FRONT-END

JAVASCRIPT

DİZİ İTERASYON METOTLARI

DİZİLERDE İTERASYON

- JS'de bir dizi içerisinde iterasyon yapmak için çok farklı yollar kullanılabilir.

- **Döngüler ile**
  - Klasik **for** döngüsü
  - **for in** döngüsü
  - **for of** döngüsü

- **Dizi iterasyon metotları ile (En çok kullanılanlar)**
  - Array.forEach()
  - Array.map()
  - Array.filter()
  - Array.reduce()

FOR DÖNGÜSÜ İLE DİZİ İTERASYONU

ÖRNEK: Bir dizideki sayıların toplamını hesaplayan uygulamayı For döngüsü ile yazınız.

```
const rakamlar = [-5, 15, 22, -4, 45, 78, -25];

let toplam = 0;

for (let i = 0 ; i <rakamlar.length ; i++) {
  toplam += rakamlar[i];
}

console.log(toplam);
```

NOT: Daha dinamik bir program için **.length** metodu ile döngünün dizinin elaman sayısı kadar dönmesi sağlanır.

FOR DÖNGÜSÜ İLE DİZİ İTERASYONU

ÖRNEK: Bir dizideki pozitif ve negatif sayıların toplamını hesaplayan uygulamayı For döngüsü ile yazınız. **topla** adında bir fonksiyonda hesaplamayı yapınız.

```
const dizi = [-5, 15, 22, -4, 45, 78];

const topla = (dizi) => {
  let negatifler = 0;
  let pozitifler = 0;
  for (let i = 0; i < dizi.length; i++) {
    if (dizi[i] < 0) {
      negatifler += dizi[i];
    } else {
      pozitifler += dizi[i];
    }
  }
  console.log(`Dizideki Pozitif Sayıların Toplamı: ${pozitifler}`);
  console.log(`Dizideki Pozitif Sayıların Toplamı: ${negatifler}`);
};
topla(dizi);
```

FOR IN DÖNGÜSÜ İLE DİZİ İTERASYONU

- **For** döngüsünün kısaltılmış halidir. (sayaç ve koşul kullanmaya gerek yok.)
- Özellikle **dizi** ve **nesnelerin** **iterasyonu** için geliştirilmiştir.

```
for ( değişken in diziAdı) {
    // Döngü içi
}
```

ÖRNEK: Bir dizideki tüm elemanların toplamını bulan programı **FOR IN** ile yazınız.

```
const sayılar = [-5, 15, 22, -4, 45, 78, -25];
let sonuç = 0;
for (let i in sayılar) {
  sonuç += sayılar[i];
}
console.log(sonuç);
```

FOR IN DÖNGÜSÜ İLE DİZİ İTERASYONU

ÖRNEK: İki ayrı dizideki eşleşen indis elemanları birleştirerek ayrı bir diziye saklayan uygulamayı **FOR IN** ile yazınız.

```
const adlar = ["Ahmet", "Can","Mustafa","Ayşe","Elif"];
const soyAdlar= ["Öztürk", "Yılmaz", "Arı", "Çalı", "Yazı"];
```

```
const birleştir = (x,y) => {

  let adVeSoyadlar = [];
  for (let i in x) {
    adVeSoyadlar[i] = `${x[i]} ${y[i]}`;
  }
  return adVeSoyadlar;
};

console.log(birleştir(adlar, soyAdlar));
```

FOR OF DÖNGÜSÜ İLE DİZİ İTERASYONU

- **FOR OF** Döngüsü **FOR IN** döngüsünün Syntax'ına çok benzemektedir.

- Ancak, **FOR OF**, **bir çok veri yapısı** üzerinde çalışabildiği için **FOR IN'e** göre daha geniş kullanım alanına sahiptir.
  - Diziler, Stringler, Map'ler v.b bir çok veri yapısında kullanılabilir.

```
for ( değişken of veriYapısı) {
    // Döngü içi
}
```

FOR OF DÖNGÜSÜ İLE DİZİ İTERASYONU

ÖRNEK: Dizideki elemanların toplamını bulan programı **FOR OF** ile yazınız.

```
const sayılar = [-5, 15, 22, -4, 45, 78, -25];
let sonuç = 0;
for (let i of sayılar) {
  sonuç += sayılar[i];
}
console.log(sonuç);
```

ÖRNEK: Dizideki elemanları birleştirerek tek bir **String** haline getiren uygulamayı **FOR OF** ile yazınız.

```
let arabalar = ["BMW", "Volvo", "Mini"];
```

```
let yazı = "";
for (let x of arabalar) {
  yazı += x + " ";
}
console.log(yazı);
```

FOREACH METODU İLE DİZİ İTERASYONU

- **Array.forEach()** bir döngü deyimi değil bir dizi **İTERASYON** metodudur.

