FUNDAMENTAL
JAVASCRIPT

Aaron Gustafson
@AaronGustafson
slideshare.net/AaronGustafson

Variables

(i.e. buckets)

Variables

```
var my_var;

var another_var, yet_another_var;
```

Variables

```
var MYVAR,
    myvar,
    myVar,
    MyVar,
    MyVaR;
```

Variables: Scope

```
function myFunc()
{
    var my_var = false;
}

my_var;  // undefined
```

Data Types

(i.e. stuff that goes in buckets)

Data type: Strings

```
var single_quoted = 'my text',
    double_quoted = "more text";

var no_escape_necessary = 'some "text"',
    escaped = 'some \'text\'';

var numeric_string = '06517';
```

Data type: Numbers

```
var  positive = 34,
     negative = -1,
     decimal  = 3.14;
```

Data type: Booleans

```
var yes = true,
    no  = false,
    also_yes = 1, // truthy
    also_no  = 0; // falsey
```

Data type: Arrays

```
var my_cats = [];

my_cats[0] = 'Sabine';
my_cats[1] = 'Dakota';

my_cats; // ['Sabine','Dakota']
```

Data type: Arrays

```
var sabine = [
  'Sabine',     // 0 = name
  'cat',        // 1 = type
  'female',     // 2 = gender
  17,           // 3 = age
  true          // 4 = spayed/neutered
];

sabine[2]; // 'female'
```

Data type: Arrays

```
var sabine = ['Sabine', 'cat', 'female', 14, true],
    dakota = ['Dakota', 'cat', 'male', 13, true];

pets = [ sabine, dakota ];

pets[1][0]; // 'Dakota'
```

Data type: Hashes

```
var sabine = [];

sabine['name']   = 'Sabine';
sabine['type']   = 'cat';
sabine['gender'] = 'female';
sabine['age'] = 14;
sabine['fixed'] = true;

sabine;          // []
sabine['name']; // 'Sabine'
sabine.name;    // 'Sabine'
```

Data type: Objects

```
var sabine = {};

sabine.name = 'Sabine';
sabine.type = 'cat';
sabine.gender = 'female';
sabine.age = 14;
sabine.fixed = true;

sabine;          // Object
sabine['name']; // 'Sabine'
sabine.name;    // 'Sabine'
```

Operators

(i.e. telling JS what to do)

Operators: Arithmetic

```
var one = 2 - 1,
    two = 1 + 1,
    three = 9 / 3,
    four  = 2 * 2,

five  = three + two;
```

Operators: Concatenation

```
'This is a ' + 'string';   // 'This is a string'
```

Operators: Shorthand

```
var my_var = 1;

my_var += 2; // 3
my_var -= 2; // 1
my_var *= 2; // 2
my_var /= 2; // 1

my_var++;   // 2 (after eval.)
my_var--;   // 1 (after eval.)
++my_var;   // 2 (before eval.)
--my_var;   // 1 (before eval.)
```

Operators: Comparison

```
var my_var = 1;

my_var > 2;    // false
my_var < 2;    // true
my_var == 2;   // false
my_var >= 2;   // false
my_var <= 2;   // true
my_var != 2;   // true
my_var === 2; // false
my_var !== 2; // true
```

Operators: Identity

```
function isTrue( value )
{
    return value === true;
}

isTrue( true );  // true
isTrue( false ); // false
isTrue( 1 );     // false
isTrue( 0 );     // false
```

Operators: Logical

```
if ( ! my_var )
{
    // my_var is false, null or undefined (not)
}

if ( my_var > 2 && my_var < 10 )
{
    // my_var is between 2 and 10 (exclusive)
}

if ( my_var > 2 || my_var < 2 )
{
    // my_var is greater or less than 2
    // (i.e. my_var != 2)
}
```

Operators: Logical

```
if ( ! ( my_var < 2 ) )
{
    // my_var is not less than 2
    // (or my_var >= 2)
}

if ( ( my_var > 2 &&
       my_var < 10 ) ||
     my_var == 15 )
{
    // my_var is between 2 and 10 (exclusive)
    // or my_var is 15
}
```

Data type: Dynamic typing

```
var my_var = false;     // boolean

my_var = 14;            // number

my_var = "test";        // string

my_var = [];            // array

my_var = {};            // object

my_var = function(){}; // function
```

