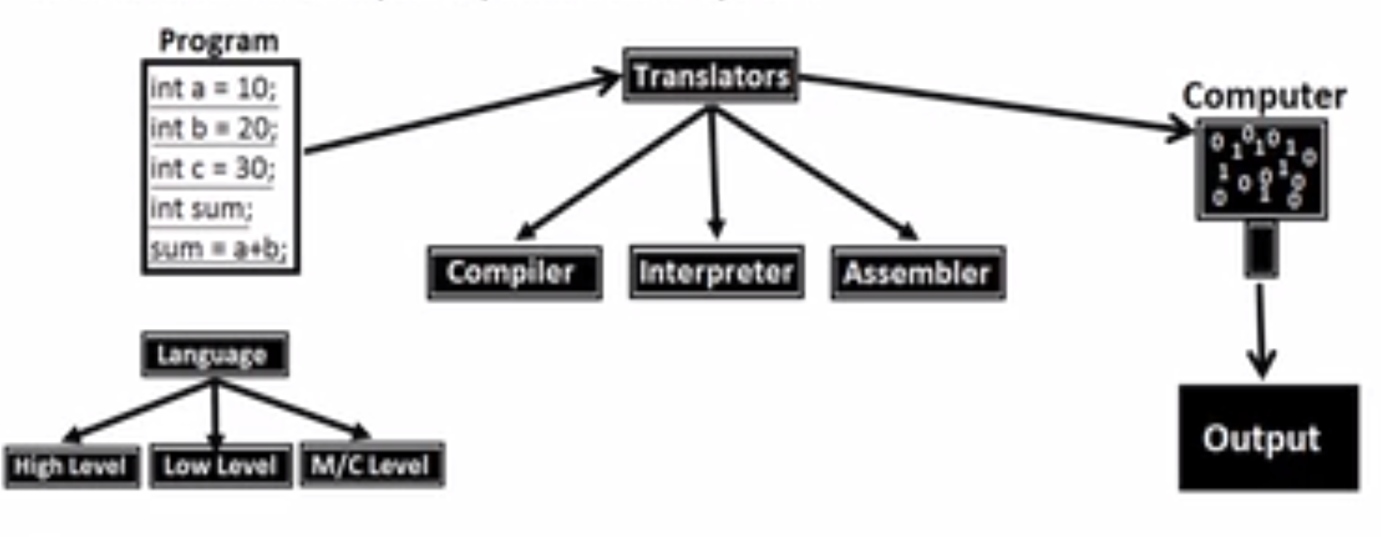
**JavaScript**:

**CONTENTS**

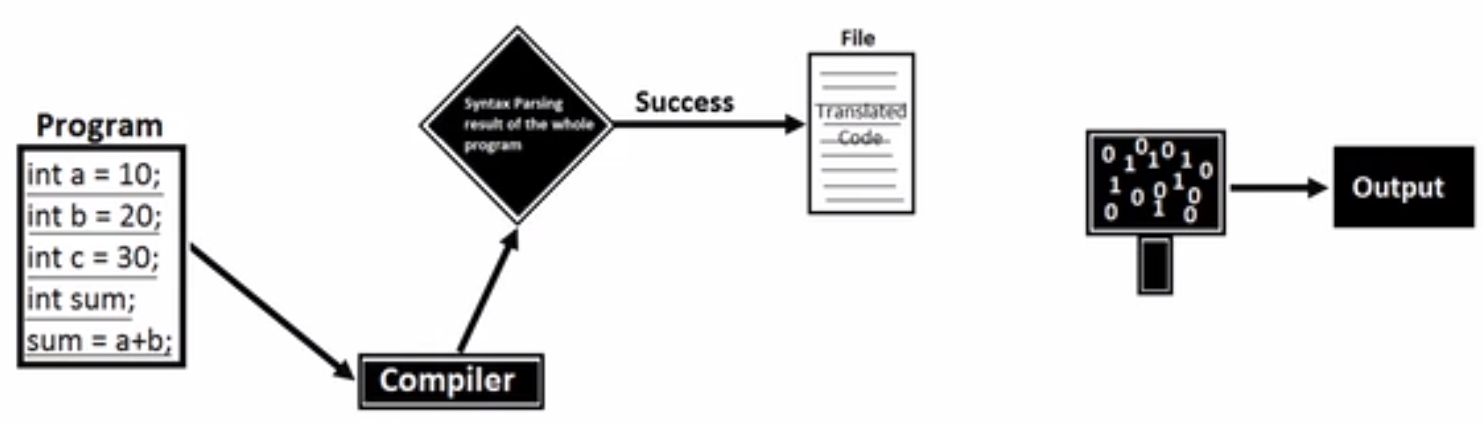
* **INTRODUCTION TO JAVA SCRIPT**
* **HISTORY OF JAVASCRIPT**
* **TOKENS**
* **VARIABLES AND DATATYPES**
* **CALLBYVALUE AND CALLBYREFERENCE**
* **PROGRAMM EXECUTION IN MEMORY**
* **VARIABLE HOISTING**
* **FUNCTION EXECUTION IN MEMORY**
* **CONDITIONAL STATEMENTS**
* **LOOPING STATEMENTS**
* **FUNCTIONS**
* **CLOSURES**
* **CALLBACK FUNCTION**
* **OBJECTS**
* **THIS KEYWORD**
* **INHERITANCE**
* **PROTOTYPE PROPERTY**
* **ARRAY**
* **DATE**
* **MATH**
* **JSON**
* **BOM**
* **DOM**
* **PREDEFINED OBJECTS**
* **JAVASCRIPT CALL STACK**
* **ES6 FEATURES**
* **Introduction to Java script:**
* **JavaScript:**
* JavaScript is a scripting language (interpretation and execution occurs at the run time)
* JavaScript is case sensitive
* using js one can create interactive webpages/dynamic web pages
* mainly developed for client side validation
* is also called interpreted programming language (as it uses interpreter for code translations)
* **PROGRAMS**:

