JavaScript Tutorial

Fundamental Concepts

[JavaScript Definition](#javascript-definition)
|
[Environment set up](#environment-set-up)
|
[What can JavaScript do?](#what-can-javascript-do)
|
[Where to use JavaScript?](#where-to-use-javascript)
|
[JavaScript output element](#javascript-output-element)
|
[JavaScript Statements](#javascript-statements)
|
[Variable Definition & syntax.](#variable)
|
[JavaScript identifiers or Variable Name](#javascript-identifiers-or-variable-name)
|
[Variable Data Types](#variable-data-types)
|
[Type Conversion](#type-conversion)
|
[False Values](#falsy-values)
|
[JavaScript Operators](#javascript-operators)
|
[Conditional Statements](#conditional-statements)
|
[Declarations](#declarations)
|
[Loops and iteration](#loops-and-iteration)
|
[JavaScript Comment](#javascript-comment)
|
[Math Object](#math-object)
|
[Date Object](#date-object)
|
[String](#string)
|
[Array](#array)
|
[Object](#object)
|
[Function](#function)

JavaScript ES6

[let & const](#let-and-const)
|
[Template and Multiline string](#template-and-multiline-string)
|
[Destructuring](#destructuring)
|
[Optional chaining](#optional-chaining)
|
[object property & function assignement](#object-property-and-function-assignement)
|
[Spread & Rest operator](#spread-and-rest-operator)
|
[Exponent operator](#exponent-operator)
|
[Default parameter](#default-value-and-parameter)
|
[arrow Function](#arrow-function)
|
[Method](#method)
|
[Classes]()
|
[Promises]()
|
[Objects Shorthand syntax]()
|
[Modules]()
|
[Generators]()

Dom

[Dom definition](#dom-definition)
|
[Selection](#element-selection-using-javaScript)
|
[Traversing Dom Elements](#traversing-dom-elements-using-javascript-dynamically)
|
[Create & Append Elements](#create-and-append-elements-using-javascript-dynamically)
|
[Remove Elements](#remove-elements-using-javaScript-dynamically)
|
[Add Events](#add-events-using-javascript-dynamically)
|
[Event Listener vs Event Handler](#difference-between-an-event-listener-and-event-handler)
|
[Deferent Event types](#deferent-type-of-event)

|
[Capturing and Bubbling](#capturing-and-bubbling)
|
[Add or remove element using event bubbling](#add-or-remove-element-using-event-bubbling)
|
[Event Delegation](#event-delegation)
|
[Add or Remove Classes](#add-or-remove-classes-using-javascript-dynamically)

Bom

[Window Object](#window-object)
|
[Window Object Methods](#window-object-methods)
|
[Alert](#alert)
|
[Confirm](#confirm)
|
[Prompt](#prompt)
|
[setTimeout](#settimeout)
|
[setInterval](#setinterval)
|
[Location Object](#location-object)
|
[Redirect to a new URL](#redirect-to-a-new-url)
|
[Navigator Object](#navigator-object)
|
[Screen Object](#screen-object)
|
[History Object](#history-object)

Web API

[Client Storage](#client-storage)
|
[Cookies](#cookies)
|
[localStorage](#localstorage)
|
[sessionStorage](#sessionstorage)
|
[IndexedDB](#indexeddb)
|
[FormData](#formdata)
|

[Drag and Drop API](#drag-and-drop-api)
|
[FileReader API](#filereader-api)
|
[Geolocation API](#geolocation-api)
|
[Notification API](#notification-api)
|
[Canvas API](#canvas-api)
|
[History API](#history-api)
|
[Network Requests](#network-requests)

[Problem Solving](#problem-solving)

Learn JavaScript

Javascript Definition

<small>Top</small>
<small>Next</small>

> JavaScript is high-level, often just-in-time compiled and multi-paradigm. It has dynamic typing, prototype-based object-orientation and first-class functions. The language was invented by Brendan Eich in 1995. It is also known as a dynamic computer programming language. The official name of JavaScript is ECMAScript. It became ECMA standard in 1997.

Environment set up

<small>Top</small>
<small>Next</small>

> Every programming language needs an environment set up to work with it. Like compilers, text editors etc. In JavaScript, the environment is easy to set up. If you have a browser and a text editor, you can run JavaScript code.

> In JavaScript, to run code, a text file is needed, where the JavaScript code will be written. The extension of a JavaScript text file will be ".js" and to see an output we have to type "console.log()".

What can JavaScript do?

<small>Top</small>
<small>Next</small>

> 1. JavaScript Can Change HTML Content

> 2. JavaScript Can Change HTML Attribute Values
> 3. JavaScript Can Change HTML Styles (CSS)
> 4. JavaScript Can Hide HTML Elements
> 5. JavaScript Can Show HTML Elements

Where to use JavaScript?

<small>Top</small>
<small>Next</small>

> 1. In HTML, JavaScript code is inserted between <script> and </script> tags.
> 2. JavaScript in \<head>
> 3. JavaScript in \<body>
> 4. External JavaScript

> Placing scripts at the bottom of the \<body> element improves the display speed, because script interpretation slows down the display.

JavaScript output element.

<small>Top</small>
<small>Next</small>

> 1. Writing into an HTML element, using innerHTML.
> 2. Writing into the HTML output using document.write().
> 3. Writing into an alert box, using window.alert().
> 4. Writing into the browser console, using console.log().
> 5. How to use consol.log() & print data.

JavaScript Statements

<small>Top</small>
<small>Next</small>

> A computer program is a list of "instructions" to be "executed" by a computer. In a programming language, these programming instructions are called statements. The statements are executed, one by one, in the same order as they are written. Semicolons separate JavaScript statements. When separated by semicolons, multiple statements on one line are allowed.

> JavaScript statements are composed of Values, Operators, Expressions, Keywords, and Comments. There are many type of statements like Block, break, continue, Empty, if...else, switch, throw, try...catch. We discuse in details in statement lessaion.

```
```

```
let x, y, z; // Statement 1
x = 5; // Statement 2
y = 6; // Statement 3
z = x + y; // Statement 4

```

Variable

<small>Top</small>
<small>Next</small>

> Variable means a container that can store. JavaScript has the functionality of variables that hold the data value and it can be changed anytime. In JavaScript, var, a reserved keyword is used to declare a variable. A variable must have a unique name. One can assign a value to a variable using equal to (=) operator when you declare it or before using it.

> JavaScript uses the keywords var, let and const to declare variables.

JavaScript identifiers or Variable Name

<small>Top</small>
<small>Next</small>

> 1. All JavaScript identifiers are case sensitive.
> 2. Hyphens are not allowed in JavaScript. They are reserved for subtractions.
> 3. Names can contain letters, digits, underscores, and dollar signs.
> 4. Names must begin with a letter.
> 5. Names can also begin with $ and _ (but we will not use it in this tutorial)
> 6. Reserved words (like JavaScript keywords) cannot be used as names.

Variable Data Types

<small>Top</small>
<small>Next</small>

> In a program, data types specify what kind of data can be stored and manipulated. In JavaScript, there are six data types. It can be divided into two main categories.

Primitive data type

> 1. String - "this is text" "this is text"

> 2. Number - 20, 20.5
> 3. Boolean - True or False
> 4. Undefined - var name;
> 5. Null- var name = "";

non-Primitive data type

> 1. Object
> 2. Array
> 3. Function.

- Number

  > The most common primitive data type is numbers. To declare numbers in JavaScript is easy. Both integer numbers and floating numbers are the same in JavaScript. Both are considered float numbers. JavaScript provides 64 bits for every single number.

  ```

  var num = 4546;
  var num2 = 55.5;
  var num3 = Number("55.5");
  var num4 = Number("55");
  console.log(Number.parseFloat(num4));
  console.log(Number.parseInt(num3));
  console.log(Number.MAX_VALUE);
  console.log(Number.MAX_SAFE_INTEGER);
  console.log(1 / 0);

  ```

- String

  > In the JavaScript programming language, anything wrapped between quotes is considered as string. It can be a single quote or double quote. Even by using a backtick sign, string can be declared in ES6. JavaScript provides two types of string functionality. A. string literal and B. string constructor.

  ```

  var str = "This is text";
  var str2 = "This is text";
  var str3 = `this is text`;
  var str4 = String("This is text");
  var str5 = String(50);
  var str6 = String(5.5);
  ```

- Boolean

> In JavaScript, Boolean is a primitive data type. The Boolean data type has two values only. It deals with true or false. Controlling program flow using conditional statements like - while, do...while, switch, if...else, the Boolean data type is used.

```

var isRaining = true;
var isComming = false;
var isEqual = Boolean(true);
var isNotEqual = Boolean(false);

```

- Null and Undefined

> Generally, in a programming language, we can assign any primitive or non-primitive type of value to a variable. But JavaScript adds the functionalities of two additional primitive type values - null and undefined. These can be assigned to a variable that has a special meaning.

```

var abc;
var text = null;

```

Type Conversion

<small>Top</small>
<small>Next</small>

- Number Conversion: Number().

```

Number(" 123 "); // 123
Number("123z"); // NaN (error reading a number at "z")
Number(true); // 1
Number(false) ); // 0
Number(undefined) // NaN
Number(null) // 0

```

> Decimal Number to Integer Number: parseInt()
> Integer Number to Decimal Number: parseFloat()

- String Conversion: String() or toString()

  ```

  String(true); // now true is a string "true"
  String(x) // returns a string from a number variable x
  String(123) // returns a string from a number literal 123
  String(100 + 23) // returns a string from a number from an expression

  ```

- Boolean Conversion: Boolean()

  ```

  Boolean(1)); // true
  Boolean(0); // false
  Boolean("0"); // true
  Boolean("hello"); // true
  Boolean(""); // false
  Boolean(" "); // spaces, also true (any non-empty string is true)

  ```

Falsy Values

<small>Top</small>
<small>Next</small>

> A falsy value is a value that is considered false when encountered in a Boolean context.

