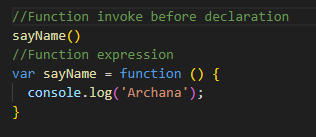
### **Hoisting**

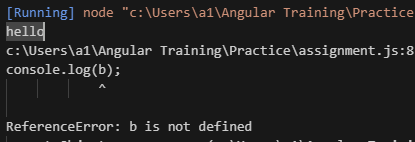
1. **List down techniques with examples where hoisting does not work as expected in JS.**
   1. It would not work for function expressions as those are not hoisted.



1. **Give output:**

****

Output:

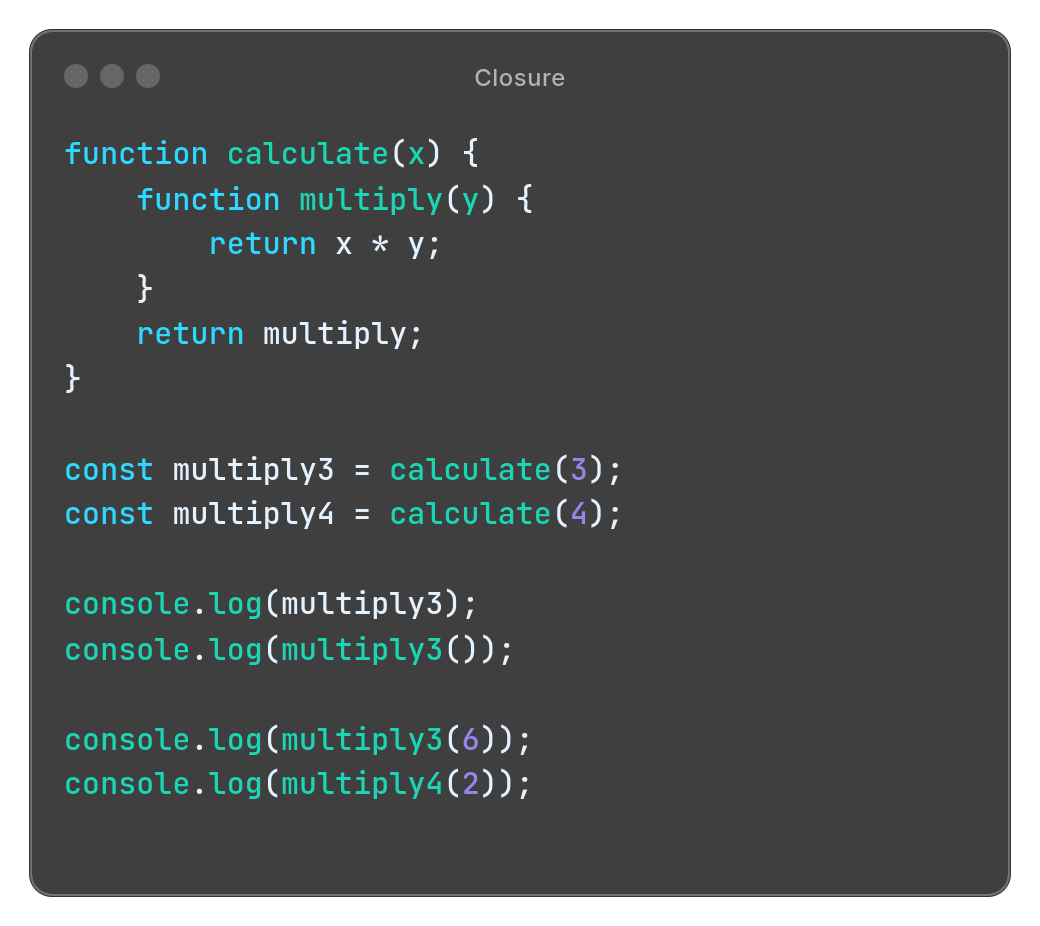
****

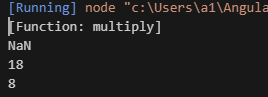
Explanation:

b = "hello";: This line assigns the value "hello" to a variable b, but here var, let, const is not used with b which makes it a global scope. And var b;: This declares a variable b inside the greet() function. Which makes it a function cope and the output shows not defined.

