Eric Elliott Follow
Compassionate entrepreneur on a mission to end homelessness.
Jan 7, 2016 · 5 min read

Master the JavaScript Interview: What is a Closure?

> *"Master the JavaScript Interview" is a series of posts designed to prepare candidates for common questions they are likely to encounter when applying for a mid to senior-level JavaScript position. These are questions I frequently use in real interviews.*

Go JS!
@JS_Cheerleader

#JavaScript #JSTweetInterview
What is a closure?

David K. 🎹
@DavidKPiano

Replying to @JS_Cheerleader
@JS_Cheerleader a stateful function.

I'm launching the series with a question that is often my first and last question in my JavaScript interviews. Frankly, you can't get very far with JavaScript without learning about closures.

You can muck around a bit, but will you really understand how to build a serious JavaScript application? Will you really understand what is going on, or how the application works? I have my doubts. Not knowing the answer to this question is a **serious red flag**.

Not only should you know the mechanics of what a closure is, you should know why it matters, and be able to easily answer several possible use-cases for closures.

Closures are **frequently used in JavaScript** for object data privacy, in event handlers and callback functions, and in partial applications, currying, and other functional programming patterns.

I don't care if a candidate knows the word "closure" or the technical definition. I want to find out if they understand the basic mechanics. If they don't, it's usually a clear indicator that the developer does not have a lot of experience building actual JavaScript applications.

> *If you can't answer this question, you're a junior developer. I don't care how long you've been coding.*

That may sound mean, but it's not. What I mean is that most competent interviewers will ask you what a closure is, and most of the time,

getting the answer wrong will cost you the job. Or if you're lucky enough to get an offer anyway, it will cost you potentially tens of thousands of dollars per year in pay because you'll be hired as a junior instead of a senior level developer, regardless of how long you've been coding.

Be prepared for a quick follow-up: "Can you name two common uses for closures?"

What is a Closure?

A **closure** is the combination of a function bundled together (enclosed) with references to its surrounding state (the **lexical environment**). In other words, a closure gives you access to an outer function's scope from an inner function. In JavaScript, closures are created every time a function is created, at function creation time.

To use a closure, simply define a function inside another function and expose it. To expose a function, return it or pass it to another function.

The inner function will have access to the variables in the outer function scope, even after the outer function has returned.

Using Closures (Examples)

Among other things, closures are commonly used to give objects data privacy. Data privacy is an essential property that helps us program to an interface, not an implementation. This is an important concept that helps us build more robust software because implementation details are more likely to change in breaking ways than interface contracts.

> *"Program to an interface, not an implementation."*
> *Design Patterns: Elements of Reusable Object Oriented Software*

In JavaScript, closures are the primary mechanism used to enable data privacy. When you use closures for data privacy, the enclosed variables are only in scope within the containing (outer) function. You can't get at the data from an outside scope except through the object's **privileged methods**. In JavaScript, any exposed method defined within the closure scope is privileged. For example:

```
const getSecret = (secret) => {
  return {
    get: () => secret
  };
};

test('Closure for object privacy.', assert => {
  const msg = '.get() should have access to the closure.';
  const expected = 1;
  const obj = getSecret(1);

  const actual = obj.get();

  try {
    assert.ok(secret, 'This throws an error.');
```

Play with this in JSBin. *(Don't see any output? Copy and paste this HTML into the HTML pane.)*

In the example above, the `.get()` method is defined inside the scope of `getSecret()`, which gives it access to any variables from `getSecret()`, and makes it a privileged method. In this case, the parameter, `secret`.

Objects are not the only way to produce data privacy. Closures can also be used to create **stateful functions** whose return values may be influenced by their internal state, e.g.:

```
const secret = msg => () => msg;
```

```
1  // Secret - creates closures with secret messages.
2  // https://gist.github.com/ericelliott/f6a87bc41de31562d0f9
3  // https://jsbin.com/hitusu/edit?html,js,output
4
5  // secret(msg: String) => getSecret() => msg: String
6  const secret = (msg) => () => msg;
7
8  test('secret', assert => {
9    const msg = 'secret() should return a function that retur
10
11   const theSecret = 'Closures are easy.';
12   const mySecret = secret(theSecret);
13
```

Available on JSBin. *(Don't see any output? Copy and paste this HTML into the HTML pane.)*

In functional programming, closures are frequently used for partial application & currying. This requires some definitions:

Application: The process of *applying* a function to its *arguments* in order to produce a return value.

