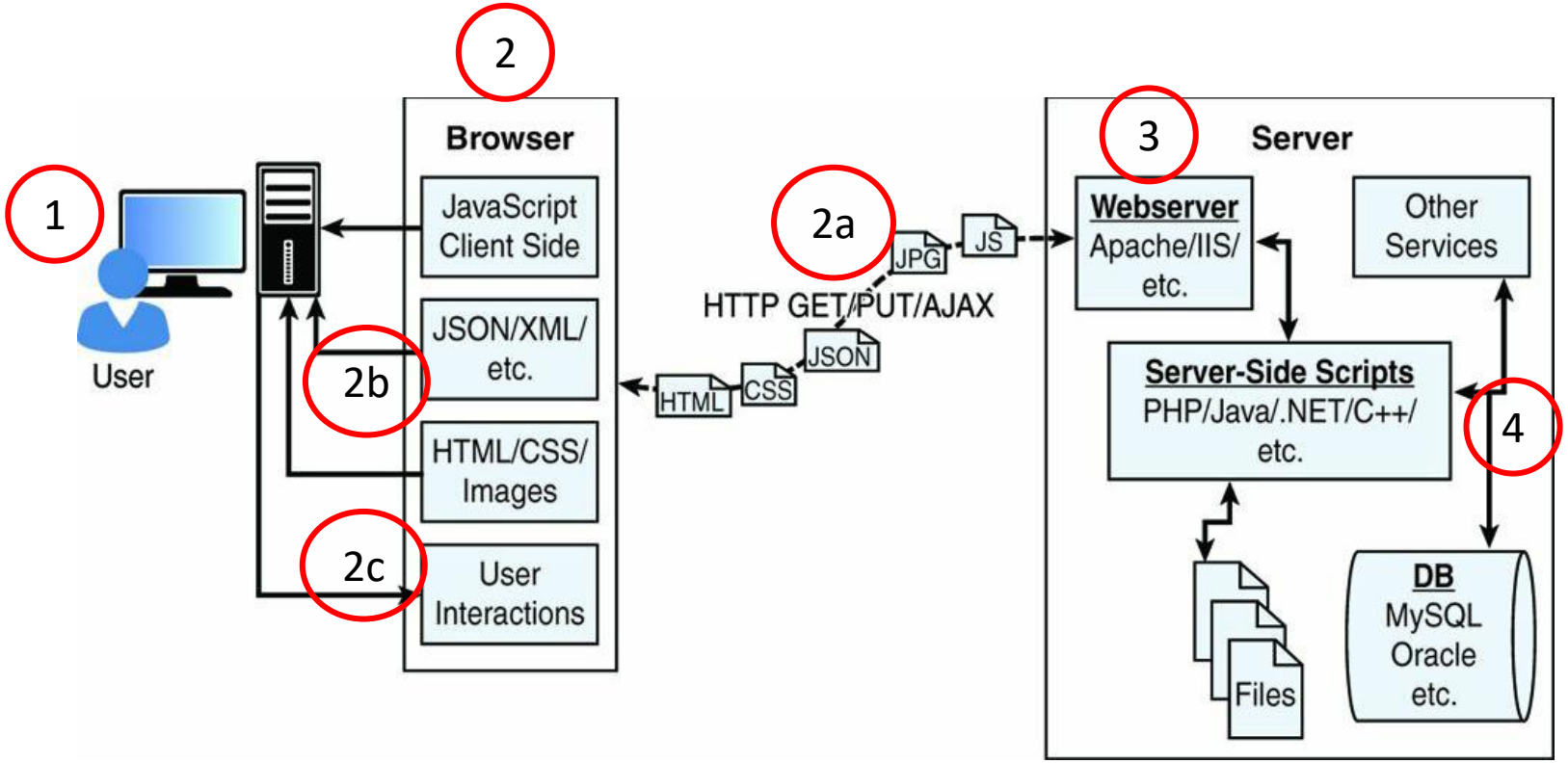
MERN

MangoDB, Express.js, React, and Node.js

Understanding the Basic Web Development Framework

The fundamental components of Web Application is

1. User
2. Browser
   a. Browser-to-webserver communication
   b. Rendering the browser view (displaying the page)
   c. User interaction
3. Web server
4. Backend services



1. Role of user

- The user role in a web framework is to sit on the visual output and interaction input of webpages.