> List of falsy values: false, 0, -0, ""''``, nul, undefined, NaN

```

if (false)
if (null)
if (undefined)
if (0)
if (-0)
if (0n)
if (NaN)
if ("")

```

JavaScript Operators

<small>Top</small>
<small>Next</small>

> In JavaScript, an operator is a special symbol used to perform operations on operands (values and variables). For example, 2 + 3; // 5. Here + is an operator that performs addition, and 2 and 3 are operands. In JavaScript has list of operators it`s given bellow.

> List of JavaScript Operators:
>
> 1. Assignment operators,
> 2. Comparison operators,
> 3. Arithmetic operators,
> 4. Logical operators,
> 5. Conditional operators
> 6. Bitwise operators,
> 7. String operators,
> 8. Conditional (ternary) operators,
> 9. Comma operators,
> 10. Unary operators,
> 11. Relational operators.

1. **Assignment Operators symbol: =, +=, -=, _=, /=, %=, **=, <<=, >>=, >>>=, &=, ^=, |=, &&=, ||=, ??=**

```

let x = 50; // 50
x += 10; // 60
x -= 5; // 55
x *= 2; // 110
x /= 2; // 55
x %= 15; // 10
x **= 2; // 100
x <<= 2; // 400
x >>= 5; // 12
x &= 5; // 4
x ^= 10; // 14
x |= 10; // 14
x &&= 10; // 10
x ||= 10; // 10
x ??= 10; // 10

```

2. **Comparison Operators symbol: ==, !=, ===, !==, >, >=, <, <=.**

```

let number = 50;
if (number == 50) //Returns true if the operands are equal.
if (number != 50) // Returns true if the operands are not equal.
if (number === 50) // Returns true if the operands are equal and
of the same type.
if (number !== 50) // Returns true if the operands are of the
same type but not equal, or are of different type
if (number > 50) // Returns true if the left operand is greater
than the right operand.
if (number >= 50) // Returns true if the left operand is greater
than or equal to the right operand.
if (number < 50) // Returns true if the left operand is less
than the right operand.
if (number <= 50) // Returns true if the left operand is less
than or equal to the right operand.

```

3. **Arithmetic Operators symbol: %, ++, --, -, +, **.**

```

let number = 51;
number % 2; // Returns the integer remainder of dividing the two
operands.
number++; //  Adds one to its operand
number--; // Subtracts one from its operand.
-number; // Returns the negation of its operand
+"3"; // Attempts to convert the operand to a number, if it is
not already.
number ** 2; // Calculates the base to the exponent power, that
is, base^exponent

```

4. **Logical Operators symbol: &&, ||, !.**

```

var a = true;
var b = false;
a && a; // t && t returns true
a && b; // t && f returns false
b && a; // f && t returns false
b && b; // f && f returns false

a || a; // t && t returns true
a || b; // t && f returns true
b || a; // f && t returns true
```

```
b || b; // f && f returns false

!a; // !t returns false
!b; // !f returns true
```

5. **Bitwise operators symbol: &, |, ^, ~, <<, >>, >>>.**
6. **String operators**
7. **Conditional (ternary) operators: condition ? val1 : val2**

```
var n = 10;
console.log(n % 2 == 0 ? "Even" : "Odd");
var str = "";
str = n % 2 == 0 ? "Even" : "Odd";
console.log(str);
```

8. **Comma operators symbol: ,**
9. **Unary operators: delete.**
10. **Relational operators: in.**

Conditional Statements

<small>Top</small>
<small>Next</small>

> Conditional statements are used to perform different actions based on different conditions. Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

> In JavaScript we have the following conditional statements:

> 1. Use if to specify a block of code to be executed, if a specified condition is true
> 2. Use else to specify a block of code to be executed, if the same condition is false
> 3. Use else if to specify a new condition to test, if the first condition is false
> 4. Use switch to specify many alternative blocks of code to be executed

- if Statement

```
var a = 50;
```

```
var b = 60;
// a > b true or false
if (a > b) {
console.log("A is greater than B");
}
if (a < b) {
console.log("B is greater than A");
}
```

- else Statement

```
var a = 60;
var b = 40;
// a > b true or false
if (a > b) {
console.log("A is grater then B");
} else {
console.log("B is grater then A");
}
```

- else if Statement

```
var a = 20;
var b = 20;
// a > b true or false
if (a > b) {
console.log("A is greater than B");
} else if (a < b) {
console.log("B is greater than A");
} else {
console.log("The both are same");
}
```

- Switch Statement

```
var date = new Date();
var today = date.getDay();
// 0 - Sunday, 1 - Manday, 2 - Tuesday
switch (today) {
```

```
case 0:
console.log("Today is Sunday");
break;
case 1:
console.log("Today is Monday");
break;
case 2:
console.log("Today is Tuesday");
break;
case 3:
console.log("Today is Wednesday");
break;
case 4:
console.log("Today is Thursday");
break;
case 5:
console.log("Today is Friday");
break;
case 6:
console.log("Today is Saturday");
break;
default:
console.log("Not a Valid Number");
}
```

Declarations

<small>Top</small>
<small>Next</small>

> var // Declares a variable, optionally initializing it to a value.

> let // Declares a block scope local variable, optionally initializing it to a value.

> const // Declares a read-only named constant.

Loops and iteration

<small>Top</small>
<small>Next</small>

> Loops are handy, if you want to run the same code over and over again, each time with a different value. JavaScript supports different kinds of loops.

1. for - loops through a block of code a number of times
2. for/in - loops through the properties of an object

3. for/of - loops through the values of an iterable object
4. while - loops through a block of code while a specified condition is true
5. do/while - also loops through a block of code while a specified condition is true

- For loop

> Syntax:for (initializer; condition; increment) {
> // Statements
> }

```

for (var i = 1; i <= 50; i++) {
console.log(i + " Md. Nazmul islam");
}

```

- While Loop

> while (condition) {
> // code block to be executed
> }

```

var i = 0;
while (i <= 10) {
console.log(i + " Md. Nazmul islam");
i++;
}

```

- Do While Loop

> do {
> // code block to be executed
> } while (condition);

```

var isRanning = false;
do {
console.log("I am Ranning");
} while (isRanning);
console.log("I am Running");

```

- Nested Loop

```

var n = 5;
for (var i = 1; i <= n; i++) {
result = "";
for (var j = 1; j <= i; j++) {
result += j + " ";
}
console.log(result);
}

```

- Break Statement

```

while (true) {
var rand = Math.floor(Math.random() \* 10 + 1);
if (rand == 9) {
console.log("Winner Winner");
break;
} else {
console.log("You have got " + rand);
}
}

```

- Continue Statement

```

for (var i = 1; i <= 10; i++) {
if (i == 3 || i == 6) {
continue;
} else {
console.log(i);
}
}

```

- Infinity for Loop

```

for (;;) {
```

```
var rand = Math.floor(Math.random() \* 10 + 1);
if (rand == 9) {
console.log("You have got " + rand);
break;
} else {
console.log("You have got " + rand);
}
}
```

JavaScript Comment

<small>Top</small>
<small>Next</small>

> JavaScript comments can be used to explain JavaScript code, and to make it more readable. JavaScript comments can also be used to prevent execution, when testing alternative code. Any text in will be ignored by JavaScript (will not be executed).

1. Single Line Comments - Single line comments start with //.
1. Multi-line Comments - Multi-line comments start with /* and end with */.

Math Object

<small>Top</small>
<small>Next</small>

> The JavaScript Math object allows you to perform mathematical tasks on numbers.

