Java Script

Javascript is a programming language.

Javascript uses REPL → Read Evalute Print Loop (Using Console)
(ctrl + K) → to clear

- (ctrl + /) → for comment
- console.clear(); → delete Every thing from web

Variable

it is a name of storage location (The Things whoes values can be vary depending upon situation)

age = 23
↳ variable

prompt → To take User Input
ParsInt → Convert it into NUM

Data Types

- These 5 are the basic primitive datatype in Js

1) Number
2) Boolean
3) string
4) Undefined
5) NULL

} Immutable

- To cheak type of data we use {typeof a}
  function ↵
- These are immutable in nature

(Tab to automatic complete Name)

1) Number →

- Positive, Negative, Integer, floating Number
- there will be a limit for storing a number after hitting to that limit Number will be automatically round off.

Operations = Addition, Substraction, Multiplication, Division, modulo, Exponentiation

Modulo → Remainder Calculation (%)

Exponentiation → Power calculation (**)

NaN

- This represents Not a Number
- 0/0 ⟶ this does not Valid so answer will be NaN
  It's data type is Number

Operator Precendence

This the order of solving an expression

(1) () ⟶ Brackets

(2) ** ⟶ Power

(3) *, /, % ⟶ multi, division, modulo

(4) +, - ⟶ add, sub

Sometimes we all have the this level symbols at that time solve from 'left to Right'

let

Syntax used for declaring Variable ⟶ mostly Used in place of Var

```
let a = 5 ;
let a ;
```
} proper way to declare a variable.

Const

- Parmentaly fix the Value of a Variable
- Value of these constant can't be changed.

```
Const pi = 3.14 ;
```

Var

old way to write variables

```
Var age = 23;
```

Javascript

Assignment Operator

+ =

- =

Unary Operators

In this only one operands are required

age ++

age --

- This will be incread one by one
- one value will be added till condition is true.

1) Pre increment ⟷ (change, then Use)

let New age = ++age;

2) Post increment ⟷ (Use, then change)

let Newage = age++;

Identifier Rules

1) Name can have letter, digits, underscores, doller signs
(No space allowed)

2) Name Must begin with a letter.

3) Name can also begin with ($) & (-)

4) Names are case sensitive

5) Reserved key words can't be used as names.

Ways of writing identifiers
1) camelcase
2) snake_case
3) PascalCase

e.g.

$price
_price
price$
} Valid

-price
price-
1Price-
} Invalid

2) Boolean

true / false

Yes / No

DataType of Variable can be change at every stage

Typescript ⟶ • strict Version of Javascript
• Data types can not be changed in this
• This is degsioned by Microsoft.

3) String

- text or sequence of characters
- given by (" ") or (' ')
- Indexing start from 0 (0 based indexing)
- name of string.length ⟶ This have data type as a Number
- String also have concatination

4) Undefined

A variable that does not have any value or
No value is assigned to that variable that is undefined.

let a;
↳ This is undefined

typeof a ⟶ Undefined

when Javascript does not know the Value at that time it generate Undefined.

5) Null

Represents the intentional absence of any object Value.

```
let a = null;
```

Null also a keyword

(Negative Indexing is not allowed in Js)

Console.log ()

To write a message on the console

```
Console.log ("Hello")
```

Linking Js file

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

If it is in same folder

If this is in another folder then add the Relative Path

Template literals → In a single string operation we can avoid all Unnessary operations.

Used to add embedded expression in a sting

```
let a = 5 ;
let b = 10 ;

Console.log (`Your pay ${a+b} rupees`);
```

this is back ticks

In these 2 New symbol Used they are ar → (` `, ${})

Here the action OR mathematical Intuation are Performed.

Operators

1) Arithmatic (+, -, *, /, %, **)

2) Unary (++, --)

3) Assignment (=, +=, -=, *=, /=, %=)

4) Comparision

5) Logical (&& || !)

4) Comparision Operators

Compares 2 Values

{ >, >=, <, <=, ==, != }

Comparision operator gives us answer in true or false

- == (compare value, not the type)
- === (compare value & type)

1 == '1' ⇒ true
1 === '1' ⇒ false } main difference

Every character Has a unique code associated with them Bcz of these we can compare them.

'a' > 'b' ⟶ false
'a' > 'A' ⟶ true

(Cheak the Unicodes at Jarascript Unicode website)

trend

a < b < c < d <, ---, < z
A < B < C < D < ---< Z

} Normally these trend is followed.

JavaScript

Conditional Statement

1) If - else
2) Nested If - else
3) Switch

1) If Else

```
if (age >= 18) {
        console.log("you can vote");
    }
```

only Use of IF

If can be wrote multiple times

```
if (age >= 18) {
        console.log ("you can drive");
    }
else if (age < 18) {
        console.log ("you can not drive");
    }
```

Use of else if

else if can be wrote multiple times.

without in If condition we can not directly write else if

```
if ((color === 'red') {
        consol.log ("stop");
    } else if (color === "Green") {
        consol.log("Go");

else {
    consol.log ("light is broken")
}
```

Use of else

All conditional statement Becomes false then else executes

there will be only one else condition present

2) Nested if-else

writing if-else inside of if else

```
if (marks >= 33) {
        if (marks >= 80) {
            Console.log ("Grade A");
            }
        else {
            console.log ("Greade B");
            }
    }
else {
    console.log ("Better luck next time");
  }
```

* Logical Operators *

Used to combine expressions

1) && (logical And) ⟶ Both must be true/all must be true
2) || (logical OR) ⟶ any one must be trust
3) ! (logical not) ⟶ Reverse the Values.

truthy & falsy

Every thing in JS is in true or false

In Boolean contex they may be true or may be false

falsy values ⟶ {false, 0, -0, empty string, null, undefined, NAN}

truthy Values ⟶ Everything Else.

