# React

* <MyButton />’un büyük harf ile başladığına dikkat edin. Bu onun bir React bileşeni olduğunu belirtir.
* HTML elemanları küçük harf ile başlamak zorundayken, React bileşenleri her zaman büyük harfle başlamak zorundadır.
* Biçimlendirmenizin hepsi return kelimesi ile aynı satırda değilse, biçimlendirmenizin parantez içine almak zorundasınız:
* Sınıf bileşenler yerine fonksiyonel bileşenler kullanın.
* Projelerde genellikle tek bir stil kullanılması tercih edilir. Bu nedenle, tırnak işareti seçimi genellikle ekip tarafından belirlenen kod stiline bağlıdır.
* export default bir componentin içinde olan bir çok fonksiyondan hangisinin varsayılan olarak çalıştırılacağını belirtir.
* Biçimlendirme sözdizimine JSX denir.
* React kaynakları
  + <https://react.dev/reference/react>
  + <https://react.dev/learn>
  + React typescript cheat sheat <https://react-typescript-cheatsheet.netlify.app/docs/basic/setup>
  + <https://create-react-app.dev/>
* Pratikte, React’ta state’leri mutasyona uğratarak genellikle problemlerden “kurtulabilirsiniz”, ancak bu yaklaşım göz önünde bulundurularak geliştirilen yeni React özelliklerini kullanabilmeniz için bunu yapmamanızı şiddetle tavsiye ederiz.
* Döndürmek isteğiniz elemanları <div>...</div> veya boş <>...</> gibi ortak bir üst eleman içine koymanız gerekmektedir:
* **Prop’lar**, JSX etiketine ilettiğiniz bilgilerdir. Örneğin, className, src, alt, width ve height, bir <img>’ye aktarabileceğiniz prop’lardan bazılarıdır.
* Prop’ları değiştiremezsiniz. Etkileşimli bir bileşen için, state ayarlaması yapmanız gerekecektir.

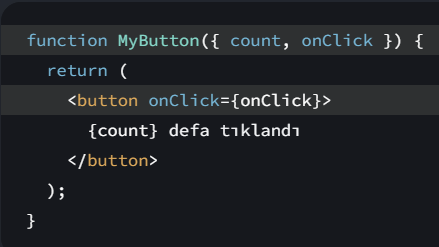


* Bileşeninizin bazı bilgileri ekran göstermek için “hatırlamasını” isteyeceksiniz. Örneğin, bir butona kaç defa tıklandığını takip etmek istiyorsunuz. Bunu yapmak için bileşeninize state ekleyin. (useState)
* const [count, setCount] = useState(0);
* useState’ten iki şey alacaksınız: mevcut state (count) ve state’i güncellemenizi sağlayan (setCount). Yaygın kullanış [something, setSomething] şeklindedir. Buton ilk defa görüntülendiğinde count değeri 0 olacaktır çünkü useState()’e 0 değerini ilettiniz.

