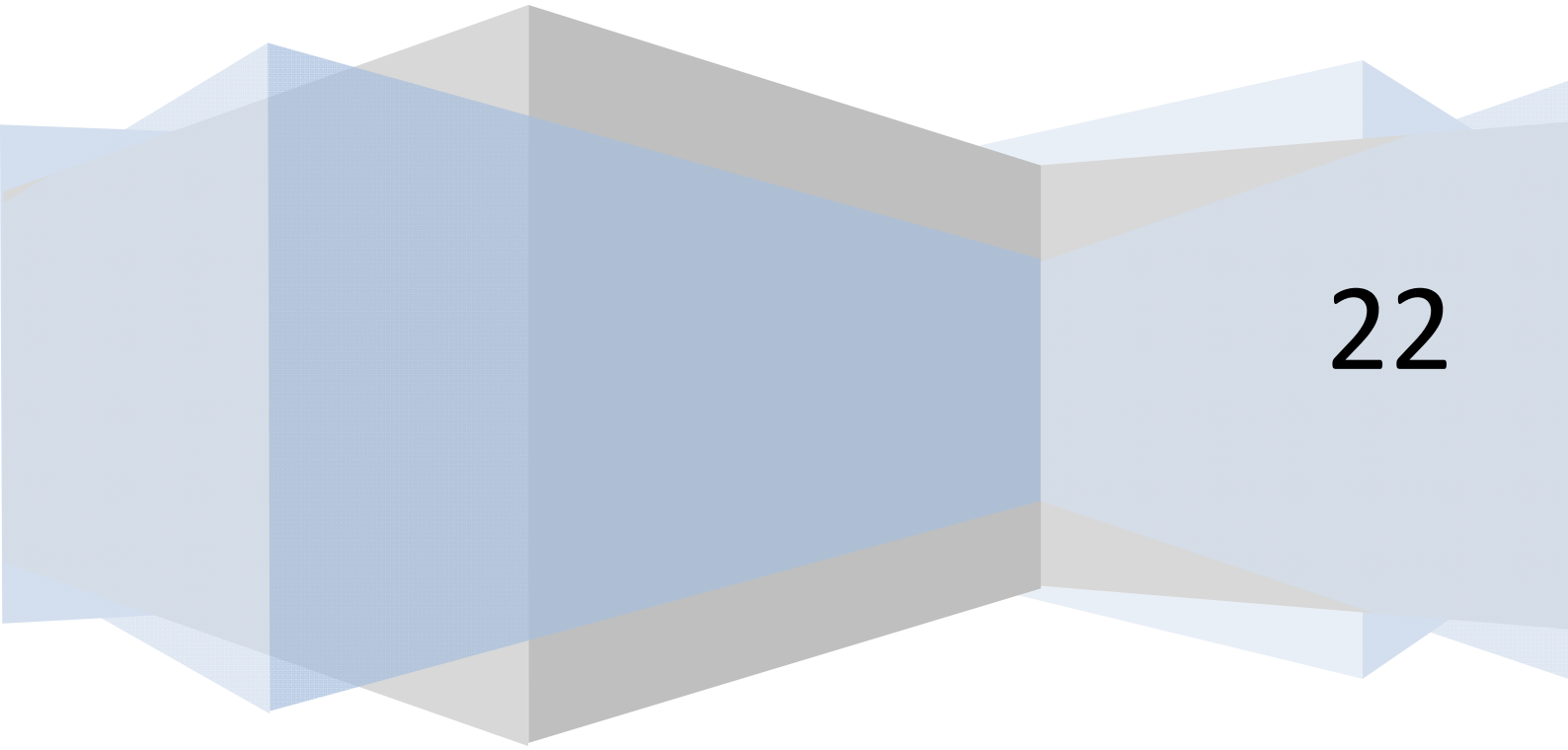
TestYantra Software Solutions pvt ltd

JAVASCRIPT

Training notes

Abhishek K H



22

DAY-1

Programming language

A language used to give instructions to the machine or computer to get the work done.

We have three types of programming language:

1. *Machine language:*
   - It is low level language made up of binary numbers (0's and 1's)
   - It is also called as machine code or object code
   - Computer only understands this language

2. *Assembly level language*
   - Assembly level language is a type of low-level programming language that is intended to communicate directly with a computer's hardware
   - Assembly level languages are readable

3. *High level language*
   - It is a programming language which humans can understand and user friendly
   - Example : c, c++, java, javascript etc.

****

Translators

→What is a translator?

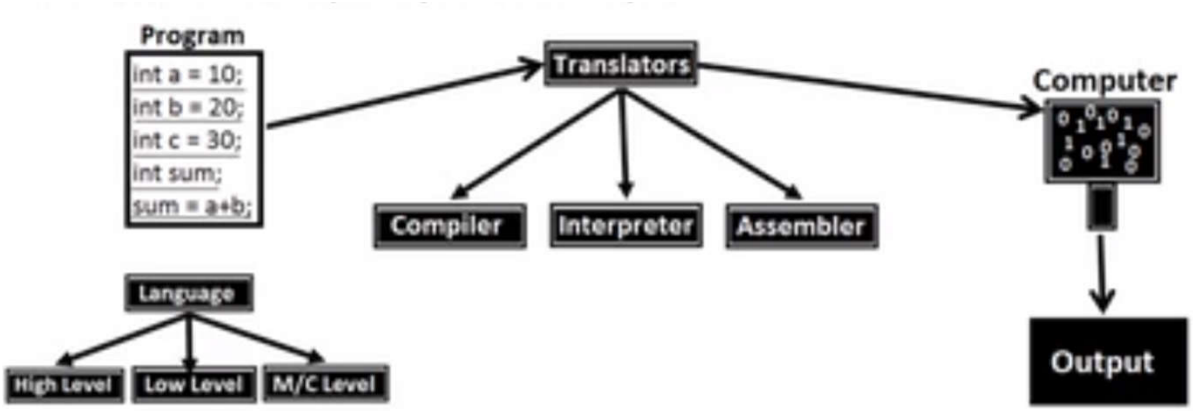
Translators convert the written word from one language to another.

