JavaScript

(V1) :-

Html ⇒ skeleton

CSS ⇒ styling

Js ⇒ logics & functioning

*what is Js?

→ It is a light weight prog lang (or) scripting lang.

→ It is a client side scripting language (Js if like react & server

*what can we do? with Js?

→ web app, mob app, ~~card~~ CLI interface, network apps, games etc., can be made.

→ So to run JavaScript we need to have Js Engine.

Ex:- firefox uses ⇒ spiderMonkey.

chrome ⇒ V8

→ is nothing but an environment where we can run over Js script (like JVM)

One line definition —

inside browser :- JS running under browser rules.

outside browser :- JS running directly on the OS using Node.js

(Just like JVM)

Q) where should we need to keep script tag either in head or body?

(or)

should Html to render first or JS to run first?

A) To improve user experience we need to render Html first then run JS. (body key andhar dalo vobi downside)

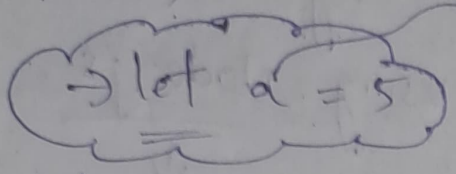
→ To link js file in 'script tag'

<script src = "filename.js" > </script>

→ To run in our IDE instead of browser console then → go to 'pwd' in cmd the " node filename.js"

*Variable:- named memory location is called variable.

How?



Note:- JS is a dynamically typed language. ie we dont need to explicitly mention datatype.

→ We can also create variables using keyword "Var".

→ The difference b/w Var & let is

Scope

Var → global scope
let → local scope.

Keyword	Redeclaration	Reassign	Scope
Var	Yes	Yes	function
let	No (same scope)	Yes	Block
const	No	No	Block

* Constants :- values that are fixed.

Ex:- const a = 5; ✓

a = 10; ✗

→ we can declare variable using let, var, const but on different purpose.

* Variable naming :-

→ Do not use Reserved keyword.

→ meaningful names.

→ cannot start with number.

→ Contain space (or) - (✗)

→ use camelCase for good practice.

* declaring multi variables :-

let a, b or but good practice is

let a;
let b;

primitive types

→ String
→ Number (1, 2, ... 1.4, 2.5 ---- all)
→ Boolean (true, false)
→ Undefined (let a; console.log(a))
→ Null
↓
value assign aye undhi but that value is 'null'.

→ mathlab, variable abi bi define (value assigned) nahi hai.
C++ lo → garbage value isthundhi
JS lo → 'undefined' isthundhi instead of garbage value.

Reference types

→ objects
→ Arrays
→ functions.

*Objects:- multiple variables are linked together is called.

object ⇒ properties + fun

Ex:- let Adithya = {

fname: "Adithya",

age: 22

}

Adithya ki eee properities unnai

→ how to access inside values?

Two ways ⇒

i) dot :-

adithya.age

ii) bracket :-

adithya['age']

Why two types of Notations to access the values in object? (:) ⇒ future lo chudham

* Array:-

A list of similar kind of items (these is in c++) ;).

but in JS =>

It is a Datastructure used to contain a list of items of any type.

Ex:- let create = ['a', 'b', 1, true].

index => 0 1 2 3

create[0]

Create[1]

create[1000] //undefined.

* functions:-

Note:- To execute JS on all browsers then all browsers should follow standard rules. Those rules are defined by an organisation called 'ECMA'

Now present browser following 'ES6' standard.

operators

→ Arithmetic:-

+, -, *, /, %, **

↳ 2**3

2^3

→ post & pre increment:-

++a, a++

→ Anything that is not falsy

→ Assignment operators

=,

→ Comparision operators

>, <, >=, <=, ==, !==, !=

↓ equal to (strict equality) (with datatype check)

↓ Not equal

(or) use '==' to exclude

i.e 3 == "3"
↳ True
but
3 === "3"
↳ false

→ Ternary operator :-

condition ? True : false

Ex:-

```
let age = (age>18) ? "teen" : "child";
```

→ Logical operator :-

- → AND (&&)
- → OR (||)
- → Not (!)

Q) but how does these logical operators work in non boolean ?

A) It follows concept of falsy and Truthy.

falsy	Truthy
→ undefined → null → 0 → false → NaN (all give 'false')	→ Anything that is not falsy

Ex:- console.log(false || 5) // 5

console.log(true || 5) // true

console.log(false || 5 || 1) // 5.

↓

('or' will not check further bcz '5' is not falsy.

