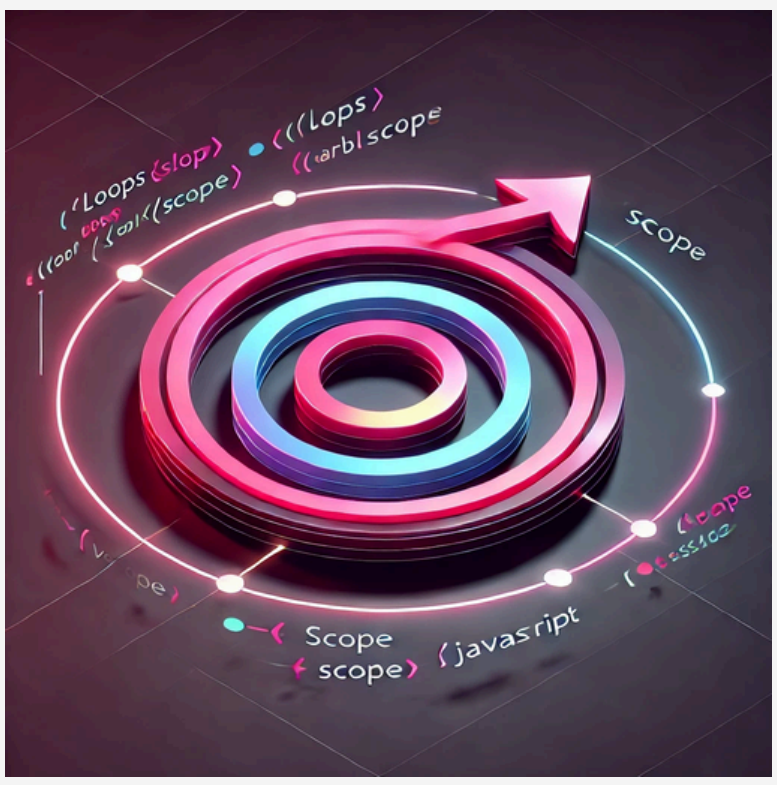
REIHANEH SADATSHOKOUHI

JavaScript Loops and Scope



امروز میخوام درمورد LOOP SCOPE
تو جاوااسکریپت با هم صحبت کنیم.

[01] به دو کد زیر و خروجی اونها دقت کنین:

```
for (let i = 0; i < 3; i++) {
  setTimeout(() => {
    console.log(i);
  }, 1000);
}
```

```
output:
0
1
2
```

```
let i = 0;
for (; i < 3; i++) {
  setTimeout(() => {
    console.log(i);
  }, 1000);
}
```

```
output:
3
3
3
```

یعنی چی ؟ چطور چنین چیزی ممکنه؟ چرا نباید خروجی این دوتا عین هم باشه؟ [02]

در حقیقت بعد از اینکه بدنه‌ی حلقه اجرا شد، این اتفاق‌ها می‌افته:

1. یه Scope جدید با متغیرهای تازه‌ای که با let تعریف شدن ساخته می‌شه.
2. مقادیر متغیرهای آخرین تکرار حلقه، دوباره به این متغیرهای جدید اختصاص داده می‌شه.
3. بعد، afterthought توی این Scope جدید ارزیابی می‌شه. (afterthought همون قسمت اخر شرطمونه ++i)

حالا با این اطلاعات جدید لوپ هارو پیمایش کنیم ببینیم درسته یا نه.

[03]

لوپ اول:

تکرار اول (i = 0):

1. مقدار i برابر با ۰ است.
2. یک setTimeout با زمان‌بندی ۱۰۰۰ میلی‌ثانیه ایجاد می‌شود.
   - این تابع به مقدار فعلی i (یعنی ۰) دسترسی دارد.
3. پس از پایان این تکرار، یک **Scope جدید** برای let i ساخته می‌شود.
4. مقدار i از ۰ به ۱ افزایش پیدا می‌کند (این افزایش بعد از اجرای بدنه اتفاق می‌افتد).

تکرار دوم (i = 1):

1. مقدار i اکنون برابر با ۱ است.
2. یک setTimeout دیگر ایجاد می‌شود.
   - این تابع به مقدار فعلی i (یعنی ۱) دسترسی دارد.
3. مجدداً یک **Scope جدید** ساخته می‌شود و مقدار i از ۱ به ۲ تغییر می‌کند.