Static methods

- Math.abs() `The Math.abs() function returns the absolute value of a number.`

```
Math.abs('-1');     // 1
Math.abs(-2);       // 2
Math.abs(null);     // 0
Math.abs('');       // 0
Math.abs([]);       // 0
Math.abs([2]);      // 2
Math.abs([1,2]);    // NaN
Math.abs({});       // NaN
Math.abs('string'); // NaN
Math.abs();         // NaN
```

```

- Math.floor() `The Math.floor() function returns the largest integer less than or equal to a given number.`

```

Math.floor(45.95); // 45
Math.floor(45.05); // 45
Math.floor(4   ); //   4
Math.floor(-45.05); // -46
Math.floor(-45.95); // -46

```

- Math.ceil() `The Math.ceil() method rounds a number UPWARDS to the nearest integer, and returns the result.`

```

Math.ceil(.95);   // 1
Math.ceil(4);   // 4
Math.ceil(7.004); // 8
Math.ceil(-0.95); // -0
Math.ceil(-4);   // -4
Math.ceil(-7.004); // -7

```

- Math.round() `The Math.round() function returns the value of a number rounded to the nearest integer.`

```

Math.round(20.49); // 20
Math.round(20.5); // 21
Math.round(42   ); // 42
Math.round(-20.5); // -20
Math.round(-20.51); // -21

```

- Math.max() `The Math.max() method returns the number with the highest value.`

```

Math.max(10, 20);   // 20
Math.max(-10, -20); // -10
Math.max(-10, 20); // 20
```
```

```

- Math.min() `The Math.min() returns the number with the lowest value.`

```

Math.min(10, 5); // 5

```

- Math.pow() `The Math.pow() static method, given two arguments, base and exponent, returns` base<sup>exponent</sup>.

```

// simple
Math.pow(7, 3);   // 343
// fractional exponents
Math.pow(2, 0.5); // 1.4142135623730951 (square root of 2)
// signed exponents
Math.pow(7, -2);   // 0.02040816326530612 (1/49)
// signed bases
Math.pow(-7, 2);   // 49 (squares are positive)
Math.pow(-7, 3);   // -343 (cubes can be negative)

```

- Math.sqrt() `The Math.sqrt() method returns the square root of a number.`

```

Math.sqrt(9); // 3
Math.sqrt(2); // 1.414213562373095

```

- Math.random() `Math.random() returns a random number between 0 (inclusive), and 1 (exclusive)`

```

function getRandomInt(max) {
  return Math.floor(Math.random() * max);
}

console.log(getRandomInt(3));
// expected output: 0, 1 or 2

```
```

- Math.round() `The Math.round() method rounds a number to the nearest integer.`

```

Math.round( 20.49); //  20
Math.round( 20.5 ); //  21
Math.round( 42   ); //  42
Math.round(-20.5 ); // -20
Math.round(-20.51); // -21

```

Date Object

<small>Top</small>
<small>Next</small>

> Date objects are created with the new Date() constructor. Date objects represent a single moment in time in a platform-independent format.

Date Function

- new Date() `new Date() creates a new date object with the current date and time`
- date.toDateString() `Returns the "date" portion of the Date as a human-readable string like 'Thu Apr 12 2018'.`
- date.toTimeString() `Returns the "time" portion of the Date as a human-readable string.`
- date.toLocaleString() `Returns a string with a locality-sensitive representation of the time portion of this date, based on system settings.`
- date.getFullYear() `Returns the year (4 digits for 4-digit years) of the specified date according to local time.`
- date.getMonth() `Returns the month (0–11) in the specified date according to local time.`
- date.getDate() `Returns the day of the month (1–31) for the specified date according to local time.`
- date.getHours() `Returns the hour (0–23) in the specified date according to local time.`
- date.getMinutes() `Returns the minutes (0–59) in the specified date according to local time.`
- date.getSeconds() `Returns the seconds (0–59) in the specified date according to local time.`
- date.getTime() `Returns the numeric value of the specified date as the number of milliseconds since January 1, 1970, 00:00:00 UTC. (Negative values are returned for prior times.)`
- date.getDay() `Returns the day of the week (0–6) for the specified date according to local time.`

String

<small>Top</small>
<small>Next</small>

> In the JavaScript programming language, anything wrapped between quotes is considered as string. It can be a single quote or double quote. Even by using a backtick sign, string can be declared in ES6. JavaScript provides two types of string functionality. A. string literal and B. string constructor.

- String Literals // var str = "Something";
- String Constructor // var str = String("Something");

Instance methods

[concat()](#concat) | [substr()](#substr) | [charAt()](#charAt)

concat()

`The concat() method joins two or more strings. This method does not change the existing strings and returns a new string.`

```

let text1 = "Hello";
let text2 = "world!";
let text3 = "Have a nice day!";
let result = text1.concat(" ", text2, " ", text3);

```

substr()

`The substr() method extracts a part of a string. This method begins at a specified position, and returns a specified number of characters and does not change the original string.`

```
substr(start, length)

const str = "Mozilla";

console.log(str.substr(1, 2));
// expected output: "oz"

console.log(str.substr(2));
// expected output: "zilla"

```

charAt()

`The charAt() method returns the character at a specified index (position) in a string.`

```

charAt(index)
let text = "HELLO WORLD";
let letter = text.charAt(1); // E

```

- startsWith() `The startsWith() method returns true if a string starts with a specified string. Otherwise it returns false. This method is case sensitive.`

```
startsWith(searchString)
startsWith(searchString, position)

const str1 = "Saturday night plans";
console.log(str1.startsWith("Sat")); // expected output: true
console.log(str1.startsWith("Sat", 3)); // expected output: false

```

- endsWith() `The endsWith() method returns true if a string ends with a specified string.Otherwise it returns false.The endsWith() method is case sensitive.`

```

endsWith(searchString)
endsWith(searchString, length)
let str = 'To be, or not to be, that is the question.'
console.log(str.endsWith('question.'))  // true
console.log(str.endsWith('to be'))      // false
console.log(str.endsWith('to be', 19))  // true

```

- toUpperCase() `The toUpperCase() method converts a string to uppercase letters.`

```

const sentence = 'The quick brown fox jumps over the lazy dog.';
```

```
console.log(sentence.toUpperCase()); // expected output: "THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG."
```

- toLowerCase() `The toUpperCase() method does not change the original string.`

```
const sentence = "The quick brown fox jumps over the lazy dog.";
console.log(sentence.toLowerCase()); // expected output: "the quick brown fox jumps over the lazy dog."
```

- trim() `The trim() method removes whitespace from both sides of a string. This method does not change the original string.`

```
var orig = "   foo  ";
console.log(orig.trim()); // 'foo'
```

- split() `The split() method splits a string into an array of substrings. This method returns the new array. This method does not change the original string.`

```
split()
split(separator)
split(separator, limit)
const str = "The quick brown fox jumps over the lazy dog.";
const words = str.split(" ");
console.log(words[3]); // expected output: "fox"
```

- includes() `The includes() method returns true if a string contains a specified string.Otherwise it returns false. This method is case sensitive.`

```
includes(searchElement)
includes(searchElement, fromIndex)
let text = "Hello world, welcome to the universe.";
let result = text.includes("world", 12);
```

```

- indexOf() `The indexOf() method returns the position of the first occurrence of a value in a string. This method returns -1 if the value is not found. This method is case sensitive.`

```

let text = "Hello world, welcome to the universe.";
let result = text.indexOf("welcome");

```

- match() `The match() method matches a string against a regular expression. This method returns an array with the matches. This method returns null if no match is found.`

```

match(regexp);
const paragraph = "The quick brown fox jumps over the lazy dog. It barked.";
const regex = /[A-Z]/g;
const found = paragraph.match(regex);
console.log(found); // expected output: Array ["T", "I"]

```

- replace() `The replace() method searches a string for a value or a regular expression. This method returns a new string with the value(s) replaced. This method does not change the original string.`

```

replace(regexp, newSubstr);
replace(regexp, replacerFunction);
const p ="The quick brown fox jumps over the lazy dog. If the dog reacted, was it really lazy?";
console.log(p.replace("dog", "monkey")); // expected output: "The quick brown fox jumps over the lazy monkey. If the dog reacted, was it really lazy?"


```

- search() `The search() method matches a string against a regular expression ** This method returns the index (position) of the first match.This method returns -1 if no match is found.`
```

```

search(regexp);
let str = "hey JudE";
let re = /[A-Z]/g;
console.log(str.search(re)); // returns 4, which is the index of
the first capital letter "J"

```

- slice() `The slice() method returns selected elements in an array, as a new array. The slice() method selects from a given start, up to a (not inclusive) given end. The slice() method does not change the original array.`

```

slice();
slice(start);
slice(start, end);
const animals = ["ant", "bison", "camel", "duck", "elephant"];
console.log(animals.slice(2)); // expected output: Array
["camel", "duck", "elephant"]
console.log(animals.slice(2, 4)); // expected output: Array
["camel", "duck"]


```

- split() `The split() method splits a string into an array of substrings. The split() method returns the new array. The split() method does not change the original string.`

```

split();
split(separator);
split(separator, limit);
const str = "The quick brown fox jumps over the lazy dog.";
const words = str.split(" ");
console.log(words[3]); // expected output: "fox"
const chars = str.split("");
console.log(chars[8]); // expected output: "k"

```

- toString() `The toString() method returns a string as a string. This method does not change the original string. The toString() method can be used to convert a string object into a string.`

- str.length `The length property returns the length of a string. The length property of an empty string is 0.`

```

const str = 'Life, the universe and everything. Answer:';
console.log(str.length); // expected output:  42

```

Escape Notation

- \0 // null character
- \' // single quote
- \" // double quote
- \\ // backslash
- \n // newline
- \v // vertical tab
- \t // tab
- \b // backspace

String Comparison

Array

<small>Top</small>
<small>Next</small>

> An array is a special variable, which can hold more than one value. An array can hold many values under a single name, and you can access the values by referring to an index number. It is a common practice to declare arrays with the const keyword.