Javascript

3) Switch

Used when we have some fixed values that we need to compare.

```
let color = "red";

switch (color) {
    case "red":
1)  consol.log ("stop");
    break;

    case "green":
2)  consol.log("Go");
    break;

3)  default:
        consol.log("Broken light");
}
```

at a time only one switch can be turned on

another way to write If, Else

Alert

displayes an alert message on the Page

```
alert("Something is wrong");
```
→ syntax

error

- displayes error message in the console
- Always it is shown in red color
- comes with console

```
Console.error("this is an error message");
```
→ syntax

Warning (warn)

- displayes warning message on console
- Always shown in yellow color
- comes with console.

```
console.warn("This is a warning msg");
```
→ syntax

Prompt

Used to ask user input

```
prompt("please entre your roll No");
```
→ syntax

---

write a program to cheak if 2 numbers have the same last digit

e.g. 32 and 47852 2=2

modulo Technique. ← if (num1 % 10) == (num2 % 10) { then they are same }

String method

methods are the actions that can be performed on objects.

```
StringName.method()
```
→ format

↳ This shows us the method

Javascript

String methods

strings are immutable in Javascript

1) trim ()

trims white spacey from both the end (starting and Ending)

string.trim(); —— syntax

- in these new trimmed string is shrown to us
- String is immutabl.
- string = string.trim() —— to store permentaly

2) toUpperCase ()

All the letters of the string will becomes/converted into UpperCase letter.

String.toUpperCase(); —— syntax

3) toLowerCase ()

All the letters of the string will be converted into lowercase letter

string.toLowerCase(); —— syntax

* String methods which Requires Argument *

Argument is the Value which we pass in method

stringName.method(Argu)

String with Argument

4) indexOf (Argument)

Returns the first index of occurance of some value in string

if value is not found then "-1" is writen

```
Str = "IloveCoading";

str.indexOf (Argument)
```

Thing you want to find ↑

↳ syntax

```
Str.indexOf("love");
```

U I l o v e C o a d i n g
0 1 2 3 4 5 6 7 8 9 10 11 12

→ ② → This index is given as output

```
Str.indexOf("J");
```

→ (-1) → Because J is not Present in this

- method chaning → • Use one or more method after another.
  - order of execution is from left to Right

```
String.trim().toUpperCase();
```

Two methods are used
These will execute from left to Right

5) slice

a subpart of the original string as a new string

~~String(5)~~

```
string.slice(5)
```

→ start from 5 and goes till last

```
String(
```

```
string.slice(1, 4)
```

→ start from 0 & end at 3

4 is non inclusive it will not be included.

```
String.slice( start, end );
```

— syntax

~~String slice~~

string.slice(-2); → Total length − 2
= final answer

slice(final Answer) → starting from these Variable

let msg = "omdarkunde";

msg.slice(-2); → ((10-2)
= 8 → from this index
slice(8) → de → final Answer

6) Replace

Searches a value in the string & return new value in place of that

~~String~~(

syntax — String.replace("Thing need to change", "with what you want to change");

7) Repeat

Returns a string with the number of copies of a string

let String.repeat(3) —— syntax

↳ this much time it will get repeated.

Array — Data structure

linear collection of things — any type of data type can be stored in this

```
let array_name = [ element1 , element2, element3 ]
```
— syntax

- arrayes have data type as object

```
typeof array-name ⟶ "object"
```

```
let array_name = [ ] ⟶ empty array
```

- arrayes are mutable

```
let fruit = ["apple", "banana", "chiko"];
fruit [0] = "Graphes";
```

} In original array there will change Happens.

```
fruit [10] = "watermelon";
```
⟶ This will also work and it will store the value at 10th index
3 to 9 index will remain empty

Javascript

Array method

1) Push = add to end
2) Pop = delete from end & return it } END
3) Unshift = add to start
4) shift = delete from start & return it } start

array_name.method( ); — syntax

5) indexOf = returns index of asked Value
   If that is not present in array then (-1) will be shown.
   it is case sensitive

6) includes = search for a value in array
   if that present then → true
   otherwise → false

7) Concat = merges 2 array

array1.concat(array2); — syntax

- this will be placed first
- these will be placed after that

- original array will remain as it is
- New array must be created to store concatination

8) reverse = reverse the array

Javascript

9) slice = copies a portion of an array

array_name.slice(start, end); — syntax

(1)
(1,5) } same as string
(-2)

No change wil
Happens in
original arr

difference

10) splice = remove / replace / add element in place
changes are happening in original array

array_name.splice(start, deleteCount, item0 --- itemN);

start → Starting from
deleteCount → That much you want to delete
item0 --- itemN → items that much you want to add

Color = ["red", "yellow", "blue", "orange"];

