**Day-34 27/09/2022(Tue) Day-1**

JavaScript team Segregation – Put into JavaScript Batch

Introduction-Rules by Rekha Mam

Selenium topics pickup – Daily presentation of selenium topics

Download **Nodejs** from google - Installation

Download url : <https://nodejs.org/en/download/>

Screen record for installation

Download **Visual code** – Installation

**Nodejs** : Provides us the environment to run the javaScripts

**Visual code :** Provides us the environment to write the javaScripts.

It is a standalone source code editor.

Create a repository / folder on desktop 🡪 Name it as js\_sdet or js\_pjtname.

**How to open the VSCode :**

Open the folder and open command prompt by typing in the path bar.

Step1 : In the VSCode Editor, Create folder with name “Basic”

Step2 : Create a file Demo.js in the Basic folder

Step3 : Copy the relative path by right clicking on the file and select copy relative path

Step4 : Click on terminal 🡪 New Terminal 🡪Execute the javascripts here using the command :

**[node <relative path of the javascript file>]**

The output will be displayed in the terminal.

**Types of languages**

There are 3 types of languages.

1. Machine level language
2. Assembly level Language
3. High Level language
4. **Machine level Language:** It is a language readable or understandable only by machines. It contains binary digits/instructions ‘0’ and ‘1’.
5. **Assembly level Language:** This language needs to be translated to machine level language so that the machine can understand and perform the tasks as specified. For this, we required Assemblers as translators to translate the language to machine understandable/level language.
6. **High level Language:**  This language is not understandable by the machine, so this language also needs to be translated with the help of Compiler which is a translator for High level language.

**Translators – Types of translators**

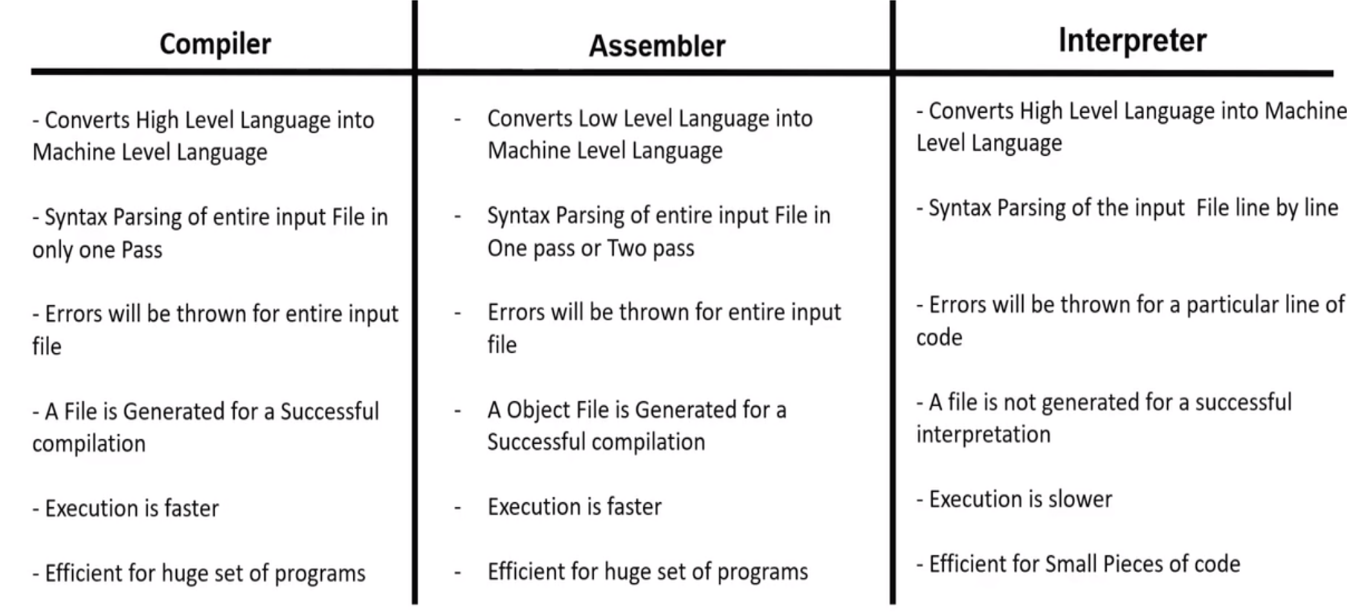
Translators are software components which are used to convert or translate high level/assembly level languages to machine level or low level language.

