



C o m m u n i t y E x p e r i e n c e D i s t i l l e d

Learning JavaScript Data Structures and Algorithms

Understand and implement classic data structures and algorithms using JavaScript

Loiane Groner

[PACKT] open source*
PUBLISHING community experience distilled

Learning JavaScript Data Structures and Algorithms

Understand and implement classic data structures and algorithms using JavaScript

Loiane Groner



BIRMINGHAM - MUMBAI

Learning JavaScript Data Structures and Algorithms

Copyright © 2014 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: October 2014

Production reference: 1201014

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78355-487-4

www.packtpub.com

Credits

Author

Loiane Groner

Project Coordinator

Neha Thakur

Reviewers

Yaroslav Bigus
Maxwell Dayvson Da Silva
Vincent Lark
Vishal Rajpal

Proofreaders

Simran Bhogal
Ameesha Green

Indexers

Mariammal Chettiyar
Tejal Soni

Commissioning Editor

Kunal Parikh

Graphics

Ronak Dhruv
Valentina D'silva
Abhinash Sahu

Acquisition Editor

Subho Gupta

Content Development Editor

Akashdeep Kundu

Production Coordinator

Conidon Miranda

Technical Editors

Madhuri Das
Nikhil Potdukhe

Cover Work

Conidon Miranda

Copy Editors

Maria Gould
Ameesha Green
Paul Hindle

About the Author

Loiane Groner lives in São Paulo, Brazil, and has over 8 years of software development experience. While at university, she demonstrated a great interest in IT. She worked as an assistant teacher for two and a half years teaching Algorithms, Data Structures, and Computing Theory. She represented her university at the ACM International Collegiate Programming Contest – Brazilian finals (South America regionals), and also worked as student delegate of the Brazilian Computing Society (SBC) for 2 years. She won a merit award in her senior year for being one of the top three students in the Computer Science department and graduated with honors.

She has previously worked at multinational companies such as IBM. Her areas of expertise include Java SE and Java EE as well as Sencha technologies (Ext JS and Sencha Touch). Nowadays, she works as a software development manager at a financial institution, where she manages overseas solutions. She also works as an independent Sencha consultant and coach.

Loiane is also the author of *Ext JS 4 First Look*, *Mastering Ext JS*, and *Sencha Architect App Development*, all published by Packt Publishing.

She is passionate about Sencha and Java, and is the Campinas Java Users Group (CampinasJUG) leader and Espirito Santo Java Users Group (ESJUG) coordinator, both Brazilian JUGs.

Loiane also contributes to the software development community through her blogs at <http://loianegroner.com> (English) and <http://loiane.com> (Brazilian Portuguese), where she writes about her IT career, Ext JS, Sencha Touch, PhoneGap, Spring Framework, and general development notes as well as publishing screencasts.

If you want to get in touch, you can find Loiane on Facebook (<https://www.facebook.com/loianegroner>) and Twitter (@loiane).

Acknowledgments

I would like to thank my parents for giving me education, guidance, and advice over all these years and for helping me to be a better human being and professional. A very special thanks to my husband, for being patient and supportive and giving me encouragement.

I would also like to thank my professors from FAESA, who taught me about the algorithms and data structures presented in this book.

Also, a big thanks to my friends and readers for all the support. It is really nice when people get in touch at conferences and by any social network mentioning that they have read one of my books and give me feedback. Thank you very much!

About the Reviewers

Yaroslav Bigus is an expert in building cross-platform web and mobile applications. He has over 5 years of experience in development and has worked for companies in Leeds and New York. He has been using the .NET Framework stack to develop backend systems; JavaScript, AngularJS, jQuery, and Underscore for the frontend; and Xamarin for mobile devices.

He is currently working for an Israeli start-up named yRuler (Tangible). Previously, he has reviewed *Xamarin Mobile Application Development for iOS* and *iOS Development with Xamarin Cookbook*, both by Packt Publishing.

I am thankful to my friends and family for their support and love.

Maxwell Dayvson Da Silva is a native Brazilian who works as a software architect for The New York Times. He has more than 11 years of experience working for two of Brazil's leading digital media companies. His work at Terra, a global digital media company, helped reach an audience of over 100 million people monthly with entertainment, sports, and news content. Later, he was part of Globo.com, the largest media conglomerate in Latin America. His contribution in the digital media field is only a portion of how he spends his time. Combining his passion for art and science, he creates games and interactive art installations. Inspired by his son, Arthur, he continually searches for new ways to spread science in a fun way to children both in NYC and Brazil. He can be reached at <https://github.com/dayvson>.

I would like to say thank you to Juliane Inês do Nascimento for taking care of Arthur in such an amazing way while I am away. You are an amazing mother!

Vincent Lark is an experienced programmer who has worked as a backend and frontend web developer for web start-ups in Luxembourg and France. He now focuses on creating modern web UIs and develops games as a hobby. He has also reviewed the book *WebGL HOTSHOT*, Packt Publishing.

Vishal Rajpal is an experienced software engineer who started developing professional software applications in 2011. He has worked primarily on Java, JavaScript, and multiplatform mobile application development platforms including PhoneGap and Titanium.

At present, he is pursuing his Master's degree in Computer Science from Northeastern University, Seattle, and has been working on Scheme (Lisp), Objective-C, computer systems, and algorithms.

