Python'da denetim yapıları, döngüler ve fonksiyonlar, programın akışını kontrol etmek ve belirli görevleri tekrar etmek için kullanılan temel araçlardır. Aşağıda bu konulara dair detaylı açıklamalar ve örnekler bulabilirsiniz:

1. Denetim Yapıları

Denetim yapıları, programın akışını koşullara göre yönlendiren yapılardır. Python'da en yaygın kullanılan denetim yapıları `if`, `elif` ve `else` ifadeleridir.

If-Else Yapısı

Bir koşulun doğru olup olmadığını kontrol eder ve buna göre farklı kod bloklarını çalıştırır.

Örnek:

```
x = 10

if x > 5:
    print("x, 5'ten büyük.")
elif x == 5:
    print("x, 5'e eşit.")
else:
    print("x, 5'ten küçük.")
```

Nested If (İç içe If)

İç içe if yapıları, daha karmaşık koşullar için kullanılır.

Örnek:

```
x = 15

if x > 10:
    if x < 20:
```

```
        print("x, 10 ile 20 arasında.")
    else:
        print("x, 20'den büyük.")
else:
    print("x, 10'dan küçük.")
```

2. Döngüler

Döngüler, bir kod bloğunu belirli bir sayıda veya koşul sağlandığı sürece tekrar tekrar çalıştırmak için kullanılır. Python'da iki temel döngü yapısı vardır: `for` ve `while`.

For Döngüsü

For döngüsü, bir dizi ya da liste gibi tekrarlanabilir bir yapı üzerinde gezinmek için kullanılır.

Örnek:

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

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

Range ile For Döngüsü

Belirli bir aralıkta döngü oluşturmak için `range()` fonksiyonu kullanılabilir.

Örnek:

```
for i in range(5):
    print(i)
```

Bu döngü 0'dan başlayarak 4'e kadar olan sayıları yazdırır.

While Döngüsü

While döngüsü, koşul doğru olduğu sürece tekrar eden bir döngüdür.

Örnek:

```
i = 0

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

Bu döngü, `i` 5 olana kadar çalışır.

Döngü Kontrol İfadeleri

Döngülerde döngüyü kontrol etmek için `break` ve `continue` ifadeleri kullanılır.

- **break**: Döngüyü sonlandırır.
- **continue**: Döngünün o anki iterasyonunu atlayıp bir sonrakine geçer.

Örnek (break):

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

Örnek (continue):

```
for i in range(10):
    if i == 5:
        continue
    print(i)
```

3. Fonksiyonlar

Fonksiyonlar, tekrar tekrar kullanılabilir kod bloklarıdır. Python'da fonksiyonlar `def` anahtar kelimesi ile tanımlanır.

Basit Fonksiyon Tanımlama

Bir fonksiyon, belirli bir görevi yerine getiren bir kod bloğudur.

Örnek:

```
def selam_ver():
    print("Merhaba!")
```

Bu fonksiyon çağrıldığında ekrana "Merhaba!" yazdırır.

```
selam_ver()
```

Parametre Alan Fonksiyonlar

Fonksiyonlara parametreler vererek farklı girdilerle çalıştırılabilir.

Örnek:

```
def toplama(a, b):
    return a + b
```

Bu fonksiyon iki sayıyı toplar ve sonucu döndürür.

```
sonuc = toplama(3, 5)
print(sonuc)  # 8
```

Varsayılan Parametreler

Fonksiyonlarda varsayılan parametreler de tanımlanabilir.

Örnek:

```
def selam_ver(isim="Dünya"):
    print(f"Merhaba, {isim}!")
```

Parametre verilmediğinde varsayılan olarak "Dünya" kullanılır.

```
selam_ver()  # Merhaba, Dünya!
selam_ver("Ali")  # Merhaba, Ali!
```

Anonim Fonksiyonlar (Lambda)

Kısa ve tek satırlık fonksiyonlar için `lambda` kullanılır.

Örnek:

```
topla = lambda a, b: a + b
print(topla(3, 4))  # 7
```

Fonksiyonlarda `*args` ve `**kwargs` kullanımı, fonksiyonların esnek parametrelerle çalışabilmesini sağlar. Bu sayede fonksiyonlara değişken sayıda parametre gönderebiliriz.

`*args`

`*args`, bir fonksiyona **değişken sayıda** konumlu (positional) argüman göndermek için kullanılır. `*args`, fonksiyon içinde bir **tuple** olarak tutulur. Bu sayede kaç tane argüman gönderildiğini bilmemize gerek kalmaz; fonksiyon, tümünü işleyebilir.

Örnek:

```
def toplama(*args):
    return sum(args)

# Farklı sayıda parametre ile fonksiyonu çağırabiliriz
print(toplama(1, 2, 3))  # 6
print(toplama(4, 5))     # 9
print(toplama(10))       # 10
```

Bu örnekte `*args` sayesinde fonksiyona kaç parametre verileceğini bilmemize gerek yoktur. Fonksiyon, aldığı tüm parametreleri bir tuple içinde toplar ve `sum()` ile bunların toplamını döner.

`**kwargs`

`**kwargs` ise bir fonksiyona **değişken sayıda anahtar-değer çiftleri** (keyword arguments) göndermek için kullanılır. `**kwargs`, fonksiyon içinde bir **dictionary** olarak tutulur.

Örnek:

```
def kisi_bilgisi(**kwargs):
    for anahtar, deger in kwargs.items():
        print(f"{anahtar}: {deger}")

# Fonksiyona farklı anahtar-değer çiftleri ile veri gönderebiliriz
kisi_bilgisi(isim="Ali", yas=30, sehir="İstanbul")
# isim: Ali
# yas: 30
# sehir: İstanbul
```

Bu örnekte, `**kwargs` ile fonksiyona farklı sayıda anahtar-değer çifti gönderebiliriz. Fonksiyon, bu çiftleri bir sözlük (dictionary) olarak alır ve işlemler yapar.

`*args` ve `**kwargs` Birlikte Kullanımı

Bir fonksiyonda hem `*args` hem de `**kwargs` birlikte kullanılabilir. Bu durumda `*args` konumlu argümanları, `**kwargs` ise anahtar-değer argümanlarını temsil eder.

Örnek:

```
def karisik_fonksiyon(*args, **kwargs):
    print("Positional arguments (args):", args)
```

```
    print("Keyword arguments (kwargs):", kwargs)

karisik_fonksiyon(1, 2, 3, isim="Ali", yas=30)
```

Çıktı:

```
Positional arguments (args): (1, 2, 3)
Keyword arguments (kwargs): {'isim': 'Ali', 'yas': 30}
```

Bu örnekte, `*args` ile 1, 2 ve 3 konumlu argüman olarak alınırken, `**kwargs` ile "isim" ve "yas" anahtar-değer çiftleri alınır.

Özet

- `*args`: Değişken sayıda konumlu argüman alır ve bir tuple içinde saklar.
- `kwargs`**: Değişken sayıda anahtar-değer çifti alır ve bir dictionary içinde saklar.

Bu kavramlar, fonksiyonlara daha esnek bir yapı kazandırarak, dinamik ve esnek kod yazmayı sağlar.

Özet

Python'da denetim yapıları, döngüler ve fonksiyonlar, programın akışını yönetmek, tekrar eden işlemleri gerçekleştirmek ve kodu daha modüler hale getirmek için kullanılır. Denetim yapıları koşullara göre farklı kod bloklarını çalıştırırken, döngüler belirli işlemleri tekrarlar. Fonksiyonlar ise kodun daha düzenli ve yeniden kullanılabilir olmasını sağlar.