**1. What is Array ?**

Arrays in JavaScript are the data type used to store a list of values. JavaScript array objects can be stored in variables and dealt with in the same way you deal with any other data type. The difference is that we can access each value inside the list individually, and perform various activities, such as looping over it.

Array is a collection of both Homogeneous and Hetrogeneous elements.

JavaScript array is an object that represents a collection of similar type of elements.

In Javascript array is declared in 3 ways.

1.var arrayname = [ ] By array literal

2.var arrayname1 = new Array(5); By using an Array constructor (using new keyword)

3.By creating instance of Array directly (using new keyword)

**2. How to Access Elements from Array ?**

These are the following ways to Access Elements in an Array:

1. Using Square Bracket Notation

2. Using forEach Loop

3. Using map() Method

4. Using find() Method

5. Using Destructuring Assignment

6. Using filter() Method

**1. Using Square Bracket Notation**

We can access elements in an array by using their index, where the index starts from 0 for the first element. We can access using the bracket notation.

**const a = [10, 20, 30, 40, 50];**

**const v = a[3];**

**console.log(v);**

**Output:**

**40**

**2. Using forEach Loop**

In this approach, we will use a loop for accessing the element. We can use for, forEach, or for…of methods for looping. The forEach() method allows you to iterate over all elements in the array and perform an operation on each element.

**const a = [100, 200, 300, 400, 500];**

**a.forEach((e, i) => {**

**console.log(e);**

**});**

**Output:**

**100**

**200**

**300**

**400**

**500**

**3. Using map() Method**

The Javascript [map()](https://www.geeksforgeeks.org/javascript-array-map-method/) method in JavaScript creates an array by calling a specific function on each element present in the parent array.

**const a = [10, 20, 30, 40, 50];**

**const r = a.map((e, i) => {**

**console.log(e);**

**});**

**Output:**

**10**

**20**

**30**

**40**

**50**

**4. Using find() Method**

The [find()](https://www.geeksforgeeks.org/javascript-array-find-method/). method returns the first element in the array that satisfies a provided testing function.

**const a = [10, 20, 30, 40, 50];**

**const r = a.find((e) => e > 30);**

**console.log(r);**

**Output**

**40**

**5. Using Destructuring Assignment**

[Destructuring Assignment](https://www.geeksforgeeks.org/destructuring-assignment-in-javascript/) is a JavaScript expression that allows us to unpack values from arrays, or properties from objects, into distinct variables data can be extracted from arrays, objects, and nested objects and assigned to variables.

let [FN, , TN] = ["alpha", "beta", "gamma", "delta"];

console.log(FN);

console.log(TN);

**Output**

alpha

gamma

### 

### **6. Using filter() Method**

The [filter() method](https://www.geeksforgeeks.org/javascript-array-filter-method/) in JavaScript creates a new array containing elements that pass a specified condition. It iterates through each element of the array, executing the condition for each element and including elements that return true in the filtered array.

**const a = [1, 2, 3, 4, 5];**

**const res = a.filter(e => e > 2);**

**console.log(res);**

**Output**

**[ 3, 4, 5 ]**

**3. Why Index starts from 0 ?**

We always assign a variable name to an array. So, that variable name references to a memory location where array elements are stored.

The reason why index start from zero is that index is used as an offset.

Suppose we have an array

Arr = [1,2,3,4,5]

The reason why index start from zero is that index is used as an offset. Suppose we have an array Arr = [1,2,3,4,5] Arr[0] actually means that first element is 0 element away from the memory location where the arr points as elements in an array are stored in a contiguous manner.

* **Why Array Index starts from 0**

**Reason 1 :**

Consider int arr[100]. The answer lies in the fact how the compiler interprets arr[i] ( 0<=i<100).

arr[i] is interpreted as \*(arr + i).

Now, arr is the address of the array or address of 0th index element of the array.

So, address of next element in the array is arr + 1 (because elements in the array are stored in consecutive memory locations), further address of next location is arr + 2 and so on.

Going with the above arguments, arr + i mean the address at i distance away from the starting element of the array.

Therefore, going by this definition, i will be zero for the starting element of the array because the starting element is at 0 distance away from the starting element of the array.

To fit this definition of arr[i], indexing of array starts from 0.

#include <iostream>

using namespace std;

int main()

{

int arr[] = { 1, 2, 3, 4 };

// Below two statements mean same thing

cout << \*(arr + 1) << " ";

cout << arr[1] << " ";

return 0;

}

**Output**

2 2

The conclusion is, we need random access in the array. To provide random access, compilers use pointer arithmetic to reach i-th element.

**Reason 2 :**

Modern languages, especially **C++** use row-majorordering for storing two-dimensional arrays.

Let us assume a 2D array and write a row-major formula with two different approaches:

1. array indices starting from 1
2. array indices starting from 0

let the 2D array be arr[m][n] of type int

let &arr be “address”

case 1 ( array indices start from 1 ) :

&( arr[i][j] ) = address + [ ( i-1 )\*n + ( j-1 ) ]\*( sizeof(int) ) ] so here we are performing 6 operations

case 2 ( array indices start from 0 ) :

&( arr[i][j] ) = address + [ ( i )\*n + ( j ) ]\*( sizeof(int) ) ] and here we are performing only 4 operations

So we see here that we are performing 2 less operations when we store 2D arrays and obtaining an element’s address.

This looks like it doesn’t make sense but it does! While handling with huge size data this may improved performance and speed.

**case 1** may look user-friendly but **case 2** is more efficient. That’s why most languages like **C++, PYTHON, JAVA** use arrays starting with index 0 and rarely languages like **Lua** arrays starting with index 1.

