JavaScript Intermediate Level

Created Date : 21/03/2023
Created By : GOVE ACADEMY
Level : Intermediate

Version History

S. No	Revision Date	Version	Modifications	Modified By

Table of Contents

3. Variables & Operators

- 3.1 Keywords
- 3.2 Variable Initialization & Declaration./ Diff
- 3.3 Variable scope with the function
- 3.4 Assignment operator -
- 3.5 Diff b/w == & ===
- 3.6 Ternary operator
- 3.7 Conditional operator
- 3.8 Diff b/w undefined & null
- 3.9 Logical operator.

4. Class

- 4.1 Un exported Class
- 4.2 Exported Class, Named export, Default export.
- 4.3 Object creation
- 4.4 Constructor – with & without parameters
- 4.5 Static Variables & Functions
- 4.6 Global Variables
- 4.7 This Keyword
- 4.8 Accessing function & Variable inside a class

5. Object Manipulation

5.1 Initialize Object (Empty & Value)

5.2 Reading a Value from Object –dot & Square

5.3 Inserting Key-Value pair in Object

5.4 Updating Existing Key Value Pair

5.5. Delete Key Value Pair in Object

5.6 Object Module Function

5.7 Merge Two Objects

5.8 Deep ,Shallow Clone

5.9 Json Stringify

5.10 Json Parse

6. Array Manipulation

6.1 Initialize

6.2 Array of String , Number , Objects and Array

6.3 Reading a Value from Array using index

6.4 Inserting Value in Array

6.5 Updating Existing in Array

6.6 Delete Array

6.7 In-Build Function

6.8 Merging Two Arrays

7. Control Flow

7.1 If, If else , else, nested if

7.2 loop

7.3 Map

7.4 While

7.5 Switch

8. Asynchronous

8.1 Defining an asynchronous function

8.2 Keywords

8.3 Creating async function

8.4 Async function call

8.5 Promise Keyword

9. Exception Handling

9.1 Function with try catch

9.2 Function with try catch and finally

9.3 Asynchronous function with try catch

9.4 Difference b/w Throw & return

9.5 Catch block without & error capturing

9.6 Exception handling with .catch

1.Node Project Creation

- ❑Node.js is the run time environment to run JavaScript.
- ❑JavaScript is used for both backend and frontend.

There are two approach to create a node project:

Approach 1:

Step1: Install a node.js Click Here - Node.js Installation

Step2: Check the version in command prompt

```
C:\Users\GOVE-LAP-12>node -v
v18.12.1
```

Step3: Choose a path in your local desktop and open in command prompt by typing cmd in the folder path.

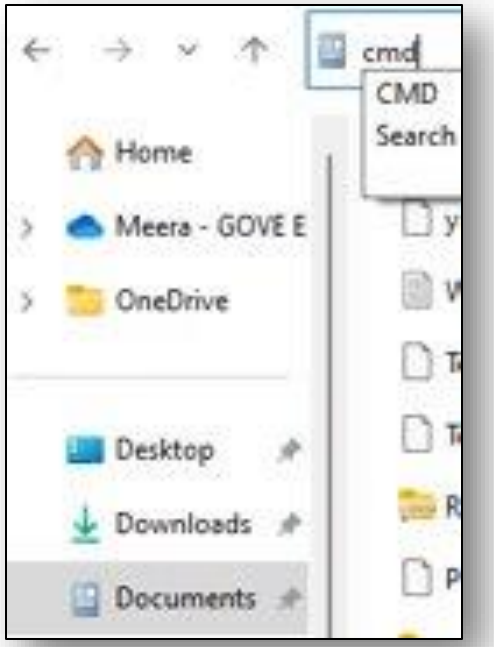
```
C:\Users\GOVE-LAP-12\Documents>mkdir Nodeproject22

C:\Users\GOVE-LAP-12\Documents>cd Nodeproject22

C:\Users\GOVE-LAP-12\Documents\Nodeproject22>
```

Step6: To open in Visual studio code, write as code . In command prompt.

```
C:\Users\GOVE-LAP-12\Documents\Nodeproject22>code .
```

Step7: Write a command ***npm init –y*** to install node project.

```
PS C:\Users\GOVE-LAP-12\Documents\Nodeproject22> npm init -y
```



In the above Screenshot, selected the document folder and entered the cmd in the folder path and press enter to open in Command prompt.

Step4: Write mkdir folder name in command prompt.

Step5: To access the above folder, write cd folder name

```
Wrote to C:\Users\GOVE-LAP-12\Documents\Nodeproject22\package.json:

{
  "name": "nodeproject22",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Once you execute the command , the above set of code will display in the terminal & you will get a package.json file automatically.

What is npm?

npm is a tool that helps to easily install and manage packages of code (libraries, frameworks, and other tools) written in JavaScript.

What is init?

To Initialize a new project

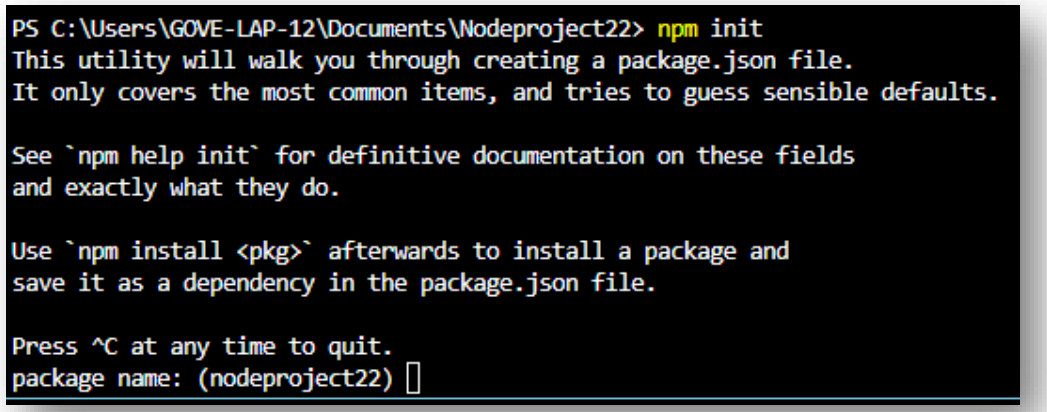
What is –y in npm init –y?

Assign a default values for all the fields in the package.json.

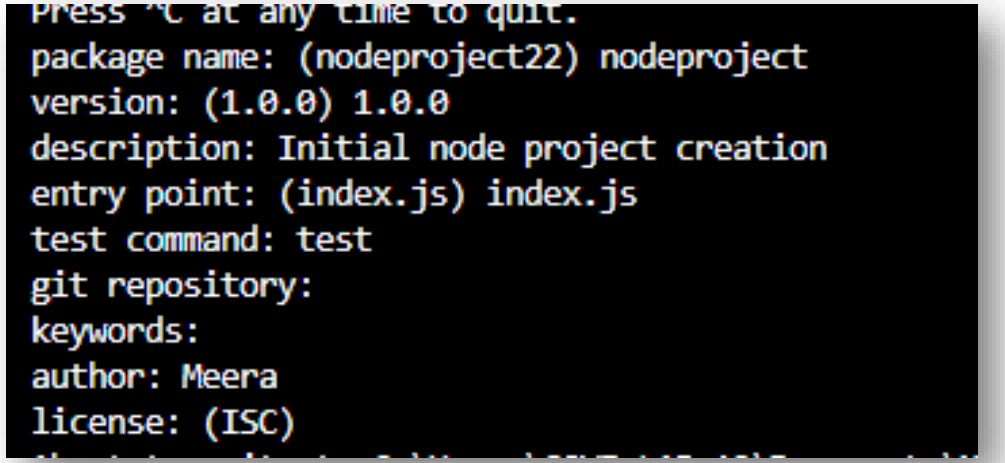
Approach 2:

Follow Step1 to Step 6 as it is like ***Approach 1***

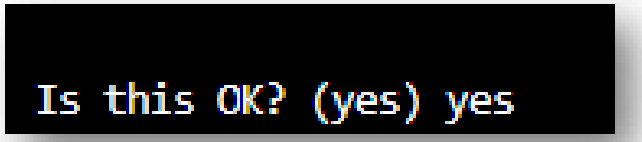
Step7: Write a command ***npm init*** to install node project.



Once you execute the command , the above set of code will display in the terminal.



The above fields are needs to be fill by ourself, Once it is done enter yes in terminal to create a package.json file



1.1 Package.json

Meta Information about the project, dependencies, dev - dependencies, script etc.

- **"name": "nodeproject", -** Name of the Project
- **"version": "1.0.0", -** Version of project release .

1 – Major Release – Drastic change

0 – Minor Release - Feature

0 – Patch Release – Bugs/Defect

- **"description": "", -** Description about the project.
- **"main": "index.js",** - Main file to access the project
- **"scripts": {**

"test": "echo \"Error: no test specified\" && exit 1"

}, - script can be write for run the application , test & build.

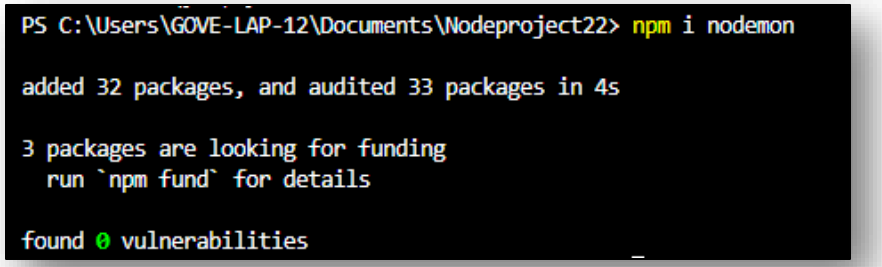
❑**"keywords": [],** - Keyword used in a project.

❑**"author": "",** - One who writes the code in the project.

❑**"license": "ISC"** -like MIT license open-source software can be used to modify and distributing.

1.2 Package lock.json

❑ Write npm I nodemon in terminal for package installing.



Package lock.json file and node modules will be installed in the project.

❑Automatically generating when installing and updating the dependencies.

❑List of all packages will be display.

❑**For Example:** nodemon --> need list of dependencies which is in package lock.json.

1.3 npm i

- npm i or npm install.
- To install the dependencies
- Read the package.json file to install what type of dependency.
- If the package lock.json file is present , it will investigate the file to check whether same version of dependency is used.

1.4 Script

Script can be written for to run the application ,test & build

Build - convert all the code into single plain js file.

For Example: "start": "node index.js"

