6/19/2024

Java Script-Questions

Shenthuri Maran (UKI_STU_843)

An Individual Task done by myself (Shenthuri Maran-UKI_STU_843)

1. **What is a variable in JavaScript?**
    - *A variable in JavaScript is a named container that will be a storage location for holding data. Or value, which can be data like numbers, strings, objects, etc. It allows you to store and manipulate data in your programs.*
    - *It is a fundamental concept in programming.*
    - *It allows a fundamental concept in Java script.*
    - *There are some key characteristics of variables in Java Script, they are Declaration, Naming, Assignment, Scope and Data Types.*

2. How do you declare a variable in JavaScript?

- *We can declare a variable in JavaScript using the var, let, or const keywords. For example: var name = "Alice"; let age = 25; const isStudent = true;*
- *var myVariable;*
- *let anotherVariable;*
- *const PI = 3.14;*
- *'var' declares a variable globally or locally to a function scope.*
- *'let' declares a block-scoped variable.*
- *'const' declares a block-scoped, read-only constant.*

3. What are the differences between var, let, and const?

- *var: Function-scoped or globally scoped, can be redeclared, and is hoisted.*
- *Let: Block-scoped, cannot be redeclared within the same scope, and is not hoisted in the same way as var.*
- *const: Block-scoped, cannot be reassigned after initialization, and like let, is not hoisted.*

4. Explain variable hoisting in JavaScript?

- *Variable hoisting in JavaScript means that variable declarations (not initializations) are moved to the top of their containing scope during the compile phase. This means you can use variables before they are declared in the code, though their value will be undefined until the actual declaration is reached.*

5. What are the scoping rules for var, let, and const?

1. *var:*

- *Function-scoped.*
- *It is accessible anywhere within the function it is declared.*
- *Function-scoped or globally scoped.*
- *Hoisted to the top of their scope.*

- *Can be re-declared and updated.*

2. *Let :*

• Block-scoped (within { } like loops or conditionals).

- *Not hoisted.*
- *Cannot be re-declared within the same scope, but can be updated.*
- *They are accessible only within the block (delimited by {}) they are declared.*

3. *Const:*

- *They are accessible only within the block (delimited by {}) they are declared.*
- *Block-scoped.*
- *Requires initialization with a value.*
- *Cannot be reassigned or re-declared within the same scope.*

6. How can you use template literals in JavaScript?

- *Template literals are enclosed by backticks (`) and allow embedded expressions using ${expression} syntax.*
- *For example: let name = "Alice"; let greeting = `Hello, ${name}! `; console.log (greeting); // Output: Hello, Alice!*

7. List the primitive data types in JavaScript.

- *String*
- *Number*
- *Boolean*
- *Null*
- *Undefined*
- *Symbol*
- *BigInt*

8. What is the difference between null and undefined?

- *Null: Represents the intentional absence of any object value. It is an assignment value.*
- *Undefined: Represents a variable that has been declared but not yet assigned a value.*

9. How do you check the type of a variable in JavaScript?

- *Can check the type of a variable using the 'typeof' operator.*

Example1:
```
let x = 42;
let y = 'Hello';
let z = true;
console.log(typeof x); // Output:"number"
console.log(typeof y); // Output:"string"
console.log(typeof z); // Output: "boolean"
```

Example2:
```
let age = 25; console.log(typeof age); // Output: "number"
```

10. Explain the difference between primitive and reference data types?

- *The difference between primitive and reference data types in JavaScript lies primarily in how they store and reference data in memory.*

- *Primitive Data Types:*

1. ***Stored Value**: Primitive data types store their values directly in the location that the variable accesses.*
2. ***Immutability**: Primitive values are immutable, meaning their values cannot be changed (though a variable can be reassigned).*
3. ***Examples**: Primitive types in JavaScript include number, string, Boolean, null, undefined and symbol.*

let num = 10; // Primitive type (number)

let str = "Hello"; // Primitive type (string)

let bool = true; // Primitive type (boolean)

- *Reference Data Types:*

1. ***Stored Reference**: Reference data types store references (addresses) to where the values are stored. They are stored and passed by reference.*
2. ***Mutability**: Objects, arrays, and functions (reference types) can be mutated, meaning their properties or elements can be changed after creation.*
3. ***Examples**: Reference types include object, array, and function.*

let obj = { name: "Alice", age: 30 }; // Reference type (object)

let arr = [1, 2, 3]; // Reference type (array) *let func = function() { /* function body */ }; // Reference type (function)*

Key Differences:

- *Storage: Primitive types store their actual values directly where variables access them, while reference types store references (addresses) to where the values are stored.*
- *Mutability: Primitive types are immutable, whereas reference types are mutable.*
- *Behaviour: Operations on primitive types (like assigning a new value) don't affect other variables referencing the same value. Operations on reference types (like modifying an array) affect all references pointing to the same object/array/function.*

11. How does type coercion work in JavaScript?

- *Type coercion in JavaScript is the automatic conversion of values between different data types during operations. It can be implicit (automatic) or explicit (user-defined using functions like `Number ()`, `String ()`, etc.). JavaScript uses type coercion to facilitate operations between different data types, such as strings and numbers, by converting values as needed.*

- *Type coercion is the automatic or implicit conversion of values from one data type to another.*

  Example: 5 + '9' results in '59' because the number 5 is coerced into a string.

