JAVASCRIPT DAY 19th IN DRIVE

Cheat Sheet of Day 19

1. **Prototype**

- **Definition**: A prototype is a property of constructor functions in JavaScript, used to define methods and properties shared across all instances created by that constructor.

Purpose:

- Shares properties and methods efficiently among instances.
- Reduces memory usage by avoiding duplication.
- **Key Insight**: Modifying a constructor's prototype dynamically updates behavior for all its instances.

2. **Prototype Chaining**

- **Definition**: A mechanism where objects inherit properties and methods from other objects by following the chain of their __proto__ links.

Purpose:

- Enables inheritance, facilitating code reuse.
- The chain ends at null, which represents the top of the inheritance hierarchy.
- **Key Insight**: If a property/method is not found on an object, JavaScript looks up the prototype chain until it's found or reaches null.

3. IIFE (Immediately Invoked Function Expressions)

- **Definition**: A self-executing function that runs immediately upon definition.

Purpose:

- Prevents global scope pollution.
- Encapsulates logic and creates private variables.
- **Key Insight**: Ideal for initialization code or avoiding naming conflicts in shared environments.

4. **Generator Functions**

- **Definition**: Functions that can pause execution at yield and resume later, enabling incremental computation.
- **Purpose**:
- Handles large datasets or infinite sequences gracefully.
- Simplifies asynchronous programming.
- **Key Insight**: Generators return iterators, enabling controlled execution through .next().

5. **Higher-Order Functions**

- **Definition**: Functions that accept other functions as arguments or return functions.

Purpose:

- Promote functional programming practices.
- Simplify operations like filtering, mapping, and reducing arrays.
- **Key Insight**: Functions like map(), filter(), and reduce() enhance code modularity and readability.

6. Differences Between **prototype** and **__proto__**

- **prototype:** A property of constructor functions, defining shared behaviors for their instances.
- **__proto__:** A property of all objects that links them to their prototype.
- Key Insight: The prototype is used during instance creation, while __proto__ represents the actual inheritance link.

CODING QUESTIONS OF DAY 19

Assignment Coding Solutions

1) Person Constructor with a Greeting Method:

```
function Person(name, age) {
   this.name = name;
   this.age = age;
}
Person.prototype.greet = function() {
   return `Hello, my name is ${this.name} and I am ${this.age} years old.`;
};
const person = new Person("Alice", 30);
console.log(person.greet()); // Output: Hello, my name is Alice and I am 30 years old.
```

2) Car Constructor with Remaining Fuel Method:

```
function Car(make, fuelCapacity, fuelEfficiency) { // fuelEfficiency in km per liter
   this.make = make;
   this.fuelCapacity = fuelCapacity; // in liters
   this.fuel = fuelCapacity;
   this.fuelEfficiency = fuelEfficiency;
}

Car.prototype.drive = function(distance) {
   const fuelNeeded = distance / this.fuelEfficiency;
   if (fuelNeeded > this.fuel) {
      console.log("Not enough fuel for this trip!");
   } else {
      this.fuel -= fuelNeeded;
```

```
        console.log(`Remaining fuel: ${this.fuel.toFixed(2)} liters.`);
    }
};
const myCar = new Car("Toyota", 50, 15);
myCar.drive(100); // Remaining fuel: 43.33 liters.
```

3) Add Method to Existing Constructor Without Affecting Instances:

```
function Book(title, author) {
    this.title = title;
    this.author = author;
}

const book1 = new Book("1984", "George Orwell");
Book.prototype.getDetails = function() {
    return `${this.title} by ${this.author}`;
};

console.log(book1.getDetails()); // Output: 1984 by George Orwell
```

Prototype Chaining

4) Animal and Dog Prototype Chain:

```
function Animal(name) {
    this.name = name;
}
Animal.prototype.eat = function() {
    return `${this.name} is eating.`;
};
function Dog(name, breed) {
    Animal.call(this, name);
    this.breed = breed;
}
Dog.prototype = Object.create(Animal.prototype);
Dog.prototype.constructor = Dog;
Dog.prototype.bark = function() {
    return `${this.name} is barking.`;
};
const myDog = new Dog("Buddy", "Golden Retriever");
console.log(myDog.eat()); // Output: Buddy is eating.
console.log(myDog.bark()); // Output: Buddy is barking.
```

