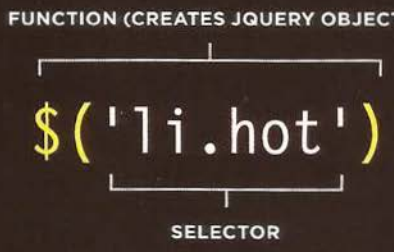
WHAT IS JQUERY?

jQuery is a JavaScript file that you include in your web pages. It lets you find elements using CSS-style selectors and then do something with the elements using jQuery methods.

1: FIND ELEMENTS USING CSS-STYLE SELECTORS

A function called **jQuery()** lets you find one or more elements in the page. It creates an object called **jQuery** which holds references to those elements. **$()** is often used as a shorthand to save typing **jQuery()**, as shown here.



The **jQuery()** function has one parameter: a CSS-style **selector**. This selector finds all of the **<li>** elements with a **class** of **hot**.

SIMILARITIES TO DOM

- jQuery selectors perform a similar task to traditional DOM queries, but the syntax is much simpler.
- You can store the **jQuery** object in a variable, just as you can with DOM nodes.
- You can use jQuery methods and properties (like DOM methods and properties) to manipulate the DOM nodes that you select.

The jQuery object has many methods that you can use to work with the elements you select. The methods represent tasks that you commonly need to perform with elements.

2: DO SOMETHING WITH THE ELEMENTS USING JQUERY METHODS

Here a **jQuery** object is created by the **jQuery()** function. The object and the elements it contains is referred to as a **matched set** or a **jQuery selection**.

You can then use the methods of the **jQuery** object to update the elements that it contains. Here, the method adds a new value to the **class** attribute.

The member operator indicates that the method on the right should be used to update the elements in the **jQuery** object on the left.

Each method has parameter(s) that provide details about how to update the elements. This parameter specifies a value to add to the **class** attribute.

KEY DIFFERENCES FROM DOM

- It's cross-browser, and there's no need to write fallback code.
- Selecting elements is simpler (because it uses CSS-style syntax) and is more accurate.
- Event handling is simpler as it uses one method that works in all major browsers.
- Methods affect all the selected elements without the need to loop through each one (see p310).
- Additional methods are provided for popular required tasks such as animation (see p332).
- Once you have made a selection, you can apply multiple methods to it.

A BASIC JQUERY EXAMPLE

The examples in this chapter revisit the list application used in the previous two chapters, and they will use jQuery to update the content of the page.

1. In order to use jQuery, the first thing you need to do is include the jQuery script in your page. You can see that it is included before the closing `</body>` tag.

2. Once jQuery has been added to the page, a second JavaScript file is included that uses jQuery selectors and methods to update the content of the HTML page.

c07/basic-example.html HTML

```
<body>
  <div id="page"
    <h1 id="header">List</h1>
    <h2>Buy groceries</h2>
    <ul>
      <li id="one" class="hot"><em>fresh</em> figs</li>
      <li id="two" class="hot">pine nuts</li>
      <li id="three" class="hot">honey</li>
      <li id="four">balsamic vinegar</li>
    </ul>
  </div>
① <script src="js/jquery-1.11.0.js"></script>
② <script src="js/basic-example.js"></script>
</body>
```

WHERE TO GET JQUERY AND WHICH VERSION TO USE

Above, jQuery is included before the closing `</body>` tag just like other scripts. (Another way to include the script is shown on p355.) A copy of jQuery is included with the code for this book, or you can download it from `http://jquery.org`. The version number of jQuery should be kept in the file name. Here, it is `jquery-1.11.0.js`, but by the time you read this book, there may be a newer version. The examples should still work with newer versions.

You often see websites use a version of the jQuery file with the file extension `.min.js`. It means unnecessary spaces and carriage returns have been stripped from the file. e.g., `jquery-1.11.0.js` becomes `jquery-1.11.0.min.js`.

It is done using a process called **minification** (hence `min` is used in the file name). The result is a much smaller file which makes it faster to download. But minified files are much harder to read.

If you want to look at the jQuery file, you can open it with a text editor – it is just text like JavaScript, albeit very complicated JavaScript.

Most people who use jQuery do not try to understand how the jQuery JavaScript file achieves what it does. As long as you know how to select elements and how to use its methods and properties, you can reap the benefits of using jQuery without looking under the hood.