* Control Statements:-

→ if-else ——→ if( ) { } else { }

→ switch ↳ else if( ){ }

```
switch(expression){
  case: [ ]
  case: [ ]
  :
  default: [ ]
}
```

Ex:- let num = 2;

```
Switch(num){
    case 1: console.log('A'); break;
    case 2: console.log('B'); break;
    case 3: console.log('C'); break;
}
```

* Loops *

→ For Loop
→ While Loop
→ Do-while Loop
→ ~~Nth~~ For in loop
→ for of Loop.

:) for loop:-

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

(11) :-

objects

→ It is a collection of key-value pairs.

→ It contains properties & methods inside of it

→ It is a a real world entity which as properties, ~~fun~~ methods.

Ex:- let person = {

age : 22,

name : "adithya",

say() { console.log(" ") } ← (or) say : function() {

console.log("my name is adithya")

}

console.log(person.age) // 22

(person.name) // adithya

(person.say()) // my ——

Now if i want 'n' number of 'person' objects then we can't write 'n' lines of codes 'n' times.

So,

→ object creation :- How?

→ factory function

→ Ctor function

① factory function :-

Ex:- function MyObj( ){

    let person = {

        age : 22,

        name : "adithya",

        say( ){ cl("myname is adithya");

    }

    return person; → will create & return obj.

}

let temp = ~~person~~ myobj( );

↳ contains obj which is inside func.

Ex:- (2)

```
function Myobj(name, age){
    name length: name,
        age : age
    say(){c.l("")}
}
```

These type of object creation through function is called factory function.

(2) ctor :- (follow PascalNotation for good prac)

Ex:-

```
function MyObj(n, a){
    this.age = a,
    this.name = n,
    this.say = function(){c.l("")}
}
let adithya = new MyObj('adithya', 22);
```

this represents current object (present it represents an empty object)

↓

will create an object.

* dynamic feature on object:-

Ex:-①:-

```
function Rectangle(len, bre){
    this.length = len,
    this.bre = bre,
    this.draw = function(){ cl(...) }
}
let a = new Rectangle(3,4);
```

⇒ Now if i want to add height also.

then,

```
a.height = 20;
```

if i want to delete then,

```
delete a.height;
```

Now if i want to know how my Rectangle object has created? then—

```
a.constructor
```

↓

will given LOC which is written inside 'Rectangle' function.

Note:- Functions are also objects in JS. i.e the function which we used to create 'Rectangle' obj is also an obj. and this obj is created using ctor called "Function".

→ we know,

Datatypes

Primitive

Ex:- let a = 10;
let b = a;

a [10] 1000 b [10] 2000

So,
++a;
c.l(a) // 11
c.l(b) // 10

Reference types.

Ex:- let a = { v : 10 }
let b = a

[v:10]
a | b ⟶ same like reference

So,
a.v++;

So,
c.l(a.v); // 11
c.l(b.v); // 11

Note:- In primitive types copy is created but in Reference types an address is passed. So no new copy is created.

→ Same in function parameters as well
pass by value ⟶ primitives types.
pass by reference → Reference types

Iterating through objects:-

→ for in
→ for of

(i) for in:-

ex:- let rect = { length : 20,
breadth : 30 }

```
for(let key in rect){
    console.log(key, rect[key])
}
```

output:- length 20
breadth 30

This is why we have two notations to access object values ☺

ii) for-of:-

only applied on iterables
i,e arrays, Maps

Notes: object is not iterable
but their is a roundabout

```
for(let i of Object.keys(rect){
    console.log(i); // will print
                    only keys.
}
```

if,

```
Object.entries(rect)
```

will given both keys, values but in array.

Notes:-

→ Use for-of only for iterable (indexed unnamed)
like array, maps

→ Use for-in for object

for good practice

Q) How can we check if a property is present in object or not?

A) Ex:-

let rect = { l: 10, b: 20 }

Now if i want to check property 'z' is present (or) not?

so, console.log(z in rect) // false.

(l in rect) // True.

---

* object cloning *

→ creation of copies of objects.

→ There are three ways to create.

① Using iteration:-

Ex:- let src = { a: 10, b: 20, c: 30 }

let temp = { } // empty obj

```
for(let key in src){
    temp[key] = src[key];
}
```

Now both src, temp contains same 'ele' but in different locations.

② Assign

here we can pass any no. of objects.
ex:- ({}, src1, src2, src3, ...)

Ex:-

```
let destobj = Object.assign({}, srcobj)
```

1) srcobj lo unna key values ani empty object lo vesthundhi.
2) The created empty object will be assigned to destobj.

③ Using spread:-

Ex:- `let dest = {...srcobj}`

Garbage collection

It

→ Job is to deallocate memory which is no longer used! (Automatic)

→ We have no control over Garbage collector, it is smart and runs in background.

V3

Math object

→ There are many predifined methods which are present in math object.

Ex:- Math.round(1.8)

math.random → generate random num.