He lives in Seattle and can be reached at vishalarajpal@gmail.com. You can also find out more about his work at <https://github.com/vishalrajpal/> and <https://www.vishal-rajpal.blogspot.com>. He has also reviewed the book *PhoneGap 3.x Mobile Application Development HOTSHOT*, Packt Publishing.

www.PacktPub.com

Support files, eBooks, discount offers, and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface	1
Chapter 1: JavaScript – A Quick Overview	7
Setting up the environment	8
The browser is enough	8
Using web servers (XAMPP)	10
It's all about JavaScript (Node.js)	11
JavaScript basics	13
Variables	14
Variable scope	16
Operators	17
Truthy and falsy	19
The equals operators (== and ===)	21
Control structures	23
Conditional statements	23
Loops	25
Functions	26
Object-oriented programming	27
Debugging and tools	29
Summary	29
Chapter 2: Arrays	31
Why should we use arrays?	31
Creating and initializing arrays	32
Adding and removing elements	33
Two-dimensional and multi-dimensional arrays	37
References for JavaScript array methods	40
Joining multiple arrays	41
Iterator functions	41

Searching and sorting	43
Custom sorting	44
Sorting strings	45
Searching	46
Outputting the array into a string	46
Summary	47
Chapter 3: Stacks	49
Creating a stack	50
The complete Stack class	53
Using the Stack class	54
Decimal to binary	55
Summary	58
Chapter 4: Queues	59
Creating a queue	60
The complete Queue class	62
Using the Queue class	63
The priority queue	64
The circular queue – Hot Potato	66
Summary	68
Chapter 5: Linked Lists	69
Creating a linked list	71
Appending elements to the end of the linked list	72
Removing elements from the linked list	74
Inserting an element at any position	77
Implementing other methods	80
The toString method	80
The indexOf method	80
The isEmpty, size, and getHead methods	82
Doubly linked lists	82
Inserting a new element at any position	83
Removing elements from any position	86
Circular linked lists	89
Summary	90
Chapter 6: Sets	91
Creating a set	92
The has (value) method	93
The add method	93
The remove and clear methods	94
The size method	95
The values method	96
Using the Set class	96

Set operations	97
Set union	97
Set intersection	99
Set difference	100
Subset	102
Summary	103
Chapter 7: Dictionaries and Hashes	105
Dictionaries	105
Creating a dictionary	106
The has and set methods	107
The remove method	107
The get and values methods	108
The clear, size, keys, and getItems methods	109
Using the Dictionary class	109
The hash table	110
Creating a hash table	111
Using the HashTable class	113
Hash table versus hash set	115
Handling collisions between hash tables	115
Separate chaining	117
Linear probing	121
Creating better hash functions	124
Summary	126
Chapter 8: Trees	127
Trees terminology	128
Binary tree and binary search tree	129
Creating the BinarySearchTree class	129
Inserting a key in a tree	130
Tree traversal	134
In-order traversal	134
Pre-order traversal	136
Post-order traversal	137
Searching for values in a tree	138
Searching for minimum and maximum values	138
Searching for a specific value	140
Removing a node	142
Removing a leaf node	144
Removing a node with a left or right child	144
Removing a node with two children	145
More about binary trees	146
Summary	147

Chapter 9: Graphs	149
Graph terminology	149
Directed and undirected graphs	151
Representing a graph	152
The adjacency matrix	152
The adjacency list	153
The incidence matrix	153
Creating the Graph class	154
Graph traversals	156
Breadth-first search (BFS)	157
Finding the shortest paths using BFS	160
Further studies on the shortest paths algorithms	163
Depth-first search (DFS)	164
Exploring the DFS algorithm	167
Topological sorting using DFS	169
Summary	171
Chapter 10: Sorting and Searching Algorithms	173
Sorting algorithms	173
Bubble sort	174
Improved bubble sort	177
Selection sort	178
Insertion sort	180
Merge sort	182
Quick sort	185
The partition process	186
Quick sort in action	188
Searching algorithms	191
Sequential search	191
Binary search	192
Summary	194
Index	195

Preface

JavaScript is currently the most popular programming language. It is known as "the Internet language" due to the fact that Internet browsers understand JavaScript natively, without installing any plugins. JavaScript has grown so much that it is no longer just a frontend language; it is now also present on the server (Node.js) and database as well (MongoDB).

Understanding data structures is very important for any technology professional. Working as a developer means you have the ability to solve problems with the help of programming languages and data structures. They are an indispensable piece of the solutions we need to create to solve these problems. Choosing the wrong data structure can also impact the performance of the program we are writing. This is why it is important to get to know different data structures and how to apply them properly.

Algorithms play a major role in the art of Computer Science. There are so many ways of solving the same problem, and some approaches are better than others. That is why it is also very important to know the most famous algorithms.

Happy coding!

What this book covers

Chapter 1, JavaScript – A Quick Overview, covers the basics of JavaScript you need to know prior to learning about data structures and algorithms. It also covers setting up the development environment needed for this book.

Chapter 2, Arrays, explains how to use the most basic and most used data structure arrays. This chapter demonstrates how to declare, initialize, add, and remove elements from an array. It also covers how to use the native JavaScript array methods.

Chapter 3, Stacks, introduces the stack data structure, demonstrating how to create a stack and how to add and remove elements. It also demonstrates how to use stacks to solve some Computer Science problems.

Chapter 4, Queues, covers the queue data structure, demonstrating how to create a queue and add and remove elements. It also demonstrates how to use queues to solve some Computer Science problems and the major differences between queues and stacks.

Chapter 5, Linked Lists, explains how to create the linked list data structure from scratch using objects and the pointer concept. Besides covering how to declare, create, add, and remove elements, it also covers the various types of linked lists such as doubly linked lists and circular linked lists.

Chapter 6, Sets, introduces the set data structure and how it can be used to store non-repeated elements. It also explains the different types of set operations and how to implement and use them.

Chapter 7, Dictionaries and Hashes, explains the dictionary and hash data structures and the differences between them. This chapter covers how to declare, create, and use both data structures. It also explains how to handle collisions in hashes and techniques to create better hash functions.

Chapter 8, Trees, covers the tree data structure and its terminologies, focusing on binary search tree data as well as the methods trees use to search, traverse, add, and remove nodes. It also introduces the next steps you can take to delve deeper into the world of trees, covering what tree algorithms should be learned next.

Chapter 9, Graphs, introduces the amazing world of the graphs data structure and its application in real-world problems. This chapter covers the most common graph terminologies, the different ways of representing a graph, how to traverse graphs using the breadth-first search and depth-first search algorithms, and its applications.