- users view the results of the web framework processing and then provide interactions using mouse clicks, keyboard input, and swipes and taps on mobile devices.

2. Role of browser

The browser plays three roles in the web framework.

- Communication: It provides communication to and from the webserver.
- Rendering(displaying): It interprets the data from the server and renders it to visualize the user
- User interaction : It handles user interaction through the keyboard, mouse, touchscreen, or other input device and takes the appropriate action.

2 a) Communication between Browser and webserver

The browser makes three main types of requests to the server:

- **GET:** The GET request is used to retrieve data from the server, such as .html files, images, or JSON data.
- **POST:** POST requests are used when sending data to the server, such as adding an item to a shopping cart or submitting a web form.
- **AJAX:** Asynchronous JavaScript and XML (AJAX) is actually just a GET or POST request done directly by JavaScript running in the browser. Despite the name, an AJAX request can receive XML, JSON, or raw data in the response.

Why JSON and XML?

JSON(Java Script Object Notation)

- JSON is commonly used in web applications and API calls to transfer data between a server and a web application.
- JSON is compact and easy to parse in web browsers.
- JSON can be parsed by a standard JavaScript function.

```
{"employees":[
  { "firstName":"John", "lastName":"Doe" },
  { "firstName":"Anna", "lastName":"Smith" },
  { "firstName":"Peter", "lastName":"Jones" }
]}
```

XML (Extensible Markup Language)

- Extensible Markup Language (XML) is a markup language used to store, transport, and exchange data between applications, platforms, and organizations
- XML has to be parsed with an XML parser.

```
<employees>
  <employee>
    <firstName>John</firstName>
    <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName>
    <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName>
    <lastName>Jones</lastName>
  </employee>
</employees>
```

2 b) Rendering: Displaying the data from server in browser

- The browser reads data from the initial URL and then renders the HTML document to build a Document Object Model (DOM).

```
document
 + html
    + head
    + body
       + div
       + p
```

HTML files: These provide the fundamental structure of the DOM.
CSS files: These define how each of the elements on the page is to be styled; for example, font, color, borders, and spacing.
Client-side scripts: These are typically JavaScript files. They can provideadded functionality to the webpage, manipulate the DOM to change the look of the webpage, and provide any necessary logic required to display the page and provide functionality.
Media files: Image, video, and sound files are rendered as part of the webpage.
Data: Any data, such as XML, JSON, or raw text, can be provided by the webserver as a response to an AJAX request. Rather than sending a request back to the server to rebuild the webpage, new data can be retrieved via AJAX and inserted into the webpage via JavaScript.
HTTP headers: The HTTP protocol defines a set of headers that can be used by the browser and client-side scripts to define the behavior of the webpage. For example, cookies are contained in the HTTP headers. The HTTP headers also define the type of data in the request as well as the type of data expected to be returned back to the browser.

