Interview Questions

Javascript/ React

General Questions

- Tell me about yourself, What have you been working on in recent years?
- Have you used agile methodologies? Which ones?
    - Scrum, Canvan, etc
    - How do you estimate tickets? Planning
    - Teamwork?
- Did you have leadership roles in the past?
    - tech lead, do code reviews, do some coaching, create app from scratch
- What would you do if you have a lot of things on your plate and you know you will not have the time to finish it all?

Exercises

1. If you have 2 arrays A and B where you don't know how many elements they have,

    * Write a function that returns a new array alternating between both.

    * For example:

    * If you receive A [a, b, c, d] B [1,2,3,4,5], it must return C[a,1,b,2,c,3,d,4,5]

    * If you receive A[a,b,c,d,e] B[1,2] it should return C[a,1,b,2,c,d,e]

2. Write a javascript function that returns a boolean when the received parameter is valid.

    * The received parameter must be a string, otherwise it must throw an exception.

* Let supose the received string is code. The code is valid only if every opened bracket has its corresponding closing bracket.

Example of valid inputs:

{}, {kkkk{}}, {{gggooo}}{ll{i}}

Examples of invalid inputs

{}{, }}}{{{, {}{}oooo{ppp}}}

3.

```
var myAge = 31;
function greet() {
  var myName = 'Max';
  function printInfo() {
    console.log('Hello ' + myName);
    console.log('I am ' + myAge + ' years old.');
  }
  printInfo();
}

greet();
console.log(myName);
printAge();
```

4.Shadowing

```
const myAge = 31;
function printAge() {
  let myAge = 35;
```

```
        console.log(myAge);
    }
    printAge();
```

```
5.// Predict WHAT the output will be and WHY that is the case
console.log("1. 1 + '5'");
console.log(1 + '5'); // '1' + '5' => '15'

console.log("2. 3 - '5'");
console.log(3 - '5'); // 3 - 5 => -2

console.log("3. 'hi' + ' there'");
console.log('hi' + ' there'); // 'hi there'

console.log("4. true * 3");
console.log(true * 3); // 1 * 3 => 3

console.log("5. 'it is ' + true");
console.log('it is ' + true); // 'it is ' + 'true' => 'it is true'

console.log("6. 5 == '5'");
console.log(5 == '5'); // '5' == '5' => true

console.log("7. 0 == false");
```

```
console.log(0 == false); // 0 == 0 => true


console.log("8. 'true' == true");

console.log('true' == true); // NaN == true => false

// Boolean('true'), if('true'), !!'true' => true


console.log("9. '' == false");

console.log('' == false); // 0 == 0 => true


console.log("10. '' === false");

console.log('' === false); // '' === false => false


console.log("11. [] + 1");

console.log([] + 1); // '' + 1 => '' + '1' => '1'


console.log("12. [] == false");

console.log([] == false); // '' == false => 0 == 0 => true


console.log("13. Boolean([])");

console.log(Boolean([])); // true


console.log("14. [] == {}");

console.log([] == {}); // false

// [] == [] ==> false because of that primitive/ reference value thing
```

```
console.log("15. [] * {}");

console.log([] * {}); // [] = '' = 0, {} = '[object Object]' = NaN => 0 * NaN => NaN


console.log("16. [] * 3");

console.log([] * 3); // '' = 0 => 0 * 3 => 0


console.log("17. {} + ' this works'");

console.log({} + ' this works'); // '[object Object]' * ' this works' => '[object Object] this works'
```

JavaScript

- Callback?
    - A callback function is a function which is:
        - accessible by another function, and
        - Is invoked after the first function if that first function completes
- Promise?
    - A promise is an object that may produce a single value some time in the future: either a resolved value, or a reason that it’s not resolved . A promise may be in one of 3 possible states: fulfilled, rejected, or pending
- What are some benefits of using promises instead of callbacks?
    - Avoid callback hell
    - - Better error handling
    - - Better definition of control flow
    - - Better readability