Data type: Dynamic typing

```
'This is a ' + 'string';   // 'This is a string'

10 + '20';                 // '1020'
```

Control Structures

(i.e. conducting the symphony)

Conditional Action

```
if ( condition )
{
    statement;
}
```

Semicolons: Use them

```
first statement
second statement
```

Semicolons: Use them

first statement
second statement

compression

first statement second statement

Semicolons: Use them

```
first statement;
second statement;
```

compression

```
first statement;  second statement;
```

Conditional Action

```
if ( 1 > 2 )
{
    console.log( 'something is terribly wrong' );
}
```

Conditional Action

```
if ( 1 > 2 )
{
    console.log( 'something is terribly wrong' );
}
else
{
    console.log( 'everything is okay' );
}
```

Conditional Action

```
console.log(
  1 > 2 ? 'something is terribly wrong' : 'everything is okay'
);
```

Conditional Action

```
if ( height > 6 )
{
   console.log( 'you are tall' );
}
else if ( height > 5.5 )
{
   console.log( 'you are of average height' );
}
else
{
   console.log( 'you are shorter than average' );
}
```

Conditional Action

```
var msg = 'You are ';

switch ( true )
{
   case height > 6:
      msg += 'tall';
      break;
   case height > 5.5:
      msg += 'of average height';
      break;
   default:
      msg += 'shorter than average';
      break;
}

console.log( msg );
```

For Loops

```
for ( initialization ; test condition ; alteration )
{
    statement ;
}
```

For Loops

```
for ( var i=1; i<=10; i++ )
{
    console.log( i );
}
// 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

For Loops

```
for ( var i=1; i<=10; i++ )
{
    console.log( i );
}
// 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

var i = 1;
for ( ; i<=10; )
{
    console.log( i++ );
}
// 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

While Loops

```
initialization;
while ( test condition )
{
    statement;
    alteration;
}
```

While Loops

```
var i = 1;
while ( i < 10 )
{
    console.log( i );
    i += 2;
}
// 1, 3, 5, 7, 9

i; // 11
```

While Loops

```
var i = 11;
while ( i > 10 )
{
    console.log( i++ );
}
// infinite loop (condition is always met)
```

While Loops

```
var i = 10;
while ( i )
{
    console.log( i-- );
}
// 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
```

Functions

(i.e. reusable bundles of logic)

Functions

```
function name( arguments )
{
   statements;
}
```

Functions

```
function isTrue( value )
{
    return value === true;
}

isTrue( true );  // true
isTrue( false ); // false
isTrue( 1 );     // false
isTrue( 0 );     // false
```

Functions

```
function add( a, b )
{
    return a + b;
}

add( 1, 2 );    // 3
add( 4, 5 );    // 9
add( 1, 2, 3 ); // 3
```

Functions

```
function add( a, b, c )
{
   return a + b + c;
}

add( 1, 2 );    // Not a number (NaN)
add( 4, 5 );    // NaN
add( 1, 2, 3 ); // 6
```

Functions

```
function add( a, b, c )
{
   c = c || 0;
   return a + b + c;
}

add( 1, 2 );    // 3
add( 4, 5 );    // 9
add( 1, 2, 3 ); // 6
```

Functions

```
function add()
{
    var total = 0,
        i = 0;
    while ( arguments[i] )
    {
        total += arguments[i++];
    }
    return total;
}

add( 1, 2 );          // 3
add( 1, 2, 3 );       // 6
add( 1, 2, 3, 8 );    // 14
```

Functions

```
function add()
{
    var total = 0,
        i = 0;
    while ( arguments[i] )
    {
        total += arguments[i++];
    }
    return total;
}

add( 1, 2 );           // 3
add( 1, 2, 3 );        // 6
add( 1, 2, 3, 8 );     // 14
add( 1, 2, 'foo', 8 ); // 3foo8
```

Functions

```
function add()
{
   var total = 0,
       i = 0;
   while ( arguments[i] )
   {
      if ( typeof arguments[i] == 'number' )
      {
         total += arguments[i];
      }
      i++;
   }
   return total;
}

add( 1, 2 );           // 3
add( 1, 2, 3 );        // 6
add( 1, 2, 3, 8 );     // 14
add( 1, 2, 'foo', 8 ); // 11
```

Variables: Scope

```
function myFunc()
{
  my_first_var = true;
  var my_second_var = false;
}

window.my_first_var;  // undefined

myFunc();

window.my_first_var;  // true
window.my_second_var; // undefined
```

