Functions in JavaScript

Understanding Function Concepts,
Recursion, and Arrow Functions

Introduction to JavaScript Functions

What is a JavaScript Function?

- A block of code designed to perform a specific task.
- Functions are executed when they are invoked or called

Function Syntax

JavaScript Function Syntax:

- **Keyword**: `function`
- **Name**: Function identifier.
- **Parameters**: Listed in parentheses `()`.
- **Body**: Code to be executed, enclosed in curly brackets `{}`

Example:

```
function myFunction(p1, p2) {
  return p1 * p2;
}
```

Function Parameters and Arguments

Parameters:

- Variables listed as a part of the function definition.
- Example: `function name(parameter1, parameter2) { ... }`

Arguments:

- Values passed to the function when it is invoked.
- Inside the function, they behave as local variables.

Function Invocation

How to Invoke a Function:

- **Events**: Invoked when an event occurs (e.g., button click).
- **Direct Call**: Invoked from JavaScript code.
- **Automatically**: Self-invoked

Example:

```
myFunction(4, 3); // Direct call
```

Function Return

Return Statement:

- Stops the execution of the function.
- Returns a value to the caller.

Example:

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

let x = myFunction(4, 3); // x will be 12
```

Why Use Functions?

- **Reuse Code**:
  - Write code once, use it multiple times.
  - Use the same function with different arguments to produce different results.

The () Operator

Invoking Functions:

- The ( ) operator is used to call the function.

Example:

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit - 32);
}

let value = toCelsius(77); // 25°C
```

Functions as Variable Values

Using Functions in Variables:

- Functions can be used directly as variable values.

Example:

```
let text = "The temperature is " + toCelsius(77) + " Celsius";
```

Local Variables

Local Variables:

- Declared within a function and can only be accessed inside that function.

Example:

```
function myFunction() {
  let carName = "Volvo";
  // carName is accessible here
}
// carName is not accessible here
```

Summary

Key Points:

- Functions are reusable blocks of code.
- They are invoked using the `()` operator.
- Parameters and arguments allow flexibility.
- Functions return values and can be used in expressions.
- Local variables are confined to the function scope.

Introduction to Function Types

Overview:

- JavaScript supports various types of functions, each with different use cases.
- Understanding different function types is crucial for writing clean, efficient, and reusable code.

Named Functions

Definition:

- Functions that have a name associated with them.

Syntax:

- Defined using the `function` keyword followed by the function name

Example:

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

let result = add(5, 10); // 15
```

Anonymous Functions

Definition:

- Functions without a name.
- Often assigned to a variable or used as an argument in other functions.

Example:

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

let result = multiply(5, 10); // 50
```

Arrow Functions

Definition:

- A concise syntax for writing functions using the `=>` (arrow) operator.

Syntax:

- Often used for short functions.

Example:

```
const subtract = (a, b) => a - b;

let result = subtract(10, 5); // 5
```

Immediately Invoked Function Expressions (IIFE)

Definition:

- Functions that are executed immediately after being defined.

Syntax:

- Wrapped in parentheses and followed by `()` to invoke them.

Example:

```
(function() {
  console.log("This function runs immediately!");
})();
```

Recursive Functions

Definition:

- Functions that call themselves during their execution.

Syntax:

- Typically used for problems that can be broken down into smaller, similar problems.

Example:

```
function factorial(n) {
  if (n === 0) return 1;
  return n * factorial(n - 1);
}

let result = factorial(5); // 120
```

Higher-Order Functions

Definition:

- Functions that take other functions as arguments or return them as results.

Syntax:

- Commonly used in functional programming.

Example:

```
function applyOperation(a, b, operation) {
  return operation(a, b);
}

let result = applyOperation(5, 10, (x, y) => x + y); // 15
```

Closure Function

```
let counter = 0;

function add() {
  counter += 1;
}

add();
add();
add();

console.log(counter); // Output: 3
```

Summary of Functions

Nested Functions: Functions within functions.

Closures: Functions with persistent lexical scope.

Arrow Functions: Concise syntax for functions.

IIFE: Functions that execute immediately.

Function Definition Methods: Declaration, Expression, Arrow, IIFE.

Recursion: Functions calling themselves.

Parameters & Return Types: Passing values and getting results from functions.