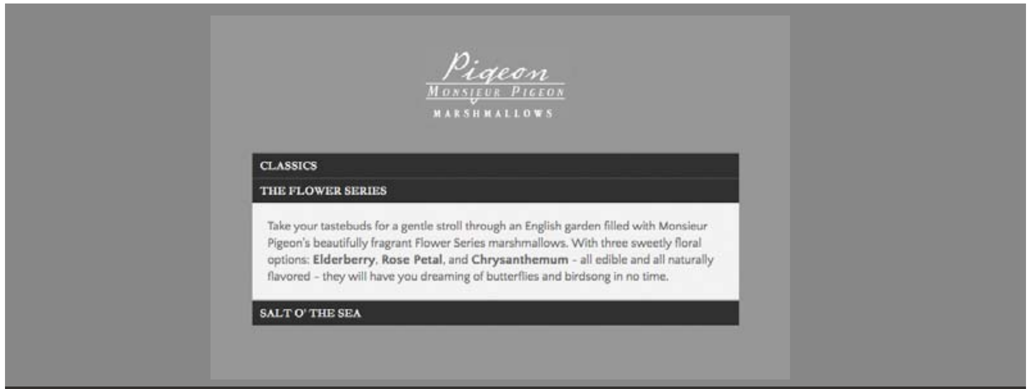
JAVASCRIPT
& JQUERY
interactive front-end
web development

CHAPTER 11

CONTENT PANELS

Content panels let you showcase extra information in a limited amount of space.


Pigeon
Monsieur Pigeon
MARSHMALLOWS

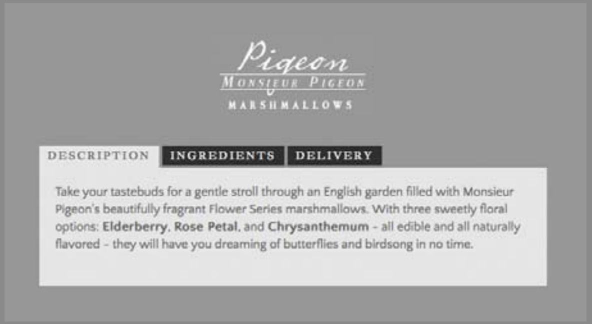
CLASSICS

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.

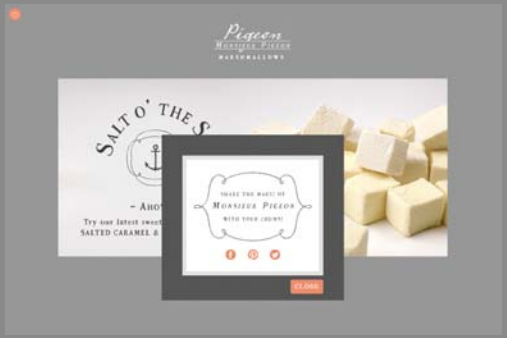
SALT O' THE SEA

Accordions feature titles which, when clicked, expand to show a larger panel of content.

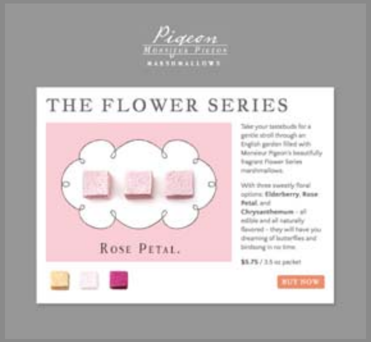


Take your tastebuds for a gentle stroll through an English garden filled with Monsieur Pigeon's beautifully fragrant Flower Series marshmallows. With three sweetly floral options: **Elderberry**, **Rose Petal**, and **Chrysanthemum** – all edible and all naturally flavored – they will have you dreaming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

Tabbed panels automatically show one panel, but when you click on another tab, switch to showing a different panel.



Modal windows (or ‘lightboxes') display a hidden panel on top of the page content when their links are activated.



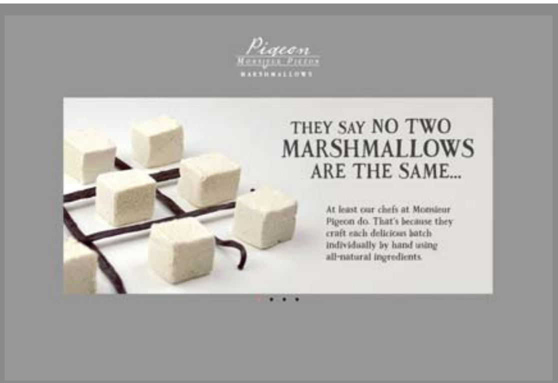
THE FLOWER SERIES

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

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



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.

Sliders show multiple panels of content that slide into view as the user navigates between them.

When creating content panels, remember to maintain a separation of concerns:

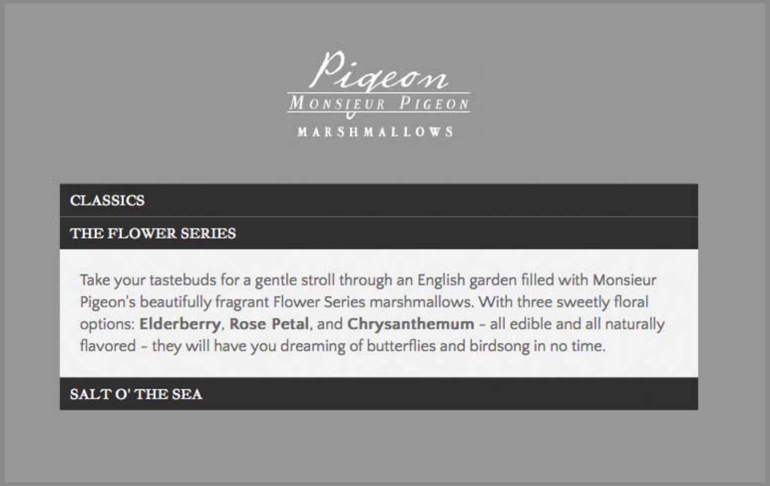
Content in HTML file
Presentation in CSS rules
Behaviors in JavaScript

It is also important to keep your code accessible:

If users can interact with an element, use `<a>` or a button.
Make sure content is available if JavaScript is disabled.