Set of instructions that we are going to give for a computer to perform a

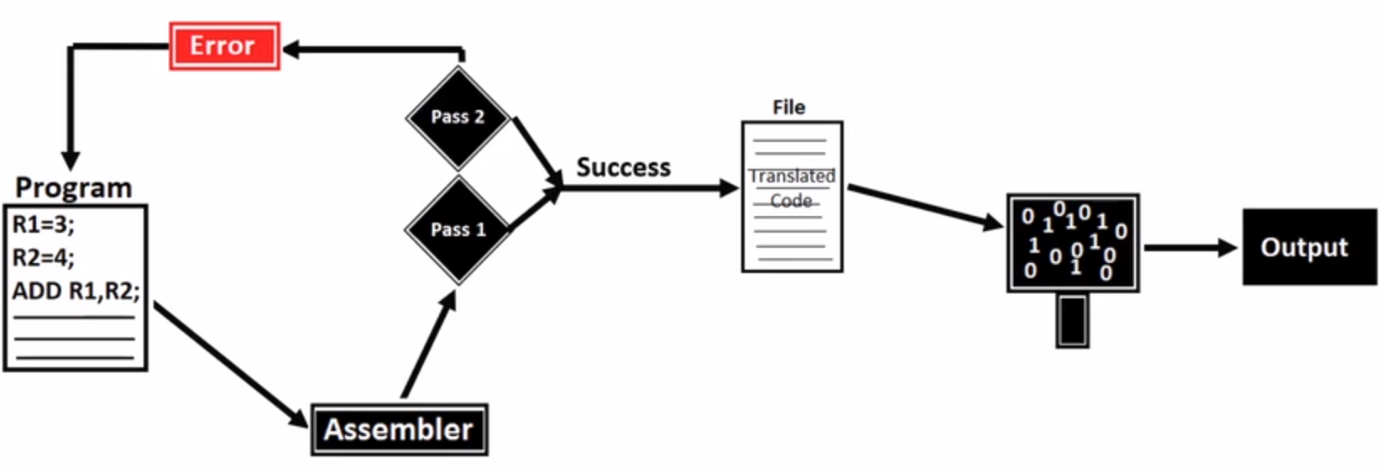
particular task.

