Web Engineering LAB

Lab # 07
JavaScript: Object, DOM, Events

Instructor: Hurmat Hidayat

Email: hurmathidayat@nu.edu.pk

Course Code: SL3003

Semester Spring 2022

Department of Computer Science,
National University of Computer and Emerging Sciences FAST
Peshawar Campus

Content

What is Object?

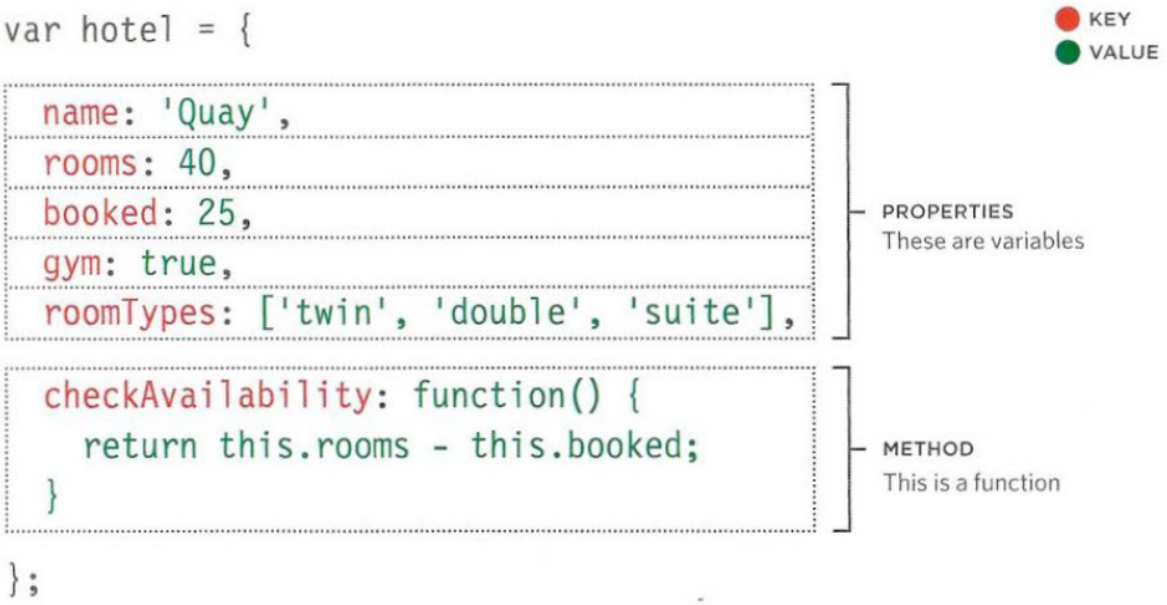
Java Script is not full fledge Object Oriented Language but it supports objects. Objects group together a set of variables and functions to create a model of a something you would recognize from the real world. In an object, variables and functions take on new names.

Properties:
If a variable is part of an object, it is called a property. Properties tell us about the object, such as the name of a hotel or the number of rooms it has.

Methods:
If a function is part of an object, it is called a method. Methods represent tasks that are associated with the object. For example, you can check how many rooms are available in the hotel by subtracting the number of booked rooms from the total number of rooms.

Hotel Object

This object represents a hotel. It has five properties and one method. The object is in curly braces. It is stored in a variable called hotel.



- Like variables and named functions, properties and methods have a name and a value. In an object, that name is called a key.
- An object cannot have two keys with the same name
- The value of a property can be a string, number, Boolean, array, or even another object. The value of a method is always a function

Creating an Object: Literal Notation

Literal Notation is the easiest and most popular way to create objects. The object is the curly braces and their contents. As in above example object is stored in a variable name hotel. Separate each value from its value using a colon. Separate each property and method with a comma.

```
var hotel = {
    name:  'Quay',
    rooms: 40,
    booked: 25,
    gym: true,
    roomType: ['twin', 'double','suite'],

    checkAvailability: function(){
    return this.rooms - this.booked;
     }
};

var elName = document.getElementByld('hotelName');
elName.textContent = hotel.name;

var elRooms = document.getElementByid{'rooms');
elRooms .textContent = hotel.checkAvailability();
```

Creating an Object: Constructor Notation

The `new` keyword and the object constructor create a blank object. You can then add properties and methods to the object. You create a new object using a combination of the `new` keyword and the `Object()` constructor function.

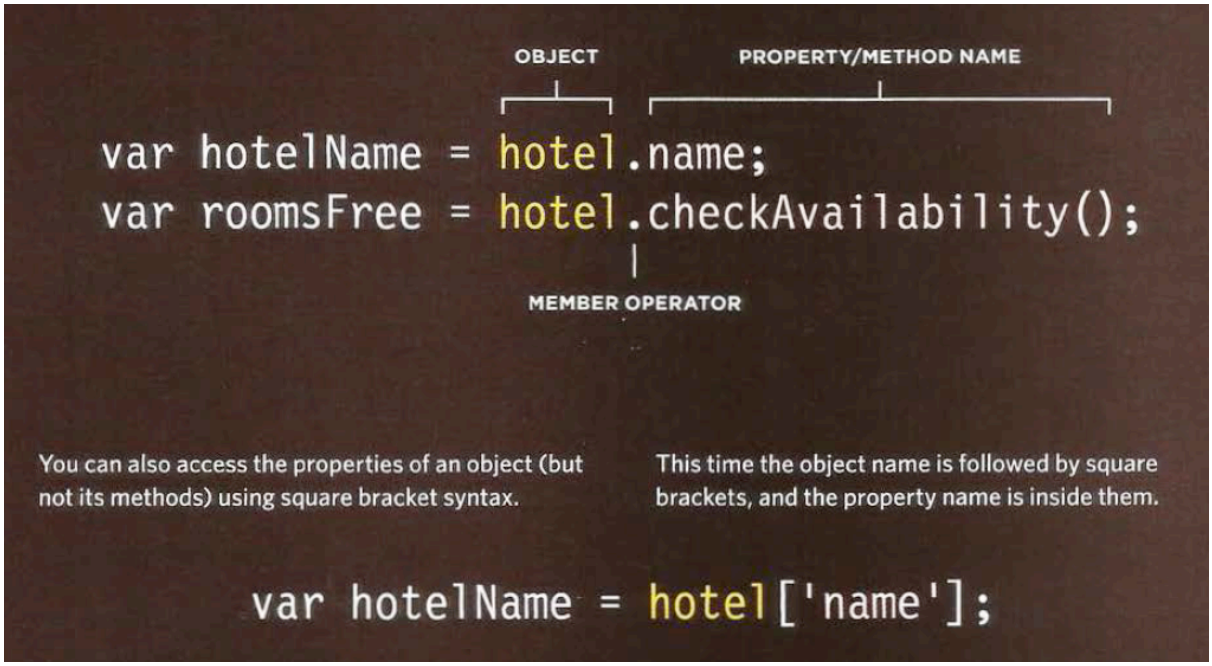
Note: this function is part of the JavaScript language and is used to create objects.

```
1   /* Create object using constructor notation */
2   var hotel = new Object();
3
4   hotel.name = 'Quay';
5   hotel.rooms = 40;
6   hotel.booked = 25;
7
8   hotel.checkAvailability = function(){
9       return this.rooms - this.booked;
10  };
11
```

To create empty object using literal notation use:
Var hotel = {}
And then add properties and method to the object following the same syntax.

