 3)); // Output: 2
```

Importing All Exports: अगर आपको किसी file के सारे exports को एक object के रूप में import करना हो, तो आप * का use कर सकते हैं।

Note: Module use करने के लिए HTML file में आपको Main File (जहाँ सारे imports lies करते हैं) src में देकर type में "module लिखना होता है।

```
// app.js
import * as MathUtils from './mathUtils.js';
console.log(MathUtils.add(5, 3));     // Output: 8
console.log(MathUtils.subtract(5, 3));   // Output: 2
```

Dynamic Imports JavaScript: में आपको modules को dynamically import करने का option भी मिलता है using import() function. इसका use तब होता है जब आपको modules को runtime पे load करना हो, या जब आपको performance को optimize करना हो।

```
// app.js
import('./mathUtils.js')
  .then(module => {
    console.log(module.add(5, 3));    // Output: 8
  });
```

Error Handling

JavaScript में error handling के लिए throw, try, catch, और finally का use होता है। यह mechanism आपको allow करता है की आप unexpected errors को handle कर सकें बिना पूरे program को crash किये। इससे आप specific errors को handle कर सकते हैं और program का flow control कर सकते हैं।

throw Statement:

- throw का use manually error generate करने के लिए किया जाता है।
- जब भी आप throw statement का use करते हैं, code execution वही stop हो जाता है और control catch block को pass हो जाता है (अगर defined है)।

Syntax:

```
throw "Error Message";
throw 101;
throw new Error("Custom Error");
```

Example:

```
function checkAge(age) {
  if (age < 18) {
    throw "Age must be 18 or above";
  } else {
    console.log("Access granted");
  }
}
checkAge(16);     // This will throw an error
```

try...catch Statement:

- try block में आप risky code रखते हैं जो error throw कर सकता है।
- अगर try block में कोई error आता है, तो control catch block को pass हो जाता है।
- catch block error को handle करता है और program को crash होने से बचाता है।

Syntax:

```
try {
   // Risky code
} catch (error) {
   // Error handling code
}
```

The Error Object

- JavaScript में जब कोई error throw होता है, तो एक Error object बनाया जाता है जो error के बारे में useful information contain करता है। इस object में error का **name** और **message** होता है जो error को identify और explain करता है।
- Error object को manually create करने के लिए new Error() constructor का use होता है। इसका use करके आप custom error messages भी generate कर सकते हैं।

Syntax: **let error = new Error(message);**

Example:

```
try {
   throw new Error("Something went wrong");
} catch (error) {
   console.log("Error Name: " + error.name);     // Output: Error
   console.log("Error Message: " + error.message);     // Output: Something went wrong
}
```

Example: **Validating User Input**

```
function validateAge(age) {
  if (isNaN(age)) {
   throw new Error("Age must be a number.");
  }
  if (age < 0 || age > 120) {
   throw new Error("Age must be between 0 and
120.");
  }
  return "Valid age: " + age;
}
try {
  let userAge = validateAge(25);
  console.log(userAge);
  userAge = validateAge(-5);    // This should trigger
an error
} catch (error) {
  console.log("An error occurred:", error.message);
}
```

finally Block:

- finally block हमेशा execute होता है, चाहे try block में error आये या नहीं।
- finally का use clean-up code के लिए होता है, जैसे की files को close करना या resources को free करना।

Syntax:

```
try {
   // Risky code
} catch (error) {
   // Error handling code
} finally {
   // Code that will always execute
}
```

Example:

```
try {
   let result = 10 / 0;      // Division by zero, may cause logical error
   throw new Error("divisibility will cause infinity");
} catch (error) {
   console.log("An error occurred: " + error.message);
} finally {
   console.log("This will always run, no matter what");
}
```

```
An error occurred: divisibility will cause infinity
This will always run, no matter what
```

Error Name Values: JavaScript में कुछ predefined error names होते हैं जो अलग -अलग types के errors को represent करते हैं। ये error names Error object के name property में stored होते हैं।

Common Error Name Values:

EvalError: जब eval() function में कोई error होता है, तब ये error throw होता है। Modern JavaScript में इसका use rare है।

```
try {
   eval("invalid code");
} catch (error) {
   console.log(error.name);      // Output: EvalError
}
```

RangeError: ये error तब throw होता है जब कोई value allowed range के बाहर होती है।

- Example: Array length negative हो या Number.toFixed() में allowed precision से ज़्यादा value दी गयी हो।

```
try {
   let num = 1;
   num.toPrecision(500); // Precision limit exceeded
} catch (error) {
   console.log(error.name);    // Output: RangeError
}
```

ReferenceError: जब कोई variable या function defined नहीं होता और use किया जाता है, तब ये error throw होता है।

```
try {
   console.log(undeclaredVariable);
} catch (error) {
   console.log(error.name);// Output: ReferenceError
}
```

SyntaxError: जब कोई code syntactically गलत होता है, तब ये error throw होता है। ये compile time error होता है।

```
try {
   eval("let x === 10;");      // Invalid syntax
} catch (error) {
   console.log(error.name);    // Output: SyntaxError
}
```

TypeError: जब किसी variable या value का type गलत होता है या किसी object property को access किया जाता है जो exist नहीं करती, तब ये error throw होता है।

```
try {
   let num = 42;
   num.toUpperCase(); // Invalid method for a number
} catch (error) {
   console.log(error.name);      // Output: TypeError
}
```

URIError: जब URI handling functions (जैसे decodeURI() या encodeURIComponent()) में कोई invalid URI component होता है, तब ये error throw होता है।

```
try {
   decodeURIComponent("%"); // Invalid URI component
} catch (error) {
   console.log(error.name);      // Output: URIError
}
```

Custom Error: JavaScript में अगर आपको अपना custom error create करना है, तो आप built-in Error class को extend करके अपने custom errors बना सकते हो। यह आपको एक specific type का error create करने का मौका देता है, जो की आपके program के logic के लिए ज़्यादा meaningful हो।

Steps for Creating Custom Error Types:

- आप Error class को extend करेंगे।
- constructor() method का use करेंगे अपने custom error message को handle करने के लिए।
- super() का use करेंगे Error class को call करने के लिए।
- Custom error का नाम set करेंगे।

Syntax:

```
class CustomError extends Error {
  constructor(message) {
    super(message);
    this.name = "CustomError";
  }
}
```

Example

```
class ValidationError extends Error {
  constructor(message) {
    super(message);
    this.name = "ValidationError";
  }
}
throw new ValidationError("Invalid input");
```

```
▶Uncaught ValidationError:
Invalid input
```

Program1: Form Validation

```
class ValidationError extends Error {
  constructor(message) {
    super(message);
    this.name = "ValidationError";
  }
}
function validateAge(age) {
  if (isNaN(age)) {
    throw new ValidationError("Age must be a number.");
  }
  if (age < 0 || age > 120) {
    throw new ValidationError("Age must be between 0 and 120.");
  }
  return "Valid age: " + age;
}
try {
  let userAge = validateAge(25);
  console.log(userAge);
  userAge = validateAge(-5);      // This should trigger an error
} catch (error) {
  if (error instanceof ValidationError) {
    console.log(`Validation error: ${error.message}`);
  } else {
    console.log(`Other error: ${error.message}`);
  }
}
```

```
Valid age: 25
Validation error: Age must be between 0 and 120.
```

JavaScript JSON

JSON (JavaScript Object Notation) एक lightweight data-interchange format है जो human-readable और machine-readable दोनों होता है। इसका use data को store और transfer करने के लिए किया जाता है, especially जब client और server के beech communication होती है, जैसे API responses में।

Key Points About JSON

Syntax Similar to JavaScript Objects: JSON का syntax JavaScript objects की तरह होता है, लेकिन JSON में सिर्फ data store होता है, कोई function नहीं।

Language-Independent: JSON JavaScript पे based ज़रूर है, लेकिन ये almost हर programming language में parse और generate किया जा सकता है, जैसे Python, Java, C#, etc.

Text-Based Format: JSON data को string के form में store करता है, जो easily web और server के बीच send किया जा सकता है।

Example of JSON

```
{
  "name": "Om Mishra",
  "age": 27,
  "isStudent": false,
  "courses": ["Math", "Physics", "Chemistry"],
  "address": {
    "street": "123 Main St",
    "city": "New York"
  }
}
```

JSON Syntax Rules: JSON में data key-value pairs के format में होता है, जहाँ :

- **Keys:** Always strings enclosed in double quotes (" ") होते हैं।
- **Values:** Strings, numbers, booleans (true/false), arrays, objects या null हो सकते हैं।

a. Arrays: JSON में values को arrays में store किया जा सकता है, और array में strings, numbers, objects, etc. हो सकते हैं।

```
{
  "fruits": ["Apple", "Banana", "Mango"]
}
```

b. Nested Objects: JSON objects को दूसरे objects के अंदर nest किया जा सकता है।

```
{
  "person": {
    "name": "Alice",
    "age": 25
  }
}
```

c. No Comments: JSON में comments allow नहीं होते। सिर्फ data ही store किया जा सकता है।

Working with JSON in JavaScript: JavaScript में JSON के साथ काम करने के लिए JSON.parse() और JSON.stringify() का use होता है।

1. Converting JSON String to JavaScript Object (JSON.parse()): Jab server से JSON data string के form में आता है, तब उसे JavaScript object में convert करने के लिए JSON.parse() का use होता है।

```
const jsonString = '{"name": "Om", "age": 27}';
const jsonObject = JSON.parse(jsonString);
console.log(jsonObject.name);      // Output: Om
console.log(jsonObject.age);      // Output: 27
```

2. Converting JavaScript Object to JSON String (JSON.stringify()): JavaScript object को JSON string में convert करने के लिए JSON.stringify() का use होता है, जो JSON data को easily store या server पे भेजने के लिए ready करता है।

```
const person = {
  name: "Om",
  age: 27,
  isStudent: true
};
```

```
{"name":"Om","age":27,"isStudent":true}
```

```
const jsonString = JSON.stringify(person);
console.log(jsonString);
```

Use Cases of JSON

Data Exchange Between Client and Server: JSON का सबसे common use client और server के बीच data exchange करना है। Server API response के form में JSON data भेजता है, और client side JavaScript इस data को process करता है।

Example: Fetch API से server से data retrieve करना।

```
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => console.log(data));
```

Configuration Files: JSON को configuration files के लिए भी use किया जाता है। जैसे package.json file Node.js projects में dependencies और project configuration को define करती है।

Storing Data Locally (LocalStorage): JSON को browser के localStorage या sessionStorage में string के form में store किया जा सकता है।

```
const person = { name: "Om", age: 27 };
localStorage.setItem("person", JSON.stringify(person));

const retrievedPerson = JSON.parse(localStorage.getItem("person"));
console.log(retrievedPerson);     // Output: { name: "Om", age: 27 }
```

APIs & Web Services: JSON mostly REST APIs के responses में use किया जाता है, जो web applications के लिए बहुत common है। जब भी किसी API से request करते हैं, आपको JSON data मिलता है।

Example: Error Handling in reading a JSON File

```
function parseJSON(jsonString) {
 try {
  let data = JSON.parse(jsonString);
  console.log("Parsed JSON:", data);
 } catch (error) {
  console.log("An error occurred:", error.message);
 }
}

let jsonString = '{"name": "Om", "age": 27}';
parseJSON(jsonString);

jsonString = '{"name": "Om", "age": 27'; // This should trigger an error
parseJSON(jsonString);
```

```
Parsed JSON: { name: 'Om', age: 27 }
An error occurred: Unexpected end of JSON input
```

Introduction to DOM (Document Object Model)

The DOM (Document Object Model) एक programming interface है जो web pages के structure को represent करता है। जब browser किसी HTML page को load करता है, तो वो page का एक DOM tree create करता है, जिसमे page के सभी elements को objects के रूप में represent किया जाता है। DOM के ज़रिये हम JavaScript का use कर के web page के content, structure, और style को dynamically change कर सकते हैं।

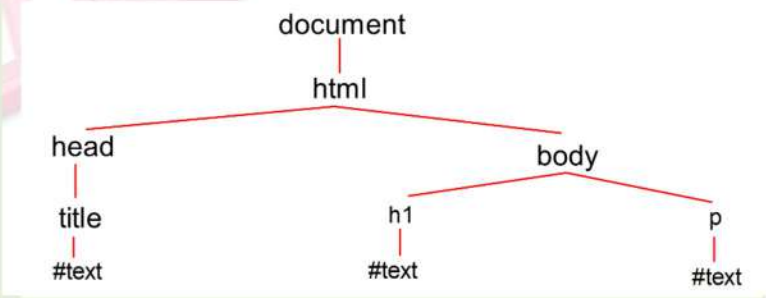
What is DOM?

- Document Object Model एक hierarchical representation है जो web page के elements को एक tree-like structure में organize करता है।
- DOM browser के अंदर dynamically create होता है जब HTML document load होती है। ये एक API (Application Programming Interface) है जो हमारे HTML और XML documents के साथ interact करने की सुविधा देती है।
- DOM को JavaScript, Python, या किसी भी programming language से manipulate किया जा सकता है जो browser को support करती है।

DOM Tree: DOM tree में हर element, attribute, और text content को एक node के रूप में represent किया जाता है। इस tree के top पे root node होती है, जो HTML document को represent करती है, और उसके अंदर हर element का अपना child-parent relationship होता है।

DOM Structure Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h1>Hello World</h1>
    <p>This is a paragraph.</p>
  </body>
</html>
```



- इस HTML document के लिए एक tree-like structure create होता है, जिसे हम DOM के रूप में देख सकते हैं। इसमें document root node होता है, जिसमे बाकी सब elements जैसे <html>, <body>, <h1>, और <p> child nodes के रूप में होते हैं।

Types of Nodes in DOM: DOM में अलग-अलग types के nodes होते हैं जो HTML document के different parts को represent करते हैं।

- **Element Node:** ये HTML elements जैसे , <h1>, <p> को represent करते हैं।
- **Text Node:** ये HTML elements के अंदर लिखा गया text content represent करते हैं।
- **Attribute Node:** ये elements के attributes (जैसे id, class, href) को represent करते हैं।
- **Comment Node:** ये HTML comments को represent करते हैं जो HTML में लिखे जाते हैं (<!-- comment -->).

Accessing DOM Elements: DOM में elements को access करने के लिए कुछ common methods होते हैं जो JavaScript में दिए गए हैं। इन methods का use करके हम specific elements को select कर सकते हैं और उनके properties को modify कर सकते हैं।

1. getElementById(): ये method किसी element को उसके unique id attribute से access करता है।

Syntax: document.getElementById("elementId");

```
<p id="myPara">Hello, World!</p>
<script>
 const element = document.getElementById("myPara");
 console.log(element);    // <p id="myPara">Hello, World!</p>
</script>
```

Hello, World!

2. getElementsByClassName(): ये method एक class name के basis पर matching elements का collection return करता है।

Syntax: document.getElementsByClassName("className");

```
<div class="box">Box 1</div>
<div class="box">Box 2</div>
<script>
 const elements = document.getElementsByClassName("box");
 console.log(elements);
</script>
```

Box 1
Box 2

```
▼ HTMLCollection(2) [div.box, div.box]
  ▶ 0: div.box
  ▶ 1: div.box
    length: 2
  ▶ [[Prototype]]: HTMLCollection
```

3. getElementsByTagName(): इस method का use किसी specific HTML tag के elements को access करने के लिए होता है।

Syntax: document.getElementsByTagName("tagName");

```
<h1>Heading 1</h1>
<h1>Heading 2</h1>
<script>
 const elements = document.getElementsByTagName("h1");
 console.log(elements);
</script>
```

Heading 1
Heading 2

```
▼ HTMLCollection(2) [h1, h1] i
  ▶ 0: h1
  ▶ 1: h1
    length: 2
  ▶ [[Prototype]]: HTMLCollection
```

4. querySelector(): ये method CSS selector का use करके first matching element को return करता है।

Syntax: document.querySelector("selector");

```
<p class="text">First Paragraph</p>
<p class="text">Second Paragraph</p>
<p id="text">Third Paragraph</p>
<script>
 const element = document.querySelector(".text");
 console.log(element);   //<p class="text">First Paragraph</p>
</script>
```

First Paragraph
Second Paragraph
Third Paragraph

5. querySelectorAll(): ये method CSS selector का use करके सभी matching elements का NodeList (array-like structure) return करता है।

Syntax: document.querySelectorAll("selector");

```
<p class="text">First Paragraph</p>
<p class="text">Second Paragraph</p>
<p id="text">Third Paragraph</p>
<script>
 const elements = document.querySelectorAll(".text");
 console.log(elements);
</script>
```

First Paragraph
Second Paragraph
Third Paragraph

```
NodeList(2) [p.text, p.text]
▶ 0: p.text
▶ 1: p.text
  length: 2
▶ [[Prototype]]: NodeList
```

onclick Event: onclick एक JavaScript event attribute है जो HTML elements पर apply किया जा सकता है। जब user किसी element को click करता है, तो उस event को trigger करने के लिए JavaScript function call होता है या कोई code execute होता है।

Syntax in HTML:

```
<element onclick="javascript_function()">
```

Syntax in JS:

```
element.onclick = function() {
   // Code to execute on click
};
```

Example of onclick in HTML:

```
<button onclick="alert('Button clicked!')">Click Me!</button>
```

- यहाँ जब user "Click Me!" button पर click करेगा, एक alert box display होगा जिसमे message होगा "Button clicked!".

Example of onclick in JavaScript:

```
<button id="myButton">Click Me!</button>
<script>
    document.getElementById("myButton").onclick = function() {
       alert('Button clicked!');
    };
</script>
```

- इस example में, JavaScript का use करके myButton element को onclick event assign किया गया है। जब user button को click करेगा, तो alert box show होगा।

Methods to Change HTML Content

1. innerHTML: innerHTML किसी element के HTML markup (tags के साथ) को access या modify करने के लिए use होता है। अगर element के अंदर कोई HTML tags हैं, तो innerHTML उन tags को भी include करता है।

Use case: जब आपको किसी element के अंदर HTML structure change करना हो या render करना हो (जैसे tags के साथ)।

Syntax: **element.innerHTML = "new content";**

```
<div id="myDiv">
  <p>Hello, <strong>World!</strong></p>
</div>
<script>
  let content = document.getElementById("myDiv").innerHTML;
  console.log(content);    // Output: "<p>Hello, <strong>World!</strong></p>"
</script>
```

- यहाँ innerHTML ने myDiv के अंदर का पूरा HTML markup (including <p> और tags) को return किया।

Key Points:

- HTML markup को modify कर सकता है।
- Dangerous हो सकता है अगर आपको user input handle करना हो, क्यूंकि Cross-Site Scripting (XSS) attacks का risk होता है।

Example:

```
<p id="myPara">Original Content</p>
<button onclick="changeContent()">Change Content</button>
<script>
 function changeContent() {
   document.getElementById("myPara").innerHTML = "New Content!";
 }
</script>
```

2. innerText: innerText किसी element के अंदर का visible text को access या modify करता है। ये text को display के according return करता है। अगर element के अंदर कोई CSS के through hidden content है, तो innerText उसको ignore करेगा।

Use case: जब आपको सिर्फ visible text को access करना हो, बिना किसी HTML tag के।

Syntax: **element.innerText = "new text";**

Example:

```
<div id="myDiv">
   <p>Hello, <strong>World!</strong></p>
</div>
```

```
<script>
   let content = document.getElementById("myDiv").innerText;
   console.log(content);     // Output: "Hello, World!"
</script>
```

- यहाँ innerText ने सिर्फ text content को return किया, बिना किसी HTML tag के।

Key Points:

- Only visible text को return करता है (CSS से hidden elements को ignore करता है)।
- Text को rendering के according handle करता है।
- Reflows (performance issue) का कारण बन सकता है, क्यूंकि ये browser के current layout को calculate करता है।

Example:

```
<h1 id="heading">Original Heading</h1>
<button onclick="changeText()">Change Text</button>
<script>
 function changeText() {
   document.getElementById("heading").innerText = "Updated Heading!";
 }
</script>
```

3. textContent: textContent किसी element के अंदर के पूरे text content को access या modify करता है, चाहे वो text hidden हो या visible.। ये all text nodes को return करता है, लेकिन कोई HTML tags को render नहीं करता।

Use case: जब आपको पूरे text content को access या modify करना हो, बिना किसी HTML के।

Syntax: **element.textContent = "new text";**

Example:

```
<div id="myDiv">
  <p>Hello, <strong>World!</strong></p>
</div>
```

```
<script>
  let content = document.getElementById("myDiv").textContent;
  console.log(content);      // Output: "Hello, World!"
</script>
```

- यहाँ textContent ने भी text return किया, लेकिन बिना किसी HTML markup को consider किये।

Key Points:

- All text, including hidden elements, को access करता है।
- Faster होता है as compared तो innerText क्यूंकि ये reflow/paint cycle को trigger नहीं करता।
- सिर्फ text nodes return करता है, कोई HTML नहीं।

Example:

```
<div id="content">
 <p>Hidden Text <span style="display:none;">(This is hidden)</span></p>
</div>
```

```
<script>
 document.getElementById("content").textContent = "Text changed!";
</script>
```

Creating and Manipulating DOM Elements: JavaScript का use करके हम नए DOM elements dynamically create कर सकते हैं और उन्हें existing elements के अंदर insert कर सकते हैं।

1. createElement(): ये method नया HTML element dynamically create करता है।

Syntax: **document.createElement("tagName");**

Example:

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

```
<script>
 const newElement = document.createElement("p");
 newElement.innerText = "This is a dynamically created paragraph!";
 document.getElementById("container").appendChild(newElement);
</script>
```

2. appendChild(): ये method किसी parent element के अंदर एक नए child element को add करता है।

Syntax: **parentElement.appendChild(newElement);**

Example:

```
<ul id="myList">
 <li>Item 1</li>
```

```
</ul>
<script>
 const newItem = document.createElement("li");
 newItem.innerText = "Item 2";
 document.getElementById("myList").appendChild(newItem);  // Add new item to list
</script>
```