1) ~~Color~~[Color.splice[0,1]; → from These index element
1 element will be removed

→ [yellow, blue, orange]

2) color.splice[0, 1, "green", "voilet"]

→ These will be placed at that given starting index

→ [green, voilet, blue, orange]

11) Sort = sort an array (ascending or decending)
work good on string
But not that much on Number (only use for string elements)

In number scenerio first it converts them into string and then sorting happens

Javascript

Array References

[1] === [1] or [1] == [1] ⟶ false

- These happens because the References of array
- values of array are stored in different location the array name Have the knoledge of the address only
- It does not have the knowledge of that actual element so it can't compare Because of these it's generates false.

let arr = ['a','b','c'];

let arr_copy = arr;

⟶

arr == arr_copy;

arr === arr_copy;

} True

These Happens Bcz the Both arrays Has element on same address

Constant Array

const Array = [_,_,_]

In these the actual original elements can not be deleted

↳ new array from existing can't be created

all Other action can be performed on these.

Nested Array

multidimensional Array
Array inside Array
} other names

```
let num = [[1,2], [3,4], [5,6]];
```

	0,0	0,1	
	1	2	
(1,0)	3	4	1,1
2,0	5	6	2,1

Javascript

Loops

1) for loop

Used to iterate a piece of code

```
for (let initilization; condition; updatation){
        // do something
}
```

} Syntax

```
for (let i=1 ; i<=5 ; i++) {
        consol.log(i) ;
}
```

} example

- Dry Run → solving the code on paper and pen

Infinite loop

①
```
for (let i=1 ; i>=0 ; i++) {
}
```

②
```
for (let i=1 ; i<=5 ; i--){
}
```

example of Infinite loop

③
```
for (let i=1 ;  ; i++) {
}
```

Nested for loop

```
for (let i=1 ; i<=3; i++){
    for (let j=1 ; j<=3 ; j++){
            Console.log(j);
    }
}
```

1 — 2 loop inside a loop

any No of nested loops can be written

While loop

```
while (condition){
        // do something
}
```

} syntax

```
let i=1 ;
while (i<=5) {
        console.log (i);
        i++;
    }
```

} example

Javascript

break keyword

get us out of a loop execution
most of the time written with wile loop

```
let i=1 ;
while(i<=5){
    if(i==3){
        break
    }
    console.log(i);
    i++;
}
```

} if 3 comes then stop

loops with Arrays

```
for(let i=0; i< array.length ; i++) {
    console.log( i , fruit[i]);
}
```

} To print All the elements of the Array

```
for( let i = array.length -1 ; i>=0 ; i--){
    console.log( i , array[i] );
}
```

} Reverse print the array

Nested loops with Nested Array

```
for ( let i=0 ; i< array.length ; i++){
        Console.log(`List #${i}`);
    for( let j=0 ; j< array[i].length ; i++){
            console.log (array[i][j]);
    }
```

→ Nested array length

*for of loop *

```
for (element of collection) {
        //do something
}
```

} Also used in nested syntax condition

```
for ( fruit of fruits){
        console.log(fruit);
}
```

↑ New variable

↑ individually print

day 18

for in → return index Num

```
for (elemen in collection) {
        // do
}
```

Javascript

Object

- Used to stored keyed collection & complex entities.

  (key, value) Pair

- objects are collection of properties
- No order present in this (It is unordered)

```
let student = {
        name : "om",
        year : "4"
    };
```

key ← name : "om", → value } example

```
const student = {
        name : "om2",
        year : "4",
        surname : "Darkunde"
    }
```

} In this case can not change the address (can not be reassign)

we can change name, year and surname.

- In objects we can store array

```
student["name"];
student.name ;
```

} ways to access the values

- Js automatically converts objects keys to string
- Even if we made the number as a key the number will be converted to string

Update in object

Student. city = " " ;

Student["city"] = " " ;

object. key_Name = "New Value" ;

Object[key_Name] = "New Value" ;

Add in object

Student. gender = "Male" ;

Student['gender'] = "Male";

↓ This is not present then it will be automatically get created

↳ This value is then stored in that

Object. Key_Name = "New Value";

Delete from object

Delete object. key_name ; —— syntax

Array of objects

```
const classinfo = [
    {
    },          } 1st obj
    {
    },          } 2nd obj
    {
    }           } 3rd obj
]
```

classinfo[1]. name

↓ 1st info ↳ fetch Name

Javascript

Math objects

Math.PI = 3.14

Math.E = 2.718

} Properties

1) Math.abs(Number) = convert this into positive

2) Math.power(Number, Power)

3) Math.floor(Number) = remove the decimal

math.floor(-5.5) ⟶ −6 — small No is given

4) Math.ceil(Number) = remove the decimal & forward to higer No

Math.ceil(5.0001) ⟶ 6

5) Math.random() = give random value from 0 to 1

1 is not given any time

} Methods

Generate Random Integers

```
let num = Math.random();  ——— step 1
```

```
num = num x 10  ——— step 2
```

↳ upto what range we want to generate random Num

```
num = Math.floor(num);  ——— step 3
```

Till this Number will generate in range of 0 to 9

to convert the range upto 10 we need to add 1

```
num = num + 1;  ——— step 4
```

ct

```
random = Math.floor(Math.random() * 10) + 1;
```

↳ upto 10 random No will be shown

Javascript

function

```
function Defination ( ) {
        // do something
}
```
} syntax

```
functionName ();  ⟶ calling of function
```

```
function helo (){
        console.log ("welcome to world");
    }
hello ();
```
} example

Argument

Valuey we pass in the function

```
function function_name (arg1, arg2) {
        //do something
}
```
} syntax

order is important in Argument

Return

this keyword is used to return some value from the function

2)

```
function funcName (argu1, argu2) {
        //do something
        return value;
}
```

} syntax

- Return must be at end
- After Return the execution of the function stopes.