Array Declaration

- array literal // Using an array literal is the easiest way to create a JavaScript Array.
- Array Constructor // The Array() constructor is used to create Array objects.

Accessing Array Elements

```

let fruits = ['Apple', 'Banana']
console.log(fruits) // Apple Banana
let first = fruits[0] // Apple
let last = fruits[fruits.length - 1] // Banana

```

array traversing

> To traverse an array means to access each element (item) stored in the array so that the data can be checked or used as part of a process.

```

const arr = [1, 2, 3, 4, 5, 6, 7, 8, 9];
for (var i = 0; i < arr.length; i++) { // Regular For
  console.log(arr[i]);
}

arr.forEach(function (item, index, array) { // forEach
  console.log(item, index);
});

for (const x of arr) { // for/of
  console.log(x);
}

```

- ### Instance methods

- push()
- pop()
- shift()
- unshift()
- indexOf()
- splice()
- [...arr]
- slice()
- Array.from()
- join()
- fill()
- concat()
- Array.isArray()
- Array.from()
- entries()
- every()
- filter()
- find()
- forEach()
- includes()
- keys()
- map()
- reduce()
- reverse()
- some()
- sort()

- toString()
- values()

Multidimensional Array

```

var arr = [
[80, 85, 90, 87],
[81, 83, 70, 80],
[70, 80, 75, 65],
];
console.log(arr);

```

JS Object

<small>Top</small>
<small>Next</small>

> You have already learned that JavaScript variables are containers for data values. Objects are variables too. But objects can contain many values. The values are written as name:value pairs (name and value separated by a colon).

Object Literal vs Constructor
```
var obj = {
x: 10,
Y: 20,
};
obj.z = 30;
console.log(obj);

var obj2 = Object();
obj2.a = 40;
console.log(obj2);

var obj3 = new Object();
obj3.name = "Md. Nazmul islam";
console.log(obj3);
```
Accessing Object Properties
```
var point = {
a: 10,
b: 20,
c: 30,
};
console.log(point.a); // Dot Notation.
console.log(point["c"]); // array notation
```
Setting Object Properties
```
var point = {
```

```
a: 10,
b: 20,
c: 30,
};
point.a = 50; // update object value
point.d = 60; // add object property
point["e"] = 70;
console.log(point);
Remove Object Properties
var point = {
a: 10,
b: 20,
c: 30,
};
point.a = undefined;
console.log(point);
delete point.a;
console.log(point);
Comparing Two Objects
const obj = {
a: 10,
b: 20,
};
const obj2 = {
a: 10,
b: 20,
};
console.log(obj == obj2); // Not work for compared
console.log(JSON.stringify(obj) === JSON.stringify(obj2));
Iterate Object Properties
const obj = {
a: 10,
b: 20,
c: 30,
};
console.log("c" in obj);
for (var i in obj) {
console.log(i + ":" + obj[i]);
}

### Object Methods

const obj = {
a: 10,
b: 20,
c: 30,
};
console.log(Object.keys(obj));
console.log(Object.values(obj));
console.log(Object.entries(obj));
```

```
const obj2 = Object.assign({}, obj);
console.log(obj2);
```

Functions

<small>Top</small>
<small>Next</small>

> A JavaScript function is a block of code designed to perform a particular task. A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().

Input, Output and Processing

```
// define a Function
function functionName() {
console.log("I am a Function");
}
function add() {
var i = 10;
var z = 20;
console.log(i + z);
}
function sub() {
var i = 10;
var z = 20;
console.log(i - z);
}
// Call or Invoke a function
functionName();
add();
sub();
```

Arguments and Parameters

```
// Arguments and Parameters
function add(a, b) {
// a and b is parameters
var result = a + b;
console.log(result);
}
add(10, 10); // 10 and 20 is arguments
add("Md. ", "Nazmul islam"); // concat the text
var arr = [10, 20, 30, 40, 50];

function sumOfArr(arr) {
var sum = 0;
for (var i = 0; i < arr.length; i++) {
sum += arr[i];
}
console.log(sum);
}
sumOfArr(arr);
```

```
function test() {
// console.log(arguments);
for (var i = 0; i < arguments.length; i++) {
console.log(arguments[i]);
}
}
test(10, 20, 30);

function addAll() {
var sum = 0;
for (var i = 0; i < arguments.length; i++) {
sum += arguments[i];
}
console.log(sum);
}
addAll(1, 2, 3, 4, 5, 6);
Return Something from a Function
function addAll() {
var sum = 0;
for (var i = 0; i < arguments.length; i++) {
sum += arguments[i];
}
return sum;
}
var result = addAll(1, 2, 3, 4, 5, 6);
console.log(result);
console.log(addAll(1, 2, 3, 4, 5, 6, 7, 8, 9, 10));

function person(name, email) {
return {
name: name,
email: email,
};
console.log("don't run this statement"); // Not Run
}
var p1 = person("Md. Nazmul islam",
"developer.nazmulislam@gmail.com");
console.log(p1);
Function Expression
var addition = function (a, b) {
return a + b;
};
addition(20, 20);

setTimeout(function () {
console.log("I will call after 5 Second");
}, 5000);

var another = addition(20, 30);
console.log(another);
```

Inner Function

```
function something(greet, name) {
function sayHi() {
console.log(greet, name);
}
sayHi();
}
something("Good Morning", "Nazmul Islam");

function somthing2(greet, name) {
function getFirstName() {
if (name) {
return name.split(" ")[0];
} else {
return "";
}
}
var message = greet + " " + getFirstName();
console.log(message);
}
somthing2("Good Morning", "Nazmul Islam");
```

Function Scoping

Functional Programming

> Functional Programming is a programming paradigm where you mostly construct and structure your code using functions. These functions take input which is called as arguments then shows the output based on the inputs being taken which, given the same input always results in the same output.
> Three Main Terms of Functional Programming.

Pure Function
First Class Function
Higher order function
Pure Functions
Pure Function is a function (a block of code) that always returns the same result if the same arguments are passed. It does not depend on any state, or data change during a program's execution rather it only depends on its input arguments.

```
function sqr(n) {
return n \* n;
}
console.log(sqr(5));
console.log(sqr(5));
```

First Class Function
A programming language is said to have First-class functions when functions in that language are treated like any other variable. For example, in such a language, a function can be

passed as an argument to other functions, can be returned by another function and can be assigned as a value to a variable.

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

A function can be Stored in a variable.

```
var sum = add;
console.log(sum(5, 5));
```

A function can be stored in an Array.

```
var arr = [];
arr.push(add);
console.log(arr[0](5, 10));
```

A function can be stored in an Object.

```
var obj = {
sum: add,
};
console.log(obj.sum(7, 10));
```

Higher Order Function in JavaScript

We can create functions as needed.

We can pass functions as arguments.

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

function mainpulate(a, b, func) {
var c = a + b;
var d = a - b;
return function () {
var m = add(a, b);
return c _ d _ m;
};
}
var multiply = mainpulate(10, 5, add);
console.log(multiply());
```

Closure

A closure is the combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment). ... In JavaScript, closures are created every time a function is created, at function creation time.

```
var a = 10;
function b() {
var x = 20;
return function () {
console.log(x);
};
}
var abc = b();
console.dir(abc);
```

Callback Function

> A callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action. ... A good example is the callback functions executed inside a . then() block chained onto the end of a promise after that promise fulfills or rejects.

