**Assignments For JavaScript Fundamentals**

**Submitted by : - SANDEEP KURARIA**

Q1) Find invalid given identifier?

Ans:-

Identifiers given below are Invalid

var 2ndPlace;

var my-variable;

var var;

var first name;

var @variable;

Identifiers given below are valid

var \_counter;

var $totalAmount;

Q2) What is the scope of a variable declared using var, let, and const?

Ans:-

var has function-level scope or is globally scoped.

let and const have block-level scope.

var variables are hoisted, while let and const variables are not hoisted.

const variables cannot be reassigned after initialization.

Q3) What will output for this given programme?

Ans:-

function testVariables() { console.log(a);

//Output:undefined (a is hoisted but not initialized yet)

if (true)

{ var a = 10; // Variable a is hoisted and initialized here

let b = 20;

const c = 30;

}

console.log(a); // Output: 10 (a is now 10 after the assignment inside the block)

console.log(b); // Error: b is not defined (b is not accessible outside the block)

console.log(c); Error: c is not defined (c is not accessible outside the block)

}

testVariables();

Q4) What are data types in javaScript?

Ans:-

In JavaScript, data types represent the kind of data that can be stored and manipulated in a program. JavaScript is a dynamically typed language, meaning you do not need to specify the data type explicitly when declaring variables. Instead, the data type is determined automatically based on the value assigned to the variable. JavaScript has the following basic data types:  
  
Primitive Data Types:

Number: Represents numeric values, including integers and floating-point numbers.

String: Represents sequences of characters, enclosed in single or double quotes.

Boolean: Represents a logical value, either true or false.

null: Represents an intentional absence of any object value.

undefined: Represents a variable that has been declared but not assigned a value.

Symbol: Represents a unique and immutable value that may be used as the key of an object property (added in ECMAScript 6).  
  
for example:-

let age = 30; // Number

let name = "John"; // String

let isStudent = true; // Boolean

let person = null; // null

let job; // undefined

const id = Symbol("unique\_id"); // Symbol

Object Data Type:

Object: Represents a collection of key-value pairs, also known as properties, or it can be an instance of a class (including arrays, functions, etc.).

let person = { name: "Alice", age: 25, isStudent: true }; // Object literal

let numbers = [1, 2, 3, 4, 5]; // Array

function greet() { console.log("Hello!"); } // Function

Q5) What are the primitive data types in JavaScript?

Ans:-

In JavaScript, there are six primitive data types:

Number: Represents numeric values, including integers and floating-point numbers. It can be positive, negative, or zero. JavaScript uses double-precision 64-bit binary format to represent numbers.

Example:

let age = 30;

let price = 19.99;

String: Represents sequences of characters, enclosed in single or double quotes. Strings can contain letters, numbers, symbols, and spaces.

Example:

let name = "John";

let message = 'Hello, World!';

Boolean: Represents a logical value, which can be either true or false. It is used for making logical decisions and controlling flow in the code.

Example:

let isStudent = true;

let isWorking = false;

null: Represents an intentional absence of any object value. It is a special keyword that indicates the absence of a value.

Example:

let person = null;

undefined: Represents a variable that has been declared but not assigned a value. It is also used to indicate the absence of a specific property in an object or the absence of a return value in a function.

Example:

let name;

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

Symbol: Represents a unique and immutable value that may be used as the key of an object property. Symbols are primarily used to avoid naming collisions in objects.

Example:

const id = Symbol("unique\_id");

These primitive data types are immutable, which means their values cannot be changed once they are created. Additionally, they are compared by value, not by reference. When you assign a primitive value to a variable or pass it as an argument to a function, a copy of the value is created.

It's important to note that in JavaScript, objects (including arrays and functions) are not considered primitive data types. They are considered reference types because their values are stored as references to memory locations where the data is stored.

Q6) Explain the difference between primitive data types and reference (complex) data types in JavaScript.

Ans:-

In JavaScript, data types are categorized into two main groups: primitive data types and reference (complex) data types. The primary difference between them lies in how they are stored and accessed in memory.

Primitive Data Types:

Primitive data types are simple data types that hold a single value. They are immutable, which means their values cannot be changed after they are created. When you assign a primitive value to a variable or pass it as an argument to a function, a copy of the value is created.

