LC 101

Unit 3 - JavaScript

March 2, 2017

JavaScript Arrays

- Arrays in JavaScript are similar to lists in Python
  - Sequence of values
  - Direct access to values using square brackets
  - Variable length
    - In many other languages, arrays are declared with a length and fixed at that length

```
var myArray = [ 0, 1, 2 ];
var x = myArray[0];
```

JavaScript Arrays

- Can add to and remove from the end of an array using **`push`** and **`pop`**

```
var myArray = [ 0, 1, 2 ];
myArray.push(3);  // myArray is now [ 0, 1, 2, 3 ]
var x = myArray.pop();  // x is 3, myArray is [ 0, 1, 2]
```

- Can add to and remove from the front of an array using **`unshift`** and **`shift`**, but this is generally *much slower* and should be avoided
  - Adding or removing at the front requires copying the entire array in memory while adding or removing at the end usually does not

JavaScript Arrays

- The length property of an array is calculated every time it is accessed
  - When looping over an array, avoid accessing length every loop

```
var myArrayLength = myArray.length;
for (var i = 0; i < myArrayLength; i++) {
  // do something with myArray[i]
}
```

- Since multiple variable declarations can be done in the same statement, this is often abbreviated as

```
for (var i = 0, len = myArray.length; i < len; i++) {
  // do something with myArray[i]
}
```

JavaScript Objects

- Objects in JavaScript are vaguely similar to dictionaries in Python
    - Properties of objects are key:value pairs
        - Keys are strings
        - Values can be any JavaScript value, including functions

```
var circle = {
  radius: 1,
  area: function() { return radius * radius * Math.PI }
};
console.log(circle.area());  // outputs 3.14…
circle.radius = 2;
console.log(circle.area());  // outputs 12.56...
```

JavaScript Objects

- If the property name does not have spaces or special characters then it can be accessed using dot notation
- Otherwise, it requires square bracket notation

```
var x = circle.area();
var y = circle["area"]();
```

Function Arguments

- Argument passing in functions is flexible
    - You can pass more or less arguments than a function declares

```
function sum(x, y) { ... }
sum(1);  // this is legal, but y will be undefined in sum
sum(1, 2, 3);  // so is this, but no variable points to 3
```

- Every function has an **`arguments`** object in its environment that contains off the passed values, indexed by position number 0, 1, 2, etc.
    - Looks like an array, but is not
    - Can loop over it, but methods like **`slice`** or **`indexOf`** are missing

Function Arguments

```
function sum() {
  var total = 0;
  for (var i = 0, len = arguments.length; i < len; i++) {
    total += arguments[i];
  }
  return total;
}

console.log(sum(1, 2, 3));  // outputs 6
```

Math Object

- The browser environment has many built-in objects available, such as the Math object
- The Math object contains many math constants and methods

```
var pi = Math.PI;
var x = Math.abs(-2);
```

Global Object

- JavaScript has a global scope. In a browser this global scope is the **`window`** object
    - In other environments (e.g., node.js) the global scope might have a different name
- Creating a variable in the global scope actually adds it to the **`window`** object

JavaScript and HTML

- JavaScript can be included in HTML pages via either script tags or certain attributes
    - A script tag can either embed JavaScript directly or have a src attribute pointing to a separate js file

```
<script>alert('hello');</script>

<script src="js/main.js"></script>

<button onclick="alert('hello');">Hello</button>
```

JavaScript and HTML

- The **`type="text/javascript"`** attribute is no longer needed
- Scripts are loaded and executed as they are encountered in the page
- For external scripts that do not modify the DOM on load the **`async`** attribute can be used to speed loading
  - But scripts will not necessarily be executed in order which could be problematic for scripts that depend on others
  - Execution can also happen before DOM is complete
- For external scripts that can wait to be loaded until after the HTML page is loaded the **`defer`** attribute can be used
  - Will be executed in order after the DOM is built

