

## **LAB ASSIGNMENT NO – 04(b)**

**Aim:** Write a Program on inheritance, Iterators and Generators

### **Theory:**

#### **1. Inheritance-**

Inheritance enables you to define a class that takes all the functionality from a parent class and allows you to add more.

Using class inheritance, a class can inherit all the methods and properties of another class.

Inheritance is a useful feature that allows code reusability.

To use class inheritance, you use the extends keyword.

#### **2. Iterators-**

Javascript Iterator is an object or pattern that allows us to traverse over a list or collection. Iterators define the sequences and implement the iterator protocol that returns an object by using a next() method that contains the value and is done. The value contains the next value of the iterator sequence and the done is the boolean value true or false if the last value of the sequence has been consumed then it's true else false.

In Array.prototype you will find Symbol(Symbol.iterator): *f* values() method. The array is by default iterable. Also, String, Map & Set are built-in iterables because their prototype objects all have a Symbol.iterator() method.

#### **3. Generators-**

A generator-function is defined like a normal function, but whenever it needs to generate a value, it does so with the yield keyword rather than return. The yield statement suspends the function's execution and sends a value back to the caller, but retains enough state to enable the function to resume where it is left off. When resumed, the function continues execution immediately after the last yield run.

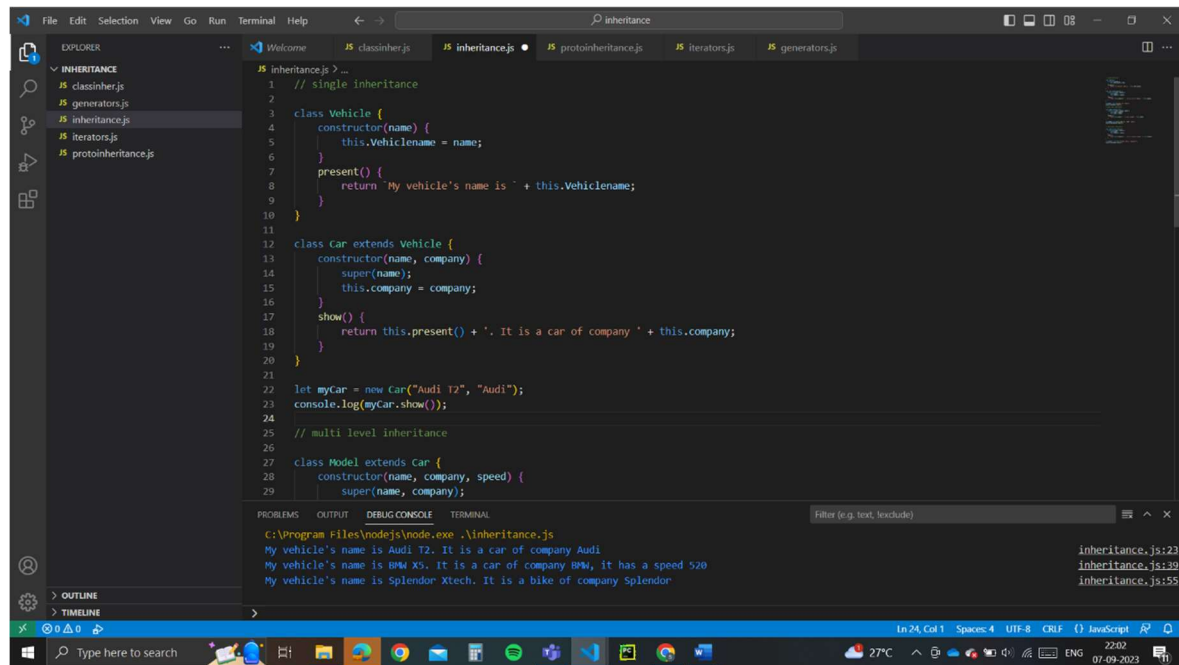
Syntax :

// An example of generator function

```
function* gen(){  
  yield 1;  
  yield 2;  
  ...  
  ...  
}
```

**Conclusion:** this experiment demonstrated the power of inheritance, iterators, and generators in JavaScript for enhancing code reusability and managing complex data structures efficiently. By leveraging these features, we achieved more elegant and modular code, simplifying the manipulation of data and the creation of custom iterable objects. This experiment underscores the significance of these JavaScript capabilities in modern software development.

