Karar Yapıları

Python'da en yaygın kullanılan karar yapısı **"if-else"** yapısıdır. Bu yapı, belirli bir koşulun doğru veya yanlış olmasına göre programın ne yapacağını belirler. Örneğin, "eğer bir sayı 10'dan büyükse **'büyük'** yaz, değilse **'küçük'** yaz" gibi bir durumu yönetmek için **"if-else"** kullanılır. Koşullar, mantıksal ifadelerle (örneğin, `==`, `>`, `<`) belirlenir ve bu ifadeler doğru olduğunda ilgili kod bloğu çalıştırılır. Python ayrıca **"elif"** anahtar kelimesiyle çoklu koşulların kontrol edilmesini sağlar. Böylece, birden fazla durum kontrol edilebilir.

"if" komutu tek başına bir koşulun sağlandığı durumlar ile kullanılırken, **"if-else"** çifti bir koşulun sağlandığı ve sağlanmadığı şeklinde iki farklı durumlar için kullanılır. **"if-elif"** komutu ise 2'den fazla koşulların sağlandığı durumlar için kullanılır ve iç-içe geçmiş birçok koşulların kontrolü için de kullanır. Bu durumların genel olarak yazım düzenleri şöyledir:

Tek koşullu durum

```
if koşul:
    .
    . işlem
    .
```

İki koşullu durum

```
if koşul:
    .
    . işlem
    .
else:
    .
    . işlem
    .
```

Çok koşullu durum

```
if koşul1:
    .
    . işlem
    .
elif koşul2:
    .
    . işlem
    .
else:
    .
    . işlem
    .
```

"if" ve **"elif"** komutlarındaki koşul durumları karşılaştırma operatörleri ile kontrol edilir. Karşılaştırma operatörleriyle bir değişken belirli sabitlerle kıyaslandığı gibi farklı değişkenler de birbirleriyle kıyaslanabilir. Yazım düzeninde her bir koşuldan ve else kelimesinde sonra ":" sembolü zorunludur ve ":" dan sonraki satırda girinti olmalıdır.

if Koşul örnekleri

"Eğer n sayısının değeri 10'dan büyükse, 'sayı 10'dan büyüktür' mesajı yazılsın" şeklinde bir ifadeyi dikkate alalım. Burada koşulumuz sayısının 10'dan büyük olmasıdır. Bu durumu Python'da ifade edersek,

```
if n > 10:
      print("sayı 10'dan büyüktür")
```

şeklinde olacaktır. Burada ":" ve sonraki satırdaki girintiye dikkat edilmelidir. IDLE editöründe yeni dosya oluşturup aşağıdaki kodları yazalım ve **sayikontrol.py** olarak kaydedelim.

```
sayı = int(input("Bir sayı giriniz: "))

if sayı > 10:
    print("Girdiğiniz sayı 10'dan büyüktür!")

if sayı < 10:
    print("Girdiğiniz sayı 10'dan küçüktür!")

if sayı == 10:
    print("Girdiğiniz sayı 10'dur!")
```

Gördüğünüz gibi, art arda üç adet **"if"** bloğu kullandık. Bu kodlara göre,

- eğer kullanıcının girdiği sayı 10'dan büyükse, ilk **"if"** bloğu işletilecek,
- eğer sayı 10'dan küçükse ikinci **"if"** bloğu işletilecek
- eğer sayı 10'a eşit ise üçüncü **"if"** bloğu işletilecektir.

