JAVASCRIPT
& JQUERY
interactive front-end
web development

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:

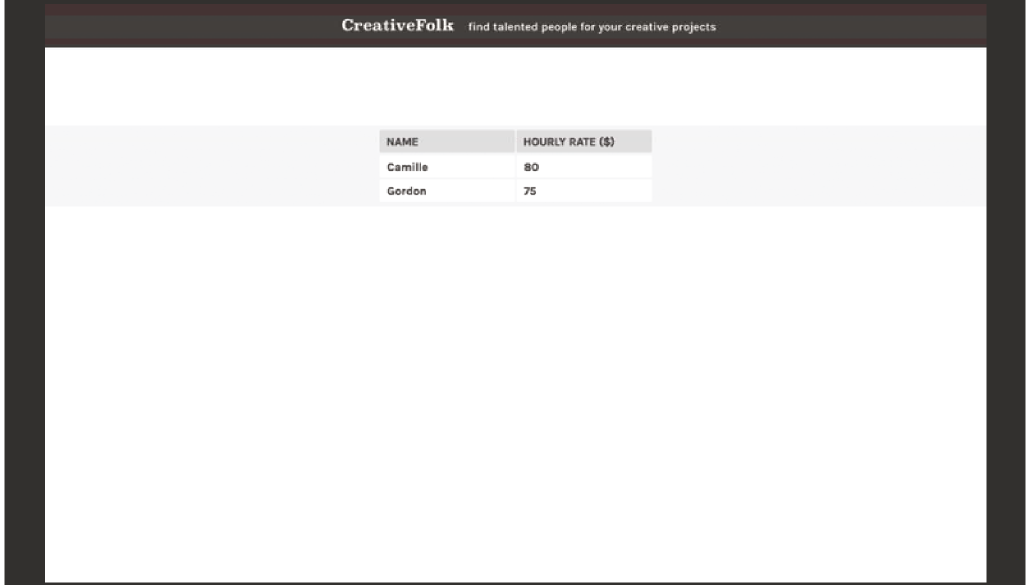
ADD / COMBINE:	`.add()`
REMOVE:	`.not()`
ITERATE:	`.each()`
FILTER:	`.filter()`
CONVERT:	`.toArray()`

WHEN TO USE ARRAYS VS. OBJECTS

Arrays allow you to store items in order.

Objects allow you to select items by name.

FILTERING



CreativeFolk find talented people for your creative projects

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
  }
});
```

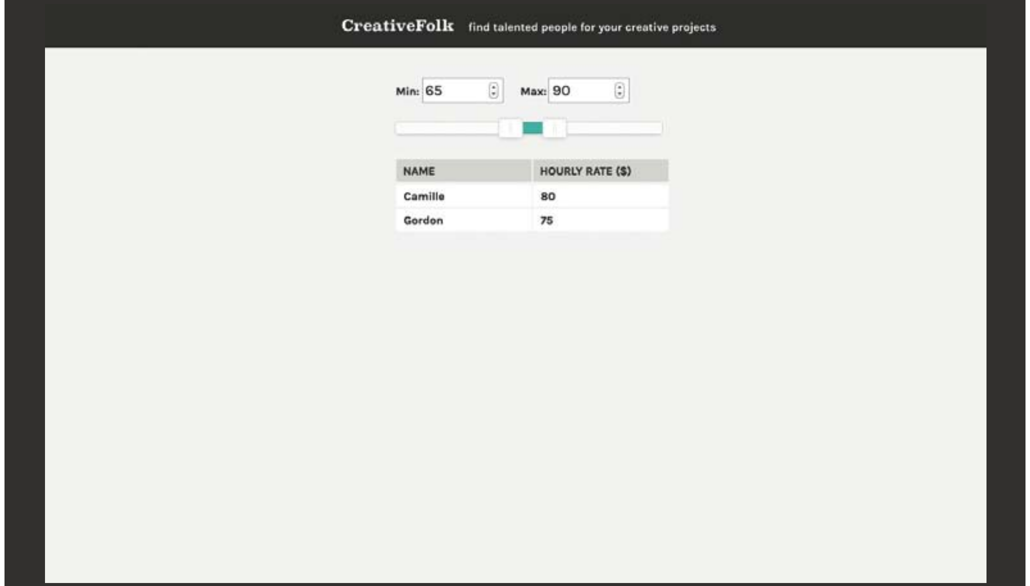
Create a table, then loop through the array (called `results`) adding a row for each person in the array:

```
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
}

$(‘thead').after($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

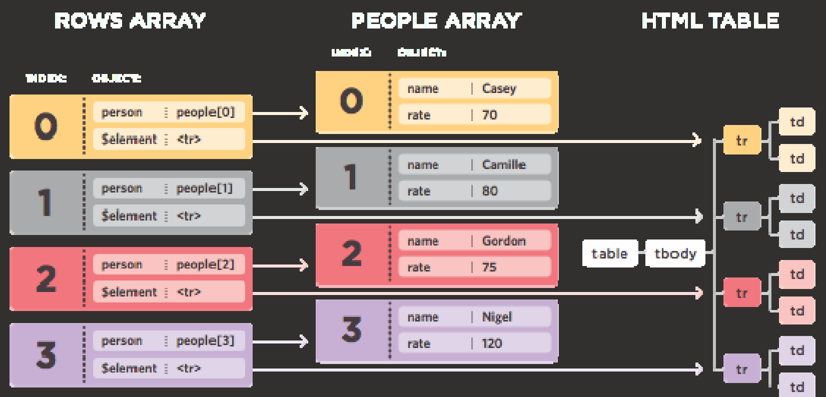
Min: 65 Max: 90

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



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
}
```

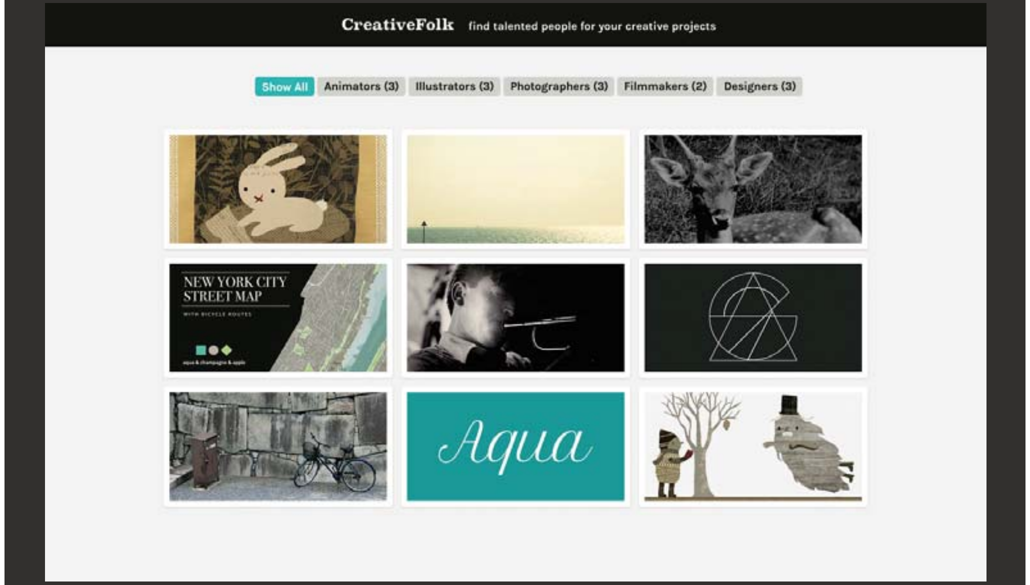
To update the table content:

```
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



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
    });
  }
});
```

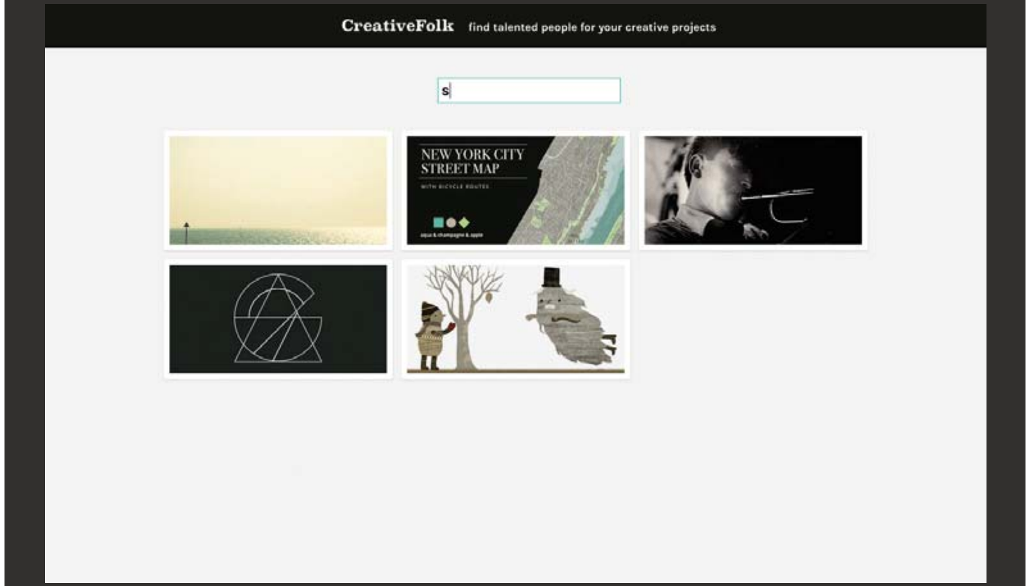
The “Show All” button:

```
$('<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



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
  cache.forEach(function(img) {               // Each cache entry
    var index = 0;                            // Set index to 0

    if (query) {                              // If there’s a query
      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…

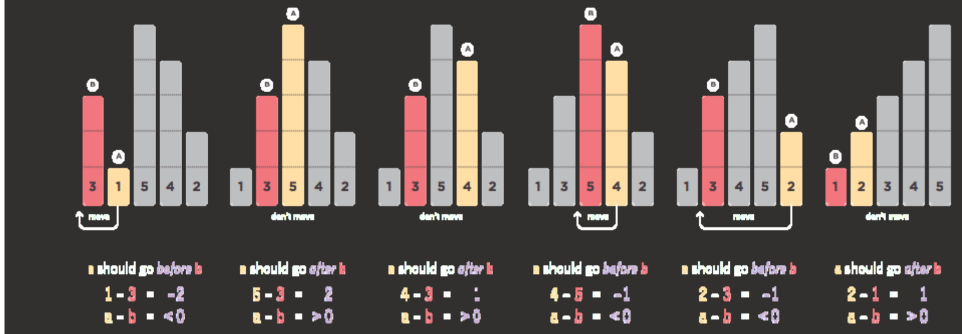
1, 2, 14, 19, 125, 156

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.



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 price = [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	-	1	0	same order
2	-	1	-1	a before b