## **Code:**



```
1 // single inheritance
2
3 class Vehicle {
4   constructor(name) {
5     this.Vehiclename = name;
6   }
7   present() {
8     return 'My vehicle's name is ' + this.Vehiclename;
9   }
10 }
11
12 class Car extends Vehicle {
13   constructor(name, company) {
14     super(name);
15     this.company = company;
16   }
17   show() {
18     return this.present() + '. It is a car of company ' + this.company;
19   }
20 }
21
22 let myCar = new Car("Audi T2", "Audi");
23 console.log(myCar.show());
24
25 // multi level inheritance
26
27 class Model extends Car {
28   constructor(name, company, speed) {
29     super(name, company);
```

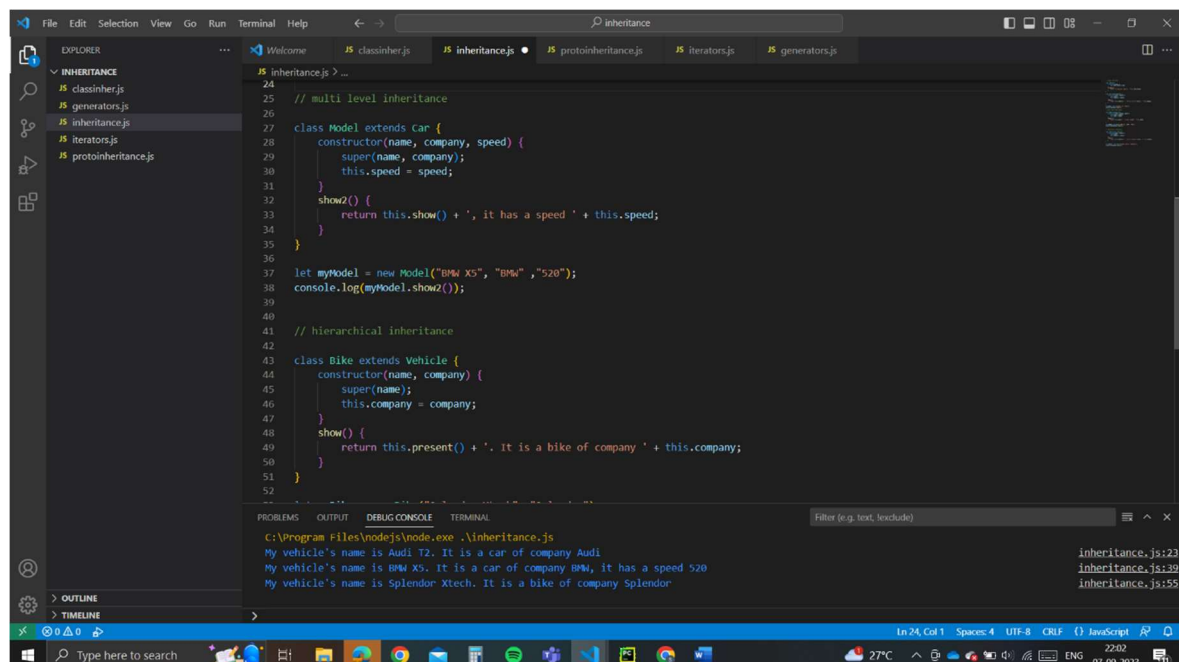
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

C:\Program Files\nodejs\node.exe .\inheritance.js

My vehicle's name is Audi T2. It is a car of company Audi

My vehicle's name is BMW X5. It is a car of company BMW, it has a speed 520

My vehicle's name is Splendor Xtech. It is a bike of company Splendor



```
24
25 // multi level inheritance
26
27 class Model extends Car {
28   constructor(name, company, speed) {
29     super(name, company);
30     this.speed = speed;
31   }
32   show2() {
33     return this.show() + ', it has a speed ' + this.speed;
34   }
35 }
36
37 let myModel = new Model("BMW X5", "BMW", "520");
38 console.log(myModel.show2());
39
40
41 // hierarchical inheritance
42
43 class Bike extends Vehicle {
44   constructor(name, company) {
45     super(name);
46     this.company = company;
47   }
48   show() {
49     return this.present() + '. It is a bike of company ' + this.company;
50   }
51 }
52
```