3. removeChild(): ये method किसी parent element के अंदर से एक specific child element को remove करता है।

Syntax: parentElement.removeChild(childElement);

Example:

```
<ul id="myList">
 <li id="item1">Item 1</li>
 <li id="item2">Item 2</li>
</ul>
<script>
 const list = document.getElementById("myList");
 const item = document.getElementById("item2");
 list.removeChild(item);  // Removes "Item 2" from the list
</script>
```

4. replaceChild(): ये method किसी parent element के एक child element को replace करने के लिए use होती है।

Syntax: parentElement.replaceChild(newChild, oldChild);

Example:

```
<div id="myDiv">
 <p id="oldPara">Old Paragraph</p>
</div>
<script>
 const newPara = document.createElement("p");
 newPara.innerText = "New Paragraph!";
 const oldPara = document.getElementById("oldPara");
 document.getElementById("myDiv").replaceChild(newPara, oldPara);
</script>
```

Modifying Attributes of DOM Elements: DOM elements के attributes (जैसे class, id, src, href, etc.) को modify करना भी possible है।

1. setAttribute(): ये method किसी element का attribute set या modify करता है।

Syntax: element.setAttribute("attributeName", "value");

Example:

```
<img id="myImage" src="oldImage.jpg">
<script>
 const image = document.getElementById("myImage");
 image.setAttribute("src", "newImage.jpg");    // Changes the image source
</script>
```

2. Changing Attributes Directly: आप directly attribute को modify कर सकते हो जैसे element.attributeName = newValue.

Syntax: element.attributeName = "new value";

Example:

```
<a id="myLink" href="https://oldsite.com">Visit Old Site</a>
<button onclick="changeLink()">Change Link</button>
<script>
 function changeLink() {
   document.getElementById("myLink").href = "https://newsite.com";
 }
</script>
```

3. getAttribute(): ये method किसी element के attribute की value return करता है।

Syntax: element.getAttribute("attributeName");

Example:

```
<a href="https://example.com" id="myLink">Go to Example</a>
<script>
 const link = document.getElementById("myLink");
 console.log(link.getAttribute("href"));    // Prints the value of "href"
</script>
```

4. removeAttribute(): ये method किसी element के attribute को remove करता है।

Syntax: element.removeAttribute("attributeName");

Example:

```
<input id="myInput" type="text" placeholder="Enter text">
<script>
 const input = document.getElementById("myInput");
 input.removeAttribute("placeholder");   // Removes the placeholder attribute
</script>
```

Changing Styles of DOM Elements: JavaScript के ज़रिये हम HTML elements के styles (CSS) को dynamically change कर सकते हैं। DOM (Document Object Model) का use करके, हम किसी भी element का style update कर सकते हैं, चाहे वो color हो, size हो, या किसी element का position हो। ये चीज़ web pages को interactive और engaging बनाती है।

Changing Inline Styles: ये DOM element के style property का use करके उसके CSS properties को modify करता है।

Syntax: element.style.propertyName = "value";

Example:

```
<p id="myPara">This is a paragraph.</p>
<script>
 const para = document.getElementById("myPara");
 para.style.color = "red";     // Changes text color to red
 para.style.fontSize = "20px";     // Changes font size
</script>
```

Note: CSS properties को modify करते वक़्त आपको camelCase format का use करना पड़ेगा अगर वो properties में hyphen (-) आता है। जैसे : background-color को backgroundColor लिखना पड़ेगा।

Changing Multiple Styles: आप multiple CSS properties को एक ही function में change कर सकते हैं।

Example:

```
<div id="myDiv" style="width:150px; height:150px; background-color:lightgray;"></div>
<button onclick="updateStyles()">Update Styles</button>
<script>
 function updateStyles() {
   const div = document.getElementById("myDiv");
   div.style.backgroundColor = "green";
   div.style.border = "2px solid black";
   div.style.width = "300px";
 }
</script>
```

Adding or Removing CSS Class: अगर आपको एक ही बार में multiple styles apply करना हो, तो classList का use करके आप CSS classes को add, remove, या toggle कर सकते हो।

1. To add CSS Class: आप CSS class को JavaScript के classList.add() method का use करके किसी भी HTML element पर add कर सकते हो।

```
  .highlight {
    background-color: yellow;
  }
```

```
 <p id="myParagraph">It's a paragraph.</p>
 <button id="addClassButton">Add Highlight</button>
```

```
 <script>
  document.getElementById('addClassButton').addEventListener('click', function() {
    document.getElementById('myParagraph').classList.add('highlight');
  });
 </script>
```

2. To remove CSS Class: CSS class को remove करने के लिए हम classList.remove() का use करते हैं।

```
<button id="removeClassButton">Remove Highlight </button>
```

```
<script>
 document.getElementById('removeClassButton').addEventListener('click', function() {
   document.getElementById('myParagraph').classList.remove('highlight');
 });
</script>
```

3. To Toggle CSS Class: अगर आप class को add या remove करना चाहते हो based on its current state, तो classList.toggle() का use कर सकते हो। ये check करता है की class पहले से present है या नहीं, अगर है तो remove कर देता है, और अगर नहीं है तो add कर देता है।

```
<button id="toggleClassButton">Toggle Highlight</button>
```

```
<script>
 document.getElementById('toggleClassButton').addEventListener('click', function() {
   document.getElementById('myParagraph').classList.toggle('highlight');
```

```
 });
</script>
```

4. To check Class Exist or not: कभी कभी आपको यह check करना होता है की एक specific class किसी element में है या नहीं। इसके लिए classList.contains() का use होता है।

```
if (document.getElementById('myParagraph').classList.contains('highlight')) {
  console.log('Paragraph is highlighted!');
} else {
  console.log('Paragraph is not highlighted.');
}
```

- method true return करता है अगर class element में हो, और false return करता है अगर ना हो।

5. To Add/Remove Multiple Classes: आप एक ही time पर एक से ज़्यादा classes को add या remove कर सकते हो, multiple arguments pass करके।

```
document.getElementById('myParagraph').classList.add('highlight', 'bold');
document.getElementById('myParagraph').classList.remove('highlight', 'bold');
```

- इस example में, highlight और bold दोनों classes एक साथ add या remove की जा रही हैं।

Using CSS Variables: आप CSS custom properties (variables) को भी JavaScript के ज़रिये change कर सकते हो। CSS variables को getPropertyValue() और setProperty() से manipulate किया जा सकता है।

```
 <p id="chng">hello my name is om shankar mishra</p>
 <button onclick="changeColor()">Change Color</button>
```

hello my name is om shankar mishra

Change Color

```
:root{
   --bgmClr : magenta;
   --txtClr : green;
}
#chng{
   background-color: var(--bgmClr);
   color: var(--txtClr);
}
<script>
function changeColor(){
   document.getElementById("chng").style.setProperty("--bgmClr", "yellow");
   let rootStyles = getComputedStyle(document.documentElement);
   let x = rootStyles.getPropertyValue("--txtClr");
   console.log(x);   // green
}
</script>
```

hello my name is om shankar mishra

Change Color

- इस example में, button click होने पर CSS variable --bgmClr का value "yellow" हो जाता है, जो chng का background color change कर देता है।

DOM Events in JavaScript: JavaScript DOM events काफी important होते हैं web pages को interactive बनाने के लिए। Events वो actions होते हैं जो किसी user या browser के interaction से trigger होते हैं, जैसे mouse click, keyboard press, page load, etc. JavaScript के ज़रिये आप इन events को handle कर सकते हो और कुछ specific actions perform कर सकते हो।

Common DOM Events:

Mouse Events:

- **click:** जब element पे click होता है।
- **dblclick:** जब element पे double click होता है।
- **mouseover:** जब mouse pointer element के ऊपर आता है।
- **mouseout:** जब mouse pointer element से बहार जाता है।
- **mousedown:** जब user mouse का button किसी element पर press करता है।
- **mouseup:** जब mouse का button release होता है किसी element के ऊपर।
- **mousemove:** जब user mouse pointer को किसी element के अंदर move करता है (हिलाता है)।
- **mouseenter:** जब mouse pointer किसी element के अंदर enter करता है।
- **mouseleave:** जब mouse pointer किसी element के बाहर चला जाता है।
- **contextmenu:** जब user किसी element पर right-click करता है, जिससे context menu (right-click menu) खुलता है, यह event trigger होता है।

Keyboard Events:

- **keydown:** जब कोई key press होती है।
- **keyup:** जब pressed key release होती है।
- **keypress:** जब कोई key press और release होती है (अब यह event deprecated है, इसका use कम होता है)।

Form Events:

- **submit:** जब form submit होता है।
- **change:** जब input field का value change होता है।
- **focus:** जब input field पे focus आता है।
- **blur:** जब input field से focus चला जाता है।

Window Events:

- **load:** जब page fully load हो जाता है।
- **resize:** जब window का size change होता है।
- **scroll:** जब page scroll होता है।

Event Listeners in JavaScript: JavaScript में addEventListener() का use करके events को handle किया जा सकता है। ये function किसी भी DOM element के लिए event listen करता है और जब event trigger होता है, तब specific action perform करता है।

Syntax: **element.addEventListener("event", function);**

- **element:** वो DOM element जिसमे आप event listener लगाना चाहते हो।
- **event:** वो event जो आप listen करना चाहते हो (जैसे click, keydown, etc.)।
- **function:** वो function जो event trigger होने पर run होगा।

Note: Always write internal JavaScript code or if you are using external JS then use defer keyword for loading JS after HTML code get loaded.

Example: Button Click Event

```
<button id="myButton">Click Me!</button>

document.getElementById("myButton").addEventListener("click", function() {
  alert("Button clicked!");
```

```
});
```

- इस example में, जब user button पे click करेगा, alert box show होगा।

Event Object: जब भी कोई event trigger होता है, एक event object automatically pass होता है event handler function को। इस object में उस event की detailed information होती है, जैसे कोनसा button click हुआ, mouse की position क्या थी, कौनसी key press हुई, etc.

Example: Using Event Object in Click Event

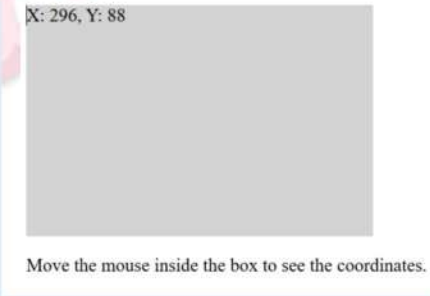
```
<button id="myButton">Click Me!</button>

document.getElementById("myButton").addEventListener("click", function(event) {
  console.log(event);     // All details of event object will be at console
});
```

Some useful properties of Event object :

- **event.type:** Event का type, जैसे click, keydown, etc.
- **event.target:** वो element जिसमे event trigger हुआ।
- **event.clientX, event.clientY:** Mouse pointer का position during event.

Ex: Track the mouse's position within a box and displays the coordinates inside it.

```
#myBox {
  width: 300px;
  height: 200px;
  background-color: lightgray;
  position: relative;
}
#coords {
  position: absolute;
  top: 0;
  left: 0;
}
```




```
<div id="myBox"> <span id="coords"></span> </div>
<p>Move the mouse inside the box to see the coordinates.</p>

document.getElementById("myBox").addEventListener("mousemove", function(event) {
  let x = event.clientX - this.offsetLeft;  // The offsetLeft property returns the left position (in pixels) relative to the parent.
  let y = event.clientY - this.offsetTop;  // The offsetTop property returns the top position (in pixels) relative to the parent.
  document.getElementById("coords").textContent = "X: " + x + ", Y: " + y;
});
```

Mouse Events: Mouse events तब trigger होते हैं जब user mouse से किसी element के ऊपर interact करता है।

```
<div id="box" style="width: 100px; height: 100px; background-color: coral;"></div>

document.getElementById("box").addEventListener("mouseover", function() {
  this.style.backgroundColor = "lightblue";    // background color will change during Mouseover
});
```

- जब user mouse pointer को box के ऊपर ले जाता है, उसका background color change हो जाता है।

Ex 1: **This example shows a dblclick (double-click) event that increases the size of a box when double-clicked.**

```
#myBox {
  width: 100px;
  height: 100px;
  background-color: lightblue;
}
```

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

```
<p>Double-click the box to enlarge it.</p>
```

```
document.getElementById("myBox").addEventListener("dblclick", function() {
  this.style.width = "200px";
  this.style.height = "200px";
});
```

Ex 2: **This example changes the box color when the mouse button is pressed down (mousedown) and releases it (mouseup).**

```
#myBox {
  width: 100px;
  height: 100px;
  background-color: coral;
}
```

Press and release the mouse button on the box to change its color.

```
<div id="myBox"></div>
<p>Press and release the mouse button on the box to change its color.</p>
```

```
document.getElementById("myBox").addEventListener("mousedown", function() {
  this.style.backgroundColor = "red";
});
document.getElementById("myBox").addEventListener("mouseup", function() {
  this.style.backgroundColor = "lightblue";
});
```

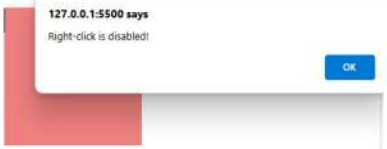
Ex 3: **This example disables the right-click context menu for a specific element.**

```
#myBox {
  width: 100px;
  height: 100px;
  background-color: lightcoral;
}
```

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

```
<p>Right-click is disabled on the box.</p>
```

```
document.getElementById("myBox").addEventListener("contextmenu", function(event) {
  event.preventDefault();
  alert("Right-click is disabled!");
});
```



Right-click is disabled on the box.

Keyboard Events: Keyboard events tab trigger होते हैं जब user कोई key press या release करता है।

Ex 1: **This example uses the keyup event to update a text input field in real-time. The value is displayed in a paragraph below as the user types.**

```
<input type="text" id="inputField" placeholder="Type something...">
<p>Text: <span id="displayText"></span></p>
```

```
document.getElementById("inputField").addEventListener("keyup", function() {
  let text = this.value;
  document.getElementById("displayText").textContent = text;
});
```

om shankar mishra

Text: om shankar mishra

Ex 2: **This example detects when the user presses a key combination, such as Ctrl + S, and displays a message.**

```
<p>Press "Ctrl + S" to trigger the event.</p>
```

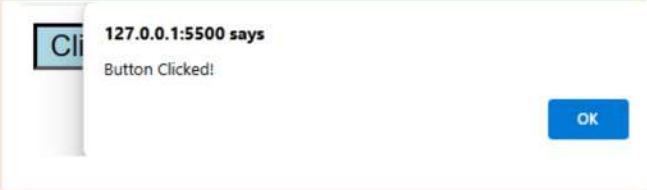
```
document.addEventListener("keydown", function(event) {
  if (event.ctrlKey && event.key === "s") {
```

```
    event.preventDefault();     // Prevent default browser behavior (saving page)
    alert("You pressed Ctrl + S!");
  }
});
```

Multiple Event Listeners on Same Element: एक DOM element पर एक से ज़्यादा event listeners add किये जा सकते हैं बिना किसी conflict के। हर event listener independently काम करता है ।

Example: Button with Multiple Events

```
<button id="myButton">Click Me!</button>

<script>
 var button = document.getElementById("myButton");
 button.addEventListener("click", function() {   // Click event
  alert("Button Clicked!");
 });
 button.addEventListener("mouseover", function() {    // Mouseover event
  button.style.backgroundColor = "lightblue";
 });
 button.addEventListener("mouseout", function() {    // Mouseout event
  button.style.backgroundColor = "";
 });
</script>
```

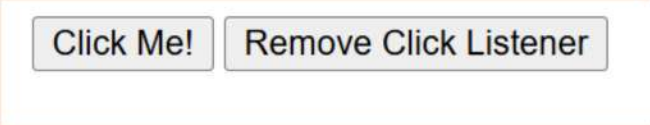


- इस example में, button पे click करने पर alert आता है, mouse को button के ऊपर ले जाने पर background color change होता है, और mouse button से बाहर ले जाने पर background color reset हो जाता है।

Removing Event Listeners: अगर आपको किसी event listener को remove करना हो, तो removeEventListener() का use किया जा सकता है। इसके लिए आपको same function reference देना होता है जो addEventListener() में दिया था।

Example: Removing Event Listener

```
<button id="myButton">Click Me!</button>
<button id="removeBtn">Remove Click Listener</button>

 var button = document.getElementById("myButton");
 function clickHandler() {
  alert("Button Clicked!");
 }
 button.addEventListener("click", clickHandler);     // Add Event Listener
 document.getElementById("removeBtn").addEventListener("click", function() {   // Remove Event Listener
  button.removeEventListener("click", clickHandler);
  alert("Click listener removed!");
 });
```



- इस example में, जब removeBtn button पे click होता है, तो click event listener को myButton से remove कर दिया जाता है।

Event Propagation: JavaScript में events propagate होते हैं in two phases:

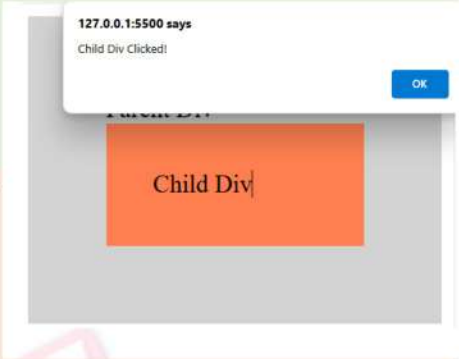
Capturing Phase: जब event outer elements से start होकर inner elements तक travel करता है।

Bubbling Phase: जब event inner element से trigger होकर outer elements तक bubble करता है।

- By default, events bubbling phase में propagate होते हैं, लेकिन addEventListener() का third parameter true set करके आप capturing phase को भी enable कर सकते हो।

Example: Bubbling Phase

```
<div id="parent" style="padding: 50px; background-color: lightgray;">
 Parent Div
 <div id="child" style="padding: 30px; background-color: coral;">
  Child Div
 </div>
</div>

 document.getElementById("parent").addEventListener("click", function() {
  alert("Parent Div Clicked!");
 });

 document.getElementById("child").addEventListener("click", function() {
  alert("Child Div Clicked!");
 });
```



- इस example में, जब आप child div पे click करोगे, तो पहले "Child Div Clicked!" alert आएगा, उसके बाद "Parent Div Clicked!" क्यूंकि event bubbling हो रहा है।

Example: Capturing Phase

```
<div id="parent" style="padding: 50px; background-color: lightgray;">
 Parent Div
 <div id="child" style="padding: 30px; background-color: coral;">
  Child Div
 </div>
</div>

 document.getElementById("parent").addEventListener("click", function() {
  alert("Parent Div Clicked!");
 }, true);     // Capturing phase

 document.getElementById("child").addEventListener("click", function() {
  alert("Child Div Clicked!");
 });
```

- अगर आप useCapture को true set करते हो, तो पहले "Parent Div Clicked!" alert आएगा और फिर "Child Div Clicked!" क्यूंकि अब event capturing phase में propagate हो रहा है।

Event Delegation: Event delegation एक technique है जिसमे आप एक common parent element को event listener assign करते हो, और inner elements में events को handle करते हो। ये useful होता है जब आप dynamically generated elements के events को handle करना चाहते हो।

Example: Event Delegation Using target

```
<ul id="myList">
 <li>Item 1</li>
```

```
 <li>Item 2</li>
 <li>Item 3</li>
</ul>
```

```
document.getElementById("myList").addEventListener("click", function(event) {
  if (event.target.tagName === "LI") {   // The value returned is always in uppercase letters
    alert(event.target.textContent + " clicked");
  }
});
```

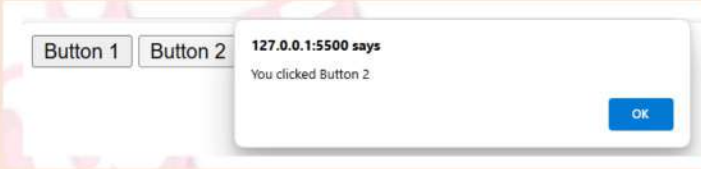
- इस example में, ul पर event listener set किया गया है और जब user किसी li element पे click करेगा, तो उसके text को alert कर दिया जायेगा।

Using this in Event Listeners: this keyword का use DOM events में करके आप current element को refer कर सकते हो जिसमे event trigger हुआ है।

Example: Using this

```
<button class="myButton">Button 1</button>
<button class="myButton">Button 2</button>
```

```
var buttons = document.getElementsByClassName("myButton");

for (var i = 0; i < buttons.length; i++) {
  buttons[i].addEventListener("click", function() {
    alert("You clicked " + this.textContent);
  });
}
```



- इस example में, जब आप किसी भी button को click करते हो, तो this.textContent उस button का text show करता है जो click हुआ था।

DOM Navigation: JavaScript के through आप DOM (Document Object Model) में elements के बीच easily navigate कर सकते हो। DOM navigation का use करके आप किसी भी HTML element के parent, child, या sibling elements को access कर सकते हो। ये functionality web pages के dynamic content को manipulate करने के लिए काफी useful होती है।

DOM Tree Structure: जब HTML page load होता है, browser उसको एक tree-like structure में convert करता है, जिसे DOM tree कहते हैं। DOM में हर HTML element एक node होती है, और elements के बीच hierarchical relationships होते हैं :

- **Parent:** जो एक element को contain करता है।
- **Child:** जो parent element के अंदर होता है।
- **Siblings:** जो same parent के under होते हैं।

DOM Navigation Properties: DOM navigation के लिए कुछ important properties हैं जिनके through आप elements के बीच traverse कर सकते हो।

- **parentNode:** Current element के direct parent को return करता है।
- **childNodes:** Current element के सारे child nodes का NodeList return करता है।
- **firstChild:** Current element के पहले child node को return करता है।

- **lastChild:** Current element के last child node को return करता है।
- **nextSibling:** Current element के बाद वाला sibling को return करता है।
- **previousSibling:** Current element के पहले वाला sibling को return करता है।

