# 11

# CONTENT PANELS

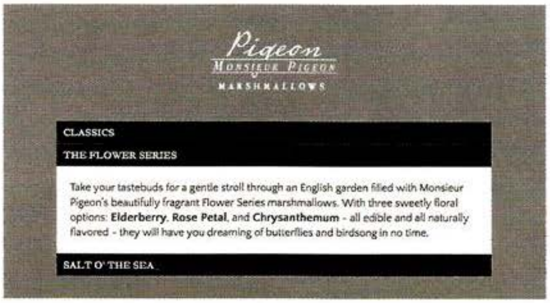
Content panels allow you to showcase extra information within a limited space. In this chapter, you will see several examples of content panels that also give you practical insight into creating your own scripts using jQuery.

In this chapter, you will see how to create many types of content panels: accordions, tabbed panels, modal windows (also known as a lightboxes), a photo viewer, and a responsive slider. Each example of a content panel also demonstrates how to apply the code you have learned throughout the book so far in a practical setting.

Throughout the chapter, reference will be made to more complex jQuery plugins that extend the functionality of the examples shown here. But the code samples in this chapter also show how it is possible to achieve techniques you will have seen on popular websites in relatively few lines of code (without needing to rely on plugins written by other people).

## ACCORDION

An accordion features titles which, when clicked, expand to show a larger panel of content.



CLASSICS

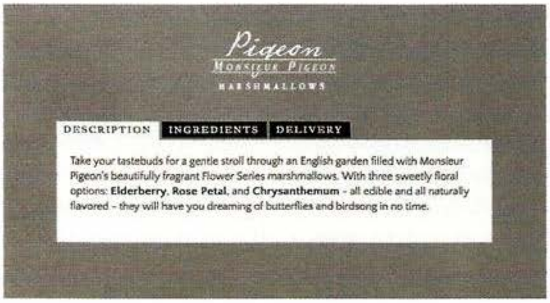
THE FLOWER SERIES

Take your tastebuds for a gentle stroll through an English garden filled with Monsieur Pigeon's beautifully fragrant Flower Series marshmallows. With three sweetly floral options: **Elderberry**, **Rose Petal**, and **Chrysanthemum** - all edible and all naturally flavored - they will have you dreaming of butterflies and birdsong in no time.

SALT O' THE SEA

## TABBED PANEL

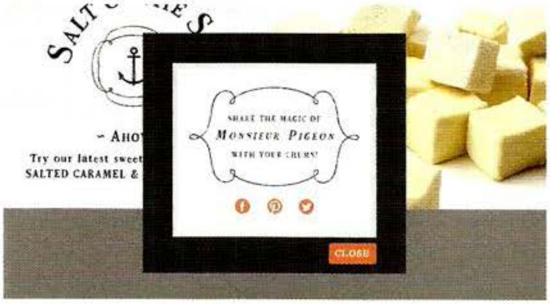
Tabs automatically show one panel, but when you click on another tab, the panel is changed.



Take your tastebuds for a gentle stroll through an English garden filled with Monsieur Pigeon's beautifully fragrant Flower Series marshmallows. With three sweetly floral options: **Elderberry**, **Rose Petal**, and **Chrysanthemum** - all edible and all naturally flavored - they will have you dreaming of butterflies and birdsong in no time.

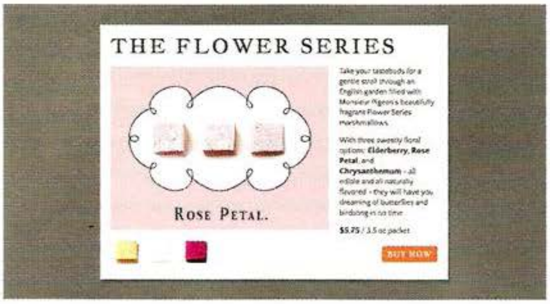
## MODAL WINDOW

When you click on a link for a modal window (or "lightbox"), a hidden panel will be displayed.



## PHOTO VIEWER

Photo viewers display different images within the same space when the user clicks on the thumbnails.


THE FLOWER SERIES

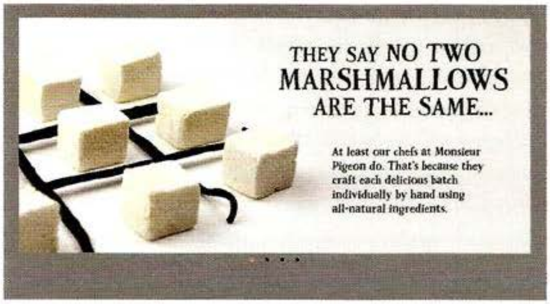
Take your tastebuds for a gentle stroll through an English garden filled with Monsieur Pigeon's beautifully fragrant Flower Series marshmallows.

With three sweetly floral options: **Elderberry**, **Rose Petal**, and **Chrysanthemum** - all edible and all naturally flavored - they will have you dreaming of butterflies and birdsong in no time.

**$5.75** / 3.5 oz packet

## RESPONSIVE SLIDER

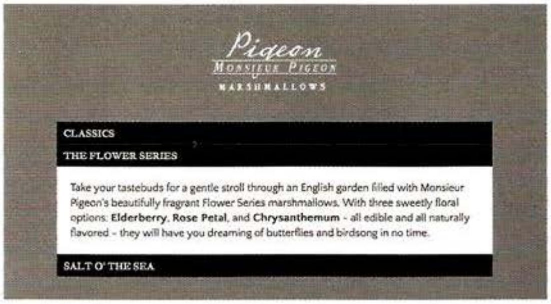
The slider allows you to show panels of content that slide into view as the user navigates between them.


THEY SAY NO TWO MARSHMALLOWS ARE THE SAME...

At least our chefs at Monsieur Pigeon do. That's because they craft each delicious batch individually by hand using all-natural ingredients.

## CREATING A JQUERY PLUGIN

The final example revisits the accordion (the first example) and turns it into a jQuery plugin.



CLASSICS

THE FLOWER SERIES

Take your tastebuds for a gentle stroll through an English garden filled with Monsieur Pigeon's beautifully fragrant Flower Series marshmallows. With three sweetly floral options: **Elderberry**, **Rose Petal**, and **Chrysanthemum** - all edible and all naturally flavored - they will have you dreaming of butterflies and birdsong in no time.

SALT O' THE SEA

# SEPARATION OF CONCERNS

As you saw in the introduction to this book, it is considered good practice to separate your content (in HTML markup), presentation (in CSS rules), and behaviors (in JavaScript).

In general, your code should reflect that:

- HTML is responsible for structuring content
- CSS is responsible for presentation
- JavaScript is responsible for behavior

Enforcing this separation produces code that is easier to maintain and reuse. While this may already be a familiar concept to you, it's important to remember as it is very easy to mix these concerns in with your JavaScript. As a rule, editing your HTML templates or stylesheets should not necessitate editing your scripts and vice versa.

You can also place event listeners and calls to functions in JavaScript files rather than adding them to the end of an HTML document.

If you need to change the styles associated with an element, rather than having styles written in the JavaScript, you can update the value of the `class` attributes for those elements. In turn, they can trigger new rules from the CSS file that change the appearance of those elements.

When your scripts access the DOM, you can uncouple them from the HTML by using `class` selectors rather than tag selectors.

# ACCESSIBILITY & NO JAVASCRIPT

When writing any script, you should think about those who might be using a web page in different situations than you.

## ACCESSIBILITY

Whenever a user can interact with an element:

- If it is a link, use <a>
- If it acts like a button, use a button

Both can gain focus, so users can move between them focusable elements using the Tab key (or other non-mouse solution). And although any element can become focusable by setting its `tabindex` attribute, only <a> elements and some input elements fire a `click` event when users press the Enter key on their keyboard (the ARIA `role="button"` attribute will not simulate this event).

## NO JAVASCRIPT

This chapter's accordion menu, tabbed panels, and responsive slider all hide some of their content by default. This content would be inaccessible to visitors that do not have JavaScript enabled if we didn't provide alternative styling. One way to solve this is by adding a `class` attribute whose value is `no-js` to the opening `<html>` tag. This class is then removed by JavaScript (using the `replace()` method of the String object) if JavaScript is enabled. The `no-js` class can then be used to provide styles targeted to visitors who do not have JavaScript enabled.

HTML c11/no-js.html

```html
<!DOCTYPE html><html class="no-js"> ...
  <body>
    <div class="js-warning">You must enable JavaScript to buy from us</div>
    <!-- Turn off your JavaScript to see the difference -->
    <script src="js/no-js.js"></script>
  </body>
</html>
```

JAVASCRIPT c11/js/no-js.js

```javascript
var elDocument = document.documentElement;
elDocument.className = elDocument.className.replace(/(^|\s)no-js(\s|$)/, '$1');
```

# ACCORDION

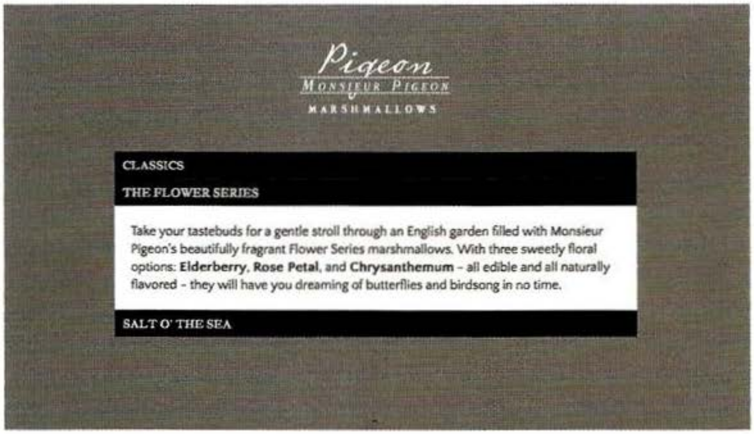
When you click on the title of an accordion, its corresponding panel expands to reveal the content.

An accordion is usually created within an unordered list (in a `<ul>` element). Each `<li>` element is a new item in the accordion. The items contain:

- A visible label (in this example, it is a `<button>`)
- A hidden panel holding the content (a `<div>`)

Clicking a label prompts the associated panel to be shown (or to be hidden if it is in view). To just hide or show a panel, you could change the value of the `class` attribute on the associated panel (triggering a new CSS rule to show or hide it). But, in this case, jQuery will be used to animate the panel into view or hide it.

HTML5 introduces `<details>` and `<summary>` elements to create a similar effect, but (at the time of writing) browser support was not widespread. Therefore, a script like this would still be used for browsers that do not support those features.



CLASSICS

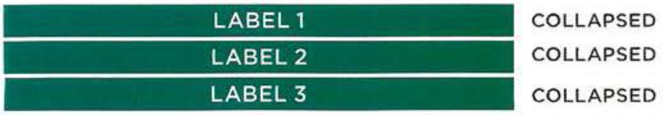
THE FLOWER SERIES

Take your tastebuds for a gentle stroll through an English garden filled with Monsieur Pigeon's beautifully fragrant Flower Series marshmallows. With three sweetly floral options: **Elderberry**, **Rose Petal**, and **Chrysanthemum** - all edible and all naturally flavored - they will have you dreaming of butterflies and birdsong in no time.

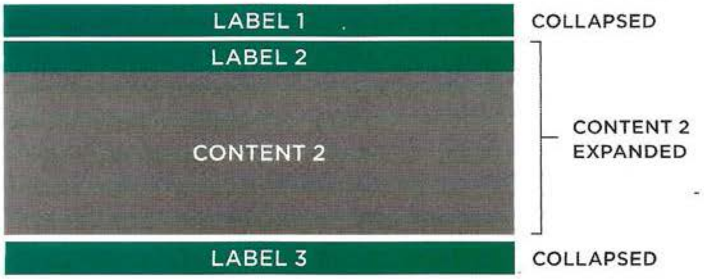
SALT O' THE SEA

Other tabs scripts include liteAccordion and zAccordion. They are also included in jQuery UI and Bootstrap.