- Bu metot, bir fonksiyonu parametre olarak alır ve bu fonksiyona göre bir belirtilen dizi üzerinde **iterasyon** yapılabilir.

- Avantajı kullanımı kolaydır. Dezavantajı ise döngüyü kırmak ve atlamak mümkün değildir.

- Ayrıca **forEach** metodu orijinal diziyi **değiştirmez**.

FOREACH METODU İLE DİZİ İTERASYONU

ÖRNEK: Bir dizideki elemanları her birini ayrı ayrı yazdıran uygulamayı **forEach** metodu ile yazınız.

```
let öğrenciler = ["John", "Ali", "Ahmet"];
```

```
öğrenciler.forEach(yazdır);

function yazdır(v) {
  console.log(v);
}
```

Daha kısa hali (arrow function)

```
öğrenciler.forEach((k) => console.log(k));
```

FOREACH METODU İLE DİZİ İTERASYONU

ÖRNEK: Bir dizideki elemanların toplamını bularak bunu h2 elamanı olarak web sayfasına yazdıran uygulamayı FOR EACH metodu ile yazınız.

index.html

```
<h2>Dizinin Toplamı: <span class="toplam"></span></h2>
```

diziler.js

```
const dizi3 = [65, 44, 12, 4, -20, 19];

let toplama = 0;
dizi3.forEach(hesapla);

function hesapla (s) {
  toplama += s;
  document.querySelector(".toplam").innerHTML = toplama;
};
```

FOREACH METODU İLE DİZİ İTERASYONU

ÖRNEK: Bir dizideki elemanların toplamını bularak bunu **h2** elamanı olarak web sayfasına yazdıran uygulamayı **FOR EACH** metodu ile yazınız.

Daha kısa hali (ArrowFunction)

```
const dizi4 = [65, 44, 12, 4, -20, 19];

let t = 0;

dizi4.forEach((s) => (t += s));

document.querySelector(".toplam").innerHTML = t;
```

FOREACH METODUNDA İNDİS KULLANIMI

- **forEach()** metodu içerinde çağrılan ya da tanımlanan fonksiyon aslında **3 adet parametre** alabilmektedir.

```
Array.forEach ( function(şuankiDeğer, indis, dizi))
```

- **şuankiDeğer:** Seçilen dizi elemanının mevcut değerini göstermektedir. Kullanımı **ZORUNLUDUR**.
- **İndis:** Şu anki dizi elemanının sırasını (index) gösterir. **OPSİYONEL.**
- **Dizi:** Şu an ki elemanın ait olduğu dizi nesnesidir. **OPSİYONEL.**

NOT: Bu parametrelerin isimlerini kullanıcı belirler ancak sırası önemlidir. 1. parametre değer, 2. si index, 3.sü ise dizidir.

FOREACH METODUNDA İNDİS KULLANIMI

ÖRNEK: Belirtilen dizinin her bir elamanının 5 katını alarak ayrı bir dizide saklayan uygulamayı **forEach()** metodu ile yazınız.

```
const dizi5 = [-5, 24, -13, 7];
const yeniDizi = [];
dizi5.forEach(katAl);

function katAl(değer, indis, dizi5) {
  yeniDizi[indis] = değer * 5;
}
console.log(yeniDizi);
```

Daha kısa hali (ArrowFunction)

```
const dizi6 = [-5, 24, -13, 7];
const yeniDizi = [];
dizi6.forEach( (d,i) => yeniDizi[i] = d * 5);
console.log(yeniDizi);
```

NOT: Bu örnekte fonksiyon içeresinde dizi5 nesnesi ile işlem yapmadığımız için parametre listesinden silebiliriz.