Example: Basic DOM Navigation

```
<div id="parentDiv">
 <p id="firstChild">This is the first child</p>
 <p id="secondChild">This is the second child</p>
</div>
```

```
var parent = document.getElementById("parentDiv");

// Accessing parent node
console.log(parent.parentNode);  // Document or parent element

// Accessing child nodes
console.log(parent.childNodes);  // NodeList of all child nodes including text nodes

// Accessing first and last child
console.log(parent.firstChild);  // First child (can be a text node)
console.log(parent.lastChild);   // Last child (can be a text node)

// Accessing siblings
var firstChild = document.getElementById("firstChild");
console.log(firstChild.nextSibling);  // Second child
```

- इस example में, parentDiv के different properties को access करके हमने parent node, child nodes, और sibling elements को console में print किया है।

parentElement vs parentNode:

- **parentElement:** ये सिर्फ element nodes को return करता है। अगर parent element null हो, तो ये null return करेगा।
- **parentNode:** ये किसी भी node को return करता है, चाहे वो element हो या text node.

```
<div id="parentDiv">
 <p>This is a paragraph.</p>
</div>
```

```
var paragraph = document.querySelector("p");
console.log(paragraph.parentElement);  // Returns <div>
console.log(paragraph.parentNode);     // Also returns <div>
```

- यहाँ दोनों properties same result return कर रही हैं क्यूंकि parent node एक element है।

Accessing Child Elements: अगर आपको सिर्फ child elements (excluding text nodes) access करने हैं, तो children property का use कर सकते हो।

Example: Accessing Child Elements

```
<div id="parentDiv">
 <p>This is the first paragraph.</p>
 <p>This is the second paragraph.</p>
</div>
```

```
var parent = document.getElementById("parentDiv");
var children = parent.children;  // Returns HTMLCollection of child elements

console.log(children[0]);  // First <p> element
console.log(children[1]);  // Second <p> element
```

- इस example में, children property सिर्फ element nodes को return करती है, और उनको index के through access किया जा सकता है।

firstElementChild and lastElementChild

- **firstElementChild:** पहला child element (excluding text nodes) को return करता है।
- **lastElementChild:** Last child element (excluding text nodes) को return करता है।

```
<div id="parentDiv">
  <p>This is the first paragraph.</p>
  <p>This is the second paragraph.</p>
</div>
```

```
var parent = document.getElementById("parentDiv");
console.log(parent.firstElementChild);  // Returns first <p> element
console.log(parent.lastElementChild);   // Returns last <p> element
```

Sibling Navigation: अगर आप किसी element के next या previous sibling element को access करना चाहते हो, तो nextElementSibling और previousElementSibling का use कर सकते हो। ये सिर्फ element nodes को consider करते हैं।

Example: Accessing Sibling Elements

```
<ul id="myList">
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

```
<script>
  var secondItem = document.querySelectorAll("li")[1];
  // Accessing next and previous sibling elements
  console.log(secondItem.previousElementSibling);  // Returns first <li> (Item 1)
  console.log(secondItem.nextElementSibling);      // Returns third <li> (Item 3)
</script>
```

- इस example में हमने li elements के siblings को navigate करके access किया ।

Traversing Through Child Nodes Using Loops: आप for loop या forEach loop का use करके child nodes के through traverse कर सकते हो।

```
<div id="parentDiv">
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
  <p>Paragraph 3</p>
</div>
```

```
var parent = document.getElementById("parentDiv");
var children = parent.children;
for (var i = 0; i < children.length; i++) {   // Using for loop
  console.log(children[i].textContent);  // Logs each child element's text
}
```

- यहाँ children HTMLCollection का use करके हर child element को traverse करके उनका text content console में print किया गया है।

JavaScript DOM Collections: JavaScript में DOM (Document Object Model) के elements को access और manipulate करने के लिए browser हमें कुछ collections provide करता है। इन collections का use करके आप आसानी से multiple DOM elements को access और manipulate कर सकते हो। DOM collections dynamically update होते हैं जब भी DOM structure में कोई changes होते हैं। DOM collections को primarily दो types में categorized किया गया है:

1. HTMLCollection: HTMLCollection एक live collection होता है जो केवल HTML elements को represent करता है। यह collection real-time में DOM changes को reflect करता है, मतलब जब भी DOM में कोई बदलाव होता है, HTMLCollection automatically update हो जाता है।

HTMLCollection कैसे मिलता है: getElementsByTagName(), getElementsByClassName() & children property

```
<div class="container">
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
</div>
```

```
var paragraphs = document.getElementsByTagName("p");

// HTMLCollection से elements को access करना
console.log(paragraphs[0].textContent);  // Output: "Paragraph 1"
console.log(paragraphs[1].textContent);  // Output: "Paragraph 2"

// HTMLCollection is live (real-time update)
var newPara = document.createElement("p");
newPara.textContent = "Paragraph 3";
document.querySelector('.container').appendChild(newPara);

console.log(paragraphs.length);  // Output: 3, collection updated automatically
```

- इस example में paragraphs एक HTMLCollection है जो सभी <p> elements को represent करता है। जैसे ही हमने DOM में एक नया paragraph element जोड़ा, collection automatically update हो गया।

Properties of HTMLCollection:

- **length:** Collection में elements की total count return करता है।

```
console.log(paragraphs.length);  // Output: 3
```

- **item(index):** Specific index पर element को return करता है।

```
console.log(paragraphs.item(0).textContent);  // Output: "Paragraph 1"
```

namedItem(name): ID या name attribute के basis पर element को return करता है।

```
<div id="container">
  <input type="text" name="username">
  <input type="text" name="email">
</div>
```

```
var inputs = document.getElementsByTagName("input");

var emailInput = inputs.namedItem("email");  // To access input element with name "email"
console.log(emailInput);  // Output: <input type="text" name="email">
```

- इस example में हमने inputs नामक HTMLCollection में से "email" नाम के attribute के साथ input element को access किया है।

2. DOM NodeList: JavaScript में NodeList एक ordered collection होता है जो DOM (Document Object Model) के nodes को represent करता है। NodeList एक ऐसा collection है जिसमें सिर्फ HTML elements ही नहीं, बल्कि text nodes और comment nodes भी हो सकते हैं। यह एक indexed list के रूप में होता है, यानी हर node एक specific position (index) पर होती है।

Types of NodeLists: NodeLists दो प्रकार के हो सकते हैं:

- **Static NodeList:** यह list static होती है, यानी DOM में changes होने के बाद update नहीं होती। जब NodeList को create किया जाता है, तब उसके elements fix हो जाते हैं। Example: querySelectorAll() method से बनी NodeList static होती है।
- **Live NodeList:** यह list live होती है, यानी DOM में होने वाले updates को dynamically reflect करती है। अगर DOM में कोई changes होते हैं, तो NodeList अपने आप update हो जाती है। Example: childNodes property से मिली NodeList live होती है।

Creating NodeList in JavaScript: JavaScript में NodeList बनाने के कुछ commonly used methods हैं:

- **querySelectorAll():** यह method एक static NodeList return करता है जो दिए गए CSS selector से match करने वाले सारे elements को represent करता है।
- **childNodes:** यह property एक live NodeList return करती है, जो किसी parent element के सभी child nodes (including text nodes और comment nodes) को represent करती है।

1. Static NodeList (Using querySelectorAll()): जब आप querySelectorAll() method का use करते हो, तो आपको एक static NodeList मिलती है। इसका मतलब अगर आप DOM में कोई नया element insert करते हो, तो यह list automatically update नहीं होती।

```
<div class="container">
 <p>Paragraph 1</p>
 <p>Paragraph 2</p>
</div>
```

```
var paragraphs = document.querySelectorAll("p");     // Static NodeList using querySelectorAll
// Add new Paragraph in DOM
var newPara = document.createElement("p");
newPara.textContent = "Paragraph 3";
document.querySelector('.container').appendChild(newPara);
console.log(paragraphs.length);    // Output: 2, new paragraph is not included
```

- इस example में, querySelectorAll() static NodeList return करता है, जो DOM में नया paragraph जोड़ने के बाद भी update नहीं होता।

2. Live NodeList (Using childNodes): जब आप childNodes property का use करते हो, तो आपको एक live NodeList मिलती है। इसका मतलब अगर आप DOM में कोई नया element insert या remove करते हो, तो यह list automatically update हो जाती है।

```
<div class="container">
 <p>Paragraph 1</p>
```

```
 <p>Paragraph 2</p>
</div>
```

```
// Live NodeList using childNodes
var container = document.querySelector(".container");
var children = container.childNodes;

// Add new Paragraph in DOM
var newPara = document.createElement("p");
newPara.textContent = "Paragraph 3";
container.appendChild(newPara);

console.log(children.length);  // Output: Updated count, new paragraph included
```

- इस example में, childNodes live NodeList return करता है, जो DOM में होने वाले changes के साथ dynamically update हो जाता है।

Accessing NodeList Elements: NodeList में elements को access करने के लिए आप index का use कर सकते हो, बिल्कुल array की तरह। हालांकि NodeList एक array नहीं होती, यह array-like structure होती है।

```
<div class="container">
 <p>Paragraph 1</p>
 <p>Paragraph 2</p>
</div>

var paragraphs = document.querySelectorAll("p");
```

```
// Accessing element through Index
console.log(paragraphs[0].textContent);
// Output: "Paragraph 1"
console.log(paragraphs[1].textContent);
// Output: "Paragraph 2"
```

Properties of NodeList

- **length:** यह property NodeList में total nodes की count return करती है।

```
var paragraphs = document.querySelectorAll("p");
console.log(paragraphs.length);    // Output: Total number of paragraphs
```

- **item(index)**: यह method NodeList के specific index पर node को return करता है।

```
var firstPara = paragraphs.item(0);
console.log(firstPara.textContent);  // Output: "Paragraph 1"
```

- **forEach():** यह method NodeList के elements को loop through करता है (सिर्फ static NodeList के लिए available)।

```
<div class="container">
 <p>Paragraph 1</p>
 <p>Paragraph 2</p>
 <p>Paragraph 3</p>
</div>
```

```
var paragraphs = document.querySelectorAll("p");

// Using forEach to iterate over NodeList
paragraphs.forEach(function(para) {
 console.log(para.textContent);
});
```

- इस example में हमने forEach() का use करके NodeList के प्रत्येक element का text content console में print किया।

Converting NodeList to Array: NodeList array-like behavior दिखाता है, लेकिन यह actual array नहीं होती। अगर आपको array के methods जैसे map(), filter(), या reduce() का use करना है, तो पहले NodeList को array में convert करना होगा।

```
var paragraphs = document.querySelectorAll("p");
var paragraphArray = Array.from(paragraphs); // NodeList को array में convert करना using Array.from()
var paragraphTexts = paragraphArray.map(function(p) { // अब array methods का use कर सकते हैं जैसे map()
  return p.textContent;
});
console.log(paragraphTexts);
```

NodeList Traversal: NodeList में elements को traverse करने के लिए आप index का use कर सकते हो, या forEach() method या for loop का use कर सकते हो।

```
<div class="container">
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
  <p>Paragraph 3</p>
</div>
```

```
var paragraphs = document.querySelectorAll("p");
// Traversing using for loop
for (var i = 0; i < paragraphs.length; i++) {
  console.log(paragraphs[i].textContent);
}
```

Difference Between HTMLCollection and NodeList

Feature	HTMLCollection	NodeList
Type of Nodes	केवल HTML elements	HTML elements, text nodes, comment nodes, etc.
Live or Static	Always live	Static (querySelectorAll()) या live (childNodes)
Methods	Array methods supported नहीं होते	forEach() static NodeList में supported है
Use Cases	Commonly used for getElementsByTagName(), getElementsByClassName()	Commonly used for querySelectorAll()

JavaScript Forms: Introduction and Manipulation:

Forms web development का एक important part है, जो users को data submit करने का मौका देता हैं, जैसे login details, search queries, etc. JavaScript के ज़रिये हम forms को dynamically manipulate कर सकते है, input validate कर सकते है, और form submission को efficiently handle कर सकते है। जब आपको form submit होने पर या किसी specific event (like submit, input, etc.) के होने पर custom logic execute करना होता है, तब addEventListener का use किया जाता है।

event.preventDefault(): जब भी कोई HTML element default action perform करता है (जैसे form submit होने पर page reload होता है, या link click होने पर दूसरे page पर जाता है), event.preventDefault() का use करके आप उस default behavior को रोक सकते हो।

Use Cases of event.preventDefault()

- Form Submit के Default Behavior को रोकना।
- Anchor Tag (<a>) का Default Navigation Stop करना।
- Button का Default Behavior Control करना।

Manipulating Form Values: JavaScript का use करके आप form field values को dynamically set या reset कर सकते है।

Get Value: `const inputValue = document.getElementById("inputID").value;`

Set Value: `document.getElementById("inputID").value = "New Value";`

To access Form and Input Fields

- **By ID:** id का use करके किसी form या input element को getElementById() से access कर सकते है।

```
const inputElement = document.getElementById("inputID");
```

- **By Name:** name attribute के ज़रिये form elements को access किया जा सकता है using document.forms या document.getElementsByName().

```
const formElement = document.forms["formName"];
```

```
const inputElement = formElement["inputName"];
```

Form Events: Forms में काफी सारे events होते है जो आप JavaScript के ज़रिये handle कर सकते है :

submit Event: जब form submit होता है तब trigger होता है।

```
<h3>Submit Event Example</h3>
<form id="myForm">
  <label for="name">Name: </label>
  <input type="text" id="name" placeholder="Enter your name" required>
  <button type="submit">Submit</button>
</form>
<p id="message"></p>
```

Submit Event Example

Name: omi Submit

Form submitted! Name: omi

```
const form = document.getElementById("myForm");
const message = document.getElementById("message");

form.addEventListener("submit", function (event) {
  event.preventDefault();   // Prevent form submission
  const name = document.getElementById("name").value;
  message.textContent = `Form submitted! Name: ${name}`;
});
```

input Event: जब भी input का value change होता है तब ये event trigger होता है।

```
<h3>Input Event Example</h3>
<label for="name">Name: </label>
<input type="text" id="name" placeholder="Type your name">
<p>Live Preview: <span id="preview"></span></p>
```

Input Event Example

Name: om shankar mishra

Live Preview: om shankar mishra

```
const input = document.getElementById("name");
const preview = document.getElementById("preview");

input.addEventListener("input", function() {
  preview.textContent = input.value;   // Show the live value of the input
});
```

change Event: जब input field का value change होता है और वो focus lose करता है तब trigger होता है।

```
<h3>Change Event Example</h3>
<label for="color">Select Color: </label>
<input type="color" id="color">
<p id="selectedColor">Selected Color: None</p>
```

Change Event Example

Select Color:

Selected Color: #930b0b

```
const input = document.getElementById("color");
const selectedColor = document.getElementById("selectedColor");

input.addEventListener("change", function() {
  selectedColor.textContent = `Selected Color: ${input.value}`;
});
```

focus aur blur Events: जब input field focus gain (focus) या lose (blur) करता है।

```
<h3>Focus Event Example</h3>
<label for="email">Email: </label>
<input type="email" id="email" placeholder="Enter your email">
<p id="message"></p>
```

Focus Event Example

Email: om

Input field is focused. Start typing...

```
const input = document.getElementById("email");
const message = document.getElementById("message");

input.addEventListener("focus", function() {
  message.textContent = "Input field is focused. Start typing...";
  message.style.color = "blue";
});
```

```
<h3>Blur Event Example</h3>
<label for="username">Username: </label>
<input type="text" id="username" placeholder="Enter your username">
<p id="status"></p>
```

Blur Event Example

Username: Enter your username

Input cannot be left blank!

```
const input = document.getElementById("username");
const status = document.getElementById("status");

input.addEventListener("blur", function() {
  if (input.value === "") {
    status.textContent = "Input cannot be left blank!";
    status.style.color = "red";
  } else {
    status.textContent = "Looks good!";
    status.style.color = "green";
  }
});
```

Example:

```
<form id="myForm">
  <label for="name">Name: </label>
  <input type="text" id="name" placeholder="Enter your name">
  <br> <span id="message"></span> <br>
  <button type="submit">Submit</button>
</form>
```

Name: om shankar mishra

Successfull

Submit

```
document.getElementById("myForm").addEventListener("submit", function () {
  event.preventDefault();
  const name = document.getElementById("name").value;
  const message = document.getElementById("message");
  function validateName(name) {
    const regex = /^[A-Za-z]+( [A-Za-z]+)*$/;
```

```
        return regex.test(name);
      }
      if (name === "") {
        message.innerText = "Name should not be blanked";
        message.style.color = "red";
      } else {
        valid = validateName(name);
        if (valid === true) {
          message.innerText = "Successfull";
          message.style.color = "green";
        }
        else {
          message.innerText = "Enter valid name";
          message.style.color = "red";
        }
      }
    })
```

Question: Dynamic Form with Validation

State Selection: Two states: Madhya Pradesh and Prayagraj.
Dynamic City Selection: Based on the selected state, the list of cities is updated.
Auto-Filled Pincode: Selecting a city auto-populates the corresponding pincode.
Phone Number Validation: Phone numbers are validated using regex:
Form Submission: If all inputs are valid, it shows a success message in green, else an error in red.

```
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
    }
    label, select, input {
      display: block;
      margin-bottom: 10px;
    }
    #message {
      margin-top: 10px;
      font-weight: bold;
    }
  <h2>Dynamic Form with Validation</h2>
  <form id="myForm">
    <!-- State Selection -->
    <label for="state">State:</label>
    <select id="state" required>
      <option value="">--Select State--</option>
      <option value="madhya_pradesh">Madhya Pradesh</option>
      <option value="prayagraj">Prayagraj</option>
    </select>
    <!-- City Selection -->
    <label for="city">City:</label>
```

Dynamic Form with Validation

State:

Madhya Pradesh

City:

Satna

Pincode:

485001

Phone Number:

07672111111

Submit

Form submitted successfully!

```
    <select id="city" required>
      <option value="">--Select City--</option>
    </select>
    <!-- Pincode -->
    <label for="pincode">Pincode:</label>
    <input type="text" id="pincode" placeholder="Pincode" readonly>
    <!-- Phone Number -->
    <label for="phone">Phone Number:</label>
    <input type="text" id="phone" placeholder="Enter phone number" required>
    <!-- Submit Button -->
    <button type="submit">Submit</button>
    <div id="message"></div>
  </form>
    const stateSelect = document.getElementById("state");
    const citySelect = document.getElementById("city");
    const pincodeInput = document.getElementById("pincode");
    const phoneInput = document.getElementById("phone");
    const messageDiv = document.getElementById("message");
    const data = {       // Data for states, cities, and pincodes
      "madhya_pradesh": {
        "Satna": { pincode: "485001", phoneCode: /^07672\d{6}$/ },
        "Rewa": { pincode: "486001", phoneCode: /^07662\d{6}$/ }
      },
      "prayagraj": {
        "Civil Lines": { pincode: "211001", phoneCode: /^05322\d{6}$/ },
        "Naini": { pincode: "211008", phoneCode: /^05321\d{6}$/ }
      }
    };
    // Populate city dropdown based on state selection
    stateSelect.addEventListener("change", function () {
      const selectedState = stateSelect.value;
      citySelect.innerHTML = '<option value="">--Select City--</option>';
      pincodeInput.value = "";
      phoneInput.value = "";
      if (selectedState) {
        const cities = Object.keys(data[selectedState]);
        cities.forEach(city => {
          const option = document.createElement("option");
          option.value = city;
          option.textContent = city;
          citySelect.appendChild(option);
        });
      }
    });
    citySelect.addEventListener("change", function () {       // Auto-fill pincode based on city selection
```

```
    const selectedState = stateSelect.value;
    const selectedCity = citySelect.value;
    if (selectedCity) {
      pincodeInput.value = data[selectedState][selectedCity].pincode;
    } else {
      pincodeInput.value = "";
    }
  });
  // Form submission validation
  document.getElementById("myForm").addEventListener("submit", function (event) {
    event.preventDefault();
    const selectedState = stateSelect.value;
    const selectedCity = citySelect.value;
    const phoneNumber = phoneInput.value;
    if (!selectedState || !selectedCity) {
      displayMessage("Please select state and city.", "red");
      return;
    }
    // Validate phone number using regex
    const phoneRegex = data[selectedState][selectedCity].phoneCode;
    if (!phoneRegex.test(phoneNumber)) {
      displayMessage(`Invalid phone number for ${selectedCity}.`, "red");
      return;
    }
    displayMessage("Form submitted successfully!", "green");
  });
  function displayMessage(message, color) {      // Function to display messages
    messageDiv.textContent = message;
    messageDiv.style.color = color;
  }
```

Window - The Browser Object Model (BOM)

JavaScript में BOM (Browser Object Model) एक ऐसा model होता है जो browser के window को represent करता है और उसके related objects और properties को manage करता है। BOM का main object window होता है, जो browser window के अंदर सब कुछ control करता है। इसमें browser का history, location, navigator, और document जैसी properties और methods शामिल हैं।
JavaScript में window object browser का global object होता है, और browser में जितनी भी global properties और functions होते हैं, वो सब window object का part होते हैं।

Key Properties and Methods of the window Object

window.document: ये property DOM (Document Object Model) का entry point है, जो web page का structure को represent करता है। इससे आप DOM के elements को manipulate कर सकते हो।

```
console.log(window.document.title);   // Page title को print करता है
```

window.innerHeight & window.innerWidth: ये properties browser window का height और width pixels में return करती हैं।

```
console.log(window.innerHeight);  // Browser window का height
console.log(window.innerWidth);   // Browser window का width
```

window.alert(): ये method एक alert dialog box को screen पर display करता है, जो user को एक message show करता है।

```
window.alert("This is an alert message!");  // Alert box show करता है
```

window.confirm(): ये method एक confirmation dialog box display करता है जिसमे "OK" और "Cancel" buttons होते हैं। ये user से confirmation के
लिए use होता है।

```
var result = window.confirm("Are you sure?");
console.log(result);  // True अगर user OK press करे, false अगर Cancel
```

window.prompt(): ये method एक prompt dialog box show करता है जिसमे user से input मांगता है।

