



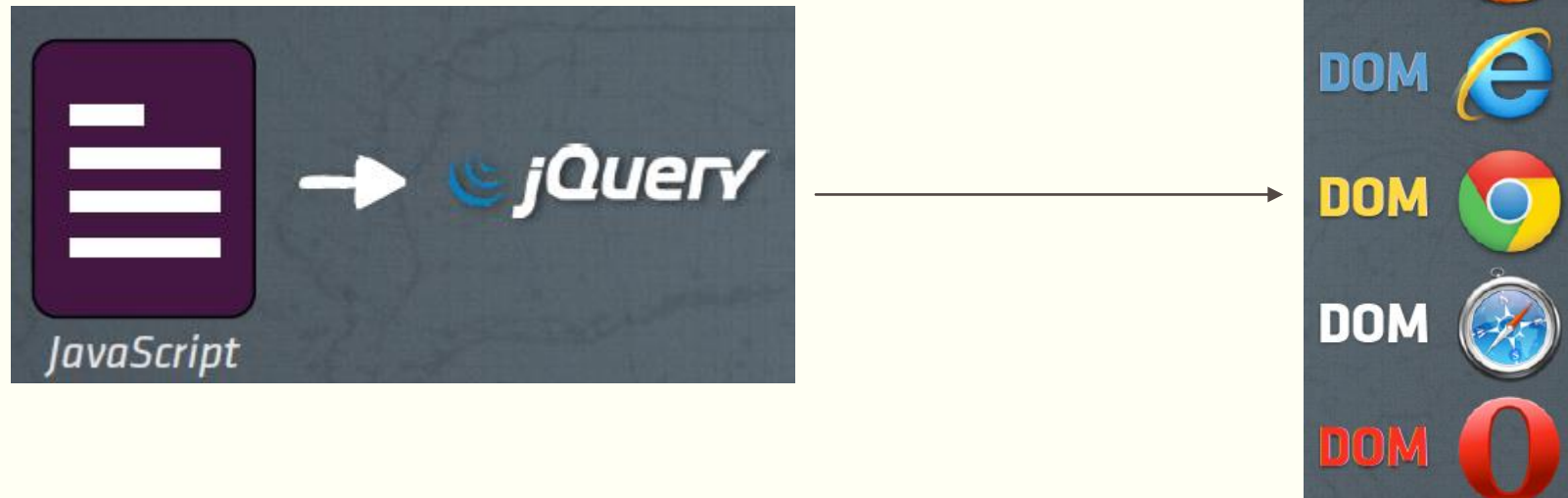
JQUERY

banuprakashc@yahoo.co.in








Why jQuery?

- Each Browser has a slightly different DOM interface
- If our JavaScript uses jQuery to interact with the DOM then it will work on most modern browsers.



Why jQuery?

- jQuery makes it easy to

	find	elements in an HTML document
	change	HTML content
	listen	to what a user does and react accordingly
	animate	content on the page
	talk	over the network to fetch new content

Why jQuery?

- jQuery makes the DOM less scary

The raw JavaScript way

I'm talking to the document
(aka the big D in DOM).

Get me all of the
elements that have
the tag name of "p."

```
document.getElementsByTagName("p")  
[0].innerHTML = "Change the page.";
```

Get me the
zeroth element.

Set the HTML
inside that element...
...to this stuff.

The jQuery way

Grab me a
paragraph element.