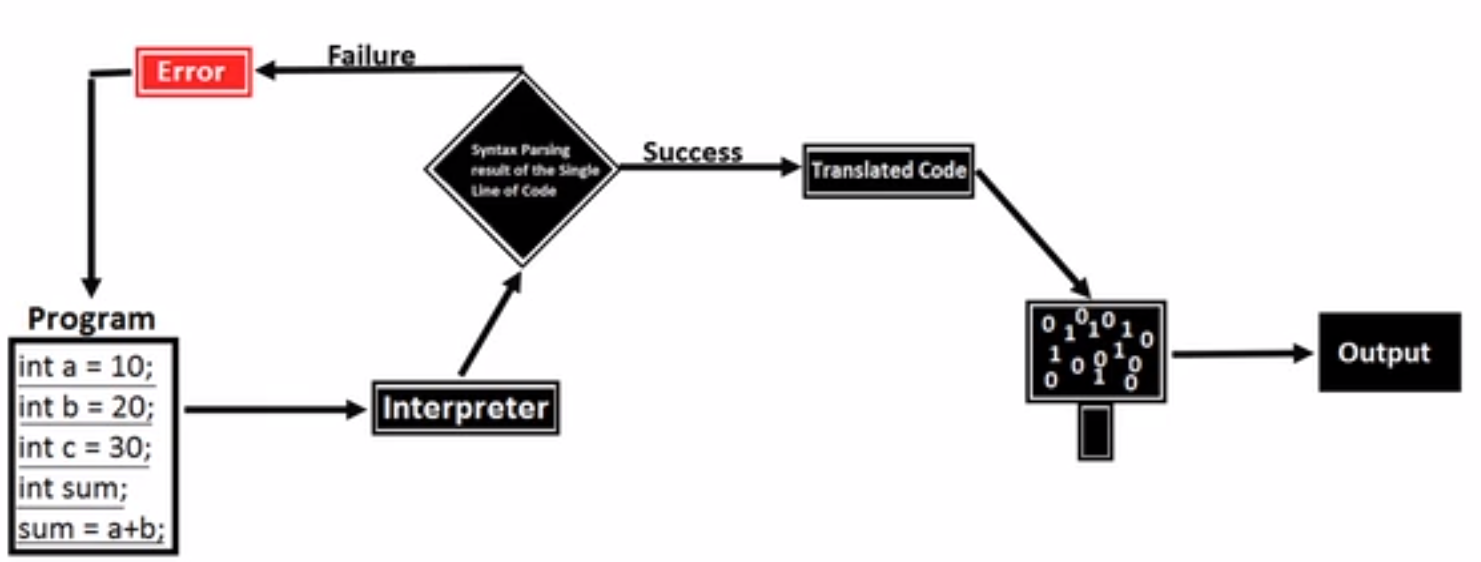


* **How compiler works:**

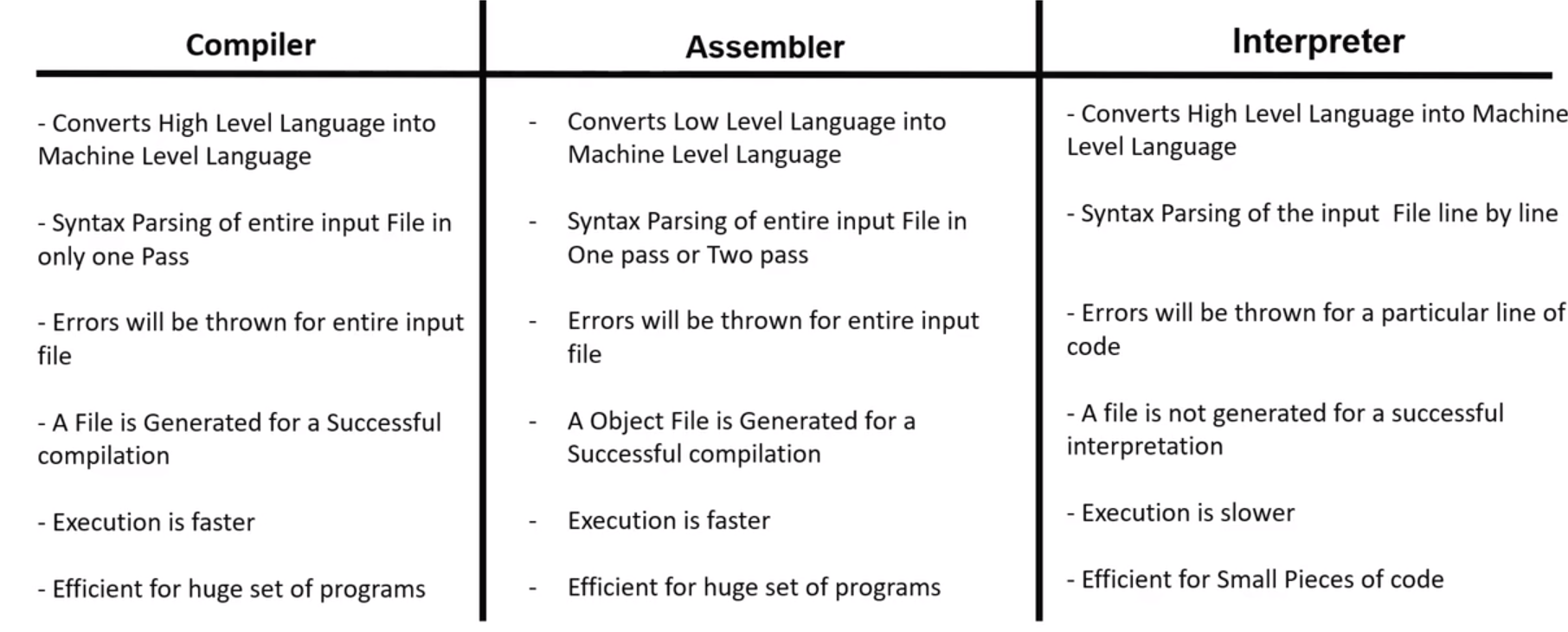


errors

* **How assembler works:** 
* **How interpreter works:**



* **Difference between compiler, assembler and interpreter:**



* **Execution of JavaScript:**

we can execute JavaScript code in 2 platforms

1]on browser

2]off browser i.e. node

we can execute our JavaScript code on the browser because it has an interpreter called JS\_engine

for e.g.

* chrome browser has js\_engine called v8
* Mozilla Firefox has js\_engine called spider-monkey
* IE has js\_engine called chakra
* safari has js\_engine called JavaScript core
