15. Asinxron dasturlash



Dasturlar ba'zan **kutishga** majbur: server javobi, fayl o'qilishi, taymer. Asinxron dasturlash — kutish paytida dastur **qotib qolmasligi** uchun usullar. Bu dars uni noldan, restoran misoli bilan tushuntiradi.

> **Bu darsda nimani o'rganasiz**
>
> Sinxron va asinxron kod farqi.
>
> `setTimeout` va `setInterval`.
>
> Callback va callback hell.
>
> Promise: nima va nima uchun.
>
> `then`, `catch`, `finally`.
>
> `async/await` bilan tanishuv.

1.1. Sinxron kod: navbat bilan

Hozirgacha yozgan kodingiz **sinxron** edi: har qator oldingisi tugagach bajariladi. Bu oddiy va tushunarli:

```
console.log("1-qadam");
console.log("2-qadam");
```

```
console.log("3-qadam");
```

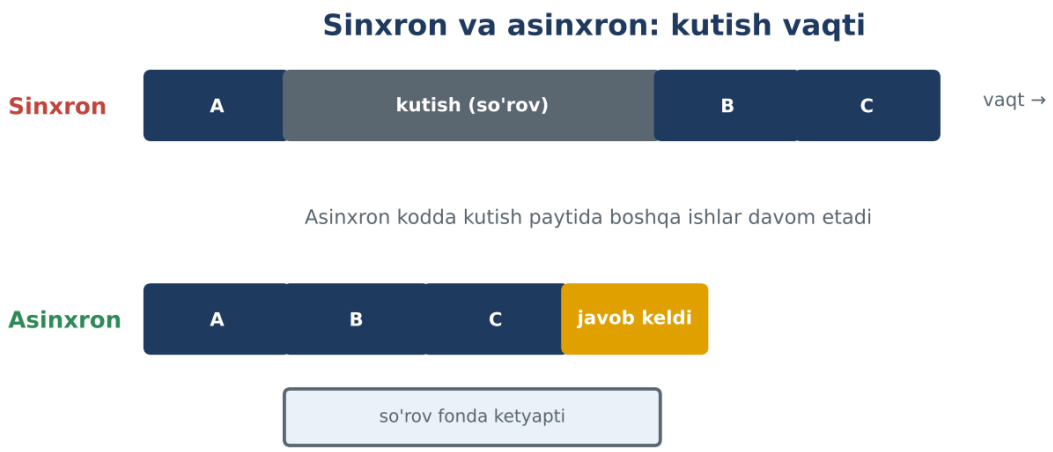
Natija:

```
1-qadam
2-qadam
3-qadam
```

Lekin ba'zi ishlar **uzoq davom etadi**: serverdan ma'lumot olish (1–2 soniya), faylni o'qish. Agar JavaScript shuni kutib tursa, **butun sahifa qotib qoladi**: tugmalar bosilmaydi, animatsiya to'xtaydi. Yechim — **asinxron** kod.

1.2. Restoran misoli

Sinxron restoran: ofitsiant bitta mijozning ovqati tayyor bo'lguncha oshxona oldida kutib turadi, boshqa mijozlarga qaramaydi. **Asinxron restoran**: ofitsiant buyurtmani oshxonaga beradi va **boshqa mijozlarga xizmat ko'rsatadi**; ovqat tayyor bo'lganda esa uni olib boradi.



1-rasm. Sinxron va asinxron

JavaScript ham shunday: uzoq ishni "oshxonaga" (brauzer yoki Node.js) topshiradi, o'zi keyingi kodni davom ettiradi, ish tugagach esa natijani qayta ishlaydi.

