are not available outside of this file because of node modular sys.

## * Modules:

In the client-side JS that we run inside of browsers when we declare a var. / fun. that is added to a global scope.

Eg: var sayHello = fun() {

}

window.sayHello();

→ It's added to the global scope & available via the window obj.

There's a problem with global scope. i.e. in real world application we often split our JS code into multiple files so it's possible that we've two file & we've define the same fun. (sayHello) with the exact same name.

Because this fun. is added to the global scope when we define this fun. in another file that new definition is going to overwrite the previous definition. So this is the problem with global scope.

So, in order to build reliable & maintainable application, we should avoid defining var. & fun. in to global scope.

Instead, we need modularity to create small building blocks / modules our var. & fun. so

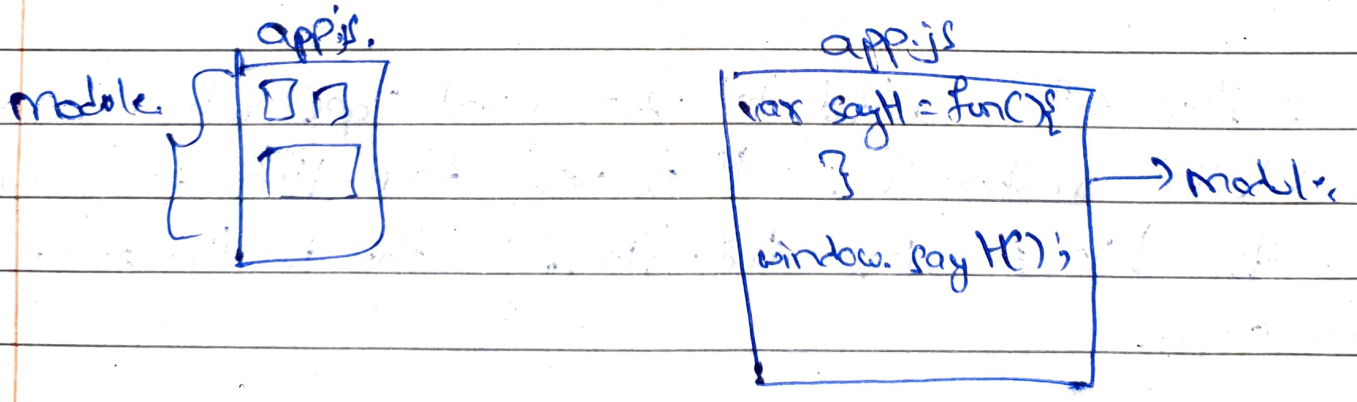
So, two var. / two fn. with the same name don't overwrite another var. / fn. defined somewhere else that encapsulated inside of that module.

→ Every file in a node application is considered a module, the var. & fn. would define in that file/ that module are scoped to that file.

In OOPs, concept we can say that they're private( not available outside that container / module).

→ We need to make a var/ fn. explicitly export it & make it public, if we need to use them outside that module.

→ Every node application has at least one file / one module which we call a main module.




app.js is main module.

→ So, in node every file is a module & the variables & fn. defined in that file are scoped to that module., they're not available outside of that module.