Back **Module 2 Quiz**
Graded Quiz • 30 min

Due May 16, 12:29 PM IST

Congratulations! You passed!

Grade received 87.50% **To pass** 80% or higher

Retake the assignment in **7h 59m**

Go to next item

Module 2 Quiz

Latest Submission Grade 87.5%

1. Which statements are true about prototypical inheritance in JavaScript? (Select all that apply.) **1 / 1 point**

- ☐ JavaScript is not object oriented, so prototypical inheritance is just a work around.
- ☑ Objects can inherit properties from other objects.

  Correct
  Prototypical inheritance is different from classical inheritance, but it can be a very powerful tool in object oriented programming.
- ☑ Objects can have their own properties and methods in addition to properties and methods inherited from other objects.

  Correct
  Prototypical inheritance is different from classical inheritance, but it can be a very powerful tool in object oriented programming.
- ☑ Objects can inherit methods from other objects.

  Correct
  Prototypical inheritance is different from classical inheritance, but it can be a very powerful tool in object oriented programming.
- ☐ You can't overwrite an object's inherited method with a different one.
- ☐ Prototypes in JavaScript are used to test your functions before putting them into production

2. Identify true statements about constructor functions in JavaScript: (Select all that apply.) **1 / 1 point**

- ☐ Constructor functions must have at least one property or method or they will throw an error.
- ☑ When you add a method to a constructor function, that method can be accessed by an object further down the prototypical chain.

  Correct
  Constructor functions are useful in object oriented programming.
- ☑ Constructor functions can start as empty function declarations

  Correct
  Constructor functions are useful in object oriented programming.
- ☑ Properties and methods can be added programmatically to constructor functions.

  Correct
  Constructor functions are useful in object oriented programming.
- ☐ Constructor functions can only include properties and not methods.
- ☐ Constructor functions must be created as function expressions.

3. In the iPhone object example, how did you delete the stocksApp method from the iPhone? **1 / 1 point**

- ○ You use the .delete() method on the parent object to delete the stocksApp.
- ○ There is no way to remove a property or method that an object has inherited through the prototypical chain.
- ● By setting the value of the stocksApp key to undefined.
- ○ You use the .remove() method on the stocksApp to delete the app.

  Correct
  Undefined is essentially the same as deleting because there is nothing to access for that key.

4. When you inspect an object and see __proto__, what is that?

0 / 1 point

- __proto__ in the inspector is a pointer to the object the current object inherited from.
- __proto__ is a constructor function used for creating prototypes.
- __proto__ is an indicator that there is an error in the prototypical chain and you should investigate.
- [✓ __proto__ is a method you can use to add a property to the parent of an object in JavaScript.]

Incorrect
Please revisit the lesson **Overriding Inheritance.**

5. In JavaScript, how does the for...in loop differ from the for...of loop?

1 / 1 point

- The for.. in loop can only be used when modifying methods inherited through the prototypical chain.
- The for.. of method is best used with objects where hasOwnProperties is false.
- [✓ The for...in loop is used to loop over each member in an object, whereas the for...of loop is used to loop over each member in an array.]
- The for.. in loop is only used when dealing with objects that inherit from a prototype. The for.. of loop can deal with local properties only.

Correct
Arrays and objects are different structures for holding data in JavaScript.

6. Identify true statements about the arrow functions. (Select all that apply.)

1 / 1 point

- Arrow functions should not be used for callback functions.
- [✓ Arrow functions provide a different syntax for writing function expressions.]

Correct
The scope of the 'this' keyword is different when used in arrow function expressions.

- Arrow functions make your code more compact, but harder to read and should be avoided.
- Arrow functions can only be used if the function has only one parameter and a return statement.
- [✓ Arrow functions can provide more compact and easier to read JavaScript when used properly.]

Correct
The scope of the 'this' keyword is different when used in arrow function expressions.

- [✓ Arrow functions can not be used in constructor functions.]

Correct
The scope of the 'this' keyword is different when used in arrow function expressions.

7. Identify true statements about the code below: (Select all that apply.)

1 / 1 point

```
const makeUpperCase = (aString) => {

    return aString.toUpperCase();

}
```

- This function expression can not be further simplified without causing confusion and errors.
- [✓ This function expression could be further simplified by the removal of the curly braces and the return keyword.]

Correct
This is an appropriate use of an arrow function expression.

- [✓ This function expression could be further simplified by the removal of the parenthesis around the parameter.]

Correct
This is an appropriate use of an arrow function expression.

- This function expression would run more efficiently as a function declaration.

8. Identify the errors in the code snippet below:

1 / 1 point

```
const fruit = ["banana", "apple", "lemon", "kiwi"];
const fruit = ['banana', 'apple', 'lemon', 'kiwi'];

const upperFruit = [];

fruit.forEach( thisFruit <= {

    upperFruit.push(thisFruit.toUpperCase());

});

console.log(upperFruit);
```

○ The toUpperCase() method will only work on the first item in the array.

○ The variable 'thisFruit' has to go inside parentheses.

○ You can't use an arrow function as a callback function for the forEach() method.

[✓ The arrow indicating the arrow function is pointing the wrong direction, and is actually the less than or equal to operator.]

✓ **Correct**
Good job spotting the error!