# Chapter 1: The ABC of Programming

-A script is a series of instructions that a computer can follow to achieve a goal.

-To write a script, you need to 1st state your goal and then list the tasks that need to be completed in order to achieve it:  
+Define the goal.

+Design the script.

+Code each step.

-Designing a script: tasks: The high-level view of tasks can be presented using a flowchart.

-Designing a script: steps: When you ready to code the script, steps can then be translated into lines of code.

-The document object represents an HTML page. Using document object, we can access and change what content users see on page and respond. Like other objects, document has: properties + methods + events

-How a browser sees a web page

+The browser receives an HTML page

+It creates a model of page and store it in memory.

+It shows the page on screen using a rendering engine (may use CSS)

-How HTML, CSS & JavaScript fit together

+HTML (Content Layer): give the page structure and adds semantics

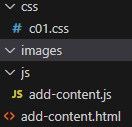
+CSS (Presentation Layer): enhances the HTML page with rules that state how HTML content is presented

+JS (Behavior Layer): where we change how the page behaves, adding interactivity.

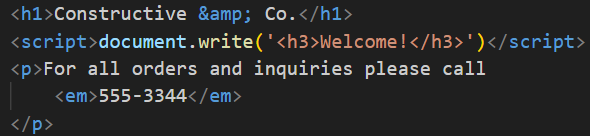
-Progressive enhancement: 3 layers form the basis of this popular approach to building web page: Start with HTML, then add CSS rules, JS is added last.

-Sample code: www.javascriptbook.com

-Creating a basic JS:



-Linking to a JS file from HTML page: Use <script src>  
-Placing the script in the page:



-JS runs where it is found in the HTML: When browser comes across a <script>, it stops to load the script and then checks to see if it needs to do anything.

# Chapter 2: Basic JavaScript Instructions

-Statements: each individual instruction or step, should end with a semicolon.

-Comment: //, /\* \*/

-Variable: store data

-Variables: how to declare them

+variable keyword + variable name: var quantity;

-Varibles: how to assign them a value

+variable name + assignment operator + variable value: quantity = 3;

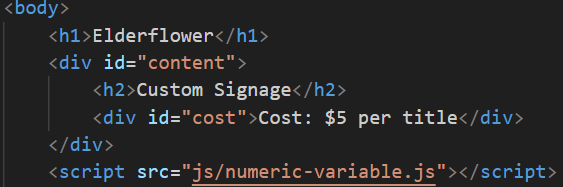
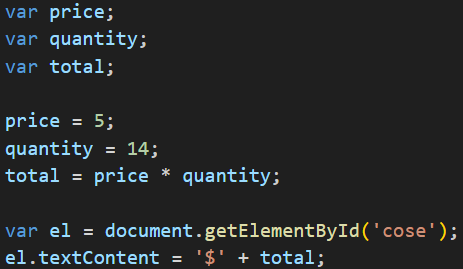
-Data types

+Numeric data type

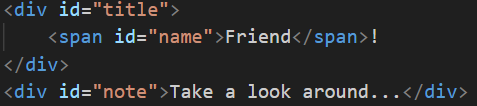
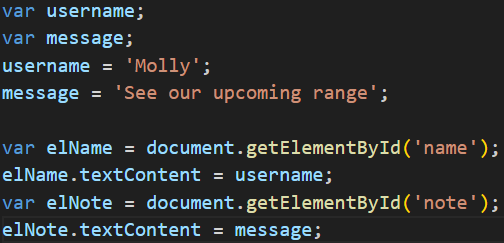
+String data types: can be single or double quotes

+Boolean data type

-Using a variable to store a number



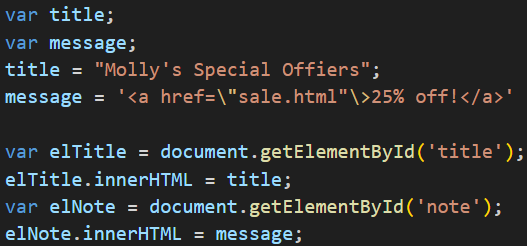
-Using a variable to store a string



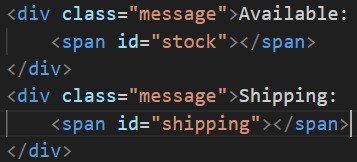
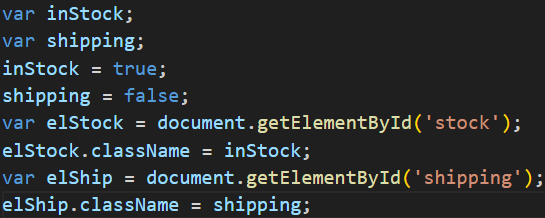
-Using quotes inside a string

