**Function creation:**

Reusable code is that executed whenever it is called. It saves time. Perform repetitive task multiple times. Js provides many built in functions.

**Built-in functions:**

* **Is Nan ():** if object is a number or not.
* **ParseFloat (string):** if strings contains number or not.
* **String (object):** convert object into string.
* **Eval ():** returns evaluation of functions.
* **Date ()**: Returns today's date and time

**User defined functions:** created by users according to need.

**//without parameter**

add();

function add(){

var a = 10;

var b = 20;

sum = a + b;

document.write("the sum of two number is"+ " " + sum);

}

**//with parameter**

sub(20, 10)

function sub(x, y){

var sub = x - y;

document.write("the difference of two number is" + " " + sub);

}

**//with return**

var stored = mul(12, 4);

document.write("the product of two number is"+ " " + stored)

function mul(m, n){

var mul = m \* n ;

return mul;

}

**How is a function executed on a event in Js?**

<input type:”button” onclick=”add()” value=”buttonvalue”>

* Onclick = event handler
* Add() = function name

**Example:**

<script type = “text/javascript”>

Function add() {

Var a = 2, b = 3;

Var sum = a + b;

Document.write(“Addition :” + sum)’

}

</script>

<p>click the button</p>

<input type:”button” onclick=”add()” value=”click”>

**Arrow function**: Arrow functions were introduced in ES6. Also called fat arrow. Arrow functions allow us to write code shorter.

**Before:**

var myfun = function show ( ) {

document.write(“hello”); };

**Anonymous function:**

var myfun = function ( ) {

document.write(“hello”); };

**Arrow function:**

var myfun = ( ) => { document.write(“hello”); };

**Var, const, let:**

**Var:** The scope of a variable declared with var is its current execution context, which is either enclosing function declared outside any function, global.

var x = “hello”

Var x = “world“ can be declared

X = “wow” can be re-assigned

**Let:** Let allows you to declare variables that are limited in scope to the block, statement or expression on which it is used.

var x = “hello”

Var x = “world“ can’t be declared

X = “wow” can be re-assigned

**Const:** var x = “hello”

Var x = “world“ can’t be declared

X = “wow” can’t be re-assigned

**Promises:** In JavaScript, a promise is a good way to handle asynchronous operations. It is used to find out if the asynchronous operation is successfully completed or not. A promise may have one of three states.

* Pending
* Fulfilled
* Rejected

A promise starts in a pending state. That means the process is not complete. If the operation is successful, the process ends in a fulfilled state. And, if an error occurs, the process ends in a rejected state.

For example, when you request data from the server by using a promise, it will be in a pending state. When the data arrives successfully, it will be in a fulfilled state. If an error occurs, then it will be in a rejected state.

**async and sync**

**callback**