1. How does React Work? How Virtual-DOM works in React?
Ans: **React creates a virtual DOM**.
Ans: When we setState() it calls the render() - React creates a virtual dom, sees the updated state, and updates DOM
When the state changes in a component it firstly runs a "diffing" algorithm, which identifies what has changed in the virtual DOM. The second step is reconciliation, where it updates the DOM (tree structure) with the results of diff.
The virtual DOM is an in-memory representation of the actual HTML elements that make up your application's UI
ReactElements lives in the virtual DOM. They make the basic nodes here. Once we defined the elements, ReactElements can be render into the "real" DOM.
Whenever a ReactComponent is changing the state, diff algorithm in React runs and identifies what has changed. And then it updates the DOM with the results of diff. The point is - it's done faster than it would be in the regular DOM.
https://medium.com/@vigowebs/frequently-asked-react-js-interview-questions-and-answers-36f3dd99f486

2. What is JSX?
Ans: JSX is a syntax **extension to JavaScript** and comes with the full power of JavaScript. **JSX compiles into React "elements".** You can embed any JavaScript expression in JSX by wrapping it in curly braces. After compilation, JSX expressions become regular JavaScript objects.
https://medium.com/@vigowebs/frequently-asked-react-js-interview-questions-and-answers-36f3dd99f486

3. What are the differences between Functional Programming vs Object Oriented Programming?
Funct Ans: **Functional programming** is the form of programming that attempts to **avoid changing state and mutable data.** In a functional program, the output of a function should always be the same, given the same exact inputs to the function. This is because the outputs of a function in functional programming purely relies on arguments of the function, and there is no magic that is happening behind the scenes. It thrives in situations where the state is not a factor and there is very little to no involvement with mutable data.
OO Ans: **Object oriented programming** is a programming paradigm in which you program using objects to represent things you are programming about (sometimes real world things). **These objects could be data structures. The objects hold data about them in attributes. The attributes in the objects are manipulated through methods or functions that are given to the object.**
https://medium.com/@shaistha24/functional-programming-vs-object-oriented-programming-oop-which-is-better-82172e53a526#:~:text=Both%20Functional%20programming%20and%20object,data%20is%20stored%20in%20objects.

4. What are the differences between class components and functional components?
Class Ans: **Class components** allows us to use additional features such as **local state** and lifecycle hooks. Also, to enable our component to have **direct access to our store and thus holds state.**
Functional Components: When our component just **receives props and renders them to the page**, this is a 'stateless component', for which a pure function can be used. These are also called dumb components or presentational components.
https://medium.com/@vigowebs/frequently-asked-react-js-interview-questions-and-answers-36f3dd99f486

5. Difference between state and props?
State Ans: The state is a data structure that starts with a default value when a Component mounts. It may be mutated across time, mostly as a result of user events.

Props Ans: Props (short for properties) are a Component's configuration. Props are how components talk to each other. **They are received from above component and immutable as far as the Component receiving them is concerned.** A Component cannot change its props, but it is responsible for putting together the props of its child Components. Props do not have to just be data — callback functions may be passed in as props.
https://medium.com/@vigowebs/frequently-asked-react-js-interview-questions-and-answers-36f3dd99f486

6. Arrow functions - Arrows support both statement block bodies as well as expression bodies which **return the value of the expression.** Unlike functions, arrows share the same lexical **this** as their surrounding code.
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions
https://github.com/lukehoban/es6features

