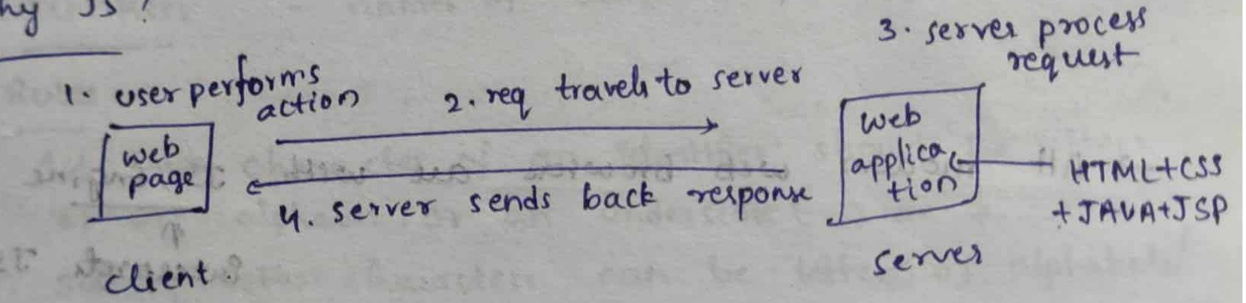
Java Script - full fledged client-side language

✓ to develop a dynamic web application, use event-based programming - create platform independent applications with security

Features

- easy to learn, debug and test
- event-based
- platform-independent
- intepreted language with all procedural programming capabilities.
- capable of executing user requests on the client side.
- helps in creating dynamic
  interactive
  scalable web applications

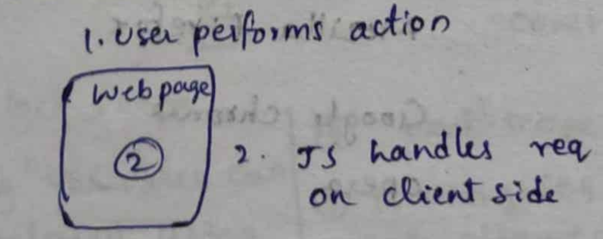
Why JS?



Server side languages - limitations

✓ Multiple req-resp cycles to handle multiple user req
✓ More n/w bw consumption
✓ Increased response time.

→ Using JS, this can be done on client-side without consulting server



advantages

1. less req-res cycles
2. less b/w ~~conm~~
3. In comp to Java: 35% ↓ in avg response time and pages being served 200ms faster)

JS - clientside scripting lang

ECMAScript - established a std for scripting language in 1997.

ES └ parent of JS, TS, JScript, ActionScript

ES6 - faster because of features:

- ✓ OOP
- ✓ new progⁿ constructs
- ✓ modules
- ✓ templates
- ✓ support for promises

HTML + CSS → static presentation of web page

HTML + CSS + JS = Dynamic web page
content presentation Action

→ All modern web browsers are with JS Engine,
↑ interprets JS code.

There is absolutely no need to incl. any file or import any langua pkg inside the browser for JS interpretation.

JS Engine	~~JS Engine~~ Browser
SpiderMonkey	NetScape Navigator Mozilla Firefox
V8	Google chrome opera
JavaScriptCore	Safari
Chakra and ChakraCore	Internet Explorer.

3 ways of writing JS - embedding within HTML or writing in ext. file

- ✓ Inline
- ✓ Internal
- ✓ External

	Internal Script	Vs External Scripting
load time	faster	slower
re-usable	no	yes
maintainability	difficult	easy

Identifiers - names of elements in the JS

Rules

1. 1st ~~letter~~ character of an identifier should be letters of an alphabet or an underscore (_) or $.
2. subseq ~~letter~~ characters can be letters of alphabets/ digits or _ or $.
3. case sensitive
4. keywords can not be used.

types of identifiers

let	const	var
① block scope ② looping variables can be declared using let keyword ③ redeclaring same variable with let throws an error	① block scope ② re-assigning not allowed ③ same as let	① identifiers declared to hold data that vary are called 'Variables' ② redeclaring same identifier won't throw an error ③ takes function scope i.e globally available to function.

