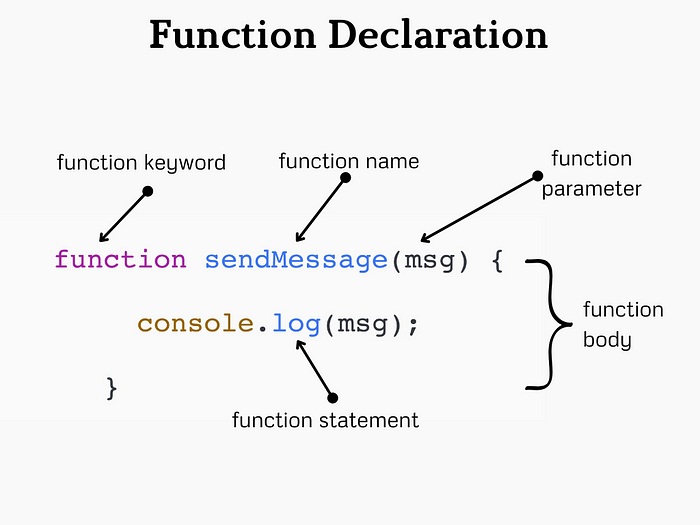
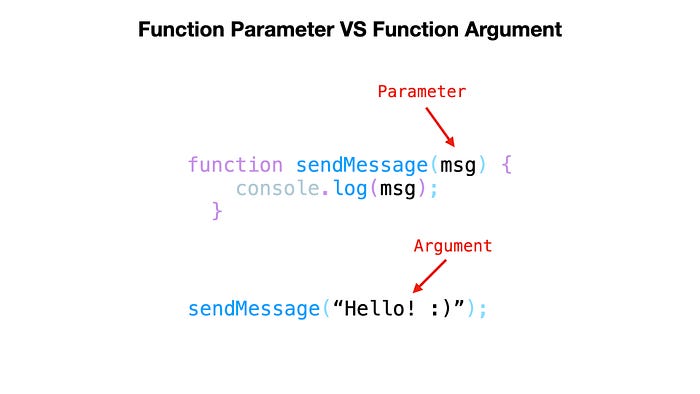
**Function:-**

**A function is a reusable block of code that is used to perform a specific task when something invokes it.**

A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().The code to be executed, by the function, is placed inside curly brackets: {}



Function parameters are listed inside the parentheses () in the function definition. Function arguments are the values received by the function when it is invoked. Inside the function, the arguments (the parameters) behave as local variables.



Named functions can be hoisted

Ex 1:-

        function myfun(a){

            return console.log(a);

            alert("it will not execute because it was in void")

        }

        myfun(20);

above program shows the output of 20 and alert will not be showed because it was returned after the statement of return means it was in voids

Points needs to be noted in functions

1) function name stores function definition

2) log off the function calling stores return value

3) statement after return will not execute because It was in void

4) Named function can only hoisted

5) function definition act as value because in js functions are first class functions

**Anonymous function:-**

Anonymous function is a function that is defined without a name

*var* anonfun=function(){

            return "this is anonymous function";

        }

        console.log(anonfun());

above program prints “this is anonymous function” in the output.it is similar to the named function but the difference is hoisting is not applied,it is also called as function expression.

**Arrow function:-**

Arrow function is a concise way of writing function in shorter way.

        var arrowfun=()=>{

            return "this is arrow function";

        }

        console.log(arrowfun());

above program prints “this is arrow function” in the output. It is a also have same functionality , it is also not hoisted.

Function with default parameters

        function hello(a="this is a function ",b="with default parameters"){

            return a+b;

        }

        console.log(hello());

above program shows the output “this is a function with default parameters” .Above program takes default parameters to print the output because we haven’t gave the parameters while calling the function.

**Immediately Invoked Function Expression**

An IIFE (Immediately Invoked Function Expression) is a JavaScript function that runs as soon as it is defined.

Following shows the syntax

(function(){

*//code goes here*

})()

        (function(){

            console.log("self invoking function invoked by itself") ;

        })();

**Callback Function**

Callback function is a function definition passed into a another function as an argument which is then invoked inside the outer function to complete some kind of task.

        function hello1(){

            return "hello1 function is triggered by main function";

        }

        function hello2(j){

            var a=j();

         return a;

        }

        console.log(hello2(hello1));

in following program hello1 is passed as an argument to a hello2 function .hello2 starts execution line by line. hello2 stores the hello1 function in j . In the next statement variable invokes the function stores the ouput in it and displays the output in a console “hello1 function is triggered by main function”.

**Global execution context:**

Global Execution Context is the first context that gets created when the JavaScript engine starts executing code.

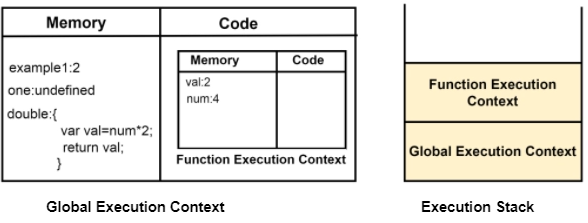
**Key Components**

**1. Memory Allocation (Creation Phase)**:

* During this phase, the engine sets up the memory for variables and functions.
* **Variables** declared with **var** are hoisted and initialized with **undefined**.
* **Function declarations** are hoisted and their definitions are stored in memory.
* Variables declared with **let** and **const** are also hoisted but are not initialized. They remain in a temporal dead zone until they are assigned a value

**2. Code Execution (Execution Phase)**:

* + The JavaScript engine executes the code line by line.
  + Variables declared with **var** are assigned their values.
  + Functions are available to be called.
  + Variables declared with **let** and **const** are assigned values when their declaration is encountered in the code.



**How it works on the functions**

When a function is invoked, a new **Execution Context(function execution context)** is created specifically for that function. This context is separate from the Global Execution Context but follows similar principles.

* Each function invocation creates a new execution context.
* Variables declared inside a function are local to that function and are not accessible outside it.
* The scope chain allows inner functions to access variables from their parent functions and the global context.

**Closures**

A closure is a function that has access to its own scope, the scope of the outer function, and the global scope. This means a closure can remember and access variables from its outer function even after that function has finished executing.

When an inner function have access to the variable which is declared in outer function even after the execution

      function ahello() {

        var a = "variable inside a outer function";

        function ahi() {

          var b = "varaible inside inner function";

          console.log(a);

console.log(b);

        }

        ahi();

      }

      ahello();