JAVASCRIPT

Complete Handwritten notes with Syntax and Examples

PAGE NO. :
DATE : / /

JAVASCRIPT

- Emascript:- Emascript is a Standard on which javascript is based. (ES6).

- Javascript is used to make interactive website.
- Javascript is dynamically typed language, i.e. the values of the variables can be changed during the runtime.

- Variables:- Variables is a container that stores values

The values of the javascript can be changed during the execution of the program.

The values of a javascript variable can be changed during the execution of the program.

Var a = 7; → Litral

let a = 7; → => Declearing variables

Identifier Assignment operator.

PAGE NO. :
DATE : / /

- Rules for choosing Variable names:-

(i) Letters, digits, underscores & sign is allowed.
(ii). Must begin with a $, _ or a letter.
(iii) Javascript reserved words cannot be used as a variable name.
(iv) Javascript is case sensitive.

- Var Vs let :-

(i) Var is globally scoped while let & constant are block specific.

(ii) Var can be updated and redeclared within its scope.

(iii). let can be updated but not - redeclared.
(iv). const. can neither be updated nor be red-declared.
(v). const must be initialized during declaration, unlike var, and let.

PAGE NO. :
DATE : / /

- Primitive data types :-

(N) - Null
(N) - Number
(S) - Symbol
(S) - String
(B) - Boolean
(B) - Bigint
(U) - Undefined.

Ex:-

```
                                    //mm   bb . ss  u
let a = null;
let b = 345;
let c = true;
let d = BigInt("720")
let e = "Pratyush"
let f = Symbol("I am a nice person")

console.log(a, b, c, d, e, f)
console.log(typeof d)
```

PAGE NO. :
DATE : / /

- Object :-

```
const item = {

    "Pratyush" : true,
    "Bittle" : false,
    "Jagdish" : undefined

}

console.log(item["Pratyush"])
```

- operators :-

```
+
-
*
%
/
**
```

PAGE NO.:
DATE: / /

&& logical and
|| logical or
!& logical not.

- Conditional operator :-

(i). if statement
(ii). if ... else statement
(iii). if .. else if ... else statement

(iv) Switch case statement.

prompting :-

let a = prompt("What's your age")

alert :-

Alert("Contains virus")

Type casting :-

let a = prompt("Age?").
a = Number.parseInt(a).

- Ternary operator:-

```
console.log("You can", age<18 ? "not drive" : "drive");
```

- Loops:- Loops are used to perform repeated actions

Types of loops in javascript :-

(i) for loop
(ii) for in loop
(iii) for of loop
(iv) while loop
(v) do while loop

(i) Syntax for - for loop:-

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

PAGE NO. :
DATE : / /

Syntax for in loop :-

```
let marks={

    Pratyush : 24,
    Jagdish : 56,
    Sonal : 72,
    Rishit : 100,
}
for (let a in maks){

    console.log(a + marks[a]).    // Acessing keys.
```

- Functions in javascript :- Chunk of code that can be used over and over again.

```
function addition(x,y){

    return (x+y)/2
    }
let a=24
let b=25
console.log(addition(a,b)).
```

OR. you can pass the parameter here also

```
const sum = () => {
    return p+q
}
```

// Program to print the keys and marks of students:-

```
let marks = {
    Harry: 95,
    Pratyush: 76,
    Jagdish: 74,
    Sonal: 92,
}
for (let i = 0; i < Object.keys(marks).length; i++){
    console.log( Object.keys(marks)[i] + marks[Object.keys
}
```

PAGE NO. :
DATE : / /

- Strings:- collection of characters.

```
Let name = "Pratyush"
console.log("name.length)
```

- Template literals:- Use brackets instead of quotes to define a string.

```
let boy1 = "Pratyush"
let boy2 = "Jagdish"

let sentence = `${boy1} is a friend of ${boy2}`
console.log(sentence)
```

- Escape sequences in Javascript:-

\n → new line

\t → Tab

\r → Carriage return

PAGE NO. :
DATE : / /

- String methods:-

(i). string.length

(ii). string.toUpperCase() => console.log(string.toUp

(iii). string.toLowerCase()

(iv). string.slice(0, 2).
index

(v). String.replace("Pratyush", "Shrey")

(vi). String concatination:-

let friend = "Pratyush"

console.log(friend.concat("Jagdish"))

PAGE NO. :
DATE : / /

- Arrays :- Arrays are variables that hold more than one value. It can consist of different data types.