```
"scripts": {
  "test": "test",
  "start":"node index.js"
```

npm start

```
"scripts": {
  "test": "test",
  "meera":"node index.js"
```

npm run meera

1.5 Dependencies

- ❑ List out the dependency that used in project. [Eg: nodemon/express]
- ❑ Nodemon is a package that can be installed as a dependency in a Node.js project using npm

1.6 Dev Dependencies:

1. Used only for development and Testing not for production use.
2. It can be added directly dev dependencies in package.json or with a command as ;

   npm install --save-dev // jest // [For example: jest, eslint, precommit]

```
PS C:\Users\GOVE-LAP-12\Documents\Nodeproject22> npm install --save-dev jest

added 275 packages, and audited 308 packages in 1m

33 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

```
"devDependencies": {
  "jest": "^29.5.0"
}
```

1.7 Peer Dependencies

- It is a conflict of two dependencies.
- **For example:** If you are using nodemon and jest dependency, both dependency used a common dependency like ***picomatch*** **,** then there is conflict with respect to version.
- Need to add peer dependency in the package.json file by exploring the common version for these two dependency.

```
"peerDependencies": {
  "picomatch": "^3.0.4"
}
```

1.8 Node Modules

1. Node modules are installed and stored.
2. Node modules are essential to run the project.

Functions

2. Functions :

- A JavaScript function is a block of code designed to perform a particular task.
- Functions are a fundamental concept in programming and are used to organize code, make it reusable, and reduce duplication.
- A function gets input parameters, performs some operations on those inputs, and then returns a result.
- The result can be a value, an object, or even another function
- Functions are useful for breaking down complex tasks into smaller, more manageable pieces of code
- Functions can also be reused across different parts of a program, saving time and effort in the development

2.1.Functions without class – Unexport

- An un exported function is a function that is not exposed to the outside world and is only accessible within the module in which it is defined.
- This can be particularly useful for functions that are only used internally within a module and should not be accessed by other parts of the program.

```
//function with multiple parameter
function getStudentDetails(name, no) {
    let RollID = name + '-' + no
    return RollID

}
let RollID = getStudentDetails("Devi", 11)
console.log("Output-3: ", RollID)
```

Output

```
PS C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code> node src/functions/unexported.function.js
Output-3:  Devi-11
PS C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code>
```

2.2 Functions without class - Export.

- functions can be defined without classes and can also be exported from a module.
- Exporting a function makes it available to other parts of the program that require it.
- Exporting functions can be useful for creating reusable and modular code.
- It allows you to define functions in one module and then use them in other modules without having to rewrite the code.

```
module.exports = function studentDetail(name, college) {
    return name + '-' + college
}
```

```
const studentDetail = require('./src/functions/exported-1.function')
const detail = studentDetail("Devi", "FX")
console.log("Detail", detail)
```

Output

```
PS C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code> node main.js
Detail Devi-FX
PS C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code>
```

2.3 Function with & without return statement

- Function with Return Statement:
  - Functions with a return statement are used to return a value back to the caller.
  - The return statement is followed by the value that the function should return

```
function getTodayDate() {
    let currentDate = new Date()
    let message = `Hai All , Today Date is ${currentDate}`
    return message
}
getTodayDate()
let date = getTodayDate()
console.log("Date :", getTodayDate())
```

Output

```
PS C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code> node src/functions/withandWithoutreturn.function.
js
Date : Hai All , Today Date is Wed Mar 22 2023 06:22:54 GMT-0500 (Central Daylight Saving Time)
```

Without Return Statement:

A function without a return statement is also called a "void function".

It is defined using the "function" keyword followed by the function name, parentheses, and a set of curly braces containing the code to be executed.

Return Statement :

```
function getLogDetails() {
    let logDetail = {
        name: {
            user1: "meera",
            user2: "nishanth",
            user3: "kousalya"
        },
        year: {
            meera: 2001,
            nishanth: 2000,
            kousalya: 1999
        },
        college: {
            "kousalya-1999": "FXEC",
            "nishanth-2000": "NEC",
            "meera-2001": "HOLY CROSS"
        }
    }
    let log =logDetail["year"]["meera"]
    return log
}
console.log("Log Details", getLogDetails() )

const getYear=(getYear)=>{
    const year = getYear + 1
    console.log("Year-",year)
}

console.log("Year : ", getYear(getLogDetails()))
```

2.4 Function with & without parameters

Functions with Parameter

- Functions are reusable blocks of code that perform a specific task.
- They can take inputs, called parameters or arguments

```
//function with multiple parameter
function getStudentDetails(name, no) {
    let RollID = name + '-' + no
    return RollID

}
let RollID = getStudentDetails("Devi", 11)
console.log("Output-3: ", RollID)
```

Functions with Parameter-Name & No

Output

```
PS C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code> node src/functions/unexported.function.js
Output-3:  Devi-11
PS C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code>
```

Functions without Parameter

- This function is named displayName ( ) and does not take any parameters.