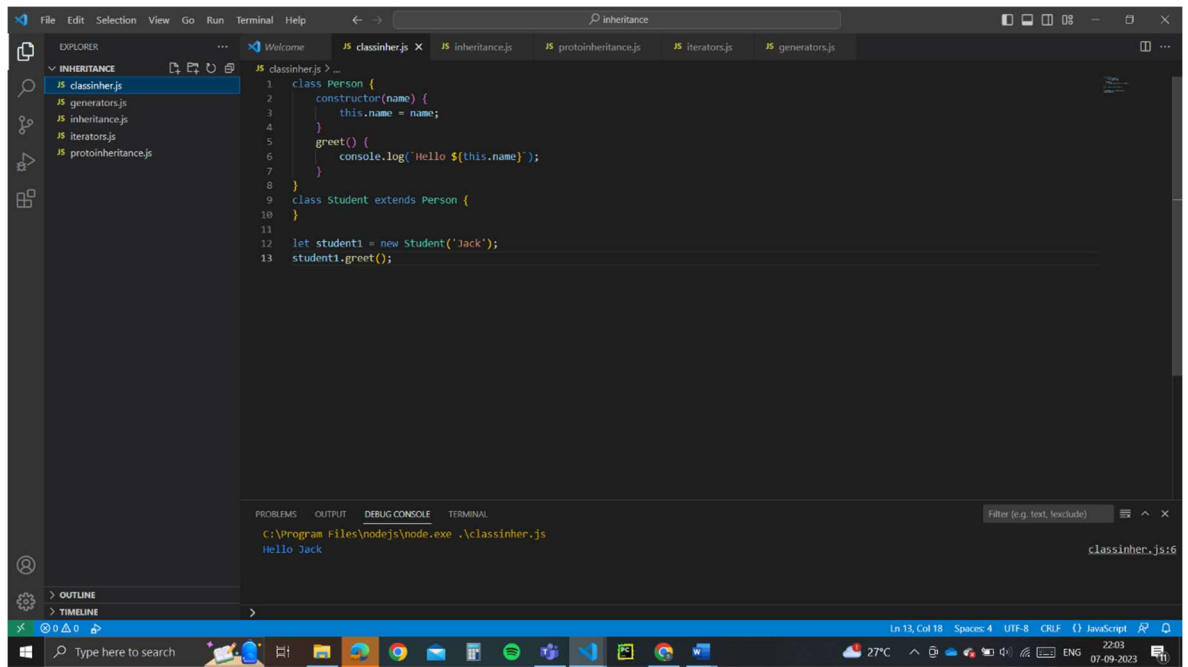
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

C:\Program Files\nodejs\node.exe .\inheritance.js

My vehicle's name is Audi T2. It is a car of company Audi

My vehicle's name is BMW X5. It is a car of company BMW, it has a speed 520

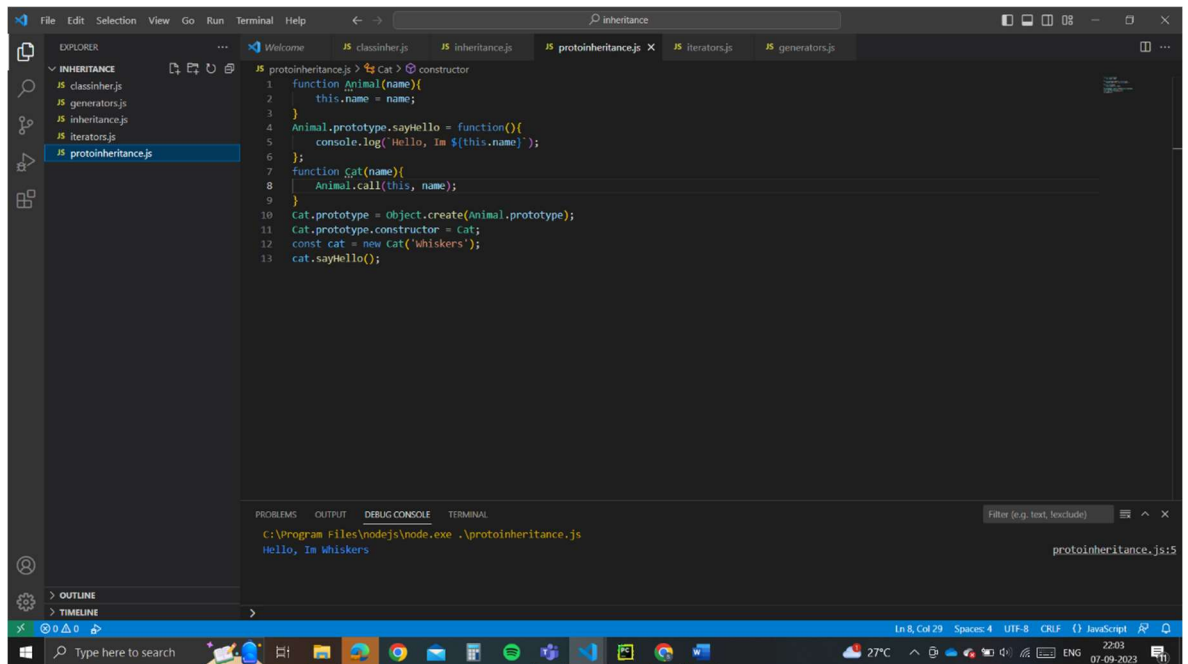
My vehicle's name is Splendor Xtech. It is a bike of company Splendor