Input: prompt()

Output:
console.log() → in console
document.write()

Datatypes
- primitive (if it contains an individual value)
- non-primitive (collection of multiple values)

primitive

① datatype number : can hold values of integer, long, float, double

constant of type number can be declared like

Ex:
```
const a=10;
const pi=3.14;
```

let result=0/0 (not a legal number) so it's val is NaN
let res = "Ten"+5

② String singe / double quotes , indexing

③ Boolean T/F ~~——~~ 0, "", NaN, undefined, null
100, "c", 1

④ Undefined - no value

⑤ null - no object

⑥ BigInt

⑦ Symbol()

Non-primitive - Object and Array

Object: key-value pairs, unordered

Array: heterogenous ordered collection

```
let A = new Array();
//OR
let B = [];
//OR
let B = [1, 2, "Sima"]
```

Operators

Types

- Arithmetic
- Assignment
- Relational or comparison
- logical ! && ||
- conditional

Statements and expressions

Statements are instructions in JavaScript that have to be executed by a web browser

statements

- Conditional
  - if-else
  - switch
- Non-conditional
  - comment
    - Single line //
    - multiline /* */
  - break
  - continue

Loops

Types

- for
- while
- do...while

JS Engine can execute JS in 2 different modes:

✓ Immediate mode

✓ Deferred mode

Functions : set of stat" that perform a specific task
reusable unit of code.

In JS, functions are first-class objects i.e, functions can be passed as arguments to another functions, it can be real value of another function or can be assigned as a value to a variable.

Types of functions

User-Defined
→ user - write own fun
→ can also be arrow functions

Built-in
- pre-defined functions like
- displaying dialog boxes
- passing a string argument
- timing related ~~problems~~ op"

defining a function:

function keyword is used.

arrow function & anonymous function

functions are first-class objects. This means, that you can assign a function as a value to a variable.

1) anonymous function

```
let fun1 = function() {
        console.log("Welcome to JS");
   fun1();
```

2) Arrow function

```
let fun1 = () => {
        console.log(" Welcome ");
};

fun1();
```

Anonymous function

Function without a name is called A.F

Ex:
```
let sayHello = function() {
        console.log("Welcome to JS")
};
sayHello();
```

Arrow function - to declare anonymous function

```
let sayHello = () => {
        console.log("Welcome to JS");
};
sayHello();
```
↑ sayHello is a variable which is a function

(or)

```
let sayHello() => { ~~console.log~~ "say Hello"
console.log(sayHello())
```

parameters - variables that defined in function definitio
arguments - values passed to function when it is invoked.

① JS → doesn't throw any error if no.of parameters don't match

Ex:
```
function mul(n1, n2){
    if(n2 == undefined)
        n2=1;
    return n1*n2;
}
console.log( mul(5,6)); //30
console.log( mul(5)); //5
```

② JS introduces an option to assign default values in fun

```
function multiply(n1, n2=1){
        // same as above
}
```

③ Rest parameter syntax allows to hold indefinite no. of arguments in the form of an array

```
function(a, ...args) {            | function ABC(a, b, ...c) {
    // return b                   |     return c
}                                 | }
console.log( f                    | console.log( ABC(1,2,3,4)) //
                                  |                          [3,4]
```

Rest parameter → should always be the last parameter

④ Destructuring - gives a syntax which makes it easy to unpack ~~from~~ values from arrays, or properties from objs into diff values

```
let l = [ 'A', 'B', 'C' ];                      | function show({name, value})
function show( [ arg1, arg2 ] ){                | {
        console.log(arg1);    // A              |
                   (arg2);    // B              | }
}                                               |
show ( l )                                      | ( {             } )
```

Nested functions in JS - perfectly normal

private to container function and cannot be invoked from outside the container function

Built-in functions

alert()
confirm()
prompt ()
isNan()
parseInt()
parseFloat()   parseFloat("10.34") // 10.34   parseFloat("10 yrs") // 10
eval() ←   eval("let n1=1; let n2=2; let res=n1+n2; console.log(res)");

8. setTimeout() → takes 2 params
① function to be executed
① no.of milliseconds after which given func should be executed

9. clearTimeout() → cancels timeout estb prev by ⑧.

