1.Defining Functions

function in JavaScript is a block of reusable code that performs a specific task. Functions allow us to organize code into logical units, which can be called and reused throughout a program.

**A function declaration defines a function with a specified name.**

function sayHello() {

console.log("Hello, World!");

}

sayHello is the function name.

console.log("Hello, World!"); is the block of code the function executes.

**A function expression creates a function and assigns it to a variable.**

const greet = function() {

console.log("Hello, World!");

};

greet is the variable, and it holds a function as its value.

This function can then be invoked using greet().

2.Function Syntax

Basic Syntax:

function functionName(parameters) {

// Function body

}

function add(a, b) {

return a + b;

}

console.log(add(3, 4)); // Outputs: 7

The function add takes two parameters a and b, adds them together, and returns the result.

3. Function Parameters and Arguments

Parameters are placeholders used in the function definition.

Arguments are the actual values passed to the function when it is called.

function greet(name) {

console.log("Hello, " + name);

}

greet("John"); // Argument: "John"

greet("Alice"); // Argument: "Alice"

name is the parameter and "John" or "Alice" are the arguments passed to the function.

4. Return Statement

The return statement is used to return a value from a function. It terminates the function and sends a value back to the caller.

function multiply(x, y) {

return x \* y;

}

let result = multiply(4, 5); // result holds the value 20

console.log(result); // Outputs: 20

The return keyword sends the result back to the caller.

5.Function Expressions

A function expression is a function assigned to a variable. It can be anonymous (without a name) or named.

const sum = function(x, y) {

return x + y;

};

console.log(sum(3, 4)); // Outputs: 7

Function expressions can also be used as arguments to other functions.

6. Arrow Functions

Arrow functions provide a more concise syntax for writing functions. They are especially useful when you need to write functions that are used inline or as callbacks.

Syntax:

const functionName = (parameters) => {

// Function body

};

Arrow functions can also be used for single-expression functions, where curly braces and return are optional

Arrow functions do not have their own this value. They inherit the this value from the surrounding context.

7. Anonymous Functions

An anonymous function is a function that is defined without a name. These are often used when you want to pass a function as an argument or define it on the fly.

setTimeout(function() {

console.log("Executed after 2 seconds");

}, 2000);

Here, the function doesn't have a name but is passed directly to setTimeout.

8. IIFE (Immediately Invoked Function Expressions)

An IIFE is a function that is executed immediately after it is defined. This pattern is useful for creating a scope that doesn't pollute the global space.

(function() {

// Code here

})();

The function is wrapped in parentheses () and is immediately executed.

9. Function Scope

Scope refers to where variables are accessible in your program. Variables declared inside a function are only accessible within that function, while variables declared outside are global.

function example() {

let insideVar = "Inside Function";

console.log(insideVar); // Works fine

}

console.log(insideVar); // Error: insideVar is not defined

insideVar is scoped to the function example() and is not accessible outside.

10. Closures

A closure occurs when a function retains access to its lexical scope, even after the function has finished executing. Closures are often used for data encapsulation.

function outer() {

let count = 0;

return function inner() {

count++;

console.log(count);

};

}

const counter = outer();

counter(); // Outputs: 1

counter(); // Outputs: 2

The inner function retains access to the count variable from the outer function even after outer has finished executing.

11. Higher-Order Functions

A higher-order function is a function that can take other functions as arguments or return functions as values.

function processArray(arr, callback) {

for (let i = 0; i < arr.length; i++) {

callback(arr[i]);

}

}

processArray([1, 2, 3], function(item) {

console.log(item \* 2); // Outputs: 2, 4, 6

});

processArray takes a function (callback) and applies it to each item in the array.

12. Callback Functions

A callback function is a function passed into another function as an argument, which is then invoked after the outer function completes its execution.

function fetchData(url, callback) {

setTimeout(function() {

callback("Data fetched from " + url);

}, 1000);

}

fetchData("https://api.example.com", function(response) {

console.log(response);

});

fetchData takes a callback function and calls it once the data is fetched.

13. Recursion

Recursion is the process where a function calls itself in order to solve a problem. Recursion is useful when you can break a problem down into smaller, similar subproblems.

function factorial(n) {

if (n === 0) {

return 1;

}

return n \* factorial(n - 1); // Recursion

}

console.log(factorial(5)); // Outputs: 120

factorial calls itself with n - 1 until the base case (n === 0) is reached.