24 April 2024

1.What is Javascript?
Ans–Javascript is a client side as well server side scripting language.It is dynamically typed language.Javascript is functionality object oriented multiparadigm language.It is mainly used to enhance the interest of the user.It is developed by branded each in 1995.

2.How Browsers execute javascript?
Ans–When we open a webpage browsers use their built in Javascript engines to understand and run the JavaScript code on that page.

Example–:

3.What are features of Javascript?
Ans– 1.Dynamic Type
2.Prototype based object oriented programming
3.Functional programming support.
4.Event driven programming
5.Asynchronous programming.
6.DOM Manipulation.
7.Cross Platform compatibility.
8.Standardized.

4.What is DOM?
Ans–DOM stands for Document Object Model, which is a programming interface for web documents. It represents the structure and content of a document as nodes and objects, which allows programs to change the document's structure, style, and content.

5.How to perform click event on DOM using javascript?
Ans–We can perform a click event on a DOM element by selecting the element and then triggering the click event programmatically using Javascript.

Example–:

```
<button onclick=”myfun()”>click me</button>
<script>
    function myfun(){
```

```
        alert(“hii”);
        }
  </script>
```

6.What is alert() and confirm() ?
Ans–alert() and confirm() are functions in Javascript used to display dialog boxes on a webpage. alert() shows an informational message, while confirm() displays a message with OK and Cancel buttons, allowing the user to confirm or cancel an action.

Example–:

7.What is difference between Undefined and null?
Ans–Undefined means a variable has been declared but has not been assigned a value. null is an assignment value representing a variable with no value.

Example–:

```
var x;
console.log(x);//undefined

var y = null;
console.log(y);//null
```

8.What is difference between == and ===?
Ans– == is an equality operator that performs type coercion, while === is a strict equality operator that checks both value and type without coercion.

Example–:

```
1==”1”//true

1===”1”//false
```

9.What is a function?Can we overload function in javascript?
Ans–A function is a block of reusable code. JavaScript does not support function overloading like some other languages.

Example–:

```
function add(a,b){
return a+b;
}
const sum = add(5,8);
console.log(sum);//output=13
```

10.What is a callback ? Explain with suitable example?
Ans–A callback is a function passed as an argument to another function which will be executed later for example in asynchronous programming you might use a callback function to handle the result of an asynchronous operation, like fetching data from a server.

Example–:

11.What is promise?What are advantage of promise over callback.
Ans–Promise in JavaScript is an asynchronous programming concept that is used to handle multiple operations in JavaScript.Promise fails on three parameters, resolves reject and pending

Advantage– 1.improved error handling.
2.more efficient flow control
3.chaining.
4.cleaner code.

Example–:

```
var  promise new = Promise (function(resolve,reject){
        setTimeout(function(){
        resolve(“resolve”);
},3000);
.then(result){
        console.log(“promise resolve”);
}).catch(error)
        console.log(“promise reject”);
}
)
```

12.What is callback hell?

Ans–Callback hell is a phenomenon where a Callback is called inside another Callback. It is the nesting of multiple Callbacks inside a function. If you look at the design of the code, it seems just like a pyramid. Thus the Callback hell is also referred to as the 'Pyramid of Doom'.

Example:--

Asynchronous–

```
console.log(“start”);
setTimeout(function(){
        console.log(“kapil”);
},2000)
console.log(“end”);
```

Output–:start
end
kapil

13.What is closure?
Ans–A closure is the combination of a function bundled together with references to its surrounding state (the lexical environment).

Example–:

```
function student(){
        var name = ‘kapil’;
        function employee(){
                console.log(name);
        }
        employee();
}
student();
```

14.What is IIFEs(Immediately Invoked Function expression).

Ans–IIFE stands for Immediately Invoked Function Expression. It is a JavaScript function that runs as soon as it is defined. IIFEs are created by wrapping a function expression in parentheses and following it with another set of parentheses.

Example–:

```
(function (){
        var name = “kapil”;
        console.log(name);
})();
```

15.What is anonymous function?
Ans–An anonymous function is a function without a name. It's often used as a callback or immediately invoked without being stored in a variable.

Example–

```
const add = function (x,y){
        return x+b;
};
const sum = add(5,5);
console.log(sum);
```

16.What is hoisting in javascript.
Ans–Hoisting is a javascript mechanism where variables and function declarations are moved to the top of their containing scope during the compilation phase.
function expressions and class expressions are not hoisted.
Example–

```
employee()
function employee(){
        console.log(“kapil”);
}
```

17.What is the difference between var,let and const keywords?

Ans–var (ES6) has function scope, while let and const have block scope. const variables cannot be reassigned, while let variables can excess.

18.Explain use strict?
Ans–use strict is a directive used to enable strict mode in javascript, which helps catch common coding errors and restricts some unsafe features.

19.What is event bubbling and event capturing?
Ans–Event bubbling–:When an event happens on a child element, it's handled first on that child element, then it moves up to its parent elements. So, the event travels from bottom to top.

Event capturing–:When an event occurs on a child element, it's initially handled by its parent elements before reaching the child element itself. This means the event moves from top to bottom.

20.What are the primitive data type in javascript?
Ans–The primitive data types in JavaScript are string, number, boolean, undefined, null, symbol and BigInt.

