Computer Science Principles

Web Programming

JavaScript Programming Essentials

CHAPTER 9: DOM AND JQUERY

DR. ERIC CHOU

IEEE SENIOR MEMBER

Overview

LECTURE 1

Objectives

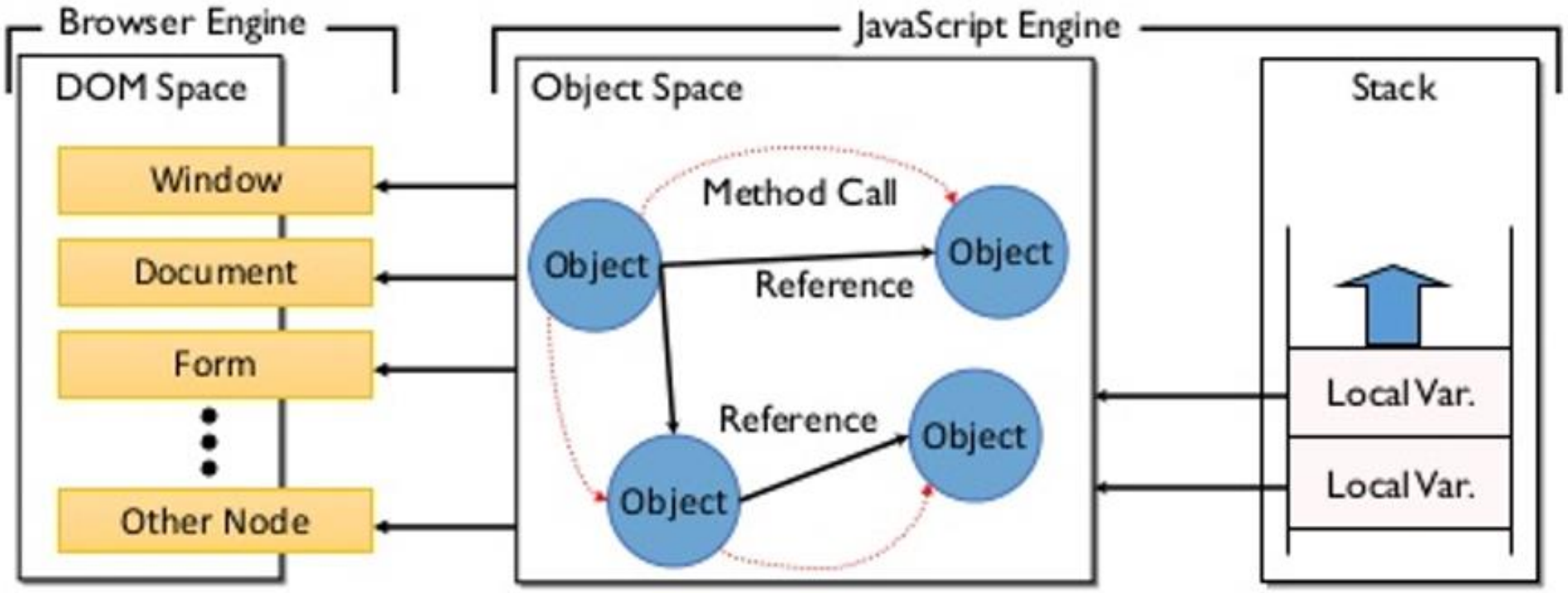
- So far, we've been using JavaScript to do relatively simple things like print text to the browser console or display an alert or prompt dialog.
- But you can also use JavaScript to manipulate (control or modify) and interact with the HTML you write in web pages. In this chapter, we'll discuss two tools that will allow you to write much more powerful JavaScript: the **DOM** and **jQuery**.

DOM

- The **DOM**, or document object model, is what allows JavaScript to access the content of a web page. Web browsers use the **DOM** to keep track of the elements on a page (such as paragraphs, headings and other **HTML** elements), and JavaScript can manipulate **DOM** elements in various ways.
- For example, you'll soon see how you can use JavaScript to replace the main heading of the **HTML** document with input from a prompt dialog.