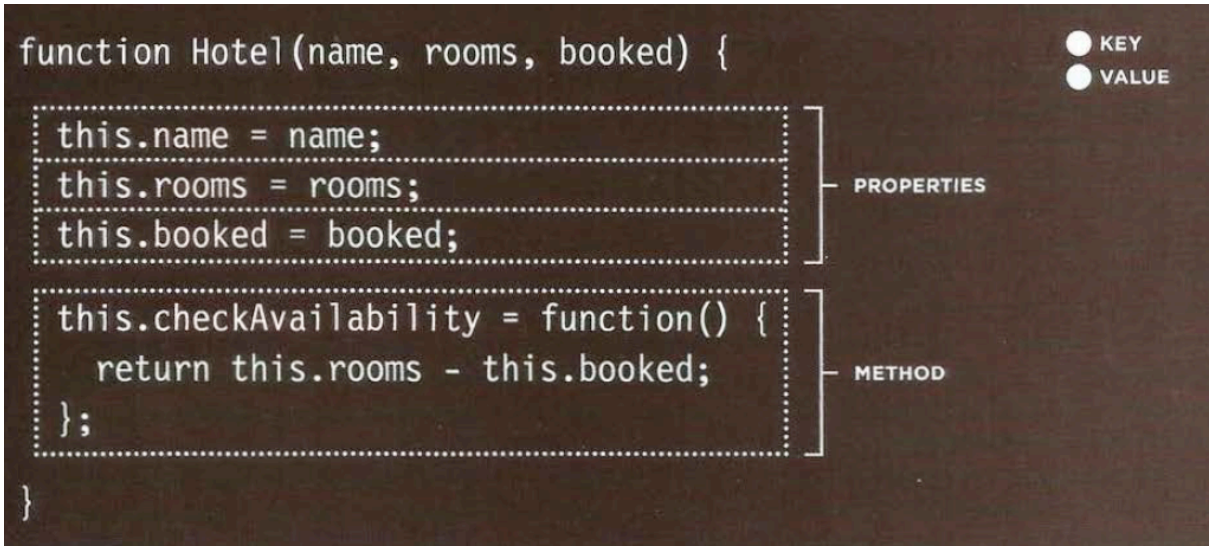
Accessing an Object and Dot Notation

You can access properties or method of object using dot notation or square brackets.

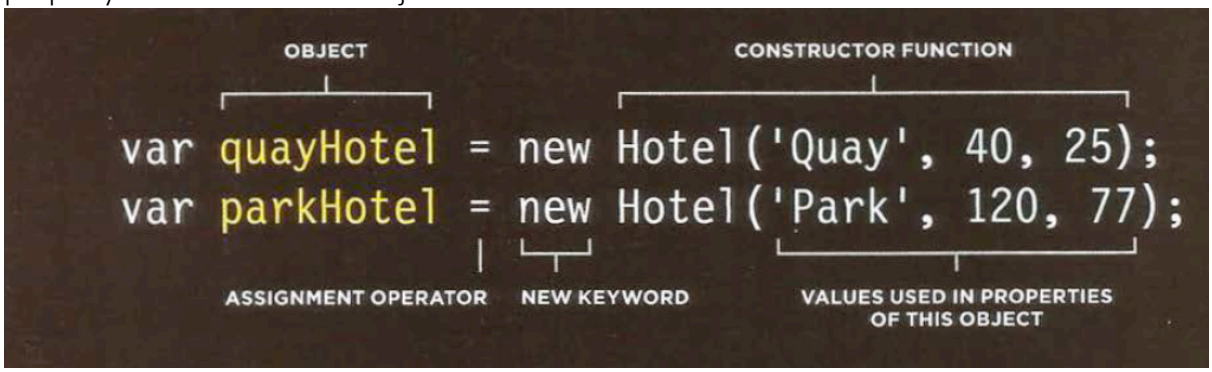


Creating Many Object: Constructor Notation

Sometimes we need to create multiple objects to represent similar things. Object constructors can use a function as a template for creating objects. First, create the template with the object's properties and methods.



A function called Hotel will be used as a template for creating new objects that represent hotels. The `this` keyword is used instead of the object name to indicate that the property or method belongs to the object that this function creates. Each statement that create new property or method for this object ends in a semicolon.



Example: Create and access objects constructor notation

```
<!DOCTYPE html>
<html>
  <head>
    <title>Objects</title>
    <link rel="stylesheet" href="css/c03.css" />
  </head>
  <body>
    <h1>TravelWorthy</h1>
    <div id="info">
      <h2>hotel availability</h2>
      <div id="hotel1"></div>
      <div id="hotel2"></div>
    </div>
    <script src="js/multiple-objects.js"></script>
  </body>
</html>
```

```
// Create the template for objects that are hotels
function Hotel(name, rooms, booked) {
  this.name = name;
  this.rooms = rooms;
  this.booked = booked;
  this.checkAvailability = function() {
    return this.rooms - this.booked;
  };
}


// Create two hotel objects
var quayHotel = new Hotel('Quay', 40, 25);
var parkHotel = new Hotel('Park', 120, 77);


// Update the HTML for the page
var details1 = quayHotel.name + ' rooms: ';
    details1 += quayHotel.checkAvailability();
var elHotel1 = document.getElementById('hotel1');
elHotel1.textContent = details1;

var details2 = parkHotel.name + ' rooms: ';
    details2 += parkHotel.checkAvailability();
var elHotel2 = document.getElementById('hotel2');
elHotel2.textContent = details2;