- `What is the difference between == and ===?`
    - == -> allows coercion
    - === -> compares also the datatype (doesn't allow coercion).
- `What is the difference between let, const and var?`
    - Keyword Scope Hoisting Can Be Reassigned Can Be Redeclared
    - var: (Function scope), has hoisting, can be reassigned, can be redeclared.
    - let: (Block scope) No hoisting, can be reassigned, can not be redeclared.
    - const (Block scope) No hoisting, can not be reassigned (it is inmutable) (only its props or children can be reassigned), can not be redeclared.
- `Explain "destructuring" and why is it useful/`
    - The destructuring assignment syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.
- `Explain the difference between spread operator and rest operator`
    - The spread operator allows us to spread the value of an array (or any iterable) across zero or more arguments in a function or elements in an array (or any iterable). The rest parameter allows us to pass an indefinite number of parameters to a function and access them in an array.
- `Using ES6 modules, how can you share code between files?`
    - Using the import keyword and default/named export object:
      import foo from '/modules/foo.js';
    - Using the require keyword and the module.exports object:
      const foo = require('./foo');
    - Using the require keyword from a named function in a designated file:
      const foo = require('./foo');
    - Using the module keyword:
      from 'foo.js' import foo;

- Describe event bubbling
- if you try to get a 'type of' of string, you get a string. If you try to get a 'type of' of 'null', what do you get?
    - The answer is an object underneath with no assigned value explicitly.

Senior Questions

- Coercion?
    - Type coercion is the process of converting value from one type to another
- Please mention which are the primitive and non primitive data types
- What does variable hoisting mean?
    - JavaScript hoisting occurs during the creation phase of the execution context that moves the variable and function declarations to the top of the script.

React

Cuál es la diferencia entre props y state?

El state se refiere al valor de datos predeterminado en un componente de React. El state de un componente puede cambiar con el tiempo y un desarrollador puede cambiarlo manualmente. Props, por otro lado, describe cómo se configura un componente de React. Los accesorios no cambian.

Qué es JSX?

- JSX es una herramienta que le permite insertar plantillas HTML sin procesar dentro del código JavaScript. (JSX is a shorthand for JavaScript XML)

Why can’t browsers read JSX?

Browsers can only read JavaScript objects but JSX is not a regular JavaScript object. So to enable a browser to read JSX, first, we need to transform a JSX file into a JavaScript object using JSX transformers like Babel and then pass it to the browser.

El navegador no puede leer JSX, por lo que debe utilizar herramientas como webpack y Babel para traducirlo a JavaScript legible por navegador.

What are refs used for in React?

- *Refs* are an escape hatch which allow you to get direct access to a DOM element or an instance of a component. In order to use them you add a ref attribute to your component whose value is a callback function which will receive the underlying DOM element or the mounted instance of the component as its first argument.

List some of the cases when you should use Refs.

- When you need to manage focus, select text or media playback
- To trigger imperative animations
- Integrate with third-party DOM libraries

Define Reducers in React?

Reducers are the pure functions that clearly state as to how the application state changes when certain actions are made. This way, it takes into account the previous state and action to turn out to a new state.

¿Usaste alguna vez Redux Saga?

The basic idea of Redux is that the entire application state is kept in a single store which is simply a JavaScript object

What is a store in Redux?

A store in Redux is a JavaScript object that can hold applications state and provide help to access them and apply dispatch actions along with register listeners.

What are the lifecycle methods of ReactJS?

- **componentWillMount**: Executed before rendering and is used for App level configuration in your root component.
- **componentDidMount**: Executed after first rendering and here all AJAX requests, DOM or state updates, and set up eventListeners should occur.
- **componentWillReceiveProps**: Executed when particular prop updates to trigger state transitions.
- **shouldComponentUpdate**: Determines if the component will be updated or not. By default it returns true. If you are sure that the component doesn't need to render after state or props are updated, you can return false value. It is a great place to improve performance as it allows you to prevent a rerender if component receives new prop.
- **componentWillUpdate**: Executed before re-rendering the component when there are pros & state changes confirmed by shouldComponentUpdate which returns true.
- **componentDidUpdate**: Mostly it is used to update the DOM in response to prop or state changes.
- **componentWillUnmount**: It will be used to cancel any outgoing network requests, or remove all event listeners associated with the component

What are React Hooks?

with Hooks, you can extract stateful logic from a component so it can be tested independently and reused. Hooks allow you to reuse stateful logic without changing your component hierarchy. This makes it easy to share Hooks among many components or with the community.

Will it work if we remove the [] argument from `useEffect`?

