String

- String is used to representing the sequence of characters.
- It can hold the data and manipulate the data which can be represented in the text form.
- We can perform different operations on strings like checking their length, also concatenate then by using +=, +, string operators.

Creating Strings

- There are two ways to create a string in javascript
  - Using String Literal

    A string can be created just by initializing with a string literal. There are three ways to write a string literal

    1. Inside single quotes(‘ ’)
    2. Inside double quotes (“”)
    3. Inside template literals or backticks(``)

Example:

```
let singleQuotes = 'string created with single quotes.';

let doubleQuotes = "string created with double quotes.";

let templateLiteral = `string created with template literals.`;

console.log(singleQuotes);

console.log(doubleQuotes);

console.log(templateLiteral);
```

Output:

```
string created with single quotes.

string created with double quotes.

string created with template literals
```

  - Using String Object

A string can also be created using a String Object

Example:

```
// Using the String object

const str = new String("Using string object");

console.log(str);
```

Output :

```
[String: 'Using string object']
```

Methods

1. slice()

extracts a part of the string based on the given starting-index and ending-index and returns a new string.

Example :

```
// Define a string variable

let A = 'Try hard Fail better';

// Use the slice() method to extract a substring

let b = A.slice(0, 3);

let c = A.slice(4,8);

let d = A.slice(9);

// Output the value of variable

console.log(b);

console.log(c);

console.log(d);
```

Output :

```
Try

Hard

Fail better
```

2. substring()

returns the part of the given string from the start index to the end index. Indexing starts from zero (0).

Example:

```
// Define a string variable

let str = "Mind, Power, Soul";

// Use the substring() method to extract a substring

let part = str.substring(6, 11);

// Output the value of variable

console.log(part);
```

Output:

```
Power
```

3. substr()

This method returns the specified number of characters from the specified index from the given string. It extracts a part of the original string.

Example:

```
// Define a string variable 'str'

let str = "Mind, Power, Soul";

// Use the substr() method to extract a substring f

let part = str.substr(6, 5);

// Output the value of variable

console.log(part);
```

Output:

```
Power
```

4. replace()

replaces a part of the given string with another string or a regular expression. The original string will remain unchanged.

Example :

```
// Define a string variable 'str'

let str = "Mind, Power, Soul";

// Use the replace() method to replace the substring

let part = str.replace("Power", "Space");

// Output the resulting string after replacement

console.log(part);
```

Output:

```
Mind, Space, Soul
```

5. replaceAll()

returns a new string after replacing all the matches of a string with a specified string or a regular expression. The original string is left unchanged after this operation.

Example :

```
// Define a string variable 'str'

let str = "Mind, Power, Power, Soul";

// Use the replaceAll() method to replace all occurrences

//of "Power" with "Space" in the string 'str'

let part = str.replaceAll("Power", "Space");

// Output the resulting string after replacement

console.log(part);
```

Output :

```
Mind, Space, Space, Soul
```

6. toUpperCase()

converts all the characters present in the String to upper case and returns a new String with all characters in upper case.

Example:

```
let str = 'abc';

// Convert the string 'geeks' to uppercase using the toUpperCase() method

console.log(str.toUpperCase());
```

Output:

```
ABC
```

7. toLowerCase()

converts all the characters present in the so lowercase and returns a new string with all the characters in lowercase.

Example :

```
let str='ABC';

// Convert the string 'geeks' to lowercase using the toLowerCase() method

console.log(str.toLowerCase());
```

Output:

```
abc
```

8. concat()

combines the text of two strings and returns a new combined or joined string. This method accepts one argument. The variable contains text in double quotes or single quotes.

Example :

```
let str1 = ‘KBFC';

let str2 = 'stands for Kerala Blasters Football Club';

console.log(str1.concat(str2));
```

Output:

```
KBFC stands for Kerala Blasters Football Club
```

9. trim()

is used to remove either white spaces from the given string. This method returns a new string with removed white spaces.This method doesn’t accept any parameter.

