JAVASCRIPT CHEAT SHEET

Table of Contents

Basics

1.1. On page Script

```
    <script type="text/javascript">  ...
    </script>
```

1.2. Include External JS File

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

1.3. Delay 1 second Timeout

```
   setTimeout(function () {
    console.log("Abdullah Ansari"); //run after one
second
    }, 1000);
```

1.4. Functions

```
function addNumbers(a, b) {
        return a + b; ;
        }
        x = addNumbers(1, 2);
```

1.5. Edit Dom Elements

```
    document.getElementById("elementID").innerHTML = "Hello
World!";
```

1.6. Output

```
1.console.log(Abd);
// write to the browser console

2.document.write(Abd);
// write to the HTML
```

```
3.alert(Abd);
// output in an alert box

4.confirm("Save Data?");
// yes/no dialog, returns true/false depending on
user click

5.prompt("Your age?","0");
// input dialog. Second argument is the initial
value
```

1.7. Comments

```
1: For Commenting Multiple Line Below syntax is used
/* Multi line
comment */

2: 1: For Commenting Single Line Below syntax is used
// One line
```

Variables

2.1. var Variable

It can be redeclared and its value can be reassigned, but only inside the context of a function. When the JavaScript code is run, variables defined using var are moved to the top.

```
var a=10;
```

2.2. const Variable

const variables in JavaScript cannot be used before they appear in the code. They can neither be reassigned values, that is, their value remains fixed throughout the execution of the code, nor can they be redeclared.

```
const a=10;
```

2.3. let Variable

The let variable, like const, cannot be redeclared. But they can be reassigned a value.

```
let a=10;
```

2.4. Examples

```
// variable
1: var a;

// string
2: let b = "init";

3: let c = "Hi" + " " + "Abd";     // = "Hi Abd"
```

```
4: let d = 1 + 2 + "3";
5: let e = [2,3,5,8];
// array

6: let f = false;
// boolean

7: let g = /()/;
// RegEx

8: let h = function(){};           // function object

9: const PI = 3.14;                // constant

10: var a = 1, b = 2, c = a + b;    // one line

11: let z = 'zzz';     // block scope local variable
```

2.5. Strict Mode

```
1: "use strict";   // Use strict mode to write secure
code

2: x = 1;    // Throws an error because variable is
not declared
```

2.6. Values

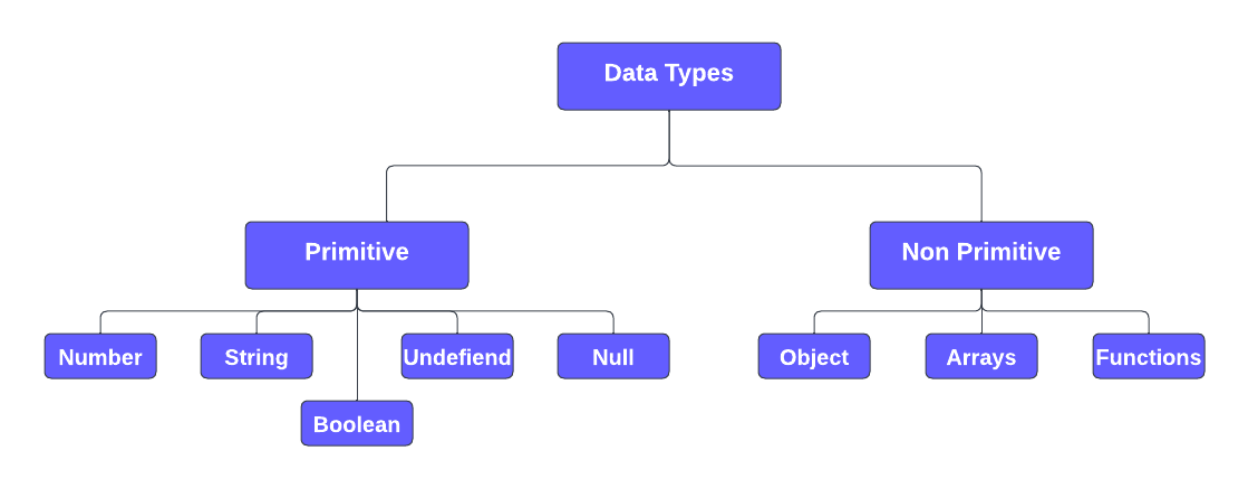
```
1: false, true                     // Boolean

2: 18, 3.14, 0b10011, 0xF6, NaN    // number

3: "flower", 'John'                // string

4: undefined, null , Infinity      // special
```

