What is JavaScript ?

1. Interpreted - Executed directly in the browser without needing compilation.
2. Dynamic Typing - variables do not require a specific type(`int`,`string`,etc)automatically determined at run time.
3. Programming Language - It is used in server and client side scripting.
4. Synchronous(default) - Line by line execution.
5. Single thread - One task at a time.
6. Dynamic content - AJAX(no need to reload)

Primitive & Non primitive Data types

1. Primitive Types - Immutable (cannot be changed after creation)
2. Non-Primitive Types - Mutable (can be changed)

Pass by value & Pass by reference

1. Pass by value - (Primitive) Does not affect the original variable outside the function.
2. Pass by Reference - (Non-primitive) Affects the original data outside the function.

Scope of variables

1. Global Scope: Accessible everywhere in the code.
2. Function Scope: Accessible only within the function.
3. Block Scope: Accessible only within the block `{}` where they are defined (`let` and `const`).
4. Lexical Scope: Inner functions can access variables from their outer function's scope.

Shadowing

1. JavaScript occurs when a variable is declared within a certain scope.(like a function or block) has the same name as a variable in an outer scope. The inner variable "shadows" or overrides the outer variable within that scope, meaning that within the inner scope, the outer variable is inaccessible.
2. **Inner Scope Overrides**: The variable in the inner scope takes precedence over the one in the outer scope when both have the same name.
3. **Limited to Scope**: The shadowing only happens within the inner scope. The outer variable remains unchanged and accessible outside of the inner scope.

Escape sequence

1. Newline (`\n`) - console.log("Hello\nWorld!"); - // Hello // World!
2. Tab (`\t`) - console.log("Name:\tJohn"); - // Name: John
3. Backslash (`\\`) - console.log("This is a backslash: \\"); - // This is a backslash:
4. Single Quote (`\'`) - console.log('It\'s a sunny day!'); - // It's a sunny day!
5. Double Quote (`\"`) - console.log("He said, \"Hello!\""); - // He said, "Hello!"
6. Carriage Return (`\r`) - console.log("Hello\rWorld!"); - // Output: World!
7. Backspace (`\b`) - console.log("Hello\bWorld!"); - // Output: HellWorld!
8. Form Feed (`\f`) console.log("Hello\fWorld!");

Execution context

1. The environment in which your JavaScript code runs. It keeps track of all the variables, functions, and the value of `this`.
2. Type of execution (global and function)

Temporal Dead Zone (TDZ)

1. JavaScript refers to the period between the time a variable is declared using `let` or `const` and the time it is initialised with a value
2. During this period, the variable exists but cannot be accessed, and trying to use it will result in a `ReferenceError`.

Call stack

The call stack in JavaScript is a data structure that keeps track of function calls and helps manage the execution of code. Here's a simple breakdown:

1. **Call Stack**: Keeps track of function calls, managing the order of execution.
2. **LIFO Order**: Functions are executed in a Last In, First Out manner.
3. **Function Management**: Manages adding and removing function calls as they are executed.
4. **Error Handling**: Can result in a stack overflow if function calls are too deep or recursive.

Closures - Advantage & drawbacks

1. **Definition**: A closure is a function that remembers the variables from the scope in which it was created, even after that scope has finished executing.
2. **Benefit**: Closures allow you to create private variables that cannot be accessed from outside the function.
3. **Use Case**: Useful in creating secure APIs or modules where you want to expose certain functionalities while keeping some data hidden
4. **Advantages**: Data privacy, maintaining state, and effective use in callbacks.
5. **Drawbacks**: Increased memory usage, more complex debugging, and potential for unintended behaviour.