Bu programı çalıştırıp farklı sayılar girerek test edelim.

if-else Koşul örnekleri

Şimdide bir sisteme şifre ile giriş durumunu dikkate alalım. Burada kullanıcı bir şifre girecek ve doğru ise "Sisteme hoş geldiniz" mesajı aksi durumda "Yanlış şifre girdiniz" mesajı iletilecek olsun. IDLE editöründe yeni dosya oluşturup aşağıdaki kodları yazalım ve **sistem.py** olarak kaydedelim.

```
print("""
Üniversite kütüphane hizmetine
Hoş geldiniz. Yalnız hizmetimizden
yararlanmak için önce sisteme giriş
yapmalısınız.
""")

sifre = input("Sifre: ")

if sifre == "12345678":
    print("Sisteme hoş geldiniz!")

else:
    print("Yanlış şifre girdiniz!")
```

Programın başında üç tırnak işaretlerinden yararlanarak uzun bir metni kullanıcıya gösterdik. Bu bölümü, kendiniz göze hoş gelecek bir şekilde süsleyebilirsiniz de. Eğer kullanıcı, kendisine parola sorulduğunda cevap olarak "12345678" yazarsa kullanıcıyı sisteme alıyoruz. Kullanıcının girdiği parola “12345678” ise kendisine "Sisteme hoş geldiniz " mesajını aksi durumda "Yanlış şifre girdiniz" mesajını gösteriyoruz.

İf-else koşul örnekleri

Klavyeden kullanıcının girdiği bir metinin, arzu edilen metin olup olmadığını kontrol eden bir program yazmaya çalışalım. Eğitici bir oyun yazmaya çalışalım, oyunumuzda bir tanım ve karşılığında bir kelime girilsin. Girilen kelime doğru ise bunu kullanıcıya bildirelim. IDLE editöründe yeni dosya oluşturup aşağıdaki kodları yazalım ve **cevaptest**.py olarak kaydedelim.

```
print("Çorum, Sivas ve Kayseri ile komşu olan ilimiz hangisidir?")
girilen = input("Cevabınız: ")

if girilen == 'yozgat':
   print("Tebrikler cevabınız doğru")
```

Program çalıştırıldığında aşağıdaki şekilde görünecektir.

```
Çorum, Sivas ve Kayseri ile komşu olan ilimiz hangisidir?
Cevabınız: yozgat
Tebrikler cevabınız doğru
```

Burada dikkat edilmesi gereken bir durum vardır. Eğer kullanıcı şehir isminin ilk harfini büyük olarak girerse cevap uygun olamayacaktır. Çünkü Python büyük-küçük harf ayırımı yapmaktadır, yani "yozgat" ile "Yozgat" aynı değildir. Bu nedenle "if" satırı şu şekilde düzenlenmelidir.

```
if (girilen == 'yozgat') or (girilen == 'Yozgat'):
```

Bu şekilde istenilen doğru cevap her iki hal için de kabul edilmiş olacaktır.

İf-else koşul örnekleri

Şimdi önceki örnekte verilen cevabın uygun olmadığı durumu dikkate alalım. IDLE editöründe **cevaptest.py** kodunu aşağıdaki gibi düzenleyelim ve **cevaptest2**.py olarak kaydedelim.

```
print("Çorum, Sivas ve Kayseri ile komşu olan ilimiz hangisidir?")
girilen = input("Cevabınız: ")

if (girilen == 'yozgat') or (girilen == 'Yozgat'):
    print("Tebrikler cevabınız doğru.")

else:
    print("Üzgünüm cevabınız doğru değil. Yeniden deneyin")
```

Program çalıştırılıp cevap olarak "bursa" verildiğinde çıktı aşağıdaki şekilde görünecektir.

```
Çorum, Sivas ve Kayseri ile komşu olan ilimiz hangisidir?
Cevabınız: bursa
Üzgünüm cevabınız doğru değil. Yeniden deneyin
```

Ancak cevap olarak "yozgat" ya da "Yozgat" verildiğinde ise çıktı "Tebrikler cevabınız doğru." olacaktır.