Data Types



3.1. Primitive Data Types

```
1- let age = 18;            // number

2- let name = "Jane         // string

3- let truth = false;      // boolean

4- let a; typeof a        // undefined

5- let a = null           // value null
```

3.2. Non Primitive Data Types (object data type)

```
// Object
1- let name = {first: "Abdullah",
last:"Ansari"};

// Array
1- let name = ["React","Vue","JS"];

// Function
3- function myfunc(){
  Console.log("Abdullah Ansari");}
```

Operators

4.1. Fundamental Operators

```
1- a = b + c - d;       // addition, substraction

2- a = b * (c / d);     // multiplication, division

3- x = 100 % 48;     // modulo. 100 / 48 remainder=4

4- a++; b--;     // postfix increment and decrement
```

4.2. Bitwise Operators

- **& :** The bitwise AND operator returns a 1 in every bit position where both operands' corresponding bits are 1.

- **| :** The bitwise OR operator (|) returns a 1 in each bit position where either or both operands' corresponding bits are 1.

- **~ :** The bitwise NOT operator reverses the operand's bits. It turns the operand into a 32-bit signed integer, just like other bitwise operators.

- **^ :** The bitwise XOR operator () returns a 1 in each bit position where the corresponding bits of both operands are 1s but not both.

- **<< :** The left shift operator shifts the first operand to the left by the provided number of bits. Extra bits that have been relocated to the left are discarded. From the right, zero bits are shifted in.

- **>> :** The right shift operator (>>) moves the first operand to the right by the provided number of bits. Extra bits that have been relocated to the right are discarded. The leftmost bit's copies are shifted in from the left. The sign bit (the leftmost bit) does not

change since the new leftmost bit has the same value as the old leftmost bit. As a result, the term "sign-propagating" was coined.

4.3. Comparison Operators

```
1- a == b                 // equals

2- a === b               // strict equal also check type

3- a != b                 // not equal

4- a != b                 // unequal

5- a !== b                // strict unequal

6- a < b   a > b          // less and greater than

7- a <= b  a >= b        // less or equal, greater or eq
```

4.4. Logical Operators

```
1- !                        // logical not

2- a && b                 // logical and

3- a || b                 // logical or
```

IF Else

5.1. If Else

```
if ((age >= 14) && (age < 19)) { // logical condition

status = "Eligible.";// executed if condition is true

}
else {               // else block is optional

status = "Not eligible.";//execute condition is false

}
```

Switch Statement

6.1. Switch

```
switch (new Date().getDay()) { // input is current day

          case 6:                 // if (day == 6)
           text = "Saturday";
           break;

          case 0:                 // if (day == 0)
           text = "Sunday";
           break;

         default:                  // else...
           text = "Whatever";
        }
```

Loops

7.1. For Loop

```
    for (let i = 0; i < 10; i++) {
        console.log("Run",i);
        }
```

7.2. While Loop

```
    let i = 1;           // initialize

    while (i < 100) {// enters if statement is true

    i *= 2;             // increment to avoid infinite loop
    console.log(i + ", ");
    }
```

7.3. Do While Loop

```
    let i = 1;      // initialize

    do {       // enters cycle at least once
    i *= 2;         // increment to avoid infinite loop

    console.log(i + ", ");

    } while (i < 100)  /* repeats cycle if statement is true at
the end*/
```

7.4. Break

```
    for (var i = 0; i < 10; i++) {

    if (i == 5) { break; }      // stops and exits the cycle

    console.log(i + ", ");      // last output number is 4
    }
```

7.5. Continue

```
    for (var i = 0; i < 10; i++) {

        if (i == 5) { continue; } /* skips the rest of the
cycle*/
        document.write(i + ", ");        // skips 5

        }
```

Arrays

8.1. Array

```
1- var arr = ["Shirt", "Pent", "Shoes"];

//Decleration
2- var arr = new Array("Shirt", "Pent", "Shoes");

// access value at index, first item being [0]
3-alert(arr[1]);

// change the first item
4- arr[0] = "Lamp";

//Parsing with array.length
5- for (var i = 0; i < arr.length; i++) {
    console.log(arr[i]);
}
```

