VIVA QUESTIONS IP

1. Write an Arrow Function.
2. Tell features of React.
3. Inheritance.(all 3)
4. Advantages of expreejs, react,node js
5. Iterator generator
6. Promise in react
7. Create textbox in react
8. Hooks definition
9. What is referencing
10. Initialise buffer
11. Sample program for promises
12. Difference in session and cookies
13. Create class rectangle and print area
14. Repl code
15. Cookies code
16. Functional component
17. Inheritance code
18. Es5 es6
19. Variable hoisting

20. what are events? sessions?

21. write switch case in js

22. Features of react

23. What is promise

24. What are iterators

ANSWERS

1. Arrow Function:

An arrow function in JavaScript is a concise way to write anonymous functions. It uses the `=>` syntax. For example:

```javascript
const add = (a, b) => a + b;
```

2. Features of React:

React is a JavaScript library for building user interfaces. Some of its key features include a virtual DOM, component-based architecture, JSX for templating, unidirectional data flow, and a rich ecosystem of libraries and tools.

3. Inheritance:

Inheritance is an Object-Oriented Programming (OOP) concept that allows a class to inherit properties and methods from another class. There are three types of inheritance: single, multiple, and multilevel.

4. Advantages of Express.js, React, and Node.js:

- Express.js: A minimal and flexible Node.js web application framework that simplifies building web applications and APIs.

- React: A fast and efficient UI library for building interactive user interfaces.

- Node.js: A runtime environment for server-side JavaScript that is non-blocking and ideal for building scalable network applications.

5. Iterator and Generator:

An iterator is an object that provides a way to access elements sequentially from a collection. A generator is a function that can pause its execution and later resume, making it useful for lazy and asynchronous programming.

6. Promise in React:

Promises are used in React for handling asynchronous operations, such as data fetching. Promises represent a value that may not be available yet but will be at some point.

7. Create a Textbox in React:

In React, you can create a textbox using JSX and controlled components. Here's a simple example:

```jsx
import React, { useState } from 'react';

function TextBox() {
    const [text, setText] = useState('');

    return (
    <input
  type="text"
  value={text}
  onChange={(e) => setText(e.target.value)}
    />
    );
}
```

8. Hooks in React:

Hooks are functions that let you "hook into" state and lifecycle features in functional components. They allow functional components to manage state and side effects.

9. Referencing:

"Referencing" typically means creating a reference to a variable or object, allowing you to access or modify that variable or object elsewhere in your code.

10. Initialize Buffer:

In Node.js, you can initialize a Buffer using the `Buffer.from()` method or by creating a new Buffer instance. For example:

```javascript
const buffer = Buffer.from('Hello, World!', 'utf8');
```

11. Sample Program for Promises:

Here's a simple JavaScript program using promises to simulate asynchronous operations:

```javascript
const fetchData = () => {
return new Promise((resolve, reject) => {
setTimeout(() => {
resolve('Data fetched successfully');
}, 2000);
});
}
```

```
fetchData()
.then((data) => {
  console.log(data);
})
.catch((error) => {
  console.error(error);
});
```

12. Difference between Sessions and Cookies:

Cookies are small pieces of data stored in the user's browser, while sessions are stored on the server. Sessions are more secure and can store larger amounts of data.

13. Create a Class Rectangle and Print Area:

Here's an example of a Rectangle class in JavaScript:

```javascript
class Rectangle {
  constructor(width, height) {
    this.width = width;
```

```
    this.height = height;
    }

    getArea() {
    return this.width * this.height;
    }
    }

    const rect = new Rectangle(5, 10);
console.log('Area:', rect.getArea());
    ```

14. REPL Code:

REPL stands for Read-Eval-Print Loop and is commonly used in JavaScript for interactive coding. You can use Node.js's built-in REPL or use an online REPL like the one provided by various code editors and websites.

15. Cookies Code:

To work with cookies in JavaScript, you can use the `document.cookie` property to set, get, and delete cookies. Here's a simple example:

```javascript
```

```
document.cookie = "username=John";

    const username = document.cookie;

    ```
```

16. Functional Component:

A functional component in React is a JavaScript function that returns JSX to define a part of the user interface. Functional components are stateless and can receive props as arguments.

17. Inheritance Code:

Here's a simple JavaScript code example demonstrating inheritance:

```javascript

    class Animal {

  constructor(name) {

    this.name = name;

    }


    speak() {

  console.log(`${this.name} makes a sound.`);

    }

    }
```
```

```
class Dog extends Animal {
speak() {
console.log(`${this.name} barks.`);
}
}

const dog = new Dog('Buddy');
dog.speak();
```

18. ES5 vs. ES6:

ES5 and ES6 refer to different versions of ECMAScript, the standard for JavaScript. ES6 introduced new features like arrow functions, classes, and let/const for variable declaration.

