**Unit-1: ES6**

**1. What is the function of the let keyword in ES6, and how does it improve variable declaration over var?**

* **Answer**: The let keyword allows you to declare variables that are limited in scope to the block, statement, or expression where they are used. Unlike var, it avoids issues related to hoisting and makes variables accessible only within the block in which they are defined.

js

Here’s the code

if (true) {

let x = 10;

console.log(x); // Output: 10

}

console.log(x); // ReferenceError: x is not defined

* **Description**: let improves code readability and helps prevent issues caused by unintended global or function-scoped variables.

**2. Define an arrow function in ES6. In what situations would you prefer an arrow function over a traditional function?**

* **Answer**: Arrow functions are a shorthand syntax for writing functions and they don’t bind their own this context, which makes them ideal for use in callbacks and methods where you don't want to redefine the this context.

js

Here’s the code

// Traditional Function

function sum(a, b) {

return a + b;

}

// Arrow Function

const sum = (a, b) => a + b;

* **Description**: Arrow functions are preferred when you want concise code and lexical scoping for this, which is particularly useful in React components or event handling.

**3. What are the primary differences in scope between var and let in JavaScript?**

* **Answer**: var is function-scoped, while let is block-scoped. This means that var variables are accessible outside the block they were defined in, whereas let variables are only accessible within the block where they are declared.

js

Here’s the code

if (true) {

var x = 5;

let y = 10;

}

console.log(x); // Output: 5

console.log(y); // ReferenceError: y is not defined

* **Description**: let provides better control over variable scope, making it a safer and more predictable choice.

**4. How does the let keyword in ES6 support block-scoped variables, and why is this useful in modern JavaScript development?**

* **Answer**: Block-scoped variables declared with let are only accessible within the block they are defined in. This prevents variable leakage and ensures that variables are only used where they are relevant.

js

Here’s the code

for (let i = 0; i < 5; i++) {

console.log(i); // Output: 0 1 2 3 4

}

console.log(i); // ReferenceError: i is not defined

* **Description**: This feature helps in modern JS development by keeping the scope cleaner and avoiding accidental overwrites of variables outside of their intended usage.

**6. In what cases would you prefer to use the const keyword over let in ES6?**

* **Answer**: Use const when the variable's value should not change. For example, when declaring constants like configuration values or references to objects that shouldn't be reassigned.

js

Here’s the code

const API\_URL = "https://api.example.com";

API\_URL = "https://newapi.example.com"; // TypeError: Assignment to constant variable.

* **Description**: const ensures that variables cannot be reassigned, promoting safer, immutable code.

**7. Categorize variable declarations in ES6 (var, let, const) based on their scope and mutability.**

* **Answer**:
  + **var**: Function-scoped, mutable.
  + **let**: Block-scoped, mutable.
  + **const**: Block-scoped, immutable (cannot be reassigned).
* **Description**: Understanding the scope and mutability of these declarations helps write more predictable and stable code.

**10. Describe how arrow functions in ES6 differ from traditional functions, especially with regard to this keyword behavior.**

* **Answer**: Arrow functions do not have their own this context; instead, they inherit this from the surrounding lexical scope. This contrasts with traditional functions, which bind their own this.

js

Here’s the code

function TraditionalFunction() {

console.log(this); // refers to the calling object

}

const ArrowFunction = () => {

console.log(this); // refers to the surrounding context (lexical scope)

}

* **Description**: Arrow functions are often used in callbacks and methods in React components where the this context of the parent scope is desired.

**11. Demonstrate the use of the extends keyword to create inheritance between two classes in ES6, with an example.**

* **Answer**: The extends keyword allows a class to inherit methods and properties from another class.

js

Here’s the code

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(); // Output: Buddy barks.

* **Description**: This approach promotes reusable code and follows the principles of object-oriented programming.

