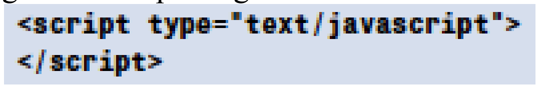
Chapter 2 Programming with JavaScript 13

Running a JavaScript Program 14

- 

  ```
  <script type="text/javascript">
  </script>
  ```
- **<script type="text/javascript" src="example.js"></script>**
- **<script type="text/javascript"src="http://www.example.com/script.js"></script>**
- XHTML and Embedded JavaScript don't Mix

Statements: Bite-sized Chunks for your Browser 17

- Statement one;
- Statement 2.0;

Comments: Bite-sized Chunks Just for You 18

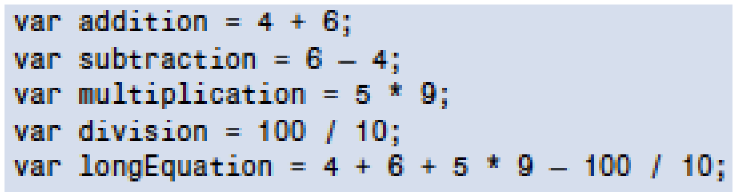
- Single Line //
- Mulitple line /*text*/

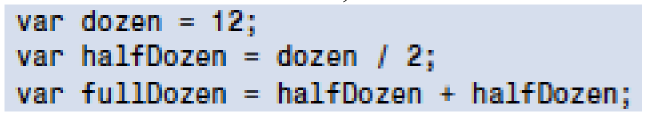
Variables: Storing Data for your Program 19

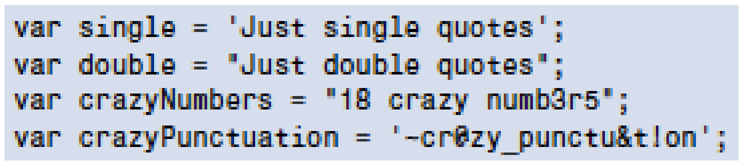
- Declaring Variables
  - var chameleon;
- Assignment Operator
  - Chameleon = "blue";
- Declaring variables and assign variables at the same time
  - Var chameleon = "blue";
- Calling Variable
  - alert(chameleon);
- Variable Rules
  - Alphanumeric
  - No spaces
  - $, _ are allowed
  - case sensitive
- Standards for variables
  - multi_word_variable
  - camel casing

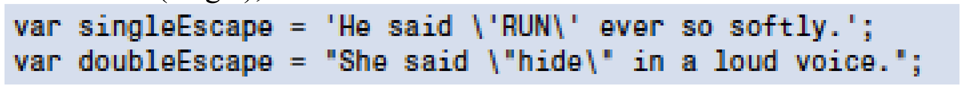
Variable Types: Different Types for Different Data 23

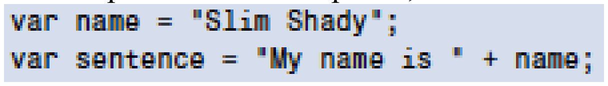
- JavaScript is a loosely type – meaning you don't have to explicitly declared the data type
  - Numbers
    - Whole number – integer or int
      - var whole = 3;
    - Decimal – float
      - var decimal = 3.14;
    - can have negative in the front
      - var whole = -3;
      - var decimal = -3.14;
  - Mathematical Operations
    - Written in natural notation

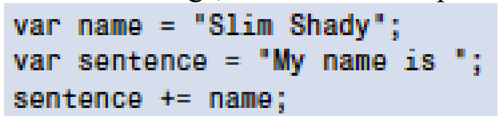
    - 

      ```
      var addition = 4 + 6;
      var subtraction = 6 – 4;
      var multiplication = 5 * 9;
      var division = 100 / 10;
      var longEquation = 4 + 6 + 5 * 9 – 100 / 10;
      ```
    - Operators
      - —+, -, *, and /
    - It follows PMDAS
    - Float vs Integer
      - Float wins (conversion happens automatically)
      - var decimal = 5 / 4;
    - 

      ```
      var dozen = 12;
      var halfDozen = dozen / 2;
      var fullDozen = halfDozen + halfDozen;
      ```
    - incrementing variable
      - age = age + 1;
      - age += 1;
      - Other Method
        - age++; add to the number
        - ++age; increments and then add to the number
      - all results are the same
    - You can *=, -=. /=
  - Strings
    - 

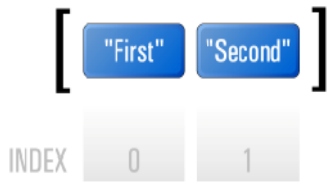
      ```
      var single = 'Just single quotes';
      var double = "Just double quotes";
      var crazyNumbers = "18 crazy numb3r5";
      var crazyPunctuation = '-cr@zy_punctu&t!on';
      ```
    - var single = 'Just single quotes';
      - alert(single);
    - 

      ```
      var singleEscape = 'He said \'RUN\' ever so softly.';
      var doubleEscape = "She said \"hide\" in a loud voice.";
      ```
    - String Operations
      - Concatenate
        - var complete = "com" + "plete";
        - 

          ```
          var name = "Slim Shady";
          var sentence = "My name is " + name;
          ```
      - += operator with strings, but not the ++ operator
        - 

          ```
          var name = "Slim Shady";
          var sentence = "My name is ";
          sentence += name;
          ```
      - String vs numbers
        - String wins
          - var sentence = "You are " + 1337
          - You are 1337
  - Booleans

        -
          ```
          var lying = true;
          var truthful = false;
          ```
- Arrays – for variables that are group together

    -
      ```
      var rack = [];
      rack[0] = "First";
      rack[1] = "Second";
      ```
        - 

        - alert(rack[1]);
            - Second
        - Element – First and Second
    - var rack = ["First", "Second", "Third", "Fourth"];
    - var numberArray = [1, 2, 3, 5, 8, 13, 21, 34];
    - var mixedArray = [235, "Parramatta", "Road"];
    - multi-dimensional array

        -
          ```
          var subArray1 = ["Paris", "Lyon", "Nice"];
          var subArray2 = ["Amsterdam", "Eindhoven", "Utrecht"];
          var subArray3 = ["Madrid", "Barcelona", "Seville"];

          var superArray = [subArray1, subArray2, subArray3];
          ```

        -
          ```
          var city = superArray[0][2];
          ```

          If we translate that statement, starting from the right side, it says:

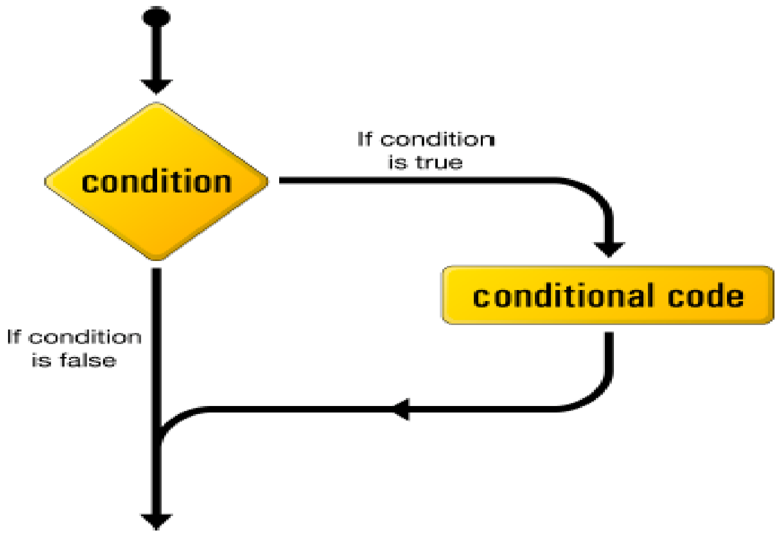
`[2]`	Get the third element ...
`[0]`	of the first array ...
`superArray`	in `superArray` ...
`var city =`	and save that value in a new variable, `city`.
    - To know if there's element inside the array

        -
          ```
          var shortArray = ["First", "Second", "Third"];
          var total = shortArray.length;
          ```
            - total = 3
    - To get the index of the last element in the array
        - var lastItem = shortArray[shortArray.length – 1];
    - To add another element inside the array
        - shortArray[shortArray.length] = "Fourth";
- Associative Array – makes it easy to extract data inside the array

-
  ```
  var postcodes = [];
  postcodes["Armadale"] = 3143;
  postcodes["North Melbourne"] = 3051;
  postcodes["Camperdown"] = 2050;
  postcodes["Annandale"] = 2038;
  ```
    - alert(postcodes["Annandale"]);