Scope

Determines the accessibility of Variables, objects and functions from different parts of code.

1) function → only inside function
2) Block
3) Lexical
4) Global (Normally described any where)

Global Vs function → function

1) function scope →
   - we can Use this only within the function
   - Variables defined inside function comes under function scope
   - Global Vs function = function scope.

2) Block Scope

- Variables declared inside a { } block cannot be accessed from outside the block
- After 2015 these scope is introduced
- Block scope is only applicable on let & const

```
for (let i=1 ; i<=5 ; i++){
        consol.log(i);
}
```

Block scope variable

3) Lexical Scope (Recursive function)

function call itself inside function

- Variable defined in outer function can be accessible inside another function defined after the variable decleardation.
- outer can not access inner function variable.
- Inner function can not directly accessed

Hoisting

function Expressions

store function inside a variable

```
const Variable = function (arg--) {
                    //do something
                 }
```

} syntax

variable ⟶ function syntax will be show

variable (); ⟶ function will work

Higher Order function

function that does one or both

- takes one or more functions as argument
- returns a function

```
function multipleGreet (func, n) {
        for (let i=1 ; i<=n ; i++){
                func ();
        }
}
```

} Higher order function
Unlimited time greeting

```
let greet = function () {
        Console.log ("Hello");
}
```

} fun expression

```
multipleGreet ( greet ,2) ;
```

⟶ calling

Javascript

Methods

Action that can be performed on an object.

```
const calculator = {
        add : function (a,b) {
                return a+b;
        };
key ← sub : function (a,b) {   → value
                return a-b;
        };
}
```

calculator.add ⟶ it will show us the syntax

calculator.add (2,3) ⟶ this will call the function

array, string } This have method
Because internally they are objects.

another way for method ⟶

```
Const calculator = {
add (a,b) {                  → without key value & function
        return a+b;
    },
sub (a,b) {
        return a-b;
    }
};
```

This keyword

- This keyword refers to an object that is executing the current piece of code.
- to asces variable of object "This" keyword is used.
- by default value of this is window object
  window object is the Higest level of object on ~~on~~ a window.
  All code comes under this.

Try & catch

Try → Try statement allows you to define a block of code to be tested for errors while it is being executed.

Catch → allows us to define a block of code to be executed, if an error occurs in the try block.

```
try {
    console.log (a);
}
catch {
    console.log ("does not present");
}
```

example

only try writing is not a good practice.

Javascript

Arrow function (lambda function) → Anonymous function

const func = (arg1, arg2) => {function defination}; → syntax

Used as callback fun^n

- Arrow function is a small way to write a function
- Arrow function does not have Names

```
const sum = (a,b) => {
                console.log (a+b);
    };
```
} example.

- for single Argument we can skip the () parenthesis
- () is compulsary if there is no argument and there are multiple argu

Implicit return

Implicit → automatic → पहिलेच माहीत असते
explicit →

```
const mul = (a,b) => {
            a*b
```

→ No return
No consol.log
} this will automatically Understand

(1)

→ In place of {} we are going to use { }

→ without this it can also be executed

```
const mul =(a,b) => a * b ;
```

Set Timeout

it will wait for the timeout and only one time execute after the timeout time

Inbuit function

setTimeout (function , timeout)

- function → callback
- timeout → upto what time we make delay
  - it is in the form of milisecond

```
setTimeout( () => {
        Console.log ("om darkunde");
}, 4000);
```

example

4000 → 2nd parameter

Set Interval

after some time of interval this will contineously execute the program

setInterval (function, timeout)

timeout → milisecond

- multiple time execute the program in given time of intervals
- every interval has an unique ID
- To stop this ⟶ clearInterval (Interval ID)

```
let id = setInterval( () => {
        Console.log ("Hello world");
}, 3000);
```

id → Id of the interval

Tutorial

(21)
(18)

Javascript

18 - 23 solve the questions :)

This with Arrow functions

- This uses lexical scope

explore more

Array methods

1) forEach
2) map
3) filter
4) some
5) every
6) reduce

} Higher order function
Uses callback method.

1) forEach

array.forEach(some function definition or name);

```
let Print = function (ele){
            console.log (ele);
};

array.forEach( Print );
```

↳ function

example

In place of these for of loop is used.

2) Map

- New array of same size is created
- a function is applied on all values of an array

let newArr = arr.map(some function or name of function)

↳ same size of previous

- If function does not perform any op action then undefined values will be stored in this new array

example.

```
let num = [1, 2, 3, 4];

let double = num.map (function (ele) {
                return ele*2;
});
```

3) filter

- New array is generated and result is stored in that
- If callback gives true then only the result is stored in new array

```
let newArray = array.filter ( some function) — syntax

let num = [ 2,4,1,6,9]

let even = num. filter ( (num) => (num %. 2 ==0) );
```

↳ If these Becomes true then only number is stored in new array even.