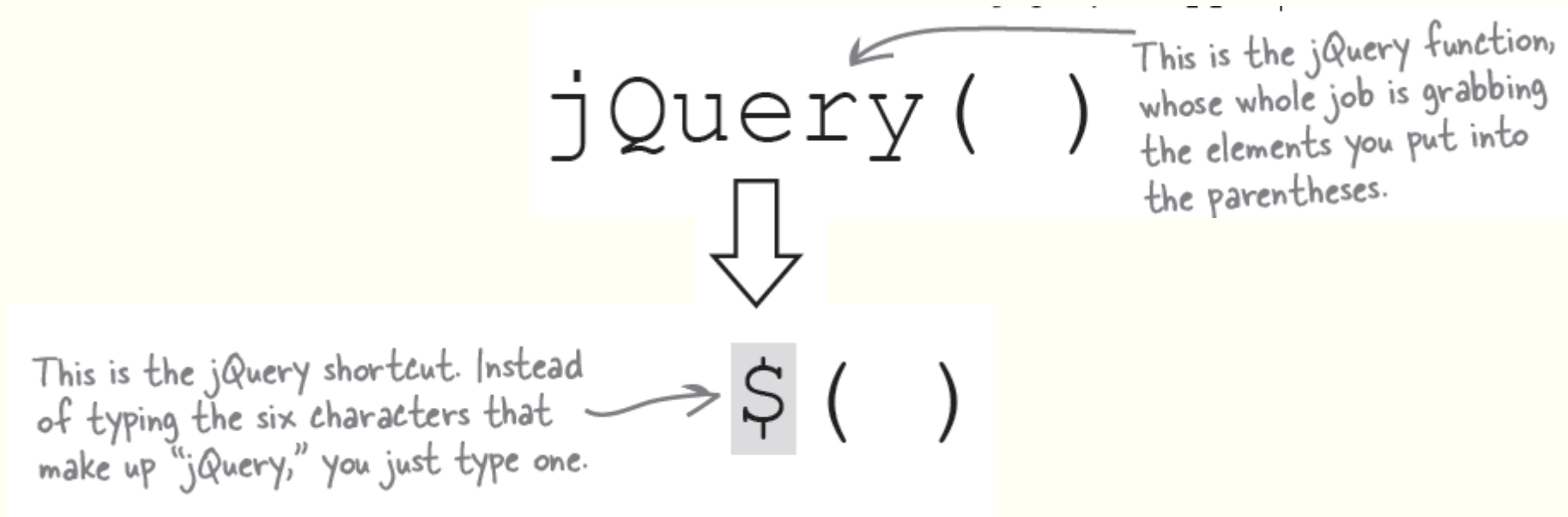
Change the HTML of
that element to what's
in these parentheses.

```
$("p").html("Change the page.");
```

jQuery uses a "selector engine,"
which means you can get at stuff
with selectors just like CSS does.

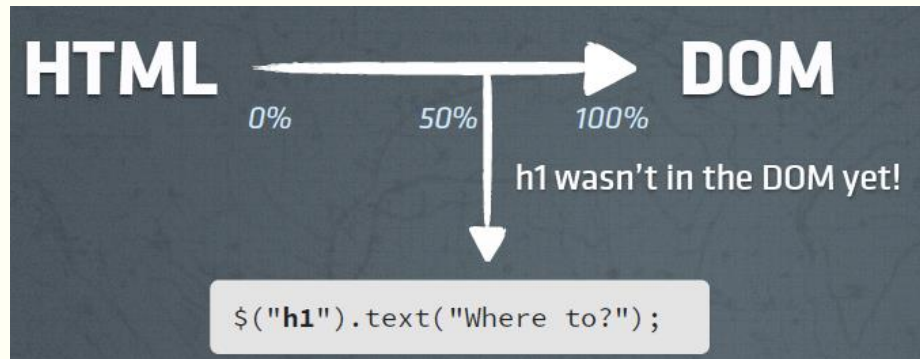
The jQuery function

- The dollar sign with the parentheses is the shorter name of the jQuery **function**.
- This shortcut saves us from writing “jQuery()” every time we want to call the jQuery function. The jQuery function is also often referred to as the jQuery **wrapper**.



jQuery(document).ready()

- A page can't be manipulated safely until the document is "ready."
- jQuery detects this state of readiness for you.
- Code included inside jQuery(document).ready() will only run once the page Document Object Model (DOM) is ready for JavaScript code to execute.



```
jQuery(document).ready(function(){  
    $("h1").text("Where to?");  
});
```

jQuery(document).ready()

