```
// Problem 1
// Create an object constructor Person that takes name and age as parameters and
initializes them. Also, add
// a method sayHello to greet the person.


// Define the Person constructor
function Person(name , age){
    this.name = name;
    this.age = age;
}
// Add the sayHello method to the Person prototype
Person.prototype.sayHello = function(){
    console.log('Hello, my name is ' + this.name + ' and I am ' + this.age + '
years old.');
}

let person1 = new Person("Jayant", 24)
person1.sayHello()

let person2 = new Person("Kirti", 21)
person2.sayHello()
```

```
// Problem 2
// Create a constructor Employee that inherits from the Person constructor of
problem 1. Add an additional
// property designation and a method getDetails to display the employee details.
// Define the Person constructor
function Person(name, age) {
    this.name = name;
    this.age = age;
}

// Method to greet the person
Person.prototype.sayHello = function() {
    console.log("Hello, my name is " + this.name + " and I am " + this.age + "
years old.");
};

// Define the Employee constructor that inherits from Person
function Employee(name, age, designation) {
    // Call the Person constructor to initialize name and age
    Person.call(this, name, age);
    this.designation = designation;
}

// Link the prototype of Employee to an instance of Person
Employee.prototype = Object.create(Person.prototype);

// Set the constructor property of Employee to refer back to itself
Employee.prototype.constructor = Employee;

// Add a method to display employee details
Employee.prototype.getDetails = function() {
    console.log("Name: " + this.name + ", Age: " + this.age + ", Designation: " +
this.designation);
};

// Creating an instance of Employee
var employee1 = new Employee("Jayant", 24, "Software Engineer");

// Greeting the employee
employee1.sayHello(); // Output: Hello, my name is Alice and I am 25 years old.

// Displaying employee details
employee1.getDetails(); // Output: Name: Alice, Age: 25, Designation: Software
Engineer
```

```
// Problem 3
// Create an object Calculator with methods add, subtract, multiply, and divide. Demonstrate the usage of this
// within these methods such that method chaining of add, subtract, multiply and divide is possible.


// Define the Calculator object
var Calculator = {
    // Method to add two numbers
    add: function(x) {
        this.result += x;
        return this; // Return the Calculator object for method chaining
    },
    // Method to subtract a number from the result
    subtract: function(x) {
        this.result -= x;
        return this; // Return the Calculator object for method chaining
    },
    // Method to multiply the result by a number
    multiply: function(x) {
        this.result *= x;
        return this; // Return the Calculator object for method chaining
    },
    // Method to divide the result by a number
    divide: function(x) {
        this.result /= x;
        return this; // Return the Calculator object for method chaining
    },
    // Method to get the current result
    getResult: function() {
        return this.result;
    }
};

// Example usage of the Calculator object
var result = Calculator.add(5).multiply(2).subtract(3).divide(2).getResult();
console.log("Result:", result); // Output: Result: 7
```

```
// Problem 4
// Define a base class Shape with a method draw. Create two subclasses Circle and
Rectangle that override
// the draw method. Demonstrate polymorphism using instances of these classes.

// Define the base class Shape
function Shape() {}

// Method draw for Shape (to be overridden by subclasses)
Shape.prototype.draw = function() {
    console.log("Drawing a generic shape");
};

// Define the subclass Circle
function Circle(radius) {
    this.radius = radius;
}
// Circle inherits from Shape
Circle.prototype = Object.create(Shape.prototype);
Circle.prototype.constructor = Circle;

// Override the draw method for Circle
Circle.prototype.draw = function() {
    console.log("Drawing a circle with radius " + this.radius);
};
// Define the subclass Rectangle
function Rectangle(width, height) {
    this.width = width;
    this.height = height;
}
// Rectangle inherits from Shape
Rectangle.prototype = Object.create(Shape.prototype);
Rectangle.prototype.constructor = Rectangle;
// Override the draw method for Rectangle
Rectangle.prototype.draw = function() {
    console.log("Drawing a rectangle with width " + this.width + " and height " +
this.height);
};

// Demonstrate polymorphism
var shapes = [new Circle(5), new Rectangle(3, 4)];
shapes.forEach(function(shape) {
    shape.draw(); // This will call the appropriate draw method depending on the
actual object type
});
```

```
// Problem 5
// Create a simple polyfill for the Array.includes method by the name of
customIncludes.

// Define the customIncludes method if it doesn't exist
if (!Array.prototype.customIncludes) {
    Array.prototype.customIncludes = function(searchElement, fromIndex) {
        // Default value for fromIndex
        var startIndex = fromIndex || 0;

        // Loop through the array starting from startIndex
        for (var i = startIndex; i < this.length; i++) {
            // If searchElement is found, return true
            if (this[i] === searchElement) {
                return true;
            }
        }
        // If searchElement is not found, return false
        return false;
    };
}

// Example usage
var array = [1, 2, 3, 4, 5];
console.log(array.customIncludes(3)); // Output: true
console.log(array.customIncludes(6)); // Output: false
```