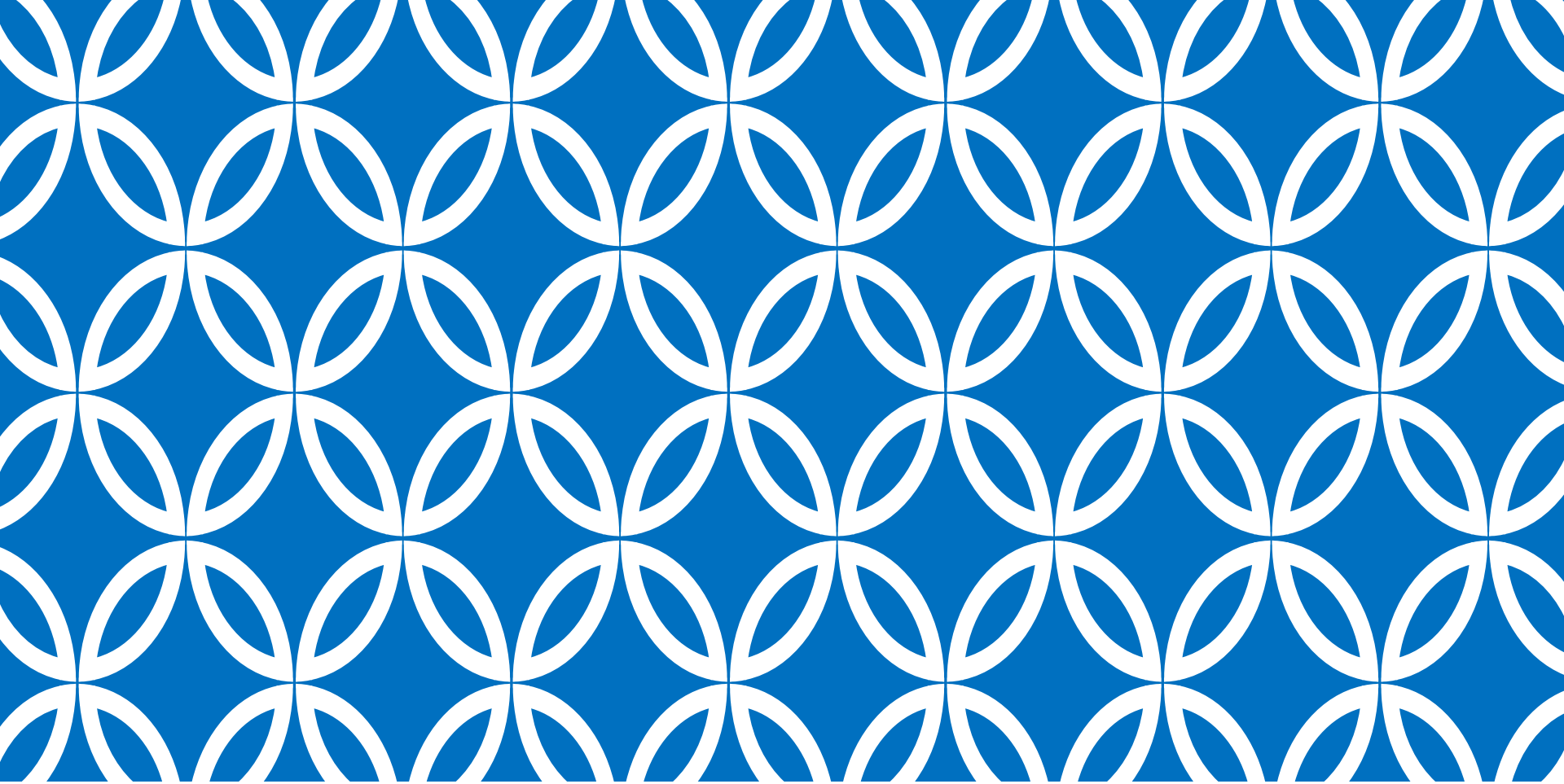
ACADGILD

FRONT END DEVELOPMENT (WITH ANGULARJS)

Website : http://www.acadgild.com
LinkedIn : https://www.linkedin.com/company/acadgild
Facebook : https://www.facebook.com/acadgild



Session 4 – Advanced JavaScript (contd.)