Math.PI → π value

Math.max(1,6) // 6

Math.min(6,0) // 0

Note:- Yaadh karne ki need nahi hai ☺

String object

→ but string is primitive right?

Ex:- let n="aadi"

↳ 'n' is now primitive string.

→ JS has two types of strings

i) primitive string.

ii) Object string → (object type String)

Ex:- let a = new String("adithya")

used ctor called 'String'.

→ Some methods —

Ex:- let n = new String("adithya")

=

i) n.length => length-Return

ii) n[0] => a

iii) n.includes("za") // false

4) n.startsWith("Bab") // false

5) n.endsWith("Bab") // false

6) n.indexOf("ad") // 0

7) n.toUpperCase();

8) n.toLowerCase();

9) n.trim() => excludes all start & end spaces if any!

10) n.replace("ad", "AD") // now 'ad' is replaced with 'AD'.

11) n.trimStart()
12) n.trimEnd() } → specifying where to trim.

Note:- all primitive types are immutable. So, whatever the methods which we have used above will not effect (or) change original string. instead they will create an temp string that will be returned.

*template literal :- we use back ticks.

: whatever and how we write the same will be stored.

Ex :- let s = `my
=
name is pa
van aditya
so
am good`

Output :- same like this only points.

Date and Time

Array

→ It is also a type of object (Reference type)

→ Hetegeneous type can be stored.

① Creation:

let num=[1, 3, 5, 7]

index => 0 1 2 3

② Insertion:

a) adding an ele at end:-

[1 3 5 7] —to→ [1 3 5 7 10]

so,

num.push(10)

b) at start: want to add '100'.

so,

num.unshift(100) // [100 1 3 5 7 10]

c) at any position in b/w.

so,

num.splice(2, 0, 99, 98, 97)

2 → index

0 → no. of ele to be deleted

99, 98, 97 → These are the ele to be added

so, [100 1 99 98 97 3 5 7 10]

→ indexOf( ) :- To find the index of the particular element

Ex:- let n = [1, 3, 5, 9, 6]

```
n.indexOf(5); // 2
n.indexOf(100); // -1
n.indexOf(5, 3) // -1
```

searchele ↳ start searching from index?

~~→ what if~~

→ includes() :- To know whether a particular element is present in an array or not!

Ex:- n.includes(5) // true
⇐ n.includes(100) // false.

→ what if we try to find the object inside an array?

A) The above method fails.

So there we use callback functions.

analogy:- inside of main function → ask → mini function

based on mini func ans main function executes

This is called call back function.

Ex:- let person = [

{ SNO : 1 , name : 'adithya' },

{ SNO : 2 , name : 'pavan' }]

So,

indexOf, includes method fails here

cuz if we try person.indexof({SNO:1,name:'adi

will return ~~undefined~~ '-1' cuz we are using an object(address) with other other object(which is in other address)

So, it will return '-1'

but in case of primitives values are compared, but here address are compared!

So, to get 1st object at index '0'.

we have 'find()' method.

So, let ans = person.find(function(p){

if(p.name === 'adithya')

{ return p

});}

write as follows –

```
let ans = person.find((p) =>{ if(p.name===='adithya')
                                return p }
                            )
c.l(ans)
```

output:-

{ sno: 1, name: 'adithya' }

Now the above shortcut is only called Arrow function.

---

*Removing elements from an Array:-

Ex:- let num = [1, 3, 5, 7]

i) removing at end:-

num.pop() // [1, 3, 5]

ii) removing at start:-

num.shift() // [3, 5, 7]

iii) removing from at b/w

num.splice(2, 1) // [1 5 7]

start index — no. of ele to remove.

* Emptying an Array:-

→ We have multiple ways to empty an Array.

i) using empty brackets:- (not recommended)

Ex:- let num1 = [1, 2, 3, 4]

=

~~let~~ num1 = [ ]

c.L(num1) // [ ] (empty)

but it is bad practice, it has one problem.

Suppose,

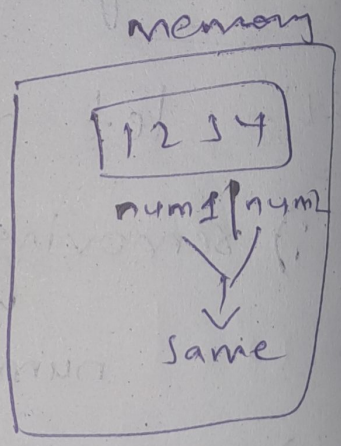
let num1 = [1, 2, 3, 4]

let num2 = num1

num1 = [ ]

c.L(num1) // [ ]

c.L(num2) // [1, 2, 3, 4]



but num1 is empty right? then how come num2 has ele?.

