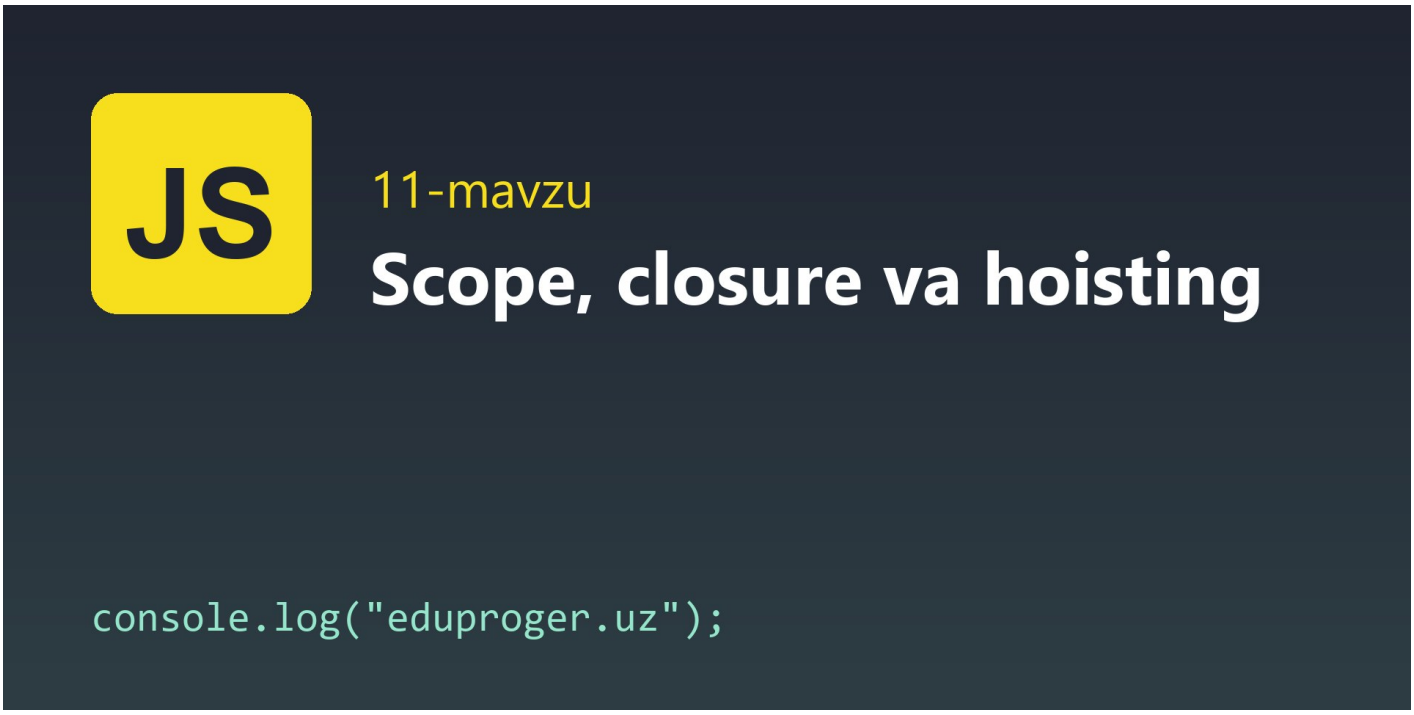
11. Scope, closure va hoisting



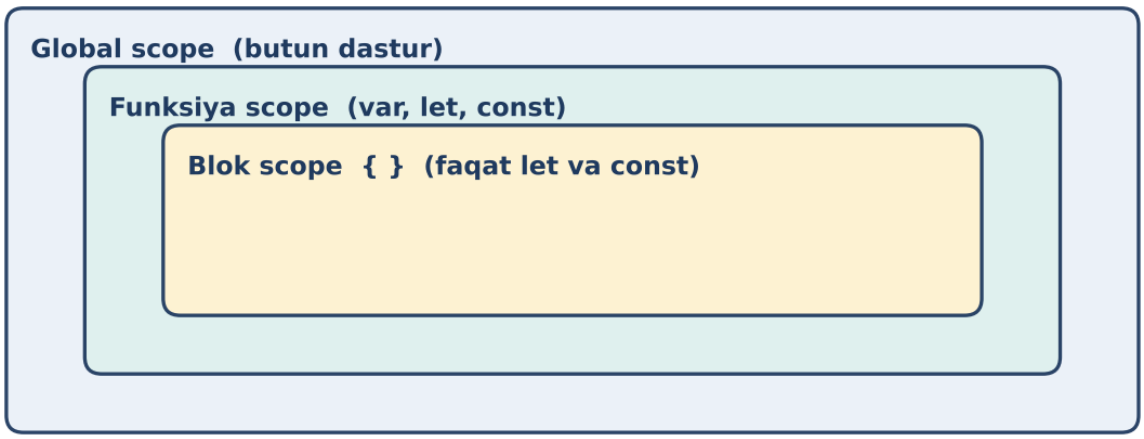
Scope (qamrov) — o'zgaruvchi **qayerda ko'rinishi** haqidagi qoidalar. Bu dars scope ni, `var`/`let`/`const` farqini va hoisting bilan tanishuvni noldan tushuntiradi.