Ex :- let a = [1, 2, 3, 4, 7]

console.log(a.length) ⟶ length of array
console.log(typeof a) ⟶ Type of array : object

Array methods :-

(i). to string :-

```
let num = [1, 2, 3 4, 46].
let b = num.toString().
```

(ii) join :-

```
let c = num.join("_")
```

(iii). pop :-

```
let r = num.pop().
```

(iv). push :-

```
let r = num.push(56)
```

PAGE NO. :
DATE : / /

Shift - Removes first element and returns it.

(v). let r = num.shift()
console.log(r, nums)

(vi). unshift() - Adds element to beginning. returns new array length.

(vii). delete() - Deletes an element. But the length of the deleted array remains same as the original array.

(viii). concate():- joins 2 arrays:-

Ex:- let num = [1, 2, 3, 4, 5, 6, 7, 8, 9]
let num_extends = [11, 12, 13]
let newarray = concat num.concat(num_extends
console.log(new array).

(ix). sort():- Sorts elements alphabetically.

PAGE NO. :
DATE : / /

Sorting numbers in an array using sort. (in descending order).

```
                a,b
let compare = ()=>{
    return a-b
}
let num= [ 3, 4, 7, 8, 2, 1]
num.sort (compare)
console.log (num).
```

(x). Splice :- Splice can be used to add new elements to an array.

```
let num= [ 22, 33, 44, 56, 78 ]
num.splice ( 2, 1, 36, 48, 39)
```

position to add elements to remove

no. of no. to be added.

(xi) slice.() :- Slices the array.
It creates a new array.

Ex:-

```
let num = [561, 762, 232, 41].
let newNum = num.slice (3,5)
console.log (newNum)
```

- Applying for loop :-

```
let num = [3,54,1,2,4].
for (let i=0; i<num.length; i++) {

  console.log(num[i]).

}
```

- Printing square of elements using for each loop :-

```
let num= [2, 4, 6, 8, 10].
num.forEach((element) => {
    console.log(element * element)
})
```

- Array.from :- Used to create an array from any other object.

```
let name = "Pratyush"
let arr = Array.from(name)
console.log(arr)
```

- for of :- for of loop can be used to get the values from an array.

```
let num = [1,2,3,4,5,6,7].
for(let i in num){
    console.log(i)
}
```

PAGE NO. :
DATE : / /

- for in :-

```
let num = [1, 2, 3, 4, 5, 6];
for (let i in num){
  console.log(i)
}
```

- for each loop :- calls a function once for each array count.

```
const a = [1, 2, 3]
a.forEach((value, index, array) => {
        //function logic
});
```

- map() :- creates a new array by performing some operation on each array element.

```
const a = [1, 2, 3]
a.map((value, index, array) => {
        return value * value;
})
```

PAGE NO.:
DATE: / /

- filter :-

```
let arr =[ 4, 6, 65, 72]
let newarr = arr.filter ((a) => {

    return  a<10

})
console.log(newarr)
```

- Reduce :- Reduces an array to a single value.

```
const n =[1, 8, 2, 11]

let q2 = n.reduce((h1, h2) => {
    return  h1+h2

})
```

PAGE NO. :
DATE : / /

Add no numbers entered by user in an array.

```
let arr = [1, 24, 26, 29, 30].
let a = prompt("Enter a number").
a = Number.parseInt(a)
arr.push(a)
console.log(arr)
```

- Javascript in browser :- Javascript was initially created to make a web pages alive. JS can be written in a web page's HTML to make it interactive.

The browser has embedded engine - called Javascript or javascript runtime.

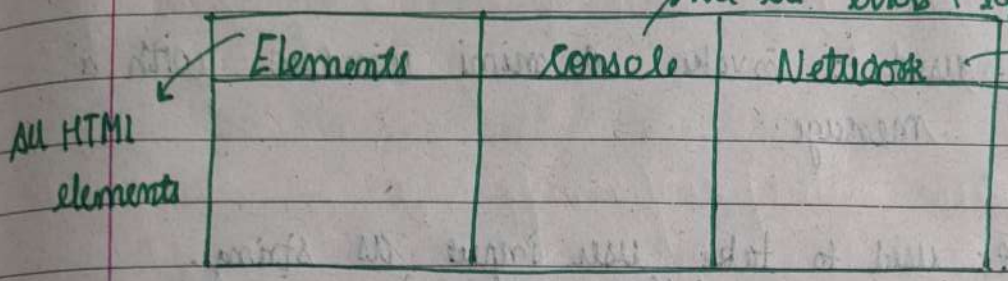
HTML :- Skeleton

CSS :- Styling

JS :- logic

PAGE NO. :
DATE : / /

- Developer tools :- Every browser has some developer tools.



- <Script>
  </Script>
  } Script tag is used to add javascript to a HTML website.

Or

<Script src="<address>.js"></Script>

- Javascript, console object :-

console.error("This is an error") → gives an error. (Check).

