1. **What is lexical structure?**

A programming language’s lexical structure specifies a set of some basic rules about how code should be written in it. Rules like what variable name look like the delimiter characters for comments, and how one program statement is separated from the next.

For how we write Whitespaces and Line Breaks, Case Sensitive, Literals and Identifiers.

1. **What is Unicode?**

Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language. It has been adopted by all modern software providers and now allows data to be transported through many different platforms, devices and applications without corruption.

1. **Explain all the keywords present in the JavaScript with examples.**
   1. **goto keyword :** Used to return execution control to a specific location. In general, the goto can be accomp the break and continue keywords.

**Ex:**

var no=0;

sposition

document.write(" something print here ");

no++;

if(no < 10) goto sposition;

* 1. **in keyword :** It is an operator returns true if the specified property is present in the specified object, else it returns false.

**Ex:**

var fruits={f1: “apple”, f2: “banana”, f3: “orange”};

console.log('apple' in fruits);

3. **instanceof :** Returns true if the object is an instance of the class otherwise false

**Ex :**

var fruits=["apple", "banana", "orange"];

fruits instanceof Object;

fruits instanceof Array;

fruits instanceof String;

4. **argument :** Represents the list of parameters passed to the function when calling the function.

**Ex :**

const func = function(p1, p2, p3) {

const param = Array.from(arguments);

console.log(param)}

func(11, 22, 33);

5. **public :** It is an access modifier that can be used with attributes, classes, constructors and methods which make it accessible to other classes.

**Ex :**

public class Employee {

public String efn = "Joseph";

public String eln = "Doe";

}

class MainClass {

public static void main(String[] args) {

Employee obj = new Employee ();

System.out.println("Name= " + obj.efn + " " + obj.lname);

}

}

6. **Do :** Used to define a do-while loop.

**Ex :**

var a=1;

do {

document.write("loop is running for " + a + "times</p>");

a++;

}

while(a <= 10);

7. **Function :** Used to define a function to execute a block of code.

**Ex :**

var func = function(){

return "Hello";

}

alert(func());

8. **class :** Used to define a class.

**Ex :**

public class Employee {

public String efn = "Joseph";

public String eln = "Doe";

}

9. **return :** Used to return from the function or method with or without a value.

**Ex :**

var func = function(){

return "Hello";

}

10. **Eval :** Used to evaluate a specified string. The eval use as a global function eval().

**Ex:**

function fun( ) {

var str1=2;

var str1=3;

var res = eval(new String(str1 + str2));

document.write(res);

}

fun();

11. **For :** Used to define a loop, for loop to repeatedly execute a block of code until a condition true.

**Ex :**

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

document.write("The loop is running for " + a + " times");

}

12. **if :** Used to define a conditioned construct. If the statement is used to run a block of code if the condition is true.

**Ex :**

var date = new Date();

var day = date.getDay();

if(day==5) {

alert("This is weekend!");

} else {

alert("This is non-weekend!");

13. **Break :** Used into a loop break or stop the execution of the loop.

**Ex :**

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

if(a == 5)

break;

document.write("The loop is running for " + a + " times");

}

14. **debugger :** Used to stop the execution of javascript code and call debugging function if define. Debugger keyword word the same as the break.

**Ex :**

var prod = 10 \* 10;

debugger;

document.getElementbyId("id").innerHTML = prod;

15. **Yield :** Used to pause and resume a generator function. The generator function is the same as a normal function but for returning a value in place of return it uses yield keyword.

**Ex :**

function\* iter( a ) {

while (a < 4) {

yield a++;

}

}

const i = iter ( 1 );

console.log(i.next().value);

console.log(i.next().value);

16. **Continue :** Used to a loop to continue the loop and skip the following statements inside the loop.

**Ex :**

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

if(a == 5)

continue;

document.write("The loop is running for " + a + " times");

}

17. **null** : Used to represent a special data type no value.

**Ex :**

var age = null;

alert(age);

18. **New :** Used to create an object.

**Ex :**

Employee obj = new Employee();

19. **throw :** Used in a try block to explicitly throw an exception object.

**Ex :**

var i=1

try {

if(i == "") throw "is Empty";

if(x > 0) throw "positive";

if(x < 0) throw "negative";

}

catch(msg) {

message.innerHTML = "Input " + msg;

}

20. **while :** Used for while loop, while loop executes the block of code until the condition is true.

**Ex :**

var a=1;

while(a <= 10)

{

document.write("loop is running for " + a + "times</p>");

a++;

}

1. **What are shorthand operators, explain with a suitable example?**

If you’re adding, subtracting, multiplying, dividing, or remaindering two values, there’s no need to type out the entire equation. Just use the shortand operators.

**Operators :**

* Assignment Operators
  + Addition assignment – X = X + Y
  + Subtraction assignment – X = X-Y
  + Multiplication assignment – X = X\*Y
  + Division assignment – X = X/Y
* Comparison Operators
  + Equal (==)
  + Not equal (!=)
  + Strict Equal (===)
  + Strict not equal (!==)
  + Greater than ( > )
  + Greater than or equal ( >= )
  + Less than ( < )
  + Less than or equal ( <= )
* Arithmetic Operators
  + Remainder ( % )
  + Increment ( ++ )
  + Decrement ( -- )
  + Exponentiation operator ( \*\* )
* Bitwise Operators
  + Bitwise AND – a & b
  + Bitwise OR – a | b
  + Bitwise XOR – a^b
  + Bitwise NOT - ~a

* Logical Operators
  + Logical AND ( && )
  + Logical OR ( || )
  + Logical NOT ( ! )
* String Operators
* Conditional Operators
* Comma Operators
* Unary Operators
* Relational Operators

1. **What is “use Strict” in JavaScript?**

The purpose of "use strict" is to indicate that the code should be executed in "strict mode". With strict mode, you can not, for example, use undeclared variables.

**Ex :**

"use strict";

myFunction();

function myFunction() {

y = 3.14; // This will also cause an error because y is not declared

}