Day 1

Tuesday, December 21, 2021 10:18 AM

1. What is JS why do we need JS?
2. What is dynamic page?

- ECMA Script is nothing but JS. European Computer Manufacturer Association
- JS will help to provide behaviour or functionality to a web page.
- Webpage becomes dynamic Because of JS.
- Webpage performs actions without loading the webpage.
- JS is a programming langanguage which the browser can understand and makes the webpage dynamic.
- Dynamic webpages are changes the happen on the webpage after the webpage has been loaded without refreshing the page.
- JS was made by Netscape. But ECMA is a standard created for JS.
- ---
- JS is a high level language which gets executed on the browser.
- Browser is a application which behaves like client.
- Browser itself acts like a compiler or interpreter.
- Browser gives an environment for JS to run.
    - Browser has a sub application called as JS engine.
    - JS engine is responsible to translate JS instructions into machine understandble language within the browser.
    - We need JS engine to execute JS.
- Browser and its sub applications
    - JS Engine
    - Timer
    - Console
    - Datatypes
- ---
- Notes: (Sir)
    - JS is a high level language.
    - To execute JS instructions it is mandatory to translate JS instruction into machine understandable language.
    - This job is done by JS Engine.
    - Every browser will have a JS Engine to run the JS code. Therefore browser becomes an environment to run JavaScript.
Chrome	V8 (fastest and most efficient according to industry)
Firefox	SpiderMonkey
JavaScriptCore	Safari
IE or Edge	Chakra
- Can we run JS outside a browser?
    - Yes we can run JS outside the browser with the help of Node.
    - Node
        - Node is a bundle of Google's V8 Engine and built in methods using C++.
        - It serves as an environment to run JavaScript outside the browser.
        - This invention helped JavaScript to gain its popularity in useage as a back end language.
        - Node is an environment to execute JS without the help of a browser.
- Variables, operators, blocks and functions fundamentals of any programming language.
- Variable is a named block of memory to store some data.
- Operator is a sybmol which has some predefined task to be performed on a data.
- Block is a set of instructions which gets executed when you call them.
- <u>Characteristics of JS</u>
    - Purely object oriented.
    - Interpreted Language.
    - JS uses Just in time compiler(combination of compiler and interpreter)
Compiled	Interpreted
Checks entire code if correct. It generates an executable file	Line by line checks syntax and then execution
    - JS is synchronous in nature(Single threaded architecture).
- Object is a block of memory which has states (variable) and behaviours (functions).
- . Is a access operator or member operator. It will help you to use the variables and functions present inside an object.
Console	.	log()
Object	Operator	Function of console object

  Log() is an function which can accepts argument (argument is an data passed to a function).
- First code

```
<html>
    <head>
        <title>Document</title>
    </head>
```

```
        <body>
            <h1>Program 2</h1>
            <script>
                console.log("hello world");
            </script>
        </body>
    </html>
```

- Use <script src="path"> </script> to include all JS files.

TO EXECUTE JS On Browser

1. We can execute JS instructions directly in the console provided by the browser.
2. We can execute JS Instructions by embedding it in html page.
   a. Inline
   b. External

Inline:

With the help of script tag we can embedd JS instruction syntax

Html code……

```
<script>
        Js code……..
</script>
```

Html code…..

Eg:	<html> <head> <title>Document</title> </head> <body> Program 2 <script> console.log("hello world"); </script> </body> </html>

External:

1. We can create a separate file for JS with extension .js
2. Link the JS file with html file using the src attribute of script tag.

Syntax:

Html code……

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

Html code…..

Task: List all the keywords of JS!

Token: smallest unit of any programming language is known as Tokens.

1.keywords: A predefined reserved word which understandable by your JS engine is known as keywords.

Note:

- every keyword must be in lower case
- A keyword cannot be used as an identifier.
- Eg: keyword
  - If, for, let, null, null

2.identifiers: The name given to the components of JS like variables, functions, class, etc are known as identifers.

Rules for identifers:

a. An identifier cannot start with a number.
b. Except $ and _ no other special character is allowed.
c. We cannot use keywords as identifers.

3.literals/ values:The Data which is used in the javascript program is called as Values or Literals.

Datatypes	Values
number	Any mathematical number
String	Anything enclosed in ' ', " ". ` `
Boolean	False, true

Null	
Undefined	It is a keyword and a value.
object	
BigInt	Suffix the number with n.
Symbol	

According to JavaScript all the types except object is considered as Primitive types
Object is considered as non primitive type.

Note:

- The numbers between the range -(2^53 -1) and (2^53 -1) is considered as number type.
- We can use BigInt type to store numbers beyond the given range. By suffixing the number n.
    - Eg: 1 ----------> number
        - 1n----------> BigInt

Typeof operator:

1. It is a keyword used as an unary operator, to identify the type of data.
2. Syntax: typeof value
3. Examples

console.log(typeof "done with file structure");	//string
console.log(typeof `done with file structure`);	//string
console.log(typeof 'done with file structure');	//string
console.log(typeof 47);	//number
console.log(typeof true);	//boolean
console.log(typeof false);	//boolean
console.log(typeof null);	//object
console.log(typeof undefined);	//undefined
console.log(typeof 1n);	//BigInt

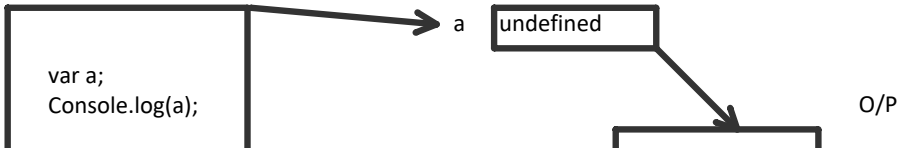
4. typeof null is never null it is considered as object | V.Imp

Variables:

1. A named block of memory which is used to store a value is know as Variable (container to store data).
2. Note:
    a. In JS variables are not strictly typed it dynamically typed.Therefore it is not necessary to specific type of data during variable declaration.
3. In a variable we can store any type of value.
4. It is mandatory to declare avariable before using.
5. Syntax for varaible declaration:
   Var/let/const identifer;
   var identifier;
   let identifer;
   Const identifier;
6.

var a;	Declaration statement
a=10;	Initialization/ assignment

7. Examples:
    a. let b;
       b=20;
       console.log (b) ;
    b. const c=30;
       console.log(c);
8. Note:
   When a variable is declared in Javascript, JS Engine implictly assigns undefined value to it.
   Example:



Q)Write a JS code to store your following details.
Name, age, gender, degree aggregate and display them on the console.

Ans)

```
var name=`bhuvan`;
var age=23;
var agg=7.0;
console.log( `My name is ${name}, I'm ${age} years old and my aggregate is ${agg}`);
```

Understanding Execution in JS

1. Everytime when JS Engine runs a JS code, it will first create a global execution context.
   Global Execution Context

Variable area	Function area/ Execution area

   The Global Execution Context has two parts.
   a. Variable area
   b. Functional area or execution area
2. JS Engine generally uses two phases to execute a JavaScript code
   a. 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.
   b. Phase 2: All the instructions get executed in the top to bottom order in Execution area of Global Execution Context.

   (JS Engine goes through the code twice)

   Tracing Execution

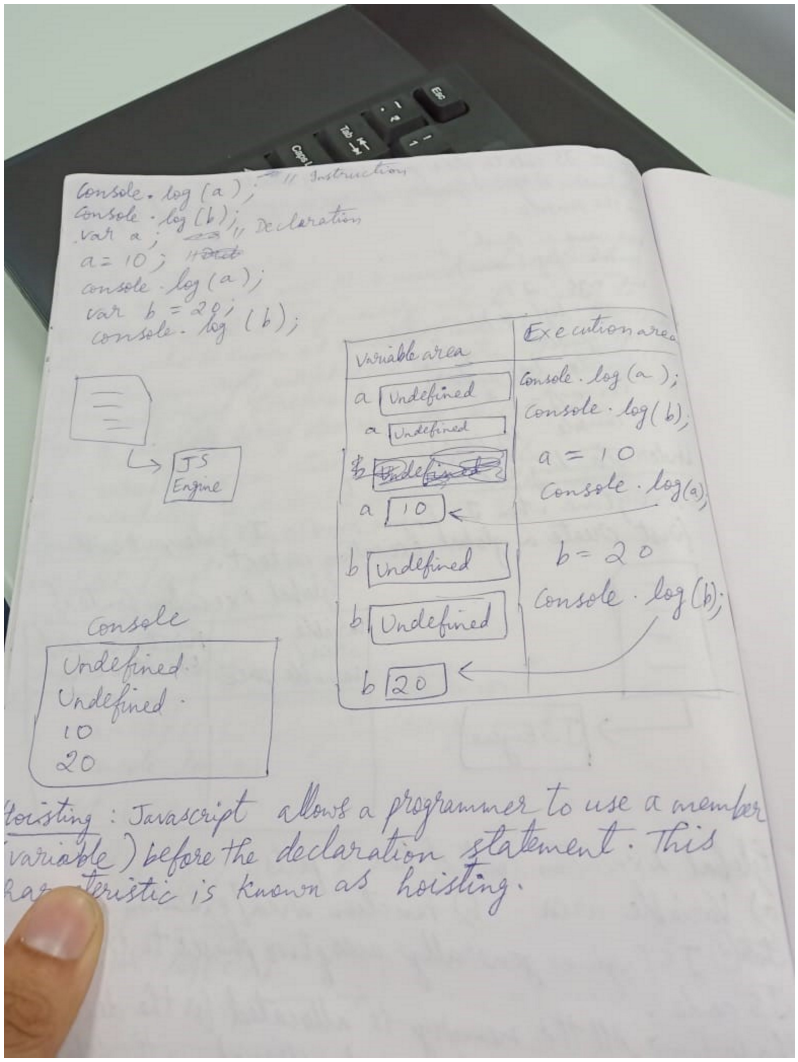
1	console.log(a);
2	console.log(b);
3	var a;
4	a=10;
5	console.log(a);
6	var b=20;
7	console.log(b);

   Step 1) Global Execution Context is created

Variable area	Function area/ Execution area

   Step 2)All the declarations are allocated with undefined. Assignment is not taken into consideration as it comes in Execution.

Variable area	Function area/ Execution area



Hoisting: JavaScript allows a programmer to use a member (variable) before the declaration statement. This charactersitic is known as Hoisting.

Script tag without src always should be within the body at the end`.
Script tag with src your code add at the end of the html body.
Script tag with src third party code at beginning in the head tag.

Day 2

Wednesday, December 22, 2021 10:39 AM

Strings:

In JS Strings can be represented using ' ', " " and ` `.

Note:

1. The start and end of a string must be done with the help of same quote.(' ", " '.` ", these are not allowed.)
2. If a text contains a single quote then it can be represented using " ".

Example:	text:	I'm a doctor.
	In js:	"I'm a doctor."

Example:	Text:	"JS" is awesome
	In js:	' "JS" is awesome'

For jumping to next lin in the console use \n (backslash n)

```
console.log("My name is: \n Bhuvan");
```

//with variables

```
var str="I love my country:\n INDIA";
console.log(str);
```

Backtick:

A Backtick can be a multi line string.

Eg:	//code snippet
	`var hobbies=`my hobbies:
	1.swimming
	2.cycling
	3.running
	4.gyming`;
	console.log(hobbies);
		//output
		my hobbies:
		1.swimming
		2.cycling
		3.running
		4.gyming

+ operator:

+ operator behaves like a concat operator if the operand is a string.

```
console.log("I'm "+name+" and I'm "+age+" years old");
```

Note:

The Strings represented using backtick is also known as Template String. The advantage of template string is we can subsitute an expression in a string using ${ expression }.

```
//using backtick or template string
console.log(`I'm ${name} and I'm ${age} years old`);
```

Question)What are the advantages of using backtick to represent a string over Single quote and double quote?

1.multi line

2.supports expressions

NOTE:

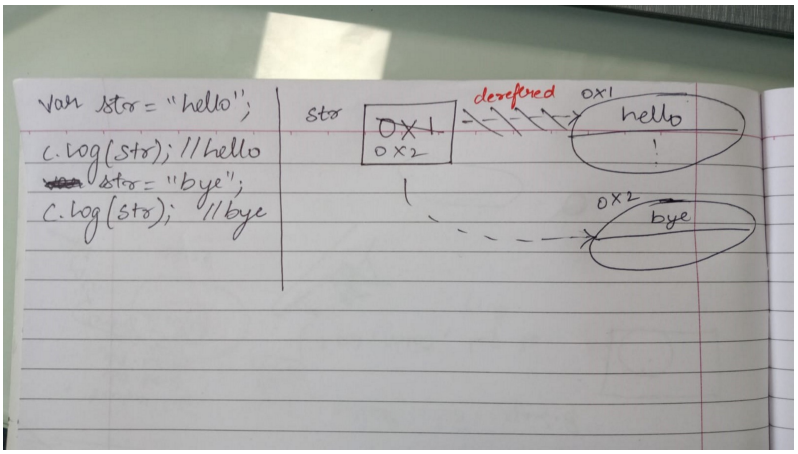
1. In JS, Strings is an object (it is an immuteable object) Therefore String objects will have state (variable) and behaviours(functions).
2. We can use the state and functions of a string obejct with the help of . Operator

to find length of a string:	We have a length variable which provides the length of a String.
	
	Eg:
	
	`var str="hello";`
	`var n=str.length;`
	`console.log(`str has ${n} characters`);`

Assignment1) List out 5-10 functions of strings.

Immutable Object: The object whose state once created cannot be modified is known as Immutable object.

Note: The variables in JS other than const is mutable in nature.



Program1) read a text from the browser using the prompt and display number of characters present in the text.

```
//code
var str = prompt();
var n = str.length;
Console.log(str);//bhuvan
console.log(`number of characters in text are ${n}`);


//output
bhuvan
number of characters in text are 6
```

Program2)read a text from the browser using a prompt and print the text back in console in upper case.

```
//code

var str = prompt();
console.log(str.toUpperCase());
```

Program3) read first name and last name from the browser using a prompt and display full name on the console by concatinating it.

```
//code

var str1 = prompt();//bhuvan
var str2 = prompt();//raj
console.log(str1 +" "+str2);

//output
bhuvanraj
```

Operators: Using which we can perform a specific task.

1. Airthmetic

+	addition
-	subtraction
*	multipilcation
/	division
%	modulus
++	Increment
--	Decrement

Eg:

```
var a=10;
var b=50;
console.log("Addition is",a+b);//60
console.log("subtraction is",a-b);//-40
console.log("multiplication is",a*b);//500
console.log("division is",b/a);//5
console.log("modulus is",a%b);//10
```

```
console.log("Increment:post" ,a++);//10
console.log("Increment:pre" ,++a,);//12
console.log("decrement:post", a--);//12
console.log("decrement:pre",--a);//10
```

2. Logical

&&	And
||	OR
!	NOT- returns true for false and false for true.

AND

T	T	T
F	T	F
T	F	F
F	F	F

OR

T	T	T
F	T	T
T	F	T
F	F	F

3. Assignment

=	Assignment operator
+=	Additional assignment
-=	Subtractal assignment
*=	Mutiplicational assignment
/=	Divisonal assignment
%=	Modulus assignment

4. Conditional

Ternary operator	(condition ? 'true' : 'false')

?	:	Ternary-works on three operands

5. Comparison

==	Equal to - it checks only the value
===	Strict equal to - it checks for both value and datatype
!=	Not equal to
!==	Strict not equal to
>	Greater than
<	Less than
>=	Greater than equal to
<=	Less than equal too

Var x=10;	Number
Var y='10';	String
x= =y	True
X = = =y	False
Eg2: a=100 b=100	(a===b) true
Eg3: a=50 b='50'	(a===b) false

6. Concatination

+	Concats when one operand as string

String Functions:

1. charAt()

   Returns a character at the specific index of the given string.
   Var str ="Hello";

console.log(str.charAt(1));//H

2. charCodeAt()
   It provides the Unicode value of a character present at the specified index.
   The default value of the index is 0. If the index value is not given or it can't be converted to a string the 0 is used by default.
   If the index value is out of range then this method returns NaN.
   It takes index as an argument.
   conststr ="Hello";// using charCodeAt() method
   console.log(str.charCodeAt(0));//72
   console.log(str.charCodeAt(1));//101
3. concat()
   It provides a combination of two or more strings.
4. endsWith()
   in javascript is used to determines whether the string ends with a specified substring or not.
   If it ends with a specified string then it returns true, else returns false.

   var question ="What is DOM?";
   console.log(question.endsWith("?"));
5. Includes()
   in javascript is used determines whether a given substring is present in the calling string or not.
   If the string is present then the method returns true, if not substring is not present then the method returns false.
   The matching of includes() method for the string is case-sensitive.

   constsentence ="Carbon emission is increasing Day by day.";// check if the string contains words
   console.log(sentence.includes("day"));// true
   console.log(sentence.includes("Day"));// true
   console.log(sentence.includes("DAY"));// false
6. Substring()

   ```
   substring(0,4)
   ```

   4 is exclusive

Difference between substring and slice functions of string object

Type Coercion/if

Thursday, December 23, 2021 9:59 AM

How to use CMD

cd/ - goes to beginning of drive
C: changes the drive
Cd - change directory
mkdir - to make a new folder
dir- to check whats present in the folder
mkdir filename1 filename2 to create two folders at once
use tab to get auto suggests

Note:

1. In JavaScript all non zero numbers are considered as true.
2. Number 0, null, NaN, ""(empty string), undefined all these values are considered as false, when they are converted to boolean.
3. A non empty string is considered as true.

Type Coercion

The process of converting one type of data into another type of data by JSEngine Implicitly when a wrong type of data is used in the expression is known as Type Coercion.

Eg:

```
console.log(5-'1');
```

```
The String is converted to number, result is 4
```

```
console.log(5+'1');
```

```
The number 5 is converted to string, result is 51
```

```
console.log(5-'a');
```

```
The String is converted to Number, the string does not have a valid numeric value. Hence it is converted to a special number NaN.
1. Any arithmetic operation with NaN ----> result is NaN.
2. Two NaN's are never considered as the same
```

```
console.log(NaN===NaN);//false
```

```
Note:
    Logical or ||
        Logical || behaves differently if the LHS value or RHS value is non boolean.
            Step1) It converts the LHS value to Boolean.
            Step2) If the converted LHS value is true then it returns the original value present in the LHS.

            Eg:

            console.log(10||20);//10

            Step2)If the converted LHS value is false then it returns the original value present in the RHS.

            console.log(0||20);//20

WAJS code to achieve the following requirement:-
Assume we have a container which keeps the count of items. If the count is zero display message empty. If the count is non-zero display the count.

Ans)
```

```
var count= 100;
console.log(count||'empty');//100
```

```
var count= 0;
console.log(count||'empty');//empty
```

```
Explicit Type Casting
```

The process of converting one type of a value to another type of a value is known as Explicit type casting.

1.Conversion of any type to Number

Syntax: Number(data to be converted)

String to Number

1. If a String is valid number we get the real number.

```
console.log(Number('123'));//123
```

2. If the String consists of any other character then we get NaN as the output.

```
console.log(Number('a'));//NaN
```

3. Boolean to Number

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

1)WAJSCode to log sum of two Numbers on the console by reading the numbers from the browsers prompt

2)WAJSCode to read source location, destination location, distance between source and destination from the browsers prompt assuming the cost per km is 8.5 display the approximate cost the user has to pay to reach destination from source on the console.

```
var destination = prompt("Enter Destination location");
var distance = Number(
  prompt("Enter the distance between the source and destinaton")
);
console.log(
  `The cost for travelling from ${source} to ${destination} with ${distance}km distance is ${
    8.5 * distance
  }Rs`
);
```

3)WAP to read age of the user from the prompt and display a message on the console wether the user is eligible to access adult content or not.

```
var age=Number(prompt('Enter you age'));
console.log(age>=18?"you can access":"you are a child go back");
```

4)WAP to accept total number of products bought and the total cost from the prompt and display cost of one product on the console.

```
var count_prod=Number(prompt('Enter number of products'));
var bill=Number(prompt('Enter the total bill amount'));
console.log(`cost of one product is ${bill/count_prod}Rs`);
```

5)WAP to read a name of a person using prompt and display whether the persons name is of even length or odd length on the console.

```
var name = prompt("enter your name");
console.log(name.length % 2 == 0 ? "even" : "odd");
```

6)Read name of a city from the user using prompt. Read a single digit lucky number from a user.Display the character present in that index postion on the console.

```
var city = prompt("Enter city name");
var index = Number(prompt("Enter lucky number"));
console.log(city.charAt(index));
```

7)Read a name of a fruit from prompt. Read a character from a user from the prompt.Check whether the character is present in the fruit name or not and suitable message on the console.

```
var fruit = prompt("enter fruit name");
var charac = prompt("enter character");
console.log(fruit.indexOf(charac) > -1 ? "present" : "not present");
console.log(fruit.includes(charac) ? "present" : "not present");
```

Decision Statements

Decision statements helps to skip a block of instructions when we don't have a favouring situation.

Eg: The instructions of loading home page should be skipped if the entered password is incorrect.

Decision Statements of JS

1. if

Syntax : if(condition) -------> any expression whose result is boolean

```
{
    statements;
}
```

2. if else

Syntax: if(condition)

```
{
    statements;
}
else
{
    statements;
}
```

Program 8) Read two numbers from the prompt and Find the largest number

```
var num1 = Number(prompt());
var num2 = Number(prompt());
if (num1 > num2) console.log(num1);
else console.log(num2);
```

3. else if ladder

At the end else block is not mandatory

Syntax: if(condition)

```
{
    statements;
}
else if
{
    statements;
}
else if
{
    statements;
}
else if
{
    statements;
}
.
.
.
else{
}
```

Eg:

```
var a=10;
var b=20;
if(++a)
{
    console.log(1);
}
else if(++b){// this part code is not even seen by the JS engine
    console.log(2);
}
else{// this part is also not seen
    console.log(3);
}
console.log(a);//11
console.log(b);//20
```

Program 9)Read four numbers using prompt and log minimum number in the console.

```
var n1 = Number(prompt());
var n2 = Number(prompt());
var n3 = Number(prompt());
var n4 = Number(prompt());
if (n1 < n2 && n1 < n3 && n1 < n4) {
  console.log(n1, " is the smallest");
} else if (n2 < n1 && n2 < n3 && n2 < n4) {
  console.log(n2, " is the smallest");
} else if (n3 < n1 && n3 < n2 && n3 < n4) {
  console.log(n3, " is the smallest");
```

```
} else {
  console.log(n1, " is the smallest");
}
```

4. Switch

Do not pass conditions in the arguments.
You can write default wherever you want inside the switch block.
Syntax:

```
Switch(value)
{
    Case value:
            {
                Statements;
            }
    Case value:
            {
                Statements;
            }
    Case value:
            {
                Statements;
            }
    default:{
    }
}
```

Note:
1. A case block gets executed if the value passed to switch matches with value present in case.
2. When a case is favourable, the case block gets executed as well as all the blocks present below in the switch gets executed.

Eg:

```
switch(1){
    case 1:{console.log('case 1');}
    default:{console.log('default');}
    case 2:{console.log('case 2');}
    case 3:{console.log('case 3');}
}
/*output
case 1
default
case 2
case 3
*/
```

3.We can have only one default inside a switch.
4.Default can be written anywhere in switch.

break

1. It is a control transfer statement.
2. It can be used either switch block or loop block only.
3. When a break statement is encountered the control is transferred outside the current switch or loop block.

Eg:

```
switch(6){
    case 1:{console.log('case 1');}
            break;
    default:{console.log('default');}
    case 2:{console.log('case 2');}
            break;
    case 3:{console.log('case 3');}
}
/*output
default
case 2
*/
```

Program 10) Design a JS code such that when a user enters any number between 1 and 7 the name of the week day must be displayed.Sunday starts from one. If the user enters any other number an invalid message to be displayed.

Loops

Friday, December 24, 2021 10:33 AM

Loops: it is also called as Iterations

The process of executing either a statement or a block of statement repeatedly multiple times is known as Loop or Looping.

Note:-> When we design a loop it is the responsibility of a programmer to break the loop after achieving the desired task. If not the program gets into infinte loop state.

Loop Statements:

1. While
2. Do-while
3. For
4. For-in, etc

While

No of minimum iterations is zero.

Syntax:	while(condition)
	{
	Statements to be repeated;
	}

Program 11)Write a program to print odd numbers between the range m and n, read the values the m and n from the browsers prompt.
Program 12)write a program to read name of a city from the prompt and print all the characters line by line present in even postion.
Program 13)Write a program to read a name from the prompt, generate a string which contains the reverse of the name and display it on the console.

```
var name='sandy';
name.split('').reverse().join('');
```

Program 14)Read a string and check whether it is palindrome or not.

Do while loop: The do while loop loops through a block of code once, then the condition is evaluated. If the condition is true, the statement is repeated as long as the specified condition is true.

No of minimum iterations is one.

Syntax:	do { statement; } while(condition);

Note:

1. The body of the loop is executed first then the condition is evaluated.
2. If the condition evaluated is true, body of the loop inside the do statement is executed again.
3. If the conditions evaluates to true, the statements inside the do statement is executed again.
4. This process continues until the condition evaluates to false. Then the loop stops.

Program 14)wap to print MERN stack 5 times within the console using dowhile loop.
Program 15)wap to display the numers from one to 10

Whats the difference a while loop and a do while loop?

For loop:

Syntax:	for(initialization; condition; updation)
	{
	//statements;
	}

Program 14) WAP to find sum of numbers from one to n.

```
var n = Number(prompt());
var sum = 0;
for (var count = 1; count <= n; count++) {
  sum = sum + count;
}
console.log(sum);
```

Program 15) WAJS program to find factorial of n.

Scope-Window object-Functions

Monday, December 27, 2021 10:21 AM

Understanding the Scope of Variables

Scope: The Visibility of a member to access it is known as scope.

Global Scope: It has the highest visibility can be accessed anywhere.

Block scope: the Visibility of a member is only within the block where it is declared is known as block scope, A member with block scope can be used only inside the block where it is declared it cannot be used outside.
Note: The variables declared with let and const have block scope. Therefore

Eg:	{
	let a=10;
	console.log(a);//can be used
	}
	console.log(a);//cannot be used
		{
		const a=10;
		console.log(a);//can be used
		}
		console.log(a);//cannot be used
			{
			var a=10;
			console.log(a);//can be used
			}
			console.log(a);//can be used

Differences between var, let and const

slno	Var	Let	Const
1	Var is global Scope.	Let is block Scope.	Const is also block scope.
2	We can declare multiple variables with the same name(the most recently created variable will be used)	We cannot declare two variables within the same name within a block.	We cannot declare two variables within the same name within a block.
3	The value of the variable can be modified.	The value of the variable can be modified.	The value of the variable cannot be modified.

4	Eg: var a=10;
	a=20;
	console.log(a);//20
		Eg: let a=10;
		a=20;
		console.log(a);//20
			Eg: const a=10;
			a=20;// Type error(Assignment to constant variable)
			console.log(a);

5	We can declare a variable without initialization.	We can declare a variable without initialization.	We cannot declare a variable without initialization.
6 (imp)	The variable declared using var belongs to global object (window). We can access them using window object.	The variable declared using let does not belong to global object. We cannot use them with the help of window.	The variable declared using const does not belong to global object. We cannot use them with the help of window.

7	The variable declared using var is hoisted, does not belong to temporal dead zone, can be used before initialization.
	Eg:
	
	`console.log(a);//undefined`
	`var a=10;`
		The variable declared using let is hoisted, the variable goes to temporal dead zone, therefore it cannot be used before initialization.
		Eg:
		
		`console.log(b); //ReferenceError: Cannot access 'b' before initialization`
		`let b = 10;`
			The variable declared using const is hoisted, the variable goes to temporal dead zone, therefore it cannot be used before initialization.

The jsEngine creates the window object using which we can access other global variables with var keyword.
Variables using let and const declaration are present in script scope.
Example:
var a=10;
let b=20;
Const c=30;
clg(window.a);//10
clg(window.b)//undefined
clg(window.c)//undefined

All var, let and const in javascript are always hoisted.

Temporal dead zone: the time between the variable which is created and initialization is known as temporal dead zone.
In temporal dead zone period we cannot use a variable.
The problem occurs only if the variable is declared before declaration.

Example:	`//before declaration`
	`console.log(b);//ReferenceError`
	`let b;`
	`b=10;`
		`//after declaration`
		`let a;`
		`console.log(a);//undefined`
		`a=10;`

Justification for let and var	`var age=24;`
	`for(var age=1; age<18;age++){`
	`}`
	`console.log(age);//18`
		`let age=24;`
		`for(let age=1; age<18;age++){`
		`}`
		`console.log(age);//24`

Global execution context->window variable which holds the address of global window object

Window Object

1. When a javaScript file is given to JSEngine on a browser by default a global window object is created and the reference is stored in the window variable.
2. The global object consists of predefined members(functions and variables) which belong to the browser window.
3. (IMP) Any members (variables or functions) created in the global scope will be added into the window object implicitly by JSEngine.
   Eg: var a=10;
   Variable a is added into the global window object hence we can also access it as----> window.a

FUNCTIONS

1. Function is a block of instructions which is used to perform a specific task.
2. A Function gets executed only when it is called.
3. The main advantage of a function is we can achieve code reuseability.

Note: In JS functions are beautiful, every function is nothing but an object.

Syntax to create a function

Generally we can create a function in two ways.

1. Using function declaration statement. (function statement)
2. Function Expression

Function Declaration Statement or Function Statement

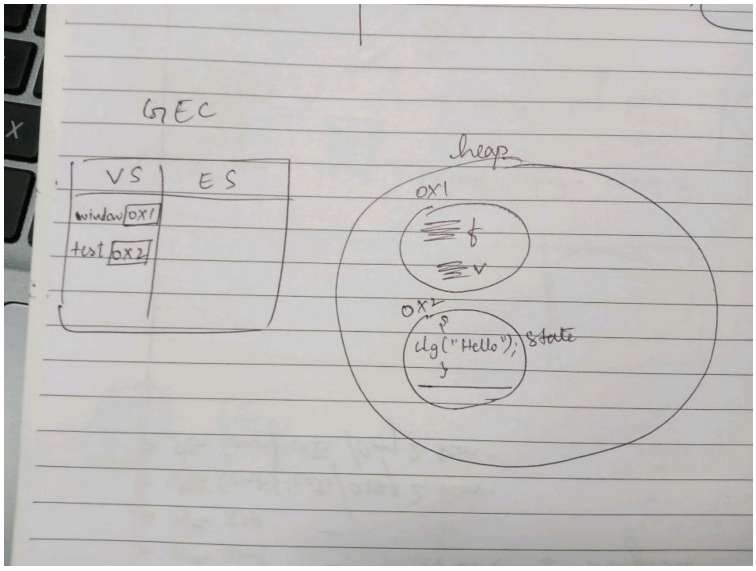
```
function  identifier([list_of_param,...})
{
      Statements;
}
```

Note:

1. Function is an object.
2. Name of a function is a variable, which holds the reference of Function object.
3. Creating a function using function statement supports function hoisting.
   Therefore we can also call a function before function declaration.
4. When we try to log a function name the entire function definiton is printed.

Example

```
console.log('start');
console.log(test);//function variable
function test(){
    console.log('hello');
}
console.log("stop");
```



To call a function of function call statement: function_name([argument_list,...])

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

Parameters:

- The variables declared in the function definition is known as Parameters.
- The parameters have local scope (they can be used only inside the function body).
- Parameters are used to hold the values past while calling the function.
- Example:
  - function sum(a,b){
    console.log(a+b);}

    a and b are variables local to the function sum

Arguments:

The values passed in the method called statement is known as statement.
An argument can be a literal, variable or an expression which results in a value.1
Eg: sum(10,20);
10, 20 are the literals used as argument.
Eg2: sum(-10+3,20)
-10+3 is an expression
Eg3: let a=10, b=30;
sum(a,b);
Here a and b are variables used as arguments.

