Assignment For Day 2   

1. What is lexical structure?

   Ans- In the world of computer programming langauge's lexical structure is a programming language's lexical structure which specifies a set of some basic rule about how code should be written in it. Rules like what variable names look like, the delimiter character of comment and how one program statement is separated from the next. It is the lowest-level of syntax of a language

   Some of the lexical structure of javascript are as follows -

   1) Unicode- JavaScript is written in Unicode. This means you can use emojis as variable names, but more importantly, you can write identifiers in any language, for example Japanese or Chinese with some rules
   2) Semicolon - JavaScript has a very c-like syntax, lot of sample code ends with semicolon, Semicolon aren't mandatory, and javascript does not have any problem that does not use them.
   3) White space- JavaScript does not consider white space meaningful, space and line break can be added in any fashion you might like. Even though is in theory.
   4) Case sensitive- JavaScript is case sensitive. A variable named something is different. The same goes to identifier.
   5) Comments - you can use two types of comment in JavaScript
   6) Literals and Identifier - An Identifier is a sequence of characters that can be use to identify a variable and function, an object

      Literals- a literals is a value that is written in the source code, for example a String boolean and also more constructs, like object literals or Arry list

   7) Reserved words - Those words which cant be use as identifier in javascript are called Reserved words.

      Some of them are as follow -

      break

      Do, instanceof, typeof, case, else,new, var, catch, finally, return

      Void, continue,for,switch, while,debugger, function

This, with, default, if, hrow ,delete,in, try, class, enum, extende. Super, const, export, import, implements, let, private,,public,interface, package, protected,static, yield

2. What is Unicode?

   Ans- Unicode is a Standard for character encoding. The introduction of ASCII characters was not enough to cover all the language. Therefore to overcome this situation, it was introduced. The Unicode Consortium introduced this encoding scheme.

3. Explain all the keywords present in the JavaScript with examples.

   Ans - All the keyword in JavaScript are as follow -

   1) break- The break statement terminates the current loop, switch or label statement and transfers the program control to the statement following the terminated statement.

      Eg.

JavaScript Demo: Statement - Break

```
let i = 0;

while (i < 6) {
  if (i === 3) {
    break;
  }
  i = i + 1;
}

console.log(i);
// expected output: 3

```

   2) case- the case is associated with the switch statement which evaluates an expression, matching the expression's value to a case clause, and executes statements associated with that case, as well as statement in case s the following matching case

      Eg.

JavaScript Demo: Statement - Switch

```
const expr = 'Papayas';
switch (expr) {
  case 'Oranges':
    console.log('Oranges are $0.59 a pound.');
    break;
  case 'Mangoes':
  case 'Papayas':
    console.log('Mangoes and papayas are $2.79 a pound.');
    // expected output: "Mangoes and papayas are $2.79 a pound."
    break;
  default:
    console.log(`Sorry, we are out of ${expr}.`);
}

```

3) catch- The catch block is associated with try block, if the code in the try blocks throws an exception then the code in the catch block will be executed.

Eg.

JavaScript Demo: Statement - Try...Catch

```
try {
  nonExistentFunction();
} catch (error) {
  console.error(error);
  // expected output: ReferenceError: nonExistentFunction is not defined
  // Note - error messages will vary depending on browser
}

```

4) class- classes are the template for creating objects.

Eg.

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
}
```

5) const- constant are block-scoped, much like variable declare using let keyword

   Eg.

   const numb = 44

6) continue- the continue statement terminate execution of the statements in the current iteration of the labelled loop and continues execution of the loop with the next iteration

   Eg.

7) debugger - the debugger statement invokes any available debugging functionality, such as setting a break point. If no debugging functionality is available, this statement has no effect.
8) default- use in a switch expression to specify the actions to be performed if no case.
9) delete - the delete operator removes a property from an object; if no more references to the same property are held. It is eventually released automatically.

   Eg.

10) do- it refer to the the do-while loop statement which create a loop that executes a specific statement until the test condition evaluates to false.The condition is evaluated after executing the statement, resulting in a specified statement executing at least once
11) else- it refers to the if and else statement, the else block is executed when the if statement fails.
12) export- the export declaration is used when creating javascript modules to export live bindings to functions, object or primitive values from the module so they can be used by other programs with the import declaration.

    Eg. export let name1, name2 name3

13) extends- the extends keyword is used in class declaration or class experience to create a class that is a child of another class.

    Eg. class Dateformatter extends Date {}

14) finally- the final statements are the statements that are executed after the try statement completes regardless of whether an exception was thrown or caught.
15) for- it relate with the for loop things
16) function - function are block of code performing specific task

    Eg.

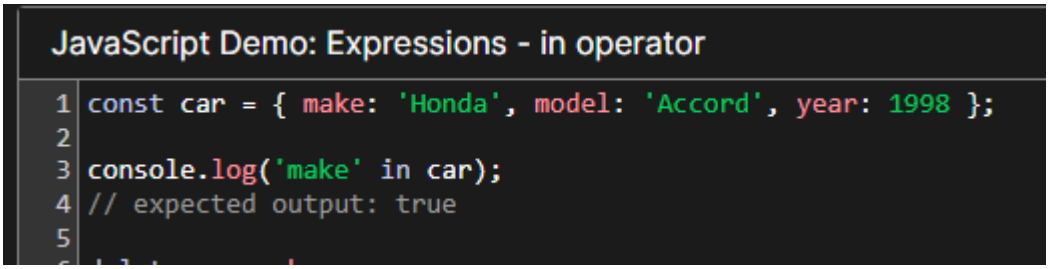
    function add(x, y){

    return x+y

    }

17) If - condition statement
18) import - the static import declaration is used to import read only live binding which is exported by another module.
19) in- the in operator return true if the specified property is in the specified object or its prototype chain.

    Eg.



20) Instanceof: The instanceof operator tests to see if the prototype property of a constructor appears anywhere in the prototype chain of an object. The return value is a boolean value.

Eg.

JavaScript Demo: Expressions - instanceof

