Kontrol Akışı ve Döngüler

Table of contents

Bölüm 2: Kontrol Akışı ve Döngüler

2.1 Kontrol Akışı (Control Flow)

Programlamada, kod satırları genellikle yukarıdan aşağıya doğru sırayla çalıştırılır. Ancak, bazen programın akışını kontrol etmek ve farklı durumlarda farklı kod bloklarını çalıştırmak isteyebiliriz. İşte tam bu noktada **kontrol akışı** kavramı devreye girer.

Kontrol akışı, programın hangi kod satırlarının hangi sırayla çalıştırılacağını belirler. **Karar yapıları** ve **döngüler**, kontrol akışını yönetmek için kullanılan temel araçlardır.

2.1.1 Karar Yapıları (if-elif-else)

Karar yapıları, belirli bir koşula bağlı olarak farklı kod bloklarını çalıştırmamızı sağlar. Python'da en sık kullanılan karar yapısı `if-elif-else` yapısıdır.

`if` İfadesi

`if` ifadesi, belirli bir koşul doğruysa (True) bir kod bloğunu çalıştırır. Eğer koşul yanlışsa (False), kod bloğu çalıştırılmaz.

Sözdizimi:

```
if koşul:
    # Koşul doğruysa çalıştırılacak kod bloğu
```

Örnek:

```
yas = 20

if yas >= 18:
    print("Oy kullanabilirsiniz.")
```

Bu örnekte, `yas` değişkeninin değeri 18'den büyük veya eşit olduğu için `if` bloğundaki kod çalıştırılır ve ekrana "Oy kullanabilirsiniz." yazdırılır.

`else` İfadesi

`else` ifadesi, `if` ifadesindeki koşul yanlışsa çalıştırılacak kod bloğunu belirtir.

Sözdizimi:

```
if koşul:
    # Koşul doğruysa çalıştırılacak kod bloğu
else:
    # Koşul yanlışsa çalıştırılacak kod bloğu
```

Örnek:

```
yas = 16

if yas >= 18:
    print("Oy kullanabilirsiniz.")
else:
    print("Oy kullanamazsınız.")
```

Bu örnekte, `yas` değişkeninin değeri 18'den küçük olduğu için `else` bloğundaki kod çalıştırılır ve ekrana "Oy kullanamazsınız." yazdırılır.

`elif` İfadesi

`elif` ifadesi, birden fazla koşulu kontrol etmek için kullanılır. Eğer ilk `if` ifadesindeki koşul yanlışsa, `elif` ifadesindeki koşul kontrol edilir. Bu işlem, tüm `elif` ifadeleri kontrol edilene kadar devam eder. Eğer hiçbir koşul doğru değilse, `else` bloğundaki kod çalıştırılır.

Sözdizimi:

```
if koşul1:
    # Koşul1 doğruysa çalıştırılacak kod bloğu
elif koşul2:
    # Koşul2 doğruysa çalıştırılacak kod bloğu
elif koşul3:
    # Koşul3 doğruysa çalıştırılacak kod bloğu
else:
    # Hiçbir koşul doğru değilse çalıştırılacak kod bloğu
```

Örnek:

```
puan = 75

if puan >= 90:
    print("Harf notunuz: AA")
elif puan >= 80:
    print("Harf notunuz: BA")
elif puan >= 70:
    print("Harf notunuz: BB")
else:
    print("Harf notunuz: CB")
```

Bu örnekte, `puan` değişkeninin değeri 75 olduğu için `puan >= 70` koşulu doğru olur ve "Harf notunuz: BB" yazdırılır.

İç İçe `if` Yapıları

Bir `if` ifadesinin içine başka bir `if` ifadesi yerleştirebiliriz. Bu, **iç içe `if` yapıları** olarak adlandırılır. İç içe `if` yapıları, birden fazla koşulu kontrol etmemizi sağlar.

