Salesforce JavaScript Developer I Certification Practice Exam (Sample)

Study Guide

Everything you need from our exam experts!

Table of Contents

Copyright **1**

Table of Contents **2**

Introduction **3**

How to Use This Guide **4**

Questions **5**

Answers **8**

Explanations **10**

Next Steps **16**

Introduction

Preparing for a certification exam can feel overwhelming, but with the right tools, it becomes an opportunity to build confidence, sharpen your skills, and move one step closer to your goals. At Examzify, we believe that effective exam preparation isn't just about memorization, it's about understanding the material, identifying knowledge gaps, and building the test-taking strategies that lead to success.

This guide was designed to help you do exactly that.

Whether you're preparing for a licensing exam, professional certification, or entry-level qualification, this book offers structured practice to reinforce key concepts. You'll find a wide range of multiple-choice questions, each followed by clear explanations to help you understand not just the right answer, but why it's correct.

The content in this guide is based on real-world exam objectives and aligned with the types of questions and topics commonly found on official tests. It's ideal for learners who want to:

- **Practice answering questions under realistic conditions,**
- **Improve accuracy and speed,**
- **Review explanations to strengthen weak areas, and**
- **Approach the exam with greater confidence.**

We recommend using this book not as a stand-alone study tool, but alongside other resources like flashcards, textbooks, or hands-on training. For best results, we recommend working through each question, reflecting on the explanation provided, and revisiting the topics that challenge you most.

Remember: successful test preparation isn't about getting every question right the first time, it's about learning from your mistakes and improving over time. Stay focused, trust the process, and know that every page you turn brings you closer to success.

Let's begin.

How to Use This Guide

This guide is designed to help you study more effectively and approach your exam with confidence. Whether you're reviewing for the first time or doing a final refresh, here’s how to get the most out of your Examzify study guide:

1. Start with a Diagnostic Review

Skim through the questions to get a sense of what you know and what you need to focus on. Your goal is to identify knowledge gaps early.

2. Study in Short, Focused Sessions

Break your study time into manageable blocks (e.g. 30 – 45 minutes). Review a handful of questions, reflect on the explanations.

3. Learn from the Explanations

After answering a question, always read the explanation, even if you got it right. It reinforces key points, corrects misunderstandings, and teaches subtle distinctions between similar answers.

4. Track Your Progress

Use bookmarks or notes (if reading digitally) to mark difficult questions. Revisit these regularly and track improvements over time.

5. Simulate the Real Exam

Once you're comfortable, try taking a full set of questions without pausing. Set a timer and simulate test-day conditions to build confidence and time management skills.

6. Repeat and Review

Don’t just study once, repetition builds retention. Re-attempt questions after a few days and revisit explanations to reinforce learning. Pair this guide with other Examzify tools like flashcards, and digital practice tests to strengthen your preparation across formats.

There’s no single right way to study, but consistent, thoughtful effort always wins. Use this guide flexibly, adapt the tips above to fit your pace and learning style. You've got this!

