

# Javascript

Many examples from Kyle Simpson: Scope and Closures  
(<https://github.com/getify/You-Dont-Know-JS>)

# What is JavaScript?

- Not related to Java (except that syntax is C/Java- like)
- Created by Brendan Eich at Netscape later standardized through ECMA
- ECMAScript3 in 1999 is the baseline for modern day JS
- Microsoft showed no intention of implementing proper JS in IE stalled progress on ES4
- 2005 work restarted on ECMAScript 4 amid continued controversy

# What is JavaScript?

- 2005 Jesse James Garrett's white paper coined the term Ajax
  - web apps could load data in the background
  - led to the development of JQuery, Prototype, Dojo
- ES6 came out in 2015
- See <http://kangax.github.io/compat-table/es6/> for browser compatibility

# Which version to use?

- Use ECMAScript 6 (also known as ES 6 or ES 2015)
- Don't worry about incompatibilities with old browsers until you have to, and then use transpilers

# JavaScript Engines

- An **engine** is the program that interprets and executes JavaScript code
- V8 - Google's open source engine used in Chrome and Node
- Spidermonkey - Mozilla
- JavaScriptCore (Nitro, Squirrelfish) - Safari, Webkit
- Chakra - MS Edge

# Transpilers

- Transpiler = source-to-source compiler
- Babel - ES6 (ES2015 to ES5)
  - allows you to write ES6 JS without worrying about browser compatibility problems.
- CoffeeScript
- TypeScript
- ClojureScript

# Language highlights

- Many things are as you would expect from your experience with Python, C and Java
- We will focus on features that are not quite what you might expect
- Always “use strict”;

# Scoping Rules

```
a = 3;    // global scope
var b = 3; // global scope
function foo() {
    c = 10 // global
    var d = 5; // function scope (lexical)
    function nested() {
        d = 6; // function (foo) scope
    }
    console.log(d);
}
```



```
"use strict";  
var a = 3; // global scope  
var b = 3; // global scopes  
function foo() {  
    var c = 10 // function scope  
    var d = 5; // function scope  
    function nested() {  
        var d = 6; // function scope  
    }  
    console.log(d);  
}
```

## ES6

```
"use strict";  
var a = 3; // global scope  
var b = 3; // global scope  
function foo() {  
    let c = 10 // block  
    let d = 5; // block scope  
    function nested() {  
        let d = 6; // block scope  
    }  
    console.log(d);  
}
```

*let gives you block scope rather than function scope*

# Functions are first-class objects

```
let sq = function(x) {  
    return x * x;  
}
```

```
function cube(x) {  
    return x * x * x;  
}
```

# Anonymous functions

```
let sq = function(x) {  
    return x * x;  
};
```

```
// If you can give a brief descriptive  
// name you should for readability
```

# IIFE

## Immediately Invoked Function Expressions



```
let a = 2;

function foo() {
    let a = 3;
    console.log(a);
}

foo();

console.log(a);
```

```
let a = 2;

(function foo() {
    let a = 3;
    console.log(a);
})();

console.log(a);
```

# Naming anonymous functions?

```
setTimeout( function(){  
    console.log( "I waited 1 second!" );  
}, 1000);
```

- No function name to display in stack traces
- Can't refer to itself for recursion or unbinding and event handler
- A name helps self-document the code

# Naming anonymous functions?

```
setTimeout( function timeoutHandler(){  
    console.log( "I waited 1 second!" );  
}, 1000);
```

# Closures

- “Closures happen as a result of writing code that relies on lexical scope.” Kyle Simpson
- An unsurprising result of first-class objects and lexical scope..



# Normal code

```
function foo() {  
    var a = 2;  
  
    function bar() {  
        console.log( a ); // 2  
    }  
  
    bar();  
}  
  
foo();
```

# Closure

```
function foo() {  
    var a = 2;  
  
    function bar() {  
        console.log( a ); // 2  
    }  
  
    return bar;  
}  
  
var baz = foo();  
baz();
```

Effectively carries its lexical scope  
with the function reference

# Another example

```
function foo() {  
    var a = 2;  
    function baz() {  
        console.log( a ); // 2  
    }  
    bar( baz );  
}
```

```
function bar(fn) {  
    fn();  
}
```

# For real

```
function wait(message) {  
  
    setTimeout( function timer(){  
        console.log( message );  
    }, 1000 );  
}  
  
wait( "Hello, closure!" );"  
// Implementation of setTimeout has a call  
//to its parameter
```

# Loops

```
for (var i=1; i<=5; i++) {  
    setTimeout( function timer(){  
        console.log( i );  
    }, i*1000 );  
}
```

//Output?