ACCORDION

When the user clicks on a label, an anonymous function gets the label the user clicked on. It selects the panel after it and either shows or hides it.



Pigeon
MONSIEUR PIGEON
MARSHMALLOWS

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

ACCORDIAN WITH ALL PANELS COLLAPSED

LABEL 1

LABEL 2

LABEL 3

ACCORDIAN WITH FIRST PANEL EXPANDED

LABEL 1

CONTENT 1

LABEL 2

LABEL 3

ACCORDIAN WITH SECOND PANEL EXPANDED

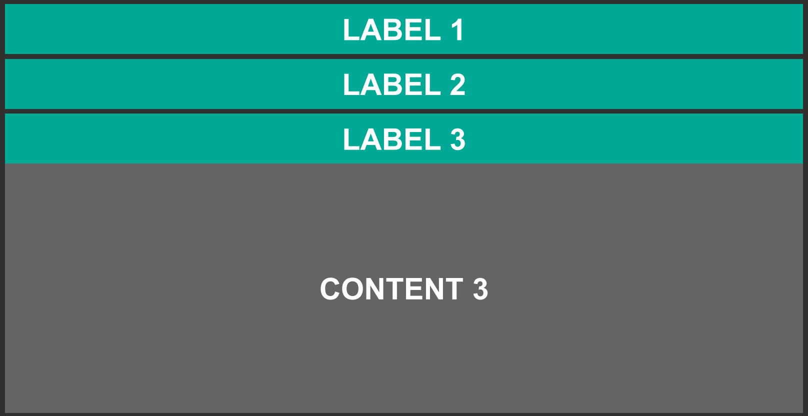
LABEL 1

LABEL 2

CONTENT 2

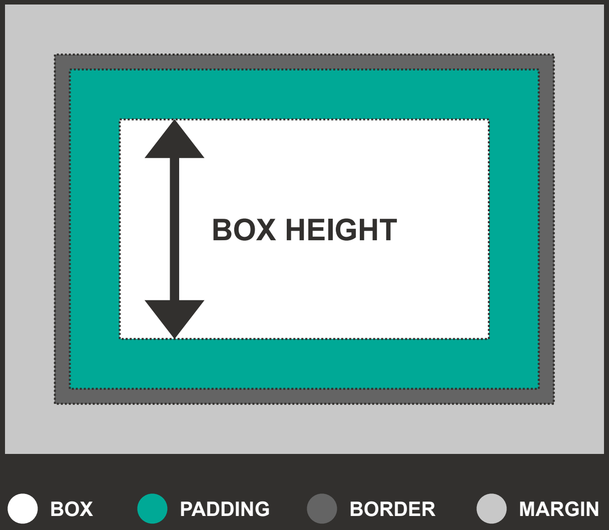
LABEL 3

ACCORDIAN WITH THIRD PANEL EXPANDED



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

They also calculate the size of the box including its content and any margins and padding.



HTML specifies the structure:

```
<ul class="accordion">

  <li>
    <button class="accordion-control">
      Label 1
    </button>
    <div class="accordion-panel">
      <!-- Content goes here -->
    </div>
  </li>

</ul>
```

CSS hides the panels:

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

jQuery handles the `click` event:

```
$('.accordion').on('click', 'accordion-control', function(e) {
  e.preventDefault();
  $(this)
    .next('.accordion-panel')
    .not('animated')
    .slideToggle();
});
```

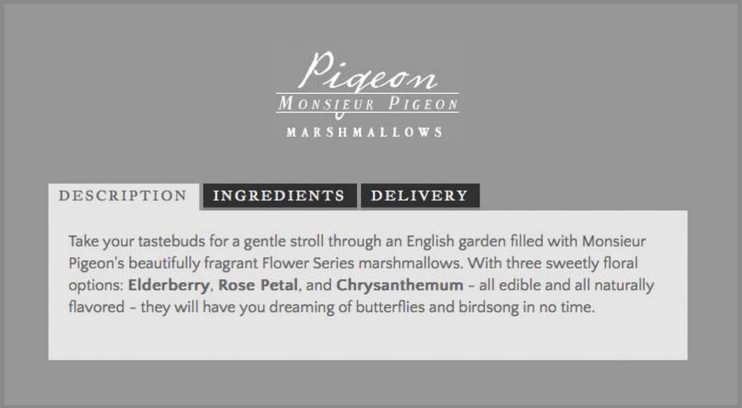
The default action of the link is stopped:

```
$('.accordion').on('click', 'accordion-control', function(e) {
  e.preventDefault();
  $(this)
    .next('.accordion-panel')
    .not('animated')
    .slideToggle();
});
```

Open or close panels:

```
$('.accordion').on('click', 'accordion-control', function(e) {
  e.preventDefault();
  $(this)
    .next('.accordion-panel')
    .not('animated')
    .slideToggle();
});
```