When called, it will print the string “Welcome to Gove Enterprise"

```
//Function Without Parameter
function displayName() {
    console.log("Output-1 : Welcome to Gove Enterprise");
}
displayName()
```

Output

```
PS C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code> node src/functions/unexported.function.js
Output-1 : Welcome to Gove Enterprise
```

2.5 Function with default Parameter

- A default parameter is a feature in programming that allows you to set a default value for a function parameter

src > functions > JS defaultparameter.function.js > ...

```
1   function getDefaultParameter(name, place = "Paris",) {
2       const team = (` ${name} likes ${place}`)
3       return team
4   }
5   let location = getDefaultParameter("Devi")
6   console.log("Location :", location)
```

Output

```
PS C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code> node src/functions/defaultparameter.function.js
Position :  Devi like Paris
PS C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code>
```

Function with Exact Parameter

- Exact Parameter

Exact Parameter takes the input from the function call skips the default parameter

src > functions > JS defaultparameter.function.js > ...

```
1   function getDefaultParameter(name, place = "Paris",) {
2       const team = (` ${name} like ${place}`)
3       return team
4   }
5   let location = getDefaultParameter("Devi", "America")
6   console.log("Position :", location)
```

Output

```
PS C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code> node src/functions/defaultparameter.function.js
Location :  Devi likes America
PS C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code>
```

2.6 Function with function as parameter.

- In JavaScript, you can pass a function as a parameter to another function. This is known as a "higher-order function" or "callback function".

```
function getDetailList(){
    let userDetail = {
        userName  : "Devi",
        password : 123
    }
    return userDetail
}
getDetailList()

function getFunctionAsInput(getCount){
    console.log("Callback : ", getCount())
}
getFunctionAsInput(getDetailList)
```

Output

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Callback :  { userName: 'Devi', password: 123 }
PS C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code> ^C
```

2.7 Arrow Function

An array function expression is a compact alternative to a traditional function expression

Arrow function have shorter syntax than regular function syntax

```
const getLogDetails = () => {
    let logDetail = {
        name: {
            user1: "meera",
            user2: "nishanth",
            user3: "kousalya"
        },
        year: {
            meera: 2001,
            nishanth: 2000,
            kousalya: 1999
        },
        college: {
            "kousalya-1999": "FXEC",
            "nishanth-2000": "NEC",
            "meera-2001": "HOLY CROSS"
        }
    }
    let log = logDetail["year"]["meera"]
    return log
}
console.log("Log Details", getLogDetails())

const getYear = (getYear) => {
    const year = getYear + 1
    console.log("Year-", year)
}

getYear(getLogDetails())
```

Output

```
Log Details 2001
Year- 2002
>> [ 'Devi' ]
PS C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code>
```

Difference Between Call Back and Closure

Callback :

A Callback is executable Code that is passed as an argument to other Code

Closure :

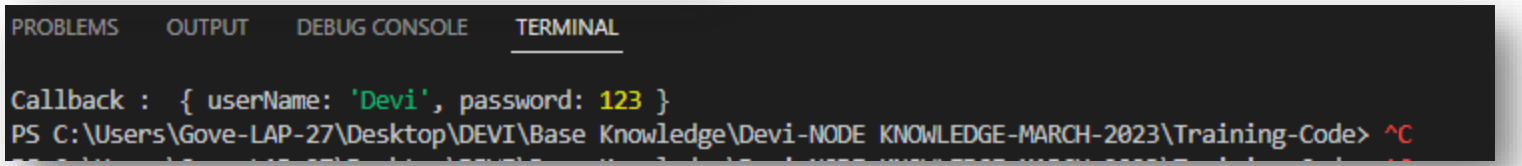
A Closure is a function that is evaluated in an environment containing one or more bound variables.

Callback ( )

- It is passed as an argument to another function
- Once its parent function completed, the function passed as an argument

```
function getDetailList(){
    let userDetail = {
        userName  : "Devi",
        password : 123
    }
    return userDetail
}
getDetailList()

function getFunctionAsInput(getCount){
    console.log("Callback : ", getCount())
}
getFunctionAsInput(getDetailList)
```

Output



2.8 Closure

- Closure is a function that has access to its own private variables, as well as to the variables of its outer functions,
- Closures are used in many ways in JavaScript, such as creating private variables and creating callbacks

Output

```
//Parent to Child Concept. Child Component have access to parent Component
function outerFunction(name, role="Engineer") {
    let outerVariable = "Devi"
    const message = `${name} is currentlyworking in  ${role} `
    innerFunction()
    function innerFunction() {
        let innerVariable = "Kousalya"
        console.log("Inner Value", innerVariable)
        console.log("Outer Variable", outerVariable)
    }
    return message

}
let result = outerFunction("BHUVIKA", "DESIGNER")
console.log("Closure Output : ", result)
```

Closure

- A function keyword inside another function, you are creating a closure
- A function return to another function we can say the closure

functions > JS closure.function.js > ...

```
 */
//Parent to Child Concept. Child Component have access to parent Component
function outerFunction(name, role="Engineer") {
    let outerVariable = "Devi"
    const message = `${name} is currentlyworking in  ${role} `
    innerFunction()
    function innerFunction() {
        let innerVariable = "Kousalya"
        console.log("Inner Value", innerVariable)
        console.log("Outer Variable", outerVariable)
    }
    return message

}
let result = outerFunction("BHUVIKA", "DESIGNER")
console.log("Closure Output : ", result)
```

Output

```
PS C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code> node src/functions/closure.function.js
Inner Value Kousalya
Outer Variable Devi
Closure Output :  BHUVIKA is currentlyworking in  DESIGNER
PS C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code>
```

Variables & Operators

3.1 Keywords

- A keyword is a reserved word that has a specific meaning and purpose in the language.
- Keywords cannot be used as variable names or function names because they are already predefined and reserved for specific tasks.
- For example, the **`'if'`** keyword is used for conditional statements, **`'function'`** is used to declare a function, and **`'var'`**, **`'let'`**, and **`'const'`** are used to declare variables. If you try to use a keyword as a variable name, you will get a syntax error.

3.2 Variable Initialization & Declaration

- What is a Variable Initialization ?

  Variable initialization refers to the process of assigning an initial value to a variable when it is declared.

```
//initialize mark with a value of 5
mark=5;
```

- What is Variable Declaration ?

  Variable declaration is the process of creating a variable and assigning it a value.

```
//Declare a variable named 'mark'
let mark;
```

3.4 Assignment operator.

The assignment operator is used to assign a value to a variable. In JavaScript, the assignment operator is the equal sign ' ='.

```
// Assign the value 5 to x
let varun = 5;
// Assign the value 2 to y
let rahul = 2;
// Assign the value x + y to z:
let totalage = varun + rahul;

console.log("Total age is",totalage)
```

```
PS D:\Projects\Variables-operators\variables\operators> node assignmentoperator.js
Total age is 7
```

3.3 Scope of a variable inside and outside a function JavaScript

- The scope of a variable declared inside a function is limited to that function. This means that the variable cannot be accessed outside the function.
- If the variable is called outside the function scope the variable cannot be accessed.

```
function person() {
    var age = 10;
    console.log(age); // outputs 10
  }

  person(); // calling the function
  console.log(age); // ReferenceError: age is not defined
```

```
10
D:\Projects\Variables-operators\variables\scopeofvariableinafunction\scopeinfunction.js:7
  console.log(age);
              ^

ReferenceError: age is not defined
    at Object.<anonymous> (D:\Projects\Variables-operators\variables\scopeofvariableinafunction\scopeinfunction.js:7:15)
    at Module._compile (internal/modules/cjs/loader.js:1085:14)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1114:10)
    at Module.load (internal/modules/cjs/loader.js:950:32)
    at Function.Module._load (internal/modules/cjs/loader.js:790:14)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:76:12)
    at internal/main/run_main_module.js:17:47
```

3.5 Difference between '==' & '==='

The main difference is '==' compares only the value, while '===' compares both values and the data-type of the variable.

```
let age = 5;
let age2 = "5";

console.log(age == age2); // does not check the data type
console.log(age === age2); // checks the data type
```

```
PS D:\Projects\Variables-operators\variables\operators> node differenceassignmentoperator.js
true
false
```

3.7 Conditional Operator

- The conditional operator, also known as the ternary operator, is a shorthand way of writing an if-else statement in some programming languages. It takes three operands, hence the name "ternary".

