JAVASCRIPT

08-May-24

What is JS?

- → High level programming language
- → Scripted language
- → Interpreted language
- → Synchronous (single threaded)
- → client-sided & server-sided
  frontend backend
- → Object-oriented language

Why JS?

- → To make dynamic webpage
- → To add functionalities
- → To make webpage responsive

Where?

Client sided → Browser (JS-Engine)

server sided → node-JS (V8 + C++)

↓

JRE

Chrome → V8

Mozilla firefox → Sys-Spyder Monkey

Edge → Chakra

- JS is H.L.L (One can easily read)
- JS is mainly used to create dynamic web page
- Browser: It is an application which provides environment to run JS instructions from HLL to MLL
- JS code can run both inside and outside the browser.
- Inside browser — JS Engine
  Outside browser — Node JS Engine
- Node:- It is a combination of C++ and V8 (Chrome JS Engine) with bundles of methods and rules.
- Node provides environment to run JS outside the browser.
- This invention helps the JS to gain its popularity in usage as backend language.
- We can run JS both inside and outside the browser.

Characteristics of JS:-

1. Purely Object Oriented
2. Interpreted language
3. Synchronous (Bz, it follows single threaded architecture which only one ~~code~~ stack)

09-May-24

To execute JS on browser:

1. We can execute js instructions directly in the console provided by the browser.
2. We can execute js instruction by embedding it in html page.
   types: a. internal
          b. external

a. internal:-
- With the help of script tag, we can embedd js instructions:

Syntax:

```
html code...
<script>
  js code...
</script>
html code...
```

b. external:-
- We can create a seperate file for js with an extension ".js"
- link the js file with html page using src attribute of script tag

Syntax:-

```
html code....
<script src="path/filename.js"> (if js file and html file in the same folder path is required)
  js code...
</script>
html code...
```

Token:- Smallest unit of any programming language. And consists of understandable

1. Keywords:- a pre-defined reserved word which is understandable by the js engine is known as a keyword.

Rules for id

Note: every keyword must be in the lower-case and is not used as identifiers. Ex: if, let, for, while

2. Identifiers:- name given by the programmers to the components of js like variables, functions, class etc are known as identifiers.

Rules for identifiers:

- an identifier can't start with number.
- except '$' & and '_' not other special characters are allowed.
- we cannot use keywords as identifiers.

3. Literals/values:- data which is used in the js program is called as literals/values.

types of values in js:

1. Number
2. String
3. Boolean
4. Null
5. undefined
6. Object (non-primitive)
7. Bgint

10-May-24

Variables:-

- A named block of memory which is used to store a value is known as variables (container of store data).

Img Note:-

- in js variables are not strictly typed, it is dynamically typed. Therefore it is not necessary to specify type of data during variable declarations.
- in a variable we can store any type of values
- it is mandatory to declare a variable before using.
- Syntax: var/let/const identifier;

ex:

```
var a; //declaration.
a=10; //initialization or assignment
console.log(a); //10

let b;
b=10;
console.log(b); //10
```

```
const c=30;
console.log(c);//30
```

Note:
When a variable is declared in js, js engine implicitly assigns undefined value to it.

Ex:
```
var a;
console.log(a);//undefined
```

Understanding Execution in JS:

1. Everytime when js engine runs a js code, it will 1st create a 'global execution context'.
2. The Global Execution context has 2 parts:
   a. Variable area
   b. functional/executional area
3. JS engine generally uses 2 phases to execute a JS code.
   phase 1: all the memory is allocated for the declarations in top to bottom order and assigned with the default value 'undefined' in variable area of global execution context.
   phase 2: all the instructions get executed in the top to bottom order in execution area of global execution context.

Ex:
```
c.lg(a); //UD
clg(b); //UD
var a;
a=10;
log(a); //10
var b=20;
log(b); //20
```

Tracing of above code:-

1. Creation of global execution context
2. In the variable area, only variable declarations are stored.
3. for clg(a); //undefined. Because, by default for var a, it is "undefined". Similarly for clg(b)
4. for clg(a); //10. Because, the default value "undefined" is replaced by 10. Similarly for clg(b). It happens in execution/functional area.
5. So, the JS interpreter scans the js code 2 times. Once for variable area and another one for execution area.