```

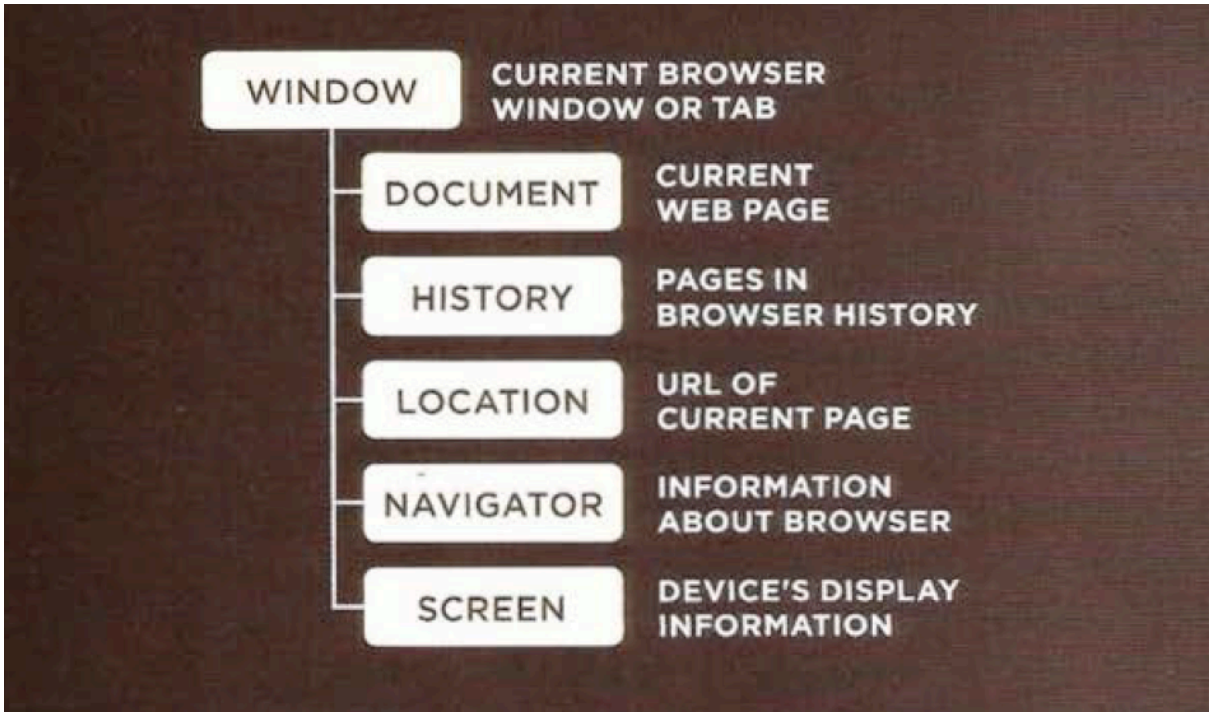
Built-In Objects In JavaScript

Browsers come with a set of built-in objects that represent things like the browser window and the current web page shown in that window. These built-in objects act like a toolkit for creating interactive web pages. These built-in objects help you get a wide range of information such as the width of the browser window, the content of the main heading in the page, or the length of text a user entered into a form field. The three set of built in objects are:

1. Browser Object Model
2. Document Object Model
3. Global JavaScript Objects

Browser Object Model

The Browser Object Model creates a model of the browser tab or window. The topmost object is the window object, which represents current browser window or tab. Its child objects represent other browser features.