10. setInterval(fun, time)
11. ClearInterval (fun, time)

Variable scope in fun — 3 types: Global, local, block

↑ accessibility of a variable

Global variables → variables defined outside fun and they are accessible anywhere in the program

local — inside functions, Can't be accessed outside the declared function block

block-scoped — variables inside function declared with 'let' and 'const'

Classes and Objects in JS

In 2015, ECMASCRIPT introduced the concept of classes to JS.

```
class A {
    // constructor , methods and all
}
```

static values can be accessed only using class name and not using 'this' keyword, outside the class

Inheritance

extends keyword
sub class inherits all methods (static & non-static) of the parent class.

enables reusability and extensibility of a given class.

super - to refer base class methods | constructors from a subclass.

Event Handling

when ~~[illegible]~~interaction happens, the event triggers. JS event handlers enable the browser to handle them.

Deferred mode (DM) → when execution of JS is delayed / deferred till some event occurs, the exe^n is called DM exe^n

Built-in events and handlers

Event	Event Handler
click	onclick
keypress	onkeypress
keyup	onkeyup
	onload
	onblur
	onchange

← associated with HTML elements
responsible to handle / listen to the event taking place on the respective element

• html

```
<script src="test.js"></script>
:
<p onclick="executeMe();"> Para says!!! </p>
:
```

test.js

```
function executeMe() {
    alert('A click event has been triggered by
        the user');
}
```

Exception Handling

To avoid termination of a prog in unfriendly manner when an exception is encountered - try...catch statu

try
catch
throw → to create custom exception
finally

Objects

A variable to hold data that represents the collection of propts is required.

object - consists of state and behavior.

- state ↑ represents properties that can be modeled as key-value pairs
- behavior ↑ rep observable effect of an operation performed on it and is modeled using funcs

JS Objects Type

- User defined
- In-built
  - Global
    - Primitive type — wrapper objs for their corresponding primitive data types — Number, String, Boolean
    - Miscellaneous — Date, Array, Math — utility objects provided by JS
  - Browser — HTML, DOM

Create objects 2 ways
1. Object literals
2. constructor

```
① objName = {
      key1: val1;
      key2: val2;
      :
      key-f1: function(param){
      },
      :
  }
```

To create multiple objects with same set of prop. and methods, function constructor can be used.
invoked using new keyword

Ex:

```
function Car( name, model, color, curSpeed, curGear )
{
        this.name=name;
        this.model=model;
        this.color =color;
        this.accelerate = function( speedCounter) {
              this.curSpeed  = curSpeed + speedCounter );
        }
        :
        :
}
```

can retrieve state / behavior of obj using
. or [.]] operator

Spread operator

combines 2 or more objects.

```
let objname1 = {
        // props
};
let objname2 = {
        // props
};
let objname3 = {
        ... object1Name,
        ... object2Name
};          // combined obj   - all props of obj1, obj2
```

to get a copy of existing object - use spread operator

```
let copyobj = { ...originalobj };
```

Accessing object properties using for.. in loop :

To work with all the keys of an object, there is a particular form of the loop: for.. in.

Ex:
```
let user = {
        name : "Rekha",
        age : 24
        isConfirmed : true
};
for ( let key in user) {
        console.log(key);
        console.log(user[key]);
}
```

Date
String
Math
RegEx
JSON

} are the built-in global objects

Date

methods

getDate()
getDay()
getFullYear()
getHours()
getMonth()
getMilliseconds()
getTime() → returns no. of ms since 1/1/1970 at 12:00 am

try setters methods

length

→ let s = "Hello"
  lengthofS = s.length
→ charAt(index)
→ match() - make use of regex to look for a specific string and returns all the strings that match

```
s = "Are you enjoying JS?"
console.log( s.match(/you/));
/* returns an array:
[       'you' ,
        index : 4 ,
        input : ' - - ' ,
        groups : undefined
]*/
```

→ replace(a1 , a2)
→ slice() returns part of string

```
s = "Hello"
p = s.slice( 2,4)    // ll
            ↑
         optional
p = s.slice(2)  // llo
```

