Koʻrinish doirasi va tutashish (Scope, Execution Context, Closure) 9-dars. Koʻrinish doirasi va tutashish (Scope, Execution Context, Closure) Scope

JavaScript Variable Scope

- 1. Global Scope
- 2. Local Scope

```
// program to print a text
let a = "hello";
function greet () {
    console.log(a);
}
greet(); // hello
```

Local Scope

```
// program showing local scope of a variable
let a = "hello";

function greet() {
    let b = "World"
    console.log(a + b);
}

greet();
console.log(a + b); // error
```

let is Block Scoped

```
// program showing block-scoped concept
// global variable
let a = 'Hello';

function greet() {

    // local variable
    let b = 'World';

    console.log(a + ' ' + b);

if (b = 'World') {

    // block-scoped variable
    let c = 'hello';
```

```
console.log(a + ' ' + b + ' ' + c);
}

// variable c cannot be accessed here
console.log(a + ' ' + b + ' ' + c);
}

greet();
```

Execution Context

```
Global Execution Context

window: global object

this: window
```

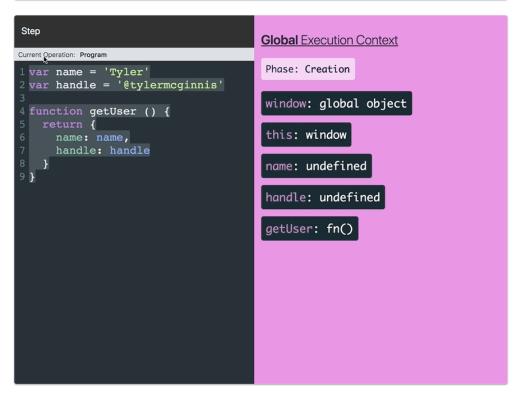
Global Execution Context
Creation Phase (Web Browser)
Global Object: window
this: window
x: undefined
timesTen: function(){...}
y: undefined

```
1 var name = 'Tyler'
2 var handle = '@tylermcginnis'
3
4 function getUser () {
5   return {
6    name: name,
7    handle: handle
8  }
9 }

this: window
name: undefined
handle: undefined
getUser: fn()
```

```
1 var name = 'Tyler'
2 var handle = '@tylermcginnis'
4 function getUser () {
5   return {
6    name: name,
7    handle: handle
8  }
9 }

This: window
name: "Tyler"
handle: "@tylermcginnis"
getUser: fn()
```



- 1. Visualize the code yourself
- 2. Visualize the code yourself
- 3. Visualize the code yourself
- 4. Visualize the code yourself
- 5. Visualize the code yourself
- 6. <u>Visualize the code yourself</u>

Closure

In JavaScript, a <u>function</u> can also contain another function. This is called a nested function. For example,

```
// nested function example

// outer function
function greet(name) {
```

```
// inner function
function displayName() {
    console.log('Hi' + ' ' + name);
}

// calling inner function
    displayName();
}

// calling outer function
greet('John'); // Hi John
```

JavaScript Closures

In JavaScript, closure provides access to the outer scope of a function from inside the inner function, even after the outer function has closed. For example,

```
// javascript closure example

// outer function
function greet() {

    // variable defined outside the inner function
    let name = 'John';

    // inner function
    function displayName() {

        // accessing name variable
        return 'Hi' + ' ' + name;

    }

    return displayName;
}

const g1 = greet();
console.log(g1); // returns the function definition
console.log(g1()); // returns the value
```

```
// closure example

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

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

console.log(multiply3); // returns calculate function definition
console.log(multiply3()); // NaN

console.log(multiply3(6)); // 18
console.log(multiply4(2)); // 8
```

```
function sum() {
    let a = 0;
    function increaseSum() {

        // the value of a is increased by 1
        return a = a + 1;
    }
    return increaseSum;
}

let x = sum();
let a = 5;
console.log(x()); // 1
console.log(x()); // 2
console.log(a); // 5
```