Code Nation

JavaScript++

Scope and Higher Order Functions

{codenation}®

Learning objectives

- To understand what variable scope is
- To understand the difference between let and var
- To understand what Higher Order Functions are and what are closures


We've all experienced it...
Sometimes we have access to
a variable we've created, and
sometimes we don't.. why?


{cn}®
Let's look at 3 types of variable scope:
1. Global Scope
2. Function Scope
3. Block Scope

1. Global Scope

Global Scope

```
let globalVar = "globalVar";

console.log(`Global Scope: ${globalVar}`);
//Output: Global Scope: globalVar

const callVar = () => {
    console.log(`Inside function: ${globalVar}`);
    //Output: Inside function: globalVar
}

callVar();
```

Global Scope

```
let globalVar = "globalVar";

console.log(`Global Scope: ${globalVar}`);
//Output: Global Scope: globalVar

const callVar = () => {
    console.log(`Inside function: ${globalVar}`);
    //Output: Inside function: globalVar
}

callVar();
```

Global Scope

2. Function Scope

Function Scope

```
let globalVar = "globalVar";

console.log(`Global Scope: ${globalVar}`);

const callVar = () => {
    let localVar = "localVar";

    console.log(`Inside function: ${globalVar}`);
    console.log(`Inside function: ${localVar}`);
}

callVar();
console.log(`Global Scope with local variable: ${localVar}`);
//ReferenceError: localVar is not defined
```

Function Scope

Global Scope

```
let globalVar = "globalVar";

console.log(`Global Scope: ${globalVar}`);

const callVar = () => {
    let localVar = "localVar";

    console.log(`Inside function: ${globalVar}`);
    console.log(`Inside function: ${localVar}`);
}

callVar();
console.log(`Global Scope with local variable: ${localVar}`);
//ReferenceError: localVar is not defined
```

Function Scope

```
let globalVar = "globalVar";

console.log(`Global Scope: ${globalVar}`);

const callVar = () => {
    let localVar = "localVar";

    console.log(`Inside function: ${globalVar}`);
    console.log(`Inside function: ${localVar}`);
}

callVar();
console.log(`Global Scope with local variable: ${localVar}`);
//ReferenceError: localVar is not defined
```

Function Scope

Scope chain

Scoping works outwards

*JS looks for variables in the **current** scope, if it doesn't find it, it will then go to look outward to the previous scope until the global scope if needed.

This is called Scope chain.

Scope chain

```
let globalVar = "globalVar";

console.log(`Global Scope: ${globalVar}`);

const outerFun = () => {
    let outerVar = "outerVar";

    console.log(`Outer function: ${globalVar}`);
    console.log(`Outer function: ${outerVar}`);
    console.log(`Outer function: ${innerVar}`); //ReferenceError: innerVar is not defined

    const innerFun = () => {
        let innerVar = "innerVar";

        console.log(`Inner function: ${globalVar}`);
        console.log(`Inner function: ${outerVar}`);
        console.log(`Inner function: ${innerVar}`);

    }
    innerFun();
}
outerFun();
innerFun(); //ReferenceError: innerFun is not defined (as it's inside outerFun())
```

Scope chain

Global Scope

```
let globalVar = "globalVar";

console.log(`Global Scope: ${globalVar}`);

const outerFun = () => {
    let outerVar = "outerVar";

    console.log(`Outer function: ${globalVar}`);
    console.log(`Outer function: ${outerVar}`);
    console.log(`Outer function: ${innerVar}`); //ReferenceError: innerVar is not defined

    const innerFun = () => {
        let innerVar = "innerVar";

        console.log(`Inner function: ${globalVar}`);
        console.log(`Inner function: ${outerVar}`);
        console.log(`Inner function: ${innerVar}`);

    }
    innerFun();
}
outerFun();
innerFun(); //ReferenceError: innerFun is not defined (as it's inside outerFun())
```

Scope chain

Global Scope

```
let globalVar = "globalVar";

console.log(`Global Scope: ${globalVar}`);
```

OuterFun Scope

```
const outerFun = () => {
    let outerVar = "outerVar";

    console.log(`Outer function: ${globalVar}`);
    console.log(`Outer function: ${outerVar}`);
    console.log(`Outer function: ${innerVar}`); //ReferenceError: innerVar is not defined

    const innerFun = () => {
        let innerVar = "innerVar";

        console.log(`Inner function: ${globalVar}`);
        console.log(`Inner function: ${outerVar}`);
        console.log(`Inner function: ${innerVar}`);

    }
    innerFun();
}
outerFun();
innerFun(); //ReferenceError: innerFun is not defined (as it's inside outerFun())
```