Here, the JavaScript file uses the `$()` shortcut for the `jQuery()` function. It selects elements and creates three `jQuery` objects that hold references to the elements.

The methods of the `jQuery` object fade the list items in, and remove them when they are clicked on. Don't worry if you don't understand the code yet.

First, you will learn how to *select* elements using jQuery selectors, and then how to *update* those elements using the methods and properties of the `jQuery` object.

JAVASCRIPT c07/js/basic-example.js

```
① $(':header').addClass('headline');
② $('li:lt(3)').hide().fadeIn(1500);
③ $('li').on('click', function() {
      $(this).remove();
  });
```

1. The first line selects all of the `<h1>` - `<h6>` headings, and adds a value of `headline` to their `class` attributes.

2. The second line selects the first three list items and does two things:

- The elements are hidden (in order to allow the next step).
- The elements fade into view.

3. The last three lines of the script set an event listener on each of the `<li>` elements. When a user clicks on one, it triggers an anonymous function to remove that element from the page.

RESULT

Here is a reminder of the colors used to convey the priority and status of each list item:

HOT COOL
NORMAL COMPLETE

WHY USE JQUERY?

jQuery doesn't do anything you cannot achieve with pure JavaScript. It is just a JavaScript file but estimates show it has been used on over a quarter of the sites on the web, because it makes coding simpler.

1: SIMPLE SELECTORS

As you saw in Chapter 5, which introduced the DOM, it is not always easy to select the elements that you want to. For example:

- Older browsers do not support the latest methods for selecting elements.
- IE does not treat whitespace between elements as text nodes, while other browsers do.

Such issues make it hard to select the right elements on a page across all major browsers.

Rather than learn a new way to select elements, jQuery uses a language that is already familiar to front-end web developers: CSS selectors. They:

- Are much faster at selecting elements
- Can be a lot more accurate about which elements to select
- Often require a lot less code than older DOM methods
- Are already used by most front-end developers

jQuery even adds some extra CSS-style selectors which offer additional functionality.

Since jQuery was created, modern browsers have implemented the `querySelector()` and `querySelectorAll()` methods to let developers select elements using CSS syntax. However, these methods are not supported in older browsers.

2: COMMON TASKS IN LESS CODE

There are some tasks that front-end developers need to do regularly, such as loop through the elements that have been selected.

jQuery has methods that offer web developers simpler ways to perform common tasks, such as:

- Loop through elements
- Add / remove elements from the DOM tree
- Handle events
- Fade elements into / out of view
- Handle Ajax requests

jQuery simplifies each of these tasks, and allows you to write less code to achieve them.

jQuery also offers chaining of methods (a technique which you will meet on p311). Once you have selected some elements, this allows you to apply multiple methods to the same selection.

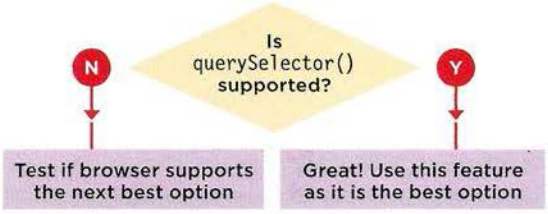
jQuery's motto is "Write less, do more," because it allows you to achieve the same goals but in fewer lines of code than you would need to write with plain JavaScript.

3: CROSS-BROWSER COMPATIBILITY

jQuery automatically handles the inconsistent ways in which browsers select elements and handle events, so you do not need to write cross-browser fallback code (such as that shown in the previous two chapters).

To do this, jQuery uses **feature detection** to find the best way to achieve a task. It involves the use of many conditional statements: if the browser supports the ideal way to achieve a task, it uses that approach; otherwise, it tests to see if it supports the next best option to achieve the same task.

This was the technique used in the last chapter to determine whether or not the browser supported event listeners. If event listeners were not supported, an alternative approach was offered (aimed at users of Internet Explorer 8 and older versions of IE).



Here, a conditional statement checks if the browser supports `querySelector()`. If it does, that method is used. If it doesn't, it checks to see if the next best option is supported and uses that instead.

JQUERY 1.9.X+ OR 2.0.X+

As jQuery developed, it built up a lot of code to support IE6, 7, and 8; which made the script bigger and more complicated. As version 2.0 of jQuery was approaching, the development team decided to create a version that would drop support for older browsers in order to create a smaller, faster script.

