JavaScript Cheat Sheet - A Basic Guide to JavaScript

Last Updated : 08 Mar, 2025

JavaScript is a lightweight, open, and cross-platform programming language. It is omnipresent in modern development and is used by programmers across the world to create dynamic and interactive web content like applications and browsers

- JavaScript (JS) is a versatile, high-level programming language primarily used for creating interactive and dynamic content on websites.
- It runs in browsers, enabling features like form validation, animations, and real-time updates.

This article provides an in-depth JavaScript Cheat Sheet, a must-have for every web developer.

JavaScript Chear Sheet

What is JavaScript Cheat Sheet?

A **JavaScript Cheat Sheet** is a concise reference guide that provides a quick overview of essential JavaScript concepts, syntax, and commands.

It is designed to help developers, especially beginners, recall important topics or features of the language without delving into detailed documentation.

Fundamentals

- To use JavaScript on a website, attach the JavaScript file to the HTML file. Enclose the code within the `<script>` tag for the browser to recognize and execute it:

```
<script type="text/javascript">
  // Your JavaScript code
</script>
```

- You can also use an external file:

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

- JavaScript comments help explain and improve code readability.
    - **Single-line comments:** Start a single-line comment with "//".
    - **Multi-line comments:** Wrap the comment in /* and*/ if it spans multiple lines.

Variables

Variables in JavaScript are containers for storing data. JavaScript allows the usage of variables in the following three ways

Variable	Description	Example
var	Used to initialize to value, redeclared and its value can be reassigned.	var x= value;
let	Similar to var but is block scoped	let y= value;
const	Used to declare a fixed value that cannot be changed.	const z= value;

```
console.log("Using var Keyword");
var x = 1;
if (x === 1) {
    var x = 2;
    console.log(x);
}
console.log(x);


// Using let keyword

console.log("Using let Keyword");
let x1 = 1;
if (x1 === 1) {
    let x1 = 2;
    console.log(x1);
}
console.log(x1);


// Using const keyword

console.log("Using const Keyword");
const number = 48;

// Changing const value will display TypeError
try {
    const number = 42;
} catch (err) {
    console.log(err);
}
console.log(number);
```

Output

```
Using var Keyword
2
2
Using let Keyword
2
1
```

```
Using const Keyword
48
```

- Use `let` for variables whose values will change.
- Use `const` for variables with constant values.
- Avoid using `var` in modern JavaScript due to potential scoping issues.

Datatypes

- In JavaScript, data types define the type of value a variable can hold.
- Understanding the types is crucial for evaluating expressions and ensuring the correct operations are performed on variables.
- JavaScript has both **primitive** and **non-primitive** data types:

Datatype	Description	Example
Number	Numeric values can be real number or integers.	var x= number;
String	Series of multiple characters written in quotes.	var x= "characters";
Boolean	Has only two values true or false.	var x= true/false;
Null	Special value that represents that the variable is empty.	var x= null;
Undefined	Represents a variable which is declared but not assigned any value.	let x; / let x= undefined;

Object	Complex data type that allows us to store a collection of data.	var x= {
		key: "value";
		key: "value";
		}

Datatype	Description	Example
Array	Stores multiple values of same type in a single variable.	var x =['y1', 'y2','y3','y4']; y: any datatype

Function	Functions are objects that can be called to execute a block of code.	function x(arguments){
		block of code
		}

```
// string
let s = "hello geeks";
console.log(s);

// Number
const n = 10;
console.log(n);

// Boolean
const x = "true";
console.log(x);

// Undefined

let name;
console.log(name);

// Null

const num = null;
console.log(num);

// Symbol
const val1 = Symbol("hello");
const val2 = Symbol("hello");
console.log(val1);
console.log(val2);

// Here both values are different
```

```
// as they are symbol type which
// is immutable object

const obj = {
    firstName: "geek",
    lastName: null,
    batch: 2,
};
console.log(obj);
```

Output

```
hello geeks
10
true
undefined
null
Symbol(hello)
Symbol(hello)
{ firstName: 'geek', lastName: null, batch: 2 }
```

- **Primitive types** (like `Number`, `String`, `Boolean`, `Null`, and `Undefined`) store single values.
- **Non-primitive types** (like `Object`, `Array`, and `Function`) can store collections of data or executable code.

Operators

- JavaScript operators are symbols used to perform various operations on variables (operands). Following are the different types of operators:

Operators	Description	Symbols
Arithmetic	Used to perform basic arithmetic operations on variables(operands).	+,-,*,/,%,++,--
Comparison	Comparison operator is used to compare two operands.	==, ===,!=,>,<,>=,<=