3 different types of translators:

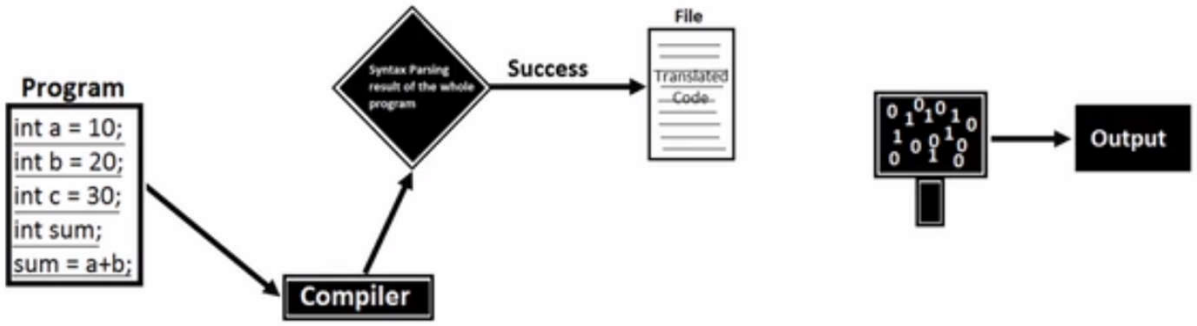
1. Compilers
2. Assemblers
3. Interpreters

→What is a program?

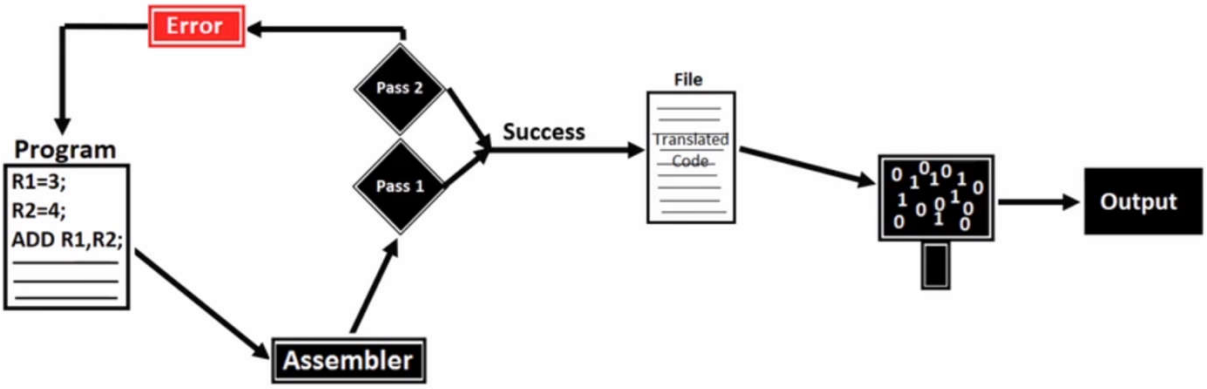
Set of Instructions given to a machine to do a work.



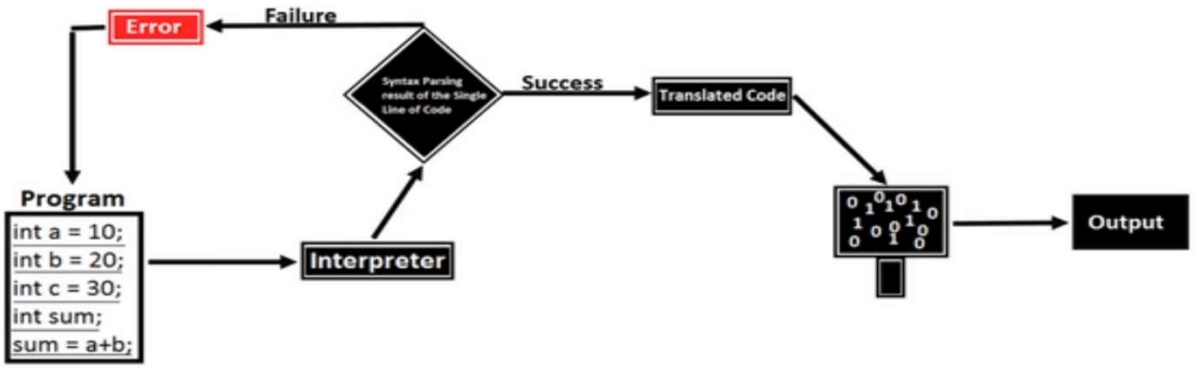
How Compiler works?



How Assemblers works?



How Interpreter works?



Difference between Compiler, Assembler, Interpreter

Compiler	Assembler	Interpreter
- Converts High Level Language into Machine Level Language	- Converts Low Level Language into Machine Level Language	- Converts High Level Language into Machine Level Language
- Syntax Parsing of entire input File in only one Pass	- Syntax Parsing of entire input File in One pass or Two pass	- Syntax Parsing of the input File line by line
- Errors will be thrown for entire input file	- Errors will be thrown for entire input file	- Errors will be thrown for a particular line of code
- A File is Generated for a Successful compilation	- A Object File is Generated for a Successful compilation	- A file is not generated for a successful interpretation
- Execution is faster	- Execution is faster	- Execution is slower
- Efficient for huge set of programs	- Efficient for huge set of programs	- Efficient for Small Pieces of code

****

History of JavaScript

- 1990's
  Internet for advertisements.
  HTML author→develop static web pages for advertisements and post it on internet so people can access.

- 1993
  As Browser was necessary to view the webpages, "Netscape" company created their own browser called "Netscape Navigator". To create completion Microsoft created its own browser called "Internet Explorer" and made it free whereas the Netscape Navigator was paid browser. If people wanted to use Netscape Navigator, they have to buy whereas people can use Internet Explorer as a free browser. So people opted for Internet Explorer.