**4. How to apply styles for console**

**console:**The console is a very useful part of every development process. We use it to log items for various reasons, to view data, to keep certain data for later use, and so on.

As a result, it is only right that we find a way to give it an appealing look and feel, given how constantly we interact with it directly and indirectly.

We are going to know how to apply styles for console

**Format Specifier:** Format specifiers contain the % symbol, followed by a letter that specifies the kind of formatting that should apply to the value.

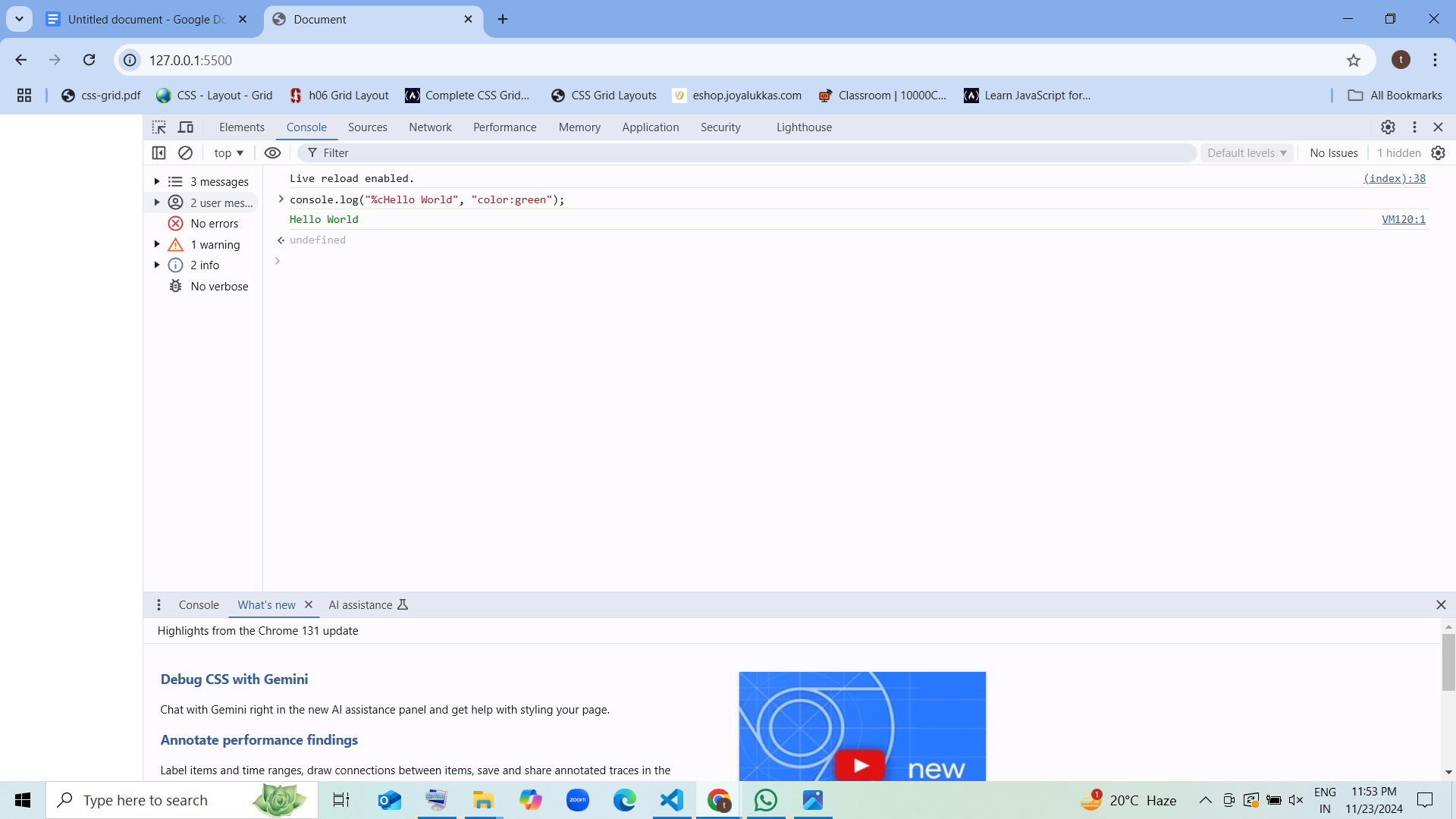
We can pass in properties as the second parameter to effect changes on the String (console message) in respective order or to insert values into the output String.