Conditions and Loops: Controlling Program Flow 35
Conditions: Making Decisions 36

- if Statements



  -
  ```
  if (condition)
  {
    conditional code;
  }
  ```
  -
    - condition:
      - normally bollean and expressions that evaluates as a bollean
        - true or false
        - var effect = 4 + 6;
      - Comparison Operators

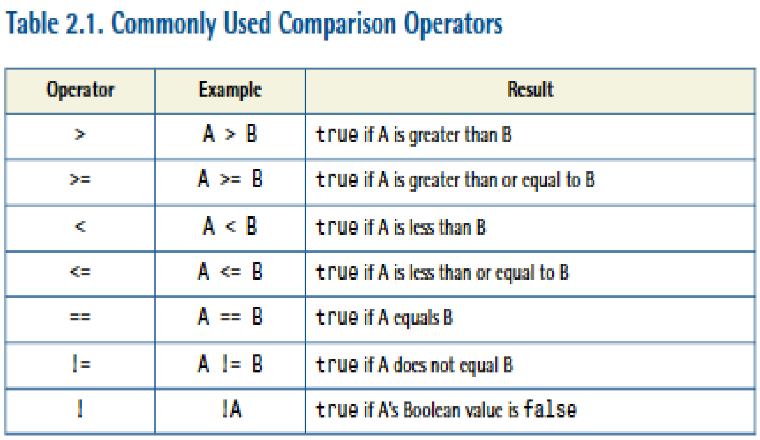
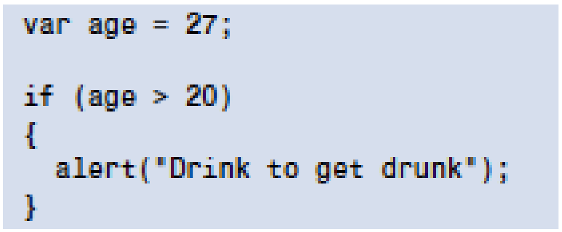
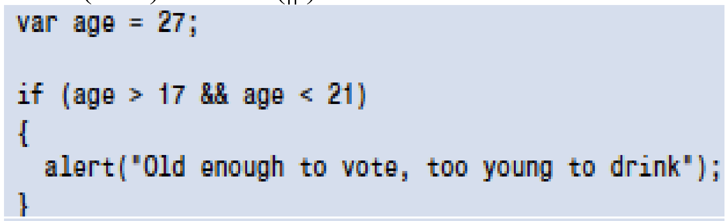


Table 2.1. Commonly Used Comparison Operators

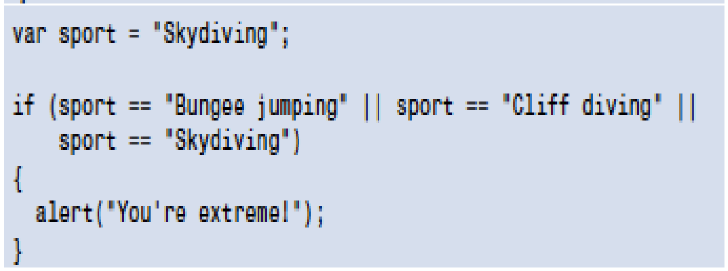
Operator	Example	Result
>	`A > B`	`true` if A is greater than B
>=	`A >= B`	`true` if A is greater than or equal to B
<	`A < B`	`true` if A is less than B
<=	`A <= B`	`true` if A is less than or equal to B
==	`A == B`	`true` if A equals B
!=	`A != B`	`true` if A does not equal B
!	`!A`	`true` if A's Boolean value is `false`

- 

```
var age = 27;

if (age > 20)
{
  alert("Drink to get drunk");
}
```

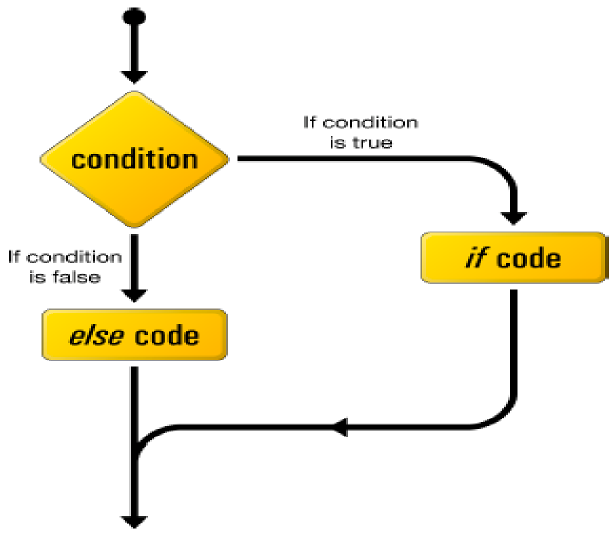
- ▪ Multiple Conditions
  - AND (&&) and OR (||)
  - 

```
var age = 27;

if (age > 17 && age < 21)
{
  alert("Old enough to vote, too young to drink");
}
```

  - 

```
var sport = "Skydiving";

if (sport == "Bungee jumping" || sport == "Cliff diving" ||
    sport == "Skydiving")
{
  alert("You're extreme!");
}
```

- if-else statements



- Figure 2.9. The logical flow of an `if-else` statement

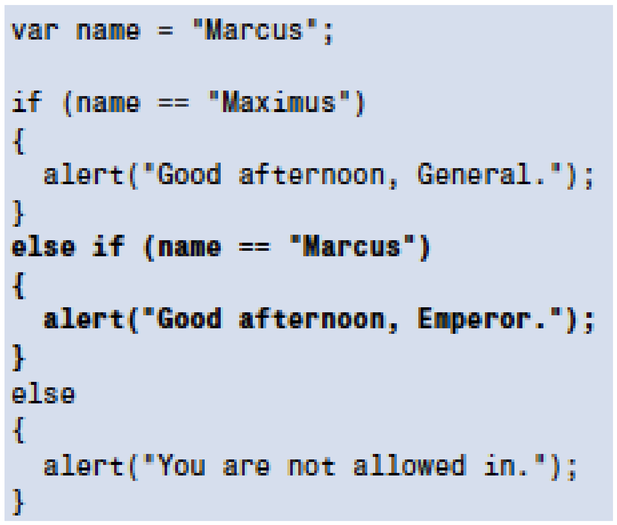
  ```
  if (condition)
  {
    conditional code;
  }
  else
  {
    alternative conditional code;
  }
  ```

-
  ```
  var name = "Marcus";

  if (name == "Maximus")
  {
    alternative("Good afternoon, General.");
  }
  ```

-
  ```
  else
  {
    alert("You are not allowed in.");
  }
  ```

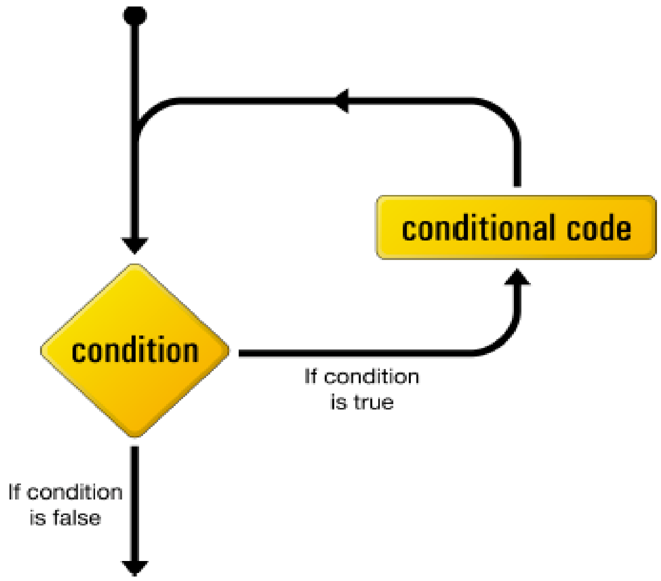
-

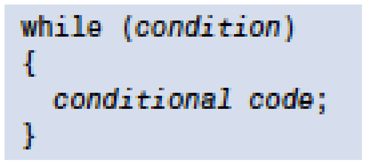
* else-if statements

- 

```
var name = "Marcus";

if (name == "Maximus")
{
  alert("Good afternoon, General.");
}
else if (name == "Marcus")
{
  alert("Good afternoon, Emperor.");
}
else
{
  alert("You are not allowed in.");
}
```

Loops: Minimizing Repetition 43

- repeat a set of actions for as long as a specified condition is true .
- while Loops
  - 

  - 

```
while (condition)
{
  conditional code;
}
```