```
function Car(make, model, year) {
  this.make = make;
  this.model = model;
  this.year = year;
}
const auto = new Car('Honda', 'Accord', 1998);

console.log(auto instanceof Car);
// expected output: true
```

21) New: the new operator lets developers create an instance of a user-defined object type or of one of the built-in object types that has a constructor function.

Eg.

JavaScript Demo: Expressions - new operator

```
function Car(make, model, year) {
  this.make = make;
  this.model = model;
  this.year = year;
}

const car1 = new Car('Eagle', 'Talon TSi', 1993);

console.log(car1.make);
// expected output: "Eagle"
```

22) Return- the return statement ends function execution and specifies a value to be returned to the function caller

Eg.

JavaScript Demo: Statement - Return

```
function getRectArea(width, height) {
  if (width > 0 && height > 0) {
    return width * height;
  }
  return 0;
}

console.log(getRectArea(3, 4));
// expected output: 12

console.log(getRectArea(-3, 4));
// expected output: 0
```

23) Super- The super keyword is used to access properties on an object literal or class's [[Prototype]], or invoke a superclass's constructor.

Eg.

```
super([arguments]) // calls the parent constructor.

super.propertyOnParent

super[expression]
```

24) Switch- The switch statement evaluates an expression, matching the expressions value to a case clause and executes statements associated with that case.

Eg.

JavaScript Demo: Statement - Switch

```
const expr = 'Papayas';
switch (expr) {
  case 'Oranges':
    console.log('Oranges are $0.59 a pound.');
    break;
  case 'Mangoes':
  case 'Papayas':
    console.log('Mangoes and papayas are $2.79 a pound.');
    // expected output: "Mangoes and papayas are $2.79 a pound."
    break;
  default:
    console.log(`Sorry, we are out of ${expr}.`);
}

```

25) This- this keyword refers to an object that is executing the current piece of code. It references the object that is executing the current function. If the function being referenced is a regular function, ‘this” references the global object.

Eg.

JavaScript Demo: Expressions - this

```
const test = {
  prop: 42,
  func: function() {
    return this.prop;
  },
};

console.log(test.func());
// expected output: 42
```

26) Throw- The throw statement throws a user-defined exception. Execution of the current function will stop and control will be passed to the first catch block in the call stack.

Eg.

JavaScript Demo: Statement - Throw

```
function getRectArea(width, height) {
  if (isNaN(width) || isNaN(height)) {
    throw 'Parameter is not a number!';
  }
}

try {
  getRectArea(3, 'A');
} catch (e) {
  console.error(e);
  // expected output: "Parameter is not a number!"
}
```

27) try- relates with the try and catch statement
28) typeof- the typeof operator returns a string indicating the type of the unevaluated operand.

Eg.

JavaScript Demo: Expressions - typeof

```
1 console.log(typeof 42);
2 // expected output: "number"
3
```

29) var - the var statement declares a function-scoped or globally-scoped variable, optionally initializing it to a value.

Eg. var x =1;

30) void- the void operator evaluates the given expression and then returns undefined.

Eg.

JavaScript Demo: Expressions - void operator

```
1 const output = void 1;
2 console.log(output);
3 // expected output: undefined
```

31) while- the while statement creates a loop that executes a specified statement as long as the test condition evaluates to true. The condition is evaluated before executing the statement.

Eg.

JavaScript Demo: Statement - While

```
1 let n = 0;
2
3 while (n < 3) {
4   n++;
5 }
6
7 console.log(n);
8 // expected output: 3
9
```

32) with- the with statement extends the scope chain for a statement.

Eg.

```
with (expression)
  statement
```

33) yield- the yield keyword is used to pause and resume a generator function.

Eg.

JavaScript Demo: Expressions - yield

```
function* foo(index) {
  while (index < 2) {
    yield index;
    index++;
  }
}

const iterator = foo(0);

console.log(iterator.next().value);
// expected output: 0
```

34) enum- refer to the enumerates in javascript, used to define a predefined list.

Eg.