SORTING NUMBERS: RANDOM

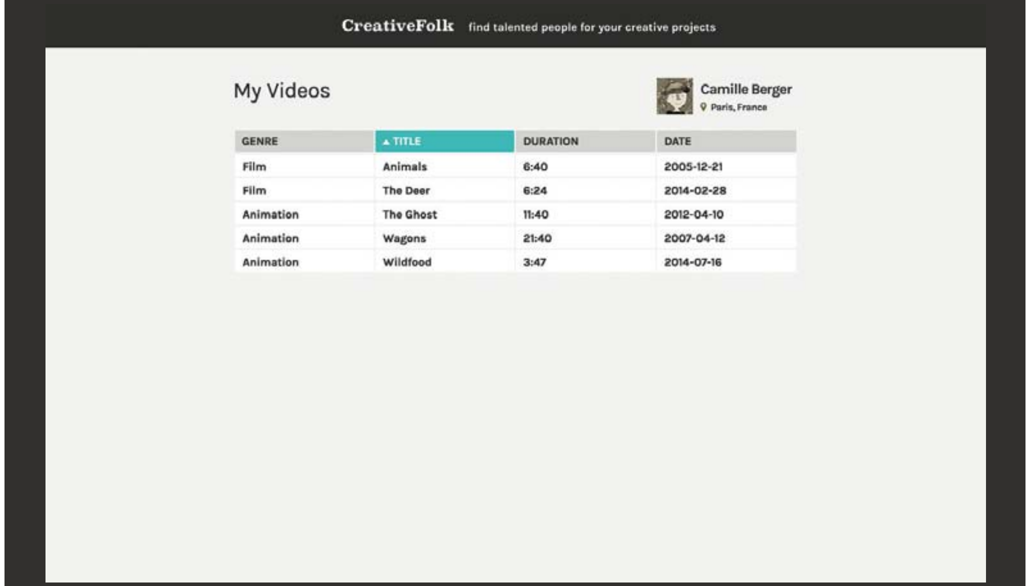
```
prices.sort(function(a,b){
  return 0.5 - Math.random();
});
```