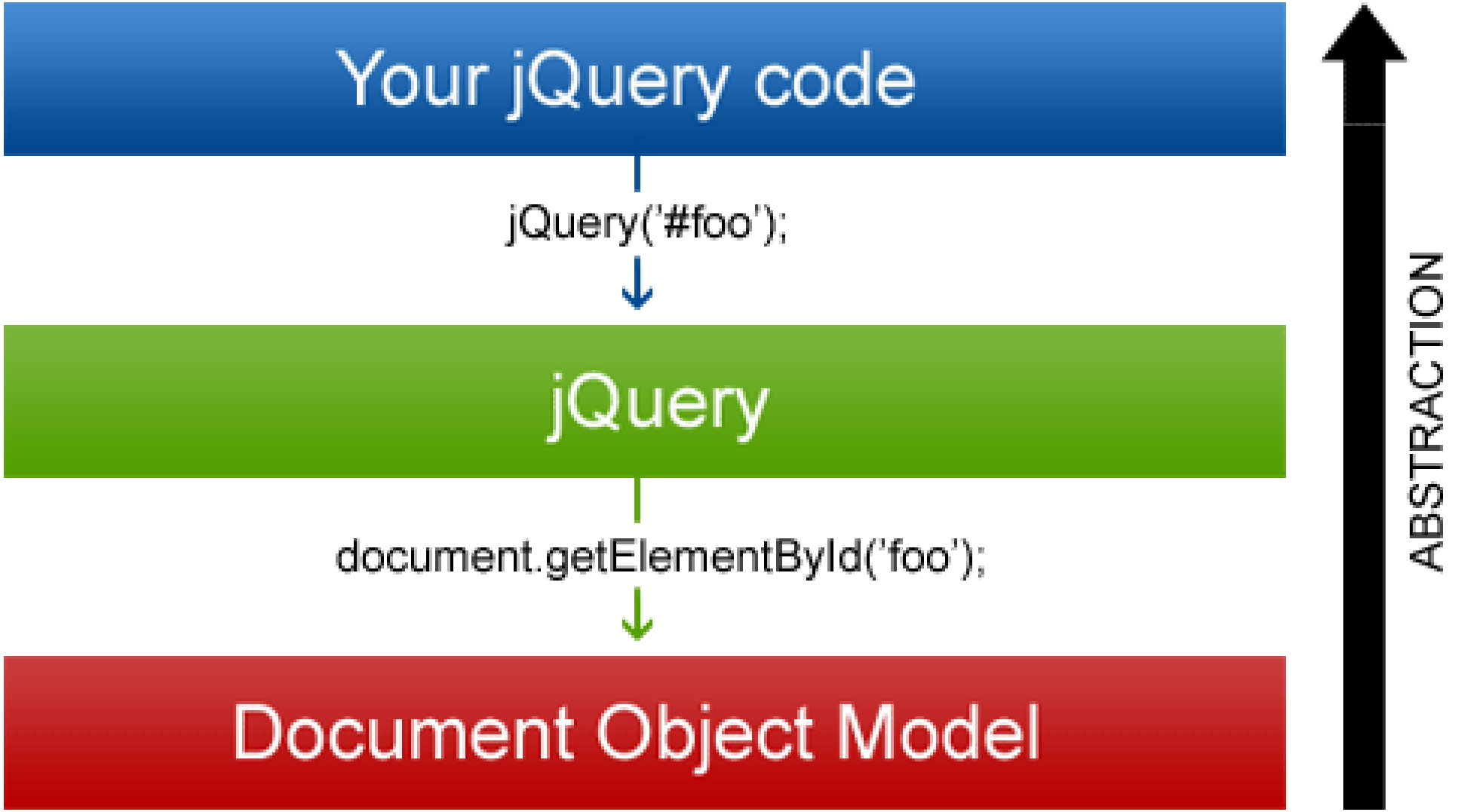
JavaScript Memory Model

- DOM Space: the space where the Document Object Model representing the HTML's layered structure is represented.
- Object Space: the space where all JavaScript objects are located.
- Stack: short-term memory



jQuery

- We'll also look at a useful tool called **jQuery**, which makes it much easier to work with the **DOM**.
- **jQuery** gives us a set of functions that we can use to choose which elements to work with and to make changes to those elements.
- In this chapter, we'll learn how to use the **DOM** and **jQuery** to edit existing **DOM** elements and create new **DOM** elements, giving us full control over the content of our web pages from JavaScript. We'll also learn how to use **jQuery** to animate **DOM** elements — for example, fading elements in and out.


Your jQuery code
jQuery('#foo');
jQuery
document.getElementById('foo');
Document Object Model
ABSTRACTION

Selecting DOM Elements

LECTURE 1

Selecting DOM Elements

- When you load an **HTML** document into a browser, the browser converts the elements into a tree-like structure. This tree is known as the **DOM** tree.
- Figure 9-1 shows a simple **DOM** tree — the same tree we used in Chapter 5 to illustrate the hierarchy of **HTML**.
- The browser gives JavaScript programmers a way to access and modify this tree structure using a collection of methods called the **DOM**.

Selecting DOM Elements

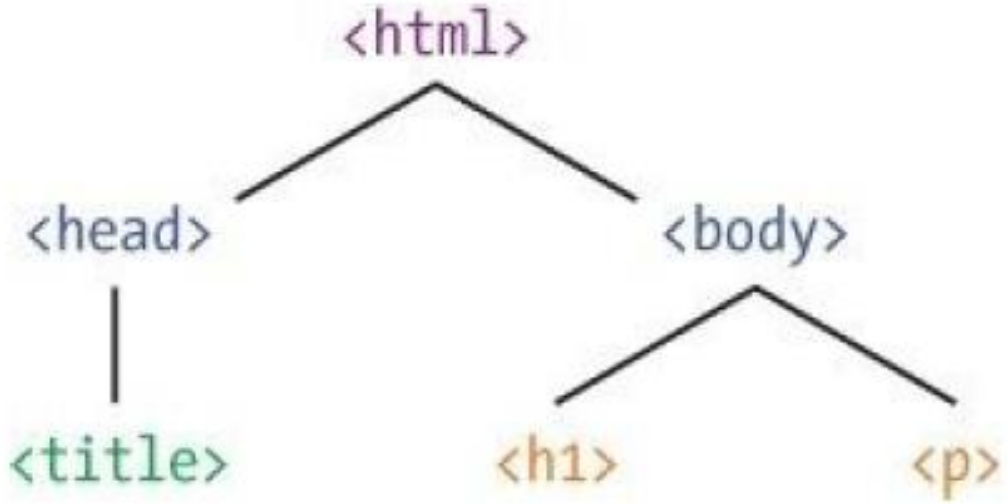


Figure 9-1. The DOM tree for a simple HTML document, like the one we made in Chapter 5