```
let num = 5;
let result = (num > 10) ? "Greater than 10" : "Less than or equal to 10";
console.log(result);  // Output: "Less than or equal to 10"
```

```
The output is : Number is less than or equal to 10
```

3.6 What is ternary Operator?

It allows you to evaluate a condition and return one of two values depending on whether the condition is true or false.

For Example:

```
let age = 7
var isEligible = (age >= 18) ? "Eligible" : "Not Eligible";
console.log("The candidate is ",isEligible)
```

```
PS D:\Projects\Variables-operators\variables\operators> node ternaryoperator.js
The candidate is  Not Eligible
```

3.9 What is logical Operators?

- Logical operators are used to perform Boolean logic on one or more values and return a Boolean value as the result. JavaScript supports three logical operators:' **&&** ' (AND), '**||**' (OR), and '**!**' (NOT).

1. && - This operator returns **`true`** if both operands are **`true`**, else **`false`**.
2. **||**-This is used to evaluate expressions that return a Boolean value and returns a Boolean value. The logical OR operator returns **`true`** if at least one of the operands is **`true`**. If both operands are **`false`**, it returns **`false`**.
3. **!** -The logical NOT operator returns **`true`** if the operand is **`false`**, and **`false`** if the operand is **`true`**.

3.8 What is the difference between null and undefined

- **'undefined'** is a value that is automatically assigned to a variable that has been declared but not initialized with a value, or to a function parameter that has not been passed an argument.

```
var car; // car is undefined
function transport(bus) {
  console.log(bus); // bus is undefined
}
transport();
```

- **'Null'**, is an object value that represents the absence of any object value.

```
var myObj = {
    name: "Nishanth",
    age: null,
 };

 console.log(myObj.name); // Output: "Nishnath"
 console.log(myObj.age); // Output: null
```

Class

4.Class

- A class is a blueprint or a template for creating objects that share common properties and behaviors. It is a fundamental concept of object-oriented programming (OOP)
- Un exported Class
- Exported Class

4.1 Un exported Class :

- Un exported classes are classes that are not used outside of the module
- To define un Exported class in JavaScript , ignore the export keyword before the class keyword

Example:

```
class unexportedClass {
    getStudentDetails(time) {
        if (time <=12){
            return "Good Morning"
        }
        else if(time >12 && time <20){
            return "Good Evening"
        }
        else {
            return "Good Night"
        }
    }
}

const UnexportedClass = new unexportedClass()
console.log("Output-1 :", UnexportedClass.getStudentDetails(11))
```

Output:

```
C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code>node src/classes/unexporte
d.class.js
Output-1 : Good Morning

C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code>
```

4.2 Exported Class

- Exported classes can be accessed by other module by importing the module that contains the class.
- **Example** :

```
module.exports= class exportedClass{

    getNameUsingID (){
        return "Kousalya"
    }
}
```

```
const exportedClass = require('./src/classes/exported.class')

const ExportedClass = new exportedClass()
console.log("Exported Class :", ExportedClass.getNameUsingID())
```

Output:

```
C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code>node main.js
Exported Class : Kousalya

C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code>
```

4.3 Object Creation

- We all are using object creation
    - Using new Keyword

```
//Class with Constructor -parameter
class unexportedClassWithConstructor{
    constructor(roll){
        this.college = "FXEC"
        this.student = "Devi"
        this.rollno =  roll
    }

    getCollegeNameWithParameter (input){
        let collegeName = this.college + '-' + this.student + '-' + input
        return collegeName
    }
}

const UnexportedClassWithConstructor = new unexportedClassWithConstructor()
console.log("Output-2 :",UnexportedClassWithConstructor.getCollegeNameWithParameter(2))
```

4.4 Constructor With Parameter

- The Constructor method is a special method for creating and initializing an object created with a class.

Example :

```
//Class with Constructor -parameter
class unexportedClassWithConstructor{
    constructor(roll){
        this.college = "FXEC"
        this.student = "Devi"
        this.rollno =  roll
    }

    getCollegeNameWithParameter (input){
        let collegeName = this.college + '-' + this.student + '-' + input
        return collegeName
    }
}

const UnexportedClassWithConstructor = new unexportedClassWithConstructor()
console.log("Output-2 :",UnexportedClassWithConstructor.getCollegeNameWithParameter(2))
```

Output:

```
C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code>node src/classes/unexporte
d.class.js
Output-2 : FXEC-Devi-2

C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code>
```

Constructor –without Parameter

Constructor can be created without parameter by creating and initializing an object created with a class

```
//Function with Constructor - without parameter
class unexportWithoutParameter{
    constructor(){
        this.rollID = 1
        this.degree = "Engineering"
        this.color= "black"
    }

    getDetails(){
        return this.degree
    }
}

const UnexportWithoutParameter = new unexportWithoutParameter()
console.log("Output-3", UnexportWithoutParameter.getDetails())
```

Output:

```
C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code>node src/classes/unexporte
d.class.js
Output-2 : FXEC-Devi-2
Output-3 Engineering
```

4.5 Static variables & Function

- A static variable is a class property that is used in a class and not on the instance of the class.
- Static Variable

```
module.exports= class exportedClass{
    static college = "NEC"
    getNameUsingID (){
        return "Kousalya"
    }
}
```

```
const exportedClass = require('./src/classes/exported.class')
const ExportedClass = new exportedClass()
console.log("Exported Class :", ExportedClass.getNameUsingID())
console.log("Static Variale :", exportedClass.college)
```

Output:

```
C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code>node main.js
Exported Class : Kousalya
Static Variale : NEC
```

Static Function

1.Static methods are generally used to create utility functions.

2.To declare a static method, we can simply use static keyword with the method .

3.The static method are not called on the instance of class they are made to call directly on the class.

```
module.exports= class exportedClass{
    static college = "NEC"
    getNameUsingID (){
        return "Kousalya"
    }

    static getApplicationName(){
        let applications =[ "Insights", "Platform", "CNN"]
        return applications
    }
```

```
const exportedClass = require('./src/classes/exported.class')
console.log("Static Functions :",exportedClass.getApplicationName())
```

Output:

```
C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi-NODE KNOWLEDGE-MARCH-2023\Training-Code>node main.js
Exported Class : Kousalya
Static Variale : NEC
Static Functions : [ 'Insights', 'Platform', 'CNN' ]
```

4.8 Accessing function & variable inside a class

Access functions and variables inside a class using this keyword. This keyword refers to the current instance of the class.

Accessing function :

```
class dynamicShape{
    constructor(color){
        this.shapeName = "Square"
        this.shapeColor = color
    }

    getDynamicShapeName(){
        return this.shapeName
    }

    static getStaticShapeName(){
        return this.shapeName
    }

    getDynamicColor(){
        return this.shapeColor
    }
}

const DynamicShape = new dynamicShape("red")
console.log("Output-2", DynamicShape.getDynamicShapeName())
console.log("Output-3", dynamicShape.getStaticShapeName())
console.log("Output-4", DynamicShape.getDynamicColor())
```

```
class unexportedClass {


    getNameToString() {
        return this.getNameDisplay
    }

    getNameDisplay() {
        return this.name + '-' + this.age
    }

    getDataFromArray() {
        const Colors = ['green', 'blue', 'red']
        console.log("Colors", Colors)
    }
}

const result = new unexportedClass("Devi", 20)

console.log("Result-1", result)
```

4.6 Global Variables

- Global variable in JavaScript is a variable declared in the global scope. It is available to use everywhere inside the program.

4.7 This. Keyword

- The "this" keyword in JavaScript classes refers to the current instance of the class. When a class is instantiated using the "new" keyword, "this" refers to the specific instance that was created.

