Object Oriented Programming in JavaScript

As JavaScript is widely used in Web Development, here we would explore some of the **Object Oriented** mechanism supported by **JavaScript** to get most out of it.

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

- **Object**
- **Classes**
- **Encapsulation**
- **Inheritance**

1. Object–

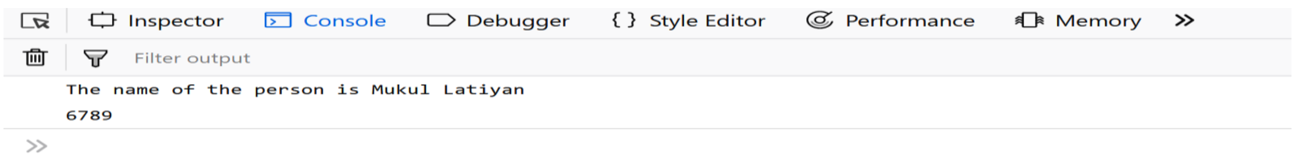
An Object is a **unique** entity which contains **property** and **methods**. For example "car" is a real life Object, which have some characteristics like color, type, model, horsepower and performs certain action like drive. The characteristics of an Object are called as Property, in Object Oriented 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, arrays and string.

Object can be created in two ways in JavaScript:

- 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
```

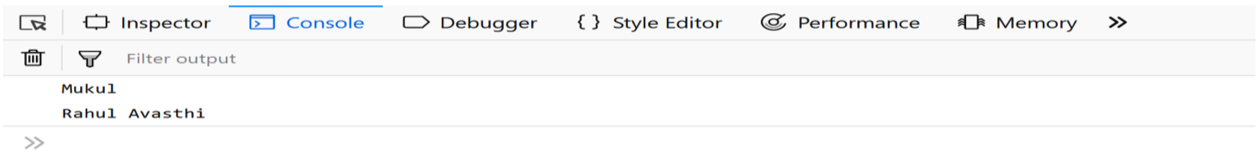
- Using an **Object Constructor:**

```
//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
```

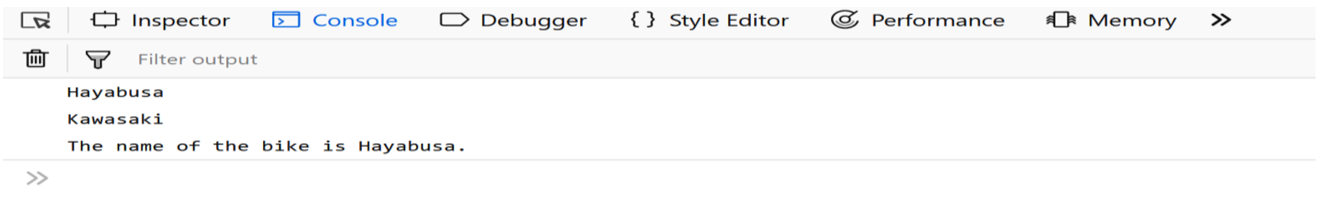
2. **Classes–**

Classes are **blueprint** of an Object. A class can have many Object, because class is a **template** while Object are **instances** of the class or the concrete implementation. Before we move further into implementation, we should know unlike other Object Oriented Language there is **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 define behaviors using constructor function and then reuse it using the prototype.

Example:
Lets use ES6 classes then we will look into traditional way of defining 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:



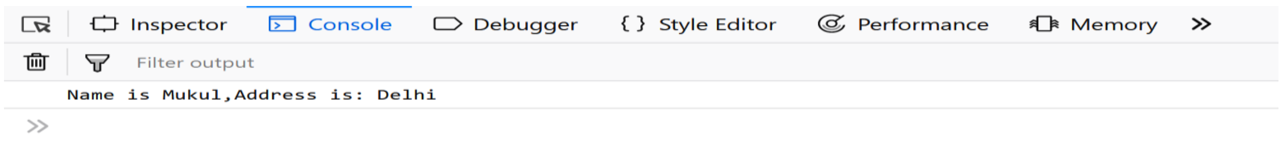
```
Hayabusa
Kawasaki
The name of the bike is Hayabusa.
```

3. **Encapsulation –**

The process of **wrapping property and function** within a **single unit**is known as encapsulation.
Let’s understand encapsulation with an example.

```
//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:



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

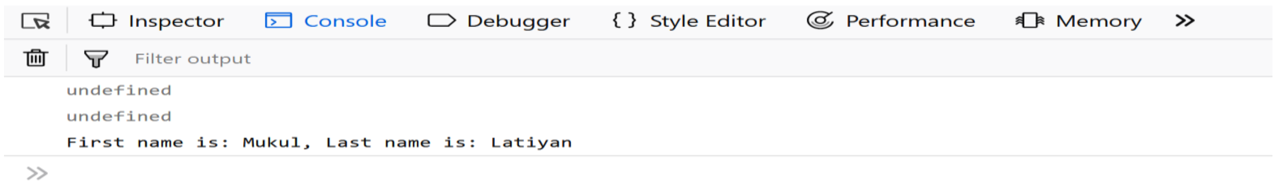
Sometimes encapsulation refers to **hiding of data** or **data Abstraction** which means representing essential features hiding the background detail. Most of the OOP languages provide access modifiers to restrict the scope of a variable, but there are no such access modifiers in JavaScript but there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

Example:

```
// 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:



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

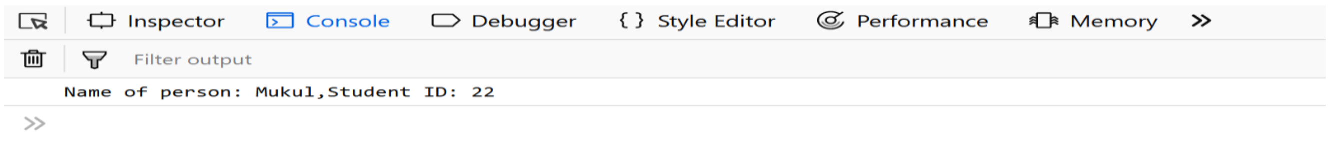
4. Inheritance –

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

Lets's understand inheritance with example:

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

Output:



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