8.2. Methods

```
// convert to string: results "Shirt,Pent,Shoes"
• arr.toString();

//join: "Shirt * Pent * Shoes"
• arr.join(" * ");

//remove last element
• arr.pop();

//add new elements to the end
• arr.push("Hoodie");
```

```
//Same as push
• arr[arr.length]= "Hoodie";

//remove first element
• arr.shift();

//add new element to the beginning
• arr.unshift("Hoodie");

//change element to undefined (not recommended)
• delete arr[0]

//add elements(where,how many remove,element list)
• arr.splice(2,0,"Light","Glasses");

//Join two arrays(arr are followed by cats and
birds)
• let animals = arr.concat(cats,birds);

//elements from [1] to [4-1]
• arr.slice(1,4);

//sort array alphabetically
• arr.sort();
```

```
//sort array in descending order
• arr.reverse();

//numeric sort
• x.sort(function(a,b){return a-b}


//numeric descending sort
• x.sort(function(a,b){return b-a}
```

JavaScript Functions

9.1. Functions For throwing Data as Output

`1- prompt():` This function is used for creating a dialogue box for taking input from the user.

`2- alert()`: This function is used for outputting information in an alert box in the browser window.

`3- console.log():` This function is used for writing data to the browser's console and is used for the purpose of debugging code by developers.

`4- document.write():` This function is used for writing straight to our HTML document.

`5- confirm():` This function is used for opening up a yes or no dialogue box and for returning a Boolean value depending upon the user's click.

9.2. Global Functions

Every browser that can run JavaScript has a set of global functions built-in.

`1- parseFloat():` This function is used for parsing the argument passed to it and returning a floating-point number.

`2- parseInt():` This function is used for parsing the argument passed to it and returning an integral number.

`3- encodeURI():` This function is used for encoding a URI into a UTF-8 encoding scheme.

`4- decodeURI():` This function is used for decoding a Uniform Resource Identifier (URI) made by encodeURI() function or similar functions.

`5- encodeURIComponent():` This function is used for the same purpose as encodeURI() only for URI components.

`6- decodeURIComponent():` This function is used for decoding a URI component.

`7- isNaN():` This function is used for determining if a given value is Not a Number or not.

`8- Number():` This function is used for returning a number converted from what is passed as an argument to it.

`9- eval():` This function is used for evaluating JavaScript programs presented as strings.

`10- isFinite():` This function is used for determining if a passed value is finite or not.

Scope

10.1. Scope

The accessibility or visibility of variables in JavaScript is referred to as scope. That is, which sections of the program can access a given variable and where the variable can be seen.

10.2. Global Scope

The global scope includes any variable that is not contained within a function or block (a pair of curly braces). Global scope variables can be accessed from anywhere in the program.

```
var hello = 'Hello!';
function sayHello() {
 console.log(hello);
}
// 'Hello!' gets logged
sayHello();
```

10.3. Local or function Scope

Variables declared inside a function are local variables. They can only be accessed from within that function; they are not accessible from outside code.

```
function sayHello() {
    var hello = 'Hello!';
    console.log(hello);
   }
   // 'Hello!' gets logged
   sayHello();
```

10.4. Block Scope

Unlike var variables,let and const variables can be scoped to the nearest pair of curly brackets in ES6. They can't be reached from outside that pair of curly braces, which means they can't be accessed from the outside.

```
{
    let hello = 'Hello!';
    var language = 'Urdu';
    console.log(hello); // 'Hello!' gets logged
   }
   console.log(language); // 'Urdu!' gets logged
   console.log(hello); // Uncaught ReferenceError:
hello is not defined
```

Scope Chain

11.1. Scope Chain

When a variable is used in JavaScript, the JavaScript engine searches the current scope for the variable's value. If it can't find the variable in the inner scope, it will look into the outer scope until it finds it or reaches the global scope.

If it still can't identify the variable, it will either return an error or implicitly declare the variable in the global scope (if not in strict mode).

```
let a = 'a';
function foo() {
let b = 'b';
console.log(b); // 'b' gets logged
console.log(a); // 'a' gets logged
randomNumber = 33;
console.log(randomNumber);  // 33 gets logged
}
foo();
```

When the function foo() is called, the JavaScript engine searches for the 'b' variable in the current scope and finds it. Then it looks for the 'a' variable in the current scope, which it can't find, so it moves on to the outer scope, where it finds it (i.e. global scope).

After that, we assign 33 to the 'randomNumber' variable, causing the JavaScript engine to search for it first in the current scope, then in the outer scope.

If the script isn't in strict mode, the engine will either create a new variable called randomNumber and assign 33 to it, or it will return an error (if not in strict mode). As a result, the engine will traverse the scope chain till the time when a variable is found.

