OOPS In JavaScript

- Getting a full understanding of how OOP works in JavaScript is a bit difficult especially in ES5 syntax, ES6 class made it a lot easier to use object constructor.
- JavaScript's object-oriented programming system can be difficult, especially if your background is in a language that uses a class-based system such as Python, Java, or C++.

Ways to implement OOPS in JavaScript

We have four ways for working with OOP in JavaScript :-

- Factory function.
- Function constructor.
- Object.create().
- ES6 classes.

Factory function

In this, we create a function that receives parameter and in return, it provides an object.

Example

```
let personA = {
   first_name: "Rahul",
   last_name: "Singh",
   getFullName() {
     return `Full Name is: ${this.first_name} ${this.last_name}`;
  }
};

console.log(personA.getFullName())

let personB = {
   first_name: "Rishabh",
   last_name: "Mittal",
   getFullName() {
      return `Full Name is: ${this.first_name} ${this.last_name}`;
   }
};

console.log(personB.getFullName())
```

Here we are repeating "getFullName" method
Therefore, we can create a Factory function for avoiding code duplicacy

```
let person = function(first_name,last_name) {
   return {
      first_name,
      last_name,
      getFullName() {
        return `Full Name is: ${this.first_name} ${this.last_name}`;
      }
   }
}
```

```
let personC = person('Rohit','Singh');
let personD = person('Yash','Veer');

console.log(personC.getFullName());
console.log(personD.getFullName());
```

note :-

- Here in the example, personA and personB both have getFullName function definition which indicate duplication of the code which is not good programming practice.
- To avoid that we have created a factory function called person in which we pass the first_name and last_name as a parameter and it will provide the required object.

- By using factory function we have avoided repeating our code. But here is a problem with memory efficiency as getFullName function will be created each time the person function is called which is not good because if we can place getFullName in common place in memory where each new person object can call it then it would also be memory efficient, to achieve this let's move to constructor function.

Function constructor

A function that is initiated with a "new" keyword is a constructor function and the constructor function should start with a capital letter.

Example

```
let Person = function(first_name,last_name) {
   this.first_name = first_name;
   this.last_name = last_name;
}

Person.prototype.getFullName = function() {
   return `Full Name is: ${this.first_name} ${this.last_name}`;
}

let personA = new Person('Rohit','Singh');
let personB = new Person('Rahul','Singh');

console.log(personA.getFullName())
console.log(personB.getFullName())
```

In each instance personA, personB getFullName function is shared means when personA.getFullName() is called, the javascript engine will first look in personA object if it is not found there then it will look in the personA parent that is Person.prototype from where it get resolved

note :-

- In the above example Person is the constructor function and it has getFullName function in its prototype object, I have created two instance personA, personB from Person constructor.
- We can see that the code is resuable by having a constructor and prototype.
- Constructor function has that code which is unique to an instance that means personA and personB has there own first_name and last_name properties while the prototype has that code which is shared by the instance and also prototype properties are not copied to the instance, they are resolved through prototype chain which makes constructor function memory efficient.

Object.create() Method

- The "Object.create()" method creates a new object, using an existing object as the prototype of the newly created object.
- In the previous method we've learned how to create prototype using constructor function, now let see how we can create prototype using "Object.create()".

Example

```
let person = {
    first_name:'lorem',
    last_name:'ipsum',
    getFullName(){
      return `Fullname is ${this.first_name} ${this.last_name}`;
  }
}

let personA = Object.create(person);
personA.first_name = 'Rohit';
personA.last_name = 'Singh';
console.log(personA.getFullName());
```

note :-

- In the above example, I have created a person object and use it as the prototype of the personA object using "Object.create(person)".
- Object.create will create a prototype chain where "__proto__" property of personA will point to person object.

ES6 classes

- ES6 brings the keyword class which is familiar to most of the OOPS developers.
- Class in JavaScript is like a syntactical sugar behind the scenes it still follows prototypal inheritance.
- The class keyword was brought to bring simplicity and easiness for the developers to write OOP programming in JavaScript.

Example -- With Class as Function Constructor

```
let Person = function(first_name,last_name) {
   this.first_name = first_name;
   this.last_name = last_name;}
Person.prototype.getFullName = function() {
   return `Full Name is: ${this.first_name} ${this.last_name}`;}
let personA = new Person('Rohit','Singh');
let personB = new Person('Rahul','Singh');
console.log(personA.getFullName());
console.log(personB.getFullName());
```

Example -- With "class" Keyword

```
class Person{
   constructor(first_name,last_name){
      this.first_name = first_name;
      this.last_name = last_name;   }}
Person.prototype.getFullName = function() {
   return `Full Name is: ${this.first_name} ${this.last_name}`;}
let personA = new Person('Rohit','Singh');
let personB = new Person('Rahul','Singh');
console.log(personA.getFullName());
console.log(personB.getFullName());
```

note :-

- Above is an example of class.
- I have created Person class which contains the constructor function whose properties and method will be copied to each instance while the rest of the method or properties like getFullName is shared.