25 April 2024

1.What are different types of popup boxes available in javascript?
Ans–:They are three types popup boxes available in javascript – alert(), confirm() and prompt().

2.What will happen if an infinite while loop is run in javascript?
Ans–: If we run an infinite loop in JavaScript, it will keep running forever, using a lot of CPU and memory, possibly causing your browser or JavaScript environment to hang.

3.List HTML DOM mouse event

onclick , onmouseover , onmouseup , onmove , in javascript?

Ans–:

4.Explain await and async ? How to use await and async?
Ans–:Async/await is a feature in js that allows you to work with asynchronous code in a more synchronous like manner,making it easier to write and understand asynchronous code.Async function always return a promise Await keyword is used only in async function to wait for promise

Example–

```
async function fetchData(){
        try{
                const response = await fetch('https://kpl.example.com/data');
                const data = await response.json();
                console.log(data);
        }catch (err){
                console.log('error fetching data',err);
        }
}
 fetchData();
```

5.How to handle the exception in javascript?
Ans–:We can handle exceptions in javascript using try, catch, and finally blocks. try is used to test a block of code for errors, catch is used to handle the error, and finally is used to execute code after the try and catch regardless of the result.

6.How to get the last index of a string in javascript lastIndexOf?
Ans–:We can use the lastIndexOf() method to find the index of the last occurrence of a specified value within a string.

7.Describe negative infinity in javascript?
Ans–:Negative infinity is a special numeric value that is returned when an arithmetic operation or mathematical function generates a negative value greater than the largest representable number in javascript.

Example–:

8.What is Node.js? Explain the advantage of Node.js over java and php?
Ans–:Node.js is a javascript runtime built on Chrome's V8 javascript engine. It allows developers to run javascript on the server-side. Node.js is known for its non-blocking, event-driven architecture, which makes it efficient for handling concurrent requests.

9.What are the limitation of Node.js?
Ans–;How are some of the limitation of Node.js–:

1.Single-threaded.
2.Not suitable for CPU-intensive tasks.
3.Callback based programming model.
4.Unstable API.
5.Lack of library support

10.How Node.Js Works?
Ans–:Node.js uses an event-driven, non-blocking I/O model, which makes it lightweight and efficient for handling concurrent requests. It uses the V8 JavaScript engine to execute javascript code.

Here is a step-by-step explanation of how Node.js handles a request from a client:-

1.The client sends a request to the server.
2.The server receives the request and places it in the event queue.
3.The event loop checks if the event queue is empty. If it is not empty, the event loop removes the first request from the queue and executes it.
4.If the request is an I/O operation, the server initiates the operation and provides a callback function.
5.The server continues to process other requests while waiting for the I/O operation to complete.
6.When the I/O operation completes, the callback function is executed.

7.The callback function performs the necessary actions to handle the request and sends a response to the client

11.How Node.js is single threaded?
Ans–:Node.js is single-threaded in the sense that it uses a single event loop to handle all asynchronous operations. However, it can offload certain tasks to worker threads or child processes to take advantage of multiple CPU cores.

12.Explain any five built-in package/Dependency name in node.js?
Ans–:Some built-in packages in Node.js include http for creating HTTP servers, fs for file system operations, path for handling file paths, os for operating system information, and events for event handling.

13.What is module in node.js?
Ans–:A module in Node.js is a reusable block of code that encapsulates related functionality. Modules can be imported and used in other modules using require().

14.What is Module.exports in javascript?
Ans–:Module.exports is a special object in Node.js used to define what a module exports as its public interface. It can be used to export functions, objects, or primitive values from a module.

15.How to create server in node.js?
Ans–:We can create a server in Node.js using the http module. First, create an instance of http.Server, then use its listen() method to start listening for incoming requests.

16.How Node.js handle multiple request?
Ans–:Node.js uses an event-driven, non-blocking I/O model, which allows it to handle multiple requests concurrently by utilizing a single thread and asynchronous operations.

17.How to use url module in node.js?

Ans–:We can use the url module in Node.js to parse URLs and access their components. we can create a URL object using url.parse() and then access its properties

18.What is setInterval,setTimeout ?
Ans–:The setTimeout() method is used to call a function after a certain period of time. The setInterval() javascript method is used to call a function repeatedly at a specified interval of time.

19.What is __dirname and __filename in javascript ?
Ans–:__dirname is a global variable in Node.js that represents the directory name of the current module. __filename represents the filename of the current module.

20.What is synchronous/Blocking and Asynchronous/Non-blocking code in node.js?
Ans–:Synchronous/blocking code execute one task at a time,blocking further execution until the current task is completed.Asynchronous/non-blocking code allows multiple task to be execute concurrently,without waiting for the previous task to complete.

27 April 2024

1.What is file system in node.js?
Ans–:The file system module in node.js provide function for interacting with the file system,such as reading and writing,a for appending and for updating.

2.What are the difference types of flags used in node.js?
Ans–:In node.js flags are used to modify the behavior of the fs module function common flags include r for reading w for writing a for appending and + for updating.

3.What is a stream in node.js?explain the types of streams.
Ans–:A stream is an abstract interface for working with streaming data in node.js there are four types of stream:-Readable,Writable,Duplex and Transform.