Design a function which can accept radius of a circle and prints diameter on the console.

```
function dia(radi){
    return radi*2;
}
```

Design a function which can accept centimeters and print data in meters

```
function toMeter(cm) {
  return cm * 1000;
}
```

return( ):

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

```
function toMeter(cm) {
  return cm / 100;
}
var cms = 2546;
console.log(toMeter(cms));
var m = toMeter(cms);
console.log(m);
```

Design a function which can accept length and breadth of a rectangle and return the area of the rectangle.

Design a funciton which can accept a number and return its square.

```
function square(number) {
  return number * number;
}
console.log(
  `The squared number is${square(
    Number(prompt("enter number to find square"))
  )}`
);
```

Design a function which can accept m and n and returns power of m and n. m multiplied n times.

```
function pow(m,n){
    let prod=1;
    while(n>0){
        prod=prod*m;
        n--;
```

```
    }
    return prod;
}
let m=Number(prompt('enter m'));
let n=Number(prompt('enter n'));
console.log(`${m} to the power ${n} is ${pow(m,n)}`);
```

H.W.->Design a function which can accept a string of any length as the first argument, a string of length 1 as the second argument. The function must check whether the second string is present in the first string. If yes return its index position. If the second string is more than length one directly return -1. If the second string is not present in the first string return -2.
Note: without using the built in string function indexOf().

```
function occurence(word, letter) {
       if (letter.length > 1) {
         return -1;// to check second string length -1
       }
       for (let i = 0; i < word.length - 1; i++) {
         if (word.charAt(i) === letter) {
           return i;// get index position
         }
       }
        return -2;//not present
     }
     console.log(occurence(prompt("enter string"), prompt("enter letter")));
```

Function Expression

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

In this syntax the function is used as a value.
Disadvantage: The function is not hoisted we cannot use a function before declaration.
Reason:

```
a();//Error
var a=function(){
    console.log('fun');
}
```

In the above example the function is not hoisted instead the variable is hoisted and assigned with default value undefined.
Therefore the typeOf a is not a function it is undefined.

What is the difference between function statement and function expression explain with an example.

Local Scope-

Tuesday, December 28, 2021 11:09 AM

Assignment)Design a function which accepts a password and checks whether the password is valid or not.

A password is said to be valid if the following conditions are statisfied

1. Minimum length must be 8
2. Password must have atleast one special character.

Scope with respect to function

1. Any member declared inside a function will have local scope.
2. Local scope: The scope within the function block is known as local scope.
3. Any member with local scope cannot be used outside the funciton block.

Examples:

1. A parameter of a function will have local scope.

```
function test(a){
}
/* a is a variable whose scope is local to test.
it can be used only inside test function block.

*/
```

2. Variable declared with var keyword inside a function

```
function test(){
    var a;
}
/* variable a is local to test function
   it cannot be used outside.
*/
```

3. Function defined inside a function

```
function test(){

    function insideTest(){
    }
}
/*
    insideTest() function is local to test( ) function.
    insideTest() function cannot be called from outside test().
*/
```

Note:

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

Example:

```
var city="bangalore";
function display(name)
{
    Console.log(`${name} belongs to ${city}`);
}
display('sheela');
```

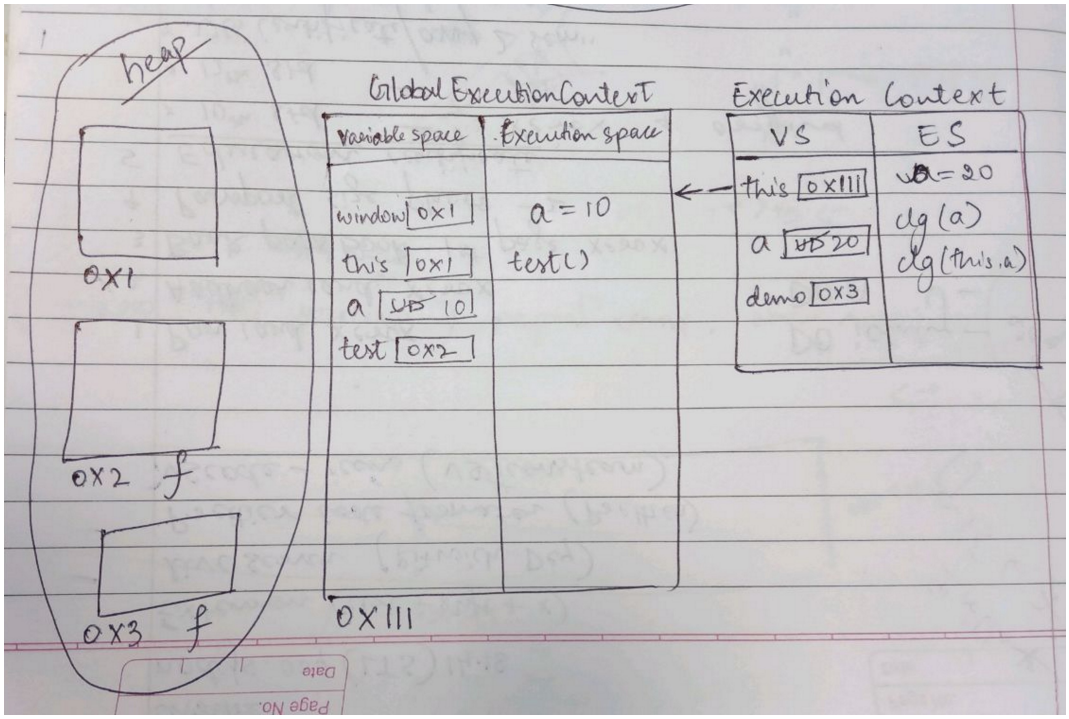
In the above example, variable name has local scope and variable city has global scope. It is observed that inside the function body we can use both the variables city and name.

this

1. It is a keyword used a variable.
2. It holds the address of global window object.
3. Therefore with the help of this variable we can use the members of global window object.
4. In JS, this is a property of every function. (every function will have this keyword).
5. Used to access the members of global execution context.

```
var a = 10;
function test() {
  var a = 20;
  function demo() {
    console.log("demo");
  }
  console.log(a); //20
  console.log(this.a); //10
}
test();
```

```
console.log("stop");
```



Call stack

Wednesday, December 29, 2021 10:16 AM

Call Stack (also called main stack or Execution stack)

Stack:

1. Stack is an organised block of memory where multiple objects can be stored.
2. Stack follows a specific order for entry and exit of objects.(LIFO : Last in first out or FILO: First in last out)
3. The object inserted at the first is removed at the last.

JS is synchronous in nature so it supports only one call stack.

Call Stack:

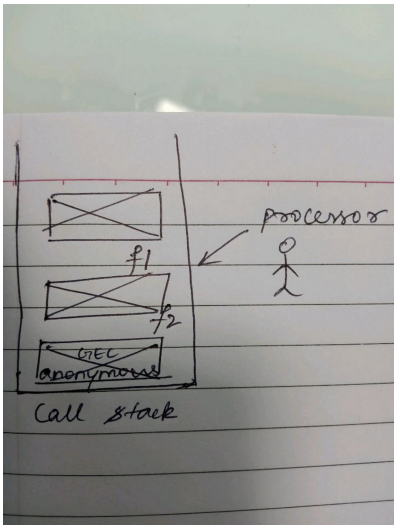
1. Call stack is used for execution for execution of instructions in JavaScript.
2. When the stack is empty processor is idle. JS never lets the call stack to be empty or idle the processor by deleting the call stack enhancing performance.
3. Call stack remembers the caller of the function

Internal working of call stack with code snippet
At any given time only one instruction can be executed in JS, it not possible to execute more than one instructions in JS.

```
console.log("start");
function f1() {
  console.log("f1");
}
console.log("GEC");
function f2() {
  console.log("f2");
  f1();
  console.log("F2");
}
f2();
console.log("GEC");
console.log("Stop");
```

Console

Start
GEC
f2
f1
f2
GEC
Stop



Execution starts with creation of Global Execution Context which is anonymous, now the Execution Current execution context (GEC named Anonymous) is paused as f2() is called by function call and the execution context of f2 is created. The statements inside f2 function is being executed, here when f1() function call is attended by pausing the current execution context f2 and new execution context for the called function f1 is created. Later, once every instruction in f1 is executed, the f1 execution context is removed from the call stack and the previously paused f2 is resumed where it left off. This process continues untill all the execution context are completed and the call stack is made empty.

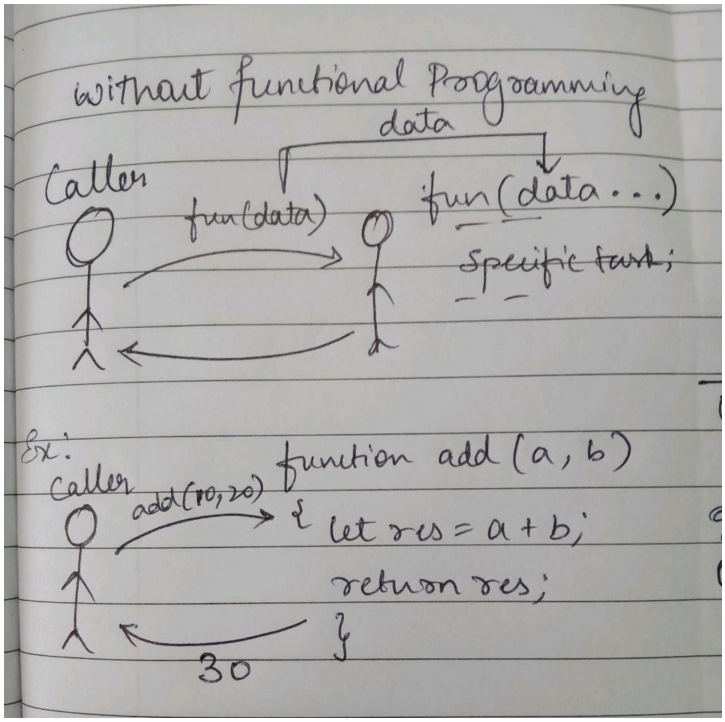
Functional Programming

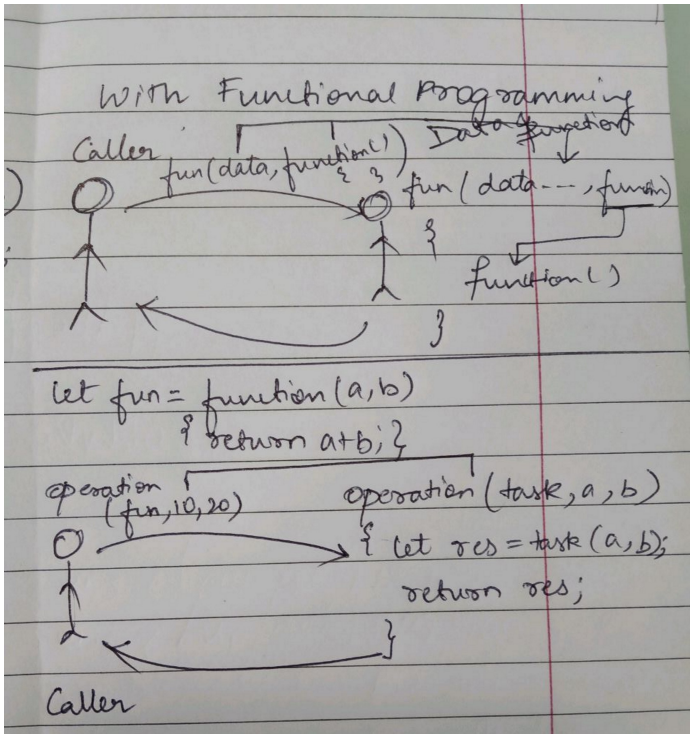
(Running notes)
A function calling another function which can accept a function as a argument is called as functional programming.
The function which is passed as a argument is called as a Call-back function.
The function which is can accept a function as a prameter Is called as a higher order function.

(Sir notes)
Passing a function as an argument to another function is known as Functioncal Programming.

Advantage of functional programming:
Instead of creating a function with specific task, functional programming helps the programmer to create functions which can perform generic task by accepting a function as a parameter.

Without functional programming	With Functional programming





In the above example:
The function operation() which can accept a function as a parameter is known as higher order function.
The function fun() which is passed as an argument to another function is known as Call back function.

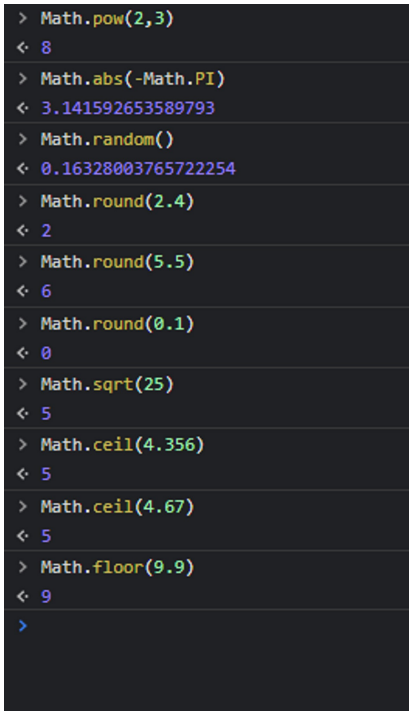
Higher-order function: A function which accepts a function as a parameter is known as a Higher-order function.

Exercise: Create a higher order function which can accept a function and three numbers. It should perform the given task on the three numbers and return the result.

```
function threeDigitOp(perform, a, b, c) {
  return perform(a, b, c);
}
//example1: to use higher order function (addition of three)
let res = threeDigitOp(function sum (a, b, c) {return a + b + c;},10,20,30);
console.log(res);
//example2: to find average of three numbers using hihger order function
let avg= threeDigitOp(function (a,b,c){return (a+b+c)/3},2,2,2);
console.log(avg);
```

Math object

WAP to generate a random Quotes.



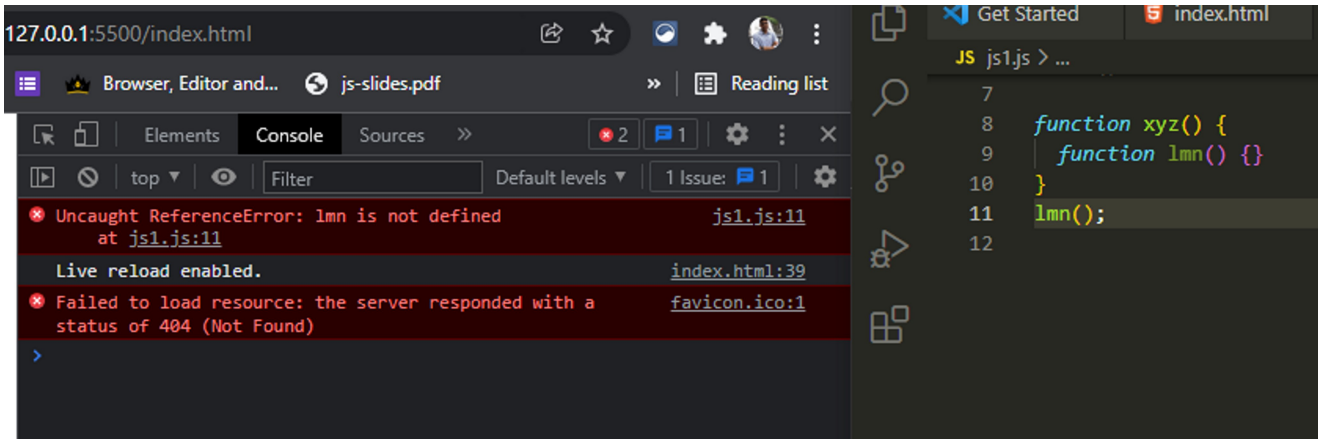
Date Object:

```
Date(Year, Month, day,hrs,mins,secs,milliseconds,timezone )
```

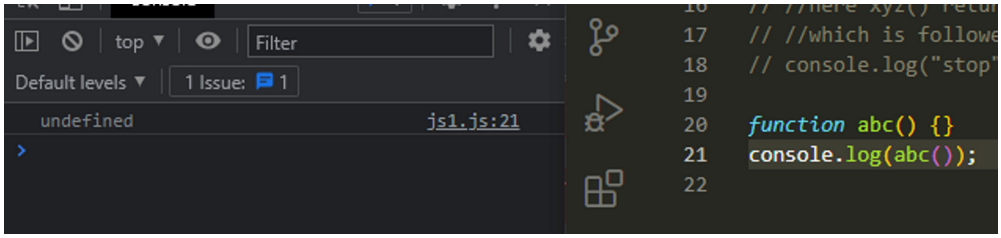
Months starts from 0 to 11
Day starts from 0 to 6
Hours starts from 0 to 23
Minutes-> 0 to 59
Second->0 to 59
Milliseconds->0 to 999

Closure-Scope-chaining

Thursday, December 30, 2021 11:51 AM

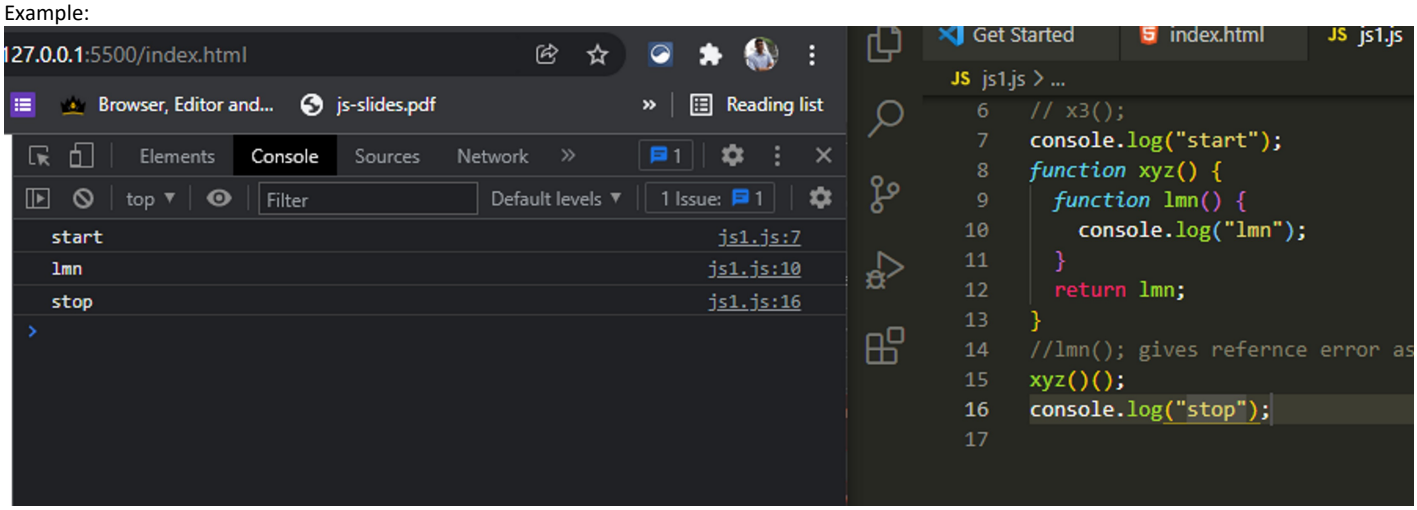


A child function called outside a parent function, throws reference error.



When a function is empty with no return statement it gives undefined back as return value.

Note: A function can return a function
Example:




```
console.log("start");

function xyz() {

  function lmn() {
    console.log("lmn");
  }
  return lmn;
}
//lmn(); gives refernce error as the scope is limited to the parent function
xyz()();
//here xyz() returns address of lmn when called
//which is followed by () to be executed so lmn gets executed
console.log("stop");
```

Self example

```
function twoDigit(a, b) {
  function add() {
    return a + b;
  }
  function sub() {
    return a - b;
  }
  return add;
}
console.log(twoDigit(10, 20)());
```

Nested Functions

In JS we can define a function inside another function is known as nested function.

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

    }
}
```

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

Note:

1. The inner function is local to outer function, it cannot be accessed from outside.
2. To use inner function outside, the outer function must return the reference of inner function.

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

    }
    return inner;
}
```

3. We can now call inner function from outside as follows

Type 1:

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

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

Type 2:

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

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

Scope Chaining (lexical Scope)

The ability of JavaScript Engine to search for a variable in the outer scope when it is not avialable in the local scope is known as Scope Chaining.

The scope chain is generally established between

1. A function and the global object.

   Eg:

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

   When test function is executed JSE looks for a in the local scope since it is not avilable, it will look for a in the outer scope i.e. global window object with the help of this keyword.

2. A child function and the parent function (Nested Function) with the help of a closure.

   Eg:

```
let a=10;
function x(){
  let b=20;
  function y(){
    console.log(b);//20
    console.log(a);//10
  }
  return y;
}
x()();
```

When the function y() is executed and console.log(b) is encountered, JSE looks for b in the local scope of function y, since b is not present and the function y is a child of function x JSE will search for b in the parent function x scope with the help of a ***closure.***

Closure

Closure is a binding of parent functions variables with the child function. This enables a JS programmer to use parent function's member inside child function.

Eg:

```
function person(){
  let age=21;
  function addAge(){
    return ++age;
  }
  return addAge;
}
let add1=person();
console.log(add1());//22
```

In the above example addAge() is a nested 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.

Alternative definition: the binding of inner function with its *Lexical Environment(Parent function's state)* is known as Closure.

Points to remember on Closure:

1. Closure helps to achieve scope chain(lexical Environment) from child function to parent function.
2. Closure preserves the state of parent function even after the execution of parent function is completed.
3. A child function will have reference to the closure.
4. Everytime a parent function is called a new closure is created.

Disdavantage of closure:

1. High memory consumption.

Example:

```
function counter() {
  let count = 0;
  return function () {return ++count;};
}
let count1 = counter();
let count2 = counter();//new closure for same parent is
console.log(count1());//1
console.log(count1());//2
console.log(count1());//3
console.log(count1());//4
console.log(count1());//5
console.log(count2());//1
```

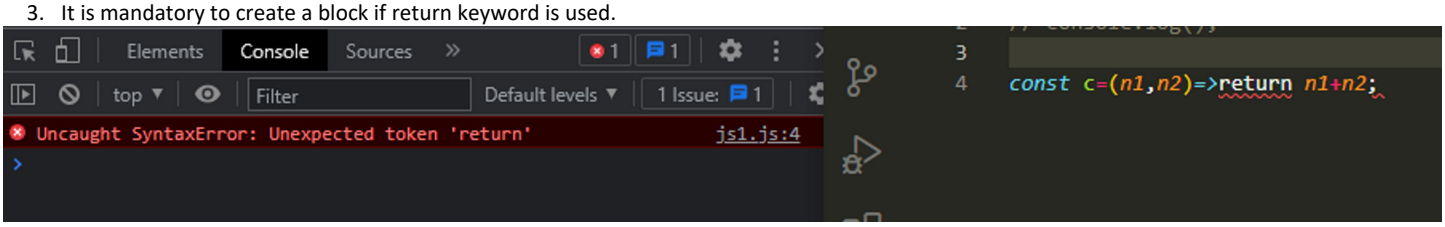
Arrow-IIFE

Friday, December 31, 2021 11:01 AM

Arrow functions

Arrow functions was introduced from ES6 version of JS
Main purpose is to reduce the syntax.
(parameter_list,. . .)=>{ }
Notes

1. Parameter is optional.
2. If function has only one statement, then block is optional.
3. It is mandatory to create a block if return keyword is used.



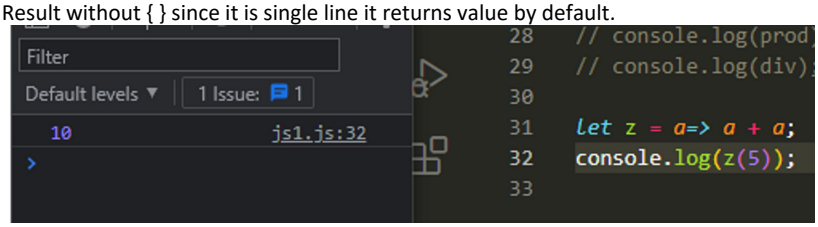

```
const c=(n1,n2)=>return n1+n2;//SyntaxError
```

Solution 1:
(a,b)=>{return a+b};
Solution 2:
(a,b)=> a+b;// executes expression and returns
The return type of the function is same as the expression return type.

Important note
Arrow functions can have two types of return

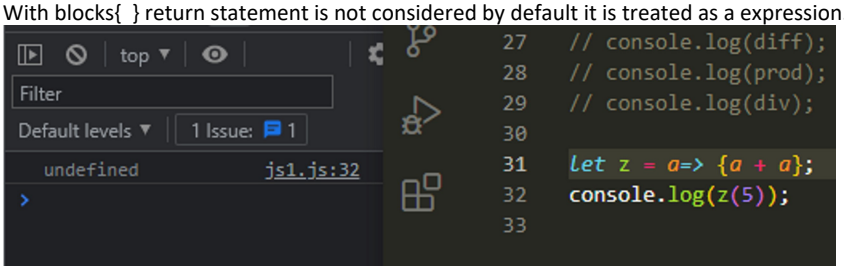
1. Implicit return
   Using return keyword Is not required.
   Syntax: (parameter list,…)=> expression;
2. Explicit return
   Using return keyword is mandatory inside the block. If return keyword is not used undefined is given back
   If a block is created then the arrow function behaves like explicit return.
   By default the return type of any function Is undefined.
   Syntax:(parameter list,…)=>{return expression};

Result without { } since it is single line it returns value by default.



Result with{ }

With blocks{ } return statement is not considered by default it is treated as a expression.



1. Create a function to return sum of two number

```
const c1 = (a, b) => a + b;
console.log(c1(10, 20));
```

2. To return product of two numbers

```
const c2 = (a, b) => a * b;
```

```
console.log(c2(10, 10));
```

3. To return difference of two numbers

```
const c3 = (a, b) => a - b;
console.log(c3(10, 5));
```

4. To return quotient of two numbers

```
const c4 = (a, b) => a / b;
console.log(c4(10, 2));
```

5. To return remainder of two numbers

```
const c5 = (a, b) => a % b;
console.log(c5(10, 3));
```

6. To return x^n

```
const c6 = (a, b) => a ** b;
console.log(c6(25, 4));
```

7. Using callback function

```
const operation=(cb)=>{
    let a=Number(prompt());
    let b=Number(prompt());
    return cb(a,b);
}
let sum=operation((a,b)=>a+b);
let diff=operation((a,b)=>a-b);
let prod=operation((a,b)=>a*b);
let div=operation((a,b)=>a/b);
console.log(sum);
console.log(diff);
console.log(prod);
console.log(div);
```

Assignment design a higher order function which can accept a call back function and three numbers, the higher order function should return the computed result generated by calling the call back function.
Call the higher order function to perform the following task

1. Sum of 10,20 and 30
2. Prod of 2,3 and 4
3. Sum of first two numbers divided by the third number
4. Largest of -10, 20 and -30

Immediate Invoking Function Expression-IIFE

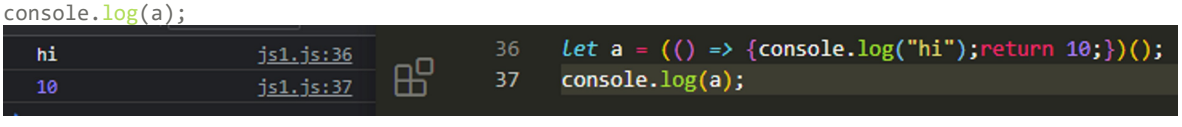
When the function is called immediately as soon as the function object is created it is known as Immediate Invokation Function.
Steps to achieve it

1. Treat a function like an expression by declaring it inside a pair of brackets.
2. Add another pair of brackets next to it which behaves like a function call statement.

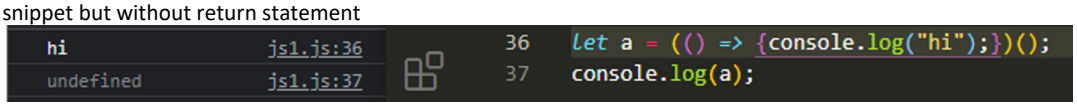
Ex:

```
(function abc() {
  console.log("hello");
})();
```

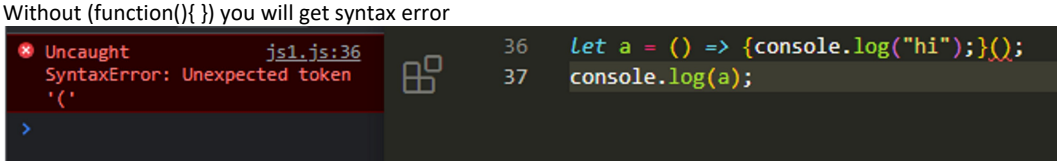
```
let a = (() => {console.log("hi");return 10;})();
console.log(a);
```



Same snippet but without return statement



Without (function(){ }) you will get syntax error



ARGUMENT object
The argument object is a local variable avilable within all non-arrow functions.
It returns an array like object.

Arrays

Saturday, January 1, 2022 10:13 AM

Arrays

1. Array is a block of memory which is used to store multiple values of same type.
2. To create an array
   a. In JS array is an object.
   b. We can create array object in three different ways
      i. Using array literal
      Syntax: let arr=[value1, value2,….]
      Example:

```
let arr=[10,20,30];
console.log(typeof arr);//object
```

      ii. By creating an instance of array using new operator
      Syntax: let arr2=new Array();
      Example:

      iii. By creating an instance of array and initializing the elements using array constructor.
      Syntax: let arr3=new Array(10,20,30);
      Example:

To access the array elements

We can access the array elements with the help of array object reference, array operator and index

Array_object_ref [index]

Index: it is an number, starts with 0 and wnds with length of the array object-1.

EX: let arr=[10,20,30];

//to print second element, index is 2-1 = 1

console.log(arr[1]);

Programs

1. Create an array with following elements 10,20,30 and 40 and display the elements in the reverse order.
2. Write an array to store names of atleast three cities, display all the cities whose length is even.
3. Create an array of fruits, display the names of fruits which is starting vowels.

```
let vowels = ["a", "i", "e", "o", "u"];
let fruits = ["apple", "mango", "eanana", "papaya", "apricot", "orange"];
for (let i = 0; i < fruits.length; i++) {
  for (let j = 0; j < vowels.length; j++) {
    if (fruits[i].charAt(0).toLowerCase() === vowels[j]) console.log(fruits[i]);
  }
}
```

4. Create an arry of food dish names, read a dish from the user using a prompt check whether the dish read from the user is avilable in the array or not.and print suitable message.

Array methods

Monday, January 3, 2022 11:20 AM

Pop()
Push(items)
Shift: shift method helps to remove an element from the first index of an array.

var arr=[10,20,30,40,50]
arr.shift( )//[20,30,40,50]

Unshift: helps to add an element at the first index.
var arr=[10,20,30];
arr.unshift(40,80);//[40,80,10,20,30]

Note: To retreive an element from an array we use index.

indexOf( ) - The index of method returns the index of a particular element which is passed to it.