**Closures**

1. **Write a factorial program of given range: 0 - 10 using closure.**
2. **Write an output with explanation.**

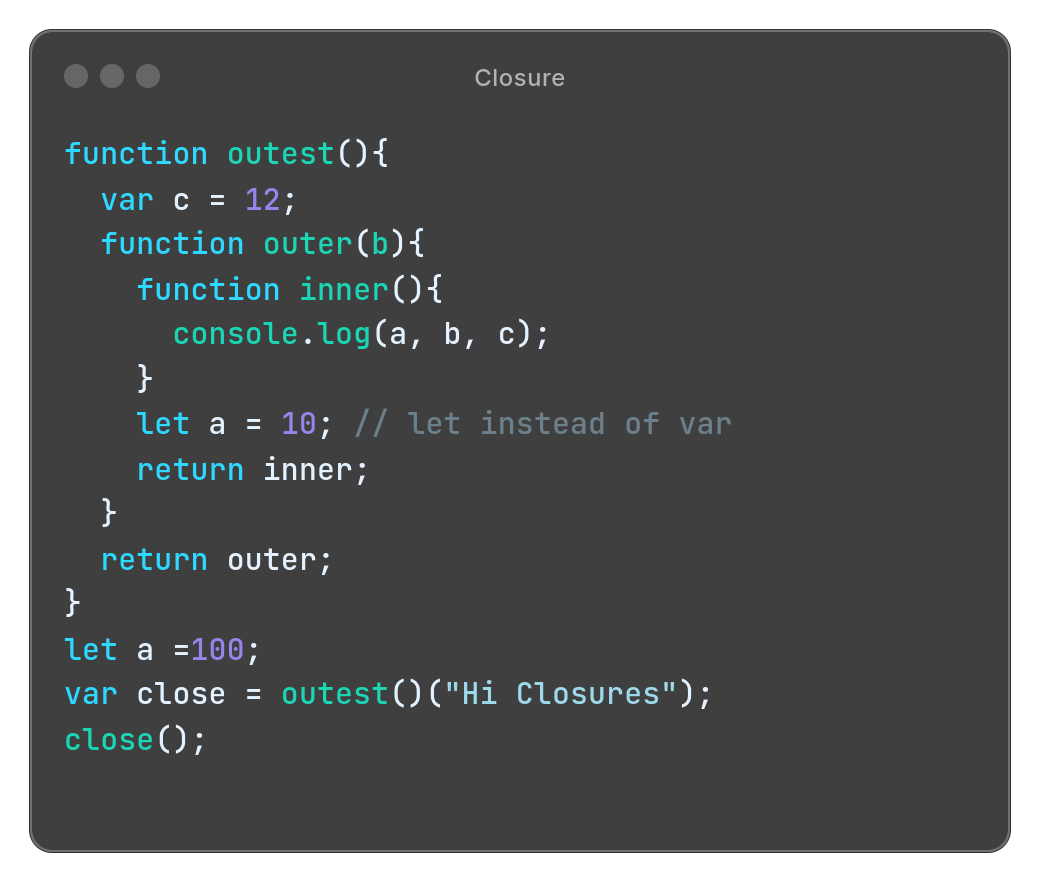
****

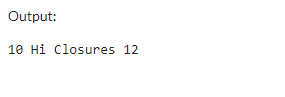
Output**:   
 **

Explanation:

Here calculate function and another function inside it. It takes x and returns multiply. The inner function takes y parameter and returns x\*y.  
Multiply3: it will return function definition of multiply as it has no parameter  
Multiply(): here The NaN appears because multiply3() is called without an argument, leading to a multiplication operation involving undefined, which results in NaN.  
Multiply3(6): this one has an argument 6 so it will perform operation with 6 and return 3\*6=18.  
Multiply4(2): similarly this one has argument 2 so it will perform operation with 2 and return 4\*2=8.

1. **Write an output with explanation:**

****

****

Explanation**:**

Here in function outer(b),Inside the outer function, b receives the value "Hi Closures".

When inner is called close(), it logs the values of a, b, and c to the console.

