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.

```
function Person(name, age) {
  this.name = name;
  this.age = age;

  this.sayHello = function () {
    console.log("Hello, my name is " + this.name + " and I am " + this.age + " years old.");
  };
}

const person1 = new Person("Alice", 30);
person1.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.

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

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

function Employee(name, age, designation) {
  Person.call(this, name, age);
  this.designation = designation;
}

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

Employee.prototype.constructor = Employee;

Employee.prototype.getDetails = function () {
  console.log(
    "Name: " +
      this.name +
      ", Age: " +
      this.age +
      ", Designation: " +
      this.designation
  );
};

const employee1 = new Employee("Bob", 25, "Developer");
employee1.sayHello();
employee1.getDetails();
```

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.

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

  this.add = function (number) {
    this.value += number;
    return this;
  };

  this.subtract = function (number) {
    this.value -= number;
    return this;
  };

  this.multiply = function (number) {
    this.value *= number;
    return this;
  };

  this.divide = function (number) {
    if (number !== 0) {
      this.value /= number;
      return this;
    } else {
      throw new Error("Cannot divide by zero");
    }
  };
}

const calc = new Calculator();
calc.add(10).subtract(3).multiply(5).divide(2);
console.log(calc.value);
```

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.

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

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

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

function drawShape(shape) {
  shape.draw();
}

const circle = new Circle();
const rectangle = new Rectangle();

drawShape(circle);
drawShape(rectangle);
```

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

```
if (!Array.prototype.customIncludes) {
  Array.prototype.customIncludes = function (searchElement, fromIndex) {
    var start = fromIndex || 0;

    for(var i=start; i<this.length; i++){
        if(this[i]===searchElement){
            return true;
        }
    }
    return false;
  };
}

const array = [1, 2, 3, 4, 5];
console.log(array.customIncludes(3));
console.log(array.customIncludes(6));
```