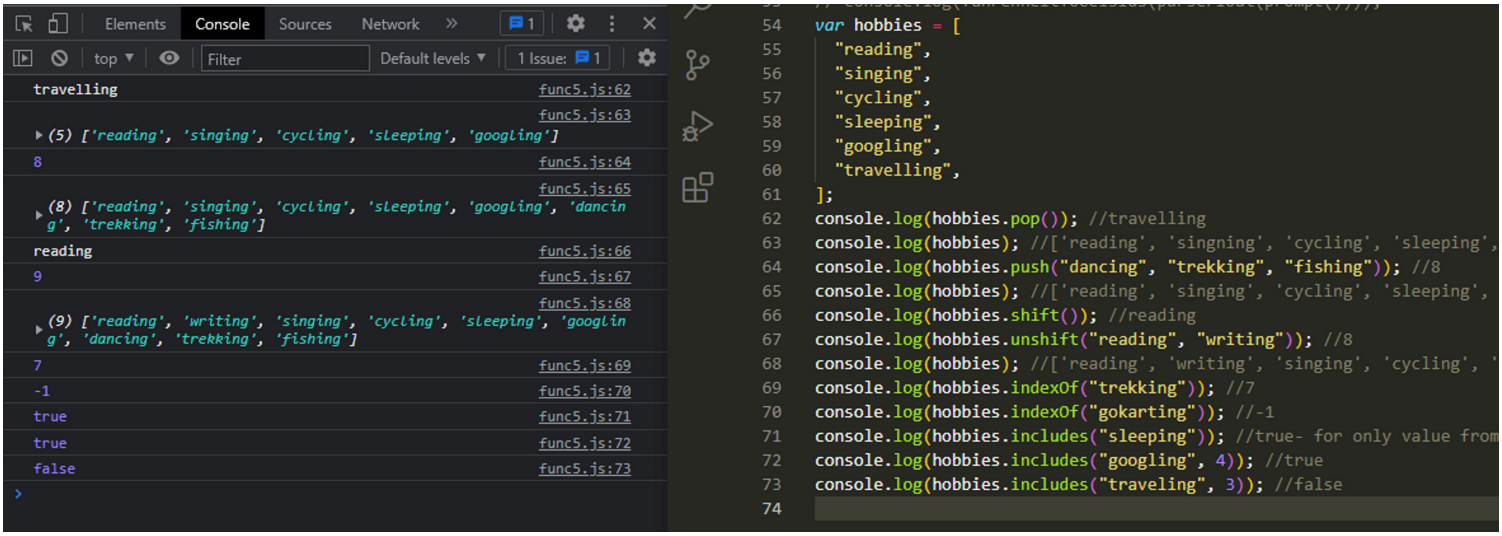
```
var hobbies=['reading','singing','cycling']
Hobbies.indexOf('singing');//1
Hobbies.indexOf('praying');//-1
```

includes(search value, from index or exact position): the includes method of an array represent whether the search element or value is present in an array along with its index value.

```
var hobbies=['reading','singning','cycling','sleeping','googling','travelling']
hobbies.includes('sleeping')//true- for only value from an array
hobbies.includes('googling',4)//true
hobbies.includes('traveling',3)//false
```

Check for includes it has some
Example: for all above methods

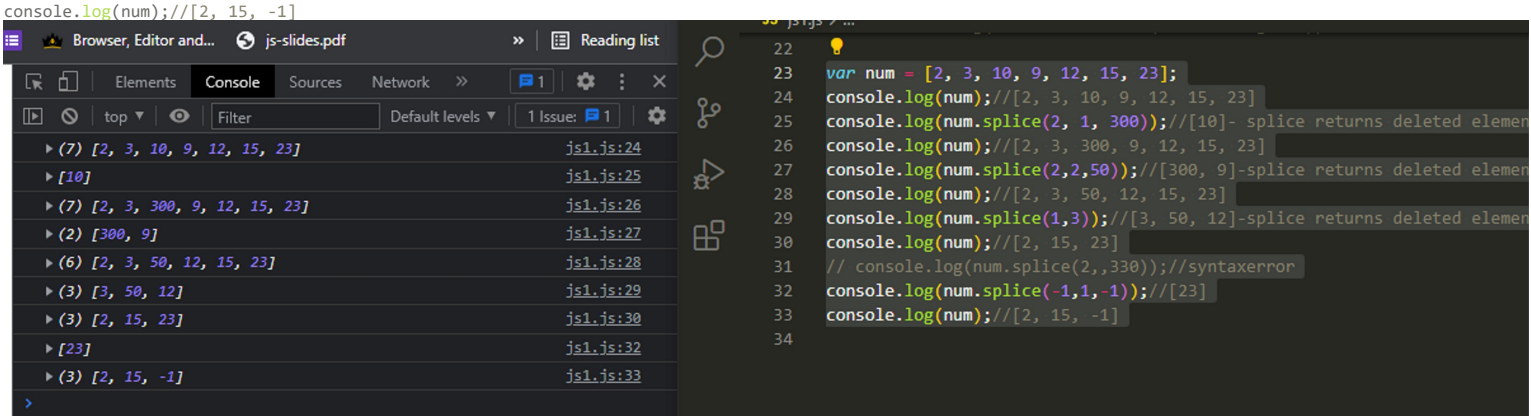
```
var hobbies = [
  "reading",
  "singing",
  "cycling",
  "sleeping",
  "googling",
  "travelling",
];
console.log(hobbies.pop()); //travelling
console.log(hobbies); //['reading', 'singning', 'cycling', 'sleeping', 'googling']
console.log(hobbies.push("dancing", "trekking", "fishing")); //8
console.log(hobbies); //['reading', 'singing', 'cycling', 'sleeping', 'googling', 'dancing', 'trekking', 'fishing']
console.log(hobbies.shift()); //reading
console.log(hobbies.unshift("reading", "writing")); //8
console.log(hobbies); //['reading', 'writing', 'singing', 'cycling', 'sleeping', 'googling', 'dancing', 'trekking',
'fishing']
console.log(hobbies.indexOf("trekking")); //7
console.log(hobbies.indexOf("gokarting")); //-1
console.log(hobbies.includes("sleeping")); //true- for only value from an array
console.log(hobbies.includes("googling", 4)); //true
console.log(hobbies.includes("traveling", 3)); //false
```



splice( ):-> helps to modify an array
Syntax: splice(start_index, Delete_count, items);
It is used to add and remove items

```
var num = [2, 3, 10, 9, 12, 15, 23];
console.log(num);//[2, 3, 10, 9, 12, 15, 23]
console.log(num.splice(2, 1, 300));//[10]- splice returns deleted element
console.log(num);//[2, 3, 300, 9, 12, 15, 23]
console.log(num.splice(2,2,50));//[300, 9]-splice returns deleted element
console.log(num);//[2, 3, 50, 12, 15, 23]
console.log(num.splice(1,3));//[3, 50, 12]-splice returns deleted element
```

```
console.log(num);//[2, 15, 23]
// console.log(num.splice(2,,330));//syntaxerror
console.log(num.splice(-1,1,-1));//[23]
console.log(num);//[2, 15, -1]
```



slice() used to create a new array
Syntax: slice(start index, end index)
it will count from the start index to end index reject the last value.
The slice method considers the start index and the end index element will be ignored while slicing.

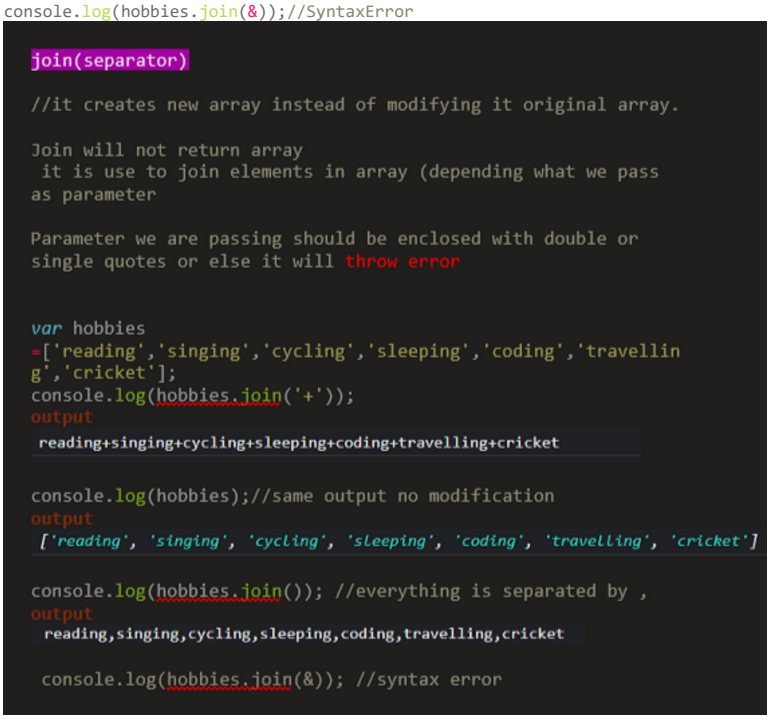
```
var num = [10, 20, 30, 40, 50, 60, 80, 90];
console.log(num.slice(2, 3)); //[30]
var a = num.slice(2, 3);
console.log(a);//[30]
console.log(num.slice(1,4));//[20, 30, 40]
[30];
console.log(num.slice(-1, 3));
```

What is the difference between slice and splice

Splice	Slice
Modify array	Create a new array
Original array modified	Original array is not modified
Can accept negative value	Will accept negative values

Join()
Helps to join array

```
console.log(hobbies.join());//reading,singing,cycling,sleeping,googling,travelling
console.log(hobbies.join('+'));//reading+singing+cycling+sleeping+googling+travelling
console.log(hobbies.join(&));//SyntaxError
```



Array.isArray(array_name):

Gives true or false

```
var num = [10, 20, 30, 40, 50, 60, 80, 90];
console.log(Array.isArray(num));
```

Q1) WAP to remove a specific item from an array.

```
var num = [10, 20, 30, 40, 50, 60, 80, 90];
console.log(num);
console.log("enter element to delete");
function dete(n) {
  var x = num.indexOf(n);
  if (x == -1) return "not present";
  else {
    num.splice(x, 1);
    return num;
  }
}
console.log(dete(Number(prompt())));
```

Q2)WAP to insert an item in an array.

```
var num = [10, 20, 30, 40, 50, 60, 80, 90];
console.log(num);
num.push(33);
num.unshift(99);
console.log(num);
num.splice(3,0,99);
console.log(num);//[10, 20, 30, 99, 40, 50, 60, 80, 90]
```

Q3)WAP to check if an array is an array.

```
var num = [10, 20, 30, 40, 50, 60, 80, 90];
console.log(Array.isArray(num));
```

Q4)WAP to empty an array.(imp three way)

```
var num = [10, 20, 30, 40, 50, 60, 80, 90];
pop
for (let i = num.length; i>-1; i--) {
  num.pop(i);
  console.log(num);
}
//splice method
num.splice(0, num.length);
console.log(num);
//shift
for (let i = num.length; i>-1; i--) {
      num.shift(i);
      console.log(num);
}
```

Q5)WAP to add element to start of an array.

```
var num = [10, 20, 30, 40, 50, 60, 80, 90];
num.unshift(1000);
console.log(num);
num.splice(0,0,9999);
console.log(num);
```

Q6)WAP to get random item form an array.

```
var num = [10, 20, 30, 40, 50, 60, 80, 90];
var x = num.length;
var rand = Math.round(Math.random() * num.length - Math.random());
console.log(rand);
console.log(num[rand]);
```

Q7)WAP to split array into smaller chunks.

```
var hobbies = [
      "reading",
      "singing",
      "cycling",
      "sleeping",
      "googling",
      "travelling",
];
var a=hobbies.slice(0,3)
console.log(a);
```

For each method- it is a special method
forEach() helps to reterive all the array elements at once.
forEach() method takes a callback function as an argument and helps to iterate its values along with its index.
Syntax: forEach(callback(values, index))

```
var num = [10, 20, 30, 40, 50, 60, 80, 90];
num.forEach((value,index)=>{console.log(value,index);});
```

WAP to multiply 50 for all the array elements within the range of 30 to 40

MAP and filter methods

1. Design a function to add three to all elements and display the new array.

```
var arr = [10, 20, 30];
var test2=arr.map((element)=>element+3);
console.log(test2);

var a=[10,20]
function high(fun){
    let b=new Array();
    for(let i=0;i<arr.length;i++){
        b[i]=fun(a[i]);
    }
    return b;
}
let c=high((v)=>v+3);
console.log(c);
```

2. Design a function to print the even values in the array.

```
var num = [11, 21, 31, 32, 34, 46, 38, 41, 51, 61, 81, 90];
function filter(fun){
    let b=new Array();
    let j=0;
    for(let i=0;i<=num.length-1;i++){
        if(fun(num[i])) b[j++]= num[i];
// if you assign the b[i]=nu[i] you will get array with empty position instead take new variable
    }
    return b;
}
var ev=filter((v)=>v%2==0?true:false);
console.log(ev);//[32, 34, 46, 38, 90]
var e=num.map((v) => {if (v % 2 == 0) return v;} );
console.log(e);//[undefined, undefined, undefined, 32, 34, 46, 38, undefined, undefined, undefined, undefined, 90]
var even = num.filter((v) => {if (v % 2 == 0) return v;} );
console.log(even);//[32, 34, 46, 38, 90]
```

3. Design a function to add 5 and find even numbers from the addition and display

```
var num = [11, 21, 31, 32, 34, 46, 38, 41, 51, 61, 81, 90];
console.log(num);//[11, 21, 31, 32, 34, 46, 38, 41, 51, 61, 81, 90]
var num1 = num.map((v) => v + 5);
console.log(num1);//[16, 26, 36, 37, 39, 51, 43, 46, 56, 66, 86, 95]
var num2=num.filter(v=>v%2==0?true:false);
console.log(num2);//[32, 34, 46, 38, 90]
```

4. Difference between map and forEach()

Map( )	forEach( )
Return array	Logs the result of the expression

5. Difference between map and filter

map( )	filter( )
The map() method creates a new array with the results of calling a function for every array element. The map method allows items in an array to be manipulated to the user's preference, returning the conclusion of the chosen manipulation in an entirely new array.	The filter() method creates an array filled with all array elements that pass a test implemented by the provided function.
New array of same size is created	New array of size only considering true outcomes is created
If the condition is not satisfied undefined is placed as default in array index positions	If the condition is not satisfied then they won't make it new array or falsy values are neglected.

6. Write 5 different example on forEach( )
7. Write 5 different example on filter( )
8. Write 5 different example on map( )

9. Array.from( ): it is used to create a new array from an array-like parameter
   The from() method takes in:
   arraylike - Array-like or iterable object to convert to an array.
   mapFunc (optional) - Map function that is called on each element.
   thisArg (optional) - Value to use as this when executing mapFunc.
   Note: Array.from(obj, mapFunc, thisArg) is equivalent to Array.from(obj).map(mapFunc, thisArg).
   Note: This method can create Array from:
   Array-like objects - The objects that have length property and have indexed elements like strings.
   Iterable objects like Map or Set.

```
// Array from String
let arr1 = Array.from("abc");
console.log(arr1); // [ 'a', 'b', 'c' ]
```

- **Array.from()**
  The Array.from() static method creates a new, shallow-copied Array instance from an array-like or iterable object.

  Syntax - Array.from(*arrayLike* [, *mapFn* [, *thisArg*]])

  arrayLike - An array-like or iterable object to convert to an array.
  mapFn - [Optional] Map function to call on every element of the array.
  thisArg - [Optional] Value to use as this when executing *mapFn*.

  Example :
  ```
  console.log(Array.from('foo'));
  // expected output:    Array ["f", "o", "o"]

  console.log(Array.from([1, 2, 3], x => x + x));
  // expected output: Array [2, 4, 6]
  ```

```
function createArr(arraylike, mapFunc) {
 return Array.from(arraylike, mapFunc);
}

// using arrow function for mapFunc
let arr1 = createArr("123456", (x) => 2 * x);
console.log(arr1); // [ 2, 4, 6, 8, 10, 12 ]
```

10. Array.of( ):
    The JavaScript Array of() method creates a new Array instance from the given arguments.

    The syntax of the of() method is:
    Array.of(element0, element1, ..., elementN)
    The of() method, being a static method, is called using the Array class name.
    of() Parameters
    The of() method takes in an arbitrary number of elements that is then used to create the array.
    Note: The difference between Array.of() and the Array constructor is the handling of the arguments. For example, Array.of(5) creates an array with a single element 5 whereas Array(5) creates an empty array with a length of 5.
    - **Array.of()**
      The **Array.of()** method creates a new Array instance from a variable number of arguments, regardless of number or type of the arguments.

      Syntax - Array.of(*element0*[, *element1*[, ...[, *elementN*]]])

      element - Elements used to create the array.

      Example :
      ```
      Array.of(7);              // [7]
      Array.of(1, 2, 3);        // [1, 2, 3]

      Array(7);                 // array of 7 empty slots
      Array(1, 2, 3);           // [1, 2, 3]
      ```

```
Array.of(5);      // [5]
Array.of(1, 2, 3); // [1, 2, 3]

Array(5);         // array of 5 empty slots
Array(1, 2, 3);   // [1, 2, 3]
```

11. Array.find( ): Returns the value of the first element in the array that satisfies the given function.
```
let numbers = [1, 2, 3, 4, 5];
console.log(numbers.find(e => e % 2 == 0));//2
```
    - **Array.find()**
      The **find()** method returns the **value** of the **first element** in the provided array that satisfies the provided testing function

      Syntax - *arr*.find(*callback(element[, index[, array]])*[, *thisArg*])

      callback - Function to execute on each value in the array, taking 3 arguments:
      element - The current element in the array.

      Example :
      ```
      const array1 = [5, 12, 8, 130, 44];

      const found = array1.find(element => element > 10);

      console.log(found);    // expected output: 12
      ```

12. Array.findIndex( ):
```
let ranks = [1, 5, 7, 8, 10, 7];
let index = ranks.findIndex(rank => rank === 7);
console.log(index);
```

- **Array.findIndex()**

  The findIndex() method returns the index of the first element in the array that satisfies the provided testing function. Otherwise, it returns -1, indicating that no element passed the test.

  Syntax - *arr*.findIndex(*callback*(*element*[, *index*[, *array*]])[, *thisArg*])

  callback - A function to execute on each value in the array until the function returns true, indicating that the satisfying element was found. It takes three arguments:
  element - The current element being processed in the array.

  Example :

```
const array1 = [5, 12, 8, 130, 44];
const isLargeNumber = (element) =>
        element > 13;
console.log(array1.findIndex(isLargeNumber));
// expected output: 3
```

13. Array.some( ):The JavaScript Array some() method tests whether any of the array elements pass the given test function

```
function checkMinor(age) {
  return age < 18;
}
const ageArray = [34, 23, 20, 26, 12];
let check = ageArray.some(checkMinor); // true
```

- **Array.some()**

  The **some()** method tests whether at least one element in the array passes the test implemented by the provided function. It returns a Boolean value.

  Syntax - *arr*.some(*callback*(*element*[, *index*[, *array*]])[, *thisArg*])

  callback - A function to test for each element, taking three arguments:
  element - The current element being processed in the array.

  Example :

```
const array = [1, 2, 3, 4, 5];
// checks whether an element is even
const even = (element) => {
                element % 2 ===0;
        }
console.log(array.some(even));
// expected output: true
```

14. Array.every( ):The JavaScript Array every() method checks if all the array elements pass the given test function.

```
function checkAdult(age) {
  return age >= 18;
}

const ageArray = [34, 23, 20, 26, 12];
let check = ageArray.every(checkAdult); // false
```

- **Array.every()**

  The **every()** method tests whether all elements in the array pass the test implemented by the provided function. It returns a Boolean value.

  Syntax - *arr*.every(*callback*(*element*[, *index*[, *array*]])[, *thisArg*])

  callback - A function to test for each element, taking three arguments:
  element - The current element being processed in the array.

  Example :

```
const isBelowThreshold = (currentValue) => currentValue < 40 ;

const array1 = [1, 30, 39, 29, 10,13];
console.log(array1.every(isBelowThreshold));
// expected output: true
```

Reduce method

```
const inventory={
    pname:['parleg','goodday','hidenseek','oreo','dark fantasy']
    ,price:[10,20,15,30,40]
};
let total=inventory.price.reduce((acc,value)=>acc+value);
console.log(total);
//reduce(callback function,inital value )- if no initial value is passed by default the first element in the array is the
initial value
```

First parameter is accumalator
second parameter is value

Objects Oriented Programming

Tuesday, January 4, 2022 12:09 PM

Object
Any substance which has its exsistence in the real world is known as an object.
Every object will have properties (attributes or fields) which describe them, every object will have actions (behaviours)
Example:

- Car
  - Properties
    - Color, price, model
  - Actions
    - Drive, brake and moving
- Webpage
  - Properties
    - URL, content
  - Actions
    - Click, scroll

Note: In programming world an object is a block of memory which represents a real world object, the properties of real world object are represented using variables.
The actions of real world object are represented using methods(functions).
In JS object is a collection of attributes and actions in the form of key value pairs enclosed within curly brackets.

Key represents properties or attributes of an object.
Value represents state of an object.

In JS we can create objects in three different ways

1. Using Object literals
2. Using a function
3. Using a Class

Creating an object using object literal

let variable=	{ key1:value1, key2:value2, key3:value3,.....}

Holds the reference of the object	Creates an object and returns it reference
Variable 0x1	Key1
	Key 2
	Key 3

We can access the values of an object with the help of dot operator, period, object reference and key.
Syntax: variable.key1:--------> gives values 1

```
let emp1 = { eid: 1, ename: "smitha", sal: 2000 };
console.log(emp1);
console.log(emp1.eid);
console.log(emp1.ename);
console.log(emp1.sal);
```

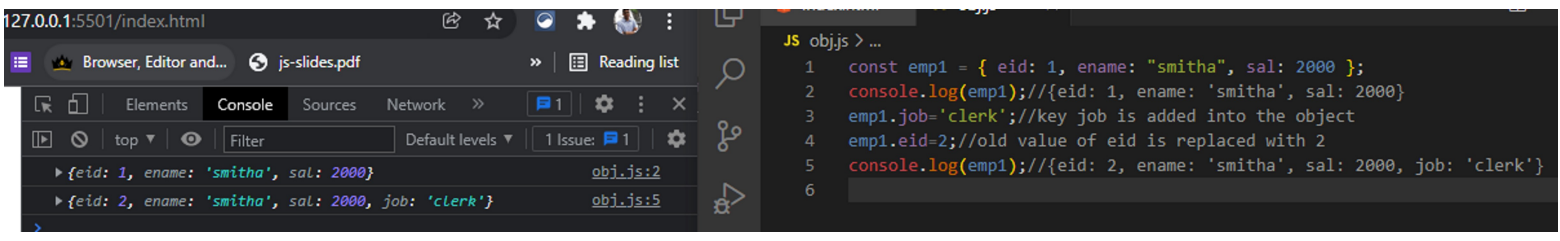
Note:

1. If the key is not present in the object then we get undefined as the value.
   Ex:
   ```
   let emp1 = { eid: 1, ename: "smitha", sal: 2000 };
   console.log(emp1.eid);//1 is output
   console.log(emp1.job);// not present so undefined
   ```
2. The objects of JS is mutable in nature, the state of the object can be modified.
   Eg:
   ```
   const emp1 = { eid: 1, ename: "smitha", sal: 2000 };
   console.log("current salary", emp1.sal);//2000
   emp1.sal = 20000;
   console.log("current salary", emp1.sal);//20000
   ```

To add a key value pair into an object

Syntax: object_reference.key = value;
Note: if the key is exists in the object old value is replaced with new value.
If the key does not exist in the object then a new pair of key:value is added into the object.

```
const emp1 = { eid: 1, ename: "smitha", sal: 2000 };
console.log(emp1);//{eid: 1, ename: 'smitha', sal: 2000}
emp1.job='clerk';//key job is added into the object
emp1.eid=2;//old value of eid is replaced with 2
console.log(emp1);//{eid: 2, ename: 'smitha', sal: 2000, job: 'clerk'}
```



To delete a key from a object

We can remove a key from an object with the help of delete operator
Syntax: delete object_ref.key

```
const emp1 = { eid: 1, ename: "smitha", sal: 2000 };
delete emp1.sal;
console.log(emp1);
```

Task1 let us consider a real world object marker with the following attributes

1. Ink, brand, price their states are black, camlin, 25
2. Create JS object to store them
3. Display the entire object
4. Log price of the marker
5. Change the ink from black to blue and log it
6. Add extra attribute shape into the marker object with value as cylinder.
7. Remove the attribute brand from the marker object
8. Display the entire final object.

```
const marker={ink:'black', brand:'camlin', price:25};
console.log(marker);
console.log(marker.price);
marker.ink='blue';
console.log(marker.ink);
marker.shape='cylinder';
delete marker.brand;
console.log(marker);
```

Dot notation everything above
Bracket notation
Ex:

```
const emp1 = { eid: 1, ename: "smitha", sal: 2000 };
delete emp1.sal;
console.log(emp1["ename"]);


console.log(emp1[ename]);//you will get syntax error
```

Create an object and reterive the data's from an object using dot notation and bracket notation.

Nested objects

```
const emp1 = {
  eid: 1,
  ename: "smitha",
  sal: 2000,
  details: { dn0: 123, city: "blore" },
};
delete emp1.sal;
console.log(emp1.details.city);//blore
```

Reduce method

```
const inventory={
    pname:['parleg','goodday','hidenseek','oreo','dark fantasy']
    ,price:[10,20,15,30,40]
};
inventory.pname.push('marry gold');
inventory.price.push(50);
// for(let i=0;i<=inventory.pname.length-1;i++){
//     console.log(inventory.pname[i],inventory.price[i]);
// }
//t2
console.log(`there are ${inventory.pname.length} in the inventory`);
//calculate total cost of the invenotry

let total=inventory.price.reduce((acc,value)=>acc+value);
console.log(total);
//reduce(callback function,inital value )- if no initial value is passed by default the first element in the array is
the initial value
```

Create an object with following details
Kart:

```
{
    Pname:[]
    Price:[]
}
```

T1: add atleast 4 products into the kart
T2:Find total cost of the kart
T3:what will be the price of the products if there is an inflation of 5%
T4:display names of the products whose length is even

```
let kart={
    pname:['amazon-ebook','flip-phon','phone','book'],
    price:[30,60,90,10]
};
let cost=kart.price.reduce((a,v)=>a+v,0);
console.log(cost);
let infla=kart.price.map((v)=>v*1.05);
console.log(infla);
let ev=kart.pname.filter((v)=>v.length%2==0);
console.log(ev);
```

Destructuring of objects

The process of extracting the values from the object into the variables is known as Destructuring of objects.

Syntax:

let/var/const {keyname,key_name, . . .} = obj_ref;

Note:

1. All the key names provided on the LHS are considered as variables.
2. JSEngine will search for the key inside the object, if the key is present the value is extracted and copied into the variable. If the key is not present undefined is stored in the variable.

Ex:

```
const emp={eid:1,ename:'sheela',sal:2000};

// case 1 :
const emp = {
 eid : 1,
 ename : "vikas",
 sal : 20000
};
 let {eid, ename}=emp;
 console.log(eid);
 console.log(ename);


// case 2 :
const emp = {
 eid : 1,
 name : " vikas",
 sal :20000
}
 let {a, sal}=emp;
 console.log(a);//undefined
 console.log(sal);//20000
```

- ```
  // case 3 : given default values incase the key is not present given value is printed on console
   const emp = {
   eid : 1,
   name : " vikas",
   sal :20000
   }
   let {address="address is not found", sal}=emp;
   console.log(address);//address is not found
   console.log(sal);//20000
  ```
  ```

Rest parameter-spread operator

Thursday, January 6, 2022 12:24 PM

Rest parameter

Rest parameter is used to accept multiple values as an array of elements.
Generally it converts multiple values into an array.

Syntax: . . . identifier

Uses of rest parameter

1. Rest parameter can be used in function definition, parameter list, to accept multiple values.
2. It can also be used in array and object destructuring.

Using rest parameter in a function

1. The function can receive multiple arguments passed by the caller and store them in an array.

Eg:	function test(…a){
	clg(a);
	}

Case 1: we have a caller who calls t
Paste picture

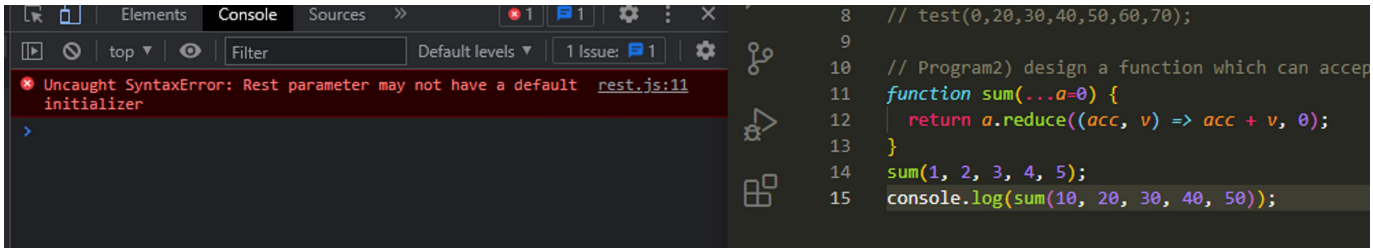
Rules of rest parameter

1. Rest parameter should be always defined at the last.

Eg:	function test(…a,b){
	clg(a);
	}//syntax error
		function test(b,…a){
		//correct way
		}

We cannot give default values to rest parameter.

```
function sum(...a=0) {//syntax error
```



Program 1) Design a function which can accept multiple arguments and list them on the console.

```
function test(b=5,...a){
    console.log(b);
    console.log(a.forEach(v=>console.log(v)));
}
test();
test(0,20,30,40,50,60,70);
```

Program2) design a function which can accept any number of arguments and returns the summation of all the arguments

```
function sum(...a) {
  return a.reduce((acc, v) => acc + v, 0);
}
sum(1, 2, 3, 4, 5);
console.log(sum(10, 20, 30, 40, 50));
```

Assignment questions

1. Design a function which can accept any number of arguments and returns the maximum element.
   Own logic and reduce
2. Design a function which can accept multiple arguments and return an array of cube of all the arguments.

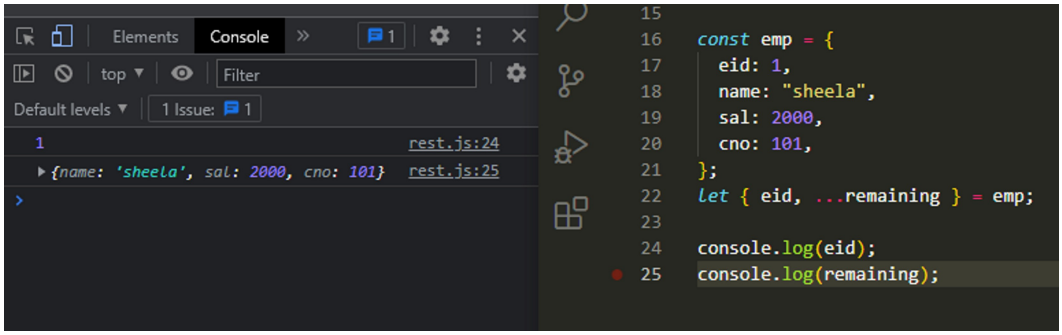
Using rest parameter for destructuring
Rest parameter can be used to destructure a JS object.
Syntax: { variable1, variable2, . . .identifer} = object_reference;
Note: When rest parameter is used for destructuring of object, all the key value pairs which is not yet extracted will be destructured as an object.
Ex:

```
const emp = {
  eid: 1,
  name: "sheela",
  sal: 2000,
  cno: 101,
};
```

```
let { eid, ...remaining } = emp;
```

The key eid Is present in the object hence the value of eid is unpacked or extracted into the variable eid.
The remaining keys name, sal and cno is not destructured or unpacked. Therefore all these key value pairs will be extracted as an object and stored inside the variable remaining.