+” ‘’ ” or ‘ “ “ ’

+Escaping the quotation characters: use backward slash before any type of quote mark that appears with a string ->the following character is part of string.

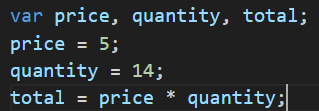
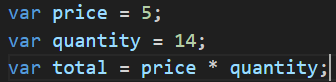


-Using a variable to store a Boolean



+The value are used in class attributes of HTML elements. These values trigger CSS class rules: true show a check, false show a cross.

-Shorthand for creating variable

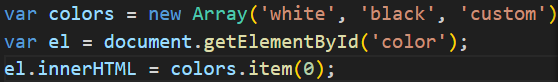
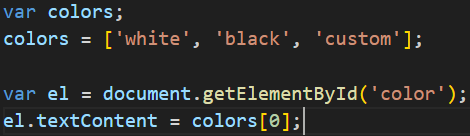




-Changing the value of a variable

-Rules for naming variable: Same in Java

-Arrays: store a list of values



+Number of items: colors.length;

-Expressions: evaluates into a single value. There are 2:

+assign a value to a variable

+use 2 or more values to return a single value

-Operators

+Assignment

+Arithmetic operators

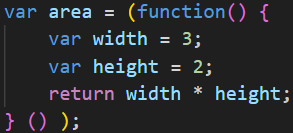
+String operators

+Comparison operators

+Logical operators

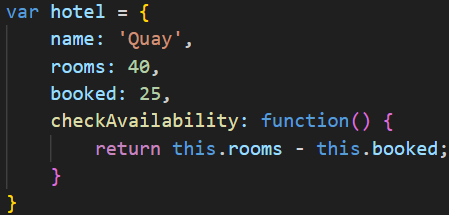
# Chapter 3: Functions, Methods & Objects

-Immediately invoked function expressions:

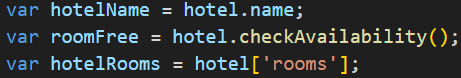


-Variable scope: local variable + global variable

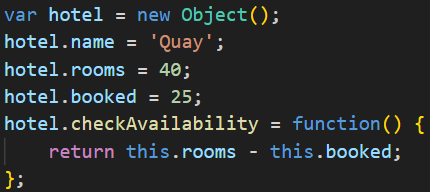
-Creating object:



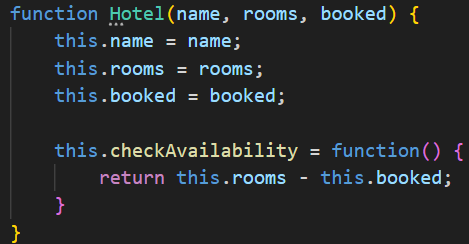
+Access object and dot notation:



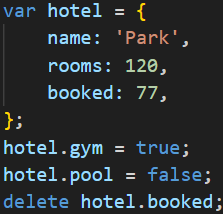
+Creating object: constructor notation



+Create many objects: constructor notation



+Add and remove properties:



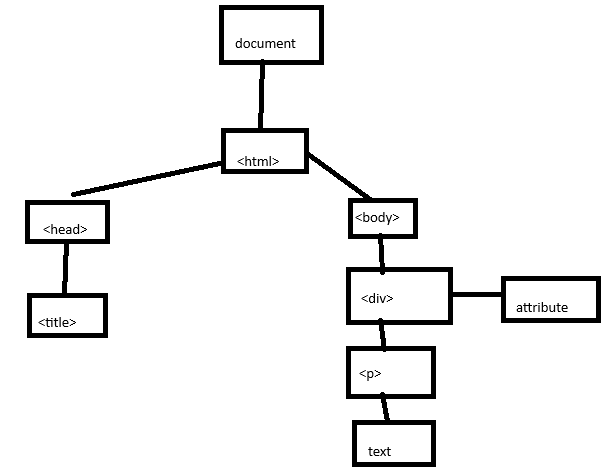
-Arrays are objects: They hold a related set of key/value pairs: costs = [420, 460, 230, 620];