4) Every

- Returns true if every element of array gives true for some function Else return false.

```
array. every (function) —— syntax

[1,2,3,4].every( (ele) => (ele %2 ==0) );
```

ANS → false

- only two values as output → true/false
- same as logical and operator

5) Some

- Returns true if some element of array give true for some function.

  Else Return false
- same as logical OR operator

array.some(function) —— syntax

[1,2,3,4].some((ele) => (ele%2==0)); —— ANS: true

6) Reduce (IMP)

- Reducej the array to a single value

array.reduce(reducer function with 2 variablej for(accumalator, element)); ← callback

accumalator → given by us

element → Normal element

[1,2,3]

acc = 1

accumalator, ele ← computation Happens

this become new acc

Return a valu.

accumalator, ele → return

again same

accumalator, element

[1,2,3,4].reduce((res,ele) => (res+ele)); ⟶ ANS: 10

(0,1) = 1

(1,2) = 3

(3,3) = 6

(6,4) = (10) ⟶ final output.

```
let max . array.reduce ( (max, ele) => {
        if (max < ele) {
            return ele;
        }
        else {
            return max;
        }
});
```

find maximum element through reduce

Default Parameter

Giving a default value to the argument

```
function fun (a, b = 2) {        // b = 2 → default
        // do something
}
```

if no parametric value is passed in for b then b automatically will take 2 as the value

Spread

- Expand an iterable into multiple Values

iterable → on which we can apply loop

```
function fun_name ( ... array) {        // array → any iterable.
        // do something                 // ... → spread
}
```

OMDAR

O M D A → ANJ

multiple values are generated

Math.min (... array_name) Math.max(... array_name) → find smallest & biggest No

we can use these with array Literals.

```
let new_array = [... arr_old];
```

↳ This will be as it is copied in new array

```
let new_char = [... "Hello"];
```

Spread in object

```
let data = {
        email : "omdarkunde@gmail.com",
        password : "om"
    }
```

```
let data_copy = {... data, id: 1}
```

- `... data` → Spread the previous object
- `id: 1` → New key value pair is added in this.
- `{}` → Create symbol changes.

Converting array into object with the help of ~~array~~ spread

```
let arr = [1, 2, 3]

let obj = { ... arr }
```

↳ This index will be treated as key

Javascript

Rest

*args, **kwargs)

Allow a function to take an indefinite number of arguments and bundle them in an array.

```
function sum(... args) {
        return args.reduce((add, ele) => ele + add);
}
```

Destructuring

storing values of array into multiple Variables.

```
let names = ["tony", "bruce", "steve"];

let [winner, Runnerup] = names;
```

→ values of names will be indexwise goes inside the variables

```
let [winner, second, ...other] = names
```

↳ spread concept can also be combined with this.

Destructuring for object

```
const student = {
                nam = "Anil",
                Roll_No = "09",
                Age = 50
        }
```

```
let { Roll_No, Age } = student ;
```

These will be searched in the object and then stored in these given variable

- If these not present then undefined will stored.
- New could be added.

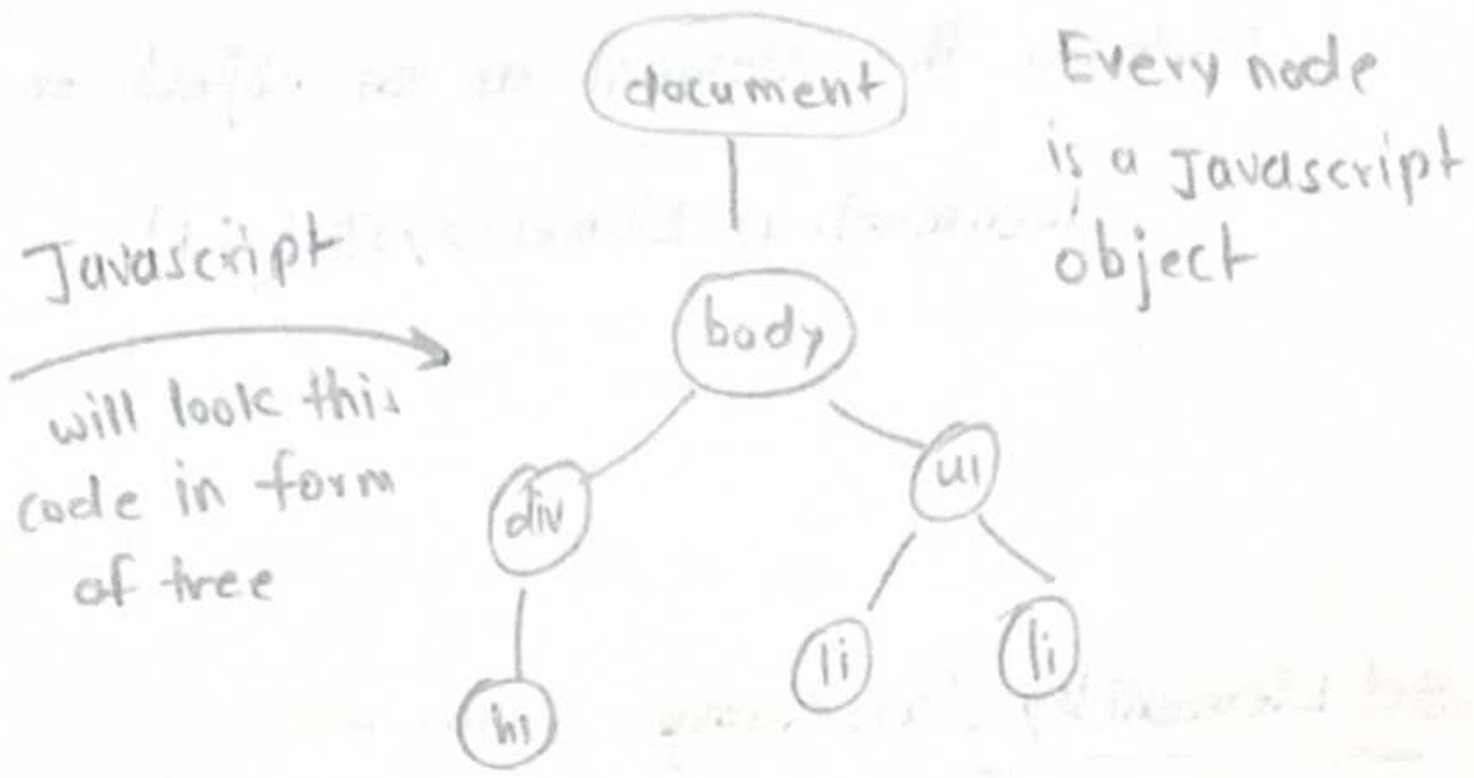
```
let { Roll, Age, location = "shriramput" } ;
```