Variables: Scope

```
function myFunc()
{
  my_first_var = true;
  var my_second_var = false;
}

window.my_first_var;  // undefined

myFunc();

window.my_first_var;  // true
window.my_second_var; // undefined

window.myFunc; // function
```

Variables: Scope

```
function Silly()
{
    a = 10;
    return a *= 2;
}

var a = 10;

a;       // 10
Silly(); // 20
Silly(); // 20
a;       // 20
```

Variables: Scope

```
function Silly()
{
    var a = 10;
    return a *= 2;
}

var a = 10;

a;       // 10
Silly(); // 20
Silly(); // 20
a;       // 10
```

Variables: Scope

```
function Silly()
{
    return a *= 2;
}

var a = 10;
Silly();  // 20
Silly();  // 40
a;        // 40
```

“Anonymous” Functions

```
window.onload = function(){
    // do something
};
```

“Anonymous” Functions

```
(function(){
    // do something
}());
```

“Anonymous” Functions

```
(

    // encapsulates some code

);
```

“Anonymous” Functions

```
(
    function(){
        // defines an anonymous function
    }
);
```

“Anonymous” Functions

```
(
    function(){

    }() // executes it immediately
);
```

“Anonymous” Functions

```
(function(){
    // do something
}());
```

Objects

(i.e. organizers)

Objects

```
var Foo = {};
```

Objects

```
var Foo = {};

Foo.value = 'bar';

Foo.value; // 'bar'
```

Objects

```
var Foo = {};

Foo.value = 'bar';

Foo.doSomething = function(){
    console.log( this.value );
};

Foo.doSomething(); // 'bar'
```

Almost everything's an object

```
var
str_a = '1 2 3 4 5',
str_b = '6 7 8 9';

str_a.length;                   // 9
str_a.concat( ' ', str_b );     // '1 2 3 4 5 6 7 8 9'
str_a.indexOf( '1' );           // 0
str_a.lastIndexOf( ' ' );       // 7
```

Almost everything's an object

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

arr.length;          // 5
arr.join( ' ' );     // '1 2 3 4 5'

arr.pop();           // 5
arr;                 // [ 1, 2, 3, 4 ]

arr.push( 6 );       // 5 (the new length)
arr;                 // [ 1, 2, 3, 4, 6 ]

arr.reverse();
arr;                 // [ 6, 4, 3, 2, 1 ]

