JavaScript Topics

Differences between var, let, and const

1. **Var**
   a. The scope of a *var* variable is functional scope.
   b. It can be updated and re-declared into the scope.
   c. It can be declared without initialization.
   d. It can be accessed without initialization as its default value is “undefined”.
2. **let**
   a. The scope of a *let* variable is block scope.
   b. It can be updated but cannot be re-declared into the scope.
   c. It can be declared without initialization.
   d. It cannot be accessed without initialization, as it returns an error.
3. **const**
   a. The scope of a *const* variable is block scope.
   b. It cannot be updated or re-declared into the scope.
   c. It cannot be declared without initialization.
   d. It cannot be accessed without initialization, as it cannot be declared without initialization.

Data types and Coercion and Type Conversion

```
  JavaScript is loosely typed language and most of the time operators
automatically convert a value to the right type but there are also cases
when we need to explicitly do type conversions.
```

There are two types of coercion in JavaScript:

Implicit Coercion: Type conversion is done implicitly or **automatic by JavaScript.**

```
result = '3' + 2;
console.log(result) // "32"
```

Explicit Coercion: Type conversion is done explicitly in code using the inbuilt functions like Number(), String(), Boolean(), etc.

```
result = Number('324');
console.log(result); // 324
```

Type Conversion

- Converting Strings to Numbers
- Converting Numbers to Strings
- Converting Dates to Numbers
- Converting Numbers to Dates

- Converting Booleans to Numbers
- Converting Numbers to Booleans

Converting Strings to Numbers

```
The global method Number() can convert strings to numbers.
 Number("3.14")    // returns 3.14
```

Number Methods

Number( ) Returns a number, converted from its argument

parseFloat( ) Parses a string and returns a floating point number

parseInt( ) Parses a string and returns an integer

The unary + operator can be used to convert a variable to a number:

```
  let y = "5";       // y is a string
  let x = + y;       // x is a number
```

Converting Numbers to Strings

```
  The global method String() can convert numbers to strings.
  String(x)         // returns a string from a number variable x
  String(123)      // returns a string from a number literal 123
  String(100 + 23)  // returns a string from a number from an
expression
```

Converting Dates to Numbers

The global method Number() can be used to convert dates to numbers.

```
  d = new Date();
  Number(d // returns 1404568027739
```

Converting Dates to Strings

The global method String() can convert dates to strings.

```
  String(Date())  // returns "Thu Jul 17 2014 15:38:19 GMT+0200 (W.
Europe Daylight Time)"
```

Converting Booleans to Numbers

The global method Number() can also convert booleans to numbers.

```
  Number(false)     // returns 0
  Number(true)     // returns 1
```

Converting Booleans to Strings

The global method String() can convert booleans to strings.

```
  String(false)     // returns "false"
  String(true)      // returns "true"
```

Function

A **Function** is a block of code that is designed to perform a task and executed when it is been called or invoked.

There are 3 ways of writing a function in JavaScript:

- Function Declaration
- Function Expression
- Arrow Function

1. Function Declaration: Function Declaration is the traditional way to define a function. It is somehow similar to the way we define a function in other programming languages. We start declaring using the keyword *"function".* Then we write the function name and then parameters.

```
function add(a, b) {
    console.log(a + b);
}

// Calling a function
add(2, 3);

Output: 5
```

2. Function Expression: Function Expression is another way to define a function in JavaScript. Here we define a function using a variable and store the returned value in that **variable**.

```
// Function Expression
const add = function(a, b) {
    console.log(a+b);
}

// Calling function
add(2, 3);

Output: 5
```

3. Arrow Functions: Arrow functions are been introduced in the **ES6 version** of JavaScript. It is used to shorten the code. Here we do not use the "**function**" keyword and use the arrow symbol.

```
// Single line of code
let add = (a, b) => a + b;

console.log(add(3, 2));

Output: 5
```

Note: When there is a need to include multiple lines of code we use brackets. Also, when there are multiple lines of code in the bracket we should write return explicitly to return the value from the function.

```
// Multiple line of code
const great = (a, b) => {
    if (a > b)
        return "a is greater";
    else
        return "b is greater";
}

console.log(great(3,5));
```