```
var userInput = window.prompt("Enter your name:");
console.log(userInput);  // User के input को console में print करता है
```

window.open(): ये method new browser window या tab को open करता है किसी specific URL के साथ।

```
window.open("https://www.example.com", "_blank");  // नए tab में URL को open करता है
```

Key Differences Between DOM and BOM

- DOM (Document Object Model): Web page का content और structure को represent करता है। ये HTML और XML documents के elements को manipulate करता है।
- BOM (Browser Object Model): Browser के window को control करता है और URL, history, navigator, और window-related properties को handle करता है।

Window Screen in JavaScript: JavaScript में window.screen एक object होता है जो user के screen के बारे में information provide करता है। ये screen की dimensions (height, width), available space, color depth, और pixel depth जैसे properties को access करने के लिए use होता है। window.screen mainly screen के specifications को check करने में help करता है, जो responsive design और user के screen size के हिसाब से layout adjustments के लिए useful होता है।

Key Properties of window.screen

- **screen.width:** ये property user के screen का total width (pixels में) return करती है।

```
console.log(screen.width);  // Screen की total width को print करता है
```

- **screen.height:** ये property user के screen का total height (pixels में) return करती है।

```
console.log(screen.height);  // Screen की total height को print करता है
```

- **screen.availWidth:** ये property screen का available width return करती है, जो screen के width में browser window और OS interface को minus करके बचा हुआ space होता है।

```
console.log(screen.availWidth);  // Available width (excluding OS taskbars etc.)
```

- **screen.availHeight:** ये property screen का available height return करती है, जो taskbars और toolbars को छोड़कर बचा हुआ space होता है।

```
console.log(screen.availHeight);  // Available height (excluding taskbars etc.)
```

- **screen.colorDepth:** ये property screen का color depth (bits per pixel) return करती है, जो बताता है की screen कितने bits पैर pixel को display कर सकती है।

```
console.log(screen.colorDepth);  // Color depth in bits per pixel (e.g., 24, 32)
```

- **screen.pixelDepth:** ये property screen का pixel depth return करती है, जो color depth की तरह ही होता है, और बताता है कितने bits हर pixel के लिए use होते हैं।

```
console.log(screen.pixelDepth);  // Pixel depth in bits (usually same as color depth)
```

Use Cases of window.screen

- **Responsive Design:** Screen की width और height को check करके आप अपने web page का layout dynamically adjust कर सकते हो, ताकि page हर screen size पर अच्छे से दिखाई दे।

```
if (screen.width < 768) {          // Mobile screen के लिए CSS rules apply करो
  document.body.style.backgroundColor = "lightblue";
}
```

- **Dynamic Content Loading:** Screen के size के हिसाब से content को load कर सकते हो। छोटी screen पर lightweight content load करो और बड़ी screen पर high-resolution content.
- **Game Development:** अगर आप browser games develop कर रहे हो, तो screen की available dimensions को use करके canvas size को adjust कर सकते हो ताकि game screen के layout के हिसाब से run हो।
- **Optimizing Video or Image Quality:** Color depth को check करके आप images या videos का quality dynamically adjust कर सकते हो, ताकि user के screen के specifications के हिसाब से best performance मिले।

Difference Between window.innerWidth & screen.width

- **window.innerWidth:** ये browser window का actual viewport width (जो दिखाई दे रहा है) return करता है, including scrollbars और sidebars.
- **screen.width:** ये user के screen का total width return करता है, चाहे browser window maximize हो या नहीं।

```
console.log(window.innerWidth);    // Browser viewport width (actual visible area)
console.log(screen.width);     // Screen का total width (entire screen size)
```

Window Location: JavaScript में window.location एक ऐसा object होता है जो current web page के URL के बारे में details provide करता है और URL को manipulate करने के लिए methods देता है। इस object का use करके आप page को reload, redirect, या URL के different parts को access और modify कर सकते हो।

- window.location को short form में location भी लिखा जा सकता है, क्यूंकि ये window object का ही एक part है।

Key Properties of window.location

- **location.href:** ये property current page का complete URL return करती है। इस property को set करके आप page को किसी new URL पर redirect भी कर सकते हो।

```
console.log(location.href);  // Current URL print करता है
// Page को किसी दूसरे URL पर redirect करने के लिए
location.href = "https://www.facebook.com/";
```

- **location.protocol:** इस property से URL का protocol (जैसे HTTP, HTTPS) access किया जा सकता है।

```
console.log(location.protocol);  // 'https:' या 'http:' print करता है
```

- **location.hostname:** ये property current page के domain name को return करती है।

```
console.log(location.hostname);  // 'www.example.com' जैसा domain name return करता है
```

- **location.port:** अगर URL में कोई specific port number set किया गया है, तो ये property उस port number को return करती है। अगर कोई port नहीं दिया गया है, तो ये blank return करती है।

```
console.log(location.port);  // Example: '8080' अगर port set हो, वरना ''
```

- **location.pathname:** इस property से URL का path (domain के बाद का part) access किया जा सकता है।

```
console.log(location.pathname);  // Example: '/products/item1'
```

- **location.search:** ये property URL के query string को (question mark के बाद वाला part) return करती है।

```
console.log(location.search);  // Example: '?id=123&name=Om'
```

- **location.hash:** अगर URL में कोई fragment identifier (hash #) है, तो ये property उस part को return करती है।

```
console.log(location.hash);  // Example: '#section1'
```

- **location.origin:** ये property full URL का origin (protocol + hostname + port) return करती है।

```
console.log(location.origin);  // Example: 'https://www.example.com:8080'
```

Key Methods of window.location

- **location.assign():** ये method current document को specified URL पर redirect करता है। ये current page को replace नहीं करता, इसलिए browser के "back" button से वापस पीछे जा सकते हो।

```
location.assign("https://www.facebook.com/");  // Page को new URL पर redirect करेगा
```

- **location.replace():** ये method भी page को redirect करता है, लेकिन location.assign() के opposite, ये current page को history में replace कर देता है। इसलिए, आप "back" button से पीछे नहीं जा सकते।

```
location.replace("https://www.facebook.com/");  // Page replace होगा, history में back नहीं जा सकते
```

- **location.reload():** ये method current page को reload करता है। अगर आप forcefully server से reload करना चाहते हो, तो true parameter pass कर सकते हो।

```
location.reload();      // Page को reload करता है
```

Free Billangual Notes Available !

HTML | CSS | JavaScript | Angular | React | Hibernate

SpringBoot | JDBC | Git & GitHub | Data Analyst

www.linkedin.com/in/omi92229

Basics of HTML

What is HTML?

Definition: HTML का full form HyperText Markup Language है। यह एक standard language है जो web pages बनाने और design करने के लिए use होती है। यह web pages को structure provide करता है अलग -अलग elements और tags का उसे करके जो content और layout define करते हैं।

History: HTML को Tim Berners-Lee ने 1991 में बनाया था। यह समय के साथ evolve होता गया, और कई versions release हुए जो इसकी capabilities को enhance करते गए। अभी जो latest version है, वह HTML5 है, जो नए elements और features introduce करता है जिससे multimedia support, semantics, और applications और robust ho गए हैं।

Key Features:

- **Structure:** HTML headings, paragraphs, lists, links, और दूसरे content types का use करके web pages का structure define करता है।
- **Elements और Tags:** HTML tags का use करता है जो angle brackets में होते हैं (जैसे <tagname>). ज़्यादातर elements के पास opening tag और closing tag होता है (जैसे *** content ***), पर कुछ self-closing भी होते हैं (जैसे ***<img src="image.jpg" />***).
- **Attributes:** Tags के पास attributes होते हैं जो element के बारे में additional information देते हैं (जैसे ***<a href="https://example.com">link</a>***). Attributes opening tag के अंदर लिखे जाते हैं।
- **Hypertext:** HTML hyperlinks बनाने की सुविधा देता है, जो users को different web pages या resources के बीच navigate करने में मदद करता है।

Importance: HTML web development का foundation है। HTML के बिना web pages के पास basic

Data Science | SQL | Mongo DB | JAVA

Om Shankar Mishra

Data Science with R Programming

Lesson 1: Introduction to Business Analytics

Business analytics का मतलब है statistical analysis, data mining, और predictive modeling का use करके data को analyze और interpret करना, ताकि informed business decisions लिए जा सकें। यह modern business strategies का एक essential part है, जो valuable insights provide करता है, जिससे organizations market trends, customer preferences, और operational efficiencies को समझ सकते हैं।

Overview

Business analytics एक process है जो data को actionable insights में convert करता है, ताकि business outcomes को improve किया जा सके। यह various techniques और methodologies को encompass करता है जो historical data को समझने, future trends को predict करने, और data-driven decisions लेने में help करता है। इसका main goal है data को leverage करके business planning और strategic initiatives को drive करना।

Business Decisions and Analytics

Analytics, business decision-making में एक crucial role play करता है , क्यूंकि यह evidence-based insights provide करता है जो strategic planning, operational improvements, और competitive strategies को guide करता है। Analytics-supported business decisions से efficiency बढ़ सकती है, costs कम हो सकती हैं, customer satisfaction improve हो सकती है, और profitability बढ़ सकती है। Decision-making process में problem को identify करना, relevant data collect करना , data analyze करना, insights generate करना, और best course of action implement करना शामिल है।

Types of Business Analytics

```
location.reload(true);    // Forcefully server से reload करेगा
```

- **location.toString():** ये method location object का complete URL string में return करता है, जो location.href के बराबर होता है।

```
console.log(location.toString());  // URL string को print करता है
```

Example: Get Query Parameters from URL

```
let queryString = location.search;
console.log(queryString);  // Example: '?id=123&name=Om'
// Query parameters को parse करना (आप manual split भी कर सकते हो )
let urlParams = new URLSearchParams(queryString);
let id = urlParams.get('id');
let name = urlParams.get('name');
console.log("ID: " + id + ", Name: " + name);  // Example output: ID: 123, Name: Om
```

Use Cases of window.location

- **Page Redirection:** window.location.href का use करके आप users को किसी दूसरे page पर redirect कर सकते हो।
- **Getting URL Parameters:** Query strings और URL fragments को access करके आप dynamic content को handle कर सकते हो, जैसे product ID के हिसाब से page update करना।
- **Reloading the Page:** window.location.reload() का use page को refresh करने के लिए किया जा सकता है, जो real-time data को fetch करने के लिए useful होता है।
- **Modifying Browser History:** location.replace() का use करके आप history में changes ला सकते हो, जो SPA (Single Page Applications) या redirects में helpful होता है।

Window History: JavaScript में window.history object browser के history stack को represent करता है। ये object उन URLs को track करता है जो user ने current browser session के दौरान visit किये हैं। इस object का use करके आप user को पिछली या अगले page पर ले जा सकते हो, या history stack में किसी specific position पर move कर सकते हो।

Key Properties and Methods of window.history

- **history.length:** ये property browser के current session की history में URLs की total count को return करती है।

```
console.log(history.length);  // History में कितने pages हैं (count)
```

- **history.back():** ये method user को browser के previous page पर वापस ले जाता है, बिलकुल वैसे ही जैसे आप browser के "back" button को press करते हो।

```
history.back();  // Previous page पर वापस जाता है
```

- **history.forward():** ये method user को browser के अगले page पर ले जाता है, बिलकुल वैसे ही जैसे आप browser के "forward" button को press करते हो।

```
history.forward();  // Next page पर जाता है
```

- **history.go():** ये method history में किसी भी specific position पर move करने के लिए use होता है। Positive number pass करने पर अगले pages में जाता है, negative number पर पिछले pages में जाता है।

```
history.go(-1);   // Previous page (same as history.back())
history.go(1);    // Next page (same as history.forward())
history.go(0);    // Current page को reload करता है
```

- **history.pushState():** ये method current page का URL change किये बिना browser के history stack में new entry add करता है। Single Page Applications (SPA) में इस method का use होता है ताकि बिना page reload किये URL change कर सकें।

```
history.pushState({page: 1}, "title 1", "/new-url");
```

- **history.replaceState():** ये method history stack में current entry को replace करता है बिना page को reload किये।

```
history.replaceState({page: 2}, "title 2", "/another-url");
```

Use Cases of window.history

- **Navigation Control:** history.back(), history.forward() और history.go() का use करके आप user के लिए navigation controls implement कर सकते हो जैसे पिछले और अगले pages पर जाना।
- **Single Page Applications (SPA):** pushState() और replaceState() का use SPA में होता है जिसमे बिना page reload किये URL को update किया जाता है। ये आपके app को fast और dynamic बनाता है।
- **Custom Back/Forward Buttons:** अगर आप अपने web application में custom back या forward buttons implement कर रहे हो, तो history methods का use कर सकते हो ताकि browser के history के हिसाब से pages navigate हो सकें।
- **Dynamic Content Without Reload:** pushState() का use करके बिना reload किये आप अपने web page के content और URL को dynamically update कर सकते हो। इससे user experience improve होता है और navigation smooth होती है।

Difference Between pushState and replaceState

- **pushState():** New entry को history stack में add करता है। इसका मतलब है की आप "back" button से पिछले URL पर वापस जा सकते हो।
- **replaceState():** Current entry को history में replace करता है। इसका मतलब है की URL change होता है लेकिन "back" button से आप पिछले URL पर नहीं जा सकते।

Window Navigator: JavaScript में window.navigator object browser के environment और user के device के बारे में information provide करता है। इस object का use करके आप browser का type, version, user's operating system, online/offline status, language, और और भी कई चीज़ें जान सकते हो। window.navigator को simply navigator भी लिखा जा सकता है, क्यूंकि ये window object का ही एक part है।

Key Properties of window.navigator

a. navigator.appName: Browser का name return करता है। लेकिन modern browsers में यह mostly "Netscape" return करता है।

```
console.log(navigator.appName);  // Example: 'Netscape'
```

b. navigator.appVersion: Browser की version और OS के बारे में information return करता है।

```
console.log(navigator.appVersion);  // Example: '5.0 (Windows NT 10.0; Win64; x64)'
```

c. navigator.userAgent: Browser और OS के बारे में detailed string provide करता है जो mostly web analytics में use होती है।

```
console.log(navigator.userAgent);  // Example: 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)'
```

d. navigator.platform: Operating system के platform का name return करता है।

```
console.log(navigator.platform);  // Example: 'Win32' या 'MacIntel'
```

e. navigator.language: Browser के language setting को return करता है।

```
console.log(navigator.language);  // Example: 'en-US'
```

f. navigator.onLine: Boolean value return करता है जो बताता है की user online है या offline. true अगर user online है, false अगर offline है।

```
console.log(navigator.onLine);  // Example: true (if online) या false (if offline)
```

g. navigator.cookieEnabled: Boolean value return करता है जो बताता है की browser में cookies allowed हैं या नहीं।

```
console.log(navigator.cookieEnabled);  // Example: true (if cookies are enabled)
```

h. navigator.geolocation: User के current location को access करने के लिए Geolocation API provide करता है।

```
navigator.geolocation.getCurrentPosition(function(position) {
  console.log(position.coords.latitude, position.coords.longitude);  // Latitude और longitude
});
```

i. navigator.permissions: इस property का use करके आप user के permissions (जैसे location, camera) को check कर सकते हो की वो allow हैं या नहीं।

```
navigator.permissions.query({ name: 'geolocation' }).then(function(permissionStatus) {
  console.log(permissionStatus.state);  // Example: 'granted', 'denied', 'prompt'
});
```

j. navigator.javaEnabled(): Boolean value return करता है जो बताता है की browser में Java enabled है या नहीं।

```
console.log(navigator.javaEnabled());  // Example: true (if Java is enabled)
```

Use Cases of window.navigator

1. Browser and Platform Detection: आप navigator.appVersion और navigator.platform का use करके browser और OS detect कर सकते हो, जिससे आप platform-specific code या messages show कर सकते हो। For example, Windows और Mac के लिए different instructions provide कर सकते हो।

2. Online/Offline Status: navigator.onLine का use करके आप check कर सकते हो की user online है या offline. Offline web applications में ये बहुत useful होता है, जहा आपको user को proper feedback देना होता है की internet connection available नहीं है।

3. Geolocation Tracking: navigator.geolocation का use करके आप user के current location को access कर सकते हो, जो map services, weather apps, या location-based services में काम आता है।

4. User Agent Detection: navigator.userAgent का use करके आप user के browser के बारे में detailed information ले सकते हो, जो आपको specific browser के bugs या features के लिए content को adjust करने में help करता है।

5. Language Detection: navigator.language का use करके आप user के browser की language को detect कर सकते हो, और उसके hisaab से language-specific content provide कर सकते हो।

Example: Get User's Location and Display on Map

```
navigator.geolocation.getCurrentPosition(function(position) {
  let lat = position.coords.latitude;
  let lon = position.coords.longitude;
  let mapUrl = `https://www.google.com/maps?q=${lat},${lon}`;
  console.log("User's Location: " + mapUrl);  // Google Maps link with user's coordinates
});
```

Timing Events: JavaScript में timing events का use specific time interval के बाद code को execute करने के लिए होता है। ये events allow करते हैं की आप कुछ actions को delay कर के या periodic intervals पर perform कर सकें। JavaScript में timing events को handle करने के लिए दो main methods होते हैं :

1. setTimeout() Method: setTimeout() एक बार specific time delay के बाद function को execute करता है।

Syntax: setTimeout(function, delay);

- **function:** ये वो function है जो execute होगा।
- **delay:** Milliseconds में delay, जिसमे 1 second = 1000 milliseconds.

Example:

```
setTimeout(function() {
  alert("This message appears after 3 seconds");
}, 3000);  // 3000 milliseconds = 3 seconds
```

- इस example में, alert() function 3 seconds के बाद execute होगा ।

2. setInterval() Method: setInterval() specific interval पर बार -बार function को execute करता है, जब तक उसे manually stop ना किया जाए।

Syntax: setInterval(function, interval);

- **function:** ये वो function है जो repeat होगा।
- **interval:** Milliseconds में interval, हर बार कितने gap के बाद function execute होगा।

Example:

```
setInterval(function() {
  console.log("This message appears every 2 seconds");
}, 2000);  // 2000 milliseconds = 2 seconds
```

- इस example में, message हर 2 seconds में console में print होगा, जब तक clearInterval() use नहीं होता।

Stopping Timers

3. clearTimeout(): इसका use करके आप setTimeout() को cancel कर सकते हो।

```
let timeoutId = setTimeout(function() {
  console.log("This will not be shown");
}, 5000);

clearTimeout(timeoutId);  // Timeout को cancel कर दिया गया
```

- यहाँ clearTimeout() use किया गया है तो setTimeout() के execute होने से पहले ही उसे cancel कर दिया गया।

4. clearInterval(): इसका use करके आप setInterval() को stop कर सकते हो।

```
let intervalId = setInterval(function() {
  console.log("This message appears every 1 second");
}, 1000);

setTimeout(function() {
  clearInterval(intervalId);  // Interval को 5 seconds के बाद stop किया गया
}, 5000);
```

- इसमें message हर 1 second के gap में print हो रहा है, लेकिन 5 seconds के बाद interval stop हो जाता है।

Practical Use Cases of Timing Events

a. Delayed Execution: setTimeout() का use animations, notifications, या ads को delay करके दिखने के लिए किया जा सकता है।

b. Repeating Actions: setInterval() का use regular updates के लिए होता है, जैसे clocks, countdown timers, या periodic data fetch करना।

c. Slideshow or Carousel: आप setInterval() का use करके automatic image slideshow या content carousel create कर सकते हो जो हर कुछ seconds में change होता रहे।

Example 1: Using setTimeout() for Delayed Message

```
setTimeout(function() {
  alert("Welcome to our website!");
}, 2000);  // Message 2 seconds के बाद दिखाई देगा
```

Example 2: Using setInterval() for Repeating Action

```
let counter = 0;
let intervalId = setInterval(function() {
  counter++;
  console.log("Counter: " + counter);
  if (counter === 5) {
    clearInterval(intervalId);  // 5 बार के बाद interval stop हो जायेगा
  }
}, 1000);  // हर 1 second में counter increase होगा
```

Example 3: Creating a Countdown Timer

```
let countdown = 10;
let timerId = setInterval(function() {
  if (countdown > 0) {
    console.log("Time left: " + countdown);
    countdown--;
  } else {
    console.log("Time's up!");
    clearInterval(timerId);
  }
}, 1000);  // हर 1 second में countdown decrease होगा
```

- इस example में, countdown timer हर second के बाद एक number कम करता है, जब तक 0 ना हो जाए। जब countdown 0 हो जाता है, timer stop हो जाता है।

Difference Between setTimeout() and setInterval()

Method	Purpose	Execution
setTimeout()	Function को एक बार delay के बाद run करता है ।	एक बार execute होता है ।
setInterval()	Function को repeatedly हर fixed interval पर run करता है ।	हर interval के बाद repeat होता है ।

Cookies

Cookies एक small piece of data होती हैं जो browser में store की जाती हैं, ताकि website या web application user-specific information को remember कर सके। Cookies का use कर के आप user के browser में information store कर सकते हो जैसे preferences, login sessions, या shopping cart data. JavaScript के through आप cookies create, access, modify, और delete कर सकते हो।

What are Cookies?

Cookies एक simple text data होती हैं जो key-value pairs में store होती हैं। Browser के अंदर ही ये small data website की तरफ से store की जाती हैं और जब तक cookies expire नहीं होती या delete नहीं की जाती, वो persist करती हैं।

Key Attributes of Cookies:

- **Name:** Cookie का नाम (unique key).
- **Value:** Cookie की value.
- **Expiration:** Cookie कब expire होगी (optional)। अगर set नहीं किया गया, तो cookie तब तक valid रहती है जब तक browser session active है।
- **Path:** किस URL path पर cookie accessible होगी।
- **Domain:** किस domain के लिए cookie set की गयी है। Default domain current page का domain होता है।
- **Secure:** अगर true है, तो cookie सिर्फ HTTPS connection पर send की जाएगी।

- **SameSite:** Cookie को cross-site request में send होने से रोकने के लिए use होता है। Lax, Strict, और None values हो सकती हैं।

Creating a Cookie in JavaScript: Cookies को create करने के लिए आप document.cookie property का use करते हो।

Syntax: document.cookie = "name=value; expires=DATE; path=PATH";

- **name=value:** ये cookie का नाम और उसकी value define करता है।
- **expires=DATE:** Cookie कब expire होगी (optional). अगर expire date नहीं दी गयी तो cookie session के end तक valid होती है।
- **path=PATH:** जिस path के लिए cookie set की गयी है (optional). अगर नहीं दिया तो current page के लिए set होती है।

Example:

```
document.cookie = "username=OmMishra; expires=Fri, 31 Dec 2024 23:59:59 GMT; path=/";
```

- ये cookie username=OmMishra set करता है जो 31 Dec 2024 तक valid रहेगा, और ये cookie पूरी website (path /) के लिए accessible होगा।

Reading a Cookie: JavaScript में आप cookies को document.cookie property के through access कर सकते हो। ये एक string return करता है जिसमे सारी cookies semi-colon (;) से separated होती हैं।

```
console.log(document.cookie);          Output: "username=OmMishra; sessionID=12345"
```

- इस output में multiple cookies एक single string में ; से separated होती हैं।

अगर आप specific cookie को access करना चाहते हो, तो आप manually उस cookie को document.cookie string से parse कर सकते हो या custom function बना सकते हो।

Example: Get Specific Cookie Value

```
function getCookie(cookieName) {
  let name = cookieName + "=";
  let decodedCookie = decodeURIComponent(document.cookie);
  let cookieArray = decodedCookie.split(';');
  for(let i = 0; i < cookieArray.length; i++) {
    let cookie = cookieArray[i].trim();
    if (cookie.indexOf(name) == 0) {
      return cookie.substring(name.length, cookie.length);
    }
  }
  return "";
}
let username = getCookie("username");
console.log(username);  // Output: OmMishra
```

- ये function username cookie की value return करेगा, अगर वो exist करती है।

Updating a Cookie: Cookies को update करने का process create करने के जैसा ही होता है। आप simply same cookie name के साथ new value set करते हो, और पुरानी cookie overwrite हो जाती है।

```
document.cookie = "username=Omi; expires=Fri, 31 Dec 2024 23:59:59 GMT; path=/";
```

- इस example में, username cookie की value OmMishra से Omi में update हो गयी।

Deleting a Cookie: Cookie को delete करने के लिए आप उसकी expiration date को past में set करते हो।

```
document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/";
```

- इसमें username cookie delete हो गयी है by setting its expiration date to a past date.

Setting Cookie with Expiration Date: अगर आप चाहें की cookie session के बाद भी valid रहे, तो आप उसके लिए expiration date set कर सकते हैं।

```
let date = new Date();
date.setTime(date.getTime() + (7 * 24 * 60 * 60 * 1000));  // 7 के बाद expire होगी
let expires = "expires=" + date.toUTCString();
document.cookie = "username=OmMishra; " + expires + "; path=/";
```

- ये cookie 7 दिन तक valid रहेगी, और फिर automatically expire हो जाएगी।

Use Cases of Cookies

User Preferences: Cookies का use user के preferences store करने के लिए किया जा सकता है, जैसे language setting, theme preference (light/dark mode).

Session Management: Cookies का use user के session को track करने के लिए किया जा सकता है, जैसे login sessions या shopping cart data.

Personalized Experience: Cookies का use personalized user experience के लिए होता है, जैसे की user के visit history को track करके उन्हें relevant content show करना।

Tracking and Analytics: Cookies का use website के analytics या tracking tools में होता है, जैसे की Google Analytics user activity को track करता है।

Example: Set Cookie for a User

```
function setCookie(cookieName, cookieValue, days) {
  let date = new Date();
  date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));
  let expires = "expires=" + date.toUTCString();
  document.cookie = cookieName + "=" + cookieValue + ";" + expires + ";path=/";
}
setCookie("username", "JohnDoe", 7);  // 7 days के लिए cookie set
```

Example: Get Cookie Value

```
let user = getCookie("username");
if (user != "") {
  alert("Welcome back, " + user);
} else {
  alert("Welcome, new user!");
}
```

JavaScript Callbacks

JavaScript callbacks एक ऐसी technique है जिसमे एक function को दूसरे function के argument के रूप में pass किया जाता है, ताकि वो function उसे कभी भी call कर सके। Callbacks asynchronous operations (जैसे API calls, timers, file reading, etc.) को handle करने का एक common तरीका है।

What is a Callback Function?

Callback function एक function होता है जो किसी दूसरे function के अंदर argument के रूप में pass होता है और उससे specific event या condition par call किया जाता है। JavaScript में functions first-class citizens होते है, इसका मतलब है की functions को variables में store कर सकते हो, argument के रूप में pass कर सकते हो, या किसी function का return value बना सकते हो।

Real-World Example: सोचो तुम अपने दोस्त को फ़ोन करके कहते हो, "जब तुम movie tickets book कर लो, मुझे call कर देना।" यहाँ call कर देना का concept एक callback function जैसा है। तुम अपने काम (जैसे घर का काम) करते रहते हो, जब तक तुम्हारा दोस्त तुम्हे call नहीं कर लेता। जब उसने tickets book कर ली, तो उसने तुम्हे call back कर दिया।

```
function bookTickets(callback) {
  console.log("Ticket is Booked...");
  callback();
}

function callMeBack() {
  console.log("Friend is calling...");
}

bookTickets(callMeBack);
```

```
Ticket is Booked...
Friend is calling...
```

Example of a Simple Callback

```
function greeting(name) {
  console.log(`Hello, ${name}!`);
}

function processUserInput(callback) {
  const name = "Om";
  callback(name);  // यहाँ callback को call कर रहे है
}

processUserInput(greeting);  // 'greeting' function को callback के रूप में pass कर रहे है
```

```
Output:
Hello, Om!
```

- यहाँ processUserInput function एक callback function (greeting) को accept करता है और जब data (name) available होता है, तब उस callback को call करता है। greeting function को pass किया गया है as an argument और उसे बाद में call किया गया है।

Synchronous vs Asynchronous Callbacks

Synchronous Callbacks: Synchronous callbacks तुरंत call किये जाते है जब parent function execute हो रहा होता है।

```
function calculate(a, b, callback) {
  const result = a + b;
```

```
  callback(result);
}
function printResult(result) {
  console.log(`The result is: ${result}`);
}
calculate(5, 3, printResult);  // Output: The result is: 8
```

- यहाँ printResult को तुरंत call किया जाता है जब calculation complete होती है।

Common Use Cases of Callbacks:

Event Listeners DOM events जैसे clicks, keystrokes, mouse movements handle करने के लिए भी callbacks का use होता है।

```
document.getElementById("myButton").addEventListener("click", function() {
  alert("Button clicked!");
});
```

- यहाँ callback function को "click" event के response में execute किया जाता है।

Array Methods with Callbacks JavaScript arrays के map(), filter(), और forEach() methods में भी callbacks का use होता है।

```
const numbers = [1, 2, 3, 4, 5];
// Callback के साथ map method
const doubled = numbers.map(function(num) {
  return num * 2;
});
console.log(doubled);  // Output: [2, 4, 6, 8, 10]
```

Asynchronous Callback

Asynchronous JavaScript एक ऐसी programming technique है जो code को non-blocking way में execute करने की सुविधा देती है। JavaScript एक single-threaded language है, इसका मतलब है की ये एक time पर एक ही task perform करती है। अगर कोई long-running task, जैसे file read करना, API call करना, या timer set करना, synchronous तरीके से run हो, तो ये बाकी सभी operations को रोक देता है, इसलिए asynchronous programming का use किया जाता है।

Asynchronous JavaScript का main goal ये होता है की long-running tasks (जैसे network requests, timers, I/O operations) को background में perform किया जा सके बिना बाकी code को block किये।

Real-World Example: तुम pizza order करते हो। Pizza बनाने में time लगता है, तो तुम pizza के आने तक TV देखते हो या अपने काम करते हो। जब pizza आ जाता है तो delivery boy doorbell बजाता है, लेकिन तब तक तुम अपना काम कर रहे हो, बिना delivery के इंतज़ार किये बैठे।

```
console.log("Ordered Pizza...");
setTimeout(function() {
  console.log("Pizza delivered!");
}, 5000);  // Delivery will be after 5 sec
console.log("I watch TV until the pizza arrives...");
```

```
Ordered Pizza...
I watch TV until the pizza arrives...
Pizza delivered!
```

Key Concepts of Asynchronous JavaScript

Single-Threaded vs Multi-Threaded: JavaScript एक single-threaded language है, जो एक event loop का use करता है ताकि asynchronous tasks को manage किया जा सके।

Non-Blocking: Asynchronous code दूसरे code को block नहीं करता, इसलिए JavaScript event-driven और non-blocking होता है। Asynchronous tasks background में चलती हैं और जब complete होती हैं, तब result को handle करने के लिए callback या promises का use किया जाता है।

Examples of Asynchronous Tasks

- Network/API requests
- File I/O operations
- Timers (setTimeout, setInterval)
- Event listeners

Callbacks in Asynchronous JavaScript: Callbacks asynchronous tasks को handle करने का पहला basic तरीका है। जब asynchronous task complete होता है, तब एक callback function call किया जाता है जो result को handle करता है।

Example of Callback with setTimeout

Om Shankar Mishra

```
console.log("Start");
setTimeout(function() {
  console.log("This runs after 2 seconds");
}, 2000);
console.log("End");
```

```
Start
End
This runs after 2 seconds
```

- यहाँ setTimeout asynchronous है, जो 2 seconds बाद callback function को execute करता है, जबकि "Start" और "End" तुरंत print हो जाते हैं।

Callback Hell: Callback Hell तब होता है जब हम एक function के अंदर दूसरा callback function, उसके अंदर और एक callback function, और ऐसे ही multiple nested callbacks लिखते हैं। यह code को बहुत complex और messy बना देता है, जिसे समझना और maintain करना मुश्किल हो जाता है।

```
setTimeout(function(){
   console.log("Order Pizza");
   setTimeout(function(){
      console.log("order coke");
      setTimeout( () => { console.log("order ice-cream");
      setTimeout(function(){
         console.log("Party is arranged");
      },3000)
      },4000)
   },2000)
},1000)
```

```
Order Pizza
order coke
order ice-cream
Party is arranged
```

इस situation को commonly "Pyramid of Doom" या "Callback Hell" कहते हैं, क्यूंकि यह code एक pyramid या steps की तरह लगते हैं और बहुत बढ़ती हुई complexity create करते हैं।

```
// Asynchronous function to order pizza
function orderPizza(callback) {
  setTimeout(function() {
    console.log("Pizza has been ordered!");
```

```
  callback(); // Next task will happen when the pizza is ready
 }, 2000);
}
// Asynchronous function to order coke
function orderCoke(callback) {
 setTimeout(function() {
  console.log("Coke has been ordered!");
  callback();
 }, 1000);
}
// Asynchronous function to order ice cream
function orderIceCream(callback) {
 setTimeout(function() {
  console.log("Ice cream has been ordered!");
  callback();
 }, 3000);
}
```

```
// Asynchronous function to arrange the party
function arrangeParty(callback) {
 setTimeout(function() {
  console.log("Party has been arranged!");
  callback();
 }, 2000);
}
// This is where callback hell starts
orderPizza(function() {
 orderCoke(function() {
  orderIceCream(function() {
   arrangeParty(function() {
    console.log("All tasks are done, now enjoy!");
   });
  });
 });
});
```

```
Pizza has been ordered!
Coke has been ordered!
Ice cream has been ordered!
Party has been arranged!
All tasks are done, now enjoy!
```

- Callback Hell का solution Promises और async/await होता है।

JavaScript Promises: JavaScript Promises एक ऐसा object है जो asynchronous operations को handle करने का एक तरीका provide करता है। ये asynchronous tasks को ऐसे manage करता है की future में resolve या reject होने का promise देता है। Promises asynchronous operations के result को handle करने के लिए एक cleaner और more manageable approach है, जो हमारे code को callback hell से बचाता है।

What is a Promise?

Promise एक object है जो asynchronous operation के future result को represent करता है। जब हम कोई asynchronous task (जैसे API call, file reading, या timer) perform करते हैं, तो उसका result तुरंत available नहीं होता। Promise एक ऐसा structure provide करता है जो asynchronous operation के complete होने पर result को handle करता है।

Promise के 3 states होते हैं :

- **Pending:** जब promise अभी execute हो रहा है, और उसका result अभी तक available नहीं है।
- **Fulfilled (Resolved):** जब promise successfully complete हो जाता है, और result available होता है।
- **Rejected:** जब promise fail हो जाता है, और error आता है।

Creating a Promise: एक new promise create करने के लिए Promise constructor का use किया जाता है। इस constructor को एक function pass किया जाता है जो resolve और reject parameters को accept करता है।

```
const myPromise = new Promise((resolve, reject) => {
 let success = true; // Assuming task is successful
 if (success) {
```

```
  resolve("Task completed successfully!");
 } else {
  reject("Task failed.");
 }
});
```

- यहाँ myPromise एक Promise है जो resolve और reject functions को accept करता है। अगर task successfully complete होता है, तो resolve को call किया जाता है, वरना reject.

Using a Promise: Promise का result handle करने के लिए हम .then() और .catch() methods का use करते हैं।

- **.then():** जब promise successfully resolve हो जाता है, तो .then() को call किया जाता है।
- **.catch():** अगर promise reject हो जाता है (error या failure होता है), तो .catch() को call किया जाता है।

Example: Handling a Resolved and Rejected Promise

```
myPromise
 .then((message) => {
  console.log(message); // Output: Task completed successfully!
 })
 .catch((error) => {
  console.log(error);   // Output (if rejected): Task failed.
 });
```

Example:

```
let willWeMarry = new Promise(function(resolve, reject) {
 // The boy proposed
 let decisionTime = 2000; // 2 seconds thinking time
 setTimeout(function() {
  let girlSaidYes = true;  // Assume the girl says yes
  if (girlSaidYes) {
   resolve("The girl said yes, the wedding preparations begin!");
  } else {
   reject("The girl refused, the wedding will not happen!");
  }
 }, decisionTime);
});
// We are handling the promise
willWeMarry
 .then(function(result) {
  console.log(result);  // " The girl said yes, the wedding preparations begin!"
 })
 .catch(function(error) {
  console.log(error);  // "The girl refused, the wedding will not happen!"
 });
```

```
The girl said yes, the wedding preparations begin!
```

Promise Chaining: Promise chaining का use कर के हम multiple asynchronous tasks को sequentially perform कर सकते हैं। एक promise के .then() में जो return होता है, वो अगले .then() को pass होता है।

```
// Step 1: Pizza order as a promise
function orderPizza() {
  return new Promise(function(resolve) {
    setTimeout(function() {
      console.log("Order Pizza");
      resolve();
    }, 1000);  // 1 second delay
  });
}
// Step 2: Coke order as a promise
function orderCoke() {
  return new Promise(function(resolve) {
    setTimeout(function() {
      console.log("Order Coke");
      resolve();
    }, 2000);  // 2 seconds delay
  });
}
// Step 3: Ice Cream order as a promise
function orderIceCream() {
  return new Promise(function(resolve) {
    setTimeout(function() {
      console.log("Order Ice-Cream");
      resolve();
    }, 4000);  // 4 seconds delay
  });
}
// Step 4: Party arrangement as a promise
function arrangeParty() {
  return new Promise(function(resolve) {
    setTimeout(function() {
      console.log("Party is arranged");
      resolve();
    }, 3000);  // 3 seconds delay
  });
}
// Now nesting promises like callbacks
orderPizza().then(function() {
  return orderCoke().then(function() {
    return orderIceCream().then(function() {
      return arrangeParty().then(function() {
        console.log("All work is done, now enjoy!");
      });
    });
  });
});
```

```
Order Pizza
Order Coke
Order Ice-Cream
Party is arranged
All work is done, now enjoy!
```

Example:

```
new Promise((resolve, reject) => {
  setTimeout(() => resolve(5), 1000);
})
  .then((result) => {
    console.log(result); // Output: 5
    return result * 2;
  })
  .then((result) => {
    console.log(result); // Output: 10
    return result * 3;
  })
  .then((result) => {
    console.log(result); // Output: 30
  })
  .catch((error) => console.error(error));
```

- यहाँ पे हम promise chain कर रहे हैं जिसमे पहले result को process करते हैं, और फिर अगले .then() में pass करते हैं।

Promise Methods: JavaScript promises के साथ काम करने के लिए कुछ built-in methods provide करता है।

Promise.all(): ये method तभी resolve होता है जब सारे promises resolve हो जाएं। अगर कोई एक promise reject हो जाता है, तो ये पूरा Promise.all() reject हो जाता है।

Real-World Scenario: Planning a Party (Pizza, Drinks, and Cake): Imagine you're organizing a party, and you've ordered pizza, drinks, and cake from three different places. Instead of waiting for one to finish before starting the next, you're waiting for all of them to be ready simultaneously before the party starts.

```
// Promise for Pizza delivery
function orderPizza() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("Pizza delivered!");
    }, 3000);  // Pizza will deliver in  3 seconds
  });
}
// Promise for Drinks delivery
function orderDrinks() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("Drinks delivered!");
    }, 2000);  // Drinks will be delivered after 2
seconds
  });
}
// Promise for Cake delivery
function orderCake() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("Cake delivered!");
    }, 4000);  // Cake will be delivered after 4 seconds
  });
}
// Function to wait for all deliveries
function organizeParty() {
  console.log("Party ke liye pizza, drinks aur cake ka
order diya...");
  // Promise.all to wait for all orders together
  Promise.all([orderPizza(), orderDrinks(),
orderCake()]).then((results) => {
    console.log(results[0]);  // result of Pizza
    console.log(results[1]);  // result of Drinks
    console.log(results[2]);  // result of Cake
    console.log("Sab items aa gaye, ab party shuru
karte hain!");
  });
}
organizeParty();
```

```
Party ke liye pizza, drinks aur cake ka order diya...
Pizza delivered!
Drinks delivered!
Cake delivered!
Sab items aa gaye, ab party shuru karte hain!
```

- **Three Promises:** तुमने pizza, drinks, और cake का order दिया, और सभी अपने अपने time पे deliver होते हैं।
- **Promise.all():** यह method तीनो promises को एक साथ run करता है। जब सब promises resolve होते हैं, तब results एक array में मिलते हैं।
- **results[0], results[1], results[2]:** यह pizza, drinks और cake के result को show करता है जब सब ready हो जाते हैं।

Code Example with Failure:

```
function orderPizza() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("Pizza delivered!");
    }, 3000);
  });
}
// Promise for Drinks delivery
function orderDrinks() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("Drinks delivered!");
```

```
Party ke liye pizza, drinks aur cake ka order diya...
Party plan fail ho gaya: Cake delivery failed!
```

```
  }, 2000);
 });
}
// Promise for Cake delivery (this will fail)
function orderCake() {
 return new Promise((resolve, reject) => {
  setTimeout(() => {
   reject("Cake delivery failed!");
  }, 4000);
 });
}
// Function to wait for all deliveries
function organizeParty() {
 console.log("Party ke liye pizza, drinks aur cake ka order diya...");
 Promise.all([orderPizza(), orderDrinks(), orderCake()])
  .then((results) => {
   console.log(results[0]);
   console.log(results[1]);
   console.log(results[2]);
   console.log("Sab items aa gaye, ab party shuru karte hain!");
  })
  .catch((error) => {
   console.log("Party plan fail ho gaya: " + error);
  });
}
organizeParty();
```

Example:

```
const promise1 = Promise.resolve(5);
const promise2 = Promise.resolve(10);
const promise3 = Promise.resolve(15);

Promise.all([promise1, promise2, promise3])
 .then((values) => {
  console.log(values);  // Output: [5, 10, 15]
 })
 .catch((error) => {
  console.error(error);
 });
```

Promise.race(): इस method में जो promise सबसे पहले resolve या reject होता है, उसी का result return होता है।

```
const promise1 = new Promise((resolve) => setTimeout(resolve, 1000, "One"));
const promise2 = new Promise((resolve) => setTimeout(resolve, 500, "Two"));

Promise.race([promise1, promise2])
 .then((result) => {
  console.log(result);  // Output: "Two"
 });
```

Promise.allSettled(): ये method तभी complete होता है जब सारे promises settle हो जाएं, चाहे वो resolve हो या reject.