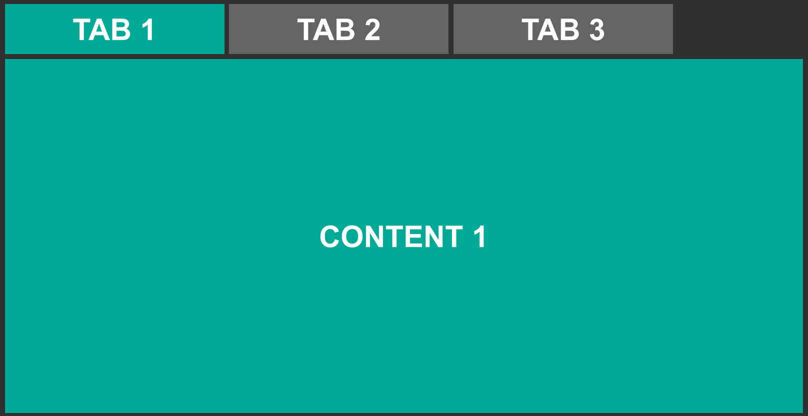
TABBED PANEL



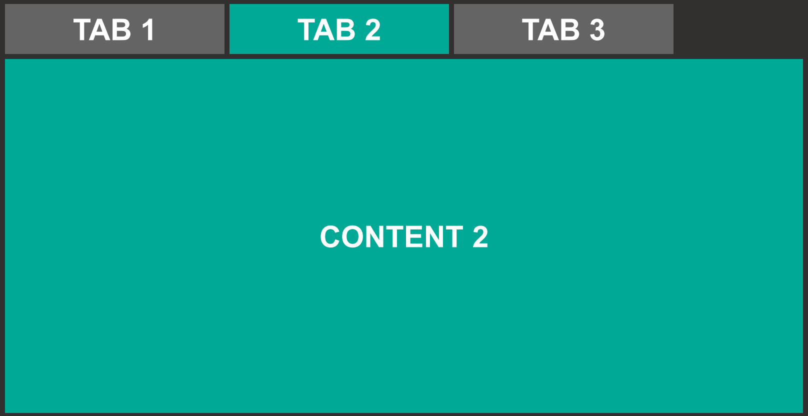
Take your tastebuds for a gentle stroll through an English garden filled with Monsieur Pigeon's beautifully fragrant Flower Series marshmallows. With three sweetly floral options: **Elderberry**, **Rose Petal**, and **Chrysanthemum** - all edible and all naturally flavored - they will have you dreaming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

Tabs have a similar concept but only one panel is shown at a time.

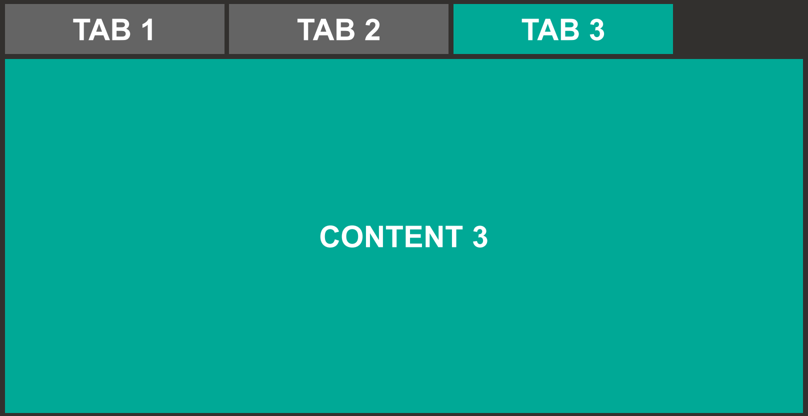
FIRST TAB SELECTED



SECOND TAB SELECTED



THIRD TAB SELECTED



HTML specifies the structure. Here are the tabs:

```
<ul class="tab-list">

  <li class="active">
    <a class="tab-control" href="#tab1">Tab 1</a>
  </li>

  <li>
    <a class="tab-control" href="#tab2">Tab 2</a>
  </li>

  <li>
    <a class="tab-control" href="#tab3">Tab 3</a>
  </li>

</ul>
```

The panels follow the tabs:

```
<div class="tab-panel active" id="tab-1">
  <!-- Content for the first panel goes here -->
</div>

<div class="tab-panel" id="tab-2">
  <!-- Content for the second panel goes here -->
</div>

<div class="tab-panel" id="tab-3">
  <!-- Content for the third panel goes here -->
</div>
```

CSS hides all of the panels except for the active one:

```
.tab-panel {
  display: none;}

.tab-panel.active {
  display: block;}
```

The same code is run for each set of tabs:

```
$(‘.tab-list').each(function() {          // Find lists of tabs

  var $this = $(this),                    // Store this list
  var $tab = $this.find('li.active'),     // Get the active li
  var $link = $tab.find('a'),             // Get its link
  var $panel = $($link.attr('href'));     // Get active panel

  $this.on('click', '.tab-control', function(e) { // Click tab

    e.preventDefault();                   // Prevent link
    var $link = $(this);                  // Store current link
    var id = this.hash;                   // Get clicked tab

    if (id && !$link.is('.active')) {     // If not active
      $panel.removeClass('active');       // Make panel and
      $tab.removeClass('active');         // tab inactive
      $panel = $(id).addClass('active'); // Make new panel and
      $tab = $link.parent().addClass('active'); // tab active
    }
  });
});
```

The variables are set:

```
$(‘.tab-list').each(function() {          // Find lists of tabs

  var $this = $(this),                    // Store this list
  var $tab = $this.find('li.active'),     // Get the active li
  var $link = $tab.find('a'),             // Get its link
  var $panel = $($link.attr('href'));     // Get active panel

  $this.on('click', '.tab-control', function(e) { // Click tab

    e.preventDefault();                   // Prevent link
    var $link = $(this);                  // Store current link
    var id = this.hash;                   // Get clicked tab

    if (id && !$link.is('.active')) {     // If not active
      $panel.removeClass('active');       // Make panel and
      $tab.removeClass('active');         // tab inactive
      $panel = $(id).addClass('active'); // Make new panel and
      $tab = $link.parent().addClass('active'); // tab active
    }
  });
});
```