1.3. setTimeout va setInterval

`setTimeout(funksiya, ms)` — funksiyani **ms millisekunddan keyin** bir marta ishga tushiradi. U kodni **to'xtatmaydi**: ro'yxatdagi keyingi qatorlar darrov bajariladi.

```
console.log("1. Boshlandi");

setTimeout(() => {
  console.log("3. 100 ms dan keyin");
}, 100);
```

```
console.log("2. Davom etdi (kutmadi!)");
```

Natija:

```
1. Boshlandi
2. Davom etdi (kutmadi!)
3. 100 ms dan keyin
```

> **Eng ko'p uchraydigan tushunmovchilik**
>
> `setTimeout(fn, 0)` ham funksiyani **darrov** ishlatmaydi: u hamma sinxron kod tugagandan **keyin** ishlaydi. Natijalar tartibi yozilgan tartibdan farq qilishi mumkin.

```
console.log("A");
setTimeout(() => console.log("B (0 ms, lekin keyin)"), 0);
console.log("C");
```

Natija:

```
A
C
B (0 ms, lekin keyin)
```

setInterval va to'xtatish

```
let n = 0;
const id = setInterval(() => {
  n++;
  console.log("Tik", n);
  if (n === 3) {
    clearInterval(id);          // to'xtatish shart!
    console.log("To'xtadi");
  }
}, 30);

// setTimeout ni ham bekor qilish mumkin
const t = setTimeout(() => console.log("bu chiqmaydi"), 1000);
clearTimeout(t);
```

Natija:

```
Tik 1
Tik 2
Tik 3
To'xtadi
```

1.4. Callback va "callback hell"

Eng qadimiy usul: asinxron ish tugaganda chaqiriladigan **funksiyani (callback)** berib qo'yish. Hozir ham ishlatiladi, lekin ketma-ket ishlar ko'payganda kod **piramida**ga aylanadi — **callback hell** ("callback do'zaxi").

```
function ovqatTayyorla(nom, ms, tayyor) {
  console.log(nom, "tayyorlanmoqda...");
  setTimeout(() => tayyor(nom + " tayyor"), ms);
}

// Ketma-ket 3 ta ish: har biri oldingisini kutadi
ovqatTayyorla("Sho'rva", 20, (n1) => {
  console.log(n1);
  ovqatTayyorla("Palov", 20, (n2) => {
    console.log(n2);
    ovqatTayyorla("Choy", 20, (n3) => {
      console.log(n3);
      console.log("Hammasi tayyor!");
    });
  });
});
```

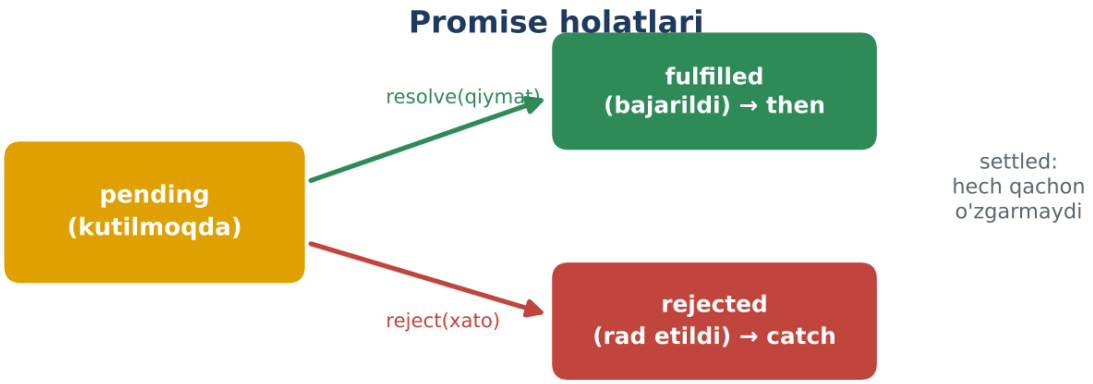
Natija:

```
Sho'rva tayyorlanmoqda...
Sho'rva tayyor
Palov tayyorlanmoqda...
Palov tayyor
Choy tayyorlanmoqda...
Choy tayyor
Hammasi tayyor!
```

Ichma-ich kod o'qishni qiyinlashtiradi, xatolarni boshqarish murakkablashadi. Promise aynan shu muammoni hal qiladi.

1.5. Promise: va'da

Promise (va'da) — hali tayyor bo'lmagan, lekin **kelajakda** tayyor bo'ladigan natijani ifodalovchi obyekt. Hayotiy misol: **buyurtma cheki** — ovqat hali yo'q, lekin chek bor; ovqat tayyor bo'lganda yoki oshxonada muammo chiqqanda (tugab qolgan) sizga xabar beriladi.



