Javascript

- Javascript is the programming language of the web
- Javascript is often used to make various web-development projects like making of swiggy website, designing an amazon clone etc...

→ ways to print in terminals

(i) console.log("Sunshine")

- It is used to print the message written under single/double bracquotes in the terminals.

eg: console.log("Sunshine")

↓

output: Sunshine

- messages can directly be written in the console section under the inspect section

→ Variables and constant

- Variables in javascript are those terms whose values can be changed.

eg: x = 5 → value

x = 5

The value of variable 'x' can be changed as x can be 9, 10 ...

x = 9, x = 5, x = 6 .....

→ Naming convention in Variables in javascript

(i) Variables names are case sensitive

eg: ankit , Ankit

These two are different cases as 'a' and 'A' differs

(ii) Variables names can only include letters, numbers, (_) and $

eg: @Ankit12_$ } ✗ Wrong practice.
Ankit@123

eg: Ankit@123 ✗

(iii) Beginning of variables can start with letter, ~~number~~ or (_) or $.

eg: Ankit123 ✓

123ankit ✗

$ankit, _ankit ✓

(iv) Reserved keywords can't be used

making of a Variable

- Variable names must be meaningful.

eg: studentName = "Ram";

variable value

Printing the above :

```
console.log(studentName)
```

↓ output

(Ram)

eg: console = "Manipur";

↓

(output : Error) (cused of Reserved keyword)

→ const (constant)

- used to assign constant values to the variables

eg: const StudentID = 9876

↓ output

(9876)

updating the values in constant

[redacted] StudentID : 1923

↓ output

(Error)

→ Let

- Values of let can be changed But within the same block
- It is a good practice to use 'let'

→ Var

- Var is an old convention to use.

eg:-

```
Var studentName= "Ankit";
```

↓ output

(Ankit)

updating the values leads to previous update also.

here redeclare occur.

```
Var studentsName= "Ankit";
↓
Var studentsName= "Ankitel";
```

output : (Ankitel)

Data types

- It specifies the type of data that the variable can store like integer, character, floating, double etc...

Types of data type

i) Primitive data type

- these are fixed data type.
- some of fixed data types are
  - i) Number

eg: (0, 100, -25, 3.14)

  - ii) String

- string must be placed within single double quotes.

eg: "a", "ABC", "123"

  - iii) Boolean.

- It can be either true a false
  - iv) undefined.

- It is not same as Null
  - v) ~~null~~ null
  - vi) Bigint
  - vii) Symbol

ii) Non-primitive data type

- these are mainly object.
- object are collection of values.
- key and value are important in object.

Operators

- the one who performs any operation are termed as "operators".

eg:

a+b → expression

a, b → operands
+ → operators

Types of operators

i) Arithmetic operators

- It includes +, −, *, /, %, etc. . . .

eg: console.log(2*5)

↓ output

10

ii) unary operator

↓

Increment operator

- increases the value by 1 (eg: a++)

↓

Post increment operator

- (a++)

↓

eg: Let a=5

console.log(a++)

↓

output: 5

Pre increment operator

- (++a)

↓

eg: Let a=5

console.Log(++a)

↓

output: 6)

Decrement operator

- Decreases the value by 1 (eg: a--)

↓

Pre decrement operator

- (--a)

Post decrement operator

- (a--)

↓

eg:

Let a=5

console.log(a

5

Note

(a+b)++
++(a+b)
} → error

iii) Assignment operator

- It assigns the value to a variable.

eg: a=5

Some of the operator are:

- (+=) operator → prior addition then assign
- (-=) " " Subtraction " "
- (*=) " " multiplication " "
- (/=) " " Division " "

eg: a+=1 → a = a+1

iv) comparison operater.

used to compare the values

(==, !=)

eg:

```
let a: 5
Let b= 10
console.log(a::b)
```

↓output

True.

- Some of comparison operaters are.

(>=, <=0...)

Logical operator

- Some of the logical operators are (AND, OR, NOT)
  - AND (&&)
  - OR (||)
  - NOT (!)
- They have similar function as if in mathematical reasoning.
- && (AND) → used when two condition
  - combines
  - returns True when both condition is true otherwise false.
- OR (||) → True when any condn is t
- ! (NOT) → returns
  - True (when condn is false)
  - false (when condn is Tr

Conditional Statement:

- used to implement some condition in our code.

