Date

History

1. Javascript was first created by Brendan Eich in just 10 days in May 1995 while he was working at Netscape communication corporation.
2. The intial release was called Mocha and later renamed to Livescript and finally Javascript.
3. Initially, Javascript was designed to be a lightweight scripting language for adding interactivity to webpages.
4. At the time webpages were mostly static and lacked interactivity and the only way to add dynamic content to a webpage was to use a server-side programming language like php or perl.

* This allowed web developers to create more engaging and interactive websites without having to rely on server-side programming language.

* In 1996, Microsoft released JScript as a competitor to Javascript which was their own implementation of the language for their internet Explorer browser.

* However, JScript ~~which~~ was written very similar to Javascript and the two languages were ~~kept~~ largely interchangable.

Javascript

* Javascrith is scripting and programming language.
* It is purely object based language. This means that variable function or even primitive data type like numbers and strings are object everything is object in Javascript

Spiral

Date

3. It is dynamically typed language, It means type of value stored in memory block is checked at runtime because of this nature we can store any type of value of variable.

④ It is object oriented programming language it means we can create our own object it is not purely object oriented programming language)

⑤. It is interpreted language

⑥ It is synchronous language, it has single threaded architecture instructions get executed line by line

⑦ It is a single call stack

⑧ Mainly introduced to interact the browser.

9. Js helps to provide behaviour and functionality to webpage and helps to develop dynamic webpage

⑩ Every browser have Js engine to run Js code. Therefore browser become environment to run Js code.

⑪ To run Js code multiple browser we just need Javascript runtime environment (Node)

⑫ Js is used to add functionality to website.

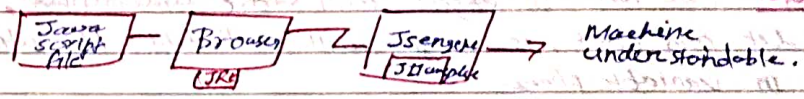
Date

JRE (Java script Run time environment)

* Every Javascript run time environment have java script engine.
* Two popular javascript runtime environment are browser and nodeJs both are application.

JS engine

Js engine is a program which reads Javascript code line by line and convert into machine understandable code.



Browser	Js engine
Chrome	V8 engine (fastest)
firefox	Spidermonkey
IE (I.E or Edge)	Chakra
Safari	Javascript Cans

Token : Smallest unit of programming language.

- keyword, :- for, double, while, if. etc.
- identifier :- rules
  - ① should be not a keyword
  - ② should be not start number
  - ③ should have special character, except '$' or '_'
  - ④ should not be space-separated.
- Literals :- value is used in program,
  Types of literals / datatype

Date

① Number: integer + Decimal
range $(-2^{53}+1$ to $2^{53}-1)$

② Bigint
integer + Decimal
range $<-2^{53}-1$ to $2^{53}-1>$
we have to suffix the number with 'n'
Boolean: true or false.

③ undefined
It is used to initialize the memory, done by JS engine implicitly in variable phase

Note: For var JS engine will store 'undefined'. for let and const, JS engine will will left the utilized in variable phase.

NaN
It stand for not a number.
It represents computational error.
when JS engine is not able compute result it returns 'NAN'.

Null
* The datatype represent a null or empty value
* It is used to mark the memory block empty intentionally

⑦ Symbol
This is represent unique Identifier.

⑧ String
It represents of character collection of characters.

Date

single line string: here linebreak and white spaces not allow. using single quotes and double quotes.

Multiline string -
→ here line break and white space allow
→ using backticks (` `)
→ It is also refferred (template string).

Primitive literals/Primitive datatype
* Primitive datatype is a single value data
* It is a immutable data.

Non primitive literals / Non primitive data.
* Non primitive data is a multi value data
* It is a muttable data.

human = 0x1
human.name = "Gowri"

0x1
name: "Gowri"
addr: record
kid: 4

Ways to coped Jscode in html.

① Internal JS:-
The Jscode written inside HTML document is refred as in ~~HTML~~ Internal JS
→ To insert Js code, we have <Script> .... Jscode. </Script>
→ script tag should be inserted at least of </body> Tag.

② External JS:-
Here we create separate file to write Jscode and file should be saved with Js extention

Date ____

To link external JS file we have to provide source path of JS file using src attribute of script tag.

Console

Console is API.
It is not method in JS code.

Type coexion

→ The process of converting one type of data to another type of data.

→ Two Type of coexion

Implicit: Implicit is convert one type of data to another type of data automatically by JS engine.
"11" + 1 ⇒ 111
"11" − 1 ⇒ 10

Explicit: Explicit is convert one type of data to another type of data by user.
Number()
String()
"11" + 1 ⇒ 111
Number("11") + 1 ⇒ 12
Number("11") − 1 = 10

Var	let	const
[Aurav]	[Rajat]	[Decream]
var Ravi = Bebo	let Sohail = "Angel"	Committed
var Ravi = 'Shelke'	Sohail = "Pratap"	const Prasant = "Chunfi";
	Sohail = "Sanam Gupta"	const Prasant
→ multiple time declaration	→ one time declaration	→ one time declaration
→ Reinitialization allowed	→ Reinitialization allowed	→ One time initialization → at the time of declaration we have to initialize the variable.

Date ____

JS Code → Browser [JRE] → JS engine → [ ] = Global execution context
window [ox1]
[ox1]

Variable phase	Execution phase
a [10]	a = [10]
	c.log(a);

output:
10

Two phase
① Variable phase: JS engine search for variable declaration statement and allocate memory
② Execution phase: All instruction set executed line by line

Hoisting

* The ability of Java script engine to access the variable before its declaration statement is called as hoisting

* Variable declared with var, let, const can be hoisted.

Temporal dead zone

* The time frame between variable declaration and initialization in this time frame we can not access because it is empty so JS engine gives us error because we are trying to access memory block which is empty. We are trying to access value of memory block, this time frame referred as Temporal deadzone.

* Variable will come out of the temporal deadzone when we initialize it.

* Variable declared with let, const goes to temporal dead zone.

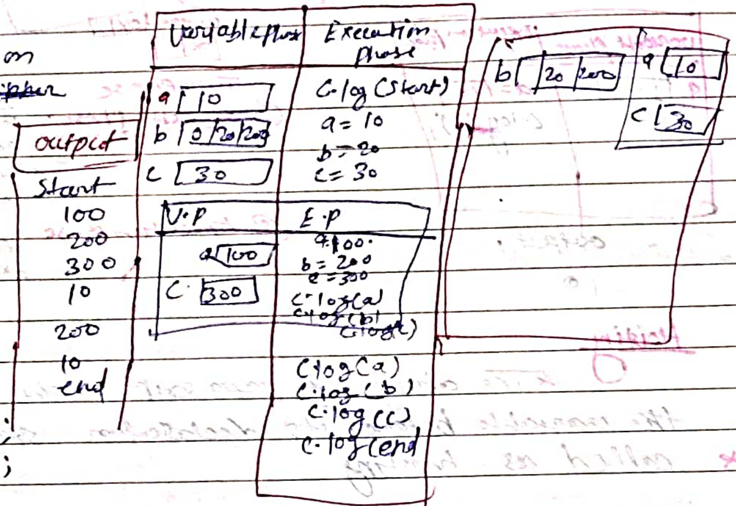
Spiral

Date

Scope

- Global
- Local
  - Function
  - Block ~~scope~~

```
c.log(start)
let a=10;
var b=20;
const c=30;
{ let a=100;
  var b=200;
  const c=300;
  c.log(a)
  c.log(b)
  c.log(c)
}
c.log(a)
c.log(b)
c.log(c)
c.log(end);
```

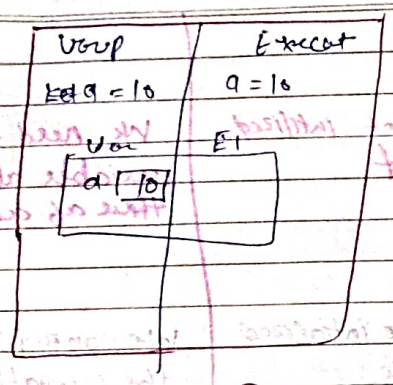


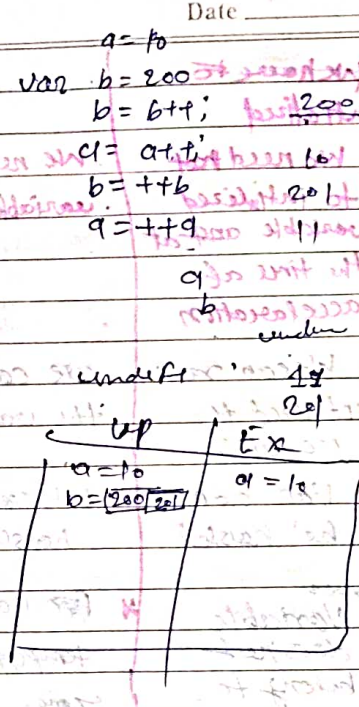
Var	Let	Const
* Variable declared with var goes to global scope	variable declared with let goes to local scope	Variable declared with let goes to local scope
* We can redeclare variable with same name in same scope	* ~~let~~ we cannot redeclare variable name and in same scope	we cannot redeclare variable with same name in same scope