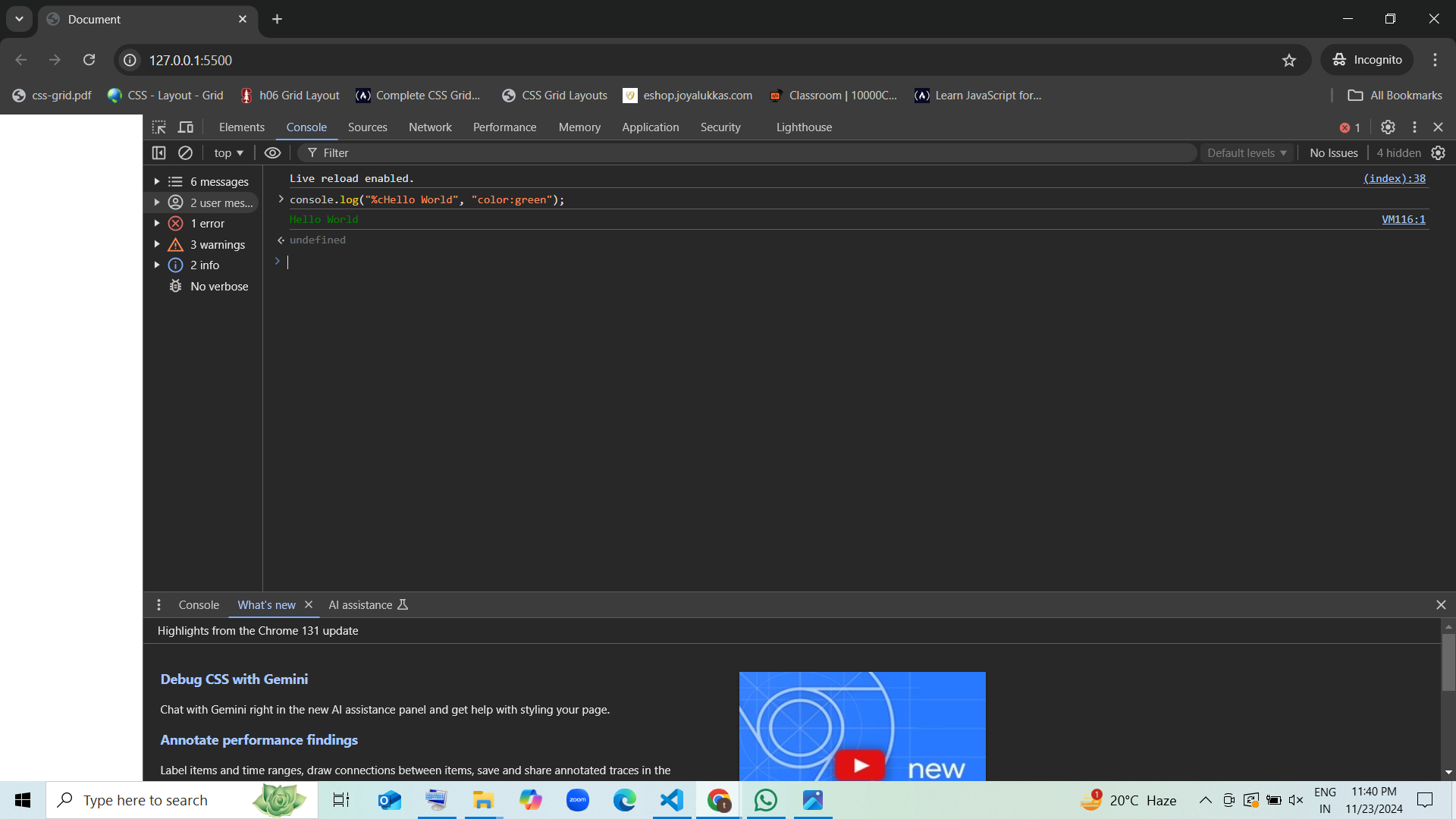
| **Specifier** | **Output** |
| --- | --- |
| %s | Formats the value as a string |
| %i or %d | Formats the value as an integer |
| %f | Formats the value as a floating point value |
| %o | Formats the value as an expandable DOM element. As seen in the Elements panel |
| %O | Formats the value as an expandable JavaScript object |
| %c | Applies CSS style rules to the output string as specified by the second parameter |

**Syntax:**To add CSS styling to the console output, we use the CSS format specifier %c.

Then we start the console message, which is usually a String with the specifier followed by the message we intend to log, and, finally, the styles we want to apply to the message:

console.log("%cHello World", "color:green");





* we have used the %c format specifier to declare that we’ll be applying CSS styles to the console output,
* we have written a String we’d like to print to the console, and finally we have defined the CSS effect we’d like to apply to the String.
* If we check the console now, we should get the String printed with green color.