Ancak, indis'i kullanmayıp dizi5 nesnesine ihyacımız olsaydı indis parametresini silemezdik. Eğer silersek sıralama karışacaktır. Bu durumda 3 parametreyi de kullanmamız gerekir.

MAP() METODU

- **Array.map()** metodu, bir fonksiyonu parametre olarak alır ve orijinal dizinin kopyasını bu fonksiyona göre modifiye ederek döndürür.

- Yani bir diziyi **transformasyondan** geçirmek için **map()** metodu kullanılabilir.

- **map()** metodu orijinal diziyi **değiştirmez**. Yeni bir dizi oluşturarak döndürür.

MAP() METODU

ÖRNEK: Bir dizideki elemanların **5 katını** alarak yeni bir diziye kaydeden uygulamayı **map()** metodu ile yazınız.

```
const rakamlar = [3, 7, 17, 8, 9, 3, 0];
```

```
const katAlınmış = rakamlar.map((x) => x * 5);
console.log(katAlınmış);
```

- Bu uygulamayı **forEach** ile de yapmıştık ancak **map()** kullanmak çok daha basit.
- **map()** metodu, güncellenmiş diziyi doğrudan bir değişkene atmaya izin vermektedir.

MAP() METODU

ÖRNEK: Bir dizideki tüm isimleri **BÜYÜK** harfe dönüştüren uygulamayı yazınız.

```
const isimler = ["Mustafa", "Murat", "Ahmet", "Mustafa", "Ayşe", "canan"];
```

```
const büyükHarfIsimler = isimler.map((x) => x.toUpperCase());
console.log(büyükHarfIsimler);
```

ÖRNEK: tlFiyatlar dizisindeki fiyatların Euro ve dolar karşılıklarını hesaplatarak yeni dizlere kaydediniz.

```
const euro = 9.68;
const dolar = 8.1;
const tlFiyatlar = [100, 150, 100, 50, 80];
```

```
const dolarFiyatlar = tlFiyatlar.map((x) => (x / dolar).toFixed(2));
const euroFiyatlar = tlFiyatlar.map((x) => (x / euro).toFixed(2));
```

MAP() METODUNDA İNDİS KULLANIMI

- **Map()** metodu forEach() **3 adet parametre** alabilmektedir.

```
Array.map( function(şuankiDeğer, indis, dizi))
```

- **şuankiDeğer:** Seçilen dizi elemanının mevcut değerini göstermektedir. Kullanımı **ZORUNLUDUR**.
- **İndis:** Şu anki dizi elemanının sırasını (index) gösterir. **OPSİYONEL.**
- **Dizi:** Şu an ki elemanın ait olduğu dizi nesnesidir. **OPSİYONEL.**

NOT: Bu parametrelerin isimlerini kullanıcı belirler ancak sırası önemlidir. 1. parametre değer, 2. si index, 3.sü ise dizidir.

MAP() METODU

ÖRNEK: tlFiyatlar dizidekisindeki ürünlere zam yapılmak isteniyor. Fiyatı 100 TL den fazla olanlara %10 zam, 100 TL den az olanlara ise %15 zam yapılmak isteniyor. Ayrıca, zamlı olan yeni değerleri örnekteki gibi diziye saklamak istiyoruz.

1.Ürün Zamlı Fiyatı:110

```
const değerler = tlFiyatlar.map(function (d, i) {
        if (d > 100) {
                return `${i + 1}. Ürün Fiyatı: ${(d * 1.1).toFixed(2)}`;
        }else {
                return `${i + 1}. Ürün Fiyatı: ${(d * 1.15).toFixed(2)}`;
        }
});

console.log(değerler);
```

FILTER() METODU

- **Array.filter()** metodu, bir fonksiyonu parametre olarak alır ve orijinal dizinin kopyasını bu fonksiyona göre **filtreleyerek** döndürür.
- Yani bir dizideki istediğimiz elemanların seçmek için kullanılır.
- **filter()** metodu orijinal diziyi **değiştirmez**. Yeni bir dizi oluşturarak döndürür.

ÖRNEK: **Koordinatlar** dizisindeki negatif koordinatları alıp yeni bir diziye saklayan uygulamayı **filter()** ile yapınız.

```
const koordinatlar = [-100, 150, -32, 43, -20]
```

```
negatifKoordinatlar = koordinatlar.filter((x) => x < 0);
console.log(negatifKoordinatlar);
```

PIPELINE (HAT)

Dizi iterasyon metotları **ardı ardına** kullanılabilir. Böylelikle ardışık bir şekilde diziler işlenebilir.