> **Bu darsda nimani o'rganasiz**
>
> Scope nima va nima uchun kerak.
>
> Global, funksiya va blok scope.
>
> `var`, `let`, `const` farqi.
>
> Lokal va global o'zgaruvchilar to'qnashuvi.
>
> Hoisting nima ekanligini sodda tushunish.

1.1. Scope nima?

Uyingizni tasavvur qiling. **Yotoqxonadagi** narsalarni faqat yotoqxonada ishlatasiz; **mehmonxonadagi** narsalar esa umumiy. Eshik yopiq bo'lsa, ichkaridagini tashqaridan ko'ra olmaysiz, lekin ichkaridan **tashqariga** qarasangiz, hamma narsa ko'rinadi.

O'zgaruvchilar ham shunday. **Scope** — o'zgaruvchining "yashash joyi": u faqat shu joyda ko'rinadi va ishlatiladi.



1-rasm. Uch xil scope

1-jadval. Scope turlari

Scope	Qayerda yaratiladi	Qayerda ko'rinadi
Global	Hamma funksiya va blokdan tashqarida	Butun dasturda
Funksiya	Funksiya `{ }` ichida	Faqat shu funksiyada
Blok	`if`, `for`, `{ }` ichida (`let`, `const`)	Faqat shu blokda

1.2. Funksiya scope

```
const global = "men global";           // hamma joyda ko'rinadi

function funk() {
  const lokal = "men lokal";           // faqat shu funksiyada
  console.log(global);                 // tashqarini ko'ra oladi
  console.log(lokal);
}
funk();

try {
  console.log(lokal);                  // tashqaridan ichkarini ko'rib bo'lmay
} catch (e) {
  console.log(e.name + ": " + e.message);
}
```

Natija:

```
men global
men lokal
ReferenceError: lokal is not defined
```

Qoida

> **Ichki** scope **tashqi** scope dagi o'zgaruvchilarni ko'radi. **Tashqi** scope esa ichki scope dagilarni **ko'rmaydi**. Xuddi yotoqxonadan mehmonxona ko'rinadi, aksincha emas.

1.3. Blok scope

`{ }` ichida (masalan, `if` yoki `for` ichida) `let` yoki `const` bilan e'lon qilingan o'zgaruvchi **faqat shu blokda** yashaydi.

```
if (true) {
  let ichki = "faqat blok ichida";
  const yana = 42;
  console.log(ichki, yana);
}

try {
  console.log(ichki);
} catch (e) {
  console.log(e.name + ": " + e.message);
}

for (let i = 0; i < 3; i++) {
  // i faqat siklda yashaydi
}
try { console.log(i); } catch (e) { console.log("i sikldan tashqarida yo'q");
```

Natija:

```
faqat blok ichida 42
ReferenceError: ichki is not defined
i sikldan tashqarida yo'q
```

1.4. var, let va const

Eski JavaScriptda faqat `var` bor edi. 2015-yildan (ES6) `let` va `const` paydo bo'ldi. Hozir **`var` ishlatilmaydi**; sabablarini ko'ramiz.