- Netscape company hired "Brenden Eich" to develop a new programming language to make webpages interactive. In 10 days a new programming language was ready and named it as "MOCHA". It was renamed as "LIVESCRIPT" during the launch. But for the marketing purpose it was again renamed it as "JAVASCRIPT"

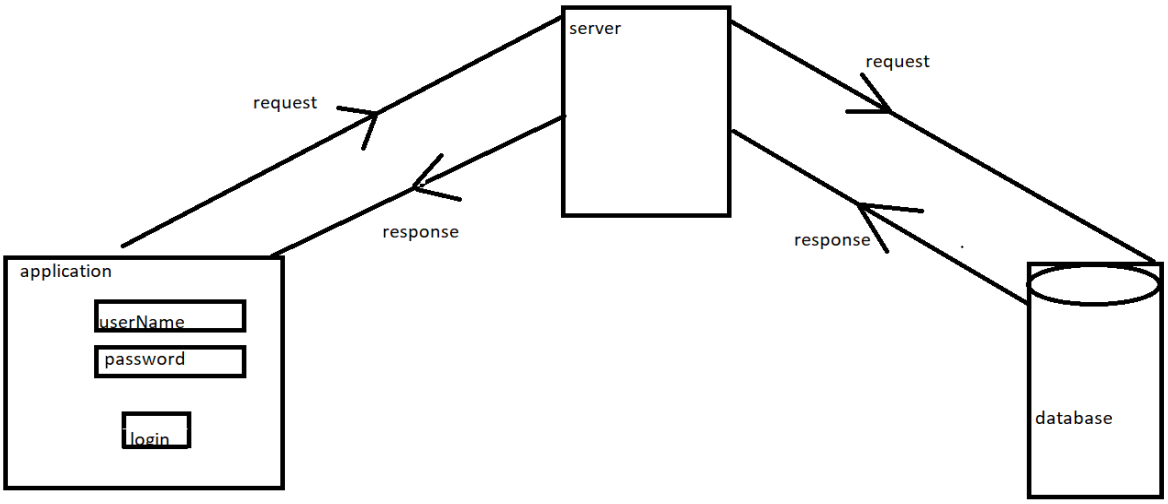
- In 1995 JavaScript was released in "Netscape Navigator2". So in order to use JavaScript people need to buy Netscape Navigator.
  But later Microsoft reverse engineered and built the similar language and named it as "Jscript".

- Netscape went to **ecma** to create standards. Ecma used JavaScript and created a standard and named it as ecmascript (es)
  1997 – es 1 was launched
  1999 – Netscape got sold and took the source code and made it as open source. The reincarnation of the Netscape Navigator is "Firefox"
  2009 - es5 was launched (can build desktop app, websites app, server app)→called "modern javascript"
  2015 – es6 was launched (even more important and new features were added)

****

JavaScript

- JavaScript is a scripting language (*interpretation and execution occurs at the run time*)
- It is also called *interpreted* programming language (as it uses interpreter for code translations)
- JavaScript is *case sensitive*
- using JavaScript one can *create interactive web pages /dynamic web pages*
- mainly developed for *client side validation*



****

NOTE

Why JavaScript file is called as "Full stack" data file?

Can be used in client side, server side, and even data base

- ➔ 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 Couch DB

****

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.

JQuery, Load#, Bootstrap, _js Etc.

****

Frameworks built using JavaScript

JS + Enhancements – Node JS (used for developing web applications)

JS + Enhancements – React JS (used for developing web applications)

JS + Enhancements – React Native (used for developing client server applications)

JS + Enhancements – angular JS (used for developing single page web applications. Eg: gmail, google maps)

JS + Enhancements – electron JS (used for developing standalone applications)

JS + Enhancements – Tensor flow (AI, machine learning applications)

****

Tokens

Smallest unit of the code is called tokens

1. Keywords
2. Identifiers
3. Literals
4. Operators
5. Separators

➔ **Keywords**

Keywords are the predefined words with predefined meaning in it (developers has defined their meaning already)

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

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**

- Values what we store inside the memory allocated.

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

****

JS Engine

- Engine is basically an Interpreter
- Each and every browser has JS engine

Eg: Chrome browser → v8
Mozilla Firefox →SpiderMonkey
IE → Chakra
Safari →JavaScriptCore

****

Declaring a variable in JavaScript

Variables are the *containers that you can store the values in it/* containers that hold reusable data

Syntax :

var(keyword) name(identifier) = value

Followed by

```
var a          // declaration
a = 10         // Initialization
console.log(a) // Utilization
```

****

Datatypes

There are 2 types of datatypes

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

JAVASCRIPT

1.Primitive datatypes/immutable:

- undefined
- Boolean
- Number
- String
- null

2. Non-primitive datatypes (object references)/mutable:

- Objects
- Functions
- Arrays
- Date
- Math

****

Flow control statements

1. Conditional Statements
2. Looping Statements

DAY-2

Loose typed:

NOTE 1: JS is a dynamically typed language.

Variable is loosely typed

String name = “abhi” ➔JAVA

var name = “abhi”; ➔JS (name is of var datatype)

name = 11; (name is of number datatype)

(ie., during the runtime, based on the input the datatype will be decided.)