ACCORDION WITH ALL PANELS COLLAPSED



ACCORDION WITH SECOND PANEL EXPANDED



When the page loads, CSS rules are used to hide the panels.

Clicking a label prompts the hidden panel that follows it to animate and reveal its full height. This is done using jQuery.

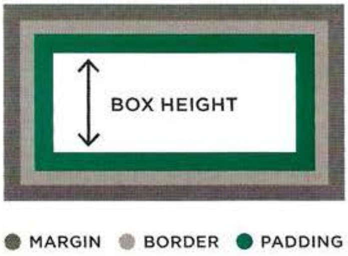
Clicking on the label again would hide the panel.

## ANIMATING CONTENT WITH SHOW, HIDE, AND TOGGLE

jQuery's `.show()`, `.hide()`, and `.toggle()` methods animate the showing and hiding of elements.

jQuery calculates the size of the box, including its content, and any margins and padding. This helps if you do not know what content appears in a box.

(To use CSS animation, you would need to calculate the box's height, margin and padding.)



`.toggle()` saves you writing conditional code to tell whether the box is already being shown or not. (If a box is shown, it hides it, and if hidden, it will show it.)

The three methods are all shorthand for the `animate()` method. For example, the `show()` method is shorthand for:

```
$('.accordion-panel')
.animate({
    height: 'show',
    paddingTop: 'show',
    paddingBottom: 'show',
    marginTop: 'show',
    marginBottom: 'show'
});
```

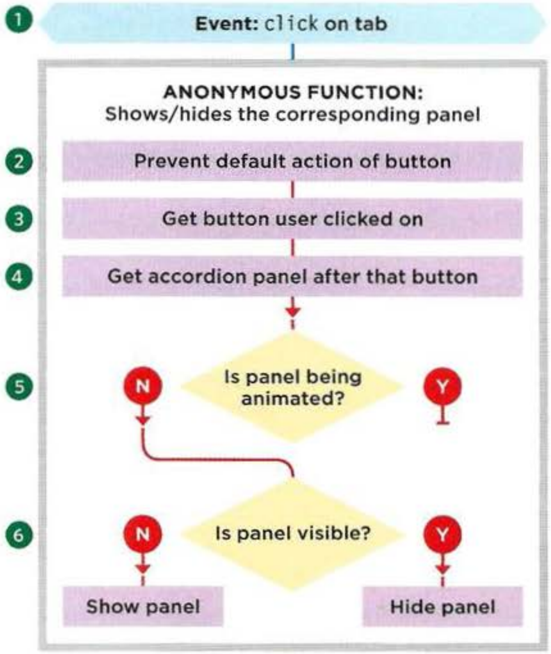
# CREATING AN ACCORDION

Below you can see a diagram, rather like a flowchart. These diagrams have two purposes. They help you:

i) Follow the code samples; the numbers on the diagram correspond with the steps on the right, and the script on the right-hand page. Together, the diagrams, steps, and comments in the code should help you understand how each example works.

ii) Learn how to plan a script before coding it.

This is not a "formal" diagram style, but it gives you a visual idea of what is going on with the script. The diagrams show how a collection of small, individual instructions achieve a larger goal, and if you follow the arrows you can see how the data flows around the parts of the script.


1
Event: click on tab
ANONYMOUS FUNCTION:
Shows/hides the corresponding panel
2
Prevent default action of button
3
Get button user clicked on
4
Get accordion panel after that button
5
N
Is panel being animated?
Y
6
N
Is panel visible?
Y
Show panel
Hide panel

Some programmers use Unified Modeling Language or class diagrams - but they have a steeper learning curve, and these flowcharts are here to help you see how the interpreter moves through the script.

Now let's take a look at how the diagram is translated into code. The steps below correspond to the numbers next to the JavaScript code on the right-hand page and the diagram on the left.

1. A jQuery collection is created to hold elements whose `class` attribute has a value of `accordion`. In the HTML you can see that this corresponds to the unordered list element (there could be several lists on the page, each acting as an accordion). An event listener waits for the user to click on one of the buttons whose `class` attribute has a value of `accordion-control`. This triggers an anonymous function.

2. The `preventDefault()` method prevents browsers treating the the button like a submit button. It can be a good idea to use the `preventDefault()` method early in a function so that anyone looking at your code knows that the form element or link does not do what they might expect it to.

3. Another jQuery selection is made using the `this` keyword, which refers to the element the user clicked upon. Three jQuery methods are applied to that jQuery selection holding the element the user clicked on.

4. `.next('.accordion-panel')` selects the next element with a class of `accordion-panel`.

5. `.not(':animated')` checks that it is not in the middle of being animated. (If the user repeatedly clicks the same label, this stops the `.slideToggle()` method from queuing multiple animations.)

6. `.slideToggle()` will show the panel if it is currently hidden and will hide the panel if it is currently visible.

HTML

c11/accordion.html

```html
<ul class="accordion">
  <li>
    <button class="accordion-control">Classics</button>
    <div class="accordion-panel">Panel content goes here...</div>
  </li>
  <li>
    <button class="accordion-control">The Flower Series</button>
    <div class="accordion-panel">Panel content goes here...</div>
  </li>
  <li>
    <button class="accordion-control">Salt O' the Sea</button>
    <div class="accordion-panel">Panel content goes here...</div>
  </li>
</ul>
```

CSS

c11/css/accordion.css

```css
.accordion-panel {
  display: none;}
```

JAVASCRIPT

c11/js/accordion.js

```javascript
① $('.accordion').on('click', '.accordion-control', function(e){ // When clicked
②   e.preventDefault();                        // Prevent default action of button
③   $(this)                                    // Get the element the user clicked on
④     .next('.accordion-panel')                // Select following panel
⑤     .not(':animated')                        // If it is not currently animating
⑥     .slideToggle();                          // Use slide toggle to show or hide it
  });
```

Note how steps 4, 5, and 6 are chained off the same jQuery selection.
You saw a screenshot of the accordion example on p492, at the start of this section.

# TABBED PANEL

When you click on one of the tabs, its corresponding panel is shown. Tabbed panels look a little like index cards.

You should be able to see all of the tabs, but:

- Only one tab should look *active*.
- Only the panel that corresponds to the active tab should be shown (all other panels should be hidden).

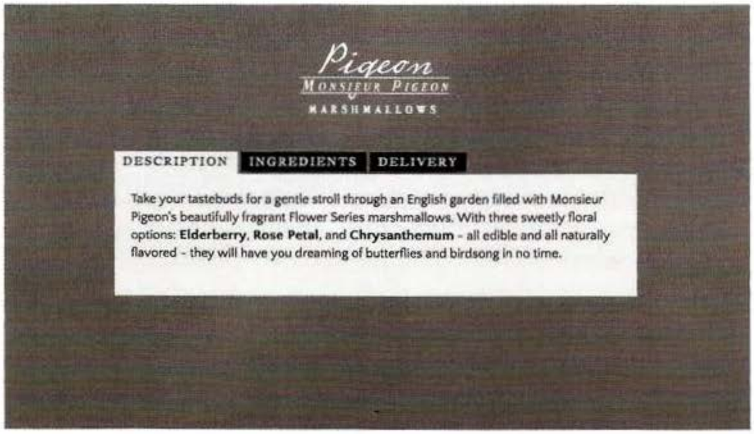
The tabs are typically created using an unordered list. Each `<li>` element represents a tab and within each tab is a link.

The panels follow the unordered list that holds the tabs, and each panel is stored in a `<div>`.

To associate the tab to the panel:

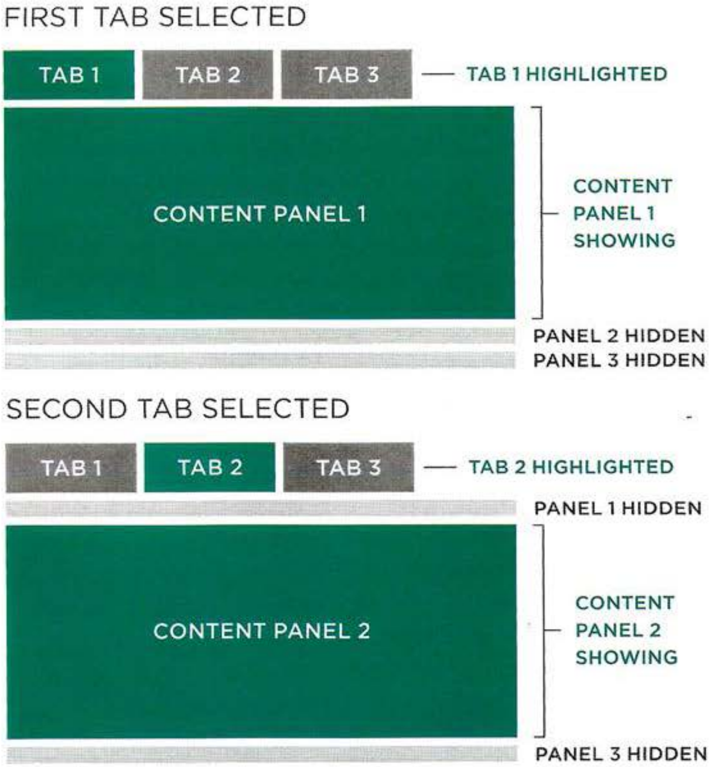
- The link in the tab, like all links, has an `href` attribute.
- The panel has an `id` attribute.

Both attributes share the same value. (This is the same principle as creating a link to another location within an HTML page.)



Take your tastebuds for a gentle stroll through an English garden filled with Monsieur Pigeon's beautifully fragrant Flower Series marshmallows. With three sweetly floral options: **Elderberry**, **Rose Petal**, and **Chrysanthemum** - all edible and all naturally flavored - they will have you dreaming of butterflies and birdsong in no time.

Other tabs scripts include Tabslet and Tabulous. They are also included in jQuery UI and Bootstrap.



When the page loads, CSS is used to make the tabs sit next to each other and to indicate which one is considered active.

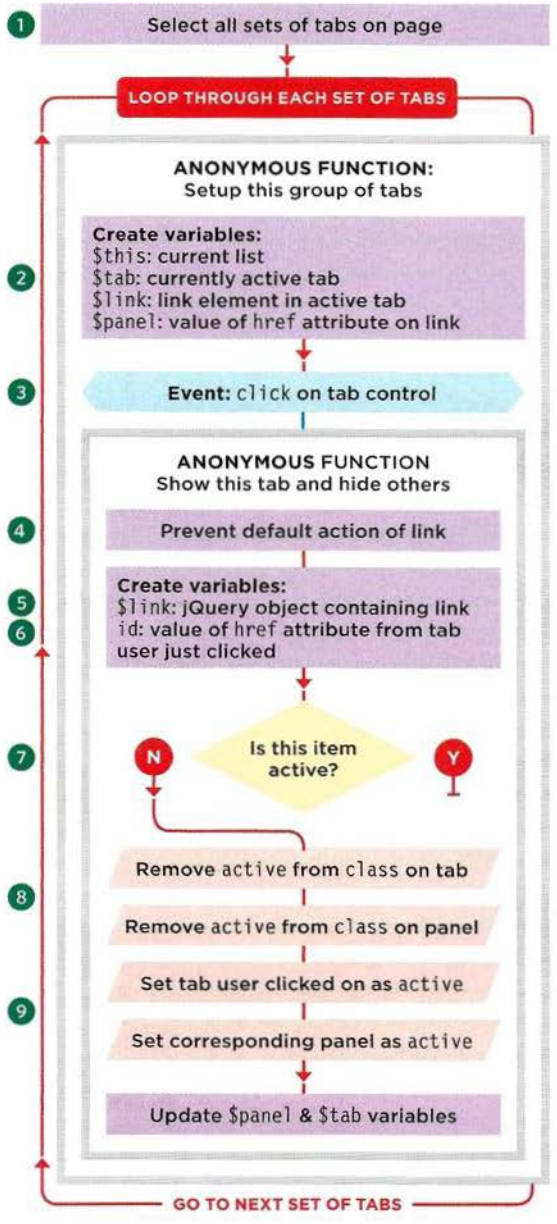
CSS also hides the panels, except for the one that corresponds with the active tab.