(i) If statement

↓

when a condⁿ takes place

eg: If(button is clicked)

Application page will get opened

```
Let person age = 20
eg: if (person age >= 18)
        {
        console.log(" Right to vote ");
        }
```

(ii) If-else statement:

- when something is dependent on something

```
eg: Let person age = 20

if (person age <= 20)
  {
  Console.log(" Teenage");
  }
else
  {
   console.log(" Youth");
  }
output => Teenage
```

(iii) else if:

- When multiple cond" are present

eg: Let studentmark = 50;

```
if( studentmark > 90)
    {
    console.log("10 CGPA");
    }
else if ( 90> studentmark > 80)
    {
    console.log(" 8 CGPA");
else if( 80> studentmark > 33)
    {
    console.log(" Average");
}
else
    {
    console.log("Fail");
}
```

output → Average

(iv) Ternary operator

- check & read two condition or mo at same time

- Not a good practice but can be used for fascinating our code

Memory

Stack
(comes under primitive data type)

- performs changes in the copies.

Heap
(comes under Non primitive data types)

- It make changes in the original file

String

- Sequence of character enclosed within single / double quotes.

  eg " Hello evening "

Str.Length

- Str.length is used to determine the length of given String.

  Let str = "Sri"

  eg: console.log(str.length);

  ↓

  output → 3

Template literals in javascript

- It is a way to embedded expression in the strings.

eg:
```
let str = "Hello";
let str1 = "Everyone"
;
console.log(" ${str} and ${str1}");
```

output:
Hello Everyone.

String interpolation

- A way to create string by doing substitution of placeholder

eg: `console.log('Sum is ${9*3}');`

output → 27.

- Escape character : eg: \n, \b....
- String length don't count Null character

Note
- String are immutable (unchangeable)
- str.toUpperCase() → used to do uppercase to all character.
- str.toLowerCase() → used to do lowercase to all the character
- str.trim() → used to eliminate unwanted spaces.

- let str = "Hello";
  let str1 = "Everyone"
  console.log(str.concat(str1))
  ↓ output
  Hello everyone.
  } concat function.

- let str = "Hello"
  console.log(str.replace("e", "O"));
  ↓ output → HOllo.

Maths and Number

eg: let marks = 609
const balance = new Number(600)
console.log(balance);
↓ output
Number.

console.log(balance.toString().length);
↓ output
3

(14)

- const balance = new Number(600.1234)
  ↓
  console.log(balance.toFixed(2));
  ↓
  output → 600.12

- Let number = 100000
  console.log(number.toLocaleString('en-IN'));
  ↓ output → 1,00,000

Math

- console.log(math)
  ↓ output
  Object

- console.log(math.abs(-55564));
  ↓ output
  55564

- console.log(math.round(5.6));
  ↓ output
  6

- console.log(math.ceil(5.1));
  ↓ output → 6.

- console.log(math.sqrt(4));
  ↓ output → 2.

- console.log(Math.random() * 10);
  ↓ output
  5 (output must be b/w 0-10)

- Let min = 50

  let max = 100

  console.log(math.floor(math.random
  * (max - min + 1)) + min);

  → output → 91

Loops

- Loop are used to execute a piece of code again and again

Types of Loop

i) for loop
ii) while loop
iii) Do while loop

i) for loop

Syntax:

```
for(initializatn; condn; after
                          through
{
   ------
   ------ } code
}
```

eg:

```
for ( let i=1; i<=5; i++)
    {
    console.log("xyz")
```

output

```
xyz
xyz
xyz
xyz
xyz
```

eg: let Sum=0;

```
for(let index=1; index<=10; index++)
    {
     Sum= Sum+index;
    }
console.log(sum);
```

↳ output → 55

Infinite loop → never ending loop (not much useful).

↳ condition → always true.