Chapter 10, Sorting and Searching Algorithms, explores the most used sorting algorithms such as bubble sort (and its improved version), selection sort, insertion sort, merge sort, and quick sort. It also covers searching algorithms such as sequential and binary search.

Chapter 11, More About Algorithms, introduces some algorithm techniques and the famous big-O notation. It covers the recursion concept and some advanced algorithm techniques such as dynamic programming and greedy algorithms. This chapter introduces big-O notation and its concepts. Finally, it explains how to take your algorithm knowledge to the next level. This is an online chapter available on the Packt Publishing website. You can download it from https://www.packtpub.com/sites/default/files/downloads/48740S_Chapter11_More_About_Algorithms.pdf.

Appendix, Big-O Cheat Sheet, lists the complexities of the algorithms (using big-O notation) implemented in this book. This is also an online chapter, which can be downloaded from https://www.packtpub.com/sites/default/files/downloads/48740S_Appendix_Big_O_Cheat_Sheet.pdf.

What you need for this book

You can set up three different development environments for this book. You do not need to have all three environments; you can select one or give all of them a try!

- For the first option, you need a browser. It is recommended to use one of the following browsers:
 - Chrome (<https://www.google.com/chrome/browser/>)
 - Firefox (<https://www.mozilla.org/en-US/firefox/new/>)
- For the second option, you will need:
 - One of the browsers listed in the first option
 - A web server; if you do not have any web server installed in your computer, you can install XAMPP (<https://www.apachefriends.org>)
- The third option is an environment 100 percent JavaScript! For this, you will need the following elements:
 - One of the browsers listed in the first option
 - Node.js (<http://nodejs.org/>)
 - After installing Node.js, install `http-server` (package) as follows:

```
npm install http-server -g
```

You can find more detailed instructions in *Chapter 1, JavaScript – A Quick Overview*, as well.

Who this book is for

This book is intended for students of Computer Science, people who are just starting their career in technology, and those who want to learn about data structures and algorithms with JavaScript. Some knowledge of programming logic is the only thing you need to know to start having fun with algorithms and JavaScript!

This book is written for beginners who want to learn about data structures and algorithms, and also for those who are already familiar with data structures and algorithms but who want to learn how to use them with JavaScript.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Inside the `script` tag, we have the JavaScript code."

A block of code is set as follows:

```
console.log("num: " + num);
console.log("name: " + name);
console.log("trueValue: " + trueValue);
console.log("price: " + price);
console.log("nullVar: " + nullVar);
console.log("und: " + und);
```


When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:


```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
</head>
<body>
  <script>
    alert('Hello, World!');
  </script>
</body>
</html>
```