Javascript

DOM (Document object model)

- DOM represents a document with a logical tree.
- allows us to manipulate webpage content (Html elements)

```
<body>
    <div>
        <H1> TODO </H1>
    </div>
    <ul>
        <li> Eat </li>
        <li> sleep </li>
    </ul>
</body>
```



- Javascript see's the Html & css code as in tree structure
- Every node of these Html & css tree is a object of Javascript
- document named a by default object is being created in a front end
- if we made changes in these document then that changes will be reflected in the Html & css page.

Consol.dir(document); ⟶ This will show us the document as a actual object

- Chnges in document will be shown on the webpages

By using Javascript in manipulation of Html & Css it is a two step process

1) select
2) change } 2 steps

get Element By ID

ID

Returns the element as an object or null

document. getElementByID (id) —— Syntax

get ElementBy class Name

class Name

- Returns the Elements as an HTML collection or empty collection

push, Pop operation can not be perfored in collection

document. getElementByclass Name (class); —— syntax

- If wrongclass is given then a Blank collection displayed tow

HTMLCollection [ ]

get ElementByTag Name

TagName

Returns the elements as an HTML collection or empty collection

document. getElementByTagName ("Tag Name"); —— syntax

Javascript (28)

Query selectors

Allows us to use any css selector

most of time we use this

```
document.querySelector('p');  ⟶ element
document.querySelector('#myID');  ⟶ ID
document.querySelector('.myclass');  ⟶ class
```

- only 1 element is select which is the first found

```
document.querySelector("div a");
```

from div
select anchor tag ? 1st one

querySelectorAll ⟶ This find find all the element and work same as query selector

Manipulating Attributes

id, class, Style, image these are some types of attributes present

- obj.getAttribute(attr) ⟶ used to find attribute
- obj.setAttribute(attr, Value) ⟶ used to change the value of Attribute

} getters & setters

Manipulate Style

Style property

object.style → syntax

We can access all the inline CSS only

element.style.color = "Red"; → example ↑ Here could be any content

↳ let element = document.queryselector('img')

- This only gives inline css

ClassList

- a object could Have multiple classes
- To cheak all classes of an object we use this method.

more change could be added with the help of these

1) classList.add() → add new class
2) classlist.remove() → remove class
3) classlist.contains() → cheak if class exists
4) classlist.toggle() → toggle bet^n add & remove

↳ just like switch

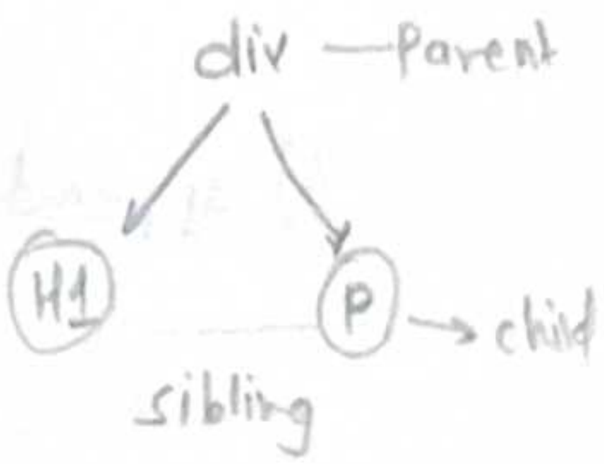
If it's current state is Yes then it become No and this viceversa Happens.