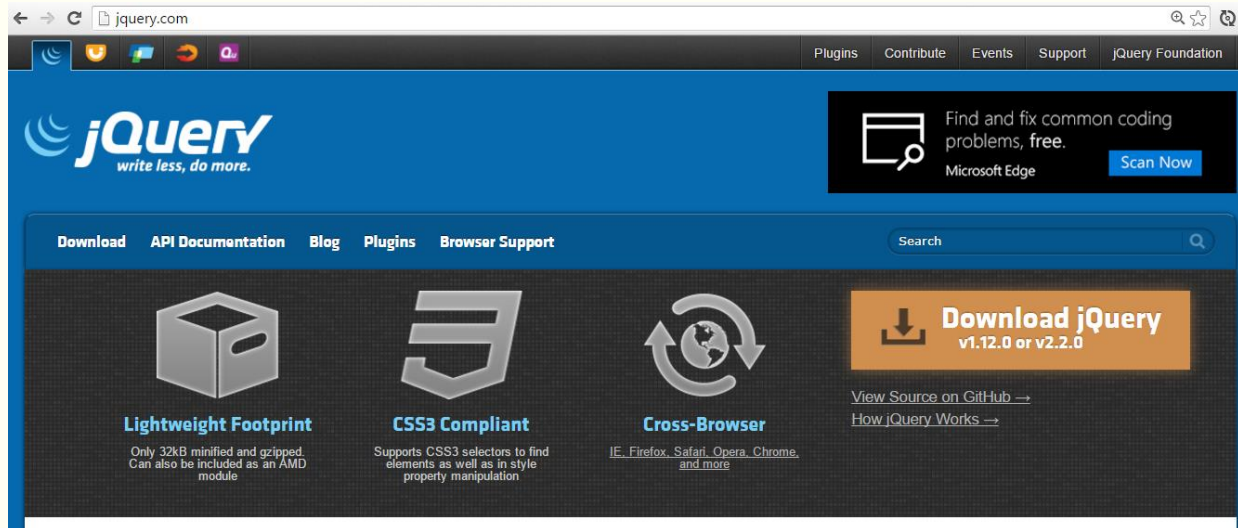
- Other syntax

```
// A $( document ).ready() block.  
$( document ).ready(function() {  
    console.log( "ready!" );  
});
```

```
// Shorthand for $( document ).ready()  
$(function() {  
    console.log( "ready!" );  
});
```


Getting Started

- Download jQuery



- Load it in your HTML document
 - `<script src="jquery.min.js"></script>`
- Start using it
 - `<script src="app.js"></script>`

Element Selector

- Select the h2 element of this simple web page.

```
<body>

<div class="container">
  <h2>Welcome to jQuery Travels - Traversing the DOM since 2006</h2>
  <p>Fly to New York today for as little as <span>$299.99</span></p>
</div>

<script type="text/javascript" src="jquery.js"></script>

</body>

$("h2")
  [<h2>Welcome to jQuery Travels - Traversing the DOM since 2006</h2>]
$("h2").text()
  "Welcome to jQuery Travels - Traversing the DOM since 2006"
```

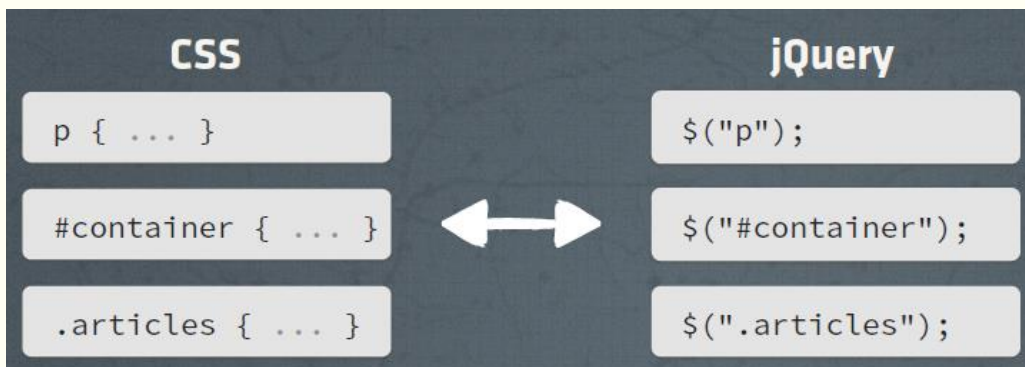
- change the price to \$100.
 - \$("span").text("\$100");