There are 3 types of translators :

1. Compilers
2. Assemblers
3. Interpreters

These translators are used based on application or the programming languages that is used by the users.

Let us see the features of these three translators in the below table.



**What is JavaScript – About JavaScript**

JavaScript is a programming language which is used to create interactive webpages / dynamic webpages.

It is called as scripted language as the interpretation and execution happens during the runtime.

It uses interpreters as translators for translating the script to machine level language. So it is called as interpreted language.

JavaScript was mainly developed for validation on client side or client side validation.

JavaScript is used for development of client side, server side and database side.

For client side we use Corejs, for webserver side we use Nodejs and for database side we use mongo DB and Couch DB. As JavaScripts can be used on all these three layers for development, JS file is known as Full Stack data file.

Some of the libraries built by using JS are: JQuery, Load#, Bootstrap, \_js Etc. Using these libraries, we can easily convert complex codes to minimal / optimized /simplified code.

We can develop frameworks by adding some enhancements to the javascript. Some of the frameworks are as below:

**NodeJS** – used to develop web applications

**React JS**– used to develop web applications

**React Native**– used to develop client server applications

**angular JS**– used to develop single page web applications. Eg: gmail, google maps

**electron JS**– used to develop standalone applications

**Tensor flow**– used to develop AI, machine learning applications

**Tokens:**

These are the smallest units of the code.

There are 5 types of tokens. They are:

**1.** **Keywords :** These are pre-defined words in the programming language

**Eg:** 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

We have some rules to follow while using the keywords.

a. JavaScript is case-sensitive

b. Keywords should always be lower case

c. We should not use keywords as identifiers.

**2.** **Identifiers :** Identifiers are the names that we give to the variables,methods/functions.

We should not use keywords as identifiers

Identifiers are case sensitive, they should not begin with numbers 0-9, or special characters except for $ and \_.

**3. Literals:** Literals are the values that we store in the variables.

The types of literals include the following:

1. Numbers

2. String

3. Null

4. undefined

5.boolean

**4. Operators:** Operators are symbols used to perform operations between two or more operands.

(+, -, \*,, /, %)

**5. Separators:** Separators are symbols used to terminate or separate the statements in the program.

(,),{,},[,] etc

**JS Engine**

JS Engine is an interpreterused as translator in the browsers. All the browsers have their own JS Engines such as :

For chrome – v8 is the JSEngine

For firefox – SpiderMonkey

For IE – Chakra

For Safari – JavaScript core

**Variable Declaration in JavaScript:**

Variables are used to store values such as numbers, string, Boolean values, etc.

We declare variables in JS as below:

var (keyword) name(Identifier) = value

After declaring the variable, we initialize it with a value and utilize it for the script.

🡪Declaration : var x

🡪 Initialization : x = 100

We can also declare and initialize in single statement : var x = 200

🡪Utilization : eg: console.log(x)

**DataTypes**

Datatype specifies the type of value that a variable contains.

There are 2 types of datatypes: **1.** Primitive datatype and **2.** Non-primitive datatype

Primitive datatype consists of :

-Numbers

-String

-undefined

-null

-boolean

Non-primitive datatypes consists of :

-Objects

-Functions

-Arrays

-Math

-Date

**Day-35 28/09/2022(Wed) Day-2**

**Flow Control Statements:**

These are the statements used to control the flow of execution of statements in the program/code.

There are two types of flow controlling statements. They are :

1. **Conditional Statements :** These statements control the flow of executions based on specified conditions. Below are the conditional statements that are used in the code.

If statement : executes the statements in the block if the specified condition is satisfied

else : executes the other set of statements in the block if the specified condition is not

satisfied.

Else-if : used to specify alternative or other conditions if previously specified conditions

don’t get satisfied.

Switch : executes statement or set of statement or block based on the choice specified by

the operator.

**If Statement:**

var a = 10

var b = 20

if(a>b){

    console.log("a is greater than");

}

**Output:** -

**Else:**

var a = 10

var b = 20

if(a>b){

    console.log("a is greater than");

}

else{

    console.log("b is greater than a");

}

**Output:** b is greater than a

**Else-if**

var a = 10

var b = 10

if(a>b){

    console.log("a is greater than");

}

else if(a==b){

    console.log("a is equal to b");

}

**Output:** a is equal to b

**Switch :**

var grade = "E"

switch(grade){

    case "A": console.log("First class");

    break;

    case "B": console.log("Second Class");

    break;

    case "C": console.log("Third Class");

    break;

    case "D": console.log("Just Passed");

    break;

    default: console.log("Enter proper grade");

    break;

}

**Output:** Enter the proper grade

1. **Looping Statements:** These statements are used to repeat execution of a block of statements with a specified criteria.