    - keeps looping while condition is true
  - multiply each number by 2

```
var numbers = [1, 2, 3, 4, 5];
var incrementer = 0;
while (incrementer < numbers.length)
{
  numbers[incrementer] *= 2;
  incrementer++;
}
```

      - variable name incrementer is frequently shortened to i

- do-while loops

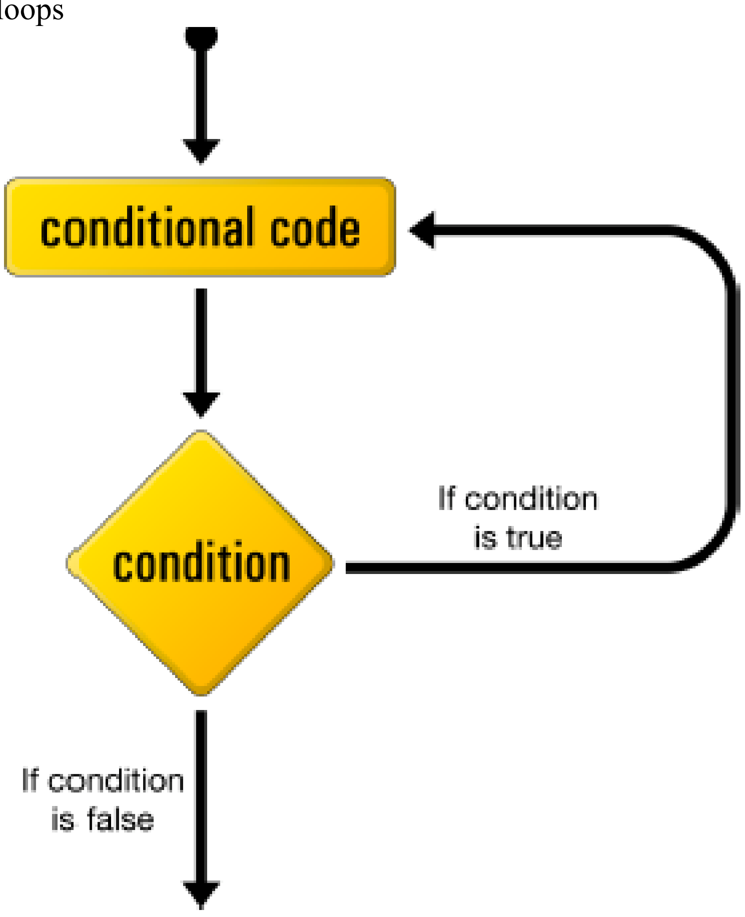
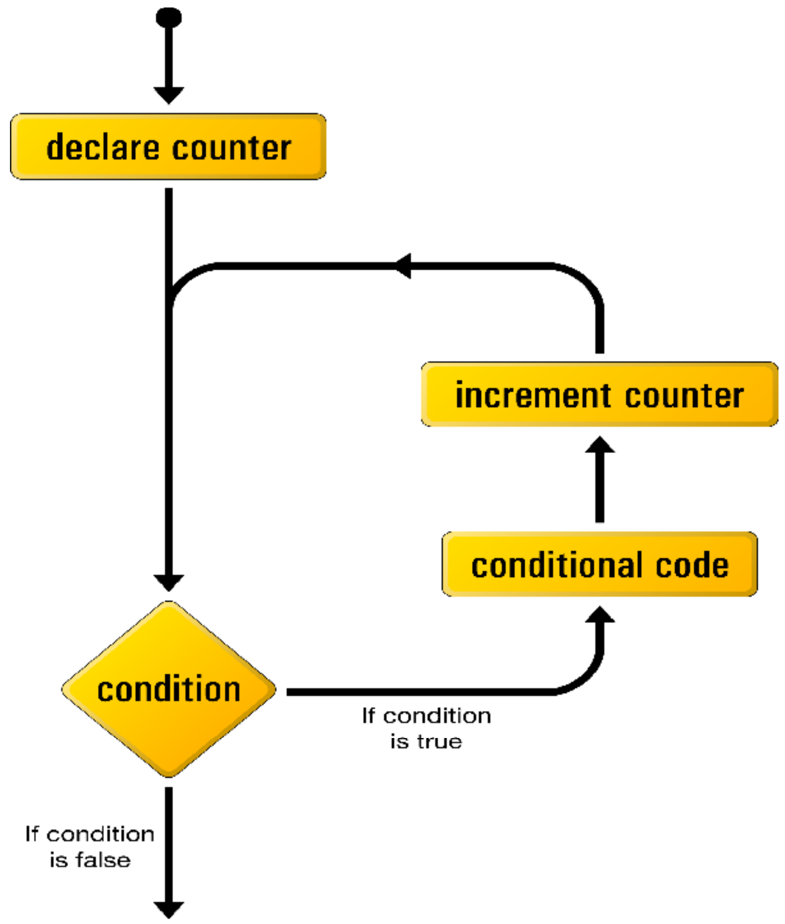


Figure 2.11. The logical flow of a do-while loop

  -
    ```
    do
    {
      conditional code;
    }
    while (condition);
    ```
  -
      - it is the same as while, but conditional code is executed at least once

- for loops



-
- They' re a lot like while loops, but they offer a couple of handy shortcuts

```
var numbers = [1, 2, 3, 4, 5];
var i = 0;
while (i < numbers.length)
{
  numbers[i] *= 2;
  i++;
}
```

-

```
var numbers = [1, 2, 3, 4, 5];

for (var i = 0; i < numbers.length; i++)
{
  numbers[i] *= 2;
}
```

-

```
function warning()
{
  alert("This is your final warning");
}
```

-
- calling the function
    - warning();

Arguments: Passing Data to a Function 50

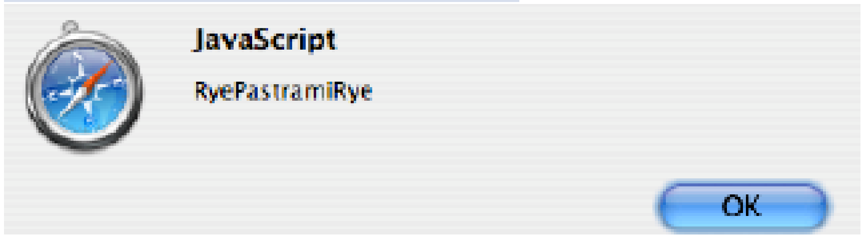
- alert("Insert and play");

```
function sandwich(bread, meat)
{
  alert(bread + meat + bread);
}
```

-

```
sandwich("Rye", "Pastrami");
```

-



JavaScript

RyePastramiRye

-

Return Statements: Outputting Data from a Function 52

- Return - data to the statement that called it

```
function sandwich(bread, meat)
{
  var assembled = bread + meat + bread;

  return assembled;
}
```

-

```
var lunch = sandwich("Rye", "Pastrami");
```

-
- Return can be expression

```
function sandwich(bread, meat)
{
  return bread + meat + bread;
}
```

  -

- Return Cuts off any statement below

```
function prematureReturner()
{
  return "Too quick";

  alert("Was it good for you?");
}
```

  -

Scope: Keeping your Variables Separate 54

```
function countWiis()
{
  stock = 5;
  sales = 3;

  return stock - sales;
}

stock = 0;
wiis = countWiis();
```

-
    - You' d probably expect it still to be **0** , which is what we set it to be before calling **countWiis** . However, **countWiis** also uses a variable called **stock** . But because the function doesn' t use **var** to declare this variable, JavaScript will go looking outside the function— in the global scope— to see whether or not that variable already exists. Indeed it does, so JavaScript will assign the value **5** to that global variable.
- Local Scope – inside the function
- Global Scope – outside the function

```
function countWiis()
{
  var stock = 5;
  var sales = 3;

  return stock - sales;
}

var stock = 0;
var wiis = countWiis();
```

-
    - The stock variable declared outside the countWiis function will remain untouched by the stock variable declared inside countWiis

Objects 55

- Contains properties and methods
    - Properties are variables that are only accessible via their object
    - methods are functions that are only accessible via their object
- var robot = new Object();
    - Variable names start with a lowercase letter, while object names start with an uppercase letter.