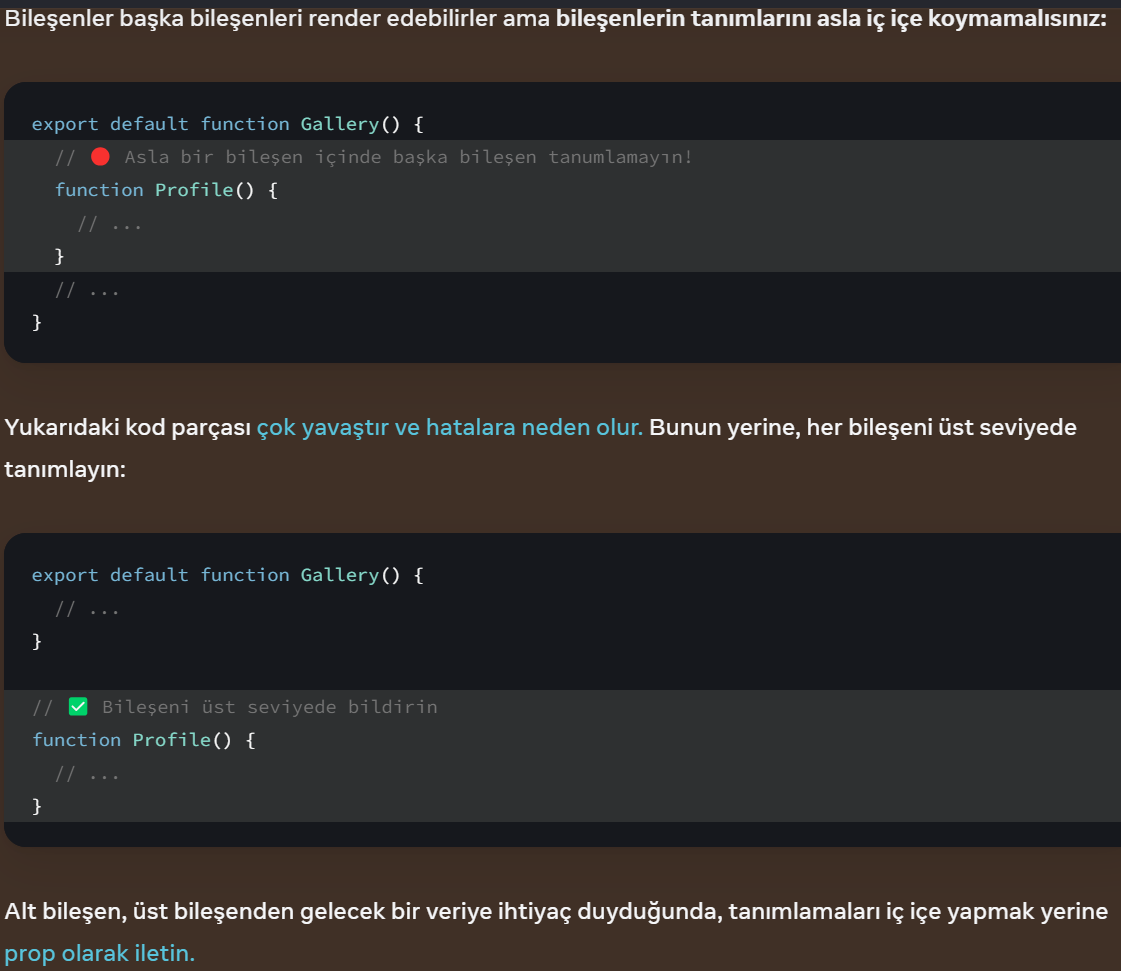


* use ile başlayan fonksiyonlara Hook denir. useState, React tarafından sağlanan bir Hook’tur.
* Eğer useState’i bir koşul ya da döngü içinde kullanmak istiyorsanız, yeni bir bileşen oluşturun ve onun içine koyun.
* Her iki MyButton bileşeninin aynı count değerini göstermesi ve aynı anda güncellenmesi için state’i bireysel butonlardan “yukarı”, hepsini içeren en yakın bileşene taşımanız gerekmektedir. ( State’i yukarı kaldırmak)
* Yeni count değeri her bir butona **prop** olarak iletilir, böylelikle hepsi yeni değeri gösterir.

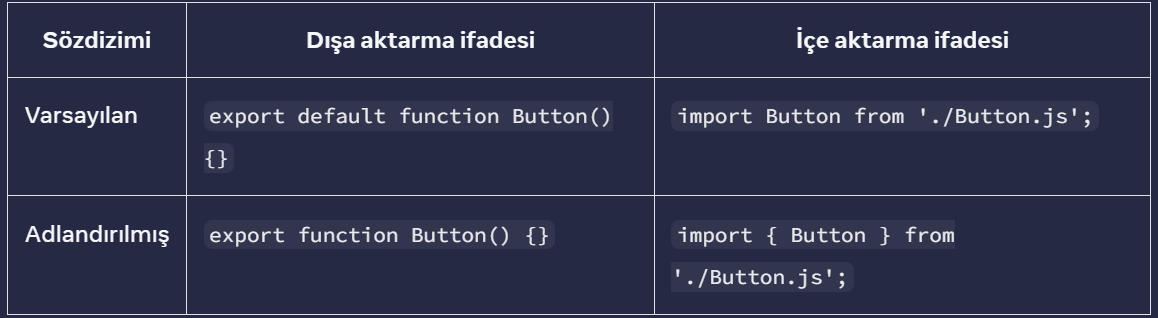




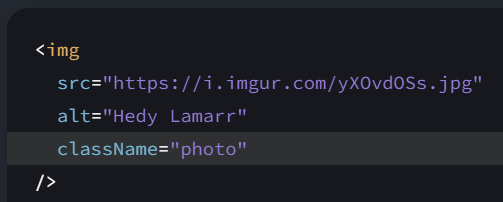
* State tanımlarken dikkat edilecek hususlar
  + Zaman içinde değişmeden mi duruyor? Eğer öyleyse, state değildir.
  + Prop’lar aracılığıyla üst bileşenden mi geliyor? Eğer öyleyse, state değildir.
  + Varolan state veya prop’lara dayalı olarak hesaplayabilir misiniz? Eğer öyleyse, kesinlikle state değildir!
* Eslint’in var olan ayarlarını kuralım
* Pretier kuralım
* useContext:
  + Bir temayı, dil ayarını veya kullanıcı bilgilerini tüm uygulama boyunca kolayca paylaşmak için kullanılabilir.
* useMemo:
  + React'te performans optimizasyonu yapmak için kullanılan bir Hook'tur. Bir bileşende hesaplanması maliyetli olan işlemlerin her render'da yeniden hesaplanmasını engelleyerek, yalnızca belirli bağımlılıkların değiştiği durumlarda bu hesaplamaların yapılmasını sağlar.
* useCallback:
  + useCallback, React'te performans optimizasyonu sağlamak amacıyla kullanılan bir Hook'tur. Özellikle bileşenlerin yeniden render edilmesi sırasında gereksiz fonksiyon yeniden oluşturulmalarını önlemek için kullanılır. useCallback, bir fonksiyonun bellekteki referansını hatırlar ve yalnızca belirli bağımlılıklar değiştiğinde bu fonksiyonu yeniden oluşturur.
* React developer tools’u kuralım
* Fonksiyonlar iç içe konulmamalıdır.



