
JAVASCRIPT®
& JQUERY
interactive front-end
web development

Objects and Classes

WHAT IS AN OBJECT?

An object is a collection of properties and methods combined together to represent a single entity

We can define methods and properties and assign them to our objects

```
let whiteRabbit = {}

whiteRabbit.color = "white";
whiteRabbit.speak = function(line){
   console.log(`The ${this.color} rabbit says "${line}"`);
};

whiteRabbit.speak("I'm late!");
```

this is a keyword that allows us to access object properties from inside of a method

Arrow methods don’t have the same access to this

Instead, they use this from the surrounding (global) scope

```
// this makes sure out speak method doesn’t output undefined
this.color = "rainbow“;

const speak = line => {
   console.log(`The ${this.color} rabbit says "${line}"`);
};

let rabbit = {
   color: 'white’,
   speak: speak
};

rabbit.speak("I'm late and a different colour!");
```

CLASSES

A class defines the template of a type of object

Objects based on a class are called an instance of the class

The class keyword was introduced in ECMAScript 2015

Also known as ES2015 or ES6

Classes you define inherit properties and methods from a prototype object

This prototype object acts as a template for your object

Prototypes can be layered on top of each other

Object.prototype is the base prototype for most objects

```
console.log(
   Object.getPrototypeOf([]) === Array.prototype);

console.log(
   Object.getPrototypeOf(Array.prototype) ===
Object.prototype);

console.log(Object.getPrototypeOf(Object.prototype));
```

Some examples of default methods in Object.prototype include

constructor
hasOwnProperty
toString
valueOf

The constructor is the method that gets called when an object is instantiated

Using the new keyword allows you to instantiate an object and call the constructor

Once the object is instantiated, it can then be used

Properties can be get and set, and methods can be called that belong to the object

Before we had the class keyword, we made objects like this:

```
function Animal(type){
    this.type = type;
}

Animal.prototype.toString = function animalToString(){
    return "This animal is a " + this.type;
}

let fluffyRabbit = new Animal("Rabbit");
```

Objects were built using their constructor definitions

```
function Animal(type){
    this.type = type;
}

Animal.prototype.toString = function animalToString(){
    return "This animal is a " + this.type;
}

let fluffyRabbit = new Animal("Rabbit");
```

Creating an instance of the object was done using the new keyword before the name of the constructor

```
function Animal(type){
    this.type = type;
}

Animal.prototype.toString = function animalToString(){
    return "This animal is a " + this.type;
}

let fluffyRabbit = new Animal("Rabbit");
```

This notation was confusing, as object definitions looked like methods

Now with the class keyword, the object definition is clearer

```
class Animal{
    type; // property belonging to the class
    constructor(type){
        this.type = type;
    }
    toString(){
        return "This animal is a " + this.type;
    }
}

let rabbit = new Animal("Rabbit");
let rabbitString = `The animal says: ${rabbit}`;

// note: console.log doesn't invoke toString
console.log(rabbit);
console.log(rabbitString);
```

GETTERS AND SETTERS

We also have access to more functionality with this notation

Some examples of this are getters and setters

Using the keyword get, you can bind an object property to a method when it is being retrieved

```
class Temperature {
    constructor(celsius) {
        this.celsius = celsius;
    }
    get fahrenheit() {
        return this.celsius * 1.8 + 32;
    }
    set fahrenheit(value) {
        this.celsius = (value - 32) / 1.8;
    }
}
let temp = new Temperature(22);
console.log(temp.fahrenheit);
```

Similarly, the keyword set will allow you to bind an object property to a method when it is being set

```
class Temperature {
    constructor(celsius) {
        this.celsius = celsius;
    }
    get fahrenheit() {
        return this.celsius * 1.8 + 32;
    }
    set fahrenheit(value) {
        this.celsius = (value - 32) / 1.8;
    }
}
let temp = new Temperature(22);
temp.fahrenheit = 85;
console.log(temp.fahrenheit);
```

The in operator can tell us if an object has access to a property:

```
console.log('celsius’ in Temperature);
// will output true
```

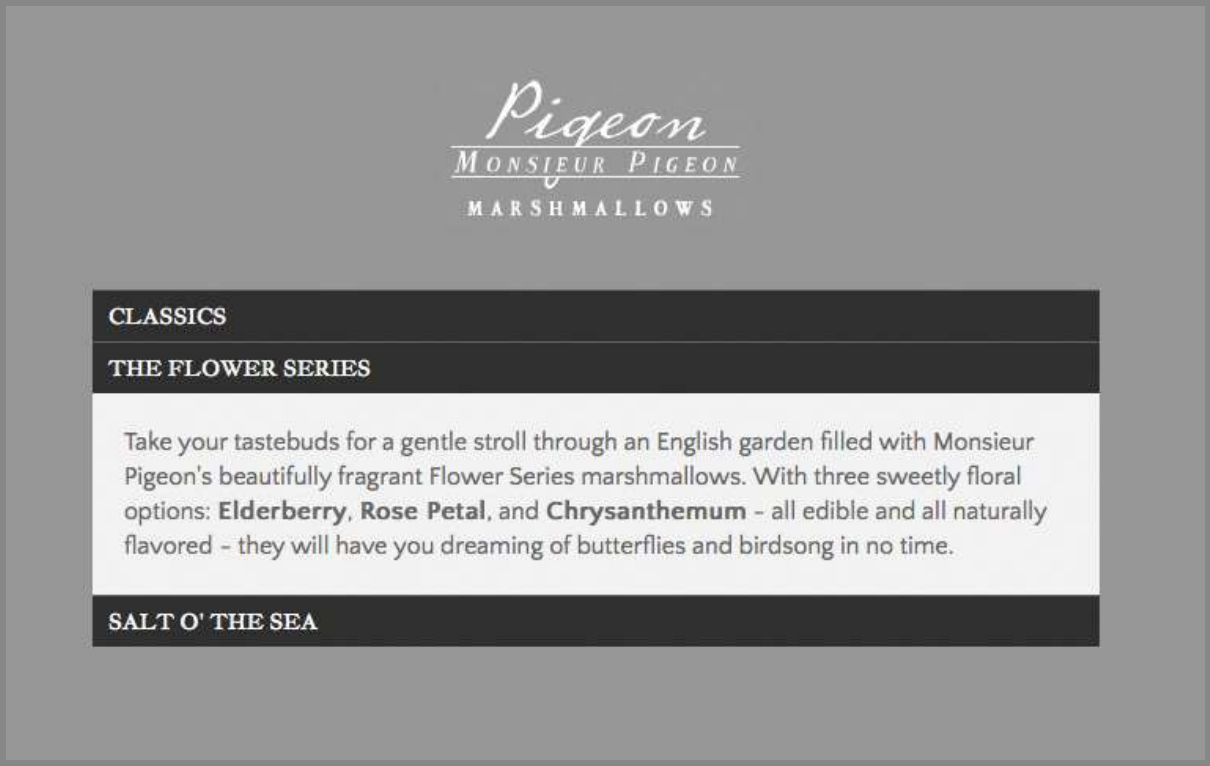
We also have ways to check all the property names in an object using a for…in statement

```
const classroom = { size: 24, year: 2, room: '80A’};

for (const property in classroom)
{
   console.log(`Property name: ${property}`);
   console.log(`Property value: ${classroom[property]}`);
}
```

CHAPTER 11

CONTENT PANELS

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



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

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.

