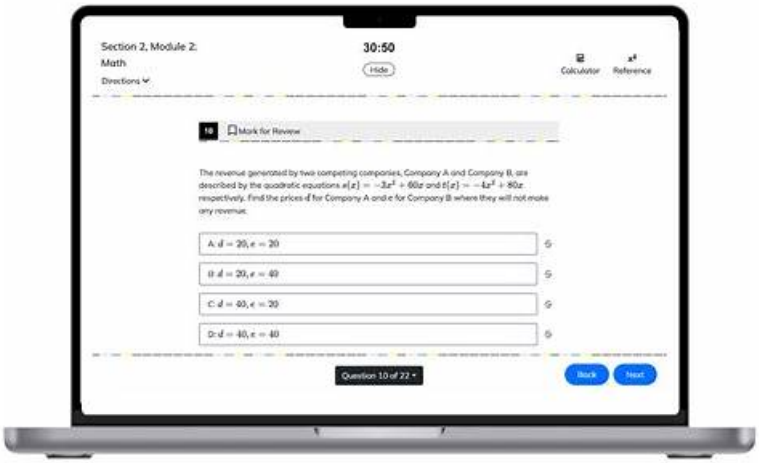
JS-Dev-101 Practice Test Online, New JS-Dev-101 Test Pass4sure



P.S. Free & New JS-Dev-101 dumps are available on Google Drive shared by PassSureExam: https://drive.google.com/open?id=1nYrdNdnHMUbHYSpPOEDZ7GE_-pDllsnQ

Constantly updated multiple mock exams with a great number of questions that will help you in better self-assessment. Memorize all your previous Salesforce Certified JavaScript Developer - Multiple Choice (JS-Dev-101) exam questions attempts and display all the changes in your results at the end of each Salesforce JS-Dev-101 Practice Exam attempt. Users will be able to customize the Salesforce Certified JavaScript Developer - Multiple Choice (JS-Dev-101) practice test software by time or question types. Supported on all Windows-based PCs.

Salesforce JS-Dev-101 Exam Syllabus Topics:

Topic	Details
Topic 1	• Browser and Events: Covers DOM manipulation, event handling and propagation, browser-specific APIs, and using Browser Developer Tools to inspect code behavior.
Topic 2	• Objects, Functions, and Classes: Covers function, object, and class implementations to meet business requirements, along with the use of modules, decorators, variable scope, and execution flow.
Topic 3	• Server Side JavaScript: Covers Node.js implementations, CLI commands, core modules, and package management solutions for given scenarios.

>> JS-Dev-101 Practice Test Online <<

Download Real Salesforce JS-Dev-101 Exam Questions And Start Your Preparation

This is a desktop-based JS-Dev-101 practice exam software that doesn't require an internet connection except for license validation during purchase. The software provides Salesforce Certified JavaScript Developer - Multiple Choice (JS-Dev-101) practice exams that are customizable, helping students prepare for the actual JS-Dev-101 Exam. The team updates the Salesforce JS-Dev-101 tests regularly and is available 24/7 to address any issues. Assessment records are saved for easy tracking. Windows computers support the desktop Salesforce JS-Dev-101 practice exam software.

Salesforce Certified JavaScript Developer - Multiple Choice Sample Questions (Q16-Q21):

NEW QUESTION # 16
Refer to the code below:
01 const myFunction = arr => {
02 return arr.reduce((result, current) => {
03 return result + current;
04 }, 10);
05 }
What is the output of this function when called with an empty array?

- A. Returns 0
- B. Returns NaN
- C. Throws an error
- D. Returns 5 ← (Text here appears to be a typo; correct value is 10, see explanation.)

Answer: D

Explanation:
We call:
myFunction([]);
Inside:
arr.reduce((result, current) => {
return result + current;
}, 10);
Key points about Array.prototype.reduce:
Signature: array.reduce(callback, initialValue)
If initialValue is provided and the array is empty, reduce:
Does not call the callback at all.
Simply returns initialValue.
Here:
arr is [] (empty).
initialValue is 10.
So:
No iterations of the callback happen (no elements to process).
The return value is the initial value: 10.
So the actual output is:
myFunction([]) === 10;
Among the options:
A: 0 - incorrect, because the initial value is 10, not 0.
B: Throws an error - reduce throws only if the array is empty and there is no initialValue. Here we have an initial value, so no error.
C: NaN - there is no arithmetic with undefined or invalid values; we just return 10.
D: Returns 5 - the numeric value given is wrong; the correct value is 10.
Given the logic, the correct conceptual result is 10. The option text "Returns 5" is almost certainly a typo for "Returns 10". Since the letter that is intended to represent the correct behavior is D, we keep:
Answe r: D
Study Guide / Concept Reference (no links):
Array.prototype.reduce behavior with initialValue
Behavior of reduce on an empty array with and without initialValue
Return value when no iterations run

NEW QUESTION # 17
A developer creates a class that represents a news story based on the requirements that a Story should have a body, author, and view count. The code is shown below:
01 class Story {
02 // Insert code here
03 this.body = body;
04 this.author = author;
05 this.viewCount = viewCount;
06 }
07 }
Which statement should be inserted in the placeholder on line 02 to allow for a variable to be set to a new instance of a Story with

the three attributes correctly populated?