Örnek:

```
yas = 25
ehliyet = True

if yas >= 18:
    if ehliyet:
        print("Araba kullanabilirsiniz.")
    else:
        print("Ehliyetiniz olmadığı için araba kullanamazsınız.")
else:
    print("Yaşınız 18'den küçük olduğu için araba kullanamazsınız.")
```

Bu örnekte, öncelikle `yas` değişkeninin değeri kontrol edilir. Eğer `yas` 18'den büyük veya eşitse, `ehliyet` değişkeninin değeri kontrol edilir. Eğer `ehliyet True` ise, "Araba kullanabilirsiniz." yazdırılır. Eğer `ehliyet False` ise, "Ehliyetiniz olmadığı için araba kullanamazsınız." yazdırılır. Eğer `yas` 18'den küçükse, "Yaşınız 18'den küçük olduğu için araba kullanamazsınız." yazdırılır.

Koşul İfadelerinde Karşılaştırma Operatörlerinin Birleştirilmesi

`and`, `or` ve `not` operatörlerini kullanarak birden fazla koşulu birleştirebiliriz.

Örnek:

```
yas = 25
ehliyet = True

if yas >= 18 and ehliyet:
    print("Araba kullanabilirsiniz.")
else:
    print("Araba kullanamazsınız.")
```

Bu örnekte, hem `yas >= 18` hem de `ehliyet` koşullarının doğru olması durumunda "Araba kullanabilirsiniz." mesajı yazdırılır.

2.1.2 Döngüler (Loops)

Döngüler, belirli bir kod bloğunu tekrar tekrar çalıştırmamızı sağlar. Python'da iki temel döngü türü vardır: `for` döngüsü ve `while` döngüsü.

`for` Döngüsü

`for` döngüsü, bir dizi (sequence) üzerinde yineleme yapmak için kullanılır. Dizi, listeler, demetler, stringler veya range nesneleri gibi birden fazla eleman içeren veri tipleri olabilir. `for` döngüsü, dizinin her elemanı için bir kez çalıştırılır.

Sözdizimi:

```
for eleman in dizi:
    # Her eleman için çalıştırılacak kod bloğu
```

Örnek:

```
meyveler = ["elma", "armut", "muz"]

for meyve in meyveler:
    print(meyve)
```

Bu örnekte, `meyveler` listesindeki her bir meyve için `print(meyve)` kodu çalıştırılır. Çıktı şu şekilde olur:

```
elma
armut
muz
```

`range()` Fonksiyonu ile `for` Döngüsü

`range()` fonksiyonu, sayı dizileri oluşturmak için kullanılır. `for` döngüsü ile birlikte sıklıkla kullanılır.

Sözdizimi:

```
range(başlangıç, bitiş, adım)
```

- **başlangıç:** Dizinin başlangıç değeri (varsayılan 0).
- **bitiş:** Dizinin bitiş değeri (bitiş değeri diziye dahil değildir).
- **adım:** Dizinin her elemanı arasındaki fark (varsayılan 1).

Örnekler:

```
# 0'dan 4'e kadar (5 dahil değil) sayıları yazdırır
for i in range(5):
    print(i, end=" ")

# 1'den 10'a kadar (10 dahil değil) 2'şer atlayarak sayıları yazdırır
for i in range(1, 10, 2):
    print(i)
```

```python
# 5'ten 1'e kadar (1 dahil değil) geriye doğru sayıları yazdırır
for i in range(5, 0, -1):
    print(i, end=" ")
```

`print()` Fonksiyonunda `end` Parametresi

`print()` fonksiyonu, varsayılan olarak her çıktıdan sonra yeni bir satıra geçer. `end` parametresi ile bu davranışı değiştirebilir ve satır sonu karakterini özelleştirebiliriz.

Örnek:

```python
for i in range(5):
    print(i, end=" ")  # Her sayıdan sonra boşluk karakteri ekler
# Çıktı: 0 1 2 3 4
```

Stringler Üzerinde `for` Döngüsü

`for` döngüsünü stringler üzerinde de kullanabiliriz. Bu durumda, döngü string içindeki her bir karakter için bir kez çalıştırılır.