(i)- console.assert() → used to assert a condition.
(ii). console.clear() → used to clear the console.
(iii). console.log() → outputs the message to the console.
(iv). console.table() → Displays tabular data.
(v). console.warn() → Displays a warning.
(vi). console.info() → Displays the entered text as information.
(vii). console.time() → Gives the time taken for execution.
console.timeEnd()

PAGE NO.:
DATE: / /

- Interaction :- alert, prompt and confirm.

(i) alert :- used to invoke a mini window with a message.

(ii) prompt :- used to take user input as string.
ing = prompt ("Hi", "No") → default optional

(iii) confirm :- shows a message and waits for the user to press "ok" or cancel.
Returns true for ok and false for cancel.

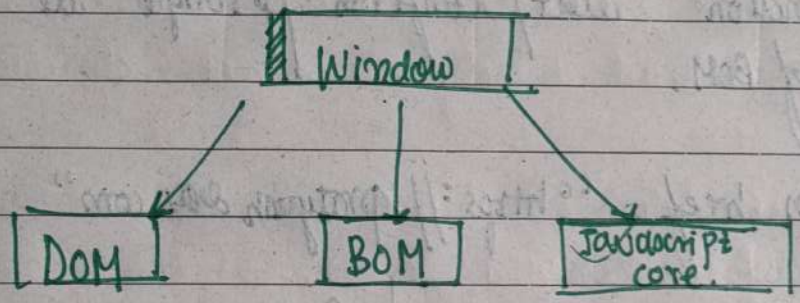
```
# alert ("Enter the value of a1")
let a = prompt ("Enter a here")
document.write(a)

# alert ("Enter the value of a")
let a = prompt ("Enter a", "724")
a = Number.parseInt(a)
alert ("You have entered a of type" + (typeof a))
let write = confirm ("Do you want to write it on the page")
if (write) {
    document.write(a).
}
```

PAGE NO. :
DATE : / /

else {
document.write ("Please allow me to write");
}.

- Window object, BoM and DOM:-



default value.

Window object represents browser window and provides methods to control it. It is a global object.

- Document Object model (DOM):- Dom represents the page content as HTML.

document.body → Page body as JS object.

document.body.style.background = "green" → changes the background colour.

PAGE NO.:
DATE: / /

- Browser object model (BOM):- The browser object model represents additional objects provided by the browser (host environment) for working with everything except the document.

The function alert/confirm/prompt are also part of BOM.

location.href = "https://pratyush.dev.com"

↳ Redirect to another URL.

- DOM- Dom tree refers to the HTML page where all the nodes are objects. There are 3 main type of nodes in DOM:

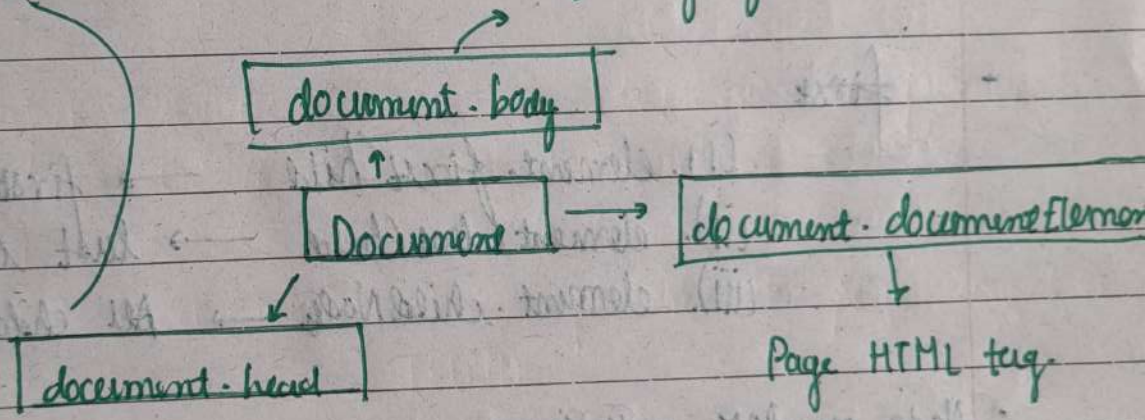
(i) Text nodes
(ii) Element nodes
(iii) comment nodes

PAGE NO. :
DATE : / /

In HTML page, <HTML> is at the root and <head> and <body> are its children, etc.

- Walking the DOM:-

```
<html>
<head>
<title> Hello </title>
</head>
<body>
  ,
  '
  '
  '
  '
  '
  '
</body>
</html>
```



PAGE NO. :
DATE : / /

- Children of an element:- Direct as well as deeply nested elements of an element are called its children.

child nodes:- Elements that are direct children.
Ex:- Head & body are direct children of <HTML>

Descendant nodes:- All nested elements, children, their children and so on.