```
const promise1 = Promise.resolve("Success");
const promise2 = Promise.reject("Failure");

Promise.allSettled([promise1, promise2])
 .then((results) => {
  console.log(results);
  // Output: [{ status: 'fulfilled', value: 'Success' }, { status: 'rejected', reason: 'Failure' }]
 });
```

Promise.any(): ये method first resolved promise का result return करता है, और अगर सारे promises reject हो जाएं, तभी ये reject होता है।

```
const promise1 = Promise.reject("Error 1");
const promise2 = Promise.reject("Error 2");
const promise3 = Promise.resolve("Success");
```

```
Promise.any([promise1, promise2, promise3])
 .then((result) => {
  console.log(result);  // Output: "Success"
 })
 .catch((error) => {
  console.error(error); // तभी catch block run करेगा जब सारे promises fail हो।
 });
```

Event Loop in Asynchronous JavaScript: JavaScript का event loop asynchronous tasks को manage करता है। Event loop synchronous code को first execute करता है, और जब asynchronous tasks complete होते हैं (जैसे setTimeout या promises), उन्हें callback queue से लेकर main execution stack पर execute करता है।

- **Call Stack:** JavaScript code जो sequentially execute होता है, वो call stack में store होता है।
- **Callback Queue:** Asynchronous tasks जब complete हो जाते हैं, तो उनके callbacks को callback queue में add किया जाता है।
- **Event Loop:** Event loop continuously check करता है की call stack empty है या नहीं। अगर call stack empty होता है, तो event loop callback queue से tasks को call stack में push करता है।

```
console.log("Start");
setTimeout(() => {
 console.log("Timeout callback");
}, 0);
console.log("End");
```

```
Start
End
Timeout callback
```

- **Timeout callback:** Event loop के कारन setTimeout callback last में execute होता है, even though उसका timer 0 ms set किया गया था।

JavaScript Async/Await: Async/Await JavaScript में asynchronous code को handle करने का एक new and clean तरीका है। ये promises पर based होता है, लेकिन syntax को ज़्यादा readable और synchronous जैसा बनाता है। जब हम asynchronous operations perform करते हैं (जैसे API calls, timers, file I/O), तो promises का use करके result का wait करते हैं। Async/Await से ये और आसान हो जाता है।

What is async and await?

async keyword: जब किसी function के आगे async keyword लगाते हैं, तो वो function एक promise return करता है। इसका मतलब है की अगर function के अंदर कुछ return किया जाता है, तो वो promise automatically resolve हो जाता है।

await keyword: await एक asynchronous operation का wait करता है जब तक वो promise resolve या reject नहीं हो जाता। await को सिर्फ एक async function के अंदर ही use किया जा सकता है।

Basic Example

```
async function fetchData() {
 return "Data fetched!";
}
```

```
fetchData().then(result => console.log(result));  // Output: Data fetched!
```

- यहाँ fetchData function asynchronous है क्यूंकि हमने उसे async keyword से define किया है, और return होने वाली value को automatically एक resolved promise में wrap कर दिया गया।

```
// Function that will make coffee (returns a promise)
async function makeCoffee() {
  console.log("Maine coffee banani shuru ki...");
  return "Coffee tayar hai!";
}

// Calling the async function
makeCoffee().then(result => {
  console.log(result);  // This will work once the coffee is made
});
```

```
Maine coffee banani shuru ki...
Coffee tayar hai!
```

Example with await

```
async function fetchData() {
  const promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve("Data fetched!"), 2000);
  });

  const result = await promise;  // Waits for promise to resolve
  console.log(result);  // Output: Data fetched! (After 2 seconds)
}

fetchData();
```

- यहाँ await keyword का use promise के result का wait करने के लिए किया गया है। जब promise resolve होता है, तब ही अगला code execute होता है।

Scenario: Ordering Coffee and Donut: अब तुम coffee के साथ donut भी order कर रहे हो। Donut 2 seconds में ready होगा, और तुम्हे coffee और donut का wait करना पड़ेगा। Here we will use await to wait for both asynchronously.

```
// Function to make coffee
async function makeCoffee() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("Coffee ban gayi!");
    }, 3000);  // It will take 3 seconds for the coffee to be made
  });
}

// Function to make a donut
async function makeDonut() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("Donut tayar hai!");
    }, 2000);  // It will take 2 seconds for the donut to be made
  });
}

// Async function जो दोनों चीज़ों का wait करेगा
async function orderFood() {
  console.log("Coffee aur donut ka order diya...");

  let coffee = await makeCoffee();  // We will wait for the coffee
  console.log(coffee);

  let donut = await makeDonut();  // We will wait for the donut
  console.log(donut);

  console.log("Coffee aur donut dono ready, enjoy!");
}

orderFood();
```

```
Coffee aur donut ka order diya...
Coffee ban gayi!
Donut tayar hai!
Coffee aur donut dono ready, enjoy!
```

Why Use Async/Await?

- **More Readable:** Async/await promises के comparison में ज़्यादा readable और maintainable होता है, especially जब multiple asynchronous operations chain करने पड़ते हैं।
- **Avoids Callback Hell:** Callbacks को use करते हुए deeply nested structures (callback hell) बन जाती हैं, जो code को मुश्किल बना देती हैं। Async/await इससे बचाता है।
- **Error Handling:** async/await के साथ error handling करना simple होता है, try...catch block का use करके।

Scenario: Ordering Pizza, Coke, and Ice Cream (Three Steps): अब imagine करो की तुमने pizza, coke, और ice cream order किया है। तुम्हे sequentially पहले pizza का wait करना पड़ेगा, फिर coke, और फिर ice cream. इस example में, हम promises को async/await के साथ chain करेंगे।

```
// Pizza making promise
function makePizza() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("Pizza is ready!");
    }, 3000);  // Pizza will be ready in 3 seconds
  });
}
// Coke preparation promise
function makeCoke() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("Coke is ready!");
    }, 2000);  // Coke will be ready in 2 seconds
  });
}
// Ice cream making promise
function makeIceCream() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("Ice cream is ready!");
    }, 1000);  // Ice cream will be ready in 1 second
  });
}
// Async function that waits for everything
async function orderPartyItems() {
  console.log("Ordering pizza, coke, and ice cream...");
  let pizza = await makePizza();  // Wait for pizza
  console.log(pizza);
  let coke = await makeCoke();  // Then wait for coke
  console.log(coke);
  let iceCream = await makeIceCream();  // Finally wait for ice cream
  console.log(iceCream);
  console.log("All orders are done, now start the party!");
}
orderPartyItems();
```

```
Ordering pizza, coke, and ice cream...
Pizza is ready!
Coke is ready!
Ice cream is ready!
All orders are done, now start the party!
```

Example: Fetching Data from API: Let's use async/await to fetch data from an API using the fetch() method.

```
async function getPost() {
  const response = await fetch('https://catfact.ninja/fact');
  const data = await response.json();
  console.log(data);   // {fact: 'When a domestic cat goes after mice, about 1 pounce in 3 results in a catch.', length: 76}
}
getPost();
```

- यहाँ await का use fetch() function के result का wait करने के लिए किया गया है। जब तक fetch() का promise resolve नहीं हो जाता, तब तक अगला code execute नहीं होता।

Advantages of Async/Await

- **Cleaner Code:** Async/await code को promises से ज़्यादा clean और readable बनाता है, especially जब asynchronous operations को sequentially handle करना हो।
- **Synchronous Flow:** Async/await asynchronous operations को synchronous flow में लिखने की सुविधा देता है, जो code को समझने में आसान बनाता है।
- **Error Handling:** try...catch के साथ error handling ज़्यादा easy और readable होती है।

Limitations of Async/Await

- **Older Browser Support:** async/await ES2017 (ECMAScript 2017) में introduce किया गया था, इसलिए पुराने browsers इसे support नहीं करते हैं। However, modern browsers में इसकी full support है।
- **Sequential Execution:** जब आप multiple asynchronous operations को sequentially await करते हैं, तो ये थोड़ा time-consuming हो सकता है। इसलिए, parallel execution के लिए Promise.all() use करना better होता है।

DOM Animation using JavaScript:

JavaScript के ज़रिये हम web pages में animations create कर सकते हैं। DOM (Document Object Model) का use करके आप elements को move, resize, fade, या color change जैसे animations apply कर सकते हो। Animations web pages को ज़्यादा interactive और engaging बनाते हैं।

Simple DOM Animation Using setInterval(): JavaScript का setInterval() function एक popular method है animation create करने के लिए। ये function बार-बार एक fixed time interval के बाद code execute करता रहता है, जो आपको एक smooth animation create करने का मौका देता है।

Example: Moving a Box

```
<div id="box" style="width: 50px; height: 50px; background-color: coral; position: relative;"></div>
<button onclick="moveBox()">Move Box</button>
```

```
function moveBox() {
  let box = document.getElementById("box");
  let position = 0;
  let interval = setInterval(frame, 10);     // 10 milliseconds का interval
  function frame() {
    if (position >= 350) {
      clearInterval(interval);    // Animation stop करने के लिए
    } else {
      position++;
      box.style.left = position + "px";   // Box को left से move करना
    }
  }
}
```

- इस example में, button click करने पर box left side से right side move करता है। setInterval() के ज़रिये हर 10 milliseconds में box थोड़ा थोड़ा move करता रहता है।

Using requestAnimationFrame() for Smooth Animation: requestAnimationFrame() एक और efficient method है जो browser के rendering cycle के साथ sync होता है, जिससे animations और smooth लगती हैं।

```
<div id="box" style="width: 50px; height: 50px; background-color: coral; position: relative;"></div>
<button onclick="animateBox()">Animate Box</button>
```

```
function animateBox() {
  let box = document.getElementById("box");
  let position = 0;
  function frame() {
    if (position < 350) {
      position++;
      box.style.left = position + "px";
      requestAnimationFrame(frame);      // request for next frame
    }
  }
  requestAnimationFrame(frame);      // To start Animation
}
```

- requestAnimationFrame() का use करके animations और smooth और efficient बनते हैं क्यूंकि ये browser के redraw cycle के according चलता है।

Animating Multiple Properties Simultaneously: आप एक से ज़्यादा properties को एक साथ animate कर सकते हो, जैसे width, height, color, etc.

Example: Resize and Move Box

```
<div id="box" style="width: 50px; height: 50px; background-color: coral; position: relative;"></div>
<button onclick="startAnimation()">Animate Box</button>
```

```
function startAnimation() {
  let box = document.getElementById("box");
  let position = 0;
  let size = 50;
  function frame() {
    if (position >= 350 || size >= 200) {
      clearInterval(interval);
    } else {
      position++;
      size++;
      box.style.left = position + "px";
      box.style.width = size + "px";
      box.style.height = size + "px";
    }
  }
  let interval = setInterval(frame, 10);
}
```

- इस example में, box का size और position दोनों एक साथ animate होते हैं, जिससे एक combined effect मिलता है।

Fade In and Fade Out Animation: Opacity property का use करके आप elements को gradually show या hide कर सकते हो, जिससे fade in या fade out effect मिलता है।

Example: Fade In Effect

```
<div id="box" style="width: 100px; height: 100px; background-color: coral; opacity: 0;"></div>
<button onclick="fadeIn()">Fade In</button>
```

```
function fadeIn() {
  let box = document.getElementById("box");
  let opacity = 0;
  function frame() {
    if (opacity >= 1) {
      clearInterval(interval);
    } else {
      opacity += 0.01;
      box.style.opacity = opacity;
    }
  }
  let interval = setInterval(frame, 10);
}
```

- इस example में, box का opacity gradually increase होता है, जिससे वो slowly screen पर दिखने लगता है।

Example: Fade Out Effect

```
<div id="box" style="width: 100px; height: 100px; background-color: coral; opacity: 1;"></div>
<button onclick="fadeOut()">Fade Out</button>
```

```
function fadeOut() {
  let box = document.getElementById("box");
  let opacity = 1;
  function frame() {
    if (opacity <= 0) {
      clearInterval(interval);
    } else {
      opacity -= 0.01;
      box.style.opacity = opacity;
    }
  }
  let interval = setInterval(frame, 10);
}
```

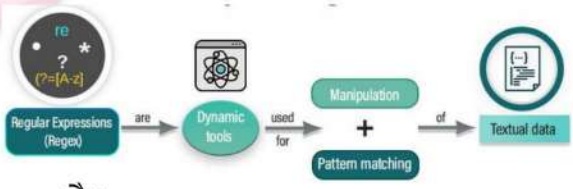
- Fade-out effect में box gradually invisible हो जाता है जब opacity 1 से 0 तक reduce होती है।

Regular Expression (regex)

Regular expression (regex) एक sequence of characters होता है जो एक search pattern को define करता है। जब आप किसी text में data search करते हैं, तो इस pattern को use करके आप specify कर सकते हैं की आप क्या search कर रहे हैं।

Syntax: /pattern/modifiers

- जहाँ **/** regex pattern को denote करता है और modifiers optional होते हैं जो search behavior को modify करते हैं।



Example: /omi92229/i

Explanation:

- **/omi92229/ :** यह omi92229 को match करने वाला pattern है।
- **i :** यह modifier है जो search को case-insensitive बना देता है।

Regular Expression Basics:

Pattern: Regex में आप जो pattern define करते हैं, वह वो string या characters को represent करता है जिसे आप search करना चाहते हैं।

Example: अगर pattern 'omi92229' है, तो यह exact omi92229 को match करेगा।

Modifiers: Modifiers (या flags) regex के search को modify करते हैं। यह specify करते हैं की search case-sensitive होगा या case-insensitive, global search होगा या not, etc.

Example: Modifier **i** का use case-insensitive search के लिए होता है।

Usage Example:

```
const text = "Visit Omi92229 for Linkedin  profile.";
const regex = /om/i;    // 'i' modifier for case-insensitive search
console.log(text.match(regex));    // Output: ["Om"]
```

Here:

- **Pattern:** omi92229 को exact match करना है।

- **Modifier i:** यह ensure करता है की search case-insensitive हो, यानी "OMi92229" या "omi92229 " दोनों को match करेगा।

String Methods

JavaScript strings के साथ regular expressions (regex) का use करते वक़्त कुछ useful string methods हैं जो text manipulation और pattern matching को efficient बनाते हैं। यहाँ पर कुछ common string methods और उनके regex के साथ use करने के तरीके दिए गए हैं :

1. .match() : String में regex pattern को match करता है और match के results को return करता है।

Syntax: string.match(regex)

```
const text = "Visit omi92229 for Linkedin profile.";
const regex = /om/;
const result = text.match(regex);
console.log(result);
```

```
[
  'om',
  index: 6,
  input: 'Visit omi92229 for Linkedin profile.',
  groups: undefined
]
```

2. .replace() : String में regex pattern को find करके उसे replace करता है।

Syntax: string.replace(regex, replacement)

```
const text = "om is a bad boy";
const regex = /bad/;
const res = text.replace(regex , "good");
console.log(res);   //output: om is a good boy
```

3. .search() : String में regex pattern को search करता है और match के start index को return करता है। अगर match नहीं होता, तो -1 return होता है।

Syntax: string.search(regex)

```
const text = "hello my name is om";
const regex = /me/;
const res = text.search(regex);
console.log(res);  //output: 11 (index of where me starts)
```

4. .split() : String को regex pattern के basis पर split करता है और एक array return करता है।

Syntax: string.split(regex)

```
const text = "one,two,three,four";
const regex = /,/;    // Split at comma
const result = text.split(regex);
console.log(result);      //  Output:  ["one",  "two",
"three", "four"]
```

```
const text = "hello my name is om";
const regex = /e/;      // Split at e
const res = text.split(regex);
console.log(res);       // Output: [ 'h', 'llo my nam',
' is om' ]
```

5. .trim() : String के start और end से whitespace को remove करता है। यह method regex pattern से directly related नहीं है, लेकिन common text cleaning के लिए useful है।

Syntax: string.trim()

```
const text = "   Hello World!   ";
const trimmedText = text.trim();
console.log(trimmedText);    // Output: "Hello World!"
```

6. .matchAll() : String में एक iterator के form में regex pattern के सारे matches को return करता है । **g** (global) flag के साथ use किया जाता है।

Syntax: **string.matchAll(regex)**

```
const text = "my name is om. my full name is om shankar mishra. my friends call me omi. my linkedin profile is omi92229";
const regex = /om/g;
const res = [...text.matchAll(regex)];
console.log(res);
```

```
▼ Array(4) i
  ▶0: ['om', index: 11, input: 'my name is om. my full name
  ▶1: ['om', index: 31, input: 'my name is om. my full name
  ▶2: ['om', index: 69, input: 'my name is om. my full name
  ▶3: ['om', index: 97, input: 'my name is om. my full name
   length: 4
  ▶[[Prototype]]: Array(0)
```

- The spread syntax ... is used to convert the iterator returned by text.matchAll(regex) into an array.

7. .replaceAll() : String में regex pattern के सारे occurrences को replace करता है एक specified replacement के साथ। **g** (global) flag के साथ use किया जाता है।

Syntax: **string.replaceAll(regex, replacement)**

```
const text = "apple banana apple";
const regex = /apple/g; // Matches all occurrences of "apple"
const result = text.replaceAll(regex, "orange");
console.log(result);   // Output: "orange banana orange"
```

How A Regex Engine Works Internally In Most Programming Languages

Regex engine काम कैसे करता है, यह समझने के लिए थोड़ा technical understanding चाहिए ।

1. Pattern Parsing: जब आप एक regular expression (regex) लिखते हो, तोह पहले engine उस pattern को parse करता है। मतलब, यह check करता है की आपने जो pattern लिखा है, उसमें क्या क्या instructions हैं, जैसे characters, quantifiers (*, +, ?), groups (()) और anchors (^, $), etc.

2. Finite Automaton Construction: Regex engines दो तरीके से काम करते हैं :

- NFA (Nondeterministic Finite Automaton) या DFA (Deterministic Finite Automaton).
- NFA ज़्यादातर programming languages में use होता है, जिसमें engine input string के साथ एक state machine बना लेता है। यह machine हर character को match करने के लिए multiple paths follow कर सकती है।

3. Matching Process: जब आप input string को regex के साथ match करते हो, engine left to right, character by character check करता है।

Backtracking: अगर कोई path fail हो जाता है, तो engine पीछे जाके दूसरे possible paths try करता है। यह process backtracking कहलाता है।

Example: अगर आपका pattern है a.*b और string है "acb", तो engine a से match करेगा, फिर .* के लिए c को match करेगा, और फिर b से match करेगा।

4. Greedy vs. Non-Greedy Matching:

- Greedy quantifiers (*, +, etc.) जितने ज़्यादा characters match कर सकते हैं, उतने करते हैं।
- Non-greedy quantifiers (*?, +?, etc.) minimum possible characters match करते हैं।

5. Returning Results: जब engine pattern को successfully match कर लेता है, तो यह results return करता है। यह result match हुई substring, groups, या capturing groups के रूप में हो सकता है।

6. Optimization: काफी advanced regex engines pattern matching को optimize करने के लिए internal tweaks करते हैं, जैसे shortcut matching, redundant checks avoid करना, etc.

Regular Expression Modifiers

Regular expression modifiers (या flags) regex patterns के behavior को modify करते हैं। यह modifiers आपके search को case-sensitive या case-insensitive बना सकते हैं, search ko global (सब occurrences को find करना) बना सकते हैं, और और भी बहुत कुछ। यहाँ कुछ common regex modifiers और उनका use case दिया गया है :

1. i (Case-Insensitive Search): यह modifier search को case-insensitive बना देता है। मतलब, upper case और lower case letters को same treat करता है।

Usage: /pattern/i

```
const text = "Hello World";
const regex = /hello/i;   // Case-insensitive search
console.log(text.match(regex));   // Output: ["Hello"]
```

- i flag के साथ, "hello" pattern "Hello" के साथ match हो जायेगा चाहे letter case different हो।

2. g (Global Search): यह modifier search को global बना देता है, यानि की string में pattern के सभी occurrences को match करता है।

Usage: /pattern/g

```
const text = "apple apple apple";
const regex = /apple/g;     // Global search
console.log(text.match(regex));     // Output: ["apple", "apple", "apple"]
```

- g flag के साथ, "apple" pattern string के हर occurrence को match करेगा।

3. m (Multiline Mode): यह modifier multiline strings में search को enable करता है, और ^ और $ anchors को line breaks के respect में interpret करता है।

Usage: /pattern/m

```
const text = "first line\nsecond line";
const regex = /^second/m;     // Multiline mode search
console.log(text.match(regex));   // Output: ["second"]
```

- m flag के साथ, ^ anchor "second" के line के start को denote करता है, not just the start of the whole string.

4. s (Dotall Mode): यह modifier dot (.) को newline characters के साथ भी match करने की permission देता है।

Usage: /pattern/s

```
const text = "hello\nworld";
const regex = /hello.world/s;     // Dotall mode
console.log(text.match(regex));     // Output: ["hello\nworld"]
```

- s flag के साथ, . dot newline characters को भी match करता है, इसलिए "hello\nworld" pattern के साथ match होता है।

5. x (Extended Mode): यह modifier regex pattern में whitespace और comments को allow करता है, जो regex को more readable बना सकता है। यह JavaScript में default नहीं होता।

Usage: /pattern/x

Example: Extended mode is not supported in JavaScript directly. This is a general example:

```
const regex = / \d{3} \- \d{2} \- \d{4} /x;      // Spaces and comments allowed
```

- x flag के साथ, आप regex में extra spaces और comments use कर सकते हैं। JavaScript में direct support नहीं है, लेकिन कुछ regex engines में available है।

6. u (Unicode Mode): यह modifier Unicode characters के support को enable करता है और \u और \p syntax को properly handle करता है।

Usage: /pattern/u

```
const text = "hello 🖐";
const regex = /\p{Emoji}/u;      // Unicode mode for emojis
console.log(text.match(regex));      // Output: [" 🖐"]
```

- u flag के साथ, regex Unicode characters को properly handle करता है, जैसे emojis.

7. d (Duplicate Mode): यह modifier regex engine के duplicate match handling को enable करता है। JavaScript में directly support नहीं है।

Usage: /pattern/d

Example: Duplicate mode is not supported in JavaScript directly. This is a general example:

```
const regex = /pattern/d;
```

Difference Between A Character And A String.

Character (Char):

- Character यानी एक अक्षर (alphabet/digit)।
- यह sirf एक ही symbol या letter को represent करता है, जैसे 'A', 'b', '1', '!', etc.
- Programming में character को single quotes (' ') में लिखा जाता है। **Example:** 'A', '1', etc.
- Size के मामले में, एक character 1 byte occupy करता है (ASCII characters के case में)।

String:

- String multiple characters का sequence होता है, यानी एक शब्द या sentence जिसमें कई अक्षर होते हैं।
- String को double quotes (" ") में लिखा जाता है। **Example:** "Hello", "1234", "omi92229".
- String का size उसमें कितने characters हैं उसपे depend करता है। Example: "Hello" एक 5-character की string है।

How To Match Simple Specific Characters.

अगर आपको specific characters को match करना है using regex, तो यह काफी simple होता है। चलिए कुछ basic examples के साथ समझते हैं।

1. Single Character Match: अगर आपको एक specific character match करना है, बस उस character को directly regex में लिख दीजिये।

```
const str = "apple";
const regex = /a/;      //patern: "a"
console.log(str.match(regex));      // Output: ["a"]
```

- यह a को match करेगा क्यूंकि string में a present है।

2. Exact Match: अगर आपको पूरा word या sequence match करना है, आप उस sequence को as it is लिख सकते हो।