7. What are controlled vs uncontrolled components?

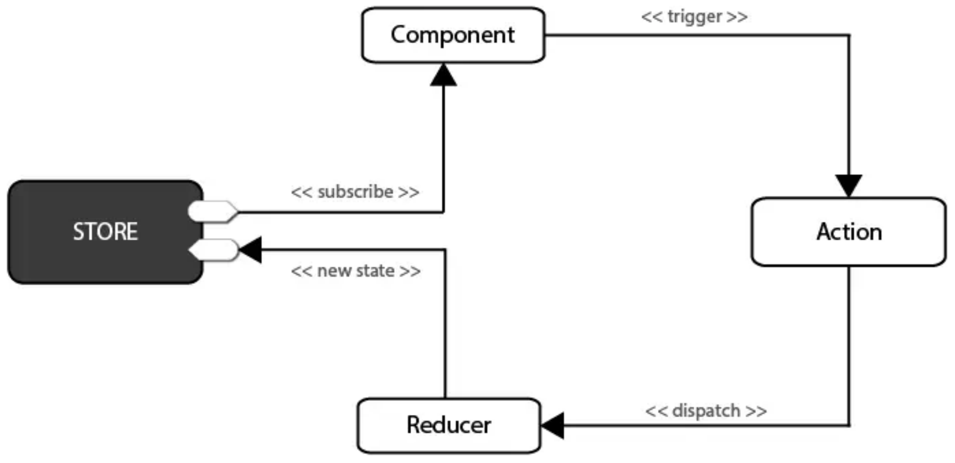
- A Controlled Component is one that takes its current value through props and notifies changes through callbacks like onChange. A parent component "controls" it by handling the callback and managing its own state and passing the new values as props to the controlled component. You could also call this a "dumb component".
- A Uncontrolled Component is one that stores its own state internally, and you query the DOM using a ref to find its current value when you need it. This is a bit more like traditional HTML.

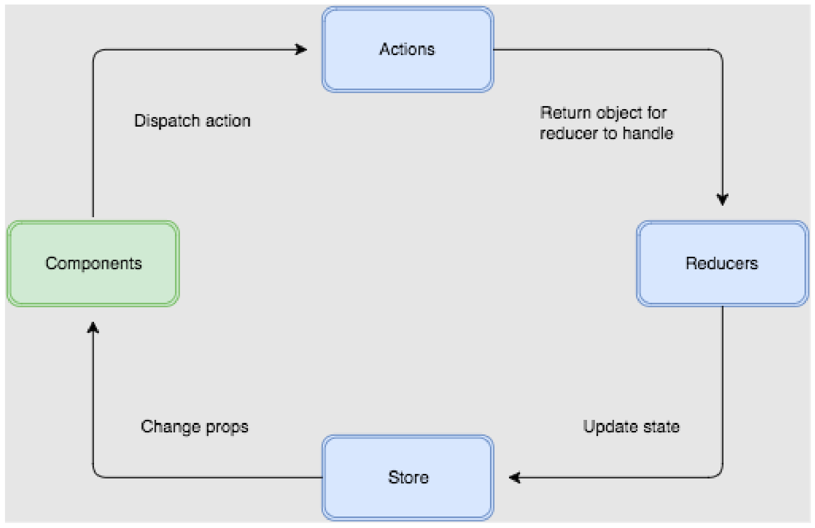
https://stackoverflow.com/questions/42522515/what-are-react-controlled-components-and-uncontrolled-components

8. Higher order components?
Ans: A higher-order component (HOC) is an advanced technique in **React for reusing component logic.** HOCs are not part of the React API. They are a pattern that emerges from React's compositional nature. **A higher-order component is a function that takes a component and returns a new component.**
https://medium.com/@vigowebs/frequently-asked-react-js-interview-questions-and-answers-36f3dd99f486

9. Redux?
Ans: the **entire application state is kept in a single store.** The **store is simply a javascript object.** The only way to change the state is by firing actions from your application and then writing reducers for these actions that modify the state. The entire state transition is kept inside reducers and should not have any side-effects.
https://medium.com/@vigowebs/frequently-asked-react-js-interview-questions-and-answers-36f3dd99f486

Post data = Component > mapDispatchToProps (dispatch action w props) > actionCreator w ActionTypes > reducer > store > component > mapStateToProps





10. What is Redux Thunk used for?
Redux thunk is **middleware** that allows us to **write action creators that return a function instead of an action**. The thunk can then be **used to delay the dispatch of an action if a certain condition is met.** This allows us to handle the asynchronous dispatching of actions. The inner function receives the store methods dispatch and getState as parameters.
https://medium.com/@vigowebs/frequently-asked-react-js-interview-questions-and-answers-36f3dd99f486