2-jadval. var, let, const

	var	let	const
Scope	**Funksiya**	**Blok**	**Blok**
Qayta tayinlash	Mumkin	Mumkin	**Mumkin emas**
Qayta e'lon qilish	Mumkin (xavfli)	Mumkin emas	Mumkin emas
E'londan oldin ishlatish	`undefined` (hoisting)	Xato (TDZ)	Xato (TDZ)
Tavsiya	Ishlatmang	O'zgaradigan qiymat uchun	**Standart tanlov**

```
// var — blok scope ga ahamiyat bermaydi
if (true) {
  var a = "var sizib chiqdi";
  let b = "let chiqmaydi";
}
console.log(a);
try { console.log(b); } catch (e) { console.log("b:", e.name); }

// var qayta e'lon qilishga ruxsat beradi — tasodifan ustidan yozish xavfi
var x = 1;
var x = 2;
console.log(x);

// const o'zgarmas bog'lanish
const pi = 3.14;
try { pi = 3; } catch (e) { console.log(e.name + ": " + e.message); }
```

Natija:

```
var sizib chiqdi
b: ReferenceError
2
TypeError: Assignment to constant variable.
```

> **Oltin qoida**
>
> Doim `const` dan boshlang. Qiymat o'zgarishi kerakligi aniq bo'lsa, `let` ga o'zgartiring. `var` dan qoching.

1.5. Bir xil nomli o'zgaruvchilar

Ichki scope da tashqaridagi bilan **bir xil nomli** o'zgaruvchi e'lon qilinsa, ikkalasi alohida bo'ladi. Ichkarida ichkisi "g'olib" bo'ladi:

```
let ism = "global Ali";

function salom() {
  let ism = "lokal Vali";       // alohida o'zgaruvchi
  console.log(ism);
}

salom();
console.log(ism);               // tashqisi o'zgarmadi
```

Natija:

```
lokal Vali
global Ali
```

Tasodifiy global o'zgaruvchi

O'zgaruvchini `let/const/var` **siz** yozsangiz, u (qat'iy rejimda bo'lmasa) **global** bo'lib qoladi. Bu — xavfli xato:

```
function xato() {
  tasodifiy = 100;     // e'lon qilmasdan! global bo'lib qoladi
}
xato();
console.log(tasodifiy);

function toGri() {
  const yaxshi = 100;  // lokal
}
toGri();
console.log(typeof yaxshi);
```

Natija:

```
100
undefined
```

> **Qat'iy rejim**
>
> Faylning boshiga `"use strict";` yozilsa, e'lon qilinmagan o'zgaruvchiga qiymat berish **xato** beradi. ES modullar va `class` ichida qat'iy rejim avtomatik yoqilgan.

1.6. Hoisting — "ko'tarilish" (tanishuv)

JavaScript kodni bajarishdan oldin uni bir marta **ko'zdan kechiradi** va barcha e'lonlarni xotiraga oladi. Shuning uchun ba'zi narsalarni e'londan **oldin** ham ishlatish mumkin:

```
// Funksiyani e'londan OLDIN chaqirish — ishlaydi
salom();
function salom() {
  console.log("Salom!");
}

// var e'londan oldin — undefined (xato emas, lekin qiymat yo'q)
console.log(son);
var son = 5;
console.log(son);
```

Natija:

```
Salom!
undefined
5

// let/const e'londan oldin — XATO
try {
  console.log(yosh);
} catch (e) {
  console.log(e.name + ": " + e.message);
}
let yosh = 20;
```

Natija:

```
ReferenceError: Cannot access 'yosh' before initialization
```

3-jadval. Hoisting natijalari

E'lon turi	E'londan oldin ishlatilsa
`function nom() {}`	**Ishlaydi** (to'liq ko'tariladi)
`var x = 5`	`undefined` beradi
`let x = 5`, `const x = 5`	`ReferenceError` (TDZ)

> **Amaliy maslahat**
>
> Hoisting haqida bilish kerak, lekin unga **tayanmang**. O'zgaruvchilarni ishlatishdan **oldin** e'lon qiling; funksiyalarni ham ishlatishdan oldin yozish o'qishni osonlashtiradi.

1.7. Kod izi

```
const x = 10;
function f() {
  const x = 20;
  if (true) {
    const x = 30;
    console.log(x);
  }
  console.log(x);
}
f();
console.log(x);
```

4-jadval. Kod izi

Qator	Qaysi `x`	Chiqadi
`console.log(x)` blok ichida	Blokdagi `x = 30`	30
`console.log(x)` funksiyada	Funksiyadagi `x = 20`	20
`console.log(x)` tashqarida	Global `x = 10`	10

```
const x = 10;
function f() {
  const x = 20;
  if (true) {
    const x = 30;
    console.log(x);
  }
  console.log(x);
}
f();
console.log(x);
```

