Lesson Weekend

Intermediate JavaScript (/intermediate-javascript) / Object-Oriented JavaScript (/intermediate-javascript/object-oriented-javascript) / Constructors and Prototypes

Text Cheat sheet

Terminology

- **Constructor:** A blueprint for creating many of the same type objects — like strings and dogs. Constructors define the properties that every instance of an object will have.
    - Most built-in JavaScript objects have a constructor function that we can use to create objects of that type. An exception is the `Math` object, which does not have a constructor function. Some built-in objects can only be accessed via the constructor.
    - Generally speaking, if data can be created with literal notation, like a string, it's most common to use literal notation instead of the constructor function.
    - We can also create constructor functions for our own custom object types.
    - The first letter of the name of a constructor function is always capitalized, following UpperCamelCase, and it matches the name of the object type.

- **Array-like object:** Objects that have numbers as their property keys to represent an index location are called **array-like objects**. Similar to arrays, we can use bracket notation to access these objects, passing in an index. While the property keys look like numbers, they are actually strings. Here is an example:

```
> let testGreeting2 = new String("Hello!");
undefined
> testGreeting2;
String { Hello! }
  0: "H",
  1: "e",
  2: "l",
  3: "l",
  4: "o",
  5: "!",
  length: 6,
  [[Prototype]]: String,
  [[PrimitiveValue]]: "Hello!"
> testGreeting2[4];
"0"
```

- **Type:** The type (https://developer.mozilla.org/en-US/docs/Glossary/Type) of an object defines what data it contains, including all of the properties and methods that an object of that type has in it. In other words, the object type is the blueprint or template for creating an object, but not an actual object.
    - Note that you'll also see the terminology **class** if you are exploring constructor functions on MDN. A class is simply an object type, and in JavaScript we can create an object type by using class syntax. In a later course section, we'll learn how to use class syntax.
- **Instance:** Objects created with a constructor are instances of the object type defined by the constructor. A constructor can be used to create many instances of the same type.

- **Prototype:** Prototypes store properties to be shared by all objects of the same type. For example, `String.prototype` is shared by all `String` objects; `Dog.prototype` is shared by all `Dog` objects. Prototypes store properties that objects of that type inherit.
    - We choose to add methods to the prototype of an object type so that we can define the method once, but all object instances still have access to that method. This saves memory and is therefore more efficient. This is in contrast to adding a method to the constructor. In this case, when an instance of an object is created, so is a new instance of that method (and any other properties) which takes up more memory.
- **Prototype Chain:** Each object has a prototype property (called `__proto__`) that is a link to another object. This creates a **prototype chain** and is the mechanism by which one object inherits from multiple other object types. The end of the prototype chain is represented by the value `null` .

Examples

Here is a constructor function for `Dog` that will initialize a new dog object with its attributes assigned to the values passed in to the constructor function.

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

Then to create a new dog we can do the following:

```
let myPuppy = new Dog("Ernie", ["brown","black"], 3);
```

We can access the name of the new dog:

```
> myPuppy.name;
"Ernie"
```

The colors of the new dog:

```
> myPuppy.colors;
["brown","black"]
```

And its age:

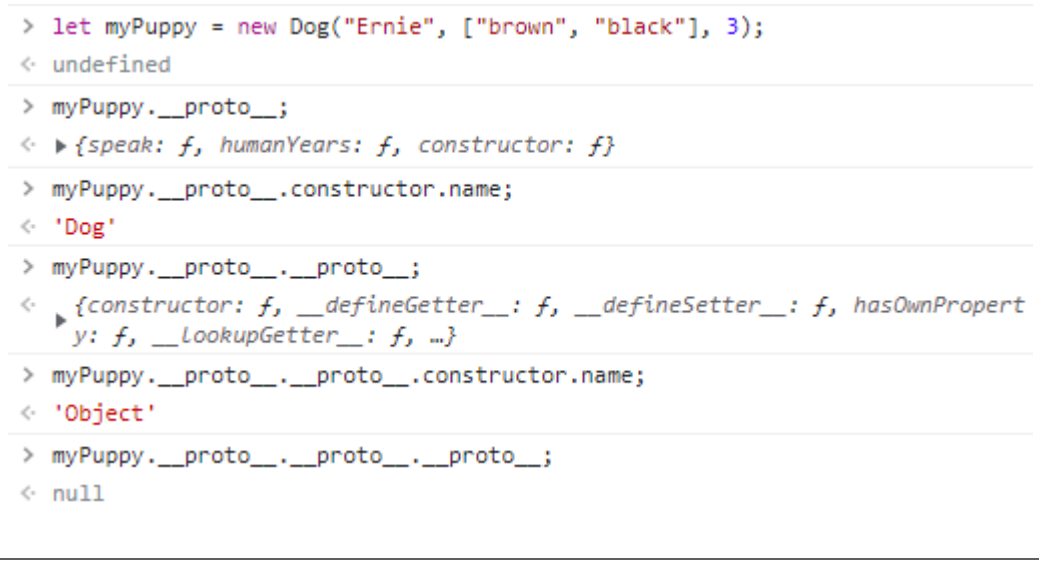
```
> myPuppy.age;
3
```

We can add a method to the `Dog` prototype and all instances of the `Dog` object, current and future, will have access to this method:

```
> Dog.prototype.speak = function() {
  console.log("Woof!");
};
```

```
> myPuppy.speak();
Woof!
```

We can access the prototype chain for the Dog object by accessing the `__proto__` property for each object in the chain. As we see in the image below, the `myPuppy` object inherits from the `Dog` object type, which itself inherits from the `Object` object type, which itself does not inherit from anything and ends the chain of inheritance.



Previous (/intermediate-javascript/object-oriented-javascript/literal-notation-versus-constructors)
Next (/intermediate-javascript/object-oriented-javascript/constructor-and-prototype-methods)

Lesson 6 of 33
Last updated March 23, 2023

disable dark mode

Epicodus (http://www.epicodus.com)