JavaScript: Objects, Methods, Prototypes

Lecture 22
What is an Object?

- **Property**: a key/value pair (aka "name"/value)
- **Object**: a partial map of properties
  - Keys must be unique
- Creating an object, literal notation
  ```javascript
  ```
- To access/modify an object's properties:
  ```javascript
  myCar.make = "Ford"; //cf. Ruby
  myCar["year"] = 2006;
  var str = "ate";
  myCar["pl"+str] = "NKR463";
  ```
Arrays vs Associative Arrays

- Arrays are accessed by index:
  - [0] = 4
  - [1] = "hi"
  - [2] = 3.14
  - [3] = true

- Associative Arrays are accessed by key:
  - age = 4
  - greeting = "hi"
  - doors = true
  - pi = 3.14

- Accessing elements by index vs key:
  - Arrays: Age[0] = 4
  - Associative Arrays: doors[true] = doors[false]
Dynamic Size, Just Like Arrays

- Objects can grow
  
  ```javascript
  myCar.state = "OH"; // 4 properties
  var myBus = {}; myBus.driver = true; // adds a prop
  myBus.windows = [2, 2, 2, 2];
  ```

- Objects can shrink
  
  ```javascript
  delete myCar.plate;
  // myCar is now {make: "Ford",
  // year: 2006, state: "OH"}
  ```
Testing Presence of Key

- Boolean operator: `in`
  
  `propertyName in object`

- Evaluates to true iff object has the indicated property key
  
  "make" in myCar //=>true
  
  "speedometer" in myCar //=>false
  
  "OH" in myCar //=>false
Iterating Over Properties

- Iterate using `for...in` syntax
  ```javascript
  for (property in object) {
    ...object[property]...
  }
  ```

- Notice `[]` to access each property
  ```javascript
  for (p in myCar) {
    document.write(p +": " + myCar[p]);
  }
  ```
Methods

- The value of a property can be:
  - A primitive (boolean, number, string, null...)
  - An object, an array, or a function

```javascript
var temp = function(sound) {
  play(sound);
  return 0;
}
myCar.honk = temp;
```

- More succinctly:

```javascript
myCar.honk = function(sound) {
  play(sound);
  return 0;
}
```
Example: Method

```javascript
var myCar = {
    make: "Acura",
    year: 1996,
    plate: "NKR462",
    honk: function(sound) {
        play(sound);
        return 0;
    }
};
```
Object Properties

myCar

- make: "Acura"
- year: 1996
- plate: "NKR462"
- honk()

```c
play(sound);
return 0;
```
Keyword "this" in Functions

- Recall distinguished parameter
  \[ x.f(y,z); //x is the distinguished param. \]
- Inside a function, keyword "this"
  ```javascript
  function report() {
    return this.plate + this.year;
  }
  ```
- At run-time, "this" is set to the distinguished parameter of invocation
  ```javascript
  myCar = {plate: "NKR462", year: 1996};
  yourCar = {plate: 340, year: 2013};
  myCar.register = report;
  yourCar.info = report;
  myCar.register(); //=>"NKR4621996"
  yourCar.info(); //=>2353
  ```
Object Properties

```
myCar

plate: "NKR462"
year: 1996

register()
report()

return this.plate + this.year;
```

```
yourCar

plate: 340
year: 2013

info()
```
 Constructors

- *Any* function can be a constructor
- When calling a function with "new":
  1. Make a brand new (empty) object
  2. Call the function, with the new object as the distinguished parameter
  3. Implicitly return the new object to caller
- A "constructor" often adds properties to the new object simply by assigning them
  ```javascript
  function Dog(name) {
      this.name = name;  //adds 1 property
      //no explicit return
  }
  var furBall = new Dog("Rex");
  ```
- Naming convention: Functions intended to be constructors are capitalized
Example

```javascript
function Circle (x, y, radius) {
    this.centerX = x;
    this.centerY = y;
    this.radius = radius;
    this.area = function() {
        return Math.PI * this.radius * 
        this.radius;
    }
}

var c = new Circle(10, 12, 2.45);
```
Creating a Circle Object

```javascript
var c = new Circle(10, 12, 2.45);
```

```
Circle()

this.centerX = x;
this.centerY = y;
... Etc ...
```
Creating a Circle Object

```javascript
var c = new Circle(10, 12, 2.45);
```

```javascript
Circle()

this.centerX = x;
this.centerY = y;
... Etc ...
```
Creating a Circle Object

```javascript
var c = new Circle(10, 12, 2.45);
```

```javascript
Circle()
```
Creating a Circle Object

```
var c = new Circle(10, 12, 2.45);
```

**Circle()**

- centerX = x;
- centerY = y;
  
  ... Etc ...

