**ARRAY AND OBJECT Q/A**

**47 .What is the drawback of declaring methods directly in JavaScript objects?**

* + In JavaScript, declaring methods directly in objects can have some drawbacks, primarily related to code organization, inheritance, and memory usage. Here are some potential drawbacks:
    - Code Organization: When methods are declared directly in objects, it can lead to less organized and harder-to-maintain code, especially as the codebase grows. This approach might make it difficult to separate concerns and follow best practices for modular and maintainable code.
    - Inability to Inherit: Objects created with methods declared directly are typically standalone instances, and they don't benefit from prototype-based inheritance. This can limit code reuse and make it harder to implement common functionality across multiple objects.
    - Memory Usage: Each object created with methods directly declared will have its own copy of those methods. This can lead to increased memory usage, especially if you have many instances of similar objects. In contrast, using prototype-based inheritance allows multiple objects to share the same set of methods, potentially reducing memory overhead.
    - Immutability Issues: If methods are declared directly in an object, they can be easily overwritten or modified elsewhere in the code, leading to unexpected behaviour. Using a more structured approach, such as using constructor functions and prototypes, can help enforce encapsulation and reduce the risk of unintended modifications.
    - Readability and Maintainability: Declaring methods directly in objects might make the code less readable, as the structure of the object becomes cluttered with both data properties and methods. This can make it harder for other developers (or even yourself) to understand and maintain the code over time.

To address these drawbacks, developers often use constructor functions, classes (introduced in ECMAScript 2015), or other design patterns to create more organized, maintainable, and extensible code. These approaches allow for better separation of concerns, improved code reuse, and adherence to principles like encapsulation and inheritance

**Q What is JavaScript?**

* + JavaScript is a object oriented programming language which can be used by developer to make web pages interactive.

**Q What is the use of isNaN function?**

* + The isNaN() function in JavaScript is used to determine whether a value is NaN (Not-a-Number) or not. NaN is a special value in JavaScript that represents an invalid number result of an arithmetic operation.
  + Here's how isNaN() works:
  + If the argument passed to isNaN() is NaN (i.e., not a number), it returns true.
  + If the argument is a valid number or a value that can be converted to a number, it returns false.
  + Example:
    - isNaN(NaN); // true
    - isNaN(123); // false (123 is a number)
    - isNaN('Hello'); // true ('Hello' cannot be converted to a number)
    - isNaN('123'); // false ('123' can be converted to a number)
    - isNaN(true); // false (true can be converted to the number 1)
    - isNaN(undefined); // true (undefined cannot be converted to a number)
  + It's important to note that isNaN() attempts to convert the argument to a number before determining if it's NaN, so non-numeric strings will return true. To check if a value is a valid number, without converting it, you can use Number.isNaN() introduced in ECMAScript 6 (ES6). This function doesn't perform type coercion and only returns true if the value is exactly NaN.
    - Number.isNaN(NaN); // true
    - Number.isNaN('Hello'); // false ('Hello' cannot be converted to a number)

**Q What is negative Infinity?**

* + In JavaScript, Negative Infinity is a special numeric value that represents negative infinity, or a value that is smaller than any other number. It is the result of certain mathematical operations that lead to a value that is too low to be represented by JavaScript's number data type.
  + Here's an example:
    - let negativeInfinity = -Infinity;
    - console.log(negativeInfinity); // Output: -Infinity
  + Negative Infinity can result from certain operations such as:
  + Dividing a negative number by zero.
  + Subtracting Infinity from any finite number.
  + Performing mathematical operations on non-numeric values which result in NaN (Not-a-Number), and then performing arithmetic operations with NaN.
  + For instance:
    - console.log(-1 / 0); // Output: -Infinity
    - console.log(Number.NEGATIVE\_INFINITY); // Output: -Infinity
    - console.log(Number.isFinite(-Infinity)); // Output: false
  + Negative Infinity is often used in JavaScript to represent the lowest possible numeric value or to indicate that a calculation has gone beyond the limits of representable numbers.