```
function sample(a, b, cb) {
var c = a + b;
var d = a - b;
var result = cb(c, d);
return result;
}

function sum(a, b) {
return a + b;
}

var result = sample(5, 8, sum);
console.log(result);

var result2 = sample(5, 8, function (c, d) {
return c - d;
});

console.log(result2);

var result3 = sample(5, 8, function (c, d) {
return c \* d;
});
console.log(result3);
Foreach Implementation
var arr = [1, 2, 3, 4, 5];

var sum = 0;
// JavaScript built-in method
arr.forEach(function (value, index, arr) {
console.log(value, index, arr);
sum += value;
});
console.log(sum);

// Create For Implementation
function forEach(arr, cb) {
for (var i = 0; i < arr.length; i++) {
cb(arr[i], i, arr);
}
}

var sum = 0;
forEach(arr, function (value, index, arr) {
```

```
console.log(value, index, arr);
sum += value;
});
console.log(sum);

forEach(arr, function (value, index, arr) {
arr[index] = value + 5;
});
console.log(arr);
Map Function
var arr = [1, 2, 3, 4, 5];

### JavaScript built-in method

var sqrArr = arr.map(function (value) {
// return Math.random() _ 100
return value _ value;
});
console.log(arr);
console.log(sqrArr);

// Create For Implementation
function myMap(arr, cb) {
var newArr = [];
for (var i = 0; i < arr.length; i++) {
var temp = cb(arr[i], i, arr);
newArr.push(temp);
}
return newArr;
}

var qb = myMap(arr, function (value) {
return value _ value _ value;
});

var mten = myMap(arr, function (value) {
return value \* 10;
});

console.log(arr);
console.log(qb);
console.log(mten);
Filter Function
var arr = [4, 8, 1, 3, 5, 6, 4, 3, 9];

// JavaScript built-in method
var filteredArr = arr.filter(function (value) {
return value > 4;
});
console.log(filteredArr);
```

```
// Create For Implementation
function myFilter(arr, cb) {
var newArr = [];
for (var i = 0; i < arr.length; i++) {
if (cb(arr[i], i, arr)) {
newArr.push(arr[i]);
}
}

return newArr;
}

console.log(
myFilter(arr, function (value) {
return value % 2 === 1;
})
);
console.log(
myFilter(arr, function (value) {
return value > 4;
})
);
Reduce Function
var arr = [67, 1, 2, 45, 3, 4, 5];

// JavaScript built-in method
var sum = arr.reduce(function (prev, curr) {
return prev + curr;
}, 100);

var max = arr.reduce(function (prev, curr) {
return Math.max(prev, curr);
}, 0);
console.log(sum);
console.log(max);

// Create For Implementation
function myReduce(arr, cb, acc) {
for (var i = 0; i < arr.length; i++) {
acc = cb(acc, arr[i]);
}
return acc;
}

var sum = myReduce(
arr,
function (prev, curr) {
return prev + curr;
},
0
);
```

```
var max = myReduce(
arr,
function (prev, curr) {
return Math.max(prev, curr);
},
0
);

var min = myReduce(
arr,
function (prev, curr) {
return Math.min(prev, curr);
},
arr[0]
);

console.log(sum, max, min);
Find and FindIndex
var arr = [7, 4, 8, 6, 9, 2, 1, 70, 3];

// JavaScript built-in method
var result = arr.find(function (value) {
return value === 9;
});
console.log(result);

var result2 = arr.findIndex(function (value) {
return value === 9;
});
console.log(result2);

// Create For Implementation
function myFind(arr, cb) {
for (var i = 0; i < arr.length; i++) {
if (cb(arr[i])) {
// return arr[i]
return i;
}
}
}

var result3 = myFind(arr, function (value) {
return value === 9;
});

console.log(result3);
Sort, Some and Every Function
var persons = [
{
name: "A",
```

```
age: 24,
},
{
name: "B",
age: 19,
},
{
name: "C",
age: 26,
},
{
name: "D",
age: 21,
},
];

var arr = [4, 8, 1, 6, 7, -9, 0, -4, 3, 5, 6, 8, 2, 1, 7, 1];

arr.sort();
console.log(arr);

persons.sort();
console.log(persons);

arr.sort(function (a, b) {
if (a > b) {
return -1;
} else if (a < b) {
return 1;
} else {
return 0;
}
});
console.log(arr);

persons.sort(function (a, b) {
if (a.age > b.age) {
return 1;
} else if (a.age < b.age) {
return -1;
} else {
return 0;
}
});
console.log(persons);

// JavaScript Every Method
var res1 = arr.every(function (value) {
return value % 2 === 0;
});
console.log(res1);
```

```
var res2 = arr.every(function (value) {
return value >= 0;
});
console.log(res2);

// JavaScript some Method
var res3 = arr.some(function (value) {
return value % 2 === 1;
});
console.log(res3);

var res4 = arr.some(function (value) {
return value < 0;
});
console.log(res4);
Return Function
function greet(msg) {
function greetings(name) {
return msg + ", " + name + "!";
}

return greetings;
}

var gm = greet("Good Morning");
var gn = greet("Good Night");
var hello = greet("Hello");
// console.log(typeof gm)
var msg = gm("HM Nayem");
console.log(gn("Twinkle Cats"));
console.log(msg);
console.log(hello("Shegufa Taranjum"));

function base(b) {
return function (n) {
var result = 1;
for (var i = 0; i < b; i++) {
result \*= n;
}
return result;
};
}

var base10 = base(10);
console.log(base10(2));

var base5 = base(5);
console.log(base5(2));
console.log(base5(3));
console.log(base5(5));
```

```
Recursive Function
function sayHi(n) {
if (n === 0) {
return;
}
console.log("Hi, I am Calling");
sayHi(n - 1);
}
sayHi(10);

function sum(n) {
if (n === 1) {
return 1;
}
return n + sum(n - 1);
}
console.log(sum(5));

function fact(n) {
if (n === 1) {
return 1;
}

return n \* fact(n - 1);
}
console.log(fact(4));

var arr = [1, 2, 3, 4, 5];

function sumOfArray(arr, lastIndex) {
if (lastIndex < 0) {
return 0;
}
return arr[lastIndex] + sumOfArray(arr, lastIndex - 1);
}
console.log(sumOfArray(arr, arr.length - 1));

Currying
function add(a, b, c) {
return a + b + c;
}
add(41, 52, 63);

function currying(a) {
return function (b) {
return function (c) {
return a + b + c;
};
};
}
```

```
var result = currying(5)(10)(15);
console.log(result);
Function Composition
function print(inp) {
console.log(inp);
}

function multiplyByFive(n) {
return n \* 5;
}

function add(a, b) {
return a + b;
}

print(multiplyByFive(add(3, 5)));
Chapter: 10 Scope And Closure
Standard built-in objects

### Object

<small><a href="#javascript-tutorial">Top</a></small>
<small><a href="#function">Next</a></small>

> Object Literal, Object Constructor, Accessing Object
Properties width (.) notation or array notation, update/Change
object value, add object property, Remove/Delete Object
Properties, Comparing Two Objects by converting, Iterate Object
Properties('x' in obj), for in loop.

### Objects Method:

> Objects.keys(), Objects.values(), Objects.entries(),
Objects.assign({}, obj).

### Function

<small><a href="#javascript-tutorial">Top</a></small>
<small><a href="#checklist">Next</a></small>

> function, function\*, async function, return, class.

> Function: Input-Output-Processing, define a Function,
Invoke/Call a function, Arguments, parameters, sum of array, use
of arguments Objects, Return, function expression, inner
function, Function Scope.

> Pure Function, First Class Function, Higher Order Function,
Callback Function, forEach, map, Filter, reduce, find,
findIndex, Sort, Some, Every, return, Recursive , Currying.
```

JavaScript ES6

> A quick overview of new JavaScript features in ES2015, ES2016, ES2017, ES2018 and beyond.

let and const

<small>Top</small>
<small>Next</small>

- let

  > The let keyword was introduced in ES6 (2015). Variables defined with let cannot be Redeclared. It must be Declared before use. Its have a Block Scope.

```

let a = 3;
if (true) {
  let a = 5;
  console.log(a); // prints 5
}
console.log(a); // prints 3

```

- const

> Constants work just like let, but can’t be reassigned.

```

const a = 50;
a = 20; // throws an error

```

Template and Multiline string

<small>Top</small>
<small>Next</small>

> Template string use back-ticks (``) rather than the quotes ("") to define a string. Its allows multiline, expressions and interpolate variables in strings.

```

const name = "Nazmul";
let price = 10;
```

```
let VAT = 0.25;

const str = `
hello
world
`; // Use multiline
let total = `Total: ${(price * (1 + VAT)).toFixed(2)}`; // Use
expressions
const message = `Hello ${name}`; // Use Interpolation
```

Destructuring

<small>Top</small>
<small>Next</small>

- array

```
let [a, b] = [20, 27];
console.log(a); // 20
console.log(b); // 27
```

- Object

```
const obj = { a: 20, b: 27 };
const { a, b } = obj;
```

- Nested Objects

```
const person = {
  name: "John Snow",
  age: 29,
  sex: "male",
  materialStatus: "single",
  address: {
    country: "Westeros",
    state: "The Crownlands",
    city: "Kings Landing",
    pinCode: "500014",
  },
```

```
};

const {
  address: { state, pinCode },
  name,
} = person;

console.log(name, state, pinCode); // John Snow The Crownlands
500014
console.log(city); // ReferenceError

```

Optional chaining

<small>Top</small>
<small><a href="#object-property-and-function-
assignement">Next</small>

```

const adventurer = {
  name: "Alice",
  cat: {
    name: "Dinah",
  },
};
const dogName = adventurer.dog?.name;
console.log(dogName); // expected output: undefined
console.log(adventurer.someNonExistentMethod?.()); // expected
output: undefined

```

object property and function assignement

<small>Top</small>
<small>Next</small>

- Property assignement

