BAB III

Looping

Looping atau perulangan adalah metode yang digunakan untuk mengeksekusi perintah secara berulang pada data yang berbentuk kumpulan, seperti list, tuple, maupun string. Dalam proses perulangan biasanya digunakan **For Loop** dan **While Loop**, dengan opsi tambahan berupa pernyataan **Break** dan **Continue** untuk mengatur jalannya perulangan.

A. For Loop

For di Python adalah perulangan berbasis iterasi yang digunakan untuk menelusuri setiap elemen dalam objek iterable (objek disebut iterable kalau ia bisa "memberikan" elemennya satu per satu saat dipakai di for) seperti list, string, atau dictionary. Jika digunakan langsung dengan iterable , **for** akan menjalankan blok kode untuk setiap item yang ada di dalamnya.

Dengan **for loop**, kita dapat mengeksekusi satu set pernyataan, sekali untuk setiap item yang ada dalam list, tuple, set, dan sebagainya.

Contoh menggunakan **for** dengan **List** :

```
Pets = ["Monkey", "Cat", "Dog"]
for x in Pets:
    print("This is my pet, ", x)
Output :
This is my pet,  Monkey
This is my pet,  Cat
This is my pet,  Dog
```

Contoh menggunakan **For** dengan **String** :

```
for huruf in "Python":
    print(huruf)
Output :
P
y
t
```

```
h
o
n
```

Contoh Menggunakan **For** dengan **range()** :

Jika digunakan bersama **range()**, **for** berfungsi untuk melakukan perulangan dengan jumlah tertentu sesuai nilai yang diberikan pada **range()**, misalnya **range(5)** berarti perulangan dilakukan sebanyak 5 kali dengan nilai 0 sampai 4.

```
for i in range(5):
    print(i)
# Output :
# 0
# 1
# 2
# 3
# 4
```

B. While Loop

While di Python adalah tipe perulangan yang akan terus dijalankan selama kondisinya bernilai True. Loop ini berbeda dengan for loop, yang biasanya menelusuri elemen dalam iterable atau melakukan perulangan sejumlah tertentu menggunakan range(). While lebih fleksibel karena tidak membutuhkan jumlah iterasi yang pasti; perulangan akan terus berjalan selama kondisi yang ditentukan terpenuhi.

While loop sangat cocok digunakan ketika kita belum tahu pasti berapa kali perulangan harus dilakukan. Perulangan hanya akan berhenti ketika kondisi yang diberikan menjadi False, sehingga kita bisa mengontrol alur program berdasarkan logika atau kondisi dinamis. Contohnya, saat meminta input dari pengguna hingga input yang valid diterima, while loop bisa terus berjalan sampai kondisi tersebut terpenuhi.

```
angka = 1
while angka <= 5:
    print(angka)
    angka += 1
Output :
1
2
3
4
```

```
5
```

Contoh penggunaan while loops yang akan menghasilkan perulangan tanpa henti :

```
angka = 1
while angka <= 5:
    print(angka)
```

Karena angka tidak di tambah, jadi angka selalu 1 dan while loops akan berjalan selamanya karena (while angka <= 5) tidak pernah false

Perbedaan **For** dan **While**, yaitu :

- **for** digunakan ketika kita ingin mengulang elemen dalam iterable atau jumlah perulangan sudah jelas (misalnya dengan range())
- **while** digunakan ketika perulangan bergantung pada kondisi tertentu dan berhenti hanya saat kondisi tersebut tidak lagi terpenuhi.

C. Break dan Continue

Break adalah perintah di Python yang digunakan untuk menghentikan perulangan secara langsung sebelum loop selesai. **Continue** adalah perintah yang digunakan untuk melewati iterasi saat ini dan langsung melanjutkan ke iterasi berikutnya. Keduanya biasanya dipakai bersamaan dengan kondisi " **if** " di dalam loop.

1. Break Statement

Break digunakan untuk menghentikan perulangan secara langsung meskipun kondisi atau iterable belum selesai. Perintah ini biasanya dipakai ketika suatu syarat sudah terpenuhi dan kita ingin menghentikan eksekusi loop lebih awal, sehingga sisa perulangan tidak dijalankan. Misalnya dalam perulangan while atau for, break dapat menghentikan proses begitu nilai tertentu tercapai, membuat eksekusi lebih efisien dan menghindari perulangan yang sia-sia.

