Abstract vs Interface

Abstract Classes:

- In JavaScript, you can create abstract classes using function constructors and prototypes.
- An abstract class can have both implemented and abstract (unimplemented) methods.
- You need to create subclasses that provide implementations for the abstract methods.
- Example:

```
function AbstractClass() {
  if (this.constructor === AbstractClass) {
      console.error("Cannot instantiate abstract class");
  }
}

AbstractClass.prototype.abstractMethod = function () {
  console.error ("Abstract method must be implemented");
};

function ConcreteClass() {}
ConcreteClass.prototype = Object.create(AbstractClass.prototype);
ConcreteClass.prototype.constructor = ConcreteClass;

ConcreteClass.prototype.abstractMethod = function () {
  console.log("Concrete implementation");
};

const instance = new ConcreteClass();
instance.abstractMethod(); // Output: "Concrete implementation"
```

Interfaces:

- Interfaces in JavaScript are often represented as a collection of method signatures.
- Objects can implicitly implement interfaces by providing the required methods.
- There's no strict enforcement.
- Example:

```
function MyInterface() {}

MyInterface.prototype.method1 = function () {
  console.error ("Method must be implemented");
};

MyInterface.prototype.method2 = function () {
  console.error ("Method must be implemented");
};

function MyClass() {}
MyClass.prototype.method1 = function () {
  console.log("Implemented method1");
};

const instance = new MyClass();
instance.method1(); // Output: "Implemented method1"
```

Inheritance in Function Constructor

In JavaScript, you can achieve inheritance between function constructors using the prototype chain.

```
function Parent(name) {
  this.name = name;
}

Parent.prototype.sayHello = function () {
  console.log(`Hello, my name is ${this.name}`);
};

function Child(name, age) {
  // Call the Parent constructor with the current instance
  Parent.call(this, name);
  this.age = age;
}

// Set up the prototype chain
Child.prototype = Object.create(Parent.prototype);
Child.prototype.constructor = Child;

Child.prototype.sayAge = function () {
  console.log(`I am ${this.age} years old`);
};

// Usage
const parentInstance = new Parent("Parent");
parentInstance.sayHello(); // Output: "Hello, my name is Parent"

const childInstance = new Child("Child", 5);
childInstance.sayHello(); // Output: "Hello, my name is Child"
childInstance.sayAge();   // Output: "I am 5 years old"
```