SALESFORCE JAVASCRIPT DEVELOPER PRACTICE EXAM (SAMPLE)

STUDY GUIDE



SAMPLE STUDY GUIDE WITH LIMITED QUESTIONS

VISIT US ONLINE FOR THE FULL VERSION STUDY GUIDE

https://salesforce-javascriptdeveloper.examzify.com

Copyright © 2026 by Examzify - A Kaluba Technologies Inc. product.

ALL RIGHTS RESERVED.

No part of this book may be reproduced or transferred in any form or by any means, graphic, electronic, or mechanical, including photocopying, recording, web distribution, taping, or by any information storage retrieval system, without the written permission of the author.

Notice: Examzify makes every reasonable effort to obtain accurate, complete, and timely information about this product from reliable sources.

SAMPLE

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 15

SAMPLE

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.

SAMPLE

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!

SAMPLE

Questions

SAMPLE

1. What will be the output when executing: console.log(typeof typeof 1);?

A. "number"

B. "string"

C. "object"

D. "undefined"

2. What is an advantage of using Lightning Web Components?

A. It requires less code to run complex applications than traditional methods

B. It eliminates the need for JavaScript entirely in Salesforce development

C. It integrates seamlessly with all existing Apex classes

D. It improves the performance due to a lightweight structure and native browser support

3. How do you define a CSS style that only applies to a specific component in LWC?

A. By including the style in the global CSS file

B. By writing the CSS in the component's own style file, which is scoped to that component

C. By importing external stylesheets in the HTML template

D. By applying inline styles within the HTML tags directly

4. How do you handle asynchronous operations in JavaScript?

A. By using synchronous calls for predictable execution

B. By using the fetch API for HTTP requests

C. By utilizing callback functions and promises

D. By avoiding the use of functions altogether

5. How does the spread operator work in the context of the object in ES6?

A. It creates a shallow copy of an object.

B. It allows merging of two objects without modifying the original.

C. It only works with arrays.

D. It prevents reference copying of objects.

SAMPLE

6. What will be the output of the following code: const arr = [1, 2, 3]; arr.push(6); console.log(arr); arr.pop(); console.log(arr); arr.shift(); console.log(arr); arr.unshift(8); console.log(arr);?

A. [1, 2, 3, 6]

B. [2, 3, 6]

C. [8, 2, 3]

D. [8, 1, 2, 3]

7. What is the output of the expression !!null;

A. false true false

B. false false true

C. false true true

D. true true false

8. What is the output of the for-in loop iterating over the object person? const person = { name: 'Lydia', age: 21 }; for (const item in person) { console.log(item); }

A. { name: "Lydia" }, { age: 21 }

B. "name", "age"

C. "Lydia", 21

D. ["name", "Lydia"], ["age", 21]

9. In which scenario will the console.assert method provide output?

A. When the assertion is always true

B. When the assertion is false

C. It only logs without checks

D. It logs all operations

10. Which of the following values are considered falsy in JavaScript?

A. 0, '', undefined

B. 0, new Number(0), '', new Boolean(false), undefined

C. 0, '', new Boolean(false), undefined

D. All of them are falsy

SAMPLE

Answers

SAMPLE

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

SAMPLE

Explanations

SAMPLE

1. What will be the output when executing: console.log(typeof typeof 1);?

A. "number"

B. "string"

C. "object"

D. "undefined"

When executing the expression `console.log(typeof typeof 1);`, the innermost `typeof` operator evaluates the expression `1`. Since `1` is a number, the first `typeof` returns the string `"number"`. Next, the outer `typeof` operator evaluates the result of the first `typeof` operation, which is the string `"number"`. The `typeof` operator determines the type of its operand, which in this case is the string `"number"`. Since the type of a string is classified as `"string"`, the outer `typeof` returns `"string"`. Therefore, the final output of the entire expression is `"string"`, making it the correct answer. This demonstrates how the `typeof` operator behaves in nested operations and highlights its return values based on the types of the evaluated expressions.