Operators	Description	Symbols
Bitwise	Used to perform bitwise operations.	&, | , ^,~,<<, >>, >>>
Logical	There are three logical operators in javascript. • logical AND: When all the operands are true. • logical OR: When one or more than one operands are true. • logical NOT: Converts true to false.	exp1&&exp2,exp1 ||exp2, !exp
Assignment	Assignment operators assign values to JavaScript variables.	=, +=,-=,*=,/=,%=

```
let x = 5;
let y = 3;

// Addition
console.log("x + y = ", x); // 8

// Subtraction
console.log("x - y = ", x - y); // 2

// Multiplication
console.log("x * y = ", x * y); // 15

// Division
console.log("x / y = ", x / y);

// Remainder
console.log("x % y = ", (x % y)); // 2

// Increment
console.log("++x = ", ++x); // x is now 6
console.log("x++ = ", x++);
console.log("x = ", x); // 7

// Decrement
console.log("--x = ", --x); // x is now 6
```

```
console.log("x-- = ", x--);
console.log("x = ", x); // 5

// Exponentiation
console.log("x ** y =", x ** y);

// Comparison
console.log(x > y); // true

// Equal operator
console.log((2 == 2)); // true

// Not equal operator
console.log((3 != 2)); // true

// Strict equal operator
console.log((2 === 2)); // true

// Strict not equal operator
console.log((2 !== 2)); // false

// Logical Operator

// Logical AND
console.log((x < 6 && y < 5)); // true

// Logical OR
console.log((x < 6 || y > 6)); // true

// Logical NOT
console.log(!(x < 6)); // false
```

Output

```
x + y =  5
x - y =  2
x * y =  15
x / y =  1.6666666666666667
x % y =  2
++x =  6
x++ =  6
x =  7
--x =  6
```

```
x-- =  6
x =  5
x ** y = 125
true
true
true
true
false
true
true
false
```

JS scope and scope chain

Scope determines where a variable is accessible in your program. It defines the context in which variables can be referenced or modified. JavaScript has three main types of scope:

- **Function Scope**: Variables declared inside a function are only accessible within that function.
- **Global Scope**: Variables declared outside any function are accessible throughout the program.
- **Block Scope**: Variables declared with `let` or `const` inside a block (e.g., loops or conditionals) are only accessible within that block.

Scope Chain:

- The **scope chain** helps the JavaScript engine resolve variable references by searching through the current scope and its outer scopes (from local to global).
- If a variable isn't found in the current scope, the engine looks in outer scopes until it either finds the variable or reaches the global scope.
- If it cannot find the variable, it either declares it in the global scope (non-strict mode) or throws an error (strict mode).

```
let a = 10;

function example() {
    let b = 20; // Exists in function scope
```

```
    if (true) {
        var c = 30; // Exists in function scope (due to 'var')
        const d = 40; // Exists in block scope
    }

    console.log(a); // Accessible from global scope
    console.log(b); // Accessible from function scope
    console.log(c); // Accessible from function scope (due to 'var')
    console.log(d); // Error: 'd' is not accessible outside block
scope
}

example();
```

Functions

- A JavaScript function is a block of code designed to perform a particular task.
- It is executed when invoked or called. Instead of writing the same piece of code again and again you can put it in a function and invoke the function when required.
- JavaScript function can be created using the functions keyword. Some of the functions in JavaScript are:

Function	Description
parseInt()	Parses an argument passed to it and returns an integral number.
parseFloat()	Parses the argument and returns a floating-point number.
isNaN()	Determines if a given value is Not a Number.
Number()	Returns an argument after converting it to number.
eval()	Used for evaluating JavaScript programs presented as strings.
prompt()	Creates a dialogue box for taking input from the user.
encodeURI()	Encodes a URI into a UTF-8 encoding scheme.

Function	Description
match()	Used to search a string for a match against regular expression.

```
// JS parseInt function:
const num1 = parseInt("100.45");
console.log('Using parseInt("100.45") = ' + num1);

// JS parseFloat function:
const num2 = parseFloat("123.45abc");
console.log('parseFloat("123.45abc") = ' + num2);

// JS isNaN function:
console.log(isNaN("hello"));

// JS Number() function:
const num3 = Number("123");
console.log("Value of '123': " + num3);

// JS eval() function:
function evalExample() {
    const expression = "2 + 3 * 4"; // Example expression to evaluate
    const result = eval(expression); // Evaluates '2 + 3 * 4' and retu
    console.log(result);
}
evalExample();

// JS encodeURI function:
const url = "https://example.com/hello world?query=javascript";
const encodedURL = encodeURI(url); // Encodes spaces as '%20' and ensu
URL.
console.log(encodedURL);
```

Output

```
Using parseInt("100.45") = 100
parseFloat("123.45abc") = 123.45
true
Value of '123': 123
14
https://example.com/hello%20world?query=javascript
```

- `parseInt()`: Converts the string to an integer by removing the decimal part.
- `parseFloat()`: Converts the string to a floating-point number until a non-numeric character is encountered.
- `isNaN()`: Checks whether the value is NaN (Not a Number).
- `Number()`: Converts a value to a number; for example, `Number("123")` results in `123`.
- `eval()`: Evaluates a JavaScript expression in string form and executes it. Be cautious with its use due to security risks.
- `encodeURI()`: Encodes a complete URL, converting special characters like spaces into their appropriate encoded forms.

Arrays

- In JavaScript, array is a single variable that is used to store different elements.
- It is often used when we want to store list of elements and access them by a single variable. Arrays use numbers as index to access its "elements".
- Declaration of an Array: There are basically two ways to declare an array.

