React Js Notes

1. What is React?

React is a JavaScript library for building user interfaces (UIs), primarily for single-page applications (SPAs). It was developed by Facebook and allows developers to create reusable components for better performance and maintainability.

- **React Features:**
  - **Declarative: React allows you to describe your UI in a clear, declarative manner.**
  - **Component-Based: The UI is broken into components, making the code modular and easier to maintain.**
  - **Virtual DOM: React uses a virtual representation of the actual DOM to optimize UI rendering.**

---

2. Key Concepts in React

Components

- **Components are the building blocks of React applications. Components can be either functional or class-based.**

Functional Components

- **Simpler components that are written as JavaScript functions.**
- **useState, useEffect, and other hooks are used for state and lifecycle management.**

Example:

```
function Greeting() {
  return <h1>Hello, World!</h1>;
}
```

Class Components

- **More feature-rich components that extend from React.Component.**
- **They are used when you need features like lifecycle methods.**

Example:

javascript

```
class Greeting extends React.Component {
  render() {
    return <h1>Hello, World!</h1>;
  }
}
```

3. JSX (JavaScript XML)

- **JSX allows you to write HTML-like syntax inside JavaScript.**
- **It gets compiled into JavaScript by React's build tools.**
- **JSX makes it easier to describe the UI structure.**

Example:

```
const element = <h1>Hello, React!</h1>;
```

---

4. Props (Properties)

- **Props allow data to be passed from a parent component to a child component.**
- **Props are read-only.**

Example:

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

function App() {
  return <Greeting name="Alice" />;
}
```

---

5. State

- **State is used to manage data that changes over time within a component.**
- **React's useState hook is used to add state to functional components.**

Example:

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

function Counter() {
  const [count, setCount] = useState(0); // Initializing state with 0
```

```
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
}
```

---

6. Handling Events

React handles events like button clicks, form submissions, etc., with camelCase syntax for event handlers.

Example:

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

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

---

7. Conditional Rendering

You can render different UI elements based on certain conditions using JavaScript logic within JSX.

Example:

javascript

Copy code

```
function Greeting({ isLoggedIn }) {
  return (
    <div>
      {isLoggedIn ? <h1>Welcome Back!</h1> : <h1>Please Log In</h1>}
    </div>
  );
}

function App() {
  return <Greeting isLoggedIn={true} />;
}
```

---

8. Lists and Keys

- **You can render lists in React using the .map() method.**
- **Keys help React identify which items have changed, are added, or are removed.**

Example:

```
function List() {
  const items = ['Apple', 'Banana', 'Cherry'];
  return (
    <ul>
      {items.map((item, index) => (
        <li key={index}>{item}</li>
      ))}
    </ul>
  );
}
```

---

9. useEffect Hook

- **useEffect() is a hook used for performing side effects, such as data fetching, DOM updates, or subscribing to events in a functional component.**
- **It acts as a replacement for lifecycle methods in class components.**

Example:

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


function Timer() {
  const [seconds, setSeconds] = useState(0);
```

```
  useEffect(() => {
    const interval = setInterval(() => {
      setSeconds(prev => prev + 1);
    }, 1000);

    return () => clearInterval(interval); // Cleanup on component unmount
  }, []);

  return <p>Timer: {seconds} seconds</p>;
}
```

---

10. Context API

- The Context API allows you to pass data through the component tree without manually passing props at every level.
- Useful for global state management (e.g., themes, authentication).

Example:

```
import React, { createContext, useContext } from 'react';

// Create a Context
const ThemeContext = createContext('light');

function ThemedComponent() {
```

```
  const theme = useContext(ThemeContext);
  return <div>The theme is {theme}</div>;
}

function App() {
  return (
    <ThemeContext.Provider value="dark">
      <ThemedComponent />
    </ThemeContext.Provider>
  );
}
```

---

11. React Router

React Router allows you to implement client-side routing in React apps. It enables navigation between different views or pages without reloading the entire page.

Example:

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

function Home() {
  return <h2>Home Page</h2>;
}
```

```
function About() {
  return <h2>About Page</h2>;
}

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

---

12. Forms in React

Forms in React are controlled components because the form input values are controlled by the component's state.

Example:

```
function MyForm() {
  const [inputValue, setInputValue] = useState('');

  const handleChange = (event) => {
    setInputValue(event.target.value);
  };

  const handleSubmit = (event) => {
    event.preventDefault();
    alert('Form submitted: ' + inputValue);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="text" value={inputValue}
onChange={handleChange} />
      <button type="submit">Submit</button>
    </form>
  );
}
```

---

13. React Lifecycle Methods (Class Components)

Lifecycle methods are special methods in class components that are called at different stages of a component's life.

Common lifecycle methods include:

- **componentDidMount() – Called when the component is mounted (after it has been rendered).**
- **componentDidUpdate() – Called when the component updates.**
- **componentWillUnmount() – Called before the component is unmounted.**

Example:

```
class MyComponent extends React.Component {
  componentDidMount() {
    console.log('Component has mounted');
  }

  render() {
    return <h1>Hello, World!</h1>;
  }
}
```

---

14. React Developer Tools

React Developer Tools is a browser extension for inspecting React component hierarchies in the Chrome Developer Tools.

- **You can inspect props, state, and component structure.**

- **It helps debug your application and understand how your components work.**

---

15. Advanced Concepts

React Hooks

- **React Hooks like useState, useEffect, useContext, useReducer, and others simplify handling state and side effects in functional components.**

Error Boundaries

- **Error boundaries are components that catch JavaScript errors in any part of their child component tree and log those errors.**

React.js Interview Questions and Answers

---

1. What is React?

Answer:
React is a popular JavaScript library used for building user interfaces, especially single-page applications (SPAs). It allows developers to create reusable UI components that can manage their own state. React uses a virtual DOM to efficiently update the real DOM when the state changes.

---

2. What is the Virtual DOM?

Answer:
The Virtual DOM is an in-memory representation of the real DOM elements. It is a lightweight copy of the actual DOM. React uses the

Virtual DOM to improve performance by updating only the parts of the DOM that have changed, rather than re-rendering the entire page.

---

3. What are Components in React?

Answer:
Components are the building blocks of a React application. A component is a JavaScript function or class that optionally accepts inputs (called props) and returns a React element describing how the UI should appear. There are two types of components:

- **Functional components: Simple functions that return JSX.**
- **Class components: ES6 classes that extend React.Component and can have lifecycle methods.**

---

4. What is JSX?

Answer:
JSX (JavaScript XML) is a syntax extension for JavaScript that allows you to write HTML-like code within JavaScript. React uses JSX to define the structure of the UI. JSX is then transpiled into JavaScript code by tools like Babel.

Example of JSX:

javascript

Copy code

const element = <h1>Hello, React!</h1>;

---

5. What are Props in React?

Answer:
Props (short for "properties") are the inputs to a React component. They are passed from a parent component to a child component and allow the child component to render dynamic content. Props are read-only and cannot be modified by the child component.

Example:

javascript

Copy code

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

function App() {
  return <Welcome name="Alice" />;
}
```

---

6. What is State in React?

Answer:
State is a way for components to manage data that can change over time. Unlike props, state is mutable and is local to a component. State changes trigger a re-render of the component.

For functional components, you can use the useState hook to manage state.

Example:

javascript

Copy code

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


function Counter() {
  const [count, setCount] = useState(0);


  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>Click me</button>
    </div>
  );
}
```

---

7. What is the difference between state and props?

Answer:

- **Props are used to pass data from a parent component to a child component. They are immutable and cannot be changed by the child component.**
- **State is used to manage data that can change within a component. It is mutable and managed within the component.**

---

8. What are React Hooks?

Answer:
React Hooks are functions that allow you to use React features such as state and lifecycle methods in functional components. Commonly used hooks include:

- **useState(): Allows you to add state to a functional component.**
- **useEffect(): Allows you to perform side effects, like data fetching or DOM manipulation.**
- **useContext(): Allows you to access context values.**

---

9. What is the useEffect Hook?

Answer:
The useEffect hook allows you to perform side effects in functional components. Side effects can include data fetching, setting up a subscription, and manually changing the DOM. By default, useEffect runs after every render.

Example:

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

function Timer() {
  const [seconds, setSeconds] = useState(0);
  useEffect(() => {
    const interval = setInterval(() => {
      setSeconds(prev => prev + 1);
    }, 1000);
```

```
    return () => clearInterval(interval); // Cleanup on component unmount
  }, []); // Empty dependency array means it runs once on mount
  return <p>Time: {seconds} seconds</p>;
}
```

---

10. What is Conditional Rendering in React?

Answer:
Conditional rendering allows you to render different UI elements based on certain conditions. You can use JavaScript operators like if, ternary operator, or && to decide what to render.