This screenshot shows a Visual Studio Code editor window with a project named 'inheritance'. The Explorer sidebar on the left shows a folder named 'INHERITANCE' containing files: 'classinher.js', 'generators.js', 'inheritance.js', 'iterators.js', and 'protoninheritance.js'. The 'classinher.js' file is open in the editor, displaying the following JavaScript code:

```
1 class Person {
2   constructor(name) {
3     this.name = name;
4   }
5   greet() {
6     console.log('Hello ${this.name}');
7   }
8 }
9 class Student extends Person {
10 }
11
12 let student1 = new Student('Jack');
13 student1.greet();
```

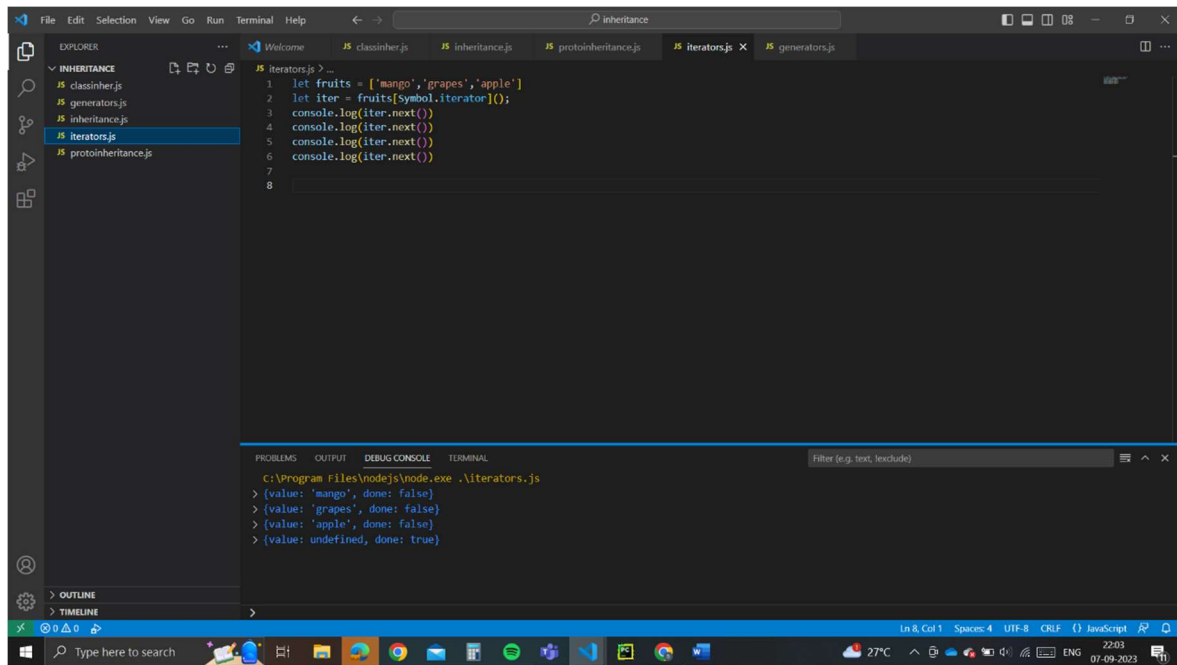
The bottom panel shows the 'TERMINAL' tab with the command `C:\Program Files\nodejs\node.exe .\classinher.js` and its output: `Hello Jack`. The status bar at the bottom indicates 'Ln 13, Col 18', 'Spaces: 4', 'UTF-8', 'CRLF', and 'JavaScript'.



