JavaScript Inheritance

"Code with Passion!"

Topics

1. Inheritance through Prototype
2. Function constructor and Prototype

Inheritance through Prototype

Prototype-based Languages

- Two different ways for supporting inheritance in programming languages
  - Scheme #1: Through traditional class hierarchy (Java, C, C++, ...)
  - Scheme #2: Through prototype (JavaScript)
- JavaScript is a prototype-based language
- In a prototype-based language, there is no concept of a "class"
  - Inheritance is provided through prototype, however
- In prototype-based language, every object has "prototype" property (the actual name of the "prototype" property is *__proto__*) that points to "prototype" object
  - Since the prototype object itself is a JavaScript object, it has its own "prototype" property, which in turn forms "prototype chain", until it reaches "Object" object, which has *null* as its prototype

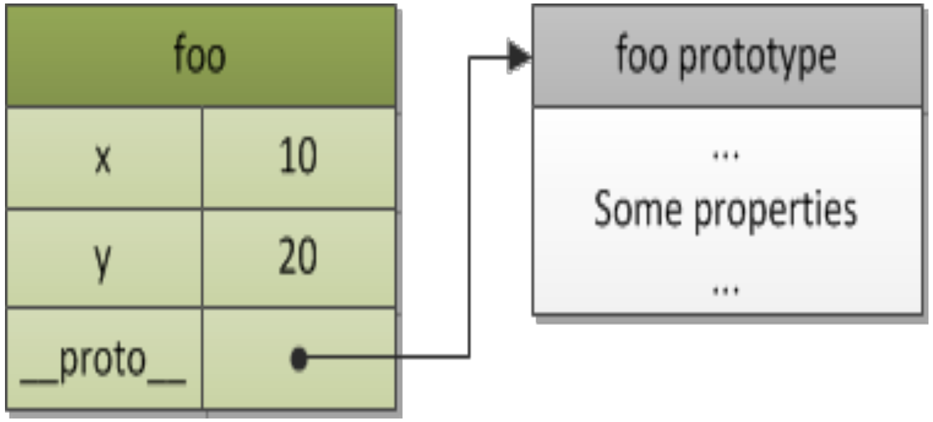
How Does Prototype Provide Inheritance?

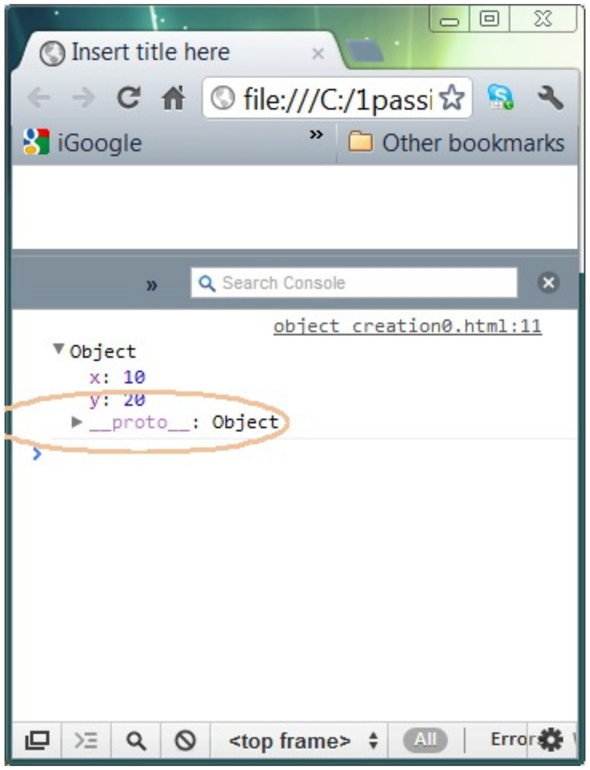
- If an object itself doesn't have a property that is requested, then the prototype objects in the “prototype chain” are searched for that property
- Let's say object “myObject” has a property “a”
  - “*myObject.a*” will access the property “a” as expected
- Let's say object “myObject” does NOT have property “b”
  - “*myObject.b*” will trigger the search of “b” in the prototype chain of the “*myObject*”
  - So, if one of the prototype objects in the prototype chain has the property “b”, then “*myObject.b*” returns the value of it
- So, prototype objects in the prototype chain can capture shared properties, thus providing inheritance

The prototype property is "__proto__"

- (As was said) Every JavaScript object has a prototype property – and it is represented by *__proto__* property

```
var foo = {
  x: 10,
  y: 20
};
```



