Javascript-Advanced

Scope: the section of code that can access a variable or function
-
Block: limited to within a non-function body (the {} in if, for, while, ...), introduced in ES6
-
Function: limited to within a function body (the {})
-
Global: all code

```
if(true){
    //block scope
}

for(var m=0;m<10;m++){
    //block scope
}

{
    //block scope
}

function functionScope(){
    //function scope
}
```

💡 var vs let vs const

Hoisting: the code is scanned and all function and variable declarations (not initialization) are done first before executing any code.

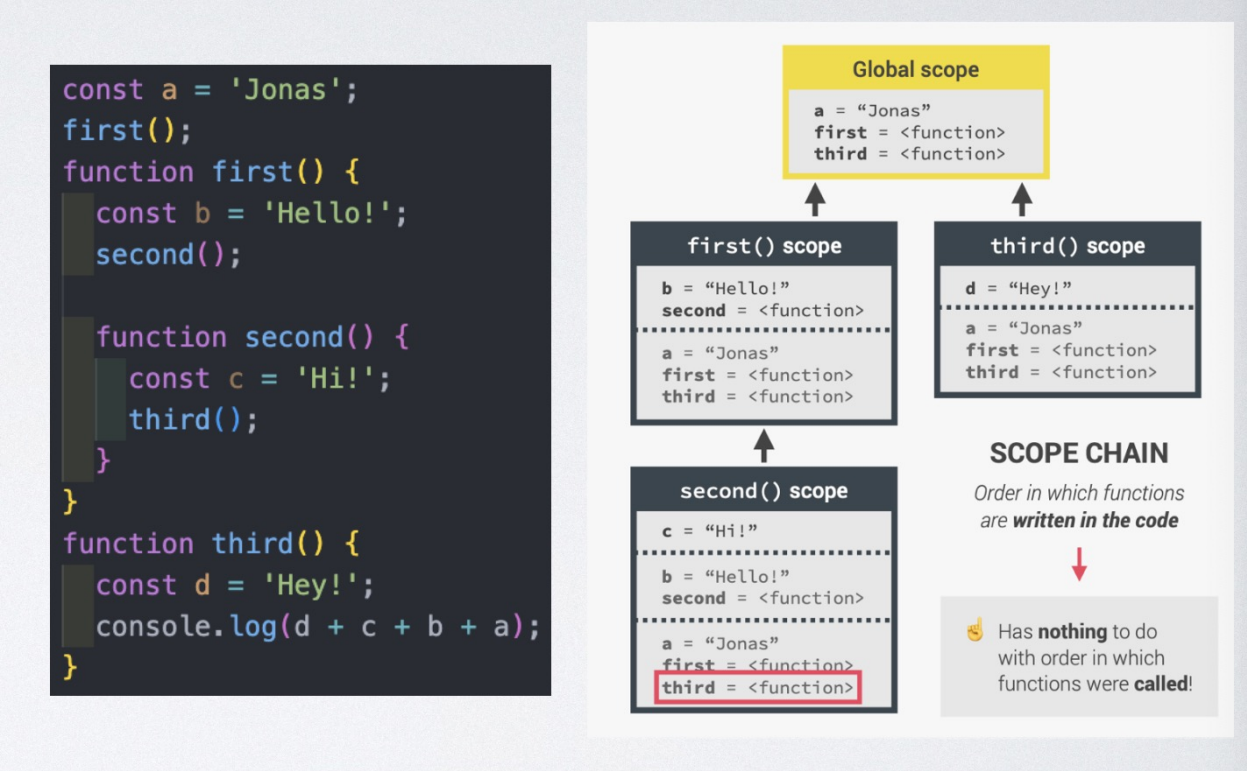
You can write code that uses a function or variable before actually declaring it.
- “Function declaration” only applies to the keyword function declaration since the function expression & arrow function are assigned to variables.

Execution context (EC): a container for all the information needed to execute some code

- By default, one global EC for the top-level code
-
new EC for each function evocation

Call stack: a stack (last-in- first-out, LIFO) that stores all EC created when code is executed
- For each function invocation, push its EC to the top of the stack

Scope chain: the cycle of looking for a variable’s value within thecurrent scope, then outer scope, and so on until finding the value orreaching the global scope



Closure: A **closure** is the combination of a function bundled together (enclosed) with references to its surrounding state (the **lexical environment**). In other words, a closure gives a function access to its outer scope.
- Use it to mimic private properties or methods in a class, for currying, or callback functions

```
function x(){
    var a = 10;
    function y(){
        console.log(a);
    }
    return y;
}
var z = x();
console.log(z);
```

Currying: break down a function that takes multiple arguments into a sequence of functions that each take a single argument.

- Each nested function is a closure.
- Function composition: functions created by functions

```
const addFn = function (a, b, c) {
    return a + b + c;
  };

  const addFnCurrying = function (a) {
    return function (b) {
      return function (c) {
        return a + b + c;
      };
    };
  };

  const arrowAddFnCurrying = a => b => c => a + b + c;

  console.log("addFn(1, 2, 3) =", addFn(1, 2, 3));
  console.log("addFnCurrying(1)(2)(3) =", addFnCurrying(1)(2)
(3));
  console.log("arrowAddFnCurrying(1)(2)(3) =", arrowAddFnCurr
ying(1)(2)(3));
```

First-class function: function that is treated like other variables
- assigned to variables, passed as arguments to a function, returned from a function

Higher-order function: function that takes in functions as arguments or returns functions

Immediately-Invoked Function Expression (IIFE): a function that is invoked when it is defined.
- Define a function, wrap it in parentheses, invoke it

```
(()=> {console.log("hello");})();
```

ES6 NEW FEATURES

- Arrow functions
- Spread/rest opearators
- Destructuring
- Template literals
- Class
- let & const
- Promises