```
const str = "I have a cat";
const regex = /cat/;
console.log(str.match(regex));     // Output: ["cat"]
```

- यह pattern "cat" को match करेगा क्यूंकि string में ये पूरा sequence present है।

Character Classes | Groups

How To Match Different Characters In A Single Position - Character Sets.

अगर आपको different characters को एक ही position पर match करना है, तो उसके लिए आप character sets का use कर सकते हो। Character sets को square brackets [] के अंदर define किया जाता है। यह bracket के अंदर दिए गए किसी भी character को match करते हैं।

Example:

- **Character Set:** [abc]
- **Explanation:** यह set a, b, या c में से किसी भी एक character को match करेगा (और अगर g modifier है तो सभी characters को) ।

चलिए इस concept को कुछ JavaScript examples के साथ समझते हैं :

Single Position, Multiple Matches: अगर आपको किसी string में a, b, या c को match करना है :

```
const str = "bat";
const regex = /[abc]/;
console.log(str.match(regex));     // [ 'b', index: 0, input: 'bat', groups: undefined ]
```

- इस example में, regex pattern [abc] string "bat" में से b और a को match करेगा क्यूंकि यह दोनों character set में include हैं।

```
let text = "hello my name is om";
let regex = /[om]/g;
let res = text.match(regex);
console.log(res);        //Output: [ 'o', 'm', 'm', 'o', 'm' ]
```

How To Define A Character Range To Match Different Characters In A Single Position - Character Ranges.

Character ranges का use आपको regular expressions में एक single position पर अलग - अलग characters को match करने के लिए करने देता है। Character range define करने के लिए square brackets [ ] के अंदर hyphen - का use होता है, जो एक range के first और last characters को specify करता है। इस range के बीच में आने वाले सारे characters match होते हैं।

Syntax:

- **[a-z]:** Lowercase letters a से z तक।
- **[A-Z]:** Uppercase letters A से Z तक।
- **[0-9]:** Digits 0 से 9 तक।
- **[a-zA-Z]:** सारे lowercase और uppercase letters.

Examples:

```
const regex = /[a-z]/;
console.log("a".match(regex));        // Output: ["a"]
```

```
console.log("z".match(regex));        // Output: ["z"]
console.log("G".match(regex));        // Output: null (because it's uppercase)
```

- [a-z] range सिर्फ lowercase letters को match करेगा, यानी "a" से "z" तक।

```
const regex = /[A-Z]/;
console.log("A".match(regex));        // Output: ["A"]
console.log("Z".match(regex));        // Output: ["Z"]
console.log("g".match(regex));        // Output: null (because it's lowercase)
```

- [A-Z] range uppercase letters को match करेगा, यानी "A" से "Z" तक।

```
const regex = /[0-9]/;
console.log("5".match(regex));        // Output: ["5"]
console.log("0".match(regex));        // Output: ["0"]
console.log("a".match(regex));        // Output: null (because it's a letter)
```

- [0-9] range में आपके सारे digits match होंगे।

```
const regex = /[a-zA-Z]/;
console.log("a".match(regex));        // Output: ["a"]
console.log("Z".match(regex));        // Output: ["Z"]
console.log("5".match(regex));        // Output: null (because it's a digit)
```

- [a-zA-Z] range में lowercase और uppercase दोनों letters match होंगे।

```
const regex = /[a-z0-9]/;
console.log("a".match(regex));        // Output: ["a"]
console.log("5".match(regex));        // Output: ["5"]
console.log("G".match(regex));        // Output: null (because it's uppercase)
```

- [a-z0-9] में आप lowercase letters और digits दोनों को match कर सकते हैं।

How To Define A Negated Character Set or Range.

Negated character set या range का use आपको उन characters को match करने के लिए करता है जो specified set या range में नहीं होते। इसका मतलब है की आप define कर सकते हैं की आपको क्या match नहीं करना है। Negated character set को define करने के लिए square brackets [ ] के अंदर caret ^ का use होता है।

Syntax:

- **[^a-z]:** Lowercase letters a से z को छोड़ कर कोई भी character match करेगा।
- **[^A-Z]:** Uppercase letters A से Z को छोड़ कर कोई भी character match करेगा।
- **[^0-9]:** Digits 0 से 9 को छोड़ कर कोई भी character match करेगा।

Examples:

```
const regex = /[^abc]/;
console.log("d".match(regex));        // Output: ["d"]
console.log("a".match(regex));        // Output: null (क्यूंकि "a" bracket के अंदर है)
```

```
const str = "hello";
const regex = /[^aeiou]/g;
console.log(str.match(regex));     // Output: ["h", "l", "l"]
```

- इस example में, regex pattern [^aeiou] non-vowels को match करेगा, यानी vowels a, e, i, o, u के अलावा बाकी characters को match करेगा। "hello" में, "h", "l", और "l" match होते हैं।

```
const regex = /[^a-z]/;
console.log("A".match(regex));        // Output: ["A"]
console.log("5".match(regex));        // Output: ["5"]
console.log("a".match(regex));        // Output: null (because it's a lowercase letter)
```

- [a-z] में जो भी characters हैं, उन्हें exclude करने के लिए [^a-z] use होता है। यहां "A" और "5" match होंगे, लेकिन "a" match नहीं होगा।

```
const regex = /[^0-9]/;
console.log("a".match(regex));        // Output: ["a"]
console.log("Z".match(regex));        // Output: ["Z"]
console.log("5".match(regex));        // Output: null (because it's a digit)
```

- [0-9] में जो digits हैं, उन्हें exclude करने के लिए [^0-9] use होता है। यहां "a" और "Z" match होंगे, लेकिन "5" match नहीं होगा।

```
const regex = /[^a-zA-Z0-9]/;
console.log("@".match(regex));        // Output: ["@"]
console.log("8".match(regex));        // Output: null (because it's a digit)
console.log("b".match(regex));        // Output: null (because it's a lowercase letter)
```

- [^a-zA-Z0-9] के साथ, आप lowercase letters, uppercase letters, और digits को exclude कर रहे हैं। इसलिए "@" match होगा, लेकिन "8" और "b" match नहीं होंगे।

```
const regex = /[^\s]/;
console.log("H".match(regex));        // Output: ["H"]
console.log(" ".match(regex));        // Output: null (because it's a space)
```

- \s whitespace characters को denote करता है, और [^\s] negated character set whitespace को exclude करता है। इसलिए "H" match होगा, लेकिन " " (space) match नहीं होगा।

Meta-Characters

How To Match Any Character With A Period Or Dot (Except For A New Line).

Regular expressions में period या dot (.) का use किसी भी single character को match करने के लिए होता है, except newline character (\n). मतलब, आप dot का use करके कोई भी character match कर सकते हैं जैसे letters, digits, special characters, etc., लेकिन newline character match नहीं होता।

Example:

```
const regex = /h.t/;
console.log("hat".match(regex));        // Output: ["hat"]
console.log("hit".match(regex));        // Output: ["hit"]
console.log("h t".match(regex));        // Output: ["h t"]
console.log("hot\n".match(regex));      // Output: ["hot"]
```

- यहां h.t pattern "h" के बाद कोई भी single character match करेगा, और फिर "t". "hat", "hit", और "h t" match होंगे, लेकिन newline (\n) match नहीं होगा।

```
const regex = /.o./;
console.log("box".match(regex));        // Output: ["box"]
console.log("fo#".match(regex));        // Output: ["fo#"]
console.log("to\n".match(regex));       // Output: null (because newline is not matched)
```

- . किसी भी character को match करेगा जो "o" के आगे और पहले हो, जैसे "box" या "fo#".

```
const regex = /a.c/;
console.log("a c".match(regex));        // Output: ["a c"]
console.log("abc".match(regex));        // Output: ["abc"]
console.log("a\nc".match(regex));       // Output: null (because newline is not matched)
```

- यहां . space को भी match करेगा, तो "a c" और "abc" match होंगे, लेकिन "a\nc" match नहीं होगा।

How To Escape (Turn OFF Special Meaning) Reserved Regular Expression Characters By Using The Backslash "\" Character.

Regular expressions में कुछ characters का special meaning होता है, जैसे . (dot), *, +, ?, ^, $, [ ], { }, |, ( ), etc. अगर आप इन characters को as a literal (यानी की उनका special meaning off करके) use करना चाहते हैं, तो आपको इन्हे escape करना पड़ता है। Escape करने के लिए backslash **** character का use होता है।

Example:

```
const regex = /a\.b/;
console.log("a.b".match(regex));  // Output: ["a.b"]
console.log("acb".match(regex));  // Output: null
```

- a\.b pattern में . as a literal dot match करेगा, तो "a.b" match होगा, लेकिन "acb" match नहीं होगा।

```
const regex = /a\*b/;
console.log("a*b".match(regex));  // Output: ["a*b"]
console.log("ab".match(regex));   // Output: null
```

- a*b pattern में * as a literal asterisk match करेगा, तो "a*b" match होगा, लेकिन "ab" match नहीं होगा।

```
const regex = /\(test\)/;
console.log("(test)".match(regex));     // Output: ["(test)"]
console.log("test".match(regex));       // Output: null
```

- \(test\) pattern में (और) as literal parentheses match करेंगे, तो "(test)" match होगा, लेकिन "test" match नहीं होगा।

```
const regex = /price\: \$5/;
console.log("price: $5".match(regex));     // Output: ["price: $5"]
console.log("price: 5".match(regex));      // Output: null
```

- \$ pattern में $ as a literal dollar sign match करेगा, तो "price: $5" match होगा, लेकिन "price: 5" match नहीं होगा।

```
const regex = /C:\\Users\\/;
console.log("C:\\Users\\".match(regex));     // Output: ["C:\\Users\\"]
console.log("C:Users".match(regex));         // Output: null
```

- Backslash को escape करने के लिए आपको text में ही double backslash \\ लिखना पड़ता है। इसलिए अगर "C\Users\" match करने के लिए ये "C:\\Users\\" match लिखना होगा।

How To Match Any Word Character (Alphanumeric).

Regular expressions में किसी भी word character को match करने के लिए \w का use होता है। Word character में आपके alphanumeric characters (letters और digits) और underscore (_) आते हैं।

Syntax:

- **\w:** Matches any single word character (alphanumeric characters + underscore).
- **\w+:** Matches one or more word characters.

Examples:

```
const regex = /\w/;
console.log("a".match(regex));        // Output: ["a"]
console.log("1".match(regex));        // Output: ["1"]
console.log("_".match(regex));        // Output: ["_"]
console.log("@".match(regex));        // Output: null (because @ is not a word character)
```

- \w pattern "a", "1", और "_" को match करेगा, लेकिन "@" को match नहीं करेगा क्यूंकि यह word character नहीं है।

```
const regex = /\w+/;
console.log("hello123".match(regex));        // Output: ["hello123"]
console.log("_test_case".match(regex));        // Output: ["_test_case"]
console.log("data-123".match(regex));        // Output: ["data"]
```

- \w+ pattern one or more word characters को match करेगा। "hello123" और "_test_case" match होंगे, लेकिन "data-123" में dash (-) non-word character है, इसलिए यह match break हो जाता है।

```
const regex = /\w\w\w/;
console.log("abc".match(regex));        // Output: ["abc"]
console.log("1a_".match(regex));        // Output: ["1a_"]
console.log("a!".match(regex));        // Output: null (because only two word characters are present)
```

- \w\w\w pattern तीन word characters के combination को match करेगा। "abc" और "1a_" match होंगे, लेकिन "a!" match नहीं होगा क्यूंकि सिर्फ दो word characters हैं।

How To Match Any Non-Word Character.

Regular expressions में किसी भी non-word character को match करने के लिए \W का use होता है। Non-word characters वो होते हैं जो alphanumeric characters (letters और digits) और underscore (_) के अलावा होते हैं। इसका मतलब है की \W special characters, spaces, punctuation marks, etc., को match करता है।

Syntax:

- **\W:** Matches any single non-word character.
- **\W+:** Matches one or more non-word characters.

Examples:

```
const regex = /\W/;
console.log("@".match(regex));        // Output: ["@"]
console.log("!".match(regex));        // Output: ["!"]
console.log(" ".match(regex));        // Output: [" "]
```

```
console.log("a".match(regex));        // Output: null (because "a" is a word character)
```

- \W pattern "@", "!", और " " (space) को match करेगा, लेकिन "a" को match नहीं करेगा क्यूंकि यह word character है।

```
const regex = /\W+/;
console.log("!@#".match(regex));        // Output: ["!@#"]
console.log("   ".match(regex));        // Output: ["   "] (multiple spaces)
console.log("hello!".match(regex));        // Output: ["!"] (matches the exclamation mark)
```

- \W+ pattern one or more non-word characters को match करेगा। "!@#" aur " " match होंगे, और "hello!" में सिर्फ "!" match होगा।

```
const regex = /\w\W\w/;
console.log("a@b".match(regex));        // Output: ["a@b"]
console.log("1!2".match(regex));        // Output: ["1!2"]
console.log("ab!".match(regex));        // Output: null (because there's no word character after the non-word character)
```

- \w\W\w pattern एक word character, एक non-word character, और फिर एक और word character के combination को match करेगा। "a@b" और "1!2" match होंगे, लेकिन "ab!" match नहीं होगा।

How To Match Any Digit.

Regular expressions में किसी भी digit को match करने के लिए \d का use होता है। \d shorthand है [0-9] के लिए, यानी यह "0" से "9" तक के कोई भी single digit को match करता है।

Syntax:

- **\d:** Matches any single digit (0-9).
- **\d+:** Matches one or more digits.
- **\d{n}:** Matches exactly n digits.
- **\d{n,m}:** Matches between n and m digits.

Examples:

```
const regex = /\d/;
console.log("5".match(regex));        // Output: ["5"]
console.log("a".match(regex));        // Output: null (because "a" is not a digit)
console.log("9".match(regex));        // Output: ["9"]
```

- \d pattern "5" और "9" को match करेगा, लेकिन "a" को match नहीं करेगा।

```
const regex = /\d+/;
console.log("123".match(regex));        // Output: ["123"]
console.log("abc123def".match(regex));        // Output: ["123"]
console.log("no digits".match(regex));        // Output: null
```

- \d+ pattern एक या उससे ज़्यादा consecutive digits को match करेगा। "123" और "abc123def" में "123" match होगा, लेकिन "no digits" में कोई match नहीं होगा।

```
const regex = /\d{4}/;
console.log("Year: 2021".match(regex));        // Output: ["2021"]
console.log("123".match(regex));        // Output: null (because only 3 digits)
```

- \d{4} pattern exactly 4 consecutive digits को match करेगा। "Year: 2021" में "2021" match होगा, लेकिन "123" match नहीं होगा।

```
const regex = /\(\d{3}\) \d{3}-\d{4}/;
console.log("(123) 456-7890".match(regex));        // Output: ["(123) 456-7890"]
console.log("123-456-7890".match(regex));          // Output: null
```

- यहां pattern एक US phone number format को match कर रहा है : "(123) 456-7890". इसमें parentheses के अंदर 3 digits, फिर space, फिर 3 digits, dash, और फिर 4 digits.

ASSIGNMENT: Try for Indian number starting from +91 and having 10 digits 11 1111 1111.

```
const regex = /\d\D\d/;
console.log("4a5".match(regex));        // Output: ["4a5"]
console.log("3 4".match(regex));        // Output: ["3 4"]
console.log("56".match(regex));         // Output: null (because there's no non-digit character between digits)
```

- \d\D\d pattern एक digit, फिर non-digit, फिर digit के sequence को match करता है। "4a5" और "3 4" match होंगे, लेकिन "56" match नहीं होगा।

How To Match Any Non-Digit.

Regular expressions में किसी भी non-digit character को match करने के लिए \D का use होता है। \D shorthand है [^\d] यानी not-digit के लिए, जो "0" से "9" के अलावा किसी भी character को match करेगा।

Syntax:

- **\D:** Matches any single non-digit character.
- **\D+:** Matches one or more non-digit characters.

Examples:

```
const regex = /\D/;
console.log("a".match(regex));        // Output: ["a"]
console.log("#".match(regex));        // Output: ["#"]
console.log("5".match(regex));        // Output: null (because "5" is a digit)
```

- \D pattern "a" और "#" को match करेगा, लेकिन "5" को match नहीं करेगा क्यूंकि यह digit है।

```
const regex = /\D+/;
console.log("abc!".match(regex));        // Output: ["abc!"]
console.log("123abc".match(regex));        // Output: ["abc"]
console.log("12345".match(regex));        // Output: null (because all characters are digits)
```

- \D+ pattern एक या उससे ज़्यादा consecutive non-digit characters को match करेगा। "abc!" और "123abc" में "abc" match होगा, लेकिन "12345" में कोई match नहीं होगा।

```
const regex = /\d\D\d/;
console.log("3a5".match(regex));        // Output: ["3a5"]
console.log("2-4".match(regex));        // Output: ["2-4"]
console.log("78".match(regex));         // Output: null (because there's no non-digit character between digits)
```

- \d\D\d pattern एक digit, फिर non-digit, और फिर digit के sequence को match करता है। "3a5" और "2-4" match होंगे, लेकिन "78" match नहीं होगा क्यूंकि यहां बीच में कोई non-digit character नहीं है।

```
const regex = /\D\d+/;
console.log("A123".match(regex));        // Output: ["A123"]
console.log("#9876".match(regex));        // Output: ["#9876"]
console.log("12345".match(regex));        // Output: null (because it starts with a digit)
```

- \D\d+ pattern एक non-digit character और उसके बाद एक या उससे ज़्यादा digits को match करता है। "A123" और "#9876" match होंगे, लेकिन "12345" match नहीं होगा क्यूंकि यह digit से start होता है।

Some more metacharacters

Regular expressions में, metacharacters वो special symbols होते हैं जो search patterns define करते हैं। यह characters आपको complex patterns बनाने में मदद करते हैं। यहां पर कुछ और common metacharacters हैं जो regular expressions में use होते हैं :

1. ( ) (Parentheses): Parentheses (( )) multiple tokens को group करने और match को remember करने के लिए use होती हैं (capturing groups).

```
const regex = /(abc)/;
console.log("abc".match(regex));          // Output: ["abc", "abc"]
```

2. \b (Word Boundary): Word boundary (\b) वो position होती है जहाँ एक word start या end होता है।

```
const regex = /\bword\b/;
console.log("a word here".match(regex));          // Output: ["word"]
console.log("sword".match(regex));          // Output: null (क्यूंकि "word" पूरा word नहीं है)
```

3. \B (Non-Word Boundary): Non-word boundary (\B) वो position होती है जहाँ word boundary नहीं होती।

```
const regex = /\Bword\B/;
console.log("swordfish".match(regex));          // Output: ["word"]
console.log("word".match(regex));          // Output: null (क्यूंकि "word" word boundary पर है)
```

4. \s (Whitespace): Whitespace (\s) space, tab, newline, या किसी भी white space character को match करता है।

```
const regex = /\s/;
console.log("a b".match(regex));          // Output: [" "]
console.log("ab".match(regex));          // Output: null (क्यूंकि white space नहीं है)
```

5. \S (Non-Whitespace): Non-whitespace (\S) कोई भी character match करता है जो white space नहीं है।

```
const regex = /\S/;
console.log("a b".match(regex));          // Output: ["a"]
console.log("   ".match(regex));          // Output: null (क्यूंकि सब white space है)
```

Quantifiers

How To Match A Character Either Zero OR One Time.

अगर आपको किसी character को match करना है जो या तो zero या एक बार appear होता हो, तो आप question mark (?) का use कर सकते हैं regular expression में।

? (Question Mark): यह metacharacter preceding element को optional बनता है। इसका मतलब यह है की वो element या तो zero बार match होगा या एक बार।

```
const regex = /colou?r/;
console.log("color".match(regex));          // Output: ["color"]
console.log("colour".match(regex));          // Output: ["colour"]
```

```
console.log("colorr".match(regex));        // Output: ["color"]
console.log("colr".match(regex));          // Output: null (क्यूंकि "u" या "o" missing है)
```

- /colou?r/ यह regular expression "colo" के बाद "u" को optional बनता है।
- यह "color" और "colour" दोनों को match करेगा क्यूंकि "u" या तो एक बार है ("colour") या नहीं है ("color").
- अगर "u" के बाद कुछ और characters आ रहे हैं, जैसा की "colorr", तो यह सिर्फ "color" को match करेगा और extra "r" को ignore करेगा।

How To Match Either Zero OR More Repetitions Of A Character.

अगर आपको किसी character को match करना है जो या तो zero बार या फिर ज़्यादा बार repeat होता हो, तो आप asterisk (*) का use कर सकते हैं regular expression में।

*** (Asterisk):** यह metacharacter preceding element को zero या उससे ज़्यादा बार match करता है। इसका matlab यह है की वो element zero बार (मतलब नहीं भी हो तो चलेगा) या multiple बार appear हो सकता है।

```
const regex = /a*/;
console.log("aaa".match(regex));     // Output: ["aaa"]
console.log("a".match(regex));       // Output: ["a"]
console.log("".match(regex));  // Output: [""] (empty string भी match करेगा क्यूंकि "a" zero times हो सकता है)
console.log("bb".match(regex));      // Output: [""] (क्यूंकि "a" zero times है यहाँ)
```

- /a*/ यह regular expression "a" character को zero या उससे ज़्यादा बार match करेगा।
- "aaa", "a", और even empty string "" भी match होगा क्यूंकि * allow करता है की "a" zero बार भी हो सकता है।
- अगर "a" string में नहीं है, जैसे "bb", तो भी यह expression एक empty match return करेगा, जो indicate करता है की "a" zero times match हो गया।

How To Match Either One OR More Repetitions Of A Character.

अगर आपको किसी character को match करना है जो कम से कम एक बार और उससे ज़्यादा बार repeat होता हो, तो आप plus (+) का use कर सकते हैं regular expression में।

+ (Plus): यह metacharacter preceding element को एक या उससे ज़्यादा बार match करता है। इसका मतलब यह है की वो element कम से कम एक बार appear होना चाहिए, और उसके बाद जितनी बार भी हो सकता है।