While loop

Syntax:
```
let i=1; → initialize
while(   ){ → condn.
    console.log("___");
     i++;
}
```

Do-while loop

→ the code must be executed atleast one time as the code is executed first and then checks condition.

eg: let index = 1;

```
do {
  console.log("abc");
  index++
} while(index >= 5);
```

↳ output → abc.

for-of loop

↳ used for iteration of string.

Syntax:

```
for (let i of str)
  {
console.log(i)
  }
```

for-in-loop.

↳ used to identify key value of object.

Syntax:

```
for (let index in stud
  {
  console.log(index
```

Array

- An array is a data structure containing a no of data values (Same data type)

creating Array

- Let Arrayname = [46, 46, 88, -]
- Let Array string = [" Raj ", " Verma ", - -]
- used to ~~create~~ store similar data type.

eg:
```
Let Studentmarks = [65, 8, 92, 47,
                    6, 91, 45]
Console.log(Studentmarks);
```

Output → 65, 8, 92, 47, 6, 91, 45.

Note:

- Array is a special type of objects in which key form contains its index.
- Arrays makes changes in its original value as it is being updated.
- Array is mutable.

Loops in Array.

Eg:
```
Let Studentmarks: [65, 8, 98, 45, 6].
for (let index: 0; index < Studentmarks.length;
     index ++)
{ console.log (Studentmarks [index]); }
```

Output →

65
8
98
45
6

Array method

Similar function as a string.

(i) Push method → Add to end.

push() [insertion at last]

eg: Let name = ["abc", "xyz", "pqr"]

console.log(name);

Studentname.push("stu")

console.log(name);

↓ output

['abc', 'xyz', 'pqr']

['abc', 'xyz', 'pqr', 'stu']

↳ Adds to last

(ii) pop() → deletion from last

(iii) toString() → change everything to string

Let

(iv) concat() → Add multiple array

eg: Let money = [10, 20, 30]

Let name = ["abc", "xyz", "par

```
Let newArray = money.concat(name)
console.log(newArray);
```

output

10 20 30

abc, xyz, par

5) unshift() → Adds from starting

6) shift() → Deletion from beginning

7) Slice →

- Syntax:

  slice(startindex, end index)

creates new array.

eg: Let money: [10, 20, 30, 40, 50]

```
console.log(money.slice(2,5));
```

output

20, 30, 40, 50

8) Splice() → makes changes in original array

Syntax:

```
Splice[a, 1,
Splice[startinder, Deletion inder]
```

Object in java

eg: const Saadiuser = {
}

Saadiuser.id = 123
Saadi.name = "abc"
Saadi.email = "abc@xyz.com"
Saadi.isloggedin = false
Console.log(Saadiuser);
// output
id: 123
name = abc
email: abc@xyz.com
Isloggedin: false

→ Nesting

eg: const uber = {
username: "abc",
email: abc@gmail.com",
username: { } → Nesting
userfullname: {
userfirstname: "qaz",
userlast name = "xyz"
}}}
consolelog(uber.name.userfullname.use
Lastname)
// output → xyz

Merge in Objects

```
* const user1 = {
        username : "abc"
        age : 51,
        contact : 9876543210,
        };

  const user2 = {
        username : "pqr",
        age : 52,
        contact : 1023456278,
        };
                    object.assign
  const user3 = {user1, user2}

  Console.log(user3);

  // output

{username : "abc", age : 51, contact : 9876543211,
username : "pqr", age : 52, contact : 1023456878}
```

Object De-Structuring

```
const product = {
        productname : "laptop",
        price : 59999,
        brand : "Hp",
        Ram : "16GB", }

const {productname : pN} = product

    Console.log(pN);

// output
```

How to create and use function

1) function definition

eg: function allrounders()
{
console.log("Ankit" + "pathak");
}
Allrounders()

output → Ankit Pathak

*) function is used in the form where we need to state a code again and again.

eg: function math(number1, number2)
{
console.log(number1+ number2);
}

math(99,2) → {math("99","2") → output 992}
output → 101

Concatanation

→ Return function can also be used to return the output of the function

→ Scoping is an important concept in function

Object as a function

```
const user = {
    username: "Abc";
    email: "ankit@xyz.com";
    }
function name(objeu) {
  console.log("my username is ${objeu.
  username) and my email is ${objeu.
  email}");
  }
```

output name(user)

→ username: Abc
email: ankit@xyz.com

Arrow function

Arrow function is a compact way to write an array.