-3 groups of built-in objects:

+Browser object model:

Window (current browser window or tab)-> document (current web page) + history (pages in browser history) + location (url of current page) + navigator (information about browser) + screen (device’s display information)

+Document object model:



-Global JS objects:

+String: for working with string values

+Number: For working with numeric values

+Boolean: for working with Boolean values

+Date: represent and handle dates

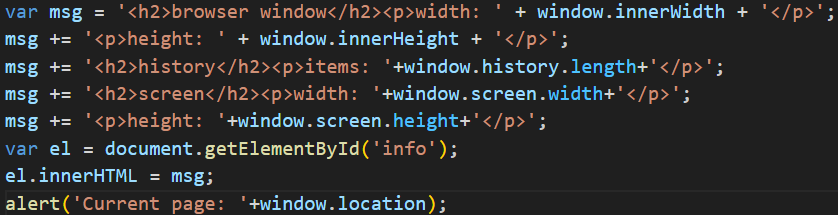
+Math: work with numbers and calculations

+Regex: match patterns within strings of text

-Browser object model: The window object

|  |  |
| --- | --- |
| Property | Description |
| window.innerHeight | Height of window |
| window.innerWidth | Width of window |
| window.pageXOffset | Distance document has been scrolled horizontally |
| window.pageYOffset | Distance document has been scrolled vertically |
| window.screenX | X-coordinate of pointer, relative to top left corner of screen |
| window.screenY | Y-coordinate of pointer, relative to top left corner of screen |
| 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 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 | Access screen object and find value of its width property |
| window.screen.height | Access screen object and find value of its height property |

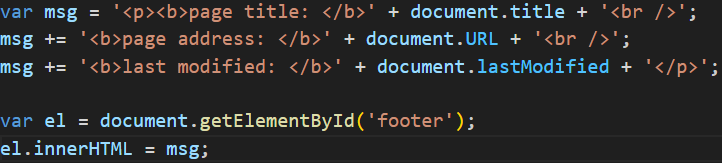
|  |  |
| --- | --- |
| Method | Description |
| window.alert() | Create dialog box with message (user must click OK button to close it) |
| window.open() | Open new browser window with URL specified as parameter |
| window.print() | Tell browser that user wants to print contents of current page |



-The document object model (**DOM**): The **document** object

|  |  |
| --- | --- |
| **Property** | **Description** |
| document.title | Title of current document |
| document.lastModified | Date on which document was last modified |
| document.URL | Return string containing URL of current document |
| document.domain | Return domain of current document |

|  |  |
| --- | --- |
| **Method** | **Description** |
| document.write() | Write text to document |
| document.getElementById() | Return element, if there is an element with value of id attribute that matches |
| document.querySelectorAll() | Return list of element that match a CSS selector, which is specified as a parameter |
| document.createElement() | Create new element |
| document.createTextNode() | Create new text node |



-Global objects: string object

+Property: length

+methods: toUpperCase(), toLowerCase(), charAt(), indexOf(), lastIndexOf(), subString(), split(), trim(), replace()

-Data types revisited:

+5 primitive data types: String, Number, Boolean, Undefined, Null

+Complex data type: Object, Arrays, Functions

-Global object: Number Object

|  |  |
| --- | --- |
| **Method** | **Description** |
| isNaN() | Check if the valiue is not number |
| toFixed() | Rounds to specified number of decimal places (return string) |
| toPrecision() | Rounds to total number of places (return string) |
| toExpotential() | Return string representing the number in exponential notation |

-Global objects: Math object

+Property: Math.PI

+Methods: Math.round(), Math.sqrt(), Math.ceil(), Math.floor(), Math.random()

-Global object: Date object

+var today = new Date();