- ~~first~~
  (i). element.firstChild ⟶ first child element
  (ii). element.lastChild ⟶ last child element
  (iii). element.childNodes ⟶ All child nodes

- Note on DOM collections:-
  (i) They are read only.
  (ii). They are live.
  elem.childNodes Variables (reference) will automatically update if childNodes of elem is changed.
  (iii) They are iterable using for...of loop.

PAGE NO. :
DATE : / /

- Sibilings and the parent:- Sibilings are nodes that are childrens of the same parent.

first child:-

```
let b = document.body
console.log("First child of b is:", b.firstChild)
console.log("First element child of b is:", b.firstElementChild)
```

- Element only navigation:- Sometimes, we dont want text or comment nodes. Some linkings only take element into account.

(i) document.previousElementSibling → Previous sibiling which is an Element.

(ii) . document.nextElementSibling
(iii) . document.firstElementChild
(iv). document.lastElementChild.

PAGE NO. :
DATE : / /

- Table Navigation :-

let t = document.body.firstElementChild.firstElementChild.
console.log(t)

(i). table.rows
(ii). table.caption
(iii). table.tHead
(iv). table.tFoot
(v). table.tbodies
(vi). tbody.rows
(vii). t.rows[0].cellIndex
(viii). t.sectionRowIndex
(ix). t.rowIndex

- Searching the DOM:- DOM is the navigation properties are helpful when the elements are close to each other.

PAGE NO.:
DATE: / /

(i) document.getElementById :- Used to get element with given id attribute.

let title = document.getElementById("<Name of the id")
title.style.color = "Red"

(ii). document.querySelectorAll :- Returns all elements inside an element matching css Selector.

let title = document.querySelectorAll(".card-title").
console.log(title).
title[0].style.color = "blue"
title[1].style.color = "green"

(iii). document.getElementsByTagName :- Returns elements with the given tag name.

(i). element.matches(css) → To check if element matches given CSS selector.

Ex:- let id1 = document.getElementsById("<id name>")
console.log(id1.matches(".class"))

(ii) elem.closest(CSS) → To look for the nearest ancestor that matches the given css-selector. The elem itself is also checked

(iii). elemA.contains(elemB) → Returns True if element B is inside element A or when elemA == elemB

• console.dir:function:- Console.log

PAGE NO. :
DATE : / /

(i). console.dir function:- console.log shows the element DOM tree.

console.dir shows the element as an object with its properties.

(ii) tag name/node name:- Used to read tag name of an element.

tagname - only exists for element nodes
nodename - defined for any node.

(iii). Inner HTML / outer HTML:- The inner HTML property allows to get the HTML inside the document as a string.

The outer HTML ~~allows to get the~~ property contains the full HTML, inner HTML + element itself

(iv) text content:- Provides acess to the text inside the element.
only text - all tags.

console.log (document.body.textContent)

(V) ## how to see hidden attributes:-

$0.hidden = false

- Attribute methods:-

(i). elem.hasAttribute(name) :- Method to check for existence of an attribute.

(ii) elem.getAttribute(name)

(iii). elem.setAttribute(name, value)

(iv) elem.removeAttribute(name)

(v). elem.attributes :- Method to get all the collection of attributes.

PAGE NO. :
DATE : / /

→ Insertion methods :- We can insert free elements in DOM.

(i) let div = document.createElement('div') // creates.

(ii) div.className = "alert" // set class.

(iii) div.innerHTML = " hello "

(iv) document.body.appen(div)

— × —

(i) node.append(a) → append at the end of the node.

(ii) node.prepend(a) → Insert at the beginning of the node.

(iii) node.before(a) → Insert before node.

(iv) node.after(a) → Insert after node.

(v) node.replaceWith(a) → Replaces node with the given node.

PAGE NO. :
DATE : / /

- Class name and class list:-

(i). elem.classList.add/remove("class") - Adds or removes a class

(ii). elem.classList.toggle("class") - Adds class if it doesn't exists, otherwise removes it.

(iii). elem.classList.contains("class") - checks for the given class, returns true/false

→ setInterval and setTimeout:-

```
let a = setTimeout(function() {
    alert("I am inside of settimeout")
}, 5000)
console.log(a)
```

clearTimeout(a)

PAGE NO. :
DATE : / /

Syntax :-

let timerId = setTimeout (function, <delay>, <arg1>, <arg2>)

→ Browser Event :- An event is a signal that something has happened. All the DOM nodes generate such signals.