Ex

HOISTING :-

- JS allows a programmer to use a member (variable) before the declaration statement. This characteristics is known as Hoisting.

	declare	initialization	re-decl.	reinitialize	Hoisting
var	✓	✓	✓	✓	✓
let	✓	✓	✗	✓	✗
const	✓	✓	✗	✗	✗

Imp Que 1. Variable Hoisting

Que 2. Diff^n b/w var & let

Strings :-

13-May-24

- In JS Strings can be represented using single quotes, double quotes or backticks.

Note:

- The start and end of string must be done with the help of same quote.
- If a text contains single quote then it can be represented using double quotes.

e.g. text: I'm a doctor.

in js: "I'm a doctor."

Backtick :

A backtick can be multiline string

Ex :

```
var hobbies = `My hobbies are:
1. Cricket
2. Movies
3. stock`;

console.log(hobbies);
```

Note:

- The string represented using backtick is also known as template string. The advantage of it is we can substitute and express it using ${ }.

Reversing the String

temp.js:

```
let x = "abc";
let splitValue = x.split('');
console.log(splitValue); //abc
let reversedValue = splitValue.reverse();
console.log(reversedValue); // ['c', 'b', 'a']
let finalValue = reversedValue.join("");
console.log(finalValue); // cba
```

another way:-

```
let x = "abc";
let finalValue = x.split("").reverse().join("");
console.log(finalValue); //cba
```

14-May-24

Arrays.

- It is a collection of both homogenious (similar data types) and heterogenous (different data types) elements.
- Representation of Array is []

ex: let arr = ['Hi', 0, true, undefined, null, {}]

in the above example inside arr variable we have store all the data types, it supports all the data types.

Array methods:-

1. pop()

- It is a method which is used to delete the last element of an

ex: let arr = [10,20,30,40,50];
arr.pop();
console.log(arr); O/P: [10,20,30,40] last element 50 has been d

2. push()

- It is a method which is used to add the element at last

ex: let arr = [10,20,30,40,50];
arr.push(100);
console.log(arr); O/P: [10,20,30,40,50,100]

3. shift()
- It is a method which is used to delete first element of an Array
ex: let arr = [10,20,30,40,50]
arr.shift();
console.log(arr); o/p: [20,30,40,50]

4. unshift()
- It is a method which is used to ~~det~~ add the element at first index
ex: let arr = [10,20,30,40,50];
arr.unshift(100);
console.log(arr); o/p: [100,20,20,30,40,50]

5. splice()
- It is a method used to add and delete the elements at the same time
- Splice method affects original Array
- It will accept 3 arguments
Syntax: splice(startIndex, deleteCount, which element to be added)
ex: let a = [10,20,30,40,50]
a.splice(1,2,100);
console.log(a); o/p: [10,100,40,50]

6. slice()
- It is a method which is used to take out sliced elements from the original Array.
- It will not affect the original Array
- It will accept 2 arguments
- end index will always be ignored.
Syntax: slice(start index, end index)
ex: let a = [10,20,30,40,50]
let b = a.slice(1,2);
console.log(b); // ~~[10,20,30,40,50]~~ [20]
console.log(a); // [10,20,30,40,50]

7. includes()
- It is used to check whatever the element is present inside an array or not.
- Includes will always returns "boolean value"
Syntax: a.includes(elements which you wanted to check)
let a = [10,20,30,40,50]
let b = a.includes(10)
console.log(b); // true (10 is present)

8. indexOf()
- It is used to check the index value of a particular element
- indexOf will returns "index value" of an array.

Syntax: a.indexOf(element which you wanted to know the index)

e.g: let a=[10,20,30,40,50];
let b=a.indexOf(20);
console.log(b); //1 (index value of 20)

15-May-24

Imp Type Coercion :-
- The process of converting one type of data into another type of data by JS Engine implicitly (implicit typecasting), when a wrong type of data is used in the expression is known as type coercion (or implicit typecasting).

ex:-
console.log(10+'20'); //1020, number is converted to string
console.log(10-'2'); //8, string is converted to number
console.log(5-'a'); //NaN, the string does not have a valid number value, hence it converted to a special number 'NaN'

What is NaN?
NaN (Not a Number) is Number !!

Any Arithmetic operation with NaN, result is NaN.