Example :

```
let str = 'KFC   ';

// with removed white spaces

let newStr = str.trim();

// Old length

console.log(str.length);

// New length

console.log(newStr.length)
```

Output:

```
6

3
```

10.trimStart()

removes whitespace from the beginning of a string. The value of the string is not modified in any manner, including any whitespace present after the string.

Example:

```
// Define a string variable
let str = "  Soul";
// Output the original value of the string
console.log(str);
// Use the trimStart() method to remove leading whitespace from the string 'str'
let part = str.trimStart();
// Output the resulting string after removing leading whitespace
console.log(part);
```

Output:

```
  Soul
Soul
```

11.trimEnd()

removes white space from the end of a string. The value of the string is not modified in any manner, including any white-space present before the string.

Example :

```
// Define a string variable
let str = "Soul  ";
// Output the original value of the string 'str'
console.log(str);
// Use the trimEnd() method to remove trailing whitespace from the string 'str'
let part = str.trimEnd();
// Output the resulting string after removing trailing whitespace
console.log(part);
```

Output:

```
Soul

Soul
```

12.padStart()

pad a string with another string until it reaches the given length. The padding is applied from the left end of the string.

Example :

```
// Define a string variable

let stone = "Soul";

// Use the padStart() method to add padding characters "Mind "

//to the beginning of the string 'stone'

stone = stone.padStart(9, "Mind ");

// Output the resulting string after padding

console.log(stone);
```

Output:

```
Mind Soul
```

13.padEnd()

pad a string with another string until it reaches the given length. The padding is applied from the right end of the string.

Example:

```
// Define a string variable

let stone = "Soul";

// Use the padEnd() method to add padding characters

//" Power" to the end of the string 'stone'

stone = stone.padEnd(10, " Power");

// Output the resulting string after padding

console.log(stone);
```

Output:

```
Soul Power
```

14.charAt()

returns the character at the specified index. String in JavaScript has zero-based indexing.

Example:

```
let str= 'Trees'

console.log(str.charAt(0));
```

Output:

```
T
```

15.charCodeAt()

returns a number that represents the Unicode value of the character at the specified index. This method accepts one argument.

Example:

```
let str=’Novavi’

console.log(str.charCodeAt(0))
```

Output:

```
78
```

16.split()

splits the string into an array of sub-strings. This method returns an array. This method accepts a single parameter character on which you want to split the string.

Example:

```
let str = 'soul-stone'

// Split string on '-'.

console.log(str.split('-'))
```

Output:

```
[ ‘soul’,’stone’ ]
```

Tasks

1. Write a function that removes any leading and trailing whitespace from a string and converts it to lowercase. Test this function with a string that has leading and trailing spaces and uppercase letters?

2. Write a function that takes a string and two indices, and returns a substring of the string from the first index to the second index. Ensure that the indices are within the bounds of the string.

3. Write a function that takes a string and a character, then replaces all occurrences of the character with another string. Make sure the original string remains unchanged.

4. Write a function that accepts a string and returns the character at the specified index. If the index is out of bounds, return undefined.

5. Write a function that adds padding to a string from both ends until it reaches a specified length. The padding string should be added from the left using a specified character.

6. Write a function that splits a string into an array of words. Test it with a string containing multiple words separated by spaces.

7. Write a function that capitalizes the first letter of each word in a string and returns the modified string.

8. Write a function that uses slice() to extract a portion of a string. The function should take a start and end index and return the corresponding substring.

9. Write a function that replaces the first occurrence of a given string with another string.

10. Write a function that returns the Unicode value of the character at a specific index.

Array Methods

An array in JavaScript is a data structure used to store multiple values in a single variable. It can hold various data types and allows for dynamic resizing. Elements are accessed by their index, starting from 0.

2 ways to create arrays

1. **Create using Literal**
   Creating an array using array literal involves using square brackets [] to define and initialize the array.

Example :

```
// Creating an Empty Array

let a = [];

console.log(a);

// Creating an Array and Initializing with Values

let b = [10, 20, 30];

console.log(b);
```

Output:

```
[]

[ 10, 20, 30 ]
```

2. **Create using new keyword (Constructor)**
   The “Array Constructor” refers to a method of creating arrays by invoking the Array constructor function.

