1.Explore and Explain the various methods of 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');

console.log(1);

console.log([1, 2, 3, 4]); // array inside log

console.log({a:1, b:2, c:3}); // object inside log

**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.

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

**console.warn()**

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

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

**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.

Ex: console.clear();

**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.

Ex: 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');

**console.table()**

This method allows us to generate a table inside a console. The input must be array or an object which will be shown as a table.

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

**console.count()**

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

Ex: for(let i=0;i<5;i++)

{

     console.count(i);

}

**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.

Ex: console.group('simple');

console.warn('warning!');

console.error('error here');

console.log('vivi vini vici');

console.groupEnd('simple');

console.log('new section');

The above code will generate a group simple which will have warning! ,error here and vivi vini vici ; after this a new section will be formed and this new section will have a different indentation.

2.What is the difference between var, let and const with code examples.

i)var and let can change their value and const cannot change its value

ii)var can be accessible anywhere in function but let and const can only be accessible inside the block where they are declared.

iii)const cannot be declared only, you need to initialize it with declaration .

iv)let and const hoist but you canno access them before the actual declaration is evaluated at runtime .So in case of let and const you cannot access them before declaration.

Hoisting a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution.

v)let and const have a block scope while var has a function scope.

vi)const cannot be declared only, we need to initialize it with declaration

Ex:

var x = 14;  
x = 12;  
console.log(x);//12  
12;

from the above code we successfully changed the value of variable from 14 to 12

let x = 14;  
x = 12;  
console.log(x); //12

from the code above we successfully changed the value of variable from 14 to 12

const y = 14;  
y = 12; //Type Error: Assignment to constant variable.  
console.log(y);

From the code above with const when we run the program got the type error that value of const cannot be changed.

function adult6(age) {  
if (age > 18) {  
let status = 'adult';  
}  
console.log(status); //ReferenceError: status is not defined  
}  
adult6(20);

The above example gives us an error that status is not defined. **we declared the status inside if block and want to access outside the if block so status with let is not accessible outside the block.**

var num = 10;  
for (var num = 0; num < 3; num++) {  
console.log(num); //0 1 2  
}  
console.log(num); //3

in the above example outside for loop value of num is 3 so num have the global scope.**if var is defined outside the function it has global scope so it can be accessed at anywhere** so inside the for loop var num is accessible so value of num is changed and when we console.log(num) outside the for loop it is value is changed.

let num = 10;  
for (let num = 0; num < 3; num++) {  
console.log(num); //0 1 2  
}  
console.log(num); //10

In the above example when we declared num inside the for loop with let then num inside the for loop has a completely different scope and num outside the for loop has different scope.

3.Write a brief intro on available data types in JavaScript.

There are 8 basic data types in JavaScript.

1.number for numbers of any kind: integer or floating-point, integers are limited by ±253.

2.bigint is for integer numbers of arbitrary length.

3.string for strings. A string may have zero or more characters, there’s no separate single-character type.

4.boolean for true/false.

5.null for unknown values – a standalone type that has a single value null.

6.undefined for unassigned values – a standalone type that has a single value undefined.

7.object for more complex data structures.

8.unique for unique identifiers.

Ex:

**Number**

let n = 123;

n = 12.345;

Besides regular numbers, there are so-called “special numeric values” which also belong to this data type: Infinity,-Infinity & NaN.

i)Infinity represents the mathematical Infinity ∞. It is a special value that’s greater than any number.

We can get it as a result of division by zero:

alert( 1 / 0 ); // Infinity

Or just reference it directly:

alert( Infinity ); // Infinity

ii)NaN represents a computational error. It is a result of an incorrect or an undefined mathematical operation, for instance:

alert( "not a number" / 2 ); // NaN, such division is erroneous

NaN is sticky. Any further operation on NaN returns NaN:

alert( "not a number" / 2 + 5 ); // NaN

So, if there’s a NaN somewhere in a mathematical expression, it propagates to the whole result.

**BigInt**

BigInt type was recently added to the language to represent integers of arbitrary length.

A BigInt value is created by appending n to the end of an integer:

// the "n" at the end means it's a BigInt

const bigInt = 1234567890123456789012345678901234567890n;

**String**

A string in JavaScript must be surrounded by quotes.

let str = "Hello";

let str2 = 'Single quotes are ok too';

let phrase = `can embed another ${str}`;

In JavaScript, there are 3 types of quotes.

1. Double quotes: ”Hello”.
2. Single quotes: ’Hello’.
3. Backticks: `Hello`.

**Boolean**

The boolean type has only two values: true an false

This type is commonly used to store yes/no values: true means “yes, correct”, and false means “no, incorrect”.

For instance:

let nameFieldChecked = true; // yes, name field is checked

let ageFieldChecked = false; // no, age field is not checked

**Null**

The special null value does not belong to any of the types described above.

It forms a separate type of its own which contains only the null value:

let age = null;

In JavaScript, null is not a “reference to a non-existing object” or a “null pointer” like in some other languages.

It’s just a special value which represents “nothing”, “empty” or “value unknown”.

The code above states that age is unknown.

**Undefined**

The special value undefined also stands apart. It makes a type of its own, just like null.

The meaning of undefined is “value is not assigned”.

If a variable is declared, but not assigned, then its value is undefined:

let age;

alert(age); // shows "undefined"

**Object and Symbol**

The object type is special.

All other types are called “primitive” because their values can contain only a single thing (be it a string or a number or whatever). In contrast, objects are used to store collections of data and more complex entities.

The symbol type is used to create unique identifiers for objects.