Important Concepts of JS

In this module, we will be discussing some exceptions and cases of javascript's syntax and it's behaviour.

▼ **What kind of language is JS?**

It is dynamic and interpreted programming language.

JS is also known as scripting language as it provides flexibility to embed in other programming languages.

It's syntax is different from traditional programming object oriented programming languages.

Those basic concept's helps us to develop a better understanding of process of designing algorithms in javascript.

▼ **JavaScript Scope?**

The `scope` is something that defines the access to javascript variables.

In js variables can below to give two scopes:

1. **Global scope :**

   `Global varaibles` are variables that belongs to the `Global` scope and are accessible from anywhere in the program.

   example:

```
//global scoped variables
//variable which are declared without using any keywords
test_variable = "test value";
// variable declared above is global scope variable at the same time
// it is the worst way to declare variable in JavaScript.
/**
 * Solution:
 * We should always use var or let to declare variables.
 * If we need to declare variables that won't be modified then we should use const keyword to create varibles.
 */
```

2. **local scope**

   a. **Functional Scope:**

   Let's take an example of declaring variables using `var` keyword.

   The variables declared by using `var` keyword `float` all the way to the top.

```
//Functional Scope of Js variables
function scope1(){
    var top = "top";
    //accessing the variable before it's declaration
    bottom = "bottom";
    console.log("The value of bottom variable is:",bottom);
    //declaring a variable using var keyword
    var bottom;
}
//executing the function
scope1();// It doesn't give any error
//Above program is the best example of hoisting
//Hoisting -> It is a concept in Js Where variable is used before it's declaration.
```

   the above program is same as the below program

```
function scope1(){
    var top = "top";
    var bottom;
    //accessing the variable before it's declaration
    bottom = "bottom";
    console.log("The value of bottom variable is:",bottom);
    //declaring a variable using var keyword
    // var bottom;
}
```

Note: The key point to note about the `var` keyword is ie `the scope of the variables is the closest function scope.` What is `closest function scope` means?

example:

```
function scope2(print){
    if(print){
        //local variable of if block
        var insideIf = '12';
    }
    console.log("The insideIf variable value is: ",insideIf);
}
scope2(true);//The insideIf variable value is:  12
```

In above example the `scope2()` function is the function scope closest to the print variable.

The above program is equivalent to the program written below:

```
function scope2(print){
    var insideIf;
    if(print){
        insideIf = '12';
    }
    console.log("The insideIf variable value is: ",insideIf);
}
scope2(true);
```

💡 IN programming language like java , c++ if we try to access a block variable outside of the block then immediately it start's throwing error.

Example:

```
//first variable
var a = 1;
function four(){
    if(true){
        //second variable
        var a = 4;
    }
    console.log("The value of a is: ",a);
}
// console.log("The value of first a: ",a);
four();
console.log("The value of first a: ",a);
```

b. **Block Scope : Declaration with let Keyword:**

Whenever a variable is declared with `let` keyword then it's scope is the `closest block scope` .

```
//variables declared with let keyword has block scope access
function scope3(print){
    if(print){
        //block scoped variable
        let insideIf = '12';
    }
    console.log('The block {} scoped variable value is: ',insideIf);
}
scope3(true);//error
```

If we try to execute the above program then we will get error as block scoped variable `insideIf` is not accessible outside of the `if` block. ie

```
ReferenceError: insideIf is not defined
    at scope3 (D:\Software Training Classes\dsa-with-js\class-4\block_scope.js:7:59)
    at Object.<anonymous> (D:\Software Training Classes\dsa-with-js\class-4\block_scope.js:9:1)
    at Module._compile (node:internal/modules/cjs/loader:1254:14)
    at Module._extensions..js (node:internal/modules/cjs/loader:1308:10)
    at Module.load (node:internal/modules/cjs/loader:1117:32)
    at Module._load (node:internal/modules/cjs/loader:958:12)
    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:81:12)
    at node:internal/main/run_main_module:23:47

Node.js v18.15.0
```

▼ JavaScript Equality & Types

Js has different data types than in traditional languages such as Java. So let's try to understand how this impacts things such as `equality comparison`.

▼ Variable Types in Js

In JS there are seven primitive data types:

1. boolean
2. number
3. string
4. undefined
5. object
6. function
7. symbol

💡 `undefined` is a primitive value that is assigned to a variable that has just been declared.

💡 `typeof` is the primitive operator used to return the `type` of a variable.

```
//primitive data types in js
var is40 = false;//false
```

```
console.log("The data type is: ",typeof is40);// boolean

var age = 25;
console.log("The data type is: ",typeof age);// number

var lastName = 'Kumar';
console.log("The data type is: ",typeof lastName); // string

var fruits = ["Apple","Banana","Orange"];
console.log("The data type is: ",typeof fruits);// object

var person = {
    firstName: "Praveen",
    lastName: "Kumar"
}
console.log("The data type is: ",typeof person); // object

var nullVariable = null;
console.log("The data type is: ",typeof nullVariable); //object

var function1 = function(){
    console.log('test function');
}

console.log("The data type is: ",typeof function1); // function

var undeclared;
console.log("The data type is: ",typeof undeclared);// undefined
```

▼ **Truthy/Falsey Check**

As we know in conditional statement's (if statement) `True/False` checking is used.

In many languages, the parameter inside the `if()` function must be a boolean type.

But in Javascript(and other dynamically typed language) is more flexible with this.

```
if(node){
    //.........
}
```

In above code **node** is some variable.

`If that variable is empty, null, or undefined, it will evaluated as false.`

Some of the commonly used expressions that evaluate to false are:

1. false
2. 0
3. empty strings(" or "")
4. NaN
5. undefined
6. null

> 💡 Some common use expressions that evaluates to true are:
> * 1. true
> * 2. Any number other than 0
> * 3. Non-empty strings
> * 4. Non-empty object

Example:

```
var printIfTrue = '';

if(printIfTrue){
    console.log("The if block is evaluated as true");
}else{
    console.log("The if block is evaluated as false");
}
```

The above program will evaluate to false ie

```
The if block is evaluated as false
```

▼ **=== vs ==**

JavaScript is a scripting language, and variables are not assigned a type during declaration.

Instead, types are interpreted as the code runs.

Hence,

`===` operator is used to check `equality` more strictly then `==` operator.

`===` operator checks for both the `type` and the `value` of the operands.

`==` operator checks only the `value` of the operands.

Example:

```
"5" == 5  returns true because here `==` operator checks only if the both values are same or not
"5" === 5 returns false as `===` operator checks the value of the operands
which is same as well as type of the operands which is different ie one
is string & other one is number .
```