```
constructor(color){
    this.shapeName = "Square"
    this.shapeColor = color
}

getDynamicShapeName(){
    return this.shapeName
}

static getStaticShapeName(){
    return this.shapeName
}

getDynamicColor(){
    return this.shapeColor
}
```

5.Object Manipulation

5.1. Initialize Object:

Initialize the object with the properties of key and values.

Example :

Initialize Object (Empty),

```
// Initialize Object as Empty{}
let data = {}
console.log("Empty object : ", data);
```

Initialize object with key and value,

```
// Object with structure values and key
const person1 = {
    name: "Arun",
    age: 22,
    place: "KTC Nagar",
    city: "Tirunelveli"
}
```

5.2. Reading a Value from Object :

5.2.1. Dot Notation :

Access the object properties using the dot notation ‘ . ‘ .

Example :

```
const person1 = {
    name: "Arun",
    age: 22,
    place: "KTC Nagar",
    city: "Tirunelveli"
}

const person2 = {
    name: "Aswin",
    age: 22,
    department: "Mech",
    address: {
        place: "Tuticorin",
        State: "TamilNadu"
    }
}

// Reading a Value from Object- dot notation

let session = person1?.name
console.log("Person Name :", session);
```

Output :

```
PS E:\FunctionManipulation> node dotNotation.js
Person Name : Arun
```

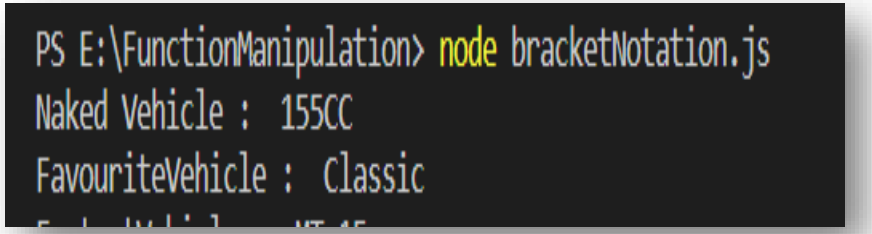
5.2.2. Bracket Notation :

- Bracket notation is a way to access or manipulate the properties of an object in JavaScript.
- It uses square brackets ‘[]‘ to specify the property name as a string or an expression that evaluates to a string.

Example :

```
// Bracket Notation example

let favouriteVehicle = vehicles["RoyalEnfield"];
console.log("FavouriteVehicle : ", favouriteVehicle);

let fastVehicle = vehicles['Yamaha']['NakeBikes'][0]['Name'];
console.log('FastestVehicle : ', fastVehicle);
```

Output:



5.4. Updating Existing Key Value pair :

Update an existing key-value pair in an object by assigning a new value to the corresponding key.

Example :

Dot Notation Updating key & value,

```
// Update the object values using dot notation
persondata.Age = 25
```

Bracket Notation Updating key & value,

```
// Update the object values using bracket notation
persondata["Address"]["Place"] = "KTC Nagar"
```

Using Object.assign() method,

```
// Object.assign() method,
Object.assign(persondata, { Gender: "Female" });
console.log("Assign : ",persondata);
```

Output :

```
PS E:\FunctionManipulation> node updateObject.js
BeforeUpdate :  23
BeforeUpdate :  Tirunelveli
AfterUpdate :  25
AfterUpdate :  KTC Nagar
Assign After Update :  {
  Name: 'Devi',
  Age: 25,
  Gender: 'Female',
  Address: { Place: 'KTC Nagar', State: 'TamilNadu' },
  Nationality: 'Indian',
  Languages: 'Tamil,English'
}
$$$$$$$$$$$$ UK
```

5.3. Inserting Key Value pair in Object :

Inserting add a new key-value pair to an object in several ways:

Example :

Dot Notation ,

```
students.name = "Aswin";

console.log("After Inserting : ", students);
```

Bracket Notation example,

```
students['staffs'] = 30;

console.log("After Inserting : ", students);
```

Example : Object.assign() method ,SpreadOperator

Output :

```
After Inserting :  {
  Department: { Mechanical: 50, Civil: 30, EEE: 30, CSC: 40 },
  Boys: 75,
  Girls: 75,
  name: 'Aswin',
  staffs: 30
}
Insert ARRAY :  [
  { Name: 'MT-15', EngineCapacity: '150CC' },
  { Name: 'R15', EngineCapacity: '155CC' },
  [ { Name: 'FZ-25', EngineCapacity: '249CC' } ]
]
Assign :  {
  RoyalEnfield: 'Classic',
  Apache: 'RTR 350',
  Yamaha: { NakeBikes: [ [Object], [Object], [Array] ] },
  Bajaj: 'Pulser NS200',
  XL: 'Heavy Duty'
}
```

5.5. Delete Key Value Pair in object :

Delete a key-value pair from an object using the delete keyword or the Object method.

Example :

Dot Notation for Delete key value ,

```
// Delete the key value - dot notation
delete newPerson.Nationality
```

Bracket Notation for Delete key value ,

```
// Delete the key value - bracket notation
delete newPerson['Address']["State"]

console.log("AfterDelete : ", newPerson)
```

Output :

```
PS E:\FunctionManipulation> node deleteKeyValue.js
BeforeDelete :  {
  Name: 'Nisanth',
  Age: 22,
  Address: { Place: 'Tirunelveli', State: 'TamilNadu' },
  Nationality: 'Indian',
  Languages: 'Tamil,English'
}
AfterDelete :  {
  Name: 'Nisanth',
  Age: 22,
  Address: { Place: 'Tirunelveli' },
  Languages: 'Tamil,English'
}
```

5.6. Object Module Function :

5.6.1. Mapping :

- The map method is used to transform elements in an array.
- It takes a callback function as an argument, which is applied to each element in the array.
- The map() method then returns a new array with the transformed elements.

Example:

Mapping the object variable,

```
// Mapping Example

const data1 = Person.map(Person => Person.Address)
console.log("Mapping :", data1);
```

Output:

```
PS E:\FunctionManipulation> node mapObject.j
Mapping : [ 'Tirunelveli', 'Madurai' ]
MapObject :  [
  { fullname: 'Nisanth B.E', Age: 38 },
  { fullname: 'Petchi B.E', Age: 38 }
]
```

5.6.2. Entries :

- The object Entries method is used to return an array of an object's key-value pairs.
- Where each key-value pair is represented as an array with two elements are key and the value.

Example :

Object entries example,

```
const collegeDetail = Object.entries(collegeNames)
console.log("Entries : ", collegeDetail)
```

Output:

```
PS E:\FunctionManipulation> node entriesObject.js
LoopingEntries :  name NEC
LoopingEntries :  place Tiruneleveli
LoopingEntries :  phno 123445
Entries :  [ [ 'name', 'NEC' ], [ 'place', 'Tiruneleveli' ], [ 'phno', 123445 ] ]
ArrayObject :  Name Petchi
```

Looping the Object entries example,

```
for (const [key, value] of Object.entries(person1)) {
    console.log("LoopingEntries : ", `${key} : ${value}`);
}
```

5.6.3. Object Keys :

- The Object keys() method is used to return an array of an object's keys.

- Each key in the object is represented as a string in the resulting array.

Example :

The Object.keys() method on an object,

```
const collegeKey = Object.keys(collegeNames)
console.log("CollegeKey : ", collegeKey)
```

Output :

```
PS E:\FunctionManipulation> node keyObject.js
CollegeKey :  [ 'name', 'place', 'phno' ]
PS E:\FunctionManipulation>
```

5.6.5. Object Freeze :

This object freeze a method using the Object. Freeze() to freeze the whole object.

Example :

Freeze the object example ,

```
createobject.college = "SRM"
console.log("FREEZE : ", createobject.college);
```

Output :

```
PS E:\FunctionManipulation> node freezeObject.js
FREEZE :  HEC
PS E:\FunctionManipulation>
```

5.6.4. Object Values :

- The Object values() method is used to return an array of an object's values.
- Each value in the object is represented as an element in the resulting array.

Example :

The Object.values() method on an object ,

```
const collegevalue = Object.values(collegeNames)
console.log("CollegeValue : ", collegevalue)
```

Output :

```
PS E:\FunctionManipulation> node valueObject.js
CollegeValue :  [ 'NEC', 'Tiruneleveli', 123445 ]
PS E:\FunctionManipulation>
```

5.6.6.Object Assign :

- The Object Assign() method is used to copy the values of all properties from one or more source objects to a original object.

- The method returns the target object with the new properties added.

Example :

Assign Object to another variable,

```
// Assign Examples
const assignobject = Object.assign(person1);
console.log("ASSIGN : ", assignobject);
```

Assign Object merge to Objects,

```
// Assign Object - Object
const object1 = Object.assign(person1, person2);
console.log("assignobject : ", object1);
```

Output :

```
PS E:\FunctionManipulation> node assignObject.js
ASSIGN :  { name: 'Arun', age: 22, place: 'KTC Nagar', city: 'Tirunelveli' }
assignobject :  {
  person1: { name: 'Arun', age: 22, place: 'KTC Nagar', city: 'Tirunelveli' },
  person2: {
    name: 'Aswin',
    age: 22,
    department: 'Mech',
    address: { place: 'Tuticorin', State: 'TamilNadu' }
  }
}
PS E:\FunctionManipulation>
```

5.6.7. Spread & Rest Operator :

- The spread & rest operator is used to expand an array or object into individual elements.

- It can be used to copy an array or object or to combine multiple arrays or objects into a single array or object.

Example :

Spread Operator Example ,

