Q1.What is typeof operator give example and explain  
 => In JavaScript, the typeof operator is used to determine the data type of a variable or an expression. It returns a string representing the data type of the operand.

The syntax for using typeof is:

let x = 5;

let y = "Hello";

let z = true;

console.log(typeof x); // Output: "number"

console.log(typeof y); // Output: "string"

console.log(typeof z); // Output: "boolean"

In this example:

typeof x returns "number" because x is assigned a numerical value.

typeof y returns "string" because y is assigned a string value.

typeof z returns "boolean" because z is assigned a boolean value.

Q2. What is equality in java script ?  
=> Equality refers to comparing two values to see if they are equal or identical. There are two main equality operators: the double equals (==) operator and the triple equals (===) operator.

1.Double Equals (==):

The double equals operator performs type coercion before comparing two values. Type coercion means that JavaScript will attempt to convert the values to the same type before comparing them.

If the types of the two values are different, JavaScript will attempt to convert one or both of the values to make them the same type.

After the type coercion, it performs a loose equality comparison. This means that it only checks if the values are equal, not necessarily of the same type.

For example:

console.log(5 == '5'); // Output: true

console.log(0 == false); // Output: true

console.log('' == false); // Output: true

2.Triple Equals (===):

The triple equals operator does not perform type coercion. It checks for strict equality, meaning both the values and their types must be the same for the comparison to return true.

It does not attempt to convert the values to the same type before comparing them.

For example:  
console.log(5 === '5'); // Output: false

console.log(0 === false); // Output: false

console.log('' === false); // Output: false

Q3.What is object type and explain in scope ?

* An object is a fundamental data type that represents a collection of key-value pairs. Objects are used to store and organize data in a structured manner. They are one of the most versatile and powerful features of the language.

Here's a brief explanation of objects in JavaScript:

1.Key-Value Pairs: An object consists of one or more key-value pairs. Each key is a unique string (or symbol in ES6+) that acts as an identifier for the corresponding value. The value can be of any data type, including other objects, functions, arrays, primitives (such as strings, numbers, and booleans), and more.

2.Properties and Methods: The keys in an object are often referred to as properties. If the value associated with a key is a function, it's typically referred to as a method. Methods can be invoked by using the dot notation or square bracket notation with the object.

3.Creating Objects: There are multiple ways to create objects in JavaScript:

* Using object literals: let obj = { key1: value1, key2: value2 };
* Using the new keyword with constructor functions: let obj = new Object();
* Using object destructuring or spread syntax to create objects from existing ones.

4.Object Prototype: Every JavaScript object has a prototype, which serves as a template for creating new objects. Objects inherit properties and methods from their prototype. You can access an object's prototype via the Object.getPrototypeOf() method or the \_\_proto\_\_ property.  
  
JavaScript has two main types of scope:

1. Global Scope: Variables and functions declared outside of any function or block have global scope, meaning they can be accessed from anywhere within the program.
2. Local Scope: Variables and functions declared inside a function have local scope, meaning they are only accessible within that function (or nested functions).

Q4.What is ‘NAN’ & ‘isNan()’ function with eg ?

* ‘NaN’ stands for "Not a Number." It is a special value representing a value that is not a valid number.

1. Performing mathematical operations that involve non-numeric values, such as dividing a string by a number.
2. Converting a non-numeric string to a number using functions like parseInt() or parseFloat() when the string does not represent a valid number.

For example:

let result = 'Hello' / 2;

console.log(result); // Output: NaN

In this example, dividing the string 'Hello' by the number 2 results in NaN because 'Hello' is not a valid numeric value.

The isNaN() function in JavaScript is used to determine whether a value is NaN. It returns true if the value is NaN, and false otherwise.

Here's an example of using the isNaN() function:

console.log(isNaN('Hello')); // Output: true

console.log(isNaN(42)); // Output: false

console.log(isNaN('42')); // Output: false.

In this example:

isNaN('Hello') returns true because the string 'Hello' cannot be converted to a number.

isNaN(42) returns false because 42 is a valid number.

isNaN('42') returns false because the string '42' can be converted to the number 42.  
  
  
Q5. How to initialize variable, assign a value to a variable and reassign new value to the same variable with eg ?

