# Golang iter Paketi - Kapsamlı Rehber

iter paketi, Go 1.21 ve sonrası için fonksiyonel tarzda iterasyonlar yapmamızı sağlayan **Iterator tabanlı bir pakettir**. Koleksiyonlar üzerinde map, filter, reduce gibi işlemleri zincirleme (chaining) ile yapmayı kolaylaştırır.

Paket, Go’nun resmi deneysel paketlerinden golang.org/x/exp/iter altında bulunur.

## 1️⃣ Iterator Oluşturma

### iter.Iter(slice)

Bir slice veya array’den iterator oluşturur.

package main  
  
import (  
 "fmt"  
 "golang.org/x/exp/iter"  
)  
  
func main() {  
 nums := []int{1, 2, 3, 4, 5}  
 it := iter.Iter(nums)  
  
 fmt.Println(it.Collect()) // [1 2 3 4 5]  
}

## 2️⃣ Map Metodu

Her elemanı verilen fonksiyona uygular ve yeni bir iterator döner.

numbers := []int{1, 2, 3}  
it := iter.Iter(numbers).Map(func(x int) int {  
 return x \* x  
})  
fmt.Println(it.Collect()) // [1 4 9]

## 3️⃣ Filter Metodu

Belirli bir koşulu sağlayan elemanları döndürür.

numbers := []int{1, 2, 3, 4, 5}  
it := iter.Iter(numbers).Filter(func(x int) bool {  
 return x%2 == 0  
})  
fmt.Println(it.Collect()) // [2 4]

## 4️⃣ Reduce Metodu

Iterator’daki tüm elemanları tek bir değere indirger.

numbers := []int{1, 2, 3, 4}  
sum := iter.Iter(numbers).Reduce(0, func(acc, x int) int {  
 return acc + x  
})  
fmt.Println(sum) // 10

## 5️⃣ Collect Metodu

Iterator’daki tüm elemanları slice olarak toplar.

numbers := []int{1, 2, 3}  
it := iter.Iter(numbers)  
collected := it.Collect()  
fmt.Println(collected) // [1 2 3]

## 6️⃣ Take ve Skip

* **Take(n int)**: İlk n elemanı alır.
* **Skip(n int)**: İlk n elemanı atlar ve kalanları döner.

numbers := []int{10, 20, 30, 40, 50}  
it := iter.Iter(numbers)  
  
firstThree := it.Take(3).Collect()  
fmt.Println(firstThree) // [10 20 30]  
  
remaining := it.Skip(3).Collect()  
fmt.Println(remaining) // [40 50]

## 7️⃣ Any ve All

* **Any(f func(T) bool)**: En az bir eleman koşulu sağlıyorsa true döner.
* **All(f func(T) bool)**: Tüm elemanlar koşulu sağlıyorsa true döner.

numbers := []int{2, 4, 6, 8}  
it := iter.Iter(numbers)  
  
fmt.Println(it.All(func(x int) bool { return x%2 == 0 })) // true  
fmt.Println(it.Any(func(x int) bool { return x > 5 })) // true

## 8️⃣ Chain – Iterator’ları Birleştirme

Birden fazla iterator’ı zincirleme şekilde birleştirir.

a := iter.Iter([]int{1, 2})  
b := iter.Iter([]int{3, 4})  
  
combined := iter.Chain(a, b)  
fmt.Println(combined.Collect()) // [1 2 3 4]

## 9️⃣ FlatMap

Her elemanı yeni bir iterator’a dönüştürür ve düzleştirir.

words := []string{"go", "lang"}  
letters := iter.Iter(words).FlatMap(func(s string) iter.Iterator[rune] {  
 return iter.Iter([]rune(s))  
})  
fmt.Println(letters.Collect()) // ['g' 'o' 'l' 'a' 'n' 'g']

## 10️⃣ Enumerate

Elemanları indeks ile birlikte döndürür.

nums := []int{10, 20, 30}  
it := iter.Iter(nums).Enumerate()  
  
for pair := range it {  
 fmt.Println(pair.Index, pair.Value)  
}  
// 0 10  
// 1 20  
// 2 30

## 11️⃣ Zip

İki iterator’ı birleştirip çiftler oluşturur.

a := []int{1, 2, 3}  
b := []string{"a", "b", "c"}  
  
it := iter.Zip(iter.Iter(a), iter.Iter(b))  
fmt.Println(it.Collect()) // [(1,a) (2,b) (3,c)]

## 12️⃣ Cycle ve Repeat

* **Cycle**: Iterator’ı sonsuz döngüye sokar.
* **Repeat**: Belirli bir değeri tekrarlar.

it := iter.Repeat(5).Take(3)  
fmt.Println(it.Collect()) // [5 5 5]

## 13️⃣ Örnek Zincirleme Kullanım

numbers := []int{1, 2, 3, 4, 5, 6}  
  
result := iter.Iter(numbers).  
 Filter(func(x int) bool { return x%2 == 0 }).  
 Map(func(x int) int { return x \* 10 }).  
 Skip(1).  
 Take(1).  
 Collect()  
  
fmt.Println(result) // [40]

## 14️⃣ Özet

* iter paketi ile **fonksiyonel ve zincirleme iterasyonlar** yapılabilir.
* Temel metodlar: Map, Filter, Reduce, Take, Skip, Collect, Any, All.
* İleri kullanım: Chain, FlatMap, Enumerate, Zip, Cycle, Repeat.
* Okunabilir ve kısa kod yazmayı sağlar, klasik for döngüleri yerine tercih edilebilir.