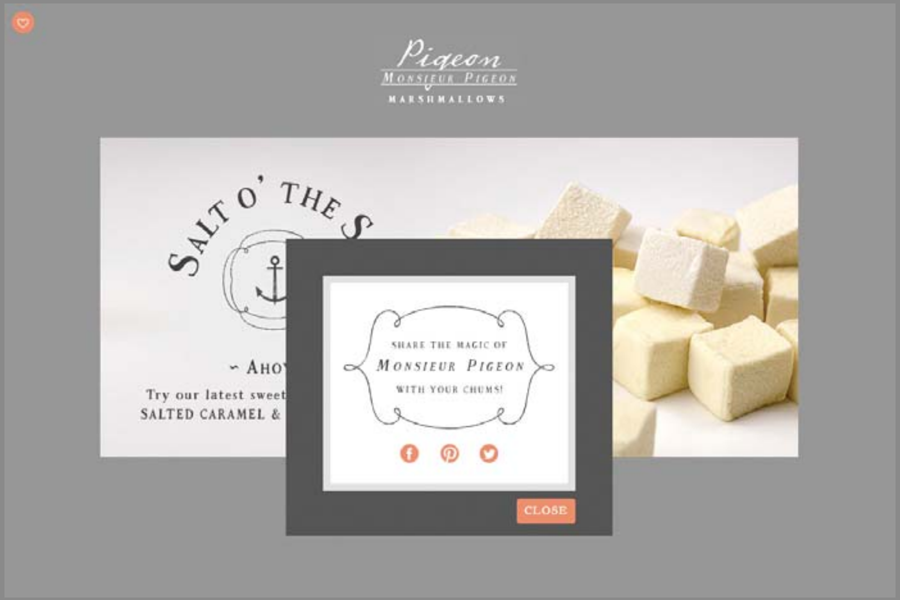
The default action of the links are stopped:

```
$(‘.tab-list').each(function() {          // Find lists of tabs

  var $this = $(this),                    // Store this list
  var $tab = $this.find('li.active'),     // Get the active li
  var $link = $tab.find('a'),             // Get its link
  var $panel = $($link.attr('href'));     // Get active panel

  $this.on('click', '.tab-control', function(e) { // Click tab

    e.preventDefault();                   // Prevent link
    var $link = $(this);                  // Store current link
    var id = this.hash;                   // Get clicked tab

    if (id && !$link.is('.active')) {     // If not active
      $panel.removeClass('active');       // Make panel and
      $tab.removeClass('active');         // tab inactive
      $panel = $(id).addClass('active'); // Make new panel and
      $tab = $link.parent().addClass('active'); // tab active
    }
  });
});
```

The panels are shown or hidden:

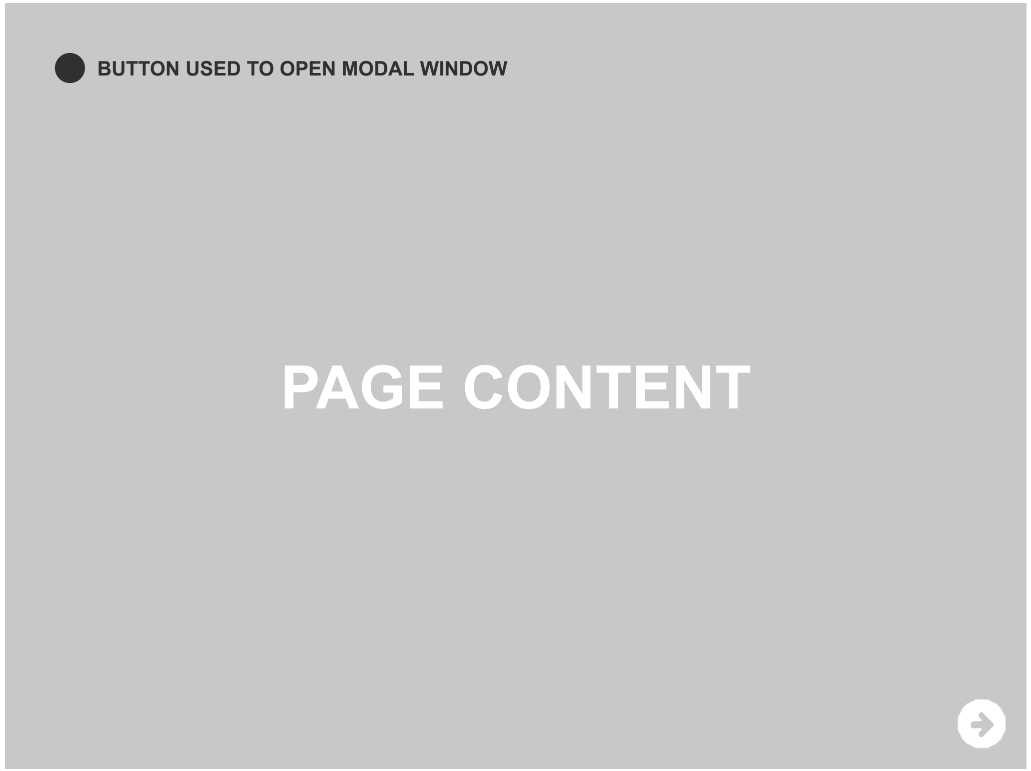
```
$('.tab-list').each(function() {          // Find lists of tabs
  var $this = $(this),                    // Store this list
  var $tab = $this.find('li.active'),     // Get the active li
  var $link = $tab.find('a'),             // Get its link
  var $panel = $($link.attr('href'));     // Get active panel
  $this.on('click', '.tab-control', function(e) { // Click tab
    e.preventDefault();                   // Prevent link
    var $link = $(this);                  // Store current link
    var id = this.hash;                   // Get clicked tab
    if (id && !$link.is('.active')) {     // If not active
      $panel.removeClass('active');       // Make panel and
      $tab.removeClass('active');         // tab inactive
      $panel = $(id).addClass('active'); // Make new panel and
      $tab = $link.parent().addClass('active'); // tab active
    }
  });
});
```