```
const regex = /a+/;
console.log("aaa".match(regex));       // Output: ["aaa"]
console.log("a".match(regex));         // Output: ["a"]
console.log("".match(regex));          // Output: null (क्यूंकि यहाँ कोई "a" नहीं है)
console.log("bb".match(regex));        // Output: null (क्यूंकि "a" नहीं है यहाँ)
```

- /a+/ यह regular expression "a" character को कम से कम एक बार match करेगा, और उसके बाद जितनी बार भी हो सकता है।
- "aaa" और "a" दोनों match हो जायेंगे, क्यूंकि "a" कम से कम एक बार appear होता है।
- अगर string में "a" नहीं है, जैसे empty string "" या "bb", तो यह match नहीं करेगा और null return करेगा।

How To Specify The Exact Amount Of Repetitions To Match.

अगर आपको regular expression में किसी character या group of characters को exact number of repetitions के साथ match करना है, तो आप curly braces (**{ }**) का use कर सकते हैं।

Explanation:

- **{n}:** यह pattern preceding element को exactly n बार match करेगा।

Example

```
const regex = /a{3}/;
console.log("aaa".match(regex));        // Output: ["aaa"]
console.log("aa".match(regex));         // Output: null (क्यूंकि "a" सिर्फ 2 बार है)
console.log("aaaa".match(regex));       // Output: ["aaa"] (सिर्फ पहले 3 "a" match होंगे)
```

How To Specify The Least (Minimum) Amount Of Repetitions To Match.

अगर आपको regular expression में किसी character या group of characters को minimum (कम से कम) number of repetitions के साथ match करना है, तो आप curly braces (**{n,}**) का use कर सकते हैं।

Explanation:

- **{n,}:** यह pattern preceding element को कम से कम n बार match करेगा। इसके बाद जितनी बार भी repetition हो सकती है, सब match होंगी।

```
const regex = /a{3,}/;
console.log("aaa".match(regex));        // Output: ["aaa"]
console.log("aaaaa".match(regex));      // Output: ["aaaaa"]
console.log("aa".match(regex));         // Output: null (क्यूंकि "a" सिर्फ 2 बार है)
```

- /a{3,}/ यह regular expression "a" को कम से कम 3 बार match करेगा, और अगर "a" उसके बाद भी हो तो उनको भी match करेगा।
- "aaa" match हो जायेगा क्यूंकि यह exactly 3 बार "a" है।
- "aaaaa" भी match होगा क्यूंकि यह 3 से ज़्यादा "a" है।
- "aa" match नहीं होगा क्यूंकि इसमें "a" सिर्फ 2 बार है, जो की minimum requirement से कम है।

How To Specify The Minimum And Maximum Amount Of Repetitions To Match.

अगर आपको regular expression में किसी character या group of characters को minimum और maximum number of repetitions के साथ match करना है, तो आप curly braces (**{n,m}**) का use कर सकते हैं।

Explanation:

- **{n,m}:** यह pattern preceding element को कम से कम n बार और ज़्यादा से ज़्यादा m बार match करेगा।

```
const regex = /a{2,4}/;
console.log("aa".match(regex));         // Output: ["aa"] (क्यूंकि यह 2 बार "a" है)
console.log("aaa".match(regex));        // Output: ["aaa"] (क्यूंकि यह 3 बार "a" है)
console.log("aaaa".match(regex));       // Output: ["aaaa"] (क्यूंकि यह 4 बार "a" है)
console.log("a".match(regex));          // Output: null (क्यूंकि यह सिर्फ 1 बार "a" है)
console.log("aaaaa".match(regex));      // Output: ["aaaa"] (क्यूंकि सिर्फ पहले 4 "a" match होंगे)
```

- /a{2,4}/ यह regular expression "a" को कम से कम 2 बार और ज़्यादा से ज़्यादा 4 बार match करेगा।
- "aa", "aaa", और "aaaa" match हो जायेंगे क्यूंकि यह range {2,4} के अंदर है।
- अगर "a" कम से कम 2 बार नहीं होगा, जैसे "a", तो match नहीं होगा और null return करेगा।
- अगर "a" 4 से ज़्यादा बार होगा, जैसे "aaaaa", तो सिर्फ पहले 4 "a" match होंगे।

Line Anchors - Assert The Start And The End Of A String.

Line anchors regular expressions में use होते हैं string के start या end को specify करने के लिए। दो primary line anchors होते हैं : ^ (caret) और $ (dollar)।

1. ^ (Caret): ^ symbol को regular expression में use करते हैं string के start से match करने के लिए।

```
const regex = /^Hello/;
console.log("Hello, world!".match(regex));          // Output: ["Hello"]
console.log("Say Hello".match(regex));          // Output: null (क्यूंकि "Hello" start में नहीं है)
```

- /^Hello/ यह regular expression सिर्फ तब match करेगा जब "Hello" string के start में हो।
- अगर "Hello" start में नहीं है, जैसे "Say Hello", तो match नहीं होगा और null return करेगा।

2. $ (Dollar): $ symbol को regular expression में use करते हैं string के end से match करने के लिए।

```
const regex = /world!$/;
console.log("Hello, world!".match(regex));          // Output: ["world!"]
console.log("world is beautiful".match(regex));          // Output: null (क्यूंकि "world!" end में नहीं है)
```

- /world!$/ यह regular expression सिर्फ तब match करेगा जब "world!" string के end में हो।
- अगर "world!" string के end में नहीं है, जैसे "world is beautiful", तो match नहीं होगा और null return करेगा।

Combination Example: अगर आपको string को exactly match करना है, जिसमे start और end दोनों specific characters से हों, तो आप ^ और $ को साथ में use कर सकते हैं।

```
const regex = /^Hello, world!$/;
console.log("Hello, world!".match(regex));          // Output: ["Hello, world!"]
console.log("Hello, world".match(regex));          // Output: null (क्यूंकि end में "!" missing है)
console.log("Say Hello, world!".match(regex));          // Output: null (क्यूंकि start में "Hello" नहीं है)
```

- /^Hello, world!$/ यह regular expression तभी match करेगा जब पूरा string exactly "Hello, world!" हो, बिना किसी extra character के।
- अगर string के start या end में कुछ और characters होंगे, तो match fail हो जायेगा।

Alteration

How To Match Alternate Strings OR Words.

अगर आपको regular expression में multiple alternate strings या words को match करना है, तो आप pipe (|) symbol का use कर सकते हैं। यह symbol "OR" के equivalent है, जो different alternatives को match करने में help करता है।

Explanation:

- **| (Pipe):** इसका use करके आप specify कर सकते हैं की आपका pattern या तो left side का string match करेगा, या right side का।
- आप multiple alternatives को भी specify कर सकते हैं, जैसे A|B|C, जो "A", "B", या "C" में से किसी को भी match करेगा।

Example 1: Matching Two Words

```
const regex = /cat|dog/;
console.log("I have a cat".match(regex));        // Output: ["cat"]
console.log("I have a dog".match(regex));        // Output: ["dog"]
console.log("I have a fish".match(regex));       // Output: null (क्यूंकि यहाँ "cat" या "dog" नहीं है)
```

Example 2: Matching Multiple Words

```
const regex = /apple|banana|orange/;
console.log("I like apple".match(regex));        // Output: ["apple"]
console.log("I like banana".match(regex));       // Output: ["banana"]
console.log("I like orange".match(regex));       // Output: ["orange"]
console.log("I like grapes".match(regex));       // Output: null (क्यूंकि यहाँ specified fruits नहीं है)
```

Example 3: Matching With Other Patterns

```
const regex = /colou?r|color/;
console.log("I like color".match(regex));        // Output: ["color"]
console.log("I like colour".match(regex));       // Output: ["colour"]
```

- /cat|dog/ यह regular expression या तो "cat" को match करेगा या "dog" को, depending on string में क्या है।
- /apple|banana|orange/ यह regular expression तीनो में से किसी भी एक fruit को match करेगा।
- आप multiple words या strings को | के साथ separate कर के उनको alternate options के रूप में specify कर सकते हैं।

Negative lookahead assertion (?! ...):

- इसका मतलब यह है की expression को ये ensure करना है की जो भी इसके बाद आ रहा है , वह specific pattern match नहीं करना चाहिए।
- अगर specified pattern match हो जाता है , तो यह assertion fail हो जाती है , और regex उस जगह पर match नहीं करेगा।

```
const regex = /a(?!b)/;
console.log("ac".match(regex));     // ["a"]
console.log("ab".match(regex));     // null
```

- /a(?!b)/: यह regex a को तब match करेगा जब उसके बाद b नहीं होगा।
- "ac" में a के बाद b नहीं है, इसलिए यह match करता है।
- "ab" में a के बाद b है, इसलिए यह match नहीं करेगा।

By Om Shankar Mishra

Projects

Validating The Username Field

User Requirements:

- Username का पहला character एक alphabetic character (uppercase या lowercase) होना चाहिए।
- Username में letters, digits (0-9), dots (.), underscores (_), और hyphens (-) allowed हैं।
- Special characters (dots, underscores, hyphens) एक के बाद एक नहीं आने चाहिए (जैसे "user..name" या "user__name" allowed नहीं हैं)।
- Username की length 5 से 20 characters के बीच होनी चाहिए।
- Username का last character एक alphanumeric character होना चाहिए (letters या digits).

```
<form id="usernameForm">
 <input type="text" id="username" placeholder="Enter username">
 <button type="submit">Submit</button>
 <p id="errorMessage" style="color: red;"></p>
</form>
```

Enter username | Submit

```
<script>
function validateUsername(username) {
   const usernameRegex = /^[a-zA-Z]([._-](?![._-])|[a-zA-Z0-9]){3,18}[a-zA-Z0-9]$/;
   return usernameRegex.test(username);
}
 document.getElementById("usernameForm").addEventListener("submit", function(event) {
   event.preventDefault();    // Form को submit होने से रोकता है
   const username = document.getElementById("username").value;
   const isValid = validateUsername(username);
   if (isValid) {
     document.getElementById("errorMessage").textContent = "Username is valid!";
   } else {
     document.getElementById("errorMessage").textContent = "Invalid username!";
   }
 });
</script>
```

Regular Expression Breakdown:

- **^:** यह regex को string के start से match करने के लिए anchor करता है, मतलब pattern string के बिलकुल starting से match करना चाहिए।
- **[a-zA-Z]:** Username का पहला character एक alphabetic character होना चाहिए, जो uppercase (A-Z) या lowercase (a-z) हो सकता है।
- **[._-]:** यह allow करता है dot (.), underscore (_), या hyphen (-).
- **(?![._-]):** Negative lookahead ensure करता है की अगर dot, underscore, या hyphen आता है, तो उसके तुरंत बाद कोई और special character (dot, underscore, hyphen) नहीं होना चाहिए। यह consecutive special characters को रोकता है।

Free Billangual Notes Available !

HTML PDF | CSS PDF | JavaScript PDF | Angular PDF | React PDF | Hibernate PDF

SpringBoot PDF | JDBC PDF

Git & GitHub PDF | Data Analyst PDF

www.linkedin.com/in/omi92229

Basics of HTML

What is HTML?

Definition: HTML का full form HyperText Markup Language है। यह एक standard language है जो web pages बनाने और design करने के लिए use होती है। यह web pages को structure provide करता है अलग -अलग elements और tags का उसे करके जो content और layout define करते हैं।

History: HTML को Tim Berners-Lee ने 1991 में बनाया था। यह समय के साथ evolve होता गया, और कई versions release हुए जो इसकी capabilities को enhance करते गए। अभी जो latest version है, वह HTML5 है, जो नए elements और features introduce करता है जिससे multimedia support, semantics, और applications और robust ho गए हैं।

Key Features:

- **Structure:** HTML headings, paragraphs, lists, links, और दूसरे content types का use करके web pages का structure define करता है।
- **Elements और Tags:** HTML tags का use करता है जो angle brackets में होते हैं (जैसे <tagname>). ज़्यादातर elements के पास opening tag और closing tag होता है (जैसे `<p>content</p>`), पर कुछ self-closing भी होते हैं (जैसे `<img src="image.jpg" />`).
- **Attributes:** Tags के पास attributes होते हैं जो element के बारे में additional information देते हैं (जैसे `<a href="https://example.com">link</a>`). Attributes opening tag के अंदर लिखे जाते हैं।
- **Hypertext:** HTML hyperlinks बनाने की सुविधा देता है, जो users को different web pages या resources के बीच navigate करने में मदद करता है।

Importance: HTML web development का foundation है। HTML के बिना web pages के पास basic

Data Science PDF | SQL PDF

Mongo DB PDF | JAVA PDF

Om Shankar Mishra

Data Science with R Programming

Lesson 1: Introduction to Business Analytics

Business analytics का मतलब है statistical analysis, data mining, और predictive modeling का use करके data को analyze और interpret करना, ताकि informed business decisions लिए जा सकें। यह modern business strategies का एक essential part है, जो valuable insights provide करता है, जिससे organizations market trends, customer preferences, और operational efficiencies को समझ सकते हैं।

Overview

Business analytics एक process है जो data को actionable insights में convert करता है, ताकि business outcomes को improve किया जा सके। यह various techniques और methodologies को encompass करता है जो historical data को समझने, future trends को predict करने, और data-driven decisions लेने में help करता है। इसका main goal है data को leverage करके business planning और strategic initiatives को drive करना।

Business Decisions and Analytics

Analytics, business decision-making में एक crucial role play करता है , क्यूंकि यह evidence-based insights provide करता है जो strategic planning, operational improvements, और competitive strategies को guide करता है। Analytics-supported business decisions से efficiency बढ़ सकती है, costs कम हो सकती हैं, customer satisfaction improve हो सकती है, और profitability बढ़ सकती है। Decision-making process में problem को identify करना, relevant data collect करना , data analyze करना, insights generate करना, और best course of action implement करना शामिल है।

Types of Business Analytics

- **|:** OR operator है। यह allow करता है या तो एक special character के बाद non-special character हो, या फिर एक alphanumeric character हो।
- **[a-zA-Z0-9]:** यह alphanumeric character को allow करता है (letter या digit).
- **{3,18}:** यह pattern 3 से 18 बार repeat हो सकता है, जिससे username की total length 5 से 20 characters के बीच रहती है (पहले और last character को include करके)।
- **[a-zA-Z0-9]:** Username का last character एक alphanumeric character होना चाहिए (letter या digit).
- **$:** यह regex को string के end पर anchor करता है, मतलब pattern string के बिलकुल end तक match करना चाहिए।

Validating Email Address

User Requirement:

Username Part:

- Lowercase letter या number से शुरू होना चाहिए।
- Username में dots (.), underscores (_), या hyphens (-) आ सकते हैं, लेकिन यह consecutive (एक के बाद एक) नहीं होने चाहिए।
- Username का total length 4 से 26 characters के बीच होना चाहिए।

Domain Part: Domain name 2 से 6 lowercase letters का होना चाहिए (जैसे gmail).

Domain Extension Part: Domain extension भी 2 से 6 lowercase letters का होना चाहिए (जैसे com).

```
function validateEmail(email) {
  // Regular expression for email validation
  const regex = /^[a-z0-9]([._-](?![._-])|[a-z0-9]){3,25}@[a-z]{2,6}\.[a-z]{2,6}$/i;
  return regex.test(email);
}
// Test the function
const email = 'omshankar92229@gmail.Com';
if (validateEmail(email)) {
  console.log(email + ' is a valid email address.');
} else {
  console.log(email + ' is not a valid email address.');
}
```

Explanation:

^[a-z0-9]: Email का username lowercase letter (a-z) या digit (0-9) से start होना चाहिए।

([._-](?![._-])|[a-z0-9]){3,25}:

- यह group special characters जैसे dots (.), underscores (_), या hyphens (-) को allow करता है, लेकिन तभी जब उनके बाद कोई और special character न हो।
- साथ ही यह group alphanumeric characters (a-z0-9) को भी allow करता है।
- {3,25} का मतलब यह pattern 3 से 25 बार repeat हो सकता है, ताकि username का total length 4 से 26 characters के बीच हो।

@: Username और domain के बीच @ symbol होना चाहिए।

[a-z]{2,6}: Domain name 2 से 6 lowercase letters का होना चाहिए।

\.[a-z]{2,6}: Domain extension part dot (.) से शुरू होता है और उसके बाद 2 से 6 lowercase letters होने चाहिए।

$: यह string के end को denote करता है, ताकि कोई extra characters ना हो domain extension के बाद।

/i: यह flag regex को case-insensitive बनता है, मतलब uppercase और lowercase letters में कोई फर्क नहीं होगा।

(?= ...) - Positive Lookahead Assertion:

(?= ...) एक positive lookahead assertion है। यह check करता है की given pattern के बाद एक specific pattern आ रहा है या नहीं, लेकिन यह actual match के part में include नहीं होता।

```
const regex = /\d(?=abc)/;
const str = "3abc";
const result = str.match(regex);
console.log(result);     // Output: ["3"]
```

- यह ensure करता है की जो digit आप match कर रहे हो (\d), उसके बाद abc होना चाहिए।
- लेकिन output में सिर्फ 3 आता है, क्यूंकि (?=abc) सिर्फ check करता है, match नहीं करता।

Validating Password:

User Requirement :

- Password length कम से कम 8 और ज़्यादा से ज़्यादा 20 characters.
- कम से कम एक digit (0-9).
- कम से कम एक uppercase letter (A-Z).
- कम से कम एक lowercase letter (a-z).
- कम से कम एक special character होना चाहिए from the set !@#$%&*()-+=^.

```
const passwordRegex = /^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?=.*[!@#$%&*()\-+=^]).{8,20}$/;
function validatePassword(password) {
   return passwordRegex.test(password);
}
// Test examples
console.log(validatePassword("Password123!"));          // true
console.log(validatePassword("password"));          // false (no uppercase, no digit, no special character)
console.log(validatePassword("Password!"));          // false (no digit)
console.log(validatePassword("Pass123!"));          // false (less than 8 characters)
console.log(validatePassword("Password123!@#"));          // true
```

Explanation:

^: यह string के start को denote करता है।

(?=.*\d): यह ensure करता है की password में कम से कम एक digit (\d) हो। .* का मतलब है की इसके आगे कुछ भी हो सकता है, लेकिन (?=.*\d) ensure करेगा की कही ना कही एक digit ज़रूर हो।

(?=.*[a-z]): यह ensure करता है की password में कम से कम एक lowercase letter ([a-z]) हो।

(?=.*[A-Z]): यह ensure करता है की password में कम से कम एक uppercase letter ([A-Z]) हो।
**(?=.*[!@#$%&*()\-+=^]):** यह ensure करता है की password में कम से कम एक special character हो जो इस set में से हो : !@#$%&*()-+=^.
.{8,20}: यह ensure करता है की password की length कम से कम 8 characters और ज़्यादा से ज़्यादा 20 characters हो।
$: यह string के end को denote करता है, ताकि password का pattern defined criteria के हिसाब से ही हो।
Note: Special characters जैसे (और) को escape (\) की ज़रुरत होती है क्यूंकि इनका regex में special meaning होता है।

Validating IP Address

IPv4 (Internet Protocol Version 4):

- IPv4 एक पुराना और widely used Internet Protocol version है जो network devices को uniquely identify करने के लिए IP addresses provide करता है।
- ये 32-bit address use करता है, जो 4 octets (यानी 4 numbers) में divided होता है। हर octet 0 से 255 के बीच एक value हो सकती है।

Limitation: IPv4 में सिर्फ 4.3 billion unique addresses possible हैं (2^32 addresses), जो modern internet के लिए काम पड़ने लगे हैं क्यूंकि ज़्यादा devices internet से connect हो रहे हैं (smartphones, laptops, IoT devices, etc.).
Leading Zeros: Decimal number के case में leading zeros का use नहीं होता है, क्यूंकि उस्से number को समझने में ambiguity हो सकती है।
Example: 092 को 92 ही मन जाता है, लेकिन इस format में लिखना incorrect है जब बात IP addresses की हो रही हो।

Octet Values: हर octet को 8-bit में लिखा जाता है। अगर आप binary में सोचो:

- 00000000 = 0 (minimum value)
- 11111111 = 255 (maximum value)

यही वजह है की octet की value 0 से 255 के बीच होनी चाहिए।

```
let ipAddress = "192.0.2.1";
let regex = /^((25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])\.){3}(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])$/;
let result = ipAddress.match(regex);
if(result){
  console.log(ipAddress+" is valid");
}else{
  console.log(ipAddress+" is invalid");
}
```

Example IPv4 Addresses:

- **Valid:** 192.168.1.1, 255.255.255.255, 172.16.0.1
- **Invalid:** 256.256.256.256, 123.045.067.089

इसमें 4 numbers होते हैं, separated by dots, और हर number 0 से 255 के बीच होता है।

IPv6 (Internet Protocol Version 6)

- IPv6 एक newer version है जो IPv4 की limitations को overcome करता है। ये 128-bit address use करता है, जो bahut ज़्यादा unique IP addresses generate कर सकता है।
- IPv6 addresses hexadecimal (0-9 और a-f letters) format में होते हैं और colons (:) से separate होते हैं।

Advantages:

- IPv6 में इतने ज़्यादा addresses available हैं की practically ये कभी ख़तम नहीं होते (2^128 addresses).
- IPv6 में better security features भी built-in होते हैं, जैसे IPsec (Internet Protocol Security).

```
let ipAddress = "2001:db8:0:1234:0:567:8:1";
let regex = /^(([0-9a-fA-F]{1,4}):){7}([0-9a-fA-F]{1,4})$/;
let result = ipAddress.match(regex);
if(result){
  console.log(ipAddress+" is valid IPv6 Address");
}else{
  console.log(ipAddress+" is invalid IPv6 Address");
}
```

Example IPv6 Addresses:

- **Valid:** 2001:0db8:85a3:0000:0000:8a2e:0370:7334, fe80::1, ::1
- **Invalid:** 2001::85a3::8a2e:0370:7334, 1200::AB00:1234::2552:7777:1313

इसमें 8 groups होते हैं, और हर group 4 hexadecimal digits का होता है।

Note: The regex pattern shared is designed to match a full IPv6 address consisting of 8 groups of 4 hexadecimal digits, separated by colons.