arr.shift();         // 6
arr.unshift( 5 );    // 5 (the new length)
arr;                 // [ 5, 4, 3, 2, 1 ]
```

The DOM

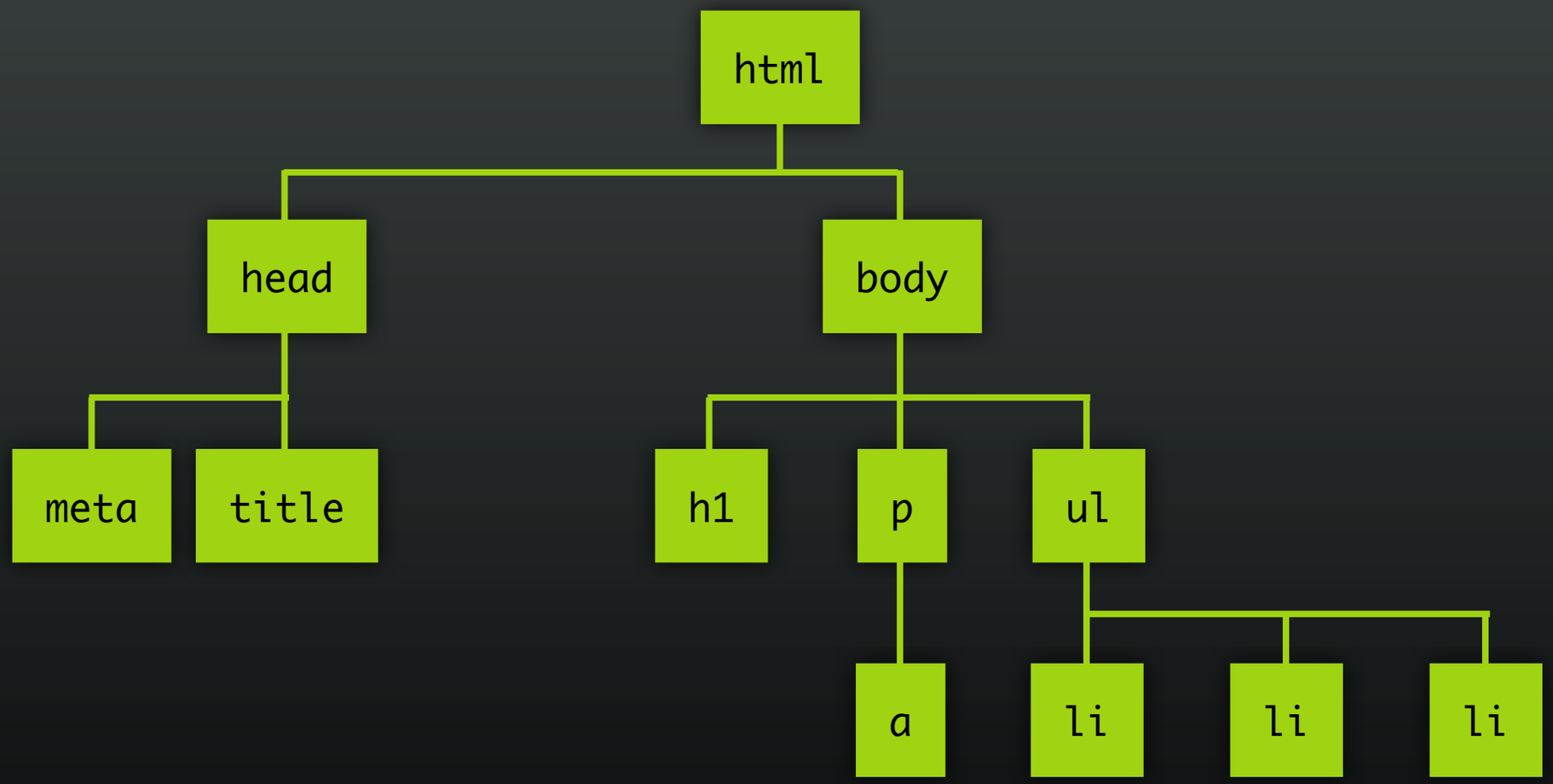
(i.e. your HTML)

HTML

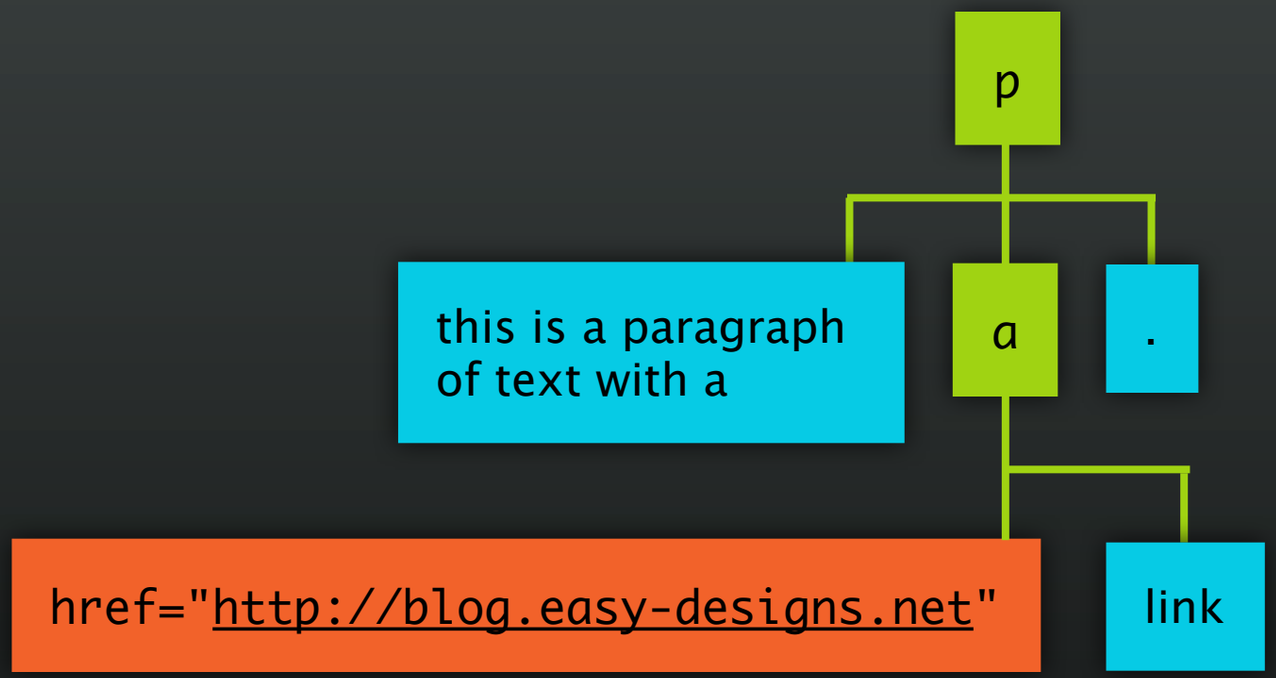
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Page Title</title>
  </head>
  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph with a
      <a href="http://blog.easy-designs.net">link</a>.</p>
    <ul>
      <li>a list item</li>
      <li>another list item</li>
      <li>a third list item</li>
    </ul>
  </body>
</html>
```