Date

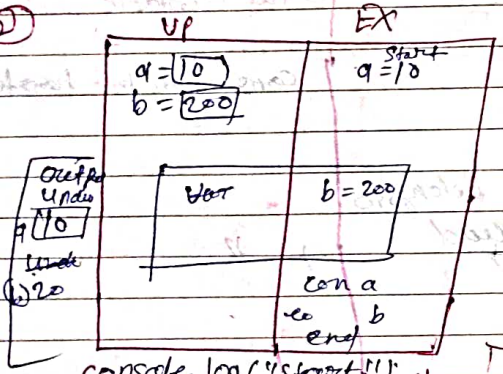
~~We have to initialized~~ We need ~~not~~ to initialized variable and at the time of acceleration	We need not initilized variable at	We need to initilized variable at the time of acceleration
* We can re initilized the variable	we can re intialized the variable	We cannot re-intialized the variable
* Var can be hoisted	Let can be hoisted	Const can be hoisted
~~for var~~ Variable does not belong to temporal dead zone	* ~~for~~ variabl belongs to temporal dead zone	→ 11
* Variable declared in block will ~~also go~~ to global space	* For variabl declared in block will be ~~avaibavailable~~ available inside block only	* variable declared in block will be available inside 'block only
* Variable declared inside function ~~can't~~ out cannot be access ~~outside~~ out	variable declared inside function cannot be access out side.	—

Date



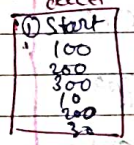


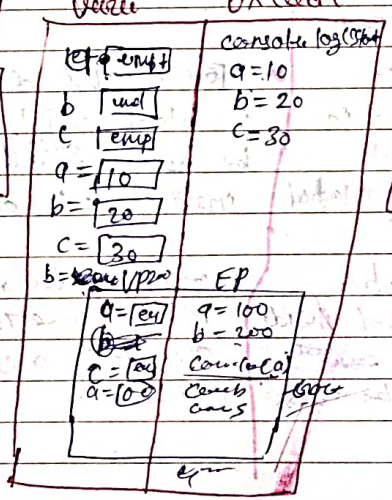
②

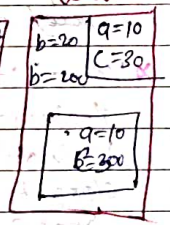


①

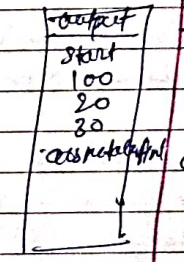
```
console.log("start");
let a=10;
var b=20;
const c=30;
{
let a=100;
var b=200;
const c=300;
console.log(a);
console.log(b);
console.log(c);
}
console.log(a);
console.log(b);
console.log(c);
console.log("end");
```

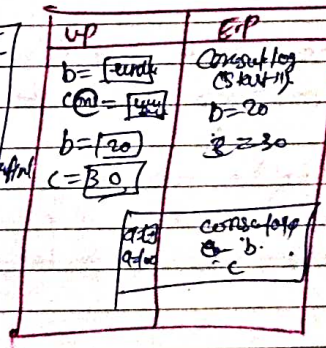




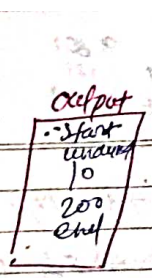


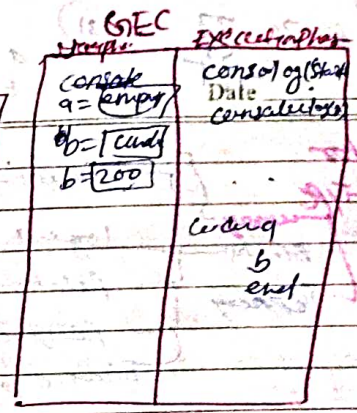

```
console.log("start");
let a=10;
var b=20;
const c=30;
{
let a=100;
console.log(a);
console.log(b);
console.log(c);
}
console.log(a);
console.log(b);
console.log("end");
```

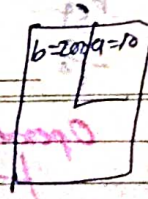





```
console.log("start");
let a=10;
console.log(b);
{
var bb=200;
}
console.log(a);
console.log(b);
console.log("end");
```

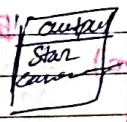


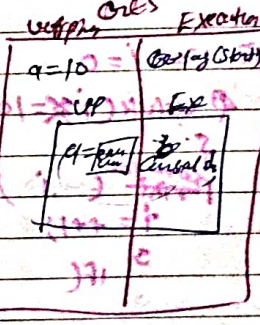


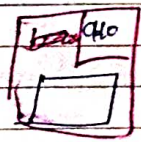


④

```
console.log("start");
let a=10;
{
console.log(a);
let a=10;
}
console.log(a);
console.log(b);
console.log("end");
```

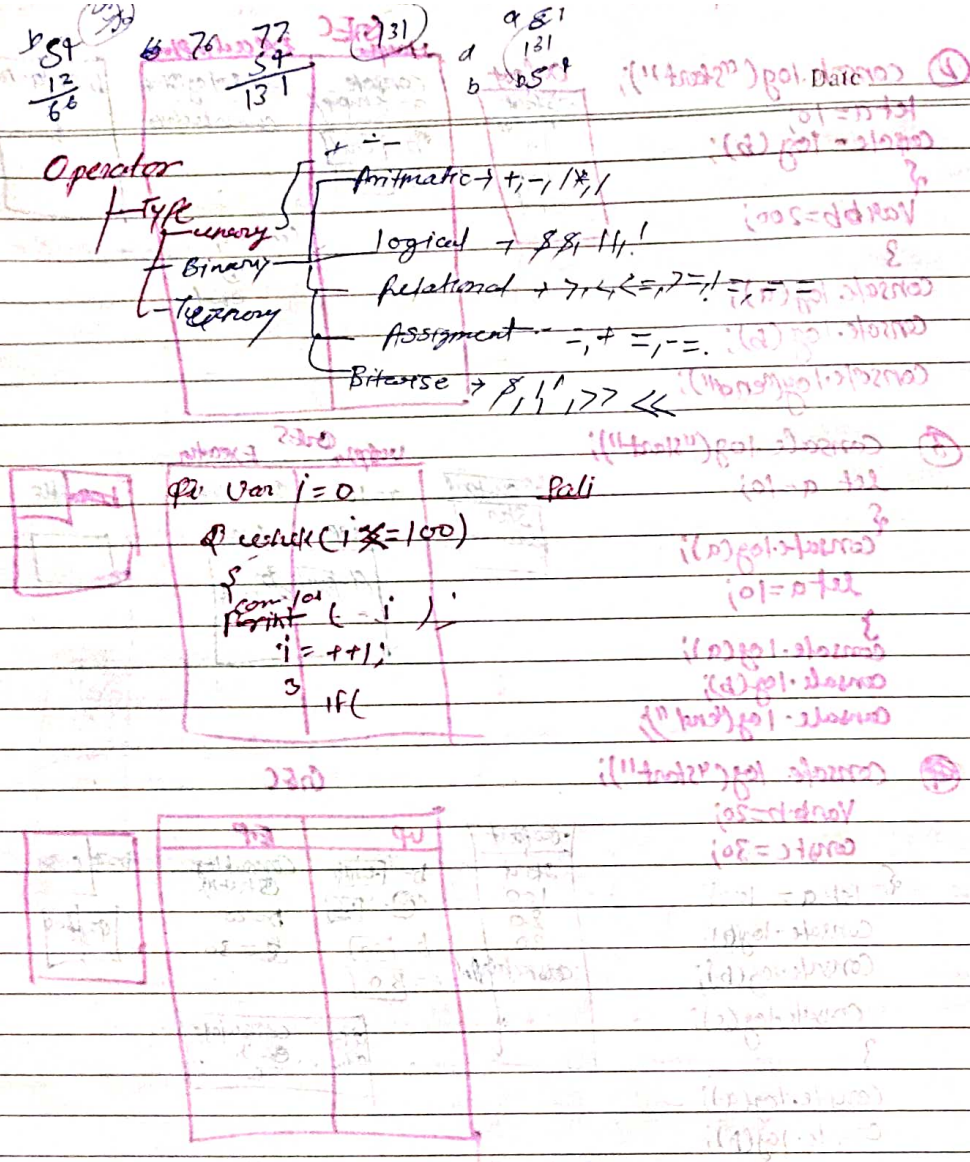






⑤

```
console.log("start");
Var b=20;
const c=30;
{
let a=100;
console.log(a);
console.log(b);
console.log(c);
}
console.log(a);
console.log(b);
console.log("end");
```



Date _____

Functions:-

1. Function is object.
2. Function is a block of instruction which is used to perform a specific task.
3. A function get executed only when it called.
4. The main advantage of function is we can achive code reusabl
5. To call a function we need its reference and ().
6. Name of function is variable which holds the reference of function object.
7. Creating of function using function keyword supports functio hoisting.
8. Therefore we can also call a function before function declarat
9. When we try to log function name the entire function defination is printed.
10. The scope within function block is known as local scope.
11. Any member with local scope cannot be used outside the function block.
12. A parameter of function will have local scope.
13. Variable written inside function even using var have local scope.
14. Inside a function we can use the member of global scope.
15. In javascript this is a property of every function. (every function will have 'this' keyword except arrow function)

Parameters

1. The variables declared in the function defination is known as parameters.
2. The parameters have local scope (can be used only inside functio body)
3. Parameters are used to hold the values passed by calling a function.