Example:

```
var House = [ ]; // Method 1
var House = new Array(); // Method 2
```

There are various operations that can be performed on arrays using JavaScript methods. Some of these methods are:

Method	Description
push()	Adds a new element at the very end of an array.
pop()	Removes the last element of an array.
concat()	Joins various arrays into a single array.
shift()	Removes the first element of an array

Method	Description
unShift()	Adds new elements at the beginning of the array
reverse()	Reverses the order of the elements in an array.
slice()	Pulls a copy of a part of an array into a new array.
splice()	Adds elements in a particular way and position.
toString()	Converts the array elements into strings.
valueOf()	Returns the primitive value of the given object.
indexOf()	Returns the first index at which a given element is found.
lastIndexOf()	Returns the final index at which a given element appears.
join()	Combines elements of an array into one single string and then returns it
sort()	Sorts the array elements based on some condition.

```
// Declaring and initializing arrays

// Num Array
let arr = [10, 20, 30, 40, 50];
let arr1 = [110, 120, 130, 140];

// String array
let string_arr = ["Alex", "peter", "chloe"];

// push: Adding elements at the end of the array
arr.push(60);
console.log("After push op " + arr);

// unshift() Adding elements at the start of the array
arr.unshift(0, 10);
console.log("After unshift op " + arr );
```

```
// pop: removing elements from the end of the array
arr.pop();
console.log("After pop op" + arr);

// shift(): Removing elements from the start of the array
arr.shift();
console.log("After shift op " + arr);

// splice(x,y): removes x number of elements
// starting from index y
arr.splice(2, 1);
console.log("After splice op" + arr);

// reverse(): reverses the order of elements in array
arr.reverse();
console.log("After reverse op" + arr);

// concat(): merges two or more array
console.log("After concat op" + arr.concat(arr1));
```

Loops

- Loops are a useful feature in most programming languages.
- With loops you can evaluate a set of instructions/functions repeatedly until certain condition is reached.
- They let you run code blocks as many times as you like with different values while some condition is true. Loops can be created in the following ways in JavaScript:

Loop	Description	Syntax

for	Loops over a block of with conditions specified in the beginning.	for (initialization condition; testing condition;increment/decrement)
		{
		statement(s)
		}

while	Entry control loop which executes after checking the	while (boolean condition) {

Loop	Description	Syntax
	condition.	loop statements... }

do-while	Exit Control Loop which executes once before checking the condition.	do
		{
		statements..
		}
		while (condition);

for-in	Another version of for loop to provide a simpler way to iterate.	for (variableName in Object)
		{
		statement(s)
		}

```
// Illustration of for loop
let x;

for (x = 2; x <= 4; x++) {
    console.log("Value of x:" + x);
}

// creating an Object
let languages = {
    first: "C",
    second: "Java",
    third: "Python",
    fourth: "PHP",
    fifth: "JavaScript",
};

// Iterate through every property of the object
for (itr in languages) {
    console.log(languages[itr]);
}

// Illustration of while loop
let y = 1;

// Exit when x becomes greater than 4
```

```
while (y <= 4) {
    console.log("Value of y:" + y);
    y++;
}

// Illustration of do-while loop
let z = 21;

do {

    console.log("Value of z:" + z);

    z++;
} while (z < 20);
```

Output

```
Value of x:2
Value of x:3
Value of x:4
C
Java
Python
PHP
JavaScript
Value of y:1
Value of y:2
Value of y:3
Value of y:4
Value of z:21
```

If-else

- If-else is used in JavaScript to execute a block of codes conditionally.
- These are used to set conditions for your code block to run. If certain condition is satisfied certain code block is executed otherwise else code block is executed.
- JavaScript allows us to nest if statements within if statements as well. i.e, we can place an if statement inside another if statement.

```
// JavaScript program to illustrate if-else statement
const i = 10;

if (i < 15)
    console.log("Value of i is less than 10");
else
    console.log("Value of i is greater than 10");
```

Output

```
Value of i is less than 10
```

Strings

- Strings in JavaScript are primitive and immutable data types used for storing and manipulating text data which can be zero or more characters consisting of letters, numbers or symbols.
- JavaScript provides a lot of methods to manipulate strings. Some most used ones are:

Methods	Description
concat()	Used for concatenating multiple strings into a single string.
match()	Used for finding matche of a string against a provided pattern.
replace()	Used for finding and replacing a given text in string.
substr()	Used to extract length characters from a given string.
slice()	Used for extracting an area of the string and returs it
lastIndexOf()	Used to return the index (position) of the last occurrence of a specified value.
charAt()	Used for returning the character at a particular index of a string

Methods	Description
valueOf()	Used for returning the primitive value of a string object.
split()	Used for splitting a string object into an array of strings.
toUpperCase()	Used for converting strings to upper case.
toLoweCase()	Used for converting strings to lower case.

```
let gfg = 'GFG ';
let geeks = 'stands-for-GeeksforGeeks';

// Print the string as it is
console.log(gfg);
console.log(geeks);

// concat() method
console.log(gfg.concat(geeks));

// match() method
console.log(geeks.match(/eek/));

// charAt() method
console.log(geeks.charAt(5));

// valueOf() method
console.log(geeks.valueOf());

// lastIndexOf() method
console.log(geeks.lastIndexOf('for'));

// substr() method
console.log(geeks.substr(6));

// indexOf() method
console.log(gfg.indexOf('G'));

// replace() method
console.log(gfg.replace('FG', 'fg'));

// slice() method
```

```
console.log(geeks.slice(2, 8));

// split() method
console.log(geeks.split('-'));

// toUpperCase method
console.log(geeks.toUpperCase(geeks));

// toLowerCase method
console.log(geeks.toLowerCase(geeks));
```

Output

```
GFG
stands-for-GeeksforGeeks
GFG stands-for-GeeksforGeeks
[
  'eek',
  index: 12,
  input: 'stands-for-GeeksforGeeks',
  groups: undefined
]
s
stands-for-GeeksforGeeks
16
-for-GeeksforGeeks
0
Gfg
an...
```

Regular Expressions

- A regular expression is a sequence of characters that forms a search pattern.
- The search pattern can be used for text search and text to replace operations.
- A regular expression can be a single character or a more complicated pattern.