When the user clicks on the link inside a tab, the script uses jQuery to get the value of the `href` attribute from the link. This corresponds to the `id` attribute on the panel that should be shown.

The script then updates the values in the `class` attribute on that tab and panel, adding a value of `active`. It also removes that value from the tab and panel that had previously been active.

If the user does not have JavaScript enabled, the link in the tab takes the user to the appropriate part of the page.

# CREATING TAB PANELS


1
Select all sets of tabs on page
LOOP THROUGH EACH SET OF TABS
ANONYMOUS FUNCTION:
Setup this group of tabs
2
Create variables:
$this: current list
$tab: currently active tab
$link: link element in active tab
$panel: value of href attribute on link
3
Event: click on tab control
ANONYMOUS FUNCTION
Show this tab and hide others
4
Prevent default action of link
5
6
Create variables:
$link: jQuery object containing link
id: value of href attribute from tab user just clicked
7
Is this item active?
N
Y
8
Remove active from class on tab
Remove active from class on panel
9
Set tab user clicked on as active
Set corresponding panel as active
Update $panel & $tab variables
GO TO NEXT SET OF TABS

The flowchart shows the steps that are involved in creating tabs when they are found in the HTML. Below, you can see how these steps can be translated into code:

**1.** A jQuery selection picks all sets of tabs within the page. The `.each()` method calls an anonymous function that is run for each set of tabs (like a loop). The code in the anonymous function deals with one set of tabs at a time, and the steps would be repeated for each set of tabs on the page.
**2.** Four variables hold details of the active tab:
**i)** `$this` holds the current set of tabs.
**ii)** `$tab` holds the currently active tab.
The `.find()` method selects the active tab.
**iii)** `$link` holds the <a> element within that tab.
**iv)** `$panel` holds the value of the `href` attribute for the active tab (this variable will be used to hide the panel if the user selects a different one).
**3.** An event listener is set up to check for when the user clicks on any tab within that list. When they do, it runs another anonymous function.
**4.** `e.preventDefault()` prevents the link that users clicked upon taking them to that page.
**5.** Creates a variable called `$link` to hold the current link inside a jQuery object.
**6.** Creates a variable called `id` to hold the value of the `href` attribute from the tab that was clicked. It is called `id` because it is used to select the matching content panel (using its `id` attribute).
**7.** An `if` statement checks whether the `id` variable contains a value, and the current item is **not** active. If both conditions are met:
**8.** The previously active tab and panel have the `class` of `active` removed (which deactivates the tab and hides the panel).
**9.** The tab that was clicked on and its corresponding panel both have `active` added to their `class` attributes (which makes the tab look active and displays its corresponding panel, which was hidden). At the same time, references to these elements are stored in the `$panel` and `$tab` variables.

HTML

c11/tabs.html

```html
<ul class="tab-list">
  <li class="active"><a class="tab-control" href="#tab-1">Description</a></li>
  <li><a class="tab-control" href="#tab-2">Ingredients</a></li>
  <li><a class="tab-control" href="#tab-3">Delivery</a></li>
</ul>
<div class="tab-panel active" id="tab-1">Content 1...</div>
<div class="tab-panel" id="tab-2">Content 2...</div>
<div class="tab-panel" id="tab-3">Content 3...</div>
```

CSS

c11/css/tabs.css

```css
.tab-panel {
  display: none;}
.tab-panel.active {
  display: block;}
```

JAVASCRIPT

c11/js/tabs.js

```javascript
① $('.tab-list').each(function(){                    // Find lists of tabs
②   var $this    = $(this);                          // Store this list
    var $tab     = $this.find('li.active');          // Get the active list item
    var $link    = $tab.find('a');                   // Get link from active tab
    var $panel   = $($link.attr('href'));            // Get active panel

③   $this.on('click', '.tab-control', function(e) { // When click on a tab
④     e.preventDefault();                            // Prevent link behavior
⑤     var $link = $(this);                           // Store the current link
⑥     var id    = this.hash;                         // Get href of clicked tab

⑦     if (id && !$link.is('.active')) {              // If not currently active
⑧       $panel.removeClass('active');                // Make panel inactive
        $tab.removeClass('active');                  // Make tab inactive

⑨       $panel = $(id).addClass('active');           // Make new panel active
        $tab   = $link.parent().addClass('active'); // Make new tab active
      }
    });
  });
```

# MODAL WINDOW

A modal window is any type of content that appears "in front of" the rest of the page's content. It must be "closed" before the rest of the page can be interacted with.

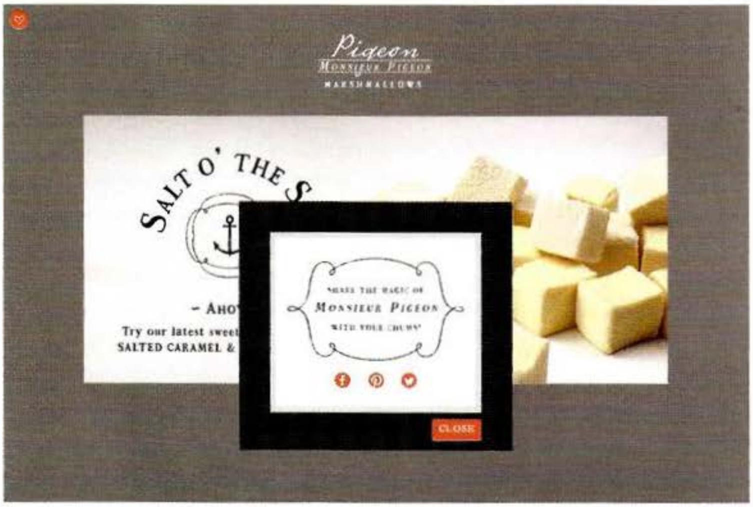
In this example, a modal window is created when the user clicks on the heart button in the top left-hand corner of the page.

The modal window opens in the center of the page, allowing users to share the page on social networks.

The content for the modal window will typically sit within the page, but it is hidden when the page loads using CSS.

JavaScript then takes that content and displays it inside `<div>` elements that create the modal window on top of the existing page.

Sometimes modal windows will dim out the rest of the page behind them. They can be designed to either appear automatically when the page has finished loading or they can be triggered by the user interacting with the page.



Other examples of modal window scripts include Colorbox (by Jack L. Moore), Lightbox 2 (by Lokesh Dhakar), and Fancybox (by Fancy Apps). They are also included in jQuery UI and Bootstrap.

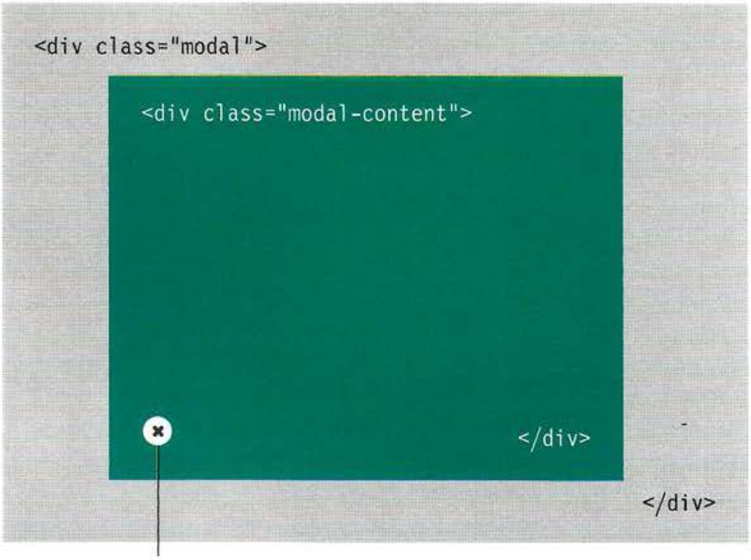
A **design pattern** is a term programmers use to describe a common approach to solving a range of programming tasks.

This script uses the **module pattern**. It is a popular way to write code that contains both **public** and **private** logic.

Once the script has been included in the page, other scripts can use its public methods: `open()`, `close()`, or `center()`. But users do not need to access the variables that create the HTML, so they remain private (on p505 the private code is shown on green).

Using modules to build parts of an application has benefits:

- It helps organize your code.
- You can test and reuse the individual parts of the app.
- It creates scope, preventing variable /method names clashing with other scripts.



This modal window script creates an object (called `modal`), which, in turn, provides three new methods you can use to create modal windows:

`open()` opens a modal window
`close()` closes the window
`center()` centers it on the page

Another script would be used to call the `open()` method and specify what content should appear in the modal window.

Users of this script only need to know how the `open()` method works because:

- `close()` is called by an event listener when the user clicks on the close button.
- `center()` is called by the `open()` method and also by an event listener if the user resizes the window.

When you call the `open()` method, you specify the content that you want the modal window to contain as a parameter (you can also specify its width and height if you want).

In the diagram, you can see that the script adds the content to the page inside `<div>` elements.

`div.modal` acts as a frame around the modal window.

`div.modal-content` acts as a container for the content being added to the page.

`button.modal-close` allows the user to close the modal window.

# CREATING MODALS

The modal script needs to do two things:

1. Create the HTML for the modal window
2. Return the modal object itself, which consists of the open(), close(), and center() methods

Including the script in the HTML page does not have any visible effect (rather like including jQuery in your page does not affect the appearance of the page).

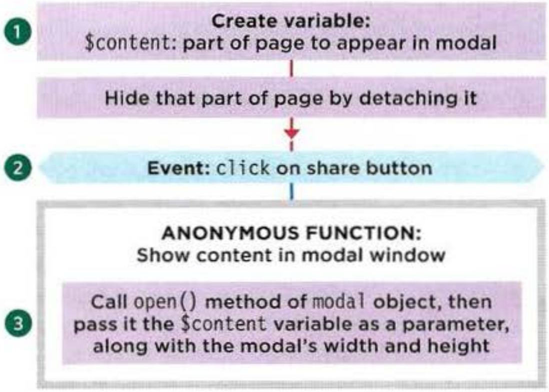
But it does allow any other script you write to use the functionality of the modal object and call its open() method to create a modal window (just like including jQuery script includes the jQuery object in your page and allows you to use its methods).

This means that people who use the script only need to know how to call the open() method and tell it what they want to appear in the modal window.

In the example on the right, the modal window is called by a script called modal-init.js. You will see how to create the modal object and its methods on the next double page spread, but for now consider that including this script is the equivalent of adding the following to your own script. It creates an object called modal and adds three methods to the object:

```
var modal = {
  center: function() {
    // Code for center() goes here
  },
  open: function(settings) {
    // Code for open() goes here
  },
  close: function() {
    // Code for close() goes here
  }
};
```

The modal-init.js file removes the share content from the HTML page. It then adds an event handler to call the modal object's open() method to open a modal window containing the content it just removed from the page. init is short for initialize and is commonly used in the name of files and functions that set up a page or other part of a script.


1
Create variable:
$content: part of page to appear in modal
Hide that part of page by detaching it
2
Event: click on share button
ANONYMOUS FUNCTION:
Show content in modal window
3
Call open() method of modal object, then pass it the $content variable as a parameter, along with the modal's width and height

**1.** First the script gets the contents of the element that has an id attribute whose value is share-options. Note how the jQuery .detach() method removes this content from the page.

