

**mtl.rb**  
**RUBY ENTHUSIASTS**

# Partial applications

# Sean Braithwaite

- Doesn't programme in lisp
- Doesn't have many github watchers
- Prefers whiskey to water

# Functional

- Is a sentence structure
- Is thoughtful
- Will make you a better programmer

# Closures

- The basis of many functional constructs
- You probably understand them already

# Closures

```
var addition = function(a) {  
    // i have a  
    return function(b) {  
        // I have access to a and b  
        return a + b;  
    }  
}
```

# Global

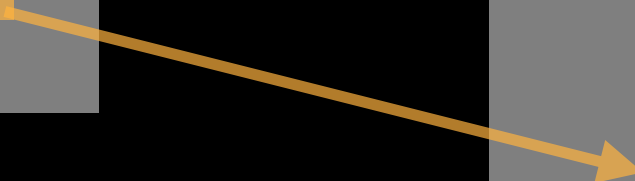
Local

More Local

# Technically

“A closure is a stack-frame which is *not deallocated* when the function returns.”

# Global



# Closures :D

```
var addition = function(a) {  
    // i have a  
    return function(b) {  
        // I have access to a and b  
        return a + b;  
    }  
}
```

```
addition(2)(2)  $\#=>$  2 + 2 = 4
```

# Partial applications

```
csv("foo, bar, gaz") // => ['foo', 'bar', 'baz']
```

```
function cvs(text){  
    return text.split(/,\s*/);  
}
```

---

VS

---

```
function csv(text){  
    return String.prototype.split.apply(text, /,\s*/)  
}
```

# Repetition

```
function csv(text){  
    return String.prototype.split.apply(text, /,\s*/)  
}
```

```
csv("foo, bar, gaz") // => ['foo', 'bar', 'baz']
```

```
function psv(text){  
    return String.prototype.split.apply(text, /\|\s*/)  
}
```

```
psv("foo | bar | baz") // => ['foo', 'bar', 'baz']
```

```
function lsv(text){  
    return String.prototype.split.apply(text, /lol\s*/)  
}
```

```
psv("foo lol bar lol baz") // => ['foo', 'bar', 'baz']
```

# Extending functionality

- Inheritance via arguments
- Stateless
- Used so often the mechanics can be abstracted...

Curry is delicious!



# Abstraction

```
Function.prototype.curry = function() {  
  var fn = this  
  var old_args = Array.prototype.slice.call(arguments);  
  return function() {  
    var new_args = old_args.concat(Array.prototype.slice.call  
(arguments));  
    return fn.apply(this, new_args);  
  };  
};
```

# Better

```
var csv    = String.prototype.split.curry(/,\s*/);  
var psv   = String.prototype.split.curry(/\|\s*/);  
var lolsv = String.prototype.split.curry(/lol\s*/);
```

```
function cvs(text){  
  return text.split(/,\s*/);  
}
```

---

VS

---

```
function csv(text){  
  return String.prototype.split.apply(text, /\s*/)  
}
```

---

VS

---

```
var csv = String.prototype.split.curry(/,\s*/);
```

# Extending functionality

- Use what you have
- Inheritance via argument *transformation*
- Maintain state by passing *transformed args*

# Chaining

```
var add = function(a,b){ return a + b};  
add4 = add.curry(4);
```

```
assert(4 + 4      == 8);  
assert(add4(4)   == 8);  
assert(add(4)(4) == 8);
```

# Whats next?

- Domain specific languages
- Compilers

```
with ( require( "fab" ) )
( fab )
  ( listen, 0xFAB )
  ( /^\/hello/ )
  ( tmpl )
    ( "Hello, <%= this %>!" )
  ( /^\/(\w+)$/ )
    ( capture.at, 0 )
    ( "world" )
  ( 404 );
```

...lolwhat?

```
var example = (reindr)('ul')
                ('li')("ichi")('/li')
                ('li')("ni")('/li')
                ('li')("san")('/li')
                ('/ul')
            (reindr)

// renders: <ul><li>ichi</li><li>ni</li><li>san</li></ul>
```

# Conclusion

- Abstract mechanics
- Build data structures with functions
- Build processing structures with function

Questions?