1. Recursion and stack:

Task 1: Implement a function to calculate the factorial of a number using recursion.

```
   function factorial(n) {
  if (n === 0 || n === 1) {
    return 1;
  }
  return n * factorial(n - 1);
}
console.log(factorial(5));           //Output: 120
```

Task 2: Write a recursive function to find the nth Fibonacci number.

```
function fibonacci(n) {
  if (n === 0) {
    return 0;
  }
  if (n === 1) {
    return 1;
  }
  return fibonacci(n - 1) + fibonacci(n - 2);
}

console.log(fibonacci(6));           // Output: 8
```

Task 3: Create a function to determine the total number of ways one can climb a staircase with 1, 2, or 3 steps at a time using recursion.

```
function climbStairs(n) {
  if (n === 0) {
    return 1;
  }
  if (n < 0) {
    return 0;
  }
  return climbStairs(n - 1) + climbStairs(n - 2) + climbStairs(n - 3);
}

console.log(climbStairs(4));         // Output: 7
```

Task 4: Write a recursive function to flatten a nested array structure.

```
function flattenArray(arr) {
  let result = [];
  arr.forEach(element => {
    if (Array.isArray(element)) {
      result = result.concat(flattenArray(element));
```

```
   } else {
      result.push(element);
    }
  });
  return result;
}
console.log(flattenArray([1, [2, [3, 4], 5], [6]]));        // Output: [1, 2, 3, 4, 5, 6]
```

Task 5: Implement the recursive Tower of Hanoi solution.

```
function towerOfHanoi(n, fromRod, toRod, auxRod) {
  if (n === 1) {
    console.log(`Move disk 1 from ${fromRod} to ${toRod}`);
    return;
  }
  towerOfHanoi(n - 1, fromRod, auxRod, toRod);
  console.log(`Move disk ${n} from ${fromRod} to ${toRod}`);
  towerOfHanoi(n - 1, auxRod, toRod, fromRod);
}

towerOfHanoi(3, 'A', 'C', 'B');


// Output:
// Move disk 1 from A to C
// Move disk 2 from A to B
// Move disk 1 from C to B
// Move disk 3 from A to C
// Move disk 1 from B to A
// Move disk 2 from B to C
// Move disk 1 from A to C
```

2. JSON and variable length arguments/spread syntax:

Task 1: Write a function that takes an arbitrary number of arguments and returns their sum.

```
      function sumAll(...numbers) {
    return numbers.reduce((sum, num) => sum + num, 0);
}
console.log(sumAll(1, 2, 3, 4));                    // Output: 10
```

Task 2: Modify a function to accept an array of numbers and return their sum using the spread syntax.

```
          function sumArray(numbers) {
    return sumAll(...numbers);
}
console.log(sumArray([1, 2, 3, 4]));                  // Output: 10
```

Task 3: Create a deep clone of an object using JSON methods.

```
     function deepClone(obj) {
  return JSON.parse(JSON.stringify(obj));
}
const original = { a: 1, b: { c: 2 } };
const clone = deepClone(original);
console.log(clone);                    // Output: { a: 1, b: { c: 2 } }
console.log(original === clone);            // Output: false
```

Task 4: Write a function that returns a new object, merging two provided objects using the spread syntax.

```
    function mergeObjects(obj1, obj2) {
  return { ...obj1, ...obj2 };
}
const objA = { a: 1, b: 2 };
const objB = { b: 3, c: 4 };
const merged = mergeObjects(objA, objB);
console.log(merged);                  // Output: { a: 1, b: 3, c: 4 }
```

Task 5: Serialize a JavaScript object into a JSON string and then parse it back into an object.

```
       function serializeAndParse(obj) {
  const jsonString = JSON.stringify(obj);
  return JSON.parse(jsonString);
}
const obj = { name: "John", age: 30 };
const newObj = serializeAndParse(obj);
console.log(newObj);               // Output: { name: "John", age: 30 }
```

3. Closure:

Task 1: Create a function that returns another function, capturing a local variable.

```
  import java.util.function.Supplier;

public class Task1 {
  public static Supplier<String> createFunction(String message) {
    return () -> "Captured message: " + message;
  }

  public static void main(String[] args) {
    Supplier<String> func = createFunction("Hello, World!");
    System.out.println(func.get());
  }
}                         // Captured message: Hello, World!
```

Task 2: Implement a basic counter function using closure, allowing incrementing and displaying the current count.

```
        import java.util.function.Supplier;

public class Task2 {
    public static Supplier<Integer> createCounter() {
        final int[] count = {0}; // Using an array to simulate mutable closure
        return () -> ++count[0];
    }

    public static void main(String[] args) {
        Supplier<Integer> counter = createCounter();
        System.out.println(counter.get()); // 1
        System.out.println(counter.get()); // 2
        System.out.println(counter.get()); // 3
    }
}
```