The jQuery team was, however, aware that many people on the web still used these older browsers, and that developers therefore needed to support them. For this reason, they now maintain two parallel versions of jQuery:

jQuery 1.9+: Encompasses the same features as 2.0.x but still offers support for IE6, 7, and 8

jQuery 2.0+: Drops support for older browsers to make the script smaller and faster to use

The functionality of both versions is not expected to diverge significantly in the short term.

The jQuery file name should contain the version number in it (e.g., `jquery-1.11.0.js` or `jquery-1.11.0.min.js`). If you don't do this, a user's browser might try to use a cached version of the file that is either older *or* newer – which could prevent other scripts from working correctly.

FINDING ELEMENTS

Using jQuery, you usually select elements using CSS-style selectors. It also offers some extra selectors, noted below with a 'jQ'.

Examples of using these selectors are demonstrated throughout the chapter. The syntax will be familiar to those who have used selectors in CSS.

BASIC SELECTORS

`*`		All elements
`element`		All elements with that element name
`#id`		Elements whose **`id`** attribute has the value specified
`.class`		Elements whose **`class`** attribute has the value specified
`selector1, selector2`		Elements that match more than one selector (see also the **`.add()`** method, which is more efficient when combining selections)

HIERARCHY

`ancestor descendant`		An element that is a descendant of another element (e.g., **`li a`**)
`parent > child`		An element that is a direct child of another element (you can use * in the place of the child to select all child elements of the specified parent)
`previous + next`		Adjacent sibling selector only selects elements that are immediately followed by the previous element
`previous ~ siblings`		Sibling selector will select any elements that are a sibling of the previous element

BASIC FILTERS

`:not(`*`selector`***`)`**		All elements except the one in the selector (e.g., **`div:not('#summary')`**)
`:first`	jQ	The first element from the selection
`:last`	jQ	The last element from the selection
`:even`	jQ	Elements with an even index number in the selection
`:odd`	jQ	Elements with an odd index number in the selection
`:eq(`*`index`***`)`**	jQ	Elements with an index number equal to the one in the parameter
`:gt(`*`index`***`)`**	jQ	Elements with an index number greater than the parameter
`:lt(`*`index`***`)`**	jQ	Elements with an index number less than the parameter
`:header`	jQ	All **`<h1>`** - **`<h6>`** elements
`:animated`	jQ	Elements that are currently being animated
`:focus`		The element that currently has focus

CONTENT FILTERS

`:contains('text')`		Elements that contain the specified text as a parameter
`:empty`		All elements that have no children
`:parent`	jQ	All elements that have a child node (can be text or element)
`:has(selector)`	jQ	Elements that contain at least one element that matches the selector (e.g., `div:has(p)` matches all `div` elements that contain a `<p>` element)

VISIBILITY FILTERS

`:hidden`	jQ	All elements that are hidden
`:visible`	jQ	All elements that consume space in the layout of the page Not selected if: `display: none`; `height` / `width`: `0`; ancestor is hidden Selected if: `visibility: hidden`; `opacity: 0` because they would take up space in layout

CHILD FILTERS

`:nth-child(expr)`	The value here is not zero-based e.g. `ul li:nth-child(2)`
`:first-child`	First child from the current selection
`:last-child`	Last child from the current selection
`:only-child`	When there is only one child of the element (`div p:only-child`)

ATTRIBUTE FILTERS

`[attribute]`		Elements that carry the specified attribute (with any value)
`[attribute='value']`		Elements that carry the specified attribute with the specified value
`[attribute!='value']`	jQ	Elements that carry the specified attribute but not the specified value
`[attribute^='value']`		The value of the attribute begins with this value
`[attribute='value']`		The value of the attribute ends with this value
`[attribute*='value']`		The value should appear somewhere in the attribute value
`[attribute|='value']`		Equal to given string, or starting with string and followed by a hyphen
`[attribute~='value']`		The value should be one of the values in a space separated list
`[attribute][attribute2]`		Elements that match all of the selectors

FORM

`:input`	jQ	All input elements
`:text`	jQ	All text inputs
`:password`	jQ	All password inputs
`:radio`	jQ	All radio buttons
`:checkbox`	jQ	All checkboxes
`:submit`	jQ	All submit buttons
`:image`	jQ	All `<img>` elements
`:reset`	jQ	All reset buttons
`:button`	jQ	All `<button>` elements
`:file`	jQ	All file inputs
`:selected`	jQ	All selected items from drop-down lists
`:enabled`		All enabled form elements (the default for all form elements)
`:disabled`		All disabled form elements (using the CSS `disabled` property)
`:checked`		All checked radio buttons or checkboxes

DOING THINGS WITH YOUR SELECTION

Once you have seen the basics of how jQuery works, most of this chapter is dedicated to demonstrating these methods.

