**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

**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 general 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 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”

**Scopes in javascript**:

Scope is a determined as a life of an variable

**Global Scope**:

Variables declared Globally (outside any function) have Global Scope. Global variables can be accessed from anywhere in a JavaScript program.

{

  var x = 2;

}

// x can be used here

**Block Scope**:

Variables declared within a block cannot be accessed from outside of the block.

{

  let x = 2;

}

// x can NOT be used here

**Local Scope**:

Variables declared inside a function have local scope. They can only be accessed within that function. They cannot be accessed from the outside.

function myFunction() {

  var a = "skills";

  // code here CAN use a

}

// code here can NOT use a

**Strings:**

Strings is a collection of words enclosed in a quotations

**Strings are immutable**

Strings are immutable we can’t change the original value

**String.length**

String.length are used to calculate the no off the characters are present in a string

**charAt()**

charAT is a string method used to find the character positioning

**at(-5)**

at is a string method used to find charcter using negative values

**charCodeAt()**

used to print the Unicode value of the charcter

**Variables :-** variables are containers used to store the data, we can declare variables with var, let and const

Programs on var in different cases.

//case - 1

//var a is "undefined" in the console because "value is not defined"

var a;

console.log(a);

//case - 2

//In output it shows "not defined" because "variable is not at all defined" in the program like this-> var b;

console.log(b);

//case - 3

//in output it shows "10" because value and variable both are defined

var c=10;

console.log(c);

//case - 4

//it shows the output "undefined" because intialisation is not hoisted

var d;

console.log(d);

d=10;

//case - 5

//it shows the output "10" because of hoisting, In hoisting declarations are moved to top

e=10;

console.log(e);

var e;

**variable declared with var have global scope can be redeclared and reassigned:-**

1) Variables declared with var inside a { } block can be accessed from outside the block because it doesn’t have a block scope it has global scope

{

  var x = 2;

}

// x CAN be used here

//because it has a global scope

2) variable declared with var can be redeclared

var x = "front end";

var x = 0;

//Variables defined with var can be redeclared.

3) variable declared with var can be re-assigned

var x;

x = 0;

**variable declared with let have block scope cannot be redeclared and must be reassigned:-**

1) Variables declared inside a { } block cannot be accessed from outside the block:

{

  let x = 2;

}

// x can NOT be used here

2) Variables defined with let can not be redeclared.

let x = "John Doe";

let x = 0;

//output – identifier has already been declared

3) Variable defined with let can be reassigned.

let x;

x = 0;

//output- 0

**variable declared with const have block scope cannot be redeclared and cannot be reassigned:-**

1) Variables declared inside a { } block cannot be accessed from outside the block:

{

 const x = 2;

}

// x can NOT be used here

2) Variables defined with const can not be redeclared.

Const x = "coding”

Const x = 0;

output – identifier has already been declared

3) Variable defined with const cannot be reassigned.

Const x;

x = 0;

//output-missing initializer in const declaration