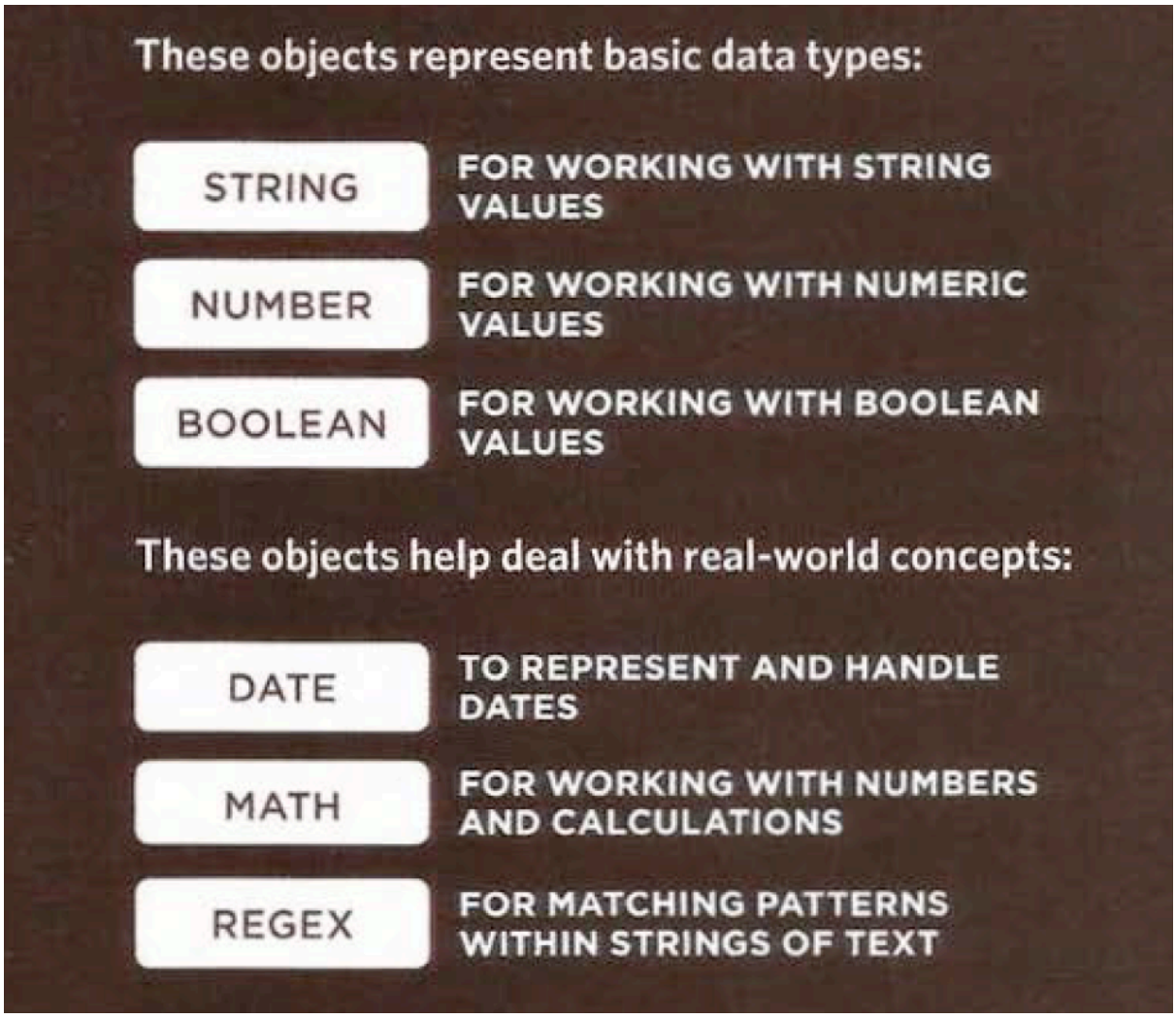


PROPERTY	DESCRIPTION
`window.innerHeight`	Height of window (excluding browser chrome/user interface) (in pixels)
`window.innerWidth`	Width of window (excluding browser chrome/user interface) (in pixels)
`window.pageXOffset`	Distance document has been scrolled horizontally (in pixels)
`window.pageYOffset`	Distance document has been scrolled vertically (in pixels)
`window.screenX`	X-coordinate of pointer, relative to top left corner of screen (in pixels)
`window.screenY`	Y-coordinate of pointer, relative to top left corner of screen (in pixels)
`window.location`	Current URL of `window` object (or local file path)
`window.document`	Reference to `document` object, which is used to represent the current page contained in window
`window.history`	Reference to `history` object for browser window or tab, which contains details of the pages that have been viewed in that window or tab
`window.history.length`	Number of items in `history` object for browser window or tab
`window.screen`	Reference to `screen` object
`window.screen.width`	Accesses `screen` object and finds value of its `width` property (in pixels)
`window.screen.height`	Accesses `screen` object and finds value of its `height` property (in pixels)

METHOD	DESCRIPTION
`window.alert()`	Creates dialog box with message (user must click OK button to close it)
`window.open()`	Opens new browser window with URL specified as parameter (if browser has pop-up blocking software installed, this method may not work)
`window.print()`	Tells browser that user wants to print contents of current page (acts like user has clicked a print option in the browser's user interface)

Global JavaScript Objects

The global objects do not form a single model. They are a group of individual objects that relate to different parts of the JavaScript language. The names of the global objects usually start with a capital letter, e.g. the String and Date objects.



Note: For detailed overview on Global JavaScript Objects read the book.

Global Object: Math Object

The Math object has properties and methods for mathematical constants and functions.

PROPERTY	DESCRIPTION
`Math.PI`	Returns pi (approximately 3.14159265359)

METHOD	DESCRIPTION
`Math.round()`	Rounds number to the nearest integer
`Math.sqrt(n)`	Returns square root of positive number, e.g., `Math.sqrt(9)` returns 3
`Math.ceil()`	Rounds number up to the nearest integer
`Math.floor()`	Rounds number down to the nearest integer
`Math.random()`	Generates a random number between 0 (inclusive) and 1 (not inclusive)

Because it is known as a **global object**, you can just use the name of the `Math` object followed by the property or method you want to access.

Typically you will then store the resulting number in a variable. This object also has many trigonometric functions such as `sin()`, `cos()`, and `tan()`.

The trigonometric functions return angles in radians which can then be converted into degrees if you divide the number by (pi/ 180).

Example: Math Object to create random numbers

```
<!DOCTYPE html>
<html>
  <head>
    <title>Math Object</title>
    <link rel="stylesheet" href="css/c03.css" />
  </head>
  <body>
    <h1>TravelWorthy</h1>
    <div id="info"></div>
    <script src="js/math-object.js"></script>
  </body>
</html>
```

```
// Create a variable to hold a random number between 1 and 10
var randomNum = Math.floor((Math.random() * 10) + 1);

// Create a variable called el to hold the element
// whose id attribute has a value of info
var el = document.getElementById('info');
// Write the number into that element
el.innerHTML = '<h2>random number</h2><p>' + randomNum + '</p>';
```

Document Object Model

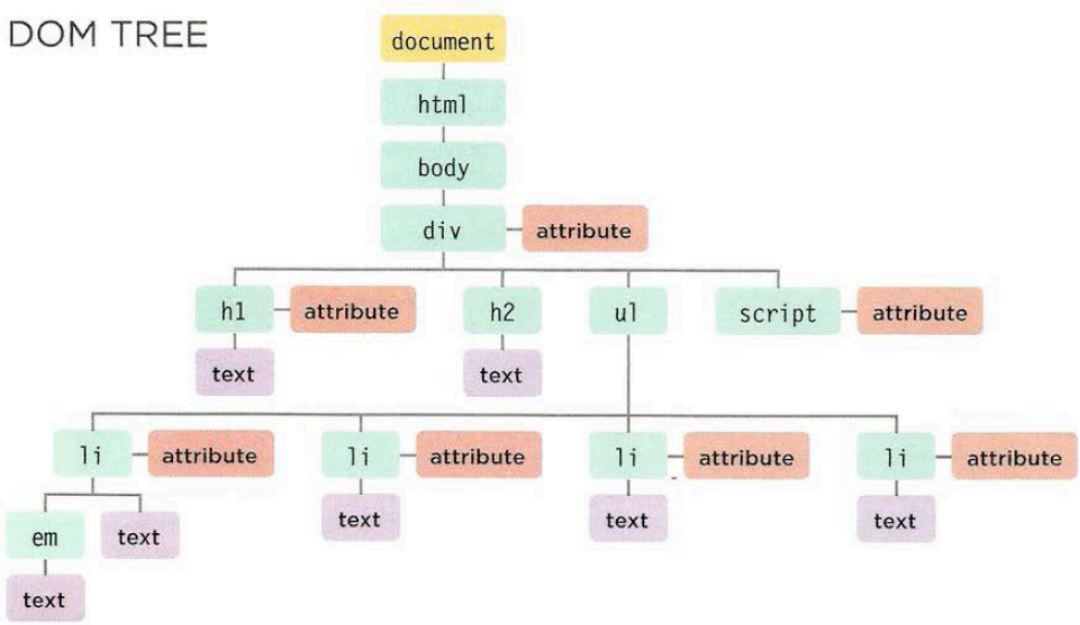
The Document Object Model (DOM) specifies how browser should create a model of an HTML page and how JavaScript can access and update the contents of a web page while it is in browser window. When the browser loads a webpage, it creates a model of the page in memory.

BODY OF HTML PAGE

```
<html>
  <body>
    <div id="page">
      <h1 id="header">List</h1>
      <h2>Buy groceries</h2>
      <ul>
        <li id="one" class="hot"><em>fresh</em> figs</li>
        <li id="two" class="hot">pine nuts</li>
        <li id="three" class="hot">honey</li>
        <li id="four">balsamic vinegar</li>
      </ul>
      <script src="js/list.js"></script>
    </div>
  </body>
</html>
```

The DOM Tree

The DOM tree is the model of the Web page. As the browser loads the web page, it create model of that page. The model is called a DOM tree.



The DOM Tree consist of four main type of nodes:

1. The Document Node
2. Element Nodes

3. Attribute Nodes
4. Text Nodes

The Document Node:
At the top of the tree a document node is added; it represents the entire page. When you access nay element, attribute, or text node, you navigate to it via document node. The document node is the starting point.

