

DAY 3 - TASK

Hoisting:

1. List down techniques with examples where hoisting does not work as expected in JS.

Var variable

- has uncertain behavior in terms of hoisting.

```
console.log(a);  
var a = 5;
```

- Output: undefined
- Variable a is hoisted but value remains undefined.

Func hoisting:

```
greet(); // Output: Hi, there.  
function greet() {  
    console.log('Hi, there.');}
```

- This will be hoisted.

```
greet(); // Output: Hi, there.  
function greet() {  
    console.log('Hi, there.');}  
var output = greet();
```

- This won't be hoisted.

Local and Global Hoisting

```
Var a = 4
function printGreeting() {
  b = 'hello';
  console.log(b);
  var b;
}
printGreeting();
console.log(b);
```

2. Give output.

hello

B is not defined

- It's because of uncertain behavior of var variable.

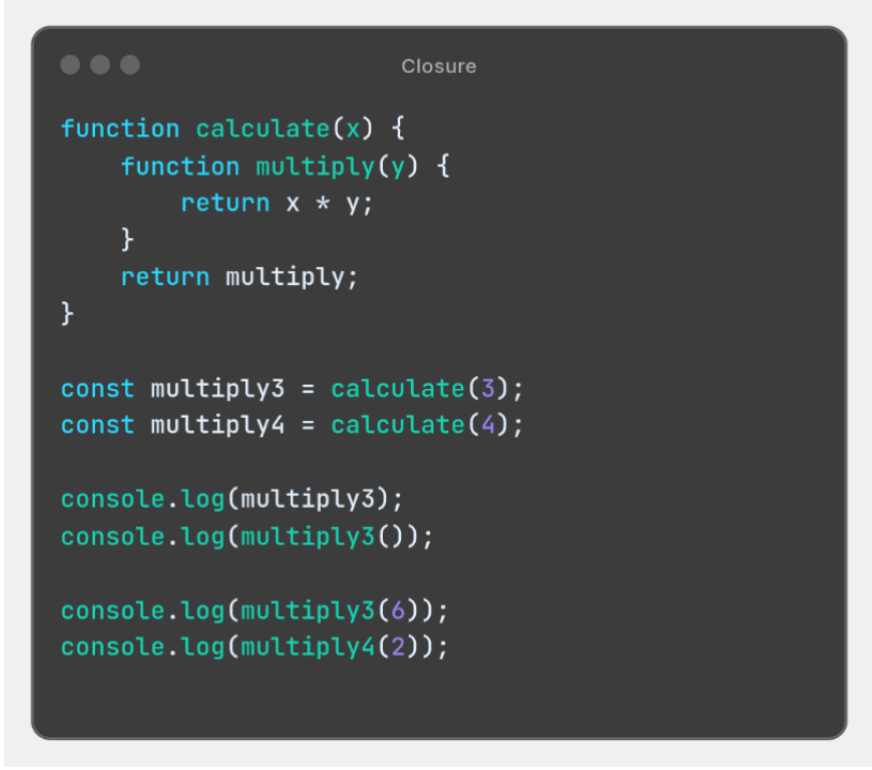
Closure:

1. Write a factorial program of given range : 0 to 10.

```
function factorial(){
  (n) => {
    if(n==0 || n==1) return 1;
    else return n * factorial(n-1);
  }
  return factorial();
}

const fact = factorial();
console.log(factorial(7));
```

2. Give output.



```
function calculate(x) {
  function multiply(y) {
    return x * y;
  }
  return multiply;
}

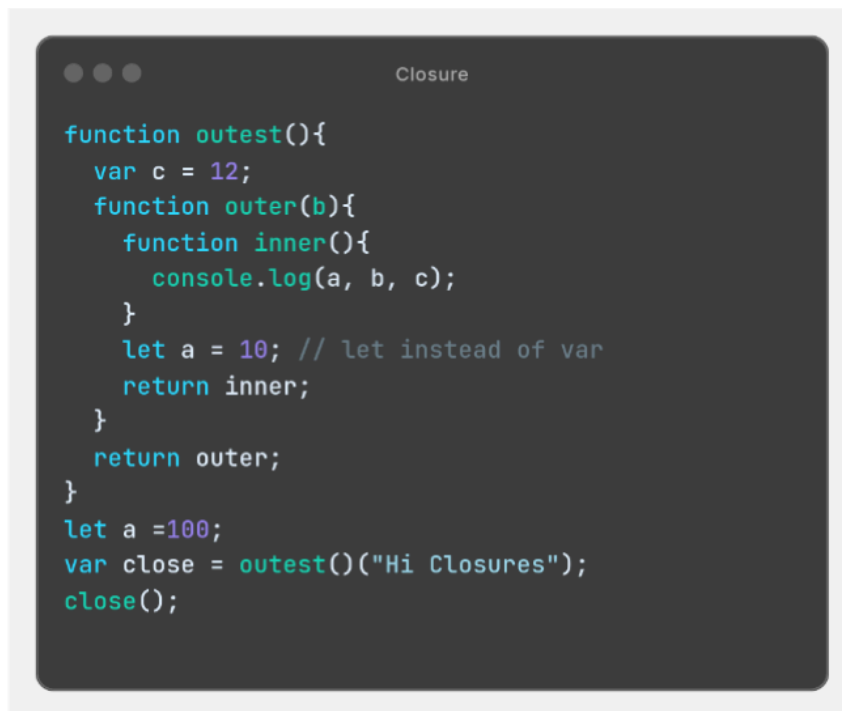
const multiply3 = calculate(3);
const multiply4 = calculate(4);

console.log(multiply3);
console.log(multiply3());

console.log(multiply3(6));
console.log(multiply4(2));
```