```
// Spread & Rest Operator Examples (append the value)
const input1 = [1, 2, 3]
const input2 = [...input1, 4, 5]

console.log("Spread Operator :", input2)
```

Rest Operator example ,

```
// Rest Operator - used to collect the multiple elements in array

const [first, ...rest] = [1, 2, 3, 4, 5];

console.log("Rest Operator : ",first);
console.log("Rest oper 2 : " , rest);
```

Output :

```
PS E:\FunctionManipulation> node spreadrestoperator.js
%%%% [ 1, 2, 3 ]
Spread Operator : [ 1, 2, 3, 4, 5 ]
Rest Operator :  1
Rest oper 2 :  [ 2, 3, 4, 5 ]
Petchi
[ 'Nisanth', 'Awin', 'Arun' ]
PS E:\FunctionManipulation>
```

5.7. Merge Two Objects :

Merge two objects in JavaScript using the spread (...) operator.

Example :

example using the spread operator,

```
let mergeObject = { ...newPerson, ...createobject }
console.log('MergeObject : ', mergeObject);
```

Also , use the object assign method to merge the object.

Output :

```
PS E:\FunctionManipulation> node mergeObject.js
MergeObject :  {
  Name: 'Aswin',
  Age: 22,
  Address: { Place: 'Tirunelveli', State: 'TamilNadu' },
  Nationality: 'Indian',
  Languages: 'Tamil,English',
  college: 'HEC',
  department: 'Mechanical'
}
```

5.8. Deep & Shallow clone :

- A shallow copy only copies the top-level properties of an object, and
- Deep copy creates a new object with all nested properties

Example :

Shallow Clone example, -- nested data can modified in original and clone copy

```
// Modify the cloned object
data2.name = 'Arun'
data1.address.state = 'PA'

console.log("OriginalFinalObject : ", data1);
console.log("CloneFinalObject", data2);
```

Output :

```
OriginalFinalObject :  PA
CloneFinalObject :  PA
OriginalFinalObject :  Vasanth
CloneFinalObject :  Arun
```

Deep Clone example , -- Did not affect the nested object

```
// Modify the original object
originalObj.name = 'Arun';
originalObj.address.city = 'Tirunelveli Town';

// Verify that the clone is independent of the original object
console.log("OriginalObject : ",originalObj.name);
console.log("CloneObject : ",cloneObj.name);

console.log("OriginalCityName : ", originalObj.address.city);
console.log("CloneCityName : ", cloneObj.address.city);
```

```
}
OriginalObject_____ :  Arun
CloneObject____ :  Vasanth
OriginalCityName_____ :  Anytown
CloneCityName_______ :  Tuticorin
PS E:\FunctionManipulation>
```

5.9. JSON Stringify :

- It converts a JavaScript object or value to a JSON string.
- This function takes an object as a parameter and returns a string that represents the object in JSON format.

Example :

```
// Json stringify
let string = JSON.stringify(createobject)
console.log('Json Stringify : ', string);
```

Output :

```
PS E:\FunctionManipulation> node jsonStringify.js
Json Stringify :  {"college":"HEC","name":"Aswin","department":"Mechanical"}
PS E:\FunctionManipulation>
```

5.10. JSON Parse :

- It converts a JSON string to a JavaScript object.
- This function takes a JSON string as a parameter and returns a JavaScript object that represents the data in the string.

Example:

```
let orgobject = JSON.parse(string)
// Json Parse
console.log('Json Parse : ',orgobject);
```

```
PS E:\FunctionManipulation> node jsonParse.js
Json Stringify :  {"college":"HEC","name":"Aswin","department":"Mechanical"}
Json Parse :  { college: 'HEC', name: 'Aswin', department: 'Mechanical' }
PS E:\FunctionManipulation>
```

6. Array Manipulation

6.1 Array Initialize :

1) Initialize an empty array
2) Initialize an array with values
3) Initialize an array with a specific length and default value
4) Initialize an array with different data types
5) Initialize a two-dimensional array

```
1   // 1) Initialize an empty array:
2
3   let myArray1 = [];
4
5   console.info("myArray1", myArray1);
```

```
// 2) Initialize an array with values:

let myArray2 = [1, 2, 3, 4, 5];

console.info("myArray2", myArray2);
```

```
// 4) Initialize an array with different data types:

let myArray4 = ['hello', 42, true, { name: 'John' }];

console.info("myArray4", myArray4);
```

```
// 3) Initialize an array with a specific length and default value:

let myArray3 = new Array(5).fill(0);

console.info("myArray3", myArray3);
```

```
// 5) Initialize a two-dimensional array:

let myArray5 = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];

console.info("myArray5", myArray5[1][1]);
```

6.2 Type of arrays :

1) Array of strings
2) Array of numbers
3) Array of objects
4) Array of array

```
// 1) Array of Strings:

const array1 = ["apple", "banana", "orange", "kiwi"];

console.log("array1",array1);
```

```
// 2) Array of numbers :

const array2 = [1, 2, 3, 4, 5];

console.log("array2",array2);
```

```
// 3) Array of objects :

const array3 = [
    { name: "Alice", age: 30 },
    { name: "Bob", age: 25 },
    { name: "Charlie", age: 35 },
];

console.log("array3",array3);
```

```
// 4) Array of array :

const array4 = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
];

console.log("array4",array4);
```

6.3 Reading a Value from Array using index

We can read each values of an array by their index position the index is starts from 0.

```
const numbers = [1, 2, 3, 4, 5];

// Reading a value from array using index
const secondNumber = numbers[1];

console.log("secondNumber",secondNumber);
```

6.4 Inserting Value in Array

1) Push
2) Unshift
3) Splice

```
const fruits = ["apple", "banana", "orange"];

// Inserting a value at the end of the array
fruits.push("grape");

// Inserting a value at the beginning of the array
fruits.unshift("kiwi");

// Inserting a value at a specific index
fruits.splice(2, 0, "pear");

console.log("fruits",fruits);
```

6.5 Updating Existing in Array

Updating existing array using index position.

```
const fruits = ["apple", "banana", "orange"];

// Updating the value at index 1
fruits[1] = "grape";

console.log("fruits",fruits);
```

6.6 Delete Array

```
let arr2 = [1, 2, 3, 4, 5];

console.log(arr2);

arr2.splice(0); // clears the array using splice method

console.log(arr2); // []

// or

arr2 = []; // clears the array by setting it to an empty array

console.log(arr2);
```

6.7 Inbuild Function - (Pop , Slice , find , filter , map , Spread)

1) Pop - Remove the last element from the array.
2) Slice - Create a new array with a slice of the original array.
3) Find – Returns the first value which satisfy the condition.
4) Filter – Returns the values which are satisfy the condition.
5) Map – Loop each value in the array.
6) Spread – Merging two arrays.

```
//1)  Pop

let fruits = ['apple', 'banana', 'orange'];

// Remove the last element from the array
let poppedFruit = fruits.pop();


console.log(poppedFruit);

console.log(fruits);
```

```
// 2) Slice

let numbers = [1, 2, 3, 4, 5];

// Create a new array with a slice of the original array
let slicedNumbers = numbers.slice(1, 4);

console.log(slicedNumbers);

console.log(numbers);
```

```
// 3) Find

let people = [
    { name: 'Alice', age: 25 },
    { name: 'Bob', age: 30 },
    { name: 'Charlie', age: 35 },
    { name: 'Dave', age: 40 },
];


let person = people.find(function (p) {
    return p.age >= 35;
});

console.log(person);
```

```
// 4) Filter

let numbers1 = [1, 2, 3, 4, 5];

let evenNumbers = numbers1.filter(function (n) {
    return n % 2 === 0;
});

console.log(numbers1);

console.log(evenNumbers);
```

```
// 5) Map

let numbers2 = [1, 2, 3, 4, 5];


let squaredNumbers = numbers2.map(function (n) {
    return n * n;
});


console.log(numbers);

console.log(squaredNumbers);
```

```
// 6) Spread


let arr1 = [1, 2, 3];
let arr2 = [4, 5, 6];


let combinedArray = [...arr1, ...arr2];


console.log(combinedArray);
```

6.8 Merging Two Arrays

1) Concat method
2) Spread

```
let arr1 = [1, 2, 3];
let arr2 = [4, 5, 6];


let mergedArray = arr1.concat(arr2);


console.log(mergedArray);
```

```
// Spread


let arr3 = [1, 2, 3];
let arr4 = [4, 5, 6];


let combinedArray = [...arr3, ...arr4];


console.log(combinedArray);
```

7.Control Flow

7.1 Control Flow Types:

- **If** - is a conditional statement that allows you to execute a block of code if a certain condition is true.