The Element Node:
HTML element describe the structure of an HTML page. (The <h1 > - <h6> elements describe what parts are headings; the <p> tags indicate where paragraphs of text start and finish; and so on.)

The Attribute Nodes:
The opening tags of HTML elements can carry attributes and these are represented by attribute nodes in the DOM tree. Attribute nodes are not children of the element that carries them; they are part of that element.

The Text Nodes:
Once you have accessed an element node, you can then reach the text within that element. This is stored in its own text node. Text nodes cannot have children. If an element contains text and another child element, the child element is not a child of the text node but rather a child of the containing element.

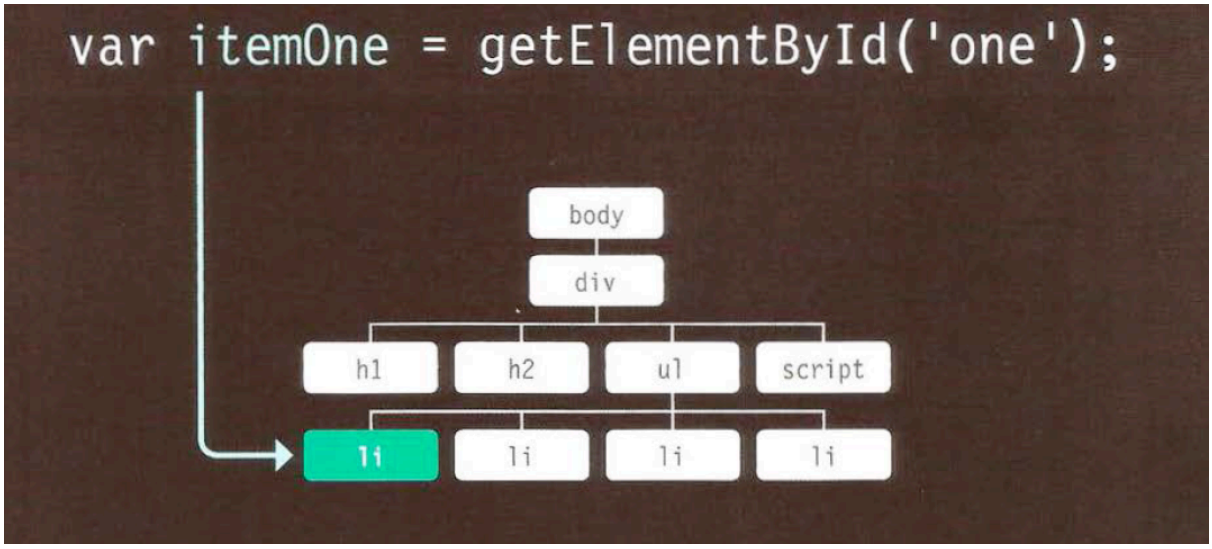
Access and Update DOM Tree
Accessing and updating DOM tree involves two steps:

1. Locate the node that represents the element you want to work with.
2. Use its text content, child element, and attributes.

DOM Queries
Methods that find elements in the DOM tree are called DOM queries. DOM queries may return one element, or they may return a Nodelist, which is a collection of nodes. When you need to work with an element more than once, you should use a variable to store the result of this query.

When a script selects an element to access or update, the interpreter must find the element(s) in the DOM tree.

Here, itemOne does not store the <li> element, it stores a reference to where that node is in the DOM tree. To access the text content of this element, you might use the variable name: itemOne.textContent



Select an Individual Element Node:

The common ways to select an individual element are:

- `getElementById('id')`
- `querySelector('css selector')`

Select Multiple Elements (NodeLists):

The common ways to select an individual element are:

- `getElementsByClassName('class')`
- `getElementsByTagName('tagName')`
- `querySelectorAll('css selector')`

Example: Selecting Elements Using ID attribute

```
<!DOCTYPE html>
<html>
  <head>
    <title>Document Object Model</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="css/c05.css">
  </head>
  <body>
    <div id="page">
      <h1 id="header">List King</h1>
      <h2>Buy groceries</h2>
      <ul>
        <li id="one" class="hot"><em>fresh</em> figs</li>
        <li id="two" class="hot">pine nuts</li>
        <li id="three" class="hot">honey</li>
        <li id="four">balsamic vinegar</li>
      </ul>
    </div>
    <script src="js/get-element-by-id.js"></script>
  </body>
</html>
```

```
// Select the element and store it in a variable.
var el = document.getElementById('one');

// Change the value of the class attribute.
el.className = 'cool';
```

```
<!DOCTYPE html>
<html>
▶<head>…</head>
▼<body>
  ▼<div id="page">
     <h1 id="header">List King</h1>
     <h2>Buy groceries</h2>
    ▼<ul>
...   ▶<li id="one" class="cool">…</li> == $0
      <li id="two" class="hot">pine nuts</li>
      <li id="three" class="hot">honey</li>
      <li id="four">balsamic vinegar</li>
    </ul>
  </div>
  <script src="js/get-element-by-id.js"></script>
 </body>
</html>
```

Example: Selecting Elements Using Classname

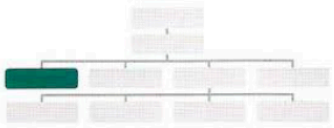
```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Document Object Model </title>
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <link rel="stylesheet" href="css/c05.css">
7    </head>
8    <body>
9      <div id="page">
10       <h1 id="header">List</h1>
11       <h2>Buy groceries</h2>
12       <ul>
13         <li id="one" class="hot"><em>fresh</em> figs</li>
14         <li id="two" class="hot">pine nuts</li>
15         <li id="three" class="hot">honey</li>
16         <li id="four">balsamic vinegar</li>
17       </ul>
18     </div>
19     <script src="js/get-elements-by-class-name.js"></script>
20   </body>
21 </html>
```

```
1  var elements = document.getElementsByClassName('hot'); // Find hot items
2
3  if (elements.length > 2) {                              // If 3 or more are found
4
5    var el = elements[2];                 // Select the third one from the NodeList
6    el.className = 'cool';                // Change the value of its class attribute
7
8  }
```

```
<!DOCTYPE html>
<html>
▶<head>…</head>
▼<body>
  ▼<div id="page">
     <h1 id="header">List</h1>
     <h2>Buy groceries</h2>
    ▼<ul>
      ▶<li id="one" class="hot">…</li>
       <li id="two" class="hot">pine nuts</li>
...    <li id="three" class="cool">honey</li> == $0
       <li id="four">balsamic vinegar</li>
    </ul>
  </div>
  <script src="js/get-elements-by-class-name.js"></script>
 </body>
</html>
```

DOM Queries that return more than one element

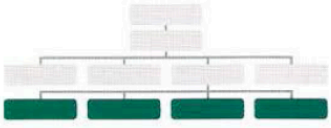
When a DOM method can return more than one element, it returns a Nodelist (even if it only finds one matching element). Following are the four different queries that all return a NodeList.