- The output would be
18
8
- When we execute the `multiply3 = calculate(3)` it simply stores value for `x` as 3 and then when we trigger `console.log(multiply3(6))` then it will calculate the value.
- Same for the 2nd case.

3. Give output.



```

function outest(){
  var c = 12;
  function outer(b){
    function inner(){
      console.log(a, b, c);
    }
    let a = 10; // let instead of var
    return inner;
  }
  return outer;
}
let a = 100;
var close = outest("Hi Closures");
close();

```

- Output would be

10 Hi Closures 12
- Here when `let` variable hoisting is being processed, it will consider `a`'s value as the newest value, in the function scope of `outest`, the know `a` would be 10, hence it will taken as 10.
- Now when we call
`Var close = outest("Hi Clousers")` it will pass the "Hi Clousers" to outer var and then we get the output.

4. Give output.

0

2

2

4

4

6

- First Of All, the `number++` operation will increment the number when the second operation of the number is encountered.
- So, for the first iteration it will be 0 itself, now pre-increment will occur, then the `++number` will take the number value as 1 and pre-increment it, hence it will be 2.
- Now other iterations will act the same.

Spread Operator:

1. Converts a specified number to an array of digits.

Input - 123

Output- [1,2,3]

```
function numberToArray(number) {  
  return [...String(number)].map(Number);  
}  
const number = 123;  
const result = numberToArray(number);  
console.log(result);
```

2. `var alphabets = ["A", ..."BCD", "E"];`
`console.log(alphabets);`

Console ×

► (5) ["A", "B", "C", "D", "E"]

3. `var newArray = [...[,]];`
`console.log(newArray);`

Console ×

► (2) [undefined, undefined]

Object/Array Destructuring:

1. `const arrValue = ['one', 'two', 'three', 'four'];`
`const [...x, y] = arrValue;`
`console.log(x);`

- It will give error
- Spread operator doesn't support this syntax

2. `const arrValue = ["one", ["two", "three"]];`
`const [x, [y, z]] = arrValue;`
`console.log(x);`
`console.log([y,z]);`
`console.log(z);`

- Output would be
One
["two", "three"]
three

3. `let arrValue = [10];`
`let [x = 5, y = 7] = arrValue;`
`console.log(x);`
`console.log(y);`

- Output would be
10
7
- Because the first parameter of the array will take value as 10 and other value will remain as it is which is 7.

4. `const [a, b, ...[length]] = [1, 2, 3];`
`console.log(a, b, length);`

- Output will be
1 2 3

5. `const [a, b, ...{ length }] = [1, 2, 3];`
`console.log(a, b, length);`

- Output will be
1 2 1

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.

Using Bind:

```
let Person1 = {
  firstname: "Parangi",
  lastname: "Rathod",
  fullname: function(prefix, suffix) {
    console.log(prefix + ' ' + this.firstname + ' ' + this.lastname + ' ' + suffix);
  }
};

let Person2 = {
  firstname: "Tulsi",
  lastname: "Lukhi"
};

let fullnameOfP2 = Person1.fullname.bind(Person2, "Hi,", "Welcome!");
fullnameOfP2();
```

Using Apply:

```
let Person1 = {  
  firstname: "Parangi",  
  lastname: "Rathod",  
  fullname: function(prefix, suffix) {  
    console.log(prefix + ' ' + this.firstname + ' ' + this.lastname + '  
' + suffix);  
  }  
};  
  
let Person2 = {  
  firstname: "Tulsi",  
  lastname: "Lukhi"  
};  
  
Person1.fullname.apply(Person2, ["Hi", "Welcome!"]);
```

Using Call:

```
let Person1 = {  
  firstname: "Parangi",  
  lastname: "Rathod",  
  fullname: function(prefix, suffix) {  
    console.log(prefix + ' ' + this.firstname + ' ' + this.lastname + '  
' + suffix);  
  }  
};
```

```
}  
};
```

```
let Person2 = {  
  firstname: "Tulsi",  
  lastname: "Lukhi"  
};
```

```
Person1.fullname.call(Person2, "Hi, ", "Welcome!");
```

Prototype:

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

```
Array.prototype.factors = function() {  
  let number = this[0];  
  let factors = [];  
  
  for (let i = 1; i <= number; i++) {  
    if (number % i === 0) {  
      factors.push(i);  
    }  
  }  
  
  return factors;  
};  
  
let numArray = [12];  
  
console.log(numArray.factors());
```