Örnek:

```python
mesaj = "Merhaba Dünya!"

for harf in mesaj:
    print(harf)
```

Bu örnekte, `mesaj` stringindeki her bir karakter için `print(harf)` kodu çalıştırılır. Çıktı şu şekilde olur:

```
M
e
r
h
a
b
a

D
ü
n
y
a
!
```

`while` Döngüsü

`while` döngüsü, belirli bir koşul doğru olduğu sürece bir kod bloğunu tekrar tekrar çalıştırır. Koşul yanlış olduğunda döngü sona erer.

Sözdizimi:

```
while koşul:
    # Koşul doğru olduğu sürece çalıştırılacak kod bloğu
```

Örnek:

```
sayac = 0

while sayac < 5:
    print(sayac)
    sayac += 1
```

Bu örnekte, `sayac` değişkeni 5'ten küçük olduğu sürece `while` döngüsü çalışır. Her döngüde, `sayac` değişkeninin değeri ekrana yazdırılır ve ardından 1 artırılır. Çıktı şu şekilde olur:

```
0
1
2
3
4
```

Döngülerde Kullanıcı Girişi

Döngüler içinde kullanıcıdan veri almak için `input()` fonksiyonunu kullanabiliriz.

Örnek:

```
toplam = 0

while True:
    sayi = input("Bir sayı girin (Çıkmak için 'q' tuşuna basın): ")
    if sayi == 'q':
        break
    sayi = int(sayi)
    toplam += sayi

print("Girdiğiniz sayıların toplamı:", toplam)
```

Bu örnekte, kullanıcı “q” tuşuna basana kadar döngü devam eder ve girilen sayıları toplar.

Sonsuz Döngüler ve `break` İfadesi

`while True` yapısı, sonsuz bir döngü oluşturur. Döngüyü sonlandırmak için `break` ifadesini kullanabiliriz.

Örnek:

```
sayac = 0

while True:
    print(sayac)
    sayac += 1
```

Bu örnekte döngüyü durduracak bir ifade olmadığı için sürekli çalışır.

2.2 Döngü Kontrol İfadeleri

Döngüler içinde, döngünün akışını kontrol etmek için `break` ve `continue` ifadelerini kullanabiliriz.

2.2.1 `break` İfadesi

`break` ifadesi, döngüyü hemen sonlandırır. Döngü içindeki kodun geri kalanı çalıştırılmaz.

Örnek:

```
sayac = 0

while True:
    print(sayac)
    sayac += 1
    if sayac == 10:
        break
```

Bu örnekte `sayac` `10`değerine ulaştığında döngü sonlanır.

2.2.2 `continue` İfadesi

`continue` ifadesi, döngünün mevcut iterasyonunu (tekrarını) atlar ve bir sonraki iterasyondan devam eder.

Örnek:

```
sayilar = [1, 2, 3, 4, 5]

for sayi in sayilar:
    if sayi == 3:
        continue
    print(sayi)
```

Bu örnekte, `for` döngüsü `sayilar` listesindeki her bir sayı için çalışır. Ancak, `sayi` değişkeni 3 olduğunda `continue` ifadesi çalışır ve bu iterasyon atlanır. Çıktı şu şekilde olur:

```
1
2
4
5
```

2.2.3 `pass` İfadesi

`pass` ifadesi, Python'da hiçbir şey yapmayan bir ifadedir. Genellikle kod bloklarının henüz tamamlanmadığı veya geçici olarak boş bırakılması gereken durumlarda kullanılır.

Örnek:

```
sayi = 5

if sayi > 10:
    # Burada henüz bir kod yok, ancak Python hata vermesin diye pass ifadesi kullanıyoruz
    pass
else:
    print("Sayı 10'dan küçük veya eşit")
```

Bu örnekte, `if` bloğu henüz tamamlanmamış. Eğer `pass` ifadesini kullanmazsak, Python bir `IndentationError` hatası verecektir. `pass` ifadesi sayesinde, kod bloğunu boş bırakabilir ve hata almadan programımızı çalıştırabiliriz.