3. Web server

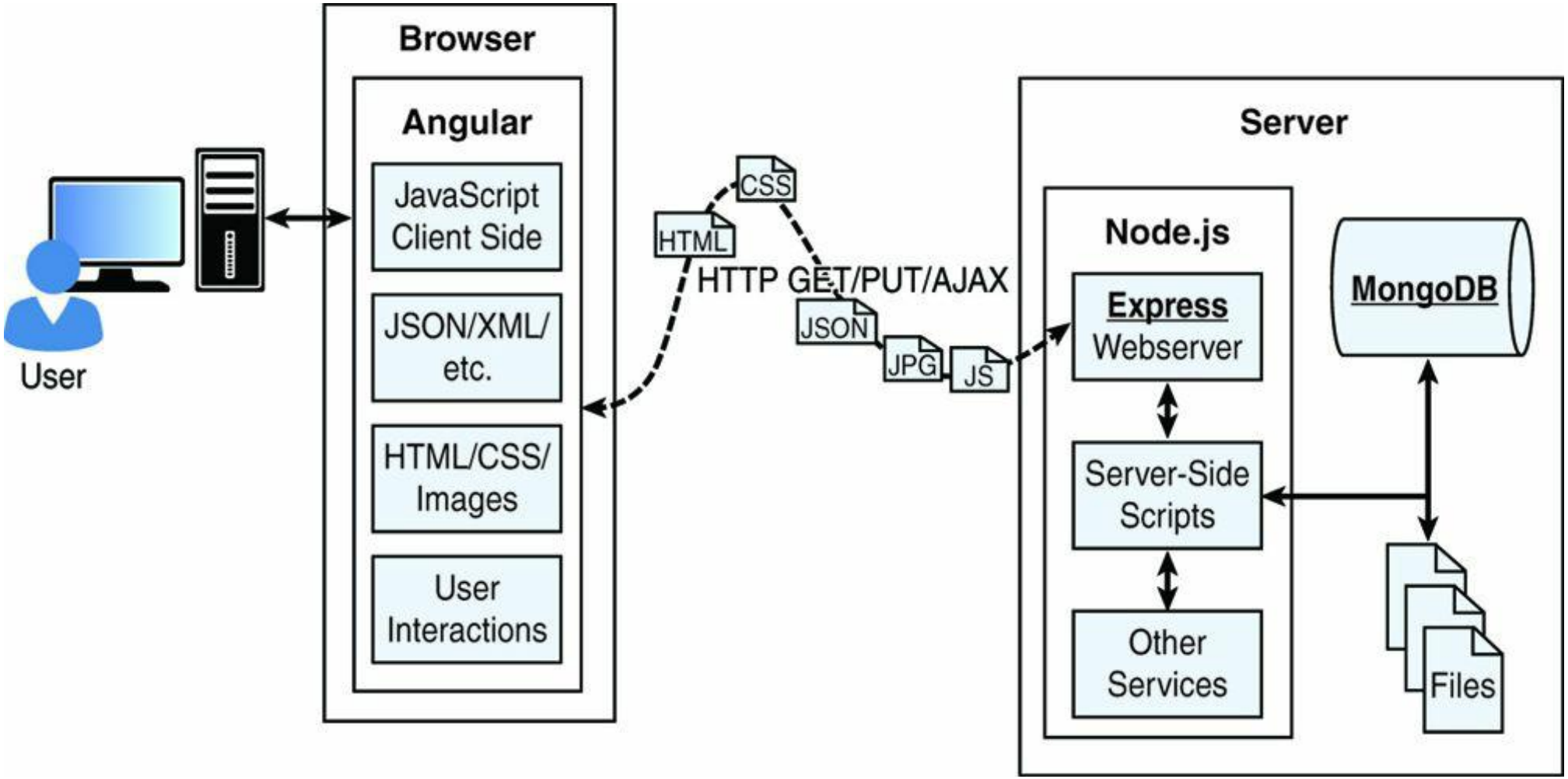
The webserver’s focus is handling requests from browsers.

- the browser may request a document, post data, or perform an AJAX request to get a data.
- The webserver uses the HTTP headers as well as the URL to determine the action to be taken.
- It contains server side scripts to handle POST requests that modify data and AJAX requests to interact with backend services.
- A server-side program that is executed by webserver to perform the browser requests. It can be written in PHP,python,C,C++,C#,Java…..
- The server-side scripts either generate the response directly by executing their code or connect with other backend servers such as databases to obtain the necessary information , which is used to build and send the response.

4. Backend Services

Backend services are services that run behind the webserver and provide data used to build responses to the browser.
Example : CRUD operations in database.

Understanding the Node.js-to-Angular Stack Components



JavaScript end-to-end

Node.js

Node.js is a development framework based on Google's V8 JavaScript engine. Node.js code is written in JavaScript and then compiled into machine code by V8 to be executed.

- Javascript end-to-end
- Scalability
- Extensibility
- Ease of installation and use

MongoDB:

MongoDB is a document database with the scalability and flexibility that querying and indexing made as per the user requirement.

- Document Orientation
- High performance
- High availability
- High Scalability
- No SQL injection

ExpressJS

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

- Route Management
- Error Handling
- Easy Integration
- Cookies
- Session and Cache Management

Angular

Angular is a client-side framework developed by Google. Angular provides all the functionality needed to handle user input in the browser, manipulate data on the client side, and control how elements are displayed in the browser view. It is written using TypeScript and uses MVC framework.

