

CLOSURES

A closure is the combination of a function bundled together (enlosed) with references to its surrounding state (the lexical environment).

In other words, a closure gives a function access to its outer scope.

- In JS, closures are created everytime a function is created, at function creation time.

- ① In JS, we can assign functions to a variable.

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

- ② In JS, we can pass a function inside a function as a parameter.

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

- ③ We can even return function.

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

```
④ function x(){  
  var a = 7;  
  function y(){  
    console.log(a);  
  }  
  return y;  
}  
var z = x();  
console.log(z);  
z();
```

```
>> y() {  
  console.log(a);  
}  
7
```

⇒ Function when they are returned from other functions, they still maintain their lexical scope.

Function along with its lexical scope → the closure was returned.

```
⑤ function x() {  
  var a = 7;  
  return function y() {  
    console.log(a);  
  };  
}  
  
var z = x();  
console.log(z);  
z();
```

Some cool developers try to do this, but this some .

Some nitty gritty's

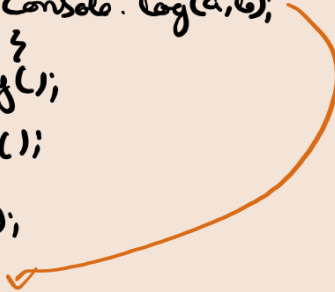
```
① function x() {  
  var a = 7;  
  function y() {  
    console.log(a);  
  }  
  a = 100;  
  return y;  
}  
  
var z = x();  
console.log(z);  
z();
```

```
>> 100
```

The value of `a` is not returned. Reference to `a` is returned.
The value of `a` doesn't persist. The reference persists.

②

```
function z(){  
  var b = 900;  
  function x(){  
    var a = 7;  
    function y(){  
      console.log(a, b);  
    }  
    y();  
    x();  
  }  
  z();  
}
```



⇒ This forms a closure with its parents' parents and its parent.

⇒ If we could have returned `y` outside somewhere, it would still have retained values of `a` & `b`.

③ Common uses of closures:-

- ① Module Design Pattern.
- ② Currying
- ③ Functions like once.
- ④ memorize
- ⑤ maintaining state in asyn world.
- ⑥ setTimeout.
- ⑦ Iterators
- ⑧ nonynore.