Regular Expression Modifiers:

Modifiers can be used to perform multiline searches. Some of the pattern modifiers that are allowed in JavaScript:

Modifiers	Description
[abc]	Find any of the character inside the brackets
[0-9]	Find any of the digits between the brackets 0 to 9
(x/y)	Find any of the alternatives between x or y separated with |

Regular Expression Patterns:
Metacharacters are characters with a special meaning. Some of the metacharacters that are allowed in JavaScript:

Metacharacters	Description
.	Used for finding a single character, except newline or line terminator
\d	Used to find a digit.
\s	Used to find a whitespace character
\uxxxx	Used to find the Unicode character specified by the hexadecimal number

Quantifiers:

They provide the minimum number of instances of a character, group, or character class in the input required to find a match.

Some of the quantifiers allowed in JavaScript are:

Quantifiers	Description
n+	Used to match any string that contains at least one n
n*	Used to match any string that contains zero or more occurrences of n
n?	Used to matches any string that contains zero or one occurrences of n
n{x}	Matches strings that contain a sequence of X n's
^n	Matches strings with n in the first place

Here is an example to help you understand regular expression better.

```
// Program to validate the email address
function validateEmail(email) {

    // Regex pattern for email
    const re = /\S+@\S+\.\S+/g;

    // Check if the email is valid
    let result = re.test(email);

    if (result) {
        console.log("The email is valid.");
    } else {
        console.log("The email is not valid.");
    }
}

// Input Email Id
let email = "abc@gmail.com"
validateEmail(email);

email = "abc#$#@45com"
validateEmail(email);
```

Output

```
The email is valid.
The email is not valid.
```

Data Transformation

- Data transformation is converts data from one format to another.
- It can be done with the usage of higher-order functions which can accept one or more functions as inputs and return a function as the result.
- All higher-order functions that take a function as input are map(), filter(), and reduce().

Method	Description	Syntax
map()	Iterates over an array and calls function on every element of array.	array.map(function(currentValue, index, arr), thisValue)
filter()	Create a new array from a given array after applying a condition.	array.filter(callback(element, index, arr), thisValue)
reduce()	Reduces the array to single value using a function	array.reduce(function(total, currentValue, currentIndex, arr), initialValue)

```
const num = [16, 25];

/* Using JS map() Method */
console.log(num.map(Math.sqrt));

const ages = [19, 37, 16, 42];

/* Using JS filter() Method */
console.log(ages.filter(checkAdult));

function checkAdult(age) {
    return age >= 18;
}
```

```
/* Using JS reduce() Method */
const numbers = [165, 84, 35];
console.log(numbers.reduce(myFunc));

function myFunc(total, num) {
    return total - num;
}
```

Output

```
[ 4, 5 ]
[ 19, 37, 42 ]
46
```

Date objects

- The Date object is an inbuilt datatype of JavaScript language.
- It is used to deal with and change dates and times.
- There are four different way to declare a date, the basic things is that the date objects are created by the new Date() operator.

Syntax

```
new Date()
new Date(milliseconds)
new Date(dataString)
new Date(year, month, date, hour, minute, second, millisecond)
```

There are various methods in JavaScript used to get date and time values or create custom date objects. Some of these methods are:

Method	Description
getDate()	Used to return the month's day as a number (1-31)
getTime()	Used to get the milliseconds since January 1, 1970

Method	Description
getMinutes()	Returns the current minute (0-59)
getFullYear()	Returns the current year as a four-digit value (yyyy)
getDay()	Returns a number representing the weekday (0-6) to
parse()	Returns the number of milliseconds since January 1, 1970
setDate()	Returns the current date as a number (1-31)
setTime()	Sets the time (milliseconds since January 1, 1970)

```
// Here a date has been assigned by creating a date obj
let DateObj = new Date("October 13, 1996 05:35:32");

// getDate()
let A = DateObj.getDate();

// Printing date of the month
console.log(A);

// getTime()
let B = DateObj.getTime();

// Printing time in milliseconds.
console.log(B);

// getMinutes()
let minutes = DateObj.getMinutes();

// Printing minute.
console.log(minutes);

// getFullYear()
let C = DateObj.getFullYear();

// Printing year
console.log(C);
```

```
// getDay()
let Day = DateObj.getDay();

// Printing day of the week
console.log("Number of Day: " + Day);

// setDate
DateObj.setDate(15);

let D = DateObj.getDate();

// Printing new date of the month
console.log(D);

// parse(), taking wrong date string as input.
let date = "February 48, 2018 12:30 PM";

// calling parse function.
let msec = Date.parse(date);
console.log(msec);
```