Task 3: Write a function to create multiple counters, each with its own separate count.

```
  import java.util.function.Supplier;

public class Task3 {
    public static Supplier<Integer> createCounter() {
        final int[] count = {0};
        return () -> ++count[0];
    }

    public static void main(String[] args) {
        Supplier<Integer> counter1 = createCounter();
        Supplier<Integer> counter2 = createCounter();

        System.out.println("Counter 1: " + counter1.get()); // 1
        System.out.println("Counter 1: " + counter1.get()); // 2

        System.out.println("Counter 2: " + counter2.get()); // 1
        System.out.println("Counter 2: " + counter2.get()); // 2
    }
}
```

Task 4: Use closures to create private variables within a function.

```
   public class Task4 {
    public static class Counter {
        private int count = 0;

        public int increment() {
```

```
      return ++count;
    }

    public int getCount() {
      return count;
    }
  }

  public static void main(String[] args) {
    Counter counter = new Counter();
    System.out.println(counter.increment()); // 1
    System.out.println(counter.increment()); // 2
    System.out.println("Current count: " + counter.getCount()); // 2
  }
}
```

Task 5: Build a function factory that generates functions based on some input using closures.

```
import java.util.function.Function;

public class Task5 {
  public static Function<Integer, Integer> createMultiplier(int factor) {
    return (value) -> value * factor;
  }

  public static void main(String[] args) {
    Function<Integer, Integer> doubleFunction = createMultiplier(2);
    Function<Integer, Integer> tripleFunction = createMultiplier(3);

    System.out.println("Double 5: " + doubleFunction.apply(5)); // 10
    System.out.println("Triple 5: " + tripleFunction.apply(5)); // 15
  }
}
```

4. Promise, Promises chaining:

Task 1: Create a new promise that resolves after a set number of seconds and returns a greeting.

```
 function delayedGreeting(seconds) {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve(`Hello after ${seconds} seconds`);
    }, seconds * 1000);
  });
}

delayedGreeting(3).then((message) => console.log(message));
```

Task 2: Fetch data from an API using promises, and then chain another promise to process this data.

```
function fetchData() {
  fetch('https://jsonplaceholder.typicode.com/posts/1')
    .then((response) => response.json())
    .then((data) => {
      console.log('Fetched Data:', data);
      return data.title.toUpperCase(); // Process data
    })
    .then((processedData) => {
      console.log('Processed Title:', processedData);
    })
    .catch((error) => console.error('Error fetching data:', error));
}

fetchData();
```

Task 3: Create a promise that either resolves or rejects based on a random number.

```
function randomPromise() {
  return new Promise((resolve, reject) => {
    const randomNumber = Math.random();
    if (randomNumber > 0.5) {
      resolve(`Success! Random number is ${randomNumber}`);
    } else {
      reject(`Failure! Random number is ${randomNumber}`);
    }
  });
}

randomPromise()
  .then((message) => console.log(message))
  .catch((error) => console.error(error));
```

Task 4: Use Promise.all to fetch multiple resources in parallel from an API.

```
    function fetchMultipleResources() {
  const urls = [
    'https://jsonplaceholder.typicode.com/posts/1',
    'https://jsonplaceholder.typicode.com/posts/2',
    'https://jsonplaceholder.typicode.com/posts/3'
  ];

  const fetchPromises = urls.map((url) => fetch(url).then((response) => response.json()));

  Promise.all(fetchPromises)
    .then((results) => {
      console.log('All Data Fetched:', results);
    })
    .catch((error) => console.error('Error fetching resources:', error));
}
```

```
fetchMultipleResources();
```

Task 5: Chain multiple promises to perform a series of asynchronous actions in sequence.

```
     function performSequentialActions() {
  new Promise((resolve) => {
    console.log('Step 1: Start');
    setTimeout(() => resolve('Step 2: Data from Step 1'), 2000);
  })
    .then((step2Data) => {
      console.log(step2Data);
      return new Promise((resolve) => {
        setTimeout(() => resolve('Step 3: Data from Step 2'), 2000);
      });
    })
    .then((step3Data) => {
      console.log(step3Data);
      return 'Step 4: Final Result';
    })
    .then((finalResult) => {
      console.log(finalResult);
    })
    .catch((error) => console.error('Error:', error));
}

performSequentialActions();
```

5. Async/await:

Task 1: Rewrite a promise-based function using async/await.

```
function delayedGreeting(seconds) {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve(`Hello after ${seconds} seconds`);
    }, seconds * 1000);
  });
}

async function asyncGreeting() {
  const message = await delayedGreeting(3);
  console.log(message);
}

asyncGreeting();
```

Task 2: Create an async function that fetches data from an API and processes it. async function fetchAndProcessData() {

```
  const response = await fetch('https://jsonplaceholder.typicode.com/posts/1');
  const data = await response.json();
  console.log('Fetched Data:', data);

  // Process the data (e.g., converting the title to uppercase)
  const processedData = data.title.toUpperCase();
  console.log('Processed Title:', processedData);
}

fetchAndProcessData();
```