- A. super(body, author, viewCount) {
- B. function Story(body, author, viewCount) {
- C. constructor(body, author, viewCount) {
- D. constructor() {

Answer: C

Explanation:
Comprehensive and Detailed Explanation From Exact Extract JavaScript knowledge:
In ES6 class syntax, the special method used to initialize a new instance is called constructor.
A class definition syntax:
class ClassName {
constructor(param1, param2) {
this.prop1 = param1;
this.prop2 = param2;
}
}
The constructor method:
Is called automatically when you create a new instance with new ClassName(...).
Receives the arguments passed in the new expression.
Assigns values to this to set instance properties.
Applying this to Story
We want to be able to write:
const article = new Story('Some body', 'Author Name', 100);
and have:
article.body === 'Some body'
article.author === 'Author Name'
article.viewCount === 100
To achieve this, the class must have:
class Story {
constructor(body, author, viewCount) {
this.body = body;
this.author = author;
this.viewCount = viewCount;
}
}
So the correct line 02 is:
constructor(body, author, viewCount) {
Why the other options are incorrect
A . constructor() {
This defines a constructor with no parameters.
The lines inside the constructor use body, author, and viewCount, which would be undefined unless they exist in an outer scope (they normally do not).
This would lead to the instance properties being set to undefined in normal usage.
B . super(body, author, viewCount) {
super(...) is used inside a constructor of a subclass to call the parent class constructor.
You cannot use super(...) { as a method definition; this is invalid syntax in a class body.
Additionally, Story as given is not shown extending any class, so super is inappropriate here.
C . function Story(body, author, viewCount) {
Inside a class definition, you do not use the function keyword to define methods.
function Story(...) here would be invalid syntax in a class body.
Even if it were allowed, the special constructor method for a class is named constructor, not the class name.
Therefore, only:
constructor(body, author, viewCount) {
correctly declares the constructor for the Story class and ensures instances created with new Story(body, author, viewCount) have all three properties populated.
Reference / Study Guide concepts (no links):
ES6 class syntax
constructor method in classes

this and instance properties in classes
Difference between class constructors and regular functions
Invalid use of super and function inside class bodies

NEW QUESTION # 18
Refer to the code below:
Let textValue = '1984';
Which code assignment shows a correct way to convert this string to an integer?

- A. Let numberValue = Integer(textValue);
- B. Let numberValue = (Number)textValue;
- C. Let numberValue = textValue.toInteger();
- D. let numberValue = Number(textValue);

Answer: D

NEW QUESTION # 19
Given a value, which two options can a developer use to detect if the value is NaN?

- A. value == NaN
- B. isNaN(value)
- C. Object.is(value, NaN)
- D. value === Number.NaN

Answer: B,C

Explanation:
Comprehensive and Detailed Explanation From Exact Extract JavaScript knowledge:
We already know NaN is special: it is not equal to itself.
Check each:
A . value === Number.NaN
Number.NaN is NaN.
NaN === NaN is always false.
This will never be true; cannot reliably detect NaN.
B . value == NaN
NaN == NaN is also always false.
Again, this never detects NaN.
C . isNaN(value)
Global isNaN converts its argument to a number and then checks if the result is NaN.
This can detect NaN, but it may also return true for non-number values that coerce to NaN, such as isNaN('foo').
Regardless, it is a standard way to detect if a value is "NaN-like" in JavaScript.
D . Object.is(value, NaN)
Object.is(NaN, NaN) returns true.
This is a strict way to detect a value that is exactly NaN (no coercion).
Therefore, among the given choices, the two viable ways to detect NaN are:
isNaN(value)
Object.is(value, NaN)

NEW QUESTION # 20
A developer publishes a new version of a package with bug fixes but no breaking changes. The old version number was 2.1.1.
What should the new package version number be based on semantic versioning?

- A. 2.2.0
- B. 2.1.2
- C. 3.1.1

- D. 2.2.1

Answer: B

Explanation:
Semantic versioning: MAJOR.MINOR.PATCH
MAJOR: incompatible API changes.
MINOR: add functionality in a backward compatible manner.
PATCH: backward compatible bug fixes.
Here:
Bug fixes only, no breaking changes → increment PATCH.
From 2.1.1 to 2.1.2.
So the correct new version is 2.1.2.

NEW QUESTION # 21
.....

How can you pass your exam and get your certificate in a short time? Our JS-Dev-101 exam torrent will be your best choice to help you achieve your aim. According to customers' needs, our product was revised by a lot of experts; the most functions of our JS-Dev-101 exam dumps are to help customers save more time, and make customers relaxed. If you choose to use our JS-Dev-101 Test Quiz, you will find it is very easy for you to pass your JS-Dev-101 exam in a short time. You just need to spend 20-30 hours on studying with our JS-Dev-101 exam questions; you will have more free time to do other things.