JavaScript Class

Summary: in this tutorial, you'll learn about the JavaScript class and how to use it effectively.

A JavaScript class is a blueprint for creating objects. A class encapsulates data and functions that manipulate data.

Unlike other programming languages such as Java and C#, JavaScript classes are syntactic sugar over the prototypal inheritance. In other words, ES6 classes are just special functions.

Classes before ES6 revisited

Before ES6, JavaScript had no concept of classes. To mimic a class, you often use the constructor/prototype pattern as shown in the following example:

```
function Person(name) {
    this.name = name;
}

Person.prototype.getName = function () {
    return this.name;
};

var john = new Person("John Doe");
console.log(john.getName());
```

Output:

```
John Doe
```

How it works.

First, create the `Person` as a constructor function that has a property name called `name`. The `getName()` function is assigned to the `prototype` so that it can be shared by all instances of the `Person` type.

Then, create a new instance of the `Person` type using the `new` operator. The `john` object, hence, is an instance of the `Person` and `Object` through prototypal inheritance.

The following statements use the `instanceof` operator to check if `john` is an instance of the `Person` and `Object` type:

```
console.log(john instanceof Person); // true
console.log(john instanceof Object); // true
```

ES6 class declaration

ES6 introduced a new syntax for declaring a class as shown in this example:

```
class Person {
    constructor(name) {
        this.name = name;
    }
    getName() {
        return this.name;
    }
}
```

This `Person` class behaves like the `Person` type in the previous example. However, instead of using a constructor/prototype pattern, it uses the `class` keyword.

In the `Person` class, the `constructor()` is where you can initialize the properties of an instance. JavaScript automatically calls the `constructor()` method when you instantiate an object of the class.

The following creates a new `Person` object, which will automatically call the `constructor()` of the `Person` class:

```
let john = new Person("John Doe");
```

The `getName()` is called a method of the `Person` class. Like a constructor function, you can call the methods of a class using the following syntax:

```
objectName.methodName(args)
```

For example:

```
let name = john.getName();
console.log(name); // "John Doe"
```

To verify the fact that classes are special functions, you can use the `typeof` operator of to check the type of the `Person` class.

```
console.log(typeof Person); // function
```

It returns `function` as expected.

The `john` object is also an instance of the `Person` and `Object` types:

```
console.log(john instanceof Person); // true
console.log(john instanceof Object); // true
```

Class vs. Custom type

Despite the similarities between a class and a custom type defined via a constructor function, there are some important differences.

First, class declarations are not hoisted like function declarations.

For example, if you place the following code above the `Person` class declaration section, you will get a `ReferenceError`.

```
let john = new Person("John Doe");
```

Error:

```
Uncaught ReferenceError: Person is not defined
```

Second, all the code inside a class automatically executes in the strict mode. And you cannot change this behavior.

Third, class methods are non-enumerable. If you use a constructor/prototype pattern, you have to use the `Object.defineProperty()` method to make a property non-enumerable.

Finally, calling the class constructor without the `new` operator will result in an error as shown in the following example.

```
let john = Person("John Doe");
```

Error:

```
Uncaught TypeError: Class constructor Person cannot be invoked without 'new'
```

> Note that it's possible to call the constructor function without the `new` operator. In this case, the constructor function behaves like a regular function.

Summary

- Use the JavaScript `class` keyword to declare a new class.
- A `class` declaration is syntactic sugar over prototypal inheritance with additional enhancements.