JavaScript

aa

# Functions in JavaScript

## [Immediately Invoked Function Expressions (IIFE)](https://www.geeksforgeeks.org/javascript-immediately-invoked-function-expressions-iife/)

Functions are one of the building blocks of any programming language and JavaScript has taken the Functions to a whole new level. Functions are said to be a collection of statements to be executed in a proper sequence in a particular context. Now JavaScript provides a variety of methods to define and execute Functions, there are named functions, anonymous functions and then there are Functions that are executed as soon as they are mounted, these functions are known as **Immediately Invoked Function Expressions or IIFEs.** Let us dive deeper to know more about IIFEs.

Syntax: IIFEs follows a particular syntax as shown below.

**(function (){**

**// Function Logic Here.**

**})();**

**(function (){**

**// Function Logic Here.**

**}());**

Now let us probe some more to understand the relation between the Name and Syntax of this type of Functions.

**Why the name Immediately Invoked Function Expressions?**

**Immediately Invoked:** This part is easy to explain and demonstrate. This type of function is called immediately invoked as these functions are executed as soon as they are mounted to the stack, it requires no explicit call to invoke the function. If we look at the syntax itself we have two pairs of closed parentheses, the first one contains the logic to be executed and the second one is generally what we include when we invoke a function, the second parenthesis is responsible to tell the compiler that the function expression has to be executed immediately.

**Function Expressions:** It is simple to understand that a Function Expression is a Function that is used as an expression. JavaScript lets us use Functions as Function Expression if we Assign the Function to a variable, wrap the Function within parentheses or put a logical not in front of a function. The following program illustrates the different ways we can create a Function Expression.

**// Creating Function Expression by assigning to a variable.**

**var myFunc = function() { return "GeeksforGeeks"; };**

**// Creating Function Expression Using Logical Not.**

**!function() { return "GeeksforGeeks"; };**

**// Creating Function Expression Using Parentheses.**

**(function() { return "GeeksforGeeks"; });**

Now if you look at the third example code it is identical to the first part of the syntax of an IIFE, hence we can conclude that the first part of the IIFE contains a Function Expression while the final part invokes it immediately.

**Converting Functions to IIFEs**

As we have already seen what an IIFE is and how its syntaxial form can be broken down to make more sense, We can convert a regular function into an IIFE following only two steps as follows.

* Given any regular function definition, wrap the definition within a closed pair of parentheses, this will create your Function Expression.
* Lastly add another pair of parentheses and a semicolon to mark the end of the statement, and you have converted your regular Function into an IIFE.

// Regular Function.

function Greet()

{

console.log("Welcome to GFG!");

};

// Execution of Regular Function.

Greet();

// IIFE creation and execution.

(function() { console.log("Welcome to GFG!"); })();

Important Points

1. IIFE follow their own scope like any other function/variable in JavaScript. The part of the name immediately invoked is sometimes confusing to new developers as they expect the IIFE to execute irrespective of function scope, this is wrong. For example, let us take the following example where the IIFE is defined within a function and will only be immediately invoked if we call the Parent Function.

**function myFunc()**

**{**

**console.log("Welcome to");**

**// This will be executed after executing the previous log.**

**(function() { console.log("GeeksForGeeks!"); })();**

**console.log("Hi There!");**

**}**

**// Calling the Function.**

**myFunc();**

1. **IIFEs have there own scope i.e. the variables you declare in the Function Expression will not be available outside the function.**
2. **Similarly to other function IIFEs can also be named or anonymous, but even if an IIFE does have a name it is impossible to refer/invoke it.**
3. **IIFEs can also have parameters. For example,**

**// Declaring the parameter required.**

**(function(dt) {**

**console.log(dt.toLocaleTimeString());**

**// Passing the Parameter.**

**})(new Date());**

**Use Cases Of IIFE**

1. **Avoid polluting the global namespace**
2. **To create closures**
3. **Avoid conflict of variable names between libraries and programs.**
4. **IIFE is used to create private and public variables and methods**
5. **It is used to execute async and await function**
6. **It is used in JQuery Library**
7. **It is used to work with require function**