

# JSX

JSX nedir, HTML ile farkları, süslü parantez { } ile JavaScript gömme, attribute'lar ve className, JSX kuralları ve conditional rendering'e kısa bir bakış.

**Başlangıç** 12 dk

Oluşturulma tarihi: 5 Ekim 2026

Çevrimiçi oku: <https://www.learnforgex.com/tr/topics/jsx>

## JSX

Bir React projesini kurmayı Bir React Projesi Oluşturmak dersinde gördük. Şimdi asıl React kodunun içine giriyoruz. React'te arayüz yazarken kullandığın söz dizimine **JSX** denir -- bu ders, JSX'in ne olduğunu ve temel kurallarını, olabildiğince basit örneklerle anlatıyor.

### JSX Nedir?

JSX, HTML'e çok benzeyen ama aslında JavaScript'in bir uzantısı olan bir söz dizimidir. React component'lerinin çoğu, arayüzü tarif etmek için JSX kullanır:

```
// JSX looks like HTML, but it's actually an extension of JavaScript.  
// The browser doesn't understand it directly -- a compiler (Babel/Vite)  
// converts it into a regular JavaScript function call behind the scenes.  
const element = <h1>Hello, React!</h1>;  
  
// The line above means the same thing as the following (if it were  
// written without JSX):  
// const element = React.createElement("h1", null, "Hello, React!");  
  
console.log(element);
```

Bu kod HTML gibi görünüyor, ama aslında bir JavaScript değişkenine ( `element` ) bir değer atanıyor. Tarayıcı JSX'i doğrudan anlamaz -- projenin build araçları (bu kursta Vite), bu kodu tarayıcının anlayacağı sade bir JavaScript fonksiyon çağrısına otomatik olarak dönüştürür. Bunu elle yapman gerekmez, arka planda otomatik olur.

## HTML vs JSX

JSX, HTML'e benziyor ama aynı şey değil. En önemli farklardan biri: JSX, sonunda bir JavaScript değeridir -- yani onu bir değişkene atayabilir, bir fonksiyondan döndürebilir, bir listeye koyabilirsin. Düz HTML bunu yapamaz.

Bir diğer fark: JSX'te bazı isimler HTML'dekinden farklıdır (birazdan "Attribute'lar ve className" bölümünde göreceğiz). Ama genel görünüm büyük ölçüde HTML ile aynı -- `<div>`, `<h1>`, `<button>` gibi etiketleri tanıdık geleceksin.

## JSX İçinde JavaScript: Süslü Parantezler { }

JSX'in en güçlü yanı, içine gerçek JavaScript kodu gömebilmen. Bunu tek süslü parantez { } ile yaparsın:

```
// Inside curly braces { } you can write ANY JavaScript expression that
// produces a result: a variable, an addition operation, a function call...
const name = "Emma";
const greeting = <h1>Hello, {name}!</h1>;

const a = 5;
const b = 3;
const total = <p>Total: {a + b}</p>;

function shout(text) {
  return text.toUpperCase();
}
const loud = <p>{shout("i am learning react")}</p>;

console.log(greeting, total, loud);
```

Süslü parantez içine, bir **sonuç üreten** herhangi bir şey yazabilirsin: bir değişken ( {name} ), bir işlem ( {a + b} ), ya da bir fonksiyon çağırısı ( {shout(...)} ). Sonuç üretmeyen şeyler (örneğin bir if bloğu) buraya yazılamaz -- bunun neden böyle olduğunu birazdan "Conditional Rendering'e Kısa Bir Bakış" bölümünde göreceğiz.

## Attribute'lar ve className

HTML'deki attribute'ları ( src , alt , class gibi) JSX'te de kullanırsın, ama iki önemli fark var:

```
// Attributes in JSX are very similar to HTML, but there are a few differences:
// 1) "className" is used instead of "class" (class has a separate meaning in JavaScript).
// 2) Attribute values can also be a JavaScript variable using { }.
const userName = "ayse";

const avatar = (
  <img className="avatar" src={`\images/${userName}.png`} alt="User avatar" />
);

// If the tag itself doesn't close (e.g. <img>, <input>, <br>), it MUST be
// "self-closing" in JSX -- a / is added at the end.
const divider = <hr />;

console.log(avatar, divider);
```

Birincisi: HTML'deki `class` yerine JSX'te `className` yazılır ( `class` , JavaScript'te zaten başka bir anlama sahip olduğu için). İkincisi: attribute değerlerini de, tıpkı metin içinde olduğu gibi, süslü parantez `{ }` ile bir JavaScript ifadesine bağlayabilirsin.