Example:

```
// Creating and Initializing an array with values

let a = new Array(10, 20, 30);

console.log(a);
```

Output:

```
[ 10, 20, 30 ]
```

Methods

1. **Length**
   The length property of an array returns the number of elements in the array. It automatically updates as elements are added or removed.

Example:

```
let a = ["HTML", "CSS", "JS", "React"];

console.log(a.length);
```

Output:

```
4
```

2. **toString()**
   The toString() method converts the given value into the string with each element separated by commas.

Example:

```
let a  = ["HTML", "CSS", "JS", "React"];

let s = a.toString();

console.log(s);
```

Output:

```
HTML,CSS,JS,React
```

3. **join()**
   This join() method creates and returns a new string by concatenating all elements of an array. It uses a specified separator between each element in the resulting string.

Example:

```
let a = ["HTML", "CSS", "JS", "React"];

console.log(a.join('|'));
```

Output:

```
HTML|CSS|JS|React
```

4. **concat()**
   The concat() method is used to concatenate two or more arrays and it gives the merged array**.**

Example:

```
let a1 = [11, 12, 13];

let a2 = [14, 15, 16];

let a3 = [17, 18, 19];

let newArr = a1.concat(a2, a3);

console.log(newArr);
```

Output:

```
[

  11, 12, 13, 14, 15,

  16, 17, 18, 19

]
```

5. **flat()**
   The flat() method is used to flatten the array i.e. it merges all the given array and reduces all the nesting present in it.

Example:

```
const a1 = [['1', '2'], ['3', '4', '5',['6'], '7']];

const a2 = a1.flat(Infinity); //Using infinity as the depth ensures that the array is
//completely flattened, regardless of how deeply nested it is.

const a3=a1.flat(1) // only flat up to depth 1

console.log(a2);

console.log(a3)
```

Output:

```
['1', '2', '3','4', '5', '6', '7']

['1', '2', '3', '4', '5',['6'], '7'];
```

6. **push()**
   The push() method is used to add an element at the end of an Array. As arrays in JavaScript are mutable objects, we can easily add or remove elements from the Array.

Example:

```
let a = [10, 20, 30, 40, 50];

a.push(60);

a.push(70, 80, 90);

console.log(a);
```

Output:

```
[

  10, 20, 30, 40, 50,

  60, 70, 80, 90

]
```

7. **unshift()**
   The unshift() method is used to add elements to the front of an Array.

Example:

```
let a = [20, 30, 40];

a.unshift(10, 20);

console.log(a);
```

Output:

```
[ 10, 20, 20, 30, 40 ]
```

8. **pop()**
   The pop() method is used to remove elements from the end of an array.

Example:

```
let a = [20, 30, 40, 50];

a.pop();

console.log(a);
```

Output:

```
[ 20, 30, 40 ]
```

9. **shift()**
   The shift() method is used to remove elements from the beginning of an array

Example:

```
let a = [20, 30, 40, 50];

a.shift();

console.log(a);
```

Output:

```
[ 30, 40, 50 ]
```

10. **splice()**
    The splice() method is used to Insert and Remove elements in between the Array.

Example:

```
let a = [20, 30, 40, 50];

a.splice(1, 3);

a.splice(1, 0, 3, 4, 5);

console.log(a);
```

Output:

```
[ 20, 3, 4, 5 ]
```

11. **slice()**

The slice() method returns a new array containing a portion of the original array, based on the start and end index provided as arguments

Example:

```
const a = [1, 2, 3, 4, 5];

const res = a.slice(1, 4);

console.log(res);
```

Output:

```
[ 2, 3, 4 ]

[ 1, 2, 3, 4, 5 ]
```

12. **reverse()**

The reverse() method is used to reverse the order of elements in an array. It modifies the array in place and returns a reference to the same array with the reversed order.

