

## 1.Tokens

Smallest unit of any programming language.

Keywords : a predefined reserved word which js engine will understand.

- a. Every keyword must be lowercase.
- b. A keyword cannot be used as identifier.
- c. Eg : if , for , let , const ,null, var

2.Identifiers : name provided by programmer to the components of js like variable, function, class etc.

Rules for identifiers

- i. An identifier cannot start with number.
  - ii. Except \$ and \_ no other special character is allowed
  - iii. We cannot use keywords as identifiers.
3. Literals/values: The data which is used in js code/program.

Operator : perform an action on data

Some popular engines

- 1. V8 for google
- 2. SpiderMonkey for Mozilla
- 3. JavaScriptCore Safari
- 4. Chakra for Internet Explorer

## Characteristics of Javascript

It is purely object oriented language (based on objects for storing most kind of data)

Interpreted language., not compiled language

- Line by line execution, like first it will check if it correct it execute and check next line

javascript uses just in time (JIT)

Javascript is synchronous in nature : single threaded architecture have one stack for execution.

## Implementation of JS

1.internal within html document by using script tag

2.External we need to create one new file extension should be .js and link with script src .

a) We can create a separate file for js with extension.js

b)Link the js file with html page using src attribute of script tag.

```
<html>

<head>

<title>js separate</title>

</head>

<body>

<h1>linking js</h1>

<!-- <script src="path/filename.js"></script> -->

<script src="../js/js1.js"></script>

</body>

</html>
```

## Typeof

It is a keyword use as unary operator to identify the type of data.

typeof - will check the value belongs to which category

(typeof 10);

Syntax

typeof value

Program

```
console.log(typeof "samsung");//string
console.log(typeof `samsung`);//string
console.log(typeof 'samsung');//string
console.log(typeof 1);//number
```

```
console.log(typeof null);//object
console.log(typeof undefined);//undefined
console.log(typeof true);//boolean
console.log(typeof false);//boolean
```

Note

Typeof null is never null, it is considered as object

## Types of values or datatypes in javascript:

### 1.Number

a. The numbers between  $-(2^{53}-1)$  and  $(2^{53}-1)$ ---number. Eg : 1----number type

### 2. String

### 3.Boolean

a. True 1

b. False 0

4. Null : it is keyword , it define value

5.Undefined : it is keyword , it define value

a. When we declare a variable in js, js engine implicitly assign undefined value to it.

### 6. Object

Can we run js outside the browser?

We can run js outside browser with help of node.

## Node

Node is a bundle(extra layer) of google v8 engine and built in methods using c++.

It serves as environment to run js without help of browser/outside browser.

We use js(node) for business logic.

This invention helped js to gain its popularity in usage as a backend language.

With the help of node we can run without browser.

# variables

A named block of memory which is used to store the value is known as variable.

(container to store data)

Js is not strictly type language as we don't have to specify what type of data we want to store. It is dynamic type

language it will understand while execution .It is not necessary to specify type of data during variable declaration.

In a variable we can store any type of value.

It is mandatory to declare a variable before using.

We create variable using variable declaration followed by identifier and we can store anything

## Syntax for variable declaration

```
var a; // declaration stmt
```

```
a=10; // initialization / assignment
```

```
console.log(a);
```

```
let b;
```

```
b=20;
```

```
console.log(b);
```

```
const c=30;
```

```
console.log(c);
```

# Understanding the scope of variable

**Scope:** The visibility of a member to access it is known as scope.

**Global scope:** it has highest accessibility can be access anywhere.

**Block scope :** The visibility of a member is only within the block where it is declared. A member with

block scope can be used only inside the block where it is declared, it cannot be used outside.

**Note:** The variable declared with let and const have block scope.

Eg1:

```
{
```

```
let a=10;
console.log(a);//can be used
}
console.log(a);//cannot be used
```

Eg2:

```
{
const a=10;
console.log(a);//can be used
}
console.log(a);//cannot be used, error variable not defined
```

Eg3:

```
{
var a=10;
console.log(a);//can be used
}
console.log(a);//can be used
```

## Var

- If we declare a variable using var it has global scope, even inside block it acts has same

## let

- A variable creating using let we can use only inside block where it is declared.
- We cannot declare more than 1 with same name in same scope.