11. What is PureComponent? When to use PureComponent over Component?
Ans: PureComponent is exactly **the same as Component except that it handles the shouldComponentUpdate method for us**. When props or state changes, PureComponent **will do a shallow comparison on both props and state.** Component on the other hand won't compare current props and state to next out of the box. Thus, the component will re-render by default whenever shouldComponentUpdate is called.
https://medium.com/@vigowebs/frequently-asked-react-js-interview-questions-and-answers-36f3dd99f486

12. Is JavaScript Synchronous or Asynchronous?
Ans: **JavaScript is always synchronous and single-threaded.**
https://stackoverflow.com/questions/2035645/when-is-javascript-synchronous#:~:text=JavaScript%20is%20always%20synchronous%20and%20single%2Dthreaded.&text=JavaScript%20is%20only%20asynchronous%20in,the%20callback%20will%20run%20synchronously.

13.Async await was introduced in ES8

```
async componentDidMount() {
    const response = await fetch(`https://api.coinmarketcap.com/v1/ticker/?limit=10`);
    const json = await response.json();
    this.setState({ data: json });
  }
```

https://www.valentinog.com/blog/await-react/

ES6
=>
let & const
classes

ES7
Exponential - (`**`) returns the result of raising the first operand to the power of the second operand. It is equivalent to `Math.pow`, except it also accepts BigInts as operands.
Array.props.includes - The array.**`includes()`** method determines whether an array includes a certain value among its entries, returning `true` or `false` as appropriate.

ES8
Async Await - for Promises in JavaScript. The **await** keyword is used with promises. This keyword can be used to pause the execution of a function till a promise is settled. The **await** keyword returns value of the promise if the promise is resolved while it throws an error if the promise is rejected.

ES9
Async Iteration
Regex stuff

14. Is setState() is async? Why is setState() in React Async instead of Sync?
Ans: **setState() actions are asynchronous. setState() does not immediately mutate this.state but creates a pending state transition**. Accessing this.state after calling this method can potentially return the existing value. There is no guarantee of synchronous operation of calls to setState and calls may be batched for performance gains.
This is because setState alters the state and causes rerendering. This can be an expensive operation and making it synchronous might leave the browser unresponsive. Thus the setState calls are asynchronous as well as batched for better UI experience and performance.
https://medium.com/@vigowebs/frequently-asked-react-js-interview-questions-and-answers-36f3dd99f486

15. What is render() in React? Explain it's purpose
Ans: Each React (class) component must have a render() mandatorily. **It returns a single React element which is the representation of the native DOM component.** If more than one HTML element needs to be rendered, then they must be grouped together inside one enclosing tag such as <form>, <group>, <div> etc. This function must be kept pure i.e., it must return the same result each time it is invoked.
https://medium.com/@vigowebs/frequently-asked-react-js-interview-questions-and-answers-36f3dd99f486

16. What are controlled and uncontrolled components in React?

Ans: In a **controlled component, the form data is handled by the state within the component.** The state within the component serves as “the single source of truth” for the input elements that are rendered by the component.

```
state = { message: '' }
updateMessage = newText => { this.setState({( message: newText }); }
<input type="text"
        placeholder="enter message"
        value={this.state.message}
        onChange{(event) => this.updateMessge(event.target.value)}
/>
```

1. The textbox has a value attribute bound to the message property in the state.
2. We have an onChange event handler declared.

These 2 points tell you that this is a controlled component.

Uncontrolled components act more like traditional HTML form elements. The data for each input element is stored in the DOM, not in the component. Instead of writing an event handler for all of your state updates, you use a ref to retrieve values from the DOM.
Have to use React.createRef();
https://itnext.io/controlled-vs-uncontrolled-components-in-react-5cd13b2075f9
Uncontrolled components can be useful when integrating with non-React code (e.g if you need to support some kind of jQuery form plugin).
https://www.sitepoint.com/react-interview-questions-solutions/