Example:

```
let a = [1, 2, 3, 4, 5];

a.reverse();

console.log(a);
```

Output:

```
[ 5, 4, 3, 2, 1 ]
```

Tasks

1. Create an array of your favorite fruits. Add two more fruits to the array and display the total number of fruits in the array.
2. You have an array ["Alice", "Bob", "Charlie"]. Convert this array into a single string separated by commas and then convert it into a string separated by |.
3. Given the array let numbers = [10, 20, 30, 40, 50];, remove the element at index 2. Then, remove the element at index 3. Compare the results.
4. Combine two arrays let arr1 = [1, 2, 3] and let arr2 = [4, 5, 6] into a single array and display the result.
5. Given a nested array let nestedArray = [1, [2, [3, 4]], 5];, flatten it by one level. Then flatten it completely.
6. Start with an array let tasks = ["Task 2", "Task 3"];. Add "Task 1" to the beginning, then remove the first task. Log the array at each step.
7. Create an array let colors = ["red", "blue", "green", "yellow"];. Remove the last element and reverse the order of the remaining elements.
8. Given the array let animals = ["dog", "cat", "rabbit", "mouse"];, replace "cat" with "lion". Then add "elephant" and "tiger" after "dog".
9. From the array let numbers = [10, 20, 30, 40, 50, 60];, create a new array containing only the first three elements. Create another array containing the last two elements.
10. You are given the array let books = ["Book A", "Book B", "Book C", "Book D"];. Add "Book E" to the end of the array, remove "Book A", combine the resulting array with let magazines = ["Magazine 1", "Magazine 2"], and convert the final array into a string where elements are separated by ;.

JSON

JavaScript Object Notation

JSON (JavaScript Object Notation) is a lightweight text-based format for storing and exchanging data. It is easy to read, write, and widely used for communication between a server and a client.

- JSON stores data in key-value pairs.
- It is language-independent but derived from JavaScript syntax.
- JSON data is written in a human-readable format.
- It supports objects { } and arrays [ ] for data representation.

Example:

```
{
   "name": "Shubham Verma",
   "age": 22,
   "city": "Haryana"
}
```

Parsing JSON data in JavaScript and accessing its values

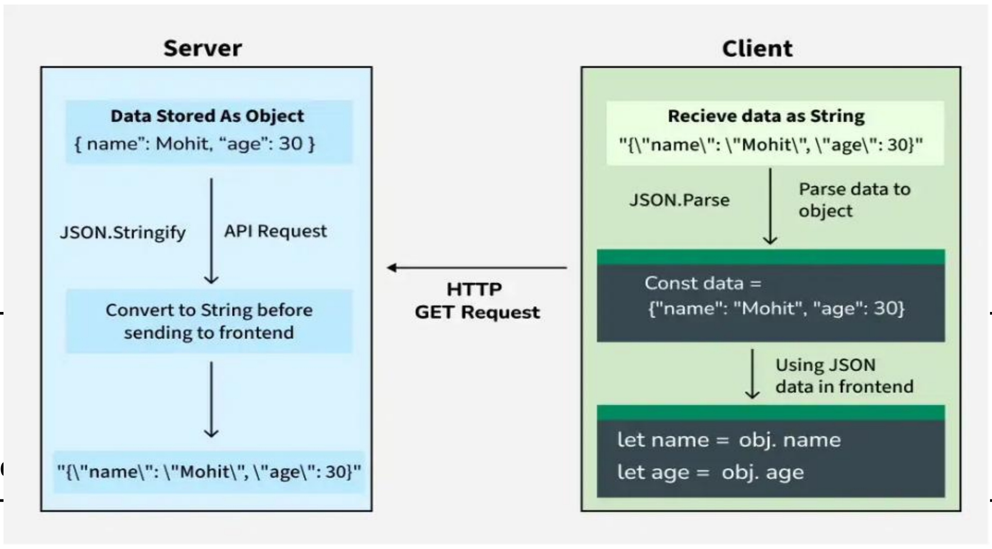
```
const jsonStr = '{"name": "Mohit", "age": 30, "city": "New Delhi"}';
// Convert JSON string into JavaScript object
const obj = JSON.parse(jsonStr);
console.log(obj)
// Accessing JSON data
console.log(obj.name);
console.log(obj.age);
console.log(obj.city);
```

Output

- JSON.parse() is used to convert the JSON string into a JavaScript object.