Selecting multiple elements

- `$("li")`

```
<h1>Where do you want to go?</h1>
<h2>Travel Destinations</h2>
<p>Plan your next adventure.</p>
  <ul id="destinations">
    <li>Delhi</li>
    <li>Paris</li>
    <li class='promo'>Rio</li>
  </ul>
```

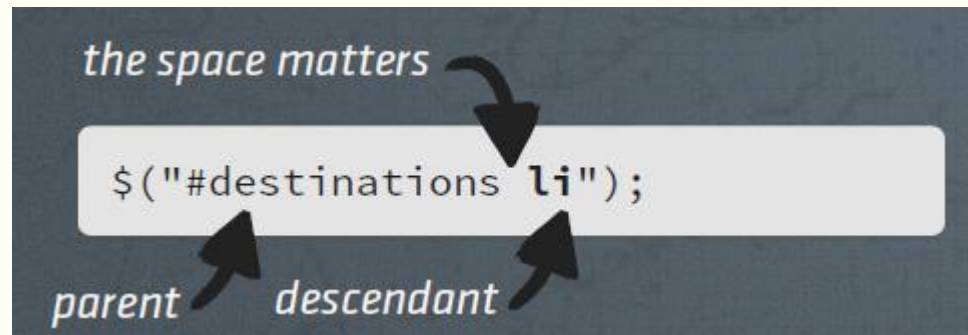
- We can find elements by ID or Class



Selecting descendants

- How do we find only the `` elements that are children of the “destinations” ``?

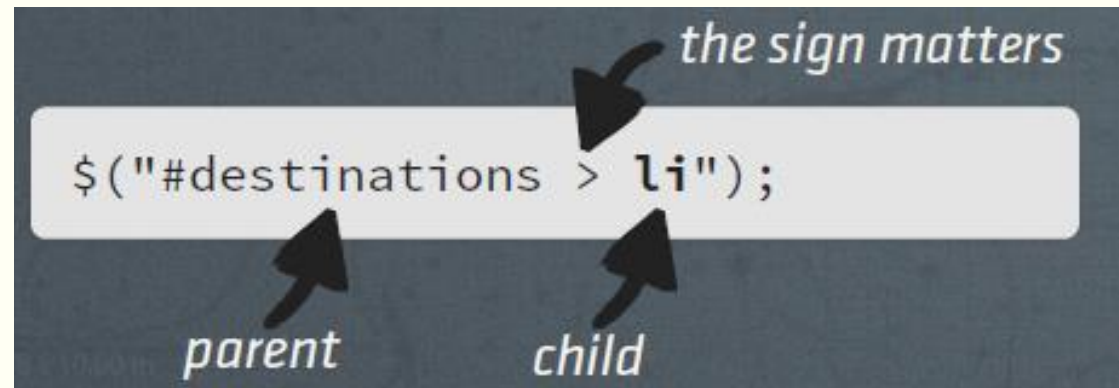
```
<ul id="destinations">  
  <li>Delhi</li>  
  <li>Bangalore</li>  
  <li>Paris</li>  
  <li>London</li>  
  <li>Rio</li>  
</ul>
```



Selecting direct children

- How do we find only the `` elements that are children of the “destinations” ``?

```
<ul id="destinations">
  <li>Delhi</li>
  <li>Bangalore</li>
  <li>
    <ul id="france">
      <li>Paris</li>
    </ul>
  </li>
  <li>London</li>
  <li class="promo">Rio</li>
</ul>
```



