Please answer the questions below.

Where we ask for you to provide code, this code can just be 'pseudo code' and does not have to be executable.

The idea of these questions are basically just to get an idea of your understanding. You are allowed to skip questions if you cannot answer but of course the more you can answer the better.

Completing all the tasks should not take more than an hour to an hour and a half.

Please create a repo on github to host your answers and test code. You can either do this on your own public repo or share 'kkroese' and 'devblazer' on your private github repo. Please do not include any OS specific files like microsoft word documents. Also do not commit binary files like zip, tar or gz files.

Please send the link to your repository to Kevin.

Javascript:

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

   Ans : Dynamic Import feature, through which we can import JS files dynamically in our applications. The standard import syntax is static and results in all of the code of the imported module being evaluated at load time. In situations where we wish to load a module conditionally or on demand, we can use a dynamic import instead.

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

   Ans : I have used dynamic import when I had to create a multilingual web application in ReactJS that supports three languages: English, French, and Russian. The app had to import all the three translation files, irrespective of the user's current locale, but used just one translation file for a particular user. In that case, all three translation files were being downloaded. The translation files are bulky, thus the application would show a poorer performance. So, instead of downloading all of these at once, I used dynamic imports and only downloaded the file that is required at the moment for that user. This is how I managed to do it:
   const userLocale = { locale: 'fr' }
   import(`./${userLocale.locale}-translations.js`)

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

Ans : Yes, there are differences between Regular functions and Arrow functions, one of the major differences is that return keyword is used to return from a regular function, while the arrow function doesn't need a return statement. Regular functions can easily construct objects but arrow functions can not be used as constructor.

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

   Ans : ++ is an increment operator, i.e., it increases the value of its operand by 1. We have two types of increment operators in Javascript:
   a). <u>Pre-increment operator</u>: ++foo represents a pre-increment operator. This means that the value of variable foo is incremented by 1 and the new incremented value is passed in the function.
   b). <u>Post-increment operator</u>: foo++ represents a post-increment operator. This means that the variable foo is passed in the function with its original value and after the execution of the function, the value of foo will be incremented by 1. The incremented value of foo will be used further in the program.

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

   Ans : A class in javascript is a representation of data members and functions. A function is a set of specific tasks while a class is a blueprint for defining one or more functions.

Css:

6. In your own words, explain css specificity.

   Ans : CSS Specificity basically defines the rank/priority of CSS value applied on different elements, and styling will be applied on the element according to the value with the highest priority.

7. 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 behavior might not be what you expect?

   Ans : !important in css is basically used for giving more importance to a particular CSS value that we want to apply, it's given the priority above others not marked !important. We just need to write !Important at the end of our rule and it will be applied, instead of any other CSS value, even if the other value has a higher priority.

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

   Ans : It depends on the type of layout we want to achieve. However, I mostly prefer

flex as it gives better flexibility and allows a lot of control with aligning items. We can easily control the directions of laying out the children.

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

   Ans : Yes, negative margins are legal. Negative margins shrink the space of the parent container, thus reducing its distance from the neighboring objects .

10. If a <div/> 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?

    Ans : This is caused by a behavior called 'collapsing margins'. The expression collapsing margins means that adjoining margins of two or more boxes combine to form a single margin, when no content, padding, or border areas, or clearance separate them.

    There are two main types of margin collapse:
    i. Collapsing margins between adjacent elements
    ii. Collapsing margins between parent and child elements

    This behavior can be prevented in the following possible ways:
    a). Giving the parent a floating property
    b). Giving the parent an absolute positioning
    c). Making the parent display: inline-block
    d). Setting the parent overflow to anything other than visible
    e). Cleared float
    f). In newer browser, it can also be prevented by using display: flow-root
    g). Using a Flexbox or Grid

Unit tests:

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

    Ans : I prefer to use Jest or cypress.io for testing react components.

12. Are there any pitfalls associated with this technology that have caused you difficulty in the past?
    Ans :
    Jest: Jest's biggest weakness is being newer and less widely used among JavaScript developers. Besides, it has less tooling and library support available

compared to more mature libraries (like Mocha).

<u>Cypress</u>**: Cypress has the following disadvantages that often cause difficulty:**
i). **Cypress don’t allow cross-browser testing**
ii). **Multiple tabs and windows not supported**
iii). **For the building of test cases, it only supports the JavaScript framework**
iv). **It is highly dependent on third-party plugins for many important features, such as XPath**

13. How do you test in your unit tests to see if the correct properties are being passed to child components.
Ans: While testing a react app in jest unit test, following steps can be followed to test if correct properties are being passed to child components.

Step1: Set up a Jest mock function to check any props. This happens outside of any test or it blocks.
Step 2: Mock the child component file, and grab all props passed to it and pass them to the Jest mock function to listen.
Step 3: Return something for React to render. I like creating an element to pass through as it helps with visually debugging if needed.
Step 4: Render the ParentComponent with the props needed to be tested.
Step 5: Check that the Jest mock function is called with an object. 'expect.objectContaining' must be used to make sure if any other default React props are ignored.