//program

```
a=10
b= "10"
c = 'abhi'
console.log(typeof(a))
console.log(typeof(b))
console.log(typeof(c))
```

//output

```
PS D:\JAVASCRIPT\VSScripts\javascriptpractice\NOTE\looslytyped> node loose.js
number
string
string
```

CHANGE DATATYPE:

NOTE 2: To change the datatype explicitly

//program

```
a=10
console.log(typeof(a))
console.log(typeof(String(a)))

console.log(typeof(String(a)))

name = "abhi"
console.log(typeof(name))
console.log(typeof(Number(name)))
a = 10;
console.log(typeof(a))
```

//output

PS D:\JAVASCRIPT\VSScripts\javascriptpractice\NOTE\changeDatatype> node changeDatatype.js

number

string

PS D:\JAVASCRIPT\VSScripts\javascriptpractice\loose> node loose.js

number

string

PS D:\JAVASCRIPT\VSScripts\javascriptpractice\loose> node loose.js

number

string

string

number

DOUBLE AND TRIPLE EQUALS:

NOTE 3: Difference b/w == and ===

//program

```
a=10
b="10"
if(a==b){
    console.log("a and b are the same")
}
else
{
    console.log("a and b are not the same")
}
```

//output

PS D:\JAVASCRIPT\VSScripts\javascriptpractice\NOTE\double and triple equals> node equals.js

a and b are the same

INCREMENT:

NOTE 4: pre increment and post increment

Pre increment will increment and then initialise

Post increment will initialise and then increment

//program

```
a=10;
res=a++
console.log(res)
b= 20
res1=++b
console.log(res1)
```

//output

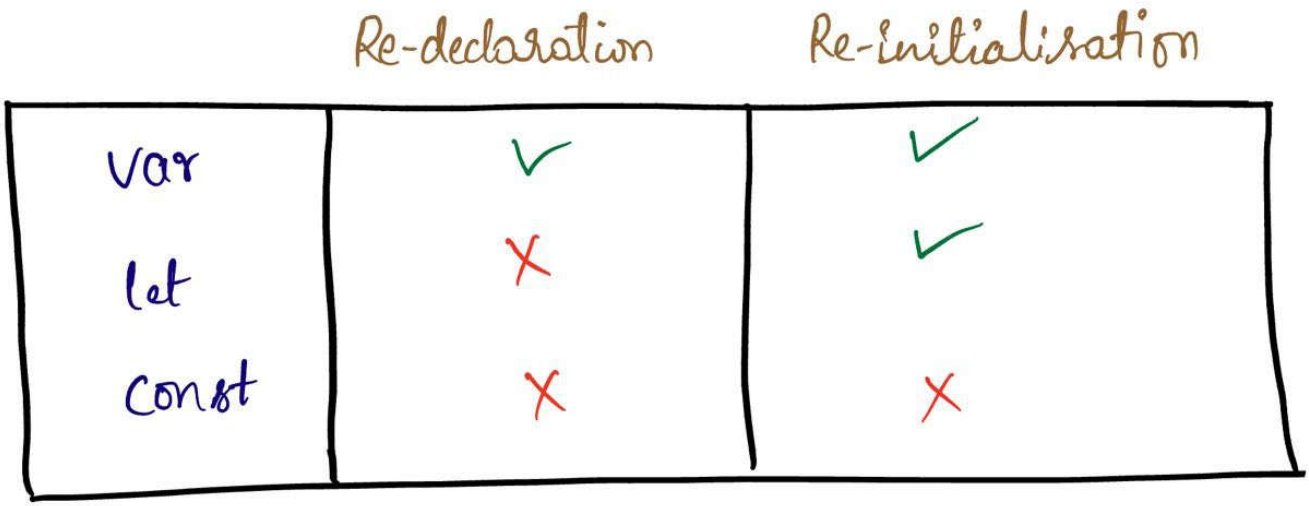
```
PS D:\JAVASCRIPT\VSScripts\javascriptpractice\NOTE\increment operator> node increment.js
10
21
```

Different types of variable declarations:

1.var

2.let

3.const



//sample program

```
var a=10
var a=11
console.log(a)

let b=20
//let b=21 //Identifier 'b' has already been declared
```

```
b=21
console.log(b)

const c= 30
//const c=31 // Identifier 'c' has already been declared
//c=31 //TypeError: Assignment to constant variable.
console.log(c)
```

//output

PS D:\JAVASCRIPT\VSScripts\javascriptpractice\varLetConst> node varLetConst.js

11

21

30

JS Execution:

1. **on browser**
   1.inline embedded

   Syntax: <script></script>

```
<!DOCTYPE html>
<html lang="en">
<head>

    <title>Document</title>
</head>
<body>
    <script>
       document.write("abhishek")
    </script>

</body>
</html>
```

   2.external embedded

   <script src = “filepath.js”></script>

```
<!DOCTYPE html>
<html lang="en">
<head>

    <title>Document</title>
</head>
<body>
    <script src = "./looping.js"></script>
    <script src = "../loose.js"></script>

</body>
</html>
```

2. **off browser**
   By using NodeJS.

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**

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
}
```

//programs on if else:

```
a=true
b="true"
if(a===b)
{
    console.log("it is equal")

}
else{
    console.log("not equal")
}

a=70
b="tru"
if(a==b)
{
    console.log("it is equal")

}
else{
    console.log("not equal")
}

a=true
b=false
if(a=b)
{
    console.log("it is equal")

}
else{
    console.log("not equal")
}
```

➔ **The else if Statement**

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
}
```

//programs on else if:

```
a=1
b="1"
if(a==b)
{
    console.log("it is equal")
}
else if(a===b)
{
    console.log("it is equal")
}
else{
    console.log("it is not equal")
}
```

➔ **The JavaScript Switch Statement**

Syntax

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

//programs on switch:

```
i=1
switch (i) {
    case 1:
       console.log("i value is 1")
        break;
    case 2:
        console.log("i value is 2")
        break;
    default:
        console.log("i avlue can be only 1 and 2");
        break;
}
```

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 itterable 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]

Syntax

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

```
//for loop
for(var i=1; i<=10;i++)
{
console.log(i*2)
}
```

➔ The For/In Loop: [returns the index]

Syntax

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

```
var providenceSMS =[2022,"Generic Utilities",'object repository',12,13,true,false]
for (var iterator of providenceSMS) {
    console.log(iterator+" "+providenceSMS[iterator])
}
```

```
PS D:\JAVASCRIPT\VSScripts\javascriptpractice> node Looping\forin.js

0 2022

1 Generic Utilities

2 object repository

3 12

4 13

5 true

6 false
```

