**Question 1: Explore and explain the various methods in console function.**

console.assert() - Log a message and stack trace to console if the first argument is false.

[console.clear()](https://developer.mozilla.org/en-US/docs/Web/API/Console/clear) - Clear the console.

[console.count()](https://developer.mozilla.org/en-US/docs/Web/API/Console/count) - Log the number of times this line has been called with the given label.

[console.countReset()](https://developer.mozilla.org/en-US/docs/Web/API/Console/countReset) - Resets the value of the counter with the given label.

[console.debug()](https://developer.mozilla.org/en-US/docs/Web/API/Console/debug) - Outputs a message to the console with the log level "debug".

[console.dir()](https://developer.mozilla.org/en-US/docs/Web/API/Console/dir) - Displays an interactive listing of the properties of a specified JavaScript object. This listing lets you use disclosure triangles to examine the contents of child objects.

[console.dirxml()](https://developer.mozilla.org/en-US/docs/Web/API/Console/dirxml) - Displays an XML/HTML Element representation of the specified object if possible or the JavaScript Object view if it is not possible.

[console.error()](https://developer.mozilla.org/en-US/docs/Web/API/Console/error) - Outputs an error message. You may use [string substitution](https://developer.mozilla.org/en-US/docs/Web/API/console#Using_string_substitutions) and additional arguments with this method.

console.exception() - An alias for error().

[console.group()](https://developer.mozilla.org/en-US/docs/Web/API/Console/group) - Creates a new inline [group](https://developer.mozilla.org/en-US/docs/Web/API/console#Using_groups_in_the_console), indenting all following output by another level. To move back out a level, call groupEnd().

[console.groupCollapsed()](https://developer.mozilla.org/en-US/docs/Web/API/Console/groupCollapsed) - Creates a new inline [group](https://developer.mozilla.org/en-US/docs/Web/API/console#Using_groups_in_the_console), indenting all following output by another level. However, unlike group() this starts with the inline group collapsed requiring the use of a disclosure button to expand it. To move back out a level, call groupEnd().

[console.groupEnd()](https://developer.mozilla.org/en-US/docs/Web/API/Console/groupEnd) - Exits the current inline [group](https://developer.mozilla.org/en-US/docs/Web/API/console#Using_groups_in_the_console).

[console.info()](https://developer.mozilla.org/en-US/docs/Web/API/Console/info) - Informative logging of information. You may use [string substitution](https://developer.mozilla.org/en-US/docs/Web/API/console#Using_string_substitutions) and additional arguments with this method.

[console.log()](https://developer.mozilla.org/en-US/docs/Web/API/Console/log) - For general output of logging information. You may use [string substitution](https://developer.mozilla.org/en-US/docs/Web/API/console#Using_string_substitutions) and additional arguments with this method.

[console.profile()](https://developer.mozilla.org/en-US/docs/Web/API/Console/profile) - Starts the browser's built-in profiler (for example, the [Firefox performance tool](https://developer.mozilla.org/en-US/docs/Tools/Performance)). You can specify an optional name for the profile.

[console.profileEnd()](https://developer.mozilla.org/en-US/docs/Web/API/Console/profileEnd) - Stops the profiler. You can see the resulting profile in the browser's performance tool (for example, the [Firefox performance tool](https://developer.mozilla.org/en-US/docs/Tools/Performance)).

[console.table()](https://developer.mozilla.org/en-US/docs/Web/API/Console/table) - Displays tabular data as a table.

[console.time()](https://developer.mozilla.org/en-US/docs/Web/API/Console/time) - Starts a [timer](https://developer.mozilla.org/en-US/docs/Web/API/console#Timers) with a name specified as an input parameter. Up to 10,000 simultaneous timers can run on a given page.

[console.timeEnd()](https://developer.mozilla.org/en-US/docs/Web/API/Console/timeEnd) - Stops the specified [timer](https://developer.mozilla.org/en-US/docs/Web/API/console#Timers) and logs the elapsed time in seconds since it started.

[console.timeLog()](https://developer.mozilla.org/en-US/docs/Web/API/Console/timeLog) - Logs the value of the specified [timer](https://developer.mozilla.org/en-US/docs/Web/API/console#Timers) to the console.

[console.timeStamp()](https://developer.mozilla.org/en-US/docs/Web/API/Console/timeStamp) - Adds a marker to the browser's [Timeline](https://developer.chrome.com/devtools/docs/timeline) or [Waterfall](https://developer.mozilla.org/en-US/docs/Tools/Performance/Waterfall) tool.

[console.trace()](https://developer.mozilla.org/en-US/docs/Web/API/Console/trace) - Outputs a [stack trace](https://developer.mozilla.org/en-US/docs/Web/API/console#Stack_traces).

[console.warn()](https://developer.mozilla.org/en-US/docs/Web/API/Console/warn) - Outputs a warning message. You may use [string substitution](https://developer.mozilla.org/en-US/docs/Web/API/console#Using_string_substitutions) and additional arguments with this method.

**Question 2: Write the difference between var, let and const with code examples.**

**Var**

The JavaScript variables statement is used to declare a variable and, optionally, we can initialize the value of that variable.

Example: var a =10;

* Variable declarations are processed before the execution of the code.
* The scope of a JavaScript variable declared with var is its current execution context.
* The scope of a JavaScript variable declared outside the function is global.
* function nodeSimplified(){
* var a =10;
* console.log(a); // output 10
* if(true){
* var a=20;
* console.log(a); // output 20
* }
* console.log(a); // output 20
* }

In the above code, you can find, when the variable is updated inside the if loop, that the value of variable "a" updated 20 globally, hence outside the if loop the value persists. It is similar to the Global variable present in other languages. But, be sure to use this functionality with great care because there is the possibility of overriding an existing value.

**let**

The **let** statement declares a local variable in a block scope. It is similar to **var**,inthat we can optionally initialize the variable.

Example: let a =10;

* The let statement allows you to create a variable with the scope limited to the block on which it is used.
* It is similar to the variable we declare in other languages like Java, .NET, etc.
* function nodeSimplified(){
* let a =10;
* console.log(a); // output 10
* if(true){
* let a=20;
* console.log(a); // output 20
* }
* console.log(a); // output 10
* }

## ****const****

const statement values can be assigned once and they cannot be reassigned. The scope of const statement works similar to let statements.

function nodeSimplified(){

const MY\_VARIABLE =10;

console.log(MY\_VARIABLE); //output 10

}

As per usual, naming standards dictated that we declare the const variable in capital letters. **const a =10**will work the same way as the code given above. Naming standards should be followed to maintain the code for the long run.

**Question 3: Write a brief intro on available datatypes in Javascript.**

String - represents sequence of characters e.g. "hello"

Number - represents numeric values e.g. 100

Boolean - represents boolean value either false or true

Undefined - represents undefined value

Null - represents null i.e. no value at all

Object - represents instance through which we can access members

Array - represents group of similar values

RegExp - represents regular expression