JavaScript Prototypes:

Introduction to Prototypes

In JavaScript, every object has a prototype. A prototype is also an object, which acts as a template object that other objects inherit properties and methods from. This inheritance mechanism is the foundation of JavaScript's prototype-based programming model.

Key Concepts

1. **Prototype Chain**
2. **Constructor Functions and Prototypes**
3. **Inheritance with Prototypes**

1. Prototype Chain

When you try to access a property or method on an object, JavaScript first looks for the property or method on the object itself. If it does not find it, it looks at the object's prototype. This process continues up the chain until the property or method is found or the end of the chain is reached (typically Object.prototype).

```
let person1 ={
   name:"Hema"
}
console.log(person1.hasOwnProperty("name"));
```

2. Constructor Functions and Prototypes

Constructor functions are special functions used to create objects. Each constructor function has a prototype property, which is shared among all instances created by that constructor.

```
function Person(name) {
   this.name = name;
}
Person.prototype.greet = function() {
   return `Hello, my name is ${this.name}`;
};
const alice = new Person('Hema');
console.log(alice.greet()); // Output: Hello, my name is Hema
```

3. Inheritance with Prototypes

Inheritance in JavaScript is prototype-based. You can set up inheritance by setting an object's prototype to another object's prototype.

```

function Person(firstName,lastName){
   this.firstName = firstName;
   this.lastName = lastName;
```

```
}
let person3 = new Person("Mahesh", "coding");
let person4 = new Person("Hema", "coding");

Person.prototype.city = "HYD"
Person.prototype.fullName = function(){
   return `My Name is ${this.firstName} and last name ${this.lastName}`
}

function Child(dateOfBirth){
   // this.dateOfBirth = dateOfBirth;
}
Child.prototype.dateOfBirth = 11
let child1 = new Child(10);

Child.prototype.eating = function(){
   return `Eating food`;
}

Person.prototype.__proto__ = Child.prototype;
console.log(child1)
console.log(child1.eating())

console.log(person3)
console.log(person3.dateOfBirth)
console.log(person3.eating())
```

Interview Questions:

Question 1: What is a prototype in JavaScript?

- **Answer:** A prototype is an object from which other objects inherit properties and methods. Every JavaScript object has a prototype, and when a property or method is not found on an object, JavaScript looks up the prototype chain to find it.

Question 2: How does prototype inheritance work in JavaScript?

- **Answer:** Prototype inheritance allows an object to inherit properties and methods from another object. When a property or method is accessed on an object, JavaScript first looks at the object itself. If it does not find the property or method there, it looks at the object's prototype, and this process continues up the prototype chain until the property or method is found or the end of the chain is reached.

Question 3: What is the __proto__ property in JavaScript?

- **Answer:** The __proto__ property is a reference to the prototype of an object. It is used to access or modify the prototype of an object. However, it is recommended to

use Object.getPrototypeOf and Object.setPrototypeOf for interacting with an object's prototype.

Question 4: What is the difference between __proto__ and prototype?

- **Answer:** __proto__ is a property of an object that points to its prototype, while prototype is a property of a constructor function that is used to set the prototype of instances created by that constructor.

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

const john = new Person('Hema');
console.log(john.__proto__ === Person.prototype); // true
console.log(Person.prototype.constructor === Person); // true
```

Question 5: How do you set up inheritance between two constructor functions in JavaScript?

- **Answer:** You set up inheritance between two constructor functions by setting the prototype of the child constructor function to an object created from the parent constructor function's prototype and then setting the constructor property back to the child constructor function.

```
function Person(firstName,lastName){
  this.firstName = firstName;
  this.lastName = lastName;
  }
  let person3 = new Person("Mahesh", "coding");
  let person4 = new Person("Hema", "coding");

  Person.prototype.city = "HYD"
  Person.prototype.fullName = function(){
    return `My Name is ${this.firstName} and last name ${this.lastName}`
  }

  function Child(dateOfBirth){
    // this.dateOfBirth = dateOfBirth;
  }
  Child.prototype.dateOfBirth = 11
  let child1 = new Child(10);

  Child.prototype.eating = function(){
    return `Eating food`;
  }

  Person.prototype.__proto__ = Child.prototype;
  console.log(child1)
  console.log(child1.eating())
```

```
console.log(person3)
console.log(person3.dateOfBirth)
console.log(person3.eating())
```