Partial Application: The process of applying a function to *some of its arguments*. The partially applied function gets returned for later use. In other words, a function that **takes a function with multiple parameters** and **returns a function with fewer parameters.** Partial application *fixes* (partially applies the function to) one or more arguments inside the returned function, and the returned function takes the remaining parameters as arguments in order to complete the function application.

Partial application takes advantage of closure scope in order to **fix** parameters. You can write a generic function that will partially apply arguments to the target function. It will have the following signature:

```
partialApply(targetFunction: Function, ...fixedArgs: Any[])
=>
  functionWithFewerParams(...remainingArgs: Any[])
```

If you need help reading the signature above, check out Rtype: Reading Function Signatures.

It will take a function that takes any number of arguments, followed by arguments we want to partially apply to the function, and returns a function that will take the remaining arguments.

An example will help. Say you have a function that adds two numbers:

```
const add = (a, b) => a + b;
```

Now you want a function that adds 10 to any number. We'll call it *`add10()`*. The result of *`add10(5)`* should be *`15`*. Our *`partialApply()`* function can make that happen:

```
const add10 = partialApply(add, 10);
add10(5);
```

In this example, the argument, *`10`* becomes a **fixed parameter** remembered inside the *`add10()`* closure scope.

Let's look at a possible *`partialApply()`* implementation:

```
// Generic Partial Application Function
// https://jsbin.com/biyupu/edit?html,js,output
// https://gist.github.com/ericelliott/f0a8fd662111ea2f569e

// partialApply(targetFunction: Function, ...fixedArgs: Any
//   functionWithFewerParams(...remainingArgs: Any[])
const partialApply = (fn, ...fixedArgs) => {
  return function (...remainingArgs) {
    return fn.apply(this, fixedArgs.concat(remainingArgs));
  };
};


test('add10', assert => {
  const msg = 'partialApply() should partially apply functi

  const add = (a, b) => a + b;
```

Available on JSBin. *(Don't see any output? Copy and paste this HTML into the HTML pane.)*

As you can see, it simply returns a function which retains access to the `*fixedArgs*` arguments that were passed into the `*partialApply()*` function.

Your Turn

This post has a companion video post and practice assignments for members of EricElliottJS.com. If you're already a member, sign in and practice now.

If you're not a member, sign up today.

Explore the Series

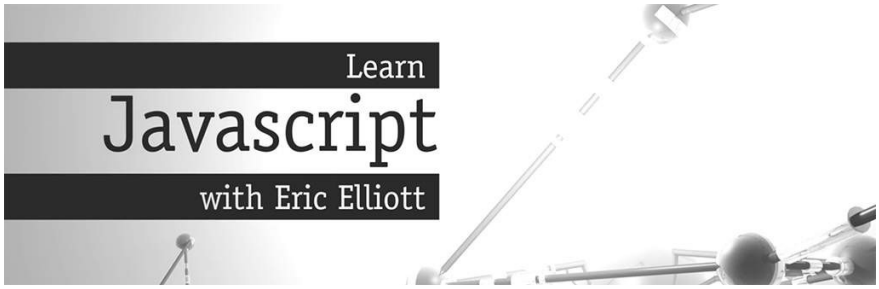
- What is a Closure?
- What is the Difference Between Class and Prototypal Inheritance?
- What is a Pure Function?
- What is Function Composition?

- What is Functional Programming?
- What is a Promise?
- Soft Skills

. . .

Learn More at EricElliottJS.com

Many more video lessons on functional programming are available for members. If you're not a member, sign up today.



. . .

Eric Elliott *is the author of "Programming JavaScript Applications" (O'Reilly), and "Learn JavaScript with Eric Elliott". He has contributed to software experiences for* ***Adobe Systems****,* ***Zumba Fitness****,* ***The Wall Street Journal****,* ***ESPN****,* ***BBC****, and top recording artists including* ***Usher****,* ***Frank Ocean****,* ***Metallica****, and many more.*

He works remote from anywhere with the most beautiful woman in the world.