Imp ~~Note~~:
Explicit Typecasting:-
The process of converting one type of value to another type of value is known as explicit typecasting.

Conversion of any datatype to number:

1. String to number:
If a string is valid number, we get the real number.
If the string consist any other character, then we get NaN as output

ex: console.log(typeof 'a');
console.log(Number('a')); //NaN
console.log('23'+10); 2310
console.log(Number('23')+10); //33, string '23' is converted to number explicitly by programmer.
console.log(Number('123a'+10)); //NaN

2. Boolean to number:
If the boolean value is false, is converted to 0.
If the boolean value is true, is converted to 1.

ex:
```
console.log(Number(false)); //0
console.log(Number(true)); //1
```

S.No.	Key	(==)	(===)
1.	Naming	Equality Operator	Strictly Equality Operator/ Identity
2.	Comparison	Type converting: The conversion with performing conversion	Strict Conversion:- without performing any conversion in operands.
3.	Syntax	eg.(a==b)	eg(a===b)
4.	Implementation	first converts the operands into same dt and compare. ie. comparison would perform once both the operands are of same type. This is also known as coercion comparison.	Donot perform any type of conversion before comparison and return true only if type and value of both operands are exactly the same.

UD== NULL //true
UD=== Null //false

Object:

16-May-24

- Any substance which has its existence in the real world is known as object.
- Every object will have properties(attributes of feeds which describe them).
- Every object will have actions (behaviours)

Ex: 1. Car:
- properties of car: model, color, price
- actions of car: accelerate, brake, start, stop

PRAJWAL SIR

16/05/2024

OBJECT

→ Any substance which has its existance in the real world is known as Object.

→ Every object will have properties (attributes of feeds which describe them).

→ Every object will have actions (behaviours)

Ex -

(1) car:

properties of car : model, color, price

actions of car : accelerate, brake, start, stop

(2) webpage:

properties of webpage : url, data/content on webpage.

actions of webpage : scroll, hover, click

⇒

NOTE -

→ In programming an object is a block of memory which represents a real world.

→ The properties of real world object are represented using variables.

→ The actions of real world object are represented using METHODS (FUNCTIONS).

→ In Javascript object is a collection of attributes and actions in the form of key & value pairs enclosed within curly braces {}

→ Key represents properties / attributes of an object.

→ In Javascript, we can create js object in

3 different ways -

(i) Object literal
(ii) using a function
(iii) using a class

(1) OBJECT LITERAL:

Syntax:

let variable = { Key1 : value1 , Key2 :value2, ... }

→ we can access the values of the object with the help of dot(.)operator (period).

Ex-

```
let eDetails = {-----> 'eDetails' stores the address
                       of an object.
    empId   : 1480,
    empName: 'Shiva',
    empsal : '2000'
}
console.log(eDetails); // whole object
console.log(typeof eDetails); // object
console.log(eDetails.empName); // Shiva
```

** NOTE - If the key is not present in the object then we get 'undefined' as the value.

Ex -

```
let eDetails = {
        empId : 1480,
        empName : 'Shiva',
        empSal: 20000
    }
console.log(eDetails.job); // undefined
```

*** NOTE: The objects in javascript are mutable (state of the object can be modified) in nature.

Ex -

```
const eDetails = {
        empId : 1480,
        empName : 'Shiva',
        empSal : '2000'
    }
```

'eDetails' stores the address of an Object.

not possible
~~let eDetails = { }~~

eDetails

```
eDetails.empSal = '20,000 rs'
console.log(eDetails); // whole object but with updated value.
```

→ Yes we can add ~~or~~ a new key value pair and we can update the value also.

→ To add a key & value pair into an Object.

Syntax :-

```
object_reference.key = value
```

17/05/2024

[NOTE: IF THE KEY & VALUE IS NOT PRESENT IN THE OBJECT, IT CREATES A NEW KEY & VALUE WHEN YOU SEARCH]

Ex -

```
const eDetails = {
        empId : 1480,
        empName : 'Shiva',
        empSal : '2000'
    }
emp.Designation = 'Software Engineer';
Console.log(eDetails); // extra key & value is added to the object.
```

TO DELETE A KEY FROM AN OBJECT

→ we can remove a key from an object with the help of delete operator.