Agenda – Advanced JavaScript

1. Enumerable Properties
2. Prototype
3. Functions and Prototype
4. _ _proto_ _ Property (Object)
5. Prototypal Inheritance
6. Multiple Inheritance
7. Parasitic inheritance
8. Copy Prototype of Inheritance
9. Deep Copy

Enumerable Properties

- **Enumerable** properties are those which can be iterated by a for…in loop.
- **Enumerable:** true or false. Whether the property shows in some loop constructs, such as

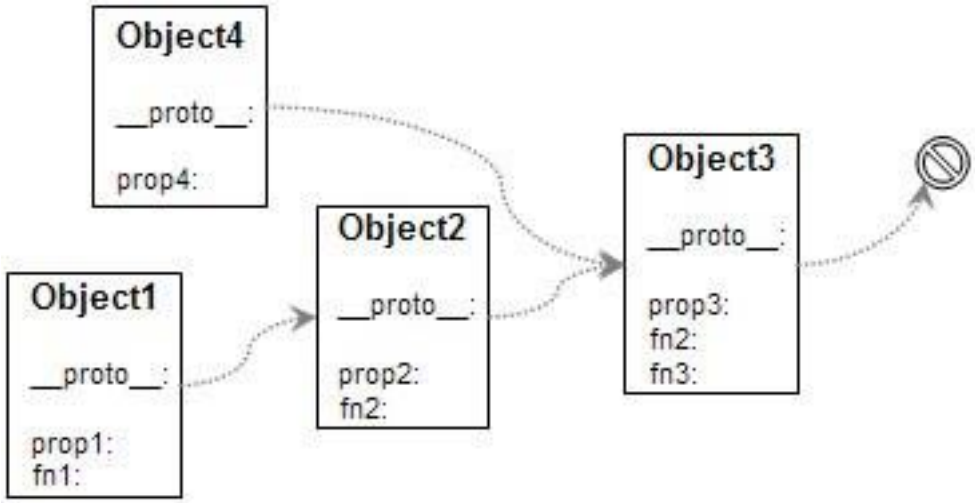
  for (var x in o) {…} and Object.keys(o)

- **Checking Property's Enumerable Attribute**

  obj.propertyIsEnumerable(p)

Prototype

- Every JavaScript object has a prototype. The prototype is also an object.
- **An ancestor of a JavaScript object is**
  - like a "super-object" instead of a superclass
  - a parent at the object level rather than at the class level



Prototype (contd.)

- **Every object contains a reference to its prototype**
  - default: Object.prototype; strings → String.prototype; etc.

- **A prototype can have a prototype, and so on**
  - an object "inherits" all methods/data from its prototype(s)
  - doesn't have to make a copy of them
  - saves memory
  - prototypes allow JavaScript to mimic classes, inheritance

Functions and Prototype

- **Every function stores a prototype object property in it**
  - **Example:** when we define our Point function (constructor), that creates a Point.prototype
  - initially this object has nothing in it ({})
  - every object you construct will use the function's prototype object as its prototype

- **Every new Point object uses Point.prototype**

```
// also causes Point.prototype to be defined
function Point(xValue, yValue) {
    ...
}
```

_ _proto_ _ Property (Object)

- Contains a reference to the internal prototype of the specified object.

  object._ _proto_ _

- **Parameters**

  object : Required. The object on which to set the prototype.

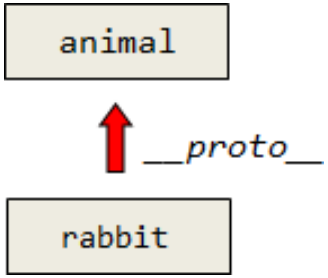
```
function Rectangle() {  }

var rec = new Rectangle();

if (console && console.log) {
        console.log(rec._ _proto_ _ === Rectangle.prototype);
        // Returns true
        rec._ _proto_ _ = Object.prototype;
        console.log(rec._ _proto_ _ === Rectangle.prototype);
        // Returns false
 }
```

Prototypal Inheritance

- In JavaScript, the inheritance is prototype-based. That means that there are no classes. Instead, an object inherits from another object
- Inheritance, the _ _proto_ _



- When an object rabbit inherits from another object animal, in JavaScript that means that there is a special property

  rabbit. _ _proto_ _ = animal.