This screenshot shows the same Visual Studio Code editor window, but now the 'protoninheritance.js' file is open. The code defines an 'Animal' constructor function and a 'Cat' constructor function that inherits from 'Animal' using prototype manipulation:

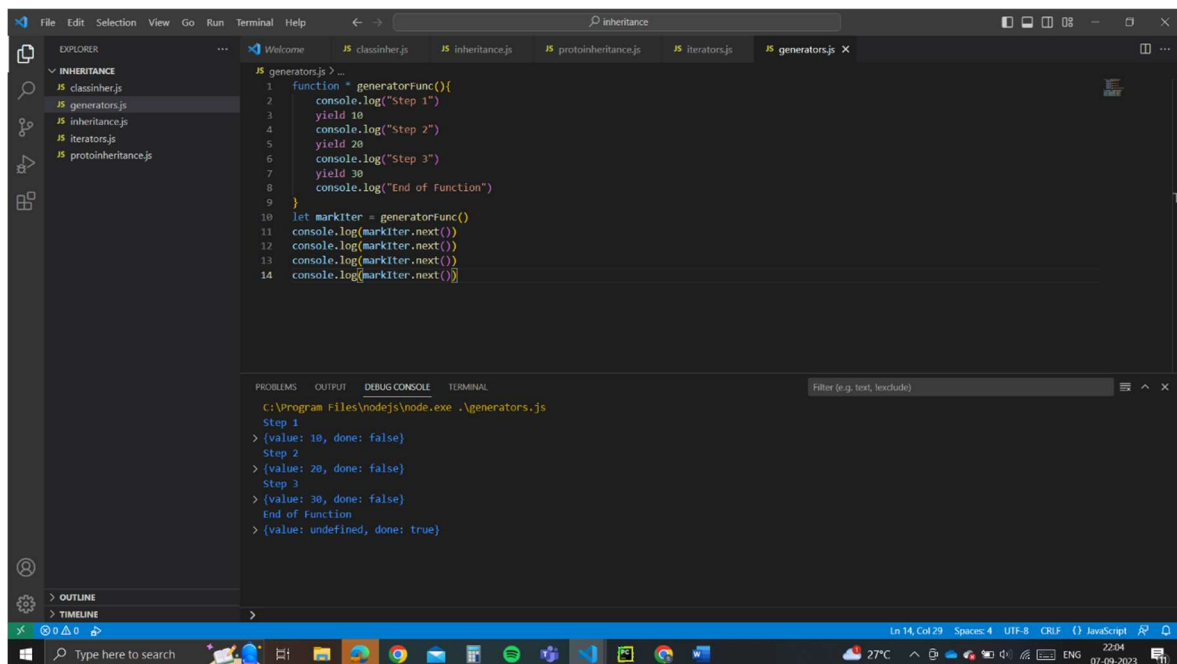
```
1 function Animal(name){
2   this.name = name;
3 }
4 Animal.prototype.sayHello = function(){
5   console.log('Hello, Im ${this.name}');
6 };
7 function Cat(name){
8   Animal.call(this, name);
9 }
10 Cat.prototype = Object.create(Animal.prototype);
11 Cat.prototype.constructor = Cat;
12 const cat = new Cat('whiskers');
13 cat.sayHello();
```

The 'TERMINAL' tab shows the command `C:\Program Files\nodejs\node.exe .\protoninheritance.js` and its output: `Hello, Im whiskers`. The status bar at the bottom indicates 'Ln 8, Col 29', 'Spaces: 4', 'UTF-8', 'CRLF', and 'JavaScript'.



```
File Edit Selection View Go Run Terminal Help
inheritance
EXPLORER
  INHERITANCE
    classinher.js
    generators.js
    inheritance.js
    iterators.js
    protoinheritance.js
  OUTLINE
  TIMELINE
iterators.js > ...
1 let fruits = ['mango', 'grapes', 'apple']
2 let iter = fruits[Symbol.iterator]();
3 console.log(iter.next());
4 console.log(iter.next());
5 console.log(iter.next());
6 console.log(iter.next());
7
8

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
c:\Program Files\nodejs\node.exe .\iterators.js
> {value: 'mango', done: false}
> {value: 'grapes', done: false}
> {value: 'apple', done: false}
> {value: undefined, done: true}
```



```
File Edit Selection View Go Run Terminal Help
inheritance
EXPLORER
  INHERITANCE
    classinher.js
    generators.js
    inheritance.js
    iterators.js
    protoinheritance.js
  OUTLINE
  TIMELINE
generators.js > ...
1 function * generatorFunc(){
2   console.log("Step 1")
3   yield 10
4   console.log("Step 2")
5   yield 20
6   console.log("Step 3")
7   yield 30
8   console.log("End of Function")
9 }
10 let markiter = generatorFunc()
11 console.log(markiter.next())
12 console.log(markiter.next())
13 console.log(markiter.next())
14 console.log(markiter.next())

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
c:\Program Files\nodejs\node.exe .\generators.js
Step 1
> {value: 10, done: false}
Step 2
> {value: 20, done: false}
Step 3
> {value: 30, done: false}
End of Function
> {value: undefined, done: true}
```

**Lab Outcome :** LO4- Use Javascript to develop interactive web pages.