- Data binding
- Extensibility
- Clean
- Reusable code
- Support
- Compatability.

Understanding the role of java script in web application

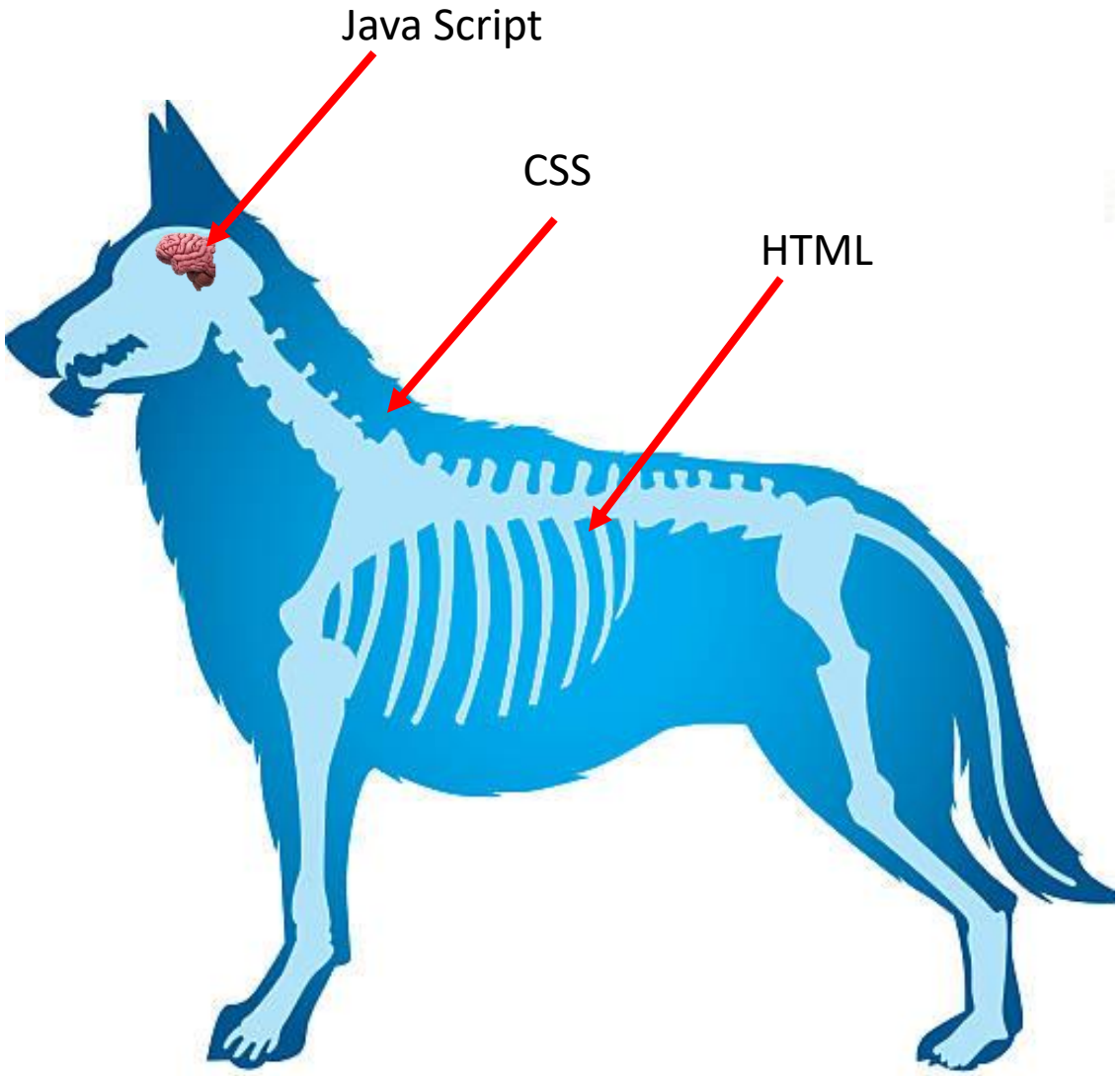
Web browser

web page

HTML
CSS
JavaScript

- Browser has in-memory representation of web page known as DOM(Document Object Model).

- JavaScript manipulates the DOM to control the behavior of web page.
- JavaScript doesn't modify the HTML document.



