**Q.1 Explore and Explain Various Methods in Console Function.**

* log()
* error()
* warn()
* clear()
* time() and timeEnd()
* table()
* count()
* group() and groupEnd()
* custom console logs

**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.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.warn()**

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

**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.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.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.count()**

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

**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.

**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.

**Q.2: Write the difference between var, let and const with code examples.**

**Var:**

The JavaScript variables statement is used to declare a variable and, optionally, we can initialize the value of that variable.

Example: var a =10;

* Variable declarations are processed before the execution of the code.
* The scope of a JavaScript variable declared with var is its current execution context.
* The scope of a JavaScript variable declared outside the function is global.

Consider the following code snippet.

function nodeSimplified(){

var a =10;

console.log(a); // output 10

if(true){

var a=20;

console.log(a); // output 20

}

console.log(a); // output 20

}

In the above code, you can find, when the variable is updated inside the if loop, that the value of variable "a" updated 20 globally, hence outside the if loop the value persists. It is similar to the Global variable present in other languages. But, be sure to use this functionality with great care because there is the possibility of overriding an existing value.

**Let:**

The **let** statement declares a local variable in a block scope. It is similar to **var**,inthat we can optionally initialize the variable.

Example: let a =10;

* The let statement allows you to create a variable with the scope limited to the block on which it is used.
* It is similar to the variable we declare in other languages like Java, .NET, etc.

Consider the following code snippet.

function nodeSimplified(){

let a =10;

console.log(a); // output 10

if(true){

let a=20;

console.log(a); // output 20

}

console.log(a); // output 10

}

It is almost the same behavior we see in most language.

function nodeSimplified(){

let a =10;

let a =20; //throws syntax error

console.log(a);

}

**Error Message: Uncaught SyntaxError: Identifier 'a' has already been declared.**

However, with var, it works fine.

function nodeSimplified(){

var a =10;

var a =20;

console.log(a); //output 20

}

The scope will be well maintained with a let statement and when using an inner function the let statement makes your code clean and clear.

I hope the above examples will help you better understand the var and let commands and if you have any queries please write me in the comment section.

## ****Const:****

const statement values can be assigned once and they cannot be reassigned. The scope of const statement works similar to let statements.

Example: const a =10;

function nodeSimplified(){

const MY\_VARIABLE =10;

console.log(MY\_VARIABLE); //output 10

}

As per usual, naming standards dictated that we declare the const variable in capital letters. **const a =10**will work the same way as the code given above. Naming standards should be followed to maintain the code for the long run.

Consider the following code snippet.

function nodeSimplified(){

const MY\_VARIABLE =10;

console.log(MY\_VARIABLE); //output 10

MY\_VARIABLE =20; //throws type error

console.log(MY\_VARIABLE);

}

**Error Message : Uncaught TypeError: Assignment to constant variable.**

The code will throw an error when we try to reassign the existing const variable.

**Q.3: write a brief intro on available Data Types in javascript.**

## Data Types in JavaScript

Data types basically specify what kind of data can be stored and manipulated within a program.

There are six basic data types in JavaScript which can be divided into three main categories: primitive (or primary), composite (or reference), and special data types. String, Number, and Boolean are primitive data types. Object, Array, and Function (which are all types of objects) are composite data types. Whereas Undefined and Null are special data types.

Primitive data types can hold only one value at a time, whereas composite data types can hold collections of values and more complex entities. Let's discuss each one of them in detail.

## The String Data Type

The string data type is used to represent textual data (i.e. sequences of characters). Strings are created using single or double quotes surrounding one or more characters, as shown below:

#### Example

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=javascript&file=string-data-type)

var a = 'Hi there!'; // using single quotes

var b = "Hi there!"; // using double quotes

You can include quotes inside the string as long as they don't match the enclosing quotes.

#### Example

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=javascript&file=include-quotes-inside-the-string)

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

You will learn more about the strings in [JavaScript strings](https://www.tutorialrepublic.com/javascript-tutorial/javascript-strings.php) chapter.

## The Number Data Type

The number data type is used to represent positive or negative numbers with or without decimal place, or numbers written using exponential notation e.g. 1.5e-4 (equivalent to 1.5x10-4).

#### Example

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=javascript&file=number-data-type)

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

The Number data type also includes some special values which are: Infinity, -Infinity and NaN. Infinity represents the mathematical Infinity ∞, which is greater than any number. Infinity is the result of dividing a nonzero number by 0, as demonstrated below:

#### Example

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=javascript&file=infinity)

alert(16 / 0); // Output: Infinity

alert(-16 / 0); // Output: -Infinity

alert(16 / -0); // Output: -Infinity

While NaN represents a special Not-a-Number value. It is a result of an invalid or an undefined mathematical operation, like taking the square root of -1 or dividing 0 by 0, etc.

#### Example

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=javascript&file=not-a-number)

alert("Some text" / 2); // Output: NaN

alert("Some text" / 2 + 10); // Output: NaN

alert(Math.sqrt(-1)); // Output: NaN

You will learn more about the numbers in [JavaScript numbers](https://www.tutorialrepublic.com/javascript-tutorial/javascript-numbers.php) chapter.