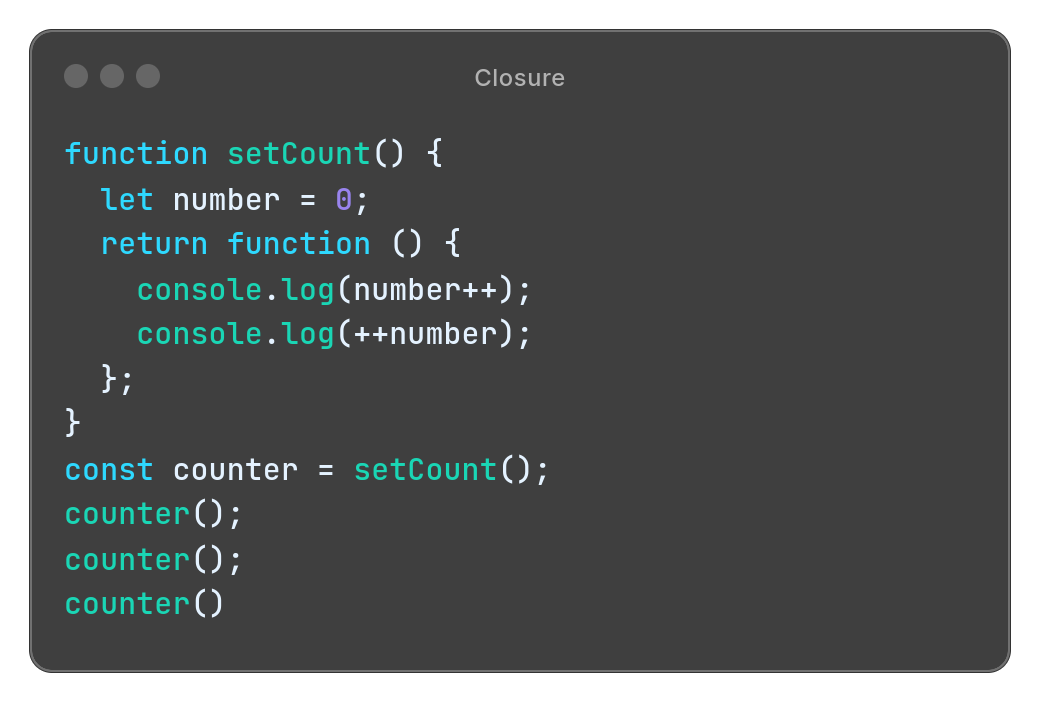
The value of a is searched first in its local scope and a is declared with let in outer function and assigned the value 10

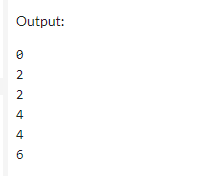
The value of b is "Hi Closures", as passed during the invocation of outer.

The value of c is 12, as it's declared in the outest.

Hence the output is “10 Hi Closures 12”

1. **Write an output with explanation:**

****

****

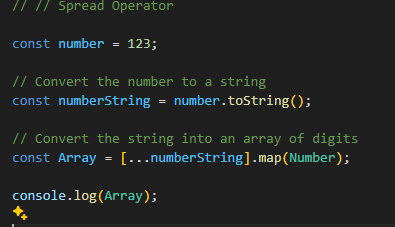
**Explanation:**Here when counter is called it performs two operations post-increment and pre-increment   
In Post increment: number++ is evaluated first. It prints the current value of (0) and then increments it by 1. So, it prints 0 and 1.

++number is evaluated next. It increments the value of number by 1 and then prints the updated value. So, it prints 2. Now when counter in called multiple times it prints the current value then increment then again print the increment value.

### **Spread Operator**

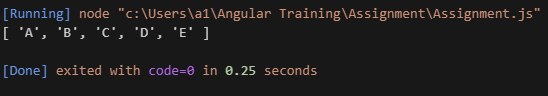
# **Converts a specified number to an array of digits.**

**Input - 123**

**Output- [1,2,3]  
 **

1. **var alphabets = ["A", ..."BCD", "E"];**

**console.log(alphabets);** Output:



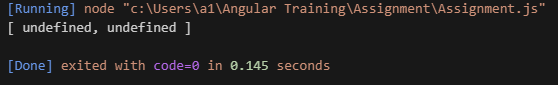
Explanation:

..."BCD" will be spread into individual elements "B", "C", and "D",

1. **var newArray = [...[,,]];**

**console.log(newArray);**

Output:

****

Explanation:

[,,] is an array with empty slots it does not contain any value, […[,,]] … would have spread the value but [,,] does not contain anything to spread.