➔ The For/Of Loop: [returns the element]

Syntax

In this for of loop we will not be getting the index and we only get values.

```
for (variable of itterable) {
  // code block to be executed
}
```

```
var providenceSMS =[2022,"Generic Utilities",'object repository',12,13,true,false]
for (var iterator of providenceSMS) {
    console.log(iterator+" "+providenceSMS[iterator])
}
```

2022 undefined

Generic Utilities undefined

object repository undefined

12 undefined

13 undefined

true undefined

false undefined

```
var providenceSMS =[2022,"Generic Utilities",'object repository',12,13,true,false]
for (const iterator of providenceSMS) {
    console.log(iterator)
}
```

2022

Generic Utilities

object repository

12

13

true

false

➔ The While Loop

Syntax

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

```
//while
var i=0
while(i<=10){
    console.log(i+"=while")
    i++
}
```

➔ 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);
```

```
//do while
var j=1
do
{
 console.log(j+"=do while")
 j++
}
while(j<10);
```

BREAK AND CONTINUE:

break statement: It is used to jump out of the loop it can be used to jump out of your switch statement. It breaks the loop and continues the execution after the loop.

Continue: It is used to skip one iteration of the statement(if the condition is not satisfied).

```
var abhi=[11,'hi',7,"abhi"]
for(var index=0;index<abhi.length;index++){
   if(index<2){
    console.log("break continue")
    continue
   }

}
console.log(abhi[index]+" "+index);

//output :
good evening
good evening
undefined 4


var abhi=[11,'hi',7,"abhi"]
for(var index=0;index<abhi.length;index++){
   if(index<2){
    console.log("break continue")
    break
   }

}
console.log(abhi[index]+" "+index);
//output :
break continue
11 0
```

Program execution in memory

- After the execution of JavaScript code, a global execution context will be created.
- In global execution context, a global object will be created named "window".
- Along with window object, global variable will be created. The global variable is called as *"this"*
- At global level, *window===this* (*window is strictly equal to this*).

There are two phases:

- ✓ Creation phase
- ✓ Execution phase

Creation phase:

- During the creation phase memory will be created for the entire code and the name of the memory is created. This name will be variable name or function name.
- The variables will be assigned as undefined by default.
- Memory will be created for the implementations of the functions.
- The code is ready to execution phase

Execution phase:

- The variables values will be overridden from undefined to actual values.
- Functions will get executed if it is called in this execution phase.

Function execution in a memory

- Whenever the function is called, a function execution context will be created on top of global execution context in call stack.
- After the function is executed, the function execution context will get popped off from the *call stack.*

Variable hoisting:

Whenever we execute a JavaScript code,

- In creation phase: a separate memory will be created for all the variables and by default, undefined will be assigned to all the variables.
- In execution phase: The variables values will be overridden from undefined to actual values

Function hoisting:

Whenever we execute a JavaScript code, in case of function

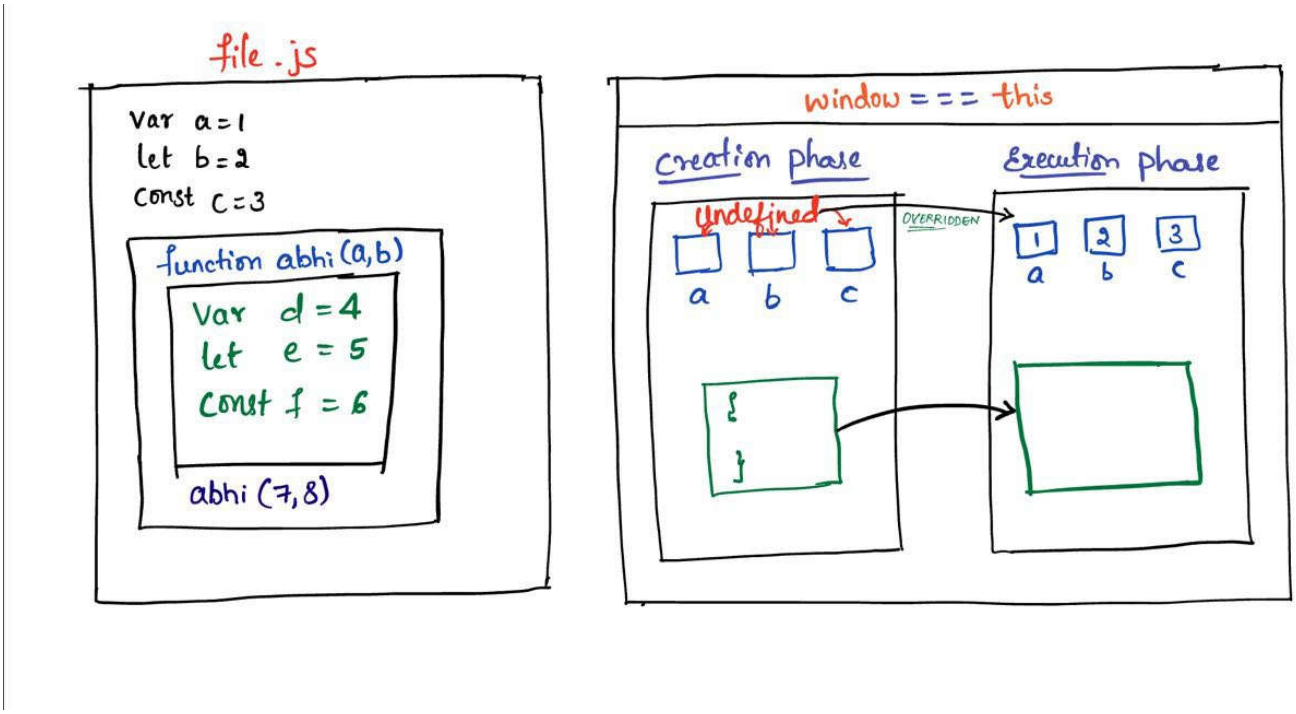
- In creation phase: a separate memory will be created for all the functions and the function implementation will be stored inside the memory.

- In the execution phase: If the function is invoked in the execution phase, the function will be executed.

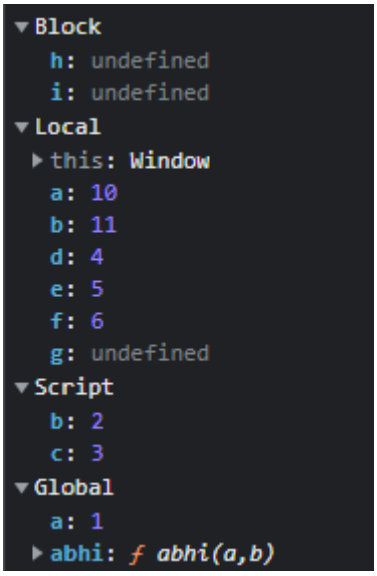
Program execution pictorial representation

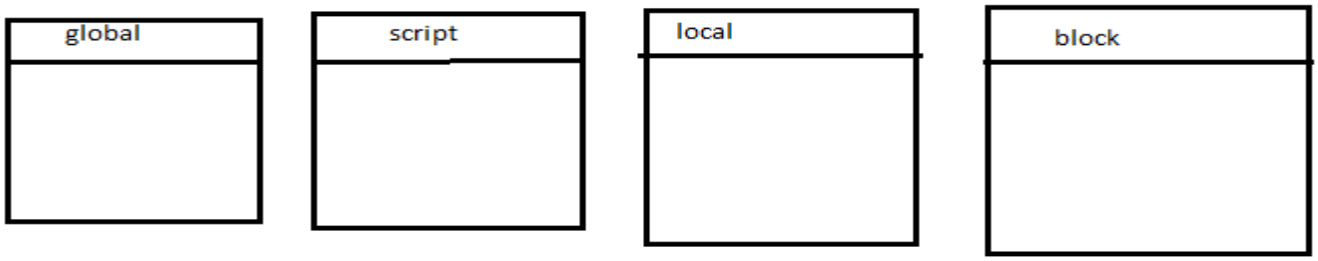
```
var   a = 1
let   b = 2
const c = 3