```
const emp = {
  eid: 1,
  name: "sheela",
  sal: 2000,
  cno: 101,
};
let { eid, ...remaining } = emp;
console.log(eid);
console.log(remaining);//the index is not denoted with numbers but key's so it not an object (while using rest parameter)
```



Ex2:

```
const emp = {
        eid: 1,
        name: "sheela",
        sal: 2000,
        cno: 101,
      };
    let{eid,sal,...oth}=emp;
    let emp2=oth;
    console.log(emp2.eid);//undefined
    console.log(emp2);//{name: 'sheela', cno: 101}
```

Array destructuring
using rest parameter in array destructuring
If rest parameter is used in array destructuring, then all the elements which Is not destructured will be copied into a new array object and that array object will be returned.

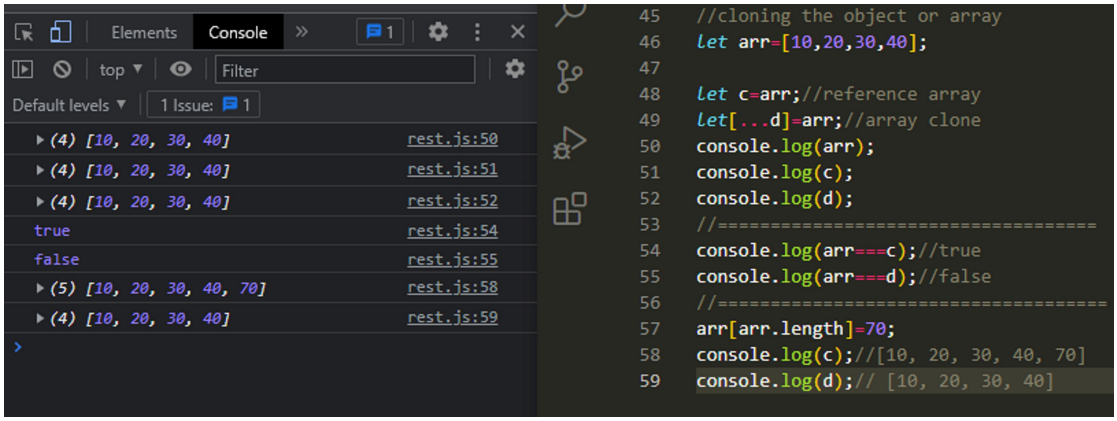
```
// rest parameter with arrays
let arr=[10,20,30,40];
let [a,b,...c]=arr
console.log(a);//10
console.log(b);//20
console.log(c);//[30,40]
```

Using rest parameter to clone array or object

1. To clone an array
   Syntax: let [. . . Identifer] = array_reference ;

Example:

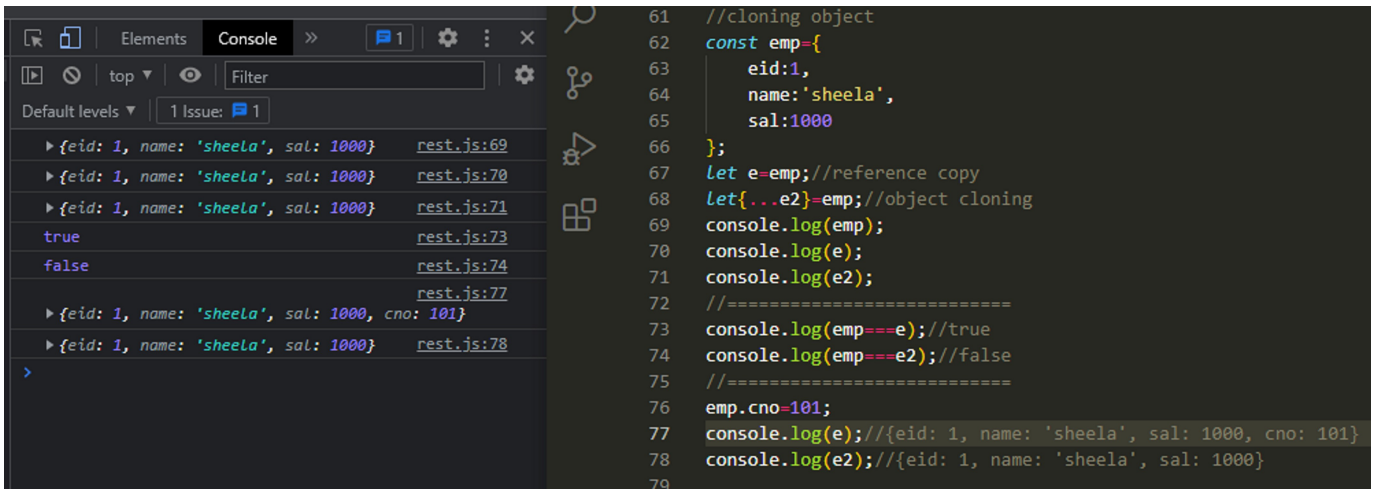
```
//cloning the object or array
let arr=[10,20,30,40];
let c=arr;//reference array
let[...d]=arr;//array clone
console.log(arr);
console.log(c);
console.log(d);
//===================================
console.log(arr===c);//true
console.log(arr===d);//false
//===================================
arr[arr.length]=70;
console.log(c);//[10, 20, 30, 40, 70]
console.log(d);// [10, 20, 30, 40]
```



2. To clone an object
   Syntax: let {. . . Identifer } = array_reference ;

Example:

```
//cloning object
const emp={
    eid:1,
    name:'sheela',
    sal:1000
};
let e=emp;//reference copy
let{...e2}=emp;//object cloning
console.log(emp);
console.log(e);
console.log(e2);
//==========
console.log(emp===e);//true
console.log(emp===e2);//false
//==========
emp.cno=101;
console.log(e);//{eid: 1, name: 'sheela', sal: 1000, cno: 101}
console.log(e2);//{eid: 1, name: 'sheela', sal: 1000}
```



Uses of rest

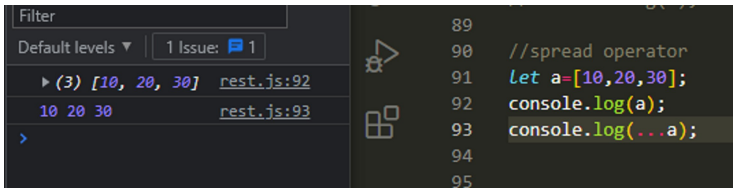
1. Can be used in a function to store excess value of arguments as an array
2. Destructuring of object n arrays
3. Clone the array and object

Program1)Design a function which can accept the start and end of a range and returns all the even numbers between them in a array format.
Call the function by passing the values 100 and 200
The result array must have all the even numbers except first three elements.

```
let a = new Array();
function ele(g,h){
for (let i = g; i <=h ; i++) {
  if(i%2==0)a.push(i);
}
return a;
}
let [e,b,c,...d]=ele(100,200);
console.log(d);
```

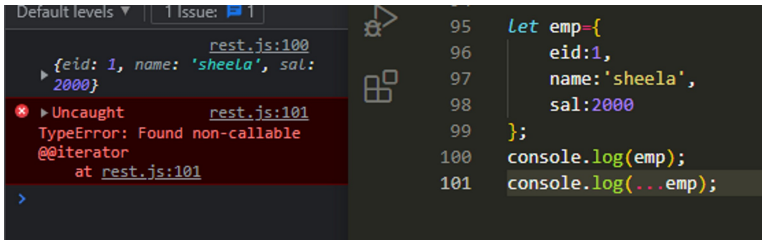
Spread operator
Spread operator is used to perform unpacking, spread operator uses an iterator to access each and every element or a property present in the array or object.

```
//spread operator
let a=[10,20,30];
console.log(a);
console.log(...a);
```



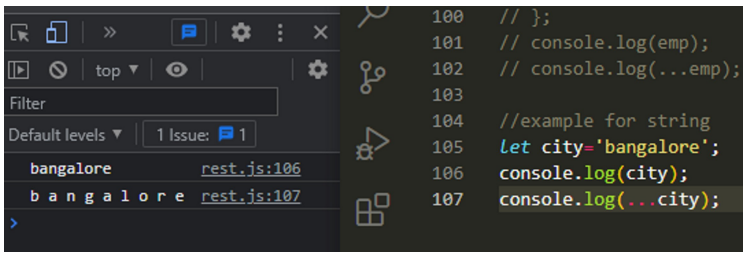
Example2:

```
let emp={
    eid:1,
    name:'sheela',
    sal:2000
};
console.log(emp);
console.log(...emp);//TYPE ERROR non-callable iterator
```



Example 3: with string

```
//example for string
let city='bangalore';
console.log(city);
console.log(...city);
```



let b=...a;//gives error
Note:
Spread operator unpacks the given iterable object (string, array, objects) and sends the unpacked data to a destination.

1. The unpacked data can be sent to a function as arguments. By using spread operator in the function call statement.
   Eg:

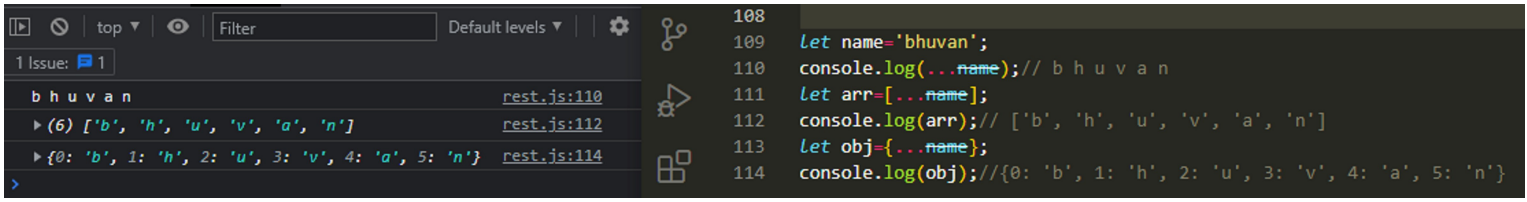
```
let name='bhuvan';
console.log(...name);// b h u v a n
```

2. We can ask the spread operator to store the elements in an array object by using spread operator inside rectangular brackets.
   Eg:

```
let name='bhuvan';
console.log(...name);// b h u v a n
let arr=[...name];
console.log(arr);// ['b', 'h', 'u', 'v', 'a', 'n']
```

3. We can ask the spread operator to store the unpacked data into an JS object by using spread operator inside curly brackets.
   Eg:

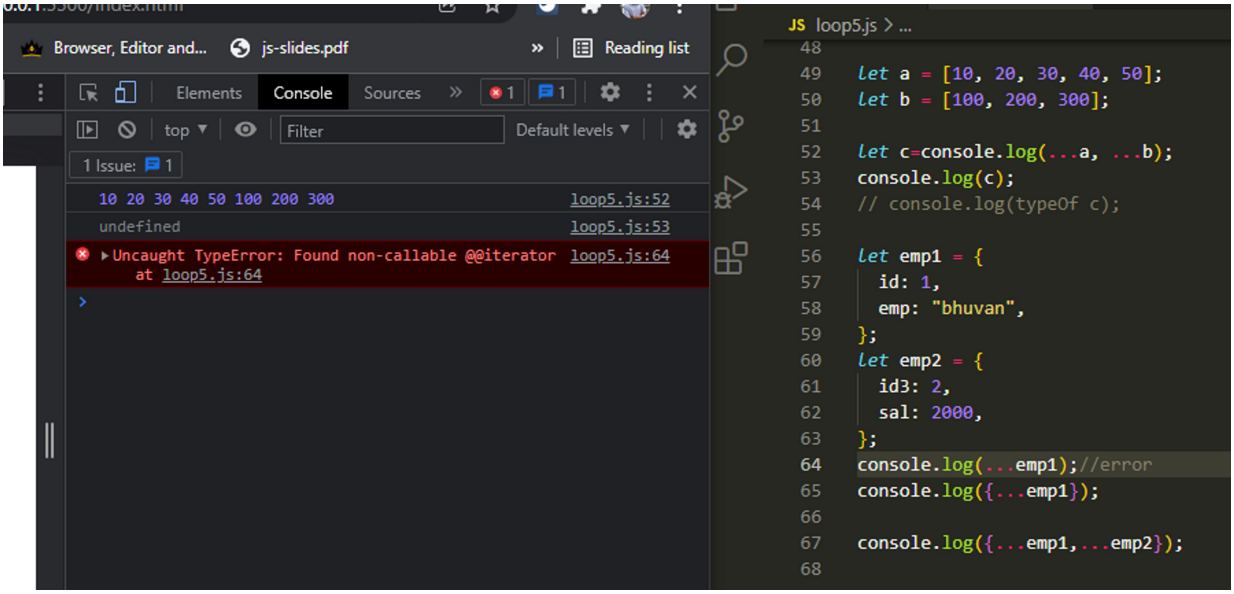
```
let name='bhuvan';
console.log(...name);// b h u v a n
let arr=[...name];
console.log(arr);// ['b', 'h', 'u', 'v', 'a', 'n']
let obj={...name};
console.log(obj);//{0: 'b', 1: 'h', 2: 'u', 3: 'v', 4: 'a', 5: 'n'}
```



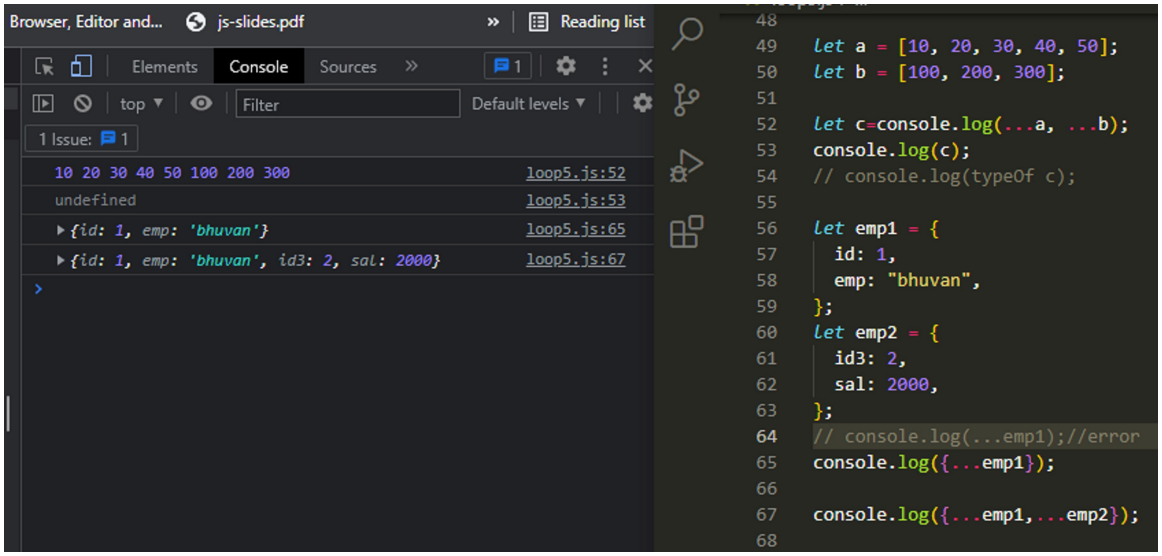
Both rest and spread are introduced in ES6

Possibilities of merging an array

```
let a = [10, 20, 30, 40, 50];
let b = [100, 200, 300];
let c=console.log(...a, ...b);
console.log(c);
// console.log(typeOf c);
let emp1 = {
  id: 1,
  emp: "bhuvan",
};
let emp2 = {
  id3: 2,
  sal: 2000,
};
console.log(...emp1);//error
console.log({...emp1});
console.log({...emp1,...emp2});
```




```
let a = [10, 20, 30, 40, 50];
let b = [100, 200, 300];
let c=console.log(...a, ...b);
console.log(c);
// console.log(typeOf c);
let emp1 = {
  id: 1,
  emp: "bhuvan",
};
let emp2 = {
  id3: 2,
  sal: 2000,
};
// console.log(...emp1);//error
console.log({...emp1});
console.log({...emp1,...emp2});
```


Browser, Editor and...
js-slides.pdf
Reading list
Elements
Console
Sources
top
Filter
Default levels
1 Issue: 1
10 20 30 40 50 100 200 300
loop5.js:52
undefined
loop5.js:53
{id: 1, emp: 'bhuvan'}
loop5.js:65
{id: 1, emp: 'bhuvan', id3: 2, sal: 2000}
loop5.js:67
48
49 let a = [10, 20, 30, 40, 50];
50 let b = [100, 200, 300];
51
52 let c=console.log(...a, ...b);
53 console.log(c);
54 // console.log(typeOf c);
55
56 let emp1 = {
57 id: 1,
58 emp: "bhuvan",
59 };
60 let emp2 = {
61 id3: 2,
62 sal: 2000,
63 };
64 // console.log(...emp1);//error
65 console.log({...emp1});
66
67 console.log({...emp1,...emp2});
68

For-in and of loop

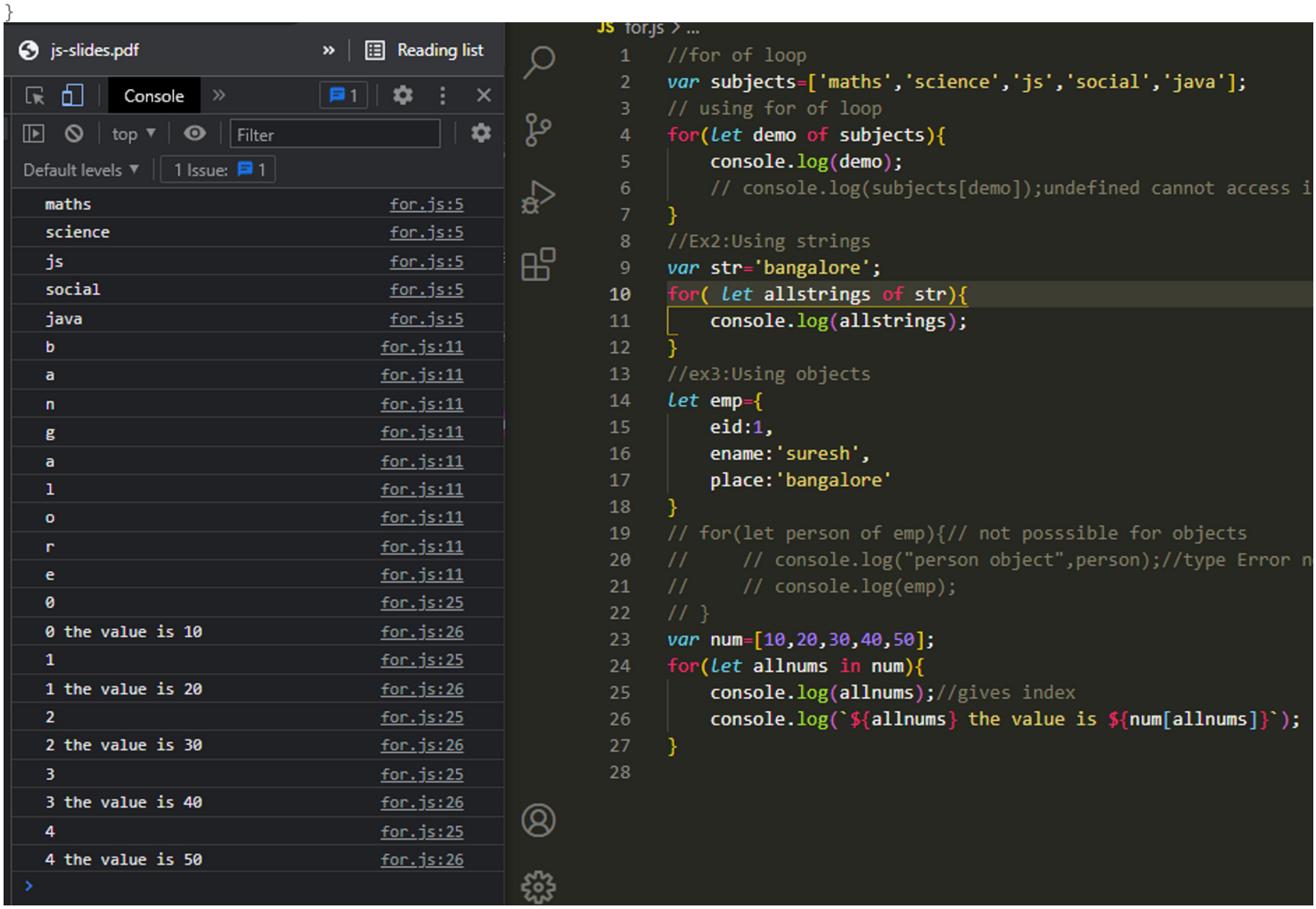
Friday, January 7, 2022 11:09 AM

Two types of loops
for of loop :
-> allows you to iterate over iterable objects (arrays, sets, maps,strings,etc)

Syntax:	for(var/let/const element of iterator){ }

Examples:
For of loop returns only the values present in the array and does not return the index positions

```
//for of loop
var subjects=['maths','science','js','social','java'];
// using for of loop
for(let demo of subjects){
    console.log(demo);
    console.log(subjects[demo]);
}
//Ex2:Using strings
var str='bangalore';
for( let allstrings of str){
    console.log(allstrings);
}
//ex3:Using objects
let emp={
    eid:1,
    ename:'suresh',
    place:'bangalore'
}
for(let person of emp){// not posssible for objects
    // console.log("person object",person);//type Error not iterable
    // console.log(emp);
}
```



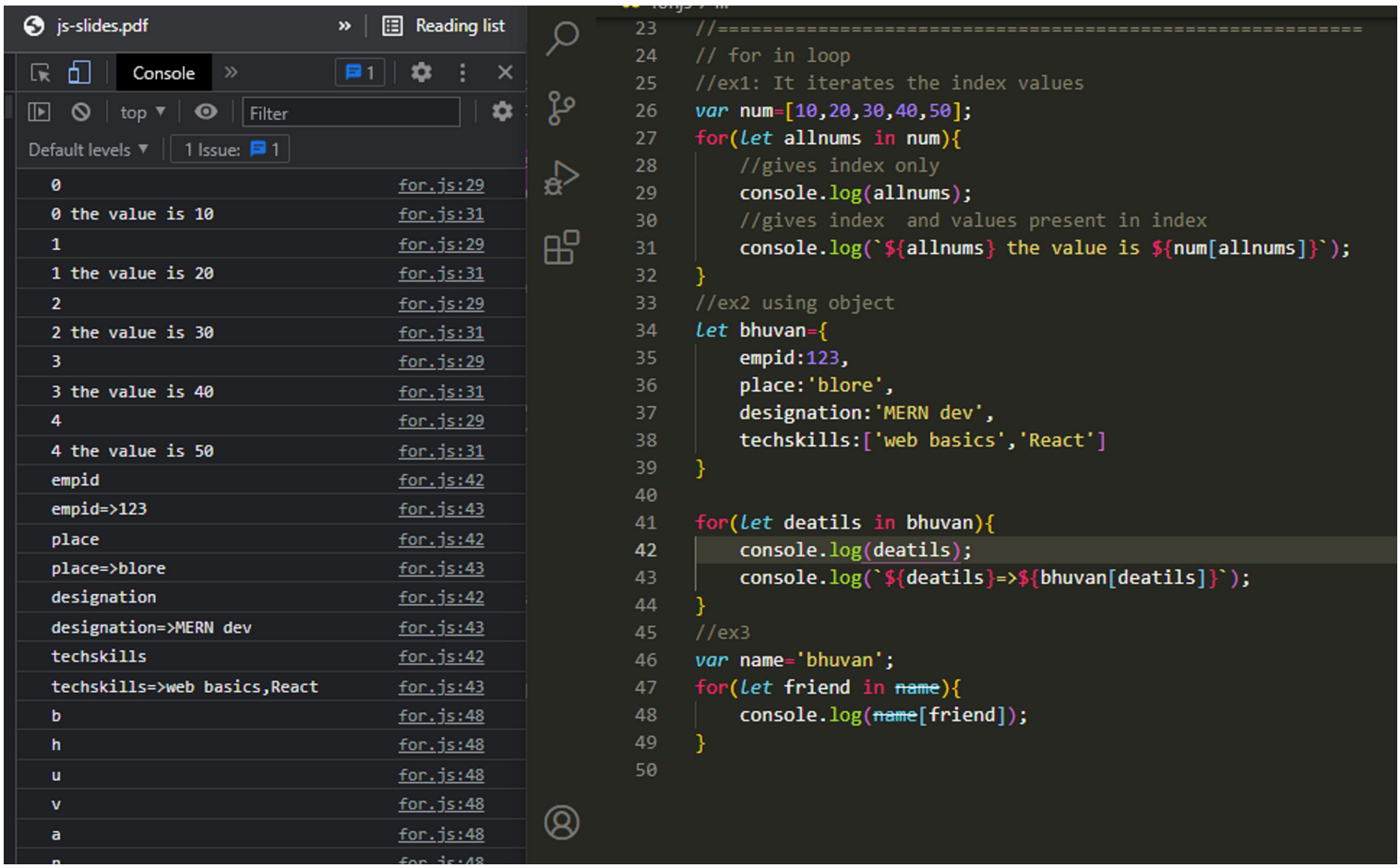
for in loop

Note:
In each iteration of the loop, a key is assigned to the key variable.
The loop continues for all object properties.

Iterate through an object

for … in with objects, when you use arrays you get only index
for … of with Arrays,strings, you will get type error if you use for objects

```
// for in loop
//ex1: It iterates the index values
var num=[10,20,30,40,50];
for(let allnums in num){
    //gives index only
    console.log(allnums);
    //gives index  and values present in index
    console.log(`${allnums} the value is ${num[allnums]}`);
}
//ex2 using object
let bhuvan={
    empid:123,
    place:'blore',
    designation:'MERN dev',
    techskills:['web basics','React']
}
for(let deatils in bhuvan){
    console.log(details);
    console.log(`${deatils}=>${bhuvan[deatils]}`);
}
//ex3
var name='bhuvan';
for(let friend in name){
    console.log(name[friend]);
}
```



1. WAP to iterate all the values in a array of numbers
2. Create an array which contain heterogenous data in it and iterate using using for of loop
3. Create an object of employee where the employee object has employee id , salary and name. update the salary of the employee from rs to dollar and iterate using for in loop
4. Write down the difference between for in and for of loop

for loop	forEach	for of	for in
Does not work with object	Does not work with object, only use with arrays	Does not work with object	Works with object and arrays
Does not ignore empty elements	Ignores empty elements	Does not ignore empty elements	Ignores empty elements
break statement is supported	break statement is not supported coz it's a method	break statement is supported	break statement is supported
Ignores extra properties which does not have index	Ignores extra properties which does not have index	Ignores extra properties which does not have index	Does not ignore extra properties which does not have index
Performance 3	Performance 4	Performance 1	Performance 2

Object method

Saturday, January 8, 2022 10:15 AM

A function which is a member of an object is called as object method.
A function defined inside an object is known as object method or method.
Encapsulation: binding of state and behaviour of an object together is known as encapsulation.

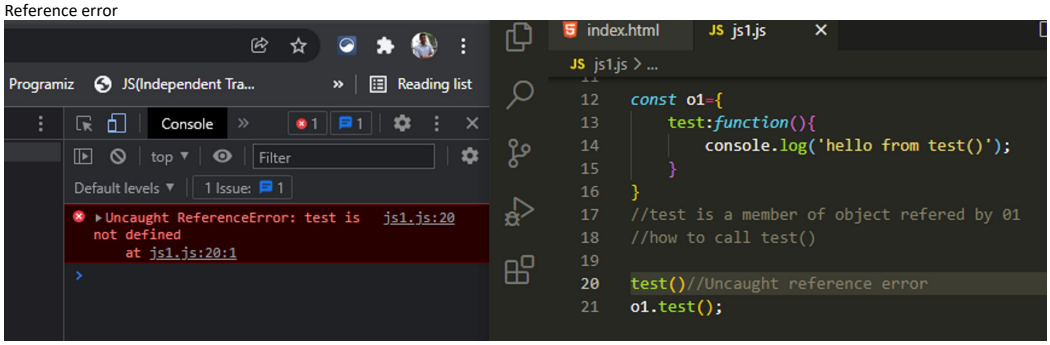
Syntax:	{
	Key: function( ) { }
	}

Syntax to call an object mehod:	Object_reference . Method_call( )

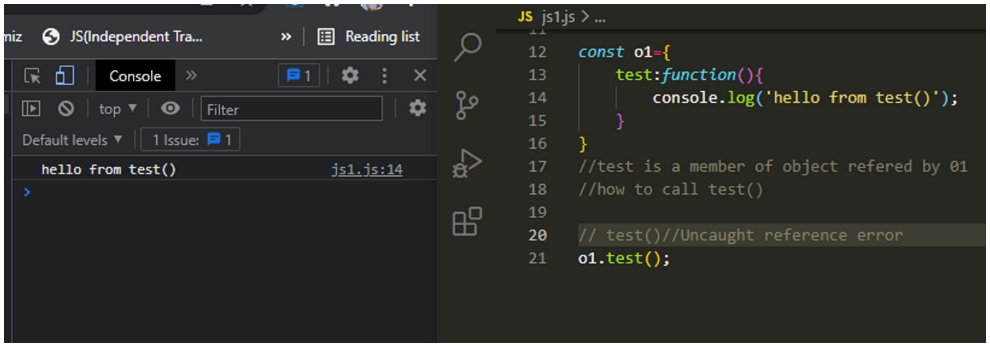
Example1

```
const o1={
    test:function(){
        console.log('hello from test()');
    }
}
//test is a member of object refered by 01
//how to call test()
// test()//Uncaught reference error
o1.test();
```

Reference error



Working condition



Using arrow function

```
const o1={
    demo:()=>console.log('arrow method')
}
o1.demo();
```

Using IIFE

```
const o1={
    demo:(()=>console.log('arrow method'))()
}
o1.demo;
```

```
/*
this:
1. it is a keyword.
2.it is a variable.
3. in GEC it holds the address of window object
4.it is a local variable of every function in js, and holds the address of window object
5. inside object methods this holds the reference of current object.
 */
```

Examples to demonstrate this keyword reference inside and outside objects