5) Vehicle -> Car -> ElectricCar Chain:

```
function Vehicle(type) {
    this.type = type;
}
Vehicle.prototype.start = function() {
    return `${this.type} is starting.`;
};
function Car(type, brand) {
    Vehicle.call(this, type);
    this.brand = brand;
}
Car.prototype = Object.create(Vehicle.prototype);
Car.prototype.constructor = Car;
Car.prototype.drive = function() {
    return `${this.brand} is driving.`;
};
function ElectricCar(type, brand, battery) {
    Car.call(this, type, brand);
    this.battery = battery;
}
ElectricCar.prototype = Object.create(Car.prototype);
ElectricCar.prototype.constructor = ElectricCar;
ElectricCar.prototype.charge = function() {
    return `${this.brand} is charging.`;
};
```

```
const tesla = new ElectricCar("Car", "Tesla", "100 kWh");
console.log(tesla.start()); // Output: Car is starting.
console.log(tesla.drive()); // Output: Tesla is driving.
console.log(tesla.charge()); // Output: Tesla is charging.
```

6) Using Object.create():

```
const animal = {
   eat() {
      return "Animal is eating.";
   }
};

const dog = Object.create(animal);
dog.bark = function() {
   return "Dog is barking.";
};

console.log(dog.eat()); // Output: Animal is eating.
console.log(dog.bark()); // Output: Dog is barking.
```

<u>IIFE</u>

1) Counter in IIFE:

```
(function() {
   let counter = 0;
   const interval = setInterval(() => {
      console.log(counter++);
      if (counter > 5) clearInterval(interval);
   }, 1000);
})();
```

2) Factorial in IIFE:

```
(function(num) {
   let factorial = 1;
   for (let i = 1; i <= num; i++) {
      factorial *= i;
```

```
    }
    console.log(`Factorial of ${num} is ${factorial}`);
})(5);
```

3) Private Variable in IIFE:

```
const counterModule = (function() {
    let count = 0;
    return {
        increment() {
            count++;
        },
        getValue() {
            return count;
        }
    };
})();

counterModule.increment();
console.log(counterModule.getValue()); // Output: 1
```

4) Reverse String in IIFE:

```
(function(str) {
    console.log(str.split("").reverse().join(""));
})("JavaScript");
```

Generator Functions

1) Fibonacci Generator:

```
function* fibonacci(n) {
    let [a, b] = [0, 1];
    for (let i = 0; i < n; i++) {
        yield a;
        [a, b] = [b, a + b];
    }
}
```

```
const fib = fibonacci(10);
for (let num of fib) {
    console.log(num);
}
```

2) Range Generator:

```
function* range(start, end) {
    for (let i = start; i <= end; i++) {
        yield i;
    }
}

const nums = range(1, 5);
console.log([...nums]); // Output: [1, 2, 3, 4, 5]
```

3) Odd Numbers Generator:

```
function* oddNumbers() {
    let num = 1;
    for (let i = 0; i < 10; i++) {
        yield num;
        num += 2;
    }
}

for (let odd of oddNumbers()) {
    console.log(odd);
}
```

4) Array Elements Generator:

```
function* arrayElements(arr) {
    for (let item of arr) {
        yield item;
    }
}
```

```
const gen = arrayElements(["a", "b", "c"]);
for (let val of gen) {
   console.log(val);
}
```

5) Traffic Light System:

```
function* trafficLight() {
   while (true) {
      yield "Red";
      yield "Yellow";
      yield "Green";
   }
}

const light = trafficLight();
setInterval(() => {
   console.log(light.next().value);
}, 1000);
```

FAQ’S OF DAY 19th

Prototypes

1. **What is a prototype in JavaScript?**

   A prototype is an object from which other objects inherit properties and methods.

2. **Why do we use prototypes?**

   To share methods and properties across multiple objects, saving memory and ensuring efficiency.

3. **Can we add methods to an object’s prototype after it is created?**

   Yes, methods or properties can be added anytime.

4. **Example: Adding a method to a prototype**

```
function Person(name) {
this.name = name;
}
Person.prototype.sayHello = function() {
return `Hi, I'm ${this.name}`;
};
```

Prototype Chaining

5. **What is prototype chaining?**

   A mechanism where objects inherit properties and methods from other objects through their prototypes.

6. **Why is prototype chaining useful?**

   It enables inheritance, allowing objects to use methods and properties from other objects.

7. **Does the chain have an end?**

   Yes, it ends when there is no __proto__, typically at Object.prototype.

8. **Example: How prototype chaining works**

```
function Animal() {
  this.alive = true;
}
Animal.prototype.eat = function() {
 console.log("Eating...");
};
function Dog(name) {
 this.name = name;
}
Dog.prototype = new Animal();
const dog = new Dog("Buddy");
console.log(dog.alive); // true
```

IIFE (Immediately Invoked Function Expressions)

9. **What is an IIFE?**

   A function that executes immediately after being defined.

10. **Why do we use IIFE?**

    To avoid polluting the global scope and to create private variables.

11. **Can an IIFE have parameters?**

    Yes, arguments can be passed to an IIFE.

12. **Example: How IIFE works**

```
(function(name) {
 console.log(`Hello, ${name}!`);
```

13. })("ExcelR");

Generator Functions

13. **What is a generator function?**

    A function that can pause and resume execution using the yield keyword.