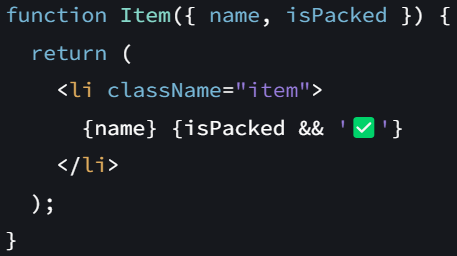
* Giderek daha fazla bileşeni iç içe yerleştirdikçe, bunları farklı dosyalara bölmeye başlamak genellikle mantıklıdır.
  + Next.js gibi dosya tabanlı yönlendirmeye sahip bir çatı (framework) kullanıyorsanız, kök bileşeniniz her sayfa için farklı olacaktır.



* Geliştiriciler eğer dosya tek bir bileşeni dışa aktarıyorsa genellikle varsayılan dışa aktarmayı, birden fazla bileşen ve değeri dışa aktarıyorsa adlandırılmış dışa aktarmayı kullanmaktadır.
  + Varsayılan ve adlandırılmış dışa aktarmalar arasındaki olası karışıklığı azaltmak için, bazı geliştirici ekipler yalnızca bir stile (varsayılan veya adlandırılmış) bağlı kalmayı veya bunları tek bir dosyada birlikte kullanmaktan kaçınmayı seçmektedir.
* JavaScript’in değişken adları konusunda sınırlamaları vardır. Örneğin, adları tire içeremez veya class gibi rezerve edilmiş sözcükler olamaz. Bu nedenle, React’te, birçok HTML ve SVG özellikleri camelCase ile yazılır. Örneğin, stroke-width yerine strokeWidth. class rezerve edilmiş sözcük olduğu için React’te, bunun yerine ilgili DOM özelliğinden sonra className yazarsınız:



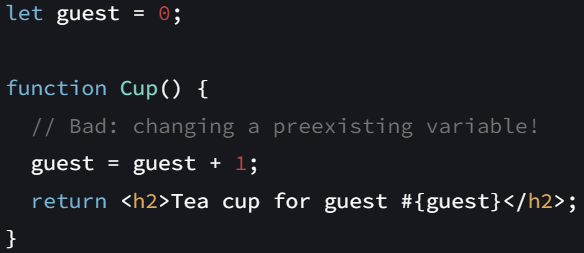
* JavaScript && operatörü eğer ifademizin sol tarafı (koşulumuz) true (doğru)ise ifadenin sağ tarafındaki değeri döndürür.

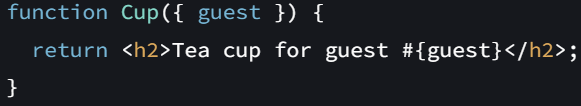


* Çift süslü parantez kullanımı
  + JSX içinde bir JS nesnesi geçmek için, objeyi başka bir çift süslü parantez içine almanız gerekir: person={{ name: "Hedy Lamarr", inventions: 5 }}.
  + Bir iç içe stil gerektiğinde, nesneyi style özelliğine iç içe parantez ile iletebilirsiniz.
  + İç içe stil özellikleri camelCase kullanılarak yazılır. Örneğin, HTML’de <ul style="**background-color**: black"> yazımı bileşeninizde <ul style={{ **backgroundColor**: 'black' }}> şeklinde yazılır.

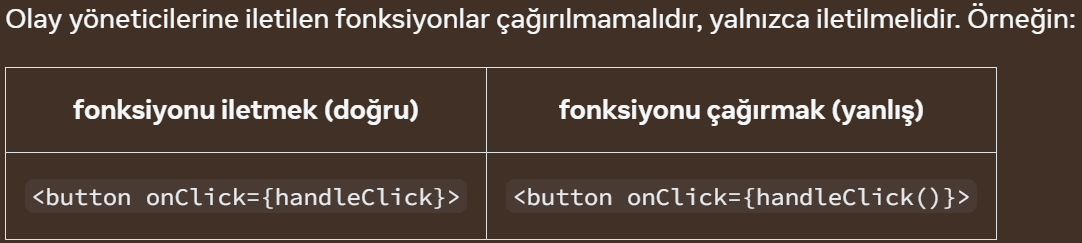


* Bileşenler saf (pure) tutulmalıdır. Aşağıdaki gibi olmamalıdır. (Dışarıdaki bir değişkeni kullanma) Bunun yerine değişken prop olarak geçilmelidir. Fakat event handler’ler püre olmak zorunda değildir.





* Eventler handle ile başlayıp olayın ismiyle devam edecek formatta isimlendirilirler. onClick={handleClick} ve onMouseEnter={handleMouseEnter} gibi,
* Geleneksel olarak, olay yönetici prop’ları on ile başlamalı ve büyük harfle devam etmelidir. Örneğin, Button bileşeninin onClick prop’u onSmash olarak da adlandırılabilir.



* children prop'u, bir bileşenin içinde yer alacak dinamik içeriği temsil eder

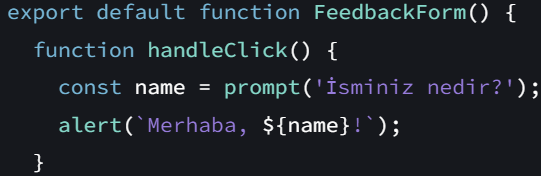


- e.stopPropagation(), üstündeki etiketlere eklenen olay yöneticilerinin tetiklenmesini önler. Örneğin buton tıklandığında butonun içinde bulunduğu <div> de tıklanmış olur. Bunu engellemek için kullanılır.