```
function m2(){
    console.log(this);// this holds the address of window object
}
m2();
```

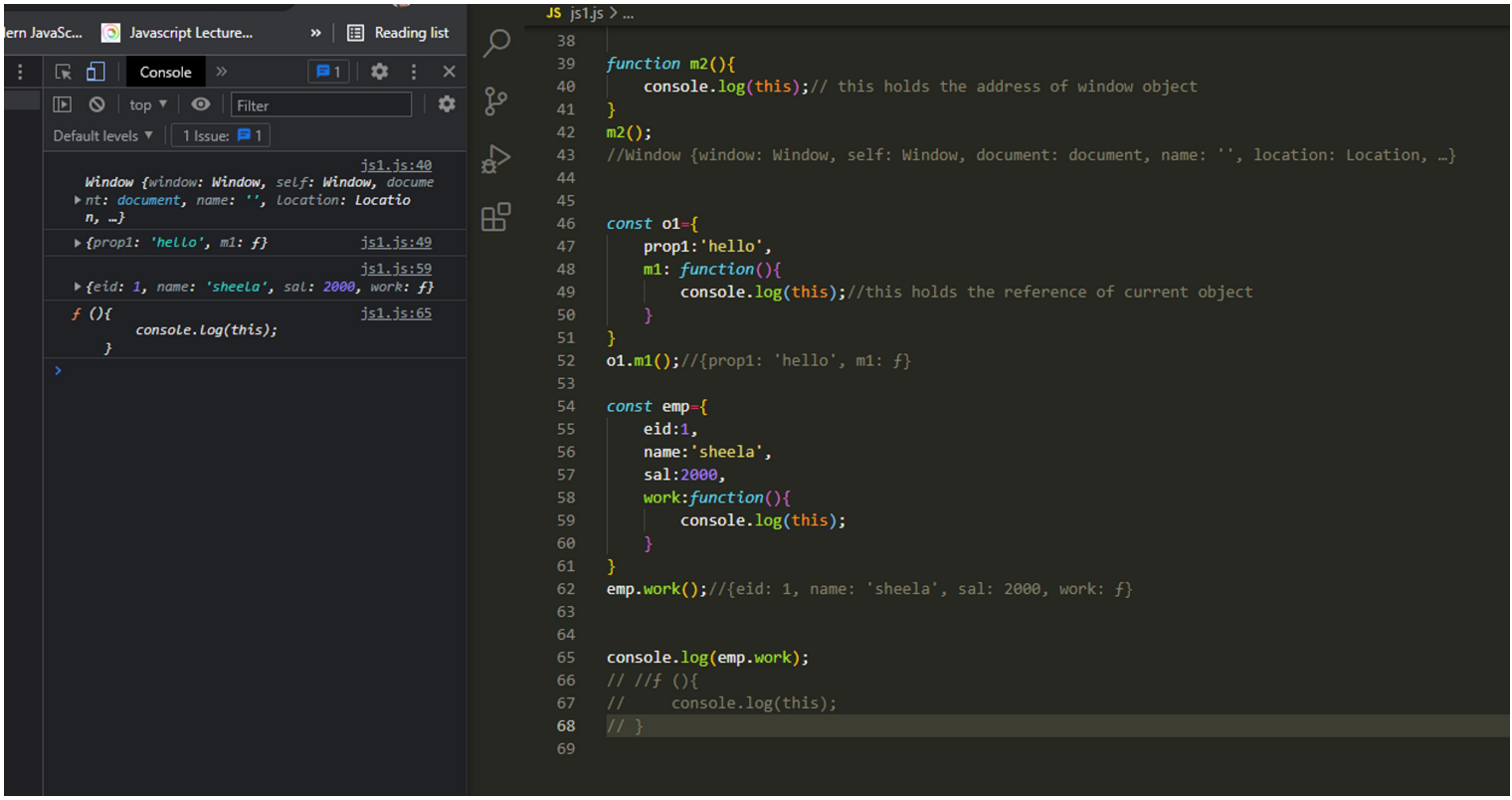
```
//Window {window: Window, self: Window, document: document, name: '', location: Location, …}


const o1={
    prop1:'hello',
    m1: function(){
        console.log(this);//this holds the reference of current object
    }
}
o1.m1();//{prop1: 'hello', m1: f}


const emp={
    eid:1,
    name:'sheela',
    sal:2000,
    work:function(){
        console.log(this);//this holds the reference of current object
    }
}

emp.work();//{eid: 1, name: 'sheela', sal: 2000, work: f}


console.log(emp.work);
// //f (){
//     console.log(this);
// }
```



Important note

1. When the object method is created using an arrow function, then this holds the reference of Global Window object and not the current object.

Example	

	const o3={
	demo:()=>{
	console.log(this);//this refers to global object or window object and not current object
	}
	}
	o3.demo();

To access members of the object inside the objects method

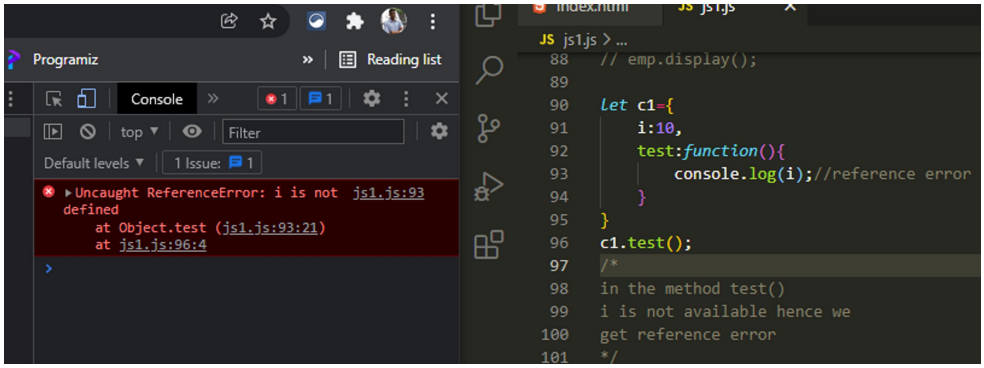
```
const emp={
    eid:1,
    ename:'sheela',
    sal:2000,
    //object method called display()---> display state of the object
    display: function(){
        //access he properties of current object emp
        //display name of emp
        console.log(this.ename);
    }
```

```
}
emp.display();
```

We can use members of an object inside the object method only with the help of this keyword.
If we try to access without this keyword it will look for the local variable of the method, if no variable is present then we get Reference error

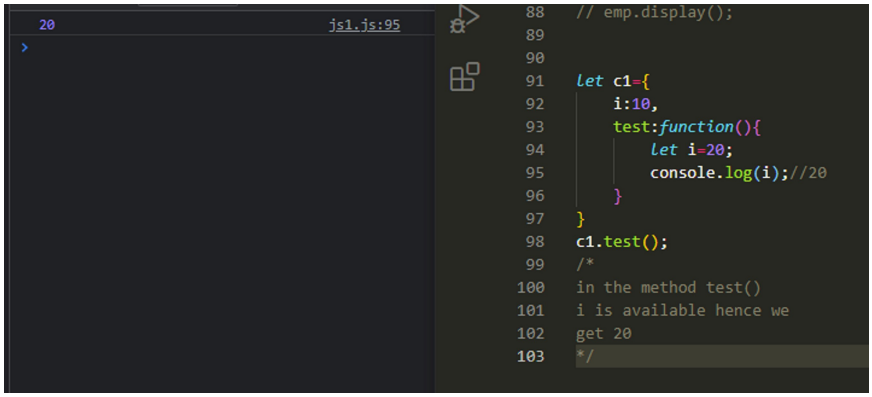
Case 1:

```
let c1={
    i:10,
    test:function(){
        console.log(i);//reference error
    }
}
c1.test();
/*
in the method test()
i is not available hence we
get reference error
*/
```



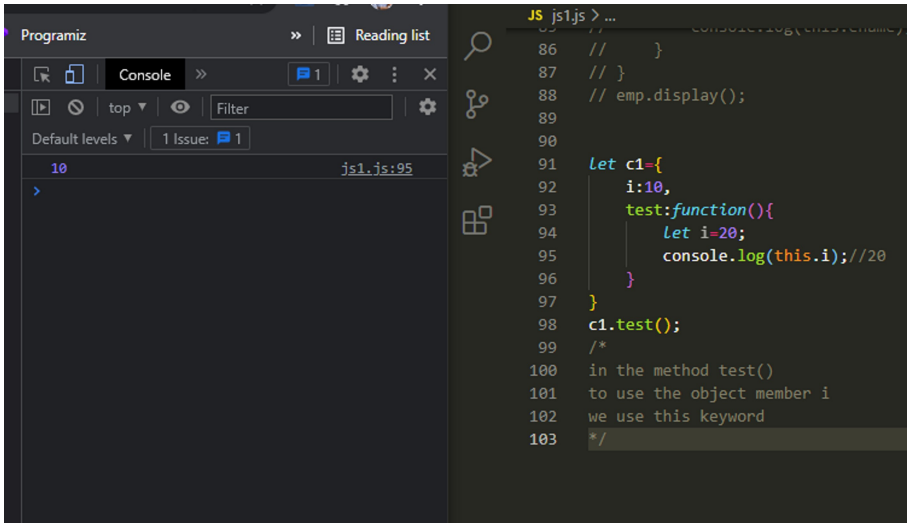
Case 2:

```
let c1={
    i:10,
    test:function(){
        let i=20;
        console.log(i);//20
    }
}
c1.test();
/*
in the method test()
i is available hence we
get 20
*/
```



Case 3:

```
let c1={
    i:10,
    test:function(){
        let i=20;
        console.log(this.i);//20
    }
}
c1.test();
/*
in the method test()
to use the object member i
we use this keyword
*/
```



If a function is not nested the lexicalscope of the function is always the window object

```
let i=100;
let c1={
    i:10,
        test:function(){
            let i=20;
            console.log(i);//20
            console.log(this.i);//10
            console.log(window.i);//undefined
        }
    }
    c1.test();
```

```
var i=100;
let c1={
    i:10,
    test:function(){
        let i=20;
        console.log(i);//20
        console.log(this.i);//10
        console.log(window.i);//100
    }
}
c1.test();
```

Not window.i is undefined as the it is declared using let and not var

AC example

```
let ac={
    cur_temp:24,
    max_temp:28,
    min_temp:18,
    increaseTemp:function(){
        this.cur_temp++;
    },
    decreaseTemp:function(){
        this.cur_temp--;
    },
    displayCurTemp:function(){
        return this.cur_temp;
    }
}
console.log(ac.displayCurTemp());//24
ac.increaseTemp();//25
ac.increaseTemp();//26
console.log(ac.displayCurTemp());//26
ac.decreaseTemp();//25
console.log(ac.displayCurTemp());//25
```

After noon session

```
const car = {
  make: "Honda",
  engine: {
    no_of_cylinder: 4,
    cc: 1200,
    displayEngine: function () {
      return `engine[cylinders: ${this.no_of_cylinder}\ncc:${this.cc}]`;
    },
  },
  displayCar: function () {
    return `make: ${this.make}\n ${this.engine.displayEngine()}`
  },
};
//details of engine
let engine_data=car.engine.displayEngine();
console.log(engine_data);
//deatils of car
let car_data=car.displayCar();
console.log(car_data);
```

Assignment: add a new function to the car object which can accept a new Engine object.

Constructor function

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

Syntax:	function identifier (parameter ,. . .)
	{
	}

Note:

1. if the function is designed to use a constructor then the name of the function should always be in upper camel case. (convention)
2. The list of parameters provided to the function, will be treated as the keys of the object (properties of the object).
3. The arguments passed when the function is called will be values of the object.
4. We can copy the values into the keys of the object from parameter using this keyword.
5. We can create an object using constructor function with the help of new keyword

Syntax	New Function( );

Note: new creates the object and returns its reference.

Used since ES6
class was intoduced in ES15

```
//define a constructor function for student
function student(sid, name){
    this.sid=sid,
    this.name=name;
}
let s1=new student(1,'a');
let s2=new student(2,'b');
let s3=new student(3,'c');
console.log(s1.sid);
console.log(s2.sid);
console.log(s3.sid);
```

Task1. Assume a book object, it has Bid, title and price using contructor function
Create a constructor function
Create atleast three book objects

==========

```
function Emp(eid,name,sal){
    this.eid=eid;
    this.name=name;
    this.sal=sal;
    this.display=function(){
        console.log(this.eid);
        console.log(this.name);
        console.log(this.sal);
    }
}
let e1=new Emp(1,'sheela',2000);
e1.display();
```

examples:
person and car object

Ways to create an Object:
1. using object literal
2. constructor function
3. using class in (ES 11)

Constructor function:

---

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

syntax:
function identifier(parameters,...){

}

note:
1. if the function is designed to use a constructor then the name of the function should always be in UpperCamelCase.(Convention)

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

3. The Arguments passed when the function is called will be the values of the object.

4. we can copy the values into the keys of the object from the parameters using this keyword.

5. we can create object using the constructor function with the help of new keyword.

syntax:
new Function();

note: new creates the object and returns its reference.

ex:

function Student(sid, name) {
this.sid = sid;
this.name = name;

}

let s1 = new Student(1, 'Anil');
let s2 = new Student(2, 'Bhuvi');

// console.log(s1 + '\n' + s2); //Objects has been created

console.log(`S1: Id is ${s1.sid} and the Name is ${s1.name}`);

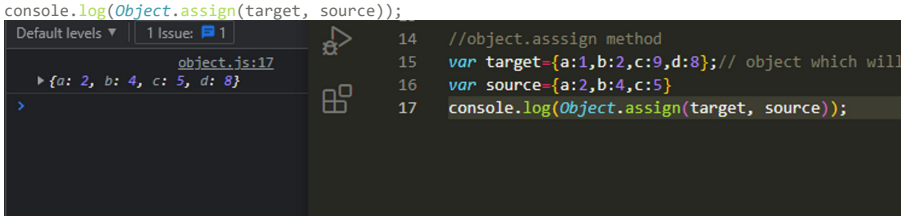
Constructor function helps us to create multiple objects of ame type.

1. Freeze method
   Freezes the object and does not let you make any changes

```
//freeze method
var obj1={
    empName:'sheela',
    age:26,
    place:'Mysore'
}
console.log(Object.freeze(obj1));
obj1.age=27;
obj1.place='blore';
console.log(obj1.age);
```

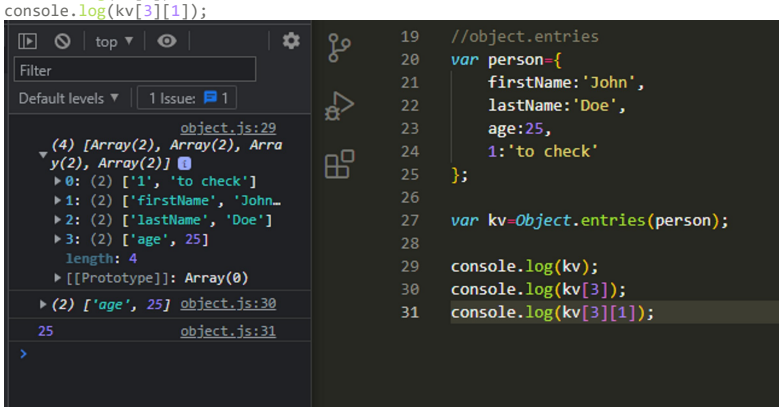
2. Assign method

```
//object.asssign method
var target={a:1,b:2,c:9,d:8};// object which will be modifed using source object
var source={a:2,b:4,c:5}
console.log(Object.assign(target, source));
```



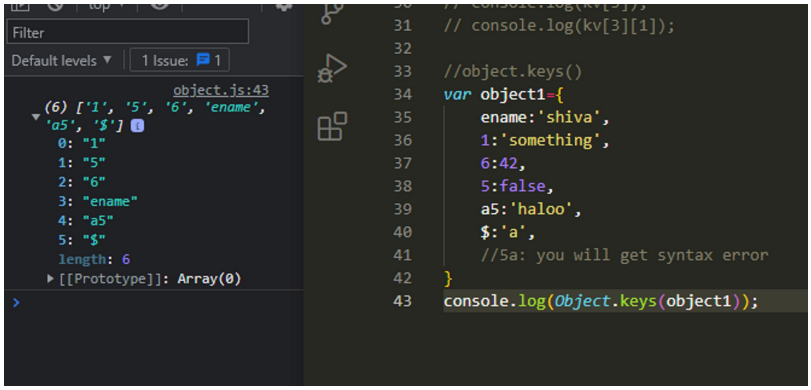
3. Enteries method

```
//object.entries
var person={
    firstName:'John',
    lastName:'Doe',
    age:25,
    1:'to check'
};
var kv=Object.entries(person);
console.log(kv);
console.log(kv[3]);
console.log(kv[3][1]);
```



4. Keys method

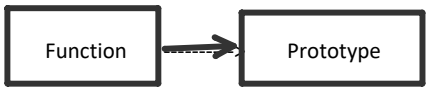
```
//object.keys()
var object1={
    ename:'shiva',
    1:'something',
    6:42,
    5:false,
    a5:'haloo',
    $:'a',
    //5a: you will get syntax error
}
console.log(Object.keys(object1));
```

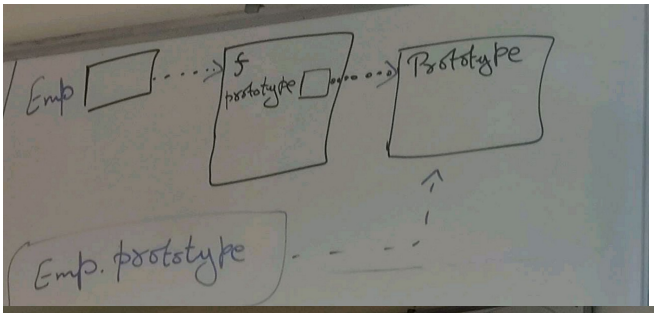

Filter
Default levels
1 Issue: 1
object.js:43
(6) ['1', '5', '6', 'ename', 'a5', '$']
0: "1"
1: "5"
2: "6"
3: "ename"
4: "a5"
5: "$"
length: 6
[[Prototype]]: Array(0)
// console.log(kv[3][1]);
//object.keys()
var object1={
ename:'shiva',
1:'something',
6:42,
5:false,
a5:'haloo',
$:'a',
//5a: you will get syntax error
}
console.log(Object.keys(object1));

prototype

Monday, January 10, 2022 3:29 PM

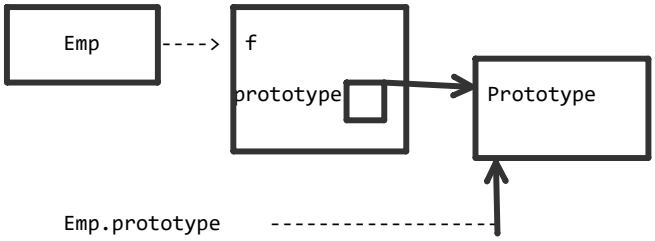
Every function in JS will be associated with an object called as Prototype.
A function object has a attribute called prototype which holds the reference of the prototype object.

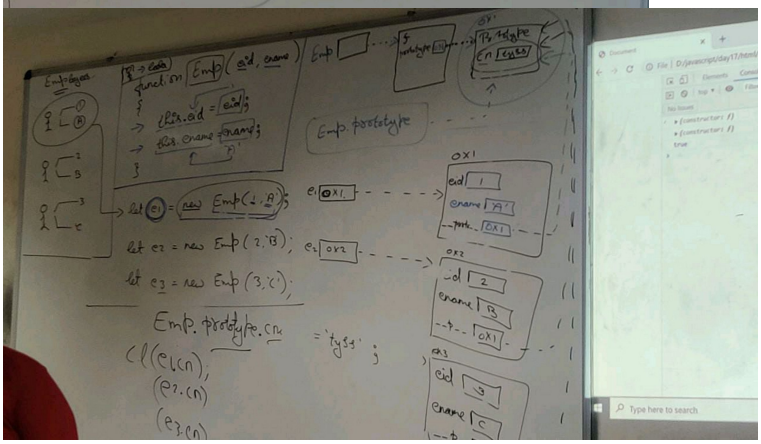




Example:

```
function Emp(eid, name){
    this.eid=eid;
    this.name=name;
}
```

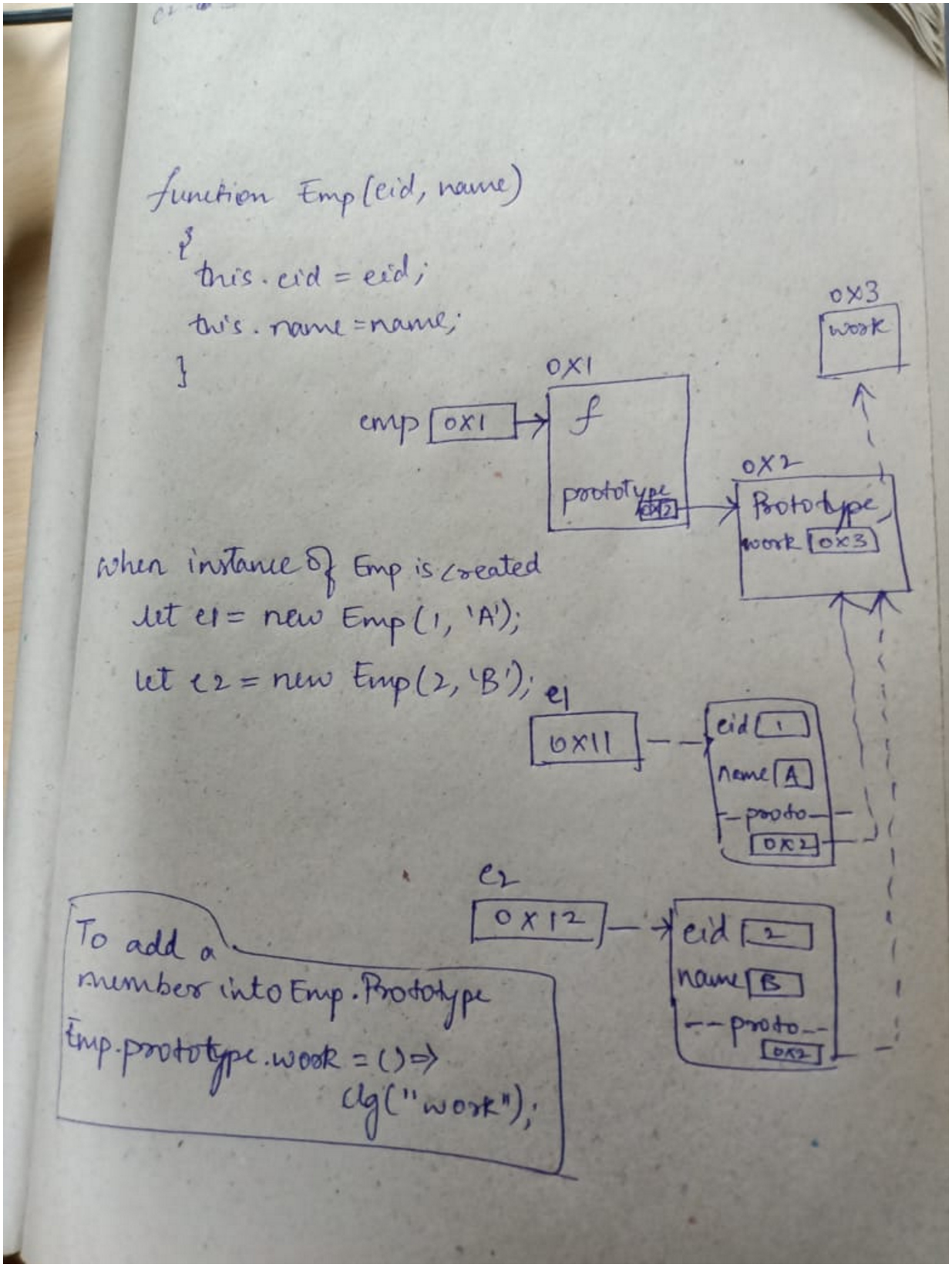




```
function Emp(eid, name){
    this.eid=eid;
    this.name=name;
}
Emp.prototype.cn='TYSS';
let e1=new Emp(1,'A');//instance created for the object e1
let e2=new Emp(2,'B');
let e3=new Emp(3,'C');
console.log(e1.cn);//TYSS
console.log(e2.cn);//TYSS
e2.cn='google';
console.log(e2.cn);//google
console.log(e3.cn);//TYSS
```

When an object is created a prototype object is not created instead the object wil have the reference of the prototype object. Refered using __proto__
Example:

```
function Emp(eid, name){
    this.eid=eid;
    this.name=name;
}
```



The advantage of adding a member in a prototype
We can acess reference of any object which belongs to that prototype.

Exercise
Design a function which can accept an aray and logs element in reverse order. The function should be aded as a member of aray so that we can call the function using the aray object reference.

```
let a=[10,20,30,40];
Array.prototype.rev=(a)=>{console.log(a.reverse());};
a.rev(a);// [40, 30, 20, 10]
```

```
let c=[10,5,7,9]
c.rev(c);//[9, 7, 5, 10]
```

Every prototype will have a reference to object's prototype.

Object

In JS we have a predefined type called Object

```
console.log(Object.prototype);
```

```
1. {constructor: ƒ, __defineGetter__: ƒ, __defineSetter__: ƒ, hasOwnProperty: ƒ, __lookupGetter__: ƒ, …}
   1. constructor: ƒ Object()
   2. hasOwnProperty: ƒ hasOwnProperty()
   3. isPrototypeOf: ƒ isPrototypeOf()
   4. propertyIsEnumerable: ƒ propertyIsEnumerable()
   5. toLocaleString: ƒ toLocaleString()
   6. toString: ƒ toString()
   7. valueOf: ƒ valueOf()
   8. __defineGetter__: ƒ __defineGetter__()
   9. __defineSetter__: ƒ __defineSetter__()
  10. __lookupGetter__: ƒ __lookupGetter__()
  11. __lookupSetter__: ƒ __lookupSetter__()
  12. __proto__: (...)
  13. get __proto__: ƒ __proto__()
  14. set __proto__: ƒ __proto__()
```

Property	Description
constructor	Returns a function that created instance.
__proto__	This is invisible property of an object. It returns prototype object of a function to which it links to.

Method	Description
hasOwnProperty()	Returns a boolean indicating whether an object contains the specified property as a direct property of that object and not inherited through the prototype chain.
isPrototypeOf()	Returns a boolean indication whether the specified object is in the prototype chain of the object this method is called upon.
propertyIsEnumerable()	Returns a boolean that indicates whether the specified property is enumerable or not.
toLocaleString()	Returns string in local format.
toString()	Returns string.
valueOf	Returns the primitive value of the specified object.

Chrome and Firefox denotes object's prototype as `__proto__` which is public link whereas internally it reference as [[Prototype]]. Internet Explorer does not include `__proto__`. Only IE 11 includes it.

The `getPrototypeOf()` method is standardize since ECMAScript 5 and is available since IE 9.

Prototypal Inheritance

In JS, inheritance is achieved using prototype hence it is known as Prototypal inheritance.
Ex:

```
function Person(pid, pname){
    this.pid=pid;
    this.pname=pname;
}
function Student(sid){
```

```
        this.sid=sid;
    }
    let p1=new Person(1,'abb');
    let s1=new Student(5);
    s1.__proto__=p1;
```

Protoype chain of P1

P1-->Person-->Object

Prototype chain of S1

S1-->P1-->Person-->Object

S1 can be used to access members of Person as it is present in the prototype chain of S1

note

When an object is created using a literal then that object __proto__ refers to prototype of Object.

Prototype is an object created for every constructor function.

Class

Class hoisting is not possible. You can use a class only after it is being declared.

Data structures-Map-Set

Monday, January 10, 2022 10:17 AM

1. Stack->LIFO (push,pop, iempty)
2. Queue-> FIFO (dequeue, enqueue, size)
3. Linkedlist-> single linked list, doubly linked list
4. Set
5. Map
6. Graph
7. Tree

Ma'am dictates
Datastructures: it is a technique for storing and organising data that make it easier to modify, navigate, access.
Datastructures determine how data is collected, the functions we can use to access it and the relationships between the data.
Few different data structures are stack, queue, linkedlist, set map graph and tree.

Set

It is an object which Is used to store unique values or data.
With respect to set key and values are the same
Set is an object with a collection of unique values.
It can of any type.
Duplicates are not allowed.

Syntax:

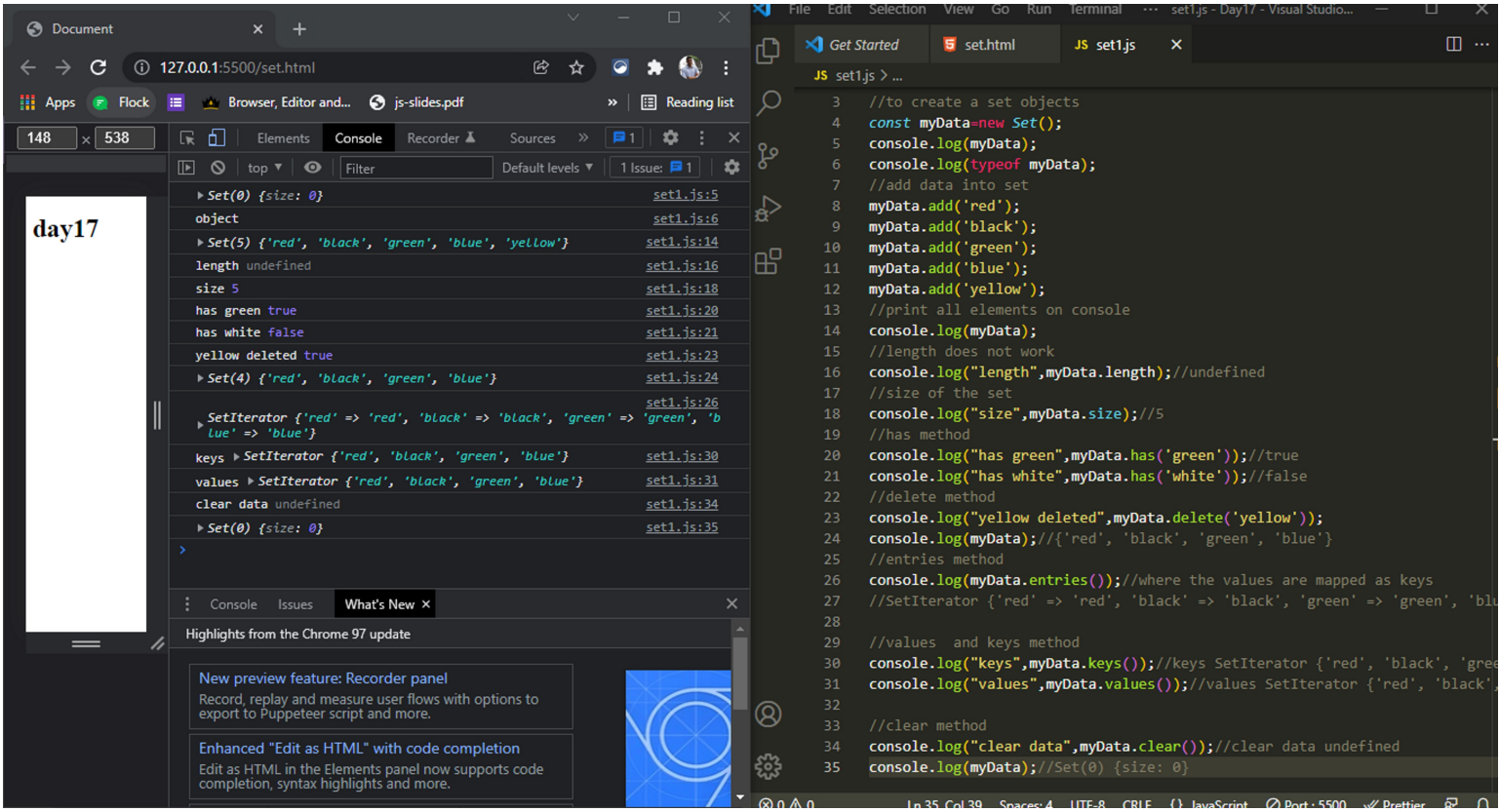
Constructor function to build set object	new Set ([iterable])

Methods for set

add(value);	Helps to insert the data
has(value);	Checks the data
delete(value);	Deletes a particular data, if you do not pass any value you will get false by default.
entries( );	It iterates the set iterator, where the values are mapped as keys
clear( );	It clears all elements in set object
foreach( );	
size	It is a property which returns the size of the set object

Code snippet

```
//set
//to create a set objects
const myData=new Set();
console.log(myData);
console.log(typeof myData);
//add data into set
myData.add('red');
myData.add('black');
myData.add('green');
myData.add('blue');
myData.add('yellow');
//print all elements on console
console.log(myData);
//length does not work
console.log("length",myData.length);//undefined
//size of the set
console.log("size",myData.size);//5
//has method
console.log("has green",myData.has('green'));//true
console.log("has white",myData.has('white'));//false
//delete method
console.log("yellow deleted",myData.delete('yellow'));
console.log(myData);//{'red', 'black', 'green', 'blue'}
//entries method
console.log(myData.entries());//where the values are mapped as keys
//SetIterator {'red' => 'red', 'black' => 'black', 'green' => 'green', 'blue' => 'blue'}
//values  and keys method
console.log("keys",myData.keys());//keys SetIterator {'red', 'black', 'green', 'blue'}
console.log("values",myData.values());//values SetIterator {'red', 'black', 'green', 'blue'}
//clear method
console.log("clear data",myData.clear());//clear data undefined
console.log(myData);//Set(0) {size: 0}
```