function abhi(a,b)
{
    var   d = 4
    let   e = 5
    const f = 6
}
abhi(7,8)
```



Different blocks which will get created during the program execution





PROGRAMS ON var, let, const in blocks and functions to determine whether it is global/script/local/block

Scenario 1 : function inside the block

```
var a = 1 //global
let b = 2 //script
const c = 3 //script


    {
        var g = 7//global
        let h = 8//block
        const i = 9//block

        function abhi(a,b)
        {
            var d = 4//local
            let e = 5//local
            const f = 6//local
        }
        abhi(10,11) //local

    }
{
    var j = 12//global
    let k = 13//block
    const l = 14//block
}
```

Scenario 2 : block is inside the function

```
var a = 1 //global
let b = 2 //script
const c = 3 //script

function abhi(a,b)
{
    var d = 4//local
    let e = 5//local
    const f = 6//local
    {
        var g = 7//local
        let h = 8//block
        const i = 9//block
    }
}
abhi(10,11) //local

{
    var j = 12//global
    let k = 13//block
    const l = 14//block
}
```

Scenario 3: block inside the block its inside the function and its inside the function and it is inside the block

```
var a = 1 //global
let b = 2 //script
const c = 3 //script


    {
        var g = 7//global
        let h = 8//block
        const i = 9//block

        function abhi(a,b)
        {
            var d = 4//local
            let e = 5//local
            const f = 6//local

        function shek(m,n)
        {
            var o = 17//local
            let p = 18//local
            const q = 19//local
            {
                var r = 20//local
                let s = 21//block
                const t = 22//block
```

```
            {
                var j = 12//local
                let k = 13//block
                const l = 14//block
            }
        }
    }
    shek(15,16)//local
    }
    abhi(10,11) //local

}
```

DAY-3 29.09.2022

Javascript codes url: https://github.com/abhi11spark/ProvidenceSMS_js.git

Mistake:
{
Abhishek Gowda@ABHISHEK-K-H-96 MINGW64 /d/JAVASCRIPT/VSScripts/javascriptpractice
$ git config--user.name abhi11spark
git: 'config--user.name' is not a git command. See 'git --help'.

Abhishek Gowda@ABHISHEK-K-H-96 MINGW64 /d/JAVASCRIPT/VSScripts/javascriptpractice
$ git config--global user.name "abhi11spark"
git: 'config--global' is not a git command. See 'git --help'.

}

Procedure to commit to github:

1) **$git config --global user.name abhi11spark :** This command sets the username

   Abhishek Gowda@ABHISHEK-K-H-96 MINGW64 /d/JAVASCRIPT/VSScripts/javascriptpractice
   $ git config --global user.name "abhi11spark"

2) **$git config --global user.** photos.pdfs825@gmail.com : This command sets the user email which helps in tracking the commit/ merge activities

   Abhishek Gowda@ABHISHEK-K-H-96 MINGW64 /d/JAVASCRIPT/VSScripts/javascriptpractice
   $ git config --global user.email photos.pdfs825@gmail.com

3) **$git init** : This command creates a hidden directory, initialization takes place

   Abhishek Gowda@ABHISHEK-K-H-96 MINGW64 /d/JAVASCRIPT/VSScripts/javascriptpractice
   $ git init
   Initialized empty Git repository in D:/JAVASCRIPT/VSScripts/javascriptpractice/.git/

4) **$git status** : This command is used to check the status of the files which are need to push, current status is No commits yet

   Abhishek Gowda@ABHISHEK-K-H-96 MINGW64 /d/JAVASCRIPT/VSScripts/javascriptpractice (master)
   $ git status
   On branch master

   No commits yet

   Changes to be committed:
    (use "git rm --cached <file>..." to unstage)
        new file: NOTE/changeDatatype/changeDatatype.js
        new file: NOTE/double and triple equals/equals.js
        new file: NOTE/increment operator/increment.js
        new file: NOTE/looslytyped/loose.js
        new file: block1.html
        new file: blocks.html
        new file: blocks1.js
        new file: blocks2.html
        new file: blocks2.js