Output

```
13
845184932000
35
1996
Number of Day: 0
15
NaN
```

DOM

- DOM stands for **Document Object Model.**
- It defines the logical structure of documents and the way a document is accessed and manipulated. JavaScript can not understand the tags in HTML document but can understand objects in DOM.
- Below are some of the methods provided by JavaScript to manipulate these nodes and their attributes in the DOM:

Method	Description
appendChild()	Adds a new child node as the last child node.
cloneNode()	Duplicates an HTML element.
hasAttributes()	Returns true If an element has any attributes otherwise,returns false.
removeChild()	Removes a child node from an element using the Child() method.
getAttribute()	Returns the value of an element node's provided attribute.
getElementsByTagName()	Returns a list of all child elements.
isEqualNode()	Determines whether two elements are same.

```
<html>
<head>
    /* CSS is used to make the output looks good */
    <style>
        #sudo {
            border: 1px solid green;
            background-color: green;
            margin-bottom: 10px;
            color: white;
            font-weight: bold;
        }

        h1,
        h2 {
            text-align: center;
            color: green;
            font-weight: bold;
        }
    </style>
</head>
```

```
<body>
    <h1>GeeksforGeeks</h1>
    <h2>DOM appendChild() Method</h2>
    <div id="sudo">
        The Good Website is learning for Computer Science is-
    </div>
    <button onclick="geeks()">Submit</button>
    <br />
    <div style="border: 3px solid green">
        <h1>GeeksforGeeks</h1>
        <h2>A computer science portal for geeks</h2>
    </div>
    <h2>DOM cloneNode() Method</h2>
    <button onclick="nClone()">
        Click here to clone the above elements.
    </button>
    <br />
    <h2>DOM hasAttributes() Method</h2>
    <p id="gfg">
        Click on the button to check if that
        body element has any attributes
    </p>
    <button type="button" onclick="hasAttr()">
        Submit
    </button>
    <br />
    <h2>DOM removeChild() Method</h2>
    <p>Sorting Algorithm</p>
    <ul id="listitem">
        <li>Insertion sort</li>
        <li>Merge sort</li>
        <li>Quick sort</li>
    </ul>
    <button onclick="Geeks()">
        Click Here!
    </button>
    <br />
    <h2>DOM getAttribute() Method</h2>
    <br />
    <button id="button" onclick="getAttr()">
        Submit
    </button>
    <p id="gfg1"></p>
    <br />
    <h2>DOM getElementsByTagName()</h2>
```

```
    <p>A computer science portal for geeks.</p>
    <button onclick="getElememt()">
        Try it
    </button>
    <h3>DOM isEqualNode() method .</h3>
    <!-- 3 div elements-->
    <div>GeeksforGeeks</div>
    <div>GfG</div>
    <div>GeeksforGeeks</div>
    <button onclick="isequal()">
        Check
    </button>
    <p id="result"></p>
    <script>
        function geeks() {
            var node = document.createElement("P");
            var t = document.createTextNode("GeeksforGeeks");
            node.appendChild(t);
            document.getElementById("sudo").appendChild(node);
        }
        function nClone() {
            // Accessing div attribute using a variable geek
            var geek = document.getElementsByTagName("DIV")[0];

            // Cloning geek variable into a variable named clone
            var clone = geek.cloneNode(true);

            // Adding our clone variable to end of the document
            document.body.appendChild(clone);
        }
        function hasAttr() {
            var s = document.body.hasAttributes();
            document.getElementById("gfg").innerHTML = s;
        }

        function Geeks() {
            var doc = document.getElementById("listitem");
            doc.removeChild(doc.childNodes[0]);
        }

        /* Using getElementById */
        function getAttr() {
            var rk =
document.getElementById("button").getAttribute("onClick");
            document.getElementById("gfg1").innerHTML = rk;
```

```
        }

        /* Using getElementsByTagName */
        function getElement() {
            var doc = document.getElementsByTagName("p");
            doc[0].style.background = "green";
            doc[0].style.color = "white";
        }

        /* Cheacking the equality */
        function isequal() {
            var out = document.getElementById("result");
            var divele = document.getElementsByTagName("div");
            out.innerHTML +=
                "element 1 equals element 1: " +
                divele[0].isEqualNode(divele[0]) +
                "<br/>";
            out.innerHTML +=
                "element 1 equals element 2: " +
                divele[0].isEqualNode(divele[1]) +
                "<br/>";
            out.innerHTML +=
                "element 1 equals element 3: " +
                divele[0].isEqualNode(divele[2]) +
                "<br/>";
        }
    </script>
</body>

</html>
```

Numbers and Math

- JavaScript provides various properties and methods to deal with Numbers and Maths.
- Number Properties include MAX value, MIN value, NAN(not a number), negative infinity , positive infinity etc.
- Some of the methods in JavaScript to deal with numbers are:

Method	Description
valueOf()	Returns a number in its original form.
toString()	Returns string representation of an integer.
toFixed()	Returns a number's string with a specified number of decimals.
toPricision()	Converts a number to a string of a specified length.
toExponential()	Returns a rounded number written in exponential notation.

```
var num = 213;
var num1 = 213.3456711;

// JS valueof() Method
console.log("Output : " + num.valueOf());

// JS tostring() Method
console.log("Output : " + num.toString(2));

// JS tofixed() Method
console.log("Output : " + num1.toString(2));

// JS topricision() Method
console.log("Output : " + num1.toPrecision(3));

// JS toexponential() Method
console.log("Output : " + num1.toExponential(4));
```