4.How to pipe stream in node.js?
Ans–:We can pipe streams together using the pipe() method.This allows we to easily transfer data from one stream to another stream without manually handling data events.

5.What is request and response in node.js?
Ans–:In node.js request represents an incoming HTTP request from a client while a response represents the server response to that request.

6.What is package.json and package-lock.json?
Ans–Package.json is a metadata file in node.js project that contain information about the project such as dependencies and scripts package-lock.json is a file automatically generated by npm to lock down the version of dependencies.

7.What is npm? How to install a dependency/module at the application level and environment level?
Ans–npm is a package manager for node.js we can install a dependency/module at the application level using npm install <package name> or at the environment level using -g flag for global installation.

8.How do you manage package in our node.js project?
Ans–We can manage package in our node.js project by using npm commands such as npm install npm uninstall npm update and npm list.

9.How in node.js better than other frameworks?
Ans–Node.js is better than other frameworks for certain use cases due to its non blocking event-driven architecture which makes it highly efficient for handling I/O heavy application and concurrents.

10. What are some commonly use timing features of node,js?
Ans–Some commonly used timing features of node.js include setTimeout,setInterval,and process.nextTick().

11.What is fork in node.js?
Ans–Fork is a method in node.js used to create child processes its commonly used to create multiple instance of a node.js application to take advantage of multiple CPU cores.

Example–

```
const { fork } = require('child process');
```

```
const childProcess = fork ('childscript.js');
childProcess.send('hello childprocess');
childProcess.on('message',(msg)=> {
console.log(msg);
});
```

12.How do you create a simple server in node.js that returns "hello world"?
Ans–We can create a simple server in node.js using the http module here an example–:

```
let http = require('http');

http.createServer(function (req,res){
        res.write('hello world');
        res.end();
}).listen(3000);
```

13.How many types of API functions are there in Node.js?
Ans–There are two types of API functions in Node.js:

1. Asynchronous(Blocking).
2. Synchronous, and Non-blocking.

15.What is the purpose of module.exports?
Ans–Module.export is a js object that is used to export function object or primitive values from a module so that they can be used in other module.it is part of the CommonJS specification which is a standard for javascript modules. A module is imported using the require().

16.What is an event loop in Node.js?
Ans–The event loop is the core of Node.js' asynchronous, event-driven architecture. It continuously checks the event queue for pending events and processes them one at a time.

17.If Node.js is single-threaded then how does it handle multiple requests/concurrency?
Ans– Node.js uses asynchronous,non-blocking I/O operations and an event loop to handle multiple requests concurrently within a single thread.

18.Differentiate between process.nextTick() and setImmediate().
Ans–process.nextTick() executes a callback function asynchronously at the end of the current event loop iteration,while setImmediate() schedules a callback function to be executed on the next iteration of the event loop.

19.What is Node.js stream?
Ans–Stream are one of the fundamental concept of Node.js Stream are a type of data-handling methods and are used to read or write input output sequentially.Streams are used to handle reading/writing files or exchanging information in an efficient way.

1.Writable–We can write data to these streams.
Exa–fs.createWriteStream().
2.Readable– We can read data from these streams.
Exa–fs.createReadStream().
3.Duplex–Streams that are both writable as well as readable.
Exa–net.socket.
4.Transform–Streams that can modify or transform the data as it is written and read.
Exa–zlib.createDeflate.

20.What is middleware?
Ans– Middleware functions are functions that have access to the request object (req), the response object (res), and the next function in the application's request-response cycle. The next function is a function in the Express router which, when invoked, executes the middleware succeeding the current middleware.

Example:--

```
app.use(function(req, res, next) {
  console.log('Request:', req);
  console.log('Response:', res);
  next();
});
```

01 May 2024

1. Explain what a reactor pattern in node.js?
Ans– The reactor pattern, also known as the event-driven architecture, is a fundamental design pattern used in Node.js and other event-driven systems to handle I/O operations asynchronously without blocking the main execution thread

2. Describes the exit code of Node.js?
Ans– The exit code is a simple way for Node.js processes to report their status when they finish running. A code of 0 means success,If we run a Node. js script, and it completes its tasks without encountering any issues, it will typically return an exit code of 0.

3. What is an EventEmitter in node.js?

Ans– EventEmitter is a class in node. js that is responsible for handling the events created using the 'events' module in node. js.

```
 import EventEmitter from 'events';

const myevent  = new EventEmitter();

myevent .on('myEvent', () => {
  console.log('Event handler for myEvent');
});
myevent .emit('myEvent');
```

removeListener(eventName, listener):
Removes a specific listener for the specified eventName.
once(eventName, listener):
Registers a one-time listener for the eventName. The listener is automatically removed after being invoked once.

4.What is a thread pool and which library handles it in node.js?
Ans– thread pool js is responsible for executing synchronous tasks such as file I/O, network operations, and other CPU-bound operations efficiently in parallel
managed by Libuv

5. What is purpose of NODE_ENV?
Ans– NODE_ENV is an environment variable that stands for Node environment in the Express server. The NODE_ENV environment variable specifies the environment in which an application is running,

```
export DATABASE_URL=mongodb://localhost/mydatabase
const databaseUrl = process.env.DATABASE_URL;

require('dotenv').config();

console.log(process.env.DATABASE_URL);
```

6.How would you connection mongodb database to node application?
Ans– import mongoose from 'mongoose'

```
mongoose.connect("mongodb://localhost:27017");
.then(()=>{
console.log("connected");
}).catch((err)=>{console.log(err)});
```

7. What are different type of http request?
Ans– GET: Retrieve data from the server.
POST: Submit High amount data to be processed to a specified resource.
PUT : Update an existing resource or create a new resource if it does not exist at a specified URL.
DELETE : Delete a specified resource at a given URL.
PATCH : Apply partial modifications to a resource.