Date

Arguments

1. The values passed in the method call statement is known as arguments.
2. Note: An argument can be a literal variable or an expression which gives a results.

Return keyword.

1. It is a keyword used as control transfer statement in a function.
2. Return will stop the execution of the function and transfer control along with data to the caller.

Ways To Create Functions

1. function declaration statement:
   Create using function key word
   * syntax:

```
function func_variable (parameter){
    // statements
}
function variable ()
```

* Example: Create a function 'greet' which should print a message "Good Morning" when it is called.

```
function greet(){
    console.log("Good Morning");
}
greet();
```

output: Good Morning

Date

* function can be hoisted

```
// Here we are accessing function before it's declaration statement.
greet();
function greet(){
    console.log("Good Morning");
}
```

output : Good Morning

* function does not belongs to temporal dead zone.

2. function as expression (expression function:
   * function which is passed to an variable as a value is called as first class function.
   * function can not be hoisted because it is objects is created in execution phase.
   * function does not belongs to temporal dead zone.

Syntax: let a = function(){ ... } — use console statement 3

Functional Programming.

1. Functional Programming is a programming technique where we pass a function along with a value to another function.
2. In this approach, we generate reusable function. Here function task is not only single task.
3. The function which accept another function as a parameter or return a function is known as 'Higher Order function'.
4. The function which is passed to another function or the function which is returned by another function is known as call back function.

Anonymous function :- when function use only one time then we are create anonymous function.
when function create function without function name, is known anonymous function

```
(function(){
})
setTimeout(function, ms)
```

Date

Types of functions :-

1. function declaration statement: using function keyword
2. function as expression / expression function
3. Immediate Invoke function (IIF)
   * When a function is called as soon as its object is created is known as immediate invoke function.

* We have to write the function inside the parenthesis to group it. [using Group operator -> (function code)].
* The function is not visible (available) outside the scope.
* After grouping it, we have to use parenthesis to called this function.
* Immediate Invoke function execute only once.

Syntax `(function(){ console.log("Hello") })();`

4. Arrow function
   * The main function of arrow function is to reduce the function syntax.
   * Arrow function is introduced in ES6.
   * If we have only single parameter, It is not necessary to use parenthesis for parameter
   * If function have single statement then block (parenthesis) is optional.
   * It does not have its own this property.
   * Implicit return :- If there is only one statement and its block is not created then JS Engine will return that statement. automatically

Date

Explicit return :-
If block is created and function is not returning any value, JS Engine will return undefined. To return a value Explicitly from block, we have to use returned keyword. If block is created the we have to use return keyword to return value otherwise JS Engine will return undefined.

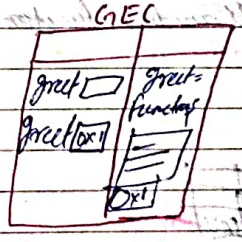
⑤ Higher Order function
* The function which accept another function as a parameter or return a function is known as higher order function.

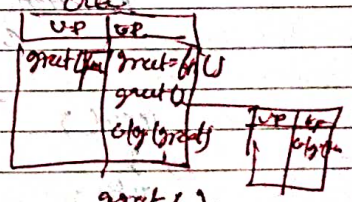
⑥ Call back function
* The function which is passed to another function or the function which is returned by another function is known as 'Call back Function'.

⑦ Eg Function as expression/expression function

```
let great = function(){
    console.log("Good Morning");
}
```

First class function





Immediate Invoke function

```
(function(){
    console.log("Good Morning");
})();
```

Date

Arrow function => ECMAScript

To reduce function syntax

```
Syntax function sum(a,b){
          return a+b
       }

let sum = (a,b) => {          ← Flat operator
          c.log(a); c.log(b)
          return a+b;
       }
```

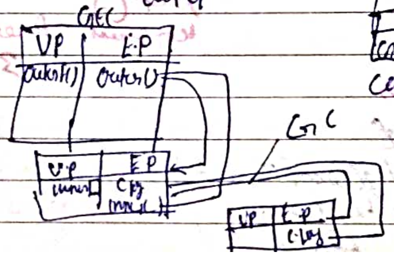
functional programming

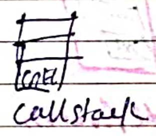
```
let cal = (task, a,b) =>{
          return task(a,b);
       }

let sum = (a,b) => a+b;
let sub = (a,b) => a-b;
let mult = (a,b) => a*b;
let Div = (a,b) => a/b;
```

→ Nested function

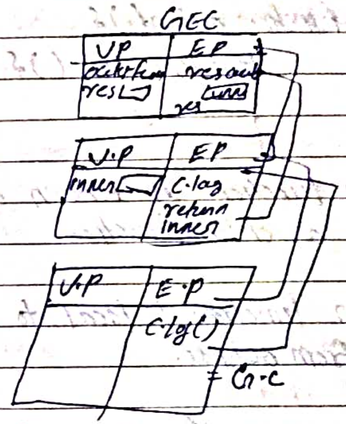
```
function outer() {
     c.log("I am outer function");
     function inner(){
          c.log("I am inner function");
     }
     inner()
}
outer.
```





Date

```
function outer(){
     c.log("I am outer function");
     function inner(){
          c.log("I am inner function");
     }
     return inner;
}
letter = outer()
         inner
res()
```



Array

① Print All string and boolean element of array.

② Print the prime number in array.

```
Let arr1 = [10,3,7,20,"MONDAY","SUNDAY", true, false, 97,
            100, 400];
For For[i=0, in arr)
{
     console.log(arr[i])
}   IF (arr
```

Date

Nested function :-

1. The function inside another function is called as nested function.

2. Example

```
function outer(){
    function inner(){
    }
}
```

3. The outer function is known as parent and the inner function is known as child function.

4. The inner function is local to outer function, it cannot be accessed from outside.

5. To use inner function outside, the outer function must return the reference of inner function:

```
function outer(){
    function inner(){
    }
    return inner;
}
```

We can now call inner function from outside as follow

1st way:

```
let fun = outer();
fun(); // ---> inner() is called
```

2nd way

```
outer()(); // ---> inner() is called.
```

Date

Lexical Scope / Scope chain :

1. The ability of JS engine to search for a variable in the outer scope when variable is not available in local scope is known as.

Lexical scope or scope chain.

2. It is ability of child to access variable from outside if its ~~or~~ not present local scope..

3. Lexical scope : A function and global object.

```
let a = 10;
function test(){
    a+1;
    console.log(a);
}
test();
```

output : 11

When test function is executed JS engine looks for a in local scope. Since it will not available it will looking for a in outer ~~space~~ scope that is global window object.

4. Lexical scope: The child function and parent function with a help of closure.

```
function outer(){
    let a = 10;
    function inner(){
        console.log(a);
    }
    return inner;
}
let res = outer();
```

output = 10

When the function inner is executed and consol.log a is encountered, JS engine looks for a in the local scope of function inner.
Since a is not present and function inner is child of function ~~order~~ outer JS engine will search for a in the parent function outer scope with the help of closure.

#Closure
1. A closure is created when a function is defined within another function and inner function need to access variables in the outer function's scope.

2. Closure helps to achieve lexical scope from child function to parent function.

3. Closure preserves the state of parent function even after the execution of parent function completed.

4. A child function will have reference to the closure.
5. Every time a parent function is called the new closure is created.
6. Disadvantage: High memory consumption.

Array
1. Array is object in javascript
2. It is non-primitive type of literals.
3. It is block of memory which is used to store multiple type of value (any type of literal) in same memory block.

4. Array size is dynamic (size is not fixed number like Java)

It means we can store 'N' number of elements and JS engine will handle memory usage automatically.

5. Values stored inside array are refered as array elements.
6. Array elements are arranged in a sequence that is represented by integer number called as index. Array index starts from zero to array size - 1 (suppose array has 5 elements its first index will be = 0 and last index will be 4).
7. We can access the array element with help of array object reference with the help of array object reference square brackets and index (array object ref[index]).

8. If we try to access the index that it greater than the array length we will get undefined.

9. Array elements should be separated by comma(,)

Ways to create Array
1. By using square braikets [] and literals.

```
let arr=[];
// empty array
let arr=[10,20,30];
// array with literals.
```

2. By using new keyword and Array() Constructor.

```
let arr = new Array();
// empty array
let arr = new Array(10,20,30)
// array with literals → [10,20,30]
```

Spread

Date

Note: Here, 'arr' is a variable which holds the reference of array object. To access ~~any~~ array element at index → 1.

Syntax: array ~~element~~ object_ref[index]

Example: console.log (arr[1]); //20

Array Methods

1. push(value) method

* It is used to insert element at last of array.
* It returns the length of array.
* Example

let arr= [10,20,30,40,50];
arr.push(100);
output: [10,20,30,40,50,100]

2. pop() Method

* It is used to delete element from last index of array.
* It return deleted element.
* Example

let arr= [10,20,30,40,50];
arr.pop();
output: [10,20,30,40]

3. unshift (value)

* It is used to insert element at first index of array
* It return array length

Date

Example

let arr =[10,20,30,40,50];
arr.unshift(200);
output: [200,10,20,30,40,50]

4. Shift() method

* It is used to delete element from first index of array.
* It return deleted element.

