Javascript

```
<head>
  <script src="js-file.js" defer></script>
</head>
//defer for js to work after the html has been parsed
```

advice

```
- always remember to flush a var after use using null and do null checks for those variables
```

falsy values in js

```
null,0,"",undefined,-0,0n,Nan,false
```

numbers

- all JavaScript numbers are 64 but floating integers
- integer without a period accurate up to 15 digits
- string+number is a string in rest the numeric string is treated as a number
- **Nan** - type of Nan is number and all mathematical operations will result in NaN or string when added with a numeric string
- **(-)infinity** - value returned if outside the range. division by zero.
- `toString` - can be used to change base
- == - is for value and === is for reference
- `+` - unary plus converts non-numbers to numbers if possible
- string with `` allows to embed variable inside them using `${..}`
- **BigInt** - can be used by adding `n` at the end of a number
- `isNaN(a)` - can be used to check if a is number of not

data types

There are 8 basic data types in JavaScript.

- Seven primitive data types:

    - `number` for numbers of any kind: integer or floating-point, integers are limited by `±(253-1)`.
    - `bigint` for integer numbers of arbitrary length.
    - `string` for strings. A string may have zero or more characters, there's no separate single-character type.
    - `boolean` for `true`/`false`.
    - `null` for unknown values – a standalone type that has a single value `null`.
    - `undefined` for unassigned values – a standalone type that has a single value `undefined`.
    - `symbol` for unique identifiers.
- And one non-primitive data type:
    - `object` for more complex data structures.

The `typeof` operator allows us to see which type is stored in a variable.

- Usually used as `typeof x`, but `typeof(x)` is also possible.
- Returns a string with the name of the type, like `"string"`.
- For `null` returns `"object"` – this is an error in the language, it's not actually an object.

strings

- `at()` - extracting char in string
- `slice(start,end)` - for substring
- `toUpperCase(),trim(),trimStart(),trimEnd(),repeat(),replace(),replaceAll(),split(),`
- `padStart(till_len,"withString")`
- `text.replace(/MICROSOFT/i, "W3Schools");`

- prompt("afjelkakfa",defaultInput);

Arrays

- `arr.toString(),` - converts array to comma separated string
- `arr.join(" anySeprator ")` - joins array to form a string
- `arr.pop(),arr.shift()` - removes last and first element respectively
- `arr.push(),arr.unshift()` - adds a element at the back and front respectively
- `delete arr[index]` - operator which deletes at a particular index
- `arr.concat(arr1,arr2)` - returns a new array with arr,arr1,arr2 combined

- `arr.sort()`
- `arr.splice(startIndex,NoOfElementsToRemove,elementsToBeAdded....)` - removes a particular no. of elements starting from a specified index and adds the mentioned numbers which are not necessary
- `arr.slice(start,end),`
- **spread operator** -

```
//for combining arrays
const newArray = [ ... arr1,..arr2];
```

A cheat sheet of array methods:

- To add/remove elements:
  - `push(...items)` – adds items to the end,
  - `pop()` – extracts an item from the end,
  - `shift()` – extracts an item from the beginning,
  - `unshift(...items)` – adds items to the beginning.
  - `splice(pos, deleteCount, ...items)` – at index `pos` deletes `deleteCount` elements and inserts `items`.
  - `slice(start, end)` – creates a new array, copies elements from index `start` till `end` (not inclusive) into it.
  - `concat(...items)` – returns a new array: copies all members of the current one and adds `items` to it. If any of `items` is an array, then its elements are taken.
- To search among elements:
  - `indexOf/lastIndexOf(item, pos)` – look for `item` starting from position `pos`, and return the index or `-1` if not found.
  - `includes(value)` – returns `true` if the array has `value`, otherwise `false`.
  - `find/filter(func)` – filter elements through the function, return first/all values that make it return `true`.
  - `findIndex` is like `find`, but returns the index instead of a value.
- To iterate over elements:
  - `forEach(func)` – calls `func` for every element, does not return anything.
- To transform the array:
  - `map(func)` – creates a new array from results of calling `func` for every element.
  - `sort(func)` – sorts the array in-place, then returns it.
  - `reverse()` – reverses the array in-place, then returns it.
  - `split/join` – convert a string to array and back.