* How js\_code can be embedded in html:

2 ways :

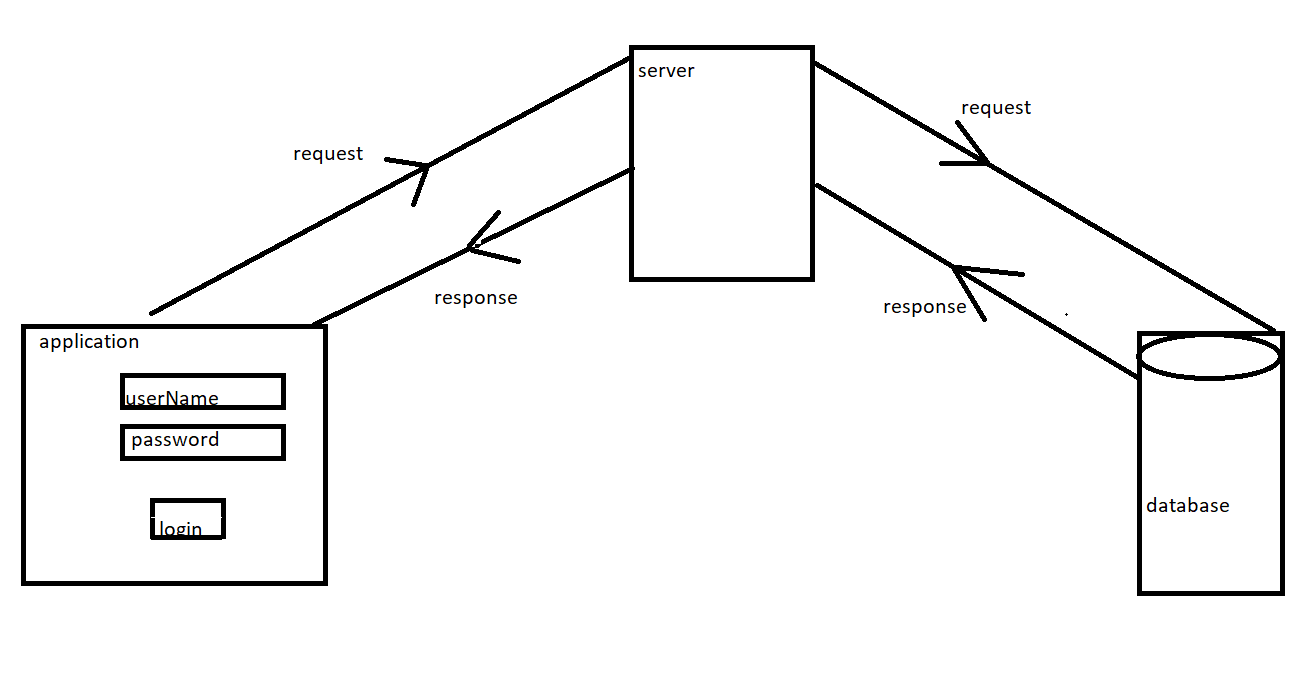
1]inline/internal:

By using script tags i.e. <script></script>

2]external/import:

By using tags i.e. <script src = “file path”></script>

* **client-side validation:**

****

we are programming in the browser end itself to perform some basic validation e.g checking @mail in email, proper number of digits in card etc…

* **Libraries built by using JavaScript:**

libraries are used to simplify a complex task i.e. js alone can perform a task in bulk code, but the same task can be performed with minimal/optimal code by using libraries.