Task 3: Implement error handling in an async function using try/catch.

```
  async function fetchWithErrorHandling() {
  try {
    const response = await fetch('https://jsonplaceholder.typicode.com/invalid-endpoint');
    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }
    const data = await response.json();
    console.log('Fetched Data:', data);
  } catch (error) {
    console.error('Error occurred:', error.message);
  }
}

fetchWithErrorHandling();
```

Task 4: Use async/await in combination with Promise.all.

```
  async function fetchMultipleResources() {
  const urls = [
    'https://jsonplaceholder.typicode.com/posts/1',
    'https://jsonplaceholder.typicode.com/posts/2',
    'https://jsonplaceholder.typicode.com/posts/3'
  ];

  try {
    const responses = await Promise.all(urls.map((url) => fetch(url)));
    const dataPromises = responses.map((response) => response.json());
    const results = await Promise.all(dataPromises);

    console.log('All Data Fetched:', results);
  } catch (error) {
    console.error('Error fetching resources:', error.message);
  }
}

fetchMultipleResources();
```

Task 5: Create an async function that waits for multiple asynchronous operations to complete before proceeding.

```
 async function waitForMultipleOperations() {
  const delayedTask = (taskName, seconds) =>
    new Promise((resolve) => {
      setTimeout(() => resolve(`${taskName} completed after ${seconds} seconds`), seconds
* 1000);
    });
```

```
   const task1 = delayedTask('Task 1', 2);
   const task2 = delayedTask('Task 2', 3);
   const task3 = delayedTask('Task 3', 1);

   const results = await Promise.all([task1, task2, task3]);
   console.log('All tasks completed:', results);
}

waitForMultipleOperations();
```

6. **Modules introduction, Export and Import:**

Task 1: Create a module that exports a function, a class, and a variable.

```
 // Exporting a function
export function greet(name) {
   return `Hello, ${name}!`;
}

// Exporting a class
export class Person {
   constructor(name, age) {
      this.name = name;
      this.age = age;
   }

   introduce() {
      return `Hi, I'm ${this.name} and I'm ${this.age} years old.`;
   }
}

// Exporting a variable
export const pi = 3.14159;
```

Task 2: Import the module in another JavaScript file and use the exported entities.

```
 import { greet, Person, pi } from './module.js';

// Using the function
console.log(greet('Alice')); // Output: Hello, Alice!

// Using the class
const person = new Person('Bob', 25);
console.log(person.introduce()); // Output: Hi, I'm Bob and I'm 25 years old.

// Using the variable
console.log(`The value of pi is ${pi}.`); // Output: The value of pi is 3.14159.
```

Task 3: Use named exports to export multiple functions from a module.

```
  export function add(a, b) {
   return a + b;
}

export function subtract(a, b) {
    return a - b;
}

export function multiply(a, b) {
    return a * b;
}
```

Task 4: Use named imports to import specific functions from a module.

```
  import { add, multiply } from './utils.js';

console.log(add(5, 3));       // Output: 8
console.log(multiply(4, 7));  // Output: 28
```

Task 5: Use default export and import for a primary function of a module.

```
    export default function logMessage(message) {
    console.log(`[LOG]: ${message}`);
}
```

7. Browser: DOM Basics:

Task 1: Select an HTML element by its ID and change its content using JavaScript.

HTML:

```
<div id="greeting">Hello!</div>

<script src="task1.js"></script>
```

JavaScript (task1.js):

```
document.getElementById('greeting').textContent = 'Hello, World!';
```

Task 2: Attach an event listener to a button, making it perform an action when clicked.

HTML:

```
<button id="myButton">Click Me</button>

<div id="output"></div>

<script src="task2.js"></script>
```

JavaScript :

```
document.getElementById('myButton').addEventListener('click', () => {

   document.getElementById('output').textContent = 'Button was clicked!';

});
```

Task 3: Create a new HTML element and append it to the DOM.

HTML:

```
<div id="container"></div>

<script src="task3.js"></script>
```

JavaScript:

```
const newElement = document.createElement('p');

newElement.textContent = 'This is a new paragraph!';

document.getElementById('container').appendChild(newElement);
```

Task 4: Implement a function to toggle the visibility of an element.

HTML:

```
<div id="toggleElement">This text can be toggled.</div>

<button id="toggleButton">Toggle Visibility</button>

<script src="task4.js"></script>
```

JavaScript:

```
document.getElementById('toggleButton').addEventListener('click', () => {
   const element = document.getElementById('toggleElement');
   if (element.style.display === 'none') {
      element.style.display = 'block';
   } else {
      element.style.display = 'none';
   }
});
```

Task 5: Use the DOM API to retrieve and modify the attributes of an element.

HTML:

```
<img id="myImage" src="example.jpg" alt="Example Image">

<button id="changeImage">Change Image Attributes</button>

<script src="task5.js"></script>
```

JavaScript:

```
document.getElementById('changeImage').addEventListener('click', () => {
   const img = document.getElementById('myImage');
   console.log('Current src:', img.getAttribute('src')); // Log current src
   img.setAttribute('src', 'new-image.jpg'); // Change the src
   img.setAttribute('alt', 'New Image'); // Change the alt attribute
});
```