Javascript (24)

Navigation

After a certain element what is comming is shown to us

- parentElement (There will be only 1 parent every time)



object.parentElement; —— syntax

- children (There could be N numbers of children)

object.children; —— syntax

object.childElementCount; —— tells us the total No of child that tag have

- previowElementSibling / nextElementSibling

object.children[0],[1] — through indexing we can also cheak children

object.children[1].previowElementSibling;
object.children[1].nextElementSibling; } syntax

Adding Element

1) document.createElement('p'); → firstly we need to create these and then below 4 step — syntax

These is type of single element which will be added only once copies will not form.

1) appendchild (element new); → Befor these always create the element

- Append always add the thing in the end

box.appendchild(btn);

box → The thing in which we are adding

btn → New thing which will be added

2) append (New element) or string or text } more useful

newp.append("this is a good Book");

3) prepend (new element) add the element in the starting

box.prepend(newp)

4) insertAdjacent (where, element)

A) beforebegin → Before the target element

B) afterbegin → inside the targetelement, before it's first child

C) beforeend → inside the target element, after it's last child

D) afterend → After the targeted element itself

} value we can put in where.

Javascript

Removing Elements

removeChild (element) → Just like append child
Removes the child

```
body = element.quarySelector('body'); — step 1
body.removeChild('btn'); — step 2
```

} In 1st step we are select the parent and in 2nd step we are removing from it's parent

remove (element) → Just like append

```
btn.remove( ); —— directly removes without any parent includation
```

DOM Events (mouse Event)

Events are signals that something has occurred (user input/action)

```
<body>
  <button onclick="console.log('button waclicked')" > click me </button>
</body>
```

↳ when we click then specific action happens

} This is inline method

inline method is not that much used
it make code Bulky & Hard to understand.

onclick ⟶ when an element is clicked

event properties name will start from "0"

onmouseenter ⟶ when mouse entre an element.

↳ there are more like this

↘ these only shows only one value

Event listener

element.addEventListener(event, callback) — syntax

action ↑ ↑ these function will activate.

- waits to happen a event on page and then react on that event.

mdn_docs ⟶ Go To this website so you can see the all types of event.

- Event listnev could be added on eny kind of element like div, P

This in Event Listnevs

When this is used in a callback of event handler of something it refers to that something.

the element which has trigge.

```
function changecolor() {
    consol.dir(this.innerText);
    this.style.backgroundcolor = "red";
}
btn.addEventlistnev('click', changcolor)
```

this will works as btn name

Javascript

event argument →

```
btn.addEventListner ("click", function (event) {
                                    ↳ this keyword is by default present in this.
    consol.log ("button clicked");
    consol.log (event); ⟶ Pointer Event
                          ↓
                          This shows different actions of these.
});
```

keybord events

- keydown
- keyup
- keypress.

There are many more

```
input.addEventListner ("keydown", function () {
                        ↳ Different operations could also performed
    consol.log ("key was pressed");
})
```

form ۲

form Events

```
let form = document.querySelector('form');

form.addEventListner("submit", function() {
                        Console.log("form Submitted");
                    });
```

"submit" → on submiting a specific action will be get performed.

alert

event.preventDefault(); → This will stop it by performing the default action,
Just like defaut action is to open a new webpage then it will be stopped.

Extracting form Data IMP

```
form.addEventListner("submit", function(event) {
    event.preventDefault();
    let inp = document.querySelector("input");
    Consol.dir(inp);
    consol.log(inp.value);
});
```

"submit" → when submit is click then we can store the value

document.querySelector("input"); → finding the input tag

inp.value → displayes the input value in consol.

innerText will not store the give input Value

.Value will store that always

Javascript

```
inp = document.querySelector("input");
```

```
inp = form.elements[0]
```
→ above code can be replaced by these

↳ "this" keyword could Be used if it is in callback function.

Change Event

- It occurs when the value of an element has been changed
- only works on input, textarea, select

only track the bigchanges

Two Box present we entre value in one and then click any where then These could be a big change so it will track that.

```
object.addEventListner("change", action);
```
— syntax
(action or function)

input Event

- The input event fires when the value of an input, select or textarea element has been changed.

Tracks the smallest of smallest change and triggers.

Non character keys does not trigger input Event.

Event Bubbling

When many items are nested together and we apply event listner on them

if we perform action on the child then all the event listner of child as well as parent will get trigger

So this phenomenon is called as event bubling

to stop this we will use a method ⟶ stopPropogation

event.stopPropagation() —— syntax

~~for (li of~~ li.addEventListner("click", function(event){

event.stopPropagation();

console.log("li was click");

});

Because of these li's parent elements event listner will not get executed.

Javascript

call stack

call → calling of a function

stack → datastructure which follows LIFO

Breakpoint

- To understand How a line of code is getting executed so we add a break point in that line for Understanding.
- Mostly used for debugging

Javascript is single Threaded language

↳ only one thing will be executed at one time

- wating like work or parallel Happening works are done by browser
- browsers are most of the time written in C++ or Jave Because of these they are multi threaded.

Call Back Hell

- Many nested function are created that is known as call Back Hell
- operation is performed but that is not a effective way

```
function changeColor (color, delay, nextChange) {
    setTimeout( () => {
        h1.style.color = color;
        nextchange();
    } delay );

changeColor( "red", 1000, () => {
    changeColor ("orange", 1000, () => {
        changeColor("white",1000)
    })
});
```

Next change

Next change

These 2 are function which are given as parameter to that changecolor function

- To avoid callback Hell we use the concept of promisses, await async
- These type of nesting is very hard to understard Because of there ~~mes~~ we must avoid it.

