Extracted from:

Rediscovering JavaScript

Master ES6, ES7, and ES8

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Rediscovering JavaScript

Master ES6, ES7, and ES8

Venkat Subramaniam

edited by Jacquelyn Carter

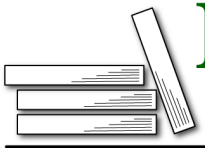
Rediscovering JavaScript

Master ES6, ES7, and ES8

Venkat Subramaniam

The Pragmatic Bookshelf

Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Managing Editor: Brian MacDonald
Supervising Editor: Jacquelyn Carter
Copy Editor: Liz Welch
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-68050-546-7
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—June 2018

CHAPTER 3

Working with Function Arguments

Calling functions is arguably one of the most frequent tasks you'll do when programming. As an author of a function, you have to decide on the parameters to receive. As a caller of a function, you have to pass the right arguments. And, from the extensibility and flexibility point of view, you may want functions to receive variable numbers of arguments. From the beginning, JavaScript is one of those few languages that has supported a variable number of arguments. But that support was very spotty—the syntax was unclear and inconsistent.

Modern JavaScript brings a breath of fresh air both for defining functions and for calling functions.

Now, when defining functions you can clearly and unambiguously convey if you intend to receive a few discrete parameters, or receive a variable number of parameters, or a mixture of both. Unlike the old `arguments`, the new rest parameter is a full-fledged `Array` object, and you can process the parameters received with greater ease; you can even use functional style code for that. And, if you choose to extend your function by adding new parameters, the default parameters makes that transition much smoother than in the past.

When calling a function, the spread operator removes the need to manually break down the values in an array into discrete parameters. That leads to less code, less noise, and more fluency. In combination with `Array`, you may also use the spread operator to combine values in arrays and discrete variables to pass arguments to functions that receive rest parameters.

In this chapter we'll quickly review the old `arguments` and how such a powerful feature is mired with issues. Then we'll see how the rest parameter replaces `arguments`, bringing all the power forward minus the perils. We'll then switch

to the function calling side and take a look at the benefits of the spread operator. Finally we'll examine default parameters and how they interplay with rest parameters.

The Power and Perils of arguments

The ability to pass a variable number of arguments to a function is a feature that's esoteric in many languages but is commonplace in JavaScript. JavaScript functions always take a variable number of arguments, even if we define named parameters in function definitions. Here's a max() function that takes two named parameters:

parameters/max.js

```
const max = function(a, b) {
  if (a > b) {
    return a;
  }

  return b;
};

console.log(max(1, 3));
console.log(max(4, 2));
console.log(max(2, 7, 1));
```

We can invoke the function with two arguments, but what if we call it with three arguments, for example? Most languages will scoff at this point, but not JavaScript. Here's the output:

```
3
4
7
```

It appears to even produce the right result when three parameters were passed—what's this sorcery?

First, we may pass as many arguments to a function as we like. If we pass fewer arguments than the number of named parameters, the extra parameters turn up as undefined. If we pass more arguments than the number of parameters, then those are merely ignored. Thus the last argument 1 was ignored in the last call to the max() method.

JavaScript has always allowed passing a variable number of arguments to functions, but receiving a variable number of parameters has been messy until recently. Traditionally, the special arguments keyword is used to process the parameters, like so:

parameters/arguments.js

```
const max = function() {
  console.log(arguments instanceof Array);

  let large = arguments[0];

  for(let i = 0; i < arguments.length; i++) {
    if(arguments[i] > large) {
      large = arguments[i];
    }
  }

  return large;
};

console.log(max(2, 1, 7, 4));
```

This version of the max() function does not have any explicitly named parameters declared. Within the function we query if arguments is an Array and then iterate over each element in that "thingy" to pick the largest value. The output from the code is shown here:

```
false
7
```

While in the past arguments has been used extensively in JavaScript, there are many issues with its use, as we see in this example:

- The method signature does not convey the intent—worse, it's misleading. While it appears that the function does not take any arguments, the actions of the implementation are quite contrary to that.
- arguments is an Array wannabe—it may be used like an Array, but only on the surface; it's largely deficient in its capabilities.
- The code is noisy and can't make use of more elegant solutions that may be used if arguments were an Array.

arguments is beyond repair since JavaScript has to preserve backward compatibility. The rest parameter solves the issues—moving forward, don't use arguments and use the rest parameter instead.

Using the Rest Parameter

A rest parameter is defined using the ellipsis (...) to signify that that parameter is a placeholder for any number of arguments. The rest parameter directly addresses the issues with arguments. First, it stands for the *rest of the parameters* and so is highly visible in the parameter list. Second, the rest parameter is of Array type. Let's convert the max() function from the previous example to use a rest parameter.

parameters/restmax.js

```
const max = function(...values) {
  console.log(values instanceof Array);

  let large = values[0];

  for(let i = 0; i < values.length; i++) {
    if(values[i] > large) {
      large = values[i];
    }
  }

  return large;
};

console.log(max(2, 1, 7, 4));
```

The two versions of max, the one that uses arguments and the one that uses a rest parameter named values, look almost identical. First, instead of an empty parameter list, we have ...values—the rest parameter name is prefixed with the ellipsis. Second, anywhere arguments appeared in the code, now there is values. At first sight, the rest parameter greatly improved the method signature and left the rest of the function mostly unaltered, except for the variable name change. Let's look at the output of this code before discussing further:

```
true
7
```

The output shows that the rest parameter is an Array. This means we can use better, more fluent, and expressive functions on the rest parameter than we could ever use on arguments. For example, we can easily change the code to the following functional style:

parameters/functionalrestmax.js

```
const max = function(...values) {
  return values.reduce((large, e) => large > e ? large : e, values[0]);
};
```

You will learn about the functional style later in this book. For now, we can appreciate how concise this code is, thanks to the fact that the rest parameter is of Array type; we can't call methods like reduce() directly on arguments.

JavaScript has some reasonable rules for the rest parameter:

- The rest parameter has to be the last formal parameter.
- There can be at most one rest parameter in a function's parameter list.
- The rest parameter contains only values that have not been given an explicit name.

Overall the rest parameter is one of the good changes to the language. It makes a very powerful feature of receiving a variable number of arguments civil and sensible from both the syntax and the semantics point of view.

The ellipsis symbol used for the rest parameter on the receiving end can also be used on the function call side; let's explore that next.