→ substring() same as slice() but can accept a -ve parameter meaning slicing should start from the end.
→ substr()   second parameter - length of slicing string.
→ toLowerCase()

Regular Expressions

Object in JS that helps us perform inspection and processing of strings with a certain pattern

can be constructed - 2 ways

1. Regexp constructor
2. / pattern/ modifiers
   ↑
   optional.

pattern	Description
[abc]	to search in given string for any of chars present within bracket
[0-9]	for any of the digits
(a|b)	for either of the characters)
[^abc]	for any of char's which are not a,b,c
[^0-9]	" " digit which is not b/w 0-9
n+	to check if given string contains atleast one n'
n*	contains atleast zero/ more occurences of n.
n?	at least zero or one occurence of n
?=n	match any string i.e followed by n
n{x}	" given string contains X n's
n{x,}	atleast X n's
n{x,y}	x to Y n's

Ex:

email pattern @ .com

```
let emailP = new RegExp("(?=.@*)(?=.+.com)");
let phoneP = new RegExp("(?=[0-9]{10})");
let pwdP = new Regexp("(?=.*[0-9])·
                      (?=.*[a-zA-Z])(?=.*[@#$!"])
```

Math

object under category of global objects

```
Math.PI        // returns 3.14
Math.SQRT2  // returns 1.4142
Math.max ( 10,9,8,7,6,5, 11)      // returns 11
Math.min
Math.ceil( 20.4)          // 21
     floor( 20.4)         //20
Math.random( )         // returns no. between 0 and 1
                          Inclusive of 0 and exclusive of 1.

Math.sqrt(9)            //3
Math.round( 30.5)      // 31
```

JSON

- JS object Notation, lightweight data-interchange format

used for storing and sharing data between client and server over the network.

```
let data = {
    customers : [
        { "firstName" : "Bob", "LastName" : "Morry"},
        { "firstName" : "Albert", "LastName" : "Smith"}
        :
        :
    ]
};
```

data → JSON object
customers → array name

→ For JSON objects, it is mandatory to put key inside double quoutes and all values of type string inside " ".

→ For JS obj, key is not put in " " if string val is ' ' or " ".

parse()

to parse string → JSON & helps prog to process obj

```
ex: let strJSON = '{"firstName": "Sam", "lastName": "Fern"}
    let obj = JSON.parse( strJSON);
    // { firstName : 'Sam', lastName: 'Fern'}
```

stringify() returns JSON string corresponding to given object

Ex: let dataJSON = { firstName = "Sam" , lastName: "Fernandes"};

let obj = JSON.stringify(dataJSON);

Browser Object Model (BOM)

dynamic manipulation of HTML page on clientside itself is achieved with the help of built-in browser objects.

↑ allow JS to programmitically control the browsers

Browser — split into parts - for prog

↑ each part - object (built-in)

window - root object

BOM (collection of objects)

$$\begin{tikzcd}
 & & \text{window} \arrow[dll, dash] \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
\text{History} & \text{Navigator} & \text{location} & \text{document}
\end{tikzcd}$$

(HTML page that gets loaded)

① document object

considers web page as a tree (DOM)

accessing elements in HTML

```
document.getElementById('id');
document.getElementByTagName('tag');
document.getElementByClassName();
document.querySelectorAll()
```

↑ finds elements by CSS selectors and return NodeList (list of element objects)

to set new content

```
document.getElementById().innerHTML = "New Heading"
```

attribute

to set new values to given attributes

```
document.getElementById("div1").setAttribute('newkey': 'value');
```

style

gives access to the style attribute of HTML element and allows it manipulate CSS modifications dynamically.

```
document.getElementById("div1").style.color='red';
```

② window object

When it is not req to update HTML page but only certain props of browser window on which it is rendered.

To navigate to different URL & display new web page

close webpage

store some data related to web page

This obj gives us access to toolbars
status bars
menus
even HTML web page currently displayed

Props

→ innerHeight — holds height of window's content area

→ innerWidth

window.innerHeight

→ outerHeight → incl. toolbars and scrollbars

→ localStorage → prop that allows access to object that stores data without any expiration date

→ sessionStorage

Methods

open() → opens a new window

```
window.open("http://.....com");
```

close() → closes current window

③ history object of window props.

to target only one of window props.