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

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

LABEL 1

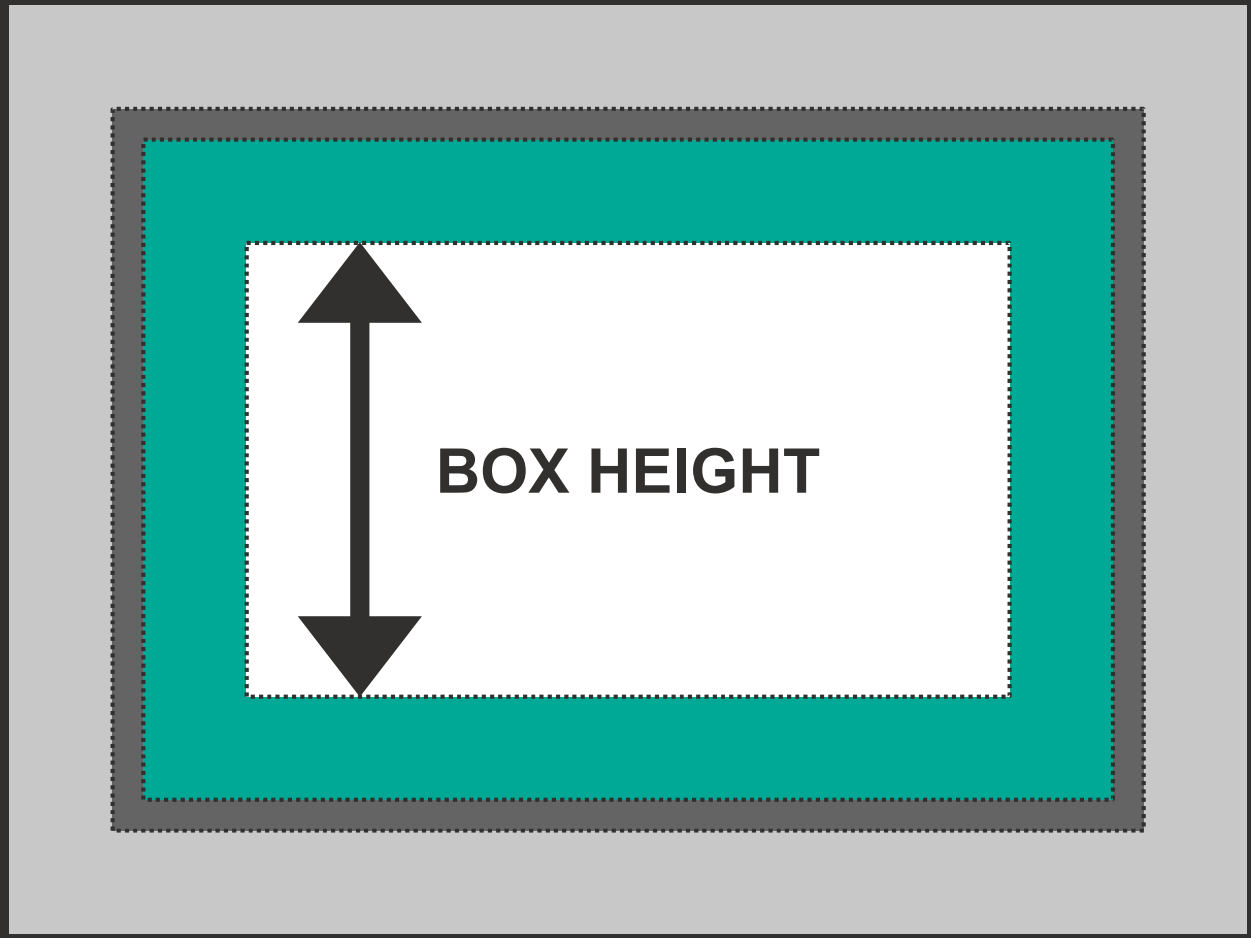
LABEL 2

LABEL 3

CONTENT 3

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.



BOX PADDING BORDER MARGIN

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

Pigeon

MONSIEUR PIGEON

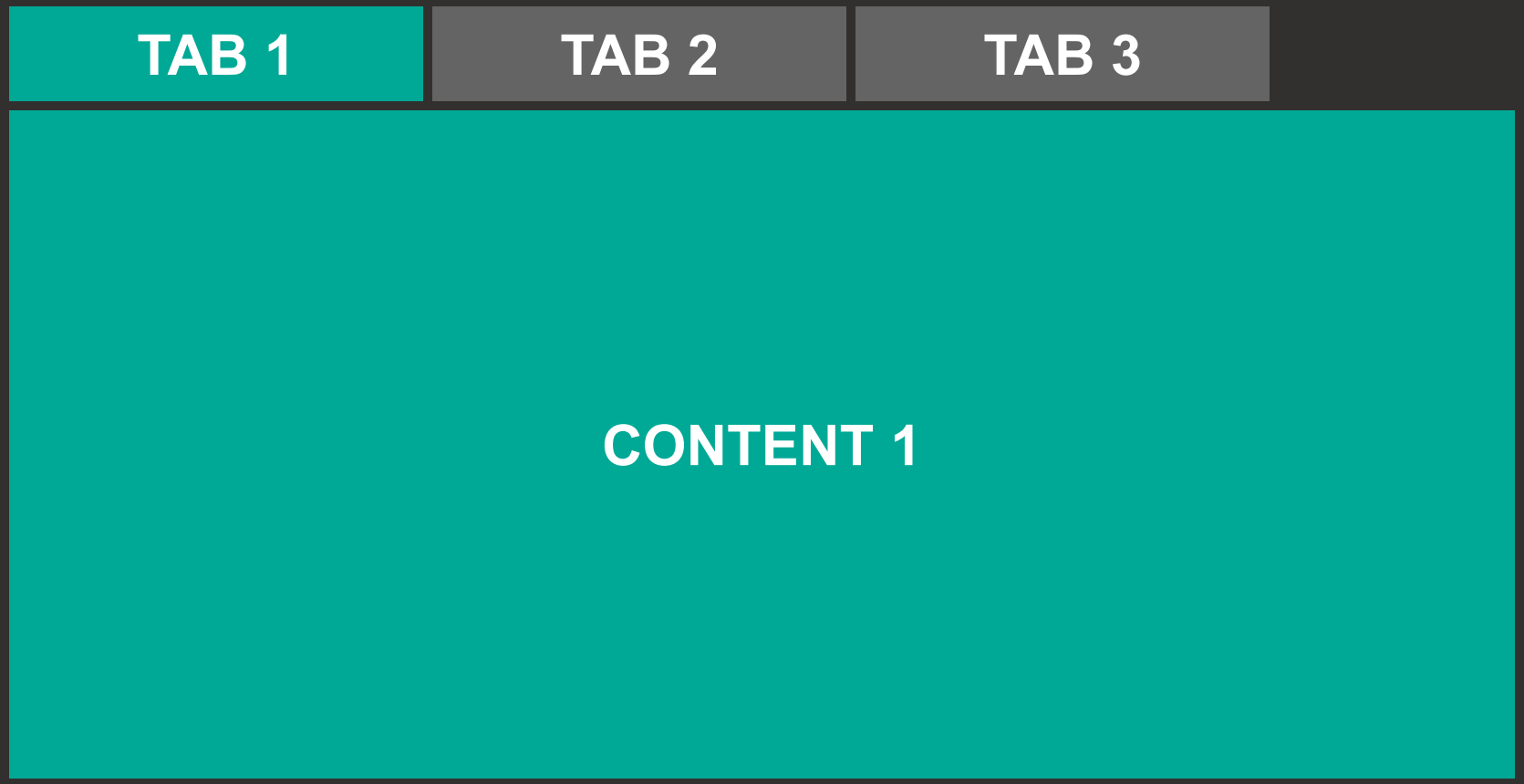
MARSHMALLOWS

DESCRIPTION | INGREDIENTS | DELIVERY

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

TAB 1 | TAB 2 | TAB 3

CONTENT 2

THIRD TAB SELECTED

TAB 1 | TAB 2 | TAB 3

CONTENT 3

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="tab1">
  <!-- Content for the first panel goes here -->
</div>

<div class="tab-panel" id="tab2">
  <!-- Content for the second panel goes here -->
</div>

<div class="tab-panel" id="tab3">
  <!-- 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.parent().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.parent().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.parent().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.parent().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


Pigeon
Monsieur Pigeon
MARSHMALLOWS
SALT O' THE S
~ AHOY
Try our latest sweet
SALTED CARAMEL &
SHARE THE MAGIC OF
Monsieur Pigeon
WITH YOUR CHUMS!
CLOSE

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.

