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 suing 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 objec**t**

## 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. |
| `var a=10;`<br>`a=20;`<br>`console.log(a);`<br><br>`//20` | `let a=10;`<br>`a=20;`<br>`console.log(a);//20`<br><br>`//reference error` | `const a=10;`<br>`a=20;`<br>`console.log(a);//we cannot modify the data with new value.`<br><br>`//typeerror` |
| We Can declare var without initialization.<br>Eg:<br><br>`var a;` | We Cannot declare let without initialization.<br>Eg:<br><br>`let b;` | We Cannot declare const without initialization.<br>Eg:<br><br>`const c;`<br>//Syntax error |
| 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 thirdtnumber=Number( prompt("enter third number"));

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

if(firstnumber<secondnumber && firstnumber<thirdtnumber &&

```
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

# Functions

• Function is a block of instruction which is used to perform a specific task

• A function get executed only when it called.

• The main advantage of function is we can achieve code reusability.

• To call a function we need its reference and ()

**Note**: in javascript functions are beautiful, every function is nothing but an object.

Syntax to create a function.

Generally we can create a function in 5 ways:

1. Function declaration statement(function statement).

2. Function expression.

Function declaration / statement(function statement).

Syntax :

function identifier ([list_of_parameter,...])

{

statements;

}

1.

Note:

i. Function is object

ii. Name of function is variable which holds the reference of function object.

iii. Creating a function using function statement supports function hoisting.

iv. Therefore we can also call a function before function declaration.

v. when we try to log function name the entire function defination is printed.

```
console.log('start');
console.log(test);
function test(){
console.log('Hello');
}
console.log('start');
```

test - it is a function variable - it will return body of function

test () - invoking a function.

To call a function - function_name();// we can also pass data

Function_name(arguments_list,.....);

```
console.log('start');
function test(){
```

```
console.log('Hello');
}
test();
test();
test();
console.log('start');
```

parameter(placeholder: variable to hold  data when function is called)

```
console.log('start');
function test(a){
console.log('Hello');
console.log(a);
}
test(10); argument : values which are passed.
console.log('start');
```

Parameters

The variables declared in the function defination is known as parameters.

The parameters have local scope (can be used only inside function body).

Parameters are used to hold the values passed by calling a function.

Eg:

```
function sum(a, b){
console.log(a + b)
}
// a and b are variables local to the function sum
```

Arguments

The values passed in the method call statement is known as arguments.

Note : An argument can be a literal, variable or an expression which gives a results.

Eg 1:

sum(10,20); // 10,20 are literals used as arguments

Eg 2:

sum(-10+3,-20);//-27

Eg 3:

sum(a ,b); //a and b are variable used as argument

## return

1. It is a keyword used as control transfer statement in a function.

Return will stop the execution of the function and transfer control along with data

to the caller.

2.

```
function toMeters(cms){
return (cms/100);
}
var cms =2546;
console.log(toMeters(cms));
var m =toMeters(cms );
console.log(m);
```

The ability of a function to use as value is called first class function

## 2. Function as Expression

Syntax :

```
var/let/const identifier = function (){
}
```

The function is used as value

Disadvantage :

• function is not hoisted

• We cannot use a function before declaration.

```
a();// error
var a = function (){ /* a has address of function*/
console.log("fun");// will get entire function definition
}
```

In the above example function is not hoisted instead variable is hoisted and assigned with default value undefined. therefore type of a is not function it is undefined.

## This

It is a keyword used as variable.

It holds the address of global window object.

Therefore with the help of this variable we can use members of global window object.

Whenever we call a function a new execution context is created. Inside that all local variable declare inside will be there and 1 more "this" will be there which is different from this

Inside this it have address of which function belongs to.

a.In javascript 'this' is a property of every function.(every function will have this keyword)

b. Generally this contains the current execution context to which the function belongs.

## 3.Arrow Functions

• Arrow function was introduce from ES6 OF JS.

• The main function of using arrow function is to reduce syntax

Syntax

```
( parameter_list,...) => {}
```

Note:

1. Parameter is optional.

2. If function has only one statement , then block is optional.

3.it is mandatory to create a block if return keyword is used.

Error if we write like this

```
const c =(n1+n2) => return n1+n2;

console.log(c(10,20));
```

Correct solution

```
const c =(n1, n2) => { return n1+n2};

console.log(c(10,20));
```

Operation taking parameter using arrow function

## 4.IIFE-(Immediate invocation function)

When a function is called immediately as soon as the function object is created.

Steps to achieve

• Treat a function like a expression by declaring in pair of bracket.

Add another pair pf braces next to it which behaves like function call statement.

○ Eg1

```
(function abc(){
console.log("Hi");
})();
```

○ Eg2

```
let a= ( ()=>{console.log("hello"); return 10})();

console.log(a);
```

## 5.Anonymous function

The function declare without any name which is called as anonymous function

Syntax:

function (){

instruction

}

# Js Engine Working principle?

-JavaScript is a client-side scripting language and one of the most efficient,

commonly used scripting languages. The term .client-side scripting language means

that it runs at the client-side( or on the client machine) inside the web-browsers,

but one important thing to remember is that client's web-browser also needs to support

the JavaScript or it must be JavaScript enabled. Nowadays, most of the modern web browsers

support JavaScript and have their JavaScript engines.

For example, Google Chrome has its own JavaScript

engine called V8.

-Now let's see how the JavaScript engine handles and runs .js code.

-In this case, we have used a chrome browser

to run our program that has the "V8" JavaScript engine,

which is also used for creating the Node.js

-As we already know, JavaScript is an interpreted language that means it gets executed in line by line manner

(or which means the JavaScript engine converts the Js code line by line and runs in the same manner instead of converting the whole program once).

Step 1: Parser

-This is the first stage of the engine, every time we run a JavaScript program,

our code is first received by the "parser" inside the JS engine.

The parser's job is to check the JavaScript code for syntactic errors in line

by line manner because JavaScript is an interpretive scripting language, so whenever an error is detected by the parser,

it throws a kind of error and stops execution of the code.

Step 2: AST

-Once the parser checks all JavaScript codes and gets satisfied that there are no mistakes/errors in the code,

it creates the data structure called AST (it stands for Abstract Syntax Tree).

Step 3: Conversion to Machine code

-Once the Abstract Syntax Tree is created by the parser, the JavaScript engine converts the

JavaScript code into the machine code (or in the language that machine can understand).

Step 4: Machine code

-When the program written in the JavaScript gets converted in the machine language (or in byte code),

the converted code is sent to the system for execution, and finally, that byte code run by the

system/engine.