HTML

HTML



HTML


p
this is a paragraph of text with a
a
.
href="http://blog.easy-designs.net"
link

Step 1:

Find Stuff

Find Stuff (in CSS)

```
p {
  color: red;
}

#footer {
  border: 1px solid;
}

#footer p {
  color: black;
}
```

Find Stuff (in JS)

```
document.getElementsByTagName( 'p' );



document.getElementById( 'footer' );



document.getElementById( 'footer' )
        .getElementsByTagName( 'p' );
```

Find Stuff (in jQuery)

```
$( 'p' );



$( '#footer' );



$( '#footer p' );
```

Find Stuff (in modern JS)

```
document.querySelector( 'p' );


document.querySelector( '#footer' );


document.querySelector( '#footer p' );
```

Libraries vs. Vanilla JS

Libraries	Vanilla JS
Write less code	Write more code
Don’t worry about browser differences	Deal with browser issues
More abstraction	More explicit
Extra Downloads	Built-in
Slower	Faster

Comparison

Syntax	Operations/second
`document.getElementsByTagName( 'p' )`	8,280,893
`$( 'p' )`	19,449
`document.getElementById( 'foo' )`	12,137,211
`$( '#foo' )`	350,557
`document.querySelector( 'ul.first' )`	350,102
`$( 'ul.first' )`	18,450

Comparison

Syntax		Operations/second
`document.getElementsByTagName( 'p' )`		8,280,893
`$( 'p' )`	← 99.7% slower	19,449
`document.getElementById( 'foo' )`		12,137,211
`$( '#foo' )`	← 97.1% slower	350,557
`document.querySelector( 'ul.first' )`		350,102
`$( 'ul.first' )`	← 95% slower	18,450

Traversing a document

```
var  a = document.getElementsByTagName( 'a' ),
     a_len = a.length,
     i,
     title;

for ( i=0; i < a_len; i++ )
{
  title = a[i].getAttribute( 'title' );
  if ( title )
  {
    console.log( title );
  }
}
```

Traversing a document

```
node.previousSibling; // node
node.nextSibling;     // node
node.parentNode;      // node
node.childNodes;      // node list
node.children;        // element collection
node.firstChild;      // node
node.lastChild;       // node
```

Digging in

```
node.nodeName;      // e.g. "em" or "#text"

node.nodeType;      // 1 = element
                    // 2 = attribute
                    // 3 = text

node.nodeValue;     // only attribute nodes
                    // and text nodes
```

Step 2:

Manipulate Stuff

Manipulate Stuff (in CSS)

```
p {
  color: red;
}

#footer {
  border: 1px solid;
}

#footer > p {
  color: black;
}
```

Manipulate Stuff (in JS)

```
var abbr = document.createElement( 'abbr' );

var text = document.createTextNode( 'TN' );

abbr.setAttribute( 'title', 'Tennessee' );

abbr.appendChild( text );
```

Manipulating the DOM

```
element.appendChild( new_node );

element.insertBefore( new_node, target );

element.replaceChild( new_node, target );
```

Manipulating elements

```
var p = document.getElementsByTagName( 'p' )[0],   // collect

abbr = document.createElement( 'abbr' ),            // generate
text = document.createTextNode( 'TN' );

abbr.setAttribute( 'title', 'Tennessee' );           // combine
abbr.appendChild( text );
p.appendChild( abbr );
```