Syntax :-

```
delete object-reference.key;
```

Ex -

```
const eDetails = {
        empId : 1480,
        empName : 'Shiva',
        empSal : '2000'
    }

delete eDetails.empSal;
console.log(eDetails); // empSal is removed.
```

20/05/2024

FUNCTION

→ named block of instructions which is used to perform a specific task.

→ function gets executed only when it is called.

→ the main advantage of function is, we can achieve code reusability.

Main Purpose -

code reusability, which means - code once declared can be called anywhere.

NOTE :-

Syntax to create a function:

Generally we can create a function in 2 ways:

1. using function declaration statement (function statement)
2. function expression.

1. FUNCTION DECLARATION STATEMENT:

```
function identifier (parameter1, parameter2, ---)
{
    statement
}
```

NOTE -

→ name of a function is a variable which holds the reference of function object.

→ creating a function using function statement supports function hoisting. therefore we can also use a function before function declaration.

ex:

```
console.log('start');
console.log(test);
function test() {
    console.log("hello");
}
console.log("end");
```

→ when we try to log the function name the entire function definition is printed.

To call a function:

```
function_name('arguments list'); --- function calling
                statement;
```

PARAMETERS

→ The variables declared in the function definition/ declaration is known as parameter.

→ The parameters have local scope. Those can be used inside the function body.

→ PARAMETERS are used to hold the values passed by a calling function.

EX :-

```
function sum(a,b)
{
    console.log(a+b);
}
```

here a&b are variables local to the function sum.

ARGUMENTS

→ The values passed in the method call statement is known as arguments.

→ An argument can be a literal, variable or an expression ~~with~~ which results a value.

EX1 : sum(10,20) --- 10,20 are literals used as arguments.

EX2 : sum(-10+3,-20); // -27

EX3 : let a = 20, b = 30
sum(a,b); // a&b are variables used as arguments.

21/05/2024

RETURN

→ It is a keyword used as control transfer statement in a function.

→ Return will stop the execution of the function and transfers the control along with data to the caller.

Ex-
```
function toMeters (centimeters)
{
    return (centimeter/100);
}
var cms = 2;
console.log (toMeters(cms)); // 0.02
var m = toMeters(cms);
console.log(m); // 0.02
```

2. FUNCTION EXPRESSION

Syntax: var/let/const identifier = function(){}

→ in this syntax the function is used as value.

DISADVANTAGE:

→ The function is not hoisted, we can not use the function before declaration.

→ Reason: function is not hoisted, instead variable is hoisted and assigned with default value undefined.

Therefore typeof a is not a function, it is undefined.

bx - a(); // error, 'a' is not a funtn

```
var a = function(){
        console.log("hi");
    }
console.log(a); //function start or body is printed.
a(); // hi (function call)
a = 10;
a();// error, 'a' is not a function, because a is
        changed to number type from funtn.
```

This we call variable is of Dynamic Type. I can change variable value during runtime.

22/05/2024

SCOPE W.R.T FUNCTION:

Any member declared inside a function will have local scope.

LOCAL SCOPE:

The scope within the function block is known as local scope. Any member with local scope can not be used outside the function block.

Ex-(i) A parameter of a function will have local scope.

function test(a){} ----> here 'a' is variable whose scope is local to test and it can be used only inside test function block.

Ex-(ii) function test(){

var a; --> it is a local to testfuth, it cannot be used outside.

ex.2(iii)

```
function test() {
    function insideTest()
    {
    }
}
```

[NOTE :- insideTest is function, local to test function & cannot be called from outside test()]

→ Inside a function we can always use the members of global scope

Ex:

```
var city = 'Bangalore';
function display(name){
    console.log( ${name} belongs to ${city});
}
display ('sheela');
```

→ in the above ex variable name has local scope and var city has global scope. It is observed inside function body we can use both the variables name & city.

24/05/2024

CLOSURE

→ The closure is a binding of parent function variables with the child function. This enables a JS programmer to use parent functions member inside child function.

Ex-

```
function person(){
    let age = 21;
    function addAge(){
        return ++age
    }
    return addAge;
}
let add = person();
console.log(add()); // 22
console.log(add()); // 23
```

→ the above ex addAge is nested to function to person() the function addAge() does not declare any local variable, but still we are able to use age variable which belongs to person this is achieved with the help of closure.