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



CreativeFolk find talented people for your creative projects

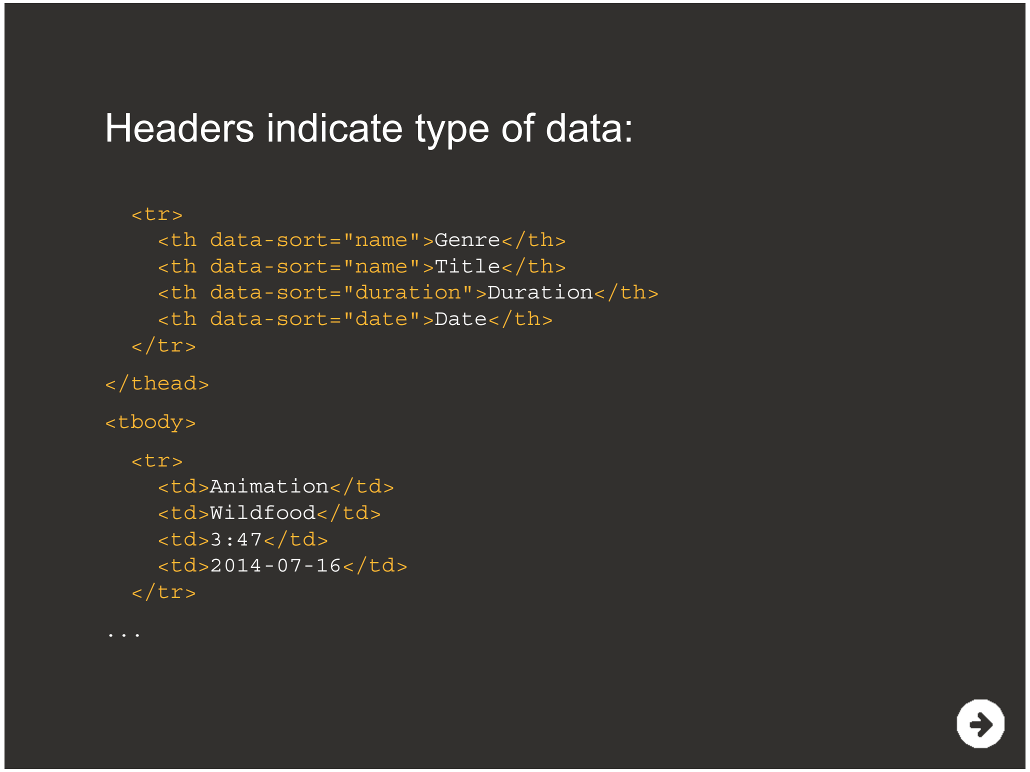
My Videos

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

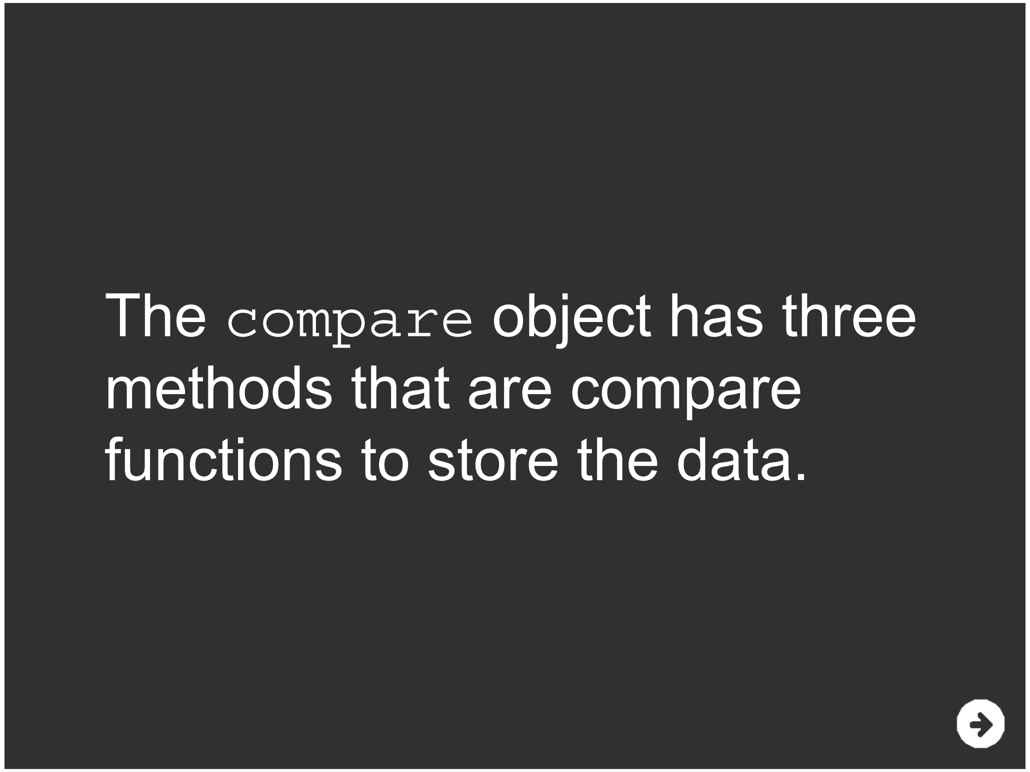
The table can be sorted by clicking on a header.