+getDate()/setDate(), getDay() 0-6, getFullYear()/setFullYear(), getHours()/setHours() 0-23, getMilliseconds()/setMilliseconds() 0-999, getMinutes()/setMinutes() 0-59, getMonth()/setMonth() 0-11, getSeconds()/setSeconds() 0-59, getTime()/setTime(), getTimeZoneOffset(), toDateString(), toTimeString(), toString()

+Use format: YYYY, MM, DD, HH, MM, SS or

MMM (Mon/Tue) DD, YYYY HH:MM:SS

# Chapter 4: Decisions & Loops

-Evaluating conditions & conditional statement: if else

-Comparison operators: == != === !== > >= < <=

-Logical operators: && || !  
-Loops: for, while, do while

# Chapter 5: Document Object Model

-Document object model (**DOM**) specifies how browsers should create a model of HTML page and how JS can access and update content of web page while it is in the browser window.

-DOM is neither part of HTML or JS, it’s separate set of rules. It’s implemented by all major browser makers, ane covers 2 primary areas:

+Make a model of HTML page: When browser loads a web page, it creates a model of page in memory. DOM specifies the way in which browser should structure this model using DOM tree.

+Access and change HTML page: DOM defines methods and properties to access and update each object in this model.

-DOM is an API: user interfaces let human interact with programs, APIs let programs (and scripts) talk to each other.

-**DOM tree is a model of a web page**

+As browser loads a web page, it creates a model of page. The model is DOM tree, and it’s stored in browser’s memory. It’s consists of 4 main types of nodes

+Document node: represent the entire page

+Element nodes: relationships between document and all element nodes are: parent, children, siblings, ancestors and descendants.

+Attribute nodes: are not children of element that carries them, they are part of that element.

+Text nodes: Can’t have children, if element contains text and another child element, child element isn’t a child of text node but rather a child of containing element.

-**Working with DOM tree**

+**Step** **1**: **Access the elements**

Select an **individual element node**: getElementById() + querySelector(css selector), traverse from one element to another within DOM tree

Select **multiple elements** (nodelists): getElementsByClassName(class), getElementByTagName(tag name), querySelectorAll(css selector)

**Traverse** between element nodes: parentNode, previousSibling/nextSibling, firstChild/lastChild

+**Step 2: Work with those elements**

Access/update **text nodes**: firstChild, nodeValue

Work with **HTML content**: innerHTML, textContent, createElement(), createTextNode(), appendChild()/removeChild()

Access or update **attribute values**: className/id, hasAttribute(), getAttribute(), setAttribute(), removeAttribute()

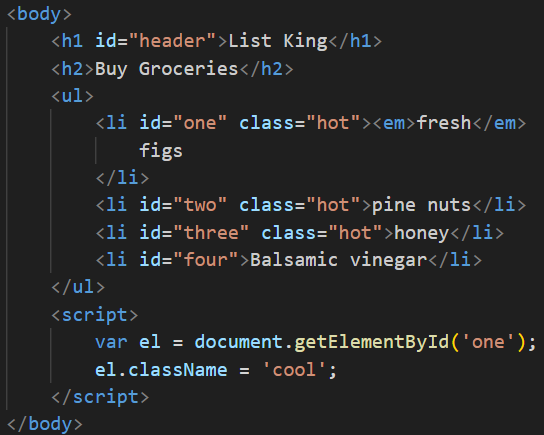
**-Caching DOM queries**

+**DOM queries**: methods that find elements in DOM tree

+When a script select an element to access/update, then interpreter must find the element(s) in Dom tree.

+variable **doesn’t store the element**, it stores a **reference** to where that node is in DOM tree.

-**Select element** using **ID attributes**:



+Use a property **className** to update value of class attribute of element stored in el variable.