> NOTE:
> THE BINDING OF CHILD/INNER/NESTED FUNCTION WITH ITS LEXICAL ENVIRONMENT (PARENT FUNCTION STATE) IS KNOWN AS CLOSURE.

POINTS TO REMEMBER ON CLOSURE :-

→ closure helps to achieve scope chain (lexical environment) from child function to parent function.

→ closure preserves the state of parent function even after the execution of the parent function is completed.

→ A child function will have reference to the closure.

→ For every parent and child function relationship a new closure is created.

27/05/2024

** ARROW FUNCTION

→ Arrow function was introduced from ES-6 version of Java Script....

Syntax-

(parameter list, ----) ⇒ { }

→ Arrow functions can have 2 types of return:

(a) Implicit return:

using 'return' keyword is not required.

Syntax:

(parameters-list ---) ⇒ expressions

Example- let add = (a, b) ⇒ a+b

(b) Explicit return:

using 'return' keyword is mandatory with curly braces. {}, if 'return' keyword is not used 'undefined' is given back.

→ If a block is created an arrow function behaves like explicit return.

Syntax:

(parameters_list ----) => { return expressions }

Ex: let add = (a,b) => {
    return a+b
}

NOTE:

→ PARAMETERS ARE OPTIONAL.

→ IF THE FUNCTION HAS ONLY ONE STATEMENT, THEN BLOCK IS OPTIONAL.

→ IT IS MANDATORY TO CREATE A BLOCK IF 'RETURN' KEYWORD IS USED.

Ex -

(a,b) => return a+b ---> syntax error, unexpected token 'return'

(a,b) => { return a+b } ---> solution 1 OR

(a,b) => a+b ; ---> solution 2

28/05/2024

CALLBACK FUNCTION

→ passing a function as an argument to another function is known as callback function.

Advantage-

→ Instead of creating a function with a specific task, functional programming to generate function which can perform a generic task by accepting a function as a parameter.

Ex -

```
function operation (a, b, task) {
    let res = task(a,b);
    return res;
}
```

```
let res1 = operation(10,20, function(a,b){
        return a+b;
});

let res2 = operation(30, 20, function(a,b){
        return a-b;
});

let res3 = operation(10, 5, function(a,b){
        return a/b;
});

console.log(res1); // 30
console.log(res2); // 10
console.log(res3); // 2
```

29/05/2024

IMMEDIATE INVOKE FUNCTION EXPRESSION (IIFE)

→ When a function is called immediatly as soon as the function object is created it is known as Immediate invokation.

STEPS TO ACHIEVE IIFE:-

(a) Treat a function like a expression by declaring inside a pair of brackets.

(b) Add another pair of bracket next to it which behaves like a function call statement.

Ex- (function (a,b) {
console.log (a+b)
})
(4,4); //8

CONSTRUCTOR FUNCTION

→ A function which is used to create an object is known as constructor function.

Syntax-

```
function identifier(parameter, ---) {
}
```

NOTE-

→ If the function is designed to use as a constructor then the name of the function should always be UpperCamelCase.

→ List of parameters provided to the function will be treated as the keys (properties) of the object.

→ The arguments passed when the function is called will be the values.

→ we can copy the value into the keys of object from parameters using 'this' keyword.

→ we can create an object using constructor function with the help of 'new' keyword.

Syntax :-

```
new function();
```

NOTE:
'new' keyword creates the object and returns its reference.

Ex -

```
// function construction

function Student (sid, sname) {
    this.sid = sid;
    this.sname = sname;
}
let s1 = new Student (1, 'Paretni')
let s2 = new Student (2, 'Totlu')
let s3 = new Student (3, 'Chinku')

console.log (s1.sid); // 1
console.log (s1.sname); // Paretni
console.log (s2.sid); // 2
console.log (s2.sname); // Totlu
console.log (s3.sid); // 3
console.log (s3.sname); // Chinku
```

TYPES OF FUNCTIONS -

(1) NORMAL FUNCTION -

```
function funthName/Identifier () {
    ===
}
```

(2) ANONYMOUS FUNCTION

```
function () {
    ===
}
```

(3) FUNCTION WITH EXPRESSIONS

```
let/var/const  variableName = function () {
    ===
}
```

(4) PARAMETERIZED FUNCTION

```
function  funtnName/identifier (parameters, ---) {
    ===
}
```