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

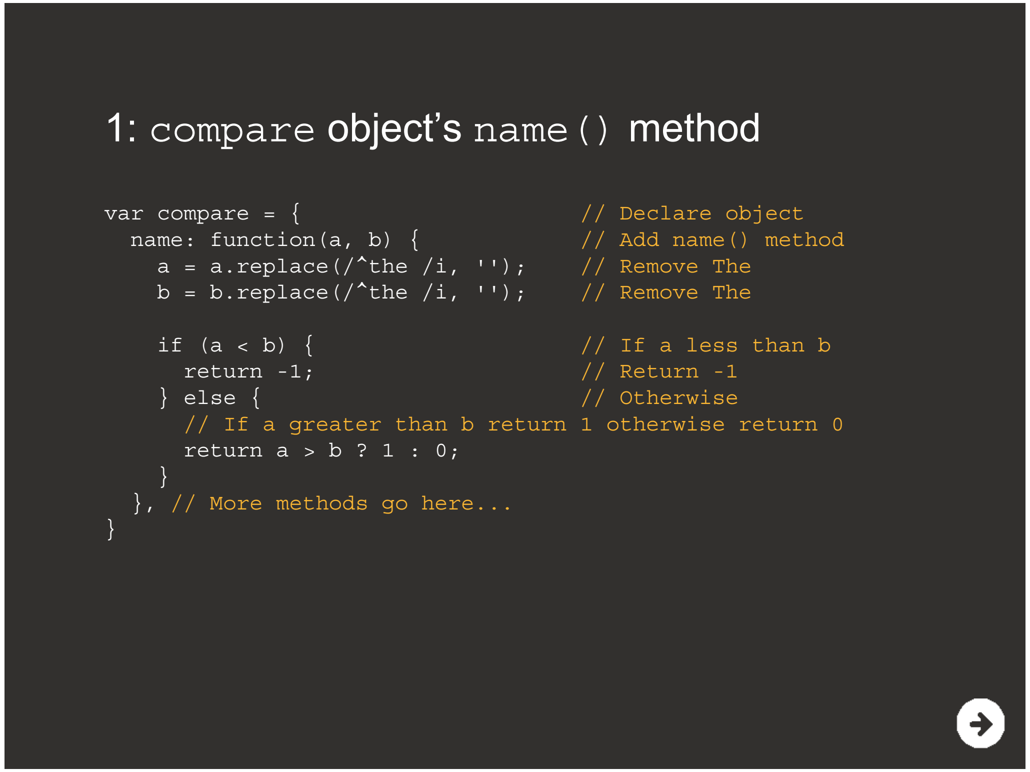


Headers indicate type of data:

```
  <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>
...
```

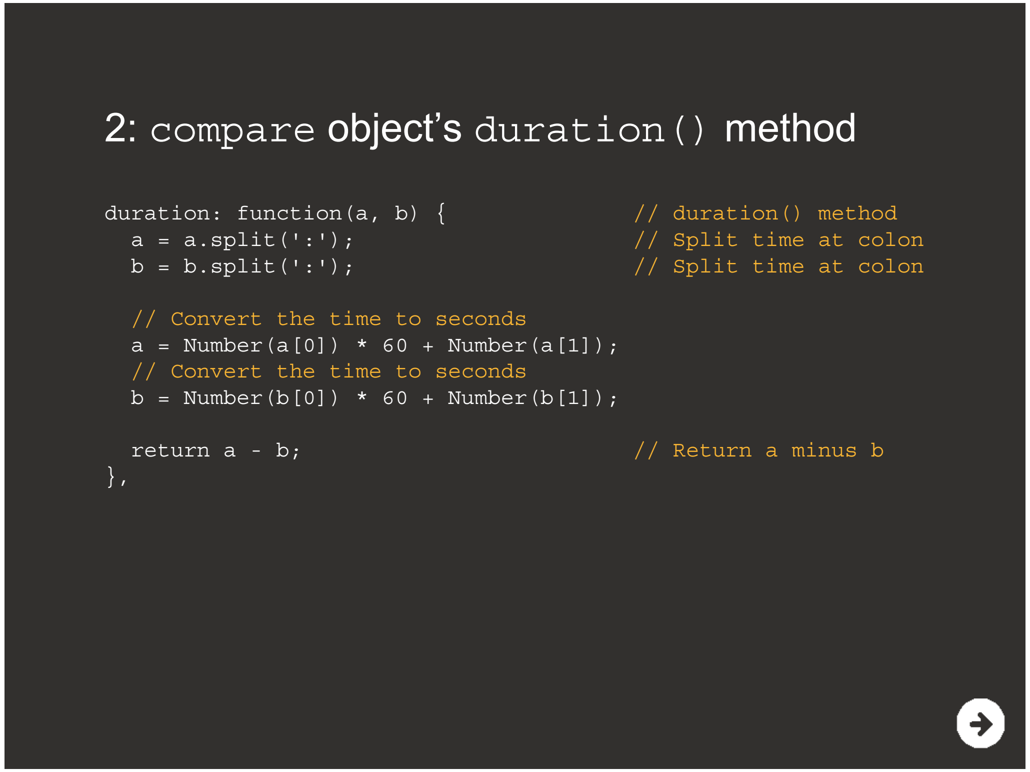


The `compare` object has three methods that are compare functions to store the data.



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);
}
```