Salesforce JavaScript Developer I Certification Practice Exam (Sample)

Study Guide

Everything you need from our exam experts!

Questions

1. How can a developer handle errors when making asynchronous API calls in JavaScript?

A. By using if-else statements to catch errors.

B. By implementing the try-catch block.

C. By ignoring the error response.

D. By using console.log to track errors.

2. What is the main use of the `@wire` decorator in LWC?

A. To read data from external APIs

B. To read data from Salesforce and automatically update components

C. To trigger events in child components

D. To handle user inputs

3. What does the 'this' keyword refer to in JavaScript?

A. The global object

B. The parent function

C. The context in which a function is called

D. The last variable defined

4. Which decorator is used to make a property reactive in LWC?

A. @api

B. @track

C. @wire

D. @component

5. If a paragraph is clicked inside a div, what will be the output in the console?

A. p div

B. div p

C. p

D. div

6. How long will the data in sessionStorage remain accessible?

A. Forever, the data doesn't get lost.

B. When the user closes the tab.

C. When the user closes the entire browser, not only the tab.

D. When the user shuts off their computer.

7. When using forEach with an array, what console output will be produced by the logArrayElements function defined as console.log('a[' + index + '] = ' + element)?

A. a[0] = 2, a[1] = 5, a[2] = 9

B. 2, 5, 9

C. 0, 1, 2

D. undefined

8. What lines of code need to be added in the Student constructor to properly inherit from Person?

A. super(fName, lName, age); and Student.prototype = Object.create(Person.prototype);

B. Student.prototype = Person.call(this, fName, lName, age);

C. Person.call(this) and Student.prototype = new Person();

D. super(); and Object.create(Student.prototype);

9. What is the return type of methods defined in Lightning Web Components?

A. Always undefined

B. Any valid JavaScript type

C. Only arrays

D. Only objects

10. What does the valueOf() method do when used on a String object in JavaScript?

A. It converts the String object to a number

B. It returns the primitive string value

C. It creates a new string

D. It concatenates the string with another value

Answers

1. B
2. B
3. C
4. B
5. A
6. B
7. A
8. A
9. B
10. B

Explanations

1. How can a developer handle errors when making asynchronous API calls in JavaScript?

A. By using if-else statements to catch errors.

B. By implementing the try-catch block.

C. By ignoring the error response.

D. By using console.log to track errors.

Implementing a try-catch block is an effective way to handle errors in asynchronous API calls in JavaScript. This method allows developers to write cleaner code and manage exceptions in a controlled way. Specifically, when you use a try-catch block, you can attempt to execute code that may throw an error within the try section. If an error occurs, execution is immediately transferred to the catch block, where the developer can manage the error appropriately. This may include logging the error, showing a user-friendly message, or attempting a retry mechanism. The use of try-catch is particularly beneficial with asynchronous operations, such as promises and async/await syntax, because it allows for a structured error handling mechanism without cluttering the code with many conditional checks. This is crucial in maintaining the flow of applications that rely on multiple network requests, enabling developers to ensure a robust user experience even when issues occur with the backend or network. The other options do not provide effective error-handling methods. For example, if-else statements do not inherently capture exceptions that occur during asynchronous operations. Ignoring error responses can lead to unhandled exceptions and degraded application performance. Lastly, while console.log can be useful for tracking errors, it does not actually handle them; it simply outputs

2. What is the main use of the `@wire` decorator in LWC?

A. To read data from external APIs

B. To read data from Salesforce and automatically update components

C. To trigger events in child components

D. To handle user inputs

The `@wire` decorator in Lightning Web Components (LWC) is primarily used to read data from Salesforce and automatically update components. When a component is wired to a Salesforce data source using this decorator, it establishes a reactive connection. This means that whenever the underlying data changes in Salesforce, the component using `@wire` will automatically re-render to reflect those changes. This behavior enhances the efficiency of data management within LWC by eliminating the need for manual data fetching or event handling for data updates. Instead of writing code to periodically check for data updates or to manage state, developers can simply utilize the `@wire` decorator to ensure that their components always display the latest information from the Salesforce back end. The other choices represent valid functionalities relevant to web development, but they do not encapsulate the purpose of the `@wire` decorator as accurately as the chosen answer. For instance, reading data from external APIs is handled differently in LWC and is beyond the scope of the `@wire` decorator, which is specifically designed for Salesforce data sources. Similarly, triggering events and handling user inputs pertains to component interaction rather than data retrieval.