-**Nodelists**: DOM queries that return more than one element

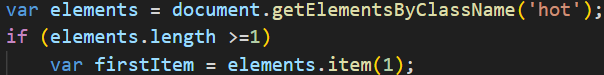
+getElementsByTagName(‘h1)

+getElementsByClassName(‘hot’)

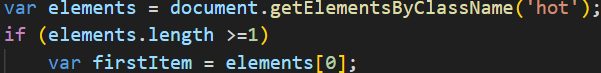
+querySelectorAll(‘li[id]’): return elements <li> having [id] attribute

-**Select** an **element** from a **nodelist**:

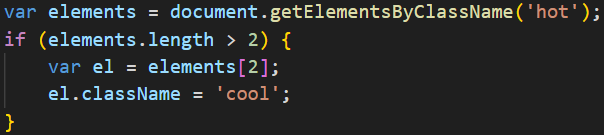
+**item()**



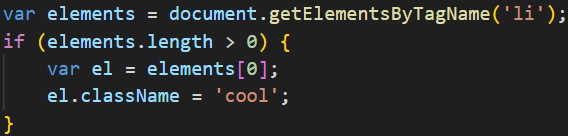
+**Array syntax**:



-Select elements using **class attributes**



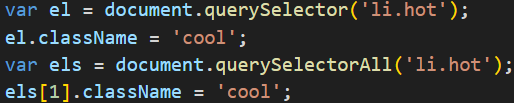
-Select elements by **tag name**



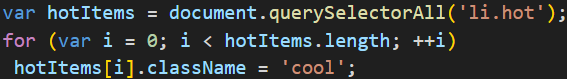
-Select elements using **CSS selectors**:

+**querySelector()**: return 1st element node that matches CSS style selector

+**querySelectorAll():** return NodeList of all matches



-**Repeat** actions for an entire **nodelist**:



-**Traverse** the **DOM**

+**parentNode**: finds the element node for the containing element in HTML

+**previousSibling, nextSibling**

+**firstChild, lastChild**

-**whitespace nodes**: Some browsers add a text node whenever they com across whitespace between elements. So properties above return different elements in different browsers. You could **strip all whitespace** out of HTML page -> harder to read.

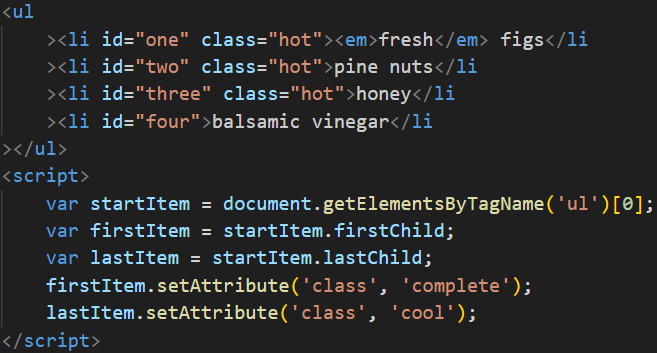
+One of the most popular ways to address this kind of problem is use **jQuery**.

-Previous & next sibling



-First & last child:

+These properties also return consistent results if there is **whitespace** between elements. We can make **closing tags put next to opening tags** of next element



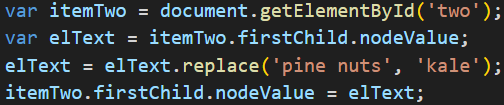
-**Get/Update** **element content**

+**Text nodes**: **nodeValue**: access text from node

+**Contain element**

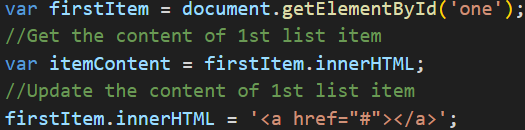
|  |  |
| --- | --- |
| **Property** | **Description** |
| innerHTML | Get/set text & markup |
| textContent | Get/set text only |
| innerText | Get/set text only |

-Access & Update a text node with **nodeValue**:

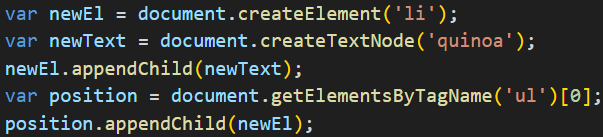


-Add or remove HTML content

+**innerHTML**: update entire fragment



+**DOM manipulation methods**: target individual nodes in DOM tree: **createElement()**, **createTextNode()**, **appendChild()**