```
Robot.metal = "Titanium";
Robot.killAllHumans = function()
{
  alert("Exterminate!");
};

Robot.killAllHumans();
```

    -
        - The first line of this code adds to our empty Robot object a metal property, assigning it a value of "Titanium" .
        - Object scope – variables and functions within the object

    - second line adds a killAllHumans method to our Robot object
    - last line of our program calls the Robot object' s killAllHumans method
- Alternative Syntax for standalone functions

```
function sandwich(bread, meat)
{
  alert(bread + meat + bread);
}
```

JavaScript lets you write this in the form of a var

```
var sandwich = function(bread, meat)
{
  alert(bread + meat + bread);
};
```

- object literal syntax – creates objects and its contents in a single statement

```
var Robot =
{
  metal: "Titanium",
  killAllHumans: function()
  {
    alert("Exterminate!");
  }
};
```

    - we represent a new object with curly braces
    - inside those braces, we list the properties and methods of the object, separated by commas
    - For each property and method, we assign a value using a colon (:) instead of the assignment operator
    - Can be difficult to read

Unobtrusive Scripting in the Real World 58

- Running JavaScript after all html are downloaded

```
var Robot =
{
  init: function()
  {
    Your HTML modifying code;
  }
};

Core.start(Robot);
```

    - By registering Robot with Core.start on the final line, you can rest assured that Robot.init will be run only when it' s safe to do

Summary 59

Chapter 3 Document Access 61

The Document Object Model: Mapping your HTML 61

- Within this model, each element in the HTML document becomes an object, as do all the attributes and text. JavaScript can access each of these objects independently, using built-in functions that make it easy to find and change what we want on the fly.

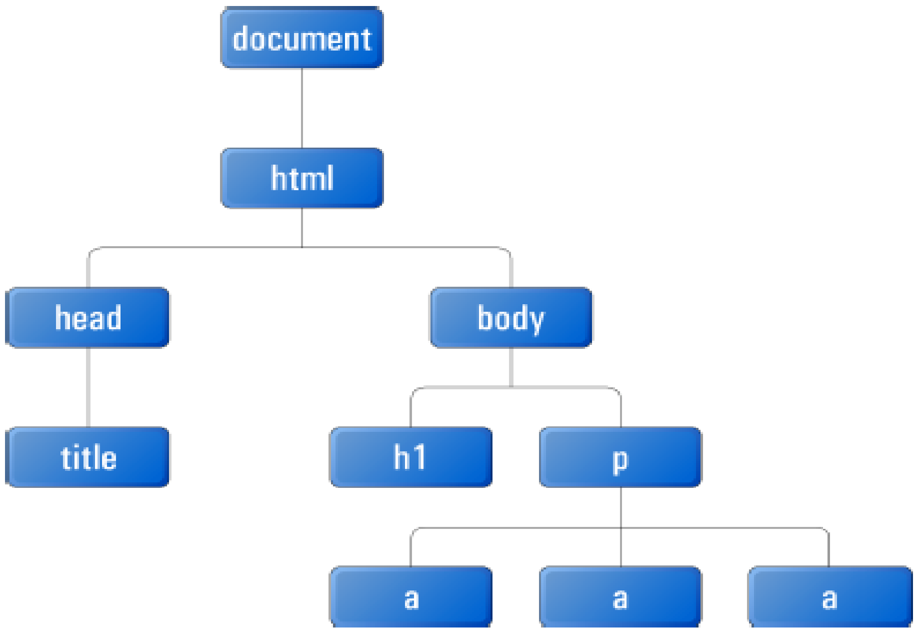


Figure 3.2. The DOM tree, including the document node

  - Type of Node
    - Element = node in javascript
    - Text nodes = anything that' s not contained between angled brackets in html; they are like lements but it doesn't have children
    - Attribute nodes
    - White space nodes
  - The highest node above html is document node

Text Nodes 64

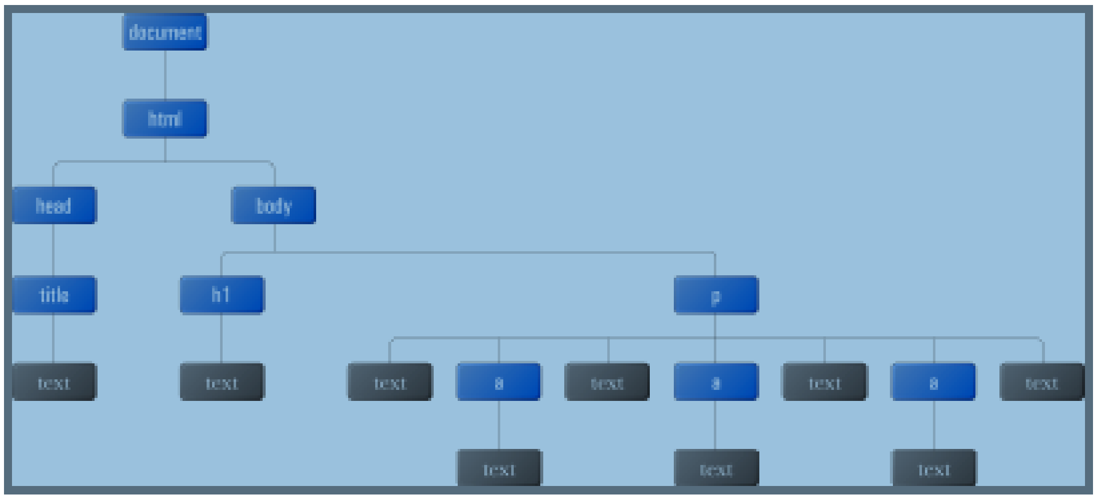


Figure 3.3. The complete DOM tree, including text nodes

Although those text nodes all look fairly similar, each node has its own value, which stores the actual text that the node represents. So the value of the text node inside the `title` element in this example would be "DOMinating JavaScript."

- Attribute Nodes 65

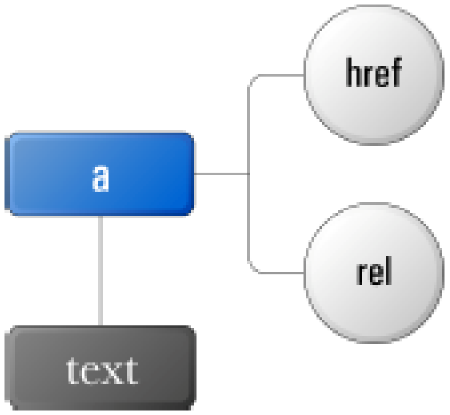


Figure 3.4. The `href` and `rel` attributes represented as attribute nodes in the DOM

- Accessing the Nodes you Want 66
- General Pattern
    - Specify the element or group of elements that you want to affect.
    - Specify the effect you want to have on them.

Finding an Element by ID 67

- In HTML
```
<p id="uniqueElement">
  ⋮
</p>
```
- In CSS
```
#uniqueElement ①
{
  color: blue; ②
}
```
- In JavaScript var target = document.getElementById("uniqueElement");

- To make sure to know what node you are targeting

```
var target = document.getElementById("berenger");
alert(target.nodeName);
```

An alert dialog will pop up displaying the tag name, as shown in Figure 3.5.

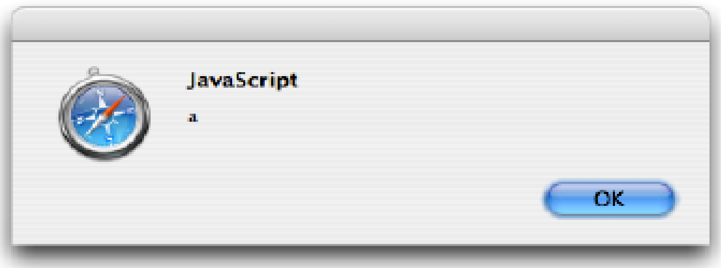


Figure 3.5. Displaying an element's tag name using the nodeName property

  -
  - If it returns null, it's an error
  - Safest way to check

```
var target = document.getElementById("berenger");

if (target != null)
{
  alert(target.nodeName);
}
```

Finding Elements by Tag Name 70

- Normally called on document node
- var listItems = document.getElementsByTagName("li");
  - to search through all of the descendants of the document node, get all the nodes with a tag name of "li" , and assign that group to the list Items variable
  - listItems ends up containing a collection of nodes called a node list.
  - You can access it like an Array

```
var listItems = document.getElementsByTagName("li");
var secondItem = listItems[1];
```

  - Node lists also have a length property

```
var listItems = document.getElementsByTagName("li");
var numItems = listItems.length;
```

  - You can use loops