Classes in ES6

- In ES6 we can create classes. If you've come from a language like PHP or Python, this will be familiar to you.
- A class in terms of OOP is a blueprint for creating objects. A class encapsulates data for the object.
- In ES6, a class is a special kind of function, that we saw in ES5 as function declaration & function expression.
- Class syntax is a syntactical sugar & behind the scenes, it still uses a prototype-based model.
- In ES6 Classes are functions, and functions are objects in JavaScript.
- A class has properties and methods in it for every instance(object) of that class that are called data members.
- ES6 Classes support prototype-based inheritance, super calls, instance and static methods and constructors.
- In ES6, we can create the class by using the "class" keyword.
- Any number of instance can be created from a class & each instance is called Object.
- ES6 classes is always executed in strict mode. So, the code containing the silent error or mistake throws an error.
- Classes in JavaScript are not a blueprint like in other languages. They just define objects that can be modified at runtime.

Features of ES6 classes

Below there are following features of ES6 classes :-

- ES6 classes make inheritance much easier.
- Decreasing the chances of bugs.

Prototypes Vs Classes

Prototypes

- A class defines a type which can be instantiated at runtime.
- A child of an ES6 class is another type definition which extends the parent with new properties and methods, which in turn can be instantiated at runtime.

Classes

- A prototype is itself an object instance.
- A child of a prototype is another object instance which delegates to the parent any properties that aren't implemented on the child.

Ways to declare classes in ES6

There are two ways by which we can declare classes in ES6 includes :-

- Class Declaration / Anonymous Class Expression
- Class Expression / Named Class Expression

a) Class Declaration / Class Literals

One way to define a class is using a class declaration. To declare a class, we use the "class" keyword with the name of the class.

Syntax

```
class className {  // Base Class, Parent Class, Super Class == className
   /* Properties and Methods */
}
```

Example -- Creating Class Instance

- In order to create an instance of the class, use the "new" keyword followed by the class name.
- The process of creating an "instance" of a class is known as instantiation.
- The new keyword is responsible for instantiation.

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script src="module.js" type="module"></script>
</body>
</html>
```

module.js

```
class Person {
    /* Properties and Methods */
}
let p = new Person(); // Here "p" is Class instance
console.log(p)
```

Example -- Checking Datatype of Class

index.html

Same as Previous Example

module.js

```
class Person {
    /* Properties and Methods */
}

let p = new Person(); // Here "p" is Class instance
console.log(typeof p);
```

Example -- Checking Class Instance

index.html

Same as Previous Example

module.js

```
class Person {
    /* Properties and Methods */
}
let p = new Person();
console.log(p instanceof Person); // true
```

Example -- Checking Class Name

index.html

Same as Previous Example

module.js

```
class Person {
    /* Properties and Methods */
}
let p = new Person();
console.log(Person.name); // Person
```

b) Class Expression

- A class expression is another way to define a class.
- Class expressions can be named or unnamed.

Syntax -- Unnamed Class Expressions

```
let className = class{  // Base Class, Parent Class, Super Class == className
    /* Properties and Methods */
}
```

Syntax -- Named Class Expressions

```
let sampleName = class className{ // Base Class, Parent Class, Super Class ==
className
    /* Properties and Methods */
}
```

Example -- Unnamed Class Expression

index.html

Same as Previous Example

module.js

```
let Person = class {
   /* Properties and Methods */
}

let p = new Person(); // instantiate an object.
console.log(p); // Person{}
```

Example -- Named Class Expression

index.html

Same as Previous Example

module.js

```
let Person = class personInfo {
   /* Properties and Methods */
}

let p = new Person(); // instantiate an object.
console.log(p); // personInfo{}
```

Example -- Calling Class Directly

```
class Meetup {
   constructor() {
      console.log("Zero Parameter Constructor")
   }
}
console.log(new Meetup());
```

Example -- Checking Datatype

index.html

Same as Previous Example

module.js

```
let Person = class{
   /* Properties and Methods */
}
let p = new Person();
console.log(typeof p); // object
```

Example -- Checking Instance of Unnamed Class

index.html

Same as Previous Example

module.js

```
let Person = class {

}
let p = new Person();
console.log(p instanceof Person);
```

Example -- Checking Instance of Named Class

```
let Person = class personInfo {

}
let p = new Person(); // Cause Object is Created Through Here
console.log(p instanceof Person)
```

Example -- Checking Unnamed Class Name

index.html

Same as Previous Example

module.js

```
let Person = class {

}
let p = new Person();
console.log(Person.name); // Person
```

Example -- Checking Named Class Name

The name given to a named class expression is local to the class's body. (it can be retrieved through the class's (not an instance's) name property, though).

index.html

Same as Previous Example

module.js

```
let Person = class personInfo {

}

let p = new Person();
console.log(Person.name); // personInfo
```

Hoisting

- Hoisting is a feature where we first access our class and then declare it.
- Class Expression & Class Declarations are not hoisted.

Example -- For Class Expression

```
let p = new Person();
console.log(Person.name); // Cannot access 'Person' before initialization

let Person = class {

}
```

Example -- For Class Declaration

```
let p = new Person();
console.log(Person.name); // Cannot access 'Person' before initialization

class Person {

}
```

Methods in Classes

- In JavaScript, a method is a function that belongs to an object or class.
- Class methods are created with the same syntax as object methods.

There are two types of methods in ECMAScript classes :-

- Instance Methods / Prototype Methods.
- Static Methods / Class Methods