Prototype

Terminology

Inheritance

An object gets access to the properties and methods of another object.

Method

A function that is specifically a part of a class (classical inheritance) or as a property on an object (prototypal inheritance)

All methods are functions, but not all functions are methods.

Classical Inheritance

Classical Inheritance

The inheritance method used for class-based languages like C++ or Java.

Class

The definition of all of the properties (data) and methods (functions) that characterize a certain set of objects.

A class is not an object, but rather, the abstraction of the object(s) it is supposed to represent.

```
using namespace std;

class Rectangle {
    int width;
    int height;
  public:
    Rectangle (int, int);
    int area () {
        return (width * height);
    }
};

Rectangle::Rectangle (int a, int b) {
  width = a;
  height = b;
}
```

Instance

The instantiation of a class. An instance has exactly the same properties (data) and methods (functions) of its parent class.

```
#include <iostream>
using namespace std;

int main () {
  Rectangle rectOne (3, 4);
  Rectangle rectTwo (5, 6);
  cout << "rectOne area: " << rectOne.area() << endl;  // rectOne area: 12
  cout << "rectTwo area: " << rectTwo.area() << endl; // rectTwo area: 30
  return 0;
}
```

Classical Inheritance

The instantiation of a class. An instance has exactly the same properties (data) and methods (functions) of its parent class.

```
#include <iostream>
using namespace std;

class Square : public Rectangle {
    public:
        Square(int side);

    private:
}

Square::Square(int side) {
    Rectangle r(side, side);
}

int main() {
    Square squareOne(4);
    cout << "squareOne area: " << squareOne.area() << endl; // squareOne area: 16
}
```

Property & Method Exercise

JavaScript is not like this at all!

Prototype

The internal property created when an object is created. This property simply links to another object. JavaScript will use these links to inherit from other objects.

Prototypal Inheritance

When it comes to inheritance, JavaScript only has one construct: objects.

Each object has an internal link to another object called its prototype.

If an object property or method is referenced, but not present on the current object, JavaScript will look for the method / property in the object linked within the prototype property.

Prototype Chain

Each prototype object has a prototype of its own, and so on until an object is reached with null as its prototype.

null, by definition, has no prototype, and acts as the final link in the prototype chain.

Code Demo

Everything in JavaScript is an Object*

*except primitives

Objects

Objects are collections of properties and values. These properties can be data, or they can be functions (called methods).

Objects in JavaScript inherit from the Object prototype which inherits from null.

Functions

Functions in JavaScript are objects. Everything you can do wtih objects, you can do with functions.

Functions can be assigned to variables, passed around as parameters and be created on the fly.

Likewise, you can attach properties and methods to a function.

Functions inherit from the Function prototype, which inherits from the Object prototype.

Methods

JavaScript does not have methods in the same way that class-based languages define them. In JavaScript, any function can be added to an object in the form of a property, and functions defined this way are called methods.

An inherited function acts just as any other property, including property shadowing, which in this case, is a form of method overriding.

Object.prototype

When a property or method is not found on the an object, JavaScript will look at the next object in the prototype chain to see if the property exists. The "top" of the prototype chain is the Object.prototype, which inherits from null.

Unless explicitly specified, the prototype property is created for objects no matter how they are created.

Nearly all objects in JavaScript are instances of (inherit from) Object.

Viewing Prototype Information

.hasOwnProperty()

To differentiate between properties or methods that exist on a given object versus which exists in the object's prototype chain, you can use the hasOwnProperty() method and supply it the property name you want to check as the argument.

```
var myObj = {
  a: 1,
  b: 2
}

var myOtherObj = {
  b: 3,
  c: 4
}

// don't ever do this for real!
myObj.__proto__ = myOtherObj;

console.log(myObj.hasOwnProperty('a'));
console.log(myObj.hasOwnProperty('c'));
```

Modifying the Prototype

Changes to Object.prototype are seen by all objects. The same is true for any change you make to anything that serves as a prototype.

Modifying built-in JavaScript objects, such as Object, String, Array, Number, etc. can introduce bugs when interacting with code that expects and unchanged built-in prototype.

```
if (typeof String.prototype.trim === 'undefined') {
  String.prototype.trim = function() {
    return this.replace(/^\s+|\s+$/g, '');
  };
}
```

```
Number.prototype.isPositive = function() {
  return (this > 0);
};
```

Prototype Exercise

Prototype Information on MDN

Performance Concerns

Finding properties or methods higher up on the prototype chain can take a long time.

Looking for properties or methods that don't exist will traverse the full chain. The exception to this is the hasOwnProperty.

Practical Examples

Basic Example

Rocket

Basic Example With Arguments

Person

instanceof

Tests whether an object has in its prototype chain the prototype property of a constructor.

```
function MyProto() {

}

function MyOtherProto() {

}

var myObj = new MyProto();
var myOtherObj = new MyOtherProto();

console.log(myObj instanceof MyProto);
console.log(myObj instanceof MyOtherProto);
console.log(myOtherObj instanceof MyProto);
console.log(myOtherObj instanceof MyOtherProto);
```

Creating a Hierarchy

Employee

Creating a Hierarchy cont.

Manager

Creating a Hierarchy cont.

Worker

Creating a Hierarchy cont.

Engineer

Creating a Hierarchy cont.

Sales

Creating a Hierarchy cont.

Exercises

Creating a Hierarchy

DIY

Shapes