(i). Mouse event :- click, context name (right click), mouseover / mouseout, mousedown / mouse up, move move

(ii) Keyboard events :- Keyboard down, up

Syntax :-

<div class = "container" on mouseenter = "alert ('contains is overlapped by mouse')>

 <button on click> click me </button>

PAGE NO. :
DATE : / /

→ addEventListeners and removeEventListeners.

addEventListner is used to assign multiple handlers to an event.

element.addEventListener(event, handler).

→ Callbacks, promises & async/await:

Asynchronous actions are the actions that we initiate now and they finish later. eg. SetTimeout. synchronous actions are the actions that initiate and finish one-by-one.

- Callback functions:- A function is a function passed into another function as an argument, which is then involved inside the outer function to complete an action.

PAGE NO. :
DATE : / /

Ex:-

```
function loadScript (src, callback) {
  let script = document.createElement('script')
  script.src = src
  script.onload = () => (callback (script)
  document.head.append(script)
}
```

loadScript ("url", function <that you have created>).

→ Handling errors:- callbacks

We can handle errors :-

```
function loadScript (src, callback) {
  - - -
  - - -
  script.onload = () => callback (null, script);
  script.onerror = () => callback (new Error('failed'));
}
```

Then inside the ~~boot~~ load Script call:-

```
loadScript('cdn/pratyush', function (error, script) {
    - - - - .
    - - - -
    if (error){
        // handle error
    } else {
        // script loaded
    }
});
```

Date: / / Page No.:

PAGE NO.:
DATE: / /

→ Pyramid of Doom:- When we have callback inside callbacks the code gets difficult to manage.

```
loadScript (' ..... ') {

    loadScript ...

        loadScript ...

            loadScript ...
```

→ Introduction to promise:- The solution to the callback hell is promises. A promise is a "promise of code execution." The code either executes or fails, in both the cases the

Syntax:-

```
let promise = new Promise (function (resolve, reject) {

    // execution

});
```

resolve (Value) → If the job is finished Successfully.

reject (error) → If the job fails.

- Properties of promise constructor :-

(i) State :- Initially pending, then changes to either "fulfilled" when resolve is called or "rejected" when reject is called.

(ii) result :- Initially undefined, then changes to value if resolved or error when rejected.

```
#let p = new Promise( resolve, reject )=>{

    setTimeout( () => {
        console.log("fulfilled")
        resolve(true)
    }, 5000)

})
```

Date / / Page No.:

PAGE NO.:
DATE: / /

→ if there's an error :-

```
# let p = new Promise ((resolve, reject) => {
    console.log("Promise is pending")
    setTimeout(() => {
        reject(new Error("There's an error"));
```

- .catch() and .then().

(i). p.then(Value) => {
    console.log(Value)
});

(ii). p.catch(err) => {
    console.log("some error occured")
})

PAGE NO. :
DATE : / /

- Promise chaining :- We can chain promise and make them pass the resolved values to one another.

```
p.then (function (result) => {
        take (result); return 2;

    }) .then.....

Ex:- let p1 = new Promise ((resolve, reject) => {
        setTimeout (() => {
            console.log ("Resolved after 2 seconds")
            resolve (56)

        }, 2000)

    })

p1. then ((value) => {
        console.log (value)
        let p2 = new Promise ((resolve, reject) => {
            resolve ("Promise 2")
        })
        return p2.

    }). then ((value) => {
        console.log ("We are done").
```

PAGE NO.: 2
DATE: / /

→ Promise API :-

6 static methods of Promise class :-

(i). Promise.all (promises) :- Waits for all promises to resolve and returns the array of their results. If anyone fails, it becomes the error & all other results are ignored.

#. let promise_all = Promise.all([p1, p2, p3])

promise_all.then((value) => {
console.log(value)

})

(ii). Promise.allSettled (promises) :- Waits for all promises to settle and returns their results as an array of objects with status and value.

PAGE NO. :
DATE : / /

(iii). Promise.race(promise) :- Waits for the first promise to settle and its result/error becomes the outcome.

(iv). Promise.any(promises) :- Waits for the first promise to fulfill (not rejected), and its result becomes the outcome. Throws AggregateError if all the promises are rejected

(v). Promise.resolved(value) :- Makes a resolved promise with the given value

(vi). Promise.reject(error) :- Makes a rejected promise with the given error.

PAGE NO. :
DATE : / /

→ Async await :- A special syntax to work with promises to work with promises in javascript.

Syntax :-

```
async function code (){
    return 7;
}
```

An async function always returns a promise.