17. Explain the components of Redux

- **Action** — Actions are **payloads of information that send data from our application to our store.** They are the only source of information for the store. We send them to the store using store.dispatch(). They are an object describes **what happened** in our app. **Require Type (String)**
- **Reducer** — Reducers specify **how the application’s state changes in response to actions sent to the store**. Remember that actions only describe what happened, but don’t describe how the application’s state changes. So this place determines how state will change to an action.
- **Store** — The Store is the object that brings Action and Reducer together. The store has the following responsibilities: **Holds application state**; **Allows access to state** via getState(); Allows **state to be updated** via dispatch(action); **Registers listeners** via subscribe(listener); **Handles unregistering of listeners** via the function returned by subscribe(listener).

https://medium.com/@vigowebs/frequently-asked-react-js-interview-questions-and-answers-36f3dd99f486

18. What is the second argument that can optionally be passed to setState and what is its purpose?
Ans: **A callback function** which will be invoked when setState has finished and the component is re-rendered. Since the setState is asynchronous, which is why it takes in a second callback function. With this function, we can do what we want immediately after state has been updated.
https://medium.com/@vigowebs/frequently-asked-react-js-interview-questions-and-answers-36f3dd99f486

19. Why call setState instead of directly mutating state?
Ans: If you try to mutate a component’s state directly, React has **no way of knowing that it needs to re-render the component**. By using the setState() method, React can update the component’s UI.
https://www.sitepoint.com/react-interview-questions-solutions/

20. How do you restrict the type of value passed as a prop, or make it required?
Used to declare the type of value that's expected and whether the prop is required or not:

```
import PropTypes from 'prop-types';
Greeting.propTypes = {
  name: PropTypes.string
};
```

https://www.sitepoint.com/react-interview-questions-solutions/

21. What's prop drilling and how can you avoid it?
Ans: Prop drilling is what happens when you need to **pass data from a parent component down to a component lower in the hierarchy,** "drilling" through other components that have no need for the props themselves other than to pass them on.
Sometimes prop drilling can be avoided by refactoring your components, avoiding prematurely breaking out components into smaller ones, and keeping common state in the closest common parent.
https://www.sitepoint.com/react-interview-questions-solutions/

22. What are React Hooks and advantages of them?
React hooks lets you **use state in Functional Components**
Advantages:
-Removing the need for class-based components, lifecycle hooks, and this keyword shenanigans
-Making it easier to reuse logic, by abstracting common functionality into custom hooks
-More readable, testable code by being able to separate out logic from the components themselves

23. Hoisting?
Ans - JavaScript's default behavior of **moving declarations to the top.** In JavaScript, a variable can be declared after it has been used. In other words; a variable can be used before it has been declared.

```
Ex: x = 5; console.log(x); /* undefined */ var x;
console.log(hoist); // Output: ReferenceError: hoist is not defined ...
let hoist = 'The variable has been hoisted.';
****
let hoist; console.log(hoist); /* Output: undefined */  hoist = 'Hoisted'
```

https://www.digitalocean.com/community/tutorials/understanding-hoisting-in-javascript#:~:text=Hoisting%20is%20a%20JavaScript%20mechanism,scope%20is%20global%20or%20local.
https://www.w3schools.com/js/js_hoisting.asp

24. Var vs Let vs Const

```
funct2 = () => {
        for (var i = 0; i < 5; i++)   {   console.log(i)     }
        console.log("Var i = ",i);   // no problems, works out of scope.
}
funct1 = () => {
        for (let i = 0; i < 5; i++)   {     console.log(i)    }
        console.log("Let i = ",i);     // i is undefined - because it is out of scope
}
```

var - globally or function/locally scoped. Var variables can be redeclared. var greeter = "hey hi";
   var greeter = "say Hello instead";

-var declarations are globally scoped or function scoped while let and const are block scoped
-var variables can be updated and re-declared within its scope; **let variables can be updated but not re-declared**; const variables can neither be updated nor re-declared.
https://www.freecodecamp.org/news/var-let-and-const-whats-the-difference/#:~:text=var%20declarations%20are%20globally%20scoped%20or%20function%20scoped%20while%20let,be%20updated%20nor%20re%2Ddeclared.