Example:

```
function Greeting({ isLoggedIn }) {
  return (
    <div>
      {isLoggedIn ? <h1>Welcome!</h1> : <h1>Please Log In</h1>}
    </div>
  );
}
```

---

11. What is React Router?

Answer:
React Router is a library used for adding client-side routing to React

applications. It enables navigation between different views or pages without reloading the entire page.

Example:

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

function Home() {
  return <h2>Home Page</h2>;
}

function About() {
  return <h2>About Page</h2>;
}

function App() {
  return (
    <Router>
      <div>
        <nav>
          <Link to="/">Home</Link>
          <Link to="/about">About</Link>
        </nav>
        <Route path="/" exact component={Home} />
        <Route path="/about" component={About} />
      </div>
    </Router>
```

```
  );
}
```

---

12. What is Context API in React?

Answer:
The Context API is a way to share state between components without having to pass props down manually at every level. It is useful for global state management, like themes or user authentication.

Example:

```
import React, { createContext, useContext } from 'react';

const ThemeContext = createContext('light');

function ThemedComponent() {
  const theme = useContext(ThemeContext);
  return <div>The theme is {theme}</div>;
}


function App() {
  return (
    <ThemeContext.Provider value="dark">
      <ThemedComponent />
    </ThemeContext.Provider>
  );
}
```

---

13. What is a Higher-Order Component (HOC)?

Answer:
A Higher-Order Component (HOC) is a function that takes a component and returns a new component with additional props or logic. HOCs are used for reusing component logic.

Example:

```
function withLoader(Component) {
   return function LoaderHOC(props) {
      if (props.isLoading) {
         return <div>Loading...</div>;
      }
      return <Component {...props} />;
   };
}
const MyComponentWithLoader = withLoader(MyComponent);
```

---

14. What is the purpose of the key prop in React?

Answer:
The key prop helps React identify which items have changed, are added, or are removed. It is used in lists to ensure efficient updates of the component tree when the state changes.

Example:

```
const items = ['Apple', 'Banana', 'Cherry'];
```

```
function List() {
  return (
    <ul>
      {items.map((item, index) => (
        <li key={index}>{item}</li>
      ))}
    </ul>
  );
}
```

---

15. What is Redux?

Answer:
Redux is a state management library for JavaScript applications. It is often used with React to manage the global state of the application in a predictable way. Redux follows the unidirectional data flow pattern and helps manage complex state logic.

- **Actions: Payloads of information that send data from the application to the Redux store.**
- **Reducers: Functions that specify how the application's state changes in response to an action.**
- **Store: The object that holds the state of the application.**

---

16. What is the difference between a functional component and a class component?

Answer:

- **Functional Components:**
  - **Simpler to write and understand.**
  - **No internal state or lifecycle methods (unless using hooks).**
- **Class Components:**
  - **Have access to state and lifecycle methods.**
  - **Require more boilerplate code.**

Example of a class component with state and lifecycle:

```
class Counter extends React.Component {
  constructor() {
    super();
    this.state = { count: 0 };
  }
  componentDidMount() {
    console.log('Component mounted');
  }
  render() {
    return (
      <div>
        <p>{this.state.count}</p>
      </div>
    );
```

```
    }
}
```

JavaScript Full Notes

---

1. Introduction to JavaScript

- **JavaScript (JS) is a high-level, interpreted programming language that is primarily used for building interactive websites.**
- **It was developed by Brendan Eich in 1995 and is now one of the core technologies of the web (along with HTML and CSS).**
- **JavaScript can be used for front-end (client-side) and back-end (server-side) development (Node.js).**

---

2. Variables and Data Types

Variables:

In JavaScript, variables are used to store data values. There are three ways to declare variables:

- **var – The oldest way to declare a variable (not commonly used now due to scoping issues).**
- **let – Declares a block-scoped local variable, which can be updated.**
- **const – Declares a block-scoped constant, which cannot be updated after being assigned.**

Example:

let name = "John"; // Can be updated

const age = 30; // Cannot be updated

Data Types:

JavaScript has several primitive data types:

- **String: Represents a sequence of characters. Example: 'Hello'**
- **Number: Represents numerical values (integer or floating-point). Example: 42**
- **Boolean: Represents true or false. Example: true**
- **Undefined: A variable declared but not assigned a value is undefined. Example: let x;**
- **Null: Represents an empty or non-existent value. Example: let y = null;**
- **Symbol: A unique identifier.**
- **BigInt: Represents large integers.**

---

3. Operators in JavaScript

JavaScript uses a wide range of operators for performing arithmetic, comparison, logical, and more operations.

Arithmetic Operators:

let sum = 5 + 2; // Addition

let diff = 5 - 2; // Subtraction

let prod = 5 * 2; // Multiplication