Looping statements used are:

**for:** used to repeat the block of code for specified number of times

**for/in**

**for/of**

**while :** repeats the execution of block of code until the specified condition is made false.

**do-while :** repeats the execution of block of code until the specified condition is made false., the execution of the block of code happens at least once.

**for:**

var data = "Good"

for(var i=0; i<10 ; i++){

console.log(" i value is " +i);

console.log(data);

}

**Output:**

i value is 0

Good

i value is 1

Good

i value is 2

Good

i value is 3

Good

i value is 4

Good

i value is 5

Good

i value is 6

Good

i value is 7

Good

i value is 8

Good

i value is 9

Good

**While:**

var a = 10

while(a<15){

    a++;

}

console.log("a value is "+a);

**Output: a value is 15**

**Do-while:**

var a = 15

do{

    a++;

}while(a<15);

console.log("a value is "+a);

**Output: a value is 16**

**Interview questions**

1. **Why variables are loosely typed in JS?**

**A:** The variables are loosely typed in JS because the variables are declared without the datatype. The datatype is decided at the time of execution/runtime.

1. **Can we explicitly convert datatypes of variables?**

**A:** Yes

1. **Difference between == and ===**

**==** checks only the value of the two operands whereas **===** checks the value as well as the data type of the two operands or variables

1. **Types of JS Execution**

**A:** There are two types of JS Execution

1. **On-Browser 2. Off-Browser**

In On-browser execution, the web browser is involved whereas in Off-browser, the execution takes place or javascript is executed without using the browser

In **On-browser** there are again 2 types:

1. In-line embedded execution
2. External line embedded execution

* **In-line embedded execution**

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <title>Html Document</title>
5. </head>
6. <body>
7. <script>
8. console.log("Welcome to JavaScript");
9. </script>
10. </body>
11. </html>

* **External Line Embedded**

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <title>Document</title>
5. </head>
6. <body>
7. <script src="./Datatypes.js"></script>
8. <script src="./CheckDataTypeExplicitly"></script>
9. </body>
10. </html>
11. In Off-browser execution the javascript execution takes place normally without using the browser and output is printed in the terminal.

**Javascript to demonstrate datatypes – check the datatype of variables**

var a = 25

var d = 'D'

var e = "Engineer"

var x = 1.5

var bn = true

var b

console.log(typeof(a));

console.log(typeof(d));

console.log(typeof(e));

console.log(typeof(x));

console.log(typeof(bn));

console.log(typeof(b));

var spl = '\*'

console.log(typeof(spl));

var n = null

console.log(typeof(n));

**Output:**

number

string

string

number

boolean

undefined

string

object

**JavaScript to demonstrate change the datatype of variable explicitly**

var a = 10

var b = "Demo"

console.log(typeof(a));

console.log(typeof(b));

console.log(typeof(Number(b)));

**Output:**

number

string

number

**Javascript to demonstrate operators == and ===**

var a =10

var b = "30"

var c = "30"

// if(a=b){

//     console.log("a and b are same");

// }

// else

// {

//     console.log("a and b are not same");

// }

 if(a===b){

     console.log("a and b are equal and are of same data type");

 }

// else

// {

//     console.log("a and b are not of same data types");

// }

// if(b==c){

//     console.log("b and c are same");

// }

// else

// {

//     console.log("b and c are of not same");

// }

if(b===c){

    console.log("b and c are equal and are of same data type");

}

**Output:** b and c are equal and are of same data type

**Declaration of variables:**

Declaration of variables is of 3 types

var, let, const

1. **var** : can be re-declared and re-initialized

var a = 10

var a = 20       // Re-declaration is possible for var

console.log(a); // Re-intialization is possible for var

**output:20**

1. **let** : cannot be re-declared but can be re-initialized
2. let b = 15
3. let b            // SyntaxError: Identifier 'b' has already been declared
4. b = 34
5. console.log(b);

**Output: SyntaxError: Identifier 'b' has already been declared**

let b = 15

//let b            // SyntaxError: Identifier 'b' has already been declared

b = 34

console.log(b);