Selecting multiple elements

```
<ul id="destinations">
  <li>Delhi</li>
  <li>Bangalore</li>
  <li>
    <ul id="france">
      <li>Paris</li>
    </ul>
  </li>
  <li>London</li>
  <li class="promo">Rio</li>
</ul>
```

the comma matters



```
$(".promo, #france");
```

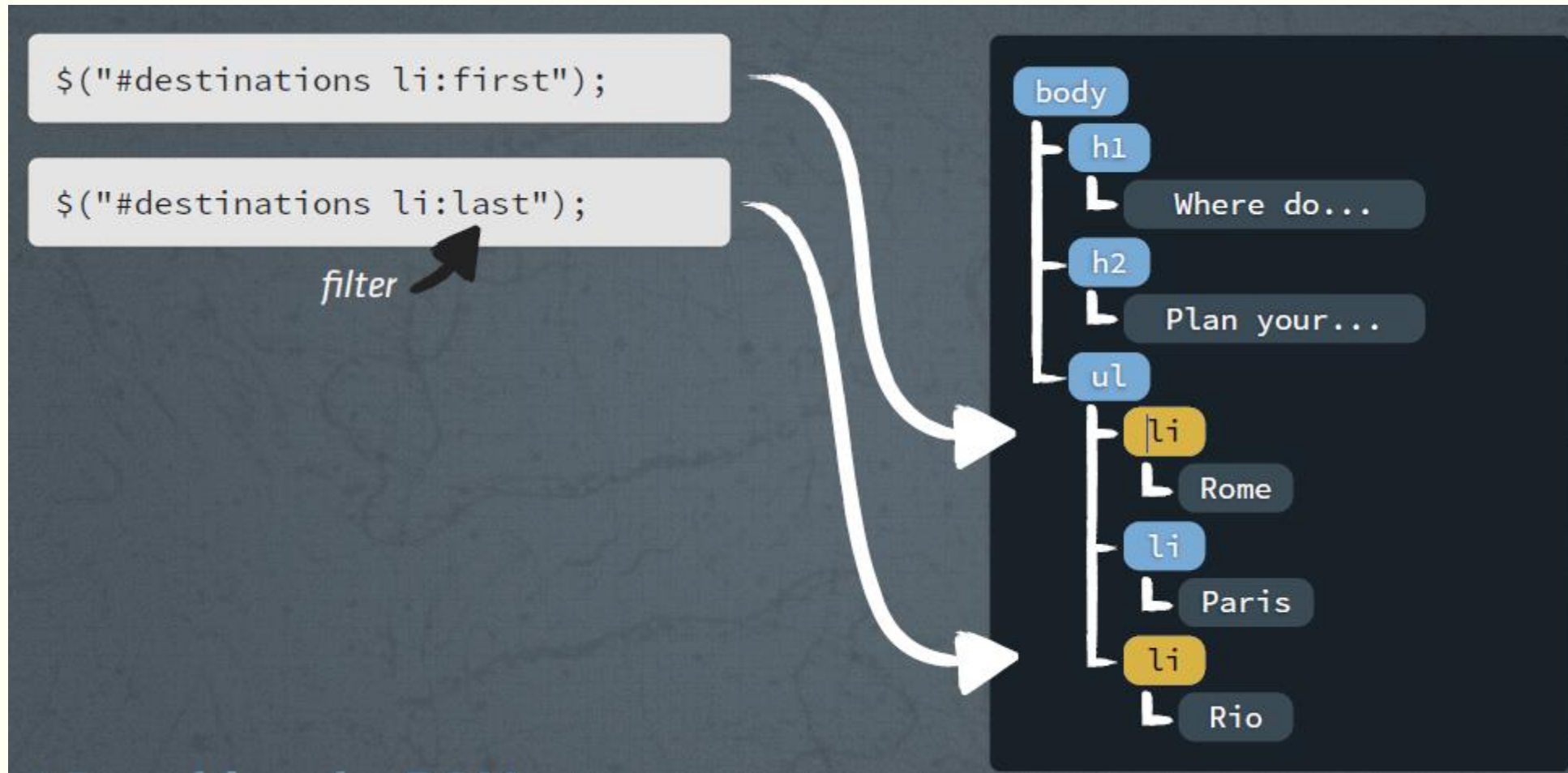
jQuery Filters

Selector	Description
<code>:first</code>	The first match of the page. <code>li a:first</code> returns the first link also under a list item.
<code>:last</code>	The last match of the page. <code>li a:last</code> returns the last link also under a list item.
<code>:first-child</code>	The first child element. <code>li:first-child</code> returns the first item of each list.
<code>:last-child</code>	The last child element. <code>li:last-child</code> returns the last item of each list.
<code>:only-child</code>	Returns all elements that have no siblings.
<code>:nth-child(<i>n</i>)</code>	The <i>n</i> th child element. <code>li:nth-child(2)</code> returns the second list item of each list.

