1. Explore and explain the various methods in console function?

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');` → `abc`
- `console.log([1, 2, 3, 4]);` → `Array(4) [1, 2, 3, 4]`
- `console.log({a:1, b:2, c:3});` → `object { a:1, b:2, c:3}`
- `console.log(true);` → `true`

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.

`E.X:- console.error('This is a simple error');`

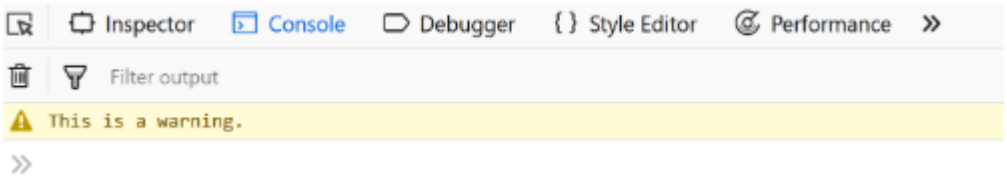
`output:`

console.warn()

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

`E.X:- console.warn('This is a warning.');`

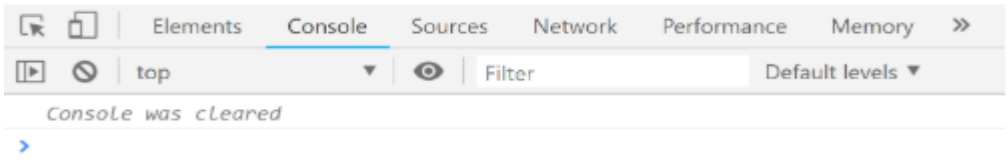
Output:



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.

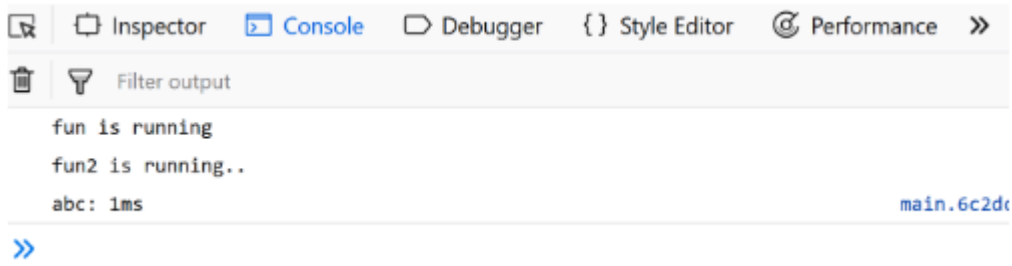
`E.X:- console.clear();`
`Output:-`



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).

E.X:-
```
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');
```
output:

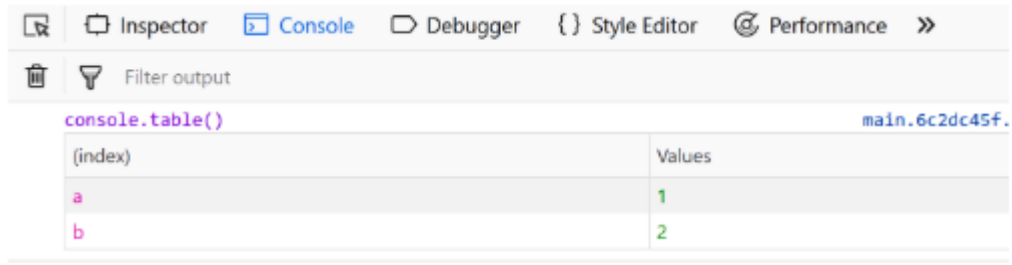


console.table()

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.

E.X:- `console.table({'a':1, 'b':2});`

Output:



console.count()

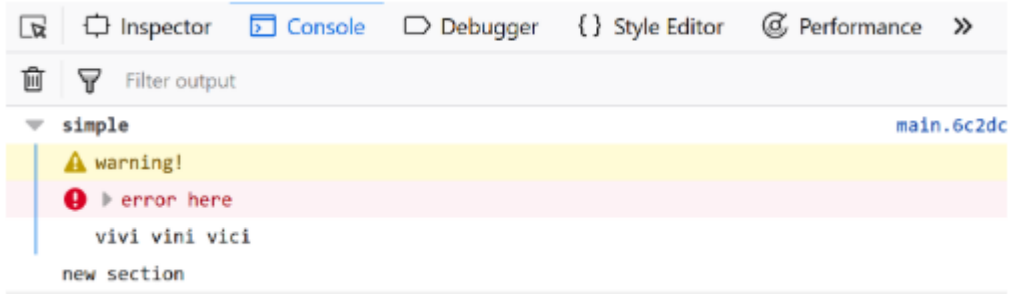
This method is used to count the number that the function hit by this counting method.

E.X:-
```
for(let i=0;i<5;i++){
    console.count(i);
}
```
Output:

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.

