Topic 2 scope and closures

- There are three types scopes in node.js: global scope, function scope and lexical scope/block scope.
    - Global scope: In Node.js, definitions do not require let, const, or var. When a variable is a global variable, it can be used arbitrarily even in a separate files. The downside of global scopes is that they pollute the namespace and cause clutter, so a better option is to use module.export.
        - Even though the global variable is in the file, we can still use require("file name") to read the variable. But the problem is if you are in a big team, for example you set the variable to i. It will affect others to set other variables to i.
    - function scope: When variable x is in function, if we want to use variable x outside of function, x cannot be accessed. If we change a variable in a file, for example when we prefix a variable with var, that variable is only restricted to be used in the file.
        - After declaring x outside the function, it's also possible to declare x again inside the function when you call the function, and print x inside the function returns the value of x declared inside the function.
        - hoisting: first print x and then use var to set the value of x will return undefined. And using let will show an error because x has not been initialized.
        - lexical scope (recommended to use): In an if or while or for loop, use let to declare the variable and access the variable in the loop. But the variable cannot be accessed outside the loop.
            - In loop, use var to declare x and let to declare y. print x and y outside the loop. x will be printed but y will not.
            - When using a for loop and declaring i with let, i is just initialized inside the curly braces and not outside the loop. Therefore it is not possible to access i outside the parentheses.
    - higher order function: Just like a function generator. We will go through one function into another to access the variables of the innermost function. A function that takes a function as input or output. In other words, it is composed of an outer layer and an inner layer by functions. When the outer function is given a variable, the outer function will drive the inner function along.
        - Closures are also used in higher order functions. Closures are used to preserve access to a and b. Closures do not retain variables that do not affect the inner layer. The variable that can access the outer function can only be the inner function.
    - Looking at closures: Google Chrome uses console.dir to help us see the range of function variables.
    - Another higher order function: By using the NOT function, you create a function and then use the NOT function to do the reverse. Take function as parameter. For example, first use the is_even function to check whether the number is even. The not function also works with the built-in function (number.isInterger). Although after the is_even function it is set to undefined and not is also set to undefined. The is_odd function still works. This will involve call stack and heap. closures prevent them from being garbage

collected. When they are set to undefined, they just set the value in the call stack to undefined, and their value can still be used by the is_odd function in the heap.

- What are closures used for: Asynchronous Programming(will discussed in next topic) and Hide some variables.
- Hiding Private Member: By giving variables in real time, it acts as a kind of single-channel property to access variables. Methods like public or private sector work effectively.
- Counter: After setting the variables of the outer function, the inner function is added and the outer variables are deleted. The inner variables are saved.