# Loops: solution

```
for (var i=1; i<=5; i++) {  
    (function() {  
        setTimeout( function timer(){  
            console.log( i );  
        }, i*1000 );  
    })();  
}
```

```
// Maybe if we wrap it in a new scope...
```

# Loops: attempt 2

```
for (var i=1; i<=5; i++) {  
    (function() {  
        setTimeout( function timer(){  
            console.log( i );  
        }, i*1000 );  
    })();  
}
```

// Maybe if we wrap it in a new scope...

The problem is there is only one i.

# Loops: solution

```
for (var i=1; i<=5; i++) {  
  (function() {  
    var j = i;  
    setTimeout( function timer(){  
      console.log( j );  
    }, i*1000 );  
  })();  
}
```



# Loops: even better

```
for (var i=1; i<=5; i++) {  
    let j = i;    // block scope!!  
    setTimeout( function timer(){  
        console.log( j );  
    }, i*1000 );  
}
```

# Objects

- An object is a container of properties, where a property has a name and a value
- You can create object literals:  

```
var point = { x: 10, y: 20};  
var point = {"x": 10, "y": 20};
```
- Quotes are optional if the name would be a legal JS name
- object properties retrieved by point.x OR point["x"]

# Methods on Objects

```
function dist_from_orig() {  
    console.log(this.x);  
    return(Math.sqrt(this.x * this.x +  
                      this.y* this.y));  
}
```

```
var p1 = {  
    x: 10,  
    y: -6,  
    dist_from_orig: dist_from_orig  
};  
console.log(p1.dist_from_orig());
```

# Function Objects/ Constructors

```
function dist_from_orig() {  
    console.log(this.x);  
    return(Math.sqrt(this.x * this.x + this.y* this.y));  
}  
  
function Point(x, y) {  
    this.x = x;  
    this.y = y;  
    this.dist = dist_from_orig;  
}  
  
var p3 = new Point(3,2);  
console.log(p3.dist_from_orig);
```

# Adding properties

```
var p3 = new Point(3,2);

p3.is_origin = function is_origin() {
    return this.x == 0 && this.y == 0;
}

p3.z = 33;

if(p3.is_origin()) {
    console.log("origin");
} else {
    console.log("not orgin");
}
```

# this

- Mostly works as you would expect, but is really different than other programming languages.
- It refers to the containing object of the call-site of a function, not where the function is defined.
- Under “use strict” the global object is not eligible for `this` binding.

# Implicit Binding

```
function bar() {  
    console.log(this.a);  
}
```

```
var obj2 = {  
    a: 42,  
    bar: bar  
};
```

```
var obj1 = {  
    a: 2,  
    obj2: obj2  
}
```

```
obj2.bar();
```

```
obj1.obj2.bar();
```

What is the result of these calls? What would you expect?

Try running it!

# Lost binding

```
var p = obj2.bar;
```

```
p(); // undefined
```

This happens because `obj2.bar` is just a reference, it doesn't *belong* to `obj2`.