8. What is difference between get and post?
Ans– GET is for fetching data, appending parameters in the URL,. POST, used for updates, sends data securely in the request body, perfect for forms

9.What is query string and how to send the data in get request?

Ans– A query string is a part of a URL that attaches information to the end. It's essentially a way to pass data to a web server when making a GET request.
http://localhost:8000/api/getProfile?parameter1=value1&parameter2=value2

10.What is the use body parser?
Ans– the body-parser middleware was commonly used to parse incoming request bodies in various formats like JSON, URL-encoded, and multipart form data and make them available in req.body for easy handling in route handlers and middleware.

Example–

```
const express = require(‘express’);
```

```
const bodyParser = require('body-parser');

const app = express()'

app.use(bodyParser.urlencoded({ extended:true}));

app.post('/submit',(req,res)=>{
        const name = req.body.name;
        const email = req.body.email;
        const message = req.body.message;

console.log(`Received form submission: Name - ${name}, Email - ${email}.
Message-${message}`);

        res.send(`Form submitted successfully!`);
        });
        const.listen(port,()=>{

        console.log(`Server is running on http://localhost:${port}`);
        })
```

12.What are types of Middleware in express ? Explain with suitable example?
Ans– application-level middleware include logging middleware, authentication middleware, and error handling middleware.

Application Level Middleware

```
app.use((req, res, next) => {
  console.log("abcd");
  next();
});
```

route Level Middleware

```
router.get('/profile', (req, res) => {
 console.log("abcd");
});
```

Error handling Middleware

```
app.use((err, req, res, next) => {
  console.error(err);
 });
```

Built-in Middleware:

```
app.use(express.json());
```

Third-party Middleware:

```
import cors from  "cors"
app.use(cors());
```

13.Does order of middleware matters in express.?
Ans– The order of middleware registration matters and each middleware function can modify the request and response objects or terminate the request-response cycle. app. use((req, res, next) => { console.log("helloo");

14. What is express.js?
Ans– Express.js is a back-end web application framework for Node.js, released as free and open-source software under the MIT License. It is designed for building web and mobile applications. It has been called the de facto standard server framework for Node.js

15.What are some distinctive features of Express?
Ans– Middleware, Routing, HTTP Interceptors, Lightweight and Unopinionated, fast Minimalist and flexible,Community Support.

16.Is Express.js front-end or backend framework?

Ans– Express.js is a backend framework. It's specifically designed to simplify the development of the server-side logic of web applications.

17.Why do we use express.js?
Ans– Express.js provides a structured and efficient way to build web applications on Node.js. It streamlines development, promotes clean code organization

Simplified Structure
Routing
Middleware
Templating
Faster Development:

18.What is difference between express.js and node.js?
Ans– Node.js is a runtime environment that executes JavaScript code on the server-side, whereas Express.js is a web framework that runs on top of Node.js,

Node.js provides low-level APIs for handling network and file system operations, whereas Express.js by providing higher-level abstractions like routing and middleware.

express do simplify our code management

19.What do you understand by Scaffolding in Express.Js?
Ans– Scaffolding in Express.js refers to the process of generating a basic directory structure and files for a new Express.js project using a tool or generator. This helps kickstart development by providing a starting point with commonly used files and folders..

20.Which are the argument available to an Express.Js route handler function?
Ans– Request Object (req):
The req object holds information about the incoming request from a client's web browser It contains details like the URL that was requested, other data like form data or parameters, and other information related to the request.

Response Object (res):
the response object is use to send response back to the client's web browser
like HTML, JSON, or plain text

Next Function (next) :
The next function is used to pass control to the next function in the sequence of request handlers.

21.How can you allow CORS in express.js?
Ans– We can allow CORS in express.js by using the cors middleware.We can install it via npm and then use it in our express.js application to enable cors for specific routes or all routes.

22.How can you deal with error handling in express.js?explain with an example?
Ans–We can deal with error handling in express.js by defining error-handling middleware functions that take four arguments:err,req,res and next.These functions can catch errors that occur during request processing and handle them appropriately,such as sending an error response to the client or logging the error.

Example:--

```
app.use((err,req,res,next)=>{
        console.log(err.stack);
        res.status(500).send('something went wrong');
});
```

23.Write code to start serving static files in express.js?
Ans–We can start serving static files in express.js by using the express.static() middleware and specifying the directory containing the static files.

Example:--

```
const express = require('express');
const app = express();

app.use(express.static('public'));

app.listen(3000,()=>{
        console.log('Server running at http://localhost:3000/');
});
```

24.How can we render plain HTML in express?
Ans– We can render plain HTML in express.js by using the res.send() method to send the HTML content as a response.