These two pages both offer an overview to the jQuery methods and will also help you find the methods you are looking for once you have read the chapter.

You often see jQuery method names written starting with a period (.) before the name. This convention is used in this book to help you easily identify those methods as being jQuery methods rather than built-in JavaScript methods, or methods of custom objects.

When you make a selection, the **jQuery** object that is created has a property called **length**, which will return the number of elements in the object.

If the jQuery selection did not find any matching elements, you will not get an error by calling any of these methods – they just won't do or return anything.

There are also methods that are specifically designed to work with Ajax (which lets you refresh part of the page rather than an entire page) shown in Chapter 8.

CONTENT FILTERS

Get or change content of elements, attributes, text nodes

GET/CHANGE CONTENT

ELEMENTS

ATTRIBUTES

FORM VALUES

FINDING ELEMENTS

Find and select elements to work with & traverse the DOM

GENERAL

FILTER/TEST

ORDER IN SELECTION

Once you have selected the elements you want to work with (and they are in a `jQuery` object), the jQuery methods listed on these two pages perform tasks on those elements.

DIMENSION/POSITION

Get or update the dimensions or position of a box

DIMENSION

POSITION

EFFECTS & ANIMATION

Add effects and animation to parts of the page

BASIC

FADING

SLIDING

CUSTOM

EVENTS

Create event listeners for each element in the selection

DOCUMENT/FILE

USER INTERACTION

There used to be methods for individual types of event, so you may see methods such as `.click()`, `.hover()`, `.submit()`. However, these have been dropped in favour of the `.on()` method to handle events.

A MATCHED SET / JQUERY SELECTION

When you select one or more elements, a `jQuery` object is returned. It is also known as a **matched set** or a **jquery selection.**

SINGLE ELEMENT

If a selector returns one element, the **jQuery** object contains a reference to just one element node.

```
$('ul')
```

This selector picks the **<ul>** element from the page. So the **jQuery** object contains a reference to just one node (the only **<ul>** element in the page):

Each element is given an index number. Here there is just one element in the object.

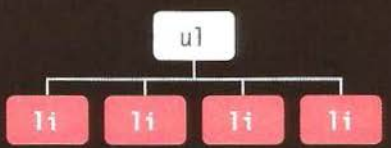
INDEX	ELEMENT NODE
0	ul

MULTIPLE ELEMENTS

If a selector returns several elements, the **jQuery** object contains references to each element.

```
$('li')
```

This selector picks all the **<li>** elements. Here, the **jQuery** object has references for each of the nodes that was selected (each **<li>** element):



The resulting jQuery object contains four list items. Remember that index numbers start at zero.

INDEX	ELEMENT NODE
0	li#one.hot
1	li#two.hot
2	li#three.hot
3	li#four

JQUERY METHODS THAT GET AND SET DATA

Some jQuery methods both retrieve information from, and update the contents of, elements. But they do not always apply to all elements.

GET INFORMATION

If a jQuery selection holds more than one element, and a method is used to get information from the selected elements, it will **retrieve information from only the first element** in the matched set.

In the list example we have been using, the following selector chooses the four **<li>** elements from a list.

```
$('li')
```

When you use the **.html()** method (which will be introduced on p316) to get information from an element, it will return the content of the first element in the matched set.

```
var content = $('li').html();
```

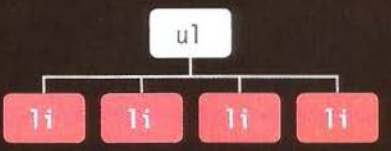
This will retrieve the content of the first list item, and store it in the variable called **content**.

To get a different element, you can use methods to traverse (p336) or filter (p338) the selection, or write a more specific selector (p302).

To get the content of all of the elements, see the **.each()** method (p324).

