12 video

Closures in JS

A function along with its lexical scope forms a closure.

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

O/P 7

Scope
local.
this: window
closure(x)
a: 7

→ The function y() is wind to variable of x(). Hence y() is forming a closure.

→ A function along lexical environment of its parents is known as closure.

→ A function bundled together with reference to its lexical environment.

→ JS allow us to pass a function as argument in a function.

```
function x(z) {
  console.log(z);
}
function y() {
  var z = 10;
  x(y);
}
```

O/P 10

→ you can pass a function into a variable.

```
var a = x();
```

JS is synchronous language.

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

var z = x();
console.log(z)
```

immediately after return x() will go out of call stack & hence y also and 'a' will also not exist.

But;

z() → o/p : 7

closure comes into action

due to this y() will remember its lexical environment.

When returned y() it also return its lexical environment. i.e. it return complete closure (it's also remembers the variable reference)

function + variables + LS

```
function y() {
  console.log(a);
}
return y;
```

==

```
return function y() {
  console.log(a);
}
```

→

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

o/p → 100 (Not 7)

function y() return reference of 'a' [not value] at 'a']

↓

Closure Return reference to variables.

→

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

Scope

local

this: window

closure(x)

a: 7

closure(z)

b: 900

debugged.

Uses of closures

module design pattern
currying [function currying]
function like once ← Runs only once
memoize
maintaining state in async world
settimeout
Iterator
and many more
data hiding & encapsulation

⇒ A closure is the combination of function bundled together (enclosed) with references to its surrounding state (the lexical environment)
that means each function has access to its surrounding environment (which includes variables & function).