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

**Ans :=>** javascript provide us to different console functions they are following:

1) log()

2) error()

3) warn()

4) clear()

5) time() and timeEnd()

6) table()

7) count()

8) group() and groupEnd()

9) custom console logs

**1. console.log()**

Mainly used to log(print) the output to the console. We can put any

type inside the log(), be it a string, array, object, boolean etc.

ex: console.log('abc');

console.log(1);

console.log(true);

console.log(null);

console.log(undefined);

console.log([1, 2, 3, 4]); // array inside log

console.log({a:1, b:2, c:3}); // object inside log

**2. console.error()**

Used to log error message to the console. Useful in testing of code.

By default the error message will be highlighted with red color.

ex: console.error('This is a simple error');

**3. console.warn()**

Used to log warning message to the console. By default the warning message will be highlighted with yellow color.

ex:console.warn('This is a warning.');

**4. console.clear()**

Used to clear the console. The console will be cleared, in case of Chrome a simple overlayed

text will be printed like : ‘Console was cleared’ while in firefox no message is returned.

ex: console.clear();

**5.console.time() and console.timeEnd()**

Whenever we want to know the amount of time spend by a block or a function, we can make use of the time() and timeEnd() methods provided by the javascript console object. They take a label which must be same, and the code inside can be anything( function, object, simple console).

|  |
| --- |
| Ex: console.time('abc');   let fun =  function(){       console.log('fun is running');   }   let fun2 = function(){       console.log('fun2 is running..');   }   fun(); // calling fun();   fun2(); // calling fun2();  console.timeEnd('abc'); |

In the above code sample, we can see that the label is ‘abc’ which is same for both the time() and the timeEnd() method. If we increase the amount of code inside the block defined by these methods, then the time will increase. It is also worth remembering that the time returned to the console will be in milliseconds and might be different each time we refresh the page.

**5. console.table()**

This method allows us to generate a table inside a console. The input must be an array or an object which will be shown as a table.

|  |
| --- |
| Ex: console.table({'a':1, 'b':2}); |

**6. console.count()**

This method is used to count the number that the function hit by this counting method.

|  |  |  |  |
| --- | --- | --- | --- |
| Ex: for(let i=0;i<5;i++){      console.count(i);  **6. console.group() and console.groupEnd()**  group() and groupEnd() methods of the console object allows us to group contents in a separate block, which will be indented. Just like the time() and the timeEnd() they also accepts label, again of same value.   |  | | --- | | Ex: console.group('simple');    console.warn('warning!');    console.error('error here');    console.log('vivi vini vici');  console.groupEnd('simple');  console.log('new section'); | |  | |  | |

**7. Custom Console Logs**

User can add Styling to the console logs in order to make logs Custom . The Syntax for it is to add the css styling as a parameter to the logs which will replace %c in the logs as shown in the example below .

|  |
| --- |
| Ex: const spacing = '10px';    const styles =          `padding: ${spacing}; background-color: white; color: green; font-style:           italic; border: 1px solid black; font-size: 2em;`;    console.log('%cGeeks for Geeks', styles); |

**8. Console Sidebar**

Console sidebar is used to organize logs and provides clarity in debugging experience.

**Question 2 :=>** Write the different between var,let and const with code examples

**Ans :=>**

In Javascript one can define variables using the keywords ***var, let****or****const.***

**var** a=10;  
**let** b=20;

**const** PI=3.14;

**var:**The scope of a variable defined with the keyword “var” is limited to the “function” within which it is defined. If it is defined outside any function, the scope of the variable is global.**var is “function scoped”.**

**Scope of var**

**Scope** essentially means where these variables are available for use. var declarations are globally scoped or function/locally scoped.

The scope is global when a var variable is declared outside a function. This means that any variable that is declared with var outside a function block is available for use in the whole window.

var is function scoped when it is declared within a function. This means that it is available and can be accessed only within that function.

To understand further, look at the example below.

var greeter = "hey hi";

function newFunction() {

var hello = "hello";

}

Here, greeter is globally scoped because it exists outside a function while hello is function scoped. So we cannot access the variable hello outside of a function. So if we do this:

var tester = "hey hi";

function newFunction() {

var hello = "hello";

}

console.log(hello); // error: hello is not defined

We'll get an error which is as a result of hello not being available outside the function.