**2.** Next an event handler is set to respond to when the user clicks on the share button. When they do, an anonymous function is run.

**3.** The anonymous function uses the open() method of the modal object. It takes parameters in the form of an object literal:

- content: the content to be shown in the modal window. Here it is the content of the element whose id attribute has a value of share-options.
- width: the width of the modal window.
- height: the height of the modal window.

Step 1 uses the .detach() method because it keeps the elements and event handlers in memory so they can be used again later. jQuery also has a .remove() method but it removes the items completely.

# USING THE MODAL SCRIPT

HTML

c11/modal-window.html

```
①     <div id="share-options">
        <!-- This is where the message and sharing buttons go -->
      </div>
      <script src="js/jquery.js"></script>
②     <script src="js/modal-window.js"></script>
③     <script src="js/modal-init.js"></script>
    </body>
  </html>
```

In the HTML above, you should note three things:

1. A `<div>` that contains the sharing options.
2. A link to the script that creates the `modal` object (`modal-window.js`).
3. A link to the script that will open a modal window using the `modal` object (`modal-init.js`), using it to display the sharing options.

The `modal-init.js` file below opens the modal window. Note how the `open()` method is passed three pieces of information in JSON format:

**i)** `content` for modal (required)
**ii)** `width` of modal (optional - overrides default)
**iii)** `height` of modal (optional - overrides default)

JAVASCRIPT

c11/js/modal-init.js

```
   (function(){
①    var $content = $('#share-options').detach();   // Remove modal from page

②    $('#share').on('click', function() {            // Click handler to open modal
③      modal.open({content: $content, width:340, height:300});
     });                          ⓘ             ⓘⓘ            ⓘⓘⓘ
   }());
```

The `z-index` of the modal window must be very high so that it appears on top of any other content.

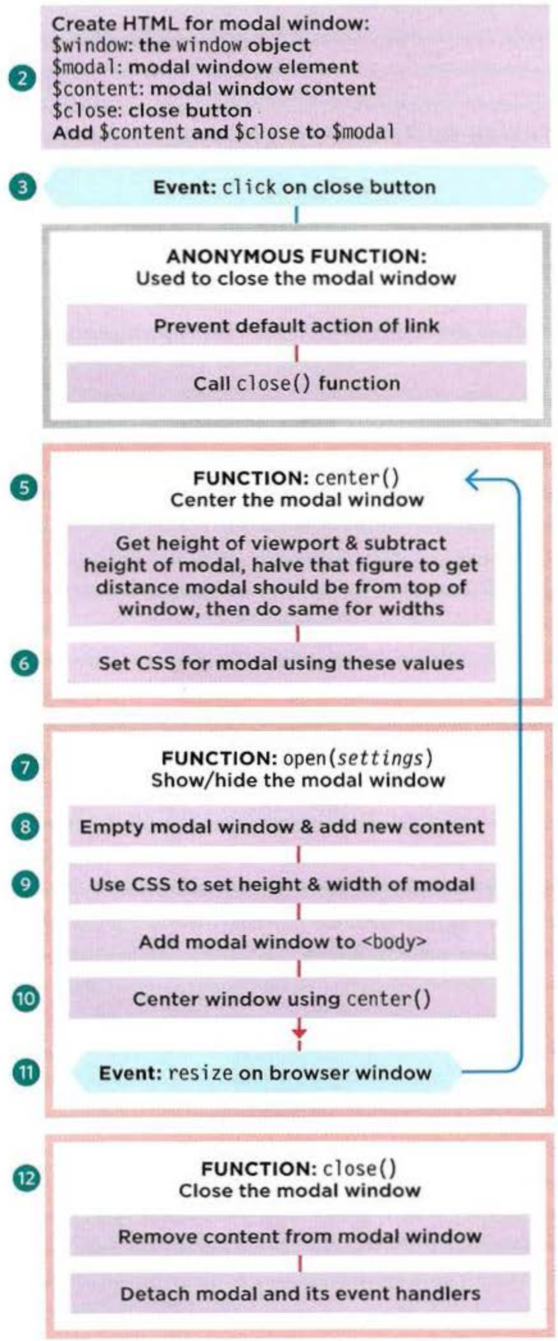
These styles ensure the modal window sits on top of the page (there are more styles in the full example).

CSS

c11/css/modal-window.css

```
.modal {
  position: absolute;
  z-index: 1000;}
```

# MODAL OBJECT


2
Create HTML for modal window:
$window: the window object
$modal: modal window element
$content: modal window content
$close: close button
Add $content and $close to $modal
3
Event: click on close button
ANONYMOUS FUNCTION:
Used to close the modal window
Prevent default action of link
Call close() function
5
FUNCTION: center()
Center the modal window
Get height of viewport & subtract height of modal, halve that figure to get distance modal should be from top of window, then do same for widths
6
Set CSS for modal using these values
7
FUNCTION: open(settings)
Show/hide the modal window
8
Empty modal window & add new content
9
Use CSS to set height & width of modal
Add modal window to <body>
10
Center window using center()
11
Event: resize on browser window
12
FUNCTION: close()
Close the modal window
Remove content from modal window
Detach modal and its event handlers

Below are the steps for creating the modal object. Its methods are used to create modal windows.

1. The modal object is declared. The methods of this object are created by an Immediately Involved Function Expression or IIFE (see p97). (This step is not shown in the flowchart.)
2. Store the current window object in a jQuery selection, then create the three HTML elements needed for the modal window. Assemble the modal window and store it in $modal.
3. Add an event handler to the close button which calls the modal object's close() method.
4. Following the return keyword, there is a code block in curly braces. It creates three public methods of the modal object. **Please note:** This step is not shown in the flowchart.
5. The center() method creates two variables:
i) top: takes the height of the browser window and subtracts the height of the modal window. This number is divided by two, giving the distance of the modal from the top of the browser window.
ii) left: takes the width of the browser window and subtracts the width of modal window. This number is divided by two, giving the distance of the modal from the left of the browser window.
6. The jQuery .css() method uses these variables to position the modal in the center of the page.
7. open() takes an object as a parameter; it is referred to as settings (the data for this object was shown on the previous page).
8. Any existing content is cleared from the modal, and the content property of the settings object is added to the HTML created in steps 1 and 2.
9. The width and height of the modal are set using values from the settings object. If none were given, auto is used. Then the modal is added to the page using the appendTo() method.
10. center() is used to center the modal window.
11. If the window is resized, call center() again.
12. close() empties the modal, detaches the HTML from the page, and removes any event handlers.

In the code below, the lines that are highlighted in green are considered **private**. These lines of code are only used within the object. (This code cannot be accessed directly from outside the object.)

When this script has been included in a page, the `center()`, `open()`, and `close()` methods in steps 5-12 are available on the `modal` object for other scripts to use. They are referred to as **public**.

JAVASCRIPT

c11/js/modal-window.js

```
①  var modal = (function() {                          // Declare modal object
     var $window = $(window);
     var $modal = $('<div class="modal"/>');          // Create markup for modal
②    var $content = $('<div class="modal-content"/>');
     var $close = $('<button role="button" class="modal-close">close</button>');

     $modal.append($content, $close);                 // Add close button to modal

     $close.on('click', function(e) {                 // If user clicks on close
③      e.preventDefault();                            // Prevent link behavior
       modal.close();                                 // Close the modal window
     });

④    return {                                          // Add code to modal
       center: function() {                            // Define center() method
⑤        // Calculate distance from top and left of window to center the modal
         var top = Math.max($window.height() - $modal.outerHeight(), 0) / 2;
         var left = Math.max($window.width() - $modal.outerWidth(), 0) / 2;
         $modal.css({                                  // Set CSS for the modal
⑥          top: top + $window.scrollTop(),             // Center vertically
           left: left + $window.scrollLeft()           // Center horizontally
         });
       },
⑦      open: function(settings) {                      // Define open() method
⑧        $content.empty().append(settings.content);    // Set new content of modal

         $modal.css({                                  // Set modal dimensions
⑨          width: settings.width || 'auto',            // Set width
           height: settings.height || 'auto'           // Set height
         }).appendTo('body');                          // Add it to the page

⑩        modal.center();                               // Call center() method
⑪        $(window).on('resize', modal.center);         // Call it if window resized
       },
       close: function() {                             // Define close() method
         $content.empty();                             // Remove content from modal
⑫        $modal.detach();                              // Remove modal from page
         $(window).off('resize', modal.center);        // Remove event handler
       }
     };
   }());
```

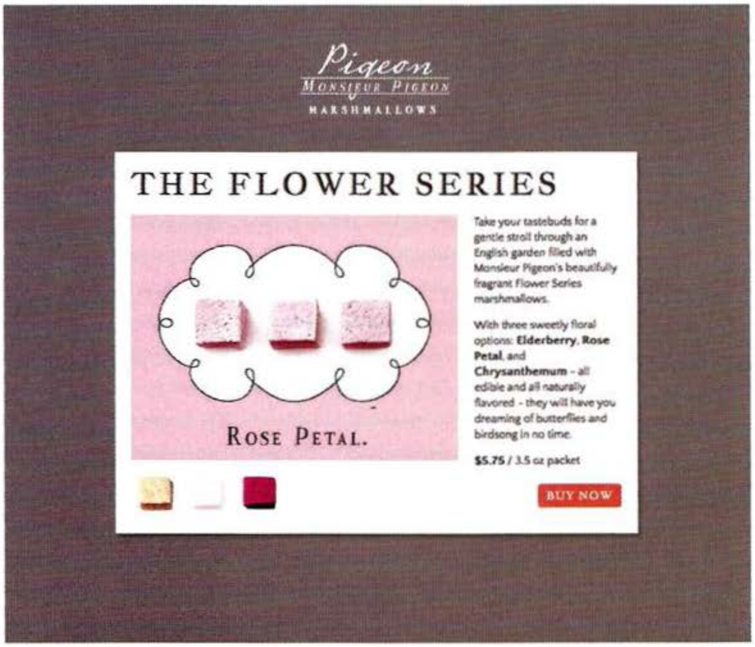
# PHOTO VIEWER

The photo viewer is an example of an image gallery. When you click on a thumbnail, the main photograph is replaced with a new image.

In this example, you can see one main image with three thumbnails underneath it.

The HTML for the photo viewer consists of:

- One large `<div>` element that will hold the main picture. The images that sit in the `<div>` are centered and scaled down if necessary to fit within the allocated area.
- A second `<div>` element that holds a set of thumbnails that show the other images you can view. These thumbnails sit inside links. The `href` attribute on those links point to the larger versions of their images.



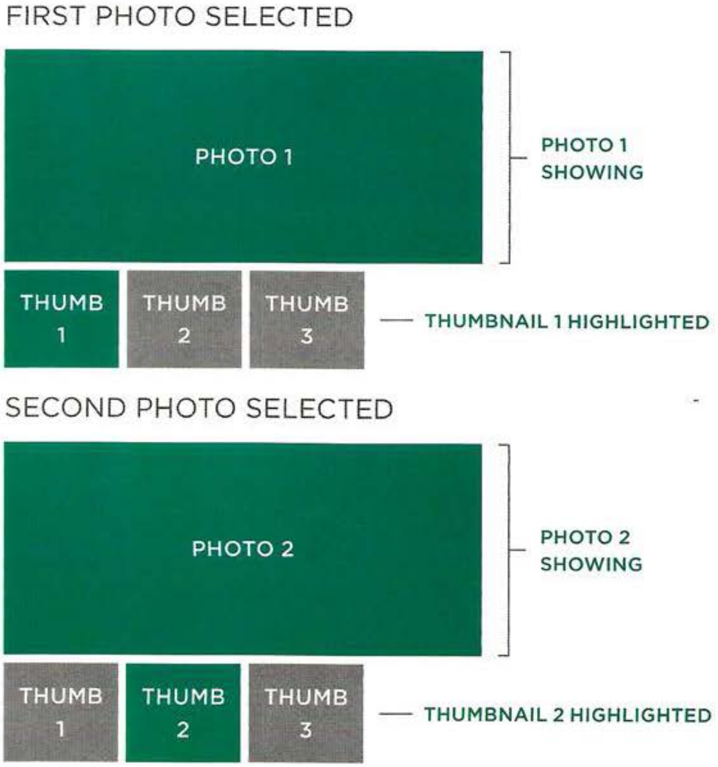
THE FLOWER SERIES

Take your tastebuds for a gentle stroll through an English garden filled with Monsieur Pigeon's beautifully fragrant Flower Series marshmallows.