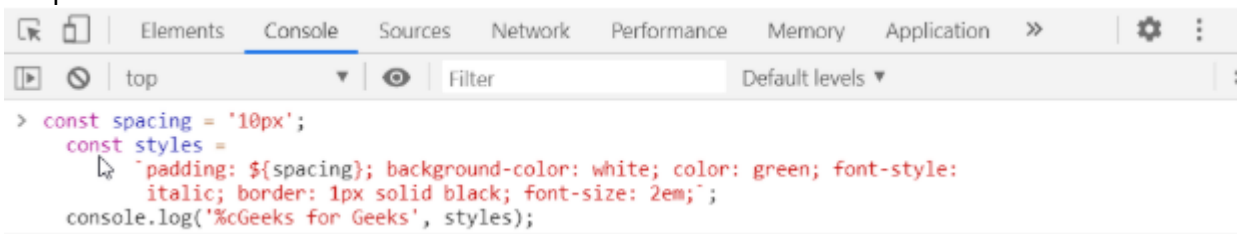
E.X:-
```
console.group('simple');
      console.warn('warning!');
      console.error('error here');
      console.log('vivi vini vici');
   console.groupEnd('simple');
   console.log('new section');
```
output:



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

E.X:-
```
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);
```
output:



2. Write the difference between var, let and const with code examples?

Var

- Before the advent of ES6, var declarations ruled. There are issues associated with variables declared with var, though. That is why it was necessary for new ways to declare variables to emerge. First, let's get to understand var more before we discuss those issues.
- 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.

```
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.

```
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

- let is now preferred for variable declaration. It's no surprise as it comes as an improvement to var declarations
- let is block scoped
- 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 .
- 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.

Const

- Variables declared with the const maintain constant values. const declarations share some similarities with let declarations.
- Like let declarations, const declarations can only be accessed within the block they were declared.

```
const greeting = {
    message: "say Hi",
    times: 4
}
```

- const cannot be updated or re-declared

```
const greeting = "say Hi";
greeting = "say Hello instead";// error: Assignment to constant
```

3.Write a brief note on available datatypes in javascript?

- In Javascript, there are five basic, or primitive, types of data.
- The five most basic types of data are
    I. Strings
    II. Numbers
    III. Booleans
    IV. Undefined
    V. Null

Strings

- A string is a collection of alphanumeric characters. I start a string by typing double quotes, single quotes, or the backtick character.
- Double quote and single quote behave identically, and the backtick character comes with some extra functionality.
- Inside of the quote, I can put whatever I want. “The dog went on a walk to 7-11”. You’ll notice that all remains the exact same color because it’s within the quote marks.
- If I were to copy this and place it outside quote marks, you’ll notice that the coloring in my editor is very differently.
- That’s because this default blue color in my current scheme is trying to look for a variable with this name. It’s not finding it. This doesn’t make any sense to it. It doesn’t know it’s trying to do 7-11 and some other weird things in here.
- But when we wrap a value in a string, in quotes, it creates a string at which point we can put any valid alphanumeric character we would like inside the string.
- We can add a whole bunch of weird special characters, letters, and numbers. That’s all valid inside of a string.
- The only thing that’s not valid is another quote mark because that ends the string and now I have a quote mark sitting out here all by itself without having a match.

```
var a = "Let's have a cup of coffee."; // single quote inside
double quotes
var b = 'He said "Hello" and left.';  // double quotes inside
single quotes
var c = 'We\'ll never give up.';     // escaping single quote
with backslash
```

Numbers

- Numbers are as straightforward as they sound.
- Numbers are for numbers. I can’t put a letter on here. It’s no longer a number, and the coloring gets funky. It’s no longer green. But I can make this number be as long as I want it.
- I cannot add a comma, but I can add a decimal point. So numbers are any integer or decimal number created in the language, and they’re used for money, age, etc. the same kinds of things that we use money for here in real life.

```
var a = 25;          // integer
var b = 80.5;        // floating-point number
var c = 4.25e+6;     // exponential notation, same as 4.25e6
or 4250000
var d = 4.25e-6;     // exponential notation, same as
0.00000425
```

Booleans

- Booleans have two values. True and false. When we create a boolean, we're simply saying it's true or it's false.
- It's like that on/off switch example that we talked about.
- It's all there is to them right now. We're gonna talk into other aspects of playing into these variables in another time.

```
var isReading = true;   // yes, I'm reading
var isSleeping = false; // no, I'm not sleeping
```

- Boolean values also come as a result of comparisons in a program.

```
var a = 2, b = 5, c = 10;

alert(b > a) // Output: true
alert(b > c) // Output: false
```

Undefined

- In JavaScript, undefined is a type.
- It means a variable declared, but no value has been assigned a value.

For example,

```
var demo;
alert(demo); //shows undefined
alert(typeof demo); //shows undefined
```

NULL

- In JavaScript, null is an object.
- Whereas, null in JavaScript is an assignment value. You can assign it to a variable

For example,

```
var demo = null;
alert(demo); //shows null
alert(typeof demo); //shows object
```