(- parameter)

Syntax:

```
const functionname = (parameter1, e...) => {
                                            }
```

Scope in javascript

- Scope is the range of any condⁿ
  - var is not effective to use
  - var is lifelong used for scoping.

eg: if (cond)
{
——— → Block Scope.
———
——— }

Scope

Global scope
↓
(can be used under any condⁿ within/outside a function)
↓
eg: if(true){
let b = 9; }
console.log(b); }
↓
output 9

but {
let b = 9;
console.log(—);
}
let b = 10;
console.log(b); → output 10

Block/Local Scop
↓
(can be used only for particu condition)
↓
eg: if(true)
{
let a = 9;
console.log(b);
}.
↓
output → 9.

Hoisting

eg: Let x = 56.

```
xyz()
console.log(x);
  Let x = 56    → Error
```

Note.

Prior declaration of function gives out error.

eg:

```
xyz()
console.log(xyz);
console.log(x);    → Error.
  var x = 56
```

This keyword

i) This keyword in Global space gives {} empty in terminal but window is displayed on the console.

ii) This keyword in function.

eg: function ABC() {
    console.log(this);
}
ABC() ⟶ window

this keyword in function depends upon whether it is in strict mode or unstrict mode. (Due to this substitution).

iii) This keyword in Objects method

eg: const dsa = {
    a: 10,
    b: function() {
        console.log(this); } };
dsa.b()

Output (a: 10 b: f)

(iv) this keyword in arrow function

eg:
```
const phy = {
    a: 10,
    b: () => {
        console.log(this);
    }
}
phy.b();
```

↓

return window

Execution in javascript

javascript execution context

- javascript code runs in two phases:-
  - (i) Global Execution context
  - (ii) functional " "
  - (iii) Eval " "

The phases are:
- (i) memory creation phase
- (ii) Execution phase

eg:

```
Let var1 = 20
Let var2 = 10

function multNum(num1, num2)
    {
      let total = num1 * num2.
      return total
    }

Let result1 = multNum(var1, var2)
```

Step 1 :- firstly global Execution is created & then this is created/allocated

Step 2 : memory creation phase

Step 3 : Execution phase

for getting the result — & new variable + Execution thread is performed.