Example:

let arr = [10,20,30,40,50];
arr.shift();
output: [20,30,40,50]

5. splice() method

* It is used to perform insertion, deletion and updation in array.
* It will modify the original array.
* It returns array of deleted elements.

Syntax:

arr_ref.splice(a,b,c)
a- Starting index
b- number of elements to be deleted
c- Element to be inserted.

* Example: Delet three elements from index 1.

let arr =[10,20,30,40,50];
arr.splice(1,3); // deleted: [20,30,40]
console.log(arr);

output: [10,50]

Date

* Example: Update value at index 3 to 500.

```
let arr = [10,20,30,40,50];
arr.splice(3,1,500);
console.log(arr);
output: [10,20,30,500,50]
```

* Example: Insert 100,200, and 300 from index 2.

```
let arr = [10,20,30,40,50];
arr.splice(2,0,100,200,300);
console.log(arr);
output: [10,20,100,200,300,30,40,50]
```

6. Slice() method
* It is used to copy array elements.
* It will not modify the original array.
* It return array of copied elements.
* Syntax:

```
arr.slice(a,b);
```

a - Starting index
b - last index

Here, last index is excluded → last index - 1

* Example: copy array from index 0 to 2.

```
let arr = [10,20,30,40,50];
let copy_elements = arr.slice(0,3);
Console.log(copy_elements);
output: [10,20,30]
```

Date

7. IndexOf() methods
* It used to get the index of array element
If element is available → it returns elements index.
* If element is not available → it returns -1;

Syntax:

```
arr.indexOf(a,b)
```

a - Value to be searched
b - Search starting index
If we does not pass last argument, it will 0 by default.

* Example: check given array has element 30 or not and search from index 0 and 3, if present print index.

```
let arr = [10,20,30,40,50];
console.log(arr.indexOf(30)); // 2
console.log(arr.indexOf(30,3)); // -1
```

8. includes() method
* It is used to check element is available or not
* if element is available → returns true.
* if element is not available → returns false.

Syntax:

```
arr_ref.includes(a,b);
```

a - value to be searched
b - search starting index
If we does not pass last argument, it will 0 by default.

Date

* Example: check given array has element 30 or not and search from index 0 and 2, if present print true.

```
let arr = [10,20,30,40,50];
console.log(arr.indexOf(30)); //true
(console.log(arr.indexOf(30,3); //false
```

9. reverse() method.
* It is used to reverse the array
* it will modify the original array

Example:

```
Let arr = [10,20,30,40,50];
(console.log(arr.reverse())
```

Output: [50,40,30,20,10]

10. Sort(Callback) method.
* It will modify the orginal convey.
* If callback returns -ve value → it will sort in ascending order.
* If callback return +ve value → it will sort indecending order.
* If callback returns 0 value → it will not sort.

Example: Sort array in ascending order.

```
let arr = [100,2000,300,990,50,0,2];
Consok.log(arr.sort((a,b) => a-b)
```

output [0,2,50,100,380,990,2000]

Date

Example: Sort array in desending order..

```
let arr = [100,2000,300,990,50,0,2];
Consok.log(arr.sort((a,b) => b-a));
```

Output: [2000,990,300,100,50,2,0]

(11) Array.isArray(literal)
* It is used to check given literal is array or not
* if it is array → It will return true.
* If it is not array → it will return false.

```
consok.log(Array.isArray(2));  // false
Consok.log(Array.isArray(10)); // false
consoke.log(Array.isArray([10,20,30])); // true
```

(12) Foreach(Callback)
* It is a higher order function.
* It is used to iterate over array elements and index.
* It does not return anything, so JS engine implicitly returns undefined.

Syntax

```
arr_ref.forEach((value, index, array) => {
        // statements
})
```

* Example: Print even number from given array.

```
Cons arr = [1,2,3,4,5];
arr.forEach(Val) => {
    if (Val % 2 === 0).
    {
    consale.log(Val+" "+"is even number");
    }
});
```

Date

Assignment - 1

1. What is Javascript (write 6 points)

1. Javascript is scripting and programming language.
2. It is purely object based language. This means that variables, functions, and even primitive and even primitive data types like numbers and string, everything is object in Javascript.
3. It is interepreted language.
4. Mainly introduced to instruct the browser.
5. It is single call stack.
6. Js helps to provide behavior and functionality to webpage and helps to develop dynamic webpage.

2. What is JRE? Name two JRE?

1. Java Script Runtime Environment provides the enviroments where we can run our Javascript code.
2. Two Javascript runtime enviroments are
   (i) Browsen
   (ii) Node.Js

3. Write name of JS engines of chrome, firfox, Edge and Safari?

1. V8 (Faster Js engine): developed by Google, used in Google Chrome and Node.js.
2. SpiderMonkey: developed by Mozilla, used in Firefox
3. JavaScriptCore: developed by Apple, used in Safari
4. Chakra: developed by Microsoft, used in Microsoft Edge and internet Explore (legacy)

Date

What is JS engine?

1. A Javascript engine is a computer program that execute Javascript code.
2. It is a core component of web browsers, server-side Javascript plakforms, and other Javascript based environ

Difference between Var, let and Const?

#Var	#let	#Const
1. Variable declared with var goes to global scope	(i) Variable declared with let is block scoped.	(i) Variable declared with const is block scope.
2. We can redeclare variable with same nam	2. We cannot redclare variable with same name in same scope.	2. We cannot redclare variable with same name in same scope.
3. We can ~~declare~~ update the value of variable	3. We can update the value of variable	3. We can not ~~declare~~ const without initialization be modified.
4. We can declare variable without In intiallization	4. We can declare variable using let without initialization. But JS engine will keep that memory block uninitialized (empty) until JS engine reads declaration Statement	4. We can not declare const without intialization
(5) Variable declared with var, can be hoisted.	(5) Because let variable is unitinitialized empty in variable phase It belongs to Temporal Dead Zone.	(5) The variable declared using const is hoisted and belongs to temporal dedd zone Therefore it cannot be used before initialization (Because at that moment it is unitialized - TDZ)

Spiral

Date

⑥ What is hoisting ?

The ability of Javascript engine to access the variable before its declaration statement is called as hoisting

Variable declaration with var let const can be hoisted.

⑦ Temporal Deadzone :-

The timeframe between variable declaration and ~~int~~ initialization in this ~~find~~ time frame. We can not access the variable because it is empty So JS engine gives us error because we are trying to access memory block which is empty. (We are trying to access value of memory block) This timeframe is refered as Temporal dead zone.

→ Variable will come out of the temporal dead zone when we initialize.

→ If variable declared with let and const goes to time dead zone.

Date

⑬ map (callback)

* It is a higher order function.
* It is used to iterate ove array.
* It will not modify original orginal array.
* It returns a new array.
* The value returned by callback function will be inserted in new array, if it doesnot return anything 'undefined' will be stored.

Syntax

```
arr_ref.map((value, index, array) => {
    // statements
})
```

* Example : Create new array where each element of given array is multiple of 8.

```
let arr = [10, 20, 30, 40, 50];
let new_arr = arr.map(value => value * 8);
console.log(new_arr);
```

Output : [80, 160, 240, 320, 400]

⑭ Filter (callback)

* It is a higher order function.
* It is used to iterate over array
* It will not modify original orginal array.
* It return a new array.
* Here element will be inserted in new array only when call back function return true.

Syntax

```
arr_ref.filter((value, index, array) => {
    // statements
})
```

Date ______

* **Example** : Create new array where elements are greater than 90.

```
let arr= [10,20,30,40,50,60,70];
let new_arr = arr. filter (value =>{
    if (value >30)
        return true;
});
console.log (new_arr);
```

output : [40,50,60,70]

Find smallest ~~number elem~~ array.

