1-Abstract VS interface:

Abstract Classes and Interfaces in JavaScript are both used to achieve abstraction, which allows a programmer to hide the implementation details and show only the essential features of an object.

The comparative analysis between Abstract Classes and Interfaces in JavaScript is as follows:

1. Definition:

* Abstract Classes are defined using the **class** keyword with at least one abstract method.
* Interfaces are not directly supported in JavaScript. However, you can use the **interface** keyword in TypeScript.

1. Implementation:

* In JavaScript, abstract classes can have both abstract and non-abstract methods.
* Interfaces in TypeScript can have only abstract methods.

1. Multiple Inheritance:

* JavaScript does not support multiple inheritance. However, you can achieve multiple inheritance-like behavior by mixing in objects using mixins.
* TypeScript allows a class to extend only one class but can implement multiple interfaces.

1. Instance Variables:

* Abstract Classes in JavaScript can have instance variables.
* Interfaces in TypeScript cannot have instance variables but can have only static readonly properties (constants).

1. Abstract Methods:

* Abstract Classes in JavaScript can have abstract methods, which are declared by setting the function body to **undefined**.
* Interfaces in TypeScript can have only abstract methods.

2-Inheritance in function constructor:

Function constructor is the way you create an object in JavaScript before ES6 introduced the class syntax. The inheritance pattern in JavaScript can be done by prototype chaining or by using a constructor function that sets the prototype of the object.

Here's an example of inheritance using the constructor function:

function Employee(name, age, gender, id) {

this.name = name;

this.age = age;

this.gender = gender;

this.id = id;

};

function Developer(name, age, gender, id, specialization) {

// Calling Employee constructor function

Employee.call(this, name, age, gender, id);

// Adding a new parameter

this.specialization = specialization;

}

console.log(Employee.prototype);

console.log(Developer.prototype);