    - `reduce/reduceRight(func, initial)` – calculate a single value over the array by calling `func` for each element and passing an intermediate result between the calls.
- Additionally:
    - `Array.isArray(value)` checks `value` for being an array, if so returns `true`, otherwise `false`.
    - `Array.from()` - convert any convertible object to array where we can also use a mapping function

Please note that methods `sort`, `reverse` and `splice` modify the array itself.

when not to use arrow function

when you need this functionality or want to use argument object

looping through array

```
//for iterating over values of iterable objects
//like maps,sets,arrays,strings etc
for (i of array) {
        //logic
}
//this can be used for any iterable items in a
//object
for (index in array) {
        if (array.hasOwnProperty(index)) {
                console.log(index,array[index]);
        }
}
//call a function once for each array element
arr.forEach((value,index,array)=>{
        //function logic
})
//creates a new array by performing some operation
//on each element
let newArray = arr.map((value,index)=>{
        return value*value;
})
//filters a array based on a condition
let newArray = arr.filter((value)=>{
        if (value>6) return true;
        else return false;
})
```

Objects

```
const JsUser = {
        name: "hitesh", //keys are by default strings
        age : 18,
        location: "Jaipur",
        "full name": "vipul chauhan"
}
//to access a value there are two ways
JsUser["name];
JsUser.name;
//but for keys with spaces in them there is only one way
JsUser["full name"]
//this is a symbol
const mySum = Symbol("key1");
//to make a symbol as a key
const JsUser = {
        [mySum]: "my Key"
}
//and it could be accessed by only one way
JsUser[mySum]

//to freeze a object from changing its value
Object.freeze(JsUser)
//it won't give error if we try to change it

JsUser.greeting = function() {
        console.log(`Hello {this.name}`)
}
console.log(JsUser.greeting())
```

combining objects

```
const newObj = Object.assign({},obj1,obj2);
//or
const newObj = {...obj1,...obj2};
```

singleton object

```
const newObj = new Object();
//this type of creation will give a singleton object
```

common function

```
Object.keys(objName);
Object.values(objName);
obj.hasOwnProperty('id');
```

destructure

```
const obj = {
        name: "vipul",
        email: "vipul@4708"
}
const {email} = obj;
console.log(email)//now it can be directly accessed
const {email: id} = obj;
console.log(id);
```

rest operator

```
function calculate( ...num) {
        console.log(num);
}
//if i don't know how many values will be given to me
//this will print
```

DOM and BOM

DOM refers to the html page where all node are objects with there being three type of nodes

- text node
- element node
- comment node

```
document.body.childNode//will give Nodelist of the child nodes
document.body.children//will give htmlCollection of the element childs
document.body.firstElementChild
document.bldy.firstElementChild.nextElementSibling
document.bldy.firstElementChild.previousElementSibling
```

- whichever element is selected $0

```
let boxes = document.getElementsByClassName("box");
console.log(boxes);
document.getElementById("red").style.backgroundColor = "red";
document.querySelector(".box").style.backgroundColor = "yellow";
document.querySelectorAll(".box").forEach((value)=>{
        value.style.backgroundColor = "pink";
})
let divs = document.getElementsByTagName("div");
console.log(divs[3].matches("#red"));//check whether it matches with the
selector or not
console.log(divs[3].closest(".container"));//will give the closes element in
the heirarchy
console.log(document.querySelector(".container").contains(divs[2]));
document.querySelector(".box").style.borderRadius = "20px";
```

- **inserting and removing elements**
  when styling background-color won't work instead use backgroundColor

```
document.querySelector(".box").innerHtml
//innerHtml - gives html markup and innertext
//innerText - returns text as it appears on screen and ignores hidden text
//textContent - returns raw text without styles even if hidden
//getAttribute(attribute) - gives the value of the attribute
//hasAttriute(attribute) - method to check the existance of the attribute
//setAttribute(attribute) - sets the value of the attribute(give the value
as string)
//removeAttribute(attribute) - remove the attribute
//attributes - for all the attributes

let div = document.creatElement("div")'
div.className = "new";
div.innerHtml = "<span>hello</span>";
document.body.append(div);
//append(e)
//prepend(e)
//after(e)
//before(e)
//replaceWith(e)
document.body.remove;
//for removing node
```

