Var:

The scope is global when a `var` variable is declared outside a function. This means that any variable that is declared with `var` outside a function block is available for use in the whole window.
`var` is function scoped when it is declared within a function. This means that it is available and can be accessed only within that function.

Example:

```
var greeter = "hey hi";

   function newFunction() {
      var hello = "hello";
   }
```

var variables can be re-declared and updated

Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution.

Problem with var:

```
var greeter = "hey hi";
   var times = 4;

   if (times > 3) {
      var greeter = "say Hello instead";
   }

   console.log(greeter) // "say Hello instead"
```

So, since times > 3 returns true, greeter is redefined to "say Hello instead". While this is not a problem if you knowingly want greeter to be redefined, it becomes a problem when you do not realize that a variable greeter has already been defined before.
If you have used greeter in other parts of your code, you might be surprised at the output you might get. This will likely cause a lot of bugs in your code. This is why let and const are necessary.

Let:

A block is a chunk of code bounded by {}. A block lives in curly braces. Anything within curly braces is a block.

So a variable declared in a block with let is only available for use within that block.

```
et greeting = "say Hi";
    let times = 4;


    if (times > 3) {
          let hello = "say Hello instead";
          console.log(hello);// "say Hello instead"
     }
    console.log(hello)  // hello is not defined
```

let can be updated but not re-declared.

Just like var, a variable declared with let can be updated within its scope. Unlike var, a let variable cannot be re-declared within its scope. So while this will work:

Hoisting of let

Just like var, let declarations are hoisted to the top. Unlike var which is initialized as undefined, the let keyword is not initialized. So if you try to use a let variable before declaration, you'll get a Reference Error

Const:

Variables declared with the const maintain constant values. const declarations share some similarities with let declarations.

const declarations are block scoped

Like let declarations, const declarations can only be accessed within the block they were declared.

const cannot be updated or re-declared

This means that the value of a variable declared with const remains the same within its scope. It cannot be updated or re-declared.

```
const greeting = "say Hi";
greeting = "say Hello instead";// error: Assignment to constant variable.
const greeting = "say Hi";
const greeting = "say Hello instead";// error: Identifier 'greeting' has already been declared
```

Every const declaration, therefore, must be initialized at the time of declaration. This behavior is somehow different when it comes to objects declared with const. While a const object cannot be updated, the properties of this objects can be updated.