Example–

```
app.get('/',(req,res)=>{
        res.send('<h1>Hello kapil</h1>');
});
```

25.How can we send data while rendering a page in express?
Ans– To send data while rendering a page in express we can use the res.render() function.This function takes two parameters: the name of the view to render and an object containing the data to be sent to the view.

Example–:

```
res.render('index.ejs',{title:'my page'});
```

26.How to enable debugging in express app?
Ans– We can enable debugging in an express app by setting the DEBUG environment variable to a specific namespace or wildcard.
For example we can use DEBUG=myapp:* to enable debugging for all modules in the myapp namespace.

27.What is routing and how dose routing work in express?
Ans– Routing is the process of determining how an application responds to a client request to a particular endpoint.This endpoint consists of a URI and an HTTP method such as GET,POST,PUT,DELETE etc.Routes can be either good old web pages or REST API endpoint.

Example–:

```
const express = require('express');
const app = express();

app.get('/',(req,res)=>{
        res.send('Hello kapil');
});
app.post('/users',(req,res)=>{
        Const user = new User (req.body);
        user.save();
        res.send(user);
});
app.listen(3000,()=>{
        console.log('app listening on port 3000');
});
```

28.How dose dynamic routing work in express.js?
Ans– Dynamic routing in express.js allows we to define routes with URL parameters,which can be used to handle variable parts of the URL.These parameters are specified using a colon(:) followed by the parameter name in the rout definition.

Example–:

```
Const  express = require('express');
Const app = express();

app.get('/users/:id,(req,res)=>{
        Const userID = req.params.id;
        res.send(``User ID : ${userId}`);
});

Const PORT = process.env.PORT || 3000;
app.listen(PORT,()=>{
        console.log(`server is running on port ${PORT}`);
});
```

React

1.What is React.Js ? Explain the features of React.Js?
Ans-React.js is a popular JavaScript library for building user interfaces, especially single-page applications. Developed by Facebook, it allows developers to create reusable UI components and manage the state of their applications efficiently.

Features of React.js--

1.JSX (JavaScript XML): A syntax extension that looks similar to XML or HTML. It allows you to write HTML structures in the same file as JavaScript code.

2.Components: Building blocks of a React application. They can be functional or class-based.

3.Virtual DOM: React maintains a virtual representation of the actual DOM, which helps in efficient updates.

4.Unidirectional Data Flow: Data flows in a single direction, making the app more predictable and easier to debug.

5.State Management: React provides local state management within components and can be extended with libraries like Redux for global state management.

6.Lifecycle Methods: Special methods that are called at different stages of a component's life

---

2.What is defference b/w Actual DOM and Virtual DOM ?
Ans-1.Actual DOM: The Document Object Model (DOM) represents the UI of your application. Updating the actual DOM is slow because it requires re-rendering the entire page or large portions of it.

2.Virtual DOM: A lightweight copy of the actual DOM. React uses it to detect changes and updates the actual DOM efficiently by only rendering the necessary parts. This makes the updates faster and improves performance.

3.What is component in React?
Ans-A Component in React is a self-contained module that renders part of the UI. Components can be thought of as custom HTML elements, and they can be reused throughout the application.

4.How many types of component in React? Explain class component and functional component?
Ans-There are two main types of components in React:

1.Class Components: These are ES6 classes that extend from React.Component. They can hold and manage their own state and lifecycle methods.

Example-:

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

2.Functional Components: These are simple JavaScript functions that take props as an argument and return React elements. With the introduction of hooks, functional components can also manage state and lifecycle.

Example-:

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

5.Explain the life cycle of class component (Mounting,Unmounting,Updating)?
Ans-(A)Mounting: When a component is being inserted into the DOM.

constructor()
static getDerivedStateFromProps()
render()
componentDidMount().

(B)Updating: When a component is being re-rendered due to changes in state or props.

static getDerivedStateFromProps()
shouldComponentUpdate()
render()
getSnapshotBeforeUpdate()
componentDidUpdate().

(C)Unmounting: When a component is being removed from the DOM.

componentWillUnmount().

6.How to implement life cycle phase(Mounting,Updating,Unmounting) using functional component?
Ans-Functional components use hooks to manage lifecycle events:

(A)Mounting-:Use useEffect with an empty dependency array.

Example-useEffect(() => {

}, []);

(B)Updating-:Use useEffect without an empty dependency array.

Example- useEffect(() => {

});

(C)Unmounting-:Use useEffect with a cleanup function.

```
Example- useEffect(() => {
  return () => {

  };
}, []);
```

7.What do you mean by statefull and stateless component ?
Ans- 1.Stateful Component: Components that can hold and manage their own state. Typically class components or functional components using hooks.

2.Stateless Component: Components that do not manage state. They simply take props and render them. Usually functional components.

8.What is the difference b/w state and props?
Ans-1.State: Managed within the component and can change over time. Used for dynamic data that affects the rendering of the component.

Example- this.state = { count: 0 };

2.Props: Short for properties, passed from parent to child components. They are read-only and cannot be modified by the child component.

Example- <ChildComponent name="John" />

9.What is the difference b/w Element and components?
Ans- Element: A plain object describing what you want to appear on the screen. Created using JSX or React.createElement().