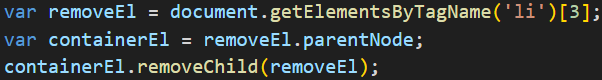


+**Remove elements via DOM manipulation**

Store element to be removed in a variable

Store parent of that element in a variable: **parentNode**

Remove element from its containing element: **removeChild()**



-Compare techniques: update HTML content

|  |  |  |  |
| --- | --- | --- | --- |
|  | document.write | element.innerHTML | DOM manipulation |
| Advantage | Quick and easy way to show beginners how content can be added to a page | Get/update the entire content of any element (including markup) as a string | Use a set of methods and properties to access, create, and update elements and text nodes |
| Disadvantagse | Only works when the page initially loads.  If you use if after page has loaded: overwrite the whole page, not add the content the page, create a new page  Cause problems with XHTML  Very rarely used these days | Not used to add content that has come from user (username or blog comment), it can pose a security risk.  Can be difficult to isolate single element that you want to update within a larger DOM fragment  Event handlers may no longer work as intended | Make a lot of changes to content of page, slower than innerHTML  Need to write more code to achieve the same thing compared with innerHTML |

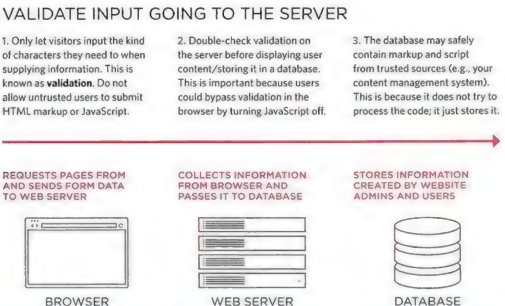
-Cross-Site Scripting (XSS) attacks

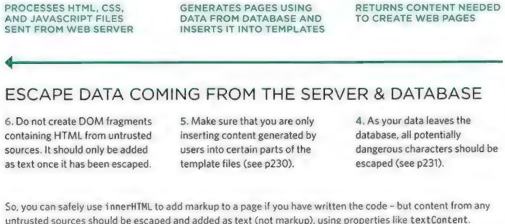
+If you add HTML to a page using innerHTML(or jQuery methods), aware of Cross-Site Scripting Attack (XSS), an attacker could gain access to users’ account.

+How XSS happesn: XSS involves an attacker placing malicious code into a site. Websites often feature content created by many different people (Users can create profiles or add comments, Files such as images and video may by uploaded, Data can come from 3rd party sites such as Facebook, Twitter, news tickers)

+What can these attacks do: XSS can give attacker access to information in (the DOM, website’s cookie, Session token: information that identifies you from other users when you log into a site)

-Defend against XSS:





-XSS: Validation & templates

+Make sure uses can only input characters they need to use and limit where this content will be shown on page

+Filter or validate input

+Limit where user content goes

-XSS: Escaping & Controlling markup

+Escaping user content: all data from untrusted sources should be escaped on server before shown on page. Most server-side languages offer helper functions that will strip-out or escape malicious code.

HTML:



JS: Never include data from untrusted sources in JS. It involves escaping all ASCII characters.

URLS: If you have links containg user input ,use **encodeURIComponent()** to encode user input. It encodes the characters:

+Adding user content

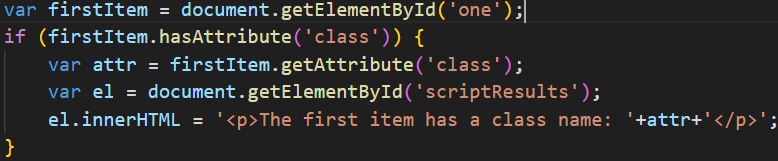
JavaScript: use textContent or innerText, don’t use innerHTML

jQuery(): use .text(), don’t use .html()

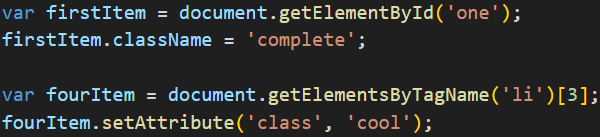
You can still use innerHTML and .html() if: control all of markup being generated (don’t allow user content that could contain markup) + user’s content is escaped and added as text using approaches

-Attribute Nodes: Once you have element node, you can use other property and methods on that element node to access and change attributes: getAttribute(), has Attribute(), setAttribute(), removeAttribute(), className, id

