episode 09

Block, Scope and shadowing in javascript

- block is also called as compound statement.
- block is defined by curly braces. { }
- we group multiple statements together in a block so that we can use it where javascript expects one statement.

```
- if (true) {
      var a=10;
      console.log(a);
  }
```

block ⟹ 10

[let and const are block scoped]

- block scope means what all variables and functions we can access inside block.

```
{
  var a = 10;        <-- Global scoped
  let b = 20;        > block scoped
  const c = 30;      > block scoped
  console.log(a);    => 10
  console.log(b);    => 20
  console.log(c);    => 30
}                    => 10
console.log(a);      => reference error : b is not defined
console.log(b);
console.log(c);
```

EEEOE

```
var a = 100;
{
  var a = 10;        shadowing
  let b = 20;        => 10   [ Here var a = 10 shadows var a = 100 ]
  const c = 30;
  console.log(a);
}
```

```
var a = 100;
{
  var a = 10;        => 10
  let b = 20;
}
console.log(a);
```

Shadowing

Page No.
Date

- `let b=100;` ← This 'b' has its scope global

```
{
    var a=10;
    let b=20;
    const c=30;
    console.log(b);
}
console.log(b);
```

This 'b' has its block scope (in our case local)

This 'b' is shadowing upper 'b's value

⟹ 20
100

⟹ i.e. let b=20; is shadowing let b=100;

- `const c=100;`

```
{
    const c=10;
    console.log(c);
}
console.log(c);
```

⟹ 10
100

[const c=10 shadows const c=100]

⟹ 'let' and 'const' same things applied

- `const c=100;`

```
function x(){
    const c=10;
    console.log(c);
}
x();
console.log(c);
```

⟹ 10
100

shadowing example

const c=10 shadows const c=100

we can shadow let using let
we cannot shadow let using var
we can shadow var using var
we can shadow var using let

Page No.
Date

- let a = 20;
{
  var a = 20; ⇒ syntaxerror : identifier 'a' has already been declared
}

illegal shadowing

EEEOE

- let a = 20;
{
  let a = 10; ⇒ valid shadowing
}

- var a = 10;
{
  var a = 20; ⇒ valid shadowing
}

- var a = 10;
{
  let a = 20; ⇒ valid shadowing
}

- let a = 20;
function x() {
  var a = 10; ⇒ This is valid because var a = 10; has its boundary or block scope
}

- const a = 10;
{
  const a = 20; ⇒ valid
}

[block scope also follows lexical scope]
[lexical scope works in same way inside block also]

Page No.
Date

-
```
const a=20;
{
    const a=100;
    {
        const a=200;
    }
    console.log(a);
}
```
⟹ 100 [lexical scope]
[this 'a' will get access from its nearest 'a']

-
```
const a=20;
{
    const a=100;
    {
        const a=200;
        console.log(a);
    }
}
```
⟹ 200 [lexical scope]
[this 'a' will get access from its nearest 'a']

-
```
const a=20;
{
        const a=100;
    {
        console.log(a);
    }
}
```
⟹ 100 [lexical scope]
[this 'a' will get access from its nearest 'a']

-
```
const a=20;
{
    const a=100; {
        const a=200;
    }
}
console.log(a);
```
⟹ 20 [lexical scope]
[this 'a' will get access from its nearest 'a']

(Normal function)

↓

- All scope rules which work on functions are exactly same on arrow function also.