MANNING PUBLICATIONS

CoffeeScript in Action
By Patrick Lee

CoffeeScript is JavaScript with non-essential syntax removed to give more simplicity and clarity to your code. In this article, based on chapter 2 of CoffeeScript in Action, author Patrick Lee shows you the basic CoffeeScript syntax and how it compares to JavaScript.

You may also be interested in…

Welcome to CoffeeScript

If you were travelling to a foreign country and you didn't speak the language you would probably spend a little bit of time before you went learning the basics of the language, perhaps by listening to something like a *Learn Esperanto while you sleep* podcast. You wouldn't learn the whole language that way but you'd be exposed to enough of it so that when you arrived in the new country you could muster up enough intelligible language output to order your dinner. It's the same with CoffeeScript. The first thing to start with is some exploration of the basic syntactic structure—just enough to set the groundwork for further explorations in CoffeeScript land.

CoffeeScript is JavaScript with non-essential syntax removed to give more simplicity and clarity to your code. Whether you are a JavaScript veteran or coming to CoffeeScript with very little experience, the best way to learn the syntax is by seeing it and by using it. Over time, your familiarity and understanding of the concepts behind the syntax would deepen.

In this article, you will learn the basic CoffeeScript syntax and how it compares to JavaScript. To demonstrate CoffeeScript programs some language features of JavaScript must be explained in terms of the new CoffeeScript syntax. Similarly, in order to learn a new programming language, you need to write programs. In the spirit of getting there quickly, you are here presented with your first CoffeeScript program followed by a comparison between CoffeeScript and JavaScript. The parts of the language that are discussed in this article teach enough for you to be comfortable with this program while also providing a broader general coverage of the syntax that will be essential for more complex scenarios.

Compiled

CoffeeScript programs are executed as JavaScript programs. They are compiled from the CoffeeScript source that you write to an equivalent JavaScript program before they are executed.

Did you know that in Italy some baristas would not serve you coffee with milk after midday? Here, in listing 1, is such a program for your quick sip of CoffeeScript.

Listing 1 No milk after midday

```
house_roast = null

has_milk = (style) ->                #1
  switch style                       #1
    when "latte", "cappucino"        #1
      yes                            #1
    else                             #1
      no                             #1
```

```
make_coffee = (requested_style) ->              #2
  style = requested_style || 'Espresso'         #2
  if house_roast?                               #2
    "#{house_roast} #{style}"                   #2
  else                                          #2
    style                                       #2

barista = (style) ->                               #3
  time = (new Date()).getHours()                   #3
  if has_milk(style) and time > 12 then "No!"      #3
  else                                             #3
    coffee = make_coffee style                     #3
    "Enjoy your #{coffee}!"                        #3
```

#1 A function to determine if a style of coffee has milk
#2 A function to 'make' the coffee, returns a string
#3 A function that coffee is requested from

This program has an equivalent JavaScript program. By looking at the equivalent JavaScript program, you can begin to see what CoffeeScript changes about JavaScript—what it adds and what it removes. More importantly, though, you can begin to see *why* CoffeeScript makes those changes and what that means for how you write programs.

What CoffeeScript removes

Claude Debussy is quoted as saying that "music is the space between the notes." CoffeeScript syntax is defined as much by what is missing as what is present. Part of the thought experiment behind CoffeeScript is to take patterns written frequently in JavaScript, look at them and ask, "How much of this is necessary?"

Comparing CoffeeScript to JavaScript

In order to explain some of the syntactic differences between CoffeeScript and JavaScript it is worth looking at an example that demonstrates the key differences. Here is a reduced version of the program about coffee serving with the CoffeeScript program side by side with a JavaScript version:

Listing 2 Comparing CoffeeScript to JavaScript

CoffeeScript

```
has_milk = (style) ->
  switch style
    when "latte", "cappucino"
      yes
    else
      no


make_coffee = ->
  style || 'Espresso'


barista = (style) ->                #2
  now = new Date()                  #3
  time = now.getHours()             #3
  if has_milk(style) and time > 12  #3
    "No!"                           #4
  else                              #3
    coffee = make_coffee style      #5
    "Enjoy your #{coffee}!"         #5
```

JavaScript

```
var has_milk = function (style) {    #1
  switch (style) {
    case "latte":
    case "cappucino":
      return true;
    default:
   return false;
  }
};

var make_coffee = function (style) {
  return style || 'Espresso';
};

var barista = function (style) {
  var now = new Date();
  var time = now.getHours();
  var coffee;
  if (has_milk(style) && time > 12) {
    return "No";
  } else {
    coffee = make_coffee(style);
    return "Enjoy your "+coffee+"!";
  }
};
```

```
barista "latte" #6                      barista("latte");
```

#1 JavaScript requires var declaration
#2 Top level, not indented
#3 Indentation inside function
#4 Indentation inside if
#5 Indentation inside else
#6 Will return either "No!" or "Enjoy your latte", depending on the time of day

Compare the syntax for the two versions. The CoffeeScript version is missing `var` statements, semicolons, `return` statements, and curly braces.

var statements

In JavaScript, unless you're in strict mode it is possible to accidentally assign variables on the global object, in the global scope. If you don't know what that means already, for now just know it's a bad thing. CoffeeScript always defines new variables in the current scope, protecting you from this unfortunate feature in JavaScript.

Semicolons

In CoffeeScript there are no semicolons[1]. While they are allowed you never need them and they should not be used.

Return statements

The `return` keyword is absent from CoffeeScript. In CoffeeScript, the last expression in the function is returned without any `return` statement. This is called an *implicit return.* Explicit `return` statements are rarely used in CoffeeScript.

Indentation over curly braces

In CoffeeScript there are no curly braces used to mark out blocks of code. In order to remove the need for curly braces and other block delimiting character, newlines and the indentation level for each line of code is meaningful. This is called *significant whitespace* and sometimes referred to as the offside rule. It will already be familiar to *Python* and *F#* programmers and anybody who has used *HAML* or *SASS*.

You can see in listing 2 that indentation in CoffeeScript is significant. You must indent for each level of function or block of code (such as inside an `if`) you are inside. To get freedom from braces, you have to make sure to use consistent indentation. Using 2, 3 or, 4 spaces is generally acceptable. Do not use tabs.

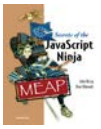
Summary

CoffeeScript provides a clean syntax over JavaScript. JavaScript is a flexible language with a small number of powerful constructs for creating abstractions—a flexible object system and first-class functions.

[1] Sure, ECMA-262 says that JavaScript parsers should do automatic semicolon insertion, but the potential for errors in JavaScript has resulted in frequent advice to always use them.

Here are some other Manning titles you might be interested in:

Ext JS in Action, Second Edition
Jesus Garcia and Jacob K. Andresen



Secrets of the JavaScript Ninja
John Resig and Bear Bibeault

Third-Party JavaScript
Ben Vinegar and Anton Kovalyov

Last updated: December 22, 2011