Introduction to Object Oriented Programming in JavaScript

Last Updated : 24 May, 2023

As JavaScript is widely used in Web Development, in this article we will explore some of the **Object Oriented** mechanisms supported by **JavaScript** to get the most out of it. Some of the common interview questions in JavaScript on OOPS include:

- How is Object-Oriented Programming implemented in JavaScript?
- How does it differ from other languages?
- Can you implement Inheritance in JavaScript?

and so on…

There are certain features or mechanisms which make a Language Object-Oriented like:

OOPs Concept in JavaScript		
Object	Classes	**Encapsulation**
Abstraction	Inheritance	Polymorphism

Let's dive into the details of each one of them and see how they are implemented in JavaScript.

Object: An Object is a **unique** entity that contains **properties** and **methods**. For example "a car" is a real-life Object, which has some characteristics like color, type, model, and horsepower and performs certain actions like driving. The characteristics of an Object are called Properties in Object-Oriented Programming and the actions are called methods. An Object is an **instance** of a class. Objects are everywhere in

JavaScript, almost every element is an Object whether it is a function, array, or string.

Note: A Method in javascript is a property of an object whose value is a function.

The object can be created in two ways in JavaScript:

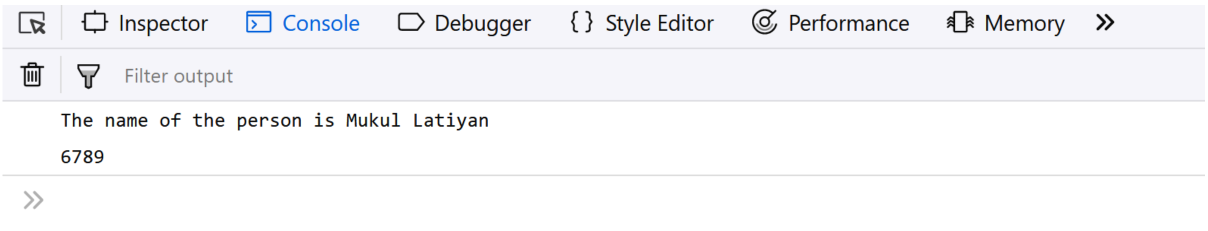
- **Object Literal**
- **Object Constructor**

Example: Using an Object Literal.

```
// Defining object
let person = {
    first_name: 'Mukul',
    last_name: 'Latiyan',

    //method
    getFunction: function () {
        return (`The name of the person is
          ${person.first_name} ${person.last_name}`)
    },
    //object within object
    phone_number: {
        mobile: '12345',
        landline: '6789'
    }
}
console.log(person.getFunction());
console.log(person.phone_number.landline);
```

Output:



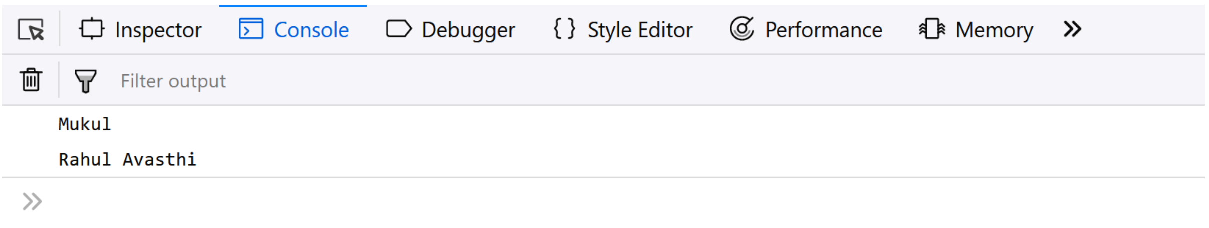
```
The name of the person is Mukul Latiyan
6789
```

Example: Using an Object Constructor.

```
// Using a constructor
function person(first_name, last_name) {
    this.first_name = first_name;
    this.last_name = last_name;
}
// Creating new instances of person object
let person1 = new person('Mukul', 'Latiyan');
let person2 = new person('Rahul', 'Avasthi');

console.log(person1.first_name);
console.log(`${person2.first_name}
${person2.last_name}`);
```

Output:



```
Mukul
Rahul Avasthi
```

Note: The JavaScript Object.create() Method creates a new object, using an existing object as the prototype of the newly created object.