14. **Why are generator functions useful?**

    They handle large or infinite data efficiently and simplify asynchronous code.

15. **How do you create a generator function?**

    Use the function* syntax.

16. **Example: How a generator function works**

```
function* numbers() {
yield 1;
yield 2;
yield 3;
}
const gen = numbers();
console.log(gen.next().value); // 1
console.log(gen.next().value); // 2
```

Higher-Order Functions

17. **What is a higher-order function?**

    A function that takes another function as an argument or returns a function.

18. **Why are higher-order functions important?**

    They make code reusable, concise, and flexible for array operations like filtering, mapping, and reducing.

19. **What are common examples of higher-order functions?**

    map(), filter(), reduce(), and event listeners.

20. **Example: How a higher-order function works**

    const numbers = [1, 2, 3, 4];

21. const squares = numbers.map(num => num * num);
22. console.log(squares); // [1, 4, 9, 16]

Differences Between prototype and __proto__

Feature	**prototype**	**__proto__**
Definition	A property of a function for attaching shared methods.	A property of an object pointing to its prototype.
Type	Available on functions only.	Available on all objects.
Purpose	Defines methods for all instances of a constructor.	Accesses or modifies an object's prototype chain.
Usage	Used to set up inheritance for constructor-created objects.	Used to traverse the prototype chain.
Default Value	Empty object ({}).	Prototype of the constructor.
Who Has It?	Only constructor functions.	All objects.
Access Level	Modifies prototype for future instances.	Inspects or modifies existing prototype chain.

Examples

1. **Using prototype:**

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

2. **Using __proto__:**

```
const obj = {};
const prototypeObj = {
sharedMethod: function() {
 return "Shared method";
}
};
obj.__proto__ = prototypeObj;
console.log(obj.sharedMethod()); // Shared method
```

Key Takeaways

- **prototype** is for defining shared methods for constructor-created objects.
- **__proto__** is for accessing or modifying an individual object's prototype chain.

MCQ'S OF DAY 19

Prototype

1. **What does the prototype property belong to?**

   Answer: 2. Functions
2. **Why do we use prototypes in JavaScript?**

   Answer: 2. To share properties and methods across objects
3. **What is the default value of the prototype property in a constructor function?**

   Answer: 3. An empty object {}
4. **How do you add a method to a prototype?**

   Answer: 2. function.prototype.method = function() { }
5. **Can the prototype property of a function be modified after the function is created?**

   Answer: 1. Yes

Prototype Chaining

6. **What does prototype chaining enable in JavaScript?**

   Answer: 2. Inheritance
7. **What is the last object in the prototype chain?**

   Answer: 2. null
8. **Which of the following is not true about prototype chaining?**

   Answer: 4. It automatically removes unused methods
9. **How do you access an object's prototype in modern JavaScript?**

   Answer: 3. Object.getPrototypeOf(object)
10. **What happens if a property is not found in an object's prototype chain?**

    Answer: 4. It returns undefined

IIFE (Immediately Invoked Function Expression)

11. **What does IIFE stand for?**

    Answer: 2. Immediately Invoked Function Expression

12. **Why is an IIFE useful?**

    Answer: 3. To avoid polluting the global scope

13. **How do you create an IIFE?**

    Answer: 2. (function() { })();

14. **Can an IIFE have parameters?**

    Answer: 1. Yes

15. **What does an IIFE return if there is no return statement?**

    Answer: 1. undefined

Generator Functions

16. **Which keyword is used to create a generator function?**

    Answer: 3. function*

17. **What does the yield keyword do in a generator function?**

    Answer: 2. Returns a value and pauses execution

18. **How do you create an instance of a generator?**

    Answer: 2. generatorFunction()

19. **What does gen.next() return?**

    Answer: 3. An object with value and done

20. **What happens when all yield statements in a generator are exhausted?**

    Answer: 3. The done property is set to true

Higher-Order Functions

21. **What is a higher-order function?**

    Answer: 1. A function that takes or returns another function

22. **Which of the following is not a higher-order function?**

    Answer: 4. console.log()

23. **What is the purpose of map()?**

    Answer: 1. To iterate over an array and modify each element

24. **What does filter() return?**

    Answer: 2. A new array with filtered elements

25. **Which method combines all elements of an array into a single value?**

    Answer: 3. reduce()

General Questions

26. **Can objects inherit from multiple prototypes in JavaScript?**

    Answer: 2. No

27. **What is the difference between prototype and proto?**

    Answer: 2. prototype is for functions, and __proto__ is for objects

28. **Which method is used to set an object's prototype?**

    Answer: 2. Object.setPrototypeOf()

29. **What is the result of calling gen.return() in a generator?**

    Answer: 1. Stops the generator and returns a value

30. **Can higher-order functions work with asynchronous callbacks?**

    Answer: 1. Yes