BUTTON USED TO OPEN MODAL WINDOW

PAGE CONTENT

BUTTON USED TO OPEN MODAL WINDOW

CLOSE

MODAL CONTENT

APPEARS ON TOP OF PAGE

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

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.

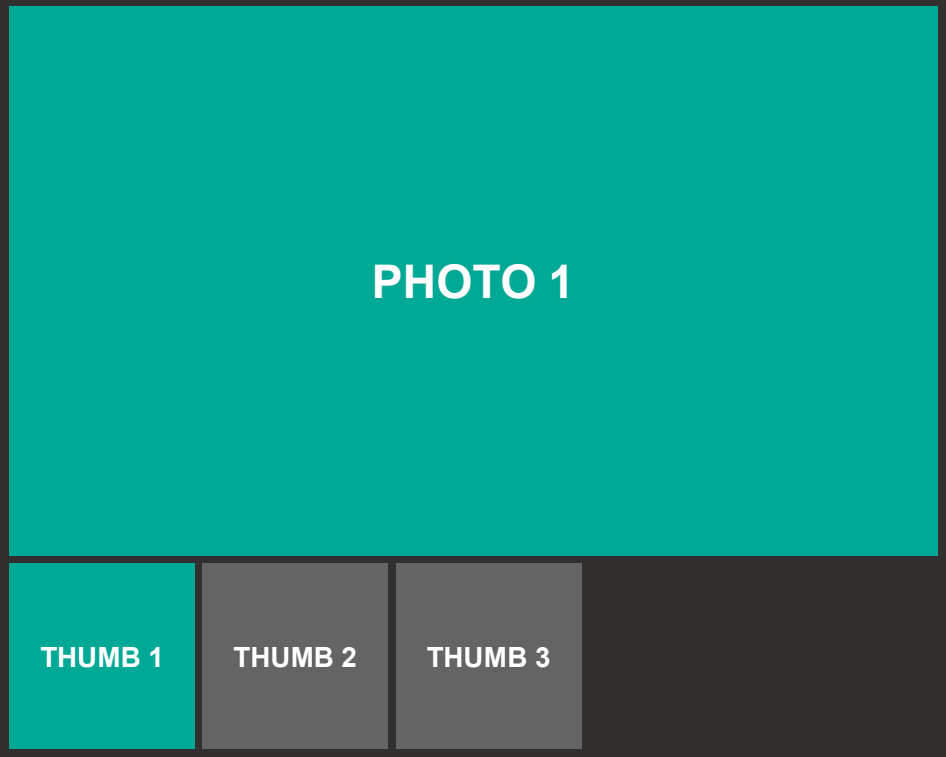
$5.75 / 3.5 oz packet

BUY NOW

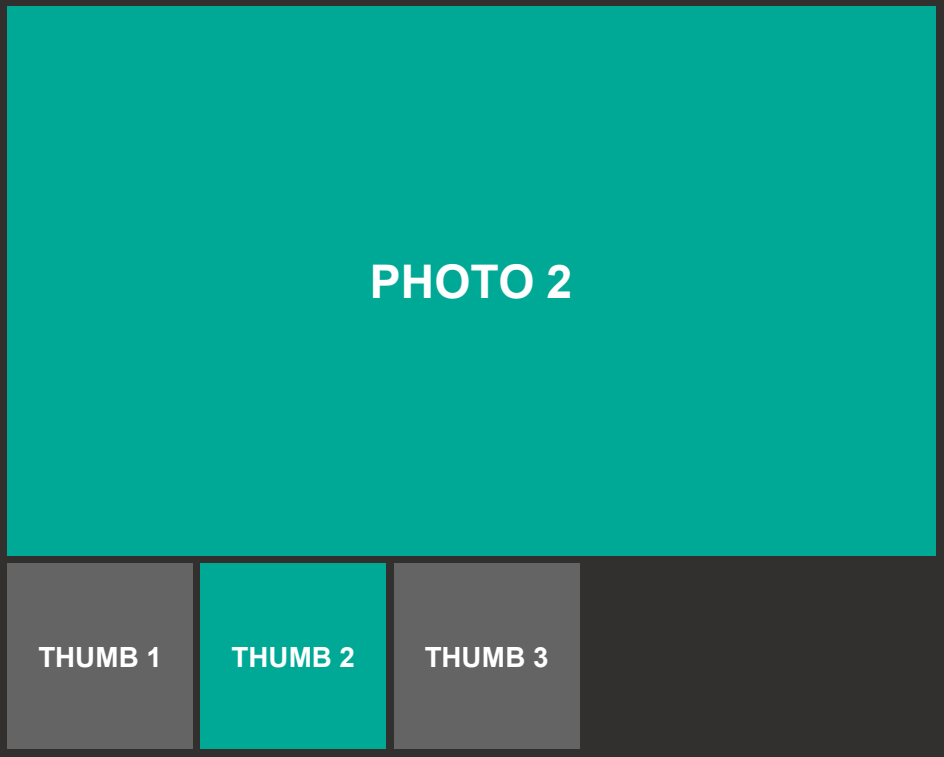
The photo viewer is an example of an image gallery.

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

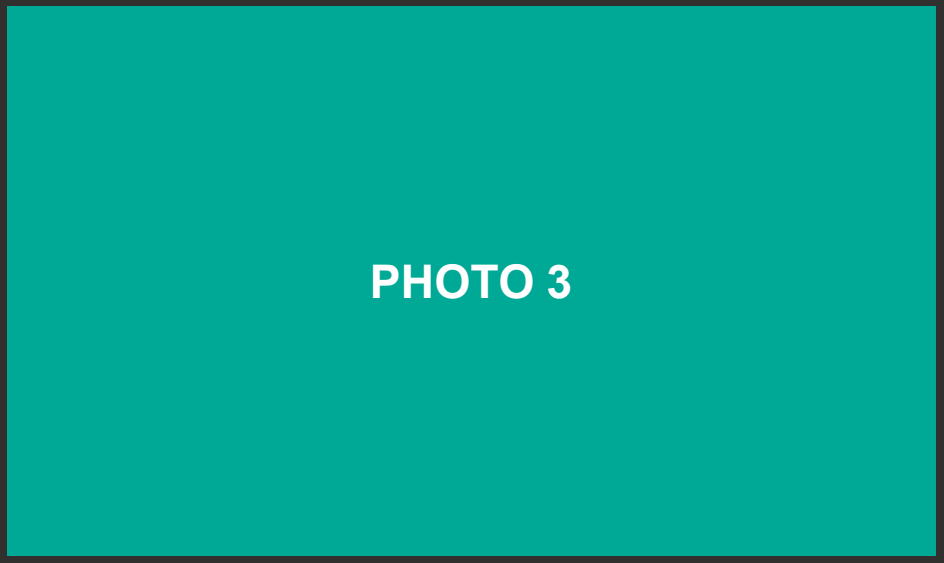
FIRST PHOTO SELECTED



SECOND PHOTO SELECTED



THIRD PHOTO SELECTED



HTML specifies the structure:

```
<div id="photo-frame"></div>

<div id="thumbnails">

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

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

  <a href="img/photo-3.jpg" class="thumb" 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-frame.is-loading:after {
  content: url('../img/load.gif');
  position: absolute;
  top: 0;
  left: 0;
}

#photo-frame 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 = {

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

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

}
```

