Andrew & Aaron present:

The Future of JavaScript

JavaScript 1.6

- Based on ECMA-262, edition 3
- Implemented in Firefox 1.5
- Added
  - ECMAScript for XML (aka E4X; ECMA-357)
  - Array extras
  - Array and String generics

Array extras

- Location methods:
  - `indexOf()`
  - `lastIndexOf()`
- Iterative methods:
  - `every()`
  - `filter()`
  - `forEach()`
  - `map()`
  - `some()`

Iteration examples

```
var ids = [ 1, 2, 3 ];
var els = ids.map( function( i ){
      return document.getElementById( 'item_' + i ); } );
for( var i = 0; i < els.length; i++ ){
  els[i].style.border = '1px solid';
}
```

and

```
var lis = document.getElementsByTagName( 'li' );
var evenLis = Array.filter( lis,
                function( li, i ){ return i % 2 == 1; } );
for( var i = 0; i < evenLis.length; i++ ){
  evenLis[i].style.background = '#ccc';
}
```

JavaScript 1.7

- Based on ECMA-262, edition 3
- Includes JS1.6 enhancements
- Introduced
  - Generators & Iterators
  - Array comprehensions
  - Block scope variables
  - Destructuring assignment

Using JS 1.7

- Implemented in Firefox 2 only
- Enable via MIME type:

```
<script type="application/javascript;version=1.7">
  // code goes here
</script>
```

or

```
<script type="application/javascript;version=1.7" »
        src="/path/to/my.js"></script>
```

Traditional iterative generation

```
for( var i = 1; i < 4; i++ ){
  var ul = document.createElement( 'ul' );
  for( var j = 1; j <= i; j++ ){
    var li = document.createElement( 'li' );
    li.appendChild( document.createTextNode( j ) );
    ul.appendChild( li );
  }
  document.getElementsByTagName( 'body' »
                                 )[0].appendChild( ul );
}
```

Which gives us

- 1

- 1
- 2

- 1
- 2
- 3

Optimized iterative generation

```
var $ul = document.createElement( 'ul' );
var $li = document.createElement( 'li' );
for( var i = 1; i < 4; i++ ){
  var ul = $ul.cloneNode( true );
  for( var j = 1; j <= i; j++ ){
    var li = $li.cloneNode( true );
    li.appendChild( document.createTextNode( j ) );
    ul.appendChild( li );
  }
  document.getElementsByTagName( 'body' »
                                 )[0].appendChild( ul );
}
```

Generator-Iterators

```
function $el( tag ){
  var el = document.createElement( tag );
  while( true ){
    yield el.cloneNode( true );
  }
}
var $ul = $el( 'ul' ),  $li = $el( 'li' );
for( var i = 1; i < 4; i++ ){
  var ul = $ul.next();
  for( var j = 1; j <= i; j++ ){
    var li = $li.next();
    li.appendChild( document.createTextNode( j ) );
    ul.appendChild( li );
  }
  document.getElementsByTagName( 'body' »
                                 )[0].appendChild( ul );
}
```

Straight Iteration

```
var me = { name: 'Aaron Gustafson', age:  29,
           'eye color': 'blue', height: '5ft 11in' };
var it = Iterator( me );
var ul = $ul.next(), li;
try{
  while( true ){
    li = $li.next();
    li.appendChild( document.createTextNode( »
                                        it.next() ) );
    ul.appendChild( li );
  }
}catch( err if err instanceof StopIteration ){
  document.getElementsByTagName( 'body' »
                                      )[0].appendChild( ul );
}catch( err ){
  alert( 'error: '+err.description );
}
```

Which gives us

- name,Aaron Gustafson
- age,29
- eye color,blue
- height,5ft 11in

Alternation

```
var li = document.getElementsByTagName( 'li' );
for( var i = 0; i < li.length; i++ ){
  if( i % 2 == 0 ){
    li[i].className = 'even';
  }
}
```

Generators and array creation

```
function range( start, end ){
  for( var i = start; i < end; i++ ){
    yield i;
  }
}

var li = document.getElementsByTagName( 'li' );

var evens = [i for ( i in range( 0, li.length ) ) »
               if ( i % 2 ==0 ) ];

for( num in evens ) li[evens[num]].className = 'even';
```

let is the new black

- Only way we currently get block scope:

```
function foo(){
  var x = 5;
  ( function(){
      var x = 10;
      alert( x );    //-> 10
    } )();
  alert( x );        //-> 5
}
```

let blocks

```
function foo(){
  var x = 5;
  let( x = 10 ){
    alert( x );          //-> 10
  }
  alert( x );            //-> 5
}
```

- or if you wanna get really crazy

```
function foo(){
  var x = 5;
  let( x = x + 5 ){
    alert( x );          //-> 10
  }
  alert( x );            //-> 5
}
```

let expressions

```
function foo() {
  var x = 5;
  alert( let( x = 10 ) x ); //-> 10
  alert( x );               //-> 5
}
```

let in loops

```
for( let i = 0; i < array.length; i++ )
  doSomethingWith( array[i] );

alert( i ); //-> undefined
```

or

```
for( let i = 0, subArray; i<array.length; i++ ){
  subArray = array[i];
  for( let i = 0; i < subArray.length; i++ )
    alert( subArray[i] );
}
```

Like a key party in your code

- Destructuring assignment:

```
var a = 1;
var b = 2;
[a, b]= [b, a];
```

or

```
var [c, d] = [a, b];
```

Return with greater flexibility

- We've always been able to return arrays

```
var result = returnsArray();
var a = result[0];
var b = result[1];
```

- But now

```
var [a,b] = returnsArray();
```

or even

```
var [,b] = returnsArray();
```

That's not all, let's play with JSON

```
var me = { name:          'Aaron Gustafson',
           age:           29,
           'eye color': 'blue',
           height:        '5ft 11in' };

var ul = $ul.next();

for( let [ key, value ] in me ){
  let li = $li.next();
  li.appendChild( document.createTextNode( 'Key: ' +
                        key + ', Value: ' + value ) );
  ul.appendChild( li );
}

document.getElementsByTagName( 'body' »
                                )[0].appendChild( ul );
```

Resulting in

- Key: name, Value: Aaron Gustafson
- Key: age, Value: 29
- Key: eye color, Value: blue
- Key: height, Value: 5ft 11in

Or iterate *safely* over an Object

```
Object.prototype.HAHAHA = "I AM THE HASH DESTRUCTOR";

for( let [ key, value ] in me )
  alert( key ); /* 'name', 'age', 'eye color',
                   'height', 'HAHAHA' */

function SafeHashIterator( hash, keysOnly ){
  for( let [key, value] in hash ){
    if( !hash.hasOwnProperty( key ) ) continue;
    yield keysOnly ? key : [key, value];
  }
  throw StopIteration;
}

for( let [key, value] in SafeHashIterator( me ) )
  alert( key ); // 'name', 'age', 'eye color', 'height'
```

So why should I care?