تکرار سوم (i = 2):

...

```
for (let i = 0; i < 3; i++) {
  setTimeout(() => {
    console.log(i);
  }, 1000);
}
```

[04]

پایان حلقه:

مقدار i به ۳ رسیده و دیگر شرط حلقه (i < 3) برقرار نیست، بنابراین حلقه متوقف می‌شود.

اجرای setTimeoutها:

بعد از گذشت ۱۰۰۰ میلی‌ثانیه، توابع setTimeout به ترتیب شروع به اجرا می‌کنند.

به دلیل اینکه هر تکرار حلقه یک Scope جداگانه ایجاد کرده و let استفاده شده، هر setTimeout به مقدار صحیح i از زمان تکرار خودش دسترسی دارد.

خروجی:

پس از ۱۰۰۰ میلی‌ثانیه، خروجی به این ترتیب نمایش داده می‌شود:

```
0
1
2
```

لوپ دوم:

[05]

تفاوت کلیدی با کد قبلی:

اینجا از let i = 0 خارج از حلقه استفاده شده است. بنابراین، متغیر i در خارج از حلقه تعریف شده و **در همه تکرارهای حلقه، فقط یک Scope** برای i وجود دارد.

هر بار که حلقه تکرار می‌شود، مقدار i تغییر می‌کند، اما این مقدار در هر setTimeout تابع **مشترک** است و از همان Scope گرفته می‌شود.

```
let i = 0;
for (; i < 3; i++) {
  setTimeout(() => {
    console.log(i);
  }, 1000);
}
```

[06]

تکرار اول (i = 0):

1. مقدار i برابر با ۰ است.
2. یک setTimeout با زمان‌بندی ۱۰۰۰ میلی‌ثانیه ایجاد می‌شود.
   - اما به دلیل اینکه i در یک Scope مشترک است، تابع هنوز منتظر مانده و اجرا نشده است.
3. مقدار i به ۱ افزایش پیدا می‌کند (afterthought اجرا شده است).

تکرار دوم (i = 1):

1. مقدار i اکنون برابر با ۱ است.
2. یک setTimeout دیگر با زمان‌بندی ۱۰۰۰ میلی‌ثانیه ایجاد می‌شود.
   - همچنان تابع منتظر اجرا است و به مقدار فعلی i (که اکنون ۱ است) دسترسی دارد.
3. مقدار i از ۱ به ۲ افزایش پیدا می‌کند.

تکرار سوم (i = 2):

...

[07]

پایان حلقه:

1. مقدار i برابر با ۳ است و شرط حلقه دیگر برقرار نیست (i < 3).
2. حلقه متوقف می‌شود.

اجرای setTimeoutها:

- پس از ۱۰۰۰ میلی‌ثانیه، تمام توابع setTimeout شروع به اجرا می‌کنند.
- اما به دلیل اینکه همه توابع setTimeout به متغیر i مشترک در یک Scope دسترسی دارند و مقدار i پس از پایان حلقه برابر با ۳ است، همه توابع console.log(i) مقدار نهایی i (یعنی ۳) را چاپ خواهند کرد.

خروجی:

پس از ۱۰۰۰ میلی‌ثانیه، خروجی به این صورت خواهد بود:

```
3
3
3
```

[08] نکته ای که وجود داره اینه که در هر تکرار از ۳ قسمت شرط ، قسمت اول دیگه خونده نمیشه و انجام نمیشه. فقط اگر let تعریف شده بود اسکوپ جدا برای اون در نظر گرفته میشه.

حالا با این توضیحات ببین منطق کد های زیر رو خودت میتونی بفهمی؟

```
for (let i = 0, getI = () => i; i < 3; i++) {
  console.log(getI());
}
// Logs 0, 0, 0
```

```
for (let i = 0, getI = () => i; i < 3; i++, getI = () => i) {
  console.log(getI());
}
// Logs 0, 1, 2
```

REIHANEH SADATSHOKOUHI

JavaScript Loops and Scope

تجربه‌های شما با SCOPE در حلقه‌ها چطور بوده؟

R.SADATSHOKOUHI@GMAIL.COM