How Does JSON work?

```
Mohit

30

New D
```


Server
Data Stored As Object
{ name": Mohit, "age": 30 }
JSON.Stringify
API Request
Convert to String before sending to frontend
"{\"name\": \"Mohit\", \"age\": 30}"
HTTP GET Request
Client
Recieve data as String
"{\"name\": \"Mohit\", \"age\": 30}"
JSON.Parse
Parse data to object
Const data = {"name": "Mohit", "age": 30}
Using JSON data in frontend
let name = obj. name
let age = obj. age

JSON is commonly used to transfer data between a server and a web application.Here's the simple process

- The server sends data as a JSON string
- The client receives the JSON string and converts into a native object for use in the application

JSON Array with Objects

```
const jsonArr = '[{"name": "Amit", "age": 30}, {"name": "Mohit", "age": 25}, {"name": "Rohit", "age": 35}]';

// Parse JSON array into JavaScript object

const obj = JSON.parse(jsonArr);

// Accessing array elements

console.log(obj[0].name);

console.log(obj[1].age);

console.log(obj[2].name);
```

Output

JSON Object to JSON String

```
const obj = {
   name: "Mohit",
   age: 30,
   city: "New Delhi"
};
// Convert JavaScript object to JSON string
const jsonStr = JSON.stringify(obj);
console.log(jsonStr);
```

Output

```
{"name":"Mohit","age":30,"city":"New Delhi"}
```

- JSON.stringify() converts a JavaScript object into a JSON-formatted string.
- Useful for sending data to servers or storing it as text

Nested JSON Data

```
const nJSON = `{
   "a": [
      {"fName": "Amit", "lName": "Kumar"},
      {"fName": "Sumit", "lName": "Dev"},
      {"fName": "Punit", "lName": "Singh"}
   ]
}`;
const obj = JSON.parse(nJSON); // Parse nested JSON string
// Access nested data
console.log(obj.a[0].fName);
console.log(obj.a[1].lName);
console.log(obj.a[2].fName);
```

Output

Applications of JSON

- Web APIs: JSON is commonly used to fetch data from RESTful APIs.
- Configuration Files: Many tools and applications use JSON for storing configuration settings.
- Data Storage: JSON is used in NoSQL databases like MongoDB.
- Cross-Language Communication: JSON provides data exchange between different programming environments.

Advantages of JSON

- Human-Readable: Easy for developers to read and debug.
- Compact: Uses a simple structure, reducing data size.
- Fast Parsing: Native support in many languages makes parsing quick.
- Standard Format: Universally accepted for data exchange.

```
Amit

Dev

Punit
```

Call by Value & Call by Reference

Call by Value

In this method, values of actual parameters are copied to the function's formal parameters, and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in the actual parameters of the caller.

Suppose there is a variable named "a". Now, we store a primitive value(boolean, integer, float, etc) in the variable "a". Let us store an integer value in "a", Let a=5. Now the variable "a" stores 5 and has an address location where that primitive value sits in memory.

Now, suppose we copy the value of "a" in "b" by assignment (a=b). Now, "b" points to a new location in memory, containing the same data as variable "a".

Thus, a=b=5 but both point to separate locations in memory.

This approach is called ***call by value*** *where 2 variables become the same by copying the value but in 2 separate spots in the memory.*

```
let a=5;

let b ;

b=a;

a=3;

console.log(a);

console.log(b);
```

Output

```
3

5
```

Example using Function

```
function changeValue(x) {
   x = 10; // Modifying the copy
   console.log("Inside function, x:", x); //Output: 10
}
let num = 5;
console.log("Before function call, num:", num); // Outputs: 5
changeValue(num); // Passes a copy of `num`
console.log("After function call, num:", num); // Outputs: 5
```

Features of Call by Value

- Function arguments are always passed by value.
- It copies the value of a variable passed in a function to a local variable.
- Both these variables occupy separate locations in memory. Thus, if changes are made in a particular variable it does not affect the other one.