```
let quotient = 5 / 2; // Division
let remainder = 5 % 2; // Modulus (Remainder)
let exponent = 5 ** 2; // Exponentiation

Comparison Operators:
let isEqual = (5 == 5); // Equal to
let isStrictEqual = (5 === '5'); // Strict Equal to (checks type and value)
let isNotEqual = (5 != 3); // Not Equal to
let isGreaterThan = (5 > 3); // Greater than
let isLessThan = (5 < 3); // Less than

Logical Operators:
let and = (true && false); // AND operator
let or = (true || false); // OR operator
let not = !true; // NOT operator
```

---

4. Control Flow and Loops

Conditional Statements:

if-else:

```
let age = 20;
if (age >= 18) {
   console.log("Adult");
} else {
```

```
    console.log("Minor");
}
switch:
let fruit = 'apple';
switch (fruit) {
    case 'apple':
        console.log("Apple is red");
        break;
    case 'banana':
        console.log("Banana is yellow");
        break;
    default:
        console.log("Unknown fruit");
}
Loops:
for loop:
for (let i = 0; i < 5; i++) {
    console.log(i);
}
while loop:
let i = 0;
while (i < 5) {
    console.log(i);
```

```
    i++;
}
```

do-while loop:

javascript

Copy code

```
let i = 0;
do {
    console.log(i);
    i++;
} while (i < 5);
```

---

5. Functions in JavaScript

Function Declaration:

```
function greet(name) {
    console.log("Hello, " + name);
}

greet("Alice");
```

Function Expression:

```
const greet = function(name) {
    console.log("Hello, " + name);
};
```

```
greet("Bob");
```

Arrow Functions:

Arrow functions are a shorthand syntax for writing functions.

javascript

Copy code

```
const greet = (name) => {
   console.log("Hello, " + name);
};

greet("Charlie");
```

Return Statement:

Functions can return values.

```
function add(a, b) {
   return a + b;
}

let result = add(5, 3);
console.log(result); // 8
```

---

6. Arrays in JavaScript

An array is an ordered list of values, which can be of any data type.

Creating Arrays:

```
let fruits = ["apple", "banana", "cherry"];
```

Array Methods:

- **push(): Adds an element to the end.**
- **pop(): Removes the last element.**
- **shift(): Removes the first element.**
- **unshift(): Adds an element to the beginning.**
- **map(): Creates a new array with the results of calling a function on every element.**
- **filter(): Creates a new array with all elements that pass a test.**
- **reduce(): Applies a function against an accumulator and every element in the array.**

Example:

```
let numbers = [1, 2, 3, 4];

let doubled = numbers.map(num => num * 2);

console.log(doubled); // [2, 4, 6, 8]
```

---

7. Objects in JavaScript

Objects are collections of properties and methods. A property is a key-value pair, and methods are functions stored as object properties.

Creating Objects:

```
let person = {
    name: "John",
```

```
    age: 30,
    greet: function() {
        console.log("Hello, " + this.name);
    }
};

console.log(person.name); // "John"
person.greet(); // "Hello, John"
```

Object Methods:

Objects can have methods:

```
const car = {
    make: "Toyota",
    model: "Corolla",
    drive: function() {
        console.log("Driving " + this.make + " " + this.model);
    }
};
car.drive(); // "Driving Toyota Corolla"
```

---

8. DOM Manipulation

The Document Object Model (DOM) represents the structure of an HTML document. JavaScript can interact with the DOM to dynamically change the content, style, and structure of a webpage.

Selecting Elements:

```
let element = document.getElementById("myElement");
let allDivs = document.querySelectorAll("div");
```

Modifying Elements:

```
element.innerHTML = "New content"; // Changes text
element.style.color = "blue"; // Changes style
```

Event Handling:

```
document.getElementById("myButton").addEventListener("click", function() {
    alert("Button clicked!");
});
```

---

9. Asynchronous JavaScript

setTimeout():

Executes a function after a specified delay.