Cheap creation

```
// find #foo
var p = document.getElementById( '#foo' );

// create the model
var abbr = document.createElement( 'abbr' );

for ( i=0; i<100; i++ )
{
  // append cheap copies to #foo
  p.appendChild( abbr.cloneNode() );
}
```

Cheap creation

```
// create the model
var abbr = document.createElement( 'abbr' ),
    a, b;

// add a child
abbr.appendChild(
  document.createTextNode('hi')
);

// make cheap copies
a = abbr.cloneNode( false );    // <abbr></abbr>
b = abbr.cloneNode( true );     // <abbr>hi</abbr>
```

Bulk manipulation

```
// good for read/write of large chunks
element.innerHTML = new_content;

// avoid in general
document.write( new_content );
```

Exercise 1

HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Example 1</title>
  </head>
  <body>
    <blockquote cite="http://bit.ly/1n9zDlG">
      <p>Progressive Enhancement, as a label for a strategy for
      Web design, was coined by Steven Champeon in a series of
      articles and presentations for Webmonkey and the SxSW
      Interactive conference.</p>
    </blockquote>
  </body>
</html>
```

The plan

1. Find all the `blockquote`s in a document
2. Get the value of the `cite` attribute
3. Create a new `anchor` element node
4. Set the `href` attribute of the anchor to the value of the `cite`
5. Create a new text node with the word “source”
6. Insert the text into the `anchor`
7. Insert the `anchor` into the `blockquote`.

HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Example 1</title>
  </head>
  <body>
    <blockquote cite="http://bit.ly/1n9zDlG">
      <p>Progressive Enhancement, as a label for a strategy for
      Web design, was coined by Steven Champeon in a series of
      articles and presentations for Webmonkey and the SxSW
      Interactive conference.</p>
    </blockquote>
    <script>
      ...
    </script>
  </body>
</html>
```

My take

```
var quotes = document.getElementsByTagName( 'blockquote' );

for ( var i=0; i < quotes.length; i++ )
{
  var source = quotes[i].getAttribute( 'cite' );

  if ( source )
  {
    var link = document.createElement( 'a' );
    link.setAttribute( 'href', source );

    var text = document.createTextNode( 'source' );
    link.appendChild( text );

    quotes[i].appendChild( link );
  }

}
```

Exercise 2

HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Example 2</title>
  </head>
  <body>
    <p>This is a <em>test</em> of a simple email obfuscation
    technique. It relies on an obfuscated email address placed in
    an emphasis element (<code>em</code>) and replaces it with a
    <code>mailto:</code> link for the valid email address.</p>

    <p>For example, this email address&#8212;<em>aaron [at]
    easy [dash] designs [dot] net</em>&#8212; should be
    converted.</p>
  </body>
</html>
```

The plan

1. Find all the `em` elements in a document
2. Make sure the content passes our obfuscation test (e.g. contains "[at]")
3. Grab the content and convert bracketed terms to their equivalents to reveal the email address (e.g. "[at]" to "@")
4. Create a new `anchor`
5. Set the content to be the email address
6. Set the mailto: `href`
7. Replace the `em` with the `anchor`

HTML

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Example 2</title>
  </head>
  <body>
    <p>This is a <em>test</em> of a simple email obfuscation
    technique. It relies on an obfuscated email address placed in
    an emphasis element (<code>em</code>) and replaces it with a
    <code>mailto:</code> link for the valid email address.</p>

    <p>For example, this email address&#8212;<em>aaron [at]
    easy [dash] designs [dot] net</em>&#8212; should be
    converted.</p>
  </body>
</html>
```

My take

```
var ems = document.getElementsByTagName('em'),
    i = ems.length, str, a;

while ( i-- )
{
  if ( ems[i].firstChild &&
       ems[i].firstChild.nodeValue.match( /\s*\[at\]\s*/g ) )
  {
    str = ems[i].firstChild.nodeValue
            .replace( /\s*\[dot\]\s*/g, '.' )
            .replace( /\s*\[at\]\s*/g, '@' )
            .replace( /\s*\[dash\]\s*/g, '-' );

    a = document.createElement( 'a' );
    a.setAttribute( 'href', 'mailto:' + str );
    a.appendChild( document.createTextNode( str ) );
    ems[i].parentNode.replaceChild( a, ems[i] );
  }
}
```