**QUESTION 1:**

Explore and explain the various methods in console function. Ex. console.log(), console.warn().

**SOLUTION:-**

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.

1. **console.warn()**  
   Used to log warning message to the console. By default the warning message will be highlighted with yellow color.
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.

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.
2. **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).

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.

1. **console.count()**

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

1. **console.group() and groupEnd()**  
   This 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.
2. **console.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 .

**QUESTION 2:**

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

**SOLUTION:**

* **Based on Scope:-**
* **var** declarations are globally scoped or function scoped   
  **Example var:-**

var greeter = "hey hi";

function newFunction()

{  
 var hello = "hello”;

}

*Here, greeter is globally scoped and hello is function scoped.*

* **let** and **const** are block scoped.  
  **Example let:-**

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

*Here,* ***let*** *variables are block scoped. So, using hello outside its block returns error.  
Similar to let,* ***Const*** *declarations are also block scoped.*

* **Based on Update and Re-declaration**
* **var** variables can be updated and re-declared within its scope;   
  **Example var:-**

var greeter = "hey hi";  
var greeter = "say Hello instead";

AND

var greeter = "hey hi";  
greeter = "say Hello instead";

* **let** variables can be updated but not re-declared; however if the same variable is defined in different scopes there will be no error because both instances are treated as different variables since they have different scopes.  
  **Example let:-**

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

let greeting = "say Hi";  
greeting = "say Hello instead";

* **const** variables can neither be updated nor re-declared.  
  **Example const:-**

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

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

* **Based on Hoisting:-**Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution.

**Example:**

var greeter;  
console.log(greeter);   
greeter = "say hello";

console.log (greeter);  
var greeter = "say hello";

**Interpreted as**

* All the three var, let and char are hoisted to the top of their scope.
* **var** variables are initialized with a value of undefined.
* **let** and **const** keyword are not initialized. So, Reference error is generated if let or const variable is used before declaration.
* **Based on initialization during declaration.**
* **var** and **let** can be declared without being initialized,
* **const** must be initialized during declaration.

**QUESTION 3**

Write a brief intro on available data types in JavaScript.

**SOLUTION:-**

JavaScript provides different data types to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type
2. Non-primitive (reference) data type

**JavaScript Primitive Data Types:-**  
There are five types of primitive data types in JavaScript.

1. **String:** represent textual data (i.e. sequences of characters). Strings are created using single or double quotes surrounding one or more characters.

**Example:**

var a = ‘Hello’;  
var b = “Hello”;

1. **Number:** represent positive or negative numbers with or without decimal place, or numbers written using exponential notation.

var a = 25; //integer  
**var b = 80.**5; //floating-point number  
**var c = 4.**25e + 6; //exponential notation, same as 4250000  
**var d = 4.**25e – 6; // exponential notation, same as 0.00000425

**Example:**

1. **Boolean:** represents only two values true or false. It is typically used to store values like yes (true) or no (false), on (true) or off (false).  
   **Example:** var a = 2, b = 5, c = 10;

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

1. **Undefined:** represents undefined value. If a variable has been declared, but has not been assigned a value, has the value undefined.

**Example:** var a ;

var b = “Hello World!”

alert (a) //Output: undefined  
alert (b) //Output: Hello World!

1. **Null:** represents null i.e. no value at all. It is not equivalent to an empty string ("") or 0, it is simply nothing.

**Example:** var a = null ;

alert (a); //Output: null

**JavaScript Non-Primitive Data Types:-**

1. **Object:** represents instance through which we can access members.

Example: <script>

Emp = { id:102,name:"Shyam Kumar",salary:40000}

document.write (emp.id+" "+emp.name+" "+emp.salary);

</script>

1. **Array:** represents group of similar values.

**Example:** var colors = ["Red", "Yellow", "Green", "Orange"];  
 var cities = ["London", "Paris", "New York"];

alert (colors[0]); // Output: Red  
alert (cities[2]); // Output: New York

1. **Function:** The function is callable object that executes a block of code. Since functions are objects, so it is possible to assign them to variables.

**Example:** var greeting = function()  
 {

return "Hello World!";

}

alert(typeof greeting) // Output: function

alert(greeting()); // Output: Hello World!