- When a *rabbit* property is accessed and the interpreter can't find it in *rabbit*, it follows the *_proto_ link* and searches in *animal*.

Multiple Inheritance

- Inheritance is all about copying properties from parent to child prototype, then why not copying properties from multiple parents.

```
function multiInheritance() {
        var n = {}, stuff, j = 0, length = arguments.length;
              for (j = 0; j <length; j++) {
              stuff = arguments[j];
                    for (var index in stuff) {
                                  if (stuff.hasOwnProperty(index)) {
                                 n[index] = stuff[index];
                                  }
                     }
               }
        return n;
}
```

Parasitic Inheritance

- This Pattern as suggested by Douglas Crockford.
- In parasitic inheritance a power constructor calls another constructor, takes the result, augments it and returns it as though it did all the work.
- Parasitic inheritance is different from prototypal inheritance which we have discussed so far.
- Prototypal inheritance is used more often because its more efficient than parasitic inheritance.

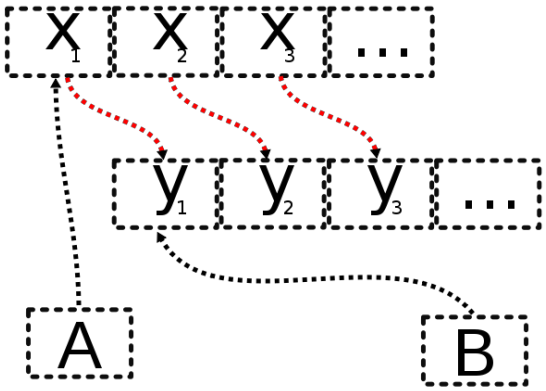
Copy Prototype of Inheritance

- Clone()
- MyClass.prototype = clone(AnotherClass.prototype);
- By cloning the prototype we get a new copy of it and assign that to MyClass's prototype so that changing the inherited properties will not affect the parent's prototype's properties.
- Like this would MyClass.prototype = AnotherClass.prototype

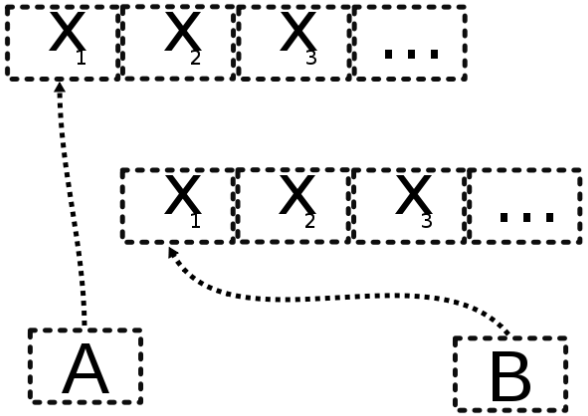
```
function clone (obj)
{
   function CloneFactory () {}
   CloneFactory.prototype = obj;
   return new CloneFactory();
}
```

Copy Prototype of Inheritance

- **copy()** makes a shallow, non-recursive copy of a single object. This implementation is interesting because it handles native types and correctly copies objects created by a user-defined class.
- **deepCopy()** is the entry point for the deep copy algorithm. Every member is recursively deep copied.



Deep Copy in Progress



Deep Copy is completed

Deep Copy

```
function deepCopy(p, c) {
        c = c || {};
        for (var index in p) {
                if (p.hasOwnProperty(index)) {
                        if (typeof p[index] === 'object') { c[index] =
                                                Array.isArray(p[index]) ? [] : {};
                        deepCopy(p[index], c[index]);
                        }
                        else {
                        c[index] = p[index];
                         }
                }
         }
 return c;
}
```

Lets Discuss Assignments

Assignment

Get in Touch with ACADGILD

Contact Info:

- **Website : http://www.acadgild.com**
- **LinkedIn : https://www.linkedin.com/company/acadgild**
- **Facebook : https://www.facebook.com/acadgild**
- **Support: support@acadgild.com**