Var	Let	const
Var is global scope.	Let is block scope	Const is also block scope
We can declare multiple variable with same name(the most recently created variable Will be used.	We cannot declare 2 variables With the same name within a block.	We cannot declare 2 variables With the same name within a block.
The value of variable can be Modified.	The value of variable can be Modified.	The value of variable cannot be Modified.
<pre>var a=10; a=20; console.log(a);  //20</pre>	<pre>let a=10; a=20; console.log(a);//20  //reference error</pre>	<pre>const a=10; a=20; console.log(a);//we cannot modify the data with new value.  //TypeError</pre>
We Can declare var without initialization. Eg:  <pre>var a;</pre>	We Cannot declare let without initialization. Eg:  <pre>let b;</pre>	We Cannot declare const without initialization. Eg:  <pre>const c; //Syntax error</pre>
The variable declared using var Belongs to global object (window), We can access them using window Object.	The variable declared using let does Not belongs to global object we cannot Use them with the help of window.	The variable declared using const does Not belongs to global object we cannot Use them with the help of window.
The variable declared using var is hoisted does not belong to temporal dead zone, can be used before initialization.	The variable declared using let is hoisted, the variable goes to temporal deadzone. Therefore it cannot be used before initialization.	The variable declared using const is hoisted, the variable goes to temporal deadzone. Therefore it cannot be used before initialization.
Declaring var inside function will not have global scope.		

## Browser

It's a application which is running over machine or processor .

1. He can understand html, css behaves like compiler and interpreter which convert all these into machine language.

2. Browser give environment to run js instruction.

3. In browser we have many sub-application.

In browser we have js engine(sub application)

a. It is used to translate js instruction to machine level language.

4.

5. Every browser will have a js engine to run js code. Therefore the browser become an environment to run js

6. Browser has a default console.

# Hoisting

Js allows a programmer to use a member (variable) before the declaration statement. This characteristic is known as hoisting.

What makes js hoisting?

It runs js in 2 phases

1. Allocates memory variables and assign value(undefined)
2. Instruction get executed (top to bottom).

Since the declaration is already done we can use.

Understanding execution in js

1. Every time when js engine runs a js code , it will first create a Global Execution context.
  - i. Global execution context is a block of memory

Functional area /

Execution area

The global execution context has 2 parts

- ii. Variable area.
- iii. Functional area or execution area.

# Typecasting

Implicit typecasting/type conversion : js engine convert one type of data to another type implicitly of data when wrong type of value is used in expression .

Number zero (0), null , NaN, empty string(""), undefined all these values are considered as false when they are converted to boolean.

- 1.In js everything rather above consider as true.

```
console.log(100?10:20); //every non zero will consider than 0
```

```
console.log('hi'?10:20);//
```

```
var a=5-'1'; // string converted to number and add
```

```
console.log(a);
```

```
var b=5+'1'; // string converted to number and concat
```

```
console.log(b);
```

```
var x=5-'a'; // hence it is converted to a special number NaN
```

```
console.log(x);  
var y=(typeof (5-'a'));  
console.log(y);
```

2.Any arithmetic operation with NaN result is NaN

Two NaN is never consider as equal

```
console.log( NaN === NaN); //false  
console.log( 10 === NaN); //false
```

## Logical OR

Logical OR behaves differently if LHS value or RHS value is non boolean

Step 1: it converts the LHS value to boolean.

Step 2 : if the converted LHS value is true than it returns the original value present in the LHS

Eg

```
console.log(20 || 10); // so output will be 20
```

if the converted LHS value is false than it returns the original value present in the RHS

```
console.log(0 || 10); // so output will be 10
```

## Explicit Type Casting :

The process of converting from one type of value to another type of value is known as explicit type casting.

