PWSkills Assignment 02

(Done By: Sunandan Sharma)

1. In JavaScript **var**, **let**, and **const** are used to declare variables but they are used in different circumstances as they also have different characteristics. Following table illustrates the different characterization and usage of the variable declaration methods in JavaScript:

Features	Var	Let	const
Scope	Function-Scoped (If declared inside a function then it is confined inside the function. Otherwise, it is declared globally)	Block-scoped (Variables are confined to the block/statement/expression where it is used)	Block-scoped (Variables are confined to the block/statement/expression where it is used)
Hoisting	Hoisted to the top of the containing scope and initialized. We can declare the variable in the method without assigning any value to it without any error. It will just be "undefined"	Hoisted but not initialized. It will show error if we use the variable before declaring.	Hoisted but not initialized. It will show error if we use the variable before declaring.
Variable Reassignment	It can be reassigned	It can be reassigned	It cannot be reassigned
Variable Redeclaration	It can be redeclared	It cannot be redeclared in the same block	It cannot be redeclared
Usage	Generally used inside function.	Generally used inside a block to avoid rehoisting or redeclaration issues.	When we want the variable remain constant and reference unchanged.

Code: (Variable Declaration "var")

```
function example() {
    console.log(x); // undefined
    var x = 5;
    console.log(x); // 5
}
console.log(example())
```

Code GitHub link: Assignment 02 - Q1 (Variable Declaration "var") Link

The Output:

```
[Running] node
"c:\Users\sunan\OneDrive\Desktop\Study\Work\Project\Projects\Software_Projects
\Programming\Web Dev\Javascript Practice\Assignment
02\Variable_Declaration_var.js"
undefined
5
undefined

[Done] exited with code=0 in 0.188 seconds
```

Code: (Variable Declaration "let")

```
function example() {
    if (true) {
        let y = 5;
        console.log(y); // 5
    }
    console.log(y); // ReferenceError: y is not defined
}
console.log(example())
```

Code GitHub link: Assignment 02 - Q1 (Variable Declaration "let") Link

The Output:

```
[Running] node
"c:\Users\sunan\OneDrive\Desktop\Study\Work\Project\Projects\Software_Projects
\Programming\Web Dev\Javascript Practice\Assignment
02\Variable_Declaration_let.js"
5
c:\Users\sunan\OneDrive\Desktop\Study\Work\Project\Projects\Software_Projects\
Programming\Web Dev\Javascript Practice\Assignment
02\Variable_Declaration_let.js:6
    console.log(y); // ReferenceError: y is not defined
```

Code: (Variable Declaration "const")

```
function example() {
    const z = 5;
    console.log(z); // 5
    z = 10; // TypeError: Assignment to constant variable.
}
console.log(example())
```

Code GitHub link: Assignment 02 - Q1 (Variable Declaration "const") Link

The Output:

```
[Running] node
"c:\Users\sunan\OneDrive\Desktop\Study\Work\Project\Projects\Software_Projects
\Programming\Web Dev\Javascript Practice\Assignment
02\Variable_Declaration_const.js"
5
c:\Users\sunan\OneDrive\Desktop\Study\Work\Project\Projects\Software_Projects\
Programming\Web Dev\Javascript Practice\Assignment
02\Variable_Declaration_const.js:4
    z = 10; // TypeError: Assignment to constant variable.
```

2. *"Hoisting"* is a JavaScript behaviour in which variable and function declarations are moved to the top of their containing scope (the global scope or the function scope), meaning we can use a variable or function before it is declared in the code. However, hoisting affects var, let, and const differently.

"var": In this case the variable is hoisted to the top of the containing scope and initialized automatically. The variable can be declared without assigning any value resulting in no error. It will just be "undefined" in the output.

"let": In this case the variable is hoisted to the top of the containing block-scope but not initialized. The variable cannot be declared without assigning any value. Doing this will result in reference error.

"const": We will observe similar hoisting behaviour to the *"let"* case. the variable is hoisted to the top of the containing block-scope but not initialized. The variable cannot be declared without assigning any value. Again, doing this will result in reference error.