2. What is an advantage of using Lightning Web Components?

A. It requires less code to run complex applications than traditional methods

B. It eliminates the need for JavaScript entirely in Salesforce development

C. It integrates seamlessly with all existing Apex classes

D. It improves the performance due to a lightweight structure and native browser support

The advantage of using Lightning Web Components lies in their improved performance, which is attributed to a lightweight structure and support from native browser APIs. This approach allows developers to create highly efficient and fast-loading components that leverage modern web standards. By operating closer to the browser's capabilities, Lightning Web Components can enhance the user experience through quicker rendering times and responsiveness compared to other frameworks that may rely on heavier abstractions. Furthermore, the lightweight architecture of Lightning Web Components reduces the overall footprint of the code, leading to lower memory consumption and faster loading times. This performance boost is critical in providing a smooth and efficient user interaction within Salesforce applications, making Lightning Web Components a preferred choice for developers looking to build high-performing applications in the Salesforce ecosystem. The other choices present limitations or misconceptions about the capabilities and requirements of Lightning Web Components. For instance, while it is true that Lightning Web Components can reduce the amount of code needed, saying it requires less code for complex applications compared to traditional methods is overly simplistic and does not capture the efficiency achieved through performance optimization. Eliminating the need for JavaScript entirely is inaccurate, as JavaScript remains a fundamental part of how Lightning Web Components operate. Integration with existing Apex classes is a feature of the Salesforce platform but does not directly pertain

SAMPLE

3. How do you define a CSS style that only applies to a specific component in LWC?

A. By including the style in the global CSS file

B. By writing the CSS in the component's own style file, which is scoped to that component

C. By importing external stylesheets in the HTML template

D. By applying inline styles within the HTML tags directly

Defining a CSS style that applies only to a specific component in Lightning Web Components (LWC) is effectively achieved by writing the CSS in the component's own style file. This approach is key because LWC encapsulates styles within the component, meaning that styles defined in a component's specific CSS file are scoped only to that component. This encapsulation prevents styles from inadvertently affecting other components or the entire application, maintaining a clear separation of styles and avoiding global conflicts. The isolation provided by using a dedicated style file promotes better maintainability and predictability of component behavior. Therefore, using the component's own style file, which adheres to LWC's styling rules, ensures that any CSS rules defined there will exclusively impact that particular component, preserving the integrity of the overall design and functionality of the application.

4. How do you handle asynchronous operations in JavaScript?

A. By using synchronous calls for predictable execution

B. By using the fetch API for HTTP requests

C. By utilizing callback functions and promises

D. By avoiding the use of functions altogether

Handling asynchronous operations in JavaScript is essential for managing tasks that may take an indeterminate amount of time to complete, such as fetching data from a server, reading files, or waiting for user input. The use of callback functions and promises is a key strategy in this regard. Callback functions allow developers to pass a function as an argument to be executed once an asynchronous operation completes. This method provides a straightforward way to structure code, where actions dependent on the completion of previous tasks can be executed in sequence. Promises, on the other hand, introduce a more robust way to handle asynchronous behavior. A promise represents a value that may be available now, or in the future, or never. It helps manage the asynchronous flow by allowing developers to attach success and error handlers, which improves code readability and maintainability. The promise pattern enables chaining multiple asynchronous operations together, enhancing the ability to handle complex workflows. Therefore, utilizing callback functions and promises is the correct approach, as they provide the necessary tools to work with asynchronous operations effectively while keeping the code organized and manageable.

SAMPLE

5. How does the spread operator work in the context of the object in ES6?

A. It creates a shallow copy of an object.

B. It allows merging of two objects without modifying the original.

C. It only works with arrays.

D. It prevents reference copying of objects.

The spread operator in ES6 is a versatile syntax that allows for expanding elements in arrays and properties in objects. When used in the context of objects, it effectively creates a shallow copy of the object. This means that it duplicates the properties of the original object into a new object, so they can be modified independently. For example, using the spread operator to copy an object would look like this: ```javascript let original = { a: 1, b: 2 }; let copy = { ...original }; ``` In the above code, `copy` is a new object that has the same properties as `original`. However, if any of those properties point to another object or array, the reference to that nested structure is copied over, not the structure itself. This is what constitutes a shallow copy—while the top-level properties are duplicated, the references to nested objects remain linked to the original object. This characteristic allows developers to operate on the new object without affecting the original, unless they manipulate the nested structures themselves. The other choices presented do not fully encapsulate the nuances of the spread operator's functionality. The ability to merge objects without modifying the original is indeed a result of creating a shallow copy, but it's more specific

6. What will be the output of the following code: const arr = [1, 2, 3]; arr.push(6); console.log(arr); arr.pop(); console.log(arr); arr.shift(); console.log(arr); arr.unshift(8); console.log(arr);?