```
var listItems = document.getElementsByTagName("li");

for (var i = 0; i < listItems.length; i++)
{
  alert(listItems[i].nodeName);
}
```

  - Unlike getElementById , getElementsByTagName will return a node list even if no elements matched the supplied tag name.
- Restricting Tag Name Selection

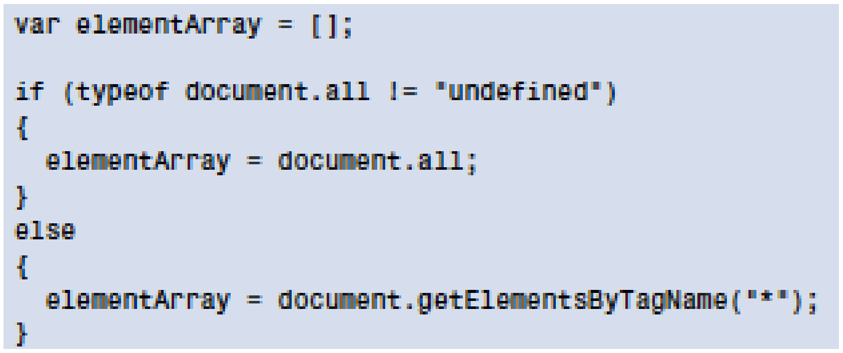
```
      There are 3 different types of element in this body:
    </p>
    <ul>
      <li>
        paragraph
      </li>
      <li>
        unordered list
      </li>
      <li>
        list item
      </li>
    </ul>
    <p>
      There are 2 children of html:
    </p>
    <ul>
      <li>
        head
      </li>
      <li>
        body
      </li>
    </ul>
  </body>
</html>
```

```
var lists = document.getElementsByTagName("ul");
var secondList = lists[1];
var secondListItems = secondList.getElementsByTagName("li");
```

Finding Elements by Class Name 74

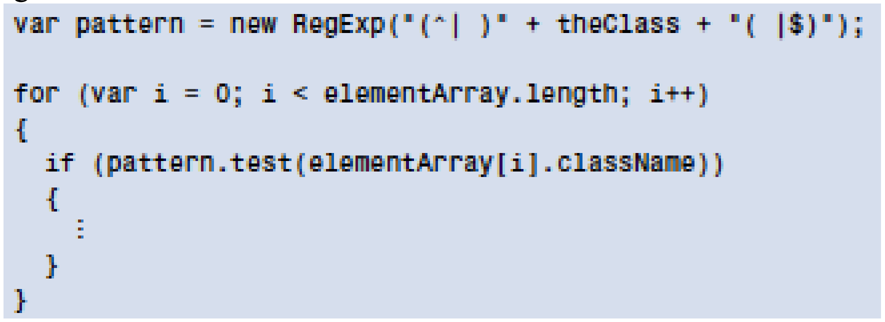
- Unfortunately, no built-in DOM function lets you get elements by class, so I think it' s time we created our first real function! Once that' s done, we can add the function to our custom JavaScript library and call it whenever we want to get all elements with a particular class.
- Starting your first function
  - Look at each element in the document
  - For each element, perform a check that compares its class against the one we're looking for
  - If the classes match, add the element to our group of elements
  - Side notes:
    - Firstly, whenever you see the phrase " for each," chances are that you' re going to need a loop.
    - Secondly, whenever there' s a condition such as " if it matches," you' re going to need a conditional statement.
    - Lastly, when we talk about a " group," that usually means an array or node list.
- Looking at All the Elements

- getElementsByTagName - get all the elements in the document
- typeof - operator to check for the existence of document.all. typeof checks the data type of the value that follows it, and produces a string that describes the value’ s type (for instance, **"number"** , **"string"** , **"object"** , etc.).
- object detection, is the safest way of testing whether an object—such as document.all—exists



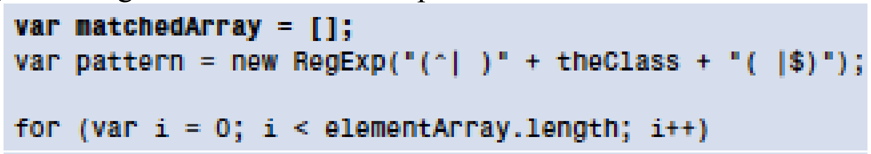
```
var elementArray = [];

if (typeof document.all != "undefined")
{
  elementArray = document.all;
}
else
{
  elementArray = document.getElementsByTagName("*");
}
```

  - we should have a collection of elements to look at
- Checking the class of each element

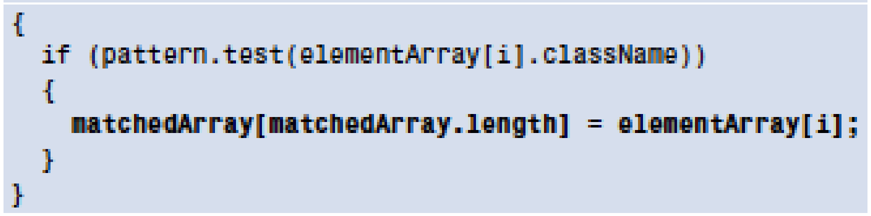


```
var pattern = new RegExp("(^| )" + theClass + "( |$)");

for (var i = 0; i < elementArray.length; i++)
{
  if (pattern.test(elementArray[i].className))
  {
    ⋮
  }
}
```

    - regular expressions help us search strings for a particular pattern
    - theClass as the class we want to match against; theClass will be passed into our function as an argument
    - When pattern.test is run, it checks the string argument that’ s passed to it against the regular expression
- Adding Matching Elements to our Group of Elements

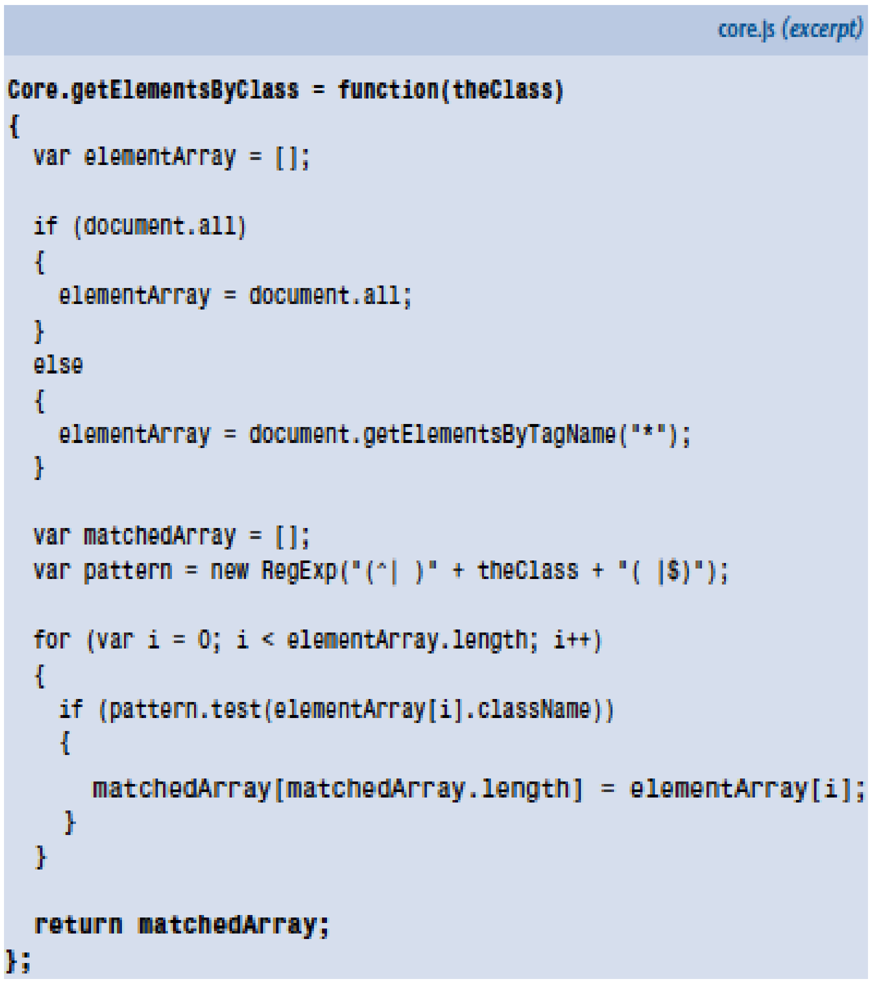


```
var matchedArray = [];
var pattern = new RegExp("(^| )" + theClass + "( |$)");

for (var i = 0; i < elementArray.length; i++)
```



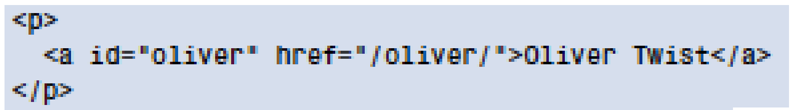
```
{
  if (pattern.test(elementArray[i].className))
  {
    matchedArray[matchedArray.length] = elementArray[i];
  }
}
```

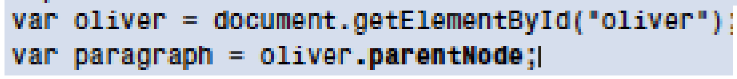
- Putting it all together

- 

  core.js (excerpt)