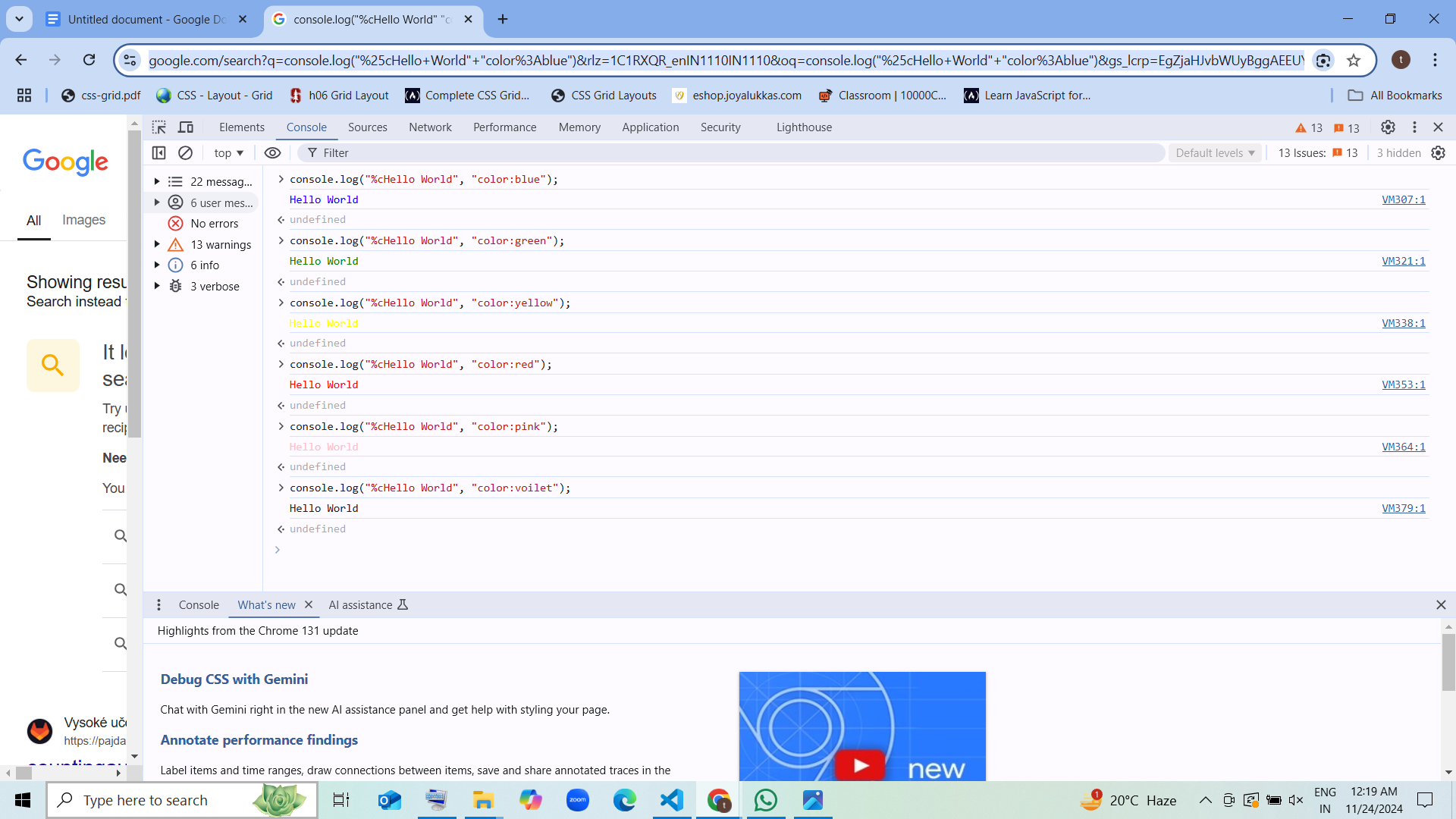
console.log("%cHello World", "color:green");

console.log("%cHello World", "color:blue");

console.log("%cHello World", "color:yellow");

console.log("%cHello World", "color:red");

This will print all the texts we have written to the console in their specified color styles like this:



## **Changing Console Output Fonts**

we changed up the text colors; here, let’s see how we can change the fonts.

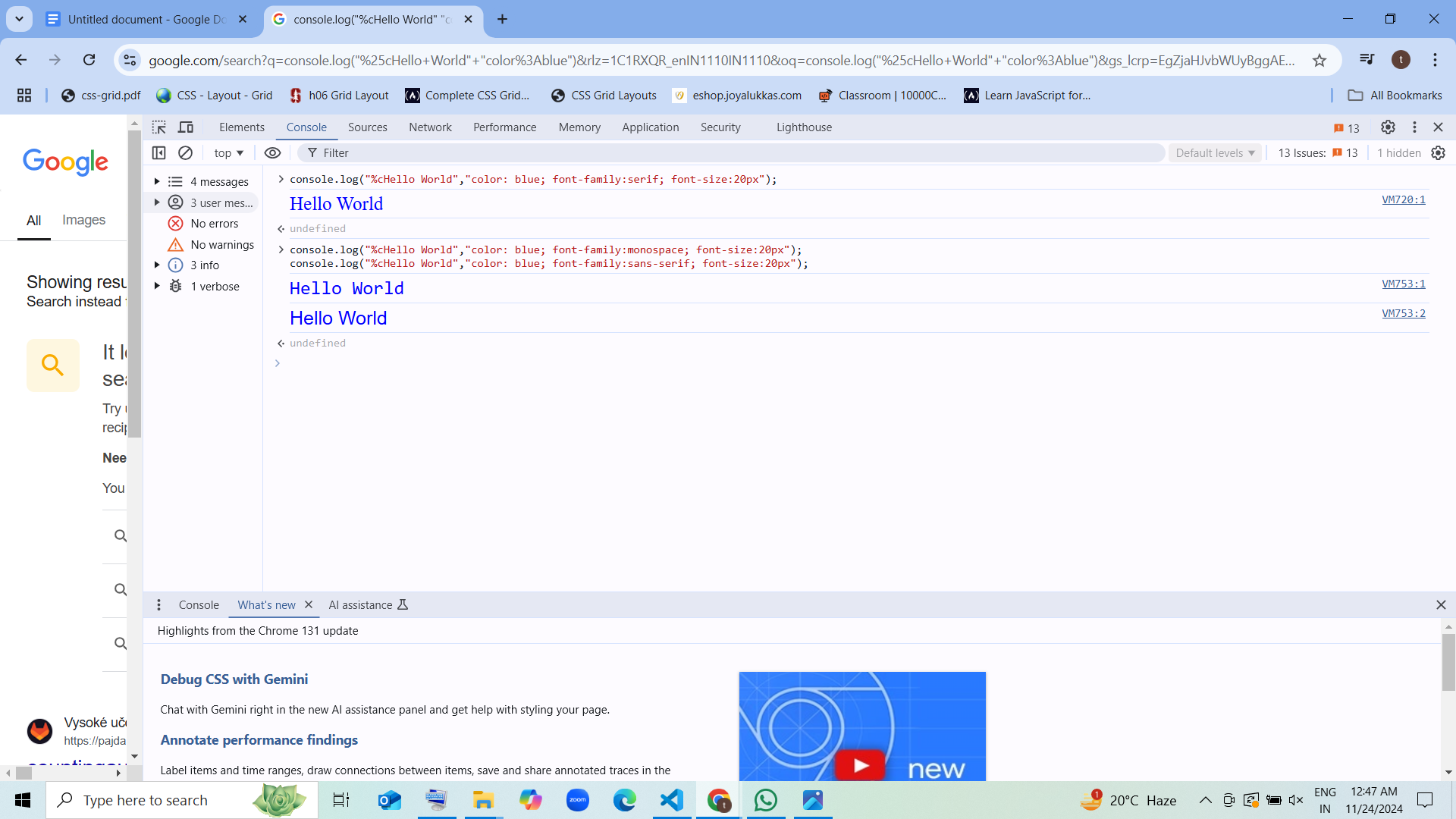
console.log("%cHello World","color: blue; font-size: 20px");