Output

```
Output : 213
Output : 11010101
Output : 11010101.010110000111110111100110101101011010100001
Output : 213
Output : 2.1335e+2
```

- Javascript provides math object which is used to perform mathematical operations on numbers
- There are many math object properties which include euler's number, PI, square root, logarithm.
- Some of the methods in JavaScript to deal with math properties are:

Method	Description
max(x,y,z...n)	Returns the highest-valued number
min(x,y,z...n)	Returns the lowest-valued number
exp(x)	Returns x's exponential value.
log(x)	Returns the natural logarithm (base E) of x.
sqrt(x)	Returns x's square root value.
pow(x,y)	Returns the value of x to the power of y
round(x)	Rounds the value of x to the nearest integer
sin(x)	Finds the sine value of x(x is in radians).
tan(x)	Finds the angle's(x) tangent value.

```
<script>
    document.getElementById("GFG").innerHTML =
        "Math.LN10: " + Math.LN10 + "<br>" +
        "Math.LOG2E: " + Math.LOG2E + "<br>" +
        "Math.Log10E: " + Math.LOG10E + "<br>" +
        "Math.SQRT2: " + Math.SQRT2 + "<br>" +
        "Math.SQRT1_2: " + Math.SQRT1_2 + "<br>" +
        "Math.LN2: " + Math.LN2 + "<br>" +
        "Math.E: " + Math.E + "<br>" +
        "Math.round: " + Math.round(5.8) + "<br>" +
        "Math.PI: " + Math.PI + "<br>" +
        "
        < p > <b>Math.sin(90 * Math.PI / 180):</b> " +
    Math.sin(90 * Math.PI / 180) + "</p>
    " +
    "
```

```
        < p > <b>Math.tan(90 * Math.PI / 180):</b> " +
    Math.tan(90 * Math.PI / 180) + "</p>
    " +
    "
        < p > <b>Math.max(0, 150, 30, 20, -8, -200):</b> " +
    Math.max(0, 150, 30, 20, -8, -200) + "</p>
    " +
    "
        < p > <b>Math.min(0, 150, 30, 20, -8, -200):</b> " +
    Math.min(0, 150, 30, 20, -8, -200) + "</p>
    " +
    "
        < p > <b>Math.pow(3,4):</b> " + Math.pow(3, 4) + "</p >
            ";
</script>
```

Events

- Javascript has events to provide a dynamic interface to a webpage.
- When a user or browser manipulates the page events occur.
- These events are hooked to elements in the Document Object Model(DOM). Some of the events supported by JavaScript:

Events	Description
onclick()	Triggers an event when an element is clicked.
onkeyup()	Executes instructions whenever a key is released after pressing.
onmouseover()	Triggers an event when mouse pointer is hovered over an element
onmouseout()	Triggers an event when mouse pointer is moved away from an element.
onchange()	Detects the change in value of any element listing to this event.
onload()	Evokes an event when an element is completely loaded.
onfocus()	Triggers when an aspect is brought into focus.

Events	Description
onblur()	Evoked an event when an element loses focus.
onsubmit()	Evokes an event when a form is submitted
ondrag()	Invokes an event when an element is dragged.
oninput()	Triggers when an input field gets any value.

```
<html>
<head>
    /* CSS is used to make the output looks good */
    <style>
        #geeks {
            border: 1px solid black;
            padding: 15px;
            width: 60%;
        }

        h1 {
            color: green;
        }
    </style>
    <script>
        function hiThere() {
            alert("Hi there!");
        }
        function focused() {
            var e = document.getElementById("inp");
            if (confirm("Got it?")) {
                e.blur();
            }
        }

        /* Mouseover event */
        document.getElementById("hID").addEventListener("mouseover",
over);

        /* Mouseoout event */
        document.getElementById("hID").addEventListener("mouseout",
out);
```

```
        /* Over on green */
        function over() {
            document.getElementById("hID").style.color = "green";
        }

        /* Leaving Out Black */
        function out() {
            document.getElementById("hID").style.color = "black";
        }

        function Geeks() {
            var x = document.getElementById("GFG").value;
            document.getElementById("sudo").innerHTML = "Selected
Subject: " + x;
        }

        /* Success alert */
        function Geek() {
            alert("Form submitted successfully.");
        }
        function Function() {
            document.getElementById("geeks").style.fontSize =
"30px";
            document.getElementById("geeks").style.color = "green";
        }
    </script>
</head>

<body>
    <!-- onload event -->
    <img onload="alert('Image completely loaded')" alt="GFG-Logo"
        src="https://media.geeksforgeeks.org/wp-content/cdn-
uploads/GeeksforGeeksLogoHeader.png" />
    <br />

    <!-- onclick event -->
    <h2>onclick event</h2>
    <button type="button" onclick="hiThere()" on>
        Click me
    </button>

    <!-- onfocus event -->
    <h2>onfocus event</h2>
    <p>Take the focus into the input box below:</p>
```

```
<input id="inp" onfocus="focused()" />

<!-- onblur Event -->
<h2>onblur event</h2>
<p>
    Write something in the input box and
    then click elsewhere in the document
    body.
</p>
<input onblur="alert(this.value)" />

<!-- onmouseover and onmouseout event -->
<h2 id="hID">onmouseover event</h2>
<h2>onchange Event</h2>
<p>Choose Subject:</p>
<select id="GFG" onchange="Geeks()">
    <option value="Data Structure">
        Data Structure
    </option>
    <option value="Algorithm">
        Algorithm
    </option>
    <option value="Computer Network">
        Computer Network
    </option>
    <option value="Operating System">
        Operating System
    </option>
    <option value="HTML">
        HTML
    </option>
</select>

<p id="sudo"></p>

<!-- onsubmit event -->
<h2>onsubmit event</h2>
<form onsubmit="Geek()">
    First Name:<input type="text" value="" />
    <br />
    Last Name:<input type="text" value="" />
    <br />
    <input type="submit" value="Submit" />
</form>
```

```
    <!--ondrag event -->
    <h2>ondrag event attribute</h2>
    <div id="geeks" ondrag="Function()">
        GeeksforGeeks: A computer science portal for geeks
    </div>
</body>
</html>
```