JavaScript Hoisting

12.1. JavaScript Hoisting

Prior to executing the code, the interpreter appears to relocate the declarations of functions, variables, and classes to the top of their scope using a process known as Hoisting in JavaScript. Functions can be securely utilized in code before they have been declared thanks to hoisting. Variable and class declarations are likewise hoisted, allowing them to be referenced prior to declaration. It should be noted that doing so can result in unforeseen mistakes and is not recommended.

12.2. Function Hoisting

Hoisting has the advantage of allowing you to use a function before declaring it in your code as shown in the code snippet given below. Without function hoisting, we would have to first write down the function display and only then can we call it.

```
display("Abdullah");
function display(inputString) {
console.log(inputString); // 'Abdullah' gets logged
    }
```

12.3. Variable Hoisting

You can use a variable in code before it is defined and/or initialized because hoisting works with variables as well. JavaScript, however, only hoists declarations, not initializations! Even if the variable was initially initialized then defined, or declared and initialized on the same line, initialization does not occur until the associated line of code is run. The variable has its default initialization till that point in the execution is reached (undefined for a variable declared using var, otherwise uninitialized).

```
console.log(x) /*'undefined' is logged from hoisted
var declaration (instead of 7)*/

var x // Declaration of variable x

x = 7; // Initialization of variable x to a value 7

console.log(x); /* 7 is logged post the line with
initialization's execution.*/
```

JavaScript Strings

13.1. Escape Sequence or Escape Characters

```
1-  \\' – Single quotes

2-  \\" – Double quotes

3-  \\t – Horizontal tab

4-  \\v – Vertical tab

5-  \\\\ – Backslash

6-  \\b – Backspace

7-  \\f – Form feed

8-  \\n – Newline

9-  \\r – Carriage return
```

13.2. String Methods

`1- toLowerCase()` — This method is used for converting strings to lower case.

`2- toUpperCase()` — This method is used for converting strings to upper case

`3- charAt()` — This method is used for returning the character at a particular index of a string

`4- charCodeAt()` — This method is used for returning to us the Unicode of the character at a given index

`5- fromCharCode()` — This method is used for returning a string made from a particular sequence of UTF-16 code units

`6- concat()` — This method is used for concatenating or joining multiple strings into a single string

`7- match()` — This method is used for retrieving the matches of a string against a pattern string which is provided

`8- replace()` — This method is used for finding and replacing a given text in the string

`9- indexOf()` — This method is used for providing the index of the first appearance of a given text inside the string

`10- lastIndexOf()` — This method is similar to the indexOf() methods and only differs in the fact that it searches for the last occurrence of the character and searches backwards

`11- search()` — This method is used for executing a search for a matching text and returning the index of the searched string

`12- substr()` — This method is pretty much the same as the slice() method but the extraction of a substring in it depends on a given number of characters

`13- slice()` — This method is used for extracting an area of the string and returning it

`14- split()` — This method is used for splitting a string object into an array of strings at a particular index

`15- substring()` — Even this method is almost the same as the slice() method but it does not allow negative positions

`16- valueOf()` — This method is used for returning the primitive value (one without any properties or methods) of a string object.

Lexical Scope

14.1. Lexical Scope

when a inner function is access the variable of outer function then its called lexical scope.

```
function Outer() {
  var a = 10;
  function inner() {
    return a;
  }
  return inner();
}
console.log(Outer());
```

Closure Scope

15.1. Closure Scope

when a inner function is access the variable of outer function and remember the variables simply we can say that the variable of outer function is access in inner function on run time.

```
    function Outer() {
        var a = 10;
        function inner() {
          return a;
        }
        return inner;
      }
      let innerFunction= Outer();
      console.log(innerFunction());
```

Document Object Modal(DOM)

16.1. Node Properties

- attributes — Gets a live list of all the
- characteristics associated with an element.
- baseURI — Returns an HTML element's absolute base URL.
- childNodes — Returns a list of the child nodes of an element.
- firstChild — Returns the element's first child node.
- lastChild — An element's final child node
- nextSibling — Returns the next node in
- the same node tree level as the current node.
- nodeName —Returns a node's name.
- nodeType — Returns the node's type.
- nodeValue — Sets or returns a node's value.
- ownerDocument — This node's top-level document object.
- parentNode — Returns the element's parent node.
- previousSibling — Gets the node that comes before the current one.
- textContent — Sets or returns a node's and its descendants' textual content.