Analogous to brain signals which controls the behavior of dog.

Brain signals doesn't modify the structure of dog.

JavaScript

- JavaScript is a programming language , which runs inside the web browser, and controls the behavior of the web page.

Programming language	• It is similar to object-oriented programming languages like C++, java, and python
Runs inside a browser	• It is just-in-time compiled and executed by the browser. • It is client-side technology, now it can run on the server side.
Behavior of the web page	• It is brain of the web page that controls the appearance, communication and interactions etc.

Fundamental features of JavaScript

- Variables
- Functions
- Loops
- Arrays
- Objects
- Input and output

Variable declaration

a = 20;	With out specifying we can declare a variable and a value to it
var a =20;	Use var keyword to declare a variable
let a =20;	Use let keyword to declare a variable, not allowed to redeclare with the same name.

Strict mode makes several changes to normal JavaScript semantics:

- Eliminates some JavaScript silent errors by changing them to throw errors.
- Fixes mistakes that make it difficult for JavaScript engines to perform optimizations: strict mode code can sometimes be made to run faster than identical code that's not strict mode.
- Prohibits some syntax likely to be defined in future versions of ECMAScript.

JavaScript Data types

Datatype	Object Wrapper	Purpose
Number	Number	For integers and real values
BigInt	BigInt	Used for Large integers specified by appending ***n*** to the number
String	String	Character data
Boolean	Boolean	true or false
Undefined	--	To identify a variable is not defined or default value of declared variables
Null	--	A variable is defined and assigned explicitly null.

JavaScript Operators

Operators	**Symbols used**
Arithmetic	+, -, *, /, %, **(exponentiation)**
Relational	<,>,<=,>=,==,!=, **===(strictly equal),!==(strictly not equal)**
Logical	**&&, ||, !, ??(nullish Coalescing operator)**
Bitwise	&, |, ^, ~, <<, >>, **>>>(unsigned right shift/ zero-fill right shift)**
Assignment	=,-=,*=,/=,%=,**=, &&= , ||= , ??= , &=,|=,^=, <<=, >>=, <<<=
Ternary	?:
Unary	-, +, ++, --, **typeof, instanceof, delete**
Other	,(comma), of, in, new

Relational Operator

==(equal) vs. ===(strictly equal)

```
console.log('100' == 100)// checks only the content
Output: true

console.log('100’ === 100)// checks content and type also
Output: false
```

Similarly , it is same for != and !==(strictly not equal)

Logical Operator

??(nullish Coalescing operator)

var res = expr1??expr2
Returns expr1 if it is neither null nor undefined; otherwise, returns expr2.

Bit-wise Operators

Expression	Result	Binary Description
15 & 9	9	1111 & 1001 = 1001
15 | 9	15	1111 | 1001 = 1111
15 ^ 9	6	1111 ^ 1001 = 0110
~15	-16	~ 0000 0000 … 0000 1111 = 1111 1111 … 1111 0000
~9	-10	~ 0000 0000 … 0000 1001 = 1111 1111 … 1111 0110
9<<2	36	0000 1001 = 0010 0100
9>>2	2	0000 1001 = **00**00 0010
-9>>2	-3	0000 1001 = 1111 0111 = 1111 1101 = -3 (1000 0011) Copy of leftmost bit are shifted in from the left
19>>>2	4	0001 0011 = **00**00 0100 **Not applicable for BigInt datatype** **zeros are shifted in from the left**

Functions in JavaScript

In JavaScript function is defined as follows

```
function functionname(parameters){
    // body of the function;
}
```

```
function add(x, y ) {
     return x+y;
}
var res = add(2,3);
Console.log("the sum of 2 and 3 is ", res);
```

In JavaScript , functions are the first class members because

- functions can be take another function as an argument
- Functions can return another function
- Function can store in a variable.