Error

- When executing JavaScript code, errors will most definitely occur when the JavaScript engine encounters a syntactically invalid code.
- These errors can occur due to the fault from the programmer's side or the input is wrong or even if there is a problem with the logic of the program.
- Javascript has a few statements to deal with these errors:

Statement	Description
try	Tests a block of code to check for errors.
catch	Handles the error if any are present.
throw	Allows construction of new errors.
finally	Executes code after try and catch.

```
<html>
<body>
    <h2>
        JavaScript throw try catch finally keywords
    </h2>
    <p>Please enter a number:</p>
    <input id="demo" type="text" />
    <button type="button" onclick="myFunction()">
        Test Input
    </button>
    <p id="p01"></p>
    <script>
        function myFunction() {
```

```
            const message = document.getElementById("p01");
            message.innerHTML = "";
            let x = document.getElementById("demo").value;

            /* Using try.. catch.. with conditions*/
            try {
                if (x == "") throw "is empty";
                if (isNaN(x)) throw "is not a number";
                x = Number(x);
                if (x > 20) throw "is too high";
                if (x <= 20) throw "is too low";
            } catch (err) {
                message.innerHTML = "Input " + err;
            } finally {
                document.getElementById("demo").value = "";
            }
        }
    </script>
</body>
</html>
```

Window Properties

- The window object is the topmost object of DOM hierarchy.
- Whenever a window appears on the screen to display the contents of document, the window object is created.
- To access the properties of the window object, you will specify object name followed by a period symbol (.) and the property name.

Syntax

```
window.property_name
```

The properties and methods of Window object that are commonly used are listed in the below tables:

Property	Description
window	Returns the current window or frame.

Property	Description
screen	Returns the window's Screen object.
toolbar	Creates a toolbar object, whose visibility can be toggled in the window.
Navigator	Returns the window's Navigator object.
frames[]	Returns all <iframe> elements in the current window.
document	Returns a reference to the document object.
closed	Boolean used to check whether the window is closed or not.
length	Represents the number of frames in the current window.
History	Provides the window's History object.

```
<html>
<body>
    <h1>The Window properties</h1>
    <h2>The origin Property</h2>
    <p id="demo"></p>
    <br />
    <button type="button" onclick="getResolution();">
        Get Resolution
    </button>
    <br />
    <button type="button" onclick="checkConnectionStatus();">
        Check Connection Status
    </button>
    <br />
    <button type="button" onclick="getViews();">
        Get Views Count</button>
    <br />
    <p>
        <button onclick="closeWin()">
            Close "myWindow"
        </button>
```

```
    </p>

    <script>
        // JS location property
        let origin = window.location.origin;
        document.getElementById("demo").innerHTML = origin;

        // JS screen property
        function getResolution() {
            alert("Your screen is: " + screen.width + "x" + screen.hei
        }

        // JS toolbar property
        var visible = window.toolbar.visible;

        // JS navigator property
        function checkConnectionStatus() {
            if (navigator.onLine) {
                alert("Application is online.");
            } else {
                alert("Application is offline.");
            }
        }
        // JS history property
        function getViews() {
            alert(
                "You've accessed " + history.length + " web pages in t
session."
            );
        }
        // JS close property
        let myWindow;
        function closeWin() {
            if (myWindow) {
                myWindow.close();
            }
        }
    </script>
</body>
</html>
```