Call by Reference

Here, Actual and copied variables are created in the same memory location. On passing variables in a function, any changes made in the passed parameter will update the original variable's reference too. This is **called by reference**. It behaves quite differently from by value. All objects interact by reference.

Features of Call by Reference

- In JavaScript, all objects interact by reference.
- If an object is stored in a variable and that variable is made equal to another variable then both of them occupy the same location in memory.
- Changes in one object variable affect the other object variable.

Example

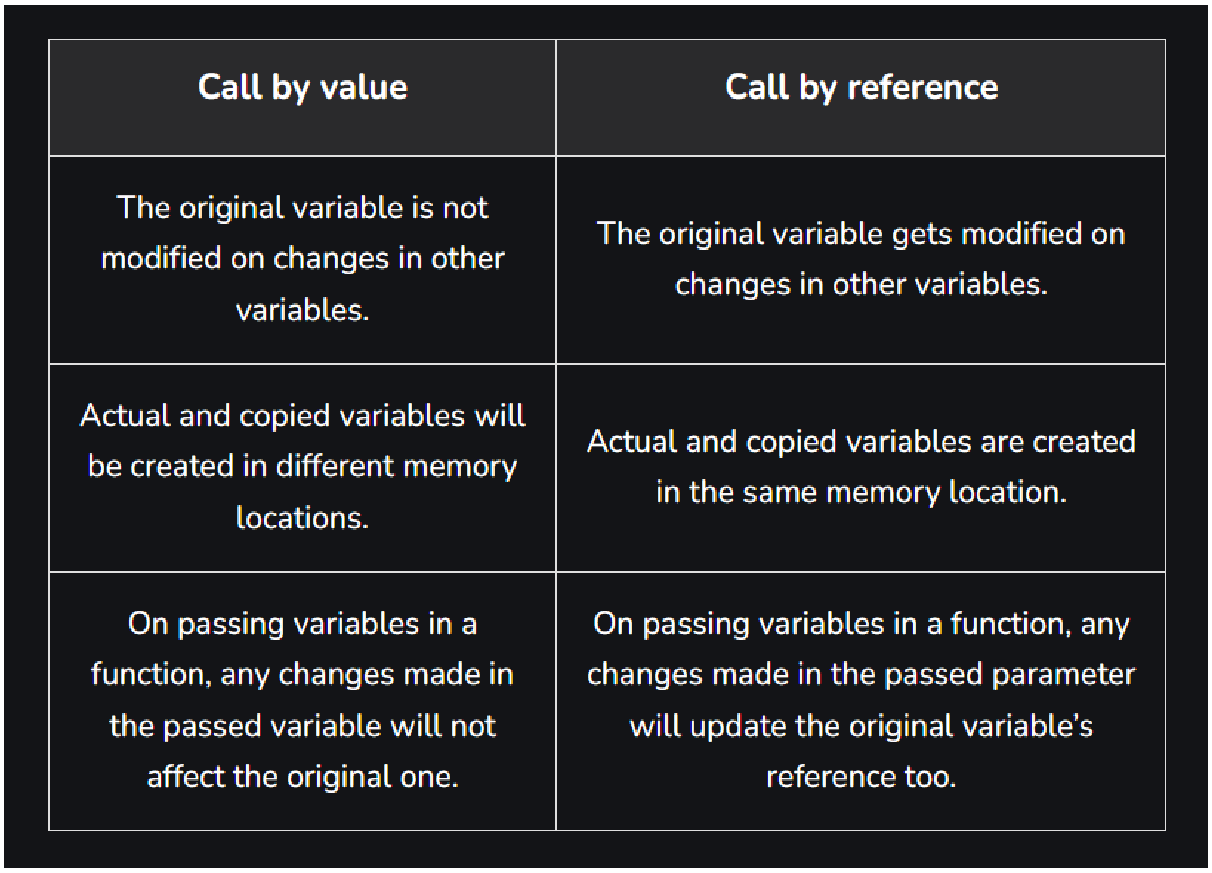
```
// By reference (all objects (including functions))
let c = { greeting: 'Welcome' };
let d;
d = c;

// Mutating the value of c
c.greeting = 'Welcome to Synnefo Solutions';
console.log(c);
console.log(d);
```

