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