16.2. Node Methods

- cloneNode() — is a function that duplicates an HTML element.
- compareDocumentPosition() — Compares two elements' document positions.
- getFeature() returns an object that implements the APIs of a feature.
- hasAttributes() — If an element has any attributes, it returns true; otherwise, it returns false.
- hasChildNodes() — If an element has any child nodes, it returns true; otherwise, it returns false.
- insertBefore() — Adds a new child node to the left of an existing child node.
- isDefaultNamespace() — returns true if a given namespaceURI is the default, and false otherwise.
- isEqualNode() — Determines whether two elements are the same.
- isSameNode() — Determines whether two elements belong to the same node.
- isSupported() — Returns true if the element supports the provided feature.
- lookupNamespaceURI() — Returns the namespace URI for a specific node.

- ❖ lookupPrefix() — If the prefix for a given namespace URI is present, lookupPrefix() returns a DOMString containing it.
- ❖ normalize() — In an element, joins neighboring text nodes and removes empty text nodes.
- ❖ removeChild() — Removes a child node from an element using the Child() method.
- ❖ replaceChild() — In an element, this function replaces a child node.
- ❖ appendChild() — Adds a new child node as the last child node to an element.

16.3. Elements Methods

- ❖ getAttribute() — Returns the value of an element node's provided attribute.
- ❖ getAttributeNS() — Returns the string value of an attribute with the namespace and name supplied.
- ❖ getAttributeNode() — Returns the attribute node supplied.
- ❖ getAttributeNodeNS() — Returns the attribute node for the specified namespace and name for the attribute.
- ❖ getElementsByTagName() — Returns a list of all child elements whose tag name is supplied.

- **getElementsByTagNameNS() — Returns a live HTMLCollection of items belonging to the provided namespace with a certain tag name.**
- **hasAttribute() — If an element has any attributes, it returns true; otherwise, it returns false.**
- **hasAttributeNS() — Returns true or false depending on whether the current element in a particular namespace has the supplied attribute.**
- **removeAttribute() — Removes an element's supplied attribute.**
- **removeAttributeNS() — Removes an attribute from an element in a specific namespace.**
- **setAttributeNode() — Sets or modifies an attribute node.**
- **setAttributeNodeNS() — Sets a new namespaced attribute node to an element with setAttributeNodeNS().**

Number and Mathematics

17.1. Number properties

1. MAX VALUE — The maximum numeric value that JavaScript can represent.
2. NaN — The "Not-a-Number" value is NaN.
3. NEGATIVE INFINITY – The value of Infinity is negative.
4. POSITIVE INFINITY – Infinity value that is positive.
5. MIN VALUE — The smallest positive numeric value that JavaScript can represent.

17.2. Number Methods

1. toString() — Returns a string representation of an integer.
2. toFixed() — Returns a number's string with a specified number of decimals.
3. toPrecision() — Converts a number to a string of a specified length.
4. valueOf() — Returns a number in its original form.
5. toExponential() — Returns a rounded number written in exponential notation as a string.

17.3. Math Properties

1. E — Euler's number is E.
2. SQRT1_2 — 1/2 square root
3. SQRT2 stands for square root of two.
4. LOG2E — E's base 2 logarithm
5. LN2 — The natural logarithm of 2 is LN2.
6. LN10 — The natural logarithm of ten is LN10.

7. LOG10E — E's base ten logarithm
8. PI — PI stands for Pianistic Integer.

17.4. Math Methods

1. exp(x) — Ex's value
2. floor(x) — x's value rounded to the nearest integer.
3. log(x) — The natural logarithm (base E) of x is log(x).
4. abs(x) — Returns the value of x in its absolute (positive) form.
5. acos(x) — In radians, the arccosine of x.
6. asin(x) — In radians, the arcsine of x.
7. pow(x,y) — x to the power of y
8. random() — Returns a number between 0 and 1 at random.
9. round(x) — Rounds the value of x to the nearest integer.
10. sin(x) — The sine of x is sin(x) (x is in radians)
11. sqrt(x) — x's square root
12. tan(x) — The angle's tangent
13. atan(x) is the numeric value of the arctangent of x.
14. atan2(y,x) — Arctangent of its arguments' quotient
15. ceil(x) — x's value rounded to the next integer
16. cos(x) – The cosine of x is cos(x) (x is in radians)
17. max(x,y,z,...,n) — Returns the highest-valued number.
18. min(x,y,z,...,n) — The number with the lowest value is the same as the number with the highest value.

