Var vs Let vs Const

- var declarations are globally scoped or function scoped
- while let and const are block scoped.
- var variables can be updated and re-declared within its scope;
- let variables can be updated but not re-declared; const variables can neither be updated nor re-declared.
- They are all hoisted to the top of their scope. But while var variables are initialized with undefined, let and const variables are not initialized.
- While var and let can be declared without being initialized, const must be initialized during declaration.

Closure

A closure in JavaScript is a function that retains access to the variables in the outer function's scope. even after the parent function has completed execution. This allows for data to be "closed over" or remembered by the inner function, even after the outer function has returned.

Callback function

A callback is a JavaScript function that is passed to another function as an argument or a parameter
callbacks are used in a way to make sure that certain code doesn’t execute until the other code finishes execution.

Function Declaration vs Function Expression

Function Declaration:

```
function greet() {
console.log('Hello, world!');
}
```

Function Expression:

```
var greet = function() {
console.log('Hello, world!');
};
```

Function declarations are hoisted, meaning they can be called before they are declared. Function expressions, on the other hand, are not hoisted, so they cannot be invoked before they are defined. Additionally, function expressions can be anonymous or named, while function declarations must always be named.

Regular functions vs Arrow functions

Regular functions duplicate named parameters are allowed but not recommended

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

Arrow functions do not allow duplicate named parameters,

Regular functions function declarations are hoisted to the top of their scope, allowing them to be called before they're defined in the code

Arrow functions are not hoisted like regular functions.

Regular functions have their own this context,

In regular functions, this refers to the object that calls the function this refers to the calling object obj. Output will be the name property.

```
const obj = {
  name: 'Geeks',
  greet: function() {
    console.log(this.name);
  }
};
obj.greet()//'Geeks'
```

Arrow functions inherit this from the outer scope, not the object itself, causing this.name to be undefined.
obj.greet(); *// Output: undefined (inherited from outer scope)*

Hoisting

hoisting is JavaScript's default behavior of moving declarations to the top of their containing scope during the compile phase, before the code has been executed. This means that you can call a function before it has been declared in your code.

Higher order function

A higher order function is a function that takes one or more functions as arguments, or returns a function as its result.

```
// Callback function, passed as a parameter in the higher order function
function callbackFunction(){
   console.log('I am  a callback function');
}
// higher order function
function higherOrderFunction(func){
   console.log('I am higher order function')
   func()
}
higherOrderFunction(callbackFunction);
```