Example- const element = <h1>Hello React</h1>;

Component: A function or class that returns a React element. Used to build more complex UIs by composing elements.

```
Example- function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

10.What happen when we call setState() method ?

Ans- Calling setState triggers a re-render of the component. React merges the new state with the existing state, and then updates the component and its children.

11.Can we re-render component without using setState?
Ans- Yes, a component can re-render if:

a)Its parent component re-renders and passes new props.
b)Using hooks like useState or useReducer in functional components.
c)Using forceUpdate() in class components (not recommended).

12.What is eager loading and lazy loading in react?
Ans- Eager Loading: Loading components or resources at the start of the application. All necessary files are downloaded upfront.

Lazy Loading: Delaying the loading of components or resources until they are needed. This improves initial load time and performance.

13.How we can load the components eagrly and lazily?
Ans-Eager Loading: Import the component normally.
Example- import MyComponent from './MyComponent';

Lazy Loading: Use React.lazy and Suspense.
Example- const MyComponent = React.lazy(() => import('./MyComponent'));

```
function App() {
  return (
    <React.Suspense fallback={<div>Loading...</div>}>
      <MyComponent />
    </React.Suspense>
  );
}
```

14.What is error boundaries in react?
Ans-Error boundaries are React components that catch JavaScript errors in their child component tree, log them, and display a fallback UI instead of crashing the whole app.

```
Example- class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }
```

```
  static getDerivedStateFromError(error) {
    return { hasError: true };
  }

  componentDidCatch(error, errorInfo) {

  }

  render() {
    if (this.state.hasError) {
      return <h1>Something went wrong.</h1>;
    }
    return this.props.children;
  }
}
```

15.How to implement routing in React?What is use of BrowserRouter and Routes?
Ans- React Router is used for client-side routing.

BrowserRouter: Wraps the app and enables routing using HTML5 history API.

Routes: Defines the different routes in the application.

Example-

```
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';

function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </Router>
  );
}
```

16.What is props deilling in React?
Ans- Props drilling refers to the process of passing data from a parent component to deeply nested child components through multiple layers of intermediate components.

17.What is pure component in React ? How create pure class component?

Ans- A pure component in React is a component that renders the same output for the same state and props. It implements shouldComponentUpdate with a shallow prop and state comparison.

Example-

```
class MyComponent extends React.PureComponent {
  render() {
    return <div>{this.props.name}</div>;
  }
}
```

18.How to use function component as pure component ?

Ans- Functional components can be used as pure components by using React.memo.

Example-

```
const MyComponent = React.memo(function MyComponent(props) {
  return <div>{props.name}</div>;
});
```

19.What is Higher Order Component (HOC) ?

Ans- A Higher Order Component is a function that takes a component and returns a new component with additional props or functionality.

Example-

```
function withLogging(WrappedComponent) {
  return class extends React.Component {
    componentDidMount() {
      console.log('Component mounted');
    }
    render() {
      return <WrappedComponent {...this.props} />;
    }
  };
}
```

20.What is protected/private route in React.js ?

Ans-Protected routes are routes that only accessible to authenticated users. They are implemented using React Router by checking authentication status before rendering the route component.

Example-

```
import { Route, Redirect } from 'react-router-dom';

function PrivateRoute({ children, ...rest }) {
```

```
  return (
    <Route
      {...rest}
      render={({ location }) =>
        isAuthenticated() ? (
          children
        ) : (
          <Redirect
            to={{
              pathname: '/login',
              state: { from: location },
            }}
          />
        )
      }
    />
  );
}
```

21.What are React fragements?

Ans- React Fragments let you group a list of children elements without adding extra nodes to the DOM. They are useful when you need to return multiple elements from a component's render method. You can use the <React.Fragment> tag or the shorthand <></>.

Example-

```
// Using React.Fragment
return (
  <React.Fragment>
    <ChildA />
    <ChildB />
  </React.Fragment>
);

return (
  <>
    <ChildA />
    <ChildB />
  </>
);
```

22.What are the hooks in React.js? Explain some hooks which used in your project ?

Ans- Hooks are special functions that let you "hook into" React state and lifecycle features from function components. Some common hooks include:

1.useState: For state management in functional components.
2.useEffect: For performing side effects in function components.
3.useContext: For accessing context values.
4.useRef: For accessing and manipulating DOM elements directly.
5.useMemo: For memoizing expensive calculations.
6.useCallback: For memoizing callback functions.

23.Explain following hooks?
Ans- a) useState- useState is a hook that allows you to add state to functional components.

Example-
```
 const [count, setCount] = useState(0);
```

b) useLocation- useLocation is a hook from react-router-dom that returns the current location object, containing information about the current URL.

Example-
```
import { useLocation } from 'react-router-dom';

const location = useLocation();
console.log(location.pathname); // Current path
```

c) useNavigate- useNavigate is a hook from react-router-dom that allows you to navigate programmatically.

Example-

```
import { useNavigate } from 'react-router-dom';

const navigate = useNavigate();
navigate('/path');
```

d) useRef- useRef is a hook that returns a mutable ref object whose .current property is initialized to the passed argument.

Example-

```
const inputRef = useRef(null);
<input ref={inputRef} />;
```

e) useMemo- useMemo is a hook that memoizes a value. It recomputes the value only when one of its dependencies changes.

Example-

