# Object Copy (Shallow)

```
// Shallow Copy using Spread operator
const original = { a: 1, b: 2 };
const copy = { ...original };

copy.a = 3; // Modify the 'a' property in the copy

console.log(original); // Output: { a: 1, b: 2 }
console.log(copy); // Output: { a: 3, b: 2 }
```

Creates a shallow copy by spreading the properties of the original object into a new object.

# Object Copy (Deep using JSON)

```
// Deep Copy using JSON Serialization
const original = { a: 1, b: { c: 2 } };
const deepCopy = JSON.parse(JSON.stringify(original));

console.log(deepCopy); // Output: { a: 1, b: { c: 2 } }
```

# Object Copy (Deep using Recursive Copy)

```
// Deep Copy using Recursion
function deepCopy(obj) {
  if (obj === null || typeof obj !== 'object') {
    return obj;
  }
  const copy = Array.isArray(obj) ? [] : {};
  for (let key in obj) {
    if (obj.hasOwnProperty(key)) {
      copy[key] = deepCopy(obj[key]);
    }
  }
  return copy;
}

const original = { a: 1, b: { c: 2 } };
const deepCopyObj = deepCopy(original);

console.log(deepCopyObj); // Output: { a: 1, b: { c: 2 } }
```

# Practice Exercise

## Objects

1. Add function isIdenticalProduct to compare two product objects.
2. Write a function that takes two objects as input and returns true if they have identical contents using JSON serialization. Test your function with objects containing different data types and explain any limitations.
3. Create an object with nested objects. Write a function that performs a shallow copy of the object. Modify a property in the nested object of the copy and observe its effect on the original object.
4. Write a function that merges two objects. If the same property exists in both objects, use the value from the second object. Test this function with objects that have both overlapping and unique properties.

# De-structuring

```
let product = {
  company: 'Mango',
  itemName: 'Cotton striped t-shirt',
  price: 861
};
// Destructuring
let company = product.company
// is same as
let { company } = product;
```

```
// Property shorthand
let price = 861;
let product = {
  company: 'Mango',
  itemName: 'Cotton striped t-shirt',
  price: price
};
// is same as
let product1 = {
  company: 'Mango',
  itemName: 'Cotton striped t-shirt',
  price
};
```

```
// Method shorthand
let product = {
  company: 'Mango',
  itemName: 'Cotton striped t-shirt',
  displayPrice: function() {
    return `$${this.price.toFixed(2)}`;
  }
};
// is same as
let product1 = {
  company: 'Mango',
  itemName: 'Cotton striped t-shirt',
  displayPrice() {
    return `$${this.price.toFixed(2)}`;
  }
};
```

1. De-structuring: Extract properties from objects easily.
2. We can extract more than one property at once.
3. Shorthand Property: {message: message} simplifies to just message.
4. Shorthand Method: Define methods directly inside the object without the function keyword.

# Spread & Rest Operator

## (Spread)

```
// Array Expansion
const arr1 = [1, 2, 3];
const arr2 = [...arr1]; // [1, 2, 3]
// [1, 2, 3, 4, 5]
const arr3 = [...arr1, 4, 5];

// Object Expansion
const obj1 = { a: 1, b: 2 };
// { a: 1, b: 2, c: 3 }
const obj2 = { ...obj1, c: 3 };

// Function Arguments
function sum(a, b, c) {
    return a + b + c;
}
const numbers = [1, 2, 3];
console.log(sum(...numbers)); // 6
```

1. Represented by three dots (...), the spread operator is used to expand elements of an iterable (like an array or string) into individual elements.
2. Useful for copying arrays and objects without modifying the original.
3. Ensures immutability in functions where modification of inputs is not desired.

# Spread & Rest Operator

## (Rest)

```
1   // Function Parameters
2   function sum(...numbers) {
3     return numbers.reduce((acc, curr) => acc + curr, 0);
4   }
5   console.log(sum(1, 2, 3, 4)); // 10
6
7   // Array Destructuring
8   const [first, second, ...rest] = [1, 2, 3, 4, 5];
9   console.log(rest); // [3, 4, 5]
10
11  // Object destructuring
12  const { a, b, ...rest } = { a: 1, b: 2, c: 3, d: 4 };
13  console.log(rest); // { c: 3, d: 4 }
```

- Used to collect the remaining elements of an array after extracting some elements.
- Used to collect the remaining properties of an object after extracting some properties.

1. Represented by three dots (...), the rest operator is used to collect multiple elements into a single array or object.
2. Allows a function to accept an indefinite number of arguments as an array.

# Practice Exercise

## Objects

1. Given an object {message: 'good job', status: 'complete'}, use de-structuring to create two variables message and status.

# Callbacks

```
// Define a callback function
function greeting(name) {
  console.log('Hello, ' + name);
}

// Define a function that takes a callback
function processUserInput(callback) {
  var name = prompt('Please enter your name.');
  callback(name);
}

// Call the function with the callback
processUserInput(greeting);
```

1. A callback is a function passed as an argument to another function, which is then invoked inside the outer function to complete some kind of routine or action.
2. Usage: Callbacks are commonly used in asynchronous programming to execute code after an asynchronous operation has completed.

# Anonymous Functions as Values

```
// syntax
(function() {
    // function body
});

// Example as a callback
setTimeout(function() {
  console.log("This is anonymous");
}, 1000);


// Assigned to a variable
const add = function(a, b) {
    return a + b;
};
console.log(add(2, 3)); // Outputs: 5
```

1. Anonymous functions are functions without a name.
2. They are often used as arguments to other functions or assigned to a variable.
3. Useful for creating function scopes and avoiding global variables.