Code: (Variable Declaration "var")

```
//"var" hoisting
console.log(a); // undefined
var a = 5;
console.log(a); // 5
```

Code GitHub link: Assignment 02 - Q2 (Variable hoisting "var") Link

The Output:

```
[Running] node
"c:\Users\sunan\OneDrive\Desktop\Study\Work\Project\Projects\Software_Projects
\Programming\Web Dev\Javascript Practice\Assignment
02\Variable_hoisting_var.js"
undefined
5

[Done] exited with code=0 in 0.221 seconds
```

Code: (Variable Declaration "let")

```
//"let" hoisting
console.log(b); // ReferenceError: Cannot access 'b' before initialization
let b = 10;
console.log(b); // 10
```

Code GitHub link: Assignment 02 - Q2 (Variable hoisting "let") Link

The Output:

```
[Running] node
"c:\Users\sunan\OneDrive\Desktop\Study\Work\Project\Projects\Software_Projects
\Programming\Web Dev\Javascript Practice\Assignment
02\Variable_hoisting_let.js"
c:\Users\sunan\OneDrive\Desktop\Study\Work\Project\Projects\Software_Projects\
Programming\Web Dev\Javascript Practice\Assignment
02\Variable_hoisting_let.js:2
console.log(b); // ReferenceError: Cannot access 'b' before initialization
            ^

ReferenceError: Cannot access 'b' before initialization
```

Code: (Variable Declaration "const")

```
// "const" hoisting
console.log(c); // ReferenceError: Cannot access 'c' before initialization
const c = 20;
console.log(c); // 20
```

Code GitHub link: Assignment 02 - Q2 (Variable hoisting "const") Link

The Output:

```
[Running] node
"c:\Users\sunan\OneDrive\Desktop\Study\Work\Project\Projects\Software_Projects
\Programming\Web Dev\Javascript Practice\Assignment
02\Variable_hoisting_const.js"
c:\Users\sunan\OneDrive\Desktop\Study\Work\Project\Projects\Software_Projects\
Programming\Web Dev\Javascript Practice\Assignment
02\Variable_hoisting_const.js:2
console.log(c); // ReferenceError: Cannot access 'c' before initialization
```

3. JavaScript is a dynamically typed language, meaning that variables are not bound to any specific data type. Instead, a variable can hold any type of data, and its type can change at runtime.

- ***Examples of Dynamic Typing:***
    - **Variable Declaration:** A variable can be declared without a type, and its type can change dynamically as the code executes.
    - **Function Parameters:** Function parameters in JavaScript do not require type annotations, allowing them to accept any type of argument

The ***"typeof"*** operator is used to determine the type of a variable or expression in JavaScript. It returns a string indicating the type.

- ***Examples of "typeof" operator usage:***
    - **Primitive Types:** The typeof operator can be used to check the type of primitive values.
    - **Reference Types:** The typeof operator can also be used with reference types like objects, arrays, and functions.

Code: (Dynamic Typing)

```
let dynamicVar = 42;       // dynamicVar is a number
console.log(typeof dynamicVar); // Output: "number"

dynamicVar = "Hello";      // Now dynamicVar is a string
console.log(typeof dynamicVar); // Output: "string"

dynamicVar = true;         // Now dynamicVar is a boolean
console.log(typeof dynamicVar); // Output: "boolean"

function printValue(value) {
    console.log(value);
  }                        // Dynamic typing for function parameters

  printValue(100);            // Output: 100
  printValue("JavaScript");   // Output: JavaScript
  printValue({ key: "value" }); // Output: { key: 'value' }
```

Code GitHub link: Assignment 02 - Q3 (Dynamic Typing) Link

The Output:

```
[Running] node
"c:\Users\sunan\OneDrive\Desktop\Study\Work\Project\Projects\Software_Projects
\Programming\Web Dev\Javascript Practice\Assignment 02\Dynamic_typing.js"
number
string
boolean
100
JavaScript
{ key: 'value' }
```

Code: (typeof operator)

```
console.log(typeof 42);             // Output: "number"
console.log(typeof "Hello");        // Output: "string"
console.log(typeof true);           // Output: "boolean"
console.log(typeof undefined);      // Output: "undefined"
console.log(typeof null);           // Output: "object" (this is a historical
bug in JavaScript)

console.log(typeof { key: "value" }); // Output: "object"
console.log(typeof [1, 2, 3]);          // Output: "object" (arrays are objects
in JavaScript)
console.log(typeof function() {});      // Output: "function"
```

Code GitHub link: Assignment 02 - Q3 (typeof operator) Link

The Output:

```
[Running] node
"c:\Users\sunan\OneDrive\Desktop\Study\Work\Project\Projects\Software_Projects
\Programming\Web Dev\Javascript Practice\Assignment 02\Typeof_operator.js"
number
string
boolean
undefined
object
object
object
function

[Done] exited with code=0 in 0.253 seconds
```