```
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);
```

f) useCallback- useCallback is a hook that returns a memoized callback function.

Example-

```
const memoizedCallback = useCallback(() => {
  doSomething(a, b);
}, [a, b]);
```

g)useDispatch- useDispatch is a hook from react-redux that returns the dispatch function to dispatch actions to the Redux store.

Example-

```
import { useDispatch } from 'react-redux';

const dispatch = useDispatch();
dispatch(someAction());
```

h) useContext- useContext is a hook that accesses the value of a React context.
Example-

```
const value = useContext(SomeContext);
```

I)useSelector- useSelector is a hook from react-redux that allows you to extract data from the Redux store state

Example -

```
const value = useSelector(state => state.someValue);
```

j) useEffect- useEffect is a hook that runs side effects in functional components. It can replace lifecycle methods like componentDidMount, componentDidUpdate, and componentWillUnmount.

Example-

```
useEffect(() => {
  // Code to run on mount and update
  return () => {
    // Code to run on unmount
  };
}, [dependencies]);
```

24.In which life cycle method you would like to make a http call?

Ans- In class components, the componentDidMount lifecycle method is typically used to make HTTP calls, as it runs after the component has been rendered.

25.Where to make a api/http/network call in functional components?
Ans- In functional components, you can make API calls inside the useEffect hook.

Example-

```
useEffect(() => {
  fetchData();
}, []); // Empty dependency array ensures this runs once on mount
```

26.Write a code using useEffect to implement ,ounting,updating and unmounting in functional component?
Ans-

```
useEffect(() => {

  console.log('Component mounted');

  return () => {

    console.log('Component unmounted');
  };
}, []);

useEffect(() => {

  console.log('Component updated');
});
```

27.Can we update props?
Ans- No, props are read-only. We can pass new props from parent components to update the data.

28.How to update parent state data from child component?
Ans- We can pass a callback function from the parent to the child component as a prop, and then call this function from the child to update the parent state.

Example-

```
// Parent Component
const Parent = () => {
  const [parentState, setParentState] = useState('');

  const updateParentState = (newState) => {
    setParentState(newState);
  };
```

```
  return <Child updateParentState={updateParentState} />;
};

// Child Component
const Child = ({ updateParentState }) => {
  return <button onClick={() => updateParentState('new state')}>Update
Parent</button>;
};
```

29.What is context api in React.js?
Ans- The Context API provides a way to pass data through the component tree without having to pass props down manually at every level. It is useful for global state management.

30.What is Redux? Explain the component of React(Store,Action,Reducer)?
Ans- Redux is a state management library for JavaScript apps. It provides a predictable state container.

1.Store: Holds the application state.
2.Action: An object that describes a change or event in the application.
3.Reducer: A pure function that takes the current state and an action and returns a new state.

31.What is difference b/w useState and useReducer ?
Ans- 1.useState: Simplest way to add state to a functional component.
2.useReducer: More complex state logic, useful for managing multiple state transitions or complex state logic.

32.What is call,bind,apply in JS ?
Ans- 1.call: Calls a function with a given this context and arguments.
2.apply: Similar to call, but takes arguments as an array.
3.bind: Creates a new function that, when called, has its this keyword set to the provided value.

33.Differnce b/w context API na Redux ?
Ans- 1.Context API: Simpler and good for passing data through the component tree.
2.Redux: More powerful and provides a more structured approach to state management with middleware and dev tools.

34.What is RTK ?
Ans- RTK stands for Redux Toolkit. It is the official, recommended way to write Redux logic, providing utilities to simplify common Redux tasks.

35.What is the use async thunk (createAsyncThunk)?
Ans- createAsyncThunk is a utility from Redux Toolkit that helps in writing Redux actions that handle asynchronous logic like fetching data.

36.What is memo in React.js?
Ans- memo is a higher-order component that memoizes a component, preventing unnecessary re-renders when the props have not changed.

Example-

```
const MemoizedComponent = React.memo(MyComponent);
```

37.What is JSX ?
Ans- JSX stands for JavaScript XML. It allows you to write HTML elements in JavaScript and place them in the DOM.

38.What is reconciliation?
Ans- Reconciliation is the process React uses to update the DOM efficiently. It compares the current elements with the previous elements and only updates what has changed.

39.What is controlled component and uncontrolled component ?
Ans- 1.Controlled Component: Form data is handled by a React component's state.
2.Uncontrolled Component: Form data is handled by the DOM itself.

40.What dose shouldComponentUpdate do and why is it important ?
Ans- shouldComponentUpdate is a lifecycle method in class components that determines whether the component should re-render or not. It can improve performance by preventing unnecessary renders.

Example-

```
shouldComponentUpdate(nextProps, nextState) {
  return nextProps.value !== this.props.value;
}
```

41.Describe how events are handled in React?
Ans- React events are handled using a declarative approach, which allows developers to create dynamic and interactive user interfaces.

Example-

```
function MyButton() {
  function handleClick() {
    alert('Button clicked!');
```

```
  }

  return (
    <button onClick={handleClick}>Click me</button>
  );
}
```

42.What is the second argument that can optionally be passed to setState and what is its purpose?
Ans- The second argument to setState is a callback function that is executed after the state has been updated and the component has re-rendered. This is useful for performing actions that depend on the updated state

Example-

