CS 4220

- Current Trends in Web Design & Development -

Cydney Auman

AGENDA

01 HW Review

02 The Last Data Type

03 Prototypes

04 ES6 JavaScript Classes

05 Code Demo

06 QUIZ! & Lab Time!

The Last Data Type: Object

The **object** is a complex data type. An object is a value in memory which is typically referenced by a variable. The data type **object** is considered a non-primitive. This means the values are mutable, meaning the value of an object and/or array can be changed after it is created. It also means objects cannot be compared by value.

- An object contains properties, which are referred to as a key-value pair.
- An array is a type of object used for storing multiple values .
- A function is callable object that executes a block of code.

Prototypes

All objects in JavaScript have a built-in internal property called a **prototype**. It contains common attributes and/or properties on all instances of the object.

This is the reason why JavaScript is described as a prototype-based language. In order to provide inheritance, objects have this built-in **prototype** which acts as the template object so that it can inherit methods and properties.

```
console.log(Object.getOwnPropertyNames(Object.prototype))
console.log(Object.getOwnPropertyNames(Array.prototype))
console.log(Object.getOwnPropertyNames(Function.prototype));
```

Understanding JavaScript Classes

Classes in JavaScript are ***not*** like classes in Java.

In languages like Java, classes are created and are templates for objects. When you want a new object, you instantiate the class. This essentially tells the language to copy the methods and properties of the class into a new entity, called an **instance**. Once this instantiation happens it no longer has an active relation with the original class.

JavaScript does not have this concept of copying mechanics. Meaning that "instantiating" a class in JavaScript does create a new object, but not one that is independent of its original class. JavaScript creates an object that is **linked** to a prototype. ***Changes to that prototype propagate to the new object, even after instantiation.***

ES6 JavaScript Classes

JavaScript classes introduced in ES6 are syntactical sugar over JavaScript's existing prototype-based inheritance. The class syntax is ***not*** introducing a new class based object-oriented model to JavaScript.

To declare a class, you use the **class** keyword. The optional **constructor** method is a special method for creating and initializing an object created with a class.

```
class Bear {
   constructor(type) {
       this.type = type;
   }
}
```

ES6 JavaScript Classes

JavaScript classes can only contain functions/method definitions. And unlike when creating object literals, you do not separate methods inside the class body with commas.

```
class Bear {
   constructor(type) {
       this.type = type;
   }

   speak(word) {
       console.log(`The ${this.type} bear says '${word}'`);
   }
}
```

ES6 JavaScript Classes

The **get** syntax binds an object property to a function that will be called when that property is looked up. With the get syntax, it does not need to be called using ( ).

```
class Bear {
    constructor(type) {
        this.type = type;
    }

    get typeOfBear() {
        console.log(`This is a ${this.type} bear.`);
    }
}
```

ES6 JavaScript Classes

The **extends** keyword is used in class declarations to create a class as a child of another defined class. The **super** keyword is used to call corresponding methods of super class.

```
class Polar extends Bear {
   constructor(type) {
        super(type);
   }

   speak() {
        super.speak();
        console.log(`${this.type} it is too cold.`);
   }
}
```

What is THIS?

In Javascript the **this** keyword refers to the current object instance. The context of Javascript's **this** depends on how and where the method/function is called.

```
class Bear {
    constructor(type) {
        this.type = type;
    }

    speak(word) {
        console.log(`The ${this.type} bear says '${word}'`);
    }
}
```

Chaining Methods

Method chaining is when **each** method returns an object, allowing the calls to be chained together in a single statement without requiring a variable to store the intermediate results.

From the previous slide, **this** is inside the object instance and has access to all the methods defined on the instance ***as well as*** access to the built-in prototype. So by return **this** from each method inside our class we can achieve method chaining.

```
const calculator = new Calculator();

const calculation = calculator

    .add(1)

    .add(5)

    .subtract(3).total;
```

```
console.log(‘Week 04’);
console.log(‘Code Examples’);
```

Lab, Homework and Prep

Lab Time

- Quiz on CSNS.
- Finish up Homework 2 Due by ***tonight*** at 11:59pm

Preparation for Next Week

- Read Eloquent Javascript Chapters 11