Scope chain

Global Scope

OuterFun Scope

innerFun Scope

```
let globalVar = "globalVar";

console.log(`Global Scope: ${globalVar}`);

const outerFun = () => {
    let outerVar = "outerVar";

    console.log(`Outer function: ${globalVar}`);
    console.log(`Outer function: ${outerVar}`);
    console.log(`Outer function: ${innerVar}`); //ReferenceError: innerVar is not defined

    const innerFun = () => {
        let innerVar = "innerVar";

        console.log(`Inner function: ${globalVar}`);
        console.log(`Inner function: ${outerVar}`);
        console.log(`Inner function: ${innerVar}`);

    }
    innerFun();
}
outerFun();
innerFun(); //ReferenceError: innerFun is not defined (as it's inside outerFun())
```

Scope chain

Global Scope

OuterFun Scope

innerFun Scope

```
let globalVar = "globalVar";

console.log(`Global Scope: ${globalVar}`);

const outerFun = () => {
    let outerVar = "outerVar";

    console.log(`Outer function: ${globalVar}`);
    console.log(`Outer function: ${outerVar}`);
    console.log(`Outer function: ${innerVar}`); //ReferenceError: innerVar is not defined

    const innerFun = () => {
        let innerVar = "innerVar";

        console.log(`Inner function: ${globalVar}`);
        console.log(`Inner function: ${outerVar}`);
        console.log(`Inner function: ${innerVar}`);

    }
    innerFun();
}
outerFun();
innerFun(); //ReferenceError: innerFun is not defined (as it's inside outerFun())
```

Scope chain

Global Scope

OuterFun Scope

innerFun Scope

```
let globalVar = "globalVar";

console.log(`Global Scope: ${globalVar}`);

const outerFun = () => {
    let outerVar = "outerVar";

    console.log(`Outer function: ${globalVar}`);
    console.log(`Outer function: ${outerVar}`);
    console.log(`Outer function: ${innerVar}`); //ReferenceError: innerVar is not defined

    const innerFun = () => {
        let innerVar = "innerVar";

        console.log(`Inner function: ${globalVar}`);
        console.log(`Inner function: ${outerVar}`);
        console.log(`Inner function: ${innerVar}`);

    }
    innerFun();
}
outerFun();
innerFun(); //ReferenceError: innerFun is not defined (as it's inside outerFun())
```

3. Block Scope

Function Scope

```
let stringLet = "let string";
var stringVar = "var string";

const newString = () => {
    let stringLet = "new let string";
    var stringVar = "new var string";

    console.log(`Inside function: ${stringLet}`); //new
    console.log(`Inside function: ${stringVar}`); //new
}

newString();

console.log(`Outside function: ${stringLet}`); //old
console.log(`Outside function: ${stringVar}`); //old
```

*Variable declared inside the function will exist inside the function only.

3. Block Scope - for loops

Block Scope (let vs var)

```
function startLet(){
    for (let i = 0; i < 5; i++){
        console.log(i); //Output: 0,1,2,3,4
    }
    console.log(i); //ReferenceError: i is not defined
}

function startVar(){
    for (var i = 0; i < 5; i++){
        console.log(i); //Output: 0,1,2,3,4
    }
    console.log(i); //Output: 5
}

console.log("Running with let:");
startLet();

console.log("Running with var:");
startVar();
```

Block Scope (let vs var – for)

let

```
function startLet(){
    for (let i = 0; i < 5; i++){
        console.log(i); //Output: 0,1,2,3,4
    }
    console.log(i); //ReferenceError: i is not defined
}
```

```
function startVar(){
    for (var i = 0; i < 5; i++){
        console.log(i); //Output: 0,1,2,3,4
    }
    console.log(i); //Output: 5
}
```

```
console.log("Running with let:");
startLet();
```

```
console.log("Running with var:");
startVar();
```

*Block Scope also applies in while loop

Block Scope (let vs var)

```
function startLet(){
    for (let i = 0; i < 5; i++){
        console.log(i); //Output: 0,1,2,3,4
    }
    console.log(i); //ReferenceError: i is not defined
}
```

var

```
function startVar(){
    for (var i = 0; i < 5; i++){
        console.log(i); //Output: 0,1,2,3,4
    }
    console.log(i); //Output: 5
}
```

```
console.log("Running with let:");
startLet();
```

```
console.log("Running with var:");
startVar();
```

Block Scope (let vs var)

let

```
function startLet(){
    for (let i = 0; i < 5; i++){
        console.log(i); //Output: 0,1,2,3,4
    }
    console.log(i); //ReferenceError: i is not defined
}
```

var

```
function startVar(){
    for (var i = 0; i < 5; i++){
        console.log(i); //Output: 0,1,2,3,4
    }
    console.log(i); //Output: 5
}
```

```
console.log("Running with let:");
startLet();
```

```
console.log("Running with var:");
startVar();
```