## JSX Kuralları

JSX'in uyman gereken birkaç basit kuralı var:

```
// JSX has a few simple but strict rules:

// 1) A JSX block must have a SINGLE root element.
// This does NOT WORK: <h1>Title</h1><p>Text</p> (two sibling elements, no single root)
// Instead, wrap it with a <div> (or with <> </> as below):
const withDiv = (
  <div>
    <h1>Title</h1>
    <p>Text</p>
  </div>
);

// <> </> ("Fragment") does the same job without adding an extra <div> --
// a wrapper invisible to the DOM, existing only to satisfy the JSX rule.
const withFragment = (
  <>
    <h1>Title</h1>
    <p>Text</p>
  </>
);

// 2) Attribute names are written in camelCase: "onClick" not "onclick",
// "tabIndex" not "tabindex".

// 3) "className" is used because "class" is a keyword in JavaScript.

console.log(withDiv, withFragment);
```

En önemlisi: bir JSX bloğunun **tek bir kök (root) elementi** olmalı -- iki kardeş element'i yan yana döndüremezsin. Bir `<div>` ile sarmalayabilir ya da fazladan bir HTML elementi

eklemek istemiyorsan `<> </>` ("Fragment") kullanabilirsin. Diğer kurallar: attribute isimleri camelCase yazılır ( `onClick` , `tabIndex` ) ve kendi kendine kapanan etiketler ( `<img>` , `<input>` ) JSX'te mutlaka `/` ile kapatılmalı ( `<img />` ).

## Conditional Rendering'e Kısa Bir Bakış

Süslü parantez `{ }` içine yalnızca "sonuç üreten" bir ifade koyabildiğini gördük -- bu yüzden JSX içinde doğrudan `if` yazamazsın ( `if` , bir sonuç üretmez, sadece bir kod bloğunu çalıştırıp çalıştırmayacağına karar verir). Bunun yerine, sonuç üreten bir ternary ( `? :` ) kullanılır:

```
// You can't write "if" inside JSX (if is a statement, not an
// expression) -- but since you can put an "expression" inside { }, tools
// based on expressions like ternary (?:) or && come in handy here. We'll
// go into this topic in much more detail in the "Conditional Rendering"
// section (in the State & Events category) -- here we're just seeing why
// JSX does it this way.
const isLoggedIn = true;

const message = isLoggedIn ? <p>Welcome!</p> : <p>Please log in.</p>;

console.log(message);
```

Burada yalnızca kısa bir örnek gördük -- koşula göre farklı arayüz göstermeyi ("conditional rendering"), State & Events kategorisindeki "Conditional Rendering" dersinde çok daha detaylı işleyeceğiz.

## Özet ve Terimler Sözlüğü

JSX, HTML'e benzeyen ama aslında JavaScript olan bir söz dizimidir. Süslü parantez `{ }` ile içine JavaScript ifadeleri gömebilirsin. `class` yerine `className` kullanılır, her JSX bloğunun tek bir kök elementi olmalıdır, ve kendi kendine kapanan etiketler `/` ile kapatılmalıdır.

### Terimler Sözlüğü

**JSX** — HTML'e benzeyen, ama aslında JavaScript'in bir uzantısı olan söz dizimi.

**Expression (İfade)** — Bir sonuç üreten kod parçası (örn. `a + b`); JSX'te `{ }` içine yalnızca bu tür kod yazılabilir.

**className** — JSX'te HTML'deki `class` attribute'unun karşılığı.

**Fragment ( `<> </>` )** — Fazladan bir HTML elementi eklemekten birden fazla element'i tek bir kökte toplamaya yarayan JSX aracı.

**Root Element (Kök Element)** — Bir JSX bloğundaki en dıştaki, tek element -- her JSX bloğunun tam olarak bir tane olmalıdır.

## Pratik Proje

Bu kategoride (What Is React?, Creating a React Application, JSX) öğrendiğimiz kavramları bir arada kullanan, gerçek ve çalıştırılabilir bir örnek proje var: [React Fundamentals Demo](#).

Proje; JSX, süslü parantez `{ }` ile JavaScript ifadeleri, attribute'lar ve ternary ile kısa bir conditional rendering önizlemesini bir arada gösteriyor. Bilgisayarına indirip çalıştırabilir, kodunu satır satır inceleyebilirsin:

```
git clone https://github.com/cdurgun/react-course-projects.git
cd react-course-projects
npm install
cd projects/react-fundamentals
npm run dev
```

`react-course-projects` deposu **npm workspaces** kullanır -- `npm install` yalnızca bir kez, depo kökünde çalıştırılır ve tüm proje klasörleri ortak bağımlılıkları paylaşır (her klasörde ayrı bir `node_modules` oluşmaz). Kök dizinde `npm install` 'ı daha önce çalıştırdıysan, doğrudan `cd react-course-projects/projects/react-fundamentals` yapıp `npm run dev` demen yeterli.