# Explicit binding

```
bar.call(obj1); // 2
```

Forces this to be obj1

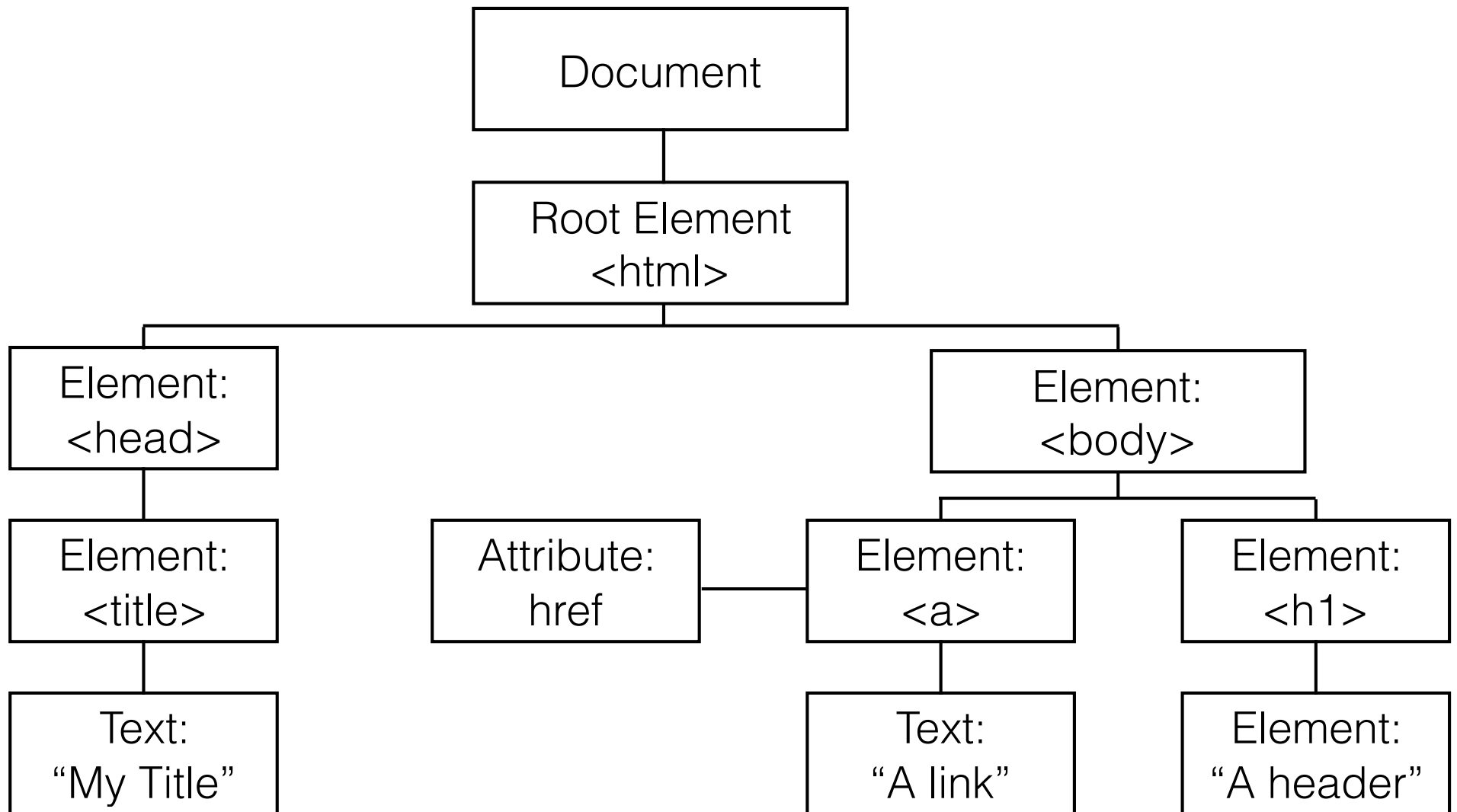
# new binding

```
function Point(x, y) {  
    this.x = x;  
    this.y = y;  
    this.dist = dist_from_orig;  
}
```

```
var p3 = new Point(3,2);
```

# DOM

# DOM - Document Object Model



# BOM: Browser Object Model

Name	Description
document	DOM (current HTML page)
history	List of pages user has visited
location	URL of current page
navigator	Info about the browser
screen	Screen area (viewport) occupied by the browser.
window	The browser window

# window

- Top level object in the Browser Object Model
- Methods include:
  - `alert, confirm, prompt` (popup boxes)
  - `setInterval, setTimeout clearInterval, clearTimeout` (timers)
  - `open, close` (popping up new browser windows)
  - `blur, focus, moveBy, moveTo, print, resizeBy, resizeTo, scrollBy, scrollTo`

# document

Current web page and its elements

- Properties:
  - `anchors`, `body`, `cookie`, `domain`, `forms`,  
`images`, `links`, `referrer`, `title`, `URL`
- Methods:
  - `getElementById`
  - `getElementsByName`
  - `getElementsByTagName`
  - `close`, `open`, `write`, `writeln`