SET INFORMATION

If a jQuery selection holds more than one element, and a method is used to update information on the page, it will **update all of the elements** in the matched set, not just the first one.



When you use the **.html()** method (which you meet on p316) to update the element, it will replace the contents of each element in the matched set. Here, it updates the content of each item in the list.

```
$('li').html('Updated');
```

This will update the content of all of the list items in the matched set with the word **Updated**.

To update just *one* element, you can use methods to traverse (p336) or filter (p338) the selection, or write a more specific selector (p302).

JQUERY OBJECTS STORE REFERENCES TO ELEMENTS

When you create a selection with jQuery, it stores a reference to the corresponding nodes in the DOM tree. It does not create copies of them.

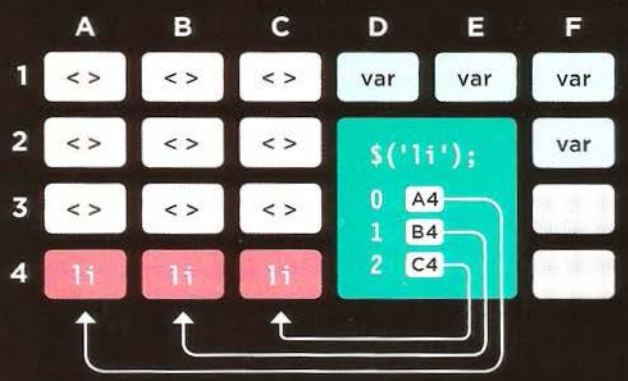
As you have seen, when HTML pages load, the browser creates a model of the page in memory. Imagine your browser's memory is a set of tiles:

- `< >` Nodes in the DOM take up a tile
- `var` Variables take up a tile
- `$` Complex JavaScript objects may take several tiles because they hold more data

In reality, the items in the browser's memory are not spread out as they are in this diagram, but the diagram helps explain the concept.

When you create a jQuery selection, the jQuery object holds **references** to the elements in the DOM – it does not create a copy of them.

When programmers say that a variable or object is storing a reference to something, what it is doing is storing the *location* a piece of information in the browser's memory. Here, the jQuery object would know that the list items are stored in A4, B4, and C4. Again, this is purely for illustration purposes; the browser's memory is not quite as simple as a checkerboard with these locations.



The jQuery object is an array-like object because it stores a list of the elements in the same order that they appear in the HTML document (unlike other objects where the order of the properties is not usually preserved).

CACHING JQUERY SELECTIONS IN VARIABLES

A jQuery object stores references to elements.
Caching a jQuery object stores a reference to it in a variable.

To create a jQuery object takes time, processing resources, and memory. The interpreter must:

1. Find the matching nodes in the DOM tree
2. Create the jQuery object
3. Store references to the nodes in the jQuery object

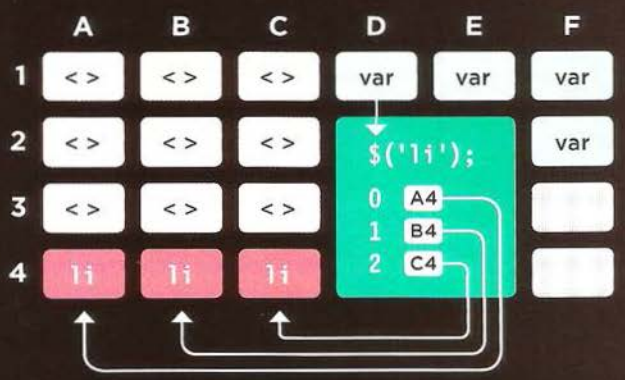
So, if the code needs to use the same selection more than once, it is better to use that same jQuery object again rather than repeat the above process. To do this, you store a reference to the jQuery object in a variable.

Below, a jQuery object is created. It stores the locations of the `<li>` elements in the DOM tree.

```
$('li');
```

A reference to this object is in turn stored in a variable called `$listItems`. Note that when a variable contains a jQuery object, it is often given a name beginning with the `$` symbol (to help differentiate it from other variables in your script).

```
$listItems = $('li');
```



Caching jQuery selections is similar to the idea of storing a reference to a DOM node once you have made a DOM query (as you saw in Chapter 5).

LOOPING

In plain JavaScript, if you wanted to do the same thing to several elements, you would need to write code to loop through all of the elements you selected.