## The Boolean Data Type

The Boolean data type can hold only two values: true or false. It is typically used to store values like yes (true) or no (false), on (true) or off (false), etc. as demonstrated below:

#### Example

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=javascript&file=boolean-data-type)

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. The following example compares two variables and shows the result in an alert dialog box:

#### Example

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=javascript&file=comparisons)

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

alert(b > a) // Output: true

alert(b > c) // Output: false

You will learn more about the comparisons in [JavaScript if/else](https://www.tutorialrepublic.com/javascript-tutorial/javascript-if-else-statements.php) chapter.

## The Undefined Data Type

The undefined data type can only have one value-the special value undefined. If a variable has been declared, but has not been assigned a value, has the value undefined.

#### Example

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=javascript&file=undefined-data-type)

var a;

var b = "Hello World!"

alert(a) // Output: undefined

alert(b) // Output: Hello World!

## The Null Data Type

This is another special data type that can have only one value-the null value. A null value means that there is no value. It is not equivalent to an empty string ("") or 0, it is simply nothing.

A variable can be explicitly emptied of its current contents by assigning it the null value.

#### Example

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=javascript&file=null-data-type)

var a = null;

alert(a); // Output: null

var b = "Hello World!"

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

b = null;

alert(b) // Output: null

## The Object Data Type

The object is a complex data type that allows you to store collections of data.

An object contains properties, defined as a key-value pair. A property key (name) is always a string, but the value can be any data type, like strings, numbers, booleans, or complex data types like arrays, function and other objects. You'll learn more about objects in upcoming chapters.

The following example will show you the simplest way to create an object in JavaScript.

#### Example

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=javascript&file=object-data-type)

var emptyObject = {};

var person = {"name": "Clark", "surname": "Kent", "age": "36"};

// For better reading

var car = {

"modal": "BMW X3",

"color": "white",

"doors": 5

}

You can omit the quotes around property name if the name is a valid JavaScript name. That means quotes are required around "first-name" but are optional around firstname. So the car object in the above example can also be written as:

#### Example

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=javascript&file=object-properties-names-without-quotes)

var car = {

modal: "BMW X3",

color: "white",

doors: 5

}

You will learn more about the objects in [JavaScript objects](https://www.tutorialrepublic.com/javascript-tutorial/javascript-objects.php) chapter.

## The Array Data Type

An array is a type of object used for storing multiple values in single variable. Each value (also called an element) in an array has a numeric position, known as its index, and it may contain data of any data type-numbers, strings, booleans, functions, objects, and even other arrays. The array index starts from 0, so that the first array element is arr[0] not arr[1].

The simplest way to create an array is by specifying the array elements as a comma-separated list enclosed by square brackets, as shown in the example below:

#### Example

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=javascript&file=array-data-type)

var colors = ["Red", "Yellow", "Green", "Orange"];

var cities = ["London", "Paris", "New York"];

alert(colors[0]); // Output: Red

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

You will learn more about the arrays in [JavaScript arrays](https://www.tutorialrepublic.com/javascript-tutorial/javascript-arrays.php) chapter.

## The Function Data Type

The function is callable object that executes a block of code. Since functions are objects, so it is possible to assign them to variables, as shown in the example below:

#### Example

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=javascript&file=function-data-type)

var greeting = function(){

return "Hello World!";

}

// Check the type of greeting variable

alert(typeof greeting) // Output: function

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

In fact, functions can be used at any place any other value can be used. Functions can be stored in variables, objects, and arrays. Functions can be passed as arguments to other functions, and functions can be returned from functions. Consider the following function:

#### Example

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=javascript&file=function-passed-as-argument-to-other-function)

function createGreeting(name){

return "Hello, " + name;

}

function displayGreeting(greetingFunction, userName){

return greetingFunction(userName);

}

var result = displayGreeting(createGreeting, "Peter");

alert(result); // Output: Hello, Peter

You will learn more about the functions in [JavaScript functions](https://www.tutorialrepublic.com/javascript-tutorial/javascript-functions.php) chapter.

## The typeof Operator

The typeof operator can be used to find out what type of data a variable or operand contains. It can be used with or without parentheses (typeof(x) or typeof x).

The typeof operator is particularly useful in the situations when you need to process the values of different types differently, but you need to be very careful, because it may produce unexpected result in some cases, as demonstrated in the following example:

#### Example

[**Try this code »**](https://www.tutorialrepublic.com/codelab.php?topic=javascript&file=typeof-operator)

// Numbers

typeof 15; // Returns: "number"

typeof 42.7; // Returns: "number"

typeof 2.5e-4; // Returns: "number"

typeof Infinity; // Returns: "number"

typeof NaN; // Returns: "number". Despite being "Not-A-Number"

// Strings

typeof ''; // Returns: "string"

typeof 'hello'; // Returns: "string"

typeof '12'; // Returns: "string". Number within quotes is typeof string

// Booleans

typeof true; // Returns: "boolean"

typeof false; // Returns: "boolean"

// Undefined

typeof undefined; // Returns: "undefined"

typeof undeclaredVariable; // Returns: "undefined"

// Null

typeof Null; // Returns: "object"

// Objects

typeof {name: "John", age: 18}; // Returns: "object"

// Arrays

typeof [1, 2, 4]; // Returns: "object"

// Functions

typeof function(){}; // Returns: "function"

As you can clearly see in the above example when we test the null value using the typeof operator (*line no-22*), it returned "object" instead of "null".

This is a long-standing bug in JavaScript, but since lots of codes on the web written around this behavior, and thus fixing it would create a lot more problem, so idea of fixing this issue was rejected by the committee that design and maintains JavaScript