25. Closures - a closure gives you **access to an outer function's scope from an inner function**. In JavaScript, closures are created every time a function is created, at function creation time.
Closures - A closure is the combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment). In other words, a closure gives you access to an outer function's scope from an inner function. In JavaScript, closures are created every time a function is created, at function creation time.
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures

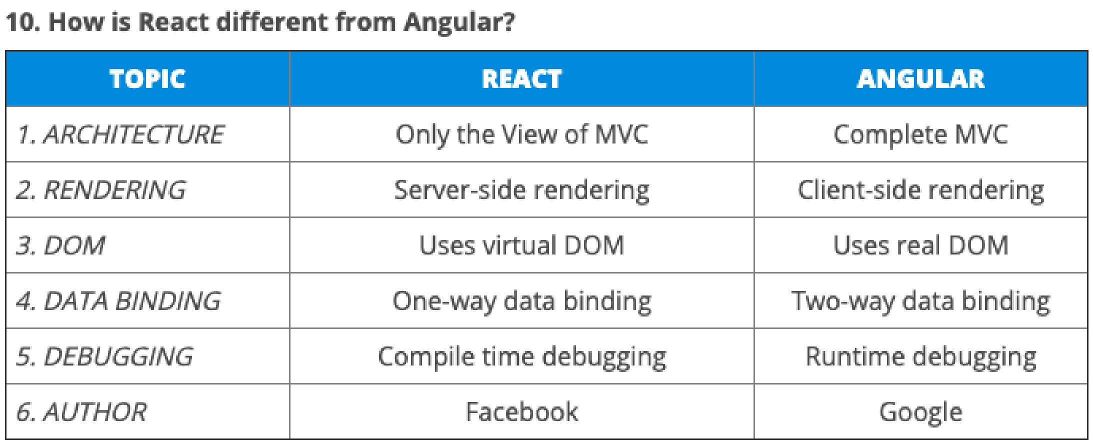
```
function init() {
  var name = 'Mozilla'; // name is a local variable created by init
  function displayName() { // displayName() is the inner function, a closure
    alert(name); // use variable declared in the parent function
  }
  displayName();
}
init();
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures

26. Errors - what happens after a REST call if there are errors. What does it do? - You need to stop the process and return a message that that specific call cannot work

27. Redux flow

28. React vs Angular -



10. How is React different from Angular?

TOPIC	REACT	ANGULAR
1. ARCHITECTURE	Only the View of MVC	Complete MVC
2. RENDERING	Server-side rendering	Client-side rendering
3. DOM	Uses virtual DOM	Uses real DOM
4. DATA BINDING	One-way data binding	Two-way data binding
5. DEBUGGING	Compile time debugging	Runtime debugging
6. AUTHOR	Facebook	Google

React vs Angular

29 JavaScript algorithms

30. JavaScript data structures

31. == vs ===
Both are comparison operators. The difference between both the operators is that,“==” is used to compare values whereas, “ === “ is used to compare both value and types.

32. **What is the difference between classical inheritance and prototypal inheritance?**
Class Inheritance: instances inherit from classes (like a blueprint — a description of the class), and create sub-class relationships: hierarchical class taxonomies. Instances are typically instantiated via constructor functions with the `new` keyword. Class inheritance may or may not use the `class` keyword from ES6.
Prototypal Inheritance: instances inherit directly from other objects. Instances are typically instantiated via factory functions or `Object.create()`. Instances may be composed from many different objects, allowing for easy selective inheritance.
https://medium.com/javascript-scene/10-interview-questions-every-javascript-developer-should-know-6fa6bdf5ad95

```
let x = 3;
let y = "3";
console.log(x - y);    //Returns 0 since the variable y (string type) is converted to a number type
var a = 12;
var b = "12";
a == b // Returns true because both 'a' and 'b' are converted to the same type and then compared.
```

33. Is javascript a statically typed or a dynamically typed language?
JavaScript is a dynamically typed language. In a dynamically typed language, the type of a variable is checked during run-time in contrast to statically typed language, where the type of a variable is checked during compile-time.
Since javascript is a loosely(dynamically) typed language, variables in JS are not associated with any type.