Find smallest greater element in array.

```
Arr = [30,40,50,10,9,40]
Function =Arr1= [ ]     let J =
    for (let i= 1, i > arr.length, i++ )
    {
        for (let J=i
        if (Arr[J] > Arr[i]
            Arr[0 ]
```

Find the sum of even number and odd number in array print the greatest sum.

```
let arr = [[10,20,30],
           [100,200]]
let arr. function sum() {
    for ( value of arr)
    {            let = J
        arr2 [J ] = Arr[i]
        for(value of arr2)
            sum = sum + Arr2[J]
    return sum }}
console(sum)
```

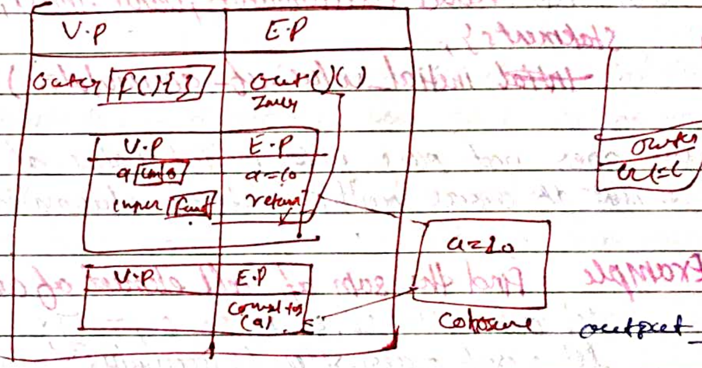
Nested function

Date ______

```
function outer()
{  let a=10;
    function inner(){
        let a=20
        c.l.o(a);
    }
    return inner
}
outer()()
```

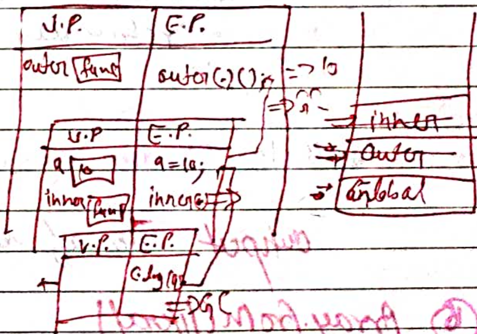
a = 10

0x1 Closure



output =10

```
Function outer( )
{
    let a=10;
    Function inner(){
        c.log(a);
    }
    inner();
}
outer()();
```



Output

Date

Reduce (callback, intial_value)

* It is a higher order function
* It is used to iterate and conclude result to single value.
* It will not modify original array
* It returns a single value.
* Here, single value is returned after complete iteration of array. Value is stored in a variable which is used to result, we refer it as accumulator.

* Syntax:

```
arr_ref.reduce(accumlator, value, index, array)=>{
    statements},
    ~~intial~~ initial_value_of_accumulator)
```

If we does not pass ~~intial~~ initial value of accumulator first element of array will be stored automatically.

* Example Find the sum of all element of array.

```
let arr = [10, 20, 30, 40, 50, 60, 70];
let result = arr.reduce((acc,value)=>{
        acc = acc+value;
        return acc;
    },0);

Console.log("sum of all elements" result);
```

output: sum of all element: 280

(16) Array.from(literal)

* It is used to convert iterable literals (like objects or string) to array.
* If literal is iterable → It return new array of elements

Date

IF literal is not iterable → it return empty array.

Example: Convert string to Array.

```
Const str = "Hello";
Const arr = Array.from(str);
Console.log(arr);
output ["h", "e", "l", "o"]
```

Object

1. An Object is a block of memory which has state (variable) behaviour (methods) and where we can store hetrogenous data.

* An object is collection of key value pairs that can contain various data types, such as numbers, strings arrays, functions, and other objects.

(2) In one objects we can have multiple key value pair and it should be separated by "Comma".

* We can access value of objects using (.) operator or squar bracket [] object reference and key-name

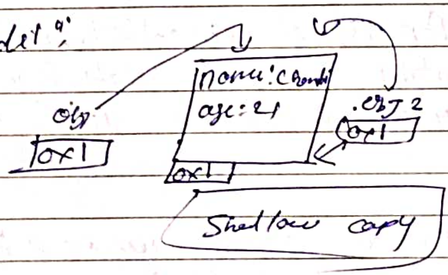
Date

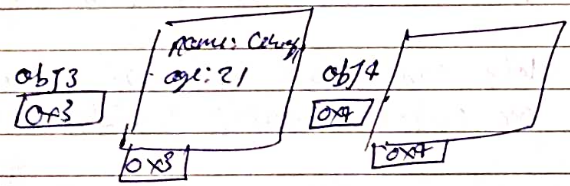
Object

```
Let obje = { name: "saurabh", age: "21", address: "Pratapgarh" }
Console.log ( obje.name );
or
Console.log ( obje["age"]);
let object2 = { name:, age:, address:; }
let name
let name = promt (" Enter your name");
let age = +Promt (" Enter your age");
let addre = Promt (" Enter your address");
console.log ( object2 ["name"] );
Console.log ( object2 ["age"]
Console.log ( object2 ["address"] );
```

copy

```
let obj = { name: "Adit",
            age: 21,
          };
let obj1 = obj1;
    0x1    0x1
obj.name = "Chombi";
c.log (obj1);
c.log (obj2);
```






```
let obj3 = { ___ }
let obj4 = { };
for ( let key in obj3)
{
  obj4[key] = obj3[key]
  obj4.name = obj3.name
              "Chombi"
}
```

JSON – Javascript object notation
It is a format of data only. Date

Two method.
→ JSON.stringfy()
  → It is converts is object into JSON.
② JSON.parse() → It convert JSON into JS object.

call, Apply, Bind method

```
Let obj1 = { name: "chombi",
             age: 21,
             details: function(a,b){
               c.log (this.name, this.age)
               c.log(a,b)
             }
obj1.details()
```

```
function getdetails(a,b){
  c.log (this.name, this.age)
  c.log (a,b);
}
```

① call
getDetails.call (obj2,1,2)

call() method
obj function.ref.call(object.ref, arg1, arg2, arg3):
obj1.details.call(obj2, 1, 2)

Apply() fun.ref.apply (obj.ref [obj-ref[arg, arg2 ...]];
obj1.details.apply(obj2 [10,20]) / args 10 20,
② get details.apply (obj3 [1,2]);

3. Bind(); fun.ref.bind (obj.ref (args, args);
let fun = obj1.details.bind(obj3, 100,200)
fun();
fun();

Date

Object Key (Property)

1. Object Key (Property) will be automatically converted into string by JS engine.

2. If key name are in Number, JS engine will convert them into string and arrange them in assending order.

3. To write space separated key name, we have to endose key name with double quotes

* If we want to give computer or user defined property then we have to use square brackets and variable name
* If key-name is same as variable name which hold the value, instead of writing two times we can write variable name only one.

```
let phone = 8800425655;
let obj = {
    Phone,
    //
        Phone: phone
    }
```

Way to Create Objects

1. By using curly braces {} and literals.

```
let obj = {}
// empty object
let obj = { name: "chombi", age: 16 }
// object with literals
```

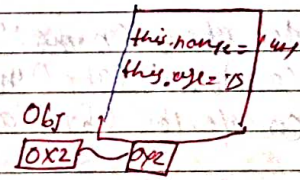
Date

By using new keyword and constructor.

```
let obj = new Object();
// {} empty object
let obj = new Object({ name: "chombi" });
// { name: "chombi" } object with literals
```

5. By using new keyword and constructor function

```
function CreateObject(name, age) {
    this.name = name
    this.age = age
}
let obj = new CreateObject("John", 20)
obj.name
let obj = {
    name: "Abc"
    age: 15,
}
obj.city = "Noida";
```



Add, update, delete

Add — objref./[] keyname = value

```
Object. city = "Noida";
```

update = objref.or[].keyname

```
obj.age = 10
obj1.["age"] = 10;
```

Date

delete

delete obj.ref.[] keyword
delete obj1.name
delete obj["name"]

Access Object Value

1. By using dot operator(.) and key name

```
let obj = {name: "chombi", age:16}
console.log(obj.name) //chombi
console.log(obj.age)// 16
```

2. By using square brackets ([]) and key name

```
let obj = {name:"chombi", age:16}
console.log(obj["name"]) //chombi
console.log(obj["age"] //16
```

3. If we try to access property which is not available in object we will get undefined.

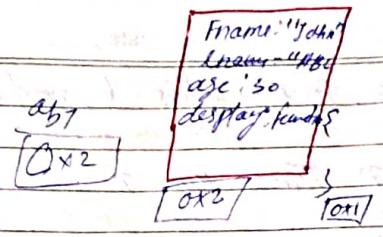
Object Method

Methods → function to attached to object, and called on the reference.

```
let obj={
    fname: "John"
    lname: "Abc"
    age: 30;
    display: function() {
        console.log("Hello I am a master");
```

(keyword)

this (called current refference of object)



Date

```
Object.ref: {[], [keyword]}
obj.display();
obj["display"]();
```

Object Methods

+ In javaScript, object methods are functions that are attached to the object, and can be called on that object reference.

+ To call a function, we use square brackets instead dot operator.

Here, speak is a variable which holds the function reference

```
let obj1 = {name: "chombi",
    age: 16,
    speak: function() {
        console.log("i can speak");
    }}
console.log(obj1["speak"]());

// I can speak
```

* Access object property inside function - function declared with function keyword.

```
let obj = {name: "chombi",
age: 16}
speak: function() {
console.log(`my name is ${this.name}, and I can speak`);}}
console.log(obj1["speak"]());
```

Date

Here, we can access object property, by using this keyword.

* Access object property inside function - Arrow function.

```
let obj1 = { name: "Chombi",
    age: 16,
  Speak: () => {
  console.log (`My name is ${obj1.name}, age ${obj1.age}
   and i can speak`);
   }}
  Console.log (obj1["speak"] ());
   //My name is chombi, age 16 and i can speak
```

//Here we can access object property, by using object reference.

~~Access~~ * Here, we can access object property, by using object reference because arrow function is not having 'this' property

Add Key Value in object

* To add key-value pair we can using dot operator and square brackets.

