

Extracted from:

# Simplifying JavaScript

Writing Modern JavaScript with ES5, ES6, and Beyond

This PDF file contains pages extracted from *Simplifying JavaScript*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit <http://www.pragprog.com>.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printed versions; the content is otherwise identical.

Copyright © 2018 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

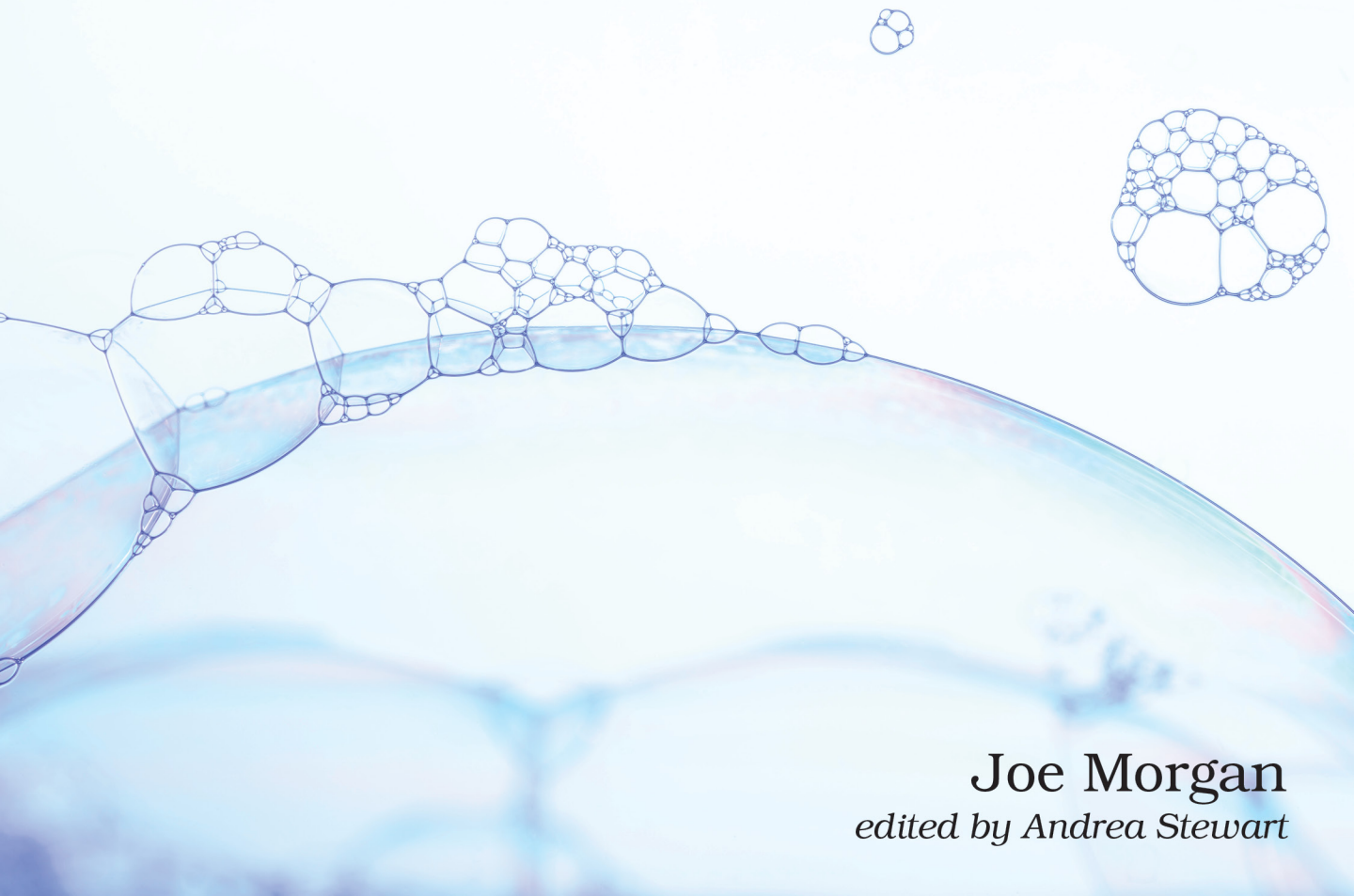
The Pragmatic Bookshelf

Raleigh, North Carolina

The  
Pragmatic  
Programmers

# Simplifying JavaScript

Writing Modern JavaScript  
with ES5, ES6, and Beyond



**Joe Morgan**  
*edited by Andrea Stewart*

# Simplifying JavaScript

Writing Modern JavaScript with ES5, ES6, and Beyond

Joe Morgan

The Pragmatic Bookshelf

Raleigh, North Carolina



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

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  
Development Editor: Andrea Stewart  
Copy Editor: Nancy Rapoport  
Indexing: Potomac Indexing, LLC  
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact [support@pragprog.com](mailto:support@pragprog.com).