`pass` ifadesi, fonksiyon veya sınıf tanımlarında da kullanılabilir. Örneğin:

```python
def benim_fonksiyonum():
    # Fonksiyonun gövdesi henüz yazılmadı
    pass

class BenimSinifim:
    # Sınıfın gövdesi henüz yazılmadı
    pass
```

Bu şekilde, fonksiyon veya sınıfı daha sonra tanımlamak üzere boş bırakabilir ve programımızı hata almadan çalıştırabiliriz.

`pass` İfadesinin Kullanım Alanları

- **Henüz tamamlanmamış kod blokları:** Kod yazarken, bazı bölümleri daha sonra tamamlamak isteyebilirsiniz. Bu durumlarda, Python'un hata vermemesi için `pass` ifadesini kullanabilirsiniz.
- **Boş fonksiyon veya sınıf tanımları:** Fonksiyon veya sınıfı daha sonra tanımlamak üzere boş bırakmak istediğinizde `pass` ifadesini kullanabilirsiniz.
- **Koşullu ifadelerde:** Belirli bir koşul sağlandığında hiçbir şey yapılmasını istemediğiniz durumlarda `pass` ifadesini kullanabilirsiniz.

2.3 İç İçe Döngüler (Nested Loops)

Bir döngü bloğu içinde başka bir döngü kullanabiliriz. Bu, **iç içe döngüler (nested loops)** olarak adlandırılır. İç içe döngüler, karmaşık problemleri çözmek için kullanılır.

Örnek:

```python
for i in range(3):
    for j in range(2):
        print(i, j)
```

Bu örnekte, dıştaki `for` döngüsü 3 kez çalışır ve her döngüde içteki `for` döngüsü 2 kez çalışır. Çıktı şu şekilde olur:

```
0 0
0 1
1 0
1 1
2 0
2 1
```

2.4 Match-Case Yapısı (Python 3.10 ve Sonrası)

`match-case` yapısı, bir değeri birden fazla desenle karşılaştırmak ve eşleşen desene göre farklı kod bloklarını çalıştırmak için kullanılır.

Sözdizimi:

```
match değişken:
    case desen1:
        # Desen1 ile eşleşirse çalıştırılacak kod bloğu
    case desen2:
        # Desen2 ile eşleşirse çalıştırılacak kod bloğu
    case _:
        # Hiçbir desenle eşleşmezse çalıştırılacak kod bloğu
```

Örnek:

```
gun = "Pazartesi"

match gun:
    case "Pazartesi":
        print("Haftanın ilk günü")
    case "Cumartesi" | "Pazar":
        print("Haftasonu")
    case _:
        print("Hafta içi")
```

Bu örnekte, `gun` değişkeninin değeri "Pazartesi" olduğu için ilk `case` ile eşleşir ve "Haftanın ilk günü" yazdırılır.

2.5 Bölüm Sonu Quiz

1. Aşağıdaki `for` döngüsü kaç kez çalışır?

   ```
   for i in range(1, 10, 2):
       print(i)
   ```

2. `while` döngüsünü sonlandırmak için hangi ifade kullanılır?

3. Aşağıdaki kod bloğunun çıktısı nedir?

```
x = 15
y = 4

if x % y == 0:
    print("x, y'ye tam bölünür")
else:
    print("x, y'ye tam bölünmez")
```

4. Aşağıdaki kod bloğunda boşluklara hangi ifadeler gelmelidir?

```
sayi = 7

_______ sayi > 10:
    print("Sayı 10'dan büyük")
_______:
    print("Sayı 10'dan küçük veya eşit")
```

5. Aşağıdaki kod bloğunun çıktısı nedir?

```
toplam = 0

for i in range(1, 6, 2):
    toplam += i

print(toplam)
```

6. Aşağıdaki kod bloğunun çıktısı nedir?

```
sayac = 1

while sayac <= 5:
    if sayac == 3:
        sayac += 1
        continue
    print(sayac)
    sayac += 1
```

7. Aşağıdaki kod bloğunun çıktısı nedir?

```
for i in range(2):
    for j in range(3):
        if j == 1:
            break
        print(i, j)
```