ÖRNEK: Koordinatlar dizisindeki negatif koordinatları seçerek bunları pozitife çevirip alt alta konsola bastıran uygulamayı yazınız.

```
koordinatlar
.filter((x) => x < 0)
.map((t) => t * -1)
.forEach((y) => console.log(y));
```

PIPELINE (HAT)

ÖRNEK:**Bireyler** dizisindeki kişilerden adı "**Belirtilen**" harf ile başlayanları seçerek ayrı bir diziye saklayan uygulamayı yazınız.

```
const bireyler = ["Mustafa", "Murat", "Ahmet", "mustafa", "Ayşe", "Canan"];

const başHarfeGöreSeç = function (harf) {
  const h = harf.toUpperCase();
  const filtrelenmiş = bireyler.filter((t) => t.startsWith(h)).map((t) => t);
  return filtrelenmiş;
};

console.log(başHarfeGöreSeç("m"));
console.log(başHarfeGöreSeç("A"));
```

REDUCE() METODU

- **Array.reduce()** metodu, bir fonksiyonu parametre olarak alır ve orijinal diziyi bu fonksiyona göre işleyerek **tek bir değer** döndürür.

- Örneğin bir dizinin değerlerinin toplamını bulmak için **reduce()** metodu kullanılabilir.

- **reduce()** metodu orijinal diziyi **değiştirmez**. Sadece bir **değer** döndürür.

REDUCE() METODU

ÖRNEK: Koordinatlar dizisindeki değerlerin toplamını hesaplayarak konsola bastıran uygulamayı **reduce()** ile yazınız.

```
const toplam = koordinatlar.reduce(function (x, y) {
  return x + y;
});
console.log(toplam);
```

```
// Arrow fonskiyonu ile Daha kısa
const toplamıBul = koordinatlar.reduce((x, y) => x + y);
console.log("KOORDİNAT TOPLAMI:" + toplamıBul);
```

REDUCE() METODUNDA İNDİS KULLANIMI

- **reduce()** metodu içerisindeki fonksiyon **4 adet parametre** alabilmektedir.

```
Array.reduce( function(toplam, şuankiDeğer, indis, dizi))
```

  - **toplam:** Her iterasyonda ardışık olarak yapılan işlemlerin kümülatif toplamını gösterir. **ZORUNLUDUR**.
  - **şuankiDeğer:** Seçilen dizi elemanının mevcut değerini göstermektedir. Kullanımı **ZORUNLUDUR**.
  - **İndis:** Şu anki dizi elemanının sırasını (index) gösterir. **OPSİYONEL.**
  - **Dizi:** Şu an ki elemanın ait olduğu dizi nesnesidir. **OPSİYONEL.**

REDUCE() METODU

ÖRNEK: Koordinatlar dizisindeki değerlerin toplamını, ara değerleri de göstererek konsola bastıran uygulamayı **reduce()** ile yazınız.

```
const toplam = koordinatlar.reduce(function (x, y, i) {
  console.log(`iterasyon ${i} ${x}`);
  return x + y;
});
console.log(toplam);
```

x : toplam değer, **y:** anlık değer, **i:** indis

ÖRNEK: Koordinatlar dizisindeki değerlerin ortalamasını hesaplayarak konsola bastıran uygulamayı **reduce()** ile yazınız.

```
const ortalama = koordinatlar.reduce((x, y) => x + y) / koordinatlar.length;
console.log("Koordinatların Ortalaması:" + ortalama);
```

PIPELINE ÖRNEK

ÖRNEK: Bir Firma, **3000 TL** den **az** olan maaşlara **%10** zam yapmak istiyor ve zam yapılan bu kişilere **toplam** kaç TL ödeneceğini bilmek istiyor. İlgili programı yazınız.

```
const maaşlar = [3000, 2891, 3500, 4200, 7000, 2500];
```

```
const zamlıToplam = maaşlar
.filter((x) => x < 3000)
.map((x) => x * 1.1)
.reduce((x, y) => x + y);

console.log(zamlıToplam.toFixed(2));
```