Different ways of creating functions in JavaScript

Function Declaration	`function multiply(x, y) {`
	`  return x * y;`
	`}`

Function Expression	`// Expression; the function is anonymous but assigned to a variable`
	`const multiply = function (x, y) {`
	`  return x * y;`
	`};`
	`// Expression; the function has its own name`
	`const multiply = function funcName(x, y) {`
	`  return x * y;`
	`};`

Function Constructor	`const multiply = new Function("x", "y", "return x * y");`
Arrow functions	`const multiply = (x, y) => x * y;`

```
let result = multiply(3,4);
console.log(“Multiplication of 3 and 4 is “, result);
```

Method	`const obj = {`
	`  multiply(x, y) {`
	`  return x * y;`
	`  },`
	`};`
		`let result = obj.multiply(3,4);`
		`Console.log(“multiplication of 3 and 4 is”, result);`

Function as an argument to another function

```
let addition = function(a,b){
    return a+b;
}
let multiplication =function(a,b){
    return a*b;
}
let division =function(a,b) {
    return a/b;
}
let subtraction =function(a,b){
    return a-b;
}
function calculate(a,b,calculationType){
    return calculationType(a,b);
}
let result = calculate(10,20,addition);
console.log("addition =",result);
```

Function expressions are convenient for passing functions as arguments to another function.

```
function map(f, a) {
    const result = new Array(a.length);
    for (let i = 0; i < a.length; i++) {
      result[i] = f(a[i]);
    }
    return result;
  }

  const cube = function (x) {
    return x * x * x;
  };

  const numbers = [0, 1, 2, 5, 10];
  console.log(map(cube, numbers)); // [0, 1, 8, 125, 1000]
```

Function return type is function

```
let fxRates ={
    INR: 84.7,
    EUR: 1.1
}
let discounts ={
    INR : 0.1,
    EUR : 0.2
}
let inCart={
    id:1,
    total: 78000,
    currency : "INR"
}
let erCart={
    id:2,
    total: 489,
    currency: "EUR"
}
```

```
function getTotalPrice(cart){
    cart.finalTotal = cart.total*(1-discounts[cart.currency]);
    function totalUSDPrice(){
        return cart.finalTotal/fxRates[cart.currency];
    }
    return totalUSDPrice;
}
let inCartTotalUSD = getTotalPrice(inCart);
console.log("total of order from india is $",inCartTotalUSD());

let eucartTotalUSD =getTotalPrice(erCart);
console.log("total of order from Europe is $",eucartTotalUSD());
```

Arrays in javascript

<table>
<tr><td></td><td colspan="2"></td></tr>
<tr><td>Creating an array</td><td colspan="2">Using array constructor
let arr = new Array(4); // creates an array of size 4
(Or)
let arr = [ ] ; // creates an empty array
arr[0] = “A” ; arr[1] = “B”; arr[2] = 2.74 ; arr[3] = “hello”</td></tr>
<tr><td>Iterating an array</td><td>for (var i =0; I < a.length; i++)
console.log(arr[i]);
<u>Output</u>
A
B
2.74
hello</td><td>for(i of arr)
console.log(i);
<u>Output</u>
A
B
2.74
hello</td></tr>
<tr><td></td><td colspan="2"></td></tr>
</table>

Rest parameter

```
function add(x,y){ // x=8, y= 4
    let s=x+y;
    return s;
}
let addition = add(8,4,5,3);
console.log(addition); //outputs 12
```

In javascript, function parameters are taken as an array

```
function add(...nums){
    let s=0;
    for(i of nums)
        s=s+i;
    return s;
}
let addition = add(8,4,5,3);
console.log(addition); //outputs 20
```

Before rest parameter any number of formal arguments may present, but after rest parameter it should not contain any formal argument.

```
function fun_name(arg1,arg2, …restparam)   ----→ allowed
function fun_name(…restparam,arg1,arg2)    ----→ not allowed
```

<table>
<tr><td>Destructing an array</td><td colspan="2">

```
let num=[10,20,30,40]
let [a,b,c,d]=num;
console.log(a+" "+b+" "+c+" "+d); // outputs 10 20 30 40
```

</td></tr>
<tr><td>Copying an array</td><td>Shallow copy – modification of copied array affects the original array

```
let prices= new Array(10,20,5,15,25);
let scopy=prices;
scopy[0]=50;
console.log(“scopy "+scopy);
//outputs scopy 50, 20, 5, 15, 25
console.log("prices "+prices);
//outputs prices 50, 20, 5, 15, 25
```