```
var a = 23;
var a = "Hello World!";
```

34. Explain passed by value and passed by reference.
In JavaScript, primitive data types are passed by value and non-primitive data types are passed by reference.
It's always pass by value, but for objects the value of the variable is a reference. Because of this, when you pass an object and change its members, those changes persist outside of the function.

35. What is an Immediately Invoked Function in JavaScript?
An Immediately Invoked Function (known as IIFE and pronounced as IIFY) is a function that runs as soon as it is defined.

```
(function(){
  // Do something;
})();
```

36. Explain Higher Order Functions in JS
Functions that operate on other functions, either by taking them as arguments or by returning them, are called higher-order functions.

```
function higherOrder2() {
  return function() {
    return "Do something";
  }
}
var x = higherOrder2();
x()   // Returns "Do something"
```

37. Explain “this” keyword.
The “this” keyword refers to the object that the function is a property of.

38. Explain Scope and Scope Chain in javascript.
Scope in JS, determines the accessibility of variables and functions at various parts in one’s code.
Global - Variables or functions declared in the global namespace have global scope, which means all the variables and functions having global scope can be accessed from anywhere inside the code
Function Scope - Any variables or functions declared inside a function have local/function scope, which means that all the variables and functions declared inside a function, can be accessed from within the function and not outside of it.
Block Scope - Block scope tells us that any variable declared inside a block { }, can be accessed only inside that block and cannot be accessed outside of it.

```
{ let x = 45; }
console.log(x); // Gives reference error since x cannot be accessed outside of the block
```

39. What are object prototypes?
All javascript objects inherit properties from a prototype. Ex: Array objects inherit properties from the Array prototype, Date objects inherit properties from the Date prototype

40. 16. What are callbacks?
A callback is a function that will be executed after another function gets executed.
Functions that are used as an argument to another function are called callback functions.

```
function divideByHalf(sum){
  console.log(Math.floor(sum / 2));  }
function multiplyBy2(sum){
  console.log(sum * 2);  }
function operationOnSum(num1,num2,operation){
  var sum = num1 + num2;
  operation(sum);  }
operationOnSum(3, 3, divideByHalf); // Outputs 3
operationOnSum(5, 5, multiplyBy2); // Outputs 20
```

41. What is memoization?
Memoization is a form of caching where the return value of a function is cached based on its parameters. If the parameter of that function is not changed, the cached version of the function is returned.

```
function memoizedAddTo256() {
  let cache = {};
  return function(num){
    if(num in cache){
      console.log("Cached value = ",cache[num]);
      return cache[num]   }
    else {
      cache[num] = num + 256;
      console.log("Saving Cashed Value = ",cache[num]);
      return cache[num];  }  }    }
let memoizedFunc = memoizedAddTo256();
console.log("First Call = ",memoizedFunc(20)); // Normal return
console.log("Second Call = ",memoizedFunc(20)); // Cached return
```

In the code above, if we run memoizedFunc function with the same parameter, instead of computing the result again, it returns the cached result.
**Note- Although using memoization saves time, it results in larger consumption of memory since we are storing all the computed results.

42. What is the use of a constructor function in javascript?
Constructor functions are used to create objects in javascript.

43. What is DOM?
DOM stands for Document Object Model. DOM is a programming interface for HTML and XML documents.
When the browser tries to render a HTML document, it creates an object based on the HTML document called DOM.
Using this DOM, we can manipulate or change various elements inside the HTML document.

pass function vs pass invoking function

main components
mapStateToProps - returns obj of state of redux into component. To access store. This triggers a rerender if state changes. Comes from Campsites reducer
which part of redux store am i listening to
mapstateToProps accesses configureStore, configureStore imports reducers. Therefore mapStateToProps accesses reducers from configureStore

mapDispatchToProps - writing data - super power actions you created in ActionCreators. From one function to another to execute
postcomment - dispatch a function - get from actionCreators
just postComment, no postComment(campsiteId, rating, ………
resetFeedBackForm, but the function is not resetFeedBackForm

connect() - to connect to the store
Currying - takes() runs it and passes values into second() for connect()(Main)

ActionCreators - Action (object (type, payload (optional)))
actionCreators - dispatch -> configureStore
all reducers listens to all actions

fetchCampsites - actionCreators - action, dispatch action, to reducer