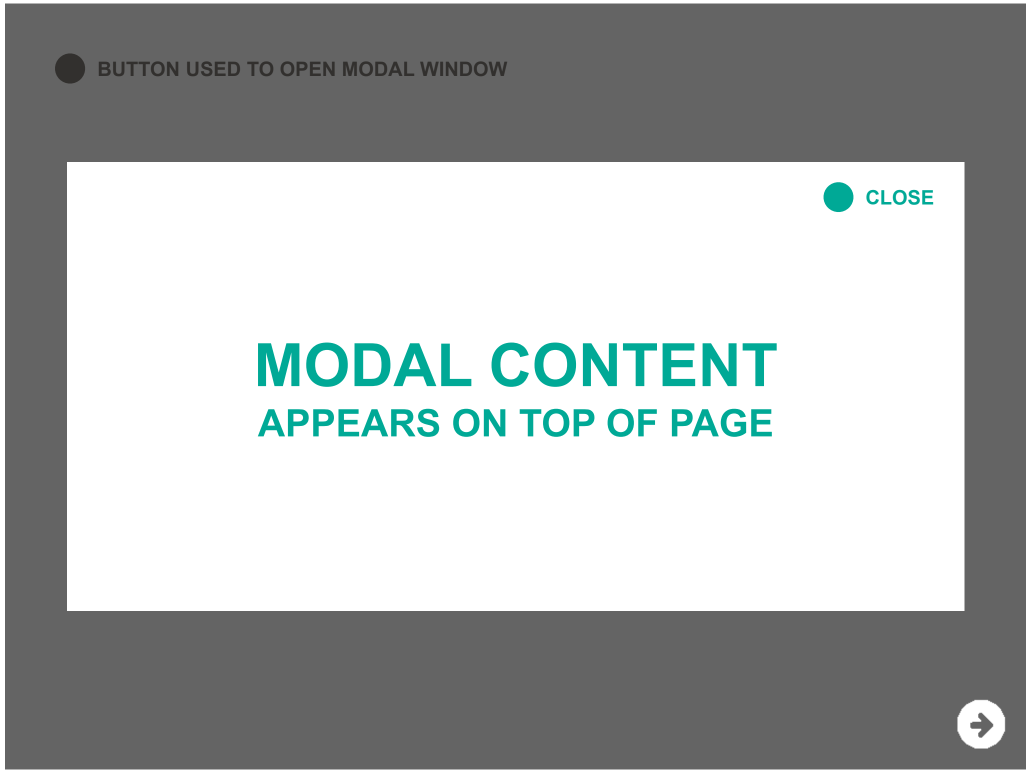
MODAL WINDOW



A modal window is content that appears "in front" of the rest of the page.

It must be closed before the rest of the page can be interacted with.





HTML specifies the structure:

```
<div class="modal">

  <div class="modal-content">

    <!- - Content goes here - ->

    <button class="modal-close">close</button>

  </div>

</div>
```

CSS positions the modal on top of all of the other content:

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

The JavaScript to create a modal window runs when the page loads.

Opening the modal window:

```
(function() {

  // Remove modal content from page and store in $content
  var $content = $('#share-options').detach();

  // Click handler calls open() method of modal object
  $('#share').on('click', function() {
    modal.open({content: $content, width:340, height:300});
  });

}());
```

The `modal` object is a custom object. It is created in another script and can be used on any page of the site.

The `modal` object's methods:

`open()`	open modal window
`close()`	close modal window
`center()`	center modal window on page

People who use the script to create a modal window only need to know how to call the `open()` method because the other methods are used by the script.

The `modal` object starts with variables only available within the object. They create the modal window and its close button:

```
var modal = (function() {

  var $window = $(window),
  var $modal = $('<div class="modal"/>'),
  var $content = $('<div class=“modal-content"/>'),
  var $close = $('<button role=“button”
      class=“modal-close">close</button>');

 $modal.append($content, $close);

}
```

If the user clicks on the close button, an event handler will close the modal window by calling the `close()` method:

```
$close.on('click', function(e) {
  e.preventDefault();
  modal.close();
});
```

The `return` statement will return the public methods:

```
$return {
  center: function() {
    // Distance from top and left to center of modal
    var top = Math.max($window.height() -
              $modal.outerHeight(), 0) / 2,
    var left = Math.max($window.width() -
               $modal.outerWidth(), 0) / 2;
    // Set CSS for the modal
    $modal.css({
        top: top + $window.scrollTop(),
        left: left + $window.scrollLeft()
    });
  }, // Other methods go here
}
```