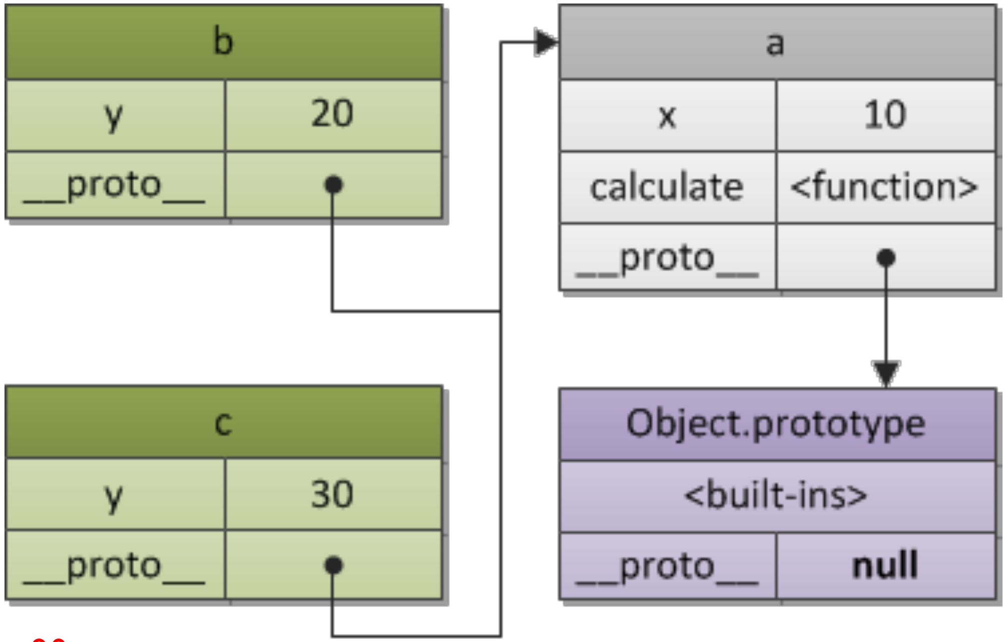


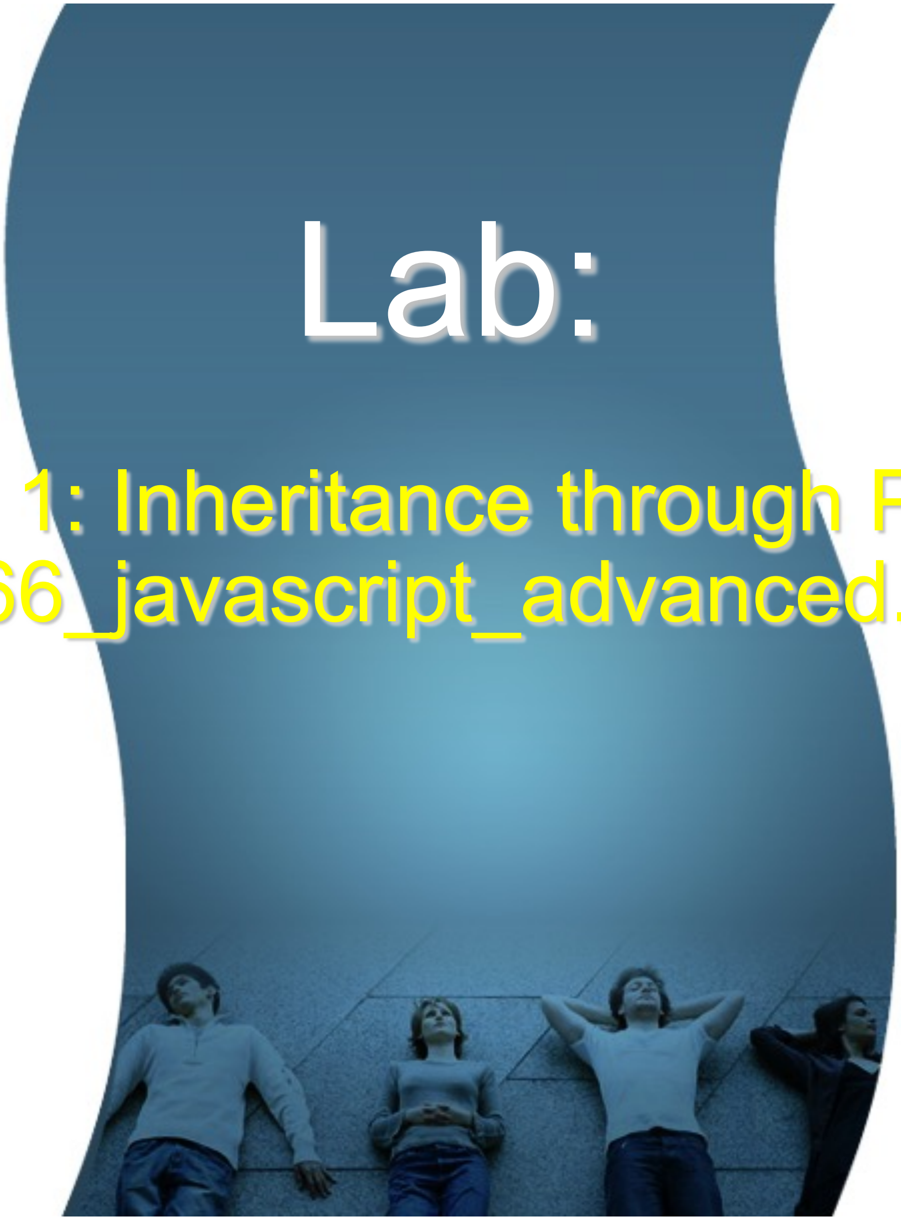
```
object_creation0.html:11
▼Object
   x: 10
   y: 20
  ▶ __proto__: Object
```