(5) RETURN TYPE FUNCTION

```
function funtnName/identifier () {(parameters, ---)
    ==
    return argument
}
```

(6) NESTED FUNCTION

```
function funtnName/identifier () {
    ===
        function funtnName/identifier () {
            ===
        }
}
```

(7) ARROW FUNCTION

```
() => {(parameters, ----)
    ===
}
```

(8) CALLBACK FUNCTION & (9) HIGHER ORDER FUNCTION

```
function funNname/identifier (parameters,...) {
                                  ↳ higher order function
}
funName (arguments,...., function funNname () {
                                  ↳ callback function
})
```

(10) IIFE (Immediate Invoke function Expression)

```
( function (parameters,...) {
    return ~~err~~ ____
}) (arguments,...)
```

(11) CONSTRUCTOR FUNCTION

```
function funNname/identifier (parameters,...) {
    this. .....
}
```

04/06/2024 SPREAD OPERATOR & REST PARAMETERS

REST ~~OPER~~ PARAMETER -

→ Rest parameter is used to accept multiple values as an array of elements.

→ generally it converts multiple value into an array.

Syntax -

```
... identifier
```

Use of Rest Parameters -

(1) rest parameters can be used in function definition parameters list to accept multiple values.

(2) it can also be used in array or object destructuring.

using rest parameters in a function -

(1) the function can relieve multiple arguments passed by the caller and store them in an array.

Ex :-
```
function test1(a, ...b){
    console.log(a); //10
    console.log(b); // [20,30]
}
test1(10, 20, 30);
```

[NOTE: REST PARAMETER SHOULD BE ALWAYS DEFINED AT THE LAST]

Ex :-
```
function test1(...a, b){
    console.log(a);
    console.log(b); // syntax error: Rest parameter must be last formal parameter.
}test1(10,20,30);
```

SPREAD OPERATOR -

→ It is used to perform unpacking, it uses an iterator to access each n every element or a property present in the array or object.

Ex: 1. For Array

```
let a = [10, 20, 30];
console.log(Math.max(...a)); // 30, using spread opa.
console.log(Math.min(...a)); // 10
console.log(Math.max(a[0], a[1], a[2]));
console.log(Math.max(a)); // NaN
console.log(a);
```

Ex: 2. For Object

```
let obj1 = {
        id: 123,
        company: 'Google',
        Salary: '5lpa'
}
let obj2 = {
        name: 'Amit',
        ...obj1
}
console.log(obj2); // {
                      name: "Amit",
                      id: 123,
                      company: 'Google',
                      Salary: '5lpa'
                   }
```

05/06/2024

LOOPING STATEMENTS

1. forEach:

→ This method is used to loop through each element of an array or object this method takes a callback function as an argument is invoked forEach element of the array or object.

Ex :-
```
arr.forEach((x, y, z) => {
    console.log(x, y, z);
})
```

2. Map :

→ This map() method used to loop through each element of an array or object this method takes a callback function as an argument.

→ callback function is involved for each element of the array or object.

[the map() is similar to the forEach() method but map returns a new array.]

Ex:-
```
let arr = [10, 20, 30, 40]
let value = arr.map((x, y, z) => {
    conole.log(x, y, z);
})
console.log(values);
```

DIFFERENCE BETWEEN forEach and map :-

1. The map() method returns a new array but the forEach() method doesnot return a new array.
2. The map() method is used to transform the elements of an array, whereas the forEach() method is used to loop through the elements of an array.

3. for of :

→ The for of method executes a loop that operates on a sequence of values sourced from an iterable object.

```
Ex :- for (let a of arr) {
        console.log(a);
      }
```

4. for in :

→ The for in method is used to return the index values.

```
Ex :- for (let a in arr) {
        console.log(a);
      }
```

06/06/2024

5. filter() :- & 6. reduce() :-

```
let Books = [
        { name: "HTML", Price : 350},
        { name: "CSS" , Price : 500},
        { name: "JS" , Price : 800},
        { name: "React", Price : 600}
        ]

// find price less than 600.
let a = Books.filter(( x ) => {
        return x.price < 600

})
console.log(a); // [{name: "HTML", price: 350},
                   {name: "CSS", price: 500}]

// Apply gst to the above price obtained
let b = a.map(( x) => {
        return x.price * 0.18 + x.price

})
console.log(b); // [{name: "HTML", Price : 413},
                   {name: "CSS", Price : 590}]

// get sum of the above price
let c = b.reduce((acc, cv) => {
        return acc + cv.price

},0)
console.log(c); // 1003
```