So, it is a bad practice.

ii) Using length=0 :- (good)

so to remove previous problem we use this method.

i.e let num1 = [1, 2, 3]

let num2 = num1

num1.length = 0

c.l(num1) // [ ]

c.l(num2) // [ ]

iii) using splice :- (it is also good prac)

num1.splice(0, num1.length)

---

* Combining & Slicing Arrays *

Ex:- let n1 = [1, 2, 3];

let n2 = [3, 4, 5];

Now i want both n1, n2 in n3

so, let n3 = n1.concat(n2)

output:- [1, 2, 3, 3, 4, 5]

*Iterating an Array :-

→ Since array is a Iterable, we need to use for-of loop.

Ex:- let n1 = [1, 2, 3, 4, 5];

```
for(let i of n1){
    console.log(i);
}
```

→ Using for-each loop, so for each loop expects an callback function.

Ex:- forEach(function(n){c.L(n)})

har ek number se kya karna chahtey ho? — bus print karna chahte hy!

How to convert about callback function to arrow function?

A) forEach((n)=>{console.log(n)})

if single para the '()' is also Optional.

if one line is their then no need of '{}' (optional)

~~Joining Arrays! Split!~~

* split()!-

Ex!- let a = "This is my number";

Now i want to split based on space.

So, let b = a.split(' '); returns an array

c.l(b) // ['This', 'is', 'my', 'number']

Now if i want to join back array to string.

Then,

let c = b.join('_')

c.l(c) // This_is_my_number

---

* Sorting arrays!-

let n1 = [4, 3, 1, 5, 2]

n1.sort() → Ascending order.

c.l(n1) // 1 2 3 4 5

→ To reverse an array (not sort!)

n1.reverse() // [2 5 1 3 4]

n1.sort((n1,n2) => n1 - n2)

[1 2 3 4 5]

Note:- n1.sort() will not sort it correctly because before sorting, the numbers are converted to string then applied sorting. So we have applied arrow function to sort.

---

* filtering Arrays:-

→ To filter ele from an array.

→ expects an callback (or) Arrow function.

Ex:- let n1 = [1, 2, -1, 0, 2, -3, 4, -6]

now i want only +ve ele.

do,

n1.filter((value) => value >= 0)

→ n1[1, 2, 0, 2, 4]

*Mapping Arrays:- (To transform data).

→ maps each element of array to something else.

Ex:- let roll = [60, 70, 80, 90]

let c = roll.map((roll) => return "rollno" + roll)

c.log(c) // ['roll60', 'roll70', 'roll80', 'roll90'].

→ map returns new Array by transforming data of each 'ele' in an array.

→ Original array is not disturbed!.

V4 :-

*functions :-

→ A block code which fullfils a specific task.

→ To create function in JS we ~~have two~~ Two ways to create functions They

i) ~~ii)~~ function funcname (paras) { }

Ex :- function Myname(name){

    Console.log(name);

}

Myname('adithya');

→ what if i write function call upside and below that function definition?

A) still it will execute, because of Hoisting concept

∞ Hoisting :- Before running Javascript the JS engine will send all functions to the top of the file. These it done automatically

ii) function can be created using function assignment

Ex:- a) named function Assignment.

```
let Myname = function say(){ CL("aditya")}
```

do, to call it ⇒ Myname(); (✓)

say(); (X)

and if we write/call Myname() before function Assignment, then it will create error. cuz JS engine will not perform hoisting process for function assignement. hoisting is performed only when functions are created using function declaration.

b) Unnamed function Assignment.

(or)

Anonymous function:-

Ex:- let Myname = function(){ C.L("Aadi")}

Myname();

Ex:- function Sum(a,b){

return a+b; }

Sum(1,2); // 3

Sum(1,2,3,4,8); // 3

---

* arguments Keyword:-

Ex:- function Sum( ){

for(let i of arguments)

{

console.log(i);

}

}

Sum(1,2,3,4,5,.....)

these all will store in a object called Arguments.

* Rest operator :-

-> Same like previous concept arguments. This also works the same.

-> There arguments is a object type

-> Now Here if we use (...) operator then those arguments can be stored in arrays.

Ex :-
```
function sum(...Para){
    for(let i in Para)
    {
        console.log(i)
    }
}
sum(1,2,3,4.....)
```

Note :- if you are using Rest operator in function as formal argument then it should be the last in parameta.

Ex :- function(a,b,...Para) ✓
function(...Para,a,b) ✗

Rest operator :- mi inavi tiskoni okka array lo pedthundhi

Rest → put remaining things inside a bag (Array).

Spread → Throw things out of the bag.

Ex:-
=

*Default Parameters:-

-> Assigning default value to formal parameters.

```
function interest(P, r, y){
    return PTR/100;
}
```

but, if i give 'r' as default para with value then from starting from 'r' to end of list all should be default parameters.

Ex:- function interest(P, r=9, y=10)
= { ---- }

interest(10);