Day - 14

20.02.2025

Functions:-

A function definition is sometimes also termed a function declaration or function statement. Below are the rules for creating a function in JavaScript:

- Begin with the keyword function followed by,
- A user-defined function name (In the above example, the name is sum)
- A list of parameters enclosed within parentheses and separated by commas (In the above example, parameters are x and y)
- A list of statements composing the body of the function enclosed within curly braces {} (In the above example, the statement is “return x + y”).

Return Statement

In some situations, we want to return some values from a function after performing some operations. In such cases, we make use of the return. This is an optional statement. In the above function, “sum()” returns the sum of two as a result.

Function Parameters

Parameters are input passed to a function. In the above example, sum() takes two parameters, x and y.

Function Definition

```
function welcomeMsg(name) {
    return ("Hello " + name + " welcome to GeeksforGeeks");
}
let nameVal = "User";
// calling the function
console.log(welcomeMsg(nameVal));
```

Function Invocation

The function code you have written will be executed whenever it is called.

- Triggered by an event (e.g., a button click by a user).
- When explicitly called from JavaScript code.
- Automatically executed, such as in self-invoking functions.

Function Expression

It is similar to a function declaration without the function name. Function expressions can be stored in a variable assignment.

```
const a = ["Hydrogen", "Helium", "Lithium", "Beryllium"];
const a2 = a.map(function (s) {
    return s.length;
});
console.log("Normal way ", a2);
const a3 = a.map((s) => s.length);
console.log("Using Arrow Function ", a3);
```

Boolean

- Boolean values are typically produced by relational operators, equality operators, and logical NOT (!).
- They can also be produced by functions that represent conditions, such as Array.isArray(). Note that binary logical operators such as && and || return the values of the operands, which may or may not be boolean values.
- Boolean values are typically used in conditional testing, such as the condition for if...else and while statements, the conditional operator (? :), or the predicate return value of Array.prototype.filter().
- You would rarely need to explicitly convert something to a boolean value, as JavaScript does this automatically in boolean contexts, so you can use any value as if it's a boolean, based on its truthiness.
- You are also encouraged to use if (condition) and if (!condition) instead of if (condition === true) or if (condition === false) in your own code so you can take advantage of this convention.

```
// Do this:

// This always returns a boolean value

const isObject = (obj) => !!obj && typeof obj === "object";



// Or this:

const isObject = (obj) => Boolean(obj) && typeof obj === "object";
```

```
// Or this:

const isObject = (obj) => obj !== null && typeof obj === "object";



// Instead of this:

// This may return falsy values that are not equal to false

const isObject = (obj) => obj && typeof obj === "object";

if (new Boolean(true)) {

  console.log("This log is printed.");

}



if (new Boolean(false)) {

  console.log("This log is ALSO printed.");

}



const myFalse = new Boolean(false); // myFalse is a Boolean object (not the primitive value
false)
```

```
const g = Boolean(myFalse); // g is true

const myString = new String("Hello"); // myString is a String object

const s = Boolean(myString); // s is true
```