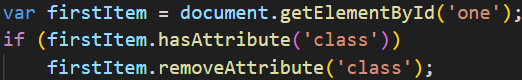
-Check for an attribute and get it’s value:



-Create attributes & change their values



-Remove attributes

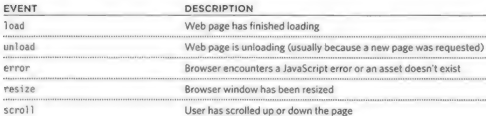


-Examine the DOM in Chrome: Inspect->Elements->Properties

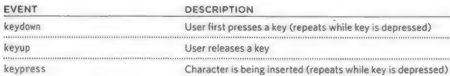
# Chapter 6: Events

-Different Event Types

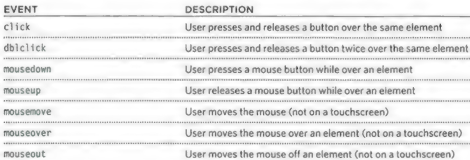
+UI events: Occur when user interacts with browser’s UI rather than web page:



+Keyboard events: user interact with keyboard



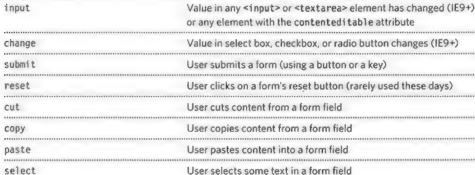
+Mouse events: user interacts with a mouse, trackpad, or touchscreen



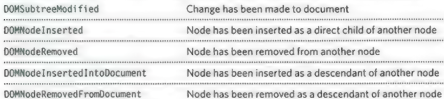
+Focus event: an element (link/form) gain or loses focus



+Form event: user interacts with form element



+Mutation events: DOM structure has been changed by a script (replaced by mutation observers)



-Terminology

+Event fire or are raised: if user is tapping on a link, a click event would fire in browser

+Events trigger scripts: when click event fires on element, it could trigger a script that enlarges the selected item.

-How events trigger JS code:

+Select element node(s) the script respond to.

+Indicate the event on node(s)

+State the code to run when event occurs

-3 ways to bind an event to an element:

+HTML event handler attributes (DO NOT USE)  
<a> can have **onclick, onmouseover, onmouseout**

<for> have **onsubmit**

<input> have **onkeypress, onfocus, onblur**

+Traditional DOM event handlers

A screen shot of a computer code

AI-generated content may be incorrect.

+DOM level 2 event listener: a more recent approach to handling events. They can deal with more than one function at a time.

A white paper with black text

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

Using parameter with event listener: wrap function call in anonymous function

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

-Event Flow

+HTML elements nest inside other elements. If you hover or click on a link, you will also be hovering or clicking on its parent elements.

+**Event bubbling**: the event starts at the most specific node and flows outwards to the least specific one. This is the default type of event flow with very wide browser support

+**Event capturing**: The event starts at the least specific node and flows inwards to the most specific one. This is not supported in IE 8 and earlier.

-Why Flow matters

+The flow of events only really matters when code has event handlers on an element and on of its ancestor or descendant elements.

+For DOM event handlers, all modern browsers default to using event bubbling rather than capturing. With event listeners, final parameter in addEventListener() lets you choose the direction to trigger event (true=capturing phase, false = bubbling phase, often default choice)

-Event object: When an event occurs, the event object tells you information about the event, and the element it happened upon.

A white paper with black text

AI-generated content may be incorrect.

-The event object in IE5-8

A computer code with text

AI-generated content may be incorrect.

-Using event listeners with the event object

A computer code with text

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

-**Event Delegation**

+If users interact with lots of elements, adding event listeners to each element can use a lot of memory and slow down performance

+You can place event handlers on containing element and use event object target to find which of its children the event happened on.

-Change default behavior:

+**preventDefault()**: prevent the default behavior of elements. IE5-8 **returnValue**

A close-up of a code

AI-generated content may be incorrect.

+**stopPropagation ()**: stop event bubbling up. IE8 is **cancelBubble**

A computer code with text

AI-generated content may be incorrect.