* libraries built by using js:

jQuery, bootstrap, load ash, \_js

* \_**Frameworks built by using js:**
* NodeJS used for server-side programming
* react native used to develop mobile applications in android and iOS
* ReactJS used to develop web application
* ElectronJS used to develop standalone application (calculator, calendar)
* AngularJS used to develop single page application (Gmail, maps)
* ReactVR used in virtual reality
* TensorFlow used to develop machine learning algorithms
* **full stack JavaScript:**
* for client side (browser) we use core js (vanilla js)
* in webserver side we can use NodeJS
* in database we can use mongo DB and CouchDB
* **History of JavaScript:**
* JavaScript was developed by Brenden Eich in 1995 at Netscape (earlier name of Mozilla Firefox)
* He already had a code called mocha and he re wrote it and named it as spider monkey
* initially official name was live script and later for marketing purpose it is named as JavaScript
* **Tokens**:

Tokens are smallest unit of the program they are of 5 types

* + - Keywords
    - Identifiers
    - Literals
    - Operators
    - Separators
* **Keywords**:

Keywords are predefined words that have special meaning in JavaScript e.g.  break, case, catch, continue, debugger, default, delete, do , else , finally , for , function , if , in , instance of , new , return , switch , this , throw , try , type of , var , void , while , and with

* **Rules to write the keywords:**
  + JavaScript is a case sensitive language
  + We should use keywords in lowercase letters
  + We cannot use keywords as identifiers
* **Identifiers:**

JavaScript Identifiers are names given to variables, functions, etc.

e.g., var x = 10

function demo () {

}

* **Rules to write the identifiers:**
* You should not use any of the JavaScript reserved keywords as a variable name. For example, break or Boolean variable names are not valid.
* JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore or dollar character. For example, 5demo is an invalid variable name but \_5demo is a valid one.
* JavaScript variable names are case-sensitive. For example, Name and name are two different variables.
* **Literals**:

Literals are the values used while coding

e.g. numeric literals –7, 5, 8

string literals – “7”, “5”, “8”, “hello”

Boolean literals – true, false

null

undefined

* **Operators**:

Used to perform some mathematical operations e.g. (+, \_, \*, \*\*, /, %) etc.

* **Separators**:

Used to separate the statements in the program e.g. (, (), {}, [],) etc

* **Variables and Datatypes**:
  + **Variables:**

Variables are the containers that you can store the values in it

Syntax :

Var(keyword) name(identifier) = value

Fallowed by

var a // declaration

a = 10 // Initialization

console.log(a) // Utilization

* **Datatypes**

1. Primitive datatypes
2. Non-primitive datatypes (object references)

1.Primitive datatypes/immutable:

* + - * [undefined](https://developer.mozilla.org/en-US/docs/Glossary/undefined)
      * [Boolean](https://developer.mozilla.org/en-US/docs/Glossary/Boolean)
      * [Number](https://developer.mozilla.org/en-US/docs/Glossary/Number)
      * [String](https://developer.mozilla.org/en-US/docs/Glossary/String)
      * null
    1. Non-primitive datatypes (object references)/mutable:
* Objects
* Functions
* Arrays
* Date
* Math

e.g.

var x =>declaration

x = 10 =>initialization

log(x) => utilization

* **callByValue and callByReference**

primitive datatypes share call by value and non-primitive datatypes share call by reference

e.g., call by value

var a = 10

var b = a

e.g., call by reference

var obj1 = {x:1, y:2}

var obj2 = obj1

* **Conditional Statements:**
* In JavaScript we have the following conditional statements:
* Use if to specify a block of code to be executed, if a specified condition is true
* Use else to specify a block of code to be executed, if the same condition is false
* Use else if to specify a new condition to test, if the first condition is false
* Use switch to specify many alternative blocks of code to be executed
* **The if Statement**

Syntax

if (condition) {  
  //  block of code to be executed if the condition is true  
}

* **The else Statement**

Use the else statement to specify a block of code to be executed if the condition is false.

Syntax:

if (*condition*) {  
  //  block of code to be executed if the condition is true} else {  
  //  block of code to be executed if the condition is false}

* **The else if Statement**

Use the else if statement to specify a new condition if the first condition is false.