jQuery Filters

Selector	Description
<code>:even</code> and <code>:odd</code>	Even and odd matching elements page-wide. <code>li:even</code> returns every even list item.
<code>:eq(n)</code>	The <i>n</i> th matching element.
<code>:gt(n)</code>	Matching elements after (and excluding) the <i>n</i> th matching element.
<code>:lt(n)</code>	Matching elements before (and excluding) the <i>n</i> th matching element.

jQuery Filters



DOM Traversing

- Filtering by traversing

The diagram illustrates the difference between selecting and traversing elements in jQuery. It features two code snippets at the top, each with a status icon: an orange warning icon for the first and a green checkmark for the second. Below the snippets are two dark blue boxes labeled 'selection' and 'traversal', each with an icon (a house and a person walking, respectively). Arrows point from these boxes to the corresponding parts of the second code snippet. At the bottom, a text line states: 'It takes a bit more code, but it's faster.'

```
$("#destinations li");
```

!

```
$("#destinations").find("li");
```

✓

selection traversal

It takes a bit more code, but it's faster.

DOM Traversing

- `$("li:first");` can be done as `$("li").first();`
- `$("li:last");` can be done as `$("li").last();`
- Walking up the DOM
 - `$("li").first();`
 - `$("li").first().next();`
 - `$("li").first().next().prev();`
 - `$("li").first().parent();`
 - `$("#destinations").children("li");`
 - `children()`, unlike `find()`, only selects direct children

Working with the DOM

- Appending to the DOM
- `$('.vacation').before(price);`
 - *Puts the price node before **.vacation***
- `$('.vacation').after(price);`
 - *Puts the price node after **.vacation***
- `$('.vacation').prepend(price);`
 - *Puts the price node at the top of **.vacation***
- `$('.vacation').append(price);`
 - *Puts the price node at the bottom of **.vacation***

```
$(document).ready(function() {  
    var price = $('<p>From $399.99</p>');  
});
```

ways to add this price node to the DOM

<code>.append(<element>)</code>	<code>.prepend(<element>)</code>
<code>.after(<element>)</code>	<code>.before(<element>)</code>

Working with the DOM

- Removing from the DOM
 - `$('button').remove();`
 - *Removes the **<button>** from the DOM*

Appending to the DOM

- `var price = $('<p>Starting from $399.99</p>');`
 - `price.appendTo($('.vacation'));`
- Methods:
 - `.appendTo(<element>)`
 - `.prependTo(<element>)`
 - `.insertBefore(<element>)`
 - `.insertAfter(<element>)`

jQuery Interactions [Events]

- jQuery Object methods
 - .on(<event>, <event handler>)



- On With a Selector



jQuery Interactions [Events]

- MouseEvents

- click
- dblclick
- focusin
- focusout
- mousedown
- mouseup
- mousemove
- mouseout
- mouseover
- mouseleave
- Mouseenter

- KeyboardEvents

- keypress
- keydown
- keyup

- Form events

- blur
- focus
- select
- change
- submit

jQuery's this

What is "this"?

- In many object-oriented programming languages, this (or self) is a keyword which can be used in instance methods to refer to the object on which the currently executing method has been invoked.

```
<div class="someDivList">  
  <div class="divItem">A div with some text with class</div>  
  <div class="divItem">Another div with class</div>  
  <div>This div doesn't have any class</div>  
  <div class="divItem">One more div with class</div>  
</div>
```

```
$('.someDivList .divItem').each(function() {  
  $(this).css('background', 'lightblue');  
});
```

jQuery's this

```
.someLinksList a {  
    display: block;  
    background: #d5d5d5;  
    color: #000;  
    text-decoration: none;  
    padding: 5px;  
    margin-bottom: 5px;  
}  
  
.someLinksList a.hover {  
    color: #FFF;  
    background: green;  
}
```

```
<div class="someLinksList">  
    <a href="#">A Simple Link</a>  
    <a href="#">Another Simple Link</a>  
    <a href="#">One More Simple Link</a>  
</div>
```

```
$('.someLinksList > a').hover(function() {  
    $(this).toggleClass('hover');  
});
```

