software
studio

prototypes

Daniel Jackson

For now, all rights reserved. Daniel Jackson, 2013.
> yellow = {red: 255, green: 255, blue: 0};
Object
1.blue: 0
2.green: 255
3.red: 255
4.__proto__: Object
...
1.hasOwnProperty: function hasOwnProperty() { [native code] }
...
2.toString: function toString() { [native code] }
3.valueOf: function valueOf() { [native code] }

> yellow.hasOwnProperty("red")
true
> yellow.hasOwnProperty("reddish")
false
object model for prototypes

› _proto is not directly accessible!
get/set along prototype chain

- **get**: up chain until match
- **set**: always immediate object
- shadowing if names match

```javascript
> green.red
0
> green.bits
24
```
how to attach a prototype

- set prototype property of constructor (or modify it)
- calls to constructor then yield object with that prototype
setting the prototype

```javascript
var Color = function (r, g, b) {
    this.red = r; this.green = g; this.blue = b;
}
Color.prototype = {bits: 24};
green = new Color(0, 255, 0);

> green.red
0
> green.bits
24
```
modifying the prototype

```javascript
var Color = function (r, g, b) {
  this.red = r; this.green = g; this.blue = b;
}
Color.prototype.toCSS = function () {
  return "rgb(" + this.red + "," + this.green + "," + this.blue + ")";
}
green = new Color(0, 255, 0);
document.body.style.backgroundColor = green.toCSS();
```

how is this bound in call to method?
→ it’s dynamic: inside m in call e.m(), bound to value of e
modifying vs setting prototype

how do these differ?

```javascript
Color.prototype.bits = 24;
Color.prototype = {bits: 24};
```

just need to track sharing between object and constructor; watch this:

```javascript
> var Color = function (r, g, b) {
>     this.red = r; this.green = g; this.blue = b;
> }
undefined
> red = new Color (255, 0, 0)
Color
> Color.prototype = {bits: 24}
Object
> green = new Color (0, 255, 0)
Color
> Color.prototype.space = "RGB"
"RGB"
> red.bits
undefined
> green.bits
24
> red.space
undefined
> green.space
"RGB"
```
extending built-ins

```javascript
Array.prototype.map = function (f) {
    var result = [];
    this.each (function (e) {
        result.push(f(e));
    });
    return result;
}

> [1,2,3].map(function (x) {return x * x;});
[1, 4, 9]
```