- e.preventDefault() , bazı olayların sahip olduğu varsayılan tarayıcı davranışını önler. Örneğin form submit olduğunda sayfa tekrar yüklenir, bunu engellemek için kullanılır.

- Hook’lar (use ile başlayan fonksiyonlar) yalnızca bileşenlerinizin en üst seviyesinde veya kendi Hook’larınızda çağrılabilir. Hook’ları koşullar, döngüler veya diğer iç içe geçmiş fonksiyonlar içinde çağıramazsınız.

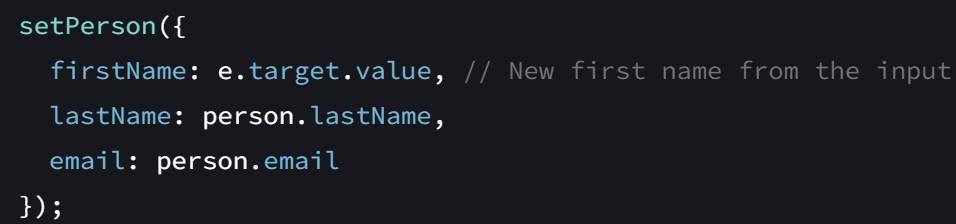
- Normal bir değişkenin iyi çalıştığı durumlarda state değişkenlerini kullanmayın.

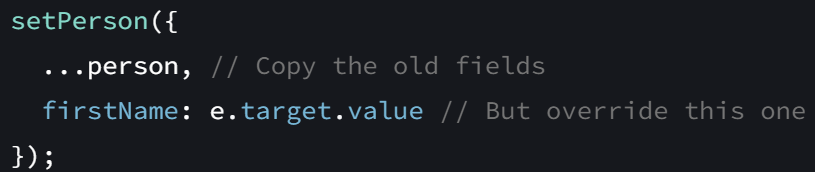


* Bir bileşen ilk defa render edildikten sonra, set fonksiyonu ile state’i güncelleyerek bileşenin tekrar render edilmesini sağlayabilirsiniz.
  + “Render etmek” React’in bileşenlerinizi çağırması demektir.
  + Örneğin, “Gönder”e bastığınızda, setIsSent(true) ifadesi React’e kullanıcı arayüzünü yeniden render etmesini söyler:
* Aşağıdaki kod number değerini 3 artırmaz, bir artırır. Bunu anlamak için number değerini sıfır ile değiştirebilirsiniz.

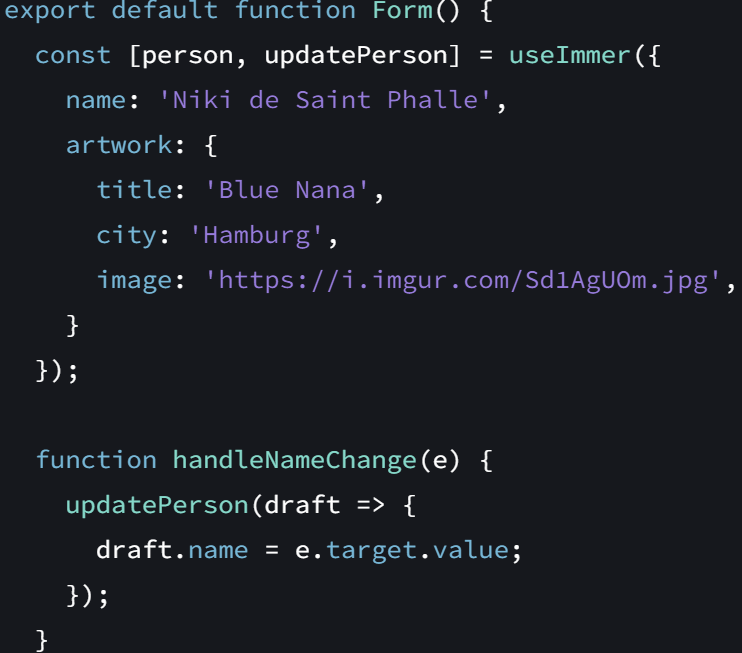
|  |  |
| --- | --- |
|  |  |

* Bu yaygın olmayan bir kullanım durumudur ancak aynı state değişkenini bir sonraki render’dan önce birden fazla kez güncellemek isterseniz, setNumber(number + 1) gibi sonraki state değerini iletmek yerine setNumber(n => n + 1) gibi kuyrukta önceki değerine dayanarak sıradaki değeri hesaplayan bir fonksiyon iletebilirsiniz. Böylelikle React’e state değerini değiştirmek yerine “state değeri ile bir şey yap” diyebilirsiniz.
* Spread söz dizimi: Her bir özelliği ayrı ayrı kopyalamak zorunda kalmadan ... nesne spread sözdizimini kullanabilirsiniz.
  + Dikkat edilmesi gereken bir nokta, ... spread sözdiziminin “yüzeysel” olmasıdır—yalnızca bir seviye derinliğe kadar kopyalar. Bu kopyalama işlemini hızlı yapar, ancak iç içe geçmiş bir özelliği güncellemek istiyorsanız, birden fazla kez kullanmanız gerekecektir.