Example:

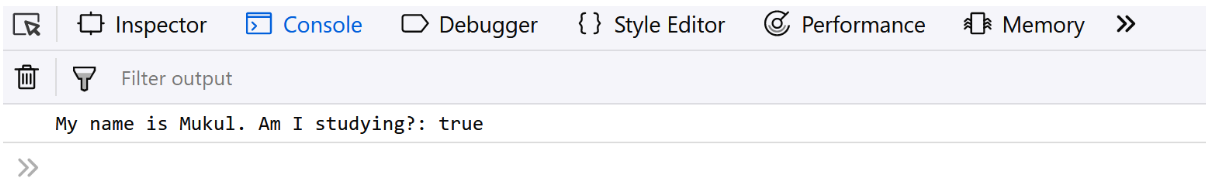
```
// Object.create() example a
// simple object with some properties
const coder = {
    isStudying: false,
    printIntroduction: function () {
        console.log(`My name is ${this.name}. Am I
              studying?: ${this.isStudying}.`)
    }
}
// Object.create() method
const me = Object.create(coder);

// "name" is a property set on "me", but not on
"coder"
me.name = 'Mukul';

// Inherited properties can be overwritten
me.isStudying = true;

me.printIntroduction();
```

Output:



```
My name is Mukul. Am I studying?: true
```

Classes: Classes are **blueprints** of an Object. A class can have many Objects because the class is a **template** while Objects are **instances** of the class or the concrete implementation.
Before we move further into implementation, we should know unlike other Object Oriented languages there are **no classes in JavaScript** we have only Object. To be more precise, JavaScript is a prototype-based Object Oriented Language, which means it doesn't have classes, rather it defines behaviors using a constructor function and then reuses it using the prototype.

Note: Even the classes provided by ECMA2015 are objects.

JavaScript classes, introduced in ECMAScript 2015, are primarily syntactical sugar over JavaScript's existing prototype-based inheritance. The class syntax is not introducing a new object-oriented inheritance model to JavaScript. JavaScript classes provide a much simpler and clearer syntax to create objects and deal with inheritance.

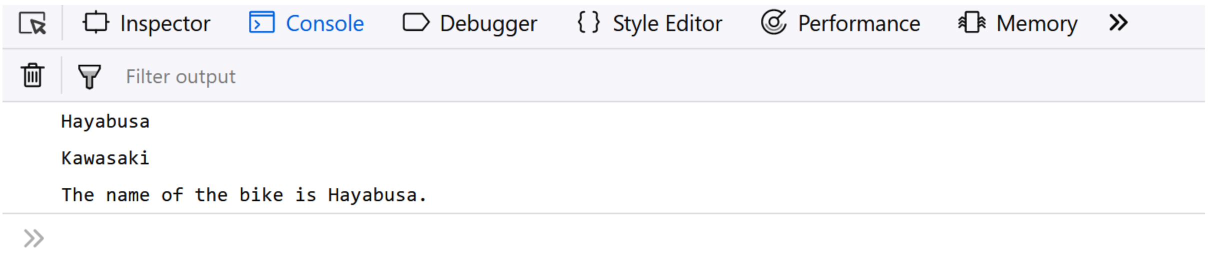
-Mozilla Developer Network

Example: Let's use ES6 classes then we will look at the traditional way of defining an Object and simulate them as classes.

```
// Defining class using es6
class Vehicle {
    constructor(name, maker, engine) {
        this.name = name;
        this.maker = maker;
        this.engine = engine;
    }
    getDetails() {
        return (`The name of the bike is
${this.name}.`)
    }
}
// Making object with the help of the constructor
let bike1 = new Vehicle('Hayabusa', 'Suzuki',
'1340cc');
let bike2 = new Vehicle('Ninja', 'Kawasaki', '998cc');

console.log(bike1.name);    // Hayabusa
console.log(bike2.maker);   // Kawasaki
console.log(bike1.getDetails());
```

Output:



```
Hayabusa
Kawasaki
The name of the bike is Hayabusa.
```

Example: Traditional Way of defining an Object and simulating them as classes.