```
new file:   day1.js
new file:   demo.js
new file:   elseif.js
new file:   ifelse.js
new file:   learningdifferentblock.html
new file:   learningdifferentblock.js
new file:   looping.js
new file:   loose.js
new file:   onbrowser/day1.js
new file:   onbrowser/demo.js
new file:   onbrowser/externalbrowser.html
new file:   onbrowser/inlinebrowser.html
new file:   onbrowser/looping.js
new file:   onbrowser/loose.js
new file:   switchcase.js
new file:   varLetConst/varLetConst.js
```

5) **$git add .** : This command is used to add all the files to local repository

```
Abhishek Gowda@ABHISHEK-K-H-96 MINGW64 /d/JAVASCRIPT/VSScripts/javascriptpractice (master)
$ git add .
```

6) **$git remote add origin <"git repository url">** : This command is used to add all the files to local repository, connects repository (Remote Github and Local); origin is a nickname assigned to particular remote repository

```
Abhishek Gowda@ABHISHEK-K-H-96 MINGW64 /d/JAVASCRIPT/VSScripts/javascriptpractice (master)
$ git remote add origin https://github.com/abhi11spark/ProvidenceSMS_js.git
```

7) **$git remote -v**➔This command is used to add all codes to global repository

```
Abhishek Gowda@ABHISHEK-K-H-96 MINGW64 /d/JAVASCRIPT/VSScripts/javascriptpractice (master)
$ git remote -v
origin  https://github.com/abhi11spark/ProvidenceSMS_js.git (fetch)
origin  https://github.com/abhi11spark/ProvidenceSMS_js.git (push)
```

8) $git commit –m “commit

```
Abhishek Gowda@ABHISHEK-K-H-96 MINGW64 /d/JAVASCRIPT/VSScripts/javascriptpractice (master)
$ git commit -m "first commit"
[master (root-commit) 327d415] first commit
 25 files changed, 329 insertions(+)
 create mode 100644 NOTE/changeDatatype/changeDatatype.js
 create mode 100644 NOTE/double and triple equals/equals.js
 create mode 100644 NOTE/increment operator/increment.js
 create mode 100644 NOTE/looslytyped/loose.js
 create mode 100644 block1.html
 create mode 100644 blocks.html
 create mode 100644 blocks1.js
 create mode 100644 blocks2.html
 create mode 100644 blocks2.js
 create mode 100644 day1.js
 create mode 100644 demo.js
 create mode 100644 elseif.js
 create mode 100644 ifelse.js
 create mode 100644 learningdifferentblock.html
 create mode 100644 learningdifferentblock.js
 create mode 100644 looping.js
 create mode 100644 loose.js
 create mode 100644 onbrowser/day1.js
 create mode 100644 onbrowser/demo.js
 create mode 100644 onbrowser/externalbrowser.html
 create mode 100644 onbrowser/inlinebrowser.html
 create mode 100644 onbrowser/looping.js
 create mode 100644 onbrowser/loose.js
 create mode 100644 switchcase.js
 create mode 100644 varLetConst/varLetConst.js
```

9) $git push origin master

```
Abhishek Gowda@ABHISHEK-K-H-96 MINGW64 /d/JAVASCRIPT/VSScripts/javascriptpractice (master)
$ git push origin master
Enumerating objects: 31, done.
Counting objects: 100% (31/31), done.
Delta compression using up to 4 threads
Compressing objects: 100% (25/25), done.
Writing objects: 100% (31/31), 3.48 KiB | 297.00 KiB/s, done.
Total 31 (delta 5), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (5/5), done.
To https://github.com/abhi11spark/ProvidenceSMS_js.git
 * [new branch]      master -> master
```

10) $git status

```
Abhishek Gowda@ABHISHEK-K-H-96 MINGW64 /d/JAVASCRIPT/VSScripts/javascriptpractice (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Functions

- It is the block of code
- Used to perform some specific task
- It is reusable

(a) Types of functions

1. Standard function / function declaration
2. Function expression
3. IIF (Immediately Invocable Function)
4. Arrow function

Syntax of functions

```
standard function
=================

syntax
======
function name_of_function(){

//**block of statements**

}

invocation
==========
name_of_function()
```

```
function expression
====================

syntax
======
var name_of_function = function (){

//**block of statements**

}

invocation
============
name_of_function()
```

```
IIF
====

syntax
======
() ()

(encloses a function) (used to invoke the function)
```

```
arrow function
==============

syntax
======
() => {}

invocation
==========
method1: var ref = () => {}
         ref()

method2: IIF
```

Function declaration

- Function declaration is the traditional way to define function.
- We start declaring a function by using the keyword “function”, followed by function name and then we can pass the parameters(optional)

```
function sum(first,last)
{
    console.log(first+" "+last);
}
sum("abhi","gowda")
```

//output : abhi gowda

****Function expression***

- We define a function using a variable and store the returned value in that variable.
- The whole function is an expression and the returned value is stored in variable.
- We use the variable name and call the function.