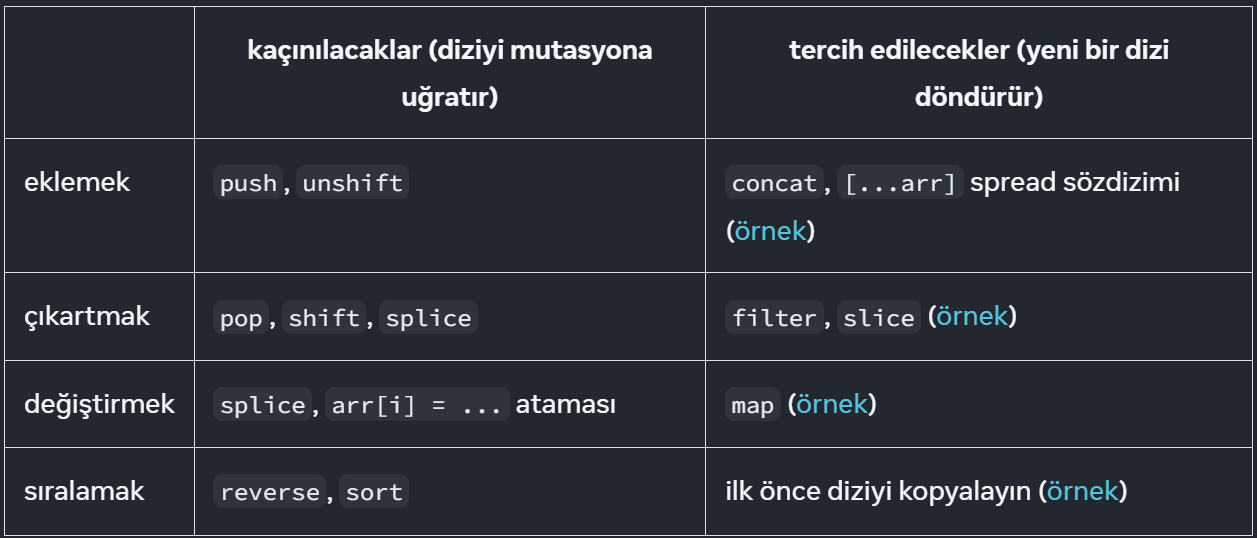




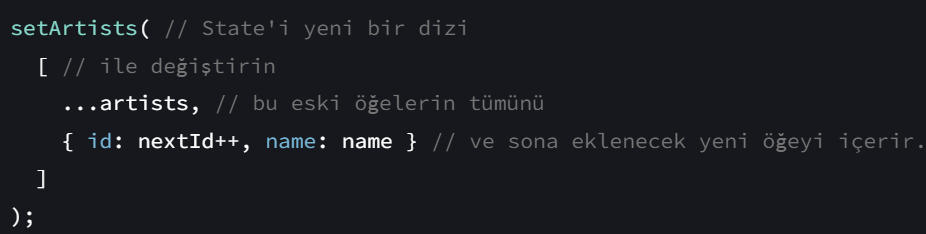
* Immer, özellikle state içerisinde iç içe geçme varsa ve nesnelerin kopyalanması tekrarlayan kodlara neden oluyorsa, olay işleyicilerini kısa tutmanın harika bir yoludur.



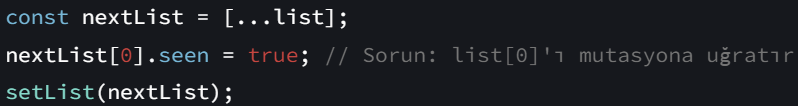
* React içerisindeki bütün state’leri değiştirilemez olarak ele alın.
  + React state içerisinde tuttuğunuz nesneleri direkt olarak değiştirmemelisiniz.
  + React’ta nesneleri state içinde sakladığınızda, nesneleri mutasyona uğratmak yeniden render işlemini tetiklemez ve önceki render “anlık görüntülerindeki” state’i değiştirir.
  + Bir nesneyi mutasyona uğratmak yerine, nesnenin yeni bir versiyonunu oluşturun, ve state’i nesneye ayarlayarak bir yeniden render oluşturun.
  + JavaScript’te, sayılar, stringler ve booleanlar gibi yerleşik temel veri tiplerinde herhangi bir değişiklik yapmak mümkün değildir.
* Diziler JavaScript’te değiştirilebilirdir, ancak bunları state içinde depolarken değiştirilemez olarak ele almalısınız. Tıpkı nesnelerde olduğu gibi, state’te depolanan bir diziyi güncellemek istediğinizde yeni bir dizi oluşturmanız (veya var olanın bir kopyasını oluşturmanız) ve ardından yeni oluşturduğunuz diziyi kullanmak için state’i güncellemeniz gerekir.
  + arr[0] = 'bird' şeklinde bir dizi içindeki öğeleri başka değerlere yeniden atamamanız, ayrıca push() ve pop() gibi dizileri mutasyona uğratan JavaScript metodlarını kullanmamanız gerektiği anlamına gelir.
  + Bu metodları kullanmak yerine, bir diziyi her güncellemek istediğinizde state setter fonksiyonunuza yeni bir dizi iletmelisiniz. Bunu yapmak için, filter() ve map() gibi diziyi mutasyona uğratmayan JavaScript metodlarını kullanarak orijinal diziden yeni bir dizi oluşturabilirsiniz.