console.log("%cHello World","color: blue; font-family:serif; font-size: 20px");

console.log("%cHello World","color: blue; font-family:monospace; font-size: 20px");

console.log("%cHello World","color: blue; font-family:sans-serif; font-size: 20px");

we paste this code in the console and run it, we will get this output:



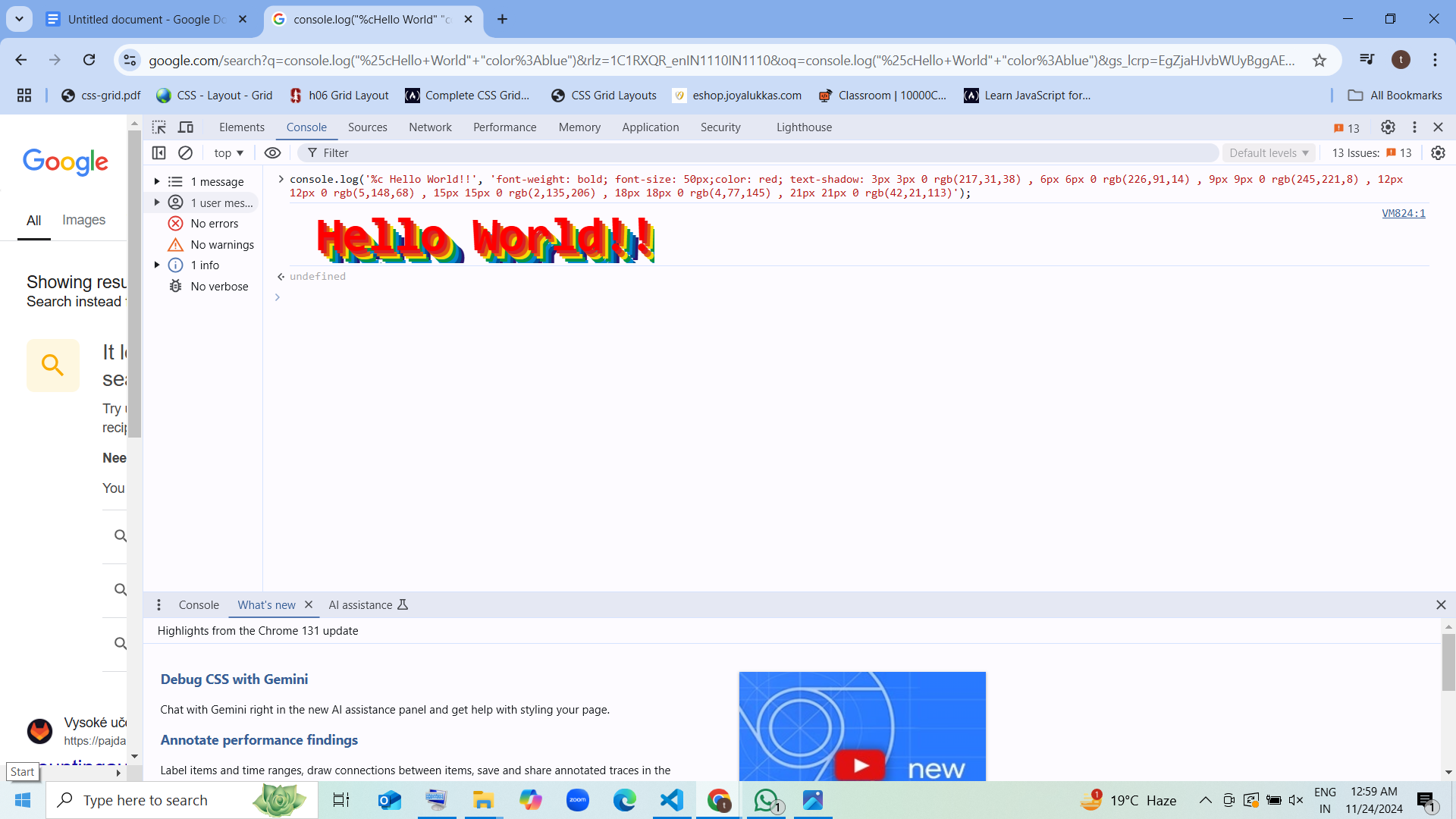
Now we have four lines of text with same color but different font styles. This goes to show that we can apply as much style as we want to our console output to produce any desired effect.