* By using dot operator (.) and key name

```
Let obj = {name:"Chombi", age : 16 }
obj.country = "India";
// new key value added in object
// $
name: "chombi",
age: 16;
country: "India"
}
```

Date

Note If property is already available with same name then updated with new value

Example

```
let obj = { name:"chombi", age:16 }
obj.age = 18;
// age property value is updated
// {
  name: "Chombi",
  age: 18,
  }
```

Check Property is Available In object or Not:

1. We can check using "in" operator.

Syntax : "property name" in object_name

```
let obj = { name : "chombi, age : 16 }
console.log ("name" in obj) // true
console.log ("age" in obj) // true
console.log ("city" in obj) // false.
```

copy of object.

We can create copy of two types:

* Shallow copy
* Deep copy

Shallow copy

* The copy of object that is directly connected with original object is called as shallow object.

Date

* Here, we store reference of original object in a new variable now new variable starts pointing to same memory block.

* So if we make any change in copy, it will be reflected to original object because both variable are pointing to same memory block.

```
let obj = { name: "chambi", age: 16 }

let obj_cpy = obj;
//reference of obj is copied in obj_cpy.
obj_copy.age = 20;
console.log (obj_copy);
// {name: "chambi", age: 20}
 console.log (obj);
 // {name: "chambi", age: 20}
```

* Deep Copy

* The copy in which original object is not connected with its copy is called as Deep Copy.

* Here, we create seperate empty object and after that we copy key-value pair of orginal object into new empty object.

* Now, if we make any changes in copy, it will not be reflected to original object because we have create seperate memory blocks.

Date

* Create copy using for loop

```
Let obj1 = {name: "chambi", age: 16}
let obj2 = { }
// new empty object
for (prop in obj1){
obj2[prop] = obj[prop]
 console.log (obj2)
}
// copy key-value into new object
obj2.age = 20;
 (console.log (obj2);
// {name: "chambi", age: 20}
 console.log (obj1);
  // {name: "chambi", age: 16}
```

Object In-Built Methods

1. Objects.key (obj_ref)

* Returns an array of given object's property names.

```
const obj = {a: 1, b: 2, c: 3};
console.log (Object.keys (obj));
// output: ["a", "b", "c"]
```

2. Object.value (obj_ref)

. returns an array of given object value.

```
console obj = {a: 1, b: 2, c: 3};
Console.log (Object.values (obj));
// output: [1,2,3]
```

Spiral

Date

③ Object.entries(objref)

* Return an array of key-value pairs in an array
* const obj = {a:1, b:2, c:3};
  console.log (Object.entries(obj));
  //output : [[a,1],[b,2],[c,3]]

④ Object.assign(target object, src, src2, ... srcn)

* copies key-value pair from one or more source objects to a target object.
* const target = {a:1, b:2};
  Const Source = {c:4, d:5};
  Const result = Object.assign(target, source);
  console.log(result);

  // output : {a:1, b:2, c:4, d:5}

Call apply Bind Introduction

① Call Apply and Bind method are used to store the object reference in 'this' keyword of function

* When function's 'this' have reference of object, then we can access states and behaviour of the object.
* For practice we will use these objects as reference.

```
let human={
    name = "chombi",
    age = 20,
    };
let human2 = {
    name : "Ninja",
    age : 19,
    };
let human3 = {
    name = "Nindi",
    age : 10,
    };
```

Below function we will use to access object's properties by using call, apply and bind methods.

```
Function details All(a,b,c){
    Console.log("Name :" + this.name);
    console.log("Age :" +this.age);
    console.log("Value of a :" +a);
    console.log("Value of b :"+b);
    console.log("Value of c :"+c);
    }
```

Call

1. Call method accepts object reference as first argument And accepts 'n' number of arguments
2. Here, arguments are passed to the function parameters 1st
3. It will call the function Immediately.
4. Example: Print name, age of object human1 and print function arguments

```
details All.call(human, 10, 20, 30);
```

output-

Name: Chombi
Age: 20
Value of a : 10
Value b : 20
Value c : 30

Date

Apply

1. Apply method accepts of 2 argument where referred is first argument and 2nd argument is the array of arguments
2. Here arguments are passed to the function parameter list.
3. It will call the function immediately.
4. Example: Print name, age of object human2 and print function argument.

```
details All. apply (human2, [11,22,33]);
output -
Name : Divya
Age : 19
Value of a : 11
Value of b : 22
Value of c : 33
```

Bind

1. Bind method accepts object reference as 1st argument and excepts 'n' number of arguments

(1) Here 'n' number of arguments are passed to the function parameter list.

(2) It will not call the function immediately
It returns a new function in which "this" keyword is pointing to the object reference which we passed

Date

* To execute the function we need function reference and parenthesis.
* Example Print name, age of object human3 and print function arguments.

```
Let Func = details All. bind (human3, 77, 88, 99);
Func ();
output -
Name : Nimbi
Age : 18
Value of a : 77
Value of b : 88
Value of c : 99.
```

Constructor function

1. A function which is used to create an object is known as constructor function.
* A constructor function behaves like blueprint or template for object and there is no need to write code again and again
* It helps us to create multiple object same type.
* Syntax : function identifier (parameter, ...) { }
* If the function is designed to use as a constructor then name of function should be upper camel case.
* The list of parameter provided to the function will be treated as key / properties of the object.

Date _____

* The argument pass when function call be value of object
* We can copy the values into the keys of the object from parameter using this keyword.
* We can create a object using the constructor function with the help of new keyword.
* To create constructor function we call not use arrow function because they does not have 'this' keyword.
* Syntax let variable = new function_name (arguments)

Example

```
function Car (model, color, engine){
this.model = model;
this.color = color;
this.engine = engine;
}
let car1 = new Car (1021, "red", "V8");
console.log (car1);
// {model: 1021, color: "red", engine: "V8"}
```

This Keyword

1. It is a keyword.
2. It is a variable which holds the reference.
3. In GEC it holds the address of window object.

Date _____

4. It is local variable of every function in JS, and holds the address of window object. Except in Arrow function (for arrow function is stores undefined)
5. Inside object methods, 'this' holds the reference of current object (not in arrow function).

Destructuring

1. The process of extracting the values from the array or object into the variables is known as destructuring
2. The two most used data structure in Javascript are object and array both allows us to unpack individual values into variables

Object destructuring

1. The process of extracting the values from the object into the variable is known as object destructuring.
2. All the key name provided on LHS are consider as variable and these variables should be declared and written inside curly braces

3. The variable name should same as object key name.
4. JS engine will search for the key inside the object
5. If the key is present, the value is extracted and copy into variable.
6. If the key is not present, undefined is store in the variable
7. After destructuring, we can directly access variable names without using object reference

Date

8 Example

```
let obj = {
name: "chombi",
age: 16,
}
let {name, age, country} = obj;
console.log (name);
//chombi
console.log(country);
// undefined
```

~~Array Destru~~ Here, we are typing extract name, age and country from obj but country is not, so inside country js engine stored undefined and for name, and age we have respective values.

Array Destructuring

1. The process of extracting the values from the array into the variables is known as array destructuring.

2. All the key names provided on LHS are consider as variable and should be written inside square brackets.

3. JS engine will extract the array values and stored them variables in the same order as they are present inside array.

④ If we try to access value which is not present inside array, js engine will store undefined inside that variable.

Date

Example

```
Let arr = [10,20,30,40,50]
let [ab,c,de,F] = arr;
console.log(a);
//10
console.log(b)
//20
console.log(c)
//30
console.log(d)
//40
console.log(e)
//50
console.log(e);
//50
console.log(F);
// undefined
```

Here, we are trying to extract value from array into variables ab,cd,ef. As we are already learnt values will be extracted and stored into variable in the same sequence they are available inside array, so we have value inside a,b,c,d,e but not inside because at sixth position no value is present and JS engine stored undefined in it.

Destructuring in function

1. We can destructuring array or object in function parameter so that we can access value directly.

② Destructuring object in function parameter.

At the time of object destructuring, we have to make sure variable name is same as object key name and write within curly braces.

Date

```
function details ({name, age}){
  console.log (name);
   // Chombi
  Console.log (age);
  // 16
  }
  let obj ={
   name: "chombi",
   age:16,
   }
   details (obj) // function call
```

Here, we have passed object as an argument to details function, and we have destructured values in parameter only.

② Destructuring array in function parameter.
At the time of array destructuring, we have to keep variable between square brackets. Values will be destructured in the same order, they are available in array.

```
function details ([a,b,c,d,e]) {
   console.log (a);
   // 10
   console.log (b);
      //20
   Console.log (c);
   // 30
   Console.log (d);
   // 40
   Console.log (e);
   //50
        $ let arr = [10, 20, 30, 40, 50];
   details (arr) // function call
```

Here, we have passed array as an argument to details function, and we have destructured values in parameter only.

Rest And Spread

1. Rest parameter

* Rest parameter is used to accept multiple value, stored them in an array and array is reference will stored the variable that we have used for rest.

* Rest can accept n number of values and stored them in an array. *

* To make a variable rest, we have to put '...' before variable name.

* Syntax
  let ... variable_name

* We can use rest in function when we don't know the exact number arguments.