Any command-line input or output is written as follows:

```
npm install http-server -g
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "The number of **Node Packages Modules** (<https://www.npmjs.org/>) also grows exponentially."

[ Warnings or important notes appear in a box like this.]

[ Tips and tricks appear like this.]

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book – what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

For this book, the code files can be downloaded or forked from the following GitHub repository as well: <https://github.com/loiane/javascript-datastructures-algorithms>.

Downloading the color images of this book

We also provide you a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from https://www.packtpub.com/sites/default/files/downloads/48740S_ColoredImages.pdf.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books – maybe a mistake in the text or the code – we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

1

JavaScript – A Quick Overview

JavaScript is a very powerful language. It is the most popular language in the world and is one of the most prominent languages on the Internet. For example, GitHub (the world's largest code host, available at <https://github.com>) hosts over 400,000 JavaScript repositories (the largest number of projects is in JavaScript; refer to <http://goo.gl/ZFx6mg>). The number of projects in JavaScript in GitHub grows every year.

JavaScript is not a language that can only be used in the frontend. It can also be used in the backend as well, and Node.js is the technology responsible for this. The number of **Node Packages Modules** (<https://www.npmjs.org/>) also grows exponentially.

JavaScript is a must-have on your résumé if you are or going to become a web developer.

In this book, you are going to learn about the most used data structures and algorithms. But why use JavaScript to learn about data structures and algorithms? We have already answered this question. JavaScript is very popular, and JavaScript is appropriate to learn about data structures because it is a functional language. Also, this can be a very fun way of learning something new, as it is very different (and easier) than learning about data structures with a standard language such as C or Java. And who said data structures and algorithms were only made for languages such as C and Java? You might need to implement some of these languages while developing for the frontend as well.

Learning about data structures and algorithms is very important. The first reason is because data structures and algorithms can solve the most common problems efficiently. This will make a difference on the quality of the source code you write in the future (including performance—if you choose the incorrect data structure or algorithm depending on the scenario, you can have some performance issues). Secondly, algorithms are studied in college together with introductory concepts of Computer Science. And thirdly, if you are planning to get a job in the greatest **IT (Information Technology)** companies (such as Google, Amazon, Ebay, and so on), data structures and algorithms are subjects of interview questions.

Setting up the environment

One of the pros of the JavaScript language compared to other languages is that you do not need to install or configure a complicated environment to get started with it. Every computer has the required environment already, even though the user may never write a single line of source code. All we need is a browser!

To execute the examples in this book, it is recommended that you have Google Chrome or Firefox installed (you can use the one you like the most), an editor of your preference (such as Sublime Text), and a web server (XAMPP or any other of your preference—but this step is optional). Chrome, Firefox, Sublime Text, and XAMPP are available for Windows, Linux, and Mac OS.

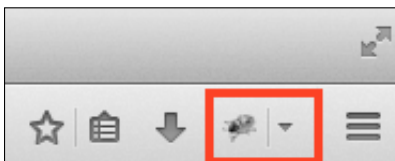
If you use Firefox, it is also recommended to install the **Firebug** add-on (<https://getfirebug.com/>).

We are going to present you with three options to set up your environment.

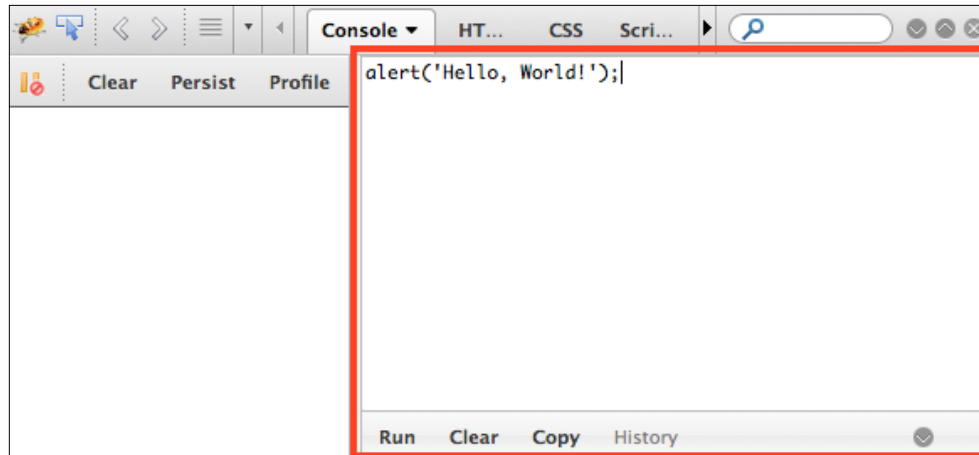
The browser is enough

The simplest environment that you can use is a browser.

You can use Firefox + Firebug. When you have Firebug installed, you will see the following icon in the upper-right corner:

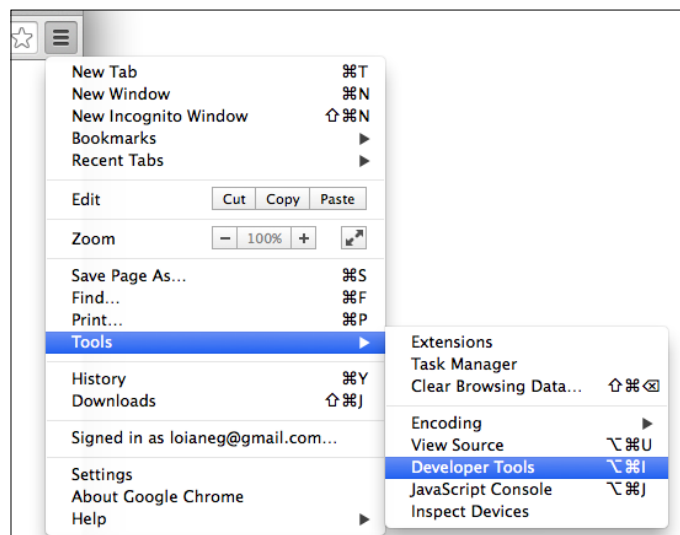


When you open Firebug (simply click on its icon), you will see the **Console** tab and you will be able to write all your JavaScript code on its command-line area as demonstrated in the following screenshot (to execute the source code you need to click on the **Run** button):

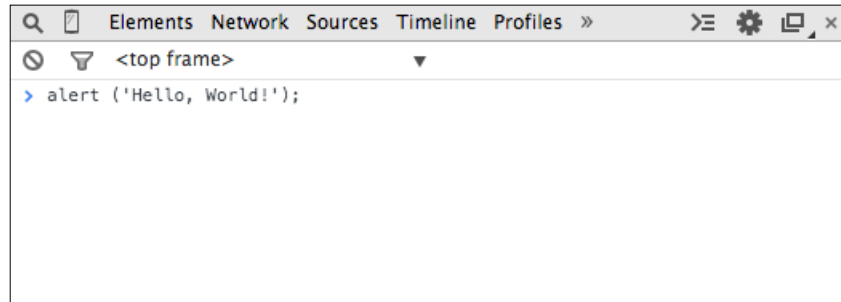


You can also expand the command line to fit the entire available area of the Firebug add-on.

You can also use Google Chrome. Chrome already comes with **Google Developer Tools**. To open it, locate the setting and control icon and navigate to **Tools | Developer Tools**, as shown in the following screenshot:



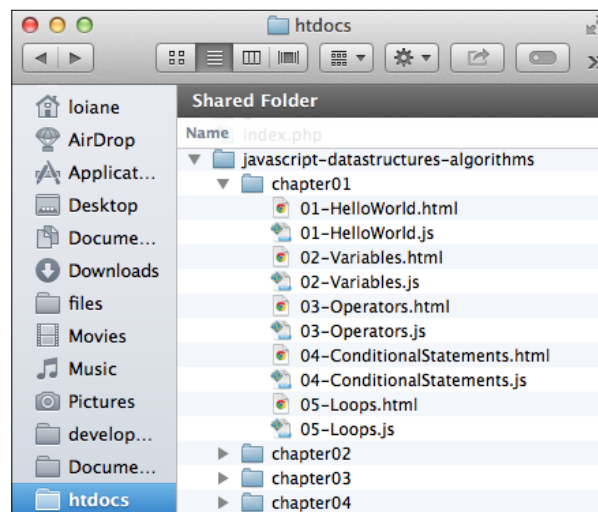
Then, in the **Console** tab, you can write your own JavaScript code for testing, as follows:



Using web servers (XAMPP)


The second environment you might want to install on your computer is also simple, but a little bit more complex than just using a browser.

You will need to install XAMPP (<https://www.apachefriends.org>) or any web server of your preference. Then, inside the XAMPP installation folder, you will find the `htdocs` directory. You can create a new folder where you can execute the source code we will implement in this book, or you can download the source code from this book and extract it to the `htdocs` directory, as follows:



Then, you can access the source code from your browser using your localhost URL (after starting the XAMPP server) as shown in the following screenshot (do not forget to enable Firebug or Google Developer Tools to see the output):



 When executing the examples, always remember to have Google Developer Tools or Firebug open to see the output.

It's all about JavaScript (Node.js)

The third option is having an environment that is 100 percent JavaScript! Instead of using XAMPP, which is an Apache server, we can use a JavaScript server.

To do so, we need to have Node.js installed. Go to <http://nodejs.org/> and download and install Node.js. After that, open the terminal application (if you are using Windows, open the command prompt with Node.js that was installed with Node.js) and run the following command:

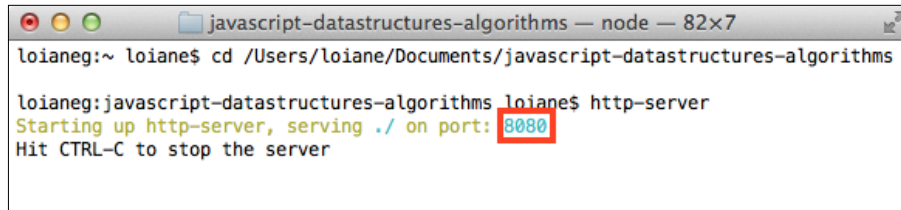
```
npm install http-server -g
```

Make sure you type the command and don't copy and paste it. Copying the command might give you some errors.

You can also execute the command as an administrator. For Linux and Mac systems, use the following command:

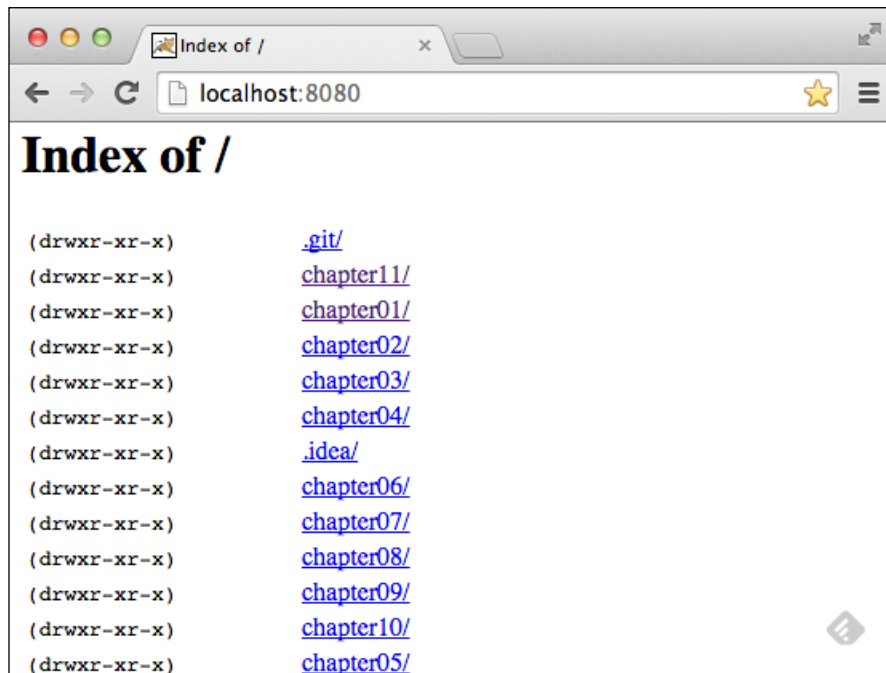
```
sudo npm install http-server -g
```

This command will install `http-server`, which is a JavaScript server. To start a server and run the examples from this book in the terminal application, change the directory to the folder that contains the book's source code and type `http-server`, as displayed in the following screenshot:



```
loianeg:~ loiane$ cd /Users/loiane/Documents/javascript-datastructures-algorithms
loianeg:javascript-datastructures-algorithms loiane$ http-server
Starting up http-server, serving ./ on port: 8080
Hit CTRL-C to stop the server
```

To execute the examples, open the browser and access the localhost on the port specified by the `http-server` command:





Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

For this book, the code files can be downloaded from this GitHub repository: <https://github.com/loiane/javascript-datastructures-algorithms>.

JavaScript basics

Before we start diving into the various data structures and algorithms, let's have a quick overview of the JavaScript language. This section will present the JavaScript basics required to implement the algorithms we will create in the subsequent chapters.

To start, let's see the two different ways we can use JavaScript code in an HTML page:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
</head>
<body>
  <script>
    alert('Hello, World!');
  </script>
</body>
</html>
```

The first way is demonstrated by the previous code. We need to create an HTML file and write this code on it. In this example, we are declaring the `script` tag inside the HTML file, and inside the `script` tag, we have the JavaScript code.

For the second example, we need to create a JavaScript file (we can save it as `01-HelloWorld.js`), and inside this file, we will insert the following code:

```
alert('Hello, World!');
```


Then, our HTML file will look like this:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
</head>
<body>
  <script src="01-HelloWorld.js">
  </script>
</body>
</html>
```

The second example demonstrates how to include a JavaScript file inside an HTML file.

By executing any of these two examples, the output will be the same. However, the second example is the best practice.



You may find JavaScript `include` statements or JavaScript code inside the `head` tag in some examples on the Internet. As a best practice, we will include any JavaScript code at the end of the `body` tag. This way, the HTML will be parsed by the browser and displayed before the scripts are loaded. This boosts the performance of the page.

Variables

Variables store data that can be set, updated, and retrieved whenever needed. Values that are assigned to a variable belong to a type. In JavaScript, the available types are **numbers**, **strings**, **Booleans**, **functions**, and **objects**. We also have **undefined** and **null**, along with **arrays**, **dates**, and **regular expressions**. The following is an example of how to use variables in JavaScript:

```
var num = 1; //{1}
num = 3; //{2}

var price = 1.5; //{3}
var name = 'Packt'; //{4}
var trueValue = true; //{5}
var nullVar = null; //{6}
var und; //{7}
```

On line {1}, we have an example of how to declare a variable in JavaScript (we are declaring a number). Although it is not necessary to use the `var` keyword declaration, it is a good practice to always specify when we are declaring a new variable.

On line {2}, we are updating an existing variable. JavaScript is not a *strongly-typed* language. This means you can declare a variable and initialize it with a number, and then update it with a string or any other data type. Assigning a value to a variable that is different from its original type is also not a good practice.

On line {3}, we are also declaring a number, but this time it is a *decimal floating point*. On line {4}, we are declaring a string; on line {5}, we are declaring a Boolean. On line {6}, we are declaring a null value, and on line {7}, we are declaring an *undefined* variable. A null value means no value and undefined means a variable that has been declared but not yet assigned a value:

```
console.log("num: " + num);
console.log("name: " + name);
console.log("trueValue: " + trueValue);
console.log("price: " + price);
console.log("nullVar: " + nullVar);
console.log("und: " + und);
```

If we want to see the value of each variable we have declared, we can use `console.log` to do so, as listed in the previous code snippet.



We have three ways of outputting values in JavaScript that we can use with the examples of this book. The first one is `alert('My text here')`, which will output an alert window on the browser; the second one is `console.log('My text here')`, which will output text on the **Console** tab of the debug tool (Google Developer Tools or Firebug, depending on the browser you are using). Finally, the third way is outputting the value directly on the HTML page that is being rendered by the browser by using `document.write('My text here')`. You can use the option that you feel most comfortable with.

The `console.log` method also accepts more than just arguments. Instead of `console.log("num: " + num)`, we can also use `console.log("num: ", num)`.

We will discuss functions and objects later in this chapter.

Variable scope

Scope refers to where in the algorithm we can access the variable (it can also be a function when we are working with function scopes). There are local and global variables.

Let's look at an example:

```
var myVariable = 'global';
myOtherVariable = 'global';

function myFunction(){
    var myVariable = 'local';
    return myVariable;
}

function myOtherFunction(){
    myOtherVariable = 'local';
    return myOtherVariable;
}

console.log(myVariable);    //{1}
console.log(myFunction()); //{2}

console.log(myOtherVariable); //{3}
console.log(myOtherFunction()); //{4}
console.log(myOtherVariable); //{5}
```

Line {1} will output `global` because we are referring to a global variable. Line {2} will output `local` because we declared the `myVariable` variable inside the `myFunction` function as a local variable, so the scope will be inside `myFunction` only.

Line {3} will output `global` because we are referencing the global variable named `myOtherVariable` that was initialized in the second line of the example. Line {4} will output `local`. Inside the `myOtherFunction` function, we are referencing the `myOtherVariable` global variable and assigning the value `local` to it because we are not declaring the variable using the `var` keyword. For this reason, line {5} will output `local` (because we changed the value of the variable inside `myOtherFunction`).

You may hear that global variables in JavaScript are evil, and this is true. Usually, the quality of JavaScript source code is measured by the number of global variables and functions (a large number is bad). So, whenever possible, try avoiding global variables.

Operators

We need operators when performing any operation in a programming language. JavaScript also has arithmetic, assignment, comparison, logical, bitwise, and unary operators, among others. Let's take a look at them:

```
var num = 0; // {1}
num = num + 2;
num = num * 3;
num = num / 2;
num++;
num--;

num += 1; // {2}
num -= 2;
num *= 3;
num /= 2;
num %= 3;

console.log('num == 1 : ' + (num == 1)); // {3}
console.log('num === 1 : ' + (num === 1));
console.log('num != 1 : ' + (num != 1));
console.log('num > 1 : ' + (num > 1));
console.log('num < 1 : ' + (num < 1));
console.log('num >= 1 : ' + (num >= 1));
console.log('num <= 1 : ' + (num <= 1));

console.log('true && false : ' + (true && false)); // {4}
console.log('true || false : ' + (true || false));
console.log('!true : ' + (!true));
```

On line {1}, we have the arithmetic operators. In the following table, we have the operators and their descriptions:

Arithmetic operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (remainder of a division operation)
++	Increment
--	Decrement

On line {2}, we have the assignment operators. In the following table, we have the operators and their descriptions:

Assignment operator	Description
=	Assignment
+=	Addition assignment (x += y) == (x = x + y)
-=	Subtraction assignment (x -= y) == (x = x - y)
*=	Multiplication assignment (x *= y) == (x = x * y)
/=	Division assignment (x /= y) == (x = x / y)
%=	Remainder assignment (x %= y) == (x = x % y)

On line {3}, we have the comparison operators. In the following table, we have the operators and their descriptions:

Comparison operator	Description
==	Equal to
===	Equal to (value and object type both)
!=	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

And on line {4}, we have the logical operators. In the following table, we have the operators and their descriptions:

Logical operator	Description
&&	And
	Or
!	Not

JavaScript also supports bitwise operators, shown as follows:

```
console.log('5 & 1:', (5 & 1));  
console.log('5 | 1:', (5 | 1));  
console.log('~ 5:', (~5));  
console.log('5 ^ 1:', (5 ^ 1));  
console.log('5 << 1:', (5 << 1));  
console.log('5 >> 1:', (5 >> 1));
```

The following table contains more detailed descriptions of the bitwise operators:

Bitwise operator	Description
&	And
	Or
~	Not
^	Xor
<<	Left shift
>>	Right shift

The `typeof` operator returns the type of the variable or expression. For example, have a look at the following code:

```
console.log('typeof num:', typeof num);
console.log('typeof Packt:', typeof 'Packt');
console.log('typeof true:', typeof true);
console.log('typeof [1,2,3]:', typeof [1,2,3]);
console.log('typeof {name:John}:', typeof {name:'John'});
```

The output will be as follows:

```
typeof num: number
typeof Packt: string
typeof true: boolean
typeof [1,2,3]: object
typeof {name:John}: object
```

JavaScript also supports the `delete` operator, which deletes a property from an object:

```
var myObj = {name: 'John', age: 21};
delete myObj.age;
console.log(myObj); //outputs Object {name: "John"}
```

In this book's algorithms, we will be using some of these operators.

Truthy and falsy

In JavaScript, `true` and `false` are a little bit tricky. In most languages, the Boolean values `true` and `false` represent the true/false results. In JavaScript, a string such as `"Packt"` has the value `true`, for example.

The following table can help us better understand how `true` and `false` work in JavaScript:

Value type	Result
undefined	false.
null	false.
Boolean	true is true and false is false!
Number	The result is false for +0, -0, or NaN; otherwise, the result is true.
String	The result is false if the string is empty (length is 0); otherwise, the result is true (length > 1).
Object	true.

Let's see some examples and verify their output:

```
function testTruthy(val){
    return val ? console.log('truthy') : console.log('falsy');
}

testTruthy(true); //true
testTruthy(false); //false
testTruthy(new Boolean(false)); //true (object is always true)

testTruthy(''); //false
testTruthy('Packt'); //true
testTruthy(new String('')); //true (object is always true)

testTruthy(1); //true
testTruthy(-1); //true
testTruthy(NaN); //false
testTruthy(new Number(NaN)); //true (object is always true)

testTruthy({}); //true (object is always true)

var obj = {name:'John'};
testTruthy(obj); //true
testTruthy(obj.name); //true
testTruthy(obj.age); //false (age does not exist)
```

The equals operators (== and ===)

The two equals operators supported by JavaScript can cause a little bit of confusion when working with them.

When using `==`, values can be considered equal even when they are of different types. This can be confusing even for a senior JavaScript developer. Let's analyze how `==` works using the following table:

Type(x)	Type(y)	Result
null	undefined	true
undefined	null	true
Number	String	<code>x == toNumber(y)</code>
String	Number	<code>toNumber(x) == y</code>
Boolean	Any	<code>toNumber(x) == y</code>
Any	Boolean	<code>x == toNumber(y)</code>
String or Number	Object	<code>x == toPrimitive(y)</code>
Object	String or Number	<code>toPrimitive(x) == y</code>

If `x` and `y` are the same type, then JavaScript will use the `equals` method to compare the two values or objects. Any other combination that is not listed in the table gives a false result.

The `toNumber` and `toPrimitive` methods are internal and evaluate the values according to the tables that follow.

The `toNumber` method is presented here:

Value type	Result
undefined	NaN.
null	+0.
Boolean	If the value is <code>true</code> , the result is 1; if the value is <code>false</code> , the result is +0.
Number	The value of the number.
String	This parses the string into a number. If the string consists of alphabetical characters, the result is NaN; if the string consists of numbers, it is transformed into a number.
Object	<code>toNumber(toPrimitive(value))</code> .

And `toPrimitive` is presented here:

Value type	Result
Object	If <code>valueOf</code> returns a primitive value, this returns the primitive value; otherwise, if <code>toString</code> returns a primitive value, this returns the primitive value; otherwise returns an error.

Let's verify the results of some examples. First, we know that the output of the following code is `true` (string length > 1):

```
console.log('packt' ? true : false);
```

Now, what about the following code? Let's see:

```
console.log('packt' == true);
```

The output is `false`! Let's understand why:

1. First, it converts the Boolean value using `toNumber`, so we have `packt == 1`.
2. Then, it converts the string value using `toNumber`. As the string consists of alphabetical characters, it returns `NaN`, so we have `NaN == 1`, which is `false`.

And what about the following code? Let's see:

```
console.log('packt' == false);
```

The output is also `false`! The following are the steps:

1. First, it converts the Boolean value using `toNumber`, so we have `packt == 0`.
2. Then, it converts the string value using `toNumber`. As the string consists of alphabetical characters, it returns `NaN`, so we have `NaN == 0`, which is `false`.

And what about the operator `===`? It is much easier. If we are comparing two values of different types, the result is always `false`. If they have the same type, they are compared according to the following table:

Type(x)	Values	Result
Number	x has the same value as y (but not NaN)	true
String	x and y are identical characters	true
Boolean	x and y are both true or both false	true
Object	x and y reference the same object	true

If x and y are different types, then the result is false.

Let's see some examples:

```
console.log('packt' === true); //false

console.log('packt' === 'packt'); //true

var person1 = {name: 'John'};
var person2 = {name: 'John'};
console.log(person1 === person2); //false, different objects
```

Control structures

JavaScript has a similar set of control structures as the C and Java languages. Conditional statements are supported by `if...else` and `switch`. Loops are supported by `while`, `do...while`, and `for` constructs.

Conditional statements

The first conditional statement we will take a look at is the `if...else` construct. There are a few ways we can use the `if...else` construct.

We can use the `if` statement if we want to execute a script only if the condition is true:

```
var num = 1;
if (num === 1) {
    console.log("num is equal to 1");
}
```

We can use the `if...else` statement if we want to execute a script if the condition is true or another script just in case the condition is false (`else`):

```
var num = 0;
if (num === 1) {
    console.log("num is equal to 1");
} else {
    console.log("num is not equal to 1, the value of num is " + num);
}
```

The `if...else` statement can also be represented by a ternary operator. For example, take a look at the following `if...else` statement:

```
if (num === 1) {  
    num--;  
} else {  
    num++;  
}
```

It can also be represented as follows:

```
(num === 1) ? num-- : num++;
```

And if we have several scripts, we can use `if...else` several times to execute different scripts based on different conditions:

```
var month = 5;  
if (month === 1) {  
    console.log("January");  
} else if (month === 2) {  
    console.log("February");  
} else if (month === 3) {  
    console.log("March");  
} else {  
    console.log("Month is not January, February or March");  
}
```

Finally, we have the `switch` statement. If the condition we are evaluating is the same as the previous one (however, it is being compared to different values), we can use the `switch` statement:

```
var month = 5;  
switch(month) {  
    case 1:  
        console.log("January");  
        break;  
    case 2:  
        console.log("February");  
        break;  
    case 3:  
        console.log("March");  
        break;  
    default:  
        console.log("Month is not January, February or March");  
}
```

One thing that is very important in a `switch` statement is the usage of `case` and `break` keywords. The `case` clause determines whether the value of `switch` is equal to the value of the `case` clause. The `break` statement stops the `switch` statement from executing the rest of the statement (otherwise, it will execute all the scripts from all `case` clauses below the matched case until a `break` statement is found in one of the `case` clauses). And finally, we have the `default` statement, which is executed by default if none of the case statements are `true` (or if the executed case statement does not have the `break` statement).

Loops

Loops are very often used when we work with arrays (which is the subject of the next chapter). Specifically, we will be using the `for` loop in our algorithms.

The `for` loop is exactly the same as in C and Java. It consists of a loop counter that is usually assigned a numeric value, then the variable is compared against another value (the script inside the `for` loop is executed while this condition is true), and then the numeric value is increased or decreased.

In the following example, we have a `for` loop. It outputs the value of `i` on the console while `i` is less than 10; `i` is initiated with 0, so the following code will output the values 0 to 9:

```
for (var i=0; i<10; i++) {  
    console.log(i);  
}
```

The next loop construct we will look at is the `while` loop. The script inside the `while` loop is executed while the condition is true. In the following code, we have a variable, `i`, initiated with the value 0, and we want the value of `i` to be outputted while `i` is less than 10 (or less than or equal to 9). The output will be the values from 0 to 9:

```
var i = 0;  
while (i<10)  
{  
    console.log(i);  
    i++;  
}
```

The `do...while` loop is very similar to the `while` loop. The only difference is that in the `while` loop, the condition is evaluated before executing the script, and in the `do...while` loop, the condition is evaluated after the script is executed. The `do...while` loop ensures that the script is executed at least once. The following code also outputs the values 0 to 9:

```
var i = 0;
do {
    console.log(i);
    i++;
} while (i<10)
```

Functions

Functions are very important when working with JavaScript. We will also use functions a lot in our examples.

The following code demonstrates the basic syntax of a function. It does not have *arguments* or the `return` statement:

```
function sayHello() {
    console.log('Hello!');
}
```

To call this code, we simply use the following call:

```
sayHello();
```

We can also pass arguments to a function. **Arguments** are variables with which a function is supposed to do something. The following code demonstrates how to use arguments with functions:

```
function output(text) {
    console.log(text);
}
```

To use this function, we can use the following code:

```
output('Hello!');
```

You can use as many arguments as you like, as follows:

```
output('Hello!', 'Other text');
```

In this case, only the first argument is used by the function and the second one is ignored.

A function can also return a value, as follows:

```
function sum(num1, num2) {  
    return num1 + num2;  
}
```

This function calculates the sum of two given numbers and returns its result. We can use it as follows:

```
var result = sum(1,2);  
output(result);
```

Object-oriented programming

JavaScript objects are very simple collections of name-value pairs. There are two ways of creating a simple object in JavaScript. The first way is as follows:

```
var obj = new Object();
```

And the second way is as follows:

```
var obj = {};
```

We can also create an object entirely as follows:

```
obj = {  
    name: {  
        first: 'Gandalf',  
        last: 'the Grey'  
    },  
    address: 'Middle Earth'  
};
```

In **object-oriented programming (OOP)**, an object is an instance of a class. A class defines the characteristics of the object. For our algorithms and data structures, we will create some classes that will represent them. This is how we can declare a class that represents a book:

```
function Book(title, pages, isbn){  
    this.title = title;  
    this.pages = pages;  
    this.isbn = isbn;  
}
```

To instantiate this class, we can use the following code:

```
var book = new Book('title', 'pag', 'isbn');
```


Then, we can access its attributes and update them as follows:

```
console.log(book.title); //outputs the book title
book.title = 'new title'; //updates the value of the book title
console.log(book.title); //outputs the updated value
```

A class can also contain functions. We can declare and use a function as the following code demonstrates:

```
Book.prototype.printTitle = function() {
    console.log(this.title);
};
book.printTitle();
```

We can declare functions directly inside the class definition as well:

```
function Book(title, pages, isbn) {
    this.title = title;
    this.pages = pages;
    this.isbn = isbn;
    this.printIsbn = function() {
        console.log(this.isbn);
    }
}
book.printIsbn();
```



In the prototype example, the `printTitle` function is going to be shared between all instances, and only one copy is going to be created. When we use class-based definition, as in the previous example, each instance will have its own copy of the functions. Using the prototype method saves memory and processing cost in regards to assigning the functions to the instance. However, you can only declare public functions and properties using the prototype method. With a class-based definition, you can declare private functions and properties and the other methods inside the class can also access them. You will notice in the examples of this book that we use a class-based definition (because we want to keep some properties and functions private). But, whenever possible, we should use the prototype method.

Now we have covered all the basic JavaScript concepts that are needed for us to start having some fun with data structures and algorithms!

Debugging and tools

Knowing how to program with JavaScript is important, but so is knowing how to debug your code. Debugging is very useful to help find bugs in your code, but it can also help you execute your code at a lower speed so you can see everything that is happening (the stack of methods called, variable assignment, and so on). It is highly recommended that you spend some time debugging the source code of this book to see every step of the algorithm (it might help you understand it better as well).

Both Firefox and Chrome support debugging. A great tutorial from Google that shows you how to use Google Developer Tools to debug JavaScript can be found at <https://developer.chrome.com/devtools/docs/javascript-debugging>.

You can use any text editor of your preference. But there are other great tools that can help you be more productive when working with JavaScript as well:

- **Aptana:** This is a free and open source IDE that supports JavaScript, CSS3, and HTML5, among other languages (<http://www.aptana.com/>).
- **WebStorm:** This is a very powerful JavaScript IDE with support for the latest web technologies and frameworks. It is a paid IDE, but you can download a 30-day trial version (<http://www.jetbrains.com/webstorm/>).
- **Sublime Text:** This is a lightweight text editor, and you can customize it by installing plugins. You can buy the license to support the development team, but you can also use it for free (the trial version does not expire) at <http://www.sublimetext.com/>.

Summary

In this chapter, we learned how to set up the development environment to be able to create or execute the examples in this book.

We also covered the basics of the JavaScript language that are needed prior to getting started with constructing the algorithms and data structures covered in this book.

In the next chapter, we will look at our first data structure, which is array, the most basic data structure that many languages support natively, including JavaScript.

