**Module (JAVASCRIPT BASIC & DOM) – 4**

**(Array and object Question)**

* Q47(Q50)\_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.
* What is JavaScript?
  + JavaScript is a object oriented programming language which can be used by developer to make web pages interactive.
* 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)
* 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.
* Which company developed JavaScript?
  + JavaScript was developed by Netscape Communications Corporation, particularly by Brendan Eich in 1995.
* 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.
* What is the difference between ViewState and SessionState?
  + ViewState is used to maintain the state of controls on a single web page across postbacks, while SessionState is used to store user-specific data across multiple requests within a session. These concepts are not directly applicable to JavaScript, as JavaScript typically operates on the client-side and does not have built-in mechanisms for server-side state management like ASP.NET. However, JavaScript can interact with server-side technologies that implement ViewState and SessionState, typically through AJAX requests or other means of communication.
* What is === operator?
  + The === operator is a strict equality operator in JavaScript. It is used to compare two values for equality without performing type coercion(Coercion in programming refers to the process of converting a value from one data type to another).
  + In JavaScript, the == operator is known as the equality operator, and it performs type coercion if the operands are of different types before comparing them. This can sometimes lead to unexpected results.
  + However, the === operator checks both the value and the type of the operands. It returns true if the operands are strictly equal, meaning they have the same value and are of the same type. If they are of different types or have different values, it returns false.
  + Here's an example:
    - console.log(5 === 5); // true, both values are 5 and both are numbers
    - console.log(5 === '5'); // false, one value is a number and the other is a string
    - console.log(5 === 10); // false, values are different
  + Using the === operator is often recommended because it helps prevent unexpected behaviour that can occur due to type coercion.