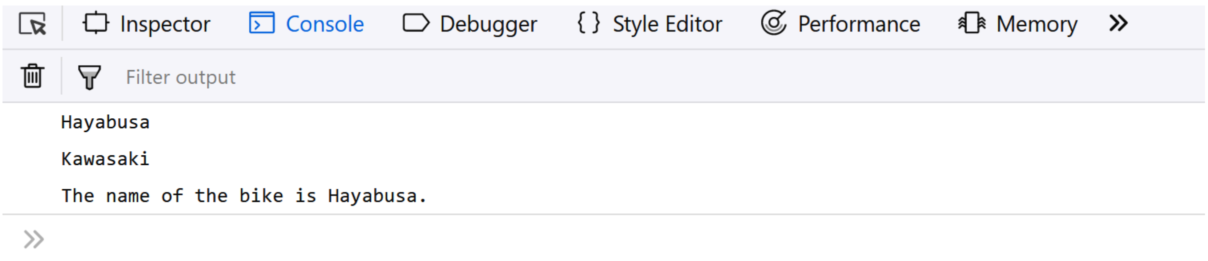
```
// Defining class in a Traditional Way.
function Vehicle(name, maker, engine) {
    this.name = name,
        this.maker = maker,
        this.engine = engine
};

Vehicle.prototype.getDetails = function () {
    console.log('The name of the bike is ' +
this.name);
}

let bike1 = new Vehicle('Hayabusa', 'Suzuki',
'1340cc');
let bike2 = new Vehicle('Ninja', 'Kawasaki', '998cc');

console.log(bike1.name);
console.log(bike2.maker);
console.log(bike1.getDetails());
```

Output:



```
Hayabusa
Kawasaki
The name of the bike is Hayabusa.
```

As seen in the above example it is much simpler to define and reuse objects in ES6. Hence, we would be using ES6 in all of our examples.

Abstraction: Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

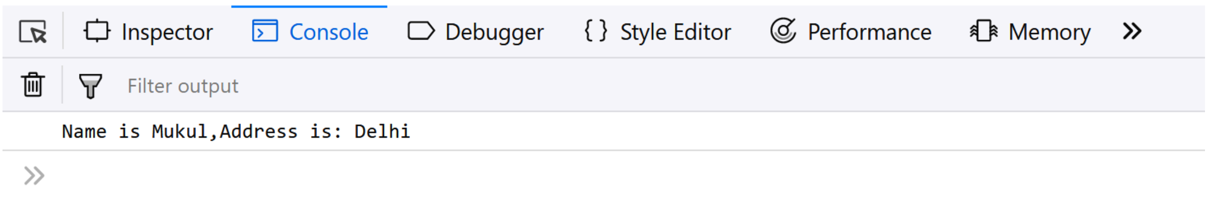
Encapsulation: The process of **wrapping properties and functions** within a **single unit** is known as encapsulation.

Example: Let's understand encapsulation with an example.

```javascript
// Encapsulation example
class person {
    constructor(name, id) {
        this.name = name;
        this.id = id;
    }
    add_Address(add) {
        this.add = add;
    }
    getDetails() {
        console.log(`Name is ${this.name},
        Address is: ${this.add}`);
    }
}

let person1 = new person('Mukul', 21);
person1.add_Address('Delhi');
person1.getDetails();
```

Output: In this example, we simply create a person Object using the constructor, Initialize its properties and use its functions. We are not bothered by the implementation details. We are working with an Object's interface without considering the implementation details.



```
Name is Mukul,Address is: Delhi
```

Sometimes encapsulation refers to the **hiding of data** or **data Abstraction** which means representing essential features hiding the background detail. Most of the OOP languages provide access modifiers to restrict the scope of a variable, but there are no such access modifiers in JavaScript, there are certain ways by which we can restrict the scope of variables within the Class/Object.