Promises

- The promis object represents the eventual completion (or failure) of an asynchronous operation and it's resulting value.
  - It is just real life promise it could be completed or it could be fail

- promiss ~~has~~ object have 2 thing → 1) Resolve
  2) Reject

States of promiss → 1) Pending
2) rejected
3) fullfiled

```
function saveToDb (data) {                          → used to handle many nested activity.
    return new promise( resolve, reject ) => {
        let intspeed = Math.floor (Math.random (1 * 10) + 1;
        if (internetSpeed > 4) : {
            ~~save~~ resolve ("succes datastored");
        } else : {
            rejected ("failure Data is not stored");
        }
    });
}
```

Promise method

promise is a object Because of these it also have method.

promises have 2 methods which are as → 1) then()

2) Catch()

then() → After the Promise is fullfiled and we want to perform so extra work then it's used.

Catch() → If the Promise is rejected and we want to perform any operation after rejection then we use the catch.

```
let request = savetoDb("om darkunde");

request
    .then(() => {
        console.log("Promise was resolved");
    }).
    .catch(() => {
        console.log("Promise was rejected");
    });
```

Javascript

Promises chaning (Then catch)

- Using multiple then
- there will be only one catch in these chaning operation

```
SavetoDb("Om darkunde")
  .then(() => {
      console.log("data1 saved");
      return savetoDb("Hello everyone");
  })
  .then(() => {
      console.log("data2 saved");
  .catch(() =>{
      console.log("Promise was rejected");
  });
```

we have again called the function in true state
If it is true then the data2 is saved otherwise it goes in catch

result

Used with then

It shows how the final result is formed

- then(result)

error

with with Catch

final result is shown

- catch(error)

Async function IMP

- Createy an Async function

```
async function greet(){
        return "hello world";
}
```

} syntax

```
let hallo = async () => { }
```
— async with arrow fun

- By defaut they all return promiss function
- does not lies if we have writen or not written the promiss statement.

throw ⟶ keyword used to through error in our code

Await keyword IMP

- Pauses the execution of it's surrounding async function until the promise is settleled (resolve or rejected)

Pauses all the nearby Async function until current function does not settle down the promise

Await is only used with the Async function

- try catch is used to handle the rejectection state of the promise.

Javascript

Http Verbs

GET ⟶ To get some information

POST ⟶ To send data from a request

DELETE ⟶ we are trying to delete something through request

Status Codes

200 — ok

404 — Not found (client error)

400 — Bad Request

500 — Internal server error

Statuscode MDN

↳ To get more information about status code

Adding information in URLs

http://www.google.com/search ? q = harry + porter

key ↓ ↳ value

If key Value pair value is wrong then that is ignored and normal execution happens.

Http headers

header, Value

headers supplies additional information

2 types → 1) Request header

2) Responce header

meta data → data about data

Addition information about data.

fetch

- XML HTTP request object are used befor fetch
- But these have lot of ~~proptem~~ problems like async & promise concepts was not working in these

fetch (URL) —— syntax

```
let URL = "https://catfact.ninja/fact";
fetch (URL);
```
} example

```
let url = "https://catfact.ninja/fact"

fetch(Url)
.then((responce) => {
        responce.json().then((data) => {
                console.log(data);
        });
});
.catch((err) => {
        console.log(err);
});
```

To read data

If error occur the error is shown.

Axios

Library to make HTTP requests

internally uses the fetch method but it is a better way

axios git hub —— CDN Link
use this in the .html file

axios.get
axios.post
} method present in this

In fetch we must need to convert the data into JSON format and then we can use that

but in Axios it is in direct JSON format so we can easily Handle that.

```
async function getfacts() {
    try {
        let res = await axios.get(URL);
        console.log(res);
    } catch(e) {
        console.log("error", e);
    }
}
```

→ direct data access

example

1 step of conversion is saved.

Sending headers in Axios

```
const config = {headers : {Accept :"application/json"}};
let res = await axios.get(url., config);
```

↑ convert html code into JSON

Updating Query string in Axios

```
await axios.get(url + Query)
```

↳ The thing you want to search

Javascript

API (Application Programming Interface)

- API is just like a waiter which serves us some services.
- It's a way of communication bet^n 2 software
- The API which uses http requests are known as the web API
- web API returns us data which is in JSON format

JSON (Javascript object Notation)

- www.json.org
- it is a format
- Before this XML was present (extensiable markup language)
- most of the current data uses the JSON format.

* All the keys in JSON format are in string and that is compulsory values could be any data type.

Accessing Data from JSON

1) JSON.parse(data) —— Syntax

To parse a string data into a JS object

parse → convert from on datatype to another datatype

2) JSON.stringify(json)

To parse a JS object data in JSON.

Testing API requests

Tools ⟶ 1) Hoppscoth — online exist

2) Postman —— Very old we need to download this

Ajax

Asynchronous Javascript and XML

it is process from which we call the API and we get data from these.

data format is JSON

The name Ajax is given because before JSON the XML was present and and the JSON combination name was not that much suitable so.