**Q Which company developed JavaScript?**

* + javaScript was developed by Netscape Communications Corporation, particularly by Brendan Eich in 1995.

**Q What are undeclared and undefined variables?**

* + In JavaScript, "undeclared" and "undefined" are terms that refer to different states of variables:
  + Undeclared Variables:
  + Undeclared variables are those that have not been declared using the var, let, or const keywords. When you attempt to use an undeclared variable, JavaScript will throw a ReferenceError.
  + Example of an undeclared variable:
    - console.log(x); // ReferenceError: x is not defined
  + Undefined Variables:
  + Undefined variables are those that have been declared but have not been assigned a value, or variables that have been explicitly assigned the value undefined.
  + Example of an undefined variable:
    - let y;
    - console.log(y); // Output: undefined
  + In this example, y is declared but not assigned a value, so its value is undefined.
  + It's important to note that undefined is also a special value in JavaScript that indicates that a variable has not been initialized or assigned a value. When a variable is declared but not assigned a value, its default value is undefined. Additionally, a function without a return statement implicitly returns undefined.

**Q 59 What is Bom vs Dom in JS?**

In JavaScript, both BOM (Browser Object Model) and DOM (Document Object Model) are

important concepts, but they serve different purposes.

1. DOM (Document Object Model):

• The Document Object Model (DOM) is a programming interface for HTML and XML documents.

It represents the structure of the document as a hierarchical tree of nodes, where each

node represents a part of the document (such as elements, attributes, and text).

• The DOM provides methods and properties for interacting with and manipulating the

structure and content of web pages. You can use DOM methods to access, create, modify,

and delete elements and attributes in an HTML document dynamically.

• Example DOM operations include selecting elements by their ID, class, or tag name,

changing element styles, adding event listeners, creating new elements, and more.

2. BOM (Browser Object Model):

• The Browser Object Model (BOM) represents the browser itself as an object, providing

access to browser-specific features and functionalities.

• Unlike the DOM, which deals with the document structure, the BOM deals with browser

windows and their properties, such as size, location, history, and the browser's

navigator object.

• Common BOM objects include window, document, location, history, navigator, screen,

localStorage, sessionStorage, etc.

• The BOM is not standardized like the DOM, and its features may vary between different

browsers. However, it provides essential functionalities for web development, such as

managing browser history, controlling window behavior, and handling user interactions.

**Q.60 Array vs object defences in JS?**

In JavaScript, arrays and objects are both used to store collections of data, but they

have different characteristics and use cases. Here's a comparison between arrays and

objects in terms of their definitions and typical uses:

1. Array:

• An array is a special type of object in JavaScript that stores data as a list of

elements, each identified by an index.

• Arrays are ordered collections, meaning the order of elements in an array is

preserved.

• Arrays are best suited for storing lists of items where the order is important, such

as a list of numbers, strings, or objects.

• Arrays provide built-in methods for manipulating and iterating over elements, such as

push(), pop(), shift(), unshift(), forEach(), map(), filter(), etc.

• Arrays are typically used when you need to store and access a collection of similar or

related items, and when you need to perform operations on the entire collection or

iterate through its elements.

2. Object:

• An object is a collection of key-value pairs, where each key is a unique string (or

symbol) that identifies a property, and each value can be any JavaScript data type,

including other objects, arrays, functions, etc.

• Objects are unordered collections, meaning there is no guaranteed order of properties

in an object.

• Objects are best suited for representing complex data structures, modeling real-world

entities, or organizing data with named properties.

• Objects provide a flexible and dynamic way to store and access data, as properties can

be added, updated, or removed dynamically.

• Objects are typically used when you need to represent entities with multiple

properties or attributes, such as a user object with properties like name, age, email,

etc., or when you need to organize data in a hierarchical or structured way.