Date Objects

18.1. Setting Dates

1. Date() — Returns a new date object that contains the current date and time.

2. Date(1993, 6, 19, 5, 12, 50, 32) — We can create a custom date object with the pattern as Year, month, day, hour, minutes, seconds, and milliseconds are represented by the numbers. Except for the year and month, we can omit anything we like.

3. Date("1999-12-22") — Date as a string declaration

18.2. Getting the values of Time and Date

1. getDate() returns the month's day as a number (1-31)

2. getTime() — Get the milliseconds since January 1, 1970

3. getUTCDate() returns the month's day (day) in the supplied date in universal time (also available for day, month, full year, hours, minutes etc.)

4. getMilliseconds() — This function returns the milliseconds (0-999)

5. getMinutes() — Returns the current minute (0-59)

6. getMonth() returns the current month as a number (0-11)

7. parse — It returns the number of milliseconds since January 1, 1970 from a string representation of a date.

8. getDay() returns a number representing the weekday (0-6)

9. getFullYear() returns the current year as a four-digit value (yyyy)

10. getHours() — Returns the current hour (0-23)

11. getSeconds() — Returns the second number (0-59)

18.3. Setting part of Dates

1. setDate() — Returns the current date as a number (1-31)
2. setFullYear() — This function sets the year (optionally month and day)
3. setMonth() — This function sets the month (0-11)
4. setSeconds() — This function sets the seconds (0-59)
5. setTime() — This function is used to set the time (milliseconds since January 1, 1970)
6. setMinutes() — This function sets the minutes (0-59)
7. setUTCDate() — According to universal time, sets the day of the month for a given date (also available for day, month, full-year, hours, minutes etc.)
8. setHours() — Changes the time (0-23)
9. setMilliseconds() — This function sets the milliseconds (0-999)

Browser Objects

19.1. Window Properties

- history — Provides the window's History object.
- innerHeight — The content area of a window's inner height.
- innerWidth — The content area's inner width.
- closed — Returns true or false depending on whether or not a window been closed.
- pageXOffset — The number of pixels offset from the centre of the page.The current document has been horizontally scrolled.
- pageYOffset — The number of pixels offset from the centre of the page.The document has been vertically scrolled.
- navigator — Returns the window's Navigator object.
- opener — Returns a reference to the window that created the window.
- outerHeight — A window's total height, including toolbars and scrollbars.
- outerWidth — A window's outside width, including toolbars and scrollbars.
- defaultStatus — Changes or restores the default text in a window's status bar.
- document — Returns the window's document object.
- frames — All <iframe> elements in the current window are returned by frames.
- length — Determine how many iframe> elements are in the window.
- location — Returns the window's location object.
- name — Sets or retrieves a window's name.
- parent — The current window's parent window is called parent.

- screen — Returns the window's Screen object.
- screenLeft — The window's horizontal coordinate (relative to the screen)
- screenTop — The window's vertical coordinate.
- self — Returns the window that is currently open.
- status — Changes or restores the text in a window's status bar.
- top — Returns the browser window that is currently at the top of the screen.
- screenX — Identical to screenLeft, but required by some browsers
- screenY — Identical to screenTop, but required by some browsers

19.2. Methods which work on user browser window

- alert() — Shows a message and an OK button in an alert box.
- setInterval() — Calls a function or evaluates an expression at intervals defined by the user.
- setTimeout() — After a specified interval, calls a function or evaluates an expression.
- clearInterval() — Removes a timer that was started with setInterval() ()
- clearTimeout() — Removes the timer that was set with setTimeout() ()
- open() — This method creates a new browser window.
- print() — Prints the current window's content.
- blur() — Removes the current window's focus.
- moveBy() — Repositions a window with respect to its present position.
- moveTo() — This function moves a window to a specific location.
- close() — This function closes the currently open window.

- confirm() — Shows a dialogue box with a message and buttons to OK and Cancel.
- focus() — Sets the current window's focus.
- scrollBy() — Scrolls the document by a certain amount of pixels.
- scrollTo() — Scrolls the document to the supplied coordinates with the scrollTo() method.
- prompt() — Shows a conversation window asking for feedback from the visitor.
- resizeBy() — Resizes the window by the number of pixels supplied.
- resizeTo() — Resizes the window to the width and height supplied.
- stop() — This function prevents the window from loading.