**Output: 34**

**const** : cannot be re-declared and also cannot be re-initialized

const c = 45

const c         // Re-declaration is not possible

c = 56             // TypeError : Assignment to constant variable

console.log(c);

**Output:** SyntaxError: Identifier 'c' has already been declared

const c = 45

//const c         // Re-declaration is not possible

c = 56             // TypeError : Assignment to constant variable

console.log(c);

**Output:** TypeError: Assignment to constant variable.

Program execution in memory:

When the user starts execute the javascript, the JSEngine receives the script file, it first creates a default Execution context/space known as “Global Execution Context” where all the JS code gets executed. In this GEC, a global object called **“Window”** is created.

A global variable **“this”** is also created along with the global object **window** and the window is strictly equal to “this”.

In this process there are two phases.

1. Creation phase
2. Execution phase

In the creation phase, all the variable and functions defined by the user are allocated memory, where these memories are named by respective variable names or function names.

All the variable are assigned with undefined values by default.

The implementation of the function will be stored inside the memory.

Once after the creation phase, the execution phase starts.

In the execution, the variables are initialized/overridden by the actual values, and if there is a function invoking statement, the respective function gets executed.

Function execution in memory:

The function execution takes place when the function is invoked in the program.

During the function execution in the memory, a function execution context will be created above the global execution context. This function execution context is temporary and lasts only until the completion of function execution. It disappears from the call stack after execution is complete.

**Variable Hoisting:**

This is similar as in program execution process, wherein,

In the creation phase, all the variable defined by the user are allocated separate memory, where these memory are named by respective variable names.

All the variable are assigned with undefined values by default.

During execution phase, the variables are initialized/overridden by the actual values.

**Function Hoisting:**

Similar to the program execution in memory, the functions will be allocated with separate memory and the implementation of the function will be stored inside the memory. Once after the creation phase, the execution phase starts.

In the execution, if there is a function invoking statement, the respective function gets executed.

**Scenarios:**

1. **Block inside a function**
2. **Function inside a block**
3. **(Block inside a function) inside a function**
4. **Block inside a function**
5. var a = 10
6. let b = 15
7. const c = 25
8. function sum(a,b){
9. var x = 11
10. let y = 21
11. const z = 31
12. {
13. var m = 51
14. let n = 61
15. const o = 71
16. }
17. }
18. sum(200,300)
19. {
20. var i = 81
21. let j = 91
22. const k = 100
23. }

**Analysis:**

a--> global

b-->script

c-->script

x-->local

y-->local

z-->local

m-->local

n-->block

o-->block

i-->global

j-->block

k-->block

**2. Function inside a block**

var a = 10

let b = 15

const c = 25

{

    var x = 11

    let y = 21

    const z = 31

    function sum(a,b){

        var m = 51

        let n = 61

        const o = 71

    }

}

sum(200,300)

{

    var i = 81

    let j = 91

    const k = 100

}

**Analysis:**

a-->global

b-->script

c-->script

x-->global

y-->block

z-->block

m-->local

n-->local

o-->local

i-->global

j-->block

k-->block

**3. (Block inside a function) inside a function**

var a = 34;

let b = 33;

const c = 10;

function sum(a,b){

    var x = 12;

    let y = 22;

    const z = 36;

    function sub(x,y){

        var p = 44;

        let q = 55;

        const r = 66;

        {

            var i = 90;

            let j = 100;

            const k = 999;

        }

    }

    sub(20,4)

}

sum (2,3)

**Analysis:**

a-->global

b-->script

c-->script

x-->local

y-->local

z-->local

p-->local

q-->local

r-->local

i-->local

j-->block

k-->block

**Day-36 29/09/2022(Thu) Day-3**

**🡪 Data Driven Testing – Generic methods for fetching data from Property file and Excel File and implementation.**

Properties file: DatadrivenByPropertiesFile

Excel file: DatadrivenByExcelFile

Write in notepad 🡪 Execute in eclipse 🡪 Send the code in whatsapp group

**🡪 Program execution and Function execution in memory presentation**

**🡪 Git: Gitbash Commands**

Create a new git repository for the project (Doordash) and keep the URL ready.

Open js\_sdet40 which is on desktop 🡪 right click 🡪 Git bash here🡪 Git bash window will open.