* In JavaScript, We can initialize a variable, assign a value to it, and reassign a new value to the same variable using the let, const, or var keywords. Here are examples demonstrating each scenario:

1. Initialize Variable: To declare a variable without assigning a value to it initially, you can use the let, const, or var keywords followed by the variable name.

let x;

console.log(x); // Output: undefined

1. Assign a Value to a Variable: To assign a value to a variable, use the assignment operator (=) followed by the value you want to assign.

let y;

y = 10;

console.log(y); // Output: 10

1. Reassign New Value to the Same Variable: You can reassign a new value to the same variable simply by using the assignment operator with the new value.

let z = 5;

console.log(z); // Output: 5

z = 20; // Reassigning a new value

console.log(z); // Output: 20

In the example above, z is first assigned the value 5. Then, it is reassigned the value 20. After reassignment, the value of z becomes 20.

Q6.What is contagion ,interpoleation,WRAP up explain with eg.

1. Contagion:

* In the context of mathematics or statistics, contagion typically refers to the spread or transmission of a particular characteristic or phenomenon from one entity to another within a system.
* This concept is often encountered in the study of infectious diseases, where contagion refers to the transfer of a disease from one individual to another.
* However, contagion can also be applied to other contexts, such as financial markets, where it describes the rapid spread of negative (or positive) effects from one market to others.

1. Interpolation:

* Interpolation is a method used to estimate unknown values that lie between known values. It involves constructing a function or curve that passes through the given data points and then determining the value of the unknown point based on its position relative to those points.
* In simpler terms, interpolation is the process of estimating values within a range of known data points.
* One common example of interpolation is linear interpolation, where a straight line is drawn between two known data points, and the value of the unknown point is determined based on its position along that line.

1. Wrap-Up:

* In the context of a discussion or presentation, a wrap-up refers to the conclusion or summary of the main points discussed.
* It involves briefly recapping the key ideas, findings, or outcomes of the discussion or presentation to provide closure and reinforce understanding.
* A wrap-up is often used to tie together various elements of a discussion and leave the audience with a clear understanding of the main takeaways.

JavaScript is a versatile programming language with a wide range of attributes and values. It's not feasible to list all of them exhaustively, but here are some key attributes and common values associated with JavaScript:

* Dynamic Typing: JavaScript is dynamically typed, meaning variables can hold values of any data type without requiring explicit declaration.
* Prototypal Inheritance: JavaScript uses prototypal inheritance, where objects inherit properties and methods from other objects (prototypes).
* First-class Functions: Functions in JavaScript are treated as first-class citizens, meaning they can be assigned to variables, passed as arguments to other functions, and returned from functions.
* Closures: JavaScript supports closures, allowing functions to retain access to variables defined in their lexical scope even after the parent function has finished executing.
* Asynchronous Programming: JavaScript is well-suited for asynchronous programming paradigms, allowing non-blocking I/O operations through features like callbacks, promises, and async/await.
* Event-Driven Architecture: JavaScript is commonly used for event-driven programming, where functions are executed in response to events such as user actions (clicks, keypresses) or system events.
* Functional Programming Features: JavaScript supports functional programming concepts such as higher-order functions, immutability, and pure functions.
* Object-Oriented Programming: Although prototype-based, JavaScript supports object-oriented programming principles through object creation, encapsulation, inheritance, and polymorphism.
* Garbage Collection: JavaScript employs automatic garbage collection to manage memory allocation and deallocation, helping to prevent memory leaks.
* Regular Expressions: JavaScript provides support for regular expressions (regex) for pattern matching and text manipulation.

Values in JavaScript can include:

1. Primitives:

* Numbers (e.g., 42, 3.14)
* Strings (e.g., 'Hello, world!')
* Booleans (true or false)
* Undefined (undefined)
* Null (null)
* Symbols (from ES6 onwards)

1. Objects:

* Arrays (e.g., [1, 2, 3])
* Functions (e.g., function() { ... })
* Dates (e.g., new Date())
* Regular Expressions (e.g., /pattern/)
* Custom Objects (defined by the programmer)

1. Special Values:

* Infinity (Infinity)
* Negative Infinity (-Infinity)
* Not-a-Number (NaN)