19.3. Screen Properties

- height — The screen's entire height.
- pixelDepth — The screen's color resolution in bits per pixel.
- width — The screen's entire width.
- colorDepth — Gets the color palette's bit depth for showing images.
- availableHeight — Returns the screen's height (excluding the Windows Taskbar).
- availableWidth — Returns the screen's width (excluding the Windows Taskbar)

JavaScript Events

20.1. Mouse Events

- onclick – When a user clicks on an element, an event is triggered.
- oncontextmenu — When a user right-clicks on an element, a context menu appears.
- ondblclick — When a user double-clicks on an element, it is called ondblclick.
- onmousedown — When a user moves their mouse over an element, it is called onmousedown.
- onmouseenter — The mouse pointer is moved to a certain element.
- onmouseleave — The pointer leaves an element.
- onmousemove — When the pointer is over an element, it moves.
- onmouseover — When the cursor is moved onto an element or one of its descendants, the term onmouseover is used.
- onmouseout — When the user moves the mouse cursor away from element or one of its descendants, it is called onmouseout.
- onmouseup — When a user releases a mouse button while hovering over an element, it is known as onmouseup.

20.2. Form Events

- onblur — When an element loses focus, it is called onblur.
- onchange — A form element's content changes.(for the input>, select>, and textarea> elements)
- onfocus – An aspect is brought into focus.

- onfocusin — When an element is ready to become the centre of attention.
- onfocusout —The element is about to lose focus.
- oninput — When a user inputs something into an element, it's called oninput.
- oninvalid — If an element isn't valid, it's called oninvalid.
- onreset — The state of a form is reset.
- onsearch — A user enters text into a search field (for input="search">).
- onselect — The user chooses some text (input> and textarea>).
- onsubmit — A form is filled out and submitted.

20.3. Drag Events

- ondrag — When an element is dragged, it is called ondrag.
- ondragend — The element has been dragged to its final position.
- ondragenter — When a dragged element enters a drop target, it is called ondragenter.
- ondragleave — When a dragged element departs the drop target, it is called ondragleave.
- ondragover — The dropped element is on top of the dragged element.
- ondragstart — The user begins dragging an element.
- ondrop — When a dragged element is dropped on a drop target, it is called ondrop.

20.4. Keyboard Events

- onkeydown — When the user presses a key down
- onkeypress — When the user begins to press a key.
- onkeyup — A key is released by the user.

20.5. Frame Events

- onabort — The loading of a media is aborted with onabort.
- onerror — When loading an external file, an error occurs.
- onpagehide – When a user leaves a webpage, it is called onpagehide.
- onpageshow — When the user navigates to a webpage
- onhashchange — The anchor component of a URL has been changed.
- onload — When an object has loaded
- onresize — The document view gets resized when onresize is called.
- onscroll — The scrollbar of an element is being scrolled.onbeforeunload — Before the document is due to be emptied, an event occurs.
- onunload — When a page is emptied, this event occurs.

20.6. Animation Events

- animationstart — The animation in CSS has begun.
- animationend — When a CSS animation is finished, it is called animationend.
- animationiteration — CSS animation is repeated using animationiteration.

20.7. Clipboard Events

- oncut — The content of an element is cut by the user.
- onpaste — When a user pastes material into an element, it is called onpaste.
- oncopy — The content of an element is copied by the user.

20.8. Media Events

- onloadeddata — Media data has been loaded
- onloadedmetadata — Metadata is loaded (such as size and duration).
- onloadstart — The browser begins looking for the media that has been specified.
- onabort — The loading of media has been halted.
- onerror — When an error occurs while loading an external file, the event onerror is triggered.
- onpause — Media is paused, either manually or automatically, by the user.
- onplay — The video or audio has begun or is no longer paused.
- onstalled — The browser is attempting to load the media, but it is not currently accessible.
- oncanplay — The browser has the ability to begin playing media (e.g. a file has buffered enough)
- oncanplaythrough — The browser is capable of playing media without pausing.
- ondurationchange — The media's duration changes.
- onended — The media's time has come to an end.

- onsuspend — The browser is not loading media on purpose.
- ontimeupdate — The situation has shifted (e.g. because of fast forward)
- onvolumechange — The volume of the media has changed (including mute)
- onwaiting — The media has taken a break, but it is anticipated to resume soon (for example, buffering)
- onplaying — Media that has been paused or halted for buffering is now playing.
- onprogress — The media is being downloaded by the browser.
- onratechange — The media's playback speed changes.
- onseeking — The user begins to move/skip.

