closure

→ A closure is a function that comes bundled with its surrounding ~~state~~ state (the lexical environment)

Ex:
```
function x()
    {
    var a=7;
    function y()
        {
        console.log(a);
        }
    return y;
    }
Var z = x();
console.log(z);
```

o/p:
```
y()
    {console.log(a);
    }
```

Now, z();

o/p: 7

→ Even when we return a function, it still remembers the environment it was created in, so it can access those variables.

→ If we write 'return function(){...}', we are returning the function itself, not its value. The function needs to be called to return a value.

```
function x()
    {
      var a=7;
      function y()
        {
          console.log(a);
        }
        a=100;
        return y;
    }
var z=x();
console.log(z);
z();
```

o/p: y()
{ console.log(a);
}
100

→ In this code, the inner function y() has access to the variable 'a' because of a closure.

→ The important thing to understand is that the closures captures the reference to 'a', not the value.

→ This means any changes to 'a' before the close is used will be reflected when I call the function later.

→ In my case, I first set 'a' to 7, but before returning the function y(), I change 'a' to 100.

→ So, when I call the returned function later, it logs 100 because that was the final value of 'a' before the closure was invoked.

In javascript, a closure is created when a function is defined inside another function, and the inner function has access to the outer function's variables, even after the outer function has finished executing.