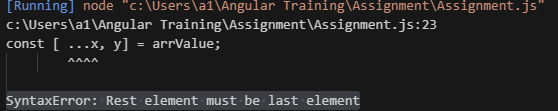
### **Object/Array Destructuring**

1. **const arrValue = ['one', 'two', 'three', 'four'];**

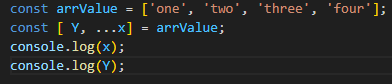
**const [ ...x, y] = arrValue;**

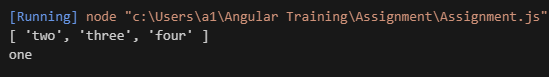
**console.log(x);**

Output:

****

Explanation:  
It does not include any skipped elements and so for that reason, the rest pattern always must be the last in the destructuring assignment because otherwise how will JavaScript know until when it should collect the rest of the array.  
But we can write it this way :





**// nested destructuring assignment in arrays**

1. **const arrValue = ["one", ["two", "three"]];**

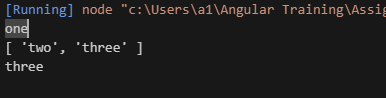
**const [x, [y, z]] = arrValue;**

**console.log(x);**

**console.log([y,z]);**

**console.log(z);**

Output:



Explanation:  
It uses nested array destructuring to extract values from arrValue. It assigns the first element of arrValue ("one") to variable x, and it destructures the second element of arrValue (which is ["two", "three"]) into variables y and z.

**// assigning default value 5 and 7**

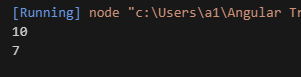
1. **let arrValue = [10];**

**let [x = 5, y = 7] = arrValue;**

**console.log(x);**

**console.log(y);**

Output:



Explanation:  
It uses array destructuring to assign values from arrValue to variables x and y, with default values provided. Since arrValue has only one element, x will have the value 10 and y is assigned the default value of 7 because there is no second element in arrValue.

1. **const [a, b, ...[ length]] = [1, 2, 3];**

**console.log (a, b, length);**

Output:



Explanation:  
a is assigned the first element of the array, which is 1.

b is assigned the second element of the array, which is 2.

...[length] uses the rest parameter syntax to gather all remaining elements of the array [3] into a new array named length.

1. **const [a, b, ... {length}] = [1, 2, 3];**

**console.log (a, b, length);**

Output:



Explanation:

a is assigned the first element of the array, which is 1.

b is assigned the second element of the array, which is 2.

...{length} uses the rest parameter syntax to gather all remaining elements of the array [3] and returns the length of that array which is 1 in this case.

### **Call, apply and bind**

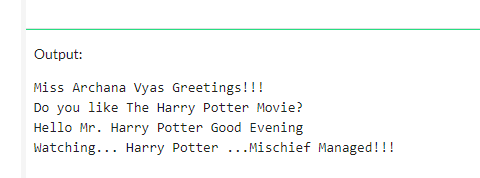
1. **Define a program with two objects person1 and person2. Person1, person2 both have firstname, lastname properties(use any name you want).**

**Add a function fullname in person1 with two arguments (prefix, suffix), which prints persons fullname using firstname lastname and adds prefix and suffix accordingly if present.**

**Note: this fullname function is present in object person1 only.**

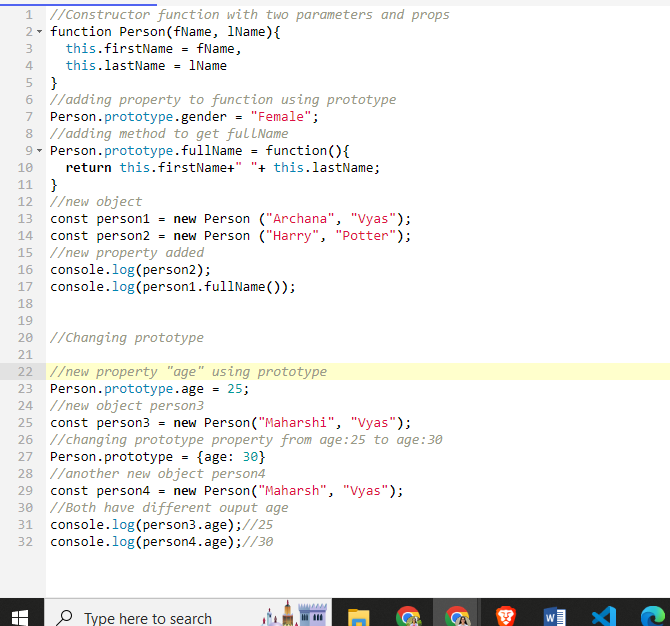
**Using call, apply, bind, print the fullname of person2 with proper parameter passed.**

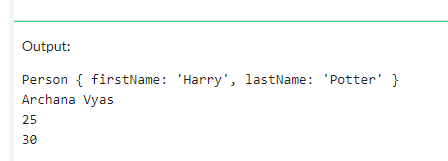
****

****

### **Prototype**

1. **Define a program that creates a custom method for the Array or Object prototype, then calls that method on its instance.**

****

****