`getElementsByTagName('h1')`

Even though this query only returns one element, the method still returns a NodeList because of the potential for returning more than one element.

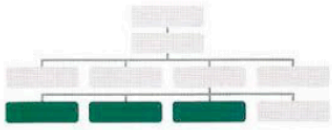
INDEX NUMBER & ELEMENT
0 `<h1>`



`getElementsByTagName('li')`

This method returns four elements, one for each of the `<li>` elements on the page. They appear in the same order as they do in the HTML page.

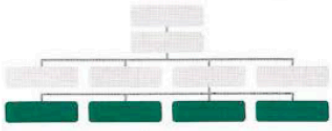
INDEX NUMBER & ELEMENT
0 `<li id="one" class="hot">`
1 `<li id="two" class="hot">`
2 `<li id="three" class="hot">`
3 `<li id="four">`



`getElementsByClassName('hot')`

This NodeList contains only three of the `<li>` elements because we are searching for elements by the value of their `class` attribute, not tag name.

INDEX NUMBER & ELEMENT
0 `<li id="one" class="hot">`
1 `<li id="two" class="hot">`
2 `<li id="three" class="hot">`



`querySelectorAll('li[id]')`

This method returns four elements, one for each of the `<li>` elements on the page that have an `id` attribute (regardless of the values of the `id` attributes).

INDEX NUMBER & ELEMENT
0 `<li id="one" class="hot">`
1 `<li id="two" class="hot">`
2 `<li id="three" class="hot">`
3 `<li id="four">`

The `item()` method:

NodeList have a method called item() which return an individual node from the NodeLists. Specify the index number of the element you want as a parameter of the method.

```
var elements = document.getElementsByClassName('hot')
if (elements.length >= 1) {
    var firstItem = elements.item(0);
}
```

1

Select elements that have a **class** attribute whose value is **hot** and store the NodeList in a variable called **elements**.

2

Use the **length** property to check how many elements were found. If 1 or more are found, run the code in the **if** statement.

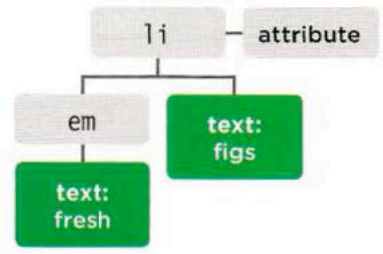
3

Store the first element from the NodeList in a variable called **firstItem**. (It says 0 because index numbers start at zero.)

Access & Update Text with textContent & innerText

Consider the following element:

```
<li id="one"><em>fresh</em> figs</li>
```




```
document.getElementById('one').textContent;
```

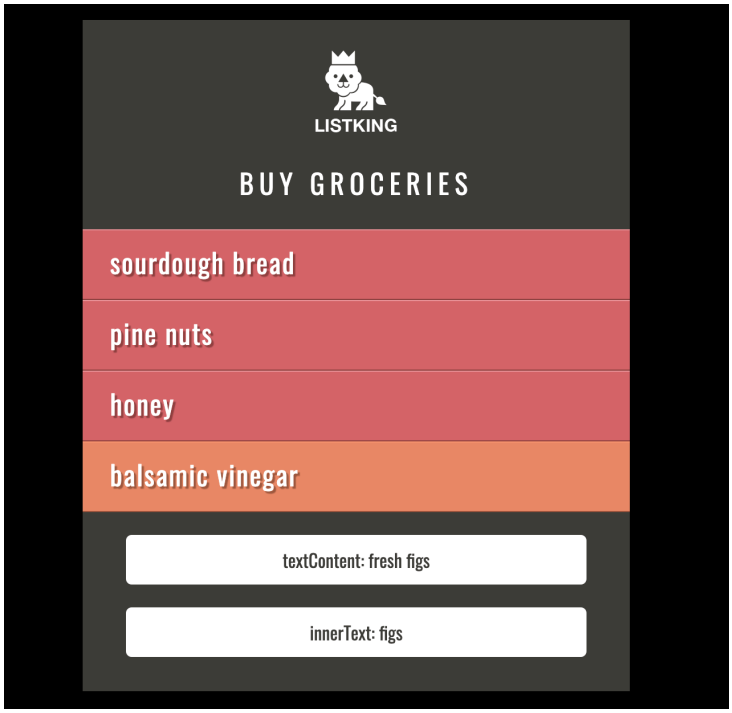
To collect the text from the <li> elements in our example (and ignore any markup inside the element) you can use the textContent property on the containing <li> element. In this case it would return the value: fresh figs.

You can also use this property to update the content of the element; it replaces the entire content of it (including any markup). In order to demonstrate the difference between textContent and innerText, this example features a CSS rule to hide the contents of the element.

Example:

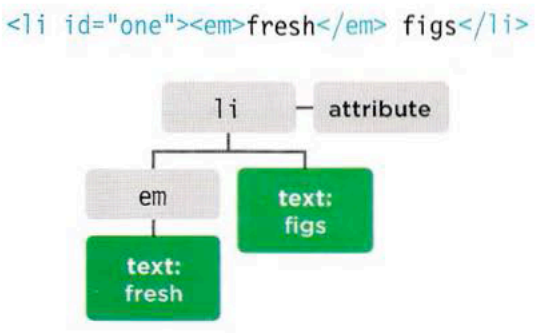
```html
<!DOCTYPE html>
<html>
  <head>
    <title>Document Object Model </title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="css/c05.css">
    <style>
      /* This page hides the <em> elements to demonstrate difference
      between innerText and textContent */
      em {display: none;}
    </style>
  </head>
  <body>
    <div id="page">
      <h1 id="header">List</h1>
      <h2>Buy groceries</h2>
      <ul>
        <li id="one" class="hot"><em>fresh</em> figs</li>
        <li id="two" class="hot">pine nuts</li>
        <li id="three" class="hot">honey</li>
        <li id="four">balsamic vinegar</li>
      </ul>
      <div id="scriptResults"></div>
    </div>
    <script src="js/inner-text-and-text-content.js"></script>
  </body>
</html>
```

```javascript
var firstItem = document.getElementById('one'); // Find first list item
var showTextContent = firstItem.textContent;    // Get value of textContent
var showInnerText = firstItem.innerText;        // Get value of innerText

// Show the content of these two properties at the end of the list
var msg = '<p>textContent: ' + showTextContent + '</p>';
    msg += '<p>innerText: ' + showInnerText + '</p>';
var el = document.getElementById('scriptResults');
el.innerHTML = msg;

firstItem.textContent = 'sourdough bread';       // Update the first list item
```



Access & Update Text & Markup with innerHTML

Using the innerHTML property, you can access and amend the contents of an element, including any child elements. When getting HTML from an element, the innerHTML property will get the content of an element and return it as one long string, including any markup that the element contains.



