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

The console object provides us with several different methods, like :

1. log()
2. error()
3. warn()
4. clear()
5. time() and timeEnd()
6. table()
7. count()
8. group() and groupEnd()
9. custom console logs
10. **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.

// console.log() method

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

1. **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.

// console.error() method

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

1. **console.warn()**

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

// console.warn() method

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

1. **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.

// console.clear() method

console.clear();

1. **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).

// console.time() and console.timeEnd() method

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.

1. **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.

// console.table() method

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

1. **console.count()**

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

// console.count() method

for(let i=0;i<5;i++){

    console.count(i);

}

1. **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.

// console.group() and console.groupEnd() method

console.group('simple');

   console.warn('warning!');

   console.error('error here');

   console.log('vivi vini vici');

console.groupEnd('simple');

console.log('new section');

1. **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.

// Custom Console log example

  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);

1. **Write the difference between var, let and const with code examples.**
2. **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”.**

{  
 var a=10;  
 console.log(a);  
} //block 1{  
 a++;  
 console.log(a);  
} //block 2

1. **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** a=10;  
 console.log(a);  
} **//block 1**{  
 a++;  
 console.log(a);  
} **//block 2**

1. **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.**
2. {  
    **const** PI=3.14;  
    console.log(PI);  
   } **//block 1**{  
    console.log(PI);  
   } **//block 2**
3. **Write a brief intro on available data types in Javascript.**

In Javascript, there are five basic, or primitive, types of data. The five most basic types of data are strings, numbers, booleans, undefined, and null. We refer to these as primitive data types. A single variable can only store a single type of data. That means it’s important for you to learn to store the data correctly.

**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.

#### Numbers

Numbers. Let’s talk about 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.

#### 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.

#### Null and Undefined

As we mentioned in the conceptual videos, null and undefined stand for empty. That means they have no value assigned to them. The difference is undefined exists when we haven’t given a value.