Ouput

```
{greeting: 'Welcome to Synnefo Solutions'}
{greeting: 'Welcome to Synnefo Solutions'}
```

Example using Function

```
function modifyObject(obj) {
    obj.name = "Alice"; // Modify the property
    console.log("Inside function:", obj); // { name: "Alice" }
}
let person = { name: "John" };
console.log("Before function call:", person); // { name: "John" }
modifyObject(person); // Passes the reference to the object
console.log("After function call:", person); // { name: "Alice" }
```



Call by value	Call by reference
The original variable is not modified on changes in other variables.	The original variable gets modified on changes in other variables.
Actual and copied variables will be created in different memory locations.	Actual and copied variables are created in the same memory location.
On passing variables in a function, any changes made in the passed variable will not affect the original one.	On passing variables in a function, any changes made in the passed parameter will update the original variable's reference too.

Closures

A closure is a combination of a function and its lexical environment (the scope in which it was declared). Closures allow functions to "remember" and access variables from their outer scope, even after the outer function has finished executing.

Key characteristics of Closures

- A closure is created every time a function is defined.
- Closures give you access to an outer function’s scope from an inner function.
- They are commonly used for data encapsulation and maintaining state in JavaScript.

Example

```
function outerFunction(outerVariable) {
    return function innerFunction(innerVariable) {
        console.log(`Outer Variable: ${outerVariable}`);
        console.log(`Inner Variable: ${innerVariable}`);
    };
}
const newFunction = outerFunction("outside");
newFunction("inside");
```

Output

```
Outer Variable: outside
Inner Variable: inside
```

- The createCounter function creates a private variable count.
- The returned function (closure) can access and modify count, even though createCounter has finished executing.

Example with counters

```
function createCounter() {
    let count = 0; // Private variable
    return function () {
        count++;
        return count;
    };
}
const counter = createCounter();
console.log(counter()); // 1
console.log(counter()); // 2
console.log(counter()); // 3
```

- The createCounter function creates a private variable count.
- The returned function (closure) can access and modify count, even though createCounter has finished executing.

Tasks

1. Counter Function
   Create a function that maintains a private count and returns two functions: one to increment the count and another to decrement it. Use these functions to perform operations on the count.

2. Discount Calculator
   Write a function that takes a discount percentage and returns another function to calculate the discounted price of a product. Use this function to apply discounts to various products.

3. Task Manager
   Create a function that manages a list of tasks with methods to add new tasks and retrieve the current task list. Use this function to manage and display a set of tasks.

4. Rate Limiter
   Design a function that limits how many times a specific action can be performed. If the limit is exceeded, it should log a message indicating the limit has been reached.

5. Personalized Greeting
   Write a function that takes a name and returns another function to generate a greeting based on the time of day (e.g., "morning", "evening"). Use this function to create personalized greetings.

Final Tasks

1. Write a function that takes a number n as input and returns an array of numbers from 1 to n, with the following rules:

    - Replace multiples of 3 with "Fizz".
    - Replace multiples of 5 with "Buzz".
    - Replace multiples of both 3 and 5 with "FizzBuzz".

2. Create a **Student Management System** using JavaScript and DOM manipulation with the following requirements:

    1. There should be three input fields for:
        - **Name** (text)
        - **Age** (number)
        - **Course** (text)
    2. Add a button labeled **"Add Student"**.
    3. When the button is clicked:
        - Validate the inputs to ensure all fields are filled.
        - Add the student's details (name, age, course) to an array.
        - Display the updated student list dynamically below the input fields.
    4. Use a traditional for loop to iterate through the array and display the list. Each student should be shown in the format:
       #1: Name: John, Age: 20, Course: Computer Science
       #2: Name: Jane, Age: 22, Course: Mathematics
    5. If no students are added, display the message: **"No students added yet."**

    Bonus: Ensure the input fields are cleared after each student is added.