</td><td>Deep copy: modification of copied array doesn’t affect the original array

```
let prices= new Array(10,20,5,15,25);
let scopy=prices.slice();
scopy[0]=50;
console.log(“scopy "+scopy);
//outputs scopy 50, 20, 5, 15, 25
console.log("prices "+prices);
//outputs prices 10, 20, 5, 15, 25
```

</td></tr>
</table>

Array Operations	• push – insert at end • unshift – insert at begin • pop – remove at end • shift – remove at begin • delete arr[index] – delete the element at index , it is replaced with undefined value • splice – used to insert the element at specified position. • slice – used to cut the array

	Syntax:
	splice(startIndex, no_ of_ elements_to_remove, element_list);
	var arr= [10,20,30,40,50]
	arr.splice(2,0,25,28);// starting from index 2 remove 0 elements and insert 25 and 28 at index 2 and 3 respectively.
	Console.log(arr) ; // [10, 20, 25, 28, 30, 40, 50]

	Syntax: slice(start,end) -> copies from *start* index to *end-1* index. slice(start) -> copies from start index to last slice(-start) -> copies from last of the array slice(-start,-end) -> copies from start to end-1 from last of the array

concat	let arr1 = [2, 11, 5]
	let arr2 = [3, 19, 7]
	let arr3 = [4, 9, 6]
	let newArray1 = arr1.concat(arr2) // newArray1 = [2, 11,5,3,19, 7]
	let newArray2 = arr2.concat(arr1, arr3) // newArray2 = [3, 19, 7, 2, 11, 5, 4, 9, 6]

sort	`let arr1=[3,10,20,30,40,50];`
	`let sortedArray = arr1.sort();`
	`console.log(arr1); // arr1 = [ 10, 20, 3, 30, 40, 50 ]`
	`console.log(sortedArray);// sortedArray= [ 10, 20, 3, 30, 40, 50 ]`
	
	Sorts the original array and perform alphabetical order.
	<u>Make a copy using spread operator/slice</u>
	
	`let arr1=[3,10,20,30,40,50];`
	`let sortedArray1= [...arr1].sort();`
	`console.log(arr1); // arr1 = [ 3, 10, 20, 30, 40, 50 ]`
	`console.log(sortedArray1); // sortedArray= [ 10, 20, 3, 30, 40, 50 ]`
	
	To sort the numbers in numerical order pass comparator function to sort method

Objects in javascript

- An object is a set of key-value pairs, where the values can also be functions.
- Each key-value pair is called a property of the object.
- They are similar to maps, dictionaries, and associative arrays.

```
let car={
    make:"volvo",
    model:"s60",
    price:40000,
};
console.log("make of the car is ", car.make);
console.log("model of the car is ",car.model);
console.log("the price of the car is ",car.price);
```

Using Function constructor

```
function car(make,model,price){
    this.make=make;
    this.model=model;
    this.price=price;
}
let myCar = new car("volvo","s60",40000);
console.log("make of the car is ", myCar.make);
console.log("model of the car is ",myCar.model);
console.log("the price of the car is ",myCar.price);
```

Using Object constructor

```
let myCar=new Object();
myCar.make="volvo";
myCar.model="s60";
myCar.price=40000;

console.log("make of the car is ", myCar.make);
console.log("model of the car is ",myCar.model);
console.log("the price of the car is ",myCar.price);
```

Objects can include functions as members

```
function car(make,model,price,engine){
    this.make=make;
    this.model=model;
    this.price=price;
    this.engine=engine
    this.details=function(){
        console.log(`make : ${this.make} model: ${this.model}price: ${this.price}`);
    }
    this.engine.details=function(){
        console.log(`Displacement: ${this.displacement} horsepower: ${this.horsepower}`);
    }
}
let s60engine={
    cylinders:4,
    displacement:2000,
    horsepower:250
}
let mycar=new car("volvo","s60",250000,s60engine);
mycar.details();
mycar.engine.details();
```

Asynchronous Programming

- By default, javascript works in synchronous

```
function makeGreeting(name) {
        return `Hello, my name is ${name}!`;
}
const name = “paul william";
const greeting = makeGreeting(name);
console.log(greeting); // "Hello, my name is paul william!"
```

makeGreeting () is a **synchronous function** because the caller has to wait for the function to finish its work and return a value before the caller can continue.