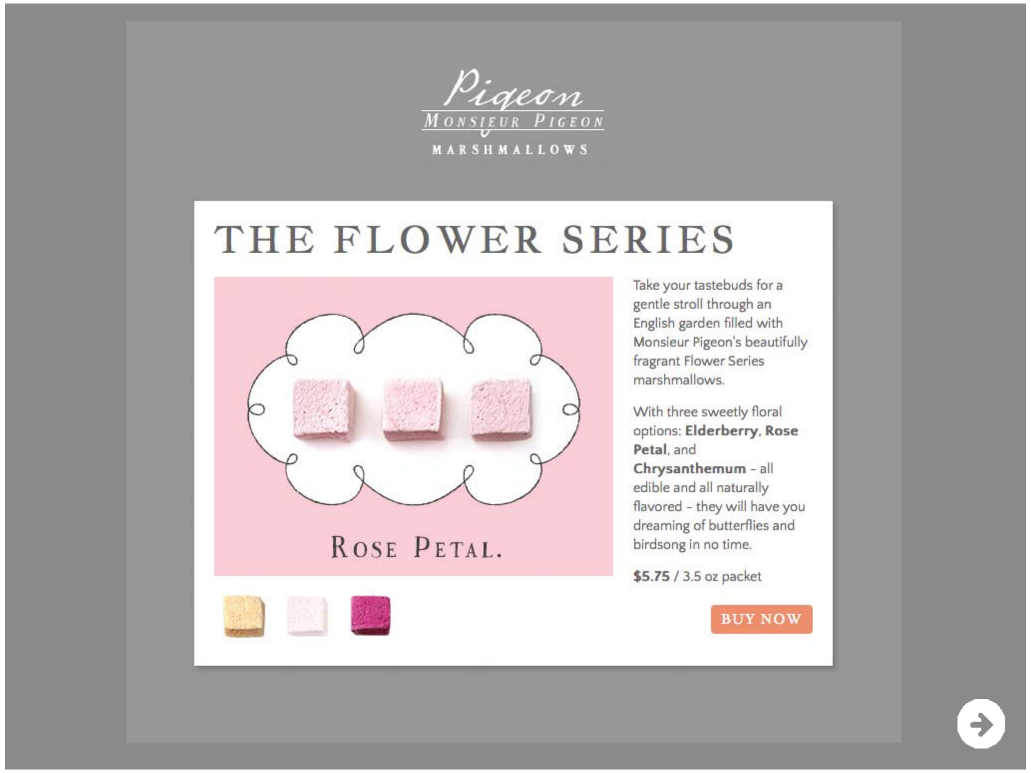
Anyone who uses the script only needs to use the `open()` method, which creates the modal window:

```
$open: function(settings) {

  $content.empty().append(settings.content);

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

  modal.center();                        // Call center() again
  $(window).on('resize', modal.center); // On window resize

},
```

The `close` method is used by the close button's event handler:

```
close: function() {

  // Remove content from the modal window
  $content.empty();

  // Remove modal window from the page
  $modal.detach();

  // Remove event handler
  $(window).off('resize', modal.center);

}
```

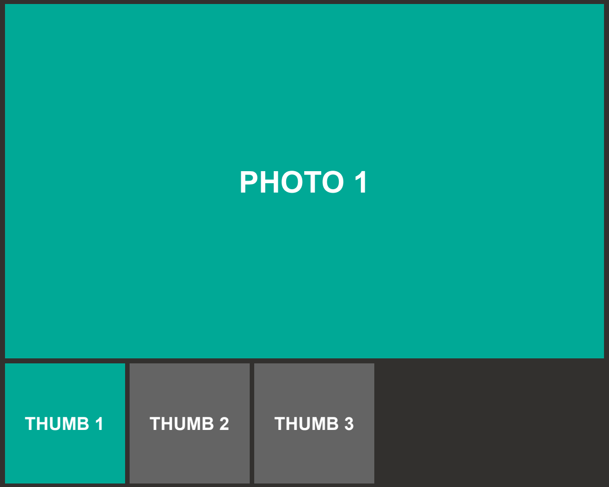
PHOTO VIEWER


Pigeon
Monsieur Pigeon
MARSHMALLOWS
THE FLOWER SERIES
Rose Petal.
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
BUY NOW

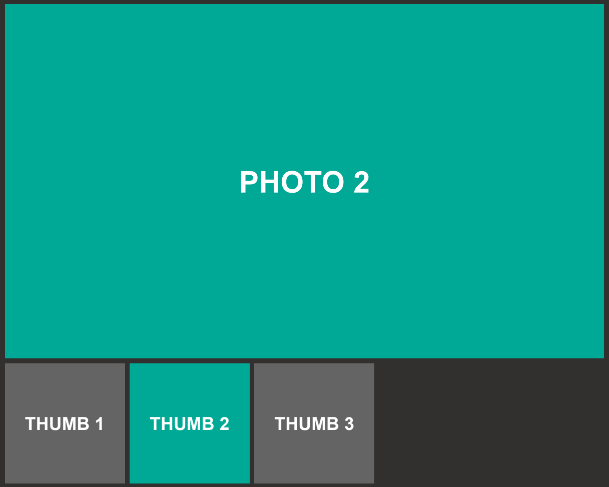
The photo viewer is an example of an image gallery.

When the user clicks on a thumbnail, the main photograph is updated.

