Javascript

1. What is your favourite new javascript feature and why?

Answer:

Private fields:

JS is getting private fields in classes.

Static and instance public fields are writable, enumerable, and configurable properties. As such, unlike their private counterparts, they participate in prototype inheritance.

2. Explain an interesting way in which you have used this javascript feature.

Answer:

you would be able to create a new validator instance whenever you had something new to validate (i.e., if you had two different forms on your website you might create one validator for each)

3. Is there any difference between regular function syntax and the shorter arrow function syntax? (Write the answer in your own words)

Answer:

Regular functions created using function declarations or expressions are 'constructible' and 'callable'. Since regular functions are constructible, they can be called using the 'new' keyword. However, the arrow functions are only 'callable' and not constructible. Thus, we will get a run-time error on trying to construct a non-constructible arrow functions using the new keyword.

4. What is the difference between 'myFunctionCall(++foo)' and 'myFunctionCall(foo++)'

Answer: I think it is simplest to understand in terms of the order of evaluation and modification.

The postfix operator (x++) evaluates first and modifies last.

The prefix operator (++x) modifies first and evaluates last.

(the order of operations is evident in the typed order of the operator and the variable)

5. In your own words, explain what a javascript 'class' is and how it differs from a function.

Answer:

Class: A class is a type of function, but instead of using the keyword function to initiate it, we use the keyword class, and the properties are assigned inside a constructor() method.

Please see syntax in attached question-5.js

Syntax :

```
class car {
    constructor(brand) {
        this.carname = brand;
    }
}
```

Difference:

class its nothing but a syntactical sugar over javascript logic class creation using function. if you are using a function as class the entire function is act as a constructor, if you want to put other member functions you need to do that in constructor

like

this.something = ..., Or

var something = ...

in case of private members (if you are not injecting from outside, assume you are creating object with other methods / properties), but in case of class the entire function is not actually act a constructor you can explicitly separate it with other member functions and data.

CSS

1. In your own words, explain css specificity.

Answer: CSS Specificity means setting rules for CSS selectors

2. In your own words, explain, what is '!important' in css. Also how does it work? Are there any special circumstances when using it, where it's behaviour might not be what you expect?

Answer:

!important: It's keyword to set highest priority of that attribute.

Key-value pair with !important will have first priority than other CSS.

3. What is your prefered layout system: inline-block, floating + clearing, flex, grid, other? And why?

Answer: I prefer floating layout because Large layout blocks that don't need equal heights and vertical centering.

4. Are negative margins legal and what do they do (margin: -20px)?

Answer: Negative margins are legal.

Margin:-20px: It will move content in left by -20px and right viewport to hide it.

5. If a has no margin or other styling and a <p/> tag inside of it has a margin top of some kind, the margin from the <p/> tag will show up on the div instead (the margin will show above the div not inside of it), why is this? What are the different things that can be done to prevent it?

Answer:

The CSS margin properties are used to create space around elements, outside of any defined borders.

 has no margins or styling, that's why <p/> margin will be show above the div.

If you want <p/> inside of , you have to set padding top to <p/> .

Unit tests

1. What technologies do you use to unit test your react components?

Answer: Jest technology

2. Are there any pitfalls associated with this technology that have caused you difficulty in the past?

Answer: It's not lightweight as Vue.JS

3. How do you test in your unit tests to see if the correct properties are being passed to child components.

Answer:

React:

14. React test step1:

Create a react component that has a with a border.

Inside this should be a that displays the 'live' width of the browser window at all times. Keep in mind that the size of the window could easily be changed by the user and you should reflect this.

Answer:

Please see example in attached question-14.js

```
class Example1 extends React.Component {
    render() {
        var divStyle = {
            border: '2px solid red'
        };
        var spanStyle ={
            width:$(window).width();
        };
        return (
          <div style={divStyle}>
            <span style={spanStyle}>live</span>
          </div>
        )
    }
}
ReactDOM.render(<Example1 />, document.getElementById('app'));
```

15. React test step2:

Inside the you created in the previous step, add a text input that, as a number is entered into it, uses that number to set the height of the div itself in pixels, live as you update the text field (keypress not change event).