  ```
  Core.getElementsByClass = function(theClass)
  {
    var elementArray = [];

    if (document.all)
    {
      elementArray = document.all;
    }
    else
    {
      elementArray = document.getElementsByTagName("*");
    }

    var matchedArray = [];
    var pattern = new RegExp("(^| )" + theClass + "( |$)");

    for (var i = 0; i < elementArray.length; i++)
    {
      if (pattern.test(elementArray[i].className))
      {
        matchedArray[matchedArray.length] = elementArray[i];
      }
    }

    return matchedArray;
  };
  ```
- Now that it' s part of our Core library, we can use this function to find a group of elements by class from anywhere in our JavaScript code:
- var elementArray = Core.getElementsByClass("dataTable");

- walking the DOM – giving directions
- finding a Parent
  - 

    ```
    <p>
      <a id="oliver" href="/oliver/">Oliver Twist</a>
    </p>
    ```
  - 

    ```
    var oliver = document.getElementById("oliver");
    var paragraph = oliver.parentNode;
    ```
- finding the children

```
<ul id="baldwins">
  <li>
    Alec
  </li>
  <li>
    Daniel
  </li>
  <li>
    William
  </li>
  <li>
    Stephen
  </li>
</ul>
```

```
var baldwins = document.getElementById("baldwins");
var william = baldwins.childNodes[2];
```

```
var alec = baldwins.firstChild;
```

```
var stephen = baldwins.lastChild;
```

- finding siblings
  - var stephen = william.**nextSibling**;
  - var daniel = william.**previousSibling**;

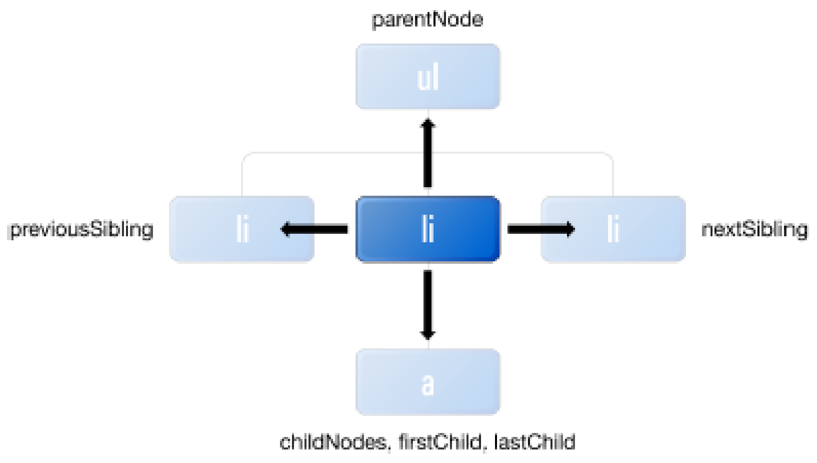


Figure 3.6. Moving around the DOM tree using the element node's DOM properties

Interacting with Attributes 82

- we don' t have DOM functions that will let you find a particular attribute node, or all attributes with a certain value.
- Attributes are more focused on reading and modifying the data related to an element.
- Getting an Attribute

```
<a id="koko" href="http://www.koko.org/">Let's all hug Koko</a>
```

```
var koko = document.getElementById("koko");
var kokoHref = koko.getAttribute("href");
```

  - value of kokoHref will now be http://www.koko.org/

  -
    ```
    var koko = document.getElementById("koko");
    var kokoId = koko.getAttribute("id");
    ```
    - value of kokoId will now be "koko" .

  -
    ```
    var koko = document.getElementById("koko");
    var kokoHref = koko.href;
    ```
    - to get the href on our anchor
- Setting an Attribute
  - To write an attribute value, we use the setAttribute method

  -
    ```
    var koko = document.getElementById("koko");
    koko.setAttribute("href", "/koko/");
    ```
    - the href for Koko' s link will change from http://www.koko.org/ to /koko/ .
  - setAttribute can be used not only to change preexisting attributes, but also to add new attributes

    -
      ```
      var koko = document.getElementById("koko");
      koko.setAttribute("title", "Web site of the Gorilla Foundation");
      ```
    - Output

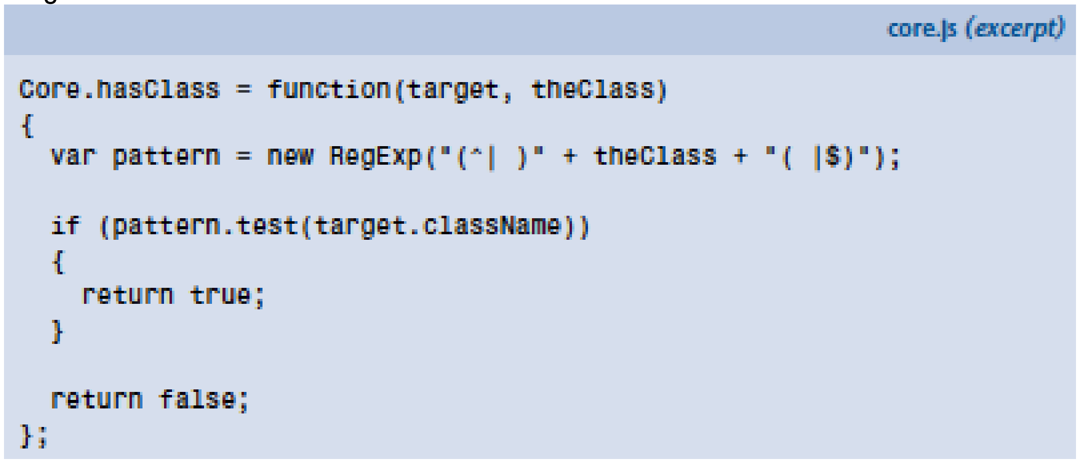
    -
      ```
      <a id="koko" href="http://www.koko.org/"
          title="Web site of the Gorilla Foundation">Let's all hug
        Koko</a>
      ```

-
  ```
  var scarlet = document.getElementById("scarlet");
  scarlet.style.color = "#FF0000";
  ```
  - To change the text color of an element, we' d use style.color :

-
  ```
  var indigo = document.getElementById("indigo");
  indigo.style.backgroundColor = "#000066";
  ```
  - To change its background color, we' d use style.backgroundColor :
- a good rule of thumb: if you wish to access a particular CSS property, simply append it as a property of the style object. Any properties that include hyphens (like text-indent) should be converted to camel case (textIndent).
- *style_object.js (excerpt)*
  ```
  var body = document.getElementsByTagName("body")[0];
  body.style.backgroundColor = "#000000";
  body.style.color = "#FFFFFF";
  ```

- The best way to change an element' s appearance with JavaScript is to change its class.
  - Advantages
    - We don' t mix behavior with style.
    - We don' t have to hunt through a JavaScript file to change styles.
    - Style changes can be made by those who make the styles, not the JavaScript programmers.

