# JavaScript Cheat Sheet

*JavaScript CheatSheet for beginners!*

# Basics - Javascript

## What is Javascript?

JavaScript is a lightweight programming language that web developers commonly use to create more dynamic interactions when developing web pages, applications, servers, and or even games.

## Features of JavaScript

The following are more characteristics/features of JavaScript:

1. Object-Centered Script Language
2. Client edge Technology
3. Validation of User's Input
4. Else and If Statement
5. Interpreter Centered
6. Ability to perform In Built Function
7. Case Sensitive format
8. Light Weight and delicate
9. Statements Looping
10. Handling Events

JavaScript Function syntax

## Where is JavaScript used?

JavaScript is highly used in

1. Web Applications
2. Games
3. Mobile Apps
4. Web Servers

## Who developed JavaScript, and what was the first name of JavaScript?

JavaScript was developed by **Brendan Eich**, who was a Netscape programmer. Brendan Eich developed this new scripting language in just ten days in the year September 1995. At the time of its launch, JavaScript was initially called Mocha. After that, it was called **Live Script** and later known as **JavaScript**.

### What are different Data types in Javascript?

1. String
2. Number
3. Bigint
4. Boolean
5. Undefined
6. Null
7. Symbol
8. Object

---

## Syntax & Meaning

### On Page Script

Adding internal JavaScript to HTML

```
<script type="text/javascript"> /*JS code goes here*/ </script>
```

### External JS File

Adding external JavaScript to HTML

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

### Functions

JavaScript Function syntax

```
function nameOfFunction () { // function body
}
```

# Conditional Statements

Conditional statements are used to perform operations based on some conditions.

## If Statement

The block of code to be executed, when the condition specified is true.

```
if (condition) {
// block of code to be executed if the condition is true
}
```

## If-else Statement

If the condition for the if block is false, then the else block will be executed.

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

## Else-if Statement

A basic if-else ladder

```
if (condition1) {
// block of code to be executed if condition1 is true
} else if (condition2) {
// block of code to be executed if the condition1 is false and condition2 is true
}
  else {
// block of code to be executed if the condition1 is false and condition2 is flase
  }
```

## Switch Statement

```
switch(expression) {
case x:
// code block
break;
case y:
// code block
break;
default:
// code block
}
```

# Iterative Statements (Loops)

Iterative statement facilitates programmer to execute any block of code lines repeatedly and can be controlled as per conditions added by the programmer.

## For Loop

For loop syntax in javascript

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

## While Loop

Runs the code till the specified condition is true

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

## Do While Loop

A do while loop is executed at least once despite the condition being true or false

```
do {
// run this code in block i++;
} while (condition);
```

# Strings

The string is a sequence of characters that is used for storing and managing text data.

### charAt method

Returns the character from the specified index.

```
str.charAt(3)
```

### concat method

Joins two or more strings together.

```
str1.concat(str2)
```

### index of method

Returns the index of the first occurrence of the specified character from the string else -1 if not found.

```
str.indexOf('substr')
```

### match method

Searches a string for a match against a regular expression.

```
str.match(/(chapter \d+(\.\d)*)/i)
```

### replace method

Searches a string for a match against a specified string or char and returns a new string by replacing the specified values.

```
str1.replace(str2)
```

### search method

Searches a string against a specified value.

```
str.search('term')
```

### split method

Splits a string into an array consisting of substrings.

```
str.split('\n')
```

### substring method

Returns a substring of a string containing characters from the specified indices.

```
str.substring(0,5)
```

## Arrays

The array is a collection of data items of the same type. In simple terms, it is a variable that contains multiple values.