Questions

1. What will be displayed in the console for the statement console.log(myDt + 10);?

A. The date ten days from now

B. 10

C. Today's date

D. Error

2. What will console.log(undefined) give during the execution of checkAge({ age: 18 })?

A. You are an adult!

B. You are still an adult.

C. Hmm.. You don't have an age I guess

D. ReferenceError

3. How can you make API calls in LWC?

A. By using the `API.fetch()` method

B. Using `fetch()` in a JavaScript file

C. Through the `@api` decorator

D. Using `@wire` only

4. How do you bind a JavaScript method to a specific object?

A. By using the `apply()` method

B. By using the `bind()` method

C. By using the `call()` method

D. By using the `link()` method

5. In JavaScript, what does the method 'bind' do?

A. Executes a function immediately

B. Returns a new function with a bound context

C. Creates an object with a specific prototype

D. None of the above

6. What is the purpose of the `Promise.all()` method?

A. To execute promises in series

B. To execute multiple promises concurrently and return a single promise

C. To convert promises into callbacks

D. To limit the number of concurrent promises

7. What command is used to handle errors in JavaScript promises?

A. catch()

B. handleError()

C. fail()

D. error()

8. 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

9. How do you prevent creating a global variable when mistyping a variable name?

A. Use 'let'

B. Use 'const'

C. Use 'use strict'

D. Use 'var'

10. What will be the output of name.giveLydiaPizza() when the method is added to String.prototype?

A. "Just give Lydia pizza already!"

B. TypeError: not a function

C. SyntaxError

D. undefined

Answers

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

Explanations

1. What will be displayed in the console for the statement console.log(myDt + 10);?

A. The date ten days from now

B. 10

C. Today's date

D. Error

The statement `console.log(myDt + 10);` will lead to the result of "The date ten days from now" because of how JavaScript handles date objects and arithmetic operations. When `myDt` is a Date object in JavaScript, adding a number to it directly performs an implicit type conversion. JavaScript first converts the Date object to its numeric representation, which corresponds to the number of milliseconds since January 1, 1970, UTC. Then, it adds the specified number (in this case, 10) to this numeric value, treating the number as milliseconds. Since there are 86,400,000 milliseconds in a day (1000 milliseconds multiplied by 60 seconds multiplied by 60 minutes multiplied by 24 hours), adding 10 to the date will effectively add 10 milliseconds rather than days. However, if the context implies adding days accurately, it often depends on how the operation is interpreted. In typical practice with dates, adding days directly does not yield a specific day; it shows how date manipulation frameworks handle it. If the intent was indeed to increase the day value by a full 10 days, one would typically use a function to adjust the date accurately. If you are seeing "today's

2. What will console.log(undefined) give during the execution of checkAge({ age: 18 })?

A. You are an adult!

B. You are still an adult.

C. Hmm.. You don't have an age I guess

D. ReferenceError

In the context of the function call checkAge({ age: 18 }), when the code attempts to log `undefined`, it indicates that the function is likely trying to access a property or variable that doesn't exist within the provided object or its context. If the function is attempting to reference a property named `age`, but for some reason, it's being referenced as `undefined`, the output would communicate that the age value is not available. For instance, if there's a check within the `checkAge` function that evaluates the age property and it finds that the passed object does not provide a valid `age` property or the reference is somehow incorrect, it would lead to this scenario. The console would print the string indicating the absence of an age value, hence outputting "Hmm.. You don't have an age I guess". This explains the correct outcome of the console logging while emphasizing the potential misinterpretation of data passed to the function. This aligns with the idea that if the age property is not accessible or was not defined correctly, it would reflect as `undefined` in the console output.

3. How can you make API calls in LWC?

A. By using the `API.fetch()` method

B. Using `fetch()` in a JavaScript file

C. Through the `@api` decorator

D. Using `@wire` only

Making API calls in Lightning Web Components (LWC) primarily involves using the built-in `fetch()` function available in JavaScript. This function is a promise-based method that allows developers to send HTTP requests and handle responses effectively. It integrates well with the modern JavaScript ecosystem, making it a preferred choice for fetching data from external APIs. When using `fetch()`, developers can specify various options such as HTTP methods (GET, POST, PUT, DELETE, etc.), headers, body content for requests, and handle responses using promises. This flexibility makes it suitable for a variety of API interaction scenarios in LWC. While the decorator `@api` is used to expose public properties and methods within a component to its parent component, it does not facilitate direct API calls. The `@wire` service is used for reactive data binding but also does not directly facilitate arbitrary API calls like the `fetch()` method does. Therefore, utilizing the `fetch()` function is the most straightforward and effective way to perform API interactions in a Lightning Web Component context.

4. How do you bind a JavaScript method to a specific object?

A. By using the `apply()` method

B. By using the `bind()` method

C. By using the `call()` method

D. By using the `link()` method

Binding a JavaScript method to a specific object is effectively accomplished through the use of the `bind()` method. When you invoke `bind()` on a function, it returns a new function where the `this` keyword is set to the specified object. This means that whenever the new function is called, it will use the provided object as its context. For instance, if you have an object with properties and methods, and you want to ensure a method always references the object it belongs to, you can bind the method to that object. This is particularly useful in scenarios where the method may be passed around as a callback or used in event handlers, where it might lose its original context. The `bind()` method is versatile and can also accept additional parameters, which will be passed to the bound function whenever it is called. This capability allows for significant flexibility in managing method contexts in your JavaScript code. The other methods mentioned don't serve the purpose of permanently binding a method to a specific object in the same manner. The `apply()` and `call()` methods are used to invoke functions immediately with a specified context, rather than creating a new function. The `link()` method, on the other hand, is not related to binding functions in JavaScript

5. In JavaScript, what does the method 'bind' do?

A. Executes a function immediately

B. Returns a new function with a bound context

C. Creates an object with a specific prototype

D. None of the above

The method 'bind' in JavaScript is designed to create a new function that, when called, has its 'this' keyword set to a specific value, which is passed as the first argument to 'bind'. This is especially useful when you want to ensure that a function operates in a specific context, regardless of how or where it is invoked. When 'bind' is utilized, it does not execute the function immediately but instead returns a new function that can be executed later. The new function retains the specified context, which can be crucial for maintaining the expected behavior in scenarios such as event handling or when passing methods as callbacks. This behavior allows for greater control over the function's execution context and can prevent common pitfalls related to the 'this' keyword in JavaScript, particularly in cases where the function might be called in a different context than intended. Thus, the correct answer indicates that 'bind' creates a new function with a predefined context, making it a powerful tool for managing function execution in JavaScript.