- **insertAdjacentHtml/Text/Element**

```
document.body.insertAdjacentHtml("afterbegin","<div>hello</div>")
//beforebegin
//afterend
//beforeend
```

- `parentNode.insertBefore(newNode, referenceNode)` - inserts *newNode* into *parentNode* before *referenceNode*.
- **class name and class list**

```
elem.classList.add/remove("class");
elem.classList.toggle("class");
elem.classList.contains("class");
```

- `hidden=true` - hides the content
- `data-*` - are custom attributes which can be accessed using `.dataset` if an element has attribute `data-one` then it can accessed by `.dataset.one`

EVENTS

various events
https://developer.mozilla.org/en-US/docs/Web/API/Element > events

```
let button = document.getElementById("btn");
button.addEventListener("click",()=>{
        document.querySelector(".box").innerHTML = "You were clicked";
})
btn.addEventListener("click", function (e) {
  e.target.style.background = "blue";
});
```

event bubbling
when we listen to a event of a element its parent's event will also be called if it has the same event. to stop it we need stop propogation

```
document.querySelector(".child").addEventListener("click",(e)=>{
        //e.stopPropagation(); three events will be called without it
        alert("child was clicked");
})
document.querySelector(".container").addEventListener("click",(e)=>{
        e.stopPropagation();
        alert("container was clicked");
```

```
})
document.querySelector(".childContainer").addEventListener("click",(e)=>{
        e.stopPropagation();
        alert("childContainer was clicked");
})
```

page load
starting of the page

- `DOMContentLoaded` - html is loaded but not the external resources
- `load`
- `beforeunload` - can be used for confirmation whether to leave the page or not
- `unload`
  mouse events
  click,mouse up and down events always occur before a dblclick so we need to handle accordingly if both are registered
  mousemove should be removed when not needed
  The `event` object passed to the mouse event handler has a property called `button` that indicates which mouse button was pressed on the mouse to trigger the event.

The mouse button is represented by a number:

- 0: the main mouse button is pressed, usually the left button.
- 1: the auxiliary button is pressed, usually the middle button or the wheel button.
- 2: the secondary button is pressed, usually the right button.
- 3: the fourth button is pressed, usually the Browser Back button.
- 4: the fifth button is pressed, usually the *Browser Forward* button.

for **modifier keys** The `event` object has four Boolean properties, where each is set to `true` if the key is being held down or `false` if the key is not pressed.

```
if (e.shiftKey) keys.push('shift');
if (e.ctrlKey) keys.push('ctrl');
if (e.altKey) keys.push('alt');
if (e.metaKey) keys.push('meta');
```

- The `screenX` and `screenY` properties return the horizontal and vertical coordinates of the mouse pointer in screen coordinates.
- The `clientX` and `clientY` properties of the `event` object returns horizontal and vertical coordinates within the application's client area at which the mouse event occurred.

key events

- When you press a character key on the keyboard, the `keydown`, `keypress`, and `keyup` events are fired sequentially. However, if you press a non-character key, only the `keydown` and `keyup` events are fired.
- The keyboard `event` object has two important properties: `key` and `code` properties that allow you to detect which key has been pressed.
- The `key` property returns the value of the `key` pressed while the `code` represents a physical key on the keyboard.

event delegation

- Having a large number of event handlers will take up memory and degrade the performance of a page.
- The event delegation technique utilizes the event bubbling to handle the event at a higher level in the DOM than the element on which the event originated.

```
let menu = document.querySelector('#menu');

menu.addEventListener('click', (event) => {
    let target = event.target;

    switch(target.id) {
        case 'home':
            console.log('Home menu item was clicked');
            break;
        case 'dashboard':
            console.log('Dashboard menu item was clicked');
            break;
        case 'report':
            console.log('Report menu item was clicked');
            break;
    }
});
```

dispatch event

- Use the specific event constructor such as `MouseEvent` and call `dispatchEvent()` method on an element to generate an event from code.
- Use `event.isTrusted` to examine whether the event is generated from code or user actions.