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.**

```
// Constructor function for 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.`);
  };
}

// Example usage:
const person1 = new Person('Alice', 30);
const person2 = new Person('Bob', 25);

person1.sayHello(); // Output: Hello, my name is Alice and I am 30 years old.
person2.sayHello(); // Output: Hello, my name is Bob and I am 25 years old.
```

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.**

```
// Person constructor function
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.`);
  };
}

// Employee constructor function that inherits from Person
function Employee(name, age, designation) {
  // Call the Person constructor with 'this' to inherit properties
  Person.call(this, name, age);

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

// Set up inheritance
Employee.prototype = Object.create(Person.prototype);
Employee.prototype.constructor = Employee;

// Example usage:
const employee1 = new Employee('Alice', 30, 'Software Engineer');
const employee2 = new Employee('Bob', 25, 'Product Manager');

employee1.sayHello(); // Output: Hello, my name is Alice and I am 30 years old.
employee1.getDetails(); // Output: Name: Alice, Age: 30, Designation: Software Engineer

employee2.sayHello(); // Output: Hello, my name is Bob and I am 25 years old.
employee2.getDetails(); // Output: Name: Bob, Age: 25, Designation: Product Manager
```

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.**

```
// Calculator constructor function
function Calculator() {
  this.value = 0;

  // Method to add a number
  this.add = function(num) {
    this.value += num;
    return this; // Return the Calculator object for chaining
  };

  // Method to subtract a number
  this.subtract = function(num) {
    this.value -= num;
    return this; // Return the Calculator object for chaining
  };

  // Method to multiply by a number
  this.multiply = function(num) {
    this.value *= num;
    return this; // Return the Calculator object for chaining
  };

  // Method to divide by a number
  this.divide = function(num) {
    if (num !== 0) {
      this.value /= num;
    } else {
      console.log("Error: Division by zero");
    }
    return this; // Return the Calculator object for chaining
  };

  // Method to get the current value
  this.getResult = function() {
    return this.value;
  };
}

// Example usage:
const calc = new Calculator();

const result = calc.add(10).subtract(5).multiply(3).divide(2).getResult();
console.log(result); // Output: 7.5

// Another example:
const anotherResult = calc.add(5).multiply(2).divide(0).getResult(); // Division by zero error
console.log(anotherResult); // Output: Error: Division by zero
```

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.**

```
// Base class Shape
class Shape {
  draw() {
    console.log("Drawing a generic shape");
  }
}

// Subclass Circle
class Circle extends Shape {
  draw() {
    console.log("Drawing a circle");
  }
}

// Subclass Rectangle
class Rectangle extends Shape {
  draw() {
    console.log("Drawing a rectangle");
  }
}

// Demonstrate polymorphism
const shapes = [new Circle(), new Rectangle()];

shapes.forEach(shape => {
  shape.draw();
});
```

5. **Create a simple polyfill for the Array.includes method by the name of customIncludes.**

```
// Check if Array.prototype.includes already exists
if (!Array.prototype.customIncludes) {
  // Define customIncludes method
  Array.prototype.customIncludes = function(searchElement, fromIndex) {
    if (this == null) {
      throw new TypeError('"this" is null or not defined');
    }

    // Convert array-like object to array
    var O = Object(this);
    var len = O.length >>> 0;

    // Default start index
    var fromIdx = fromIndex || 0;

    // Adjust start index if negative
    if (fromIdx < 0) {
      fromIdx = Math.max(len + fromIdx, 0);
    }

    // Iterate through the array
    for (var i = fromIdx; i < len; i++) {
      // Check if element exists
      if (O[i] === searchElement) {
        return true;
      }
    }

    return false;
  };
}

// Example usage:
const array = [1, 2, 3, 4, 5];

console.log(array.customIncludes(3)); // Output: true
console.log(array.customIncludes(6)); // Output: false
```