3. Block Scope - if else

Block Scope (let vs var – ifelse)

```
function startLet(){
    for (let i = 0; i < 5; i++){
       if(true){
           let colour = "red";
           console.log(i, colour); //Output: 0,1,2,3,4 with red
       }
    }
    console.log(i, colour); //ReferenceError: i is not defined
}

function startVar(){
    for (var i = 0; i < 5; i++){
        if(true){
            var colour = "blue";
            console.log(i, colour);//Output: 0,1,2,3,4 with blue
        }
    }
    console.log(i, colour); //Output: 5, blue
}

console.log("Running with let:");
startLet();

console.log("Running with var:");
startVar();
```

Block Scope (let vs var – ifelse)

let

```
function startLet(){
    for (let i = 0; i < 5; i++){
       if(true){
           let colour = "red";
           console.log(i, colour); //Output: 0,1,2,3,4 with red
       }
    }
    console.log(i, colour); //ReferenceError: i is not defined
}

function startVar(){
    for (var i = 0; i < 5; i++){
        if(true){
            var colour = "blue";
            console.log(i, colour);//Output: 0,1,2,3,4 with blue
        }
    }
    console.log(i, colour); //Output: 5, blue
}

console.log("Running with let:");
startLet();

console.log("Running with var:");
startVar();
```

Block Scope (let vs var – ifelse)

```
function startLet(){
    for (let i = 0; i < 5; i++){
       if(true){
           let colour = "red";
           console.log(i, colour); //Output: 0,1,2,3,4 with red
       }
    }
    console.log(i, colour); //ReferenceError: i is not defined
}
```

var

```
function startVar(){
    for (var i = 0; i < 5; i++){
        if(true){
            var colour = "blue";
            console.log(i, colour);//Output: 0,1,2,3,4 with blue
        }
    }
    console.log(i, colour); //Output: 5, blue
}
```

```
console.log("Running with let:");
startLet();
```

```
console.log("Running with var:");
startVar();
```

Block Scope (let vs var – ifelse)

let

```
function startLet(){
    for (let i = 0; i < 5; i++){
       if(true){
           let colour = "red";
           console.log(i, colour); //Output: 0,1,2,3,4 with red
       }
    }
    console.log(i, colour); //ReferenceError: i is not defined
}
```

var

```
function startVar(){
    for (var i = 0; i < 5; i++){
        if(true){
            var colour = "blue";
            console.log(i, colour);//Output: 0,1,2,3,4 with blue
        }
    }
    console.log(i, colour); //Output: 5, blue
}
```

```
console.log("Running with let:");
startLet();
```

```
console.log("Running with var:");
startVar();
```


{cn}®
Don't hassle the HOF.
Higher Order Functions

Higher Order Function – Ex 1

```
const whichGreeting = (timeOfDay) => {
    console.log(`Good ${timeOfDay}`);
}

const greet = (time, fn) => {
    if(time < 1200){
        fn("Morning");
    }else if (time >= 1200 && time < 1800){
        fn("Afternoon");
    }else{
        fn("Evening");
    }
}

greet(1400, whichGreeting);
```

*Higher Order Function is when you call a function and you pass another function as a parameter.

Higher Order Function – Ex 2

```
const add = () => {
    return 2 + 3;
}

add(); //logs 5

add; // logs the whole function
```

Higher Order Function - Ex 2.1

```
const add = (num1) => {
    return (num2) => {
        return num1 + num2;
    }
}
console.log(add(2)); //returns the function in the main function
```

*Higher Order Function is when you create a function that returns another function.

Higher Order Function - Ex 2

```
const add = (num1) => {
    return (num2) => {
        return num1 + num2;
    }
}
console.log(add(2)(1)); //output: 3
```

*1 was passed to the function that sits inside the main function

Learning objectives

- To understand what variable scope is
- To understand the difference between let and var
- To understand what Higher Order Functions are and what are closures

Activity (1)

- Write a simple function which logs “Hello Code Nation” to the console.
- Then write a higher order function which will run our simple function five times, even though you only call it one time.
- Hint: Pass the simple function as a parameter, and this will involve a for loop.

Activity (2)

- The array method **.map** is an example of a higher order function.
- Declare a variable with five numbers, then use **.map** to iterate through the array and multiply each array item by 3.

Activity (3)

- Test this function to make sure it works by passing a number to the doMaths function, then passing a number and one of the four maths functions, to the returned function.

```
const add = (a,b) => {
    return a+b;
}
const subtract = (a,b) => {
    return a - b;
}
const multiply = (a,b) => {
    return a*b;
}
const divide = (a,b) => {
    return a/b;
}

const doMaths = (num1) => {
    return (num2, fn) => {
        return fn(num1, num2);
    }
}
```