With three sweetly floral options: **Elderberry**, **Rose Petal**, and **Chrysanthemum** - all edible and all naturally flavored - they will have you dreaming of butterflies and birdsong in no time.

**$5.75** / 3.5 oz packet

Other gallery scripts include Galleria, Gallerific, and TN3Gallery.


FIRST PHOTO SELECTED

SECOND PHOTO SELECTED

When you click on a thumbnail, an event listener triggers an anonymous function that:

1. Looks at the value of the `href` attribute (which points to the large image)
2. Creates a new `<img>` element to hold that image
3. Makes it invisible
4. Adds it to the big `<div>` element

Once the image has loaded, a function called `crossfade()` is used to fade between the existing image and the new one that has been requested.

# USING THE PHOTO VIEWER

In order to use the photo viewer, you create a `<div>` element to hold the main image. It is empty, and its `id` attribute has a value of `photo-viewer`.

The thumbnails sit in another `<div>`. Each one is in an `<a>` element with three attributes:

- `href` points to the larger version of the image
- `class` always has a value of `thumb` and the current main image has a value of `active`
- `title` describes the image (it will be used for `alt` text)

c11/photo-viewer.html

HTML

```html
<div id="photo-viewer"></div>
<div id="thumbnails">
  <a href="img/photo-1.jpg" class="thumb active" title="Elderberry mallow">
    <img src="img/thumb-1.jpg" alt="Elderberry Marshmallow" /></a>
  <a href="img/photo-2.jpg" title="Rose Marshmallow" class="thumb">
    <img src="img/thumb-2.jpg" alt="Rose Marshmallow" /></a>
  <a href="img/photo-3.jpg" title="Chrysanthemum  Marshmallow" class="thumb">
    <img src="img/thumb-3.jpg" alt="Chrysanthemum Marshmallow" /></a>
</div>
```

The script comes before the closing `</body>` tag. As you will see, it simulates the user clicking on the first thumbnail.

The `<div>` that holds the main picture uses relative positioning. This removes the element from normal flow, so a `height` for the viewer must be specified.

While images are loading, a `class` of `is-loading` is added to them (it displays an animated loading gif). When the image has loaded, `is-loading` is removed.

If the images are larger than the viewer the `max-width` and `max-height` properties will scale them to fit. To center the image within the viewer a mix of CSS and JavaScript will be used. See p511 for detailed explanation.

c11/css/photo-viewer.css

CSS

```css
#photo-viewer {
  position: relative;
  height: 300px;
  overflow: hidden;}

#photo-viewer.is-loading:after {
  content: url(images/load.gif);
  position: absolute;
  top: 0;
  right: 0;}

#photo-viewer img {
  position: absolute;
  max-width: 100%;
  max-height: 100%;
  top: 50%;
  left: 50%;}

a.active {
  opacity: 0.3;}
```

# ASYNCHRONOUS LOADING & CACHING IMAGES

This script (shown on the next page) shows two interesting techniques:
1. Dealing with asynchronous loading of content
2. Creating a custom cache object

## SHOWING THE RIGHT IMAGE WHEN LOADING IMAGES ASYNCHRONOUSLY

### PROBLEM:

The larger images are only loaded into the page when the user clicks on a thumbnail, and the script waits for the image to fully load before displaying it.

Because larger images take longer to load, if a user clicks on two different images in quick succession:
1. The second image could load faster than the first one and be displayed in the browser.
2. It would be replaced by the *first* image the user clicked on when that image had loaded. This could make users think the wrong image has loaded.

### SOLUTION:

When the user clicks on a thumbnail:
- A function-level variable called `src` stores the path to this image.
- A global variable called `request` is also updated with the path to this image.
- An event handler is set to call an anonymous function when *this* image has loaded.

When the image loads, the event handler checks if the `src` variable (which holds the path to *this* image) matches the `request` variable. If the user had clicked on another image since the one that just loaded, the `request` variable would no longer match the `src` variable and the image should not be shown.

## CACHING IMAGES THAT HAVE ALREADY LOADED IN THE BROWSER

### PROBLEM:

When the user requests a big image (by clicking on the thumbnail), a new `<img>` element is created and added to the frame.

If the user goes back to look at an image they have already selected, you do not want to create a new element and load the image all over again.

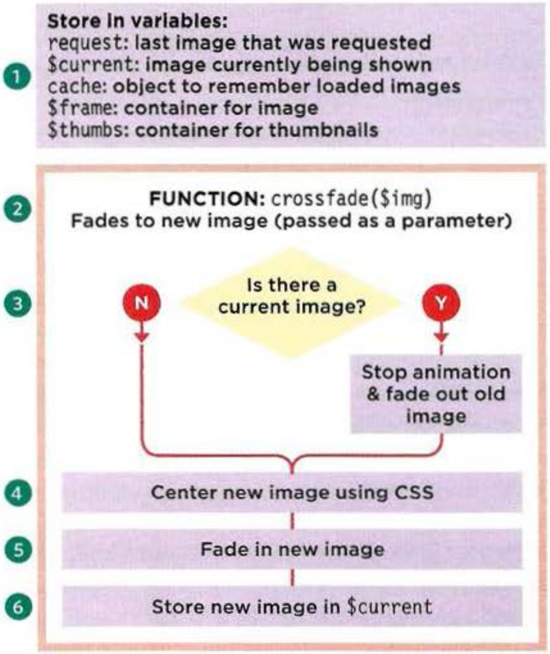
### SOLUTION:

A simple object is created, and it is called `cache`. Every time a new `<img>` element is created, it will be added to the `cache` object.

That way, each time an image is requested, the code can check if the corresponding `<img>` element is already in the cache (rather than creating it again).

# PHOTO VIEWER SCRIPT (1)

This script introduces some new concepts, so it will be spread over four pages. On these two pages you see the global variables and `crossfade()` function.



1
Store in variables:
request: last image that was requested
$current: image currently being shown
cache: object to remember loaded images
$frame: container for image
$thumbs: container for thumbnails
2
FUNCTION: crossfade($img)
Fades to new image (passed as a parameter)
3
Is there a current image?
N
Y
Stop animation & fade out old image
4
Center new image using CSS
5
Fade in new image
6
Store new image in $current

1. A set of global variables is created. They can be used throughout the script - both in the `crossfade()` function (on this page) and the event handlers (on p512).

2. The `crossfade()` function will be called when the user has clicked on a thumbnail. It is used to fade between the old image and the new one.

3. An `if` statement checks to see if there is an image loaded at the moment. If there is, two things happen: the `.stop()` method will stop any current animation and then `.fadeOut()` will fade the image out.

4. To center the image in the viewer element, you set two CSS properties on the image. Combined with the CSS rules you saw on p508, these CSS properties will center the image in its container. (See the diagrams on the bottom of p511.)

i) `marginleft`: gets the width of the image using the `.width()` method, divides it by two, and uses that number as a negative margin.

ii) `marginTop`: gets the height of the image, using the `.height()` method, divides it by two, and makes that number a negative margin.

5. If the new image is currently being animated, the animation is stopped and the image is faded in.

6. Finally, the new image becomes the current image and is stored in the `$current` variable.

## THE CACHE OBJECT

The idea of a cache object might sound complicated, but all objects are just sets of key/value pairs. You can see what the cache object might look like on the right. When an image is requested by clicking on a new thumbnail, a new property is added to the cache object:

- The key added to the cache object is the path to the image (below this is referred to as *src*). Its value is another object with two properties.
- *src*.`$img` holds a reference to a jQuery object that contains the newly created `<img>` element.
- *src*.`isLoading` is a property indicating whether or not it is currently loading (its value is a Boolean).

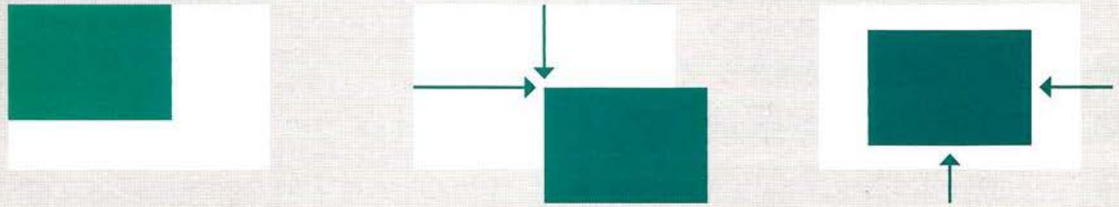
```
var cache = {
  "c11/img/photo-1.jpg": {
    "$img": jQuery object,
    "isLoading": false
  },
  "c11/img/photo-2.jpg": {
    "$img": jQuery object,
    "isLoading": false
  },
  "c11/img/photo-3.jpg": {
    "$img": jQuery object,
    "isLoading": false
  }
}
```

JAVASCRIPT

c11/js/photo-viewer.js

```
  var request;                         // Latest image to be requested
  var $current;                        // Image currently being shown
① var cache   = {};                    // Cache object
  var $frame  = $('#photo-viewer');    // Container for image
  var $thumbs = $('.thumb');           // Container for image

② function crossfade($img) {           // Function to fade between images
                                       // Pass in new image as parameter
    if ($current) {                    // If there is currently an image showing
③     $current.stop().fadeOut('slow'); // Stop animation and fade it out
    }

    $img.css({                         // Set the CSS margins for the image
④     marginLeft: -$img.width() / 2,   // Negative margin of half image's width
      marginTop: -$img.height() / 2    // Negative margin of half image's height
    });

⑤   $img.stop().fadeTo('slow', 1);     // Stop animation on new image & fade in

⑥   $current = $img;                   // New image becomes current image

  }
```

## CENTERING THE IMAGE

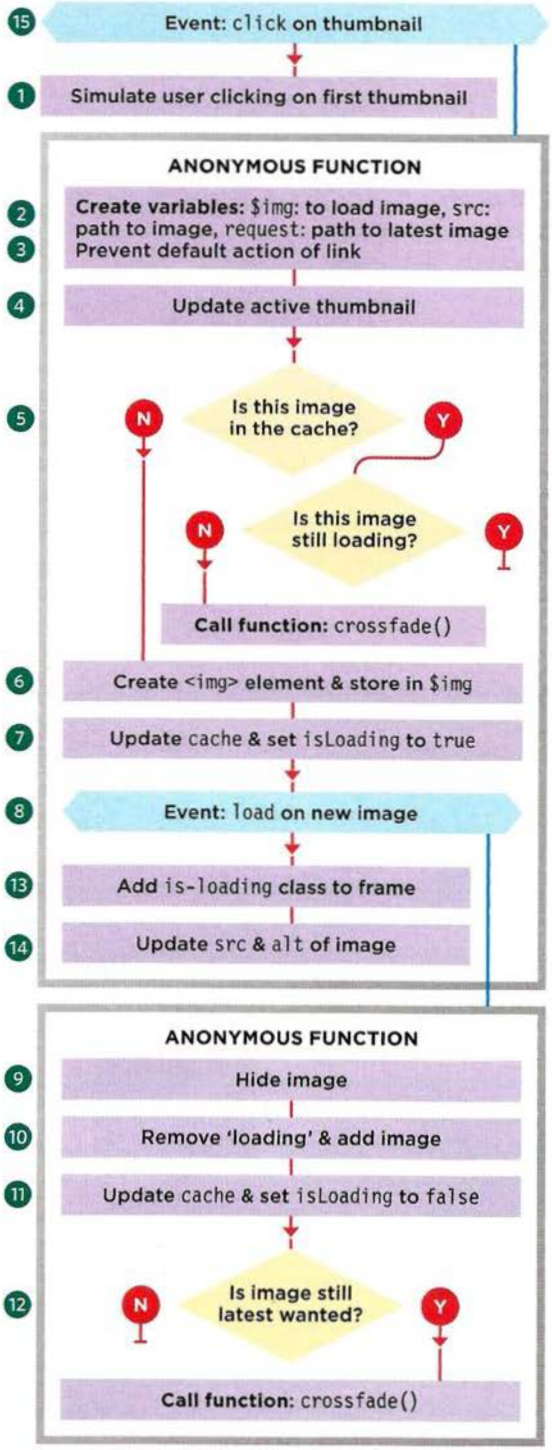


**i)** Centering the image involves three steps. In the style sheet, absolute positioning is used to place it in the top-left corner of the containing element.

