21.11.2025

İşte **Meta Android Developer Professional Certificate** programından çalıştığım derslerden aldığım notlar.
Tarih: 21.11.2025
İçerdiği Konular:

- JavaScript'te **Nesne ve Dizi Yapısını Parçalama (Destructuring)**
- **for...in** ve **for...of** Döngüleri Arasındaki Fark
- **ES6 Şablon Literalleri (Template Literals)**

Nesne ve Dizi Yapısını Parçalama (Destructuring)

- Bu işlem, bir nesnenin veya dizinin içindeki özellikleri/değerleri alıp, bağımsız değişkenlere kopyalamak gibi düşünülebilir.
- Tıpkı bir word processor'da bir metnin formatını kopyalayıp başka bir metne uygulamak gibi. Formatı (yani özellikleri) kopyalıyorsun.
- **Önemli:** Bu bir kopyalama işlemidir. Orijinal nesne veya diziye bir şey olmaz, değişmez.

Örnek:
`Math` nesnesinden `PI` değerini çekmek istiyorum.

```
let { PI } = Math;
```

- Bu satır, `Math.PI` değerini alır ve onu `PI` adında YENİ bir değişkene kopyalar.
- Dikkat: Değişken ismi, nesne içindeki property ismiyle **aynı olmak zorunda**. Büyük-küçük harf duyarlı!
  - `let {pi} = Math;` dersen `pi` değişkeni `undefined` olur, çünkü `Math` nesnesinde `pi` diye bir property yok, `PI` var.
- Kopyaladıktan sonra bu iki değer birbirinden tamamen bağımsızdır.

```
console.log(PI === Math.PI); // true döner, çünkü değerler aynı

PI = 1; // Kopya değişkeni değiştiriyorum

console.log(PI === Math.PI); // false döner, artık aynı değiller
console.log(Math.PI); // Orijinal değer hiç bozulmadı.
```

---

for...in ve for...of Döngüleri

- İkisi de döngü ama nesnelerde çalışırken çok farklı davranıyorlar.

Örnek Senaryo:
Bir araba nesnem var (`car`). Bir de bu nesneden `Object.create()` ile `sportsCar` adında yeni bir nesne yaratıyorum. Yani `sportsCar`, `car` nesnesinin mirasçısı (prototype'ı) oluyor.

```
let car = { engine: true, steering: true, speed: "slow" };
let sportsCar = Object.create(car);
sportsCar.speed = "fast"; // sportsCar'ın kendi speed özelliği
```

for...in Döngüsü (Güvenilir Değil!)

- **Hem nesnenin KENDİ özelliklerinde, hem de PROTOTYPE zincirindeki özelliklerde döner.**
- Yani `sportsCar` için döndürdüğünde, kendi `speed` özelliği ve prototype'ı olan `car`'dan miras aldığı `engine` ve `steering` özelliklerini de listeler.

```
for (prop in sportsCar) {
  console.log(prop); // "speed", "engine", "steering" yazar
}
```

- Bu, genellikle istenmeyen bir davranış. Sadece nesnenin kendi özellikleriyle ilgileniyorsak sorun yaratır.

for...of Döngüsü (Güvenilir!)

- **Sadece nesnenin KENDİ özelliklerinde döner.**
- `Object.keys()` metoduyla beraber kullanılır. `Object.keys()` bize sadece o nesneye ait property isimlerini bir dizi olarak verir.

```
for (prop of Object.keys(sportsCar)) {
  console.log(prop + ": " + sportsCar[prop]); // Sadece "speed: fast" yazar
}
```

Özet: Bir nesnenin sadece kendi property'leri üzerinde işlem yapmak için **for...of** ve **Object.keys()** ikilisi kullan. Tüm prototype zincirini dolaşmak için **for...in** kullan, ama dikkatli ol.

ES6 Şablon Literalleri (Template Literals)

- ES5'te string'ler `'tek tırnak'` veya `"çift tırnak"` ile yapılıyordu ve çok kısıtlıydı.
- **Çok Satırlı String:** ES5'te bir string'i alt satıra geçirip yazamazdın. Hata verirdi.

ES6 ile gelen ` (backtick) işareti hayatı kurtarıyor.

1. Çok Satırlı String:

```
// ES5 - HATALI!
let noMultiLine = "No multi-line
strings in ES5"; // Hata!
```

```
// ES6 - MÜKEMMEL!
let multiLine = `Bu bir
çok satırlı
string örneği`;
console.log(multiLine); // Aynen alt alta basar.
```

2. Değişken Yerleştirme (Interpolation):

- Artık `+` ile string birleştirmeye gerek yok.
- Backtick içinde `${değişkenAdı}` yazman yeterli.

```
let first = `"İlk" değişkenim`;
let second = `'İkinci' değişkenim`;

// ES5 birleştirme (zor ve karışık):
console.log("ES5: " + first + " ve " + second);

// ES6 Template Literal (süper basit ve temiz):
console.log(`ES6: ${first} ve ${second}`);
```