Ex: If concern is about list of URLs that have been visited by the user and there is no need for any other info about the browser.

prop history.length;

length
returns no.of elements in history list

Methods

history.back() → loads prev URL from history list
forward() → next
go() → loads previous URL present at given no from the history list.

④ Navigator object

info about the client i.e browser on which webpage is rendered.

props
→ navigator.appName // browser's name
→ navigator . appVersion // returns pf (os) version of client (browser)
→ navigator. platform // return OS name
.userAgent // string equiv. to HTTP user-agent rqst header

⑤ location object

↑ to programmitically refresh the current page or navigate to a new page

→ contains info about current URL in the browser window. The info can be accessed / manipulated using. →

props and methods

href

contains the entire URL as a string

hostname

contains hostname part of the URL

[URL:

← authority → port

http://www.example.com:80/ path/to/myfile.html ? key1=val1 & key2=value2 # SomewhereintheDocument

↑ Scheme

↑ parameter

→ anchor

(protocol that browser uses to request the

→ domain name → domain indicates web server ie being requested.

port : technical "gate" used to access the resources on the web server.]

port

contains port no associated with the URL.

assign() ← loads new HTML documents

location.assign('http://www.facebook.com');

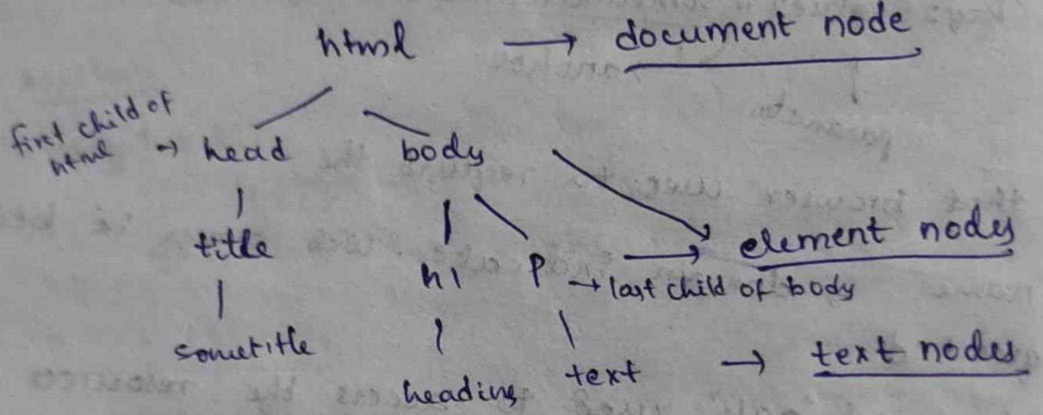
reload() ← reloads current HTML

DOM   Document Object Model

HTML page is considered as DOM tree by the browser, with every HTML element having a hiera

Using DOM, HTML elements can added / removed dynamically

Acc to W3C DOM standard, each HTML element can be referred to as a Node



parentNode.

```
document.body.parentNode        // returns <html>
```

childNodes

```
document.body.childNodes        // returns h1, p
document.getElementById("someid").firstChild;
document.getElementById("someid").lastChild;
                                  .nextSibling;
                                  .previousSibling;
```

createElement()

creates a new Element

```
let newElement = document.createElement('span');
```

createTextNode()

creates content at runtime. This node then can be appended to any node that can hold content.

```
let newTextElement = document.createTextNode('The
                       span is added just now!');
```

appendChild()

appends a newly created element to existing DOM tree

newElement.appendChild(newTextElement)

document.getElementById('div1').appendChild(newElement)

removeChild()

removes element from existing DOM tree

```
document.getElementById('div1').removeChild(document.
                    getElementById('para1'));
```

Event Handlers in JS using inline scripting with DOM API

2 ways to handle events in JS - internal
                               - external

```
document.getElementById('para1').onclick = function() {
            alert('Para one clicked');

    //OR
document.getElementById('para2').addEventListener(
            click, function(){
                    alert('Para two clicked'); } , false);
```

events in JS - objects 'event'

events are fired → obj is generated by browser

This object encapsulates all data related to that event

props

→ target - refers to HTML element that fired the even

event.target.nodeName

→ type - tells type of event that have taken place like click, load etc

methods

preventDefault()

prevents default action ass with HTML element and adds user-defined action.