### variable

Containers for storing data.

```
var fruit = ["element1", "element2", "element3"];
```

### concat method

Joins two or more arrays together.

```
concat()
```

### indexOf method

Returns the index of the specified item from the array.

```
indexOf()
```

### join method

Converts the array elements to a string.

```
join()
```

### pop method

Deletes the last element of the array.

```
pop()
```

### reverse method

This method reverses the order of the array elements.

```
reverse()
```

### sort method

Sorts the array elements in a specified manner.

```
sort()
```

### toString method

Converts the array elements to a string.

```
toString()
```

### valueOf method

returns the relevant Number Object holding the value of the argument passed

```
valueOf()
```

## Number Methods

JS math and number objects provide several constant and methods to perform mathematical operations.

### toExponential method

Converts a number to its exponential form.

```
toExponential()
```

### toPrecision method

Formats a number into a specified length.

```
toPrecision()
```

### toString method

Converts an object to a string

```
toString()
```

### valueOf method

Returns the primitive value of a number.

```
valueOf()
```

## Maths Methods

### ceil method

Rounds a number upwards to the nearest integer, and returns the result

```
ceil(x)
```

### exp method

Returns the value of E^x.

```
exp(x)
```

### log method

Returns the logarithmic value of x.

```
log(x)
```

### pow method

Returns the value of x to the power y.

```
pow(x,y)
```

### random method

Returns a random number between 0 and 1.

```
random()
```

### sqrt method

Returns the square root of a number x

```
sqrt(x)
```

## Dates

Date object is used to get the year, month and day. It has methods to get and set day, month, year, hour, minute, and seconds.

### Pulling Date from the Date object

Returns the date from the date object

```
getDate()
```

### Pulling Day from the Date object

Returns the day from the date object

```
getDay()
```

### Pulling Hours from the Date object

Returns the hours from the date object

```
getHours()
```

### Pulling Minutes from the Date object

Returns the minutes from the date object

```
getMinutes()
```

### Pulling Seconds from the Date object

Returns the seconds from the date object

```
getSeconds()
```

### Pulling Time from the Date object

Returns the time from the date object

```
getTime()
```

## Mouse Events

Any change in the state of an object is referred to as an Event. With the help of JS, you can handle events, i.e., how any specific HTML tag will work when the user does something.

### click

Fired when an element is clicked

```
element.addEventListener('click', ()=>{
// Code to be executed when the event is triggered
});
```

### oncontextmenu

Fired when an element is right-clicked

```
element.addEventListener('contextmenu', ()=>{
  // Code to be executed when the event is triggered
});
```

### dblclick

Fired when an element is double-clicked

```
element.addEventListener('dblclick', ()=>{
// Code to be executed when the event is triggered
});
```

### mouseenter

Fired when an element is entered by the mouse arrow

```
element.addEventListener('mouseenter', ()=>{
// Code to be executed when the event is triggered
});
```

### mouseleave

Fired when an element is exited by the mouse arrow

```
element.addEventListener('mouseleave', ()=>{
// Code to be executed when the event is triggered
});
```

### mousemove

Fired when the mouse is moved inside the element

```
element.addEventListener('mousemove', ()=>{
// Code to be executed when the event is triggered
});
```

## Keyboard Events

### keydown

Fired when the user is pressing a key on the keyboard

```
element.addEventListener('keydown', ()=>{
// Code to be executed when the event is triggered
});
```

### keypress

Fired when the user presses the key on the keyboard

```
element.addEventListener('keypress', ()=>{
// Code to be executed when the event is triggered
});
```

### keyup

Fired when the user releases a key on the keyboard

```
element.addEventListener('keyup', ()=>{
// Code to be executed when the event is triggered
});
```

## Errors

Errors are thrown by the compiler or interpreter whenever they find any fault in the code, and it can be of any type like syntax error, run-time error, logical error, etc. JS provides some functions to handle the errors.