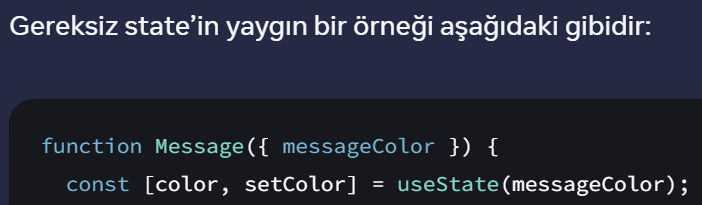
* Mevcut öğeleri ve son eleman olarak yeni öğeyi içeren yeni diziyi oluşturun.



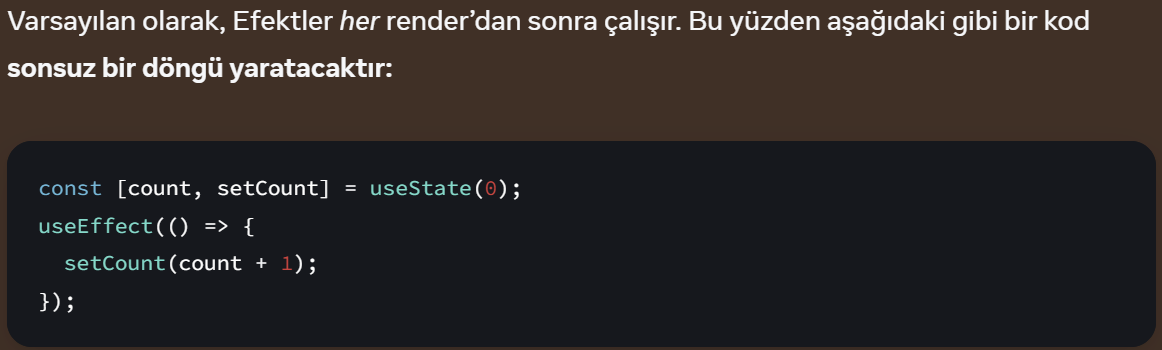
* Dizideki bazı ya da tüm öğeleri değiştirmek isterseniz yeni bir dizi oluşturmak için map() metodunu kullanabilirsiniz.
* Kopyalanan dizinin içindeki bir nesneyi değiştirdiğiniz zaman mevcut state’i de mutasyona uğratmış olursunuz.



* Prop’ları state’e yansıtmayın.

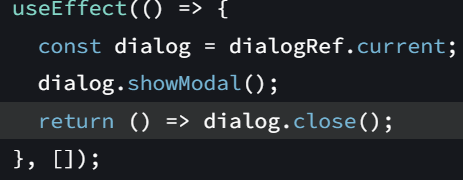


* Aynı konumda farklı bileşen tipleri arasında geçiş yapılırsa state sıfırlanır. Aynı konumda aynı bileşen tipleri arasında geçiş yapılırsa state sıfırlanmaz. <https://tr.react.dev/learn/preserving-and-resetting-state>
* İki sayaç arasında geçiş yaparken state’i sıfırlamanın iki yolu vardır.
  + Bileşenleri farklı konumlarda render edin
  + Her bileşene bir key (anahtar) prop’u verin
* Context ile veriyi aktarma
  + Bir prop’u ağacın derinliklerine aktarmak gerektiğinde veya birçok elemanın aynı prop’a ihtiyaç duyduğu durumlarda zahmetli ve uygunsuz hale gelebilir. Veriye ihtiyaç duyan elemanlar ile en yakın ortak üst bileşen arasındaki mesafe uzun olabilir ve state’i yukarı taşımak “prop drilling” adı verilen duruma yol açabilir.
  + Tüm alt ağacın ihtiyaç duyduğu her türlü bilgiyi aktarabilirsiniz: geçerli renk teması, o anda oturum açmış kullanıcı vb.
* Bir bileşenin “hatırlamasını” istediğiniz bilgi varsa, ancak bu bilginin yeni render’lar tetiklemesini istemiyorsanız, bir useRef kullanabilirsiniz.
  + state’i değiştirmek bileşeni yeniden render eder. Bir ref’i değiştirmek etmez!
* ref’lerin en yaygın kullanımı DOM elemanlarına erişmektir.
  + Örneğin, bir input’a programatik olarak odaklanmak istiyorsanız bu kullanışlıdır. JSX’te ref özelliğine bir ref geçtiğinizde, <div ref={myRef}> gibi, React karşılık gelen DOM elemanını myRef.current’e koyar. Bir eleman DOM’dan kaldırıldığı zaman, React myRef.current değerini null olarak günceller.
  + Ör: inputRef.current.focus();
* Ref’ler kaçış kapısıdır. Bunu sadece “React’in dışına çıkmanız” gerektiğinde kullanmalısınız. Bunun yaygın örnekleri arasında focus yönetimi, kaydırma konumu veya React’in göstermediği tarayıcı API’lerini çağırmak yer alır.
* React tarafından yönetilen DOM elemanlarını değiştirmekten kaçının.
* Sohbete mesaj göndermek bir olaydır çünkü direkt olarak kullanıcının belirli bir butona tıklaması ile gerçekleşir. Ancak, bir sunucu bağlantısı kurmak Efekttir.
  + Efektler ekran güncellendikten sonra işlemenin (commit) sonunda çalışır. Bu, React bileşenlerini bazı harici sistemlerle (ağ veya üçüncü parti kütüphane ile) senkronize etmek için iyi bir zamandır.
  + useEffect bir kod parçasının çalışmasını o render işlemi ekrana yansıtılana kadar “geciktirir”
* Gerekliyse temizleme (cleanup) ekleyin. Bazı Efektlerin, yaptıkları her şeyi nasıl durduracaklarını, geri alacaklarını veya temizleyeceklerini belirtmeleri gerekir. Örneğin, “bağlanmak” “bağlantıyı kese” ihtiyaç duyar, “abone ol” “abonelikten çıka” ihtiyaç duyar ve “veri getirme (fetch)” ya “iptal” ya da “görmezden gele” ihtiyaç duyar. Bir temizleme fonksiyonu döndürerek bunu nasıl yapacağınızı öğreneceksiniz.