Yes, but we'll call useEffect hook on every render which may cause performance issues.

`Examples of`

- Use Effect only on didunmount
- Use effect only on the first render
- Use effect with params
- Which methods are replaced by using the "useEffect" Hook

How can I prevent unnecessary re-rendering?

- **React.PureComponent** : Components created off of this class do a shallow comparison with the upcoming props and state and re-render if there are changes
- **React.Memo**: Higher-order component that works like React.PureComponent but used for function components
- **shouldComponentUpdate** : Lifecycle method that takes in the next props and state and returns a boolean indicating if the component should rerender

What causes a component to update?

- A re-render of the parent which may entail new Props
- setState()
- forceUpdate() (and that it should be avoided!)

What is the useState hook?

- useState is a Hook that lets you add React state to function components
- useState like all Hooks is a function
- **Argument**: the initial state
- **Returns**: Pair containing the current state and a function to update it.

What is the useEffect hook?

- useEffect lets you perform side effects in function components
- useEffect is triggered after a render
- useEffect is like the combination of componentDidMount, componentDidUpdateand componentWillUnmount
- **Arguments:** function to be called, and an array for React to check for changes in order to render

¿Unit Test?

JEST y Enzyme?

React Memo ?

What is Context?

Context enables passing data through the component tree without having to pass props down manually at every level. Which means you can deep-nest items without issue.

Vue

Que son props?

Every component instance has its own isolated scope. This means you cannot (and should not) directly reference parent data in a child component's template. Data can be passed down to child components using props. Props are custom attributes you can register on a component. When a value is passed to a prop attribute, it becomes a property on that component instance.

```
Vue.component('blog-post', {
  // camelCase in JavaScript
  props: ['postTitle'],
  template: '<h3>{{ postTitle }}</h3>'
})
```

What is filters in Vue.js?

Vue.js allows you to define filters that can be used to apply common text formatting. Filters are usable in two places: mustache interpolations and v-bind expressions (the latter supported in 2.1.0+). Filters should be appended to the end of the JavaScript expression, denoted by the “pipe” symbol:

```
<div v-bind:id="rawId | formatId"></div>
```

What are all the life cycle hooks in Vue instance?

Each Vue instance goes through series of steps when they are created, mounted and destroyed. Along the way, it will also runs functions called life cycle hooks, giving us the opportunity to add our own code at specific stage. Below are the events, a Vue instance goes through.

beforeCreate — The first hook in the creation hooks. They allow us to perform actions before our component has even been added to the DOM. We do not have access to the DOM inside of this.

created — This hook can be used to run code after an instance is created. We can access the reactive data. But templates and Virtual DOM have not yet been mounted or rendered.

beforeMount — The beforeMount hook runs right before the initial render happens and after the template or render functions have been compiled. Most likely we’ll never need to use this hook.

mounted — We will have full access to the reactive component, templates, and rendered DOM. This is the most frequently used hook.

beforeUpdate — This hook runs after data changes on our component and the update cycle begins. But runs right before the DOM is patched and re-rendered.

updated — This hook runs after data changes on our component and the DOM re-renders. If we need to access the DOM after a property change, here is probably the safest place to do it.

beforeDestroy — This hook will run right before tearing down the instance. If we need to clean up events or reactive subscriptions, this is the right place to do it.

destroyed — This hook will be used to do any last minute clean up.

What is the difference between v-show and v-if directives?

Below are some of the main differences between between v-show and v-if directives:

- v-if only renders the element to the DOM if the expression passes whereas v-show renders all elements to the DOM and then uses the CSS display property to show/hide elements based on expression.
- v-if supports v-else and v-else-if directives whereas v-show doesn't support else directives.
- v-if has higher toggle costs since it add or remove the DOM every time while v-show has higher initial render costs. i.e, v-show has a performance advantage if the elements are switched on and off frequently, while the v-if has the advantage when it comes to initial render time.
- v-if supports tab but v-show doesn't support.

What is the role of ref in Vue.js?

Despite the existence of props and events, sometimes if we still need to directly access a child component, we can assign a reference ID to the child component using the ref attribute. For example:

CSS

- Media queries
- Sass, less , css preprocessors
- z-index
- Css Bad practices
- Difference between position absolute, fixed, relative