## **Extended Console Styling**

Here shows you how to make a rainbow-like text in the console by combining colors and using CSS styles like font-weight, font-size, text-shadow, colors, etc. to produce the code:

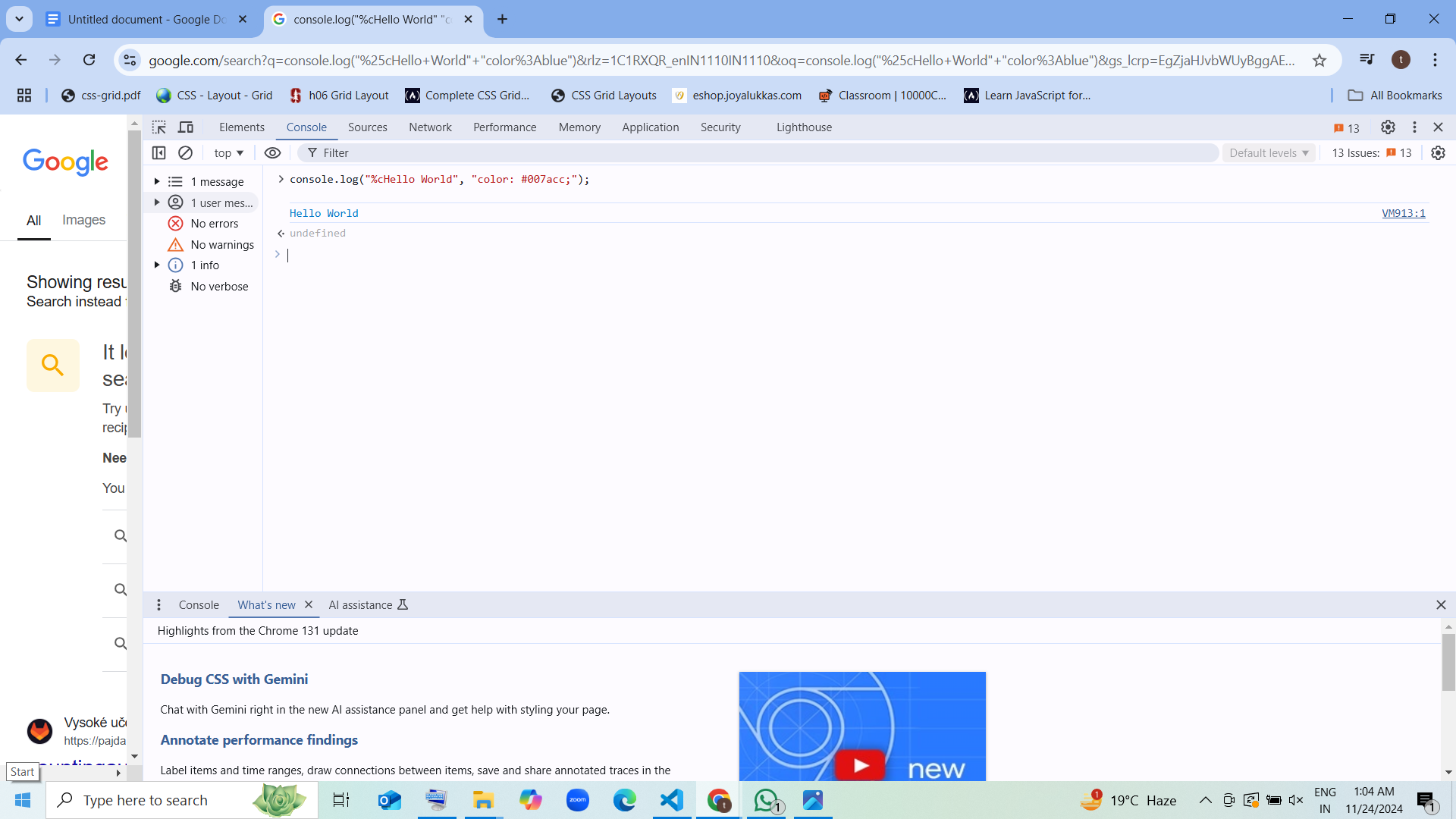
console.log('%c Hello world!!', 'font-weight: bold; font-size: 50px;color: red; text-shadow: 3px 3px 0 rgb(217,31,38) , 6px 6px 0 rgb(226,91,14) , 9px 9px 0 rgb(245,221,8) , 12px 12px 0 rgb(5,148,68) , 15px 15px 0 rgb(2,135,206) , 18px 18px 0 rgb(4,77,145) , 21px 21px 0 rgb(42,21,113)');

This way, when we check the code in the console, we should be able to get this output:



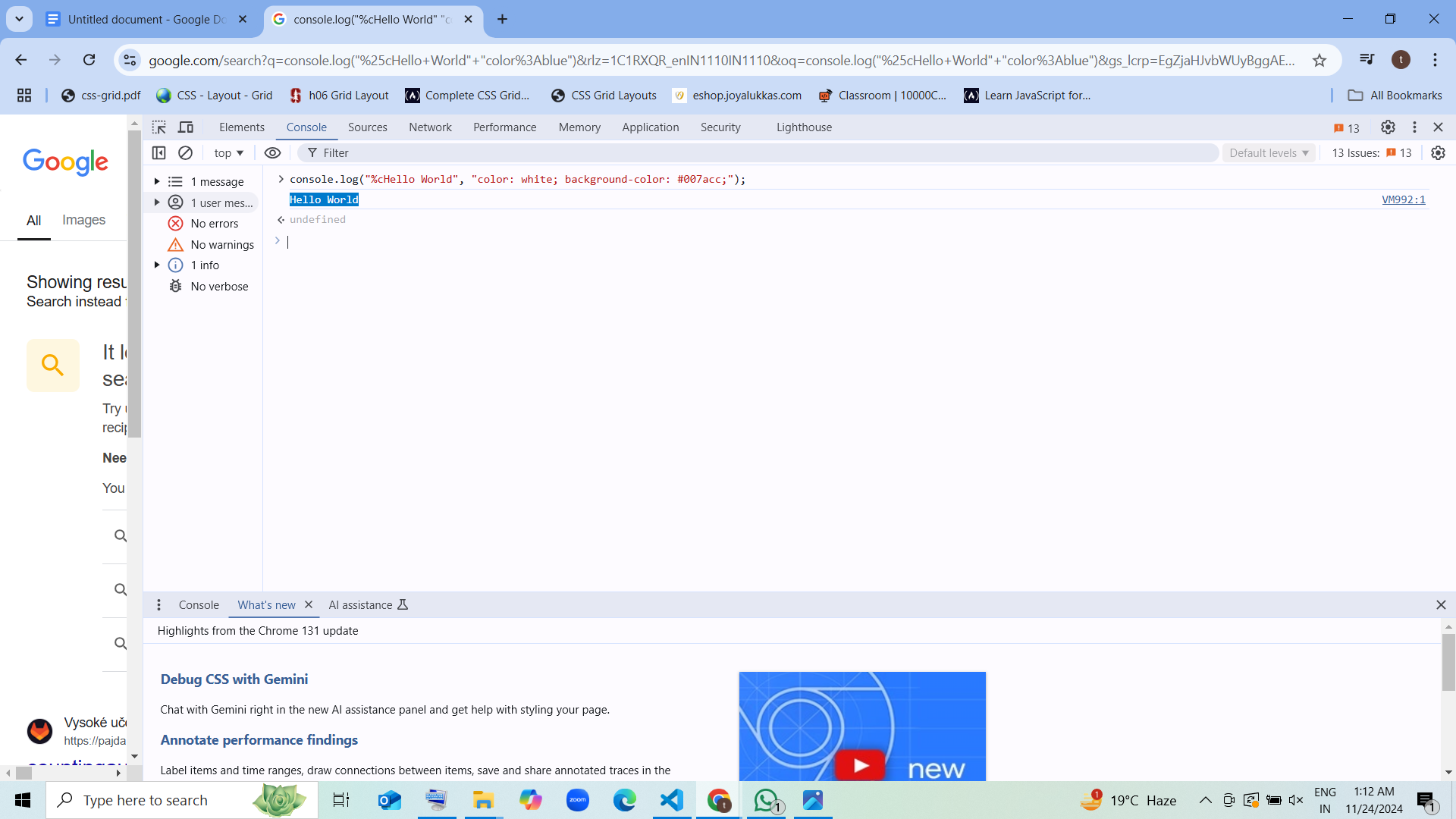
1.Simple Console Log Shortcut for JS Objects with the defined color i.e #007acc