Selain itu, break dapat digunakan di semua jenis loop dan sering dikombinasikan dengan kondisi if. Dengan cara ini, kita dapat mengontrol alur program secara presisi, misalnya menghentikan loop saat mencari nilai tertentu di list atau ketika proses perhitungan sudah mencapai batas logis. Contoh praktisnya adalah menampilkan perkalian angka 6 hingga nilai tertentu, dan berhenti saat nilai yang diinginkan sudah tercapai.

```
i = 1
while i <= 5:
    print('6 * ', (i), '=', 6 * i)
    if i >= 3:
```

```
        break
    i = i + 1
# Output:
# 6 * 1 = 6
# 6 * 2 = 12
# 6 * 3 = 18
```

2. Continue Statement

Continue digunakan untuk melewati satu iterasi saat ini dan langsung melanjutkan ke iterasi berikutnya, tanpa mengeksekusi sisa kode di dalam loop. Perintah ini berguna ketika kita ingin mengabaikan kondisi tertentu tetapi tetap melanjutkan proses perulangan, misalnya melewati angka genap saat ingin menampilkan angka ganjil saja.

Selain itu, continue juga memungkinkan loop dapat tetap berjalan secara efisien dengan melewati iterasi yang tidak perlu, dan sering dikombinasikan dengan kondisi if untuk menentukan kapan iterasi harus dilewati. Contohnya, dalam while loop yang menampilkan angka 1 hingga 10, kita bisa melewati semua angka genap dengan continue sehingga hanya angka ganjil yang dicetak, tanpa menghentikan keseluruhan perulangan.

```
num = 0
while num < 10:
    num = num + 1
    if (num % 2) == 0:
        continue
    print(num)
# Output:
# 1
# 3
# 5
# 7
# 9
```

3. Else pada loop

Di Python, **else** digunakan untuk mengeksekusi blok kode setelah perulangan selesai **tanpa adanya break**. Artinya, blok else hanya dijalankan jika loop selesai dengan normal, dan tidak dihentikan oleh break. Ini berguna ketika ingin melakukan aksi tambahan setelah loop selesai, misalnya menampilkan pesan bahwa nilai tertentu tidak ditemukan dalam list atau iterable.

Selain itu, penggunaan else sering dikombinasikan dengan break untuk membedakan antara kondisi “nilai ditemukan” dan “nilai tidak ditemukan”. Jika break dijalankan karena nilai ditemukan, maka else tidak dieksekusi. Dengan cara ini, else pada loop membantu mengatur alur program secara logis saat menangani pencarian atau validasi data dalam perulangan.

```
angka = [2, 4, 6, 8, 10]

for x in angka:
    if x == 7:
        print("Ketemu:", x)
        break
else:
    print("Angka 7 tidak ditemukan")

Output:
Angka 7 tidak ditemukan
```

Jika ada angka 7 pada list maka break akan dijalankan dan else tidak akan dijalankan.

D. Nested Loop

Nested loop adalah perulangan yang dijalankan **di dalam perulangan lain**, artinya satu loop berada di dalam loop lain. Setiap iterasi dari loop luar akan memicu seluruh iterasi dari loop dalam. Dengan cara ini, kita bisa membuat perulangan yang lebih kompleks dan mengatur proses yang bergantung pada dua level atau lebih.

Nested loop sangat berguna ketika bekerja dengan data yang **bersusun atau bertingkat**, misalnya membuat tabel, mencetak pola tertentu, atau mengakses elemen dari matriks (list 2D). Dengan nested loop, kita bisa menelusuri data dua dimensi atau lebih secara sistematis dan menghasilkan output yang sesuai dengan struktur data tersebut.

Contoh di dalam perulangan for :

```
buah = ["Apel", "Jeruk", "Mangga"]
warna = ["Merah", "Oranye", "Kuning"]

for b in buah:
    for w in warna:
        print(b, "-", w)
```

Output :

```
Apel - Merah
Apel - Oranye
Apel - Kuning
```

```
Jeruk - Merah
Jeruk - Oranye
Jeruk - Kuning
Mangga - Merah
Mangga - Oranye
Mangga - Kuning
```

Contoh didalam perulangan while :

```
baris = 1
while baris <= 2:          # loop luar untuk baris
    kolom = 1
    while kolom <= 3:      # loop dalam untuk kolom
        print("Baris:", baris, "Kolom:", kolom)
        kolom += 1
    baris += 1
    print()  # pindah baris
```

Output :

```
Baris: 1 Kolom: 1
Baris: 1 Kolom: 2
Baris: 1 Kolom: 3

Baris: 2 Kolom: 1
Baris: 2 Kolom: 2
Baris: 2 Kolom: 3
```