Natija:

```
30
20
10
```

1.8. Mini loyihalar

scope_loyiha.js

```
// 1) Hisoblagich funksiyasi — lokal o'zgaruvchi har safar yangidan
function sanoq() {
  let n = 0;
  n++;
  return n;
}
console.log(sanoq(), sanoq(), sanoq());     // har safar 1

// 2) Global hisoblagich — xavfli, lekin ishlaydi
let jami = 0;
function qosh() { jami++; return jami; }
console.log(qosh(), qosh(), qosh());

// 3) Blok ichida vaqtinchalik o'zgaruvchi
{
  const vaqtincha = "yo'qoladi";
  console.log(vaqtincha);
}
console.log(typeof vaqtincha);

// 4) Sikl o'zgaruvchisi
for (let i = 1; i <= 3; i++) {
  const kvadrat = i * i;
  console.log(i, kvadrat);
}
```

Natija:

```
1 1 1
1 2 3
yo'qoladi
undefined
1 1
2 4
3 9
```

1.9. Eng ko'p uchraydigan xatolar

5-jadval. Tipik xatolar

Xato	Sababi	To'g'ri yozuv
Blokda e'lon qilingan `let` ni tashqarida ishlatish	Blok scope	O'zgaruvchini tashqarida e'lon qiling

Xato	Sababi	To'g'ri yozuv
E'lon qilmasdan qiymat berish	Tasodifiy global	`const x = ...` / `let x`
`let` ni e'londan oldin ishlatish	TDZ — `ReferenceError`	Avval e'lon qiling
`const` ni qayta tayinlash	`const` — o'zgarmas bog'lanish	`let` ishlating
Bir nomni ikki marta `let` bilan e'lon qilish	Takroriy e'lon taqiqlangan	Bitta e'lon, keyin qayta tayinlash
`var` ni sikl ichida ishlatish	Funksiya scope: sikldan sizib chiqadi	`let` ishlating

1.10. Mashqlar

1. Funksiya ichida o'zgaruvchi e'lon qiling va uni tashqaridan chop etishga urinib, xatoni ko'ring.
2. `if` blokida `let` va `var` bilan o'zgaruvchi e'lon qilib, farqni namoyish qiling.
3. Bir xil nomli global va lokal o'zgaruvchi yarating, qaysi biri chiqishini oldindan ayting.
4. `const` massivga `push` qilib bo'ladimi? Sinab ko'ring va sababini tushuntiring.
5. Hoisting: `var` va `let` uchun e'londan oldin chop etishga urining, farqini yozing.
6. Funksiyani e'londan oldin chaqiring.
7. `for (var i...)` va `for (let i...)` dan keyin `i` ni chop eting.

Javob (4)

Ha, `push` mumkin: `const` massivning **o'zini** emas, o'zgaruvchiga **bog'lanishni** qulflaydi.

1.11. O'zingizni tekshiring

6-jadval. Qisqa test

№	Savol	Javob
1	Scope nima?	O'zgaruvchi ko'rinadigan joy
2	Uch xil scope?	Global, funksiya, blok
3	Ichki scope tashqarini ko'radimi?	Ha
4	Tashqi ichkarini ko'radimi?	Yo'q
5	Qaysi biri blok scope ga ega?	`let` va `const`
6	`var` ning scope i?	Funksiya
7	E'londan oldin `let` ishlatilsa?	`ReferenceError`

№	Savol	Javob
8	Qaysi funksiya e'londan oldin chaqiriladi?	`function` declaration

6 va undan ko'p to'g'ri javob — **2-darajaga** tayyorsiz.

1.12. Kichik lug'at

7-jadval. Lug'at

Atama	Sodda ma'nosi
Scope	O'zgaruvchi ko'rinadigan joy
Global	Hamma joyda ko'rinadigan
Lokal	Faqat ma'lum joyda ko'rinadigan
Hoisting	E'lonlarning kod bajarilishidan oldin xotiraga olinishi
TDZ	`let/const` e'londan oldingi "o'lik zona"

Xulosa

Ichki scope tashqarini ko'radi, aksincha emas. `let/const` — blok scope, `var` — funksiya scope. `const` dan boshlang. Hoisting tufayli funksiyalar e'londan oldin ishlaydi, `let/const` esa xato beradi.

Chuqurroq o'rganmoqchimisiz?

Bu mavzuning o'rta va murakkab darajalari JavaScript adaptiv kursida — daraja testi sizga mos materialni o'zi tanlaydi, oxirida sertifikat.