**12. Use the spread operator in ES6 to merge two arrays and calculate the total sum of their elements.**

* **Answer**: The spread operator (...) can be used to merge arrays and then calculate their sum.

js

Here’s the code

const arr1 = [1, 2, 3];

const arr2 = [4, 5, 6];

const merged = [...arr1, ...arr2];

const sum = merged.reduce((acc, val) => acc + val, 0);

console.log(sum); // Output: 21

* **Description**: The spread operator simplifies array manipulation, making code more concise and readable.

**Unit-2: React**

**1. What is React, and what are its most significant features?**

* **Answer**: React is a JavaScript library for building user interfaces. Its key features include:
  + **Component-based architecture**: Breaks UI into reusable components.
  + **Virtual DOM**: Efficiently updates the UI by minimizing direct manipulation of the real DOM.
  + **JSX**: Syntax that allows embedding HTML within JavaScript.

**2. List the benefits of using JSX in React over traditional JavaScript.**

* **Answer**: JSX allows developers to write HTML-like syntax within JavaScript, improving readability and making it easier to visualize UI structure. It also provides powerful features like embedding JavaScript expressions and attributes.

jsx

Here’s the code

const element = <h1>Hello, {user.name}!</h1>;

* **Description**: JSX simplifies UI development by combining HTML and JavaScript logic in a single, readable syntax.

**3. Explain the differences between Props and State in React. Provide examples where each would be used.**

* **Answer**:
  + **Props**: Used to pass data from parent to child components.
  + **State**: Represents dynamic data that changes over time within a component.

jsx

Here’s the code

// Props Example

const Greeting = ({ name }) => <h1>Hello, {name}!</h1>;

// State Example

class Counter extends React.Component {

state = { count: 0 };

increment = () => this.setState({ count: this.state.count + 1 });

render() {

return (

<div>

<p>Count: {this.state.count}</p>

<button onClick={this.increment}>Increment</button>

</div>

);

}

}

* **Description**: Props are immutable and used for passing data, while State allows components to maintain and modify their internal data.

**4. Summarize the advantages of choosing React over other front-end frameworks, such as Angular.**

* **Answer**:
  + React offers better flexibility due to its component-based architecture.
  + Virtual DOM improves performance by minimizing expensive DOM manipulations.
  + React focuses only on the view layer, allowing developers to use it with other technologies.

**5. Explain how React's virtual DOM improves performance compared to traditional DOM manipulation.**

* **Answer**: React's virtual DOM creates a lightweight copy of the actual DOM and updates only the parts that have changed. This results in fewer direct DOM manipulations, which can be slow and resource-intensive.

**6. Demonstrate how to install and set up a React environment for development.**

* **Answer**: You can install and set up a React project using Create React App, a command-line tool for quickly creating React applications.

bash

Here’s the code

npx create-react-app my-app

cd my-app

npm start

* **Description**: This will create a new React project and start a local development server, allowing for immediate feedback during development.

**7. Use JSX to create a simple React component with nested elements.**

* **Answer**:

jsx

Here’s the code

const Welcome = () => (

<div>

<h1>Welcome to React</h1>

<p>This is a nested element inside a React component.</p>

</div>

);

**8. Explain how embedding JavaScript expressions in JSX helps improve dynamic rendering in React.**

* **Answer**: JSX allows you to embed JavaScript expressions within curly braces {} to dynamically render content.

jsx

Here’s the code

const user = { name: "John" };

const element = <h1>Hello, {user.name}!</h1>;

**9. Discuss how Props validation enhances the stability and reliability of React applications.**

* **Answer**: Prop validation using PropTypes ensures that components receive the correct type and format of props, leading to fewer runtime errors.

js

Here’s the code

import PropTypes from 'prop-types';

const Greeting = ({ name }) => <h1>Hello, {name}!</h1>;

Greeting.propTypes = {

name: PropTypes.string.isRequired

};

* **Description**: Props validation helps catch potential bugs early by ensuring that the data passed to components is valid.