**let:**The scope of a variable defined with the keyword “let” or “const” is limited to the “block” defined by curly braces i.e. {} .**“let” and “const” are“block scoped”.**

### let is block scoped

A block is a chunk of code bounded by {}. A block lives in curly braces. Anything within curly braces is a block.

So a variable declared in a block with let  is only available for use within that block. Let me explain this with an example:

let greeting = "say Hi";

let times = 4;

if (times > 3) {

let hello = "say Hello instead";

console.log(hello);// "say Hello instead"

}

console.log(hello) // hello is not defined

We see that using hello outside its block (the curly braces where it was defined) returns an error. This is because let variables are block scoped .

### let can be updated but not re-declared.

Just like var,  a variable declared with let can be updated within its scope. Unlike var, a let variable cannot be re-declared within its scope. So while this will work:

let greeting = "say Hi";

greeting = "say Hello instead";

this will return an error:

let greeting = "say Hi";

let greeting = "say Hello instead"; // error: Identifier 'greeting' has already been declared

However, if the same variable is defined in different scopes, there will be no error:

**const:** The scope of a variable defined with the keyword “const” is limited to the block defined by curly braces. However if a variable is defined with keyword const, it cannot be reassigned.**“const” cannot be re-assigned to a new value. However it CAN be mutated.**

### const declarations are block scoped

Like let declarations, const declarations can only be accessed within the block they were declared.

### const cannot be updated or re-declared

This means that the value of a variable declared with const remains the same within its scope. It cannot be updated or re-declared. So if we declare a variable with const, we can neither do this:

const greeting = "say Hi";

greeting = "say Hello instead";// error: Assignment to constant variable.

nor this:

const greeting = "say Hi";

const greeting = "say Hello instead";// error: Identifier 'greeting' has already been declared

Every const declaration, therefore, must be initialized at the time of declaration.

This behavior is somehow different when it comes to objects declared with const. While a const object cannot be updated, the properties of this objects can be updated. Therefore, if we declare a const object as this:

const greeting = {

message: "say Hi",

times: 4

}

while we cannot do this:

const greeting = {

words: "Hello",

number: "five"

} // error: Assignment to constant variable.

we can do this:

greeting.message = "say Hello instead";

This will update the value of greeting.message without returning errors.

**Question 3 :=>** write a brief intro on variable datatypes in javascript

**Ans :=>**

JavaScript is dynamically typed (also called loosely typed) scripting language. That is, in javascript variables can receive different data types over time. Datatypes are basically typed of data that can be used and manipulated in a program.

**The latest ECMAScript(ES6) standard defines seven data types**: Out of which six data types are Primitive(predefined).

* **Numbers**: 5, 6.5, 7 etc.
* **String**: “Hello GeeksforGeeks” etc.
* **Boolean**: Represent a logical entity and can have two values: true or false.
* **Null**: This type has only one value : *null.*
* **Undefined**: A variable that has not been assigned a value is *undefined.*
* **Object**: It is the most important data-type and forms the building blocks for modern JavaScript. We will learn about these data types in details in further articles.

**Variables in JavaScript:**

Variables in JavaScript are containers which hold reusable data. It is the basic unit of storage in a program.

* The value stored in a variable can be changed during program execution.
* A variable is only a name given to a memory location, all the operations done on the variable effects that memory location.
* In JavaScript, all the variables must be declared before they can be used.

**Before ES2015**, JavaScript variables were solely declared using the *var* keyword followed by the name of the variable and semi-colon. Below is the syntax to create variables in JavaScript:

*var* var\_name;

*var x;*

The var\_name is the name of the variable which should be defined by the user and should be unique. These type of names are also known as **identifiers**. The rules for creating an identifier in JavaScript are, the name of the identifier should not be any pre-defined word(known as keywords), the first character must be a letter, an underscore (\_), or a dollar sign ($). Subsequent characters may be any letter or digit or an underscore or dollar sign.

Notice in the above code sample, we didn’t assign any values to the variables.We are only saying they exist.If you were to look at the value of each variable in the above code sample,it would be undefined.

Javascript is also known as **untyped** language. This means, that once a variable is created in javascript using the keyword var, we can store any type of value in this variable supported by javascript. Below is the example for this:

// creating variable to store a number

var num = 5;

// store string in the variable num

num = "GeeksforGeeks";

The above example executes well without any error in JavaScript unlike other programming languages.  
Variables in JavaScript can also evaluate simple mathematical expressions and assume its value.