3. What does the 'this' keyword refer to in JavaScript?

A. The global object

B. The parent function

C. The context in which a function is called

D. The last variable defined

In JavaScript, the 'this' keyword is used to refer to the context in which a function is called. It represents the object that is currently executing the code and can vary based on how a function is invoked. This makes 'this' highly dynamic and context-sensitive. For example, if a method is called on an object, 'this' refers to that particular object. If the function is invoked in the global scope, 'this' refers to the global object (like `window` in browsers or `global` in Node.js). In the context of an event handler, 'this' refers to the element that the event is bound to. This behavior of 'this' allows for more flexibility in JavaScript, enabling developers to write code that's adaptable to different invocation contexts. Understanding the context of 'this' is crucial for effectively working with object-oriented programming and callbacks in JavaScript.

4. Which decorator is used to make a property reactive in LWC?

A. @api

B. @track

C. @wire

D. @component

In Lightning Web Components (LWC), the decorator used to make a property reactive is @track. When you apply the @track decorator to a property in a component, it enables the component to re-render whenever the property's value changes. This is important for dynamic user interfaces that need to update based on user interactions or changes in underlying data. The @track decorator ensures that the LWC engine is aware of changes to the property and can efficiently trigger a re-render of the component when necessary. Initially, @track was frequently used to mark properties of complex data types, such as objects or arrays, to monitor changes within them. However, with more recent updates, the reactivity in LWC has been enhanced, allowing for simpler state management and reducing the need for tracking certain properties explicitly. The other decorators serve different purposes: @api is used for public properties that can be accessed by parent components, @wire is used for connecting a component to data from a Salesforce service, and @component is not a valid LWC decorator. Therefore, @track is the correct choice for making properties reactive in LWC.

5. If a paragraph is clicked inside a div, what will be the output in the console?

A. p div

B. div p

C. p

D. div

When a paragraph is clicked inside a div, the output in the console will be "p div" because of the way event propagation works in the DOM. When an event occurs, such as a click, it travels through the DOM in two phases: the capturing phase and the bubbling phase. In the bubbling phase, the event starts from the target element (the clicked paragraph) and bubbles up to its parent elements (the div, in this case). In this scenario, if you have an event listener on the paragraph element and one on the div element, and you log the event or the elements in the listener, the context in which you're logging is important. If the logging is set up in such a way that it includes both the paragraph and its parent div, it may show the tags in the logged output as "p div," representing that the event originated from the paragraph and has bubbled up to the div. This understanding highlights how event handling and propagation affect the output you observe in the console when interacting with nested elements in the DOM.

6. How long will the data in sessionStorage remain accessible?

A. Forever, the data doesn't get lost.

B. When the user closes the tab.

C. When the user closes the entire browser, not only the tab.

D. When the user shuts off their computer.

The data in sessionStorage is designed to be temporary and is specific to the current browser tab. This means that the data stored in sessionStorage will remain accessible as long as that particular tab is open. However, once the user closes the tab, all data associated with that tab will be cleared and rendered inaccessible. This feature makes sessionStorage particularly useful for maintaining data for short-term use during a browsing session without persisting it beyond that tab's lifecycle. In contrast, the other options imply longer data retention than what sessionStorage offers. The option stating that data lasts forever is incorrect as sessionStorage is not permanent. Additionally, suggesting that data persists until the entire browser is closed or when the computer is shut off also misrepresents the nature of sessionStorage, as these scenarios would retain data in localStorage rather than sessionStorage. Therefore, the correct understanding lies in recognizing that the data is wiped from sessionStorage once the user closes the tab.