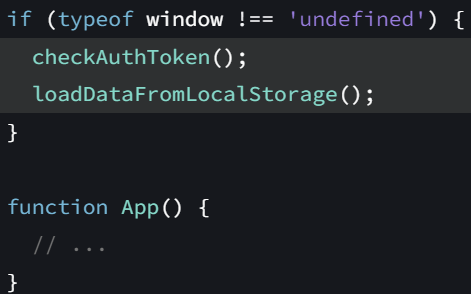


* Efektinizde bir temizleme fonksiyonu döndürün.

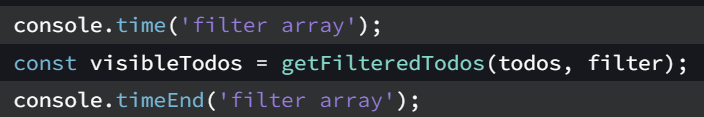




* Fazladan bir bağlan/bağlantıyı kes çağrı çifti olabilir, çünkü React, geliştirme sırasındaki hatalar için kodunuzu araştırma yapar.
* Strict Modu kapatarak geliştirme sırasındaki davranışlarından kaçınabilirsiniz ancak biz bunu açık tutmanızı tavsiye ediyoruz.
* Geliştirmede Strict modu açıkken, React kurulum ve temizleme işlemini asıl kurulumdan önce bir kere fazladan çalıştırır.
* Eğer Efektiniz bir şeye abone oluyorsa (subscribe), temizleme fonksiyonu abonelikten çıkarmalıdır.
* Modern React framework’ler verimli veri getirme mekanizmalarını entegre etmişlerdir.
  + Kullanıcı taraflı önbelleğe almayı kullanmayı ya da kendiniz kurmayı düşünün. Popüler açık kaynak çözümleri arasında React Query, useSWR ve React Router 6.4+ vardır.
* Yazdığınız bazı mantıklar uygulama başlatıldığında yalnızca bir kez çalışmalıdır. Bu mantıkları bileşenlerinizin dışına koyabilirsiniz:



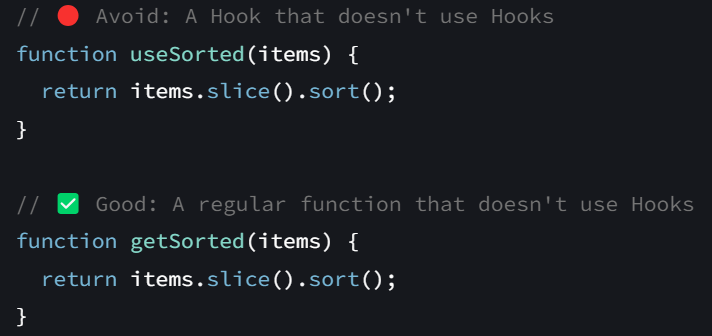
* Bir kod parçasında geçen süreyi ölçmek için bir konsol ekleyebilirsiniz.

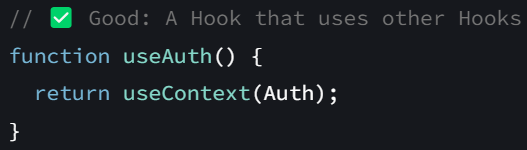


* Makinenizin kullanıcılarınızdan daha hızlı olduğunu aklınızda bulundurun bu nedenle performansınızı yapay bir yavaşlık ile test etmek daha iyi bir fikirdir. Örneğin, Chrome bunun için CPU Throttling seçeneği sunuyor.
  + En doğru ölçümleri elde etmek için, uygulamanızı üretim için derleyin ve kullanıcılarınızın sahip olduğu gibi bir cihazda test edin.
  + 1ms veya daha fazla ölçülen süre performans artırımı için değerlendirilmesi gerekir.
* Normalde, React aynı bileşen aynı noktada render edildiğinde state’i korur. Profile bileşenine bir key olarak userId ileterek, React’ten farklı userId’li iki Profile bileşenine herhangi bir state’i paylaşmaması gereken iki farklı bileşen olarak muamele etmesini istiyorsunuz. Key her değiştiğinde (userId olarak ayarladığınız), React DOM’u tekrar oluşturacak ve Profile bileşeninin ve tüm alt öğelerinin state’lerini sıfırlar.