**ii)** In the style sheet, the image is moved down and right by 50% of the **container's** width and height:
**width:** 800px ÷ **2** = 400 px
**height:** 500px ÷ **2** = 250 px

**iii)** In the script, negative margins move the image up and left by half the **image's** width and height:
**width:** 500 px ÷ **2** = 250 px
**height:** 400px ÷ **2** = 200 px

# PHOTO VIEWER SCRIPT (2)


15
Event: click on thumbnail
1
Simulate user clicking on first thumbnail
ANONYMOUS FUNCTION
2
3
Create variables: $img: to load image, src: path to image, request: path to latest image
Prevent default action of link
4
Update active thumbnail
5
N
Is this image in the cache?
Y
N
Is this image still loading?
Y
Call function: crossfade()
6
Create <img> element & store in $img
7
Update cache & set isLoading to true
8
Event: load on new image
13
Add is-loading class to frame
14
Update src & alt of image
ANONYMOUS FUNCTION
9
Hide image
10
Remove 'loading' & add image
11
Update cache & set isLoading to false
12
N
Is image still latest wanted?
Y
Call function: crossfade()

**1.** The thumbnails are wrapped in links. Every time users click on one, the anonymous function will run.

**2.** Three variables are created:

**i)** $img will be used to create new <img> elements that will hold the larger images when they load.

**ii)** src (a function-level variable) holds the path to the new image (it was in the href attribute of the link).

**iii)** request (a global variable) holds the same path.

**3.** The link is prevented from loading the image.

**4.** The active class is removed from *all* the thumbs and is added to the thumb that was clicked on.

**5.** If the image is in the cache object and it has finished loading, the script calls crossfade().

**6.** If the image has not yet loaded, the script creates a new <img> element.

**7.** It is added to the cache. isLoading is set to true.

**8.** At this point, the image has not loaded yet (only an empty <img> element was created). When the image loads, the load event triggers a function (which needs to be written *before* the image loads).

**9.** First, the function hides the image that just loaded.

**10.** It then removes the is-loading class from the frame and adds the new image to the frame.

**11.** In the cache object, isLoading is set to false (as it will have loaded when this function runs).

**12.** An if statement checks if the image that just loaded is the one the user last requested. To see how this is done, look back at step 2 again:

- The src variable holds the path to the image that just loaded. It has function-level scope.
- The request variable is updated each time the user clicks on an image. It has global scope.

So, if the user has clicked on an image since this one, the request and src variables will not be the same and nothing should be done. If they do match, then: crossfade() is called to show the image.

**13.** Having set all of this in place, it is time to load the image. The is-loading class is added to the frame.

**14.** Finally, by adding a value to the src attribute on the image, the image will start to load. Its alt text is retrieved from the title attribute on the link.

**15.** The last line of the script simulates the user clicking on the first thumbnail. This will load the first image into the viewer when the script first runs.

JAVASCRIPT

c11/js/photo-viewer.js

```
①  $(document).on('click', '.thumb', function(e){ // When a thumb is clicked on
      var $img;                                 // Create local variable called $img
②     var src = this.href;                      // Store path to image
      request = src;                            // Store path again in request

③     e.preventDefault();                       // Stop default link behavior

④     $thumbs.removeClass('active');            // Remove active from all thumbs
      $(this).addClass('active');               // Add active to clicked thumb

      if (cache.hasOwnProperty(src)) {          // If cache contains this image
⑤       if (cache[src].isLoading === false) { // And if isLoading is false
          crossfade(cache[src].$img);           // Call crossfade() function
        }
      } else {                                  // Otherwise it is not in cache
⑥       $img = $('<img/>');                    // Store empty <img/> element in $img
        cache[src] = {                          // Store this image in cache
⑦         $img: $img,                          // Add the path to the image
          isLoading: true                       // Set isLoading property to true
        };

        // Next few lines will run when image has loaded but are prepared first
⑧       $img.on('load', function() {           // When image has loaded
⑨         $img.hide();                         // Hide it
          // Remove is-loading class from frame & append new image to it
⑩         $frame.removeClass('is-loading').append($img);
⑪         cache[src].isLoading = false;        // Update isLoading in cache
          // If still most recently requested image then
          if (request === src) {
⑫           crossfade($img);                   // Call crossfade() function
          }                                     // Solves asynchronous loading issue
        });

⑬       $frame.addClass('is-loading');         // Add is-loading class to frame

        $img.attr({                             // Set attributes on <img> element
⑭         'src': src,                          // Add src attribute to load image
          'alt': this.title || ''               // Add title if one was given in link
        });

      }

    });

    // Last line runs once (when rest of script has loaded) to show the first image
⑮  $('.thumb').eq(0).click();                  // Simulate click on first thumbnail
```

# RESPONSIVE SLIDER

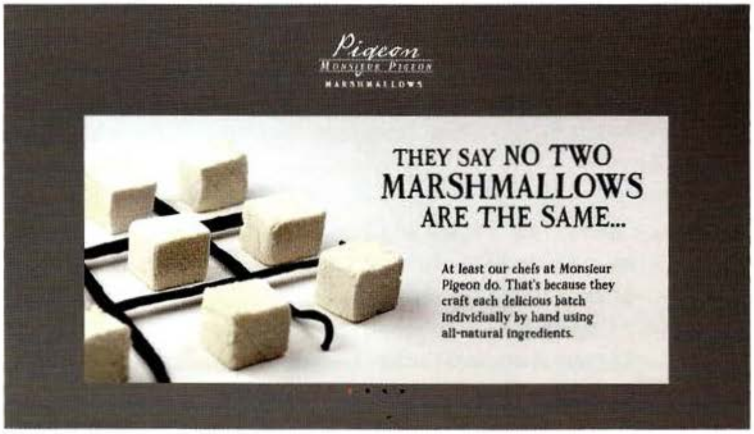
A slider positions a series of items next to each other, but only shows one at a time. The images then slide from one to the next.

This slider loads several panels, but only shows one at a time. It also provides buttons that allow users to navigate between each of the slides and a timer to move them automatically after a set interval.

In the HTML, the entire slider is contained within a `<div>` element whose `class` attribute has value of `slider-viewer`. In turn, the slider needs two further `<div>` elements:

- A container for the slides. Its `class` attribute has a value of `slide-group`. Inside this container, each individual slide is in another `<div>` element.
- A container for the buttons. Its `class` attribute has a value of `slide-buttons`. The buttons are added by the script.

If the HTML contains markup for more than one slider, the script will automatically transform all of them into separate sliders.



THEY SAY NO TWO MARSHMALLOWS ARE THE SAME...

At least our chefs at Monsieur Pigeon do. That's because they craft each delicious batch individually by hand using all-natural ingredients.

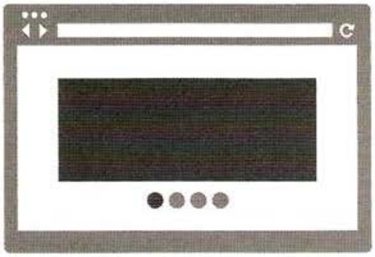
Other slider scripts include Unslider, Anything Slider, Nivo Slider, and WOW Slider. Sliders are also included in jQuery UI and Bootstrap.

When the page first loads, the CSS hides all of the slides, which takes them out of normal flow. The CSS then sets the `display` property of the first slide `block` to make it visible.

The script then goes through each slide and:

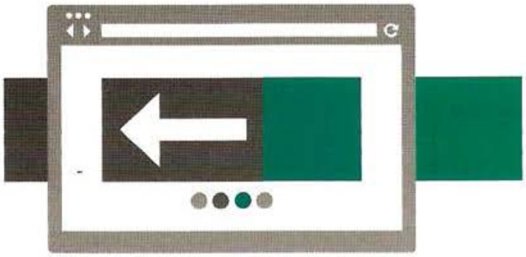
- Assigns an index number to that slide
- Adds a button for it under the slide group

For example, if there are four slides, when the page first loads, the first slide will be shown by default, and four buttons will be added underneath it.

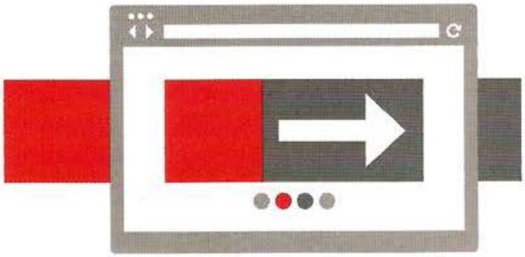


The index numbers allow the script to identify each individual slide. To keep track of which slide is currently being shown, the script uses a variable called `currentIndex` (holding the index number of the current slide). When the page loads, this is 0, so it shows the first slide. It also needs to know which slide it is moving to, which is stored in a variable called `newSlide`.

When it comes to moving between the slides (and creating the sliding effect), if the index number of the new slide is *higher* than the index number of the current slide, then the new slide is placed to the *right* of the group. As the visible slide is animated to the left, the new slide automatically starts to come into view, taking its place.



If the index number of the new slide is *lower* than the current index, then the new slide is placed to the *left* of the current slide, and as it is animated to the right, the new slide starts to come into view.



After the animation, the hidden slides are placed behind the one that is currently active.

# USING THE SLIDER

As long as you include the script within your page, any HTML that uses the structure shown here will get transformed into a slider.

There could be several sliders on the page and each one will be transformed using the same script that you see on the next double-page spread.

c11/slider.html

HTML

```
<div class="slide-viewer">
  <div class="slide-group">
    <div class="slide slide-1"><!-- slide content --></div>
    <div class="slide slide-2"><!-- slide content --></div>
    <div class="slide slide-3"><!-- slide content --></div>
    <div class="slide slide-4"><!-- slide content --></div>
  </div>
</div>
<div class="slide-buttons"></div>
```

The width of the `slide-viewer` is not fixed, so it works in a responsive design. But a height does need to be specified because the slides have an absolute position (this removes them from the document flow and without it they could only be 1px tall).

Each slide is shown at the same width and height as the viewer. If the content of a slide is larger than the viewer, the `overflow` property on the `slide-viewer` hides the parts of the slides that extend beyond the frame. If it is smaller it is positioned to the top-left.

c11/css/slider.css

CSS

```
slide-viewer {
  position: relative;
  overflow: hidden;
  height: 300px;}

.slide-group {
  width: 100%;
  height: 100%;
  position: relative;}

.slide {
  width: 100%;
  height: 100%;
  display: none;
  position: absolute;}

.slide:first-child {
  display: block;}
```

# SLIDER SCRIPT OVERVIEW

A jQuery selector finds the sliders within the HTML markup. An anonymous function then runs for each one to create the slider. There are four key parts to the function.