Syntax:

if (*condition1*) {  
  //  block of code to be executed if condition1 is true} else if (*condition2*) {  
  //  block of code to be executed if the condition1 is false and condition2 is true  
} else {  
  //  block of code to be executed if the condition1 is false and condition2 is false}

* **The JavaScript Switch Statement**

Use the switch statement to select one of many code blocks to be executed.

Syntax:

switch(expression) {  
  case x:  
    *// code block*    break;  
  case y:  
    *// code block*    break;  
  default:  
    // code block  
}

* **Looping statements:**

JavaScript supports different kinds of loops:

* for - loops through a block of code a number of times
* for/in - loops through the properties of an object
* for/of - loops through the values of an iterable object
* while - loops through a block of code while a specified condition is true
* do/while - also loops through a block of code while a specified condition is true

## **The For Loop: [returns the index]**

The for loop has the following

syntax:

for (*statement 1*;*statement 2*;*statement 3*) {  
  // *code block to be executed*  
}

## **The For/In Loop: [returns the index]**

The JavaScript for/in statement loops through the properties of an Object:

### Syntax:

for (key in object) {  
  // *code block to be executed*  
}

## **The For/Of Loop: [returns the element]**

The JavaScript for/of statement loops through the values of an iterable object.

It lets you loop over iterable data structures such as Arrays, Strings, Maps, NodeLists, and more:

### Syntax:

for (variable of iterable) {  
  // *code block to be executed*  
}

## **The While Loop**

The while loop loops through a block of code as long as a specified condition is true.

### Syntax:

while (condition) {  
*// code block to be executed*  
}

## **The Do/While Loop**

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

### Syntax

do {  
*// code block to be executed*}  
while (condition);

* **Break and continue:**

**Break statement:** The break statement is used to jump out of a loop. It can be used to “jump out” of a switch () statement. It breaks the loop and continues executing the code after the loop.

**e.**g., for (let index = 0; index < 10; index++) {

    if (index == 5) {

        break

    }

    console.log(index);

}

**continue statements:**It used to skip one loop iteration with or without a label reference.

**e.**g., for (let index = 0; index < 10; index++) {

    if (index == 5) {

        continue

    }

    console.log(index);

}

* **Program execution in memory:**
  + Whenever we run our js code a global execution context will get created
  + Js engine will create a default object in the context and the name of the objet is window
  + Along with the window object js engine creates default variable and name of the variable is this
  + In global level object is equal to variable i.e., window=this
  + When you run a code both interpretation and execution occur at the same time

|  |  |
| --- | --- |
| **CREATION PHASE** | **EXECUTION PHASE** |
| 1. Memory allocation for all the variables | Line by line execution |
| 1. Memory allocation for the functions | Assignment of actual values to the variables |
| 1. Memory allocation for the entire code | Executing the functions |
| 1. In creation phase variable will be created to in memory with default value undefined | Executing the code |

* Function execution in memory:

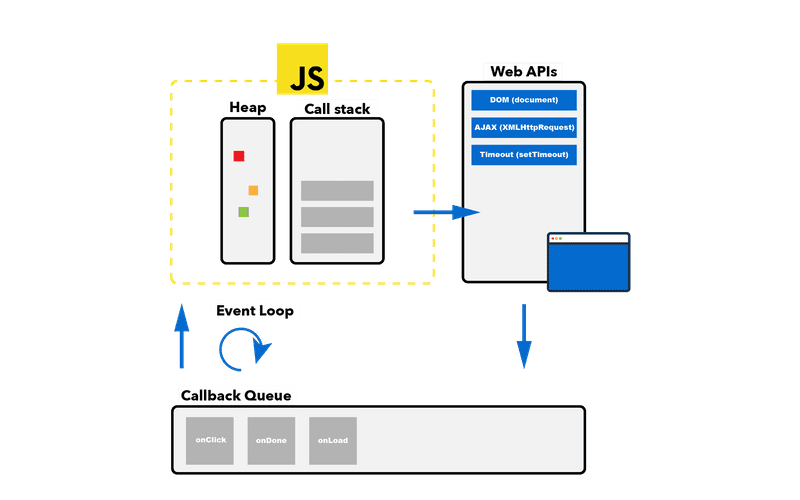
In JavaScript function the concept of function execution in memory is similar to program execution in memory

1. **Functions:**

**What is a Function ?**

A **Function** is a block of code that is designed to perform a task and executed when it is been called or invoked.