- When we have to invoke a function in another file (.js file), we have to go for function expression.

```
var sum = function (first,last)
{
    console.log(first+" "+last);
}
sum("abhi","gowda")
```

//output : abhi gowda

IIF (Immediately invocable function)

- This function is rarely used
- ()()==>2 invocations. In 1 invocation we will enclose the function and another invocation symbol is used to invoke the function
- Only one IIF can be used in one .js file. If we use more than one IIF we will get an error

```
( function (first,last)
{
    console.log(first+" "+last);
}) ("abhi","gowda")
```

//output : abhi gowda

```
(sum = function(a,b)
{
    console.log(a+b)
    console.log(arguments[4]);
})(11,55,77,88,99)
(mul = function(a,b)
{
    console.log(a*b)
    console.log(arguments[3]);
})(11,55,77,88,99)
```

//output:

```
PS D:\JAVASCRIPT\VSScripts\javascriptpractice> node Functions\iIF1.js

66

99

D:\JAVASCRIPT\VSScripts\javascriptpractice\Functions\iIF1.js:6

(mul = function(a,b)

^


TypeError: (intermediate value)(intermediate value)(...) is not a function

    at Object.<anonymous> (D:\JAVASCRIPT\VSScripts\javascriptpractice\Functions\iIF1.js:6:1)

    at Module._compile (node:internal/modules/cjs/loader:1126:14)

    at Object.Module._extensions..js (node:internal/modules/cjs/loader:1180:10)

    at Module.load (node:internal/modules/cjs/loader:1004:32)

    at Function.Module._load (node:internal/modules/cjs/loader:839:12)

    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:81:12)

    at node:internal/main/run_main_module:17:47
```

Arrow function

- Arrow function was introduced in ES6 version of JavaScript.
- This function is mainly used for code optimisation.
- In a single line of code, the function returns implicitly.
- When there are multiple lines of code we use brackets. When there are multiple lines of code in the bracket we should write return explicitly to return the value from the function.

```
var sum = (first,last) => first+last
console.log(sum(20,20));
```

//output : 40

PS D:\JAVASCRIPT\VSScripts\javascriptpractice> node Functions\arrowFunction.js

```
const sum = (first,last) => {

    if(first==last){
        return "first is equals last"
    }
    else{
        return "first is not equals last"
    }
}

console.log(sum(20,20));
```

//output: first is equals last

(Interview Q) Arguments array:

```
function sum(a,b) {
console.log(a+b);
console.log(arguments);
}
sum(1,2,3,4,5,6)
```

//output:

3

[Arguments] { '0': 1, '1': 2, '2': 3, '3': 4, '4': 5, '5': 6 }

Note: We can call the particular value in the arguments array using the index.

```
(sum = function(a,b)
{
    console.log(a+b)
    console.log(arguments[4]);
})(11,55,77,88,99)
```

//output:

66

99

Table for an Array Methods:

Sl no.	*Method Name*	*What it does?*	*Arguments*	*Returns*	*Modifies Original array or not*
1	**concat()**	Combines 2 arrays	Array or String	Array	No
2	**push()**	It will add the element at the end of the array	Element	Length of original array	Yes
3	**pop()**	It will remove the last element in the array	Element	Element	Yes
4	**unShift()**	It will add the element in the beginning of the array	Element	Length of modified array	Yes
5	**shift()**	It will remove the first element from the array	Element	Element	Yes
6	**splice()**	It will delete and as well add the given element to the specific index of an array.	Start index, Delete count, new Element	Deleted Element	Yes
7	**slice()**	It will take out part of array	Start index , end index	Part of array	No
8	**every()**	Compares the each and every element of array with specified condition (returns true only if all conditions are specified)	*Function	Boolean	No
9	**some()**	Compares all value and returns true id any one of the element satisfying condition	*Function	Boolean	No
10	**indexOf()**	Returns the index of an element (left to right)	(element, Fromindex)	Index no	No
11	**lastIndexOf()**	Returns the index of an element (right to left)	(element,index)	Index	No
12	**reverse()**	Reverse the array	------	Reversed array	Yes
13	**includes()**	Search the element in an array	Element	Boolean	No
14	**join()**	Joins the element to all the elements present in array.	Element	Concated elements with array	No
15	**forEach()**	Perform operation mentioned in callback function with all the element in array	Callback function (element, index)	Modified array	No
16	**map()**				
17	**filter()**				
18	**sort()**	Sort the array in ascending , descending order based on condition in callback function.	(a,b)	Ascending array if Return(a-b) Descending array if Return (b-a) .	yes
19	**reduce()**	Reduce the array element from total no.			
20	**reduceRight()**				

Table for String Methods:

Sl. no	Method Name	What it does?	Arguments	Returns	Modifies Original array or not
1	length()	Returns the length of the String	No args	Length	No
2	split()	Separates string	Separators	Separated string in an array	no
3	charAt()	Returns the character of specified index no	Index no	Character	No
4	concat()	Conacats element the string	Element	Concated string	No
5	endsWith()	Checks string with last character	Character	Boolean	No
6	includes()	Checks whether a string having specified string or no	String or character	Boolean	No
7	indexOf()	Checks the index of specified String	Character	Index no	No
8	lastIndexOf()		character	Index no	No
9	replace()	Replces a string with given string	(old string ,required string)	Complete changed String	No
10	repeat ()	Repeats the string by specified number of times	Number	Repeated strings	No
11	trim()	Trims the space	No args	String	No

Table for Math Methods:

Sl no	Method Name	What it does?
1	Math.abs()	It will ignore last zeros of decimal and takes absolute value
2	Math.round()	It will round off the number according to decimal value.
3	Math.min(int1,int2,int3)	It will compare the values and gives the minimum value among them.
4	Math.max(int1,int2,int3)	It will compare the values and gives the maximum value among them.
5	Math.floor(int)	It will neglect the decimal value and gives the whole no.
6	Math.ceil()	It will give the next value.
7	Math.random()	It Will generate random number.

***Note: overloading is not possible in the javascript. If we give same method with different parameters we will be doing overriding the method instead of overloading.

Example:

var cellular(“abhi”,”k h”)

var cellular(“shek”,”k h”)

in this case console.log(cellular); //output: shek k h

Data Security:

In the Object the data flow will be unprotected and public so if we have to protect the data of the object we will be using this keyword.