With jQuery, when a selector returns multiple elements, you can update all of them using the one method. There is no need to use a loop.

In this code, the same value is added to the `class` attribute for all of the elements that are found using the selector. It doesn't matter if there are one or many.

c07/js/looping.js

JAVASCRIPT

```
$('li em').addClass('seasonal');
$('li.hot').addClass('favorite');
```

In this example, the first selector applies only to one element and the `class` attribute's new value triggers a CSS rule that adds a calendar icon to the left of it.

The second selector applies to three elements. The new value added to the `class` attribute for each of these elements triggers a CSS rule that adds a heart icon on the right-hand side.

The ability to update all of the elements in the jQuery selection is known as **implicit iteration**.

When you want to get information from a series of elements, you can use the `.each()` method (which you meet on p324) rather than writing a loop.

RESULT

CHAINING

If you want to use more than one jQuery method on the same selection of elements, you can list several methods at a time using dot notation to separate each one, as shown below.

In this one statement, three methods act on the same selection of elements:
`hide()` hides the elements
`delay()` creates a pause
`fadeIn()` fades in the elements

The process of placing several methods in the same selector is referred to as **chaining**. As you can see, it results in code that is far more compact.

JAVASCRIPT c07/js/chaining.js

```
$('li[id!="one"]').hide().delay(500).fadeIn(1400);
```

RESULT

To make your code easier to read, you can place each new method on a new line:

```
$('li[id!="one"]')
  .hide()
  .delay(500)
  .fadeIn(1400);
```

Each line starts with the dot notation, and the semicolon at the end of the statement indicates that you have finished working with this selection.

Most methods used to **update** the jQuery selection can be chained. However the methods that **retrieve** information from the DOM (or about the browser) cannot be chained.

It is worth noting that if one method in the chain does not work, the rest will not run either.

CHECKING A PAGE IS READY TO WORK WITH

jQuery's `.ready()` method checks that the page is ready for your code to work with.

`$(document)` creates a jQuery object representing the page.

When the page is ready, the function inside the parentheses of the **`.ready()`** method is run.

JQUERY OBJECT

READY EVENT METHOD

```
$(document).ready(function() {
  // Your script goes here
});
```

As with plain JavaScript, if the browser has not yet constructed the DOM tree, jQuery will not be able to select elements from it.

If you place a script at the end of the page (just before the closing **</body>** tag), the elements will be loaded into the DOM tree.

If you wrap your jQuery code in the method above, it will still work when used elsewhere on the page or even in another file.

A shorthand for this is shown on the right-hand page. It is more commonly used than this longer version.

THE `load` EVENT

jQuery had a `.load()` method. It fired on the `load` event, but has been replaced by the `.on()`. As you saw on p272, the `load` event fires after the page and all of its resources (images, CSS, and scripts) have loaded.

You should use this when your script relies on assets to have loaded, e.g., if it needs to know the dimensions of an image.

It works in all browsers, and also provides function-level scope for the variables it contains.

VS

THE `.ready()` METHOD

jQuery's `.ready()` method checks if the browser supports the `DOMContentLoaded` event, because it fires as soon as the DOM has loaded (it does not wait for other assets to finish loading) and can make the page appear as if it is loading faster.

If `DOMContentLoaded` is supported, jQuery creates an event listener that responds to that event. But the event is only supported in modern browsers. In older browsers, jQuery will wait for the `load` event to fire.

VS

PLACING SCRIPTS BEFORE THE CLOSING `</body>` TAG

When you place your script at the end of the page (before the closing `</body>` tag), the HTML will have loaded into the DOM before the script runs.

You will, however, still see people using the `.ready()` method because scripts that use it will still work if someone moves the script tag elsewhere in the HTML page. (This is particularly common when that script is being made available for other people to use.)

SHORTCUT FOR READY EVENT METHOD ON DOCUMENT OBJECT

```
$(function() {
  // Your script goes here
});
```

Above, you can see the shorthand that is commonly used instead of `$(document).ready()`

A positive side-effect of writing jQuery code inside this method is that it creates function-level scope for its variables.

This function-level scope prevents naming collisions with other scripts that might use the same variable names.

Any statements inside the method automatically run when the page has loaded. This is the version that will be used in the examples in the rest of the chapter.