**10. Analyze how React’s state management works and how state changes affect component behavior.**

* **Answer**: State represents dynamic data within a component. When state changes, React re-renders the component to reflect the updated state.

js

Here’s the code

class Counter extends React.Component {

state = { count: 0 };

increment = () => this.setState({ count: this.state.count + 1 });

render() {

return (

<div>

<p>Count: {this.state.count}</p>

<button onClick={this.increment}>Increment</button>

</div>

);

}

}

**SOME IMPORTANT QUESTIONS**

**Unit-1 (ES6)**

1. **What is the function of the let keyword in ES6, and how does it differ from previous versions of JavaScript?**
2. **Define an arrow function in ES6 and explain its use cases.**
3. **Contrast the scope behavior of var and let in JavaScript.**
4. **Why is the let keyword in ES6 considered an improvement over var? Explain with examples.**
5. **What is block scope in JavaScript, and how does let support this concept?**
6. **In what situations would the const keyword be a better choice than let?**
7. **Categorize variable declarations (var, let, and const) based on their scope and mutability.**
8. **Highlight the major differences between let, const, and var in ES6, especially in terms of scope.**
9. **Compare how let and const in ES6 improve code clarity compared to var in older JavaScript versions.**
10. **Discuss how the const keyword in ES6 impacts the immutability of variables.**
11. **Classify the types of variable declarations in ES6 and describe their scope.**
12. **Summarize how arrow functions in ES6 are distinct from traditional functions, particularly in how they handle the this context.**
13. **Demonstrate the use of the extends keyword to implement class inheritance in ES6, with an example.**
14. **Write a JavaScript program using the spread operator to combine two arrays and compute their total.**
15. **Show how let and const behave differently in block-scoped code through examples.**
16. **Create a simple ES6 class that uses inheritance and includes the super keyword.**
17. **Use the extends keyword to illustrate inheritance in ES6, with a relevant example.**
18. **Develop a basic ES6 class structure with a constructor, method, and inheritance using extends.**
19. **How do traditional functions differ from arrow functions in terms of handling the this keyword? Provide examples.**
20. **How do iterators and generators enhance iteration mechanisms in ES6? Provide illustrative examples.**
21. **Explain the differences between default parameter values and the spread operator in ES6 functions. How do they contribute to better function design?**
22. **Discuss the impact of using let, const, and var on memory management in JavaScript, with examples.**
23. **Analyze the key distinctions between let, const, and var concerning hoisting, block scope, and reassignment.**
24. **Explore the benefits of using the spread operator for handling function arguments and arrays in ES6. Provide examples.**

**Unit-2 (React)**

1. **What is React, and what are its most important features?**
2. **Outline the key advantages of using JSX in React.**
3. **Explain the main differences between Props and State in React.**
4. **Compare Angular and React, and explain why React might be the preferred choice in some cases.**
5. **Summarize the advantages of React over other front-end frameworks.**
6. **Identify the features that make React a popular choice for developers.**
7. **How does JSX differ from traditional JavaScript in embedding expressions?**
8. **Contrast the component-based architecture of Angular with that of React.**
9. **Why is React considered a better alternative to traditional web development frameworks?**
10. **Convert a given HTML structure into JSX format.**
11. **How does React’s virtual DOM differ from traditional DOM manipulation?**
12. **Compare how React and vanilla JavaScript handle event management.**
13. **Outline the steps required to install and set up a React environment.**
14. **Build a basic React component that passes data via Props.**
15. **Use JSX to construct a React component with nested elements.**
16. **Explain the role of JSX in React development and its importance.**
17. **Design a component that validates Props in React.**
18. **Demonstrate how to embed JavaScript expressions in JSX.**
19. **Discuss state management in React components and explain how changes to state impact behavior.**
20. **Expand a basic React application to include dynamic state updates.**
21. **Compare the performance of JSX to traditional HTML and JavaScript in web applications.**
22. **Describe the significance of Props in React, and how validating Props improves application robustness.**
23. **Extend the functionality of a React component using both State and Props.**
24. **Compare JSX with traditional HTML, outlining the key differences in structure and use.**