```
let num = 12;

if (num > 10) {
  console.log("The number is greater than 10");
}
```

```
The number is greater than 10
```

If else

If else - is a conditional statement that allows you to execute different blocks of code depending on whether a certain condition is true or false.

```
let response = "Tomorrow meeting"

if (response === "Tomorrow meeting") {
    console.log("tomorrow meeting at 9 AM")
}
else {
    console.log("tomorrow meeting is cancel")
}
```

```
tomorrow meeting at 9 AM
```

Else

- **Else -** It is used to specify a block of code that should be executed if a certain condition in an "if" statement is false

```
let response = "Tomorrow meeting"

if (response === "Tomorrow meeting") {
    console.log("tomorrow meeting at 9 AM")
}
else {
    console.log("tomorrow meeting is cancel")
}
```

```
tomorrow meeting is cancel
```

Nested if

Nested if - a nested if statement is a conditional statement that is used within another conditional statement. It allows you to check for multiple conditions and execute different blocks of code depending on the outcome of those conditions.

```
num = 20
if (num >= 10) {
  console.log("Num is more than 10.")
  if (num > 15) {
    console.log("Num is also more than 15.")
  }
} else {
  console.log("Num is less than 10.")
}
```

```
Num is more than 10.
Num is also more than 15.
```

7.3.1 What is for loop ?

It is a programming construct that allows you to repeat over a block of code a specified number of times or over a collection of elements in an array or object.

```
for (let i = 1; i <= 10; i++) {
    if (i % 2 !== 0) {
        console.log("ODD Number " + i);
    }
}
```

```
ODD Number 1
ODD Number 3
ODD Number 5
ODD Number 7
ODD Number 9
```

7.3.2 What is for each loop ?

It is a type of loop used in programming that iterates over each element in a collection, such as an array or a list.

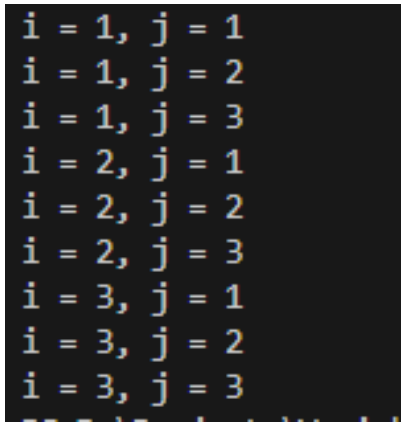
```
const fruits = ['apple', 'banana', 'cherry'];

fruits.forEach(function(listoffruits) {
  console.log(listoffruits);
});
```

```
apple
banana
cherry
```

7.3.3 What is nested for loop?

A nested for loop is simply a loop inside another loop. This can be useful when you need to perform a certain operation for every combination of two or more arrays or when you need to perform a certain operation for every combination of two or more sets of data.

```
for (let i = 1; i <= 3; i++) {
    for (let j = 1; j <= 3; j++) {
      console.log(`i = ${i}, j = ${j}`);
    }
  }
```



7.4 What is .map() ?

Creates a new array from calling a function for every array element.

```
const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
const multliplyNumbers = numbers.map(number => number * 2);
console.log(multliplyNumbers);
```

```
[
   2,  4,  6,  8, 10,
  12, 14, 16, 18, 20
]
```

7.5 What is a While Statement ?

- It is a looping statement that allows you to repeatedly execute a block of code as long as a certain condition remains true.

```
let i = 0;
while (i < 10) {
  console.log(i);
  i++;
}
```

```
0
1
2
3
4
5
6
7
8
9
```

7.6 What is Switch ?

It is a control flow statement that allows you to execute different code blocks depending on the value of a variable or expression.

```
let day = "Monday";
switch (day) {
  case "Monday":
    console.log("Today is Monday.");
    break;
  case "Tuesday":
    console.log("Today is Tuesday.");
    break;
  case "Wednesday":
    console.log("Today is Wednesday.");
    break;
  default:
    console.log("Today is not Monday, Tuesday, or Wednesday.");
    break;
}
```

```
Today is Monday.
```

8.Asynchronous

8.1 Defining an asynchronous function:

An asynchronous function is a type of function in JavaScript that allows you to perform operations that might take some time to complete, without blocking the execution of other code. You define an asynchronous function using the async keyword in front of the function declaration.

Example:

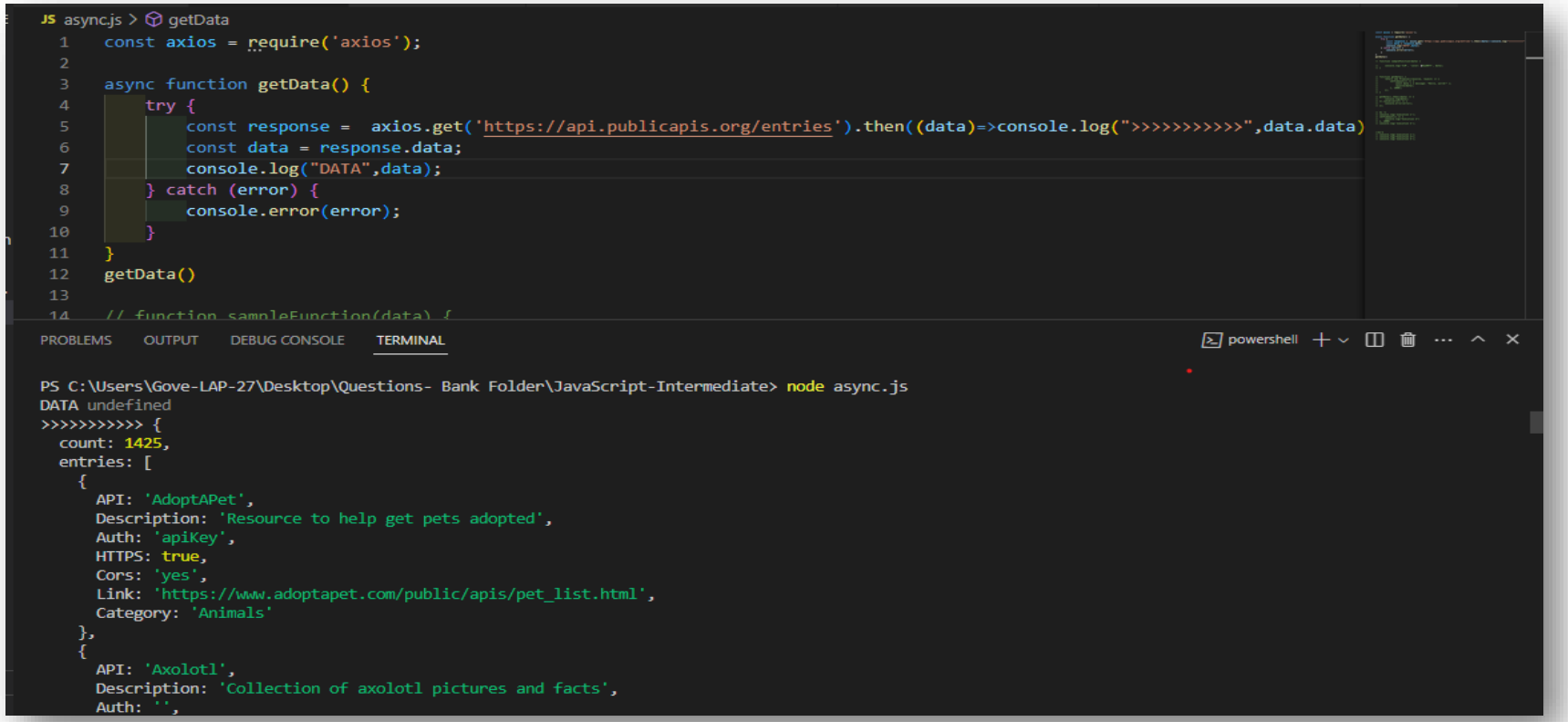
```
console.log('execution 1');
setTimeout(() => {
    console.log("execution 2")
}, 1000)
console.log('execution 3');
```

8.2 Keywords (async, await, then, Promise):

async: This keyword is used to define a function as an asynchronous function.