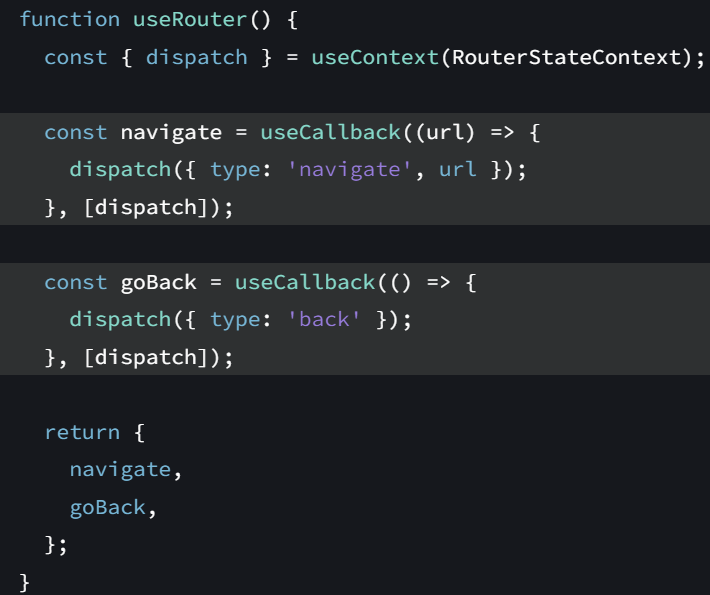


* Verileri bir Efekt içinde üst elemana iletmekten kaçının.
* If your function doesn’t call any Hooks, avoid the use prefix. Instead, write it as a regular function without the use prefix. For example, useSorted below doesn’t call Hooks, so call it getSorted instead:





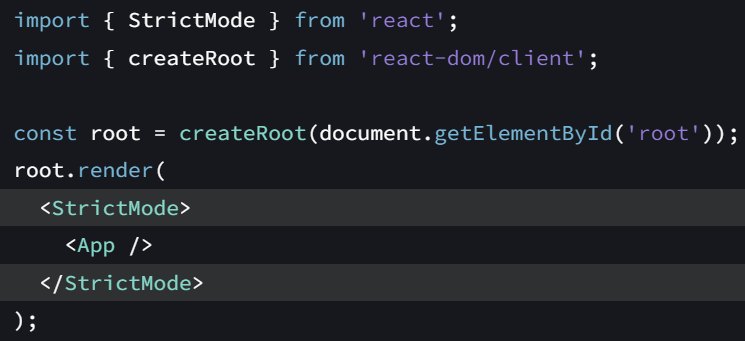
* useCallback bir fonksiyonu bağımlılıkları değişene kadar önbellekte tutar ve her istendiğinde yeni fonksiyon oluşturmak yerine aynı fonksiyonu döner.
* Özel Hook yazıyorsanız, döndürdüğü tüm fonksiyonları useCallback içine sarmanız önerilir:



* useDeferredValue: Sorgunun gecikmeli olarak atılması için kullanılır.
* Eğer harici sistemle senkronize etmeye çalışmıyorsanız, büyük ihtimalle Effect’e ihtiyacınız yoktur.
* Effect’ler ile veri getirmeye iyi alternatifler nelerdir?
  + Popüler açık kaynak çözümleri arasında React Query, useSWR ve React Router 6.4+ vardır.
* Fonksiyon effect içinde tanımlanırsa effectin tekrar çağrılması işi azalır.” Gereksiz fonksiyon bağımlılıklarını kaldırma”.

|  |
| --- |
|  |
|  |

* useMemo’ya yalnızca bir performans optimizasyonu olarak güvenmelisiniz. Kodunuz onsuz çalışmıyorsa, önce altta yatan sorunu bulun ve düzeltin. Daha sonra performansı artırmak için useMemo ekleyebilirsiniz.
* useReducer, useState ile çok benzerdir, ancak state güncelleme mantığını olay yöneticilerinden bileşeninizin dışındaki tek bir bir fonksiyona taşımanıza olanak tanır.
  + Bazı bileşenlerde yanlış state güncellemeleri nedeniyle sık sık hatalarla karşılaşıyorsanız bir reducer kullanılması önerilir.
* Eğer bir React çatısı (framework) veya yönlendirici oluşturuyorsanız, sayfa gezinmelerini transition’lar olarak işaretlemenizi öneririz.
* Eğer bir React çatısı (framework) veya yönlendirici oluşturuyorsanız, sayfa gezinmelerini transition’lar olarak işaretlemenizi öneririz.
* <StrictMode> geliştirmeleriniz sırasında bileşenlerinizdeki genel hataları erkenden bulmanızı sağlar.



* + Özellikle yeni yaratılmış uygulamalarınızda, tüm uygulamayı Strict Modu ile sarmanızı öneririz. (next.js için strict mode varsayılan olarak açık gelir)



# Next.js

* Referans sayfalar

<https://nextjs.org/docs>

<https://nextjs.org/learn/react-foundations>

<https://nextjs.org/learn/dashboard-app>

# Redux

* This is the basic idea behind Redux: a single centralized place to contain the global state in your application, and specific patterns to follow when updating that state to make the code predictable.
* You can think of a reducer as an event listener which handles events based on the received action (event) type.
* The current Redux application state lives in an object called the store .
* You can think of dispatching actions as "triggering an event" in the application.
* A "slice" is a collection of Redux reducer logic and actions for a single feature in your app.
* You can only write "mutating" logic in Redux Toolkit's createSlice and createReducer because they use Immer inside! If you write mutating logic in your code without Immer, it will mutate the state and cause bugs!
* One of the principles of Redux is that there should only be one store instance for an entire application.