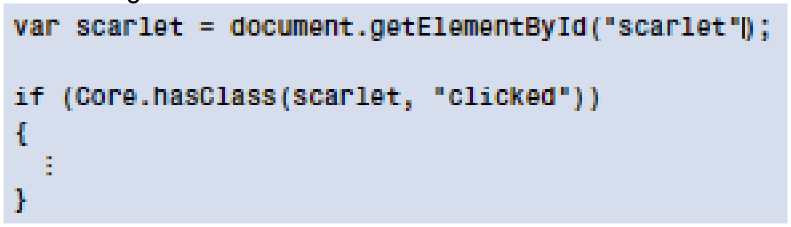
        - It’ s more succinct to write styles in CSS.
- Comparing Classes



core.js (excerpt)

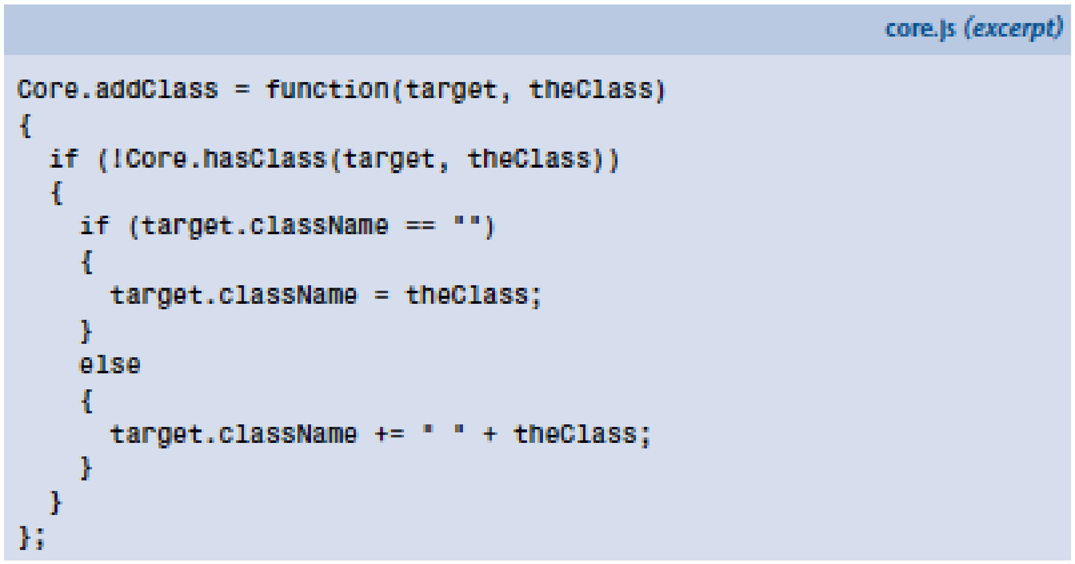
```
Core.hasClass = function(target, theClass)
{
  var pattern = new RegExp("(^| )" + theClass + "( |$)");

  if (pattern.test(target.className))
  {
    return true;
  }

  return false;
};
```

    - Core.hasClass takes two arguments: an element and a class.
        - The class is used inside the regular expression and compared with the className of the element. If the pattern.test method returns true, it means that the element does have the specified class, and we can return true from the function. If pattern.test returns false, Core.hasClass returns false by default.
    - Now, we can very easily use this function inside a conditional statement to execute some code when an element has (or doesn’ t have) a matching class:



```
var scarlet = document.getElementById("scarlet");

if (Core.hasClass(scarlet, "clicked"))
{
  ⋮
}
```

- Adding a class
    - The main thing we have to be careful about here is to not overwrite an element’ s existing classes. Also, to make it easy to remove a class, we shouldn’ t add a class to an element that already has that class.



core.js (excerpt)

```
Core.addClass = function(target, theClass)
{
  if (!Core.hasClass(target, theClass))
  {
    if (target.className == "")
    {
      target.className = theClass;
    }
    else
    {
      target.className += " " + theClass;
    }
  }
};
```

-
    - first conditional statement inside Core.addClass uses Core.hasClass to check whether or not the target element already has the class we' re trying to add. If it does, there' s no need to add the class again.
    - If the target doesn't have the class, we have to check whether that element has any classes at all. If it has none (that is, the className is an empty string), it' s safe to assign theClass directly to target.className . But if the element has some preexisting classes, we have to follow the syntax for multiple classes, whereby each class is separated by a space. Thus, we add a space to the end of className , followed by theClass . Then we' re done.

class.js *(excerpt)*

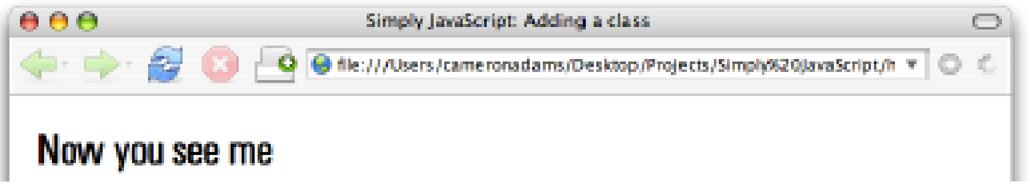
```
var body = document.getElementsByTagName("body")[0];
Core.addClass(body, "unreadable");
```

Then, we specify some CSS rules for that class in our CSS file:

class.css

```
.unreadable
{
  background-image: url(polka_dots.gif);
  background-repeat: 15px 15px;
  color: #FFFFFF;
}
```

The visuals for our page will swap from those shown in Figure 3.9 to those depicted in Figure 3.10.


Simply JavaScript: Adding a class
file:///Users/cameronadams/Desktop/Projects/Simply%20JavaScript/h

Now you see me

- ○
- Removing a Class

core.js *(excerpt)*

```
Core.removeClass = function(target, theClass)
{
  var pattern = new RegExp("(^| )" + theClass + "( |$)");

  target.className = target.className.replace(pattern, "$1");
  target.className = target.className.replace(/ $/, "");
};
```

- ○
    - In Core.removeClass , instead of using the regular expression to check whether or not the target element has the class, we assume that it does have the class, and instead use the regular expression to replace the class with an empty string, effectively removing it from className .

  - The first version of the W3C DOM specification was called Document Object Model Level 1. Since event handlers (along with a number of other nonstandard JavaScript features) predate this specification, developers like to call them Document Object Model Level 0.
  - Stepping into the 21st century, the World Wide Web Consortium (W3C) has developed the DOM Level 2 Events standard,2 which provides a more powerful means of dealing with events, called event listeners.
- Event Handlers 107
  - event handler is a JavaScript function that' s “ plugged into” a node in the DOM so that it' s called automatically when a particular event occurs in relation to that element.

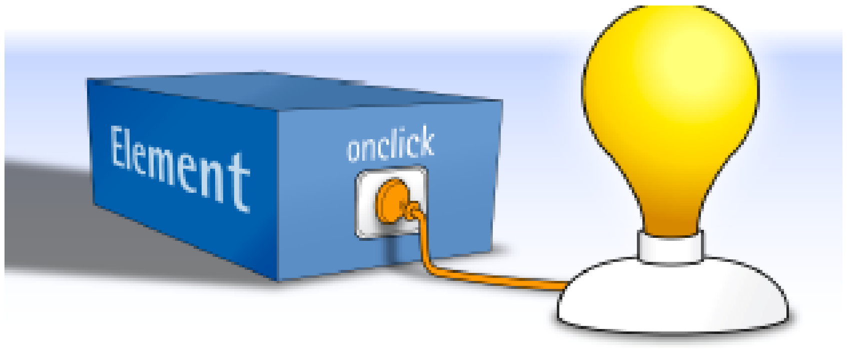
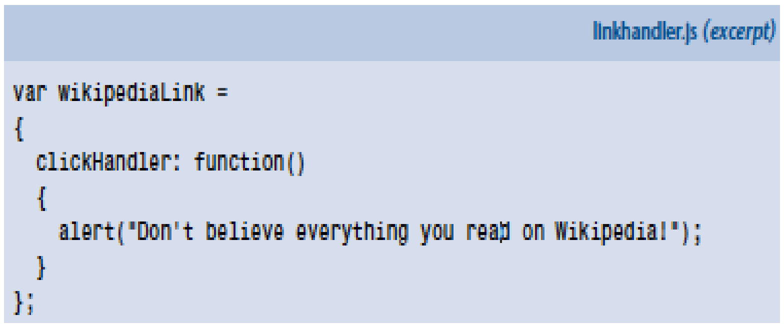


Figure 4.1. Plugging in a single event handler function to respond to a particular event

  -
    - Create an Event Function



linkhandler.js *(excerpt)*

```
var wikipediaLink =
{
  clickHandler: function()
  {
    alert("Don't believe everything you read on Wikipedia!");
  }
};
```

      -
    - Get the element
      - var link = document.getElementById("wikipedia");
    - Add the Event Function to the Element
      - link.onclick = wikipediaLink.clickHandler;
      - Convention: element**.on**event = eventHandler**;**
    - Core.start(wikipediaLink); - to request that the object' s init method be called when the whole document has loaded
  - Default Actions 111
    - Browsers take all sorts of actions like this:
      - They follow links that users click.
      - They submit forms when users click a Submit button, or hit Enter .
      - They move keyboard focus around the page when the user

hits Tab .