```

const a = 2;
const b = 5;
const obj = { a, b };
console.log(obj); // expected output: { a: 2, b: 5 }

```

- function assignement

```

const obj = {
  a: 10,
  b() {
    console.log("b");
  },
};
obj.b(); // expected output: "b"

```

Spread and Rest operator

<small>Top</small>
<small>Next</small>

- Array

```

let colors = ["red", "green", "blue"];
let rgb = [...colors];
console.log(rgb); // expected output: [ 'red', 'green', 'blue' ]

function sum(x, y, z) {
  return x + y + z;
}
const numbers = [1, 2, 3];
console.log(sum(...numbers)); // expected output: 6

```

- Object

```

const obj1 = { x: 1, y: 2 };
const obj2 = { z: 3 };
// add members obj1 and obj2  to obj3
const obj3 = { ...obj1, ...obj2 };
console.log(obj3); // {x: 1, y: 2, z: 3}

```

- Rest

```

function myBio(firstName, lastName, ...otherInfo) {
```

```
  return otherInfo;
}

let func = function (...args) {
  console.log(args);
};
func(3); // [3]
func(4, 5, 6); // [4, 5, 6]

```

Exponent operator

<small>Top</small>
<small>Next</small>

```

const byte = 2 ** 4;
// Same as: Math.pow(2, 4)

```

Default Value and parameter

<small>Top</small>
<small>Next</small>

- Value

```

const scores = [22, 33];
const [math = 50, sci = 50, arts = 50] = scores; // math === 22,
sci === 33, arts === 50

```

- Parameter

```

function print(a = 5) {
  console.log(a);
}
print(); // prints 5
print(22); // prints 22

```

arrow Function

<small>Top</small>
<small>Next</small>

- Implicit return

```

const sum = (a, b) => a + b;
console.log(sum(2, 6)); // prints 8

```

- no arguments

```

const birthday = () => "Happy Birthday";

```

- With one arguments

```

const birthday = (name) => {
  return "Happy Birthday," + name + "!";
};

```

Method

<small>Top</small>
<small>Next</small>

[map ()](#map) | [forEch()](#forech) | [filter()](#filter) | [find()](#find) | [async-await](#async-await) | [for of](#for-of) | [repeat()](#repeat) | [includes()](#includes) | [startsWith()](#startswith) | [padStart()](#padstart) | [padEnd()](#padend) | [Object.assign()](#object-assign) | [Object.entries()](#object-entries) | [Object.values()](#object-values) | [set ()](#set)

map()

<small>Top</small>
<small>Method</small>
<small>Next</small>

> map() Method creates a new array from calling a function for every array element. its does not execute the function for empty elements. Its does not change the original array.

```

map((element) => {
  /* ... */
});
map((element, index) => {
  /* ... */
});
map((element, index, array) => {
  /* ... */
});
const array = [1, 4, 9, 16];
const map = array.map((x) => x * 2);
console.log(map); // expected output: Array [2, 8, 18, 32]

```

forEch()

<small>Top</small>
<small>Method</small>
<small>Next</small>

> forEch() method calls a function for each element in an array. its not executed for empty elements and does not return anything.

```

forEach((element) => {
  /* ... */
});
forEach((element, index) => {
  /* ... */
});
forEach((element, index, array) => {
  /* ... */
});
const array = ["a", "b", "c"];
array.forEach((element) => console.log(element));

```

filter()

<small>Top</small>
<small>Method</small>

<small>Next</small>

> filter() method creates a new array filled with elements that pass a test provided by a function. The filter() method does not change the original array.

```

filter((element) => {
  /* ... */
});
filter((element, index) => {
  /* ... */
});
filter((element, index, array) => {
  /* ... */
});
const words = ["spray", "limit", "elite", "exuberant"];
const result = words.filter((word) => word.length > 6);
console.log(result);

```

find()

<small>Top</small>
<small>Method</small>
<small>Next</small>

> find() method returns the value of the first element that passes a test. The find() method does not change the original array.

```

find((element) => {
  /* ... */
});
find((element, index) => {
  /* ... */
});
find((element, index, array) => {
  /* ... */
});
const array = [5, 12, 8, 130, 44];
const found = array.find((element) => element > 10);
console.log(found); // expected output: 12

```

Async-await

<small>Top</small>
<small>Method</small>
<small>Next</small>

> async await: The keyword async before a function makes the function return a promise. The keyword await before a function makes the function wait for a promise.

```

async function run() {
  const user = await getUser();
  const tweets = await getTweets(user);
  return [user, tweets];
}

```

for of

<small>Top</small>
<small>Method</small>
<small>Next</small>

> for of loop: loops through the values of an iterable object

```

const array1 = ["a", "b", "c"];
for (const element of array1) {
  console.log(element);
}

```

repeat()

<small>Top</small>
<small>Method</small>
<small>Next</small>

> The repeat() method returns a new string with a number of copies of a string. Its method does not change the original string.

```

let text = "Hello world!";
let result = text.repeat(2);
```

```

### includes()

<small><a href="#javascript-tutorial">Top</a></small>
<small><a href="#method">Method</a></small>
<small><a href="#startswith">Next</a></small>

> The includes() method returns true if a string contains a specified string. Otherwise it returns false. Its method is case sensitive.

```

```
const array = [1, 2, 3];
console.log(array.includes(2)); // expected output: true
console.log(array.includes(5)); // expected output: false

const str = "hellow this is me";
console.log(str.includes("s")); // expected output: true
console.log(str.includes("a")); // expected output: false
```

startsWith()

<small>Top</small>
<small>Method</small>
<small>Next</small>

> startsWith() method returns true if a string starts with a specified string. Otherwise it returns false. This method is case sensitive.

```

```
const str = "Saturday night plans";
console.log(str.startsWith("Sat")); // expected output: true
console.log(str.startsWith("Sat", 3)); // expected output: false
```

### padStart()

<small><a href="#javascript-tutorial">Top</a></small>
<small><a href="#method">Method</a></small>
<small><a href="#padend">Next</a></small>

> The padStart() method pads the current string with another string (multiple times, if needed) in starting position of string.
```

```

const str = "5";
console.log(str.padStart(2, "0")); // expected output: "05"

let number = "01555555540";
let lastDigits = number.slice(-2);
lastDigits.padStart(number.length - 2, "*"); // expected
output:'*******40'

```

padEnd()

<small>Top</small>
<small>Method</small>
<small>Next</small>

> padEnd() method pads the current string with another string
(multiple times, if needed) in ending position of string.

```

const str = "5";
console.log(str.padEnd(3, ".")); // expected output: "5.."

```

Object.assign()

Object.entries()

Object.values()

set ()

Arrow function
const sum = (a, b) => a + b;
console.log(sum(2, 6)); // prints 8
let test = (a, b) => {
let result = a + b;
return result;
};
console.log(test(50, 55));
let sum = a => a * a;
console.log(sum(5));
Default parameters
function print(a = 5) {
console.log(a);
}

```
print(); // prints 5
print(22); // prints 22
Let scope
let a = 3;

if (true) {
let a = 5;
console.log(a); // prints 5
}
console.log(a); // prints 3
Const
// can be assigned only once:
const a = 55;
a = 44; // throws an error
Multiline string
console.log(` This is a multiline string`);
Template strings
const name = "Leon";
const message = `Hello ${name}`;
console.log(message); // prints "Hello Leon"
String includes ()
console.log("apple".includes("pl")); // prints true
console.log("apple".includes("tt")); // prints false
String starts With ()
console.log("apple".startsWith("ap")); // prints true
console.log("apple".startsWith("bb")); // prints false
String repeat ()
var age = 23;
var name = "Md. Nazmul islam";
console.log("ab".repeat(3)); // prints "abababab"
console.log(name.padStart(15, "\*"));
console.log(name.padEnd(15, "a"));
Destructuring array
let [a, b] = [3, 7];
console.log(a); // 3
console.log(b); // 7
Destructuring object
let obj = {
a: 55,
b: 44,
address: {
city: Mymensingh,
country: Bangladesh,
},
};

let { a, b, address:{city, country} } = obj;

console.log(a); // 55
console.log(b); // 44
console.log(city, country); // Mymensingh
```

```
function sum(...rest) {
return rest.reduce((a, b) => a + b);
}
Object property assignment
const a = 2;
const b = 5;
const obj = { a, b };

// Before es6:
// obj = { a: a, b: b }
console.log(obj); // prints { a: 2, b: 5 }
Object function assignment
const obj = {
a: 5,
b() {
console.log("b");
},
};
obj.b(); // prints "b"
Object. Assign ()
const obj1 = { a: 1 };
const obj2 = { b: 2 };

const obj3 = Object.assign({}, obj1, obj2);
console.log(obj3); // { a: 1, b: 2 }
Object. Entries ()
const obj = {
firstName: "Vipul",
lastName: "Rawat",
age: 22,
country: "India",
};

const entries = Object.entries(obj);
/_ returns an array of [key, value]
pairs of the object passed
_/
console.log(entries);
/_ prints
[
['firstName', 'Vipul'],
['lastName', 'Rawat'],
['age', 22],
['country', 'India']
];
_/
Spread & Rest operator
const a = [1, 2];
const b = [3, 4];
const c = [...a, ...b];
```