```
const allNums=new Set([num]);//passing as a single array so size will be on


//ex1
const num=[10,20,30,50,10,22,20];
const allNums=new Set(num);
console.log(typeof num);//object
console.log("size",allNums.size);//5
//using for each
allNums.forEach((val,i)=>console.log(`keys ${i}=> values ${val}`));
```

Create an array of technology stacks which contains mean mern pern mven jfs pfs

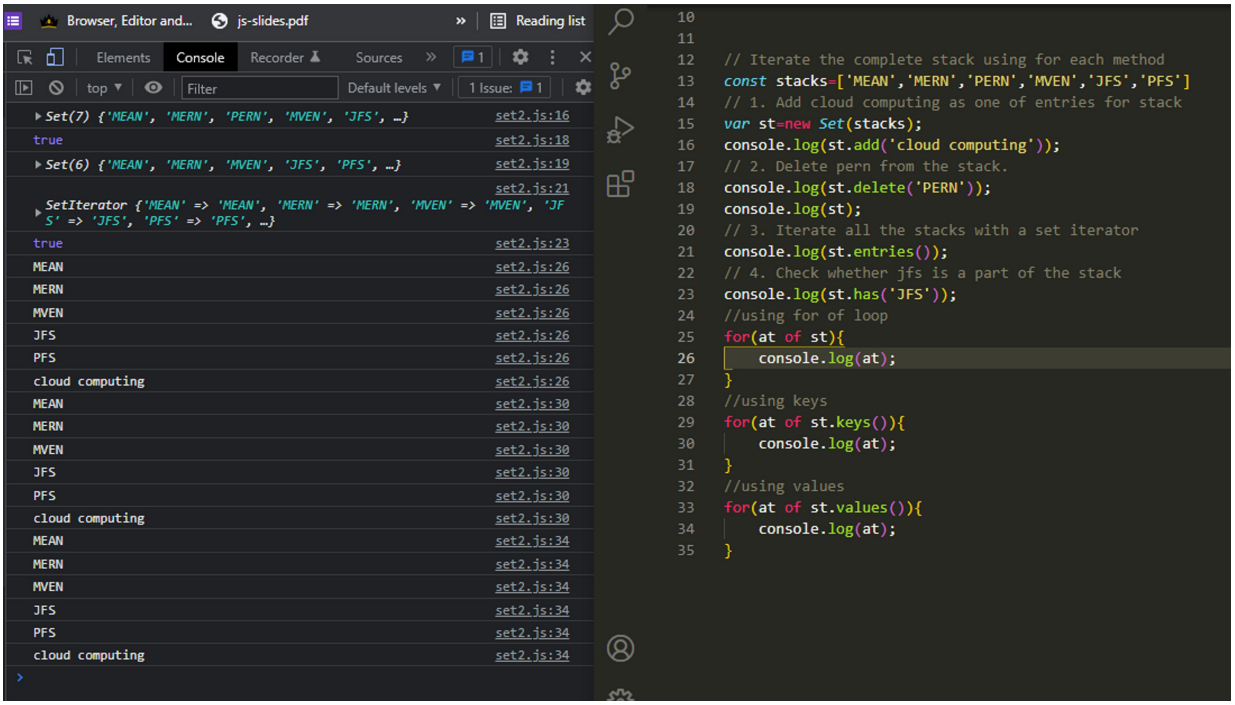
1. Add cloud computing as one of entries for stack
2. Delete pern from the stack.
3. Iterate all the stacks with a set iterator
4. Check whether jfs is a part of the stack
5. Iterate the complete stack using for each method

```
// Iterate the complete stack using for each method
const stacks=['MEAN','MERN','PERN','MVEN','JFS','PFS']
// 1. Add cloud computing as one of entries for stack
var st=new Set(stacks);
console.log(st.add('cloud computing'));
// 2. Delete pern from the stack.
console.log(st.delete('PERN'));
console.log(st);
// 3. Iterate all the stacks with a set iterator
console.log(st.entries());
// 4. Check whether jfs is a part of the stack
console.log(st.has('JFS'));
//using for of loop
for(at of st){
    console.log(at);
}
```

How to iterate using for of loop

```
        //using for of loop
        for(at of st){//case1
            console.log(at);
        }
        //using keys
        for(at of st.keys()){//case2
            console.log(at);
        }
        //using values
        for(at of st.values()){//case3
            console.log(at);
        }
```

Here at is just a variable it cannot use at.values( ) or .keys( ), at is not a Set or object it is just a variable.
Instead use one of the three ways listed above




```
// conversion of set to array
const stacks=['MEAN','MERN','PERN','MVEN','JFS','PFS']
var st=new Set(stacks);
console.log(typeof st);
console.log(st);
let arr=[...stacks];
console.log(typeof arr);
console.log(arr);
```

Write a program to remove duplicates values form an array

```
var arr = [10, 20, 30, 10, 20, 30];
let b = (function a(arr) {
  return [...new Set(arr)];
})(arr);
console.log(b);
```

weakSet( );

- It holds a weak reference to values.
- Can contain only objects.
- It uses only add, has, remove methods
- Values cannot be accessed.

Difference between set and weakset()

Map

It is collection of keyed values.
Key and value pairs can be of any type.
Insertion order is mainted
Keys can be of any type, functions, an object , array etc

Ma'am dictates

- It is collection of keyed values
- It holds key and value pairs.
- Key and value pairs can have same values.
- Key in the collection are distinct or unique and may only occur in one key or value pair within the maps collection.

Difference between map and object

Map	Object
In a map any value can be used as a key or a value.	Only strings can be used as keys
Why to use maps? It can be used other than a string as a key • String->key • Function->key • Array->key • Number->key	

Key and value pairs can have same values
Maintains insertion order

Constructor function for map

Syntax	new Map( );

Methods in map

set(key, value);	
get(key);	
entries( );	

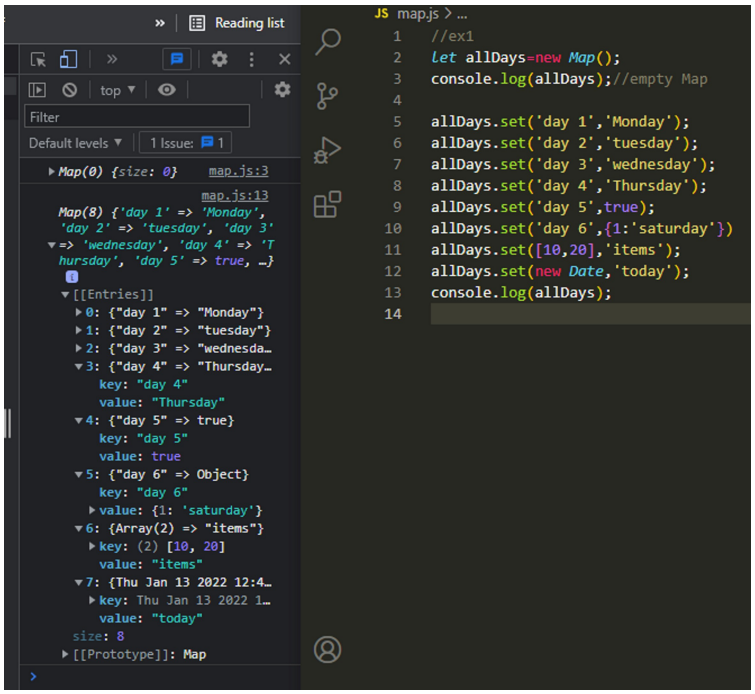
keys( );	
values( );	
clear( );	

Exampe:

1. set()

Example

```
//ex1
let allDays=new Map();
console.log(allDays);//empty Map
allDays.set('day 1','Monday');
allDays.set('day 2','tuesday');
allDays.set('day 3','wednesday');
allDays.set('day 4','Thursday');
allDays.set('day 5',true);
allDays.set('day 6',{1:'saturday'})
allDays.set([10,20],'items');
allDays.set(new Date,'today');
console.log(allDays);
```



Add code here

Note:

1. SameValueZero, which works like === strict equals to, but it considers NaN to be equal to itself.
   NaN===NaN
2. NaN can be used as a key in Maps

Example

```
allDays.set(NaN,NaN);//get(NaN);
allDays.set(NaN,123);//get(NaN);
```

3. The +0 and -0 values can be used as keys in Maps

Eg:

```
allDays.set(+0,'hi');//get(+0);
allDays.set(-0,'Bye');//get(+0);
allDays.set(0,JS);//get(NaN);
```

Assignment

1. Iterate an array of strings letters using for of and map and set
2. Create a complex array(array of objects and iterate them using set object.
3. Difference between JS object/ map object /JSON

Assignment

Tuesday, January 11, 2022 10:38 AM

Assignment questions

1. List out the rules for object keys.
2. Define array.flat() with examples.
3. Define 2D array and multidimensional array with examples.Mention the real time usage of this arrays
4. Define weak set and weak map with examples.
5. Define the web storage and discuss about the local storage, session storage, cookies, create a difference table for local storage, session storage and cookies. And check the local storage, session storage cookies in a devlopers tool.
6. Class
    a. What is class?
    b. Syntax of class creation?
    c. What Is the need of class?
    d. Class was introduced in which version of ECMA script. ES11
    e. How to instanciate a class?
    f. What is constructor? And what are its needs?
    g. How many constructor can a class have?
    h. How to create a sub class?
    i. What is the need of this in class?
7. Definition
    a. extends
    b. super
    c. this
    d. Define inheritance in class?
    e. Realtime examples for class and inheritance.

Class

Thursday, January 13, 2022 10:44 AM

Ma'am ppt explanation

A class is a type of function, but instead of using a keyword function to initiate it, we use the keyword class, and the properties are assigned inside a constructor( ) method.

Class Definition:

- Use the keyword class to create a class and always add the constructor( ) method.
- The constructor method is called each time the class object is initialized.
-

Syntax	class class_name{
	constructor( ) {
	//properties declaration and intialization
	} //end of constructor
	}//end of class

Methods

- The constructor method is special, it is where you initialize properties, it is called automatically when a class is initiated and it has to have the exact name "constructor", in fact, if you do not have a constructor method, JS will add an invisble empty constructor method.
-

Example

```
class Car{
    constructor(brand){
        this.carname=brand;
    }
    present(){
        return "i have a "+this.carname;
    }
}
var myCar=new Car("ford");//object creation
console.log(myCar.carname);//accessing the data members
console.log(myCar.present());//accessing the method
```

This

- The JS this keyword refers to the object it belongs to .
  - It has different values depending on where it is used.

- In a method, this refers to the owner object.
- Alone, this refers to the global object.
- In a function, this refers to the global object.
- In an event , this refers to the element that received the event.

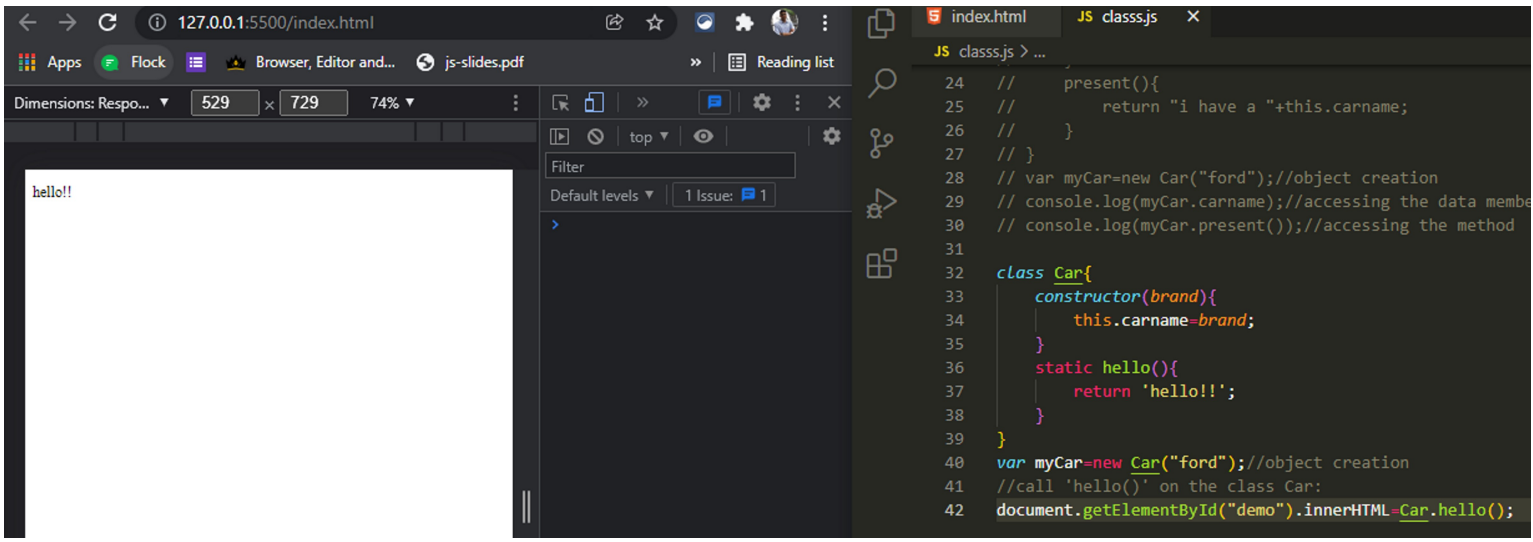
Static Methods

- Static methods are defined in the class itself, and not on the protoype. That means you cannot call a static method on the object, but on the classname.
-

Example

```
class Car{
    constructor(brand){
        this.carname=brand;
    }
    static hello(){
        return 'hello!!';
    }
}
var myCar=new Car("ford");//object creation
//call 'hello()' on the class Car:
document.getElementById("demo").innerHTML=Car.hello();
```

```
<body>
   <p id="demo">
   </p>
   <script src="./classs.js"></script>
 </body>
```



Inheritance

To create a class inheritance, use the extends keyword.

A class created with a class inheritance

Define getters and setters with example.
Rules of return keyword
Apart from whatsapp write two more examples

DOM

Tuesday, January 11, 2022 2:43 PM

Html file has two parts or Document has

1. Content to be displayed
2. Structure or elements

Once a page is already rendered by the browser and want to some modifications we use Document.
Document helps us to develop dynamic applications.

Document is an object

Life span of the Dom manipulation is untill the session ends or file reloads's/
DOM helps to modify web content

DOM- Document object model
Sir dictates

Document:

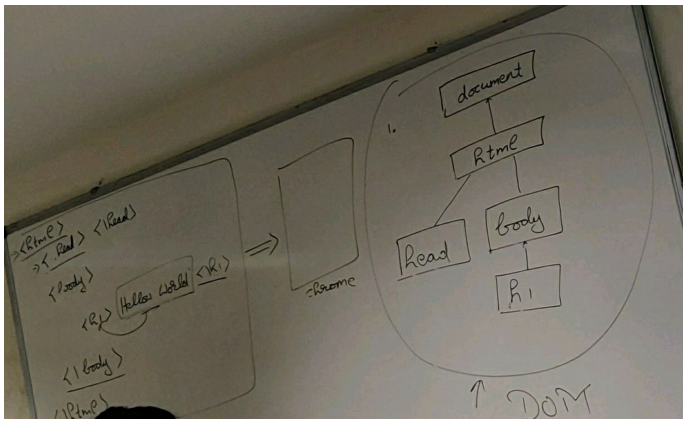
A document is an object created by Browser. The Document object is the root node of DOM tree.

DOM:

Every HTML element is considered as a node (JavaScript object) in DOM. DOM allows to modify the document content rendered by the browser without reloading the page.

Therefore DOM helps to make a page Dynamic.
Any modification done using DOM, is not updated to the original html page.
Therefore once we reload a page, all the modifications done using DOM will be lost .

Example:	<html> <head> </head> <body> Hello world </body> </html>



We can write content on the browser's page dynamically with the help of write and writeln method of Document object.
Eg: to display a message on the browser page from the javascript code.

```
document.write('hello');
```

Note:

1. DOM is an API provided by the browser.
2. DOM is not JS, it is built using JS.
3. DOM is a Object Oriented representation of an HTML file.
4. Everytime a HTML Page is given to a browser. It automatically generates the DOM tree which can be accessed and modified using the root node document in JS.

Note: this is not a dynamic page
Form events generate a brand new page upon click

```
<form action="#" name='f1'>
        Name:<input type="text" id="name" placeholder="enter name">
        <input type="submit" id="submit" onclick="test()">
    </form>


    function test() {
      document.write("submit click");
```

```
    }
```

getElementById();

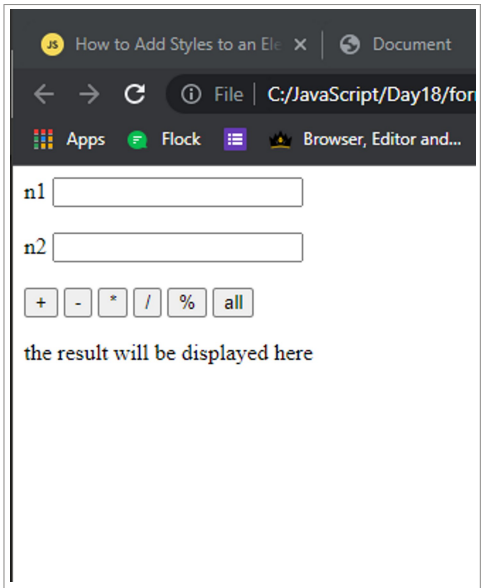
```
        function onPress(){
            document.write(document.getElementById('input1').value);
        }
```

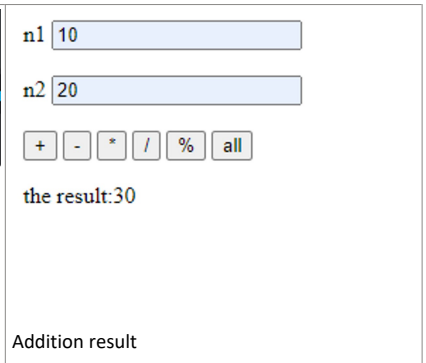
Assignment
calculator

```
<body>
      <script src="./form1.js"></script>
    n1 <input type="text" id="n1"><br><br>
    n2 <input type="text" id="n2"><br><br>
    <input type="submit" id="add" value="+" onclick="add()">
    <input type="submit" id="sub" value="-" onclick="sub()">
    <input type="submit" id="mul" value="*" onclick="mul()">
    <input type="submit" id="div" value="/" onclick="div()">
    <input type="submit" id="mod" value="%" onclick="div()">
    <!-- <input type="submit" id="all" value="all"
onclick="all1()"> -->
    <p id="p1">the result will be displayed here</p>
</body>
```

```
function opontwo(cb) {
  let n1 = Number(document.getElementById("n1").value);
  let n2 = Number(document.getElementById("n2").value);
  let res = cb(n1, n2);
  // console.log(res);
  document.getElementById("p1").textContent = ` the
result:${res}`;
}
function add() {
  opontwo((a, b) => a + b);
}
function sub() {
  opontwo((a, b) => a - b);
}
function mul() {
  opontwo((a, b) => a * b);
}
function div() {
  opontwo((a, b) => a / b);
}
function mod() {
  opontwo((a, b) => a * b);
}
```

Output





Addition result

==========

Next day

Left to right association
Top to bottom order

Draw the Dom tree here
To obtain the elements from DOM
The methods of DOM objects

1. getElementById( )
    i. For this method we need to pass the Id of an element as a String it returns the first element with a specified Id.
    ii.

Eg: fetching an element by using an Id	`let e=document.getElementById('heading');` `console.log(e);`

2. getElementsByName( )
    i. Fetching an element by using attribute name
    ii. We need to pass name of the element as a String
    iii. It returns a NodeList [] which is similar to an array but not an Array
    iv. If there is no any Element with given name then it returns an empty NodeList
    v.

Eg:	`let e=document.getElementsByName('heading');` `console.log(e);`

3. To fetch an element from DOM using classname- getElementsByClassName( )
    a. Pass the class name as a string
    b. It returns a HTMLCollection containing all the elements matching the given class name
    C.

Eg:	`let e=document.getElementsByClassName('divstyle1');` `console.log(e);//HTMLCollection(2) [div#d1.divstyle1, div#d2.divstyle1, d1: div#d1.divstyle1, d2: div#d2.divstyle1]`

4. To fetch the elements from the DOM using tagname-getElementsByTagName( )
    i. We need to pass the tagname as a String.
    ii. It returns an HTMLCollection containing all the elements matching the given Tag name.
    iii.

Eg:to count no of tags in the code	//html
	`<input type="button" id="b1" value="div" onclick="countP()">`
	//JS
	`function countP(){`
	`  let e=document.getElementsByTagName('div');`
	`  alert(`there are ${e.length} div tags in the page`)`
	`}`

To fetch the elements from DOM using CSS selectors.
We can select an element from DOM using CSS selector with the help of

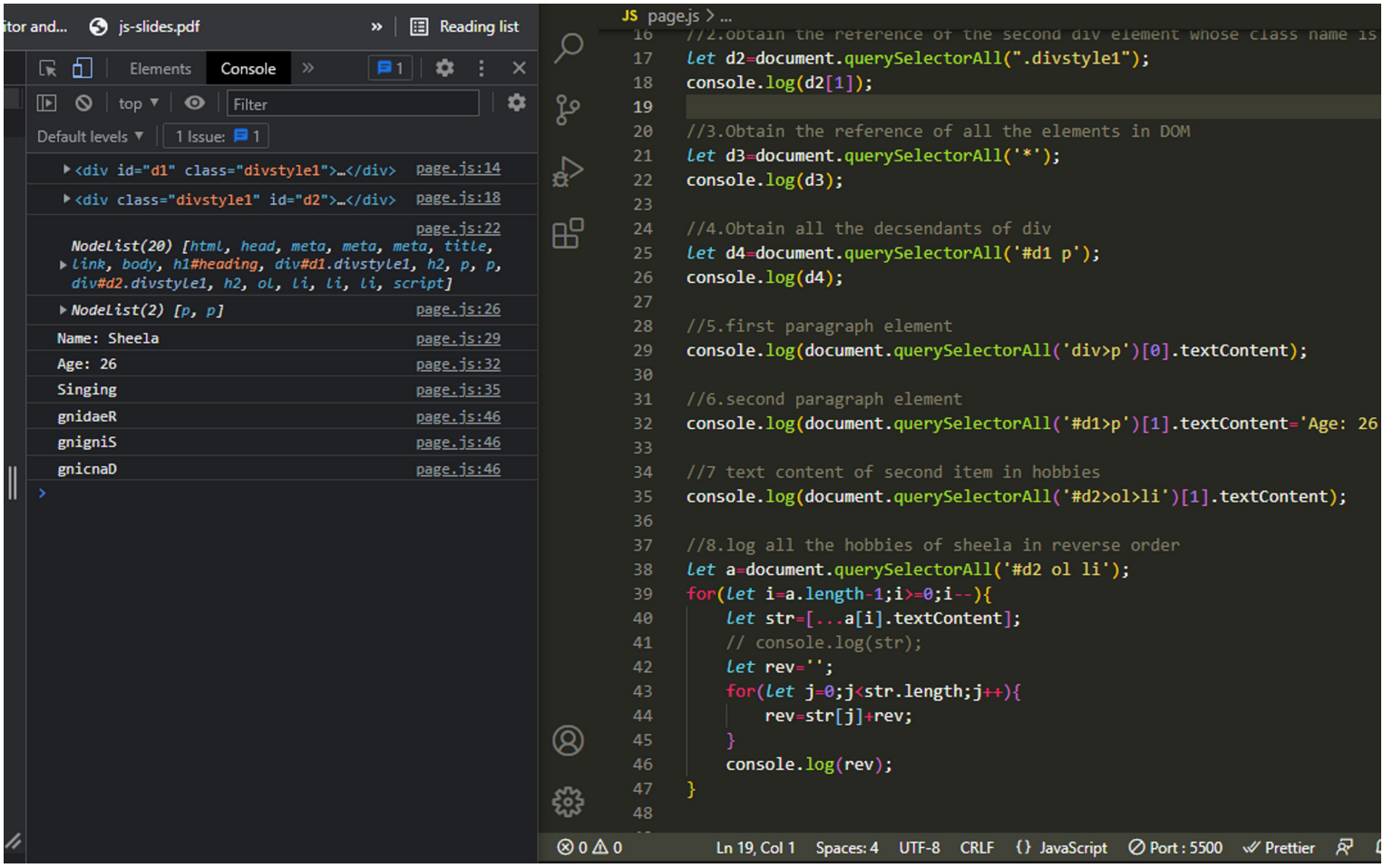
1. querySelectorAll( )
2. query Selector( )

```
let e1=document.querySelector('.divstyle1');
console.log(e1);
```

Note:

1. We need to pass CSSS selector as a String.
2. querySelector returns the reference of first element found.
3. querSeletorAll( ) returns a NodeList of all the elements found in the original order.

```
//1.obtain the reference of div tag whose id is d1 using querySelector
let d1=document.querySelector('#d1');
console.log(d1);
//2.obtain the reference of the second div element whose class name is div style 1
let d2=document.querySelectorAll(".divstyle1");
console.log(d2[1]);
//3.Obtain the reference of all the elements in DOM
let d3=document.querySelectorAll('*');
console.log(d3);
//4.Obtain all the decsendants of div
let d4=document.querySelectorAll('#d1 p');
console.log(d4);
//5.first paragraph element
console.log(document.querySelectorAll('div>p')[0].textContent);
//6.second paragraph element
console.log(document.querySelectorAll('#d1>p')[1].textContent='Age: 26');
//7 text content of second item in hobbies
console.log(document.querySelectorAll('#d2>ol>li')[1].textContent);
//8.log all the hobbies of sheela in reverse order
let a=document.querySelectorAll('#d2 ol li');
for(let i=a.length-1;i>=0;i--){
    let str=[...a[i].textContent];
    // console.log(str);
    let rev='';
    for(let j=0;j<str.length;j++){
        rev=str[j]+rev;
    }
    console.log(rev);
}
```



Properties of DOM objects which return the DOM object reference

1. firstChild:
   It is a property, it gives the reference of the first child node of the target node.

Syntax	target_node.firstChild

   Note: the first child can be anything either a text, comment or an element.

2. firstElementChild:
   It is property, it gives the reference of the element which is the first child of the target node

Syntax	target_node.firstElementChild

   It can return only the element object.

3. lastChild:
   It is property it gives the reference of last child object of the target node.

Syntax	target_node.lastChild

   It can return either a text, comment or element object.

4. lastElementChild:
   It returns the reference of an element which is a last element of the target node.

Syntax	target_node.lastElementChild

5. Children
   It is a property, it contains list of all the elements which are Children of target node.

Syntax	target_node.children

6. childNodes:
   It will give you all the child nodes including text and comments.

7. nextSibling:
   It will give you the reference of an node which is a sibling present in the immediate right of the target node.
   It includes text and comments.

8. nextElementSibling:
   It will give you the reference of an element which is a sibling present in the immediate right of the target element.

9. parentElement:
   It will give you the parent element reference of the current element.

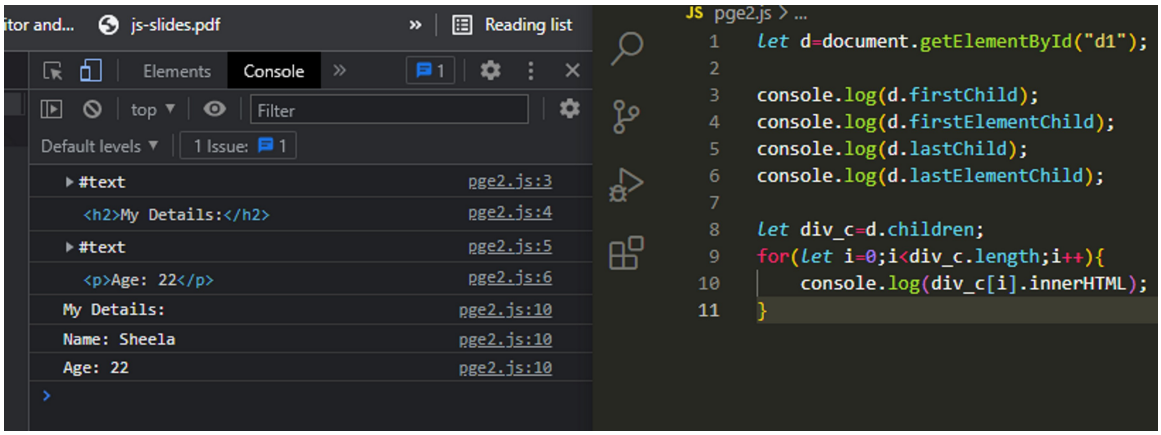
firstChild	
firstElementChild	
lastChild	
lastElementChild	
Children	
childNodes	
nextSibling	
nextElementSibling	
parentElement	
Document.all	Returns Html collection- for each does not work
queryselectorall	Returns Node list- for each works

Examples

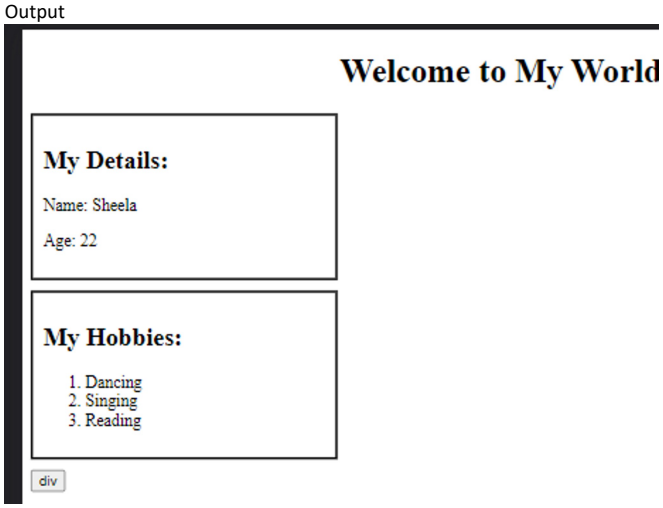
```
let d=document.getElementById("d1");
console.log(d.firstChild);
console.log(d.firstElementChild);
console.log(d.lastChild);
console.log(d.lastElementChild);
let div_c=d.children;
for(let i=0;i<div_c.length;i++){
    console.log(div_c[i].innerHTML);
}
let e2=d.nextSibling;
console.log(e2);
console.log(d.nextElementSibling);
console.log(d.parentElement);
let h=document.getElementById('d1');
console.log(h);
```

```
.divstyle1{
    border:2px solid black;
    margin-bottom: 10px;
    width:30%;
    padding:10px;
}
#heading{
    text-align: center;
}
```