Example:

```
<!DOCTYPE html>
  <html>
  <head>
    <title>Document Object Model</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="css/c05.css">
  </head>
  <body>
    <div id="page">
      <h1 id="header">List</h1>
      <h2>Buy groceries</h2>
      <ul>
        <li id="one" class="hot"><em>fresh</em> figs</li>
        <li id="two" class="hot">pine nuts</li>
        <li id="three" class="hot">honey</li>
        <li id="four">balsamic vinegar</li>
      </ul>
    </div>
    <script src="js/inner-html.js"></script>
  </body>
</html>
```

```
// Store the first list item in a variable
var firstItem = document.getElementById('one');

// Get the content of the first list item
var itemContent = firstItem.innerHTML;

// Update the content of the first list item so it is a link
firstItem.innerHTML = '<a href=\"http://example.org\">' + itemContent + '</a>';
```

Events

When you browse the web, your browser registers different types of events. It's the browser's way of saying, "Hey, this just happened." Your script can then respond to these events. Scripts often respond to these events by updating the content of the web page (via the Document Object Model) which makes the page feel more interactive.

Different Types of Events:

UI EVENTS

Occur when a user interacts with the browser's user interface (UI) rather than the web page

EVENT	DESCRIPTION
`load`	Web page has finished loading
`unload`	Web page is unloading (usually because a new page was requested)
`error`	Browser encounters a JavaScript error or an asset doesn't exist
`resize`	Browser window has been resized
`scroll`	User has scrolled up or down the page

KEYBOARD EVENTS

Occur when a user interacts with the keyboard (see also `input` event)

EVENT	DESCRIPTION
`keydown`	User first presses a key (repeats while key is depressed)
`keyup`	User releases a key
`keypress`	Character is being inserted (repeats while key is depressed)

MOUSE EVENTS

Occur when a user interacts with a mouse, trackpad, or touchscreen

EVENT	DESCRIPTION
`click`	User presses and releases a button over the same element
`dblclick`	User presses and releases a button twice over the same element
`mousedown`	User presses a mouse button while over an element
`mouseup`	User releases a mouse button while over an element
`mousemove`	User moves the mouse (not on a touchscreen)
`mouseover`	User moves the mouse over an element (not on a touchscreen)
`mouseout`	User moves the mouse off an element (not on a touchscreen)

FOCUS EVENTS	Occur when an element (e.g., a link or form field) gains or loses focus
EVENT	**DESCRIPTION**
`focus / focusin`	Element gains focus
`blur / focusout`	Element loses focus

FORM EVENTS	Occur when a user interacts with a form element
EVENT	**DESCRIPTION**
`input`	Value in any `<input>` or `<textarea>` element has changed (IE9+) or any element with the `contenteditable` attribute
`change`	Value in select box, checkbox, or radio button changes (IE9+)
`submit`	User submits a form (using a button or a key)
`reset`	User clicks on a form's reset button (rarely used these days)
`cut`	User cuts content from a form field
`copy`	User copies content from a form field
`paste`	User pastes content into a form field
`select`	User selects some text in a form field

MUTATION EVENTS*	Occur when the DOM structure has been changed by a script * To be replaced by mutation observers (see p284)
EVENT	**DESCRIPTION**
`DOMSubtreeModified`	Change has been made to document
`DOMNodeInserted`	Node has been inserted as a direct child of another node
`DOMNodeRemoved`	Node has been removed from another node
`DOMNodeInsertedIntoDocument`	Node has been inserted as a descendant of another node
`DOMNodeRemovedFromDocument`	Node has been removed as a descendant of another node

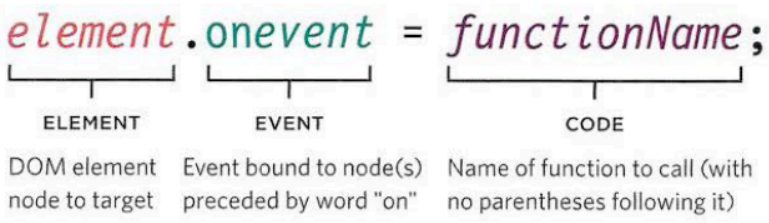
How Events Trigger JavaScript Code

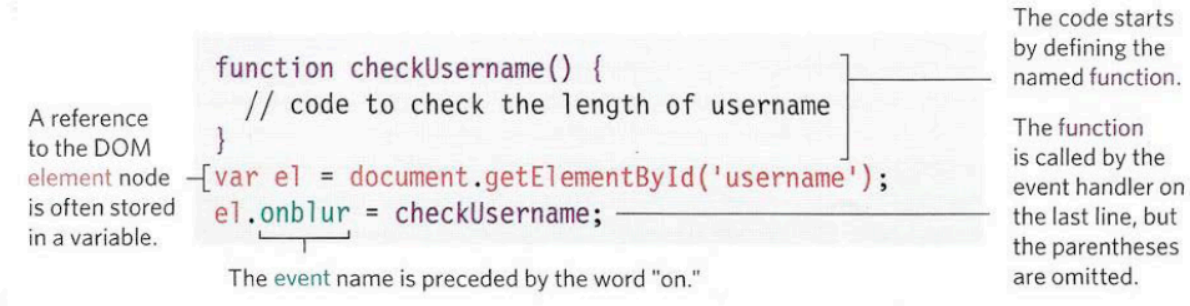
When the user interacts with the HTML on a web page, there are three steps involved in getting it to trigger some JavaScript code. Together these steps are known as event handling.

1. Select the element node(s) you want the script to respond to.
2. Indicate which event on the selected node(s) will trigger the response. Programmers call this binding an event to a DOM node.
3. State the code you want to run when the event occurs.