4. JavaScript offers various methods for string manipulation. Here are some common methods:

- ***"length"***: Returns the number of characters in a string
- ***"charAt(index)"*** and ***"str[index]"***: Returns the character at the specified index
- ***"toUpperCase()"*** and ***"toLowerCase()"***: Converts the string to uppercase or lowercase.
- ***"substring(start, end)"*** and ***"slice(start, end)"***: Extracts a part of the string from start to end. slice can also take negative indices.
- ***"split(separator)"***: Splits the string into an array of substrings based on the specified separator
- ***"replace(searchValue, newValue)"***: Replaces occurrences of searchValue with newValue
- ***"trim()"***: Removes whitespace from both ends of the string
- ***"includes(substring)"***, ***"startsWith(substring)"***, ***"endsWith(substring)"***: Checks if the string contains, starts with, or ends with the specified substring
- ***"concat(...strings)"***: Concatenates multiple strings into one
- ***"repeat(count)"***: Repeats the string count times

Code: (String Manipulation)

```javascript
// Defining a string
let str = "Hello, World!";

// 1. Length of the string
console.log("Length:", str.length); // Output: 13

// 2. Accessing characters
console.log("Character at index 0:", str.charAt(0)); // Output: "H"
console.log("Character at index 0:", str[0]); // Output: "H"

// 3. Changing case
console.log("Uppercase:", str.toUpperCase()); // Output: "HELLO, WORLD!"
console.log("Lowercase:", str.toLowerCase()); // Output: "hello, world!"

// 4. Substring extraction
console.log("Substring (1, 5):", str.substring(1, 5)); // Output: "ello"
console.log("Slice (1, 5):", str.slice(1, 5)); // Output: "ello"
console.log("Slice (-6):", str.slice(-6)); // Output: "World!"

// 5. Splitting a string
let strArray = str.split(", ");
console.log("Split by comma and space:", strArray); // Output: ["Hello",
"World!"]

// 6. Replacing a substring
let newStr = str.replace("World", "JavaScript");
console.log("Replace 'World' with 'JavaScript':", newStr); // Output: "Hello,
JavaScript!"

// 7. Trimming whitespace
let paddedStr = "   padded string   ";
console.log("Trimmed string:", paddedStr.trim()); // Output: "padded string"

// 8. Checking if a string contains a substring
console.log("Contains 'World':", str.includes("World")); // Output: true
console.log("Starts with 'Hello':", str.startsWith("Hello")); // Output: true
console.log("Ends with '!':", str.endsWith("!")); // Output: true

// 9. Concatenating strings
let str1 = "Hello";
let str2 = "World";
let concatenatedStr = str1.concat(", ", str2, "!");
console.log("Concatenated string:", concatenatedStr); // Output: "Hello,
World!"

// 10. Repeating a string
let repeatedStr = str1.repeat(3);
```

```
console.log("Repeated string:", repeatedStr); // Output: "HelloHelloHello"
```

Code GitHub link: Assignment 02 - Q4 (String Manipulation) Link

The Output:

```
[Running] node
"c:\Users\sunan\OneDrive\Desktop\Study\Work\Project\Projects\Software_Projects
\Programming\Web Dev\Javascript Practice\Assignment 02\String_Manipulation.js"
Length: 13
Character at index 0: H
Character at index 0: H
Uppercase: HELLO, WORLD!
Lowercase: hello, world!
Substring (1, 5): ello
Slice (1, 5): ello
Slice (-6): World!
Split by comma and space: [ 'Hello', 'World!' ]
Replace 'World' with 'JavaScript': Hello, JavaScript!
Trimmed string: padded string
Contains 'World': true
Starts with 'Hello': true
Ends with '!': true
Concatenated string: Hello, World!
Repeated string: HelloHelloHello

[Done] exited with code=0 in 0.222 seconds
```

5. We mention the differences and similarities between *“null”* and *“undefined”* in following two tables.

The differences table:

Features	**Null**	**undefined**
Definition	Null is a primitive value that represents the intentional absence of any object value	Undefined is a primitive value that indicates that a variable has not been assigned a value.
Type Check	typeof(null) returns object	typeof(undefined) returns “undefined”
Assignment	Can be explicitly assigned to a variable	Cannot be explicitly assigned to a variable
Usage	Used to explicitly indicate that a variable should have no value.	Default value for uninitialized variables, function parameters not passed, and missing object properties

The similarities table:

Features	**Null**	**Undefined**
Boolean Values	Both are considered false value in Boolean contexts.	Both are considered false value in Boolean contexts.
Equality Check	“null == undefined” is true	“undefined == null” is true
Usage	Used to represent the absence of a value.	Used to represent the absence of a value.