## 1: SETUP

Each slider needs some variables, they are in function-level scope so they:

- Can have different values for each slider
- Do not conflict with variables outside of the script

## 2: CHANGING SLIDE: move()

move() is used to move from one slide to another, and to update the buttons that indicate which slide is currently being shown. It is called when the user clicks on a button, and by the advance() function.

## 3: A TIMER TO SHOW THE NEXT SLIDE AFTER 4 SECONDS: advance()

A timer will call move() after 4 seconds.
To create a timer, JavaScript's window object has a setTimeout() method. It executes a function after a number of milliseconds. The timer is often assigned to a variable, and it uses the following syntax:

```
var timeout = setTimeout(function, delay);
```

- *timeout* is a variable name that will be used to identify the timer.
- *function* can be a named function or an anonymous function.
- *delay* is the number of milliseconds before the function should run.

To stop the timer, call clearTimeout(). It takes one parameter: the variable used to identify the timer:

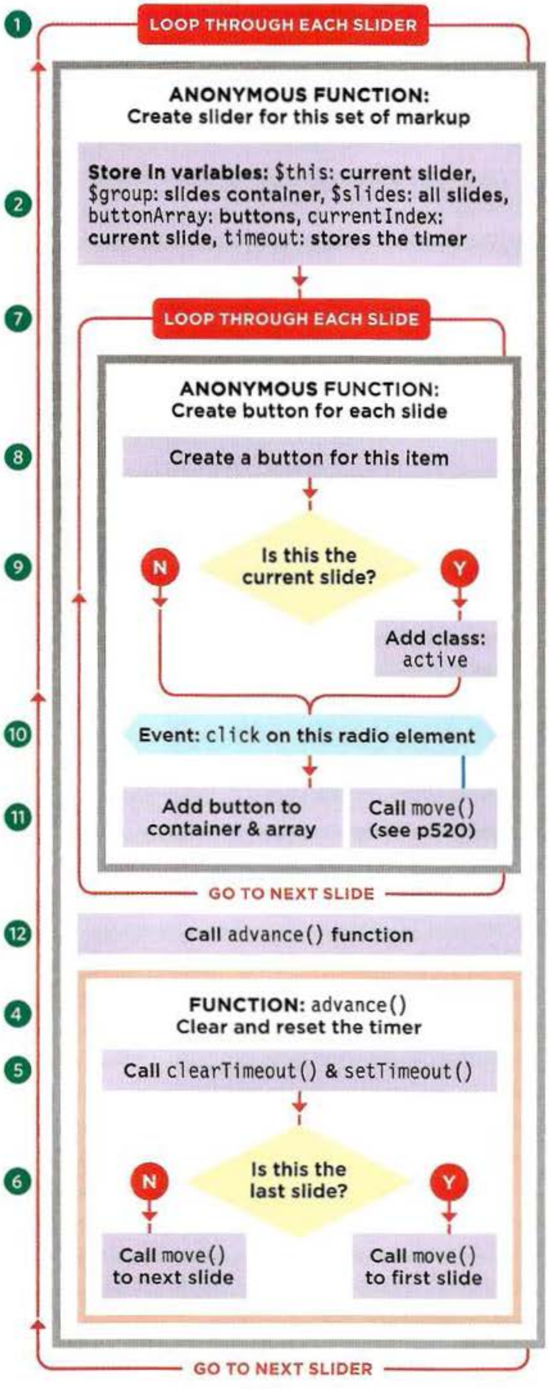
```
clearTimeout(timeout);
```

## 4: PROCESSING EACH OF THE SLIDES THAT APPEAR WITHIN A SLIDER

The code loops through each of the slides to:

- Create the slider
- Add a button for each slide with an event handler that calls the move() function when users clicks it

# SLIDER SCRIPT


1
LOOP THROUGH EACH SLIDER
ANONYMOUS FUNCTION:
Create slider for this set of markup
2
Store in variables: $this: current slider, $group: slides container, $slides: all slides, buttonArray: buttons, currentIndex: current slide, timeout: stores the timer
7
LOOP THROUGH EACH SLIDE
ANONYMOUS FUNCTION:
Create button for each slide
8
Create a button for this item
9
N
Is this the current slide?
Y
Add class: active
10
Event: click on this radio element
11
Add button to container & array
Call move() (see p520)
GO TO NEXT SLIDE
12
Call advance() function
4
FUNCTION: advance()
Clear and reset the timer
5
Call clearTimeout() & setTimeout()
6
N
Is this the last slide?
Y
Call move() to next slide
Call move() to first slide
GO TO NEXT SLIDER

1. There may be several sliders on a page, so the script starts by looking for every element whose class attribute has a value of slider. For each one, an anonymous function is run to process that slider.

2. Variables are created to hold:

i) The current slider

ii) The element that wraps around the slides

iii) All of the slides in this slider

iv) An array of buttons (one for each slide)

v) The current slide

vi) The timer

3. The move() function appears next; see p520. **Please note:** This is not shown in the flowchart.

4. The advance() function creates the timer.

5. It starts by clearing the current timer. A new timer is set and when the time has elapsed it will run an anonymous function.

6. An if statement checks whether or not the current slide is the last one.

If it is not the last slide then it calls move() with a parameter that tells it to go to the next slide. Otherwise it tells move() to go to the first slide.

7. Each slide is processed by an anonymous function.

8. A <button> element is created for each slide.

9. If the index number of that slide is the same as the number held in the currentIndex variable, then a class of active is added to that button.

10. An event handler is added to each button. When clicked it calls the move() function. The slide's index number indicates which slide to move to.

11. The buttons are then added to the button container, and to the array of buttons.

This array is used by the move() function to indicate which slide is currently being shown.

12. advance() is called to start the timer.

JAVASCRIPT c11/js/slider.js

```
①  $('.slider').each(function(){                   // For every slider
     var $this   = $(this),                          // Get the current slider
     var $group  = $this.find('.slide-group'),       // Get the slide-group (container)
     var $slides = $this.find('.slide'),             // jQuery object to hold all slides
②    var buttonArray  = [],                          // Create array to hold nav buttons
     var currentIndex = 0,                           // Index number of current slide
     var timeout;                                    // Used to store the timer

③    // move() - The function to move the slides goes here (see next page)

④    function advance() {                            // Sets a timer between slides
       clearTimeout(timeout);                        // Clear timer stored in timeout
⑤      // Start timer to run an anonymous function every 4 seconds
       timeout = setTimeout(function(){              //
         if (currentIndex < ($slides.length - 1)) {  // If not the last slide
           move(currentIndex + 1);                   // Move to next slide
⑥        } else {                                    // Otherwise
           move(0);                                  // Move to the first slide
         }
       }, 4000);                                     // Milliseconds timer will wait
     }

⑦    $.each($slides, function(index){
       // Create a button element for the button
⑧      var $button = $('<button type="button" class="slide-btn">&bull;</button>');
       if (index === currentIndex) {      // If index is the current item
⑨        $button.addClass('active');      // Add the active class
       }
⑩      $button.on('click', function(){    // Create event handler for the button
         move(index);                     // It calls the move() function
⑪      }).appendTo('.slide-buttons');     // Add to the buttons holder
       buttonArray.push($button);         // Add it to the button array
     });

⑫    advance();

   });
```

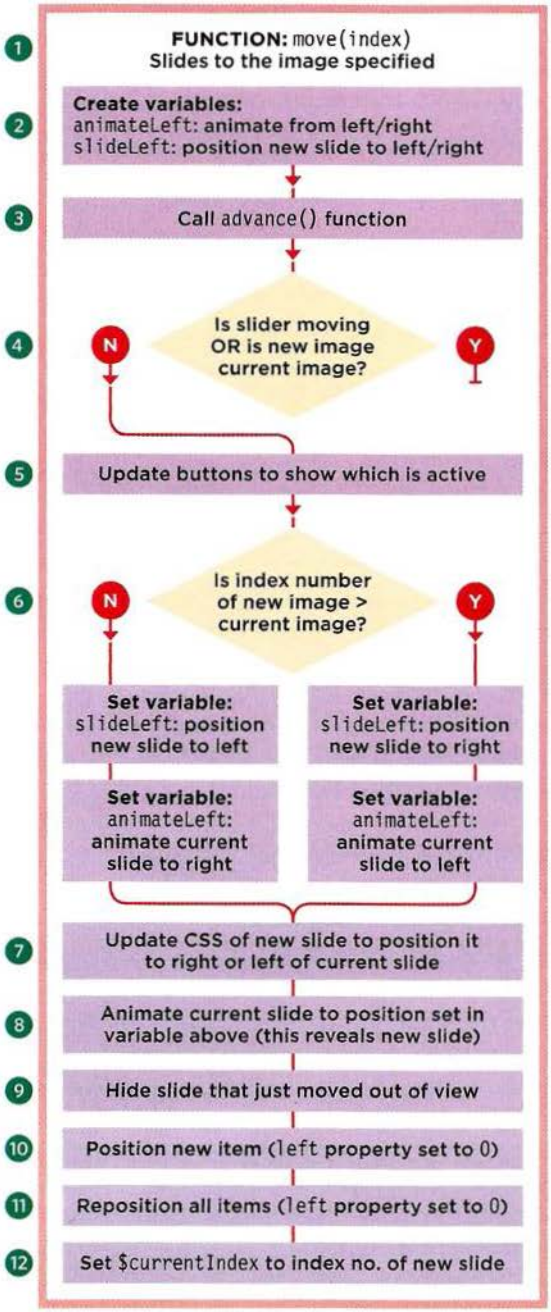
### PROBLEM: GETTING THE RIGHT GAP BETWEEN SLIDES USING A TIMER

Each slide should show for four seconds (before the timer moves it on to the next slide). But if the user clicks a button after two seconds, then the new slide might not show for four seconds because the timer is already counting down.

### SOLUTION: RESET THE TIMER WHENEVER A BUTTON IS CLICKED

The advance() function clears the timer before setting it off again. Every time the user clicks on a button the move() function (shown on the next two pages) it calls advance() to ensure the new slide is shown for four seconds.

# SLIDER MOVE() FUNCTION



1
FUNCTION: move(index)
Slides to the image specified
2
Create variables:
animateLeft: animate from left/right
slideLeft: position new slide to left/right
3
Call advance() function
4
Is slider moving OR is new image current image?
N
Y
5
Update buttons to show which is active
6
Is index number of new image > current image?
N
Y
Set variable: slideLeft: position new slide to left
Set variable: slideLeft: position new slide to right
Set variable: animateLeft: animate current slide to right
Set variable: animateLeft: animate current slide to left
7
Update CSS of new slide to position it to right or left of current slide
8
Animate current slide to position set in variable above (this reveals new slide)
9
Hide slide that just moved out of view
10
Position new item (left property set to 0)
11
Reposition all items (left property set to 0)
12
Set $currentIndex to index no. of new slide

1. The `move()` function will create the animated sliding movement between two slides. When it is called, it needs to be told which slide to move to.

2. Two variables are created that are used to control whether the slider is moving to the left or right.

3. `advance()` is called to reset the timer.

4. The script checks if the slider is currently animating or if the user selected the current slide. In either case, nothing should be done, and the `return` statement stops the rest of the code from running.

5. References to each of the buttons were stored in an array in step 11 of the script on the previous page. The array is used to update which button is active.

6. If the new item has a higher index number, then the slider will need to move from right to left. If the item has a lower index number, the slider will need to move from left to right. These variable values are set first and are then used in step 7.

