Problem 1: Person Constructor with sayHello Method

```
// Problem 1: Create an object constructor Person
function Person(name, age) {
  this.name = name;
  this.age = age;

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

// Create an instance of Person
const person1 = new Person('Alice', 30);
person1.sayHello();  // Output: Hello, my name is Alice and I am 30 years old.
```

Problem 2: Employee Constructor Inheriting from Person

```
// Problem 2: Create an Employee constructor that inherits from Person
function Employee(name, age, designation) {
  Person.call(this, name, age);  // Call the Person constructor

  this.designation = designation;

  // Method to display employee details
  this.getDetails = function() {
    console.log(`Employee Name: ${this.name}, Age: ${this.age}, Designation: ${this.designation}`);
  };
}

// Set Employee prototype to inherit from Person
```

```
Employee.prototype = Object.create(Person.prototype);
Employee.prototype.constructor = Employee;

// Create an instance of Employee
const employee1 = new Employee('Bob', 25, 'Software Engineer');
employee1.sayHello();  // Inherited method from Person
employee1.getDetails();  // Output: Employee Name: Bob, Age: 25, Designation: Software Engineer
```

Problem 3: Calculator with Method Chaining

```
// Problem 3: Calculator with method chaining
function Calculator() {
  this.result = 0;

  this.add = function(num) {
    this.result += num;
    return this;  // Return the current instance to allow method chaining
  };

  this.subtract = function(num) {
    this.result -= num;
    return this;  // Return the current instance to allow method chaining
  };

  this.multiply = function(num) {
    this.result *= num;
    return this;  // Return the current instance to allow method chaining
  };

  this.divide = function(num) {
    if (num !== 0) {
      this.result /= num;
```

```
    } else {
      console.log('Cannot divide by zero');
    }
    return this;  // Return the current instance to allow method chaining
  };

  this.getResult = function() {
    console.log('Current Result: ', this.result);
    return this.result;
  };
}

// Demonstrating method chaining
const calculator = new Calculator();
calculator.add(5).subtract(2).multiply(3).divide(2).getResult();  // Output: Current Result: 4.5
```

Problem 4: Shape Class with Polymorphism

```
// Problem 4: Shape class with polymorphism
class Shape {
  draw() {
    console.log('Drawing a shape');
  }
}

class Circle extends Shape {
  constructor(radius) {
    super();  // Call the base class constructor
    this.radius = radius;
  }

  draw() {
```

```
    console.log(`Drawing a circle with radius ${this.radius}`);
  }
}

class Rectangle extends Shape {
  constructor(width, height) {
    super();  // Call the base class constructor
    this.width = width;
    this.height = height;
  }

  draw() {
    console.log(`Drawing a rectangle with width ${this.width} and height ${this.height}`);
  }
}

// Demonstrating polymorphism
const circle = new Circle(5);
const rectangle = new Rectangle(10, 20);

circle.draw();      // Output: Drawing a circle with radius 5
rectangle.draw();   // Output: Drawing a rectangle with width 10 and height 20
```

Problem 5: Polyfill for Array.includes (customIncludes)

```
// Problem 5: Simple polyfill for Array.includes
Array.prototype.customIncludes = function(element) {
  for (let i = 0; i < this.length; i++) {
    if (this[i] === element) {
      return true;
    }
  }
```

```
  return false;
};

// Test the customIncludes method
const arr = [1, 2, 3, 4, 5];
console.log(arr.customIncludes(3));  // Output: true
console.log(arr.customIncludes(6));  // Output: false
```