Using id to Identify Elements

- The HTML id attribute lets you assign a unique name, or **identifier**, to an HTML element. For example, this h1 element has an id attribute:

```
<h1 id="main-heading">Hello world!</h1>
```

- In this example, the id of "main-heading" will let us identify, and eventually change, this particular heading without affecting other elements or even other h1 headings.

Selecting an Element Using getElementById

- Having uniquely identified an element with id (each id must have a unique value), we can use the **DOM** method document.getElementById to return the "main-heading" element:

```
var headingElement =
    document.getElementById("main-heading");
```

- By calling document.getElementById("main-heading"), we tell the browser to look for the element with the id of "main-heading". This call returns a DOM object that corresponds to the id, and we save this DOM object to the variable **headingElement**.

Selecting an Element Using getElementById

- Once we've selected an element, we can manipulate it with JavaScript. For example, we can use the innerHTML property to retrieve and replace the text inside the selected element:

```
headingElement.innerHTML;
```

- This code returns the HTML contents of **headingElement** — the element we selected using **getElementById**. In this case, the content of this element is the text **Hello world!** that we entered between the <h1> tags.

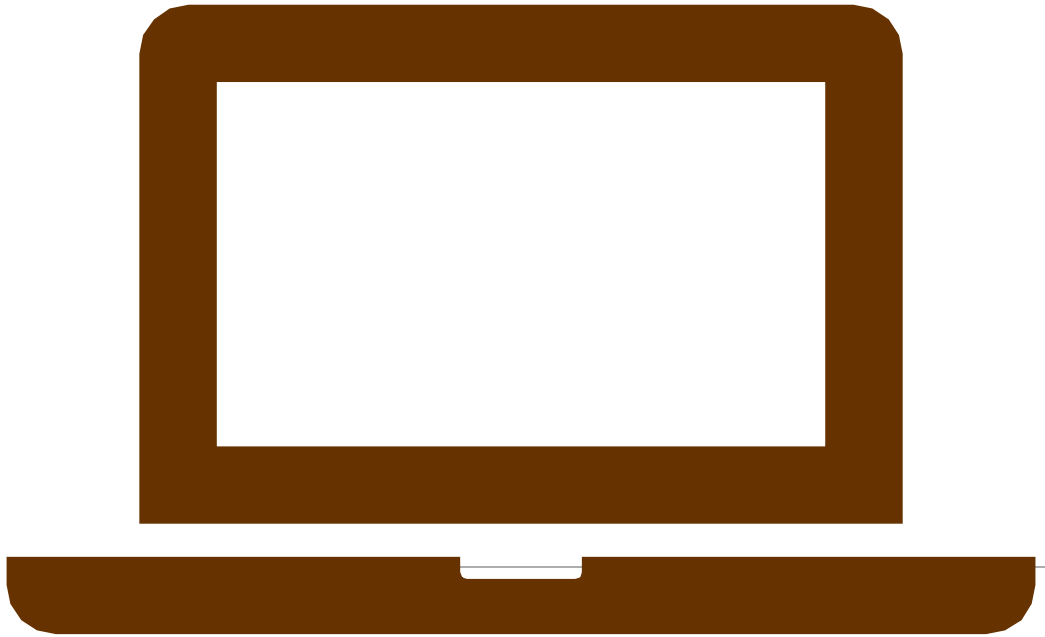
Replacing the Heading Text Using the DOM