We can do something like this :-

```
code(). then (alert)
```

#

PAGE NO. :
DATE : / /

async function weather () {

let Delwea = new Promise ([resolve, reject) => {
setTimeout (() => {
resolve ("15 Deg")
}, 1000)

})

let Blrwea = new Promise ([resolve, reject) => {
setTimeout (() => {
resolve ("20 deg")
}, 5000)

})

let delhiW = await Delwea
let BlrW = await Blrwea
return [delhiW, BlrW]

}

PAGE NO.:
DATE: / /

```
console.log ("Welcome to weather control Room")
  let a= prakash ()
  a.then(value) => {
        console.log (value)
        }
```

→ Error Handling :-
At time our script can have errors. Usually a program fails when an error occurs.

We can use try catch :-

```
try {
  //try the code
}catch (err){
    // error handling
  }
```

PAGE NO. :
DATE : / /

- The Error objice & Custom Errors.

try {

Pratyush

} catch (error) {
console.log (error.name)
console.log (error.message)

}

- creating a custom error:-

try {
console.log (Pratyush) ;
throw new ReferenceError ("Pratysh is nice")

} catch (error) {
console.log (error.name)
console.log (error.message)
console.log (error.stack)

}

PAGE NO. :
DATE : / /

→ The finally clause:-

The try..... catch construct may have one more code clause : finally.

If it exists it runs in all cases:-

If there is a return in try, finally is executed just before the control returns to the outer code.

→ Fetch API:- JS can be used to send and ~~return~~ retrieve information on from the network (AJAX).

let promise = fetch (url, [options])

without options, a get request is sent.

#. let p = fetch ("src/url")
p.then ((value1) => {
return value1.json()
console.log(value1.status)
console.log(value1.ok)

}).then((value2) => {
console.log(value2)

})

Getting a response is a 2-stage process:-

1. (i) Status :- The http status, eg :- 200.

(ii) ok :- Boolean. true if the http status code is 200-299.

2. We need to call another method to access the body in diff. formats :-

response.text() :- Read and return the text
response.json() :- Parse the response as JSON.

PAGE NO.:
DATE: / /

Other methods are:- response.formData(), response.blob(), response.arrayBuffer() etc.

→ POST request:-

To make a post request, we need to use fetch options:-

(i) method → HTTP-method, eg. POST
(ii). body → the request body.

Syntax:-

```
let response = await fetch("url", {
    method: 'POST',
    headers: {
        'Content Type': 'application/json'
    },
    body: '{"a": "Pratyush"}'
});

let result = await response.json()
```

→ Javascript cookies:- Cookies are small strings of data. Stored directly in the browser.

Cookies are set by a web browser using the set-cookie HTTP-header. Next time when the request is sent to the same domain, the browser sends the cookie using the cookie HTTP.-header.

alert(document.cookie);

adding a cookie:-

document.cookie:- "name Pratyush 6000"

- let key = prompt("Enter your key")
  let value = prompt("Enter the value");
  document.cookie = `${key}=${value}`
  console.log(document.cookie);

PAGE NO. :
DATE : / /

- Local Storage:- Local storage is a web storage object which are not sent to server with each request. This data

In Console:-

localStorage.setItem("name", "Pratyush").

Methods provided by local storage:-

(i) setItem (key, value) → Store key / value pair.
(ii) getItem (key) → get the value by key
(iii) removeItem (key) → remove the key with its value.
(iv) clear() → Delete everything.
(v) key (index) → Get the key on the index position.

Imp notes:-

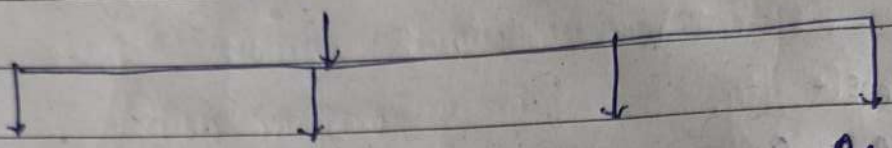
1. Both key and values must be string.

2. We can use 2 JSON methods to store objects in local storage

JSON.Stringify (object)
JSON.parse (string)

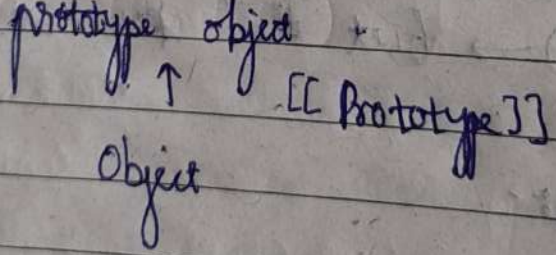
PAGE NO. :
DATE : / /

- Object oriented programming in javascript:-



Abstraction Encapsulation Inheritance Polymorphism

→ Prototype:-
Javascript objects have a special property called prototype property that is either null or references another object.