7. When using forEach with an array, what console output will be produced by the logArrayElements function defined as console.log('a[' + index + '] = ' + element)?

A. a[0] = 2, a[1] = 5, a[2] = 9

B. 2, 5, 9

C. 0, 1, 2

D. undefined

The logArrayElements function makes use of the forEach method to iterate through each element of an array. Within the forEach method, the console.log statement is structured to output the index and the element in a specific format: 'a[' + index + '] = ' + element. When this function is called on an array such as [2, 5, 9], for each element in the array, the forEach function will: 1. Pass the current element to the logArrayElements function. 2. Provide the corresponding index of that element. Thus, when the first element (which is 2) is processed, it will output "a[0] = 2". For the second element (5), it will output "a[1] = 5", and for the third element (9), it will produce "a[2] = 9". Therefore, the cumulative result printed in the console will be: - a[0] = 2 - a[1] = 5 - a[2] = 9 Hence, this results in the final output format as stated in option A: a[0] = 2, a[1] = 5, a

8. What lines of code need to be added in the Student constructor to properly inherit from Person?

A. super(fName, lName, age); and Student.prototype = Object.create(Person.prototype);

B. Student.prototype = Person.call(this, fName, lName, age);

C. Person.call(this) and Student.prototype = new Person();

D. super(); and Object.create(Student.prototype);

The correct choice involves using the `super` function call to invoke the constructor of the parent class (Person) and the `Object.create()` method to set up the prototype chain properly. This allows the Student class to inherit behaviors and properties from the Person class effectively. By invoking `super(fName, lName, age);`, the constructor of Person is called with the appropriate parameters from the Student constructor, ensuring that any properties defined in Person are initialized correctly for instances of Student. This is an essential step for proper inheritance in JavaScript, especially when using class-based syntax. The use of `Student.prototype = Object.create(Person.prototype);` establishes the prototype chain such that instances of Student will have access to methods defined on Person's prototype, promoting effective inheritance of methods. In contrast, the other options do not correctly establish the inheritance relationship or do not utilize the constructor in the proper way: - Some of the options attempt to use `call` or `new` inappropriately, which can lead to instances not being set up correctly or methods not being accessible. - Using `super()` without a proper constructor call to the parent class would also not achieve the necessary setup for inheritance. Thus, the effective combination provided in the correct answer ensures proper

9. What is the return type of methods defined in Lightning Web Components?

A. Always undefined

B. Any valid JavaScript type

C. Only arrays

D. Only objects

The return type of methods defined in Lightning Web Components can indeed be any valid JavaScript type. This flexibility allows developers to create methods that can return various data structures and types, including primitives like strings or numbers, objects, arrays, or even null and undefined. In practice, this means you can craft methods that suit the specific needs of your application without being constrained to a single return type. For example, a method could return an object representing a user profile, an array containing multiple items, or even a simple boolean value based on a condition. This versatility enhances the dynamic nature of Lightning Web Components and allows for a more engaging interaction with the data, as developers can tailor the return types to meet the requirements of the component's logic and design. Understanding that methods can return any valid JavaScript type is crucial for efficiently building user interfaces in Salesforce Lightning Web Components, as it supports a wide range of coding practices and design patterns.

10. What does the valueOf() method do when used on a String object in JavaScript?

A. It converts the String object to a number

B. It returns the primitive string value

C. It creates a new string

D. It concatenates the string with another value

The valueOf() method on a String object is designed to return the primitive string value of that object. This method is useful because it allows you to retrieve the underlying primitive data type from a String object without altering the content of the string itself. In JavaScript, when you create a string using the String object (e.g., `new String("Hello")`), you create an object wrapper around the primitive string. Using valueOf() effectively unwraps this object and provides you with the string in its simplest form, which is beneficial in scenarios where you might need to perform operations that require a primitive value rather than an object reference. This primitive value can then be used in string operations, comparisons, or even concatenations without the overhead of dealing with the String object itself. Thus, understanding what valueOf() does is crucial for working effectively with strings in JavaScript, particularly when considering type coercion and the differences between objects and primitive values.