**console.log("%cHello World", "color: #007acc;");**

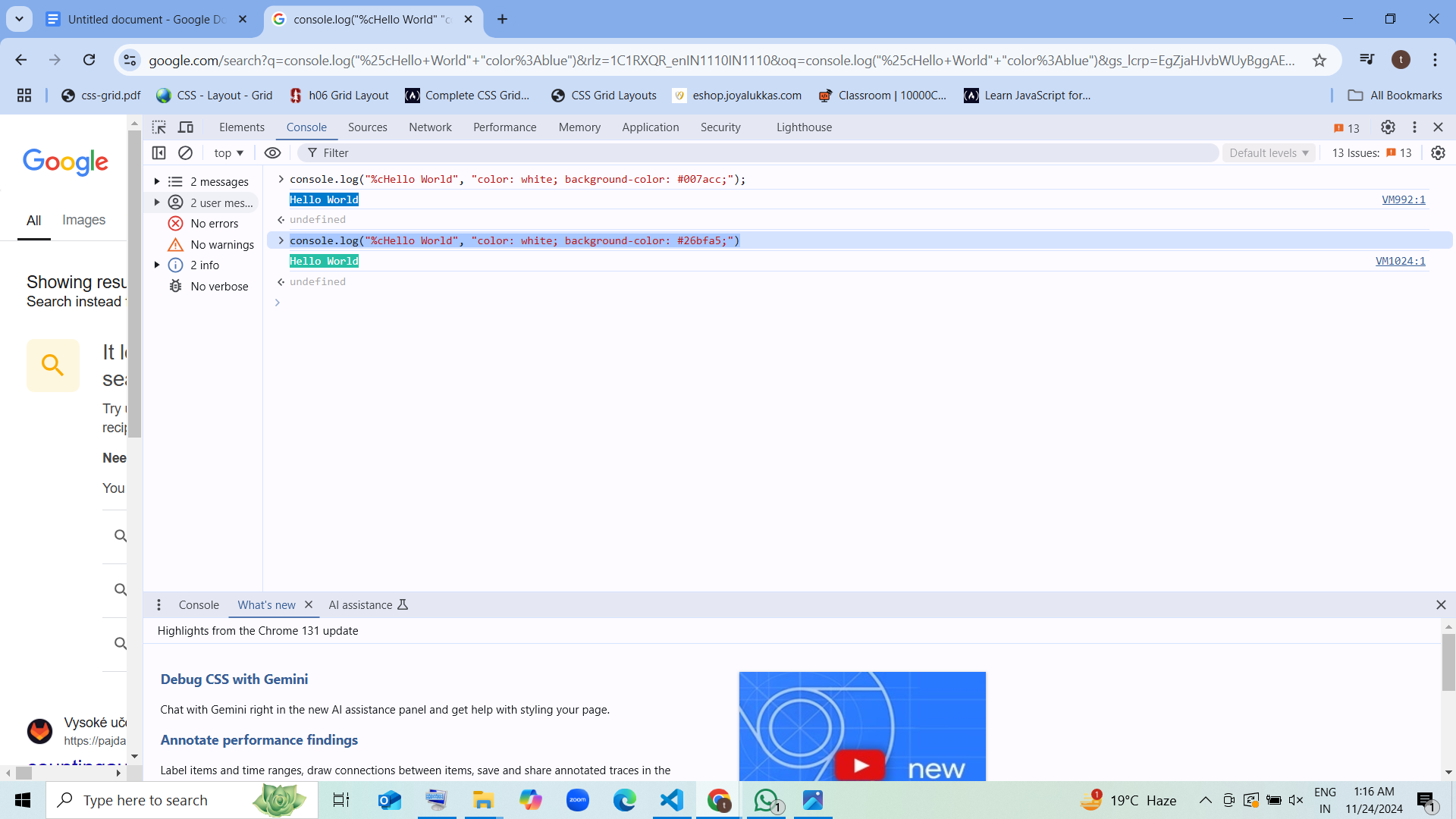
****

**2.Simple Console Log Shortcut for JS Objects - Blue Background**

**console.log("%cHello World", "color: white; background-color: #007acc;");**

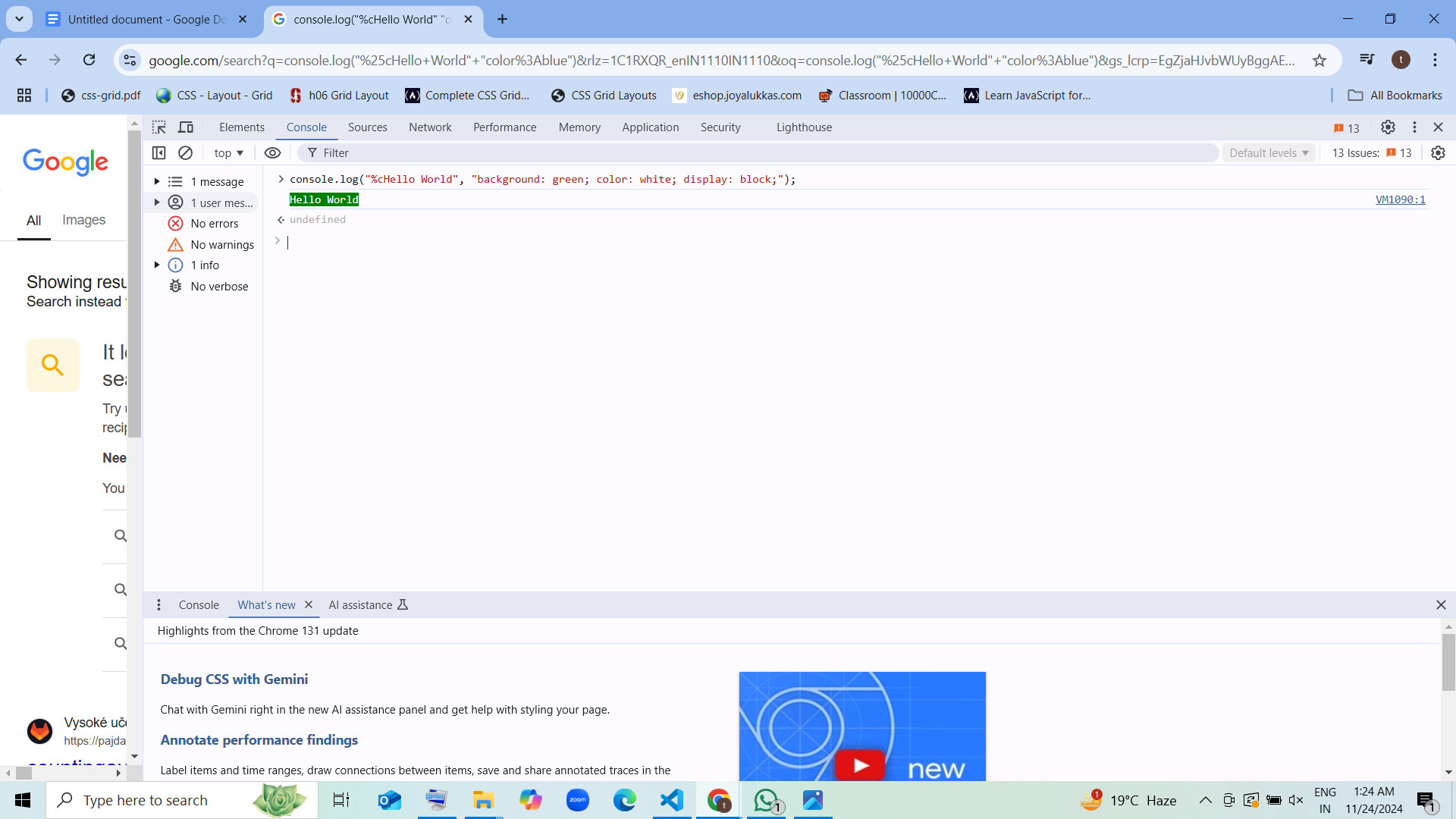
****

**console.log("%cHello World", "color: white; background-color: #26bfa5;")**

****

**3.Console logs -> Hello ${TM\_FILENAME} line:${TM\_LINE\_NUMBER} on Green Background**

**console.log("%cHello World", "background: green; color: white; display: block;");**

****

**4.we can also add images and GIF**

**Syntax**

**console.log("%c ", "font-size: 1px; padding: 166.5px 250px; background-size: 500px 333px; background: no-repeat url(https://www.capscode.in/static/media/cap.0d0af8f0.png);");**

**console.log("%c ", "font-size: 1px; padding: 215px 385px; background-size: 770px 430px; background: no-repeat url(https://i0.wp.com/i.giphy.com/media/ZVik7pBtu9dNS/giphy-downsized.gif?w=770&amp;ssl=1);");**