07/06/2024

BOM - (BROWSER OBJECT MODEL)

```
let person = {
        name : "paglait",
        id: 102
}
```

①
```
function greet() {
    console.log(this); // window
}
```
object { }

② Call

```
function greet() {
    console.log(this); // person
}
greet.call(person)
greet()
```
object { } window { }

③ apply

```
function greet(a,b) {
    console.log(this);
    console.log(a,b);
}
greet.apply(person, [1,3])
greet(10,12)
```
o/p — person { } 1 3 window { } 10 12

④ bind

```
function greet(a,b) {
    console.log(this);
    console.log(a,b);
}
grt = greet.bind(person)
grt(1,3) → grt(10,12)
```
o/p — person { } 1,3 person { } 10, 12

Object Methods

↳ keys()

```
let personKey = Object.keys(person)
console.log(personkey); // ['name', 'id']
```

↳ values()

```
let personValue = Object.values(person)
console.log(personValue); // ['paglait', 102]
```

↳ entries()

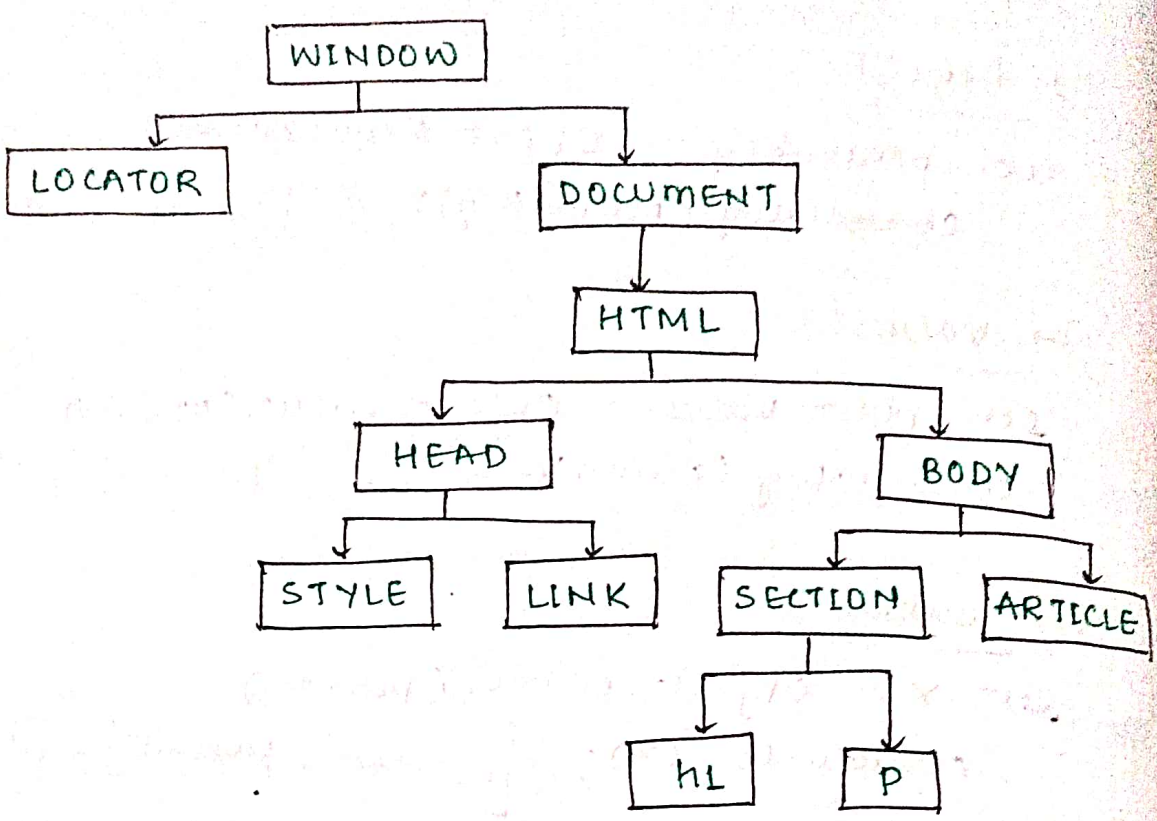
```
let x = Object.entries(person)
console.log(x); //[['name', 'paglait'], ['id', 102]]
```

11/06/2024

DOCUMENT OBJECT MODEL

"The DOCUMENT OBJECT MODEL (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document".