Note: This is just an example of what will be populated with once it's running

Start by creating variables:

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

var $frame = $('#photo-frame');          // 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
  $(this).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($(this));              // 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

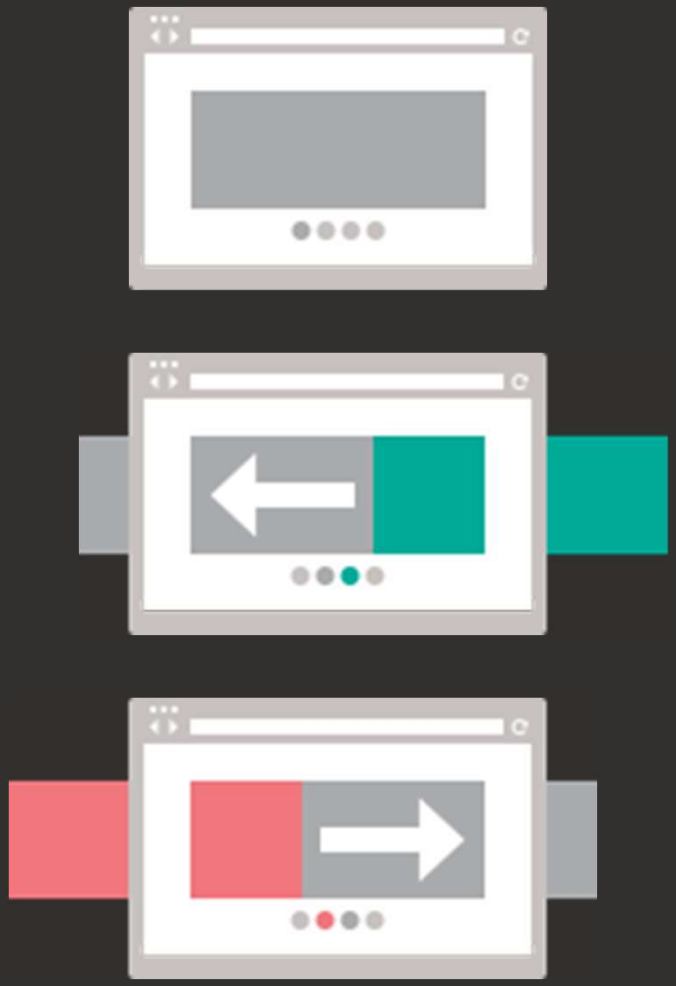
Pigeon

MONSIEUR PIGEON

MARSHMALLOWS

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.

.accordion() is a jQuery plugin:

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

Create a jQuery collection of all the elements with the class menu.

.accordion() is a jQuery plugin:

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

`.accordion()` is called on those elements.

.accordion() is a jQuery plugin:

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

The `fadeIn()` method is applied to the same selection.

jQuery has a property 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
```

CHAPTER 12

FILTERING, SEARCHING & SORTING

Filtering, searching, and sorting help users find the content they are looking for.

An array is a kind of object. It has methods and properties.

Arrays are often used to store complex data.

Array object's methods:

ADD ITEMS:	`push()`	`unshift()`
REMOVE:	`pop()`	`shift()`
ITERATE:	`forEach()`	
COMBINE:	`concat()`	
FILTER:	`filter()`	
REORDER:	`sort()`	`reverse()`

jQuery has similar methods for working with a jQuery collection:

ADD / COMBINE:	`.add()`
REMOVE:	`.not()`
ITERATE:	`.each()`
FILTER:	`.filter()`
CONVERT:	`.toArray()`

WHEN TO USE ARRAYS VS. OBJECTS

Arrays allow you to store items in order.

Objects allow you to select items by name.

FILTERING

NAME	HOURLY RATE ($)
Camille	80
Gordon	75

Filtering reduces a set of values. It creates a subset of data that meets certain criteria.

Data (people and the hourly rate they charge):

```
var people = [

  {
    name: 'Casey',
    rate: 60
  },

  {
    name: 'Nigel',
    rate: 120
  }

];
```

`forEach()`

Create a blank array called `results` and loop through the data about the people, adding anyone who charges between $65 and $90.

```
// LOOP THROUGH ARRAY ADD MATCHES TO TABLE

var results = [];                                  // Results array

people.forEach(function(person) {                  // Each person
  // Is the rate is in range
  if (person.rate >= 65 && person.rate <= 90) {
    results.push(person);                          // Add to array
  }
});
```

Create a table, then loop through the array (called `results`) adding a row for each person in the array:

```
var $tableBody = $('<tbody></tbody>');

for (var i = 0; i < results.length; i++) {
  var person = results[i];                  // Store current person
  var $row = $('<tr></tr>');                // Create row for them
  $row.append($('<td></td>').text(person.name));   // Add name
  $row.append($('<td></td>').text(person.rate));   // Add rate
  $tableBody.append( $row );
}

$('thead').after($tableBody);                       // Add body
```

The `filter()` method offers a slightly different way to select the people that match the criteria:

```
// FUNCTION ACTS AS FILTER

function priceRange(person) {
  return (person.rate >= 65) && (person.rate <= 90);
};

// FILTER PEOPLE ARRAY & ADD MATCHES TO ARRAY

var results = [];

results = people.filter(priceRange);
```

DYNAMIC FILTERING

CreativeFolk find talented people for your creative projects

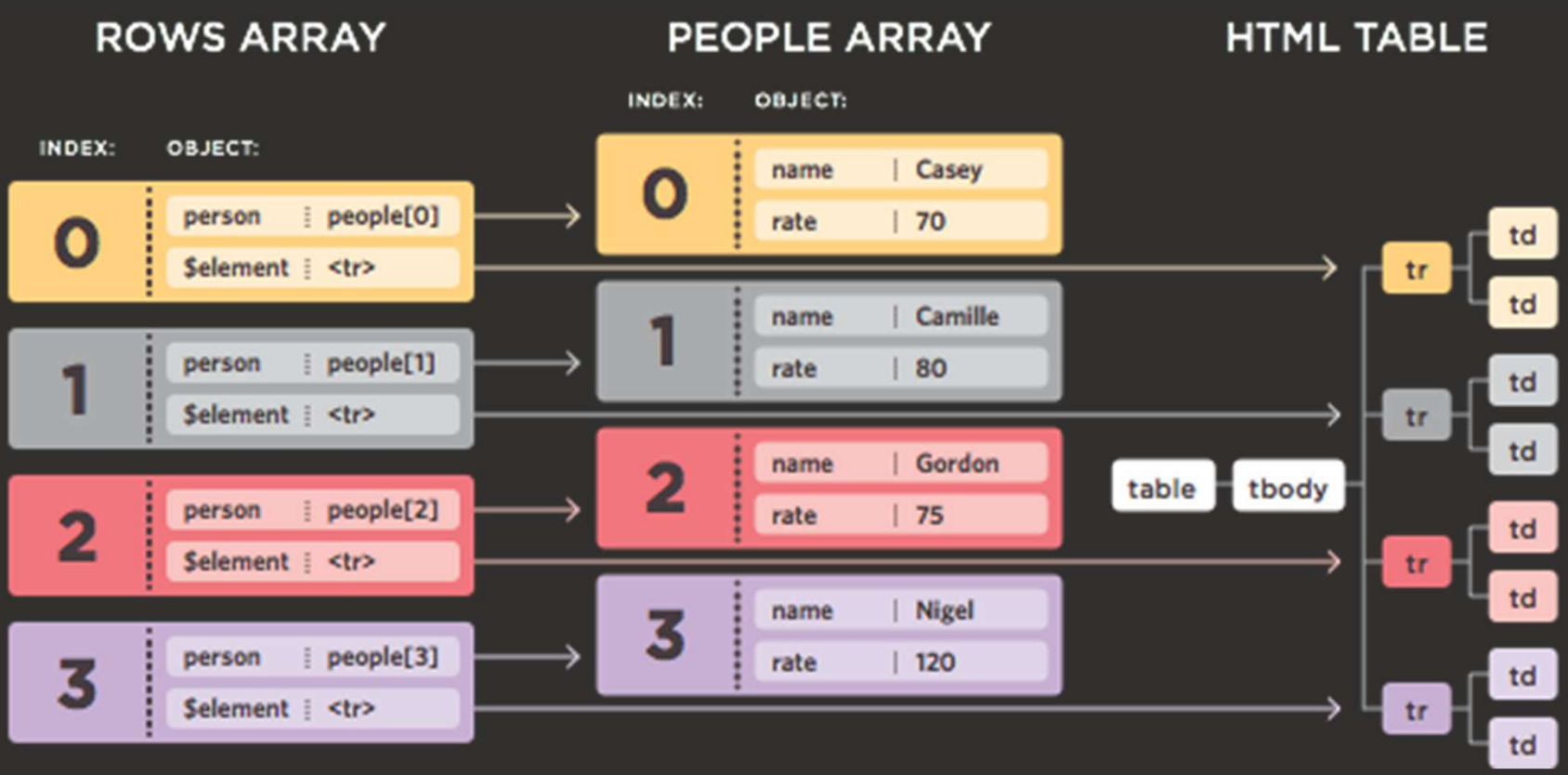
Min: 65 Max: 90

NAME	HOURLY RATE ($)
Camille	80
Gordon	75

- Doesn’t rebuild a table each time the filter runs
  Creates a row for each person, then shows or hides those rows

An array called `rows` will store references to:

The object that represents each person
A jQuery object holding the row of the table for a person


ROWS ARRAY
INDEX:
OBJECT:
0
person
people[0]
$element
<tr>
1
person
people[1]
$element
<tr>
2
person
people[2]
$element
<tr>
3
person
people[3]
$element
<tr>
PEOPLE ARRAY
INDEX:
OBJECT:
0
name
Casey
rate
70
1
name
Camille
rate
80
2
name
Gordon
rate
75
3
name
Nigel
rate
120
HTML TABLE
table
tbody
tr
td
td
tr
td
td
tr
td
td
tr
td
td

Creating the `rows` array:

```
var rows = [],                              // rows array
    $min = $('#value-min'),                 // Minimum text input
    $max = $('#value-max'),                 // Maximum text input
    $table = $('#rates');                   // Table to show results

function makeRows() {
  people.forEach(function(person) {         // For each person
    var $row = $('<tr></tr>');              // Create their row
    $row.append( $('<td></td>').text(person.name) );// Add name
    $row.append( $('<td></td>').text(person.rate) );// Add rate
    rows.push({                             // Create rows array
      person: person,                       // Person object
      $element: $row                        // jQuery object: row
    });
  });
}
```

Add a row to the table for each person:

```
function appendRows() {
 var $tbody = $('<tbody></tbody>');   // Create <tbody> element
  rows.forEach(function(row) {        // Each obj in rows array
    $tbody.append(row.$element);      // Add HTML for the row
  });
  $table.append($tbody);              // Add rows to the table
}
```

To update the table content:

```
function update(min, max) {
  rows.forEach(function(row) {                    // For each row
    // If the person's price is within range
    if (row.person.rate >= min && row.person.rate <= max) {
      row.$element.show();                        // Show the row
    } else {                                      // Otherwise
      row.$element.hide();                        // Hide the row
    }
  });
}
```

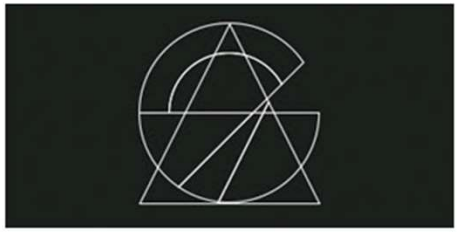
When the script first runs:

```
function init() {

  // Set up the slider
  $('#slider').noUiSlider({
    range: [0, 150], start: [65, 90],
    handles: 2, margin: 20, connect: true,
    serialization: {to: [$min, $max],resolution: 1}
  }).change(function() { update($min.val(), $max.val()); });

  makeRows();                    // Create rows and rows array
  appendRows();                  // Add the rows to the table

  // Update table to show matching people
  update($min.val(), $max.val());
}

$(init);                         // Call init() when DOM is ready
```

FILTERING AN IMAGE GALLERY

CreativeFolk find talented people for your creative projects

Show All | Animators (3) | Illustrators (3) | Photographers (3) | Filmmakers (2) | Designers (3)



In the HTML, images are tagged using attributes called `data-tags`:

```
<div id="buttons"></div>

<div id="gallery">

  <img src="p1.jpg"
       data-tags="Animators, Illustrators"
       alt="Rabbit" />

  <img src="p2.jpg"
       data-tags="Photographers, Filmmakers"
       alt="Sea" />

  <img src="p3.jpg"
       data-tags="Photographers, Filmmakers"
       alt="Deer" />

  <!-- More images go here -->

</div>
```

A set of buttons is created from the values in the attributes. An object called `tagged` stores each tag, and a reference to all of the images using that tag.

Basic set-up and creation of `tagged` object:

```
$(function() {
  var $imgs = $('#gallery img');                    // Store all images
  var $buttons = $('#buttons');                     // Store buttons
  var tagged = {};                                  // Create tagged
object

  $imgs.each(function() {                           // Loop through images
    var img = this;                                 // Store img in var
    var tags = $(this).data('tags');                // Get its tags
    if (tags) {                                     // If it has tags
      tags.split(',').forEach(function(tagName){// Split at comma
        if (tagged[tagName] == null) {              // If obj has no tag
          tagged[tagName] = [];                     // Add array to object
        }
      tagged[tagName].push(img);                    // Add image to array
    }
  });
```

The “Show All” button:

```
$('<button/>', {                          // Create button
  text: 'Show All',                       // Add text
  class: 'active',                        // Make it active
  click: function() {                     // Add click handler
    $(this)                               // Get clicked button
    .addClass('active')                   // Make it active
    .siblings()                           // Get its siblings
    .removeClass(‘active');               // Remove active class
    $imgs.show();                         // Show all images
  }
}).appendTo($buttons);                    // Add to buttons
```

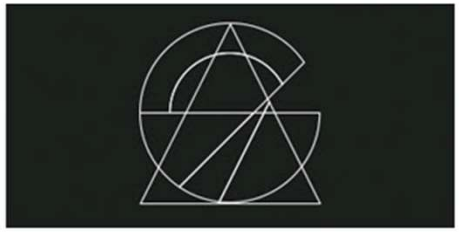
The tag buttons:

```
$.each(tagged, function(tagName) { // For each tag name
  $('<button/>', {                 // Create empty button
    // Add tag name
    text: tagName + ' (' + tagged[tagName].length + ')',
    click: function() {            // Add click handler
      $(this)                      // The button clicked on
        .addClass('active')        // Make clicked item active
        .siblings()                // Get its siblings
        .removeClass('active');    // Remove active siblings
      $imgs                        // With all of the images
        .hide()                    // Hide them
        .filter(tagged[tagName])   // Find ones with this tag
        .show();                   // Show just those images
    }
  }).appendTo($buttons);           // Add to the buttons
 });
});
```

SEARCHABLE IMAGE

CreativeFolk find talented people for your creative projects

s



The buttons from the previous example are replaced by a search box.

If tags contain characters entered into the search box, the corresponding images are shown.

Set up and create cache:

```
$(function() {
  var $imgs = $('#gallery img');                    // Get images
  var $search = $('#filter-search');                // Get input
  var cache = [];                                   // Create array

  $imgs.each(function() {                           // Each img
    cache.push({                                    // Add to cache
      element: this,                                // This image
      text: this.alt.trim().toLowerCase()           // Its alt text
    });
  });
```

Filter function:

```
function filter() {
  var query = this.value.trim().toLowerCase(); // Get query
  if (query) {                                 // If there's a query
    cache.forEach(function(img) {              // Each cache entry
      var index = 0;                           // Set index to 0
      index = img.text.indexOf(query);         // Is text in there?
      // Show / hide
      img.element.style.display = index === -1 ? 'none' : '';
    });
  }
}
```

Trigger filter when text changes:

```
  // If browser supports input event
  if ('oninput' in $search[0]) {
    // Use input event to call filter()
    $search.on('input', filter);
  } else { // Otherwise
    // Use keyup event to call filter()
    $search.on('keyup', filter);
  }

});
```

SORTING

Sorting involves taking a set of values and reordering them.

We will use the Array object's `sort()` method to do this.

The `sort()` method works like a dictionary: lexicographically e.g. Abe, Alice, Andrew, Anna

It orders items by the first letter. If two items have the same first letter, it looks at the second letter, and so on.

This doesn’t work so well with numbers…

1, 2, 14, 19, 125, 156

BECOMES

1, 125, 14, 156, 19, 2

To change the order, you use a compare function.

Compare functions always compare two values at a time and return a number.

a should go *before* **b**

1 - 3 = -2

a - b = < 0

a should go *after* **b**

5 - 3 = 2

a - b = > 0

a should go *after* **b**

4 - 3 = 1

a - b = > 0

a should go *before* **b**

4 - 5 = -1

a - b = < 0

a should go *before* **b**

2 - 3 = -1

a - b = < 0

a should go *after* **b**

2 - 1 = 1

a - b = > 0

SORTING NUMBERS: ASCENDING

a	operator	b	result	order
1	-	2	-1	`a` before `b`
2	-	2	0	same order
2	-	1	1	`b` before `a`

SORTING NUMBERS: ASCENDING

```
var prices = [1, 2, 125, 19, 14];

prices.sort(function(a,b){
  return a - b;
});
```

SORTING NUMBERS: DESCENDING

a	operator	b	result	order
2	-	1	1	`b` before `a`
2	-	2	0	same order
1	-	2	-1	`a` before `b`

SORTING NUMBERS: RANDOM

```
prices.sort(function(a,b){
   return 0.5 - Math.random();
});
```

Dates can be compared using `<` and `>` operators by turning the values into a `Date` object.

SORTING DATES

```
dates.sort(function(a,b){
  var dateA = new Date(a);
  var dateB = new Date(b);

  return dateA - dateB;
});
```

SORTING A TABLE

My Videos

Camille Berger
Paris, France

GENRE	▲ TITLE	DURATION	DATE
Film	Animals	6:40	2005-12-21
Film	The Deer	6:24	2014-02-28
Animation	The Ghost	11:40	2012-04-10
Animation	Wagons	21:40	2007-04-12
Animation	Wildfood	3:47	2014-07-16

The table can be sorted by clicking on a header.

Three compare functions will be stored in an object called `compare`.

Headers indicate type of data:

```
 <table class="sortable">
<thead>
 <tr>
    <th data-sort="name">Genre</th>
    <th data-sort="name">Title</th>
    <th data-sort="duration">Duration</th>
    <th data-sort="date">Date</th>
  </tr>
</thead>
<tbody>
  <tr>
    <td>Animation</td>
    <td>Wildfood</td>
    <td>3:47</td>
    <td>2014-07-16</td>
  </tr>
...
```

The `compare` object has three methods that are compare functions to store the data.

1: `compare` object's `name()` method

```
var compare = {                          // Declare object
  "name": function(a, b) {                 // Add name() method
    a = a.replace(/^the /i, '');         // Remove The
    b = b.replace(/^the /i, '');         // Remove The

    if (a < b) {                         // If a less than b
      return -1;                         // Return -1
    } else {                             // Otherwise
      // If a greater than b return 1 otherwise return 0
      return a > b ? 1 : 0;
    }
  }, // More methods go here...
```

2: `compare` object's `duration()` method

```
duration: function(a, b) {                    // duration() method
  a = a.split(':');                           // Split time at colon
  b = b.split(':');                           // Split time at colon

  // Convert the time to seconds
  a = Number(a[0]) * 60 + Number(a[1]);
  // Convert the time to seconds
  b = Number(b[0]) * 60 + Number(b[1]);

  return a - b;                               // Return a minus b
},
```

3: `compare` object's `date()` method

```
  date: function(a, b) {              // Add a method called date
    a = new Date(a);                  // New object to hold date
    b = new Date(b);                  // New object to hold date

    return a - b;                       // Return a minus b
  }
}
```

Set up and compare data when header is clicked:

```
$('.sortable').each(function() {
  var $table = $(this);                          // This table
  var $tbody = $table.find('tbody');             // Table body
  var $controls = $table.find('th');             // Table headers
  var rows = $tbody.find(`tr').toArray();        // Array of rows

  $controls.on('click', function() {             // Event handler
    var $header = $(this);                       // Get header
    var order = $header.data('sort');            // Get data type
    var column;                                  // Used later
```

If item’s class is ascending or descending, reverse the order:

```
if ($header.is('.ascending') || $header.is('.descending')) {
  // Toggle to other class
  $header.toggleClass('ascending descending’);
  // Reverse the array
  $tbody.append(rows.reverse());
} else {
```

Order using `compare` object's methods:

```
$header.addClass('ascending');                  // Add class to header

// Remove asc or desc from all other headers
$header.siblings().removeClass('ascending descending');

// If compare object has method of that name
if (compare.hasOwnProperty(order)) {
  column = $controls.index(this);               // Column's index no

  rows.sort(function(a, b) {                    // Call sort() on rows
    a = $(a).find('td').eq(column).text();// Text of column row a
    b = $(b).find('td').eq(column).text();// Text of column row b
    return compare[order](a, b);                // Call compare method
  });

  $tbody.append(rows);
}
}
});
```

CHAPTER 13

FORM ENHANCEMENT & VALIDATION

Form enhancement makes forms easier to use.

Validation ensures that you are getting the right information from users.

Examples in this chapter use helper functions. They add cross-browser event handlers.

Helper function to add events:

```
function addEvent(el, event, callback) {
  // If addEventListener works use it
  if ('addEventListener' in el) {
    el.addEventListener(event, callback, false);
  } else {
    // Otherwise create IE fallback
    el['e' + event + callback] = callback;
    el[event + callback] = function () {
      el['e' + event + callback](window.event);
    };
    el.attachEvent('on' + event, el[event + callback]);
  }
}
```

DOM nodes for form controls have different properties and methods than other elements.

<form> ELEMENT

properties	methods	events
action	submit()	submit
method	reset()	reset
name		
elements		

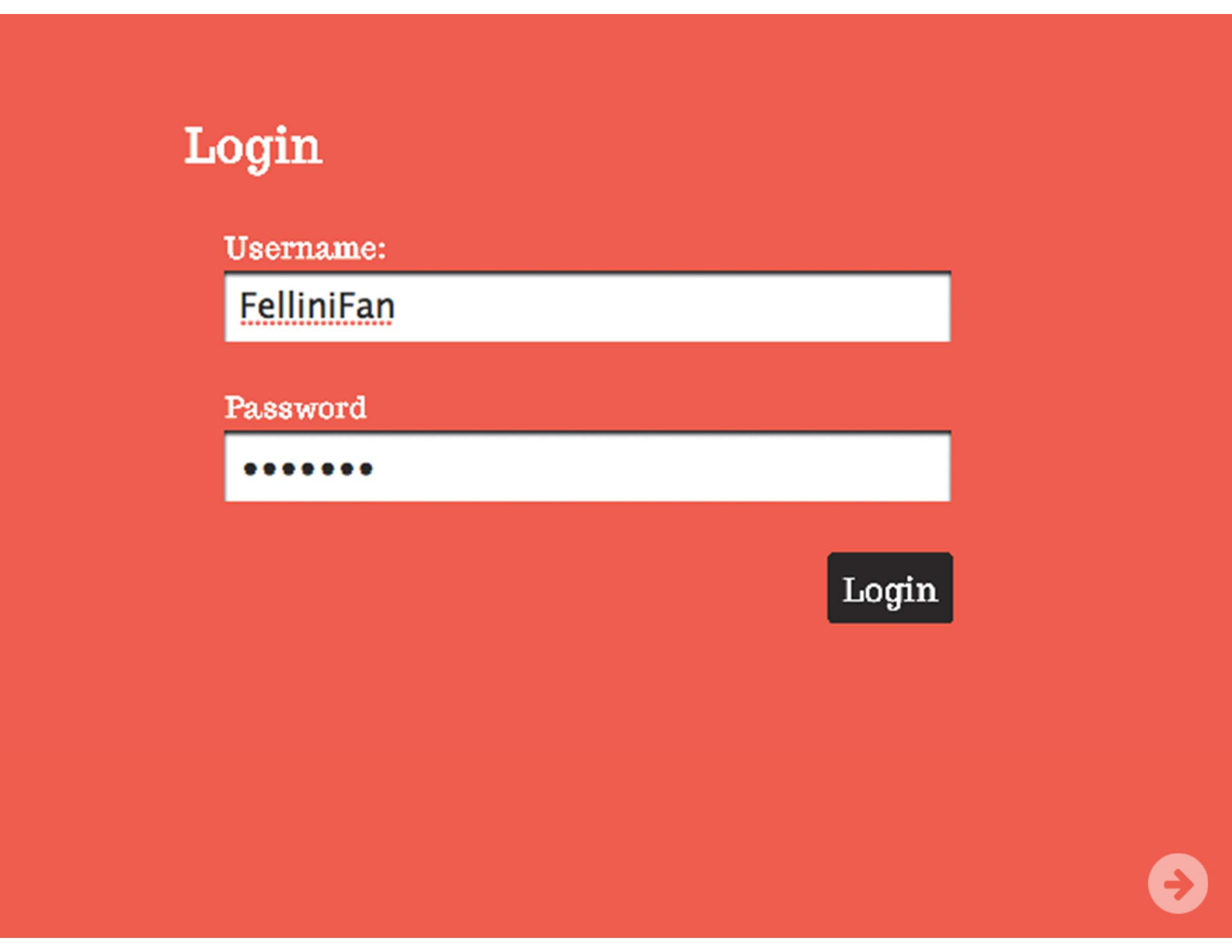
FORM CONTROLS

properties	methods	events
value	focus()	blur
type	blur()	focus
name	select()	click
disabled	click()	change
checked		input
selected		keyup
form		keydown
defaultChecked		keypress

WORKING WITH FORMS

SUBMITTING FORMS

To work with a form's content, use the `preventDefault()` method of the `event` object to stop it from being sent.


Login
Username:
FelliniFan
Password
Login

Submitting a form:

```
(function() {
  var form = document.getElementById('login'); // Form element

  addEvent(form, 'submit', function(e) { // On submit
    e.preventDefault();                  // Stop it being sent
    var elements = this.elements;        // Get form elements
    var username = elements.username.value; // Get username
    var msg      = 'Welcome ' + username;   // Welcome message

    // Write welcome message
    document.getElementById('main').textContent = msg;
  });
}());
```

TYPE OF INPUT

The `type` property of an input corresponds with the `type` attribute in HTML.

(It won't work in IE8 or earlier.)

Login

Username:

FelliniFan

Password

8point5

☑ show password

Login

Showing a password:

```
var pwd = document.getElementById('pwd');      // Get pwd input
var chk = document.getElementById('showPwd'); // Get checkbox

addEvent(chk, 'change', function(e) {     // Click on checkbox
  var target = e.target || e.srcElement; // Get that element
  try {                                  // Try following code
    if (target.checked) {                // If checked set
      pwd.type = 'text';                 // type to text
    } else {                             // Otherwise set
      pwd.type = 'password';             // type to password
    }
  } catch(error) {                       // If an error
    alert('This browser cannot switch type'); // Show warning
  }
});
```

DISABLE INPUTS

The `disabled` property of an input corresponds with the `disabled` attribute in HTML.

Reset password

New password:

submit

Disabling a submit button:

```
var form      = document.getElementById(‘newPwd'); // Form
var password  = document.getElementById('pwd');    // Password
var submit    = document.getElementById('submit'); // Submit

var submitted = false;              // Has form been submitted?
submit.disabled = true;             // Disable submit button
submit.className = 'disabled';      // Style submit button
```

CHECKBOXES

The `checked` property of an input corresponds with the `checked` attribute in HTML.

Genres

- [x] All
- [x] Animation
- [x] Documentary
- [x] Shorts

Selecting all checkboxes:

```
var form = document.getElementById('interests'); // Form
var elements = form.elements;                    // Elements in form
var options  = elements.genre;                   // Genre checkboxes
var all = document.getElementById(‘all');  // 'All' checkbox

function updateAll() {
  for (var i = 0; i < options.length; i++) { // Each checkbox
    options[i].checked = all.checked;         // Update property
  }
}
addEvent(all, 'change', updateAll);          // Event listener
```

RADIO BUTTONS

The `checked` property is also commonly used with radio buttons.

How did you hear of us?

- Search engine
- Newspaper or magazine
- Other

submit

Showing a text input:

```
options   = form.elements.heard;                    // Radio
buttons
other     = document.getElementById('other');       // Other button
otherText = document.getElementById('other-text'); // Other
text otherText.className = 'hide';                    // Hide
other

for (var i = [0]; i < options.length; i++) {     // Each option
 addEvent(options[i], 'click', radioChanged);    // Add listener
}

function radioChanged() {
 hide = other.checked ? '' : 'hide';     // Is other checked?
 otherText.className = hide;             // Text input
visibility
 if (hide) {                             // If text input hidden
  otherText.value = '';                  // Empty its contents
 }
}
```

SELECT BOXES

Select boxes have more properties and methods than other form controls.

The `<option>` elements hold the values select boxes contain.

SELECT BOXES

properties	methods
`options`	`add()`
`selectedIndex`	`remove()`
`length`	
`multiple`	
`selectedOptions`	

1

Equipment hire

type Please choose a type

model Please choose a type first

2

Equipment hire

type ✓ Please choose a type
camera
projector

model

3

Equipment hire

type camera

model Please choose a model

4

Equipment hire

type camera

model ✓ Please choose a model
Bolex Paillard H8
Yashica 30
Pathescape Super-8 Relax
Canon 512

Info for select boxes is stored in objects:

```
// Type select box
var type  = document.getElementById(‘equipmentType');

// Model select box
var model = document.getElementById('model');
var cameras = {                          // Object stores cameras
    bolex: 'Bolex Paillard H8',
    yashica: 'Yashica 30',
    pathescape: 'Pathescape Super-8 Relax',
    canon: 'Canon 512'
};
var projectors = {                       // Store projectors
    kodak: 'Kodak Instamatic M55',
    bolex: 'Bolex Sound 715',
    eumig: 'Eumig Mark S',
    sankyo: 'Sankyo Dualux'
};
```

Getting the right object:

```
function getModels(equipmentType) {
  // If type is cameras return cameras object
  if (equipmentType === 'cameras') {
    return cameras;
  // If type is projectors return projectors object
  } else if (equipmentType === 'projectors') {
    return projectors;
  }
}
```

Populating select boxes:

```
addEvent(type, 'change', function() {  // Change select box

  if (this.value === 'choose') {       // No selection made
    model.innerHTML = '<option>Please choose a type
first</option>';
    return;                            // No need to proceed
  }
  var models = getModels(this.value);  // Get right object

  var options = '<option>Please choose a model</option>';
  var key;
  for (key in models) {                // Loop through models
    options += '<option value="' + key + '">' + models[key]
               + '</option>';
  }
  model.innerHTML = options;           // Update select box

});
```

TEXTAREA

The `value` property gets and updates the value entered into a textarea or text input.

Profile

Short bio (up to 140 characters)

I first discovered the art of Super 8 in a dusty old box in my father's attic. The beautiful colors of his footage of New York in 1969

5 characters

Set-up and event handling:

```
(function () {
  var bio      = document.getElementById(‘bio');
  var bioCount = document.getElementById('bio-count');

  // Call updateCounter() on focus or input events
  addEvent(bio, 'focus', updateCounter);
  addEvent(bio, 'input', updateCounter);

  addEvent(bio, 'blur', function () {   // Leaving the element
    if (bio.value.length <= 140) {      // If bio not too long
      bioCount.className = 'hide';      // Hide the counter
    }
  });
```

Updating the counter:

```
function updateCounter(e) {
  var target = e.target || e.srcElement;// Get target of event
  var count = 140 - target.value.length; // Characters left
  if (count < 0) {                        // Less than 0 chars
    bioCount.className = 'error';         // Add class of error
  } else if (count <= 15) {              // Less than 15 chars?
    bioCount.className = 'warn';          // Add class of warn
  } else {                                // Otherwise
    bioCount.className = 'good';          // Add class of good
  }
  var charMsg = '<b>' + count + '</b>' + ' characters'; // Msg
  bioCount.innerHTML = charMsg;           // Update counter
}
```

HTML5 ELEMENTS & ATTRIBUTES

HTML5 added form elements and attributes that perform tasks that had previously been done by JavaScript.

In particular, the elements can check that the user entered the right kind of information.

If not, they show an error.
This is known as validation.

EMAIL, PHONE & URL

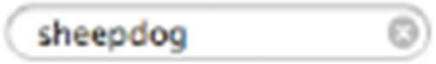
SAFARI 

FIREFOX 

CHROME 


```
<input type="email">
<input type="url">
<input type="tel">
```

SEARCH

SAFARI 

FIREFOX 

CHROME 


```
<input type="search"
     placeholder="sheepdog"
       autofocus>
```

NUMBER

SAFARI 

FIREFOX 

CHROME 


```
<input type="number"
       min="0"
       max="10"
       step="2"
       value="6">
```

RANGE

SAFARI

FIREFOX

CHROME

```
<input type="range"
       min="0"
       max="10"
       step="2"
       value="6">
```

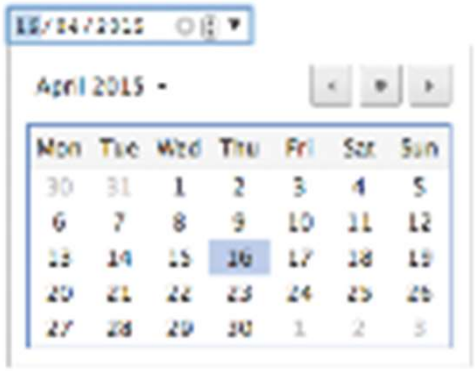
COLOR PICKER

CHROME 

```
<input type="color">
```

DATE

CHROME




```
<input type="date">
<input type="month">
<input type="week">
<input type="time">
<input type="datetime">
```

HTML5 elements are not supported in all desktop browsers. (There is much better support on mobile.)

When they are supported, they can look very different.

To get around this lack of support, you can use polyfills or feature detection.

FORM VALIDATION

Form validation checks that users enter data in the right format. If not, an error message is shown to users.

Generic checks are the kind that would be performed on different kinds of form.

If it uses the `required` attribute, does it have a value?
Does the value match what is indicated by the `type` attribute?

Custom validation tasks correspond to specific requirements of a given form.

Is the user's bio less than 140 characters?
If the user is under 13, is the parental consent checkbox selected?

Check every element before submitting the form so you can show all errors at once.

You can create an object to keep track of each element and whether its entry is valid.

To check if you can submit the form, check the `valid` object.

The last example in the book uses JavaScript for validation and HTML5 validation as a fallback. This gives maximum visual consistency, and browser compatibility.

Checking if a required input has a value uses three functions:

```
function validateRequired(el) {
  if (isRequired(el)) {
    var valid = !isEmpty(el);
    if (!valid) {
      setErrorMessage(el, 'Field required');
    }
    return valid;
  }
  return true;
}
```

The `isRequired()` function checks if it has the `required` attribute:

```
function  isRequired(el) {
  return ((typeof el.required === 'boolean') && el.required)
         ||(typeof el.required ==='string');
}
```

The `isEmpty()` function checks if the element is empty:

```
function  isEmpty(el) {
  return !el.value || el.value === el.placeholder;
}
```

Error messages can be stored with the element using jQuery's `data()` method:

```
function setErrorMessage(el, message) {
  $(el).data('errorMessage', message);
}

function showErrorMessage(el) {
  var $el = $(el);
  var $errorContainer = $el.siblings('.error');
  if (!$errorContainer.length) {
    $errorContainer = $('<span class="error">
                         </span>').insertAfter($el);
  }
  $errorContainer.text(getErrorMessage(el));
}

function getErrorMessage(el) {
    return $(el).data('errorMessage') || el.title;
}
```

The type of content in a text input is validated using the `validateTypes()` function.

In turn, this function uses an object called `validateType` which has three methods to validate email addresses, numbers, and dates.

The `validateType` object uses regular expressions:

```
var validateType = {
  email: function(el) {
    var valid = /[^@]+@[^@]+/.test(el.value);
    if (!valid) {
      setErrorMessage(el, 'Please enter a valid email');
    }
    return valid;
  },
  number: function(el) {
    // Check is a number
  },
  date: function(el) {
    // Check date format
  }
}
```

The `validateType` object is used by the `validateTypes()` function:

```
function validateTypes(el) {
  if (!el.value) return true;

  var type = $(el).data('type') || el.getAttribute('type');

  if (typeof validateType[type] === 'function') {
    return valiudateType[type](el);
  } else {
    return true;
  }

}
```

Regular expressions search for characters that form a pattern. They can also replace those characters with new ones or simply remove them.

Character Matches

`.`	single character (except newline)
`[ ]`	single character in brackets
`[^ ]`	single character not in brackets
`\d`	digit [0-9]
`\D`	non-digit character
`\w`	word character `[A-Za-z0-9_]`
`\W`	non-word character
`\s`	white space character `[ \t\r\n\f]`
`\S`	non-white space character

Anchor Matches

`^`	the starting position in any line
`$`	ending position in any line
`\b`	word boundary

These don’t match any characters, only conditions within the line.

Repeaters

`?`	preceding element 0 or 1 times
`+`	preceding element 1 or more times
`*`	preceding element 0 or more times
`{n}`	preceding element `n` times
`{n,m}`	preceding element `n` to `m` times

Checking the length of the bio is an example of custom validation:

```
function validateBio() {
  var bio = document.getElementById('bio');
  var valid = bio.value.length <= 140;
  if (!valid) {
    setErrorMessage(bio, 'Bio should not exceed 140 chars);
  }
  return valid;
}
```

VALIDATION EXAMPLE OVERVIEW

A: Set up script
B: Perform generic checks
C: Perform custom validation
D: Did it pass validation?

Was the form valid? A flag is used to check through each item in the `valid` object:

```
// Loop through every form control - are there errors?
for (var field in valid) {
  if (!valid[field]) {
    isFormValid = false;
    break;
  }
  isFormValid = true;
}

if(!isFormValid) {
  e.preventDefault;
}
```