```
<body>
    <h1 id="heading">Welcome to My World</h1>
    <div id="d1" class="divstyle1">
        <h2>My Details:</h2>
        <p>Name: Sheela</p>
        <p>Age: 22</p>
    </div>
    <div class="divstyle1" id="d2">
        <h2>My Hobbies:</h2>
        <ol>
            <li>Dancing</li>
            <li>Singing</li>
            <li>Reading</li>
        </ol>
    </div>
    <script src="./pge2.js"></script>
    <input type="button" id="b1" value="div" onclick="countP()">
</body>
```

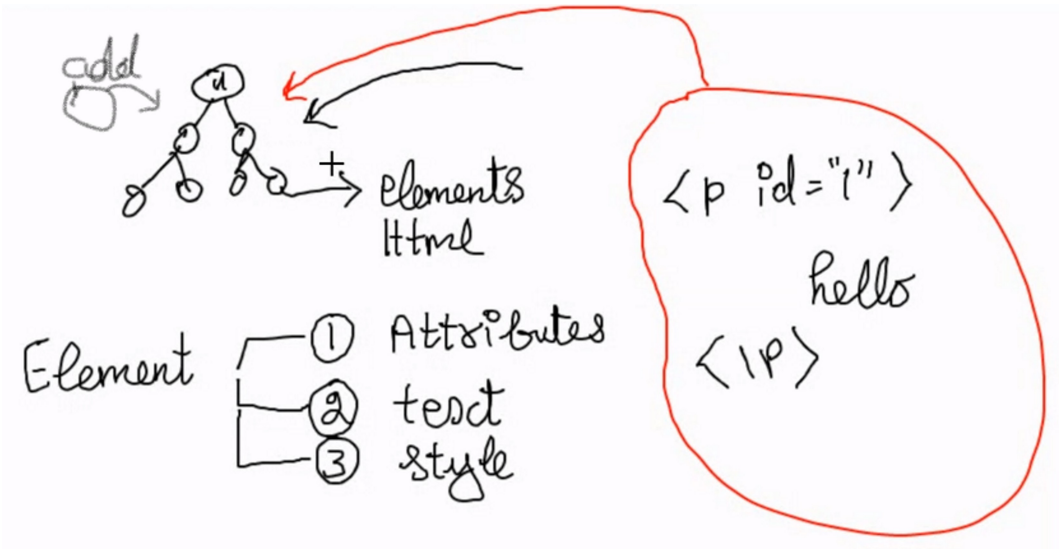


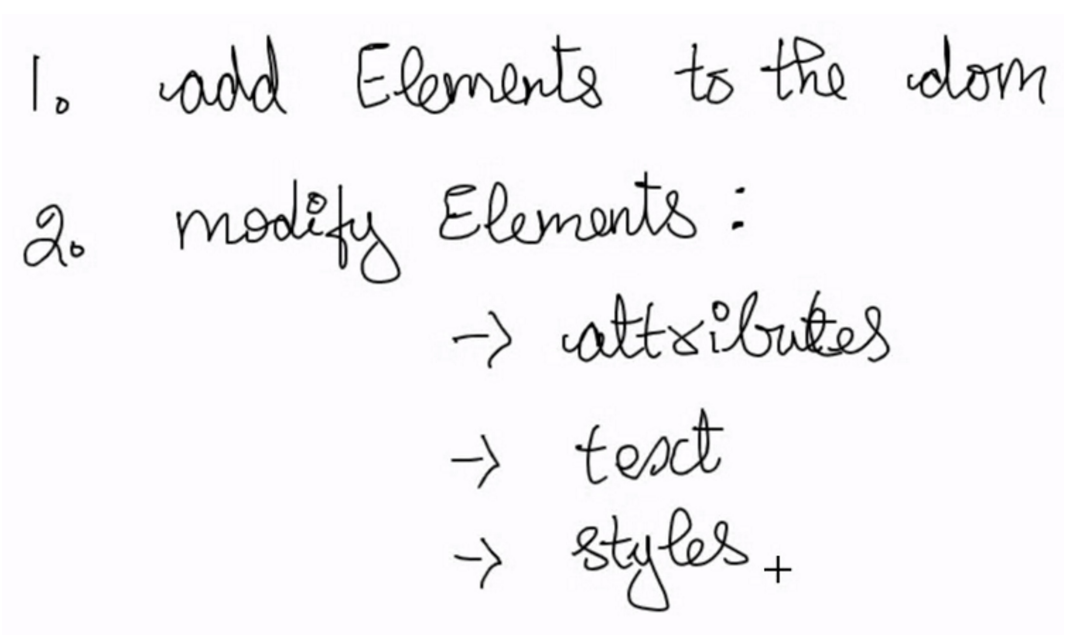
Output



==========

DOM MANIPULATION





DOM manipulations: modifying or updating the DOM tree is known as DOM Manipulation.
Two ways

1. Add elements to the DOM
2. Modify the exsisting elements in the DOM
    i. Attributes of elements
    ii. Text content of the elements
    iii. Styles of the elements

Adding elements to the DOM

We can add elemens to the DOM in the following ways

1. Using the property innerHTML
2. By creating an element using createElement( ) method of DOM objects

```
console.log(document.body);
console.log(document.body.innerHTML);
document.body.innerHTML="<h1> inserted from js</h1>";
console.log(document.body.innerHTML);
```

You will loose the old data

Adding elements to paragraph

```
    let arr=['apple','mango',"grapes"];
    for(let i=0;i<arr.length;i++){
        document.body.innerHTML+=`<p>${arr[i]}</p>`;
    }
    console.log(document.body.innerHTML);
    // ======using foreach
    let body=document.body;
    arr.forEach((s)=>{
```

```
    body.innerHTML += `<p>${s}</p>`
});
console.log(body.innerHTML);
```

InnerHTML is a property of an DOM element, the innerHTML contains everything between the html tag as a string

Example:	HTML
	<body>
	 para 1
	<script src="./DOMmani.js"></script>
	</body>
		JS
		//the adjacent html will come as output in console
		console.log(document.body.innerHTML);

We can update the property innerHTML

Syntax	Target_element.innerHTML= "string/string with html tags"	// the old content of innerHTML will be replaced with new one

//to append new content to the exsistent HTML content of innerHTML

Target_element.innerHTML += "string/string with html tags"

Example		Output on the HTML page

	console.log(document.body);
	console.log(document.body.innerHTML);
	body.innerHTML+=" para 2 ";
	console.log(document.body.innerHTML);
		Para 1
		
		Para 2

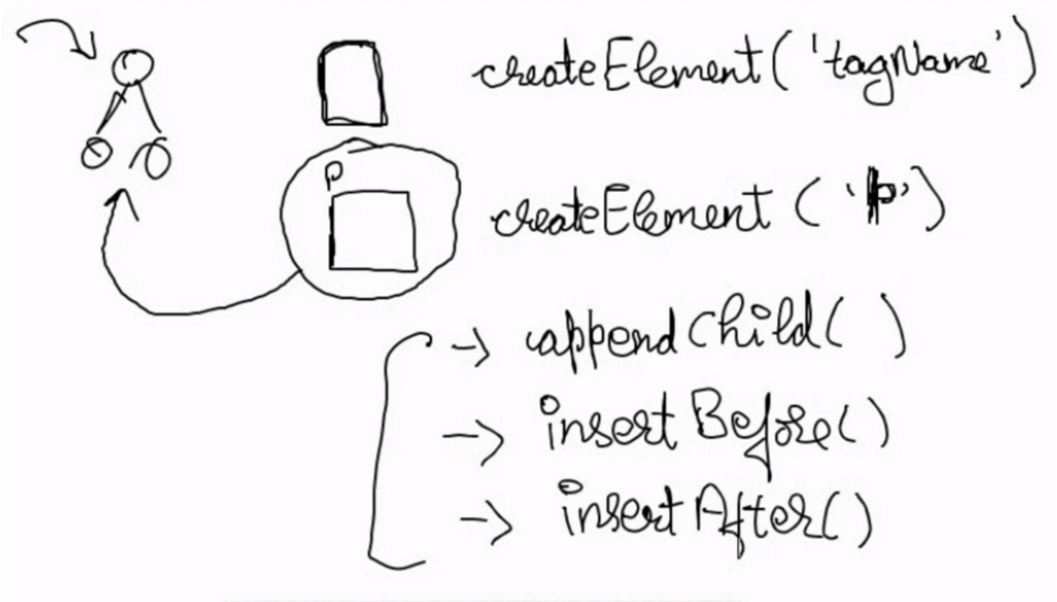
Disadvantage of using innerHTML

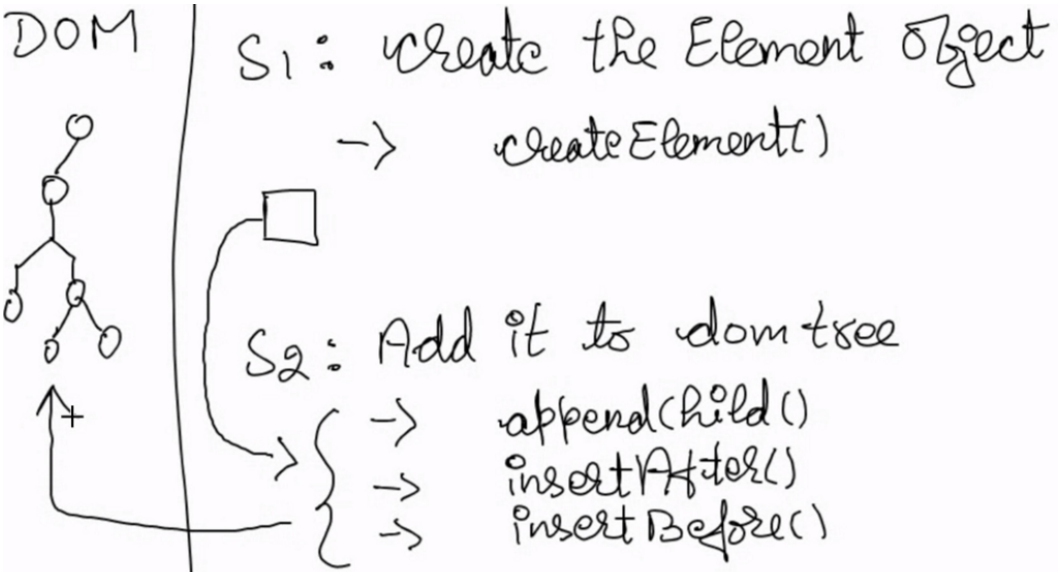
1. Security issue
2. It will reduce the efficiency of the browser, because the entire DOM tree is recreated every time innerHTML is modified.

innerHTML	textContent
Takes everything from the target node and replaces it along with the html tags and content	Takes only text part of that element
Modifies both html tags and content	Modifies only the text part of a target tag
It can accept html tags	It can't differenciate so it will print HTML tags as normal string

```
//step1 get the ref of ol element
let ol=document.getElementById("d2").lastElementChild;
moreHobbies.forEach((s)=>{
    ol.innerHTML += `<li>${s}</li>`;
})
```

Using createElement( ) method






```
// ==================================new js file
//add an extra p element to the body tag
//step1: create an element object
let e=document.createElement('p');
e.textContent="para 2";
//step 2 insert he element into dom tree
document.body.appendChild(e);//adding node to exsisting dom tree
//=======================================
let e2=document.createElement('p');
e2.textContent="para 3";
document.body.appendChild(e2);
//=======================================
let dom= document.querySelectorAll("body *");
console.log(dom);
```

Sir dictates ----------
We have two steps
Step 1. create an element object
Step 2. insert the element into the DOM

Syntax for step1:	Document.createElement('tagname');
Syntax for step2:	Target_node.appendChild(reference_of _element_object_to_be added);

Note: the given element will be inserted as a child of target_node to the right hand side.
Example:

Consider the html source

```
<body>
    <p>para 1</p>
    <script src="./DOMmani.js"></script>
</body>
```

Js file

```
let dom= document.querySelectorAll("body *");
console.log(dom);
console.log(document.querySelectorAll('body *'));
//output :
// [p, script]
//lets add another p tag to the body
//step 1:
let p= document.createElement("p");
document.body.appendChild(p);
//output :
// [p, script, p]
// note
// we can provide text, id ,name etc.. by accessing the properties of newly created element object
// to add text for newly created p EventTarget
p.textcontent=text;
// to give id
p.id="p1";

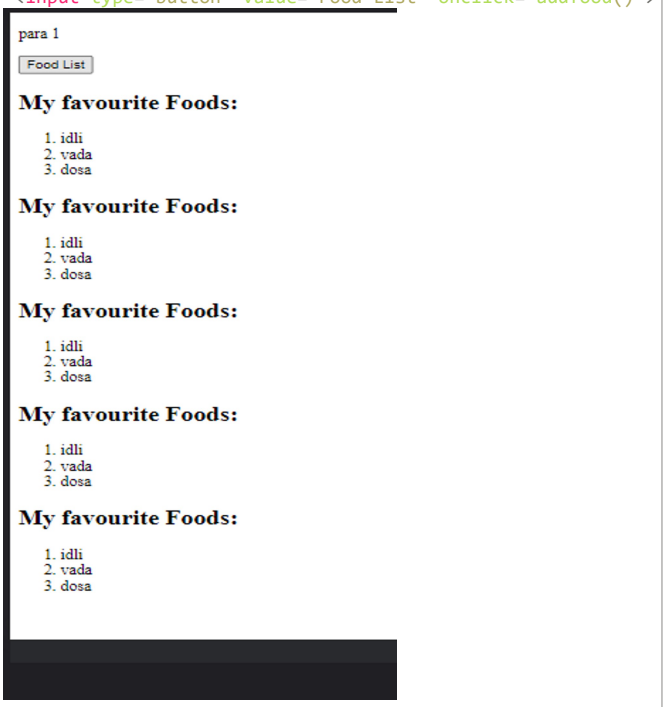
// to give class name
p.className="class name"
```

Setting attributes (same as above)

```
let e2=document.createElement('p');
e2.id="p2";//setting attributes
e2.className="para2"
e2.textContent="para 3";
document.body.appendChild(e2);
```

```
function addfood(){
//create a div tag
let div=document.createElement("div");
//add div to the body
document.body.appendChild(div);
//give class name
div.className="divstyle1";
div.id="d3";
//add h2
let h2=document.createElement("h2");
h2.textContent="My favourite Foods:"
//add h2 to div tag
div.appendChild(h2);
//add foods
let foods=['idli','vada','dosa'];
//create ol and li tags
 let ol=document.createElement("ol");
 //append li tags to ol tags
foods.forEach((food)=>{
    let li=document.createElement("li");
    li.textContent=food;
    ol.appendChild(li);
})
div.appendChild(ol);
}
```

```
<input type="button" value="Food List" onclick="addfood()">
```



insertBefore( )

Parent_node.insertBefore(node_to be _added, existing_child_node)	syntax

Example		

	`<body>`
	`<p>para 1</p>`
	`<p>para 2</p>`
	`<script src="./DOMmani.js"></script>`
	`</body>`
		`let p3=document.createElement('p');`
		`p3.textContent="para 3";`
		`let p2=document.body.firstElementChild.nextElementSibling;`
		`document.body.insertBefore(p3,p2);`

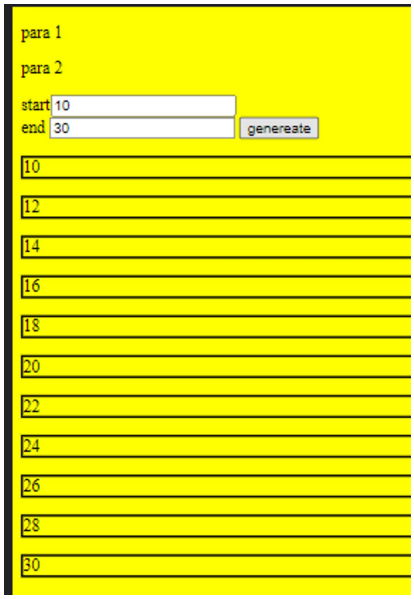
```
// ============================after lunch
//insert para 3 between p1 and p2 as a child of body
let p3=document.createElement('p');
p3.textContent="para 3";
//add it to dom
//insert before method(arg1, arg2);
//insertBefore( node to be added,existing node or last node)
let p2=document.body.firstElementChild.nextElementSibling;
document.body.insertBefore(p3,p2);
```

Task display even numbers

```
//task1
// b1.type="text";
// b1.textContent=10;
let i=document.body.lastElementChild;
function even(){
    let s=Number(document.getElementById("start").value);
    let e=Number(document.getElementById("end").value);
    for(let i=s;i<=e;i++){
        if(i%2==0){
        let p=document.createElement('p');
        p.textContent=i;
```

```
<body>
    <p>para 1</p>
    <p>para 2</p>
    start<input type="text" id="start">
    <br>
    end <input type="text" id="end">
    <input type="button" value="genereate" onclick="even()">
    <script src="./DOMmani.js"></script>
</body>
```

```
        p.style.font="25px";
        p.style.border="2px solid black";
        document.body.appendChild(p);
        }
    }
}
```



To modify or add style

Every Dom object has a property called style using which we can modify the CSS style of the element.

Syntax	target_node.style.property = "value" ;
example	Document.body.backgroundcolor =" red";

Class and class list

To modify the class of DOM elements
We can access the class attribute of an element in two ways

1. className Property
2. classList Property

className	classList
className property provides name of the class of the target element	classList provides list of all the classnames associated with the target element
	Classlist is an array type object it has built in methods to add or remove the class names `console.log(d.classList);` ▼DOMTokenList(1) app.js:16 0: "hide" length: 1 value: "hide" ▶ [[Prototype]]: DOMTokenList ▶ DOMTokenList(1) app.js:18

Task==========

```css
.show {
  border: 2px solid rgb(0, 0, 0);
  margin: 10px;
  padding: 10px;
  color: rgb(0, 0, 0);
  background-color: rgb(113, 238, 211);
}
.hide {
  display: none;
}
```

```html
<body>
    <input type="button" value="Bio" class="s1"onclick="toggle()"><br><br>
    <div class="hide" id="dis">
        name  <input type="button" id="h" value="done" onclick="hide()"><br>
        age <br>
        nati <br>
        gender
    </div>
    <br>
    <br>
    <input type="button" value="Qualification" class="s1"onclick="toggle()">
    <br><br>
    <div class="hide" id="dis">
        name  <input type="button" id="h" value="done" onclick="hide()"><br>
        age <br>
        nati <br>
        gender
    </div>
    <br>
    <br>
    <input type="button" value="Hobbies" class="s1"onclick="toggle()">
    <br><br>
    <div class="hide" id="dis">
        name  <input type="button" id="h" value="done" onclick="hide()"><br>
        age <br>
        nati <br>
        gender
    </div>
    <script src="./t1.js"></script>
</body>
```

```js
function showBio() {
  let d = document.getElementById("bio");
  //   console.log(d);
  d.className = "show";
}
function hideBio() {
  let d = document.getElementById("bio");
  d.className = "hide";
}
function showQuali() {
  let d = document.getElementById("quali");
  d.className = "show";
}
function hideQuali() {
  let d = document.getElementById("quali");
  console.log(d.classList);
  d.classList.value = "hide";
  console.log(d.classList);
}
function showHobb() {
  let d = document.getElementById("hobb");
  d.classList.value = "show";
}
function hideHobb() {
  let d = document.getElementById("hobb");
  d.className = "hide";
}
function changeName() {
  //catch value from input field
  let new_Name = document.getElementById("inName");
  console.log(new_Name.value); //caught value logged
  //to put new value in old place
  let d1 = document.getElementById("bio");
  let p = d1.firstElementChild.nextElementSibling;
  p.textContent = "name :" + new_Name.value;
  new_Name.className = "hide"; //hide text field
  new_Name.nextElementSibling.className = "hide"; //hide ok button
}
```

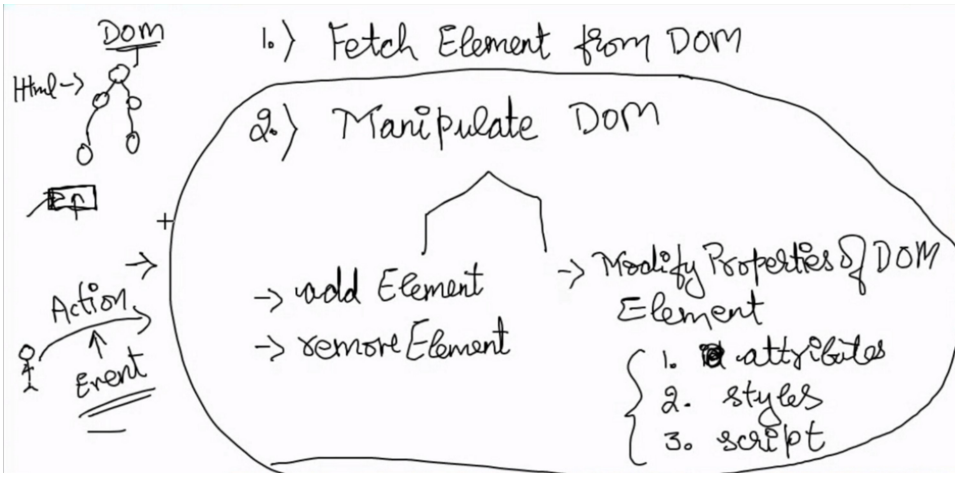
output

==========jan 15th 2022==========

Next day

removeElement

Target_element.removeElement( );

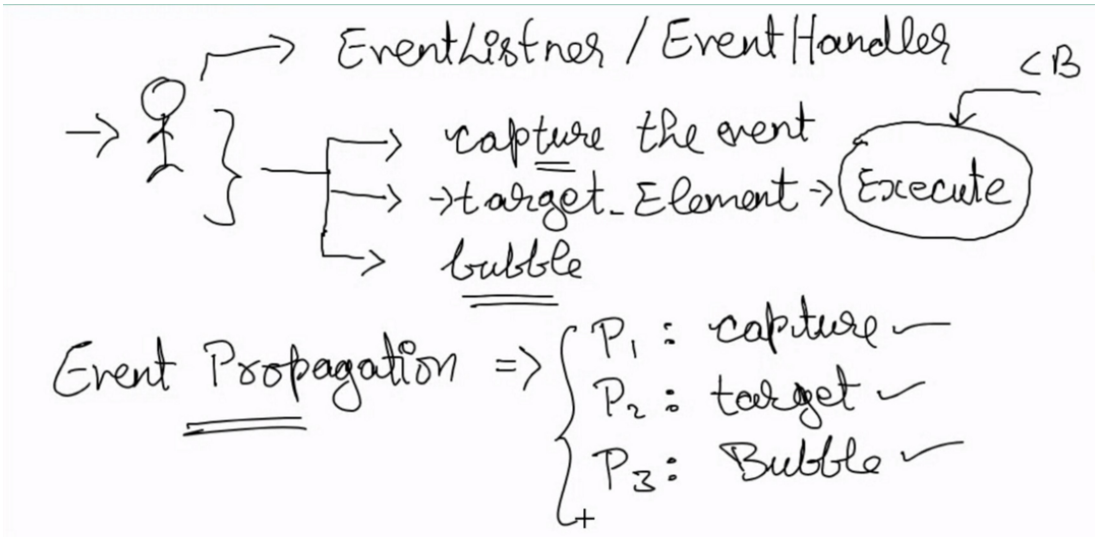


Event

Saturday, January 15, 2022 12:11 PM

an action performed by the user on the webpage is known as a Event.
We can dynamically add events to a DOM tree without adding it in the HTML file.
There are two ways

1. Js
2. Jquery



Sir dictates==========
Event and Event Listeners

1. Event is an action performed by a end user on a web page.
2. End user can perform different types of events such as mouse events, keyboard events etc.

Assignment: List all the events available.

Event Listener

Event Listener or event handler is a function which executes a task when an event is occurred on the Most specific element.

Event Flows

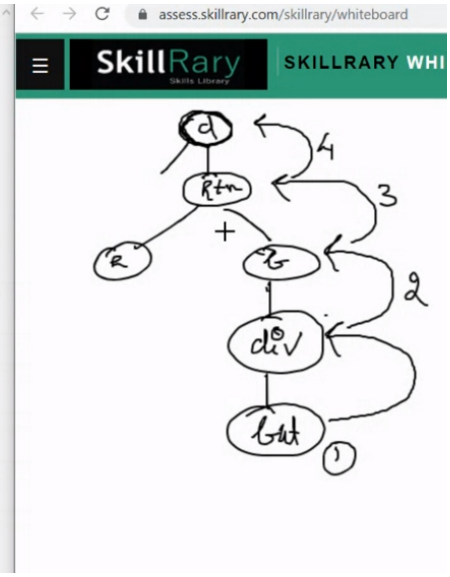
1. Event bubbling
2. Event capturing

Event bubbling

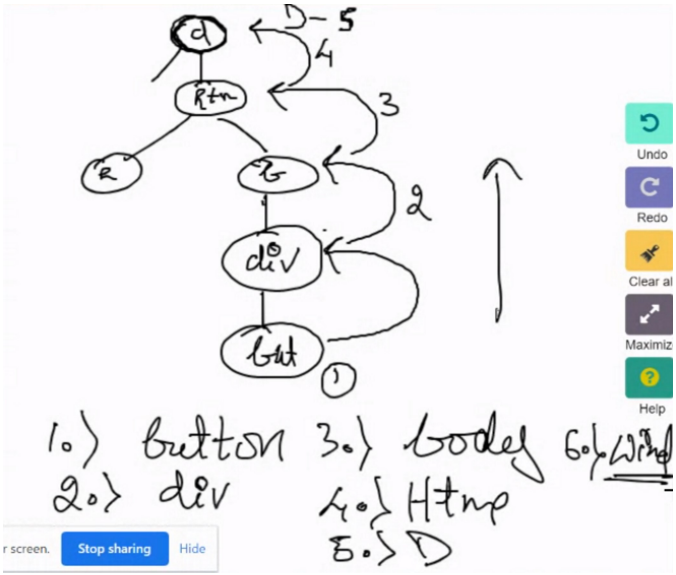
In event bubbling the event starts at the target element(most specific element) and then flows in the upward direction of the DOM tree.

Parent of document in browser is Window

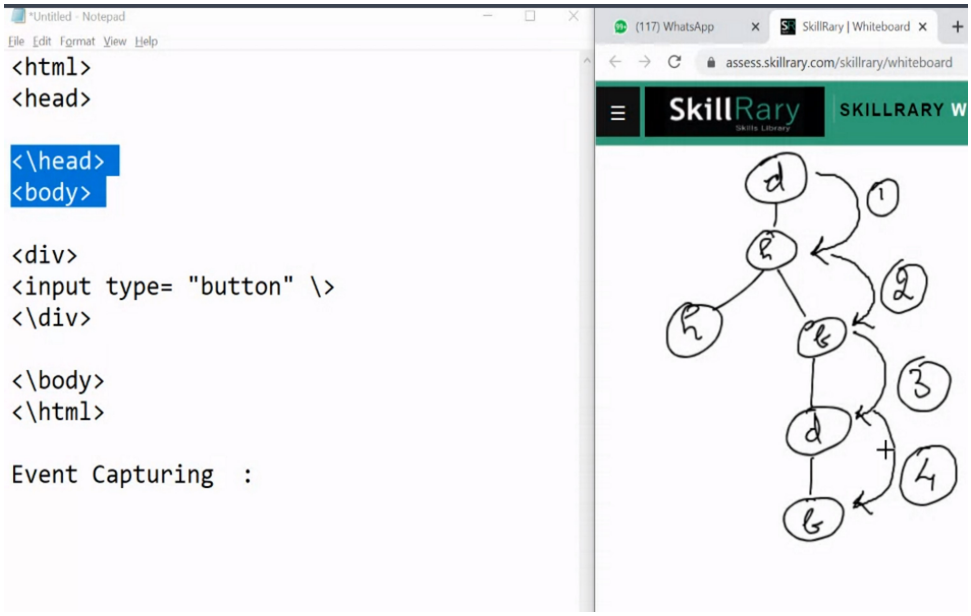
```
<html>
<head>

<\head>
<body>

<div>
<input type= "button" \>
<\div>

<\body>
<\html>
```



Event bubbling
When the event occurs on the button, the event bubbling flow is as follows
Button=>div=>body=>html=>document=>window



Event Capturing
In event capturing an event starts from the top of the DOM tree (least specific element) and flows downwards up to the target element.
Example:
When the event capturing occurs the flow is as follows
Window=>document=>html=>body=>div=>button

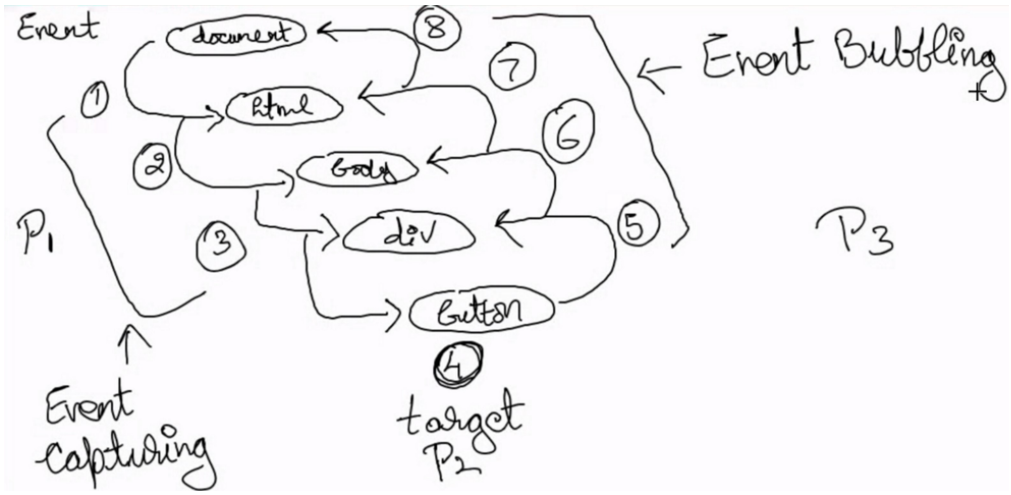


Event Propagation (event object flow)
Event Propagation has three phases
Phase1: Event capturing occurs, it provides an opportunity to intercept.
Phase 2: Actual target recieves the event.
Phase 3:Event bubbling occurs, it provides an opportunity to have a final response to the event.

Example

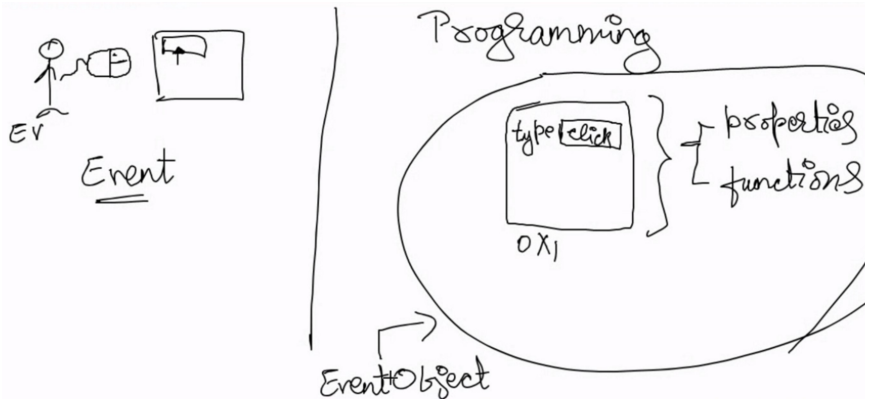
```
<html>
<head>