```
console.log(c); // [1, 2, 3, 4]

const a = {
firstName: "Barry",
lastName: "Manilow",
};
const b = {
...a,
lastName: "White",
canSing: true,
};

console.log(a); // {firstName: "Barry", lastName: "Manilow"}
console.log(b); // {firstName: "Barry", lastName: "White",
canSing: true}

function sum(...rest) {
return rest.reduce((a, b) => a + b);
}
console.log(sum(1, 2, 3, 4, 5));

// great for modifying objects without side effects/affecting
the original
Destructuring Nested Objects
const Person = {
name: "John Snow",
age: 29,
sex: "male",
materialStatus: "single",
address: {
country: "Westeros",
state: "The Crownlands",
city: "Kings Landing",
pinCode: "500014",
},
};

const {
address: { state, pinCode },
name,
} = Person;

console.log(name, state, pinCode); // John Snow The Crownlands
500014
console.log(city); // ReferenceError
Exponent operator
const byte = 2 \*\* 8;

// Same as: Math.pow(2, 8)
For of loop
const array1 = ["a", "b", "c"];
```

```
for (const element of array1) {
console.log(element);
}
```

Set & Map

cÖvq mKj †cÖvMÖvwgs fvlvq A‡bK iK‡gi WvUv ÷ªvKPvi i‡q‡Q| wKš‘ Avcwb hw` Rvevw¯Œ‡Þi w`‡K ZvKvb Zvn‡j †`L‡eb GLv‡b GKwU gvÎ WvUv ÷ªvKPvi i‡q‡Q Avi Zv n‡jv array | wKš‘ ES6 Avgiv wKQz WvKv ÷vKPvi †c‡qwQ ‡hgb Set & Map GLb Avgiv GB `yBUv welq wb‡qB Av‡jvPbv Kiv n‡e|

JavaScript Set

Set n‡”Q GKwU Kb÷ªvUi dvskb| GLv‡b Avcwb NaN & Undifined mn †h †Kvb ai‡bi WvUv msiÿb Ki‡Z cvi‡eb| Z‡e GLv‡b KL‡bv WvUv Wewj‡KU Ki‡Z cvi‡eb bv| GLv‡b Avcwb WvUv¸‡jv‡K µgvbœ‡q mvRv‡Z cvi‡eb, ‡Kvb Ie‡R‡± WvUv Av‡Q wKbv †PK Ki‡Z cvi‡eb, Mo †ei Ki‡Z cvi‡eb| GQvovI Av‡iv A‡bK KvR Ki‡Z cvi‡eb|

Creates a new Set object.

new Set()GLv‡b GKwU Array cvm Kivi gva¨‡g bZzb †mU ^Zix Ki‡Z cvi‡eb|

add()‡g*‡W values cvm K‡iI bZzb †mU ^Zix Ki‡Z cvi‡eb|

add()‡g*‡W variables cvm K‡iI bZzb †mU ^Zix Ki‡Z cvi‡eb|

Set Methods

Set Gi mv‡_ eûj e¨eüZ wKQz †g_W:

`new Set()` // নতুন সেট তৈরি করতে।

`add()` // সেট এর ভিতর নতুন কোন উপাদান যোগ করতে।

`delete()` // সেট থেকে কোন উপাদান দূর করতে।

`has()` // সেট এ যদি উপাদান থাকে তাহলে `True` রিট্রান করবে।

`clear()` // সেট থেকে সকল উপাদান রিমোভ করতে।

`forEach()` // প্রতিটি উপাদানের জন্য একটি কলব্যাক আহ্বান করে।

`values()` // উপাদানের সকল মান সহ একটি ইটারেটর প্রদান করে।

`size()` // সেটে কতটি উপাদান আছে তা রিটার্ন করবে।

JavaScript Map

Map() G mKj cÖKvi WvUv msiÿY Kiv hvq| Z‡e WvK¸‡jv Aek¨B (Key-value) AvKv‡i †Rvovq †Rvovq n‡e| GwU bZzb A¨v‡i ^Zwi K‡i|

Creates a new Map object.

Map()GLv‡b GKwU Array cvm Kivi gva¨‡g bZzb map ^Zix Ki‡Z cvi‡eb|

Map.set()‡g_‡W values cvm K‡iI bZzb map ^Zix Ki‡Z cvi‡eb|

Map Methods

`new Map()` // নতুন ম্যাপ তৈরি করতে।

`set()` // নতুন কোন উপাদান যোগ করতে।

`clear()` // সকল উপাদান রিমোভ করতে।

`delete()` // কোন একটি উপাদান রিমোভ করতে।

`has()` // যদি উপাদান থাকে তাহলে `True` রিট্রান করবে।

`forEach()` // প্রতিটি `(Key-value)`-র জন্য একটি কলব্যাক আহ্বান করে।

`get()` // কোন উপাদান পেতে।

size // কতটি উপাদান আছে তা রিটার্ন করবে।

Dom

Dom definition

<small>Top</small>
<small>Next</small>

> DOM (Document Object Model)
> The Document Object Model (DOM) is the data representation of the objects that comprise the structure and content of a document on the web. This guide will introduce the DOM, look at how the DOM represents an HTML document in memory and how to use APIs to create web content and applications.
> It is just an API to interact with HTML documents and change it later on demand.

Element Selection using JavaScript

<small>Top</small>
<small>Next</small>

> .getElementsByTagName( ) | .getElementsByClassName( ) | .getElementById( ) | .querySelectorAll( ) | .querySelector( ) | .getElementsByName().

1. .getElementsByTagName( )

   > This method will return an array of all the elements you specify between the parentheses.

   ```
   const collection = document.getElementsByTagName("p");
   const collection = document.getElementsByTagName("*");
   ```

2. .getElementsByClassName( )

   > This method will returns a array of html elements that contain the class name you specify within the parentheses.

   ```

   const collection = document.getElementsByClassName("class");
   ```

3. .getElementById( )

   > This method will return a html element that contain the id name you specify within the parentheses.

   ```

   document.getElementById("id");

   ```

4. .querySelectorAll( )

   > This method returns all elements that matches a CSS selector(s) you specify within the parentheses.

   ```

   document.querySelectorAll("css selector like p .class #id");

   ```

5. .querySelector( )

   > This method returns the first element that matches a CSS selector(s) you specify within the parentheses.

   ```

   document.querySelector("css selector like p .class #id");

   ```

6. .getElementsByName()

   > This method returns a collection of elements with a specified name.

   ```

   const firstName = document.getElementsByName("fname");

   ```

Traversing DOM elements using JavaScript dynamically

<small>Top</small>
<small>Next</small>

> .parentElement | .children | .previousElementSibling | .nextElementSibling | .firstElementChild | .lastElementChild | Array.from() | Array.prototype.slice.apply() | [...array]

```

const listItem = document.getElementById("li");
const parent = listItem.parentElement; // Get parent Element
const children = listItem.children; // Get Childrens
const previousSibling = listItem.previousElementSibling; // Get Previous Sibling Element
const nextSibling = listItem.nextElementSibling; // Get Nest Sibling Element
const firstChiled = listItem.firstElementChild; // Get First Element of child
const lastChiled = listItem.lastElementChild; // Get Last Element of child

```

convert array like object to array

> Html Collection is an array like objects so we can convarte an array for Traversing. 3 way to convarte. Array.from() | Array.prototype.slice.apply() | [...array]

Array.from()

```

const listItems = Array.from(listItem);
listItems.forEach((li, index) => {
  let text = li.innerHTML;
  li.innerHTML = `(${index + 1}) ${text}`;
});


```

Array.prototype.slice.apply()

```

const listItems = Array.prototype.slice.apply(listItem);
listItems.forEach((li, index) => {
  let text = li.innerHTML;
  li.innerHTML = `(${index + 1}) ${text}`;
});

```

[...array]

```

const listItems = [...listItem];
listItems.forEach((li, index) => {
  let text = li.innerHTML;
  li.innerHTML = `(${index + 1}) ${text}`;
});

```

Create and append Elements using JavaScript Dynamically

<small>Top</small>
<small>Next</small>

> .createElement() | .appendChild() | .append() | .insertBefore() | .insertAdjacentElement |"afterbegin", p); | .insertAdjacentElement("beforebegin", p); | .insertAdjacentElement("afterend", p); | .insertAdjacentElement("beforeend", p);

- Create a \<p> element and append it to the document

```

const p = document.createElement("p");
p.innerText = "This is a paragraph";
document.body.appendChild(p);

```

- Create a \<p> element and append it to an element

```

const p = document.createElement("p");
p.innerHTML = "This is a paragraph.";
document.getElementById("myDIV").appendChild(p);

```

- a function for create elements Daynamicly

```

function createElement(tagName, className, innerHTML) {
  let tag = document.createElement(tagName);
  tag.classList = className | "";
```

```
  tag.innerHTML = innerHTML | "";
  return tag;
}
```

- a function for append elements

```
function append(parent, children) {
children.forEach((child) => parent.appendChild(child));
}
```

Remove Elements Or Others using JavaScript dynamically

<small>Top</small>
<small>Next</small>

> .remove() | .removeChild() | .replaceChild() | .removeAttribute() | .removeEventListener()

Add Events using JavaScript Dynamically

<small>Top</small>
<small>Next</small>

Difference between an Event Listener and Event Handler

<small>Top</small>
<small>Next</small>

Deferent Type of Event

<small>Top</small>
<small>Next</small>

Capturing and Bubbling

<small>Top</small>
<small>Next</small>

Add or remove element using event bubbling

<small>Top</small>
<small>Next</small>

Event Delegation

<small>Top</small>
<small>Next</small>

Add or Remove Classes using JavaScript dynamically

<small>Top</small>
<small>Next</small>

Bom

Window Object

<small>Top</small>
<small>Bom</small>
<small>Next</small>

> The JavaScript window object which is the global object of JavaScript in the browser and exposes the browsers functionality.