Inheritance through Prototype

- As an experimentation, let's set prototype property (__proto__ property) of object “b” and “c” to “a” manually – making “a” parent object of “b” and “c”

```
// Consider “a” parent object of “b” and “c”
var a = {
  x: 10,
  calculate: function (z) {
    return this.x + this.y + z
  }
};

var b = {
  y: 20,
  __proto__: a
};

var c = {
  y: 30,
};
c.__proto__ = a;

// call the inherited method
console.log(b.calculate(30)); // 60 = 10 +20 +30
console.log(c.calculate(40)); // 80 = 10 +30 +40
```





Lab:

Exercise 1: Inheritance through Prototype
4266_javascript_advanced.zip

JavaScript Objects;
3 Different Ways of Creating JavaScript Objects

3 Ways of Creating Your Own JavaScript Objects

1. Create an object instance as Hash Literal (You have already seen this) – preferred
2. Define a function as a Constructor first and then create an instance of an object from it
3. Create a direct instance of an object by using built-in constructor of the built-in "*Object*" object

Option #1: Creating JavaScript Object as a Hash Literal

```
    // Create JavaScript object as a Hash Literal then assign to “personObj”
var personObj = {
        firstname: "John",
        lastname: "Doe",
        age: 50,
        tellYourage: function () {
             alert(“The age is ” + this.age );
        }
        tellSomething: function(something) {
             alert(something);
        }
 }

// Call methods of “personObj” JavaScript object
personObj.tellYourage();
personObj.tellSomething(“Life is good!”);
```

Option #2: Create from a Constructor Function (Template)

- A function defines the structure of a JavaScript object – it plays a role of a template

```
// Define a Constructor function
function Person(firstname,lastname,age,eyecolor){
    this.firstname=firstname;
    this.lastname=lastname;
    this.age=age;
    this.tellYourage=function(){
            alert(“This age is ” + this.age);
    }
}

// Continued in the next slide
```

Option #2: Create from a Constructor Function (Continued)

- Once you have a Constructor function (as you saw in the previous slide), you can create new instances of JavaScript object using *new* keyword

```
myFather=new Person("John","Doe",50,"blue");
myMother=new Person("Sally","Rally",48,"green");
```

- You can then add new properties and functions to new objects

```
myFather.newField = “some data”;
myFather.myfunction = function() {
    alert(this["fullName"] + ” is ” + this.age);
}
```

Option #3: Create a Direct Instance of a JavaScript Object from “Object” object

- By invoking the built-in constructor for the *Object* object

```
// Initially empty with no properties or methods
personObj=new Object();  // same as personObj = { }
```

- Add properties to it

```
personObj.firstname="John";
personObj.age=50;
```

- Add an anonymous function to the *personObj*

```
personObj.tellYourage=function(){
    alert(“This age is ” + this.age);
}
// You can call then tellYourage function as following
personObj.tellYourage();
```

Option #3: Create a Direct Instance of a JavaScript Object from “Object” (Continued)

- Add a pre-defined function

```
function tellYourage(){
    alert(“The age is” + this.age);
}
personObj.tellYourage=tellYourage;
```

- By the way, note that the following two lines of code are doing completely different things

```
// Set property with a function
personObj.tellYourage=tellYourage;
// Set property with returned value of the function
personObj.tellYourage=tellYourage();
```

Lab:

Exercise 6: Create objects
4262_javascript_basics.zip

Constructor Function & Prototype

Object Created through Constructor Function

- In our previous examples, we construct inheritance relationship between object instances
  - In other words, it is not general enough