<\head>
<body>

<div>
<input type= "button" \>
<\div>

<\body>
<\html>
```

Note p1, p2 and p3 are phases



Event capturing happens before it reaches target.
Event Bubbling happens after it reaches target.

Event Object



Every event is an object

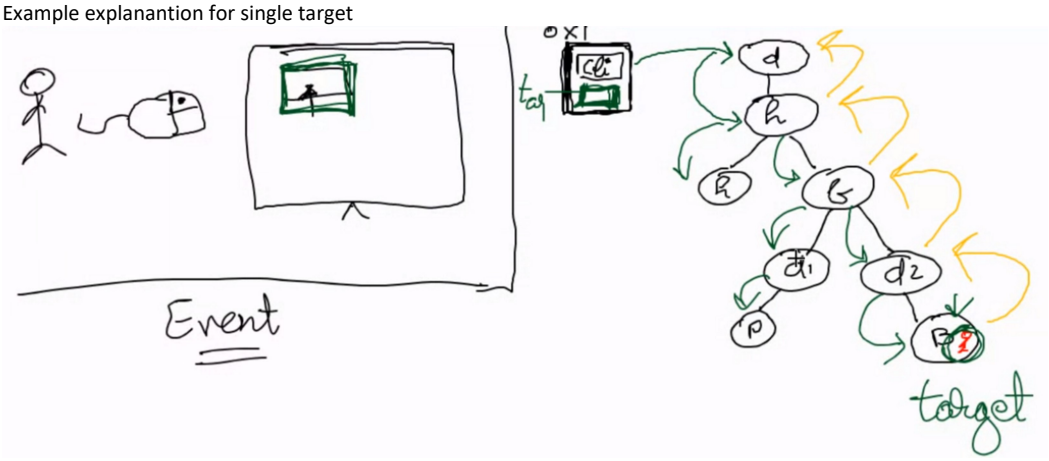
- An object which is created for an event when an event occurs is known as Event object
- Web browser creates the event object when the event occurs.
- The web browser passes the even object to the event handler.

Some of the important property or methods of Event object

1. type: it holds the type of event that is fired.
2. target: it holds the reference of target element of the event.
3. eventPhase: it holds the number which represents the event phase
    a. 1- for capturing phase
    b. 2- target
    c. 3- bubbling
4. currentTarget: holds the reference of current element on which the event is firing.
5. stopPropagation( ): it cancels an further event capturing or bubbling.

Note:
Event object can be used only inside evenHandler.

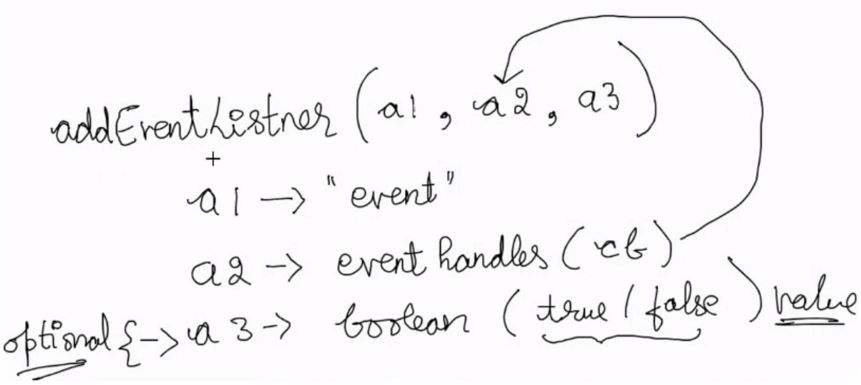
Target and type attribute of the event object must match for phase 2 to occur
Example explanantion for single target

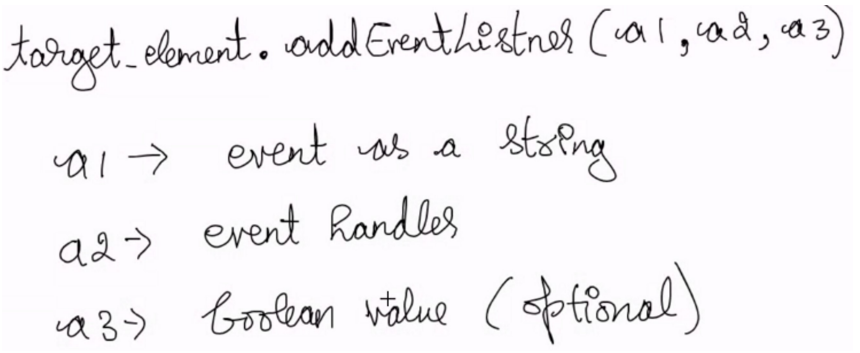


How to add an even handler to an element

addEventListner(a1,a2,a3)

A1->"event"
A2->even handler (cb)
A3->boolean (optional)





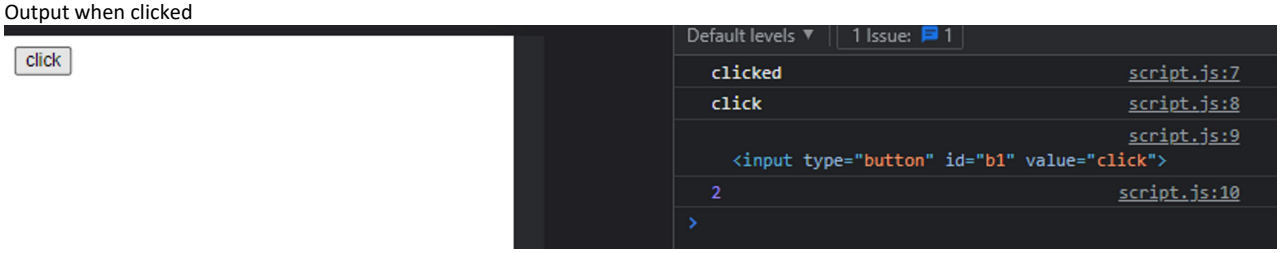
1. We can add an event handler to an element with the help of addEventListner( ) method of DOM objects

Syntax	Target_element.addEventListner(a1, a2, a3)
a1	Event as a string
a2	event handler
a3	boolean value

Example code

```
//step1 get the address of target element
let b1=document.getElementById("b1");
//step 2 design eventHandler
function eventHandler(e){
    //task to be performed when event occurs
    console.log("clicked");
    console.log(e.type);
    console.log(e.target);
    console.log(e.eventPhase);
}
//step 3add event handler to the target element
//a1=== event type
//a2--> pass eventHandler
//a3->optional
b1.addEventListener("click",eventHandler);
```

Output when clicked



Next snippet

```
//take the mouse on the button the color should change to red
let b1=document.getElementById("b1");
//desgin evnt handler for mouse hover
function eventHandler1(){
    b1.style.backgroundColor="red";
    b1.style.width="70px";
    b1.style.height="30px";
```

```
}
//addevent handler to element
b1.addEventListener("mouseover",eventHandler1);
b1.addEventListener("mouseout",()=>{
    b1.style.backgroundColor="";
    b1.style.width="40px";
    b1.style.height="20px";
});
```

Nested events

If there is an event handler designed with the same event inside the nested elements then it is known as Nested events.
Example: let us consider nested div tags where each div tag is having an event handler click

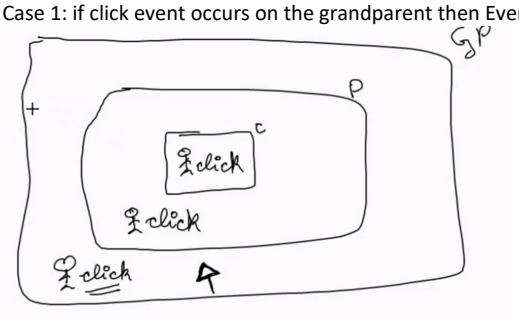
Code

```
let
gp=document.getElementById("grandparent");
let p=document.getElementById("parent");
let c=document.getElementById("child");
c.addEventListener("click",()=>{
    console.log("child is clicked");
},true);
p.addEventListener("click",()=>{
    console.log("parent clicked");
},true)
gp.addEventListener("click",()=>{
    console.log("gp clicked");
},true)//third argument not neccessary
```

```
<body>
    <div id="grandparent" class="style1">

        <div id="parent" class="style1">
            <div id="child"></div>
        </div>
    </div>
    <script src="./js1.js"></script>
  </body>
```

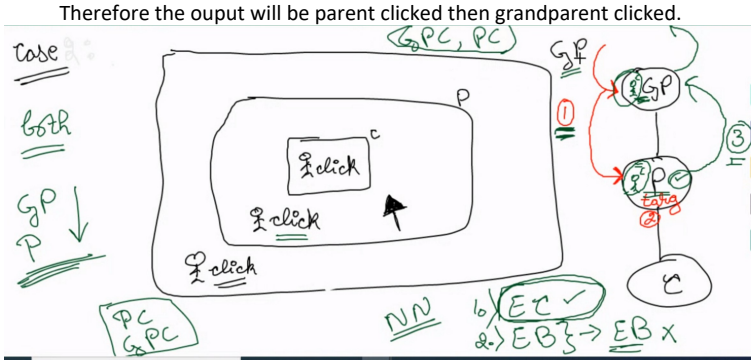
```
#grandparent{
    border: 2px solid black;
    width: 500px;
    height: 50px;
    padding: 50px;
}
#parent{
    border: 2px solid black;
    width: 350px;
    height: 30px;
    padding: 20px;
}
#child{
    border: 2px solid black;
    width: 150px;
    height: 10px;
    padding: 5px;
}
```

Use capture is the third argument
Case 1: if click event occurs on the grandparent then Event handler grand parent is executed.



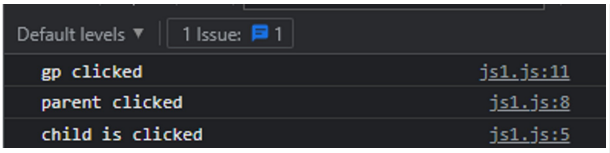
Case 2: if click event occurs in the parent

- Target is parent
- Event handlers of both parent and grandparent gets executed
- If we use event capturing flow then event handlers from top to target gets executed in phase 1.
  Therefore the ouput will be grandparent clicked and parent clicked.
- If event bubbling flow is used then the execution of event handlers happen from target (bottom) to top.
  Therefore the ouput will be parent clicked then grandparent clicked.



Case 3. if click occurs in child.
Possibility of outputs

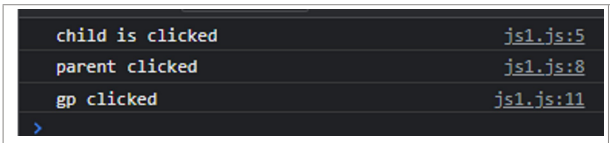
P1: gp clicked, P clicked, child clicked (outside to inside)




```
c.addEventListener("click",()=>{
    console.log("child is clicked");
},true);
p.addEventListener("click",()=>{
    console.log("parent clicked");
},true)
gp.addEventListener("click",()=>{
    console.log("gp clicked");
},true)
```

//true is passed -> capturing flow

P2: c clicked, p clicked, gp clicked (inside to outside)

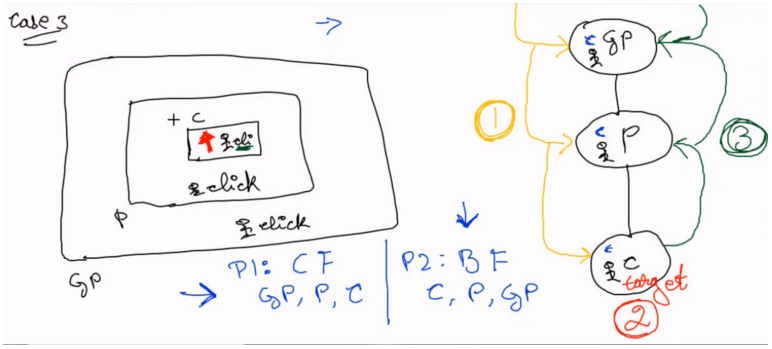



```
c.addEventListener("click",()=>{
    console.log("child is clicked");
},false);
p.addEventListener("click",()=>{
    console.log("parent clicked");
},false)
gp.addEventListener("click",()=>{
    console.log("gp clicked");
},false)
```

//false is passed -> bubbling flow

```
c.addEventListener("click",()=>{
    console.log("child is clicked");
});
p.addEventListener("click",()=>{
    console.log("parent clicked");
})
gp.addEventListener("click",()=>{
    console.log("gp clicked");
})
```

//no argument so by default -> bubbling flow



Note: By default it follows Bubbling flow

Sir's code

```
let gp = document.getElementById("grandparent");
let p = document.getElementById("parent");
let c = document.getElementById("child");

function gPHandler() {
  console.log("Grand Parent Clicked");
}

let pHandler = () => {
  console.log("parent clicked");
};

let cHandler = () => {
  console.log("child clicked");
};

c.addEventListener("click", cHandler, true);
p.addEventListener("click", pHandler, true);
gp.addEventListener("click", gPHandler, true);

// by default it uses bubbling flow
// to use capture flow
// pass the 3rd argument as true

```

For parametrized EventListener

```
function eventHandler(e){
```

```
    //task to be performed when event occurs
    console.log("clicked");
    console.log(e.type);
    console.log(e.target);
    console.log(e.eventPhase);
}
```

Used to look up properties of the thrown object by browser which is caught in local varaible e

```
e.stopPropagation();
```
//does not work for capturing phase

==========

17th Jan 2022

Browser works in 2 phases
Phase 1: load HTML simulatenously load DOM
Phase 2:loading links

Script tag attributes
Async and defer.

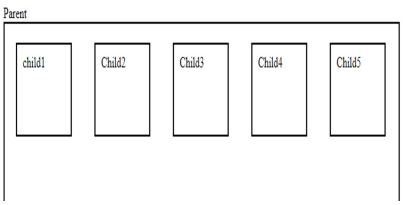
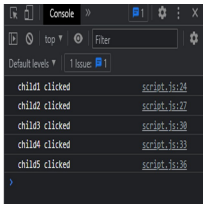
Paste image and explain

==========

Event delegation
Event delegation with the help of Event Bubbling
And do not use event capturing phase (3rd argument true) when you want to execute child and without executing parent.

Not following dry principle	Simple and follows dry principle

`let c1=document.getElementById("child1");`
`c2=c1.nextElementSibling;`
`c3=c2.nextElementSibling;`
`c4=c3.nextElementSibling;`
`c5=c4.nextElementSibling;`
`c1.addEventListener("click",()=>{`
`  console.log("child1 clicked");`
`})`
`c2.addEventListener("click",()=>{`
`  console.log("child2 clicked");`
`})`
`c3.addEventListener("click",()=>{`
`  console.log("child3 clicked");`
`})`
`c4.addEventListener("click",()=>{`
`  console.log("child4 clicked");`
`})`
`c5.addEventListener("click",()=>{`
`  console.log("child5 clicked");`
`})`
	`let p=document.getElementById("parent");`
	`p.addEventListener("click",(e)=>{`
	`  console.log(e.target.id+"is clicked");`
	`});`

Output   	Same ouput

Types of Events

16 January 2022 22:57

Types of Events

1. User Interface Events
2. Focus and blur events
3. Mouse Events
4. Keyboard Events
5. Form events
6. Mutation events and observers
7. HTML5 events
8. CSS events

9. User Interface events
   These occur as the result of any interaction with the browser window rather than the HTML page. In these events, we attach the event listener to the window object, not the document object. The various UI events are as follows.

1	load	The load event fires when the webpage finishes loading. It can also fire on nodes of elements like images, scripts, or objects.
2	unload	This event fires before the users leave the page, i.e., the webpage is unloading. Page unloading usually happens because a new page has been requested.
3	error	This event fires when the browser encounters a JavaScript error or an asset that doesn't exist.
4	resize	It fires when we resize the browser window. But browsers repeatedly fire this event, so avoid using this event to trigger complicated code; it might make the page less responsive.
5	scroll	This event fires when the user scrolls up/down on the browser window. It can relate to the entire page or a specific element on the page.

10. Focus and blur events
    These events fire when the HTML elements you can interact with gain/ lose focus. They are most commonly used in forms and especially helpful when you want to do the following tasks:

- To show tips or feedback to users as they interact with an element within a form. The tips are usually shown in the elements other than the one the user is interacting with.
- To trigger form validation as a user moves from one control to the next without waiting to submit the form.

1	focus	This event fires, for a specific DOM node, when an element gains focus.
2	blur	This fires, for a specific DOM node, when an element loses focus.

3	focusin	This event is the same as the focus event. But Firefox doesn't yet support the focusin event.
4	focusout	This is the same event as the blur event. This is a new event type in JavaScript, thus not supported in Firefox right now. The focus and blur events use the capture approach, while the focusin and focusout events use both capture and bubble approach of the event flow.

11. Mouse Events
    These events fire when the mouse moves or the user clicks a button. All the elements of the page support these events and use the bubbling approach. These actions work differently on touchscreen devices. Preventing the default behavior of mouse events can cause unexpected results. The various mouse events of JavaScript are as follows:

1	Click	This event fires when the user clicks on the primary mouse button (usually the left button). This event also fires if the user presses the Enter key on the keyboard when an element has focus. Touch-screen: A tap on the screen acts like a single left-click.
2	dblclick	This event fires when the user clicks the primary mouse button, in quick succession, twice. Touch-screen: A double-tap on the screen acts like a double left-click. Accessibility: You can add the above two events to any element, but it's better to apply it only on the items that are usually clicked, or it will not be accessible through keyboard navigation. All the mouse events discussed below cannot be triggered by the keyboard.
3	mousedown	It fires when the user clicks down on any mouse button. Touch-screen: You can use the touchstart event.
4	Mouseup	It fires when the user releases a mouse button. Touch-screen: You can use the touchend event. We have separate mousedown and mouseup events to add drag-and-drop functionality or controls in game development. Don't forget a click event is the combination of mousedown and mouseup events.
5	mouseover	It fires when the user moves the cursor, which was outside an element before, inside the element. We can say that it fires when we move the cursor over the element.
6	mouseout	It fires when the user moves the cursor, which was inside an element before, outside the element. We can say that it fires when the cursor moves off the element. The mouseover and mouseout events usually change the appearance of graphics on our webpage. A prefered alternative to this is to use the CSS: hover pseudo-class.
7	mousemove	It fires when the user moves the cursor around the element. This event is frequently triggered.

12. Keyboard Events
    These events fire on any kind of device when a user interacts with a keyboard.

1	input	This event fires when the value of an <input> or a <textarea> changes (doesn't fire for deleting in IE9). You can use keydown as a fallback in older browsers.
2	keydown	It fires when the user presses any key in the keyboard. If the user holds down the key, this event fires repeatedly.
3	keypress	It fires when the user presses a key that results in printing a character on the screen. This event fires repeatedly if the user holds down the key. This event will not fire for the enter, tab, or arrow keys; the keydown event would.

4	keyup	The keyup event fires when the user releases a key on the keyboard. The keydown and keypress events fire before a character appears on the screen, the keyup fires after it shows. To know the key pressed when you use the keydown and keypress events, the event object has a keyCode property. This property, instead of returning the letter for that key, returns the ASCII code of the lowercase for that key.

13. Form events
    These events are common while using forms on a webpage. In particular, we see the submit event mostly in form of validation (checking form values). As described in our tutorial; Features of JavaScript, if the users miss any required information or enter incorrect input, validation before sending the data to the server is faster. The list below explains the different form of events available to the user.

1	submit	This event fires on the node representing the <form> element when a user submits a form.
2	change	It fires when the status of various form elements change. This is a better option than using the click event because clicking is not the only way users interact with the form.
3	input	The input event is very common with the <input> and the <textarea> elements.

    We often use the focus and blur events with forms, but they are also available in conjunction with other elements like links.

14. Mutation events and observers
    Whenever the structure of the DOM tree changes, it triggers a mutation event. The change in the tree may be due to the addition or removal of a DOM node through your script. But these have an alternative that will replace them: mutation observers. The following are the numerous mutation events in JavaScript.

1	DOMNodeInserted	It fires when the script inserts a new node in the DOM tree using appendChild(), replaceChild(), insertBefore(), etc.
2	DOMNodeRemoved	This event fires when the script removes an existing node from the tree using removeChild(), replaceChild(), etc.
3	DOMSubtreeModified	It fires when the structure of the DOM tree changes i.e. the above two events occur.
4	DOMNodeInsertedIntoDocument	This event fires when the script inserts a node in the DOM tree as the descendant of another node already in the document.
5	DOMNodeRemovedFromDocument	This event fires when the script removes a node from the DOM tree as the descendant of another node already in the document.

    The problem with the mutation events is that lots of changes to your page can make your page feel slow or unresponsive. These can also trigger other event listeners, modifying DOM and leading to more mutation events firing. This is the reason for introducing mutation observers to the script.

    Mutation observers wait until the script finishes its current task before reacting, then reports the changes in a batch (not one at a time). This reduces the number of events that fire when you change the DOM tree through your script. You can also specify which changes in the DOM you want them to react to.

15. HTML5 events
    These are the page-level events included in the versions of the HTML5 specialization. New events support more recent devices like phones and tablets. They respond to events such as gestures and movements. You will understand them better after you master the above concepts, thus they are not discussed for now. Work with the events below for now and when you are a better developer, you can search for other events available. The three HTML5 events we will learn are as follows:

1	DOMContentLoaded	This event triggers when the DOM tree forms i.e. the script is loading. Scripts start to run before all the resources like images,

		CSS, and JavaScript loads. You can attach this event either to the window or the document objects.
2	hashchange	It fires when the URL hash changes without refreshing the entire window. Hashes (#) link specific parts (known as anchors) within a page. It works on the window object; the event object contains both the oldURL and the newURL properties holding the URLs before and after the hashchange.
3	beforeunload	This event fires on the window object just before the page unloads. This event should only be helpful for the user, not encouraging them to stay on the page. You can add a dialog box to your event, showing a message alerting the users like their changes are not saved.

16. CSS events
    These events trigger when the script encounters a CSS element. As CSS is a crucial part of web development, the developers decided to add these events to js to make working with CSS easier. Some of the most common CSS events are as follows:

1	transitionend	This event fires when a CSS transition ends in a program. It is useful to notify the script of the end of transition so that it can take further action.
2	animationstart	These events fire when CSS animation starts in the program.
3	animationiteration	This event occurs when any CSS animation repeats itself. With this event, we can determine the number of times an animation iterates in the script.
4	animationend	It fires when the CSS animation comes to an end in the program. This is useful when we want to act just after the animation process finishes.

Reference link https://data-flair.training/blogs/javascript-event-types/

BOM

Monday, January 17, 2022 3:05 PM

BOM
Make your notes
Location from one page to another page
Timer
Navigator
Screen

Asynchronous

Monday, January 17, 2022 3:05 PM

Asynchronous
The behaviour of making way for others is known as Asynchronous. (informal definintion).

Illustration!
Let us consider there are two functionalities f1 and f2 purely independent from each other.
Let us assume, f1 takes 10 mins to complete the execution, f2 takes just a minute to complete the execution.
Therefore f1 is called first and then F2.
F2 should wait for 10 mins to get execution started. This behaviour is synchronous.

To overcome this we can design our application in such a way that F1 gives way for F2 to complete its execution and then F1 can complete its execution, such a design is known as Asynchronous.

We can achieve Asynchronous behaviour with the help of setTimeOut( ) method.

setTimeOut( )
- setTimeOut( ) is a method of window object.
- setTImeOut( ) can accept two arguments
  Argument 1: a call back function
  Argument 2: delay time in milli seconds
- When a setTimeOut( ) function is called, it will the register the given call back function and starts the browser timer for the given milliseconds.
- Once the said time is completed, callback function is moved to the call back queue.
- The event loop loads the call back function from call back queue in to the stack when the call stack is idle.
- Event loop waits until the call stack becomes idle.
- Call stack becomes idle only after the all the instructions of global execution context is completed.

Problems with Asynchronous Behaviour
1. If a feature is asynchronous we cannot predict when the task is completed.
2. Since the completion of asynchronous feature is not predictable, if there is any dependent feature or task which has to be executed only after the completion of the asychronous behaviour, it becomes difficult to design.
3. To achieve the above said scenario we can use the following design technique.
   a. With the help of call backs
   b. Using promises introduced in ES6

Another method setInterval( ) it keeps on executing continosly for set interval (won't stop)

Example:

```
function print(m,n){
    setTimeout(()=>{
        for(let i=m;i<=n;i++)console.log(i)
    },1000)

}
function successMessage(){
    console.log("numbers printed successfully");
}
```

print(m, n) is an asynchronous function, therefore we cannot predict when the function will complete its task

```
console.log("js start");
print(10,15);
successMessage();
console.log("js end");
```

Expectations
We Expect success message is printed after the completion print(10,15) method call.

Reality
Since print(10,15) is asynchronous it gives away the call stack when called and cannot predict when the numbers are printed. But the success message is printed before the numbers are printed.

```
js start
numbers printed successfully
js end
Live reload enabled.
10
11
12
13
14
15
>
```

Note: From the above example it is understood success message is dependent on asynchronous print method. Therefore the above design does not promise us success message to be executed after print.

Solution1:
We can solve this problem using callbacks.
The print method must accept the dependent function, print method should take the responsibility of calling the call back function.(dependent function)

Redesigning print method using call back.

```
function print(m,n,cb){
    setTimeout(()=>{
        for(let i=m;i<=n;i++)
        console.log(i)

        if(cb!==undefined){
        cb();
        }
    },1000)
}
```

Now we can call the asynchronous print method by passing the dependent function as a call back function.

```
console.log("js start");
print(10,15,successMessage);
console.log("js end");
```

output

```
js start
js end
10
11
12
13
14
15
numbers printed successfully
>
```

Solution 2: Promises
We can solve this problem using promise.

Promise

Tuesday, January 18, 2022 12:31 PM

Promise

1. Promise is an object.
2. Promise object keeps an eye on the asynchronus task given.
3. If the asynchronous task is not yet completed the promise is considered as pending(status).
4. If the asynchronous task is successfully completed then the promise is considered as resolved.
5. If the asynchronous task is completed but not successful then it is considered as Reject.
6. In promise object we have two important methods.
    a. then(cb)- it can accept a call back function.
    The call back function passed to then method gets executed only when the promise returns resolve.
    b. catch(cb) - it can accept a call back function.
    The call back function passed to catch method gets executed only when the promise returns reject.

Syntax to create a promise object
new Promise(call_back_function)

==========

New Promise((resolve, reject) => {

//asynchronous task

})

Note:Therefore we pass the asynchronous task as a call back function to the promise object.

Realtime example is fetch( )

Steps to design a Asynchronous function

async function print(m,n) using promise

step1: create function which returns promise

Step2: pass asynchronous task as a call back function with 2 parameters
resolve and reject to the Promise function constructor.

Step3: if the asynchronous is completed successfully
callback function must return resolve( ).

Step 4(optional): if the asynchronous task is completed
but not successful callback must return reject( ).

```
function print(m, n) {
  //asynchronous task
  return new Promise((resolve, reject) =>
{
    setTimeout(() => {
      if (isNaN(m) || isNaN(n)) {
        return reject();
      }
```

```
console.log("start");
print(10, 15)
  .then(successMessage)
  .catch(() => {
    console.log("some error
occured");
  });
console.log("end");
```

```
console.log("start");
print(10, "15a")
  .then(successMessage)
  .catch(() => {
    console.log("some error
occured");
  });
console.log("end");
```

```
      for (let i = m; i <= n; i++) {
        console.log(i);
      }
      //on success
      return resolve();
    }, 1000);
  });
}


function successMessage() {
  console.log("numbers printed
successfully");
}
```

```
start
end
10
11
12
13
14
15
numbers printed successfully
>
```

```
start
end
some error occured
>
```

Design a asynchronous function which returns a promise, the function must accept m and n and print all even numbers between them, call the function for following test data
Td1 10,20
Td2 30,x

```
function even(m, n) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (isNaN(m) || isNaN(n)) {
        return reject();
      }
      for (let i = m; i <= n; i++) {
        if (i % 2 == 0) console.log(i);
      }
      return resolve();
    }, 1000);
  });
}
function message() {
  console.log("success");
}
console.log("start");
even(10, 20)
  .then(message)
  .catch(() => {
    console.log("error occured");
  });
console.log("stop");
```

Use of keywords
async and await

Await was intoduced in ES8.
Await is just a syntax sugar for using then method of promise.
Syntax

```
await  asynchronous_function_call( );

let variable = await methodCall( );
```

Note:

1. await can be used only if the function returns a promise.
2. await keyword can be used only inside a function block which is prefixed with async modifier.

Ex: let us consider an synchronous function print even.

```
//async await
function even(m, n) {
    return new Promise((resolve, reject) => {
      setTimeout(() => {
        if (isNaN(m) || isNaN(n)) {
          return reject();
        }
        for (let i = m; i <= n; i++) {
          if (i % 2 == 0) console.log(i);
```

```
        }
        return resolve();
      }, 1000);
    });
}
```

Assume we have to call even( ) method and execute some dependent task, this is done using then method of promise.

```
function stkart(){
even(10, "x")
  .then(()=>{console.log("success");})
  .then(()=>{console.log("js end");})
  .catch(()=>{console.log("something went wrong");})
}
stkart();
```

The function start can be redesigned using the keywords async and await.

```
async function start(){
    console.log("start");
    await even(10,20);
    console.log("success");
    console.log("js end");
}
```

Note: we cannot put negative scenarios or catch() method of promise here so we use try catch block.

```
async function start(){
    console.log("start");
    try{
        await even(10,"20a");
        console.log("success");
        console.log("js end");
    }
    catch(error){
        console.log("something seriously went wrong");
    }
}
start()
```

Whenever we call promises it goes to microtask queue

Giphy-API

Tuesday, January 18, 2022 5:37 PM

Step1: make account
Step2 go to api quick starts click on devlopers dashboard.

3. Create an app
   Go to API explorer

In url
Url followed by

1. Api key
2. parameters
   Parameter1&parameter2
   Key=value & key=value

```
function sum(){
 return Math.trunc(Math.random()*25);
}
// my try
async function alpha() {
  let btn = document.getElementById("click-ok");
  btn.addEventListener("click", async () => {
    let field = document.getElementById("text-field");
    let value = field.value;
    // console.log(value);
    let url = `https://api.giphy.com/v1/gifs/search?api_key=
0Sp9Atcbs0AD0NyNij7ibGRpS6RKrJ5u&q=${value}&limit=25&offset=0
&rating=g&lang=en`;
    let response = await fetch(url);
    let img=await response.json();
    let im=document.getElementById("img-container");
    im.src=img.data[sum()].images.original.url;
    console.log(im.src);
    // .data.images.original.url;
    // console.log(img);
  });
}
alpha();
```

```
#img-container
{
  border: 2px
solid black;
  height: max-
content;
  width:max-
content;
  margin:10px;
  padding:2px;
  height:400px
;
  width:400px;
}
#maincontainer
{
  border: 2px
solid black;
  background-
color: grey;
  height:500px
;
  width:500px;
  margin:10px;
  padding:2px;
}
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-
Compatible" content="IE=edge" />
    <meta name="viewport"
content="width=device-width,
initial-scale=1.0" />
    <title>Document</title>
    <link rel="stylesheet"
href="./style.css" />
  </head>
  <body>
    <script defer src="./app.js">
</script>
    <div id="maincontainer">
      <input type="text" id="text-
field" />
      <input type="button"
value="ok" id="click-ok" /><br />
<br>
      <img src="" alt="img not here"
id="img-container" />
    </div>
  </body>
</html>
```

heman ok

Call apply bind

Tuesday, March 1, 2022 10:49 AM

Function methods

Call

```
//call apply bind methods
let employee = {
  emp_id: "typ1",
  emp_name: "shashi",
};
let qsp_employee = {
  emp_id: "qsp1",
  emp_name: "priya",
};
function Testyantra() {
  console.log(this);
}
Testyantra.call(qsp_employee);
```

Another example==========

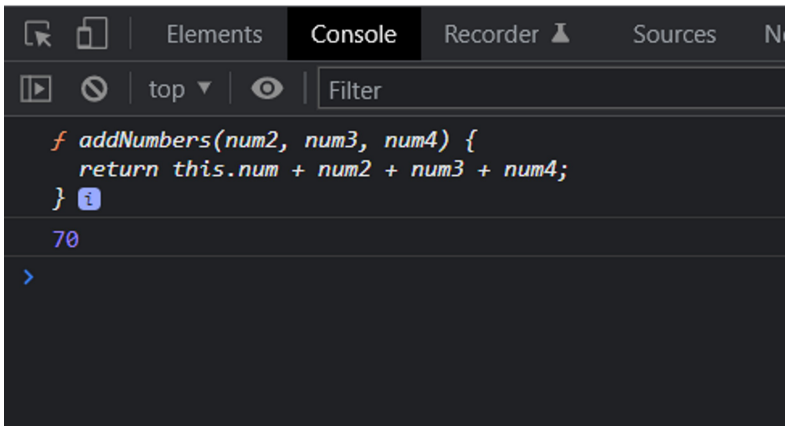
```
let obj1 = {
    num=10,
}
function addNumbers(num2) {
    return this.num + num2;
}
//20
let total = addNumbers.call(obj1, 10);
console.log(total);
let users = {
    firstname:"shashi"
}


function getFullName(lastname) {
    return this.firstname + lastname;
}
let fullname = getFullName.call(users, "kunal");
console.log(fullname);
```

Apply

```
let obj1 = {
  num: 10,
};
function addNumbers(num2, num3, num4) {
  return this.num + num2 + num3 + num4;
}
let total = addNumbers.apply(obj1, [10, 20, 30]);
console.log(total);
```

Bind

```
let obj1 = {
  num: 10,
};
function addNumbers(num2, num3, num4) {
  return this.num + num2 + num3 + num4;
}
let total = addNumbers.bind(obj1, 10, 20, 30);
console.log(total);
console.log(total());
```



Cheat sheet and terminologies

Thursday, December 30, 2021 2:39 PM

Function terminologies

1. Function
2. Function declaration
3. Function definition
4. Calling function
5. Called function
6. Function statement
7. Function literal
8. Named function
9. Immediately invoked function expression
10. Recursive function
11. Pure function
12. Function expression
13. Anonymous function
14. Parameterized function
15. Non parameterized function
16. Call back function
17. Higher order function
18. Function hoisting
19. Arrow function
20. Constructor function
21. Funcitonal Programming
22. First citizen function
23. Nested function
24. Outer function
25. Inner function

Scopes

1. Global scope
2. Local scope
3. Block scope
4. Function scope
5. Script scope
6. closure

String()

1. String function
    a. Property
        i. Length
    b. Functions

What is protoypal inheritance and prototype chaining?

Argument object: A local variable avialiable within all non- arrow functions which can refers to a function arguments.

Basic tags and forms
Selectors
Datatypes and arrays

Terminal commands to move files
move * ../dom