So, the complete test is as follows:

```
import React from "react";
import { render } from "@testing-library/react";
import ParentComponent from "./ParentComponent";

const mockChildComponent = jest.fn();
jest.mock("./ChildComponent", () => (props) => {
  mockChildComponent(props);
  return <mock-childComponent />;
});

test("If ParentComponent is passed open and has data, ChildComponent is called
with prop open and data", () => {
  render(<ParentComponent open data="some data" />);
  expect(mockChildComponent).toHaveBeenCalledWith(
    expect.objectContaining({
      open: true,
      data: "some data",
    })
  );
});

test("If ParentComponent is not passed open, ChildComponent is not called", () =>
{
  render(<ParentComponent />);
  expect(mockChildComponent).not.toHaveBeenCalled();
});
```

React:

14. React test step1:

    Create a react component that has a <div/> with a border.
    Inside this <div/> should be a <span/> 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.

15. React test step2:

    Inside the <div/> 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).

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;
    ```

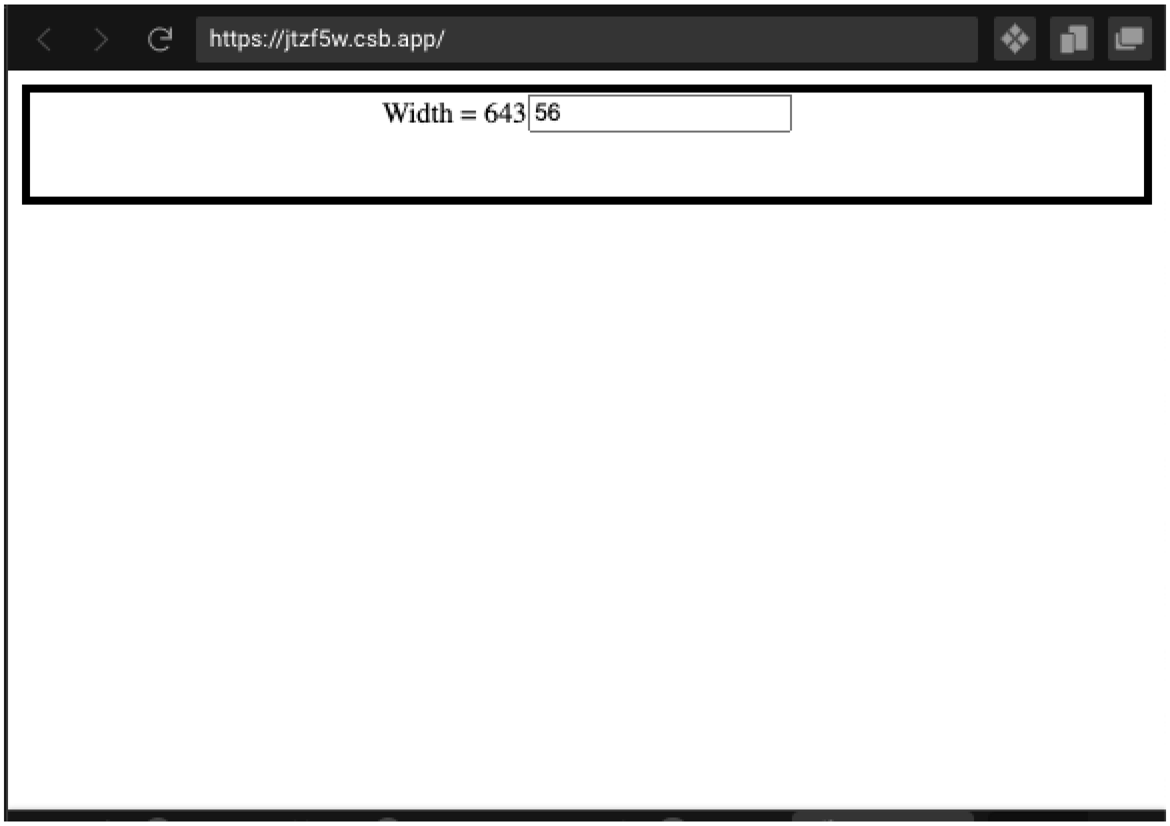
    Add a HOC for your div component that allows you to set the height of your <div/> component from the previous steps by calling that external function.

    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).

React

```
import "./styles.css";
import { useEffect, useState } from "react";

function App() {
  const [screenSize, getDimension] = useState({
    dynamicWidth: window.innerWidth,
    dynamicHeight: window.innerHeight
  });

  const setDimension = () => {
    getDimension({
      dynamicWidth: window.innerWidth,
      dynamicHeight: window.innerHeight
    });
  };

  const [divHeight, setDivHeight] = useState(25);

  useEffect(() => {
    window.addEventListener("resize", setDimension);
    return () => {
      window.removeEventListener("resize", setDimension);
    };
  }, [screenSize]);

  return (
    <div
      style={{
        border: "5px solid black",
        height: ${divHeight}px,
        textAlign: "center"
      }}
    >
      <span style={{ textAlign: "center" }}>
        Width = {screenSize.dynamicWidth}
      </span>
      <input
        type="number"
        id="test"
        value={divHeight}
        onChange={(e) => {
          setDivHeight(e.target.value);
        }}
      />
    </div>
  );
}
export default App;
```



```
https://jtzf5w.csb.app/
Width = 643 56
```