Iterables

Array - ordered
allows multiple values to store in a single Variable.
heterogeneous datatype

Ex
```
let numArr = [1,2,3,4];
let empArr = ["Johnson" , 123, "Chicago"];
```

To create array - 2 way

① Array literal Notation
```
let myArr = [ 1,2,3]
```

② Array construction
```
let myArr = new Array(arrlen);
```

Spread operator

to spread out elements of an array to a new variable.
If used in function call, it expands iterable object i.e, array into list of arguments

```
let numArr = [10,5,20];
console.log( Math.max(...numArr));     //20
           (Math.max( numArr))  ;     // NaN
```

to concatenate arrays
```
let a1 = [1,2,3]
let a2 = [4,5,6]
let a3 = [0, ...a1 , 7, ...a2];  // [0 ,1, 2, 3, 7, 4, 5, 6]
```

for .. of statement - to iterate over an array

```
for (let color of colors) {
        console.log(color);
}
```

Property

Length

methods   let a1=[1,2,3]

push()	a1.push(4)	// [1,2,3,4]
pop()	a1.pop()	// [1,2,3]
shift()	a1.shift() ← removes first element from a and returns that element // 1	
unshift()	a1.unshift(4) adds element to beg and returns new length //a1=[4,2,3] returns 3	
splice()	change the content of an array by inserting, removing and replacing elements, returns array of removed elements	
slice()	to get substring a1.slice(start, end)	
concat()	a1.concat(a2)	

a1.indexOf()

find()   returns value of 1st ele in array that passes a condi specified in callback function

a1.find(callback function)
element => element>0

findIndex()

filter()   creates list with~~at~~ elements that passes the test provided as a function

forEach(callback (item, index, arr))

map()

join()   a = ['1','2','3']
a.join('-')   // 1-2-3
a.join()   // 1,2,3

reduce()

Asynchronous Programming

Use : To make an HTTP request call
To perform any ip/op operations
To deal with client - server commn

can be achieved through

callbacks
promises
Async/wait

Callback function - function ie passed as an argument to another function
makes sure that certain func doesn't execute until another function has already finished execution

Callback hell - Pyramid of Doom
(consists of more than 1 nested callbacks

Promise - holder for a result that will become available in the future.
provides a more structured way to write asyn calls
→ a returned obj to which you can attach callbacks, instead of passing callbacks to a func
(callback hell)
→ comes to rescue when there are chained asynchronous calls that are dependent on each other

```
new Promise( function (resolve, reject) {

    //

});
```

→ 3 states
pending - result of asyncall not known yet
resolved - " " returned with success
rejected error

then, catch, finally → Error Handling

Async/await

to implement asynchronous code with promises that resemble synchronous code. "async/await" - simple, easy, readable.

can use try, catch, finally → Error Handling

Async() → returns a promise, if not promise, JS wrap value in a resolved promise

Await() → makes JS wait until promise returns a result.
works inside Async()

Fetch API

JS plays an important role in communication with the server. This can be achieved by sending a rqst to the server and obtaining info sent by the server.

fetch() ↗ does this

$$\begin{tikzcd}
\text{PromiseReturned} = \text{fetch(urlofthesite,} & \text{[options])} \\
\text{mandatory arg} \arrow[u] & \text{methods, headers} \arrow[u] \\
& \text{default - GET} \arrow[u]
\end{tikzcd}$$

returns a promise that resolves to a response if fetch() is successful
error if unsuccessful

fetch() method

getting response from fetch() - 2-step process

1. The promise object returned by fetch() needs to be resolved to an object after the server sends a response. status between (200-299) → true.
HTTP fetch successful

2. get response body using additional methods

```
response.text()
         json()
         formData()
         blob()
```

Modular Programming

Modules → help in state and global namespace isolation and enable reusability and better maintainability.

export } keywords to e/i variables / methods / objs from
import } a module.