**Location:** URL of the current page

- Properties:
  - `host`, `hostname`, `href`, `pathname`, `port`,  
`protocol`, `search`
- Methods:
  - `assign`, `reload`, `replace`

**Navigator:** Information about the web browser

- Properties:
  - `appName`, `appVersion`, `browserLanguage`,  
`cookieEnabled`, `platform`, `userAgent`



- Screen : information about the display screen
  - Properties: `availHeight`, `availWidth`, `colorDepth`, `height`, `pixelDepth`, `width`
- Difference?
  - *Screen* is what's visible to you
  - *Window* is the browser window including scrollbars, navigation, bookmark bars, etc
  - *Document* can be larger than the screen/window

- The history object keeps a list of sites that the browser has visited in this window.
- Properties: `length`
- Methods: `back`, `forward`, `go`
- Sometimes the browser won't let scripts view history properties, for security reasons.

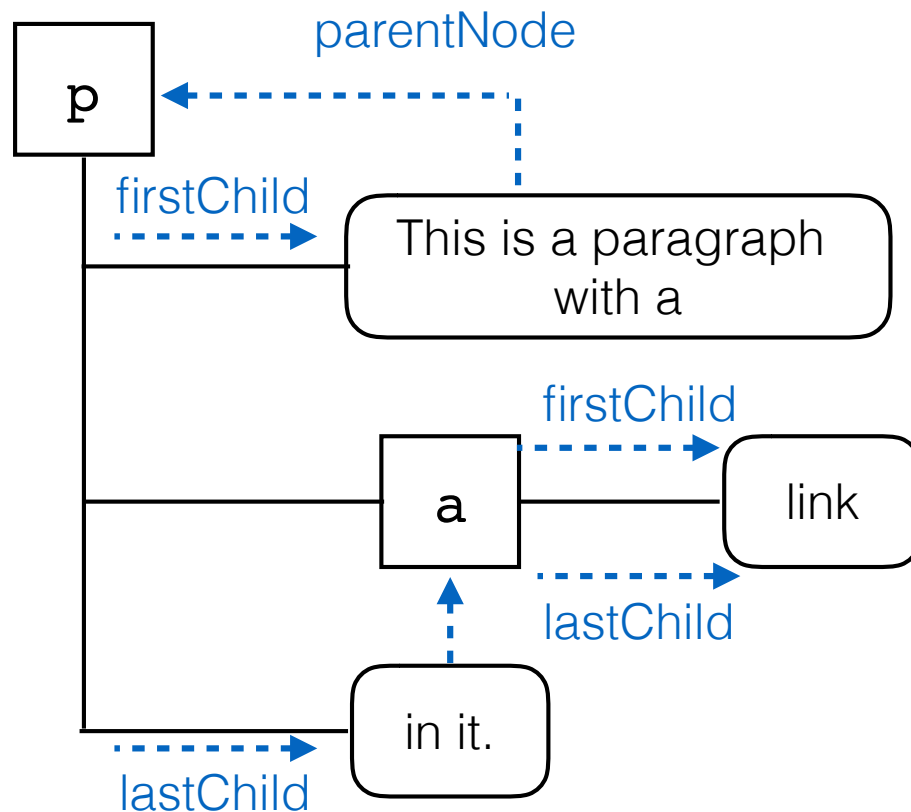
Name	Description
<code>firstChild</code> <code>lastChild</code>	Start/end of this node's list of children.
<code>childNodes</code>	Array of all this node's children
<code>nextSibling</code> <code>previousSibling</code>	Neighbouring nodes with the same parent.
<code>parentNode</code>	Node that contains this node

<p>

This is a paragraph with a

<a href="/path/page.html">link</a> in it.s

</p>



Name	Description
<code>document. createElement("tag")</code>	Create and return a new empty DOM nodes
<code>document. createTextNode("text")</code>	Create and return a text node with the given text
<code>appendChild(node)</code>	place node at end of this node's children
<code>insertBefore(new, old)</code>	place given new node just before old child
<code>removeChild(node)</code>	remove node from this child list
<code>replaceChild(new, old)</code>	replace node