To understand closures you must understand JavaScript’s variable scope. JavaScript variables can belong to the **local** or **global** scope. Global variables live as long as your application (your window / your web page) lives.

Local variables have short lives. They are created when the function is invoked, and deleted when the function is finished.

Global variables can be made local (private) with **closures**.

A closure is an inner function that has access to the outer (enclosing) function’s variables—scope chain. The closure has three scope chains: it has access to its own scope (variables defined between its curly brackets), it has access to the outer function’s variables, and it has access to the global variables.

The inner function has access not only to the outer function’s variables, but also to the outer function’s parameters. Note that the inner function cannot call the outer function’s arguments object, however, even though it can call the outer function’s parameters directly.

Example:

var add = function () {  
    var counter = 0;  
    return function () {counter += 1; return counter}  
};  
  
increment = add();// the counter is now 0  
increment (); // the counter is now 1  
increment (); // the counter is now 2

The variable **add** is assigned as a function defination.

Function only runs once. It sets the counter to zero (0), and returns a function expression.

This way increment becomes a function. The "wonderful" part is that it can access the counter in the parent scope.

This is called a JavaScript **closure.** It makes it possible for a function to have "**private**" variable i.e “counter”.

The counter is protected by the scope of the anonymous function, and can only be changed using the add function.

Reference : w3schools.com

Hoisting is JavaScript's default behavior of moving all declarations to the top. In JavaScript, a variable can be declared after it has been used.

In other words; a variable can be used before it has been declared.

var x = 5; // Initialize x  
  
elem = document.getElementById("demo"); // Find an element   
elem.innerHTML = x + " " + y;           // Display x and y  
  
var y = 7; // Initialize y

Does it make sense that y is undefined ?

Yes, this is because only the declaration (var y), not the initialization (=7) is hoisted to the top.

Because of hoisting, y has been declared before it is used, but because initializations are not hoisted, the value of y is undefined.

Above example is the same as writing:

var x = 5; // Initialize x  
var y;     // Declare y  
  
elem = document.getElementById("demo"); // Find an element   
elem.innerHTML = x + " " + y;           // Display x and y  
  
y = 7;    // Assign 7 to y

To avoid bugs, always declare all variables at the beginning of every scope.

Since this is how JavaScript interprets the code, it is always a good rule.

JavaScript in strict mode ‘use strict’ does not allow variables to be used if they are not declared.

[**Hoisting functions**](https://scotch.io/tutorials/understanding-hoisting-in-javascript#toc-hoisting-functions)

JavaScript functions can be loosely classified as the following:

1. Function declarations
2. Function expressions

These are of the following form and are hoisted completely to the top. Now, we can understand why JavaScript enable us to invoke a function seemingly before declaring it.

hoisted(); // Output: "This function has been hoisted."

function hoisted() {

console.log('This function has been hoisted.');

};

Function expressions

Function expressions, however are not hoisted.

expression(); //Output: "TypeError: expression is not a function

var expression = function() {

console.log('Will this work?');

};

As we can see above, the variable declaration var expression is hoisted but it's assignment to a function is not. Therefore, the intepreter throws a TypeError since it sees expression as a variable and not a function.

Above example is same as writing

var expression;

expression(); //Output: "TypeError: expression is not a function

expression = function() {

console.log('Will this work?');

};