19. Variable Hoisting:

Variable hoisting in JavaScript means that variable declarations are moved to the top of their containing scope during compilation. However, only the declarations are hoisted, not the initializations.

20. Features of React (Repeated):

Please refer to the previous response for the features of React.

21. Events and Sessions:

Events in web development refer to user actions (e.g., clicks, keypresses) that trigger specific actions in a web application. Sessions are a way to maintain state information between HTTP requests in a web application.

22. Writing a Switch Case in JavaScript:

Here's an example of a switch case statement in JavaScript:

```javascript
const fruit = 'apple';

switch (fruit) {
case 'apple':
console.log('It's an apple.');
break;
case 'banana':
console.log('It's a banana.');
break;
default:
console.log('It's something else.');
```

```
}
```

23. What Is a Promise:

A promise is an object representing the eventual completion or failure of an asynchronous operation. It provides a more structured way to handle asynchronous operations and their results.

24. What Are Iterators:

Iterators are objects in JavaScript used to iterate over data structures like arrays, strings, and other iterable objects. They provide a way to access elements one at a time, sequentially.

VIVA QUESTIONS:

1. Arrow Function:
```javascript
const add = (a, b) => a + b;
```

2. Features of React:
- Declarative UI.
- Component-based architecture.
- Virtual DOM.
- Reusable components.
- Unidirectional data flow.
- JSX (JavaScript XML) for rendering components.

3. Inheritance (Types):

- Single Inheritance
- Multiple Inheritance (in some languages like C++, Python)
- Multilevel Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

4. Advantages of MERN stack:
- Unified JavaScript environment.
- Fast and efficient development.
- Strong community support.
- Component-based architecture in React.
- Scalability and performance in Node.js.
- Rich ecosystem of libraries and packages.

5. Iterator and Generator (example):
- Iterator: Iterators are objects that implement the Iterable protocol, typically with a `next()` method.
- Generator: Functions that can be paused and resumed, allowing you to iterate over sequences lazily.

6. Promise in React (example):
```javascript
const fetchData = () => {
  return new Promise((resolve, reject) => {
    // Simulate an async operation
    setTimeout(() => {
      if (/* operation was successful */) {
        resolve(data);
      } else {
        reject(error);
      }
    }, 1000);
  });
};
```

7. Create a Textbox in React:
```javascript
import React from 'react';
```

```
function MyComponent() {
  return (
    <input type="text" />
  );
}
```

8. Hooks (Definition):
   Hooks in React are functions that allow functional components to access state and other React features without writing class components. Some commonly used hooks include `useState`, `useEffect`, and `useContext`.

9. Referencing (in JavaScript):
   Referencing typically refers to creating a reference or pointer to an object or value in memory. It can be done using variables or data structures like arrays and objects.

10. Initialize Buffer (in Node.js):
    ```javascript
    const buffer = Buffer.from('Hello, world!', 'utf-8');
    ```

11. Sample Program for Promises:
    Here's a simple promise-based program:

    ```javascript
    function fetchData() {
      return new Promise((resolve, reject) => {
        setTimeout(() => {
          resolve('Data received!');
        }, 1000);
      });
    }

    fetchData()
      .then((data) => {
        console.log(data);
      })
      .catch((error) => {
        console.error(error);
      });
```

```

12. Difference in Session and Cookies:
   - Cookies: Small text files stored in the client's browser. They are sent with each HTTP request and response.
   - Sessions: Server-side storage for maintaining user-specific data. Session data is typically stored on the server and associated with a unique session ID in a cookie.

13. Create a Class Rectangle and Print Area (in JavaScript):
   ```javascript
   class Rectangle {
     constructor(width, height) {
       this.width = width;
       this.height = height;
     }

     getArea() {
       return this.width * this.height;
     }
   }

   const rect = new Rectangle(5, 10);
   console.log('Area:', rect.getArea());
   ```

14. REPL (Read-Eval-Print Loop) Code:
   - Example using Node.js: Run `node` in your terminal to access a JavaScript REPL.