6. What is the purpose of the `Promise.all()` method?

A. To execute promises in series

B. To execute multiple promises concurrently and return a single promise

C. To convert promises into callbacks

D. To limit the number of concurrent promises

The `Promise.all()` method is designed to handle multiple promises concurrently and returns a single promise that resolves when all of the promises in the iterable have resolved or when the iterable contains no promises. This means that when using `Promise.all()`, you can initiate multiple asynchronous operations at once, allowing them to run simultaneously rather than sequentially. The promise returned by `Promise.all()` will fulfill if all the promises it contains succeed, and it will reject with the reason of the first promise that rejects. This behavior is particularly useful for scenarios where you need to wait for several asynchronous operations to complete before proceeding, and you want to optimize performance by running them in parallel instead of executing them sequentially. For instance, if you're fetching multiple resources from an API, using `Promise.all()` allows you to send all requests at once and then handle the responses together, improving the efficiency of your application. This capability is what distinguishes `Promise.all()` from other approaches that handle promises.

7. What command is used to handle errors in JavaScript promises?

A. catch()

B. handleError()

C. fail()

D. error()

The command used to handle errors in JavaScript promises is `catch()`. This method is specifically designed to handle any errors that occur during the execution of a promise. When a promise is rejected, the `catch()` method allows you to specify a function that will run to manage that error, providing an opportunity to implement error handling logic, such as logging the error or displaying an error message to the user. The `catch()` method is a part of the promise chain and can be called after a `then()` block to capture any rejections from the previous promise in the chain. This is particularly useful for ensuring that your code can gracefully manage unexpected situations, maintaining a robust application. The other choices do not exist as standard methods for error handling in promises. Therefore, `catch()` is the only correct and relevant option to handle errors within the context of JavaScript promises.

8. 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.

9. How do you prevent creating a global variable when mistyping a variable name?

A. Use 'let'

B. Use 'const'

C. Use 'use strict'

D. Use 'var'

The correct choice, which is to use 'use strict', is a statement in JavaScript that enables strict mode. When strict mode is enabled, it helps catch common coding errors such as the accidental creation of global variables. In traditional JavaScript, if you mistype a variable name (for example, typing `myVar` instead of `myVar1`), without strict mode enabled, the interpreter will create a global variable with the incorrect name, which can lead to difficult-to-debug issues in your code. However, when you include 'use strict' at the beginning of your script or function, the code will throw an error if you try to assign a value to an undeclared variable. This behavior helps enforce better coding practices and reduces the chance of silent failures due to typos. Using 'let' and 'const' are also ways to declare variables, but they do not inherently prevent the creation of global variables from typos unless combined with strict mode. Both options impose block scope and avoid hoisting, but strict mode specifically targets the misuse of variable declarations. As for 'var', it declares variables that are function-scoped or globally scoped, which doesn't help prevent global variable issues when a variable is mistyped. By utilizing '

10. What will be the output of name.giveLydiaPizza() when the method is added to String.prototype?

A. "Just give Lydia pizza already!"

B. TypeError: not a function

C. SyntaxError

D. undefined

When the method `name.giveLydiaPizza()` is added to `String.prototype`, it means that any string object can now utilize this function. The `String.prototype` object is the core prototype for all string instances in JavaScript. By extending `String.prototype`, you essentially create a method that can be called on any string, including the string value represented by the `name` variable. If the implementation of `giveLydiaPizza` returns the string "Just give Lydia pizza already!", then calling `name.giveLydiaPizza()` will yield that exact string as output. This development provides a clear example of how prototype inheritance works in JavaScript, which allows all string instances to access the new method as part of their prototype chain. Thus, if the method is correctly defined to return that specific string, the expected output when invoking it will indeed be "Just give Lydia pizza already!"

Next Steps

Congratulations on reaching the final section of this guide. You've taken a meaningful step toward passing your certification exam and advancing your career.

As you continue preparing, remember that consistent practice, review, and self-reflection are key to success. Make time to revisit difficult topics, simulate exam conditions, and track your progress along the way.

If you need help, have suggestions, or want to share feedback, we'd love to hear from you. Reach out to our team at hello@examzify.com.

Or visit your dedicated course page for more study tools and resources:

https://salesforcejsdev1.examzify.com

We wish you the very best on your exam journey. You've got this!