Javascript, λ-Calculus, and Functional Programming

History

Created by Brendan Eich.

Came from the need for websites to display dynamic content.

History

“Borrowed” some Java syntax (curly braces, semicolons)

First called Mocha, then Livescript, and finally Javascript, after receiving a trademark license from Sun.

History

Functions are first class citizens.

Supports Anonymous/Lambda functions.

Allows nested functions with function scope.

History

```
function init() {
    var name = "Mozilla"; // name is a local variable created by init
    function displayName() { // displayName is the inner function, a closure
        alert (name);//displayName uses variable declared in the parent function
    }
    displayName();
}
```

Lazy Evaluation

Call by need/ Don’t evaluate anything unless you need to.

Can improve performance greatly, but can also ruin performance if you do it wrong.

Lazy Evaluation

Javascript has no lazy implementation as a default language feature.

Doesn’t fit the mutable, non-pure style of programming in Javascript.

Lazy Evaluation

Possible to get some expressiveness of lazy evaluation by using some stream libraries, such as lazy.js or stream.js.

Higher Order Functions

Functions of functions!

Functions where the Input, Output, or Both are functions.

Higher Order Functions

```
function compose(f, g) {
    function ret(x) { return f(g(x)); }
    return ret;
}

function foo(x){ return 1 + x; }
function bar(x){ return x * 2; }

foobar = compose(foo, bar);
foobar(10);    // returns 21    (1 *(2 * 10))
```

Closures

Can step up the static chain to find variables.

If no matching variable is found at the global scope, one is created.

Closures

```
function init() {
    var name = "Mozilla"; // name is a local variable created by init
    function displayName() { // displayName is the inner function, a closure
        alert (name);//displayName uses variable declared in the parent function
    }
    displayName();
}
```

List Comprehension

Actually… Array Comprehension

Objects in Javascript are Associative Arrays

Really just syntactic sugar for loops

List Comprehension

```javascript
var numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9];
var tripledEven = [i * 3 for (i of numbers) if (i % 2 === 0)]; //[6,12,18,24]
```

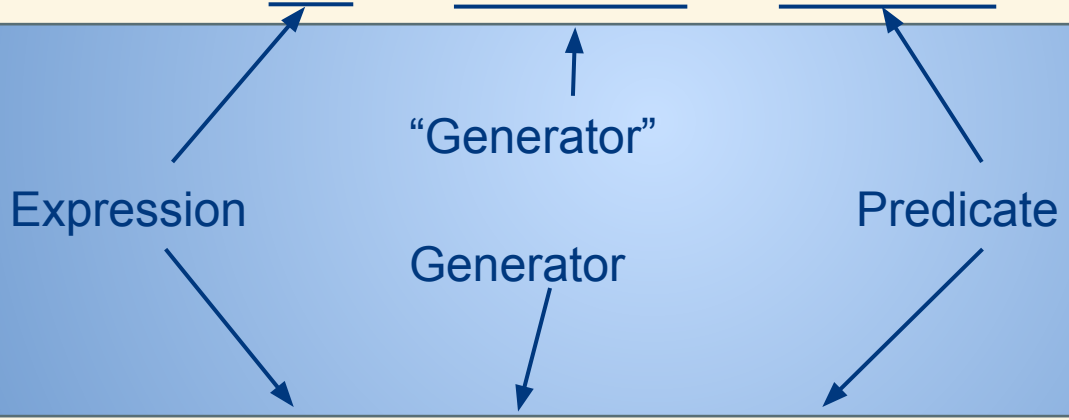
Javascript

Haskell

```haskell
let tripledEven = [x * 3 | x <- [1..], x `mod` 2 == 0]
take 4 tripledEven --[6,12,18,24]
```

List Comprehension

```
var numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9];
var tripledEven = [i * 3 for (i of numbers) if (i % 2 === 0)]; //[6,12,18,24]
```




```
let tripledEven = [x * 3 | x <- [1..], x `mod` 2 == 0]
take 4 tripledEven --[6,12,18,24]
```

Questions?