Method	Description
alert()	Shows a message and an OK button in an alert box.
print()	Prints the current window's content.
blur()	Removes the current window's focus.
setTimeout()	Evaluates an expression after a specified time interval.
clearTimeout()	Removes the timer that was set with setTimeout()
setInterval()	Evaluates an expression at intervals defined by the user.
prompt()	Shows a conversation window asking for feedback from the visitor.
close()	Closes the currently open window.
focus()	Sets the current window's focus.
resizeTo()	Resizes the window to the width and height supplied.

```
<html>
<head>
    <title>JavaScript Window Methods</title>
    <style>
        .gfg {
            font-size: 36px;
        }

        form {
            float: right;
            margin-left: 20px;
        }
    </style>
</head>

<body>
```

```
    <div class="gfg">JavaScript Window Methods</div>
    <br />
    <button onclick="windowOpen()">
        JavaScript window Open
    </button>
    <button onclick="resizeWin()">
        JavaScript window resizeTo
    </button>
    <button onclick="windowBlur()">
        JavaScript window Blur
    </button>
    <button onclick="windowFocus()">
        JavaScript window Focus
    </button>
    <button onclick="windowClose()">
        JavaScript window Close
    </button>
    <br />
    <br />
    <p id="g"></p>
    <form>
        <button onclick="setTimeout(wlcm, 2000);">
            Alert after 2 Second
        </button>
        <button onclick="geek()">Click me!</button>
        <input type="button" value="Print" onclick="window.print()" />
    </form>
    <br /><br />
    <button id="btn" onclick="fun()" style="color: green">
        JavaScript Used setTimeOut
    </button>
    <button id="btn" onclick="stop()">
        JavaScript clearTimeout
    </button>
    <script>
        var gfgWindow;

        // Function that open the new Window
        function windowOpen() {
            gfgWindow = window.open(
                "https://www.geeksforgeeks.org/",
                "_blank",
                "width=200, height=200"
            );
        }
```

```
        // Function that Resize the open Window
        function resizeWin() {
            gfgWindow.resizeTo(400, 400);
            gfgWindow.focus();
        }

        // Function that Closes the open Window
        function windowClose() {
            gfgWindow.close();
        }

        // Function that blur the open Window
        function windowBlur() {
            gfgWindow.blur();
        }

        // Function that focus on open Window
        function windowFocus() {
            gfgWindow.focus();
        }

        // Alert function
        function wlcm() {
            alert("Welcome to GeeksforGeeks");
        }

        // Prompt function
        function geek() {
            var doc = prompt("Please enter some text", "GeeksforGeeks"
            if (doc != null) {
                document.getElementById("g").innerHTML = "Welcome to "
doc;
            }
        }

        // Function setTimeout and clearTimeout
        var t;
        function color() {
            if (document.getElementById("btn").style.color == "blue")
                document.getElementById("btn").style.color = "green";
            } else {
                document.getElementById("btn").style.color = "blue";
            }
        }
```

```
        function fun() {
            t = setTimeout(color, 3000);
        }
        function stop() {
            clearTimeout(t);
        }
    </script>
</body>
</html>
```

Benefits of Using JavaScript Cheat Sheet

Here are some key benefits of a JavaScript Cheat Sheet:

- **Efficient Web Development:** A JavaScript Cheat Sheet provides a quick reference guide for web developers, enabling faster and more efficient coding. It helps in reducing the time spent on searching for syntax or functions, thereby increasing productivity.
- **Comprehensive Function Reference:** The cheat sheet includes an extensive collection of JavaScript functions, methods, and properties, covering

events. This makes it a valuable resource for both beginners and experienced developers.
- **Dynamic Content Creation:** With the inclusion of JavaScript functions in the cheat sheet, developers can create more interactive and dynamic web content. It aids in enhancing the user experience and engagement on web pages.
- **Interoperability:** JavaScript is a core technology of the web. A JavaScript Cheat Sheet can be beneficial when working with other web technologies like HTML, CSS, and various web development frameworks.
- **Optimized for Performance:** A well-structured JavaScript code using the correct functions and methods can significantly improve the performance of web applications. The cheat sheet can guide developers in creating performance-optimized scripts.
- **Event Handling:** The cheat sheet covers methods for handling user interactions and browser events, enabling more responsive and interactive web content.

Recommended Links:

- JavaScript Tutorial
- JavaScript Examples
- HTML Cheat Sheet
- CSS Cheat Sheet
- Self-paced JavaScript course

Arrays i
JavaScr

Workin
of Array
in
JavaScr

Arrays in JavaScript

Visit Course

Comment

More info

Campus Training Program

Next Article

jQuery Cheat Sheet – A Basic Guide to jQuery

GeeksforGeeks
Sanchhaya Education Private Limited

Corporate & Communications Address:
A-143, 7th Floor, Sovereign Corporate Tower, Sector- 136, Noida, Uttar Pradesh (201305)

Registered Address:

K 061, Tower K, Gulshan Vivante Apartment, Sector 137, Noida, Gautam Buddh Nagar, Uttar Pradesh, 201305

Company
About Us
Legal
Privacy Policy
Careers
In Media
Contact Us
Corporate Solution
Campus Training Program

Explore
Job-A-Thon
Offline Classroom Program
DSA in JAVA/C++
Master System Design
Master CP
Videos

Tutorials
Python
Java
C++
PHP
GoLang
SQL
R Language
Android

DSA
Data Structures
Algorithms
DSA for Beginners
Basic DSA Problems
DSA Roadmap
DSA Interview Questions
Competitive Programming

Data Science & ML
Data Science With Python
Machine Learning
ML Maths
Data Visualisation
Pandas
NumPy
NLP
Deep Learning

Web Technologies
HTML
CSS
JavaScript
TypeScript
ReactJS
NextJS
NodeJs
Bootstrap
Tailwind CSS

Python Tutorial
Python Examples
Django Tutorial
Python Projects
Python Tkinter
Web Scraping
OpenCV Tutorial
Python Interview Question

Computer Science
GATE CS Notes
Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths

DevOps
Git
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

System Design
High Level Design
Low Level Design
UML Diagrams
Interview Guide
Design Patterns
OOAD
System Design Bootcamp
Interview Questions

School Subjects
Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar

Databases
SQL
MYSQL
PostgreSQL
PL/SQL
MongoDB