Using .closest(<selector>)

■ .closest()

- For each element in the set, get the first element that matches the selector by testing the element itself and traversing up through its ancestors in the DOM tree.
- **`$(this).closest('.vacation').append(price);`**

Event object stopPropagation()

- *The browser will still handle the click event but will prevent it from “bubbling up” to each parent node.*

```
$(document).ready(function() {  
    $('.vacation').on('click', '.expand',  
    function(event) {  
        event.stopPropagation();  
        $(this).closest('.vacation')  
            .find('.comments')  
            .fadeToggle();  
    });  
})
```

Event object preventDefault()

- *The click event will “bubble up” but the browser won’t handle it*

```
$(document).ready(function() {  
    $('.vacation').on('click', '.expand',  
    function(event) {  
        event.preventDefault();  
        $(this).closest('.vacation')  
            .find('.comments')  
            .fadeToggle();  
    });  
})
```

jQuery CSS

■ jQuery Object Methods for CSS

- `.css(<attr>, <value>)`
- `.css(<attr>)`
- `.css(<object>)`
- `.addClass(<class>)`
- `.toggleClass(<class>)`
- `.removeClass(<class>)`
- `.hasClass(<class>)`
- **Examples:**
 - `$(this).css({'background-color': '#252b30', 'border-color': '1px solid #967'});`
 - `$(this).addClass('highlighted');`
 - `$(this).toggleClass('highlighted');`
 - `$(this).hasClass('highlighted') //Returns true or false`

jQuery Effects

- **.animate()**

- Perform a custom animation of a set of CSS properties.
- Example:
 - `$(this).animate({'top': '-10px'});`

- **.fadeIn()**

- Display the matched elements by fading them to opaque.

- **.fadeOut()**

- Hide the matched elements by fading them to transparent.

- **.hide()**

- Hide the matched elements.

- **.show()**

- Display the matched elements.

- **.slideDown()**

Display the matched elements with a sliding motion.

- **.slideUp()**

Hide the matched elements with a sliding motion.

- **.slideToggle()**

Display or hide the matched elements with a sliding motion.

AJAX and jQuery

The diagram illustrates the jQuery AJAX method with handwritten annotations. The code is as follows:

```
$.ajax({  
  url: "my_page.html"  
  success: function(data) {  
  }  
});
```

Annotations with arrows pointing to the code:

- The jQuery shortcut** points to `$.ajax({`
- The jQuery ajax method** points to the opening curly brace of the object.
- The URL of what you want to GET via Ajax** points to `url: "my_page.html"`
- Run this function if the Ajax method is successful. We'll put more code in here in a bit.** points to the `success: function(data) {` block.
- The data returned from the Ajax call** points to the `data` parameter in the success function.

Options for the \$.ajax() utility function

Name	Type	Description
url	String	The URL for the request.
type	String	The HTTP method to use. Usually either POST or GET. If omitted, the default is GET.
data	Object	An object whose properties serve as the query parameters to be passed to the request. If the request is a GET, this data is passed as the query string. If a POST, the data is passed as the request body. In either case, the encoding of the values is handled by the \$.ajax() utility function.
dataType	String	A keyword that identifies the type of data that's expected to be returned by the response. This value determines what, if any, post-processing occurs upon the data before being passed to callback functions. The valid values are as follows:
contentType	String	The content type to be specified on the request. If omitted, the default is <i>application/x-www-form-urlencoded</i> , the same type used as the default for form submissions.
success	Function	A function invoked if the response to the request indicates a success status code. The response body is returned as the first parameter to this function and formatted according to the specification of the dataType property. The second parameter is a string containing a status value—in this case, always <i>success</i> .
error	Function	A function invoked if the response to the request returns an error status code. Three arguments are passed to this function: the XHR instance, a status message string (in this case, always <i>error</i>), and an optional exception object returned from the XHR instance.