DOM STRUCTURE



STRUCTURE –

Document ⇒ html file

Object ⇒ tags, elements, nodes

Model ⇒ layout, structure

* When web page is loaded, the browser creates a DOCUMENT OBJECT MODEL of the page.

* The HTML DOM model is constructed as a tree of objects:

  With the object model, JavaScript gets all the power it needs to create DYNAMIC HTML:

  → JS can change all the HTML element in the page.

→ JS can change all the HTML attributes in the page.
→ JS can change all the CSS styles in the page.
→ JS can remove existing HTML elements and attributes.
→ JS can add new HTML elements and attributes.
→ JS can react to all existing HTML event in the page.
→ JS can creat new HTML events in the page.

12/06/2024

DOM SELECTORS or DOM METHODS

(1) document.getElementById() –

→ document.getElementById() function targets and return only one particular specified element based on id value.

Example –

fun.html

```
<body>
<h1 id = "head"> Hello World </h1>
</body>
```

fun.js

```
let heading = document.getElementById("head").
                        innerText
console.log(heading);
```

(2) document.getElementsByClassName() -

→ The function targets and returns all the element with matching class value.

→ It returns HTML collection object and we can access the values using index value/positions.

Example -

fun.html

```
<body>
<h1 class = "a" > Hello world </h1>
<h1 class = "a" > Hello world </h1>
<h1 class = "a" > Hello world </h1>
</body>
```

fun.js

```
let data = document.getElementsByClassName("a")
console.log(data[0]); // <h1> Hello world </h1>
console.log(data); // HTML collection.
```

(3) document.getElementsByTagName() -

→ This function is used to target or fetch an element from the document object model.

→ This function is used to target the element based on tagname.

Example -

fun.html

```
<body>
<h1> Hello world </h1>
<h1> Hello world </h1>
</body>
```

fun.js

```
let tagName = document.getElementsByTagName("h1")
console.log(tagname[0].innerText); // HelloWorld
console.log(tagName); // HTML collection.
```

(4) document.querySelector() -

→ Query Selector function is used to target an element using tagname, Id value, class value.

→ In whichever input available first that is tag name, id value, class value it returns the particular element.

Example -

fun.html

```
<body>
<div> Div1</div>
<div id="x"> Div2 </div>
<div class="a"> Div3 </div>
</body>
```

fun.js

```
// it will check for the first iteration
let a = document.querySelector(".div")
console.log(a); // <div> Div1 </div>

let b = document.querySelector("#x");
console.log(b); // <div id="x"> Div2 </div>

let c = document.querySelector(".a");
console.log(c); // <div class="a"> Div3 </div>
```

(5) document.querySelectorAll()-

→ querySelectorAll() function is used to target all the elements using tagname, id, class.

→ it returns all the elements in the form of NodeList.

→ it will return (NodeList[]) <== it is an impure Array .. we can perform only forEach Method.

→ to perform map method first we have to convert it into pure Array.

Syntax -

```
variableName
Array.from(ele)
```

→ to check array is pure or not

Syntax -

```
Array.isArray(variableName)
```

it will return boolean value.

Example -

fun.html

```
<body>
<div> Div1 </div>
<h1> Heading1 </h1>
<h1> Heading2 </h1>
<div> Div2 </div>
<div> Div3 </div>
</body>
```

fun.js

```
let div = document.querySelectorAll("div")
console.log(div); // it will target all the div
                     element and return it as a
                     Nodelist.
```

14/06/2024

DOM EVENTS

(1) onclick()

→ It is an event in DOM whenever we wanted to peyform some specific task after clicking on particulay Element.

→ Tanget the element which you wanted to click and then we an attribute "onclick" and anjom a function call to it.

Ex1: index.html

```
<body>
<h1 onclick="a()"> click </h1>
</body>
```

fun.js

```
function a(){
  console.log("h1 element is clicked")
}
```

Ex2: write a program to creat dark and light theme using onclick event.