`slideLeft` positions the new slide in relation to the current slide. (`100%` sits the new slide to the right of it and `-100%` sits the new slide to the left of it.)

`animateLeft` indicates whether the current slide should move to the left or the right, letting the new slide take its place. (`-100%` moves the current slide to the left, `100%` moves the current slide to the right.)

7. The new slide is positioned to the right or the left of the current slide using the value in the `slideLeft` variable and its `display` property is set to `block` so that it becomes visible. That new slide is identified using `newIndex`, which was passed into the function.

8. The current slide is then moved to the left or right using the value stored in the `animateLeft` variable. That slide is selected using the `currentIndex` variable, which was defined at the start of the script.

JAVASCRIPT

c11/js/slider.js

```
    // Setup of the script shown on the previous page

①    function move(newIndex) {          // Creates the slide from old to new one
②      var animateLeft, slideLeft;      // Declare variables

③      advance();                       // When slide moves, call advance() again

       // If current slide is showing or a slide is animating, then do nothing
④      if ($group.is(':animated') || currentIndex === newIndex) {
         return;
       }

⑤      buttonArray[currentIndex].removeClass('active'); // Remove class from item
       buttonArray[newIndex].addClass('active');        // Add class to new item

⑥      if (newIndex > currentIndex) {   // If new item > current
         slideLeft = '100%';            // Sit the new slide to the right
         animateLeft = '-100%';         // Animate the current group to the left
       } else {                         // Otherwise
         slideLeft = '-100%';           // Sit the new slide to the left
         animateLeft = '100%';          // Animate the current group to the right
       }
       // Position new slide to left (if less) or right (if more) of current
⑦      $slides.eq(newIndex).css( {left: slideLeft, display: 'block'} );
⑧      $group.animate( {left: animateLeft} , function() {   // Animate slides and
⑨        $slides.eq(currentIndex).css( {display: 'none'} ); // Hide previous slide
⑩        $slides.eq(newIndex).css( {left: 0} ); // Set position of the new item
⑪        $group.css( {left: 0} );               // Set position of group of slides
⑫        currentIndex = newIndex;               // Set currentIndex to new image
       });
     }

    // Handling the slides shown on p519
```

Once the slide has finished animating, an anonymous function performs housekeeping tasks:

**9.** The slide that was the `currentIndex` is hidden.

**10.** The position of the left-hand side of the new slide is set to 0 (left-aligning it).

**11.** The position of all of the other slides is set so the left-hand side is 0 (left-aligning them).

**12.** At this point, the new slide will be visible, and the transition is complete, so it is time to update the `currentIndex` variable to hold the index number of the slide that has just been shown. This is easily done by giving it the value that was stored in the `newIndex` variable.

Now that this function has been defined, as you saw on the p519, the code creates a timer and goes through each slide adding a button and an event handler for it. (Steps 4-12 on the page p519.)

# CREATING A JQUERY PLUGIN

## jQuery plugins allow you to add new methods to jQuery without customizing the library itself.

jQuery plugins have benefits over plain scripts:

- You can perform the same task on any elements that match jQuery's flexible selector syntax
- Once the plugin has done its job, you can chain other methods after it (on the same selection)
- They facilitate re-use of code (either within one project or across multiple projects)
- They are commonly shared within the JavaScript and jQuery community
- Namespace collisions (problems when two scripts use the same variable name) are prevented by placing the script is placed in an IIFE (immediately invoked function expression, which you met on p97)

You can turn any function into a plugin if it:

- Manipulates a jQuery selection
- Can return a jQuery selection

The basic concept is that you:

- Pass it a set of DOM elements in a jQuery selection
- Manipulate the DOM elements using the jQuery plugin code
- Return the jQuery object so that other functions can be chained off it

This final example shows you how to create a jQuery plugin. It takes the accordion example you saw at the start of the chapter and turns it into a plugin.

The earlier version applied to all matching markup on the page; the plugin version requires that users call the `accordion()` method on a jQuery selection.

Here a jQuery selection is made collecting elements with a class of `menu`. The `.accordion()` method is called; once that has run, `.fadeIn()` is called.

```
$('.menu').accordion(500).fadeIn();
└────────┘└──────────────┘└────────┘
    ①             ②            ③
```

1. A jQuery selection is made containing any elements which have the class of `menu`.

2. The `.accordion()` method is called on those elements. It has one parameter; the speed of animation (in milliseconds).

3. The `.fadeIn()` method is applied to the same selection of elements once `.accordion()` has done its job.

# BASIC PLUGIN STRUCTURE

## 1) ADDING A METHOD TO JQUERY

jQuery has an object called .fn which helps you extend the functionality of jQuery.

Plugins are written as methods that are added to the .fn object.

Parameters that can be passed to the function are placed inside the parentheses on the first line:

```
$.fn.accordion = function(speed) {
  // Plugin will go here
}
```

## 2) RETURNING THE JQUERY SELECTION TO CHAIN METHODS

jQuery works by collecting a set of elements and storing them in a jQuery object. The jQuery object's methods can be used to alter the selected elements.

Because jQuery lets you chain multiple methods to the same selection, once the plugin has done its job it should return the selection for the next method.

The selection is returned using:

1. The return keyword (sends a value back from a function)
2. this (refers to the selection that was passed in)

```
$.fn.accordion = function(speed) {
  // Plugin will go here
  return this;
}
```

## 3) PROTECTING THE NAMESPACE

jQuery is not the only JavaScript library to use $ as a shorthand, so the plugin code lives in an IIFE, which creates function-level scope for the code in the plugin.

On the first line below, the IIFE has one named parameter: $. On the last line, you can see that the jQuery selection is passed into the function.

Inside the plugin, $ acts like a variable name. It references the jQuery object containing the set of elements that the plugin is supposed to be working with.

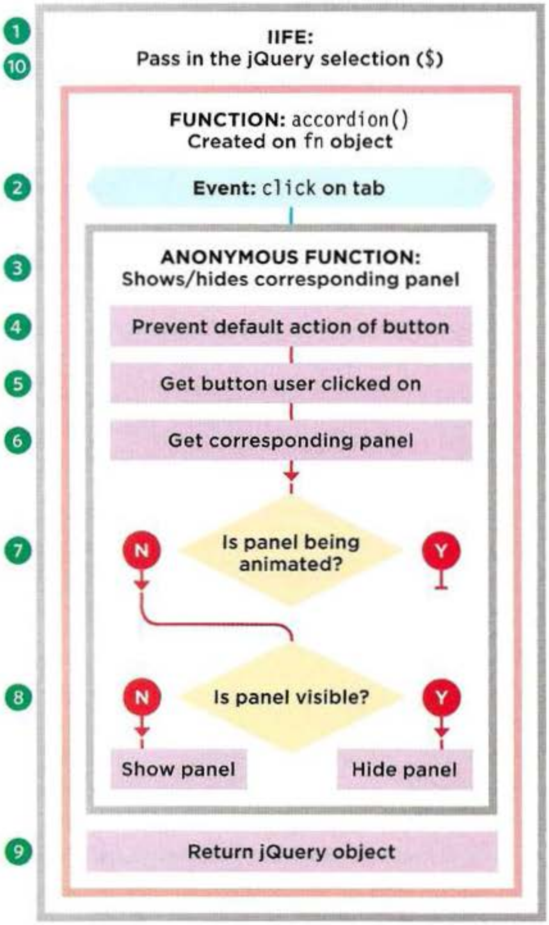
```
(function($){
  $.fn.accordion = function(speed) {
    // Plugin code will go here
  }
})(jQuery);
```

If you want to pass in more values, it is typically done using a single parameter called options.

When the function is called, the options parameter contains an object literal.

The object can contain a set of key/value pairs for the different options.

# THE ACCORDION PLUGIN



To use the plugin, you create a jQuery selection that contains any `<ul>` elements that hold an accordion. In the example on the right, the accordion is in a `<ul>` element that has a class name of menu (but you could use any name you wish). You then call the `.accordion()` method on that selection, like so:

```
$('.menu').accordion(500);
```

This code could be placed in the HTML document (as shown on the right-hand page), but it would be better placed in a separate JavaScript file that runs when the page loads (to keep the JavaScript separate from the HTML).

You can see the full code for the accordion plugin on the right. The parts in orange are identical to the accordion script at the start of the chapter.

1. The plugin is wrapped in an IIFE to create function-level scope. On the first line, the function is given one named parameter: $ (which means you can use the $ shortcut for jQuery in the function).
10. On the last line of code, the jQuery object is passed into the function (using its full name `jQuery` rather than its shortcut $). This jQuery object contains the selection of elements that the plugin is working with. Together, points 1 and 10 mean that in the IIFE, $ refers to the `jQuery` object and it will not be affected if other scripts use $ as a shorthand, too.

2. Inside the IIFE, the new `.accordion()` method is created by extending the `fn` object. It takes the one parameter of `speed`.

3. The `this` keyword refers to the jQuery selection that was passed into the plugin. It is used to create an event handler that will listen for when the user clicks on an element with a `class` attribute whose value is `accordion-control`. When the user does, the anonymous function runs to animate the corresponding panel into or out of view.

4. The default action of the link is prevented.
5. In the anonymous function, `$(this)` refers to a jQuery object containing the element that the user clicked upon.
6. 7. 8. The only difference between this anonymous function and the one used in the example at the start of the chapter is that the `.slideToggle()` method takes a parameter of `speed` to indicate how fast the panel should be shown or hidden. (It is specified when the `.accordion()` method is called.)

9. When the anonymous function has done its work, the `jQuery` object containing the selected elements is returned from the function, allowing the same set of elements to be passed to another jQuery method.

JAVASCRIPT

c11/js/accordion-plugin.js

```
① (function($){                                   // Use $ as variable name
②   $.fn.accordion = function(speed) {           // Return the jQuery selection
③     this.on('click', '.accordion-control', function(e){
④         e.preventDefault();
⑤         $(this)
⑥             .next('.accordion-panel')
⑦             .not(':animated')
⑧             .slideToggle(speed);
      });
⑨     return this;                                // Return the jQuery selection
    }
⑩ })(jQuery);                                     // Pass in jQuery object
```

Note how the filename for the jQuery plugin starts with `jquery.` to indicate that this script relies upon jQuery.

After the accordion plugin script has been included, the `accordion()` method can be used on any jQuery selection.

Below you can see the HTML for the accordion. This time it includes both the jQuery script and the jQuery accordion script.

HTML

c11/accordion-plugin.html

```
<ul class="menu">
  <li>
    <a href="#" class="accordion-control"><h3>Classics</h3></a>
    <div class="accordion-panel">If you like your flavors traditional...</div>
  </li>
  <li>
    <a href="#" class="accordion-control"><h3>The Flower Series</h3></a>
    <div class="accordion-panel">Take your tastebuds for a gentle...</div>
  </li>
  <li>
    <a href="#" class='accordion-control"><h3>Salt o' the Sea</h3></a>
    <div class="accordion-panel">Ahoy! If you long for a taste of...</div>
  </li>
</ul>
<script src="js/jquery.js"></script>
<script src="js/jquery.accordion.js"></script>
<script>
  $('.menu').accordion(500);
</script>
```

# SUMMARY

## CONTENT PANELS

- Content panels offer ways to show more content within a limited area.
- Popular types of content panels include accordions, tabs, photo viewers, modal windows, and sliders.
- As with all website code, it is advisable to separate content (HTML), presentation (CSS), and behavior (JavaScript) into different files.
- You can create objects to represent the functionality you want (as with the modal window).
- You can turn functions into jQuery plugins that allow you to re-use code and share it with others.
- Immediately invoked function expressions (IIFEs) are used to contain scope and prevent naming collisions.