Done by → memory phase

→ Execution phase

Call Stack

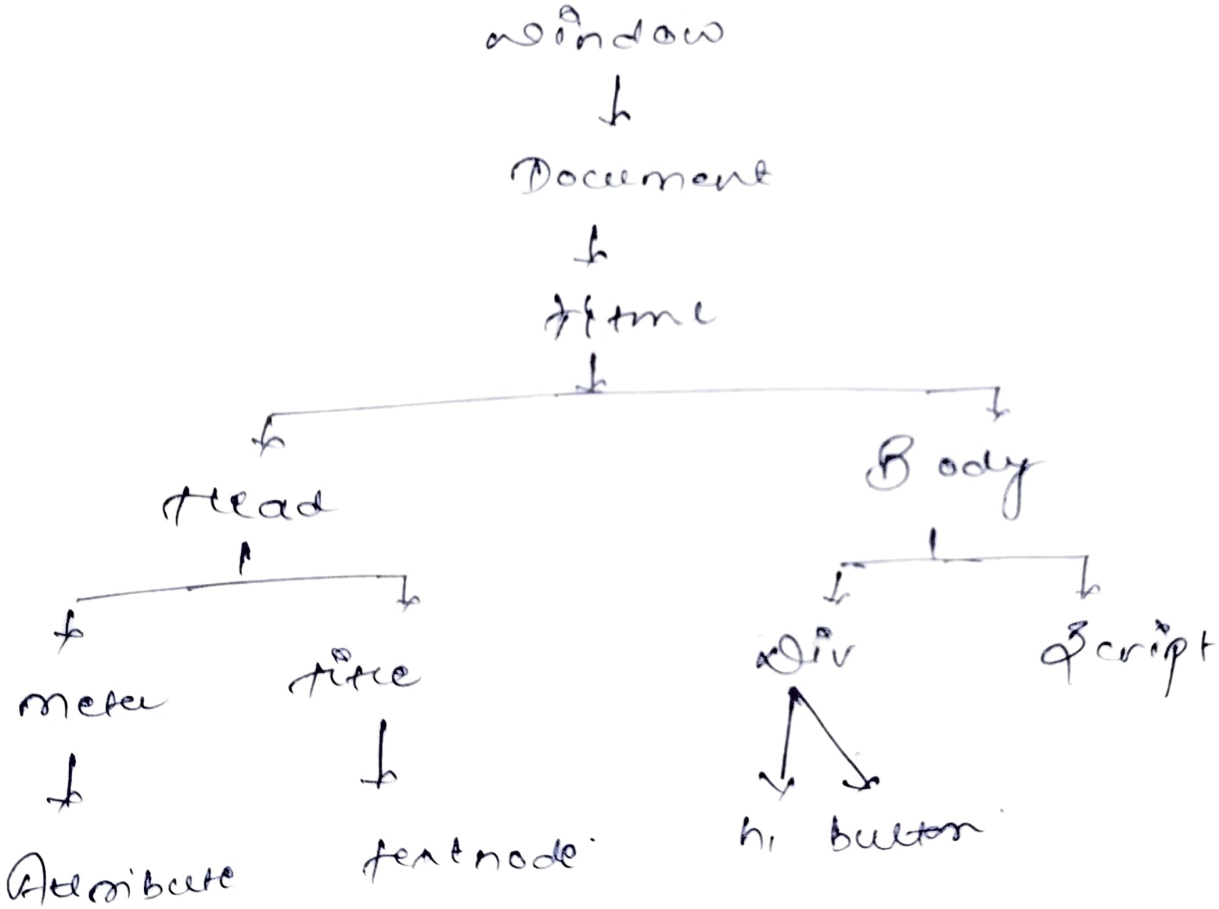
→ Call stack is based on Lifo (Last-in-first-out)

→ Here the last element arranged on the stack should be poped-out first as the name suggest Lifo. (Last-in-first-out)

DOM (Document object model)

- Dom is the representation of object that compose the structure and content of a document of the web. It is a programming interface

Dom model is Given below



- This is used to manipulate the content during the runtime.

DOM manipulation

(i) Ways to access elements:

(i) Id method

Syntax:

document.getelementById("....")

↓

It returns the value of Id

(ii) By using class

Syntax:

document.getelementsByclassName("....")

(iii) By using tagName

Syntax:

document.getelementsByTagName("....")

(iv) Query Selector

Syntax:

document.querySelector("....")

↓

It only gives out first elements

(v) Query SelectorAll

Syntax:

document.querySelectorAll("____")

(i) tagName: returns the tags used

(ii) innerText: returns text content of Element and all its children

(iii) innerHTML → plain text with tags used

Accessing Attributes

getAttributes(Attribute)

(i) getAttribute("Attribute", "value")

for creating new elements we use:

createElement("___")

done by four ways:

i) node.append("element") → Adds at end of the node.

ii) node.prepend("element") → Adds at the starting.

iii) node.before("Element")

iv) node.after("Element")

Events

→ Events are the interesting changes in javascript

Some of the events are given below!

- Keyboard event
- Mouse event
- Print event
- form event

→ These are Some of the events

→ External Event handling.

Syntax!

```
node.event = () =>
{
  ---
  ---
  ---
}
```
code

Event Object

→

```
node.event = (evt) =>
{
  ===
  ===
  ===
}
```
code

Event listener

Syntax:

```
node.addelementlistener(evt, callback)
```

Remove event listener

Syntax:

```
node.removeeventlistener(evt, callback)
```

Asynchronous javascript

→ In Asynchronous javascript execution of multiple lines occurs simultaneously

→ javascript is not Asynchronous, it is Synchronous.

eg:
```
console.log("xyz");
setTimeout(function()
{
console.log("Hello");
}, 2000)
console.log("Hello");
```

output
xyz
Hello
Hello

→ Accessing elements to main stack from the side stack is formed as "Event Loop".