Here's an example of how to replace heading text using the DOM. First, we create a new HTML document called *dom.html* containing this code:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Playing with the DOM</title>
  </head>
  <body>
  <h1 id="main-heading">Hello world!</h1>
    <script>
❶     var headingElement = document.getElementById("main-heading");
❷     console.log(headingElement.innerHTML);
❸     var newHeadingText = prompt("Please provide a new heading:");
❹     headingElement.innerHTML = newHeadingText;
    </script>
  </body>
</html>
```

Replacing the Heading Text Using the DOM

- At ❶ we use document.getElementById to get the h1 element (with the id of "main-heading") and save it into the variable headingElement. At ❷ we print the string returned by headingElement.innerHTML, which prints Hello world! to the console. At ❸ we use a prompt dialog to ask the user for a new heading and save the text the user enters in the variable newHeadingText.
- Finally, at ❹ we set the innerHTML property of headingElement to the text saved in newHeadingText.
- When you load this page, you should see a prompt dialog like the one shown in Figure 9-2.



Demonstration Program

HEADING.HTML

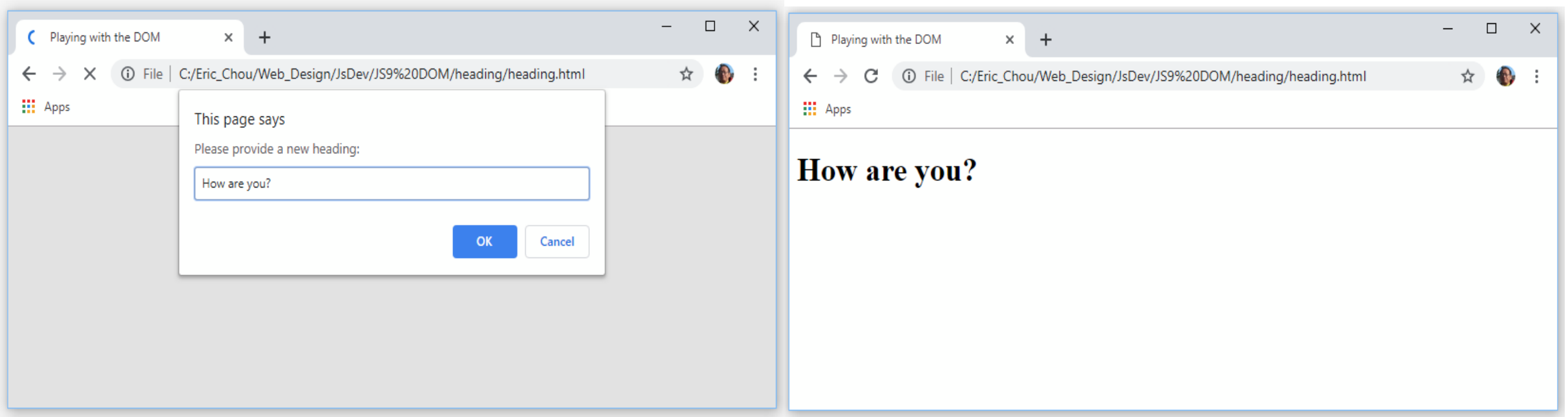


Figure 9-2. Our page with the dialog open

Replacing the Heading Text Using the DOM

- Enter the text **JAVASCRIPT IS AWESOME** into the dialog and click **OK**. The heading should update instantly with the new text, as shown in Figure 9-3.

- *Figure 9-3. Our page after the heading change*

- Using the innerHTML property, we can change the content of any DOM element using JavaScript.

Work with the DOM Tree

LECTURE 1

Using jQuery to Work with the DOM Tree

- The built-in **DOM** methods are great, but they're not very easy to use. Because of this, many developers use a set of tools called **jQuery** to access and manipulate the **DOM** tree. **jQuery** is a JavaScript library
  — a collection of related tools (mostly functions) that gives us, in this case, a simpler way to work with
- **DOM** elements. Once we load a library onto our page, we can use its functions and methods in addition to those built into JavaScript and those provided by the browser.

Loading jQuery on Your HTML Page

- To use the jQuery library, we first tell the browser to load it with this line of HTML:

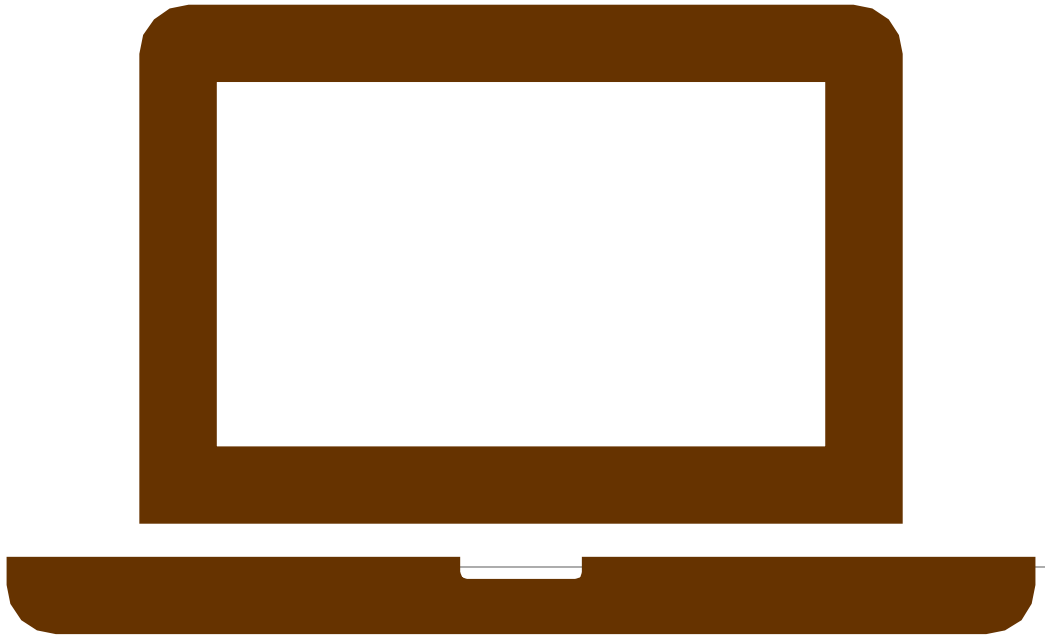
```
<script src="https://code.jquery.com/jquery-2.1.0.js">
</script>
```

- Notice that the <script> tag here has no contents, and it has a src attribute. The src attribute lets us insert a JavaScript file into our page by including its *URL* (web address). In this case, *https://code.jquery.com/jquery-2.1.0.js* is the URL for a specific version of jQuery (version 2.1.0) on the jQuery website.
- To see the jQuery library, visit that URL; you'll see the JavaScript that will be loaded when this <script> tag is added. The entire library is over 9,000 lines of complicated JavaScript, though, so don't expect to understand it all right now!

Replacing the Heading Text Using jQuery

- In Replacing the Heading Text Using the DOM, you learned how to replace text using the built-in **DOM** methods. In this section, we'll update that code to use **jQuery** to replace the heading text instead. Open **dom.html** and make the changes shown.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Playing with the DOM</title>
    </head>
    <body>
        <h1 id="main-heading">Hello world!</h1>
❶       <script src="https://code.jquery.com/jquery-2.1.0.js"></script>
        <script>
            var newHeadingText = prompt("Please provide a new heading:");
❷           $("#main-heading").text(newHeadingText);
        </script>
    </body>
</html>
```