Enter the git commands:

DASARI@DESKTOP-69P9OLH MINGW64 ~/Desktop/js\_sdet40

$ git config --global user.name DRoopesh

DASARI@DESKTOP-69P9OLH MINGW64 ~/Desktop/js\_sdet40

$ git config --global user.email roopesh469@gmail.com

DASARI@DESKTOP-69P9OLH MINGW64 ~/Desktop/js\_sdet40

$ git init

Initialized empty Git repository in C:/Users/DASARI/Desktop/js\_sdet40/.git/

DASARI@DESKTOP-69P9OLH MINGW64 ~/Desktop/js\_sdet40 (master)

$ git add .

DASARI@DESKTOP-69P9OLH MINGW64 ~/Desktop/js\_sdet40 (master)

$ git status

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: Basic/Demo.js

new file: ConditionalStatements/ConditionalStatements1.js

new file: ConditionalStatements/ConditionalStatements2.js

new file: Html Files/CreateHtml.html

new file: Html Files/Embedded.html

new file: Html Files/Inline.html

new file: LoopingStatements/DoWhileLoop.js

new file: LoopingStatements/Forloop.js

new file: LoopingStatements/WhileLoop.js

new file: ProgramExecution/ProgramExection1.js

new file: ProgramExecution/ProgramExecution2.js

new file: ProgramExecution/ProgramExecution3.js

new file: ProgramExecution/Programhtml1.html

new file: ProgramExecution/Programhtml2.html

new file: ProgramExecution/Programhtml3.html

new file: Variables/Check.js

new file: Variables/CheckDataTypeExplicitly.js

new file: Variables/CheckOperator.js

new file: Variables/Datatypes.js

new file: Variables/VariableDeclarationTypes.js

DASARI@DESKTOP-69P9OLH MINGW64 ~/Desktop/js\_sdet40 (master)

$ git remote add origin https://github.com/DRoopesh/Doordash\_JS.git

DASARI@DESKTOP-69P9OLH MINGW64 ~/Desktop/js\_sdet40 (master)

$ git remote -v

origin https://github.com/DRoopesh/Doordash\_JS.git (fetch)

origin https://github.com/DRoopesh/Doordash\_JS.git (push)

DASARI@DESKTOP-69P9OLH MINGW64 ~/Desktop/js\_sdet40 (master)

$ git commit -m "first commit"

[master (root-commit) 96029d1] first commit

20 files changed, 257 insertions(+)

create mode 100644 Basic/Demo.js

create mode 100644 ConditionalStatements/ConditionalStatements1.js

create mode 100644 ConditionalStatements/ConditionalStatements2.js

create mode 100644 Html Files/CreateHtml.html

create mode 100644 Html Files/Embedded.html

create mode 100644 Html Files/Inline.html

create mode 100644 LoopingStatements/DoWhileLoop.js

create mode 100644 LoopingStatements/Forloop.js

create mode 100644 LoopingStatements/WhileLoop.js

create mode 100644 ProgramExecution/ProgramExection1.js

create mode 100644 ProgramExecution/ProgramExecution2.js

create mode 100644 ProgramExecution/ProgramExecution3.js

create mode 100644 ProgramExecution/Programhtml1.html

create mode 100644 ProgramExecution/Programhtml2.html

create mode 100644 ProgramExecution/Programhtml3.html

create mode 100644 Variables/Check.js

create mode 100644 Variables/CheckDataTypeExplicitly.js

create mode 100644 Variables/CheckOperator.js

create mode 100644 Variables/Datatypes.js

create mode 100644 Variables/VariableDeclarationTypes.js

DASARI@DESKTOP-69P9OLH MINGW64 ~/Desktop/js\_sdet40 (master)

$ git push origin master

Enumerating objects: 28, done.

Counting objects: 100% (28/28), done.

Delta compression using up to 8 threads

Compressing objects: 100% (27/27), done.

Writing objects: 100% (28/28), 3.45 KiB | 353.00 KiB/s, done.

Total 28 (delta 3), reused 0 (delta 0), pack-reused 0

remote: Resolving deltas: 100% (3/3), done.

To https://github.com/DRoopesh/Doordash\_JS.git

\* [new branch] master -> master

Token : ghp\_C29FCCHcJCIbvuVA30tE1QVZyg1VdC0S4Qan

URL : <https://github.com/DRoopesh/Doordash_JS.git>

Username: DRoopesh

Password: rooptej8589