```
const fruits={
APPLE: 'apple',
BANANA: 'banana',
Orange: 'orange',
}
```

35) implements- relates to the Document.implementationn property returns a DOM implementation object associated with the current document. Used to implement the interface in a class

Eg.

```
var modName = "HTML";
var modVer = "2.0";
var conformTest = document.implementation.hasFeature(
modName, modVer );

alert( "DOM " + modName + " " + modVer + " supported?: " +
conformTest );
```

36) Interface-
37) let- the let declaration declares a block-scoped local variable, optionally initialising it to a value.

Eg.

JavaScript Demo: Statement - Let

```
let x = 1;

if (x === 1) {
  let x = 2;

  console.log(x);
  // expected output: 2
}

console.log(x);
// expected output: 1
```

38) package- used to define an interface (interface contains all abstract methods)
39) private- class fields are public by default, but private class members can be created by using a hash #prefix. The privacy encapsulation of these class features is enforced by javascript itself

Eg.

```
class ClassWithPrivateField {
  #privateField;
}
```

40) protected- an access modifier can be used with attributes, classes, constructors and methods which make it not accessible to other classes.

41) Public- it is an access modifier that can be used with attributes, classes, constructors and method and make it accessible to other classes.
42) Static- used to define a static method in a class. Static methods are those methods that are not called on the object.
43) await- used to wait for javascript until the promise returns its result.

Eg.

```
async function fun() {
let promise = new Promise((resolve, reject) => {
setTimeout(() => resolve(" yes, it is done!"), 100)
});
let res = await promise; // wait until the promise ret
alert(result); // output give yes, it is done
}
};
fun();
```

44) Abstract - the concept of Abstract is to hide the implementation details and highlight an objects essential features to the users
45) Boolean - primitive data types of true or false
46) Byte- used to convert the given string of number, number value or boolean into a its byte representation.
47) Double- it is a number type which is double precision 64-bit binary format value
48) Finally- used in exception handling, finally block of code always execute regardless of whether the error is generating or not.
49) Float- relates to number data types of 64 bit floating Point
50) Goto- used to return execution control to a specific location. In general the goto can be accomplished by the break and continue keywords

    Eg.

```
var no=0;
sposition
document.write(" something print here ");
no++;
if(no < 10) goto sposition;
```

51) Int - data types of number
52) Long - data types of number with large size
53) BigInt - data types of Number which can store large number.
54) Native- native refers to several built-in object. Accessible anywhere in the program and will work the same way in any browser.
55) sort - sorting of item either in ascending or descending
56) Synchronized - a process of step by step order in sequence
57) Transient - Transient activation is a window state that indicates a user recently pressed a button , move a mouse, used a menu, or performed some other user interaction.
58) Volatile6true- Used to store or represent primitive data type Boolean ‘true’

Eg. var inp = true

59) false- used to store or represent primitive data type Boolean ‘false’

Eg. var inp = false

60) null- Used to represent a special data type no value

Eg.

Var age = null

61) for- Used to define a loop, for loop to repeatedly execute a block of code until a condition true

Eg. for(let i=0; i<n; i++){}

62) Eval- used to evaluate a specified string. The eval use as a global function eval()

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

Ans -

Name	Shorthand operator	Meaning
Assignment	X = f()	x = f()
Addition Assignment	X += f()	x = x + f()
Subtraction assignment	X -= f()	x = x - f()
Multiplication Assignment	x *= f()	x = x * f()
Division Assignment	x /= f()	x = x / f()
Remainder Assignment	x %= f()	x = x % f()
Exponentiation assignment	x **= f()	x = x ** f()
Left shift assignment	x <<= f()	x = x << f()
Right shift assignment	x >>= f()	x = x >> f()
Unsigned right shift assignment	x >>>= f()	x = x >>> f()
Bitwise AND Assignment	x &= f()	x = x & f()
Bitwise XOR assignment	x ^= f()	x = x ^ f()
Bitwise OR assignment	x |= f()	x = x | f()
Logical AND assignment	x &&= f()	x && (x = f())
Logical OR assignment	x ||= f()	x || (x = f())

Logical nullish assignment	x ??= f()	x ?? (x = f())

5. What is “use Strict” in JavaScript?

   Ans - JavaScript's strict mode, introduced in ECMAScript 5, is a way to opt in to a restricted variant of JavaScript, thereby implicitly opting-out of "sloppy mode". Strict mode isn't just a subset: it intentionally has different semantics from normal code. Browsers not supporting strict mode will run strict mode code with different behavior from browsers that do, so don't rely on strict mode without feature-testing for support for the relevant aspects of strict mode. Strict mode code and non-strict mode code can coexist, so scripts can opt into strict mode incrementally.

   Strict mode makes several changes to normal JavaScript semantics: -

   1. Eliminates some JavaScript silent errors by changing them to throw errors.
   2. Fixes mistakes that make it difficult for JavaScript engines to perform optimizations: Strict mode code can sometimes be made to run faster than identical code that’s not strict mode
   3. Prohibits some syntax likely to be defined in future versions of ECMAScript.