2-rasm. Promise holatlari

1-jadval. Promise holatlari

Holat	Ma'nosi	Keyin
pending	Kutilmoqda: hali natija yo'q	Yo bajariladi, yo rad etiladi
fulfilled	Muvaffaqiyatli bajarildi (qiymat bor)	`.then(...)` ishlaydi
rejected	Xato bilan rad etildi	`.catch(...)` ishlaydi

```
const vada = new Promise((resolve, reject) => {
  setTimeout(() => {
    const muvaffaqiyat = true;
    if (muvaffaqiyat) resolve("Ovqat tayyor!");     // fulfilled
    else reject(new Error("Ovqat tugab qoldi"));   // rejected
  }, 50);
});

console.log(vada);                                   // hozircha pending

vada
  .then((natija) => console.log("Muvaffaqiyat:", natija))
  .catch((xato) => console.log("Xato:", xato.message))
  .finally(() => console.log("Tugadi (finally)"));
```

Natija:

```
Promise { <pending> }
Muvaffaqiyat: Ovqat tayyor!
Tugadi (finally)
```

1.6. then, catch, finally

2-jadval. Promise metodlari

Metod	Qachon ishlaydi
`.then(fn)`	Promise **bajarilganda** (natijani `fn` oladi)
`.catch(fn)`	Promise **rad etilganda** yoki zanjirda xato bo'lganda
`.finally(fn)`	**Doim**: natija qanday bo'lishidan qat'iy nazar (tozalash uchun)

```
const kut = (ms, qiymat) => new Promise((resolve) => setTimeout(() => resolve(

kut(30, 5)
  .then((x) => x * 2)               // 10
  .then((x) => x + 1)               // 11
  .then((x) => console.log("Natija:", x));

const xato = () => new Promise((_, reject) => setTimeout(() => reject(new Erro
xato()
  .then(() => console.log("bu chiqmaydi"))
  .catch((e) => console.log("Ushlandi:", e.message))
  .finally(() => console.log("finally"));
```

Natija:

```
Natija: 11
Ushlandi: buzildi
```

```
finally
```

1.7. async / await — oddiyroq yozuv

`async/await` — Promise bilan ishlashning **eng qulay** usuli. Kod sinxronga o'xshab ketadi: `await` — "bu Promise tayyor bo'lguncha **shu funksiya ichida** kut" degani.

```
const kut = (ms, qiymat) => new Promise((resolve) => setTimeout(() => resolve(
```

```
async function ovqatla() {
  console.log("Boshlandi");
  const a = await kut(30, "Sho'rva");        // bu yerda funksiya kutadi (bosh
  console.log(a);
  const b = await kut(30, "Palov");
  console.log(b);
  return "Hammasi tayyor";
}

ovqatla().then((n) => console.log(n));
console.log("Asosiy kod davom etdi");
```

Natija:

```
Boshlandi
Asosiy kod davom etdi
Sho'rva
Palov
Hammasi tayyor
```

Ikki qoida

1) `await` faqat `async` funksiya ichida ishlaydi (yoki ES modullarining eng yuqori darajasida). 2) `async` funksiya **doim Promise qaytaradi**.

Xatoni ushlash: try / catch

```
const rad = (ms) => new Promise((_, reject) => setTimeout(() => reject(new Err
```

```
async function yukla() {
  try {
    console.log("So'rov yuborildi");
    const data = await rad(30);
    console.log("Keldi:", data);
  } catch (xato) {
    console.log("Xato:", xato.message);
  } finally {
    console.log("Yuklash tugadi");
  }
}
yukla();
```

Natija:

```
So'rov yuborildi
Xato: Server javob bermadi
Yuklash tugadi
```

1.8. Kod izi

```
console.log("1");
setTimeout(() => console.log("2"), 0);
Promise.resolve().then(() => console.log("3"));
console.log("4");
```