For international rights, please contact [rights@pragprog.com](mailto:rights@pragprog.com).

Copyright © 2018 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.  
ISBN-13: 978-1-68050-288-6  
Encoded using the finest acid-free high-entropy binary digits.  
Book version: P1.0—April 2018

## Update Information with Object Spread

*In this tip, you'll learn how the object spread operator gives you all the advantages of `Object.assign()` with reduced syntax.*

You saw in the previous tip how you can use `Object.assign()` to make copies of objects and how you can overwrite object values with new values from another object. It's a great tool that has a lot of value. But, wow—it's ugly.

The spread operator was such a popular addition in ES6 that similar syntax is being introduced for objects. The object spread operator is not officially part of the spec, but it's so widely used that it will likely be adopted in the future. You can check out the proposal on [github](#).<sup>1</sup>

### Using Proposed Syntax

JavaScript developers love new syntax. In fact, they love it so much they often start using it before it's officially adopted. This is the case with the object spread operator and other things such as private methods in classes. You see them in many code bases even though the spec is not official.

Deciding when to use proposed syntax is a matter of preference. I tend to be pretty conservative and only adopt proposed syntax when it's very clear that the community supports the change and the proposal is unlikely to change.

Once you decide to use proposed syntax, you will need to make a few changes to your development environment. If you use Babel for compiling your code in order to be compatible across browsers, all you have to do is add a plugin and everything works fine. If you are on Node.js, it can be a little more difficult. Many features are supported using the `-harmony` flag when starting Node.js.

Feel free to experiment, but be aware you may need to refactor code if the official proposal changes.

How does the object spread operator work? Well, it's simple. It works like the array spread operator—the key-values are returned as if in a list. You can easily add information by placing it either before or after the spread. And like the array spread, you must spread it out into something.

1. <https://github.com/tc39/proposal-object-rest-spread>

```
collections/objectSpread/objectSpread.js
```

```
const book = {
  title: 'Reasons and Persons',
  author: 'Derek Parfit',
};

const update = { ...book, year: 1984 };
// { title: 'Reasons and Persons', author: 'Derek Parfit', year: 1984 }
```

But it's different from the array spread in that if you add a value with the same key, it will use whatever value is declared last.

In this way, it's like `Object.assign()` with much less typing.

```
collections/objectSpread/objectSpread.js
```

```
const book = {
  title: 'Reasons and Persons',
  author: 'Derek Parfit',
};

const update = { ...book, title: 'Reasons & Persons' };
// { title: 'Reasons & Persons', author: 'Derek Parfit' }
```

That's it! It takes the best existing features and combines them. You will not be surprised to learn that the JavaScript community embraces it enthusiastically.

Now that you have some great new syntax, try rewriting the functions from the previous tip. I'll give you the original and then the updated version. But try it yourself. It's very simple.

Here's the way to add or update information with `Object.assign()`:

```
collections/assign/assign.js
```

```
const defaults = { author: '',
  title: '',
  year: 2017,
  rating: null,
};

const book = {
  author: 'Joe Morgan',
  title: 'Simplifying JavaScript',
};

const updated = Object.assign({}, defaults, book);
```

And here it is with the object spread operator:

```
collections/objectSpread/objectSpread.js
```

```
const defaults = {
  author: '',
  title: '',
  year: 2017,
  rating: null,
};

const book = {
  author: 'Joe Morgan',
  title: 'ES6 Tips',
};

const bookWithDefaults = { ...defaults, ...book };

// {
//   author: 'Joe Morgan',
//   title: 'ES6 Tips',
//   year: 2017,
//   rating: null,
// }
```

You'll have the same deep merge problems that you have with `Object.assign()`: you don't copy nested objects—you only copy a reference creating a potential problem with mutations.

Fortunately, the fix is less painful on the eyes. Here's the original.

```
collections/assign/assign.js
```

```
const employee2 = Object.assign(
  {},
  defaultEmployee,
  {
    name: Object.assign({}, defaultEmployee.name),
  },
);

export { defaults };
```

Now, before you the look at the answer, really try this out. It's straightforward, but still a little more complex. Got it? Okay. Here's the same update with the object spread operator.

```
collections/objectSpread/objectSpread.js
```

```
const employee = {
  ...defaultEmployee,
  name: {
    ...defaultEmployee.name,
  },
};
```

The advantages are clear. The code is more readable. You're signaling your intention to create an object in a clear way. You don't have to worry about mutations because you don't need to remember to start with an empty object.

The object spread is fantastic—it's great for your code and gives you an opportunity to integrate experimental features in your code base.

That's all there is for existing collections. In the next tip, you will finally get to try out some completely new collections that should improve your code communication. First up, the Map object.