Ex:-

```
let a = {
    name: "Pratyush",
    language: "Javascript"
}

let p = {
    run: () => {
        alert("run")
    }
}

a.__proto__ = p
a.run()
```

- Classes and objects

In object oriented programming, a class is an extensible program code template for creating objects, providing initial values for state (class) and implementations of behaviour.

PAGE NO. :
DATE : /

Syntax :-

```
class MyClass{
    //class method
    constructor (){ - - - }
    method1 () { - - - 2
    method2 () { - - - - }
}
```

```
class Railway Form {
    submit (){
    alert ("Form submitted")
    }
    cancel (){
        alert ("This form is cancelled")
    }
}
```

```
submit (){
    alert("form submitted")
}
let pratyush = new RailwayForm()
let rishit = new Railwayform()

pratyush.submit()
prathyusha.submit()
rishit.cancel()

fill(name){
    this.name = name
}
}

let PratyushForm = new

Pratyush.fill("Pratyush")
```

PAGE NO.:
DATE: / /

→ Constructor:- A special function that creates and initializes an object instance of a class. A constructor gets called when an object is created using the new keyword.

Syntax:-

```
class RailwayForm{
    constructor(){
        console.log("constructor called")
    }
}
```

Example:-

```
class AirwayForm {

    constructor(givenname, planeno, address){
        console.log("constructa is called" + givenname + "  " +
    this.name = givenname.
    this.planeno = planeno
    this.address = address

    }
```

```
preview (){
    alert ("this.name + ": Your form for the plane
    number :" + this.planeno + " and your address is "
        + this.address)

}

Submit (){
    alert (this.name+": Your form is submitted for the
        plane number: " + this.planeno)

    }

}

let newForm = new AirForm ("Pratyush", 63725, "DEL AIRPORT"
planeno); newForm.preview()
newForm.submit()
```

PAGE NO.:
DATE: / /

Class

- Inheritance:- Class Inheritance is a way for one class to extend another class. This is done by using the extends keyword.

THE EXTENDS KEYWORD:-

extends keyword is used to extend another class.

Example:-

```
class Animal {
    constructor(name, color) {
        this.name = name
        this.color = color
    }
    run() {
        console.log(this.name + " is running")
    }
    shout() {
        console.log(this.name + " is shouting")
    }
}
```

```
class Monkey extends Animals {
    eatBanana () {
        console.log (this.name + " is eating banana")
    }
}

let ani = new Animal ("Brownie", "Brown")
let m = new Monkey ("Monkey", "Orange")

ani.shout()
m.eatBanana().
```

- Method overiding :- When a child class method overrides the parent class method of the same ~~name~~ name, parameters and return type.

#Example:-

PAGE NO.:
DATE: / /

- Superkey word:- Used to acess properties of on an object literal or class' [[Prototype]], or invoke a superclass' construction. The super.prop and super[exp] expressions are valid in any method defination in both classes and object literals.

- Static method:- Static methods are used to implement functions that belong to a class as a whole and not to any particular object.

#Example:-

```
class Animal {
    constructor (name) {
        this.name = name
    }

    walk() {
        alert ("Animal" + Animal.capitalize(this.name) + "is
            walking")
    }
```

```
static capitalize(name) {
    return name.charAt(0).toUpperCase() + name.substr
                (1, name.length)

    }
}
j = new Animal ("dog")
j.walk()
```

- Getters and setters :- Classes may include getters and setters to get and set the computed properties.

#Example:-

```
~~class Person~~.
class Animal {
    constructor (name) {
        this._name = name
    }
    fly () {
        ~~alert ()~~
        alert ("I am flying").
    }
}
```

```
}
get name (){
    return this._name
    }
}

let ani= new Animal ("Bruno")
  a.fly()
console.log(a.name)
```

Syntax :-

```
class Person {
    .....
    get (name){
        return this._name;
    }

    set name (new name){
        this._name= NewName;
    }
```

PAGE NO. :
DATE : / /

- Instance of operator :- The instance operator allows to check whether an object belongs to a certain class.

Syntax :-

<obj> instance of <class>

It returns true if object belongs to the class or any other class inheritance.

- IIFE - Immediately invoked function Expressions

IIFE is a JavaScript function that runs as soon as it is defined

```
(function (){
    ....
    ....
    .....
})();
```

=> IIFE syntax.

It is used to avoid polluting the global namespace, execute an async-await, etc

PAGE NO.:
DATE: / /

Example:-

```
let a = () => {

return new Promise ((resolve, reject) => {
    setTimeout (() => {
        resolve (456)
    }, 4000)
})

}
{ async () => {
    let b = await a ()
    console.log (b)

    let c = await a ()
    console.log (c)

    let d = await a()
    console.log (d)
}) ()
```

PAGE NO. :
DATE : / /

- Destructuring :- Destructuring assignment is used to unpack values from an array, or properties from objects, into distinct variables.

let [x, y] = [7, 29]

x ⟶ 7

y ⟶ 29

x will be assigned 7 and y, 20.

