JAVASCRIPT VARIABLES ASSIGNMENT-1

**NAME:** M. LOKESH REDDY

**1.What are variables and how many ways can we declare variables in JavaScript?**

Variables are containers for storing data values. In JavaScript, we can declare variables in three ways.They are:

1.Var

2.Let

3.Const

**EXAMPLE:**

**Let:**

Let x=45;

console.log(x);//it will print 45

**Var:**

Var y=23;

console.log(y);//it will print 23

**Const:**

Const pi=3.14;

console.log(pi);//it will print 3.14

**2.Difference between ‘var’,’let’,’const’ with examples?**

**Var:**

* It is a global scope,it can be redeclared and reinitialized.

EXAMPLE:

var a = 1;

var a = 2; // No error because it is a redeclared and reinitialized.

a = 3; // No error

**Let:**

* It is a block-scope, cannot be redeclared in the same scope but reinitialized.

EXAMPLE:

let b = 1;

// let b = 2; // Error: 'b' has already been declared.

b = 3; // No error

**Const:**

* It is a block-scope,and cannot be redeclared and reinitialized.

Example:

const c = 1;

// const c = 2; // Error: 'c' has already been declared.

// c = 3; // Error: Assigned a value to a constant variable.

**3.Explain the scope of variables and their usage?**

**Global Scope:** Variables declared outside any function or block using var. They can be accessed from anywhere in the code.

**Block Scope**: Variables declared within a block (e.g., inside {}) using let or const. Accessible only within that block.

**4. Explain the difference between “block level scope and global scope”?**

**Block-level scope**: Variables declared inside a block (e.g., if, for statements) using let or const are accessible only within that block.

**Global scope**: Variables declared outside any function or block are globally scoped and accessible from anywhere in the code.

**5.What is hoisting and explain variable hoisting in detail?**

Hoisting is JavaScript's default behavior of moving declarations to the top of the current scope. This means that variables and function declarations are processed before any code is executed.

* var variables are hoisted to the top of their function scope and initialized with undefined.
* let and const variables are also hoisted to the top of their block scope but they are not initialized. Accessing them before initialization results in a ReferenceError.

Example:

console.log(x); // undefined

var x = 5;

**6.What will be the output of the following code, and why?**

**console.log(x);** // undefined because var declarations are hoisted.

**var x = 5;**

**console.log(x);** // 5 because x is assigned to the value 5.

**x = 10;**

**console.log(x);** // 10 because x is updated to 10.

**7.What will be the output of the following code, and why?**

**console.log(a);** // ReferenceError: Cannot access 'a' before initialization.

**let a = 3;**

**console.log(a);** // 3 because a is assigned to 3.

**8.Explain the term "temporal dead zone" (TDZ).**

The temporal dead zone is the time between entering a scope and variable declaration where variables declared with let and const cannot be accessed. It starts from the beginning of the block until the variable is declared and initialized.

**9.How does variable shadowing work in JavaScript?**

Variable shadowing occurs when a variable declared within a certain scope (local scope) has the same name as a variable declared in an outer scope (global scope). The local variable shadows the global variable within its scope.

Example:

let a = 1;

function test() {

let a = 2; // Shadows the global variable `a`

console.log(a); // 2

}

test();

console.log(a); // 1

**10.What will be the output of the following code and why?**

**var x = 1;**

**if (function f() {}) {**

**x += typeof f;**

**}**

**console.log(x);** // 1undefined

* if (function f() {}) is true because functions are objects and objects are truthy.
* typeof f inside the if block is undefined because f is not accessible outside the function expression.
* x becomes 1 + "undefined" which results in "1undefined".

**11.What will be the output of the following code and why?**

**for (var i = 0; i < 3; i++) {**

**setTimeout(() => console.log(i), 1000);**

**}**

* The output will be 3 3 3 because var is function-scoped and not block-scoped. The same i is used in all iterations of the loop, and by the time the setTimeout callbacks are executed, i has been incremented to 3.

**12.Can you explain what happens when a const variable is declared but not initialized?**

* Declaring a const variable without initializing it results in a syntax error.

Example:

const a; // SyntaxError: Missing initializer in const declaration