There are six primitive data types in JavaScript: Number, String, Boolean, null, undefined, and Symbol.

Example:

let x = 10; // x is a primitive (Number)

let name = "John"; // name is a primitive (String)

let isStudent = true; // isStudent is a primitive (Boolean)

Reference (Complex) Data Types:

Reference data types are more complex data types that can hold multiple values. They are mutable, which means their values can be changed after they are created. When you assign a reference value to a variable or pass it as an argument to a function, you are actually assigning a reference to the memory location where the data is stored, rather than the actual data itself.

The reference data types include Object, Array, Function, and other objects that are created using constructors or classes.

Example:

let person = { name: "Alice", age: 25 }; // person is a reference (Object)

let numbers = [1, 2, 3, 4, 5]; // numbers is a reference (Array)

function greet() { console.log("Hello!"); } // greet is a reference (Function)

Key differences between primitive data types and reference data types:

Mutability: Primitive data types are immutable, while reference data types are mutable. This means you can change the value of a property within a reference data type without creating a new object.

Copying Behavior: When you assign a primitive value to a variable or pass it as an argument to a function, a copy of the value is created. For reference data types, when you assign a variable to another variable or pass it as an argument to a function, you are copying the reference to the object in memory, not the actual object itself. Both variables will point to the same object in memory.

Storage in Memory: Primitive data types are stored directly in the memory location where variables are allocated. For reference data types, the variable holds a reference (memory address) to the location in memory where the actual object is stored.

Comparison: Primitive data types are compared by their actual values, while reference data types are compared by their references. Two objects with the same key-value pairs will not be considered equal unless they are referencing the same memory location.

Passing as Arguments: When you pass a primitive data type as an argument to a function, it is passed by value. When you pass a reference data type as an argument, it is passed by reference, meaning the function can modify the object directly.

Q7) What are the different ways to create a string in JavaScript?

Ans:-

In JavaScript, there are several ways to create a string, allowing you to work with textual data in various contexts. Here are the different methods to create strings:

Using Single or Double Quotes:

You can create a string using either single quotes (`'`) or double quotes (`"`).

Example:

let str1 = 'Hello, World!';

let str2 = "JavaScript is awesome!";

Using Template Literals (String Interpolation):

Template literals, introduced in ECMAScript 6, allow you to create multi-line strings and embed expressions using `${}` inside backticks (``). It makes string interpolation more convenient.

Example:

let name = "John";

let greeting = `Hello, ${name}!`;

Using the String Constructor:

You can create a string using the String constructor. However, it's not recommended to create strings this way, as it's less efficient compared to using literals.

Example:

let str3 = new String("I am a string created with the String constructor.");

String Concatenation:

You can concatenate strings using the `+` operator.

Example:

let part1 = "Hello, ";

let part2 = "World!";

let greeting = part1 + part2;

Using String Methods:

There are several string methods that can be used to create strings from other data types, such as toString() or join().

Example:

let num = 42;

let str4 = num.toString(); // Converts the number to a string

let array = [1, 2, 3];

let str5 = array.join(","); // Converts the array to a comma-separated string

Using Escape Sequences:

Escape sequences allow you to include special characters within a string.

Example:

let specialChars = "This is a string with \"quotes\" and new lines.\nNew Line!";

Q8) How do you check the data type of a variable in JavaScript?

Ans:-

In JavaScript, you can check the data type of a variable using the `typeof` operator. The `typeof` operator returns a string representing the data type of the variable or expression. It is commonly used for type-checking and conditional logic.

Here's the syntax for using the `typeof` operator: typeof variable

Example:

let age = 30;

let name = "John";

let isStudent = true;

let person = { name: "Alice", age: 25 };

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

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

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

console.log(typeof person); // Output: "object"

It's important to note that `typeof null` returns "object", which is considered a historical bug in JavaScript. The `typeof` operator will return "object" for any object, including arrays and functions, as they are all considered objects in JavaScript.

If you want to check for specific reference types like arrays or functions, you can use additional methods like `Array.isArray()` and `typeof` with function objects.

Example:  
let numbers = [1, 2, 3, 4, 5];

let greet = function() { console.log("Hello!"); };

console.log(Array.isArray(numbers)); // Output: true

console.log(typeof greet === "function"); // Output: true