```
const f = function fun() {
    console.log("How much are you funny");
}

f();
fun();
```

This above function is only accessible by the 'f' variable. It is not accessible outside 'f'. so it is going to be bound to the scope of 'f' only. It is technically having tight boundation with 'f'. Now we will only talk about the scope of 'f' rather than scope of function 'fun()' because 'fun()' is just only accessible inside 'f'. It means 'fun()' is only accessible by a 'f' with 'f' fun() can't exist.

* What is IIFE (Immediately Invoked function expression).

In javascript, it is a function that is defined and immediately invoked without being explicitly called.

An IIFE is defined by wrapping the function in parentheses, and then adding an additional set of parentheses at the end to immediately invoke it.

Example :—

```
(function () {
    var x = 10;
    console.log(x);
})(); // output: 10
```

Date / /

* Higher order functions (HOF)

Higher order functions are functions that take other functions as arguments, or return functions as their result. are called higher order functions.

They allow us to write more flexible and reusable code by abstracting away common logic and behaviour.

Example

Higher order function

```
function f(x, fn) {
    console.log(x);
    console.log(fn);
    fn();
}

f(10, function exect() {
    console.log("I am an expression passed to a HOF");
});
```

Example:

'setTimeout()' function takes a callback function as an argument, and runs it after a specified time delay.

```
setTimeout(function() {
    console.log("Hello, world");
}, 1000); // "Hello, world" after 1 second.
```

Here 'setTimeout' function is a higher order function.

* **Lexicographical :-**

"lexicographical" refers to the way strings are ordered based on their character sequences. When comparing two strings, Javascript compares them character by character, starting from the first character. If the first character of both strings are the same, it compares next characters, and so on. If a difference is found, the string with the character that come first in the lexicographic order is considered "less than" the string.

→ Use case :-

If we want to arrange array of names then we can arrange in alphabetical orders or dictionary order.

* **Sorting :-**

Arranging an elements into a particular order (it can be increasing or decreasing order).

* **Callbacks :-**

A function that is passed on as an argument to another function is called callback function.

The callback function is called after the main function has completed its task.

Example

```
b.sort( function cmp(a, b) {
        return a < b;
});
```

Highlighted function is callback function.

Example:

callback function

```
setTimeout( function exect() {
    console.log("Running after some time");
}, 4000);
```

Higher order function.

Higher order function are those functions which take a function as an argument.
And the function that is passed as an argument are called 'callback functions.

→ biggest problem of using callback
1. Inversion of control (most biggest problem)
2. Callback hell.

- **Callback Hell** :—

Due to the syntax of callback, sometimes whan can happen is, we pass a callback to a function inside that callback we passed another callback then inside that callback we passed another callback and so an so far. It becomes very nested structure in our code and code gets a pyramid like orientation. This is called callback hell.

"When multiple nested callbacks are used, it can lead to a situation known as "callback hell" where the code becomes difficult to read, understand and maintain. This is because the control flow of the program becomes hard to follow when there are too many nested callbacks"

Callback hell is a readability problem, that is become difficult to debug, hard to write the code and understand too.

There is no issue in work execution or business logic that we have written. So callback hell is a readiability problem.

To solve the problem of callback hell is promise. but there is also issue with promise that is promise hell. :

- **Inversion of control** :— (Biggest problem)

The control of your function which is our implementation and we have passed control of how this function should be called to someone else.

"When a third party executes part of our code. We can't exactly know when and how our code will be executed. When we lose control of our code and pass it to someone else, the Inversion of control happens."