- area()

  ```
  return Math.PI * this.radius * this.radius
  ```
Creating a Circle Object

```javascript
var c = new Circle(10, 12, 2.45);
```

```javascript
Circle()
```

```javascript
this.centerX = x;
this.centerY = y;
... Etc ...
```

```javascript
return Math.PI * this.radius * this.radius
```

c

centerX

10

centerY

12

radius

2.45

area()
Prototypes

- Every object has a *prototype*
  - A hidden, indirect property ([[Prototype]])
- What is a prototype?
  - Just another object! Like any other!
- When accessing a property (*i.e. obj.p*)
  - First look for `p` in `obj`
  - If not found, look for `p` in `obj`'s prototype
  - If not found, look for `p` in that object's prototype!
  - And so on, until reaching the basic system object
Prototype Chaining

- greeting: "hi"
- doors
- pi: 3.14
- age: 4
- toString()
- hasOwnProperty()
- push()
- pop()
- etc...

Variables:
- greeting
- doors
- pi
- age
- true
- false
- push()
- pop()
- etc...
Class-Based Inheritance

interfaces

extends

implements

classes

static

extends

static

objects

static

instantiates
Example

- Consider two objects
  ```javascript
  var dog = {name: "Rex", age: 3};
  var pet = {color: "blue"};
  ```
- Assume `pet` is `dog`'s prototype
  ```javascript
  //dog.name is "Rex"
  //dog.color is "blue" (follow chain)
  pet.color = "brown";
  //dog.color is "brown" (prop changed)
  dog.color = "green";
  //pet.color is still "brown" (hiding)
  ```
Delegation to Prototype

dog "Rex" age 3 color "green"

pet color "brown"
Prototypes Are Dynamic Too

- Prototypes can add/remove properties
- Changes are felt by all children

```javascript
//dog is {name: "Rex", age: 3}
//dog.mood & pet.mood are undefined
pet.mood = "happy"; //add to pet
//dog.mood is now "happy" too
pet.bark = function() {
    return this.name + " is " + this.mood;
}
dog.bark(); //=>"Rex is happy"
pet.bark(); //=>"undefined is happy"
```
Delegation to Prototype

dog
  name: "Rex"
  age: 3

dog.bark();

pet
  color: "brown"
  mood: "happy"

pet.bark();

return this.name + " is " + this.mood;
Connecting Objects & Prototypes

- How does an object get a prototype?
  ```javascript
  var c = new Circle();
  ```
- Answer
  1. Every function has a prototype `property`
     - Do not confuse with hidden `[[Prototype]]`!
  2. Object's prototype `link`—`[[Prototype]]`—is set to the function's prototype `property`
- When a function `Foo` is used as a constructor, `i.e. new Foo()` , the value of `Foo`'s prototype property is the prototype object of the created object
Prototypes And Constructors

Prototype:

```
Circle()
```

Constructor:

```
this.centerX = x;
this.centerY = y;
... Etc ...
```
Idiom: Methods in Prototype

```
function Dog(n, a) {
    this.name = n;
    this.age = a;
}

var Canine = {
    bark: function (sound) {
        return this.name + " says " + sound;
    }
};

Dog.prototype = Canine;
```
function Dog(n, a) {
    this.name = n;
    this.age = a;
}

var Canine = {
    bark: function (sound) {
        return this.name + "says" + sound;
    }
};

Dog.prototype = Canine;
Idiom: Methods in Prototype

function Dog(n, a) {
    this.name = n;
    this.age = a;
}

Dog.prototype = {
    bark: function (sound) {
        return this.name + "says" + sound;
    }
};

//sets prototype to new anonymous object
function Dog(n, a) {
    this.name = n;
    this.age = a;
}

Dog.prototype.bark = function (sound) {
    return this.name + " says " + sound;
};

//better: extends existing prototype
function Animal() { ... }; function Dog() { ... }; Dog.prototype = new Animal(); //create prototype for future dogs Dog.prototype.constructor = Dog; //set prototype's constructor properly // (ie should point to Dog() )
Setting up Prototype Chains

```java
new Dog()
name "Rex"

new Animal()
constructor

Dog()
prototype

Animal()
prototype
```
Summary

☐ Objects as associative arrays
  ■ Partial maps from keys to values
  ■ Can dynamically add/remove properties
  ■ Can iterate over properties

☐ Method = function-valued property
  ■ Keyword this for distinguished parameter

☐ Constructor = any function

☐ Prototypes are "parent" objects
  ■ Delegation up the chain of prototypes
  ■ Prototype is determined by constructor
  ■ Prototypes can be modified