Set time out

Syntax

setTimeout(function(){}, time)

Print Something

```
setTimeout(function(){
    console.log("Ankit")
},9000)
```

↓ output

Ankit

Set Interval

Syntax

```
console.setName=function(){
setInterval(SetName,2000);
```

↓

output

The written line under

Console

API

i) API are basically used for the communication blw two languages like frontend & backend ....

ii) Github is one of the API

XMLHttpRequest

- Some values along with State and Description are given in the MDN web docs ...

value	State
0	unsent
1	opened
2	Header received
3	Loading
4	Done

V8 Engine in Javascript

- V8 engine perform several function like providing Debugging tool and var others
- It is mainly used in chrome and Node

Promise in javascript

- The promise object represent the eventual completion of an asynchronous operation and its resulting value

- promises occurs in three steps
  (i) pending
  (ii) Fulfilled
  (iii) Rejected

```
const promise1 = new promise()
                    ↓
        new promise (function(resolve, reject)
        {

        setTimeout (function()
            {
        console.log("promise1"); resolve()
        }, 1000)
        })

promise.then(function() {
        console.log('consume');
        })
```

output → Promise1

Add

output
Promise1
Consume

Other method

```
Const promise = new promise(function
                (resolve, reject){
    Settimeout(function(){
resolve({ username: "Ankit",
          password: "6325" })
    }, (000)

})

promise.then(function(user){
        console.log(user);
    });
```

output

username: "Ankit password: "632

Fetch API

- The fetch API provides an interface for fetching resources (including across the network). It is a more powerful and flexible replacement for XMLHttpRequest.
- fetch API provides a method to send and receive resources.

using promises

```
function getjokes()
    {
    fetch(URL).then(ANKIT) =>
        {
        Returns ANKIT.json()
    }).then((data) => {

        Console.log(data);
    })
}

Btn.addeventlistener("click", getjokes)
```

Output → New joke.

OOPs

(Object oriented programming)

Classes

- classes are a way to create object templates.

creation of class

Syntax:

```
class myclass {
    constructor() {}
    method() {}
}
```

```
let myobject = new myclass()
```

eg:

```
class car {
    start() {
        console.log("start");
    }
    drive() {
        console.log("drive");
    }
    stop() {
        console.log("drive");
    }
}
```

```
let bolero = new car()
output
Car()
  > Bolero
    prototype : object
      constructor: class car
      drive : f drive()
      start : f start()
      stop : f stop()
```

We can define various other parameter in this

eg: for Car
several parameter are
Speed, brands etc---

Inheritance

- Inheritance are the property that passes characters from parents to child.

- Some of the characters are property and method

Super keyword

- Superword is used to call the constructor of its parents class to access the parent's properties and methods

Call Apply and Bind in javascript

- call is a function that helps us to change the context of invoking function. In other words, it helps us to replace the value of "this" inside a function with whatever value you want.

```
eg: const p1 = {
        firstname : "Anurag",
        last name : "kashyap",
        full name : function() {
          return this.firstname
            + " " + this.lastname }}

    const p2 = {
        firstname : "Riya",
        lastname : "Verma",
      }

    console.log(p1.fullname.call(p2))
```

output

Riya Verma

- Apply is a similar concept to call but parameters are stored in array form i.e ([ ])

eg:

```
console.log(pl.fullname.apply(p2, ["xyz", ob
```

- Bind

eg:

```
const result = pl.fullname.bind(p2, "roshni",
                                "Teacher")

console.log(result());
```

output
Desired output.

Getter and Setter

- used to get and set the values and used to manage classes and object.

Lexical scope and closures in javascript

Eg:
```
function outer() {
  let user = "Ankit"
  function small() {
    let username = "xyz"
    console.log(username)
  }
  function small2() {
    console.log(username)
  }
  small2()
}
outer()
```

→ Error due to scope

Local storage and sessionstorage

eg: `let username = "abc";`

`localStorage.setItem("name", username);`

output → name ——→ abc

↳ saved

Similarly we can setItems in sessionalStorage