* There are 3 ways of writing a function in JavaScript:
* Function Declaration
* Function Expression
  + Arrow Function
* Inbuilt methods in JavaScript:
  + 1. Call()
    2. Apply()
    3. Bind()
* JavaScript call stack:



* Closures:

Inner function still having an access to the variable that has been declared in outer function even after the execution of outer function is called

function addto(passed){

    return function add(inner){

        return passed + inner

    }

}

addthree = addto(3)

console.log(addthree(2))

* callback function:

passing one function as a reference to the other function and executing/invoking that function in other function is called

// fun1(fun2)

// function fun1(a){

//     console.log("inside the fxn 1")

//     a()

// }

// function fun2(){

//     console.log("inside the fxn 2")

// }

// fun1(fun2)

// else.g,2

// E\_commerece(paymentresult)  //TypeError: paymentresult is not a function

function E\_commerece(paymentresult) {

    console.log("before payment");

    var result = paymentresult()

    if (result == "successful") {

        console.log("dispatch the product");

    }

    else if (result == "unsuccessful") {

        console.log("payment is unsuccessfull" );

    }

}

var paymentresult = () =>{return ("unsuccessful")}

// E\_commerece(paymentresult)

1. **Objects**:

JavaScript object is an entity with properties and values,

Each properties are separated by coma(,) and property and its value are separated by colon (:).

Different ways of creating an object in JavaScript:

* + 1. **Using object literals:**

//creating js objects with object literal

let car = {

    name : 'GT',

    maker : 'BMW',

    engine : '1998cc'

};

//property accessor

console.log(car.name); //dot notation

console.log(car['maker']); //bracket notatio

* + 1. **Object.create() method:**

**const coder = {**

**isStudying : false,**

**printIntroduction : function(){**

**console.log(`My name is ${this.name}. Am I studying?: ${this.isStudying}`);**

**}**

**};**

**const me = Object.create(coder);**

**me.name = 'Mukul';**

**me.isStudying = true;**

**me.printIntroduction();**

* + 1. **Creating object with a constructor:**

//simple function

function vehicle(name,maker,engine){

    this.name = name;

    this.maker = maker;

    this.engine = engine;

}

//new keyword to create an object

let car  = new vehicle('GT','BMW','1998cc');

//property accessors

console.log(car.name);

console.log(car.maker);

console.log(car['engine']);

* **prototype property**:

each and every object in JavaScript has a prototype property but it is accessible only for the objects that has been created by constructor function

it is mainly used to modify (add/delete) the properties of an object

e.g.

<script>

// function constructor

function Person(name, job, yearOfBirth){

this.name= name;

this.job= job;

this.yearOfBirth= yearOfBirth;

}

// adding calculateAge() method to the Prototype property

Person.prototype.calculateAge= function(){

console.log('The current age is: '+(2019- this.yearOfBirth));

}

console.log(Person.prototype);

// creating Object Person1

let Person1= new Person('Jenni', 'clerk', 1986);

console.log(Person1)

let Person2= new Person('Madhu', 'Developer', 1997);

console.log(Person2)

Person1.calculateAge();

Person2.calculateAge();

</script>

* + 1. **Using es6 classes:**

**//using es6 classes**

**class Vehicle {**

**constructor(name, maker, engine) {**

**this.name = name;**

**this.maker = maker;**

**this.engine = engine;**

**}**

**}**

**let car1 = new Vehicle('GT', 'BMW', '1998cc');**

**console.log(car1.name); //GT**

* **This keyword:**

Console.log(this)

// var a = "msg"

// function fxn1(){

//     // console.log(this);

//     console.log(`the value is ${this.a}`);

// }

// fxn1()

var bike = {

    make:"hero M",

    model:"xpulse",

    fxn2:function(){

        console.log(this.make+":"+this.model);

    }

}

var bike = {

    make:"hero M",

    model:"xpulse",

    fxn2:function(){

console.log(this)

        console.log(this.make+":"+this.model);

var self = this

function fxn3() {

console.log(self)

     }

fxn3()

}

* **Inheritance:**

**Deriving properties and methods of one object to other object is called,**

**JavaScript follows prototypal inheritance, each and every object in JavaScript is derived from a base object**

**Inheritance can be achieved by \_\_proto\_\_**

var bike = {

    make:"hero",

    model:"xpulse",

    fxn2:function(){

        console.log(this.make+":"+this.model);

    }

}

var bike2 = {

    make:"yamaha ",

    model:"mt\_15",

}

bike2.\_\_proto\_\_= bike

console.log(bike2);

* **Arrays:**

In JavaScript, array is a single variable that is used to store different elements. It is often used when we want to store list of elements and access them by a single variable. Unlike most languages where array is a reference to the multiple variable, in JavaScript array is a single variable that stores multiple elements.