12. What are the typeof operator and the instanceof operator used for?

*1. **typeof Operator**:*

- ***Purpose**: Used to determine the data type of a variable or expression.*
- *For example, typeof 42 returns "number".*

```
typeof 42; // "number"
typeof "Hello"; // "string"
typeof true; // "boolean"
typeof {}; // "object"
```

*2. **instanceof Operator**: **Purpose**: Checks whether an object belongs to a specific class or constructor function.*

```
let arr = [1, 2, 3];
arr instanceof Array; // true
let obj = {};
obj instanceof Object; // true
```

13. How do you convert a string to a number in JavaScript?

- *To convert a string to a number in JavaScript, you can use the parseInt ( ) or parseFloat ( ) functions for integer* *or floating-point conversion, respectively:*

1. ***parseInt()****: Converts a string to an integer.*

Example: ***let str = "42";***

let num = parseInt(str);

2. ***parseFloat()****: Converts a string to a floating-point number.*

Example: ***let str = "3.14";***

let num = parseFloat(str);

- *These functions parse the string and return a number based on its content. If the string does not start with a valid numeric format,* ***parseInt()*** *and* ***parseFloat()*** *will return 'NaN' (Not a Number).*

14. How do you convert a number to a string in JavaScript?

- *To convert a number to a string in JavaScript, you can use the toString ( ) method or concatenate the number with an empty string (" ").*

1.toString() Method:

```
let num = 42;
let str = num.toString();
```

2.Concatenation with Empty String:

```
let num = 3.14;
let str = num + "";
```

- ***Both methods will convert the numeric value `num` into a string stored in the variable str.***

15. What is implicit type conversion?

- *Implicit type conversion, also known as type coercion, is the automatic conversion of a value from one data type to another data type in JavaScript. This occurs when an operator or function requires operands of a different type than the ones provided. Examples include adding a string to a number or evaluating a non-boolean value in a conditional statement.*

16. What are the different methods to convert a string to a number? Explain with examples.

1. parseInt()*: Converts a string to an integer.*

```
let str1 = "42";

let num1 = parseInt(str1); // num1 will be 42

let str2 = "10.5";

let num2 = parseInt(str2); // num2 will be 10
```

2. parseFloat()*: Converts a string to a floating-point number.*

```
let str1 = "3.14";

let num1 = parseFloat(str1); // num1 will be 3.14

let str2 = "10.5";

let num2 = parseFloat(str2); // num2 will be 10.5
```

3. Number ()*: Converts a string to a number (integer or floating-point).*

```
let str1 = "42";

let num1 = Number(str1); // num1 will be 42

let str2 = "3.14";
```

let num2 = Number(str2); // num2 will be 3.14

***4. Unary Plus Operator (+)**: Converts a string to a number.*

let str1 = "42";

let num1 = +str1; // num1 will be 42

let str2 = "3.14";

let num2 = +str2; // num2 will be 3.14

***5. Math.floor(), Math.ceil(), Math.round()**: Converts a string representation of a number to an integer using mathematical functions.*

let str = "3.14";

let num1 = Math.floor(str); // num1 will be 3 (using Math.floor())

let num2 = Math.ceil(str); // num2 will be 4 (using Math.ceil())

let num3 = Math.round(str); // num3 will be 3 (using Math.round())

17. How can you handle type conversion when adding a number and a string?

- *To handle type conversion when adding a number and a string in JavaScript, you can use the following approaches:*

1. Using the `+` Operator (Concatenation):

- *JavaScript's `+` operator behaves differently depending on the types of the operands:*

  - If both operands are numbers, it performs addition.

  - If one or both operands are strings, it performs concatenation.

Example:

```
let num = 42;

let str = " is the answer.";

let result = num + str; // "42 is the answer."
```

2. Explicit Conversion:

- *You can explicitly convert the number to a string using methods like `toString()` or concatenating with an empty string `""`:*

Example:

```
let num = 42;

let str = " is the answer.";

let result = num.toString() + str; // "42 is the answer."

// or

let result2 = num + "" + str; // "42 is the answer."
```

18. Explain how parseInt() and parseFloat() functions work.

1. ***parseInt()***:

- *Purpose: Converts a string to an integer.*
- *Functionality: Parses the string from left to right until it encounters a non-digit character, then stops and returns the integer value parsed up to that point.*

```
let str1 = "42";

let num1 = parseInt(str1); // num1 will be 42

let str2 = "10.5";

let num2 = parseInt(str2); // num2 will be 10
```

2. ***parseFloat()***:

- *Purpose: Converts a string to a floating-point number.*
- *Functionality: Parses the string as a floating-point number, taking into account the decimal point and any subsequent digits.*

```
let str1 = "3.14";

let num1 = parseFloat(str1); // num1 will be 3.14

let str2 = "10.5";
```

let num2 = parseFloat(str2); // num2 will be 10.5

19. What are arrays and how do you declare them?

- *Arrays in JavaScript are special variables that can hold more than one value at a time. They are declared using square brackets [ ] and can contain elements of any data type, including other arrays.*

Declaration:

- *To declare an array in JavaScript:*

let fruits = ["Apple", "Banana", "Cherry"];

- *The array fruits now contains three elements: "Apple", "Banana", and "Cherry".*

- *Arrays can also be declared using the Array constructor:*

let numbers = new Array(1, 2, 3, 4, 5);

- *Arrays can hold any type of data, including numbers, strings, objects, or even other arrays.*

20. What is an object in JavaScript?

- *An object in JavaScript is a collection of key-value pairs where keys are strings (or symbols) and values can be of any data type. Objects are used to store and organize data and functionalities.*

```
let person = {

   name: "Alice",

   age: 30,

   city: "New York"

};

console.log(person.name); // Output: "Alice"

console.log(person.age); // Output: 30

console.log(person.city); // Output: "New York"
```

- *In this example, person is an object containing properties like name, age, and city, each with its corresponding value. Objects are fundamental to JavaScript for organizing and manipulating complex data.*