20.9. Miscellaneous Events

- transitionend — When a CSS transition is complete, this event is triggered.
- onmessage — The event source has received a message.
- onpopstate — When the history of the window changes
- onshow — A <menu> element is shown as a context menu when it is onshow.
- onoffline — The browser switches to offline mode.
- ononline — The browser enters the online mode.
- ontouchcancel — The user's ability to touch the screen has been halted.
- ontouchstart — The touch-screen is activated by placing a finger on it.
- onstorage — An part of Web Storage has been upgraded.
- ontoggle — The user toggles the details> element open or closed.
- onwheel — The mouse wheel moves up and down when it passes over an element.

- **ontouchend — A touch-screen user's finger is withdrawn.**
- **ontouchmove — When a finger is dragged over the screen, it is called ontouchmove.**

Event Propagation

21.1. Event Propagation

Event propagation is a technique that governs how events propagate or travel through the DOM tree to reach their destination, as well as what happens to them once they arrive.

21.2. Capturing Phase

```
    <div onClick={() => console.log('outer div')}> //Run first
  <div onClick={() => console.log('middle div')}>  //Run second
    <div onClick={() => console.log('innermost div')}> //Run Third
      Click me!
    </div>
  </div>
</div>
```

21.3. Bubbling Phase

```
<div onClick={() => console.log('outer div')}> //Run Third
  <div onClick={() => console.log('middle div')}>  //Run second
    <div onClick={() => console.log('innermost div')}> //Run first
      Click me!
    </div>
  </div>
</div>
```

Errors

22.1. Error

```
try {    // block of code to try
      undefinedFunction();
      }
      catch(err) {  // block to handle errors
      console.log(err.message);
      }
```

22.2. Throw Error

```
throw "My error message";    // throw a text
```

22.3. Input Validation

```
var x = document.getElementById("mynum").value; // get input value
try {
if(x == "")  throw "empty";  // error cases
if(isNaN(x)) throw "not a number";
x = Number(x);
if(x > 10)   throw "too high";
}
catch(err) {          // if there's an error
document.write("Input is " + err);   // output error
console.error(err);      // write the error in console
}
finally {
document.write("</br />Done");/* executed regardless of the try /
catch result*/
      }
```

22.4. Error Name Values

- RangeError: A number is "out of range"
- ReferenceError: An illegal reference has occurred
- SyntaxError: A syntax error has occurred
- TypeError: A type error has occurred
- URIError: An encodeURI() error has occurred

Promises

23.1. Promises

```
function sum (a, b) {
      return Promise(function (resolve, reject) {
       setTimeout(function () {  // send the response after 1 second
         if (typeof a !== "number" || typeof b !== "number") {
         return reject(new TypeError("Inputs must be numbers"));
         }
         resolve(a + b);
       }, 1000);
      });
      }
      var myPromise = sum(10, 5);
      myPromsise.then(function (result) {
      document.write(" 10 + 5: ", result);
      return sum(null, "foo");    // Invalid data and return another promise
      }).then(function () {      // Won't be called because of the error
      }).catch(function (err) { // The catch handler is called instead, after
another second
      console.error(err);  // => Please provide two numbers to sum.
      });
```

23.2. States

- Pending
- Fulfilled
- Rejected

23.3. Properties

- Promise.length
- Promise.prototype

23.4. Methods

- Promise.all(iterable)
- Promise.race(iterable)
- Promise.reject(reason)
- Promise.resolve(value)

Deep Copy, Shallow Copy

24.1. Deep Copy

Deep copy is a copy of an object that is independent of the original object, meaning that modifying the copy will not affect the original object. A deep copy creates a new object with its own memory address, and copies the values of the properties of the original object into the new object.

```
let obj ={
        name:"Abdullah Ansari",
        city:{
            name:"Lahore"
        }
    }
    // DEEP COPY
    let obj2=JSON.parse(JSON.stringify(obj));
    obj2.name="Abdullah";
    obj2.city.name="Karachi";
    console.log("obj",obj)
    console.log("obj2",obj2)
```

24.2. Shallow Copy

```
let obj ={
        name:"Abdullah Ansari",
        city:{
            name:"Lahore"
        }
    }
    // SHAWLOW COPY
    let obj2=obj;
    obj2.name="Abdullah";
    obj2.city.name="Karachi";
    console.log("obj",obj)
    console.log("obj2",obj2)

    let obj3 ={
        name:"Abdullah Ansari",
        city:{
            name:"Lahore"
        }
    }
    //SPREAD OPEARATOR ONE LEVEL SHAWLOW COPY
    let obj4=[...obj3];
    obj4.name="Abdullah";
    obj4.city.name="Karachi";
    console.log("obj3",obj3)
    console.log("obj4",obj4)
```


{.js}
JavaScript
Thank You!