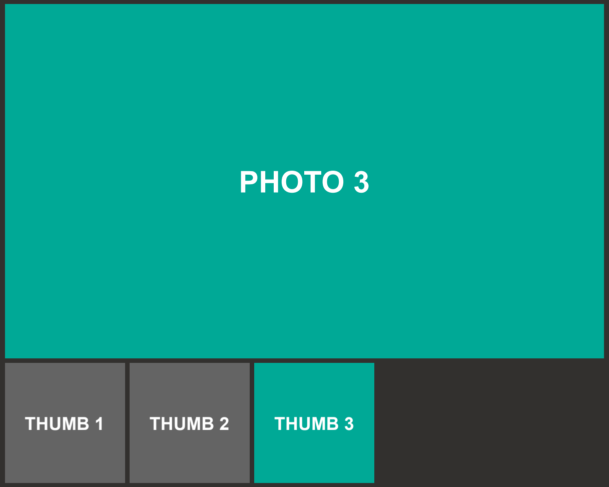
FIRST PHOTO SELECTED



SECOND PHOTO SELECTED



THIRD PHOTO SELECTED



HTML specifies the structure:

```
<div id=“photo-viewer”></div>

<div id=“thumbnails”>

  <a href=“img/photo-1.jpg” class=“active” title=“Berry”>
    <img src=“img/thumb-1.jpg” alt=“Berry”>
  </a>

  <a href=“img/photo-2.jpg” title=“Rose”>
    <img src=“img/thumb-2.jpg” alt=“Rose”>
  </a>

  <a href=“img/photo-3.jpg” title=“Mint”>
    <img src=“img/thumb-3.jpg” alt=“Mint”>
  </a>

</div>
```

CSS is used to show a loading GIF and position the images:

```
#photo-viewer.is-loading:after {
  content: url('../img/load.gif');
  position: absolute;
  top: 0;
  left: 0;}

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

Images load asynchronously.

PROBLEM:
If the user clicks on a large image and then a smaller image, the smaller one might show up first.

Images load asynchronously.

SOLUTION:
When an image has loaded, check to see if it was the last one to have been requested.

Images don't cache automatically.

PROBLEM:
If the user clicks on a large image, looks at another image, and then goes back, it creates a new element and goes through the loading process again.

Images don’t cache automatically.

SOLUTION:
Create an object called `cache`.
When a new `<img>` element is created, add it to the `cache`.
Check `cache` before showing images (if it is there, use that).

The `cache` object would look like this:

```
var cache = {

  "c11/img/photo-1.jpg" : {
    "$img": jquery object,
    "isLoading": false
  },

  "c11/img/photo-2.jpg" : {
    "$img": jquery object,
    "isLoading": false
  }

}
```

Start by creating variables:

```
var request;                              // Last image request
var $current;                             // Current image
var cache = {};                           // Cache object

var $frame = $('#photo-viewer');          // Container
var $thumbs = $('.thumb');                // Container
```

Cross-fade images:

```
function crossfade($img) {             // New image as parameter

  if ($current) {                      // If image showing
    $current.stop().fadeOut('slow'); // Stop animation & fade out
  }

  $img.css({                         // Set CSS margins for new img
    marginLeft: -$img.width() / 2, // Neg margin 1/2 image width
    marginTop: -$img.height() / 2  // Neg margin 1/2 image height
  });

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

  $current = $img;                   // New image is current one

}
```

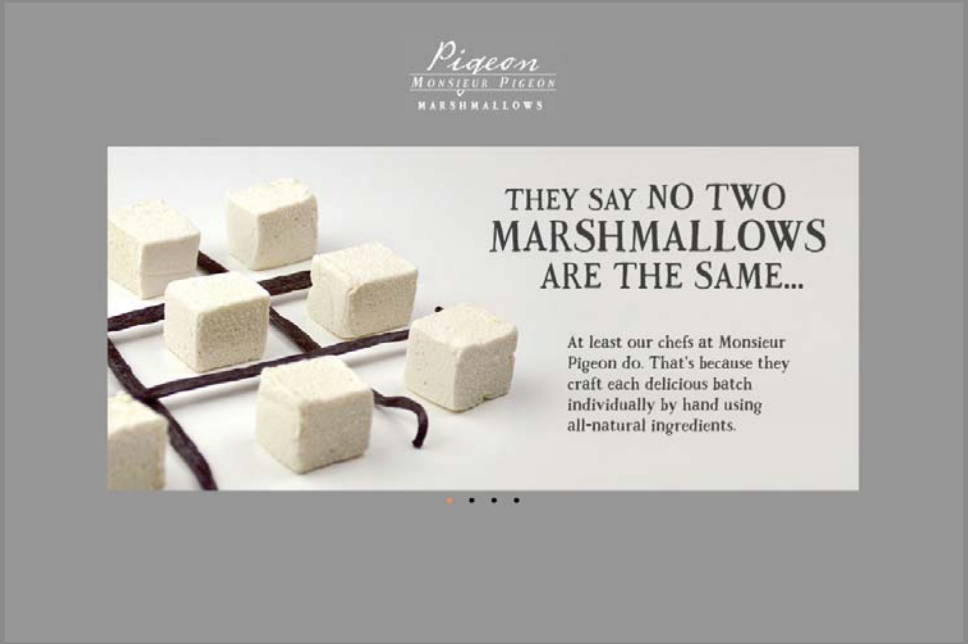
Set-up, cache, and loading image:

```
$(document).on('click', '.thumb', function(e) { // Click on thumb
  var $img;                                // Local var called $img
  var src = this.href;                     // Store path to image
  var request = src;                       // Store latest image
  e.preventDefault();               // Stop default link behavior
  $thumbs.removeClass(‘active');    // Remove active from thumbs
  $(this).addClass(‘active');       // Add active to clicked one
  if (cache.hasOwnProperty(src)) {  // If cache contains this img
    if (cache[src].isLoading === false) { // and it’s not loading
      crossfade(cache[src].$img);   // Call crossfade() function
    }
  } else {                    // Otherwise it is not in the cache
    $img = $(‘<img/>’);       // Store empty <img/> in $img
    cache[src] = {            // Store this image in cache
      $img: $img,             // Add the path to the image
      isLoading: true         // Set isLoading to false
    };
```

Set-up, cache, and loading image:

```
// When image has loaded this code runs
$img.on('load', function() {              // When image loaded
  $img.hide();                            // Hide it
  // Remove is-loading class & append image
  $frame.removeClass(‘is-loading’).append($img);
  cache[src].isLoading = false;   // Update isLoading in cache
  // If still most recently requested image then
  if (request === src) {
    crossfade($img);              // Call crossfade() function
  }                               // to solve async load issue
});
$frame.addClass('is-loading');    // Add is-loading to frame
$img.attr({                       // Set attributes on <img>
 'src': src,                      // src attribute loads image
 'alt': this.title || ''          // Add title if one given
});
```

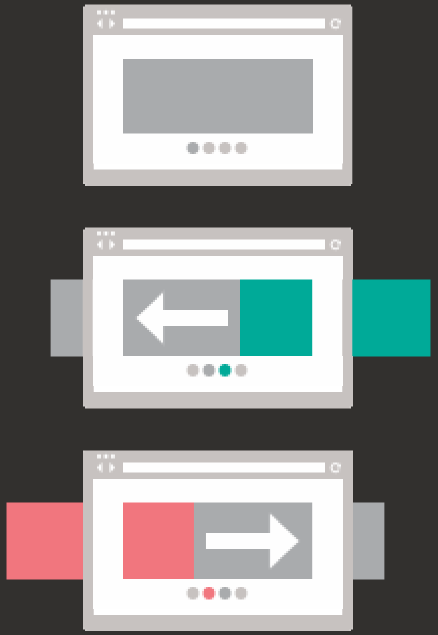
SLIDER



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

Sliders position a series of items next to each other, but only show one at a time.

The images then slide from one to the next.



HTML specifies the structure:

```
<div class=“slider">

  <div class=“slide-viewer">

    <div class=“slide-group">

      <div class="slide"><img src=“slide-1.jpg" /></div>
      <div class="slide"><img src=“slide-2.jpg" /></div>
      <div class="slide"><img src=“slide-3.jpg" /></div>
      <div class="slide"><img src=“slide-4.jpg" /></div>

    </div>

  </div>

  <div class=“slide-buttons"></div>

</div>
```

Slides are shown at the same height and width as the container:

```
.slide-viewer {
  position: relative;
  overflow: hidden;
  height: 430px;}
.slide-group {
  width: 100%;
  height: 100%;
  position: relative;}
.slide {
  width: 100%;
  height: 100%;
  display: none;
  position: absolute;}
.slide:first-child {
  display: block;}
```

Set up looping through each slider:

```
('.slider').each(function() {

  var $this    = $(this);                    // Current slider

  // Get the slide-group (container)
  var $group   = $this.find('.slide-group');

  // Create jQuery object to hold all slides
  var $slides = $this.find('.slide');
  var buttonArray  = [];                     // Array holds nav
  var currentIndex = 0;                      // Current slide
  var timeout;                               // Gap between slide

  // The move() function will go here
```

Moving the slides (part one):

```
function move(newIndex) {

  var animateLeft, slideLeft;             // Declare variables

  advance();                              // Call advance() again

  // If it is the current slide animating, do nothing
  if($group.is(':animated') || currentIndex === newIndex) {
    return;
  }

  // Remove active class from current slide button
  buttonArray[currentIndex].removeClass(‘active');

  // Add active class to new slide button
  buttonArray[newIndex].addClass('active');
```

Moving the slides (part two):

```
    if (newIndex > currentIndex) { // If new item > current
      slideLeft = '100%';          // Sit new slide to the right
      animateLeft = '-100%';       // Animate current group to left
    } else {                       // Otherwise
      slideLeft = '-100%';         // Sit the new slide to the left
      animateLeft = '100%';        // Animate current group right
    }

    // Position slide left (if less) right (if more) of current
    $slides.eq(newIndex).css( {left: slideLeft, display:
'block'} );
    $group.animate( {left: animateLeft}, function() {  // Animate
    $slides.eq(currentIndex).css( {display: 'none'} ); // Hide old
    $slides.eq(newIndex).css( {left: 0} ); // Set pos: new item
    $group.css( {left: 0} );               // Set pos: slide group
    currentIndex = newIndex;               // Set to new image

  });

}
```

The timer:

```
function advance() {                  // Set timer

  clearTimeout(timeout);              // Clear timeout

  // New timer
  timeout = setTimeout(function() {

    // If slide < total slides
    if (currentIndex < ($slides.length - 1)) {
      move(currentIndex + 1);         // Move slides
    } else {                          // Otherwise
      move(0);                        // Go to first slide
    }

  }, 4000);                           // Milliseconds timer waits

}
```

Buttons:

```
$.each($slides, function(index) {

  // Create button
  var $button = $('<button type="button"
                class="slide-btn">&bull;</button>');

  if (index === currentIndex) {          // Is it current item
    $button.addClass('active');          // Add active class
  }

  $button.on('click', function() {       // Add event handler
    move(index);                         // It calls the move()
  }).appendTo('.slide-buttons');         // Add to holder
  buttonArray.push($button);             // Add to array

});

advance();                               // Ready, move it
```

CREATING AN ACCORDION JQUERY PLUGIN

jQuery plugins add new methods to jQuery.

jQuery has a function called `fn` which you can use to extend jQuery:

```
$.fn.accordion = function(speed) {

  // Plugin code goes here

}
```

It returns the jQuery selection when it has finished running, so that other methods can be chained after it:

```
$.fn.accordion = function(speed) {

  // Plugin code goes here

  return this

}
```

The namespace:

```
(function($) {                          // $ as variable name
  $.fn.accordion = function (speed) {
    this.on('click', '.accordion-control', function (e) {
      e.preventDefault();
      $(this)
        .next('.accordion-panel')
        .not(':animated')
        .slideToggle(speed);
    });
    return this;                        // Return jQuery selection
  };
}(jQuery));                             // Pass in jQuery object
```