```
const axios = require('axios');

async function getData() {
    try {
        const response = axios.get('https://api.publicapis.org/entries');
        const data = response.data;
        console.log(data);
    } catch (error) {
        console.error(error);
    }
}
getData()
```

8.3 Creating async function



8.5 Promise Keyword

```
function getData() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            const data = { message: "Hello, world!" };
            resolve(data);
        }, 2000);
    });
}

getData().then((data) => {
    console.log(data);
}).catch((error) => {
    console.error(error);
});
```

Output :

```
PS C:\Users\Gove-LAP-27\Desktop\Questions- Bank Folder\JavaScript-Intermediate> node async.js
{ message: 'Hello, world!' }
PS C:\Users\Gove-LAP-27\Desktop\Questions- Bank Folder\JavaScript-Intermediate>
```

9.Exception Handling

- **Exception Handling :**
  - Exception handling is the process of responding to unwanted or unexpected events when a computer program runs
- **JavaScript Errors:**

  try - try statement defines a code block to run

  catch - catch statement defines a code block to handle any error.

  Finally – finally statement defines a code block to run regardless of the result

  throw - throw statement defines a customer error

Try – Catch Statements

- The try–catch statement is used to catch and handle errors during the execution of a block of code.
- The try block contains the code that may throw an error, while the catch block contains the code that handles the error.

```
try {
    let vacancyOpen = false
    if (vacancyOpen) {
        throw new Error("Eligible Candidate needes")
    }
}
catch (err) {
    throw new Error(errorMessage())
}
```

9.1 Function with try-catch

A try-catch block to handle errors that may occur within a function

```
function errorcatch(){
    try{
        let isPasswordOpen = true
        if(isPasswordOpen){
            console.log("Password Accepted")
            throw new Error("Password InCorrect")
        }
    }
    catch(err){
        console.log("InValid output:",err)
    }
}
errorcatch()
```

Output :

```
PS C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowledge\Devi
on/try.catch.js
Password Accepted
InValid output: Error: Password InCorrect
```

Settimeout()

1. It sets a timer (a countdown set in milliseconds) for the execution of a callback function, calling the function upon completion of the timer.

```
function getDevelopmentSteps() {
    setTimeout(() => {
        console.log()
        console.log("*1*Requirement Gathering")
        setTimeout(() => {
            console.log("*1*Analysis")
            setTimeout(() => {
                console.log("*1*Specification Document")
                setTimeout(() => {
                    console.log("*1*Pseudo Code")
                    setTimeout(() => {
                        console.log("*1*Code Implementation")
                        setTimeout(() => {
                            console.log("*1*Code Review")
                            setTimeout(() => {
                                console.log("*1*Code moved to develop branch")
                                setTimeout(() => {
                                    console.log("*1*Dev Deployment")
                                }, 8000);
                            }, 7000);
                        }, 6000);
                    }, 5000);
                }, 4000);
            }, 3000);
        }, 2000);
    }, 1000);
```

Output :

```
*1*Requirement Gathering
*1*Analysis
*1*Specification Document
*1*Pseudo Code
*1*Code Implementation
*1*Code Review
*1*Code moved to develop branch
*1*Dev Deployment
```

9.2 Function with try catch and finally

- The try Statement defines the code block to run (to try).
- The Catch Statement defines a code block to handle any error.
- The finally statement defines a code block to run regardless of the result

Syntax :

try { try statements }

Catch (error) { cache statements }

Finally ( ) { codes that gets executed anyway }

```
let courseAvailable = true
try {
    if (courseAvailable) {
        throw new Error("Registration Closed")
    }
}
catch (err) {
    console.log('Error Messagge :' + err)
}
finally {
    console.log("Finally Day Ends")
}
```

Function with try catch and finally

Try block will not throw any error and catch block will not have exception to throw. As expected, the finally block will execute the code.

```
//try - catch - finally
function vacancyCheck(){
let vacancyOpen = true
try {
    if (vacancyOpen) {
        console.log("Upload your Resume")
        throw new Error("Eligible Candidate")
    }
}
catch (err) {
    console.log("An Error Message");
    console.log('Error Messagge :' + err)
}
console.log("---------------------------")
}
vacancyCheck()
```

Output :

```
Upload your Resume
An Error Message
Error Messagge :Error: Eligible Candidate
---------------------------
```

9.3 Asynchronous function with try catch

- Try..catch is a javascript language that allows you to handle error in controlled way
- In asynchronous code,’try-catch provide a way to handle error occur during asynchronous operation

```
async function getData() {
    try {
        const response = await new Promise((resolve, reject) => {
            setTimeout(() => {
                resolve('Data received');
            }, 1000);
        });
        console.log(response);
    } catch (error) {
        console.error(error);
    }
}
getData()
```

Output :

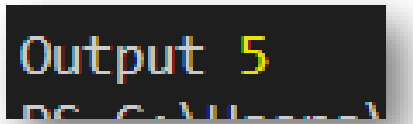
```
-----Async try catch------
Data received
```

9.4 Difference between Throw & return

- In JavaScript “throw “ and “return” are both used to control the flow of code execution , but they serve different purposes.
- Return:
  - “return “ is used to end the execution of a function and return a value to the calling code.
  - The code after the return statement is not executed

```
function add(input1 , input2) {
    let sum = input1 + input2
    return sum;
}

const output = add(2, 3);
console.log("Output",output);
```

Output :



- Throw

throw is used to signal that an error has occurred in the code and stop the normal execution of the program.

```
//throw statement sample
function divide(a, b) {
    if (b === 0) {
        throw new Error('Cannot divide by zero');
    }
    return a / b;
}

try {
    const quotient = divide(10, 0);
    console.log(quotient);
} catch (error) {
    console.error(error.message);
}
```

Throw error in code execution

Output :

```
Cannot divide by zero
PS C:\Users\Gove-LAP-27\Deskt
```

Without error facing execution

```
2
```

Difference between return and throw

return is used to retun a value and end the execution of a function.

throw is used to signal an error and stop the normal execution of the program

9.5 Catch block without error capturing

It is achieved by omitting the error parameter that is usually passed to the catch block.

```
try{
    let isPasswordOpen = true
    if(isPasswordOpen){
        console.log("Password Accepted")
        throw new Error("Password InCorrect")
    }
}
catch{
    console.log("InValid output:")
}
```

Output

```
PS C:\Users\Gove-LAP-27\Desktop\DEVI\
c/exception/try.catch.js
Password Accepted
InValid output:
```

Catch block capturing with error

The error parameter that is usually passed to the catch block.

```
try{
    let isPasswordOpen = true
    if(isPasswordOpen){
        console.log("Password Accepted")
        throw new Error("Password InCorrect")
    }
}
catch(err){
    console.log("InValid output:",err)
}
```

Output

```
PS C:\Users\Gove-LAP-27\Desktop\DEVI\Base Knowl
c/exception/try.catch.js
Password Accepted
InValid output: Error: Password InCorrect
```

9.6 Exception handling with .catch in Async function

then : when a promise is successful, you can then use the resolved data.

catch : when a promise fails, you catch the error, and do something with the error information.

```
async function getData() {
    try {
        const response = await new Promise((resolve, reject) => {
            setTimeout(() => {
                console.log("-----Async try catch------ ")
                resolve('Data received');
            }, 1000);
        });
        console.log(response);
    } catch (error) {
        console.error(error);
    }
}
getData()
    .then(() => console.log("Message"))
    .catch((error)=>console.error(error))
```

Output

```
-----Async try catch------
Data received
Message
```