What happens if synchronous function takes long time?

What we need is a way for our program to:

1.Start a long-running operation by calling a function.

2.Have that function start the operation and return immediately, so that our program can still be responsive to other events.

3.Have the function execute the operation in a way that does not block the main thread, for example by starting a new thread.

4.Notify us with the result of the operation when it eventually completes.

Need callbacks, promises and async and wait

It is known as Asynchronous programming.

Callbacks

A **callback function** is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action.

There are two ways in which the callback may be called: *synchronous* and *asynchronous*. **Synchronous callbacks** are called immediately after the invocation of the outer function, with no intervening asynchronous tasks, while **asynchronous callbacks** are called at some point later, after an asynchronous operation has completed.

It is used to provide Asynchronicity to javascript

```
document.getElementById('button').addEventListener('click', () => {
  // item clicked
});
```

Don’t know when the user clicks the button, but when the user clicks the respective action will be performed

Without callback

```
var items =[
    {name:"vegetables", price:25},
    {name:"meat", price:18},
    {name:"cake", price:10},
    {name:"drinks", price:12}
]

function getTotalCost(items){
    var totalCost=0;
    for(const item of items){
        totalCost+=item.price;
    }
    return totalCost;
}
function applyTax(totalCost){
    return 1.1*totalCost;
}
var totalCost = getTotalCost(items);
console.log("total cost of items",totalCost);
console.log("total cost of items after
tax",applyTax(totalCost));
```

With callback

```
var items =[
    {name:"vegetables", price:25},
    {name:"meat", price:18},
    {name:"cake", price:10},
    {name:"drinks", price:12}
]

function getTotalCost(items,callback){
    var totalCost=0;
    for(const item of items){
        totalCost+=item.price;
    }
    return callback(totalCost);
}
function applyTax(totalCost){
    return 1.1*totalCost;
}
console.log(“total cost of items after tax”,
getTotalCost(items,applyTax));
```

```
let regularFunction =()=> {
                    return "Hello";
}
let timeoutFunction=(msg)=>{
        var returnMsg;
        setTimeout(()=>{
            console.log("inout message to time out function: ", msg);
            returnMsg=msg+" World";
            console.log("timeout function has constructred returnMsg: ", returnMsg);
            return returnMsg;
        },2000);
}
let followUpFunction = (msg)=> {
        console.log("Follow up timeout returned: ",msg);
}
console.log('Before calling any function');
var regularReturn=regularFunction();
var timoutReturn=timeoutFunction(regularReturn);
followUpFunction(timoutReturn);
```

```
[Running] node "e:\2024-25\Mern\frontend1\synch.js"
Before calling any function
Follow up timeout returned:  undefined
inout message to time out function:  Hello
timeout function has constructred returnMsg:  Hello World

[Done] exited with code=0 in 2.145 seconds
```

Exception Handling

In javascript, exceptions are handled by using

- throw statement
- try---catch statement

Use the throw statement to throw an exception. A throw statement specifies the value to be thrown:

throw expression;

examples

```
throw "Error2"; // String type
throw 42; // Number type
throw true; // Boolean type
throw {
    toString() {
        return "I'm an object!";
     },
};
```

```
function getMonthName(mo) {
    mo--; // Adjust month number for array index (so that 0 = Jan, 11 = Dec)
    const months = [
      "Jan", "Feb", "Mar", "Apr", "May", "Jun",
      "Jul", "Aug", "Sep", "Oct", "Nov", "Dec",
    ];
    if (months[mo]) {
      return months[mo];
    } else {
      throw "InvalidMonthNo"; // throw keyword is used here
    }
  }

  try {
    // statements to try
    monthName = getMonthName(0); // function could throw exception
    console.log(monthName);
   } catch (e) {
       console.log(e);
  }
```

try...catch...finally

```
function f() {
    try {
      console.log(0);
      throw "zero";
    } catch (e) {
      console.log(1);
      // This return statement is suspended
      // until finally block has completed
      return true;
      console.log(2); // not reachable
    } finally {
      console.log(3);
      return false; // overwrites the previous "return"
      console.log(4); // not reachable
    }
    // "return false" is executed now
    console.log(5); // not reachable
  }
  console.log(f()); // 0, 1, 3, false
```