+Using both methods: prevent the default behavior of element + prevent event from bubbling up or capturing further.

-Using Event Delegation:

A diagram of a program

AI-generated content may be incorrect.

A diagram of a program

AI-generated content may be incorrect.

A close-up of a card

AI-generated content may be incorrect.

A group of text on a white background

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

A computer screen shot of a program code

AI-generated content may be incorrect.

-Different types of events:

+W3C DOM events: DOM events specification managed by W3C (also look after other specifications HTML, CSS, XML). Most of events you meet are part of DOM events

+HTML5 events: details events that browsers are expected to support that are specifically used with HTML.

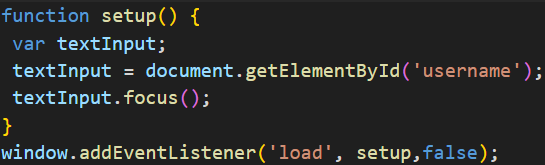
+BOM events: events as part of Browser Object Model.

-User Interface events: UI events occur as a result of interaction with the browser window rather than HTLM

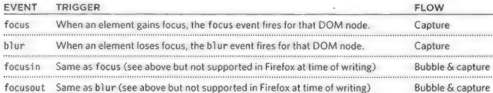
A close-up of a web browser

AI-generated content may be incorrect.

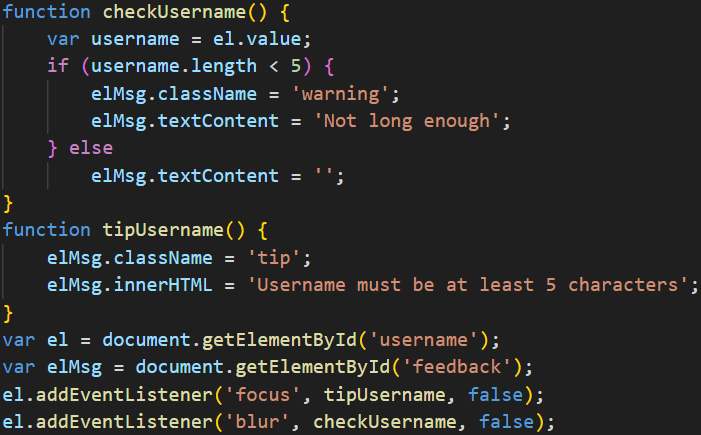
+**load**: trigger scripts that access the content of page. The event is automatically raised by window object when page finished loading HTML and all its resources



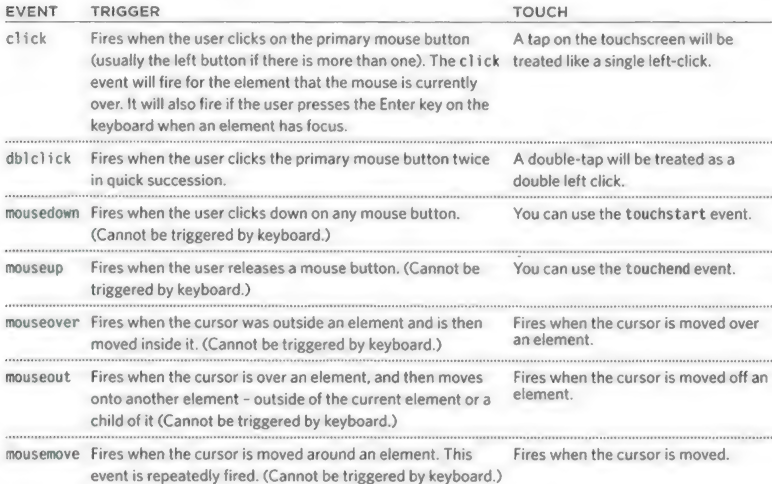
-Focus & Blur events: HTML elements you can interact with, such as links and form elements, can gain focus. These events fire when they gain or lose focus



+Focus & Blur:



-**Mouse events**: fired when the mouse is moved and also when its buttons are clicked. All elements on page support mouse events, and all of these bubble. Prevent default behavior can have unexpected results (click only fires when both mousedown and mouseup fired)



# Chapter 7: jQuery

# Chapter 8: Ajax & JSON

# Chapter 9: APIs