### try and catch

Try the code block and execute catch when err is thrown

```
try {
// code to try
}
catch(err) {
// code to handle errors
}
```

## Window Methods

Methods that are available from the window object

### alert method

Used to alert something on the screen

```
alert()
```

## blur method

The blur() method removes focus from the current window.

```
blur()
```

## setInterval

Keeps executing code at a certain interval

```
setInterval(() => {
// Code to be executed
}, 1000);
```

## setTimeout

Executes the code after a certain interval of time

```
setTimeout(() => {
// Code to be executed
}, 1000);
```

## close

The Window. close() method closes the current window

```
window.close()
```

## confirm

The window.confirm() instructs the browser to display a dialog with an optional message, and to wait until the user either confirms or cancels

```
window.confirm('Are you sure?')
```

## open

Opens a new window

```
window.open("https://instagram.com/coding_dev_");
```

### prompt

Prompts the user with a text and takes a value. Second parameter is the default value

```
let name = prompt("What is your name?", "Tilak");
```

### scrollBy

```
window.scrollBy(200, 0); // Scroll 200px to the right
```

### scrollTo

Scrolls the document to the specified coordinates.

```
window.scrollTo(200, 0); // Scroll to horizontal position 200
```

### clearInterval

Clears the setInterval. var is the value returned by setInterval call

```
clearInterval(var)
```

### clearTimeout

Clears the setTimeout. var is the value returned by setTimeout call

```
clearTimeout(var)
```

### stop

Stops the further resource loading

```
stop()
```

# JavaScript Objects

Object is a non-primitive data-type that allows you to store multiple collections of data.

## Object creation

Syntax to declare an object is:

```
const object_name = {
   key1: value1,
   key2: value2
}
```

## Accessing Object Properties

### 1. Using dot Notation

```
objectName.key
```

### 2. Using bracket Notation

```
objectName["propertyName"]
```

# Query/Get Elements

The browser creates a DOM (Document Object Model) whenever a web page is loaded, and with the help of HTML DOM, one can access and modify all the elements of the HTML document.

### querySelector

Selector to select first matching element

```
document.querySelector('css-selectors')
```

### querySelectorAll

A selector to select all matching elements

```
document.querySelectorAll('css-selectors', ...)
```

### getElementsByTagName

Select elements by tag name

```
document.getElementsByTagName('element-name')
```

### getElementsByClassName

Select elements by class name

```
document.getElementsByClassName('class-name')
```

### Get Element by Id

Select an element by its id

```
document.getElementById('id')
```

# Creating Elements

Create new elements in the DOM

### createElement

Create a new element

```
document.createElement('div')
```

### createTextNode

Create a new text node

```
document.createTextNode('some text here')
```

# Accessing & Updating Elements

### innerHTML

Get or set the HTML Content of an element

```
element.innerHTML
```

### textContent

Modify the text content  of an element

```
element.textContent
```

# Regular Expressions

Regular expressions can be defined as search patterns that can be used to match string character combinations. Text search and text to replace procedures can both benefit from the search pattern.

## Pattern Modifiers

e — This is used for evaluating replacement
i — This is used for performing case-insensitive matching
U — This is used for ungreedy pattern
g — This is used for performing global matching
m — This is used for performing multiple line matching
s — This is used for treating strings as a single line
x — This is used for allowing comments and whitespace in the pattern

## Metacharacters

. — This is used for finding a single character, except newline or line terminator
\w — This is used for finding Word characters
\W — This is used for finding Non-word characters
\s — Used for finding Whitespace characters
\S — This is used for finding Non-whitespace characters
\b — This is used for finding matches at the beginning or at the end of a word
\B — This is used for finding matches not at the beginning or at the end of a word
\0 — This is used for finding NULL characters
\n — Used for finding a new line character
\f — This is used for finding a Form feed character
\r — This is used for finding a Carriage return character
\t — This is used for finding a Tab character
\v — Used for finding a Vertical tab character
\d — This is used for finding digits
\D — This is used for finding non-digit characters
\xxx — This is used for finding characters given by an octal number xxx
\xdd — This is used for finding characters given by a hexadecimal number dd
\uxxxx — This is used for finding the Unicode character given by a hexadecimal number XXXX