- What we want in general, however, is to construct inheritance relationship for a group of objects
  - In Java, all objects of a child class inherit properties and methods of a parent class
  - In JavaScript, we want all objects created from a Constructor function to inherit all properties of the Constructor function

Object Created through Constructor Function

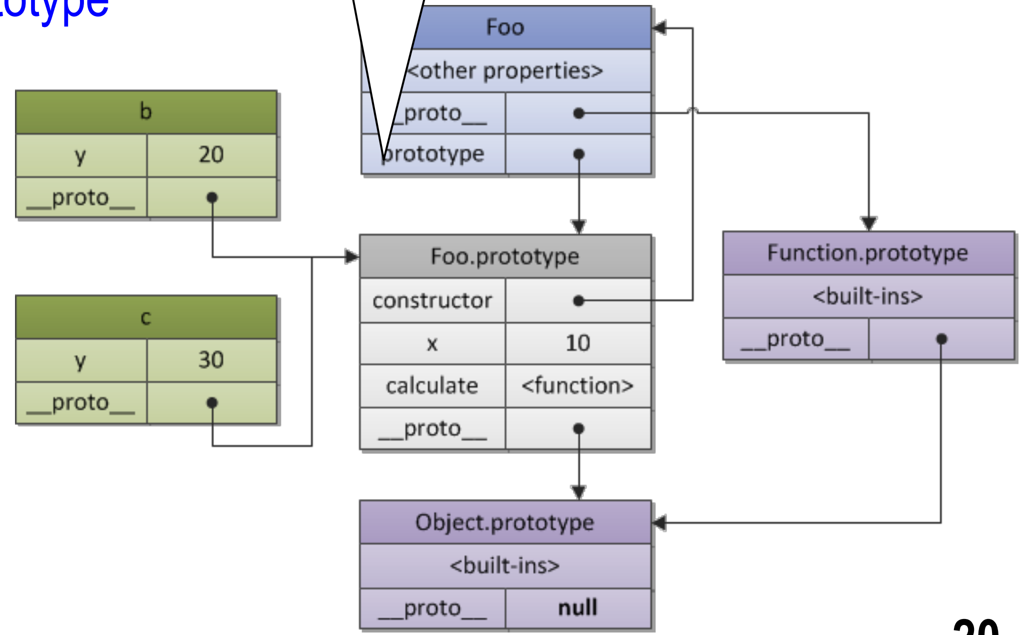
- Besides creation of objects, a constructor function does another useful thing — it automatically sets a “function prototype” object for newly created objects.
  - This function prototype object is stored in the <ConstructorFunction>.prototype property
  - This is different from __proto__ property
- When a JavaScript object is created from the Constructor Function, the __proto__ property of the resulting object points to the function prototype object
  - In other words, any properties and methods added to the function prototype object are available to the resulting object

(You really don't have to understand this internal mechanics in order to use prototype effectively, however.)

ConstructorFunction.prototype

```
 // Function constructor
function Foo(y) {
  this.y = y;
}

// Add property "x" to function prototype
Foo.prototype.x = 10;

// Add method "calculate" to function prototype
Foo.prototype.calculate = function (z) {
  return this.x + this.y + z;
};

// now create our "b" and "c"
// objects are created from “Foo”
var b = new Foo(20);
var c = new Foo(30);

// call the inherited method
b.calculate(30); // 60
c.calculate(40); // 80
```

Foo.prototype points to function prototype object – any properties added to this function prototype oject are available to objects created from Foo



Another Example: ConstructorFunction.prototype

```
// Constructor (Template) of the MyObject
function MyObject(name, size){
    this.name=name;
    this.size=size;
}
// Add a function to the prototype
MyObject.prototype.tellSize=function(){
    alert(“size of  “ + this.name+” is “ + this.size);
}

// Create an instance of the object. Note that new object
// has tellSize() method.
var myObj=new MyObject(“Desk”, “30 inches”);
myObj.tellSize();
```

More Explanation...

- __proto__ provides inheritance only between object instances
- We need inheritance in JavaScript something like Java “class” level
  - In Java, an object that is created from class inherits all properties and methods of that class
  - So, in Java, if you create a new child class with new properties and new methods, any new Java objects created from that child class have the new properties and methods
  - In JavaScript, there is no “class” concept - a Constructor function plays that role, however
  - So, if we add properties to a Constructor function in JavaScript, we want new objects to inherit those newly added properties
  - The “prototype” field of the Constructor function creates an intermediate object called “function prototype object”, the newly added properties are then added
  - The newly created object's __proto__ points to this intermediate object

Lab:

Exercise 2: Function Constructor & Prototype
4266_javascript_advanced.zip

Code with Passion!