Traditional DOM Event Handler

All modern browsers understand this way of creating an event handler, but you can only attach one function to each event handler. Here is the syntax to bind an event to an element using an event handler, and to indicate which function should execute when that event fires:



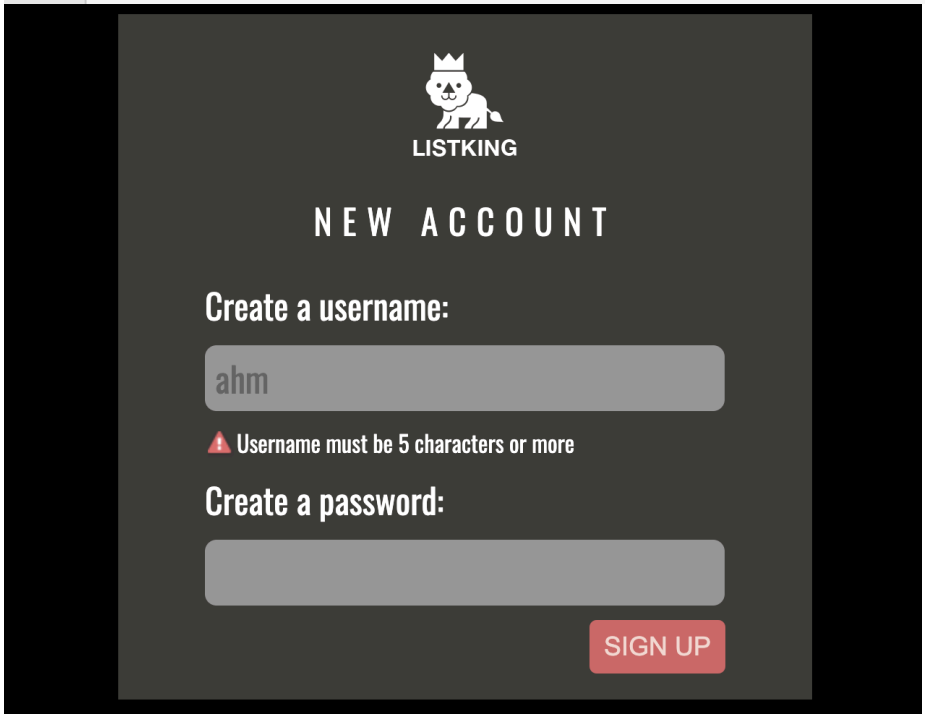


When a function is called, the parentheses that follow its name tell the JavaScript interpreter to "run this code now." We don't want the code to run until the event fires, so the parentheses are omitted from the event handler on the last line.

Example: Using DOM Event Handler

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Events </title>
5      <link rel="stylesheet" href="css/c06.css" />
6    </head>
7    <body>
8      <div id="page">
9        <h1>List King</h1>
10       <h2>New Account</h2>
11       <form method="post" action="http://www.example.org/register">
12
13         <label for="username">Create a username: </label>
14         <input type="text" id="username" />
15         <div id="feedback"></div>
16
17         <label for="password">Create a password: </label>
18         <input type="password" id="password" />
19
20         <input type="submit" value="sign up" />
21
22       </form>
23     </div>
24     <script src="js/event-handler.js"></script>
25   </body>
26 </html>
```

```
1  function checkUsername() {                              // Declare function
2    var elMsg = document.getElementById('feedback');     // Get feedback element
3    if (this.value.length < 5) {                         // If username too short
4      elMsg.textContent = 'Username must be 5 characters or more';  // Set msg
5    } else {                                             // Otherwise
6      elMsg.textContent = '';                            // Clear message
7    }
8  }
9
10 var elUsername = document.getElementById('username'); // Get username input
11 elUsername.onblur = checkUsername;  // When it loses focus call checkuserName()
12
```



In this example, the event handler appears on the last line of the JavaScript. Before the DOM event handler, two things are put in place:

1. If you use a named function when the event fires on your chosen DOM node, write that function first. (You could also use an anonymous function.)

2. The DOM element node is stored in a variable. Here the text input (whose id attribute has a value of username) is placed into a variable called elUsername.

   When using event handlers, the event name is preceded by the word "on" (onsubmi t , onchange, onfocus, onblur, onmouseover, onmouseout, etc).

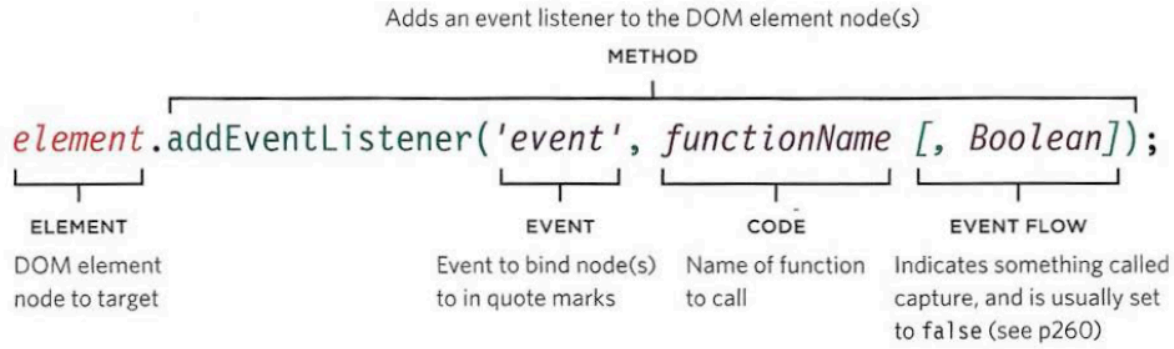
3. On the last line of the code sample above, the event handler elUsername. onblur indicates that the code is waiting for the blur event to fire on the element stored in the variable called elUsername.

This is followed by an equal sign, then the name of the function that will run when the event fires on that element. Note that there are no parentheses on the function name. This means you cannot pass arguments to this function.

Event Listeners

Event listeners are a more recent approach to handling events. They can deal with more than one function at a time but they are not supported in older browsers.

Here is the syntax to bind an event to an element using an event listener, and to indicate which function should execute when that event fires:



References

1. Duckett J. JavaScript & jQuery. Wiley VCH; 2015.

Lab Tasks:

1. The Reading List:
   - Create an array of objects using constructor notation, where each object describes a book and has properties for the title (a string), author (a string), and alreadyRead (a boolean indicating if you read it yet) and a method ReadBook(), which alter the value of property alreadyRead.
   - Iterate through the array of books. For each book, log the book title and book author like so: "The Hobbit by J.R.R. Tolkien".
   - Now use an if/else statement to change the output depending on whether you read it yet or not. If you read it, log a string like 'You already read "The Hobbit" by J.R.R. Tolkien', and if not, log a string like 'You still need to read "The Lord of the Rings" by J.R.R. Tolkien.

2. Registration Form:
   Consider the registration form created in Lab 03 and write the JavaScript codes to handle the following:
   - The minimum Length of username should be 5.
   - Address should not have any special characters
   - The phone number starts with +92

3. Calculator
   Using HTML, CSS and JavaScript create a simple calculator that will perform addition, subtraction and multiplication.