**Declaration of an Array**  
There are basically two ways to declare an array.  
**Example:**

var House = [ ]; // method 1  
var House = new array(); // method 2

**An array in JavaScript can hold different elements:**

**Accessing Array Elements:**

**Array methods:**

* + - Array.concat()
    - Array.every()
    - Array.some()
    - Array.forEach()
    - Array.index() and Array.lastIndex()
    - Array.map()
    - Array.filter()
    - Array.push()
    - Array.unshift()
    - Array..pop()
    - Array.shift()
    - Array.reduce() and Array.reduceRight()
    - Array.reverse()
    - Array.sort()
    - Array.toString()
* For of and for in loops in arrays:
  + - Difference between for, for\_of and for\_in in implementing an array as well as in objects

|  |  |  |
| --- | --- | --- |
| for loop | forEach | for of |
| Does not work with object | Does not work with object, only use with arrays | Does not work with object |
| Does not ignore empty elements | Ignores empty elements | Does not ignore empty elements |
| break statement is supported | break statement is not supported coz it’s a method | break statement is supported |
| Ignores extra properties which does not have index | Ignores extra properties which does not have index | Ignores extra properties which does not have index |

|  |
| --- |
| for in |
| Works with object and arrays |
| Ignores empty elements |
| break statement is supported |
| Does not ignore extra properties which does not have index |

* Date object:
  + Declaration :

Var date = new Date()

* + Date object methods:

|  |  |
| --- | --- |
| getFullYear() | Get the **year** as a four digit number (yyyy) |
| getMonth() | Get the **month** as a number (0-11) |
| getDate() | Get the **day** as a number (1-31) |
| getHours() | Get the **hour** (0-23) |
| getMinutes() | Get the **minute** (0-59) |
| getSeconds() | Get the **second** (0-59) |
| getMilliseconds() | Get the **millisecond** (0-999) |
| getTime() | Get the time (milliseconds since January 1, 1970) |
| getDay() | Get the weekday as a number (0-6) |
| Date.now() | Get the time. ECMAScript 5. |

* Math object:
  + Math object properties:

Math.E        // returns Euler's number  
Math.PI       // returns PI  
Math.SQRT2    // returns the square root of 2  
Math.SQRT1\_2  // returns the square root of 1/2  
Math.LN2      // returns the natural logarithm of 2  
Math.LN10     // returns the natural logarithm of 10  
Math.LOG2E    // returns base 2 logarithm of E  
Math.LOG10E   // returns base 10 logarithm of E

* Math object methods:

Math.round()

Math.ceil()

Math.floor()

Math.abs()

Math.random()

Math.max()

Math.min()

* JSON:

JSON(JavaScript Object Notation) :

JSON is a format for sharing data.

Json methods:

JSON.stringify()

JSON.parse()

JSON Data Types :

In JSON, values must be one of the following data types

a string, a number, an object, an array, a boolean, null

JSON values**cannot**be one of the following data types:

a function, a date, undefined

* **BOM**: [Browser Object Model]:

The Browser Object Model (BOM) allows JavaScript to "talk to" the browser.

**WindowObject** :

The window object is supported by all browsers. It represents the browser's window tab.

**Properties**:

innerHeight, innerWidth.

**Methods**:

prompt, alert, confirm, open, close, console.

**Location Object**:

href, hostname, pathname, protocol, port.

**History Object**:

back(), forward().

* DOM:[Document Object Model]

The Document Object Model (**DOM**) is a programming interface for HTML and XML documents, which can be modified with a scripting language such as **JavaScript.**

**Methods**:

write(), createElement( element ), getElementById( id name ), getElementsByTagName( tag name ), getElementsByClassName( class name ),querySelector( tag or id or class ),

querySelectorAll( tag or id or class ), appendChild( element ).

**Properties**:

firstChild, firstElementChild, lastChild, lastElementChild, removeChild, className, classList, childNodes.

**DOM Events** :

onclick, onmouseover, onmouseout, onkeyup, onkeydown.

* ES6 FEATUERS:
  + Let var and const keywords
  + Template literals
  + Array and Object destructuring
  + Spread and rest operators
  + Promises
  + class