```

var counter = 1;
var showCounter = () => console.log(counter);

console.log(window.counter);
window.showCounter();

```

Window Object Methods

<small>Top</small>
<small>Bom</small>
<small>Next</small>

[open()](#open) | [resize()](#resize) | [moveTo()](#moveto) | [close()](#close)

open()

> To open a new window or tab, you use the window.open() method:

```

window.open(url, windowName, [windowFeatures]);
```

```
let url = 'http://localhost/js/about.html';
let newWindow = window.open(url,'about');
```

resize()

> To resize a window you use the resizeTo(), resizeBy(), method.

```
window.resizeTo(width,height);
window.resizeBy(deltaX,deltaY);

let newWindow = window.open(
    'http://localhost/js/about.html',
    'about',
    'height=600,width=800');

setTimeout(() => {
    newWindow.resizeTo(600, 300);
}, 3000);
```

moveTo()

> To move a window you use the moveTo(), moveBy(), method.

```
window.moveTo(x, y);
window.moveBy(x, y);

let newWindow = window.open(
    'http://localhost/js/about.html',
    'about',
    'height=600,width=600');

setTimeout(() => {
    newWindow.moveTo(500, 500);
}, 3000);
```

close()

> To close a window, you use the window.close() method.

```
```

```
window.close()

let newWindow = window.open(
    'http://localhost/js/about.html',
    'about',
    'height=600,width=600');

setTimeout(() => {
    newWindow.close();
}, 3000);
```

Alert

<small>Top</small>
<small>Bom</small>
<small>Next</small>

> The alert() method displays an alert box with a message and an OK button.

```
alert("Hello! I am an JavaScrip alert!!");
```

Confirm

<small>Top</small>
<small>Bom</small>
<small>Next</small>

> The confirm() method displays a dialog box with a message, an OK button, and a Cancel button.This method returns true if the user clicked "OK", otherwise false.

```
const response = confirm('are you going to picnic?')
console.log(response)
```

Prompt

<small>Top</small>
<small>Bom</small>
<small>Next</small>

> The prompt() method displays a dialog box that prompts the user for input. this method returns the input value if the user clicks "OK", otherwise it returns null.

```

const person = prompt("Please enter your name");
console.log(person);

```

setTimeout

<small>Top</small>
<small>Bom</small>
<small>Next</small>

> The setTimeout() sets a timer and executes a callback function after the timer expires.

```

setTimeout(() => {
  console.log("this is the first message");
}, 5000);

```

setInterval

<small>Top</small>
<small>Bom</small>
<small>Next</small>

> The setInterval() repeatedly calls a function with a fixed delay between each call.

```

setInterval(function () {
  element.innerHTML += "Hello";
}, 1000);

```

Location Object

<small>Top</small>
<small>Bom</small>
<small>Next</small>

> The location object can be used to get the current page address (URL) and to redirect the browser to a new page.

> reload(), assign(), hash, host, hostname, href, origin, pathname;

Redirect to a new URL

<small>Top</small>
<small>Bom</small>
<small>Next</small>

Navigator Object

<small>Top</small>
<small>Bom</small>
<small>Next</small>

Screen Object

<small>Top</small>
<small>Bom</small>
<small>Next</small>

History Object

<small>Top</small>
<small>Bom</small>
<small>Next</small>

> The history object contains the URLs visited by the user (in the browser window).

> back, forward, go, pushState

Web API

<small>Top</small>
<small>Web API</small>
<small>Next</small>

Client Storage

<small>Top</small>
<small>Web API</small>
<small>Next</small>

Cookies

<small>Top</small>
<small>Web API</small>

```
<small><a href="#localstorage">Next</a></small>

### localStorage

<small><a href="#javascript-tutorial">Top</a></small>
<small><a href="#web-api">Web API</a></small>
<small><a href="#sessionstorage">Next</a></small>

### sessionStorage

<small><a href="#javascript-tutorial">Top</a></small>
<small><a href="#web-api">Web API</a></small>
<small><a href="#indexeddb">Next</a></small>

### IndexedDB

<small><a href="#javascript-tutorial">Top</a></small>
<small><a href="#web-api">Web API</a></small>
<small><a href="#formdata">Next</a></small>

### FormData

<small><a href="#javascript-tutorial">Top</a></small>
<small><a href="#web-api">Web API</a></small>
<small><a href="#drag-and-drop-api">Next</a></small>

### Drag and Drop API

<small><a href="#javascript-tutorial">Top</a></small>
<small><a href="#web-api">Web API</a></small>
<small><a href="#filereader-api">Next</a></small>

### FileReader API

<small><a href="#javascript-tutorial">Top</a></small>
<small><a href="#web-api">Web API</a></small>
<small><a href="#geolocation-api">Next</a></small>

### Geolocation API

<small><a href="#javascript-tutorial">Top</a></small>
<small><a href="#web-api">Web API</a></small>
<small><a href="#notification-api">Next</a></small>

### Notification API

<small><a href="#javascript-tutorial">Top</a></small>
<small><a href="#web-api">Web API</a></small>
<small><a href="#canvas-api">Next</a></small>

### Canvas API
```

<small>Top</small>
<small>Web API</small>
<small>Next</small>

History API

<small>Top</small>
<small>Web API</small>
<small>Next</small>

History API

<small>Top</small>
<small>Web API</small>
<small>Next</small>

Problem Solving

[Solution](https://github.com/dev-nazmulislam/javaScript-problem-solving)
<small>Top</small>

1. Print in Console numbers from 1 to 10
2. Print in Console the odd numbers less than 100
3. Print in Console the multiplication table with 7
4. Calculate the sum of numbers from 1 to 10
5. Calculate 10!
6. Calculate the sum of odd numbers greater than 10 and less than 30
7. Calculate the sum of numbers in an array of numbers
8. Calculate the average of the numbers in an array of numbers
9. Find the maximum number in an array of numbers
10. Create a function that receives an array of numbers and returns an array containing only the positive numbers
11. Print in Console all the multiplication tables with numbers from 1 to 10
12. Create a function that will convert from Celsius to Fahrenheit
13. Create a function that will convert from Fahrenheit to Celsius
14. Print in Console the first 10 Fibonacci numbers without recursion
15. Create a function that will find the nth Fibonacci number using recursion
16. Check Leap Year Using if...else?
17. Create a function that accepts an array and returns the last item in the array.
18. Calculate the sum of digits of a positive integer number
19. Print in Console the first 100 prime numbers
20. Check Odd or Even Number with Arguments Objects

21. Create a function that will return in an array the first "nPrimes" prime numbers greater than a particular number "startAt"
22. Rotate an array to the left 1 position
23. Reverse an array with createed function, don`t Change orginal array #24: Reverse an array with JavaScript Builtin Method
24. Reverse a string with JavaScript Builtin Method
25. Reverse a string with createed function, don`t Change orginal string
26. Create a function that takes an array and returns the types of values (data types) in a new array.
27. Create a function that will return in an array. becarefull function parameter and array length is same in count.
28. Create a function that takes the age in years and returns the age in days.
29. Create a function that takes voltage and current and returns the calculated power.
30. Given two numbers, return true if the sum of both numbers is less than 50. Otherwise return false.
31. Write a function that takes minutes and converts it to seconds.
32. Write a function that converts hours into seconds.
33. Create a function that takes an array containing only numbers and return the first element.
34. Create a function that finds the maximum range of a triangle's third edge, where the side lengths are all integers.
35. Write a function that takes the base and height of a triangle and return its area.
36. Create a function that takes a number as an argument, increments the number by +1 and returns the result.
37. Create a function that takes a base number and an exponent number and returns the calculation.
38. Create a function that takes two numbers as arguments and return their sum.
39. Create a function that will merge two arrays and return the result as a new array
40. Create a function that will receive two arrays of numbers as arguments and return an array composed of all the numbers that are either in the first array or second array but not in both