## Brackets

Group parts of a regular expression together by putting them inside round brackets or parentheses:
[abc] — This is used for finding all the characters between the brackets
(a|b|c) — This is used for finding all of the alternatives separated with |
[^abc] — This is used for finding every character that is not in the brackets
[0-9] — This is used for finding each digit from 0 to 9
[A-z] — This is used for finding each character from uppercase A to lowercase z

## Quantifiers

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

n+ — This is used for matching each string which is having one or more n
n* — This is used for matching any string which is having zero or more occurrences of
n? — This is used for matching strings which are having zero or one occurrence of
^n — This is used for matching strings with n in the first plac
?=n — This is used for matching all strings which are followed by a particular string
?!n — This is used for matching strings that are not followed by a particular string n
n{X} — This is used for matching strings that contain a sequence of X n'
n{X,Y} — This is used for matching a string that contains a sequence of X to Y n'
n{X,} — This is used for matching all strings which are having a sequence of X or more n'
n$ — This is used for matching all strings having n at the end.

# Advanced Javascript

## JavaScript Generators

A new way to work with functions and iterators.
Using a generator,

- you can stop the execution of a function from anywhere inside the function
- and continue executing code from a halted position.

```
// define a generator function
function* generator_function() {
  //....
}

// creating a generator
const generator_obj = generator_function();
```

## Closures

A closure is a function created inside another function but has access to the outer function variables. Example,

```
function makeFunc() {
let name = 'Mozilla';
function displayName() {
alert(name);
}
return displayName;
}
let myFunc =
makeFunc();
myFunc();
```

Above the displayName() inner function is returned from the outer function before being executed.

### Spread Operator

Quickly copy all or part of an existing array or object into another array or object

```
let variablename1 = [...value];
```

### Ternary Operator

Short, one-line conditional operator to replace if/else

```
condition ? 'True' : 'False'
```

## Error Handling

Various types of errors occur when we are coding in JavaScript. There are a few options for dealing with them:

**try** — We can define a code block for testing errors using the try block.
**catch** — We can set up a block of code to execute in the event of an error using the catch statement.
**throw** — Instead of the typical JavaScript errors, we can also create custom error messages using the throw statement.
**finally** — JavaScript also allows us to run our code regardless of the outcome of try and catch.

**JavaScript possesses its own inbuilt error object which has the following properties:**

**name** — It is used for setting or returning an error name.
message — It is used for setting or returning the error message as a string.

=> There are **six** types of ways in which the error property can return its name. They are as follows:

**EvalError** — It indicates that an error has occurred within the eval() method.
**RangeError** — It indicates that some number is “out of range”.
**ReferenceError** — It indicates that an illegal reference was occurring.
**SyntaxError** — It indicates that a syntax error was occurring.
**TypeError** — It indicates that a type error was occurring.
**URIError** — It indicates that an encodeURI() error was occurring.

## Promise

To create a promise object, we use the Promise() constructor.

```
let promise = new Promise(function(resolve, reject){
     //do something
});
```

## async/await

The syntax of **async** function is:

```
async function name(parameter1, parameter2, ...paramaterN) {
    // statements
}
```

The syntax to use **await** is:

```
let result = await promise;
```

## try...catch...finally

The finally block executes both when the code runs successfully or if an error occurs.

Syntax of try...catch...finally block is:

```
try {
    // try_statements
}
catch(error) {
    // catch_statements
}
finally() {
    // codes that gets executed anyway
}
```

## Recursion

Recursion is a process of calling itself. A function that calls itself is called a recursive function.
Syntax:

```
function recurse() {
    // function code
    recurse();
    // function code
}
recurse();
```

## Anonymous Functions

It is a function that does not have any name associated with it. General Syntax:

```
function() {
    // function Body
 }
```

**Anonymous function with Arrow function** syntax:

```
( () => {
    // Function Body...
} )();
```