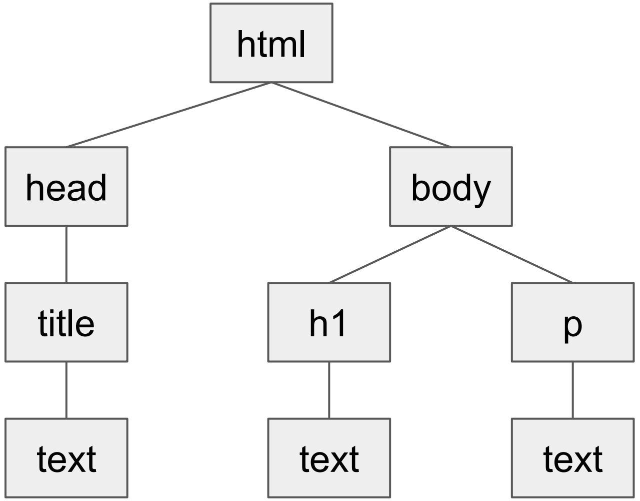
```
<script defer src="js/main.js"></script>
```

Sandbox

- JavaScript that is run in a browser is subject to sandboxing
    - This means that access to resources outside of the browser is limited
        - E.g., cannot read or write files on the client computer

The DOM

- The *Document Object Model* is the in-memory representation of the web page
  - Can be viewed abstractly as nested boxes where each tag is a box
    - This can also be viewed as a *tree* of *nodes*



The DOM

- The **`document`** global variable is a reference to the DOM
  - **`document.documentElement`** is the html object
  - **`document.head`** is the head object
  - **`document.body`** is the body object
- Each node has a **`nodeType`** property
  - Regular elements have type **`document.ELEMENT_NODE`**
  - Text elements have type **`document.TEXT_NODE`**
  - Comments have type **`document.COMMENT_NODE`**

Node References

- Nodes have a myriad of properties referencing related nodes
  - **`childNodes`** an array-like **`NodeList`** object of child nodes
  - **`parentNode`**
  - **`firstChild`**
  - **`lastChild`**
  - **`nextSibling`**
  - **`previousSibling`**
- Though these can be used to move through the document tree, there are generally better options

Finding Elements

- Various functions exist for finding elements
  - **`node.getElementsByTagName(name)`** will return all descendant nodes corresponding to a specific tag
  - **`node.getElementsByClassName(cls)`** will return all descendant nodes with a specific value in their class attribute
  - **`document.getElementById(id)`** will return the node with the specified id attribute value

```
<span id="someId">Hello</span>
```

```
var node = document.getElementById("someId");
```

Creating and Adding Nodes

- We can create new nodes
    - **`document.createElement(type)`** will create an empty regular element node of the specified type
        - It just creates a node; it does not add it to the DOM
    - **`document.createTextNode(text)`** will create a text node
        - Also just creates it; does not add it to the DOM
- And move or add nodes
    - **`parent.appendChild(node)`** will add a node to the end of the parent's children
    - **`node.insertBefore(newNode, existingNode)`**
    - **`node.replaceChild(newNode, existingNode)`**
- Each node can only exist at once place in the DOM
    - Inserting an existing node will remove it from its old location

Creating and Adding Nodes

- We can also set the HTML content of an element by setting the innerHTML property of the node

```
document.getElementById('myPara').innerHTML = '<b>Wow!</b>'
```

Setting Attributes

- Standard attributes (e.g., **`href`**) can be accessed via a property of the same name on the corresponding node
- Non-standard attributes must be accessed via **`node.getAttribute(name)`** and **`node.setAttribute(name, value)`**
    - It is fairly standard practice for custom attributes to start with "**`data-`**"

```
<span id="price" data-contract="oneYear">$10.00</span>
```

```
contract = document.getElementById("price").getAttribute("data-contract");
```

Query Selectors

- **`document.querySelectorAll(selector)`** will return all nodes that match the selector
- **`document.querySelector(selector)`** will return the first matching node
- The selectors are strings that follow the CSS selector rules

```
document.querySelectorAll('p');  // all <p> elements
document.querySelectorAll('.foo');  // all elements of class foo

// all elements of class foo that are inside <p> elements
document.querySelectorAll('p .foo');

// all elements of class foo that are direct children of <p> elements
document.querySelectorAll('p > .foo');
```