```
this.setState({ count: this.state.count + 1 }, () => {
  console.log('State updated:', this.state.count);
});
```

43.List down the variation of setState() in React.js?
Ans- 1.Object form: Pass an object to update the state.

Example-

```
this.setState({ name: 'John' });
```

2.Function form: Pass a function that receives the previous state and props, and returns an object to update the state.

Example-

```
this.setState((prevState, props) => ({
  count: prevState.count + 1
}));
```

44.What is props.childrens?
Ans- props.children is a special prop that is used to pass child elements to a component. It allows you to create components that wrap other components or elements.

Example-

```
function Wrapper({ children }) {
  return <div className="wrapper">{children}</div>;
}

function App() {
```

```
  return (
    <Wrapper>
      <h1>Hello, world!</h1>
    </Wrapper>
  );
}
```

45.Does React use HTML ?
Ans- React does not use HTML directly. Instead, it uses JSX (JavaScript XML), which looks similar to HTML but is actually JavaScript. JSX is then compiled into React.createElement calls that produce React elements.

46.Tell me most signification drawback of React.js?
Ans- One significant drawback of React is the steep learning curve for beginners, especially when learning concepts like JSX, state management, and the various lifecycle methods. Additionally, frequent updates and new features require developers to continuously learn and adapt.

47.Explaing React Router ?
Ans- React Router is a library used for routing in React applications. It allows you to define routes in your application and render different components based on the URL.

Example-

```
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import Home from './Home';
import About from './About';

function App() {
  return (
    <Router>
      <Switch>
        <Route exact path="/" component={Home} />
        <Route path="/about" component={About} />
      </Switch>
    </Router>
  );
}
```

48.How modify each request and response (Interceptor) in React.js?
Ans- To modify requests and responses, you typically use a library like Axios that supports interceptors.

Example-
```
import axios from 'axios';
```

```
axios.interceptors.request.use(request => {
  console.log('Starting Request', request);
  return request;
});

axios.interceptors.response.use(response => {
  console.log('Response:', response);

  return response;
});
```

49.What is REST API? How to call API from React?
Ans- A REST API is an application programming interface that uses HTTP requests to GET, PUT, POST, and DELETE data. To call an API from React, we can use the fetch API or Axios.

Example-

```
useEffect(() => {
  fetch('https://api.example.com/data')
    .then(response => response.json())
    .then(data => setData(data))
    .catch(error => console.error('Error:', error));
}, []);
```

50.How to send file in request from React ?
Ans- To send a file in a request, we typically use a FormData object and Axios or fetch.

Example-

```
const formData = new FormData();
formData.append('file', file);

axios.post('https://api.example.com/upload', formData)
  .then(response => console.log(response))
  .catch(error => console.error('Error:', error));
```

51.Why should we not update the state directly ?
Ans- In React, you should not update state directly because React might not detect the change and the component might not re-render as expected.

52.What is the purpose of callback function as an argument of setState()?
Ans- The callback function as an argument of setState is used to execute code after the state has been updated and the component has re-rendered.

Example-

```
this.setState({ count: this.state.count + 1 }, () => {
  console.log('State updated:', this.state.count);
});
```

53.How to bind method and event handlers in JSX?
Ans- In class components we bind methods in the constructor.
Example-

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    console.log('Button clicked');
  }

  render() {
    return <button onClick={this.handleClick}>Click me</button>;
  }
}
```

In functional components, you use arrow functions.

Example-

```
 function MyComponent() {
  const handleClick = () => {
    console.log('Button clicked');
  };

  return <button onClick={handleClick}>Click me</button>;
}
```

54.What are synthetic event in React.js?
Ans- Synthetic events are React's cross-browser wrapper around the browser's native event system. They provide a consistent API regardless of the browser.

Example-

```
function MyButton() {
  function handleClick(event) {
    console.log(event.type); // Synthetic event
```

```
  }

  return <button onClick={handleClick}>Click me</button>;
}
```

55.What are inline conditional expression ?
Ans- Inline conditional expressions allow you to conditionally render elements in JSX.

Example-

```
function MyComponent({ isLoggedIn }) {
  return (
    <div>
      {isLoggedIn ? <p>Welcome back!</p> : <p>Please log in.</p>}
    </div>
  );
}
```

56.What is key props and what is the benefit of using it in arrays of elements ?
Ans- The key prop helps React identify which items have changed, are added, or are removed. Keys should be unique among siblings to ensure efficient updates.

Example-

```
const listItems = items.map((item) =>
  <li key={item.id}>{item.name}</li>
);
```

57.What is the use refs?
Ans- Refs are a function that use to access the DOM from components.

58.How to create refs ?
Ans- To create refs in React, you can use the React.createRef() function or the useRef() hook.

Example-

```
class Form extends Component {
  inputRef = React.createRef();
}
```

59.What are forword ref ?
Ans- Forward refs are used to pass refs through a component to one of its children.

Example-

```
const MyInput = React.forwardRef((props, ref) => (
  <input ref={ref} {...props} />
));

class Parent extends React.Component {
  constructor(props) {
    super(props);
    this.inputRef = React.createRef();
  }

  componentDidMount() {
    this.inputRef.current.focus();
  }

  render() {
    return <MyInput ref={this.inputRef} />;
  }
}
```