```
setTimeout(() => {
```

```
    console.log("Executed after 2 seconds");
}, 2000);
```

Promises:

Promises are used to handle asynchronous operations.

```
let myPromise = new Promise((resolve, reject) => {
    let success = true;
    if (success) {
        resolve("Success!");
    } else {
        reject("Error!");
    }
});

myPromise
    .then(result => console.log(result))
    .catch(error => console.log(error));
```

async/await:

A modern way to handle asynchronous code.

```
async function fetchData() {
    let response = await fetch('https://api.example.com');
    let data = await response.json();
```

```
    console.log(data);
}
```

---

10. Error Handling

JavaScript provides mechanisms for handling errors using try...catch.

```
try {
    let result = riskyFunction();
} catch (error) {
    console.log("Error caught: " + error.message);
}
```

---

11. ES6 Features

ES6 (ECMAScript 2015) introduced several new features to JavaScript:

- **Let & Const: Block-scoped variable declarations.**
- **Arrow Functions: Shorter syntax for functions.**
- **Template Literals: Multi-line strings and string interpolation.**
- **Destructuring Assignment: Unpacking values from arrays or objects.**

```
let person = { name: "Alice", age: 25 };
let { name, age } = person;
```

- **Spread Operator: Copies or combines arrays/objects.**

```
let arr1 = [1, 2, 3];
let arr2 = [...arr1, 4, 5];
```

---

12. JavaScript Classes

Classes provide a way to create reusable object templates in JavaScript.

```
class Person {
   constructor(name, age) {
      this.name = name;
      this.age = age;
   }
   greet() {
      console.log("Hello, " + this.name);
   }
}
let john = new Person("John", 30);
john.greet(); // "Hello, John"
```

JavaScript Interview Questions & Answers

---

1. What are the different data types in JavaScript?

Answer:
JavaScript has 7 primitive data types:

1. **String: Represents a sequence of characters.**

2. **Number: Represents both integer and floating-point numbers.**
3. **Boolean: Represents true or false values.**
4. **Undefined: A variable that has been declared but not assigned a value.**
5. **Null: Represents an intentional absence of any object value.**
6. **Symbol: Represents a unique identifier.**
7. **BigInt: Represents integers larger than the safe integer limit of the Number type.**

Additionally, there is the Object data type, which can hold collections of data in the form of key-value pairs.

---

2. What is the difference between == and === in JavaScript?

Answer:

- **== (Equality operator) checks for value equality, but performs type coercion (converts operands to the same type before comparison).**
- **=== (Strict equality operator) checks for value and type equality and does not perform type coercion.**

Example:

```
5 == '5';   // true (because of type coercion)

5 === '5';  // false (because types are different)
```

---

3. Explain JavaScript closures.

Answer:
A closure is a function that retains access to variables from its outer

(enclosing) function even after the outer function has finished executing. This enables the inner function to access and manipulate the outer function's variables.

Example:

```
function outer() {
   let count = 0;
   return function inner() {
      count++;
      console.log(count);
   };
}

const counter = outer();
counter(); // 1
counter(); // 2
```

In this example, inner is a closure that has access to the count variable from outer, even after outer has finished executing.

---

4. What is this in JavaScript?

Answer:
The keyword this refers to the context in which the function is called. Its value can vary depending on how the function is invoked:

- **In a regular function, this refers to the global object (in non-strict mode) or undefined (in strict mode).**

- **In an object method, this refers to the object the method is called on.**
- **In an arrow function, this is lexically bound, meaning it inherits the this value from its surrounding context (not dynamically based on how it's called).**

Example:

```
const person = {
  name: "John",
  greet: function() {
    console.log(this.name);
  }
};

person.greet();  // "John"
```

---

5. What is a callback function?

Answer:
A callback function is a function that is passed as an argument to another function and is executed once the operation is complete. Callbacks are typically used for asynchronous operations, such as reading files or making API requests.

Example:

```
function fetchData(callback) {
  setTimeout(() => {
```

```
    callback("Data fetched!");
  }, 1000);
}

fetchData(function(message) {
  console.log(message);  // "Data fetched!"
});
```

---

6. What is event delegation in JavaScript?

Answer:
Event delegation is a technique where instead of attaching event listeners to individual child elements, you attach a single event listener to a parent element and let the event "bubble" up. The parent element can then decide what action to take based on the event target.

Example:

```
document.querySelector("#parent").addEventListener("click", function(event) {
  if (event.target && event.target.matches("button")) {
    console.log("Button clicked");
  }
});
```

This is more efficient when you have a large number of child elements, as it avoids adding event listeners to each individual child.

---

7. What is bind(), call(), and apply() in JavaScript?

Answer:

- **bind(): Creates a new function that, when invoked, has its this value set to the provided value.**
- **call(): Immediately calls the function with a specified this value and arguments.**
- **apply(): Similar to call(), but arguments are passed as an array.**

Example:

```
function greet() {
  console.log(`Hello, ${this.name}`);
}


const person = { name: "Alice" };
const greetPerson = greet.bind(person);  // `bind` creates a new function
greetPerson();  // Hello, Alice


greet.call(person);  // Hello, Alice
greet.apply(person);  // Hello, Alice
```

---

8. What are promises in JavaScript?

Answer:
A promise is an object representing the eventual completion or failure of an asynchronous operation. A promise can be in one of three states:

- **Pending: The operation is still in progress.**
- **Resolved (Fulfilled): The operation completed successfully.**
- **Rejected: The operation failed.**

You can use .then() to handle a resolved promise and .catch() to handle a rejected promise.

Example:

```
let promise = new Promise((resolve, reject) => {
   let success = true;
   if (success) {
      resolve("Data fetched!");
   } else {
      reject("Error occurred");
   }
});

promise.then((message) => {
   console.log(message); // "Data fetched!"
}).catch((error) => {
   console.log(error); // "Error occurred"
});
```

---

9. Explain the concept of hoisting in JavaScript.

Answer:
Hoisting is JavaScript's default behavior of moving declarations to the top of their containing scope during the compile phase. Only the declarations (not the initializations) are hoisted.

Example:

```
console.log(a);  // undefined

var a = 5;

console.log(a);  // 5
```

In the above example, var a is hoisted to the top, but its assignment a = 5 remains where it is.

---

10. What is a closure?

Answer:
A closure is a function that has access to its own scope, the scope in which it was created, and the global scope. Closures are created every time a function is created, and they can be used to preserve data between function calls.

Example:

```
function outer() {
   let counter = 0;
   return function inner() {
      counter++;
      console.log(counter);
```

```
    };
}


const counterFunction = outer();
counterFunction();  // 1
counterFunction();  // 2
```

Here, the inner function maintains access to the counter variable from the outer function.

---

11. What are JavaScript's higher-order functions?

Answer:
A higher-order function is a function that takes another function as an argument or returns a function as a result. Common examples include map(), filter(), and reduce().

Example:

javascript

Copy code

```
let numbers = [1, 2, 3];
let doubled = numbers.map(num => num * 2);  // `map` is a higher-order function
console.log(doubled);  // [2, 4, 6]
```

---

12. What is the difference between null and undefined?

Answer:

- **null is an intentional assignment of no value. It is used when a variable is expected to hold an object or value but doesn't yet.**
- **undefined means a variable has been declared but has not been assigned a value.**

Example:

```
let a = null;  // null is intentionally assigned

let b;  // undefined is default for uninitialized variables
```

---

13. What is the event loop in JavaScript?

Answer:
The event loop is a mechanism that allows JavaScript to execute code asynchronously. It continuously checks the call stack to see if there is any code to execute, and if the stack is empty, it processes the callback queue (events like setTimeout, setInterval, etc.).

This is what allows JavaScript to handle asynchronous operations like API calls, user input, and timers without blocking the main thread.

---

14. Explain the concept of "this" in arrow functions.

Answer:
In arrow functions, the value of this is lexically bound (it does not change based on how the function is called). It takes the value of this from the surrounding context where the function is defined.

Example:

```
const person = {
   name: "Alice",
```

```
    greet: function() {
        setTimeout(() => {
            console.log(this.name);  // "Alice"
        }, 1000);
    }
};

person.greet();
```