# useRef

DOM referansları, render'lar arası kalıcı değerler, useRef vs useState, ve useRef'i useEffect ile birlikte kullanmak -- basit örneklerle.

**Orta** 11 dk

Son güncelleme: 18 Ağustos 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/use-ref

---

# useRef

useEffect dersinde, component'in ekran çıktısı dışında bir şey yapmasını gördük. Bu ders, benzer bir ihtiyaca -- ekranı tetiklemeden bir değeri "hatırlamak" ya da bir DOM elementine doğrudan erişmek -- çözüm sunan bir başka hook'u, `useRef`'i anlatıyor.

## DOM Referansları (DOM References)

`useRef`'in en yaygın kullanımlarından biri, bir DOM elementine doğrudan erişmek:

```
import { useRef } from "react";

function DomReferenceExample() {
  const inputRef = useRef(null);

  function focusInput() {
    // inputRef.current is the actual DOM element itself -- we can call the
    // browser methods the input has (like focus()) directly.
    inputRef.current.focus();
  }

  return (
    <div>
      <input ref={inputRef} type="text" placeholder="Focuses when the button is clicked" />
      <button onClick={focusInput}>Focus Input</button>
    </div>
  );
}
```

`ref={inputRef}` yazarak React'e "bu elementin gerçek DOM node'unu `inputRef.current`'a koy" diyoruz. `inputRef.current`, artık gerçek bir DOM elementi -- `focus()` gibi tarayıcının kendi metotlarını doğrudan çağırabiliriz.

## Render'lar Arası Kalıcı Değerler

`useRef`'in ikinci kullanımı: bir değeri render'lar arasında "hatırlamak", ama bu değer değiştiğinde ekranı YENİDEN RENDER ETMEDEN:

```
import { useState, useRef } from "react";

function PersistentValueExample() {
  const [count, setCount] = useState(0);
  const renderCount = useRef(0);

  renderCount.current = renderCount.current + 1;
  // renderCount.current changes, but this does NOT TRIGGER A RE-RENDER --
  // it only shows the up-to-date value on the next render.

  return (
    <div>
      <p>Count: {count}</p>
      <p>This component has rendered {renderCount.current} times so far.</p>
      <button onClick={() => setCount(count + 1)}>+1</button>
    </div>
  );
}
```

`renderCount.current` her render'da artıyor ve değeri render'lar arasında korunuyor -- ama bu artış kendi başına bir re-render TETİKLEMİYOR. Değeri yalnızca bir SONRAKİ render olduğunda (başka bir sebeple, burada `count` state'i değiştiği için) güncel haliyle görüyoruz.

## useRef vs useState

`useRef` ile `useState` arasındaki temel fark şu: `useState`'i değiştirmek bir re-render TETİKLER, `useRef`'i değiştirmek TETİKLEMEZ:

```
import { useState, useRef } from "react";

function UseRefVsUseStateExample() {
  const [stateValue, setStateValue] = useState(0);
  const refValue = useRef(0);

  function incrementState() {
    setStateValue(stateValue + 1); // Updates the screen (triggers a re-render).
  }

  function incrementRef() {
    refValue.current = refValue.current + 1;
    console.log("refValue.current:", refValue.current);
    // The value really did change, but the screen is NOT UPDATED -- because
    // changing a ref does not trigger a re-render.
  }

  return (
    <div>
      <p>State: {stateValue}</p>
      <button onClick={incrementState}>Increment State</button>
      <button onClick={incrementRef}>Increment Ref (Check Console)</button>
    </div>
  );
}
```

"State'i Artır" butonuna basınca ekran güncellenir, çünkü `setStateValue` bir re-render tetikler. "Ref'i Artır" butonuna basınca değer gerçekten değişir (console'da görebilirsin), ama ekranda hiçbir şey değişmez -- çünkü ref değişikliği React'e "yeniden render et" demez. Ekranda GÖRÜNMESİ gereken bir değer state olmalı; ekranda görünmesi gerekmeyen, yalnızca "hatırlanması" gereken bir değer ref olabilir.

## useRef ve useEffect'i Birlikte Kullanmak

`useRef` ve `useEffect` sık sık birlikte kullanılır -- örneğin, "bir önceki render'daki değer neydi" sorusuna cevap vermek için:

```
import { useState, useEffect, useRef } from "react";

function PreviousValueWithRefExample() {
  const [count, setCount] = useState(0);
  const previousCountRef = useRef(undefined);

  useEffect(() => {
    // AFTER every render, we save the current value into the ref -- so we
    // can read it as the "previous value" on the next render. We use a ref
    // instead of state here because we want to record this without
    // triggering another render.
    previousCountRef.current = count;
  });

  return (
    <div>
      <p>
        Current: {count}, Previous: {previousCountRef.current}
      </p>
      <button onClick={() => setCount(count + 1)}>+1</button>
    </div>
  );
}
```

Her render'dan sonra çalışan `useEffect`, mevcut `count` değerini `previousCountRef`'e kaydediyor. Bir sonraki render'da, bu ref hâlâ BİR ÖNCEKİ render'daki değeri taşıyor -- çünkü ref güncellemesi kendi başına yeni bir render tetiklemiyor, güncelleme ancak `count` state'i değiştiğinde gerçekleşen render'ın İÇİNDE oluyor.

## Özet ve Terimler Sözlüğü

`useRef`, iki temel iş için kullanılır: bir DOM elementine doğrudan erişmek, ya da bir değeri render'lar arasında ekranı tetiklemeden hatırlamak. `useState`'in aksine, bir ref'i değiştirmek re-render tetiklemez -- bu yüzden ekranda görünmesi gereken veriler için `useState`, görünmesi gerekmeyen "arka plan" değerleri için `useRef` kullanılır.

### Terimler Sözlüğü

**`useRef`** — Bir DOM elementine erişmeni ya da bir değeri render'lar arasında, ekranı tetiklemeden saklamanı sağlayan hook.

**`.current`** — `useRef`'in döndürdüğü nesnenin, o an tutulan değeri taşıyan alanı.

**DOM Referansı** — Bir JSX elementinin gerçek tarayıcı DOM node'una doğrudan erişim; `ref` attribute'u ile kurulur.