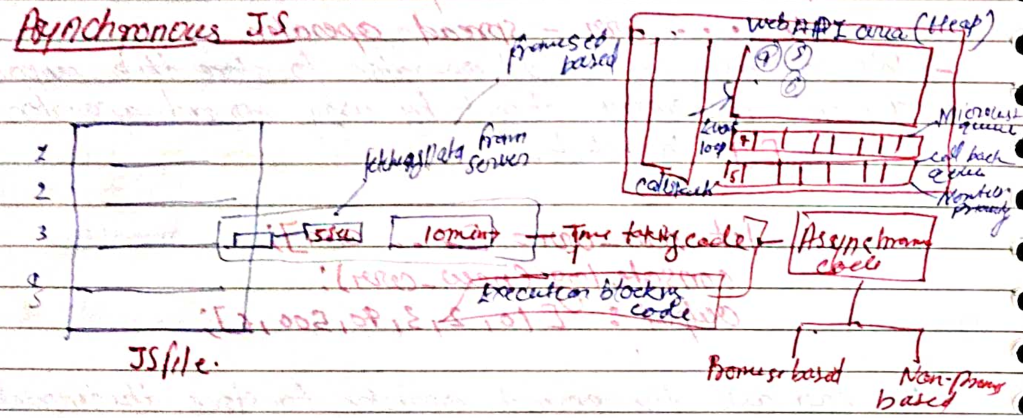
* In function, there can be only one rest parameter and it should be the last parameter.

```
* Function details (a, b, ... z) { console.log (a);
      //10
   console.log (b);
      //20
   console.log (z[0]);
      // 30
```

Date

* If we do not pass literals to three dots(...), it will unpack all literals and behave as a spread parameter.

Asynchronous JS



Asynchronous programming

AJAX
Asynchoronous Javascript and XML → Dynamical UI render
└ Resource fetch

Non-Promise Based API
They interact with browser click or key is
├ set timeout
└ set time interval

Date

Set timeout →
* Syntax: Set timeout(callback, time);
* It will execute callback when time expired.
* It return unique Id. by this Id we can Identify individual SetTimeout.
• Here, callback will execute only once.
Clear Timeout (setTimeout_id).
↳ It will execute stop execute of that individual SetTimeout

Set Interval():-
Syntax SetInterval(callback, time);
* It will execute callback after given interval of time
* It return unique Id
* Here callback will execute continously after given interval of time, untill it is stopped by clear Id.
* Clear Timeout
Clear Interval out ("SetIntervalout_Id");

Promise
It represent Eventual completion or failure of asynchronous instruction and its resulting
* Promise is Proxy.
* Promise has three states.
├ Pending → Not settled
├ fulfilled → data → then()
└ rejected → Reason of rejection → catch()
settled → finally()
State handlers method

Date

Async and Await

* **Asyn** → It is used to convert synchronous function to asynchronous function.
  * Asyn is a key word
  * Now function will always return a promise

[# We cannot use await without async
We can use async without await.

* **Await** →
  Await, it used to execute asyn-asynchronous code line by line (synchronous)
  → Await (key word) instructs the JS to wait for complete execution of instruction, where await keyword used

Asynchronous →

* The code or instruction that can blocks the execution are stuffbirud as Asynchoronous code.

* In javascript, we have time taking instruction that may blocks the execution, If not handled correctly.

* Asynchronous programming technique where we handle asynchronous code without blocking name normal execution of program.

* Asynchronous Javascript is commonly used when dealing with time-consuming tak tasks such as network requests, file operations, or accessing remote resources. Instead of blocking the entire program until the task is finished, asynchronous operation are initiated and the program continues executing other tasks. When an asynchoronous operation completes a callback function or a promise is triggered to handle the result or perform additional actions.

Date

SetTimeout

* SetTimeout is part of the Web APIs provided by web browsers. It is a method that allows you to schedule the execution of a function or code snippet after a specified delay. The setTimeout function is not a part of the Javascript language itself but is made available by the browser's runtime environment through its API

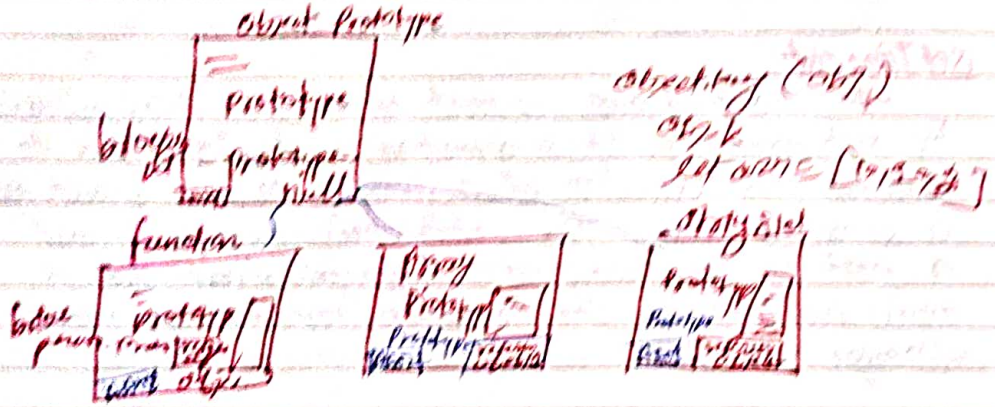
* The clearTimeout function is used to cancel a Timeout that was set using setTimeout. It takes the timer ID returned by setTimeout as an argument. When clearTimeout is called with the timer ID, It cancels the scheduled timeout, preventing the associated function or code snippet from executing.

SetInterval

* The setInterval is part of the WebAPI is provided by browser. It allows you to schedule the repeated execution of a function or code snipped at a specified interval. Similarly, the clearInterval function is used to cancel or stop the execution of an Interval that was set using setInterval.

Dom

* Prototype



Str. split(" ")
Str. Join( )

String — method
— toUpperCase( )
— toLowerCase( )
— IndexOf (substring, pos)
— Includes (substring, position)
— charAt ( )
→ CharCodeAt ( ) — e

slice (a, b) — starting point (is not included)
positive values are accepted
ending point (not included)

substring (a, b)
substring index — ending index (not included).

Prototype

1. In JavaScript, every function is associated with an object called as prototype.
2. It serves as a blueprint or a template from which other objects can inherit properties and methods.
3. Prototype are used to achieve inheritance in JavaScript.
4. When you access a property or method on an object, JavaScript first looks for that property or method directly on the object itself. If it doesn't find it there then it looks at the object's prototype, and continues up the prototype chain until it either finds the property/method or reaches the end of the chain (where the prototype is null).
5. This allows us to define common properties and methods in a prototype and all objects that inherit from that prototype will have access to those properties and methods.

__proto__

1. The reference of prototype object is stored in __proto__
2. When an object is created a prototype object is not created instead the object will have reference of the prototype object (from which properties are to be inherited) referred using __proto__.

Prototypal Inheritance.

1. Prototypal Inheritance is a fundamental concept in JavaScript's object-oriented programming model.

Date

2. It allows objects to inherit properties and methods from other objects, form a prototype chain.

3. In JavaScript inheritance is achieved using prototype hence it is known as prototypal Inheritance.

Prototype chain

1. The prototype chain is a mechanism in JavaScript that allows objects to inherit properties and methods from other objects.

2. The prototype chain forms a hierarchy of objects to inherit proto properties and methods & where each objects prototype is linked its parent object is prototype, creating a chain of inheritance.

3. By following this chain, objects can inherit properties and methods from their prototype objects.

DOM

1. The Dom Object Model (DOM) is a programming interface for web documents that represents the HTML or XML document as a the tree structure, where each node represents an element, attribute, or piece of text in the document.

2. When a webpage is loaded, the browser creates a DOM tree that represents the documents structure and content.

3. Each node in the tree is represented as JS object, which we can access and manipulate using the DOM API

Date

4. Here, HTML element comments, text, content etc are reffered as node of DOM tree.

DOM API.

1. The DOM API (Application Programming Interface) is a set of programming interface and methods that allows developers to interact with DOM tree and manipulate the content and structure of web documents.
2. The DOM API provides a standardized way to create, modify and delete elements and attributes, change styles and classes handle events, and more.