Demonstration Program

HEADING2.HTML

Replacing the Heading Text Using jQuery

- At ❶ we add a new **<script>** tag to the page to load **jQuery**. With **jQuery** loaded, we use the **jQuery** function **$** to select an **HTML** element.
- The **$** function takes one argument, called a *selector string*, which tells **jQuery** which element or elements to select from the **DOM** tree. In this case, we entered **"#main-heading"** as the argument. The **#** character in a selector string means **“ID,”** so our selector string **"#main-heading"** means **“the element with an id of main-heading.”**

Replacing the Heading Text Using jQuery

- The $ function returns a **jQuery** object that represents the elements you selected. For example,

  `$("#main-heading")` returns a jQuery object for the **h1** element (which has an **id** of "mainheading").

- We now have a **jQuery** object representing the h1 element. We can modify its text by calling the text method on the **jQuery** object at ❷, passing in the new text for that element, and replacing the text of the heading with the user input saved to the variable **newHeadingText**.

- As before, when you load this page, a dialog should prompt you to enter replacement text for the old text in the **h1** element.

New Elements

LECTURE 1

Creating New Elements with jQuery

- In addition to manipulating elements with **jQuery**, we can also use jQuery to create new elements and add them to the **DOM** tree. To do so, we call append on a jQuery object with a string containing **HTML**.
- The append method converts the string to a **DOM** element (using the **HTML** tags in the string) and adds the new element to the end of the original one.
- For example, to add a p element to the end of the page, we could add this to our **JavaScript**:

```
$("body").append("<p>This is a new paragraph</p>");
```

Creating New Elements with jQuery

- The first part of this statement uses the **$** function with the selector string "body" to select the body of our **HTML** document. The selector string doesn't have to be an id. The code **$("body")** selects the body element. Likewise, we could use the code **$("p")** to select all the p elements.
- Next, we call the append method on the object returned by **$("body")**. The string passed to append is turned into a **DOM** element, and it is added inside the body element, just before the closing tag. Figure 9-4 shows what our revised page would look like.

Figure 9-4. Our document with a new element

Creating New Elements with jQuery

- We could also use append to add multiple elements in a for loop like this:

```
for (var i = 0; i < 3; i++) {
 var hobby = prompt("Tell me one of your hobbies!");
 $("body").append("<p>" + hobby + "</p>");
}
```

- This loops three times. Each time through a loop, a prompt appears, asking users to enter one of their hobbies. Each hobby is then put inside a set of **<p>** tags and passed to the append method, which adds the hobby to the end of the body element. Try adding this code to your **dom.html** document, and then load it in a browser to test it. It should look like Figure 9-5.