3-jadval. Kod izi

Bosqich	Nima bo'ladi	Chiqadi
1	Sinxron kod ketma-ket bajariladi	1
2	`setTimeout` — keyinga qoldirildi (macrotask)	—
3	`then` — keyinga qoldirildi (microtask)	—
4	Sinxron kod davom etadi	4
5	Sinxron kod tugadi: avval microtask	3
6	Keyin macrotask (taymer)	2

```
console.log("1");
setTimeout(() => console.log("2"), 0);
Promise.resolve().then(() => console.log("3"));
console.log("4");
```

Natija:

```
1
4
3
2
```

1.9. Mini loyihalar

asinxron_loyiha.js

```
const kut = (ms, qiymat) => new Promise((resolve) => setTimeout(() => resolve(

// 1) Teskari sanash
async function sana(n) {
  for (let i = n; i > 0; i--) {
    console.log(i);
    await kut(20);
  }
  console.log("Boshladik!");
}

// 2) Soxta serverdan foydalanuvchi olish
const foydalanuvchilar = { 1: "Ali", 2: "Vali" };
function olish(id) {
```

```
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (foydalanuvchilar[id]) resolve({ id, ism: foydalanuvchilar[id] });
      else reject(new Error(`ID ${id} topilmadi`));
    }, 20);
  });
}

(async () => {
  await sana(3);
  for (const id of [1, 2, 3]) {
    try {
      const u = await olish(id);
      console.log("Topildi:", u.ism);
    } catch (e) {
      console.log("Xato:", e.message);
    }
  }
})();
```

Natija:

```
3
2
1
Boshladik!
Topildi: Ali
Topildi: Vali
Xato: ID 3 topilmadi
```

1.10. Eng ko'p uchraydigan xatolar

4-jadval. Tipik xatolar

Xato	Sababi	To'g'ri yozuv
`await` ni unutish	Natija o'rniga `Promise` obyekti keladi	`const x = await f();`
`await` ni oddiy funksiyada ishlatish	`SyntaxError`	Funksiyani `async` qiling
`setTimeout(f(), 100)`	`f` darrov chaqiriladi	`setTimeout(f, 100)`
`forEach` ichida `await`	`forEach` kutmaydi	`for...of` ishlating
`catch` yozmaslik	Xato "ushlanmagan" qoladi	`try/catch` yoki `.catch`
`clearInterval` ni unutish	Cheksiz ishlaydi	Id ni saqlab, to'xtating

```
const kut = (ms, qiymat) => new Promise((resolve) => setTimeout(() => resolve(
(async () => {
  const x = kut(10, "salom");        // await yo'q
  console.log(x);                    // Promise { <pending> }
  console.log(await x);              // await bilan natija
})();
```

Natija:

```
Promise { <pending> }
salom
```

1.11. Mashqlar

1. `setTimeout` bilan 1 soniyadan keyin "Salom" yozing.
2. A, B, C ni shunday chiqaringki, B `setTimeout(…, 0)` bilan bo'lsin: tartibini oldindan ayting.
3. `setInterval` bilan 5 tagacha sanab, to'xtating.
4. `kut(ms)` Promise funksiyasini yozing.
5. `async` funksiya: 2 marta kutib, har safar xabar yozsin.
6. Rad etiladigan Promise yarating va `.catch` bilan ushlang.
7. `try/catch/finally` bilan `async` funksiyada xatoni qayta ishlang.
8. Callback hell misolini `async/await` ga o'giring.

1.12. O'zingizni tekshiring

5-jadval. Qisqa test

№	Savol	Javob
1	Sinxron kod qanday bajariladi?	Qatorma-qator, har biri tugagach
2	`setTimeout` kodni to'xtatadimi?	Yo'q
3	Promise holatlari?	pending, fulfilled, rejected
4	Xatoni ushlash?	`.catch` yoki `try/catch`
5	`await` qayerda ishlaydi?	`async` funksiya ichida
6	`async` funksiya nima qaytaradi?	Promise
7	`finally` qachon?	Doim
8	Taymerni to'xtatish?	`clearTimeout / clearInterval`

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

Xulosa

Asinxron kod kutish paytida dasturni qotirmaydi. Callback → Promise → `async/await`: har biri oldingisidan qulayroq. `await` Promise ni kutadi, `try/catch` xatoni ushlaydi.

Chuqurroq o'rganmoqchimisiz?

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