Answer:

Please see example in attached question-15.js

```
class Example2 extends React.Component {
    constructor() {
        super();
        this.state ={
            divHeight:0
```

```
                    }
                }
            handleKeyPress(e){
                let divHeightValue= e.target.value+'px';
                this.setState({divHeight: divHeightValue});

            }
            render() {
                var divStyle = {
                    border: '2px solid red'
                    height: this.state.divHeight;
                };
                var spanStyle ={
                    width:$(window).width();
                };

                return (
                  <div style={divStyle}>
                    <span style={spanStyle}>live</span>
                    <input  type="number" value={this.state.divHeight}
onKeyPress={this.handleKeyPress.bind(this)}/>
                  </div>
                )
              }
        }
        ReactDOM.render(<Example2 />, document.getElementById('app'));
```

16. React test step3:

Add the following code to your project root (same project as in step 2, but add the code in the global / window space):

```
Let divHeight;
window.setDivHeight = (height) => divHeight = height;
```

Add a HOC for your div component that allows you to set the height of your component from the previous steps by calling that external function.

If you do not know what a HOC is or how to create one, that is also fine, just mention that in your answer and instead create a parent component that can still do this (allow you to call that function 'setDivHeight' in order to set the height of the div manually.

Bare in mind that when the height of the div is forcefully set like this, the text fields value should also update to reflect this and should still carry on working as normal (user can continue to modify its value).

Answer:

Please see example in attached question-16.js

```
class Example3 extends React.Component {
        constructor() {
                super();

        }
        render() {
                return (
                  <Example2 />
                )
         }
}
class Example2 extends React.Component {
          constructor() {
                super();
                this.state ={
                        divHeight:0
                        }
                }
        handleKeyPress(e){
                let divHeightValue= e.target.value+'px';
                this.setState({divHeight: divHeightValue});

        }
        render() {
                var divStyle = {
```

```
                    border: '2px solid red'
                    height: this.state.divHeight;
                };
                var spanStyle ={
                    width:$(window).width();
                };

                return (
                  <div style={divStyle}>
                    <span style={spanStyle}>live</span>
                    <input  type="number" value={this.state.divHeight}
onKeyPress={this.handleKeyPress.bind(this)}/>
                  </div>
                )
            }
        }
        ReactDOM.render(<Example3 />, document.getElementById('app'));
```

RXjs

17. What are the differences between Subject, BehaviorSubject and ReplaySubject? And in what situation would you use each of these (please provide example scenarios)?

Answer:

Difference:

Subject: A Subject, in contrast to an observable, is simply an observer that's also able to emit values. It's both an observable and an observer simultaneously. This is unlike an observable, as an observer that's subscribed to an observable can only read values emitted from an observable.

BehaviorSubject:

BehaviorSubject is a special type of Subject whose only different is that it will emit the last value upon a new observer's subscription.

For instance, in the above example of a regular Subject, when Observer 2 subscribed, it did not receive the previously emitted value 'The first thing has been sent' -- In the case of a BehaviorSubject, it would.

ReplaySubject:

It's like BehaviorSubject, except it allows you to specify a buffer, or number of emitted values to dispatch to observers. BehaviorSubject only dispatches the last emitted value, and ReplaySubject allows you to dispatch any designated number of values.

18. If you have an array of values in a stream and you wish to pipe it such that it will emit the arrays values individually, one by one and wait for them all to be completed before processing another array, how would you do this? Please provide a code example.

Answer: We pass a predicate as a parameter to the operator, and if it returns true for the event being streamed, the event will be passed through the pipeline, otherwise, it will be discarded.

Please see example in attached question-18.js

Example:

```
const numbers$ = of(1, 2, 3, 4, 5);

const predicate = (n) => n <= 2;

numbers$

        .pipe(

                filter(predicate)

         )

        .subscribe(console.log);

// will log 1,2
```

19. If you have a stream that receives individual values and would like to pipe it such that it builds an array out of these values, emitting the updated array each time a new value is added to it, how would you do this? Please provide a code example.

Answer: Filtering operators allow us to filter events from the stream that we want to disregard and avoid sending them to the observable's subscribers.

If we filter the events soon enough in the pipeline, we avoid passing them down to other operators and to the subscription callback.

Of course, this is especially important if the pipeline is doing heavy-computations or HTTP requests.

Please see example in attached question-19.js

```
const node = document.querySelector('input[type=text]');


const input$ = Rx.Observable.fromEvent(node, 'input');


input$.subscribe({
  next: event => console.log(`You just typed ${event.target.value}!`),
  error: err => console.log(`Oops... ${err}`),
  complete: () => console.log(`Complete!`),
});
```

Twilio:

20. Explain which of the Twilio Api's you have used. Also explain how and in what scenarios you have used them.

Answer: We have used Twilio for sending text messages, gathering number inputs on call.

- Text Message Send Scenario: To send new offers, membership due date & expiry notifications
- Gathering number inputs: OTP verification