15. Cookies Code:
   - You can set and read cookies in JavaScript using the `document.cookie` property.

16. Functional Component (in React):
   Functional components are simple JavaScript functions that accept props as input and return JSX (user interface) as output. Example:
   ```javascript
   function MyComponent(props) {
     return <div>{props.text}</div>;
   }
   ```
   ```

17. Inheritance Code:
   - Example of class inheritance:
   ```javascript
   class Parent {
     // Parent class properties and methods
   }

   class Child extends Parent {
     // Child class properties and methods
   }
   ```

18. ES5 and ES6:
   ES5 and ES6 refer to different versions of the ECMAScript standard. ES6 introduced new features like arrow functions, classes, and the `let` and `const` keywords, making JavaScript more powerful and developer-friendly.

19. Variable Hoisting (in JavaScript):
   Variable declarations are hoisted to the top of their containing function or block, making them accessible throughout the function or block even before they are declared.

20. Events and Sessions (clarification):
   - Events refer to user interactions or occurrences in a web application that can trigger JavaScript functions.
   - Sessions typically relate to server-side storage for maintaining user-specific data.

21. Switch-Case in JavaScript:
   ```javascript
   switch (variable) {
     case 'value1':
       // Code to execute when variable is 'value1'
       break;
     case 'value2':
       // Code to execute when variable is 'value2'
       break;
     default:
       // Code to execute when variable doesn't match any case
   }
   ```

22. More React Features (in MERN context):
- React Router for routing.
- State management with Redux or Mobx.
- Server-side rendering (SSR) with libraries like Next.js.
- Component libraries like Material-UI and Ant Design for UI development.

23. Promise (Definition):
A Promise is an object representing the eventual completion (or failure) of an asynchronous operation. It provides a cleaner way to handle asynchronous code than traditional callback patterns.

24. Iterators (in JavaScript):
Iterators are objects that implement the `Symbol.iterator` method, allowing you to loop over or iterate through data structures like arrays, sets, and maps.

Something Extra

1. Connecting to a MongoDB Database (Node.js with Mongoose):
To connect to a MongoDB database using Mongoose in Node.js:

```javascript
const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost/mydatabase', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

const db = mongoose.connection;

db.on('error', console.error.bind(console, 'MongoDB connection error:'));
db.once('open', () => {
  console.log('Connected to MongoDB');
});
```

2. Creating and Using RESTful APIs with Express.js:

An example of creating a RESTful API endpoint in Express.js:

```javascript
const express = require('express');
const app = express();
const port = 3000;

app.get('/api/items', (req, res) => {
  // Retrieve and send data as a response
});

app.post('/api/items', (req, res) => {
  // Create and save new data
});

app.put('/api/items/:id', (req, res) => {
  // Update data by ID
});

app.delete('/api/items/:id', (req, res) => {
  // Delete data by ID
});

app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

3. React Router (Routing in React):
Setting up routing in a React application using `react-router-dom`:

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

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

```
      <Route path="/contact" component={Contact} />
    </Switch>
  </Router>
);
```

4. Using Redux for State Management (React):
   Configuring Redux in a React application to manage application state:

   ```javascript
   import { createStore, combineReducers } from 'redux';
   import { Provider } from 'react-redux';

   // Define reducers
   const rootReducer = combineReducers({
     // Your individual reducers here
   });

   // Create the Redux store
   const store = createStore(rootReducer);

   // Wrap the app with the Redux Provider
   const App = () => (
     <Provider store={store}>
       <YourAppComponents />
     </Provider>
   );
   ```

5. Creating a Basic Express.js Server:
   Starting a basic Express.js server:

   ```javascript
   const express = require('express');
   const app = express();
   const port = 3000;

   app.get('/', (req, res) => {
     res.send('Hello, Express!');
   });
```

```
app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

6. Fetching Data in React (with `fetch` or Axios):
   Example of fetching data from an API in a React component:

```javascript
import React, { useEffect, useState } from 'react';

const MyComponent = () => {
  const [data, setData] = useState([]);

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

  return (
    <div>
      {data.map((item) => (
        <div key={item.id}>{item.name}</div>
      ))}
    </div>
  );
};
```

These are some additional syntax and concepts that may be relevant to MERN stack development and could be asked in a viva exam. Remember to study these topics thoroughly and be prepared to explain and discuss them in detail during your exam.