A. [1, 2, 3, 6]

B. [2, 3, 6]

C. [8, 2, 3]

D. [8, 1, 2, 3]

The code provided performs a series of array manipulations using methods that modify the original array. Initially, the array `arr` is declared and initialized with the elements `[1, 2, 3]`. When the `push(6)` method is called, the value `6` is added to the end of the array, resulting in `[1, 2, 3, 6]`. This is confirmed by the first `console.log(arr)`, which outputs the updated state of the array. Next, the `pop()` method removes the last element from the array. Since the last element is `6`, after this operation, the array reverts to `[1, 2, 3]`. This is echoed in the next `console.log(arr)` statement. Subsequently, `shift()` is called, which removes the first element of the array. Now, `1` is removed, leading to the array becoming `[2, 3]`, which is displayed with the next `console.log(arr)`. Finally, the `unshift(8)` method adds `8` to the front of the array, resulting in the array transforming to `[8, 2, 3]`. This last change

7. What is the output of the expression !!null;

A. false true false

B. false false true

C. false true true

D. true true false

The expression !!null is a double negation of the value null. In JavaScript, the logical NOT operator `!` converts a value to its boolean equivalent and then negates it. When you apply the first negation `!null`, the null value is coerced to a boolean. Since null is a falsy value, `!null` evaluates to true. Then, applying the second negation with another `!`, you negate true, resulting in false. Thus, the final output of the expression !!null is false. This understanding aligns with the evaluation of its logical truthiness and the behavior of the NOT operator in JavaScript. Other choices do not accurately reflect the result of negating null twice.

SAMPLE

8. What is the output of the for-in loop iterating over the object person? const person = { name: 'Lydia', age: 21 }; for (const item in person) { console.log(item); }

A. { name: "Lydia" }, { age: 21 }

B. "name", "age"

C. "Lydia", 21

D. ["name", "Lydia"], ["age", 21]

In the given question, the for-in loop is used to iterate over the properties (keys) of the object `person`. When using the for-in statement, it retrieves the keys of the object, not the values associated with those keys. In this case, the object `person` has two keys: `name` and `age`. As the loop executes, it will first assign `item` to the first key, which is `name`, and will log this to the console. In the second iteration, `item` is assigned to `age`, and this is also logged. Thus, the output of the loop will produce exactly the keys of the object, which are the strings "name" and "age". This aligns precisely with the correct choice, which indicates these strings as the output of the loop. The other options suggest outputs that either combine keys with values or format the output differently, which does not reflect the functionality of the for-in loop in this context.

9. In which scenario will the console.assert method provide output?

A. When the assertion is always true

B. When the assertion is false

C. It only logs without checks

D. It logs all operations

The console.assert method is designed to evaluate an expression and provide output only when that expression evaluates to false. This behavior is useful for debugging because it allows developers to make assertions about their code's expected state and to receive feedback only when those expectations are not met. When the assertion (i.e., the expression passed to console.assert) evaluates to true, the method does nothing and no output is generated, which helps to keep the console clean and focused on issues that actually need attention. Thus, it is particularly beneficial during development to confirm that the program behaves as expected. This functionality makes console.assert an effective tool for debugging, as it alerts the developer to assertions that fail while ignoring those that succeed, thus streamlining the debugging process.

10. Which of the following values are considered falsy in JavaScript?

A. 0, '', undefined

B. 0, new Number(0), '', new Boolean(false), undefined

C. 0, '', new Boolean(false), undefined

D. All of them are falsy

In JavaScript, the concept of "falsy" values refers to values that evaluate to false in a boolean context. The values considered falsy include 0 (the number zero), an empty string (''), null, undefined, and NaN (Not-a-Number). The correct answer identifies that 0, '', and undefined are indeed falsy values. When examining the other choices, it's important to note that new Number(0) and new Boolean(false) are not falsy because they are objects. In JavaScript, all objects are truthy regardless of their content unless they are explicitly checked for their primitive value. While the primitive number 0 and the primitive boolean false are falsy, their object counterparts (new Number(0) and new Boolean(false)) return true in conditions since they are objects. Therefore, the correct selection showcases only the universally recognized falsy values without including any that could be considered truthy due to being objects. This understanding helps in accurately determining the behavior of different types of data in JavaScript.

SAMPLE

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://salesforce-javascriptdeveloper.examzify.com

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

SAMPLE