Case 1 : Conversion of any type to number

Syntax

```
Number(data-to-be-converted)
```

i.If a string is valid number we get real number

```
console.log(Number('123')); // 123
```

ii.If the string consist any other character then we get NaN as output.



```
a. console.log( Number ('a')); //NaN
```

Boolean to Number

```
console.log(Number (false)); //0
```

```
// when we convert false to 0 the result will 0
```

```
console.log(Number (true)); //1
```

```
// when we convert true to 0 the result will 0
```

```
console.log(Number (20>10)); // 1
```

## Decision Statements

Decision statement helps to skip a block of instruction when we don't have favoring situation. Eg:  
The

instruction of loading home page should be skipped if the entered password is incorrect.

Decision statement of js:

1.If

if(condition)-----> any expression whose result is Boolean

```
{  
}
```

2.If else

if(condition)-----> any expression whose result is Boolean

```
{  
}
```

```
else{
```

```
}
```

3.Else if

```
var firstnumber=Number( prompt("enter first number"));
```

```
var secondnumber=Number( prompt("enter second number"));
```

```
var thirdnumber=Number( prompt("enter third number"));
```

```
var fourthnumber=Number( prompt("enter fourth number"));
```

```
if(firstnumber<secondnumber && firstnumber<thirdnumber &&
```

```

firstnumber<fourthnumber )
{
console.log(` ${firstnumber} is smallest` );
}
else if( secondnumber<thirdtnumber && secondnumber<fourthnumber )
{
console.log(` ${secondnumber} is smallest`);
}
else if(thirdtnumber<fourthnumber)
{
console.log(` ${thirdtnumber} is smallest`);
}
else {
console.log(` ${fourthnumber} is smallest`);
}

```

## Switch

Switch is keyword and we pass value not boolean condition

Switch is faster than if else

Case followed by value or expression followed by : and then statements

Default is optional

And we can write default anywhere , not necessary to write at end.

Syntax

```
switch(value)
```

```

{
case value : {
statement;
}
case value : {
statement;
}
}

```

.

```

.
.
default : {
statement;
}
}

```

1.A case blocks gets executed if the value passed to switch matches with value present in case

2.When a case is favorable the case blocks gets executed as well as all the blocks present below in the switch gets executed.

3. We can have only 1 default inside switch.

4.Default can be written anywhere in switch.

```

switch(1)
{
case 1 : {console.log(`case 1`);}
default : {console.log(`case 1`);}
case 2 : {console.log(`case 1`);}
case 3 : {console.log(`case 1`);}
}

```

## Break

- It is a control transfer statement.
- It can be used inside either in switch block or loop block only.
  - When a break statement is encountered the control list transferred outside the current switch or loop block.

```

switch(6)

```

```
{  
case 1 : {console.log(`case 1`);}  
break;
```

Literal : the value will not change

True and false are literal

```
default : {console.log(`case 1`);}  
case 2 : {console.log(`case 1`);}  
break;  
case 3 : {console.log(`case 1`);}
```

## Looping

It is also called iteration.

The process of executing a instruction /block of instruction repeatedly multiple times.

Note: when we design a loop it is a responsibility of a programmer to break the loop after achieving the desire task, if not the

program get into infinite loop statement.

If we don't break loop it will be infinite and control will not come out of loop.

Condition is use to break loop after some certain

Loop statements

1.While

a. Syntax

```
while(condition){  
statement to be repeated;  
}
```

2.Do-while :

The do-while loop loops through a block of code once, then the condition is evaluated. If the condition is

true the statement is repeated as long as specified condition is true.

a. number of iteration in do while is 1

Syntax

```
do{
```

```
// statement;
```

```
}
```

```
while(condition);
```

### **Note:**

a. in do while loop the body of the loop is executed first then the condition is evaluated.

b. If the condition evaluated is true the body of the loop inside the do statement is executed again.

c. The statement inside the do statement is executed again.

d. This process continue until the condition evaluates to false. Then the loop stops.

3. For

4. For-in etc