async	Boolean	If specified as <i>false</i> , the request is submitted as a synchronous request. By default, the request is asynchronous.
-------	---------	--

The ajax() options

```
// Using the core $.ajax() method
$.ajax({
    // the URL for the request
    url: "post.php",
    // the data to send (will be converted to a query string)
    data: { name: "Banu Prakash", email: "banuprakashc@yahoo.co.in"},
    // whether this is a POST or GET request
    type: "GET",
    // the type of data we expect back
    dataType: "json",
    // code to run if the request succeeds;
    // the response is passed to the function
    success: function( json ) {
        $( "<h1/>" ).text( json.title ).appendTo( "body" );
        $( "<div class=\"content\"/>" ).html( json.html ).appendTo( "body" );
    },
    // code to run if the request fails; the raw request and
    // status codes are passed to the function
    error: function( xhr, status ) {
        alert( "Sorry, there was a problem!" );
    },
    // code to run regardless of success or failure
    complete: function( xhr, status ) {
        alert( "The request is complete!" );
    }
});
```

Shortcuts for AJAX

\$.get

- `$.get("url", dataObj, someFuncnt)`
- `$.ajax({url: "url", data: dataObj, success: someFuncnt});`

\$.post

- `$.post("url", dataObj, someFuncnt)`
- `$.ajax({url: "url", data: dataObj, success: someFuncnt, type: "post"});`

\$.getJSON

- `$.getJSON("url", dataObj, someFuncnt)`
- `$.ajax({url: "url", data: dataObj, success: someFuncnt, dataType: "json"});`

Note

- get and post take the type as an optional fourth argument

Example \$.get Vs. \$.getJSON

```
$(function() {  
    $('#emailDropdown').change(function() {  
        var emailVal = $(this).val();  
        /*  
        $.get('EmployeeJsonSer', {email : emailVal}, function(data){  
            $('#detailsDisplay').html(data.name + "," +  
                data.email + "," + data.age);  
        }, "json");  
        */  
        $.getJSON('EmployeeJsonSer', {email : emailVal}, function(data){  
            $('#detailsDisplay').html(data.name + "," +  
                data.email + "," + data.age);  
        });  
    });  
});
```

Simplifying Inserting Results into HTML: the “load” Function

- `$("#result-area-id").load("url");`
- `$("#result-area-id").load("url", data);`
- `$("#result-area-id").load("url", data, handlerFunction);`

Format the data before you send it.

- jQuery offers two form helper methods for serializing data: `serialize` and `serializeArray`.

`serialize`

```
<form id="my_form">
  <input type="text" name="a" value="1" />
  <input type="text" name="b" value="2" />
  <input type="hidden" name="c" value="3" />
</form>
```

```
$("#my_form").serialize();
```

`a=1&b=2&c=3`

`serializeArray`

```
<form id="my_form">
  <input type="text" name="a" value="1" />
  <input type="hidden" name="c" value="3" />
</form>
```

```
$("#my_form:input").serializeArray();
```

```
[ {
  name: "a",
  value: "1"
},
{
  name: "c",
  value: "3"
}
]
```

Example

```
$(function() {  
    $('#btn').click(function() {  
        var data = new Object();  
        data.email = $("#email").val();  
        data.name = $("#name").val();  
        data.age = $("#age").val();  
        var jsonText = JSON.stringify(data);  
        $.ajax( { url: 'EmployeeJsonSer',  
                  data: jsonText,  
                  type: "post",  
                  success: function(retData){  
                      var content = "Employee Details <br />";  
                      $.each(retData,  
                          function(index, employee) {  
                              content += employee.name + "," +  
                                  employee.email + "," +  
                                  employee.age + "<br />";  
                          });  
                      $('#detailsDisplay').html(content);  
                  },  
                  dataType: 'json',  
                  contentType: 'application/json'  
                });  
    });  
});
```