Objects and Prototypes

Object's Prototypes

- Every JS object has a special property `__proto__` pronounced "dunder proto" (double underscore), that points to another object (the prototype object)
- When we use `Object.create` to create an object it sets the `__proto__` property to the object passed in

```
var foo = {
  a: 1,
  b: 2,
};

var bar = Object.create(foo);
bar.__proto__ === foo;                  // true
```

- **In this case we say that the object assigned to `foo` is the prototype object of the object returned by `Object.create` and assigned ot `bar`**
- `Foo.prototype` references `Function.prototype` while `Foo.__proto__` is set as the prototype object for any object returned by `Foo` when invoked as a constructor

`Object.getPrototypeOf` and `isPrototypeOf`

- The `__proto__` property first introduced by Firefox as nonstandard object property
- In actual JS specification, this property is defined as `[[Prototype]]` which is not somethin gyou can interact with directly
- `__proto__` got popular though and has been adopted by almost all modern browsers and is standardized in ECMAScript 6
- For compatibility reasons, its better to use the following two methods:
    - `Object.getPrototypeOf(obj)` to get the prototype object of `obj`
    - `obj.isPrototypeOf(foo)` to check if `obj` is a prototype object of `foo`

Prototype Chain and the `Object.prototype` Object

- We can use `Object.create` to create objects that form a prototype chain
- The `Object.prototype` object is at the end of the prototype chain for all JS objects
    - If you don't create an object from a prototype, its default prototype is `Object.prototype` object

Prototypal Inheritance and Behavior Delegation

- JS's prototype chain lookup for properties allos us to store an object's data and behaviors not just in the object but anywhere on its prototype chain

- Allows us to create objects more easily and don't have to duplicate properties on every single object
- If we need to add/remove/update behavior to apply to all dogs, we can just modify the prototype object and all instances will pick up those changes
- This pattern is known as JavaScript's Prototypal Inhertitance
- JS doesn't have true classes, but in a true object oriented way, objects can be created directly from other objects and behaviors/methods canbe shared by the prototype chain
- From a top down view, objects on the bottom of the prototype chain "inherit" properties and behaviors of all upstream objects
- From a bottom down view, objects on the bottom of the chain can "delegate" requests to the upstream objects to be handled
    - This design pattern is called Behavior Delegation

Overriding Default Behavior

- Objects created from prototypes can override shared behaviors by defining the same methods locally (polymorphism)
- Doesn't impact other objects created from the prototype

Object.getOwnPropertyNames and object.HasOwnProperty

- The following two methods allow you to check an object's own property:
- `obj.hasOwnProperty('property')` returns true/false if property is defined on the object itself
- `Object.getOwnPropertyNames(obj)` returns an array of object's own property names

Methods on Object.prototype

- The `Object.prototype` object is on the top of all JavaScript's object's prototype chain, and the methods defined there can be used from any JS object
- 3 useful ones:
    - Object.prototype.toString()`
    - Object.prototype.isPrototypeOf(obj)`
    - Object.prototype.hasOwnProperty(property)`

The `extend` function

- `extend` is JavaScripts way for the "Mixin" pattern
- ES 6 provides `Object.assign`
    - Used to copy the values of all enumerable own properties from one or more source objects to a target object and returns it