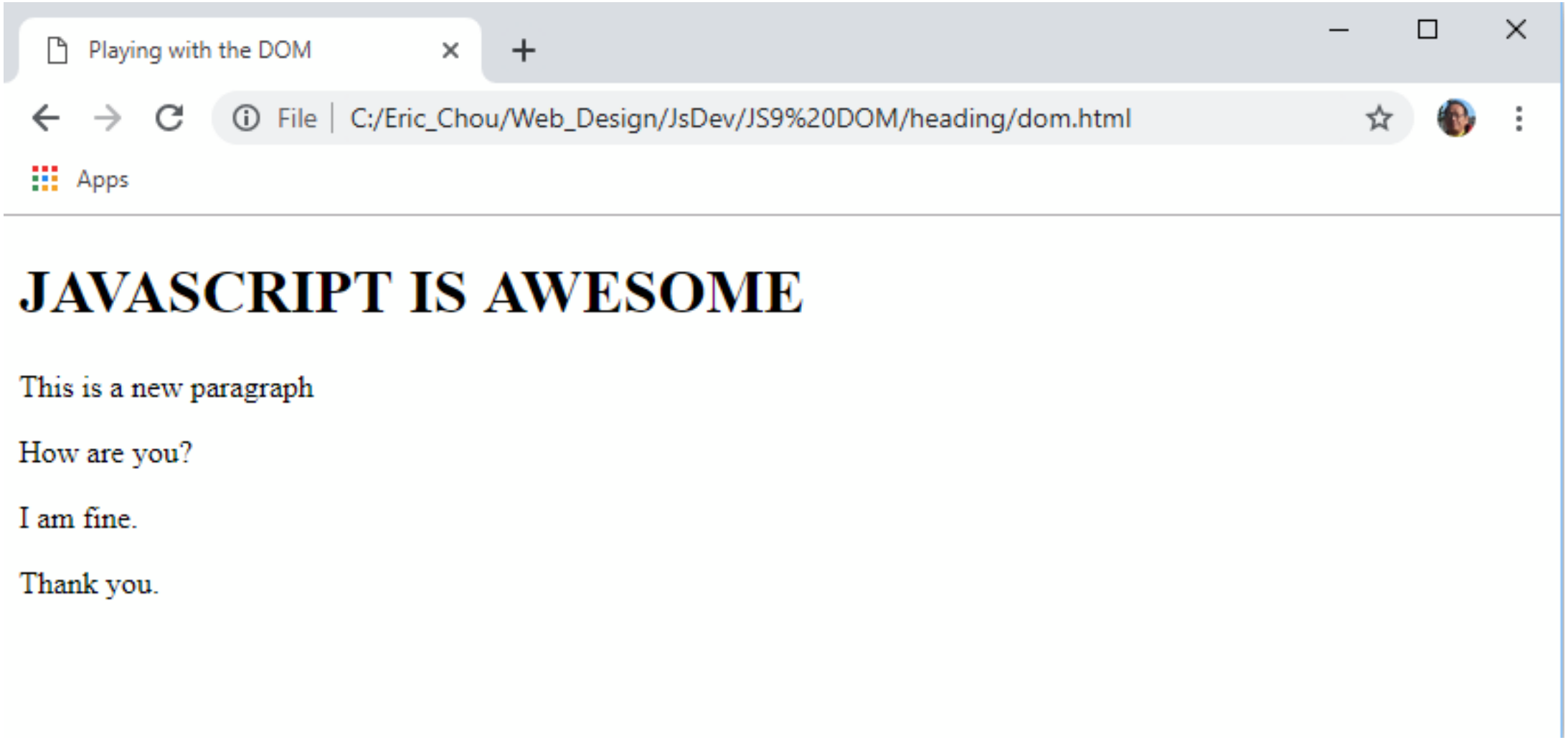


Figure 9-5. Extra elements added in a loop

Animating Elements

LECTURE 1

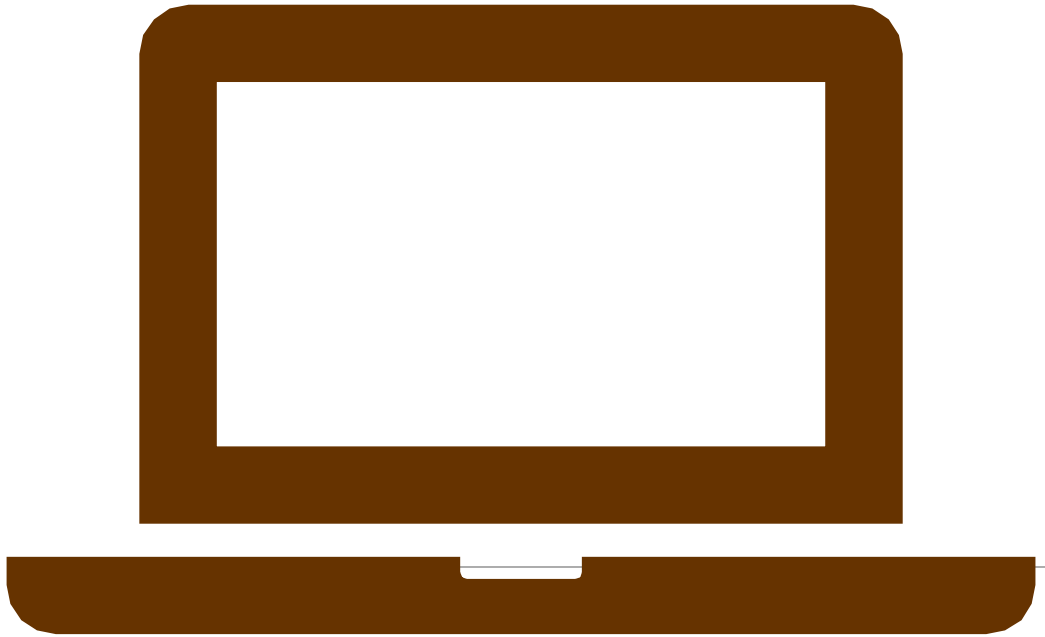
Animating Elements with jQuery

- Lots of websites use animations to show and hide content. For example, if you were adding a new paragraph of text to your page, you might want to fade it in slowly so it doesn't appear all of a sudden.
- **jQuery** makes it easy to animate elements. For example, to fade an element out, we can use the fadeout method. To test this method, replace the contents of the second script element in **dom.html** with this:

```
$("h1").fadeOut(3000);
```