- default actions—things the browser normally does in response to events
- The easiest way to stop the browser from performing a default action in response to an event is to create for that event an event handler that returns false

clickprompt.js *(excerpt)*

```
clickHandler: function()
{
  if (!confirm("Are you sure you want to leave this site?"))
  {
    return false;
  }
}
```

- The confirm function used in this code is built into the browser, just like alert . And it displays a message to the user just like alert does, except that it offers the user two buttons to click: OK and Cancel . If the user clicks OK , the function returns true . If the user clicks Cancel , the function returns false . We then use the ! operator introduced in Table 2.1 to reverse that value so that the body of the if statement is executed when the user clicks Cancel .
- Cutting Down on Code

```
clickHandler: function()
{
  return confirm(
      "Are you sure you want to leave this site?");
}
```

- The this Keyword 112

```
clickHandler: function()
{
  open("http://en.wikipedia.org/wiki/Christopher_Pike");
  return false;
}
```

  - a functions that opens the URL to another window

- The Problem with Event Handlers 115
  - Limitation - you can only assign one event handler to a given event on a given HTML element
    - Example

```
element.onclick = script1.clickHandler;
element.onclick = script2.clickHandler;
```

      - the second event handler replaces the first
  - Workaround
    - Assign an event handler a function that calls multiple event

handling functions:

```
element.onclick = function()
{
  script1.clickHandler();
  script2.clickHandler();
}
```

- But what's wrong with this approach
    - This will no longer point to the element within the clickHandler methods.
    - If either clickHandler method returns false, it will not cancel the default action for the event.
    - Instead of assigning event handlers neatly inside a script's init method, you have to perform these assignments in a separate script, since you have to reference both script1 and script2.

- Event Listeners 116
    - event listeners are just like event handlers, except that you can assign as many event listeners as you like to a particular event on a particular element, and there is a W3C specification that explains how they should work.
        - Internet explorer, and safari might be slightly buggy
        - Like an event handler, an event listener is just a JavaScript function that is " plugged into" a DOM node. Where you could only plug in one event handler at a time, however, you can plug multiple listeners in, as Figure 4.5 illustrates

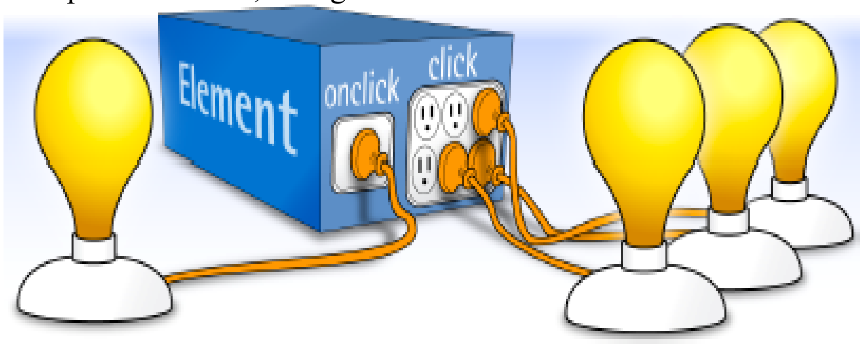


Figure 4.5. Plugging in only one handler, but many listeners

- Code Sample;

```
element.addEventListener("event", eventListener, false);
```

- addEventListener is a method that takes 3 arguments
    - the name of the event to which you want to assign the listener (e.g. **"click"**)
    - The listener function itself
    - Boolean value that you' ll usually want to set to **false** (more on this last argument in the section called " Event Propagation").

- To set up an event listener in Internet Explorer, however, you need to use a method called attachEvent and it takes slightly arguments

```
element.attachEvent("onevent", eventListener);
```

- Spot the differences? The first argument— the name of the event you' re interested in— must be prefixed with on (for example, "onclick"), and there is no mysterious third argument
- To Check if browser supports
  - if-else statement that checks if the addEventListener or attachEvent methods exist in the current browser

```
if (typeof element.addEventListener != "undefined")
{
  element.addEventListener("event", eventListener, false);
}
else if (typeof element.attachEvent != "undefined")
{
  element.attachEvent("onevent", eventListener);
}
```

  - example of the object detection technique

linklistener.js *(excerpt)*

```
var wikipediaLink =
{
  init: function()
  {
    var link = document.getElementById("wikipedia");

    if (typeof link.addEventListener != "undefined")
    {
      link.addEventListener(
          "click", wikipediaLink.clickListener, false);
    }
    else if (typeof link.attachEvent != "undefined")
    {
      link.attachEvent("onclick", wikipediaLink.clickListener);
    }
  },

  clickListener: function()
  {
     alert("Don't believe everything you read on Wikipedia!");
   }
};

Core.start(wikipediaLink);
```

- Unplugging event listeners from a DOM node

```
if (typeof element.removeEventListener != "undefined")
{
  element.removeEventListener("event", eventListener, false);
}
else if (typeof element.detachEvent != "undefined")
{
  element.detachEvent("onevent", eventListener);
}
```

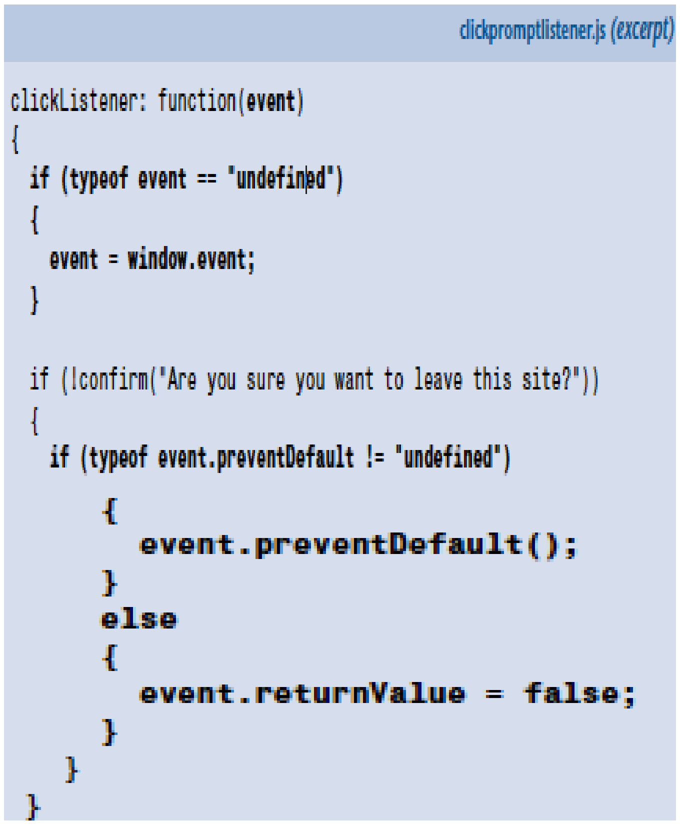
- Default Actions 119
  - event listener model, the browser will always pass an event object to the event listener function
    - The event object' s properties contain information about the event (for instance, the position of the cursor when the event occurred), while its methods let us control how the event is processed by the browser
    - To Prevent default actions

```
clickListener: function(event)
{
  if (!confirm("Are you sure you want to leave this site?"))
  {
    event.preventDefault();
  }
}
```

      - If multiple listeners are associated with an event, any one of those listeners calling preventDefault is enough to stop the default action from occurring
    - In Internet Explorer, the event object isn' t passed to the event listener as an argument; it' s available as a global variable named event . Also, the event object doesn' t have a preventDefault method; instead, it has a property named returnValue that we can set to false in order to prevent the default action from taking place:

```
clickListener: function()
{
  if (!confirm("Are you sure you want to leave this site?"))
  {
    event.returnValue = false;
  }
}
```

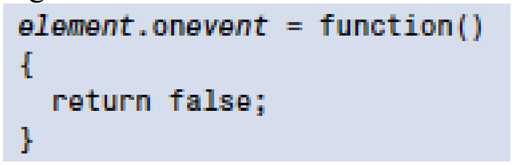
    - object detection technique



clickpromptlistener.js (excerpt)

```
clickListener: function(event)
{
  if (typeof event == "undefined")
  {
    event = window.event;
  }

  if (!confirm("Are you sure you want to leave this site?"))
  {
    if (typeof event.preventDefault != "undefined")
        {
          event.preventDefault();
        }
        else
        {
          event.returnValue = false;
        }
     }
  }
```

- Preventing Default Actions in Safari 2.0.3 and Earlier



```
element.onevent = function()
{
  return false;
}
```

- Event Propagation 122
  - Event propagation – events don't just target element that generated the event – they travel through the tree structure of the DOM
    - 3 phases of event propagation
      - Capture phase, the event travels down through the DOM tree, visiting each of the target element's ancestors on its way to the target element.
        - Example: if the user clicked a hyperlink, that click event would pass through the document node, the html element, the body element, and the paragraph containing the link.
        - At each stop along the way, the browser checks for capturing event listeners (third argument of the addEventListener method that is set to true) for that type of event, and runs them
        - Internet explorer doesn't have a capture phase