[10, x, ... rest] = [10, 80, 7, 11, 21, 88]

x will be 80 rest will be [7, 11, 21, 88]

Similarly, we can destructure objects on the left hand side of the assignment.

const obj = {a:1, b:2}

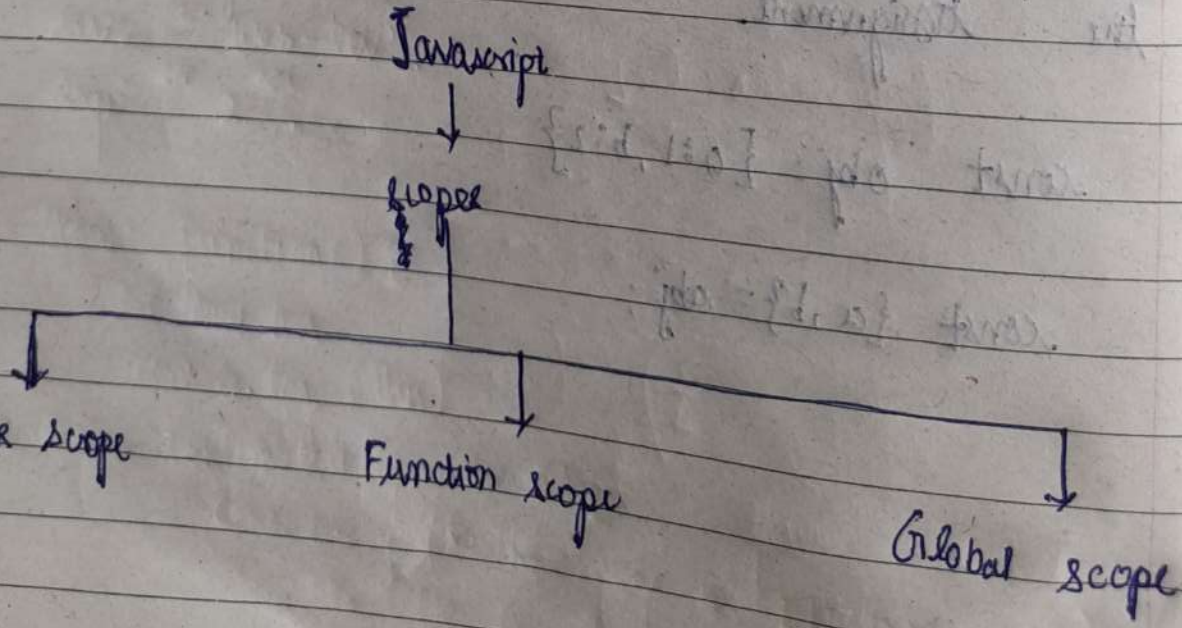
const {a, b} = obj;

PAGE NO.:
DATE: / /

- Spread Syntax:- Spread Syntax allows an iterable such as an array or string to be expanded in places where zero or more arguments are expected. In an object literal, the spread syntax enumerates the properties of an object and adds the key-value pairs to the object being created.

#Example:-

```
const arr = [1, 2, 11]
const obj = {...arr}; // {0:1, 1:7, 2:11}
console.log(obj);
```



PAGE NO. :
DATE : / /

Hoisting :- Hoisting refers to the process where the interpreter appears to move the declarations to the top of the code before execution.

Variables can thus be referred before they are declared in Javascript.

#Examples :-

```
hello ("Pratyush")

function hello (name) {
    ....
}
```

Closures :- (IMP) :- A closure is the combination of a function bundled together (enclosed) with reference to its surrounding state (the lexical environment). In other words, a closure gives you access to an outer function's scope from an inner function.

PAGE NO. :
DATE : / /

- Arrow Function:- It is an expression is a compact alternative to a traditional function expression.

Syntax :-

```
() ⟹ {
    ......
    ......
    ......
}
```

Example:-

```
const x = {
name: "Pratham",
dev: "Web-dev",
exp: 2,

show: function() {
    let that = this
    console.log(this)
    setTimeout(function() {
        console.log(`The name is ${that.name} \n
        The He is a ${dev}`)
```

PAGE NO.:
DATE: / /

```
    }, 2000)
  }
}
x.show()
```

↳ code after arrow function :-

```
show: function(){
  setTimeout(()=>{
    console.log('The name is ${this.name}\n He is a
                ${this.dev}')
  }, 2000)
}
}
x.show().
```

* - node package manager.

- REPL = Read - Evaluate - Print - Loop.