Animating Elements with jQuery

- We use the **$** function to select all h1 elements. Because **dom2.html** has only one **h1** element (the heading containing the text Hello world!), that heading is selected as a **jQuery** object. By calling **.fadeOut(3000)** on this **jQuery** object, we make the heading fade away until it disappears, over the course of 3 seconds. (The argument to **fadeOut** is in milliseconds, or thousandths of a second, so entering 3000 makes the animation last 3 seconds.)
- As soon as you load the page with this code, the **h1** element should start to fade away.



Demonstration Program

DOM2.HTML

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Playing with the DOM</title>
5  </head>
6  <body>
7      <h1 id="main-heading">Hello world!</h1>
8      <script src="https://code.jquery.com/jquery-3.4.0.js"></script>
9      <script>
10       $("h1").fadeOut(3000);
11     </script>
12   </body>
13 </html>
```

Chaining jQuery Animations

LECTURE 1

Chaining jQuery Animations

- When you call a method on a **jQuery** object, the method usually returns the original object that it was called on. For example, $("h1") returns a **jQuery** object representing all **h1** elements, and
- $("h1").fadeOut(3000) returns the *same* **jQuery** object representing all **h1** elements. To change the text of the h1 element and fade it out, you could enter:

```
$("h1").text("This will fade out").fadeOut(3000);
```

- Calling multiple methods in a row like this is known as ***chaining***.

Chaining jQuery Animations

- We can chain multiple animations on the same element. For example, here's how we could chain a call to the **fadeOut** and **fadeIn** methods to fade an element out and then immediately fade it in again:

```
$("h1").fadeOut(3000).fadeIn(2000);
```

- The **fadeIn** animation makes an invisible element fade back in. **jQuery** is smart enough to know that when you chain two animations in a row like this, you probably want them to happen one after the other. Therefore, this code fades the **h1** element out over the course of 3 seconds and then fades it back in over 2 seconds.

Chaining jQuery Animations

- **jQuery** provides two additional animation methods similar to fadeOut and fadeIn, called **slideUp** and **slideDown**. The slideUp method makes elements dis appear by sliding them up, and **slideDown** makes them reappear by sliding them down. Replace the second script element in the **dom.html** document with the following, and reload the page to try it out:

```
$("h1").slideUp(1000).slideDown(1000);
```

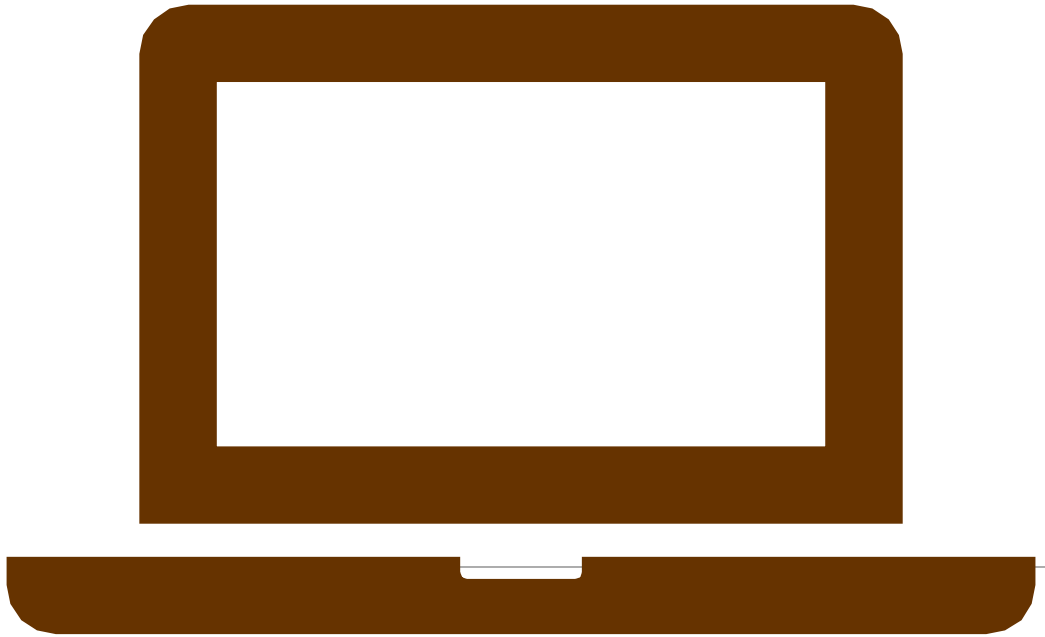
- Here we select the **h1** element, slide it up over 1 second, and then slide it down over 1 second until it reappears.

Activity

- We use **fadeIn** to make invisible elements visible. But what happens if you call **fadeIn** on an element that's already visible or an element that comes *after* the element you're animating?
- For example, say you add a new p element to your **dom.html** document after the heading. Try using **slideUp** and **slideDown** to hide and show the h1 element, and see what happens to the p element. What if you use **fadeOut** and **fadeIn**?
- What happens if you call **fadeOut** and **fadeIn** on the same element without chaining the calls? For example:

```
$("h1").fadeOut(1000);
$("h1").fadeIn(1000);
```

- Try adding the preceding code inside a for loop set to run five times. What happens?
- What do you think the show and hide **jQuery** methods do? Try them out to see if you're right. How could you use hide to fade in an element that's already visible?



Demonstration Program

DOM3.HTML / DOM4.HTML

Summary

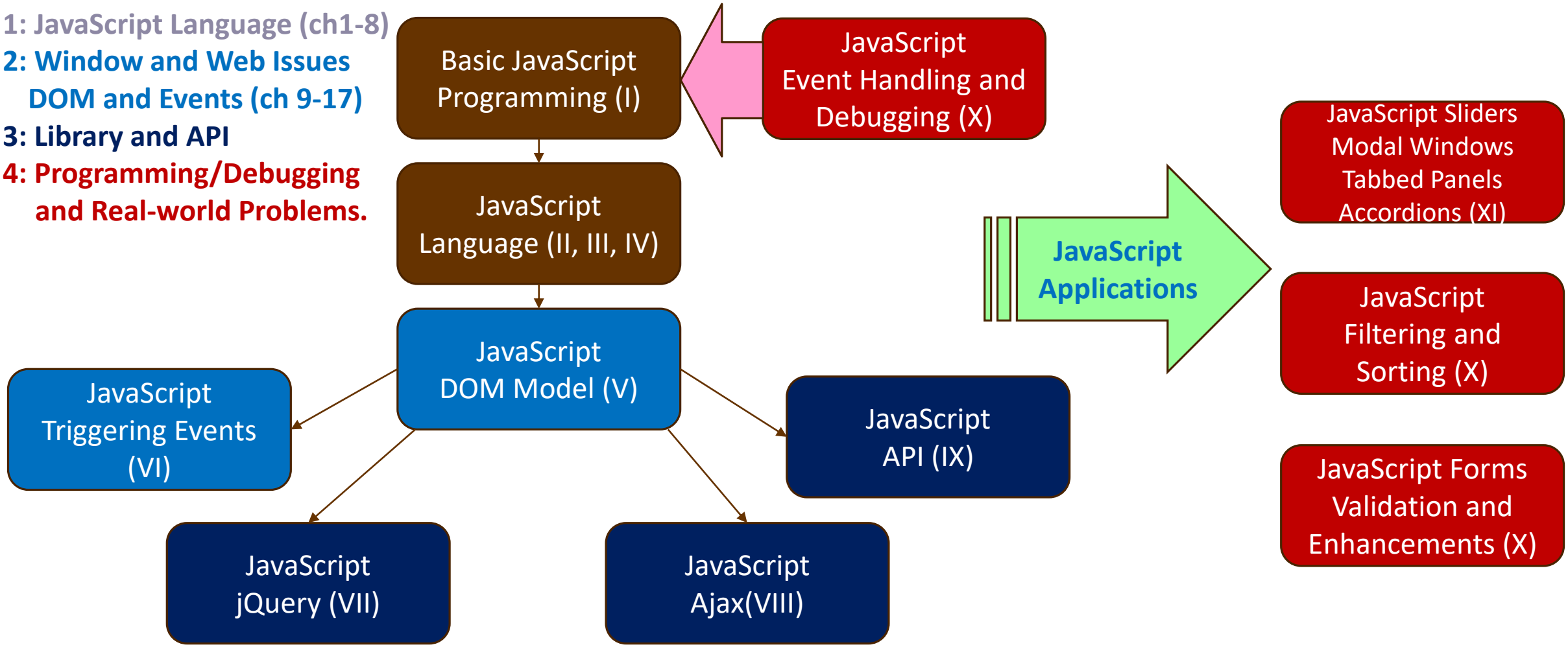
LECTURE 1

Summary

- In this chapter, you learned how to update **HTML** pages using JavaScript by manipulating **DOM** elements. As you've seen, **jQuery** gives us even more powerful ways to select elements and change or even animate them. You also learned a new **HTML** attribute, id, which allows you to give an element a unique identifier.
- In the next chapter, you'll learn how to control when your JavaScript is run — for example, once a timer has run out or when you click a button. We'll also look at how to run the same piece of code multiple times with a time delay in between — for example, updating a clock once every second.

JavaScript and jQuery Course Work

Part 1: JavaScript Language (ch1-8)
Part 2: Window and Web Issues
DOM and Events (ch 9-17)
Part 3: Library and API
Part 4: Programming/Debugging
and Real-world Problems.



jQuery Version

Download Site:

https://jquery.com/download/

Versions:

- Regular version.
- Compressed version.
- Online version