HTML Structure

1. Reference HTML Structure.

```
<body>
    <h1>Falling In love with Javascript</h1>
    <div class = "container">
    <div class = "item item1" id = "itemone">1</div>
    <div class = "item item2">2</div>
    <div class = "item item3">3</div>
    <div id = "itemFour" class = "item item4">4</div>
    <div class = "item item5">5</div>
    </div>
    <p>hello I'm paragraph</p>
```

Target Elements

1. getElementsById('id_name')
   - It return reference of single element object where id name matches
   - Example

```
let divone = document.getElementById("itemone");
console.log(divone);
```

Spiral

Date

2. getElementsByClassName('class_name')
* It return htmlcollection all elements matches with class name.
* Example: Apply background color, margin, fontsize and padding on each div.

```
let div_child = document.getElementsByClassName("item");
console.log(div_child)

div_child[0].style.backgroundColor = "red";
for(ref of div_child){
    ref.style.backgroundColor = "blue";
    ref.style.margin = "10px"
    ref.style.fontSize = "52px"
    ref.style.padding = "10px"
    ref.style.color = "white";
}
```

3. getElementsByTagName("Tag_name")
* It returns htmlcollection all elements matches with tag name.
* Example: Display parent div as flex box

```
let divs = document.getElementsByTagName("div");
console.log(divs)
divs[0].style.backgroundColor = "yellow";
divs[0].style.padding = "10px";
divs[0].style.display = "flex";
divs[0].style.gap = "10px";
divs[0].style.justifyContent = "space-between";
for(let i=1; i<divs.length; i++){
    divs[i].style.backgroundColor = "blue";
    divs[i].style.padding = "10px";
    divs[i].style.color = "white";
}
```

Date

4. querySelector('css-selector')
* It returns reference of the first element that matches a specified css selector.
* Example: Change fontSize of first div with "item" class

```
let ele = document.querySelector(".item");
ele.style.fontSize = "52px";
console.log(ele);
```

5. querySelectorAll("css-selector")
* It returns Nodelist of all elements that matches a specified css selector.
* Example: Change fontWeight all div with 'item' class.

```
let eles = document.querySelectorAll(".item");
for(ref of eles){
    ref.style.fontWeight = "bold";
}
console.log(ele);
```

Create And Insert Element

1. createElement("tag_name")
* It is used to create a new html element of the specified type and return a reference to it as a javascript object
* Example: Creat section tag

```
let sec = document.createElement("selector");
console.log(sec);
```

2. appendChild(element)
* It is used to insert the element as last child.
* Example: Insert the section tag inside div tag having class container

```
let sec = document.createElement("section");
let pdiv = document.getElementsByClassName("container")[0];
pdiv.appendChild(sec)
```

Date

3. insertAdjacentElement("position", element)
- It is used to insert an element as a child or sibling.
- Positions: beforebegin, afterbegin, beforend, afterend.
- Example: Show how to display element as child and sibling of div having class 'container'

```
pdiv.insetAdjacentElement("beforebegin", sec);
pdiv. insertAdjacentElement("afterend", sec);
pdiv. insertAdjacentElement("afterbegin, sec);
pdiv. insertAdjacentElement("beforeend", sec);
```

Insert Text And Elements

1. textContent
- It is used to insert text inside element.
- Example: Insert "Hello" text inside p tag

```
let p = document.getElementsByTagName("p")[0];
p.textContent = "Hello";
```

Example: Insert "Hello" text inside p tag and preserve previous text also.

```
let p = document.getElementsByTagName("p")[0];
p.textContent += "Hello";
```

2. innerHTML
- It is used to insert text and html tag inside element.

Example: Insert " Hello" inside p tag.

```
let p = document.getElementsByTagName("p")[0];
p.innerHTML = '<strong> Hello</strong>';
```

Example: Insert " Hello" inside p tag and preserve previous text and element also.

```
let p = document.getElementsByTagName("p")[0];
p.innerHTML += "<strong> Hello</strong>";
```

Date

Insert And Remove Attribute

1. setAttribute('attribute_name', 'value')
- It is used to insert the attribute to an element.

Example: Insert id="chumbi" to third div in the container

```
let divs = document.getElementsByClassName("items");
divs[2].setAttribute("id", "chumbi");
```

2. removeAttribute(attribute_name)
- It is used to remove attribute from an element.
- Example: Remove id attribute from third div of container

```
let divs = document.getElementsByClassName("item");
divs[2].removeAttribute("id");
```

Traverse HTML Nodes

1. parentElement
- It returns the reference of parent html element.
- Example: Print parent element of div whose class is "item"

2. nextElementSibling
- It returns the reference of next html node sibling.

Example: Print next sibling element of third div whose class is 'item'

```
let div-child = document.getElementByClassName("item");
Console.log(div-child[2].nextElementSibling);
```

3. previousElementSibling
- It returns the reference of next html node sibling.
- Example: Print previous sibling element of third div whose class is "item".

```
let div-child = document.getElementByClassName("item");
Console.log(pdiv-chil div-child[2].previousElementSibling);
```

Date _____

4. Children

* It returns HTMLCollection of html all childs element.

Example: Print children elements of div whose class is "container"

```
let pdiv = document.getElementsByClassName("container")[0];
console.log(pdiv.children);
```

5. childNodes

* It returns NodeList of all types of nodes like string, text, comment etc.
* Example. Print all child nodes of div whose class is "container"

```
let pdiv = document.getElementsByClassName("container")[0];
console.log(pdiv.childNodes);
```

Remove HTML Element

1. remove()

* It is used to delete html element
* Example Remove p element from Para.

```
let p = document.getElementsByTagName("p")[0];
p.remove();
```

* cloneNode(true)
* replaceChild(e, value, position)
* getAttribute()
* hasAttribute()

Inline style change with CSS

```
div.style.cssText = "background:red ; color:white";
```

Date _____

* Event

Event are action carried out by user on their browser. When such events occur, an object is created by the browser which carries relevant data about the event and the object on which it occured. For instance if a user clicks on an <h1> tag, an object would be automatically created which carries information about the tag and the type of event that took place (in this case, it would be a "click" event).

* The Event object

Thonfore is a container created by the browser which holds crucial data about an event, including the event's type and the element on which the event occurred.

Event listeners

Event listeners in Javascript are functions or code snippets that are attached to specific events on HTML elements. They allow developers to define how the program should respond or behave when a particular event occurs.

Ways to attach event listeners

* HTML ATTRIBUTE
  * You can also attach an event listener by adding an inline event handler as an HTML attribute. This approach is less common and not recommended for larger projects as it mixes HTML markup with Javascript logic.
* JS Property
  * HTML elements have properties that begin with "on" followed by the event name. These attributes provide direct assignment of a listener function but using this method, each event type on an element can only have one listener.

Date

- addEventListener/removeEventListener:-

+ The addEventListener() method is the preferred way to attach event listeners. It allows you to attach multiple event listeners to an element, and it provides more flexibility for even handling. You can use to attach a listener to an element for a specific event type.
* The removeEventListener() method is used to detach or remove an event listener that was previously attached to an HTML element using the addEventListener() method.

Events name

mouseover: fires when the mouse pointer enters an element
mouseout: fires when the mouse pointer leaves an element
mousemove: fires when the mouse pointer is moved over an element
click: fires when a mouse button is clicked on an element
submit: fires when a form is submitted.
input: fires when the value of an input field changes
change: fires when the value of an input field changes and losses focus
keydown: fires when a key is pressed down
keyup: fires when a key is released
keypress: fires when a key is pressed down and then released
DOM ContentLoad
load
focus
blur
Scroll (event / deltaX, deltaY)
Wheel

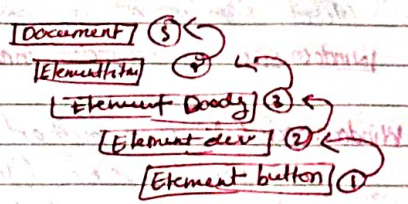
Event bubbling

In the event bubbling model, an event starts at the most specific element and then flows upward toward the least specific element (the document or even window).

Date

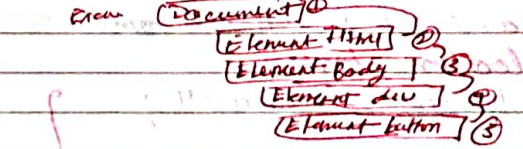
Example When you click the button, the click event occurs in the following order.

- Button
- div with the id container
- body
- html
- document



② Event capturing

In the event capturing model an event starts at the least specific element and flows downward toward the most specific element.
When you click the button, the click event occurs in the following order



- addEventListener (event, function, useCapture):
- The default value is false, which will use the bubbling propagation. When the value is set to true, the event uses the capturing propagation.

If user wants to stop propagation then pass the parameter in a function and use the method stopPropagation.

Example

```
let b = document.getElementById("btn")
b.addEventListener ("click", btnclick)
function btnclick (event){
    console.log ("button clicked..");
    event.stopPropagation()//
```

preventDefault It is a change default behaviours

Eg:

```
b.addEventListener ("click", btnclick)
function btnclick (event){
    event.preventDefault();
```

BOM Browser Object Model is used to interact with the browser.

Window Object is main object in BOM, The window object represents a window in browser.

Spiral

An Object of window is created automatically by the Browser
* All global Javascript objects, functions, and variables with the var keyword automatically become member of the window object

* Global variable are properties of the window object
Global function are methods of the window object

Window size — window.innerWidth window.outerWidth window.innerHeight window.innerHeight

Window.open () is a method open a new tab.
window.open (url, name, feature)
feature is set the style of New tab of popup
.close() is close the new tab.

Location Object properties.
Location.href :- It is provide href detail.
location.pathname. It is provide path name

Method of location object
Function { window.location = "url";
}
Location.href = " url "
Location.assign("url") → It is add the history
location.replace ("url") → It is not add history
location.reload ( )

(redirect the other site.)

Navigator properties
navigator.appName() is provide Appname of the Browser
Navigator.appVersion is provide version of the Browser
Navigator.appCodeName is provide code name of the Browser
navigator.cookieEnabled() is provide cookies Enabled or not
Navigator.userAgent() is provide user Agent of the Browser
Navigator.platform is provide platform name of the Browser

Date

function

① Import and export
If we want to use another file code in other file then we are use import and export key word

hi.JS
Let a = {
name : "Saurabh"
}
export default a

hello JS
export let b = a => hello

Jam.JS
import {a} from './hi.JS'
pathname

[Index.html]
<html>
<body>

<script type="module" src="Jam.JS"></script>

Type="module" is mandatory in script tag.

Spiral