- LaunchSchool
  An Online School for Developers

- Menu

- Courses
- Forum (2)
- Events
- Sharing (5)
- Videos
- Resources
- Exercises
- Pages

  - **Back**

  - Our Pedagogy
  - Mastery-based Learning
  - Results & Outcomes
  - For Employers
  - Is This For Me
  - Common Questions
  - The Student Experience
  - Love
  - Core Curriculum
  - Capstone
-
- Chat Room
-
- My Account
- My Assessments
- Sign Out

-
- Courses
- Forum (2)
- Events
- Sharing (5)
- Videos
- Resources
- Exercises
- Pages
  - Our Pedagogy
  - Mastery-based Learning
  - Results & Outcomes
  - For Employers
  - Is This For Me
  - Common Questions
  - The Student Experience
  - Love
  - Core Curriculum
  - Capstone

Give us your feedback
×

Suggestions, errors or compliments on this page - let us know!

Select a category
Please don't use this Feedback form to ask questions. If you have a question about the content, use the lesson forums and one of our staff will take a look. If you have a general non-technical question, send us an email at support@launchschool.com.

Your comment Enter comment

Edit Send feedback
Preview Send feedback

Part 1: Study Guide for Test

Please be comfortable with the topics mentioned below before taking the assessment. You should be able to explain these concepts with clarity.

Specific Topics of Interest

- Objects, object factories, constructors and prototypes, OLOO, and ES6 classes
- Methods and properties; instance and static members
- Prototypal and pseudo-classical inheritance
- Encapsulation
- Polymorphism
- Collaborator objects
- Single vs multiple inheritance
- Mix-ins; mix-ins vs. inheritance
- Methods and functions; method invocation vs. function invocation
- Higher-order functions
- The global object
- ~~Method and property lookup sequence~~
- Function execution context and `this`
- Implicit and explicit execution context

- ~~Dealing with context loss~~
- ~~`call`, `apply`, and `bind`~~
- ~~`Object.assign` and `Object.create`~~
- ~~Built-in constructors like `Array`, `Object`, `String` and `Number`~~
- Reading OO code

Precision of Language

Some questions require that you explain some code or concepts with words. It's essential that you use precise vocabulary. Be sure to pinpoint the causal mechanism at work. In other words, use the right words and don't be vague.

For example, let's take the following code.

Copy Code

```
class Dog {
  constructor(name) {
    this.name = name;
  }

  sayHello() {
    console.log(`Woof! My name is ${this.name}.`)
  }
}
```

If we ask you to describe this code, you may say that:

> It defines a `Dog` class with two methods: a constructor and a method that has a message as a result.

This description isn't wrong, but it's imprecise and lacks some essential details. An answer like this may receive a score of 5/10 on a 10-point question; 50% is not a passing score.

A more precise answer says:

> This code defines a `Dog` class with two methods. The `constructor` method initializes a new `Dog` object, which it does by assigning the instance property `this.name` to the dog's name specified by the argument. The `sayHello` instance method logs a message to the console that includes the dog's name in place of `${this.name}`. The instance method `sayHello` returns `undefined`.

In programming, we must always concern ourselves with outputs, return value, and object mutations. We must use the right terms when we speak and not use vague words like "result." Furthermore, we need to be explicit about even the smallest details.

When writing answers to test questions, make sure you're as precise as possible, and that you use the proper vocabulary. Doing so will help you debug and understand more complex code later in your journey. If your definitions are imprecise, you can't use them to decompose a complicated method or program. Also, you may be unable to pass the test.

Some Specific Definitions

For the purposes of this assessment, we will use some terms in very precise ways. You should be extremely precise in the language that you use as well. Doing so will prevent misunderstandings during grading. Relying on precise language will help both you and us understand each other.

These areas are outlined below.

Variables

Unless mentioned specifically, we use the term **variable** in the broadest sense possible. On this exam, that means that all of the following should be treated as variables:

- Variables declared with `let`, `const`
- Function parameters
- Function names
- Class names

Note in particular that object property names **are not** variables.

Implicit vs. Explicit Execution Context

In earlier versions of JS120, we incorrectly stated that method invocations (e.g., `obj.foo()`) provide an **explicit execution context** for the method. On the surface, this usage makes sense since we're explicitly providing the object we want to use for the execution context. However, in practice, it's more common to say that method invocations provide an **implicit execution context** -- JavaScript determines the context by looking at the object used to call the method, and that is determined implicitly. It's a bit confusing, but you should use the term implicit execution context in this situation.

In practice, you use the `call`, `apply`, and `bind` methods to set an explicit execution context. You can also set the execution context explicitly with functions that accept an argument that defines the context for a callback function. For instance, `Array.prototype.forEach` (and several other `Array.prototype` methods) take a `thisArg` argument where you can explicitly set the context for the callback.

The Private `[[Prototype]]` Property

In earlier versions of this course, we miscapitalized the name of this property as `[[prototype]]`. The correct capitalization is `[[Prototype]]`.

Online Resources

When an online resource conflicts with Launch School materials, the Launch School materials should be used on the assessment. We can't grade assessments using information that differs from what we present, especially when that information is incorrectly changed.

This is especially true with semi-official sources like the Microsoft Developer Network (MDN). Nearly anybody can update the MDN documentation, and those updates are sometimes incorrect.

Additional Tips

This assessment has a different style than the JS101 written assessment, so you should expect several open-ended questions where you will need to explain certain OOP concepts using code examples.

Some questions near the end of the exam may take longer to answer than other questions. Be sure to allow additional time for these questions.

When writing code for an assessment question, run your code often to check it. Make sure that you have some way to run JavaScript code in your terminal or via an online REPL prepared beforehand.

Be sure to format and syntax highlight your code with Markdown when including code and other special words that need to have a distinctive look to differentiate it from the surrounding text. Formatting and syntax

highlighting your code will make it easier for us to accurately gauge your performance on the exam; if you don't, you may end up with a lower grade. In particular:

- Use single backticks to format inline code and names: write `` `class` `` to get `class`.
- Use triple backticks and a language name to format multi-line code:

  ```
  ```javascript
  console.log("Hello");
  console.log("Goodbye");
  ```
  ```

- Make sure that you use backticks (`), not apostrophes ('). On most keyboards, you can find the backtick key near the top-left corner.
- Use the Preview functionality to double-check that the output looks as you expect, especially if you use copy and paste or try to use other types of markdown formatting.

You marked this topic or exercise as completed.
Assessment Format
Part 1: Start the Test

Formatting Help

Normal Markdown & GFM

Github Flavored Markdown

---

bold

~~~~strike through~~~~

`single line of code`

Supports Emoji

Code Blocks

You can write code blocks using three backticks (```) followed by the language you want to show. For instance, if you wanted to write a Ruby code block you would write:

Copy Code

```
```ruby
class Foo
end
```
```

Copy Code

```
class Foo
end
```
~~~~

You can also leave out the language identifier, doing this will make it so your code isn't highlighted at all, and all text will be white with no coloring.

Copy Code

```
```
class Bar
end
```
```

Copy Code

```
class Bar
end
```

Highlighting

You may highlight lines within your code blocks. This is a different type of highlighting, and isn't related to how the code within your code block may be colored depending on the language identifier used.
To highlight one or more lines of code, add `# highlight` on a separate line just before the first line you wish to highlight. To end highlighting, add `# endhighlight` on a separate line immediately after the last line you wish to highlight.

Copy Code

```
```ruby
def say_hello_world
# highlight
  puts 'Hello World!!!'
# endhighlight
end
```
```

Copy Code

```
def say_hello_world
  puts 'Hello World!!!'
end
```

×