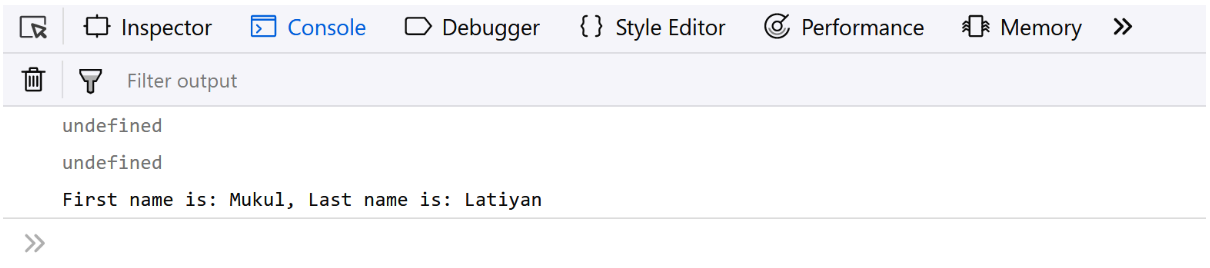
Example:

```javascript
// Abstraction example
function person(fname, lname) {
    let firstname = fname;
    let lastname = lname;

    let getDetails_noaccess = function () {
        return (`First name is: ${firstname} Last
            name is: ${lastname}`);
    }

    this.getDetails_access = function () {
        return (`First name is: ${firstname}, Last
            name is: ${lastname}`);
    }
}
let person1 = new person('Mukul', 'Latiyan');
console.log(person1.firstname);
console.log(person1.getDetails_noaccess);
console.log(person1.getDetails_access());
```

Output: In this example, we try to access some property(person1.firstname) and functions(person1.getDetails_noaccess) but it returns undefined while there is a method that we can access from the person object(person1.getDetails_access()). By changing the way we define a function we can restrict its scope.



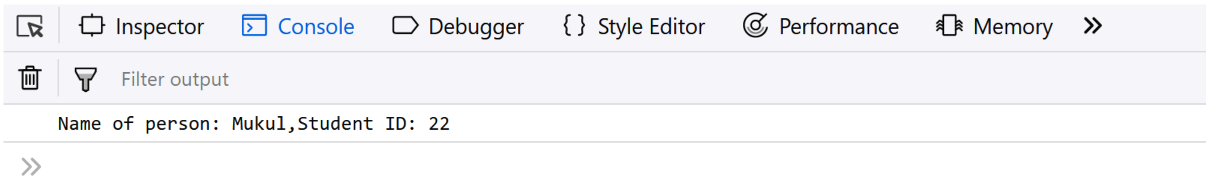
```
undefined
undefined
First name is: Mukul, Last name is: Latiyan
```

Inheritance: It is a concept in which some properties and methods of an Object are being used by another Object. Unlike most of the OOP languages where classes inherit classes, JavaScript Objects inherit Objects i.e. certain features (property and methods) of one object can be reused by other Objects.

Example: Let's understand inheritance and polymorphism with an example.

```javascript
// Inheritance example
class person {
    constructor(name) {
        this.name = name;
    }
    // method to return the string
    toString() {
        return (`Name of person: ${this.name}`);
    }
}
class student extends person {
    constructor(name, id) {
        // super keyword for calling the above
        // class constructor
        super(name);
        this.id = id;
    }
    toString() {
        return (`${super.toString()},
        Student ID: ${this.id}`);
    }
}
let student1 = new student('Mukul', 22);
console.log(student1.toString());
```

Output: In this example, we define a Person Object with certain properties and methods and then we inherit the Person Object in the Student Object and use all the properties and methods of the person Object as well as define certain properties and methods for the Student Object.



```
Name of person: Mukul,Student ID: 22
```

Note: The Person and Student objects both have the same method (i.e toString()), this is called **Method Overriding**. Method Overriding allows a method in a child class to have the same name(polymorphism) and method signature as that of a parent class.

In the above code, the super keyword is used to refer to the immediate parent class's instance variable.

Polymorphism: Polymorphism is one of the core concepts of object-oriented programming languages. Polymorphism means the same function with different signatures is called many times. In real life, for example, a boy at the same time may be a student, a class monitor, etc. So a boy can perform different operations at the same time. Polymorphism can be achieved by method overriding and method overloading

JavaScript is best known for web page development but it is also used in a variety of non-browser environments. You can learn JavaScript from the ground up by following this JavaScript Tutorial and JavaScript Examples.

Comment More info

Campus Training Program

Next Article

JavaScript Objects