Böyle farklı yazılış durumlarını dikkate aldığımızda bir çok koşulu ayrı ayrı yazmak yerine girilen cevap metnini "küçük harflere çevirmek" daha kolay bir çözümdür. Burada "if lower(girilen) == 'yozgat':" şeklinde yazmak daha uygun olacaktır.

if-elif koşul örnekleri

Bir kullanıcının yaşını alıp yaşına göre belirli yorumlar yapılan bir durumunu dikkate alalım. IDLE editöründe yeni bir dosya oluşturup aşağıdaki kodları ayrı ayrı yazalım ve **yaskontrol1.py** ve **yaskontrol2.py** olarak kaydedelim.

```
yaş = int(input("Yaşınız: "))

if yaş == 18:
    print("18 iyidir!")

elif yaş < 0:
    print("Yok canım, daha neler!...")

elif yaş < 18:
    print("Genç bir kardeşimizsin!")

elif yaş > 18:
    print("Eh, artık büyüdün sayılır!")
```

```
yaş = int(input("Yaşınız: "))

if yaş == 18:
    print("18 iyidir!")

if yaş < 0:
    print("Yok canım, daha neler!...")

if yaş < 18:
    print("Genç bir kardeşimizsin!")

if yaş > 18:
    print("Eh, artık büyüdün sayılır!")
```

Bu iki programın da aynı işlevi gördüğünü düşünebilirsiniz. Ancak ilk bakışta pek belli olmasa da, aslında yukarıdaki iki program birbirinden farklı davranacaktır. Örneğin ikinci programda eğer kullanıcı eksi değerli bir sayı girerse hem if yaş < 0 bloğu, hem de if yaş < 18 bloğu çalışacaktır. Bunu görmek için yukarıdaki programı çalıştırıp, cevap olarak eksi değerli bir sayı verelim.

if-elif diğer bir örnek

if ile elif arasında uygulama açısından önemli farklılıklar bulunur. if fonksiyonu bize olası bütün sonuçları listeler, elif fonksiyonu ise sadece doğru olan ilk sonucu verir. Aşağıdaki örneklere bir bakalım

ifkodu_1.py

```
a = int(input("Bir sayı giriniz: "))

if a < 100:
    print("verdiğiniz sayı 100'den küçüktür.")

if a < 50:
    print("verdiğiniz sayı 50'den küçüktür.")

if a == 100:
    print("verdiğiniz sayı 100'dür.")

if a > 100:
    print("verdiğiniz sayı 100'den büyüktür.")

if a > 150:
    print("verdiğiniz sayı 150'den büyüktür.")
```

ifkodu_2.py

```
a = int(input("Bir sayı giriniz: "))

if a < 100:
    print("verdiğiniz sayı 100'den küçüktür.")

elif a < 50:
    print("verdiğiniz sayı 50'den küçüktür.")

elif a == 100:
    print("verdiğiniz sayı 100'dür.")

elif a > 100:
    print("verdiğiniz sayı 100'den büyüktür.")

elif a > 150:
    print("verdiğiniz sayı 150'den büyüktür.")
```

ifkodu_1.py ve ifkodu_2.py kodlarını çalıştırılıp ve sayı olarak 40 verildiğini varsayalım. Buna göre çıktılar şöyle olacaktır:

ifkodu_1.py çıktısı

```
verdiğiniz sayı 100'den küçüktür.
verdiğiniz sayı 50'den küçüktür.
```

ifkodu_2.py çıktısı

```
verdiğiniz sayı 100'den küçüktür.
```

if-elif diğer bir örnek

Bir kullanıcının girdiği ders puanını kontrol eden ve puana göre harf notunu yazan bir durumu dikkate alalım. IDLE editöründen bir dosya açıp aşağıdaki kodları yazalım ve **notkontrol.py** olarak kaydedelim.

```
puan = int(input("Öğrenci puanının girin: "))

if puan > 85:
    karne_notu = "A"

elif puan > 70:
    karne_notu = "B"

elif puan > 60:
    karne_notu = "C"

elif puan > 50:
    karne_notu = "D"

else:
    karne_notu = "F"

print("Karne notu: %s", karne_notu)
```

```
puan = int(input("Öğrenci puanının girin: "))

if   puan > 85: karne_notu = "A"
elif puan > 70: karne_notu = "B"
elif puan > 60: karne_notu = "C"
elif puan > 50: karne_notu = "D"
else: karne_notu = "F"

print("Karne notu: %s", karne_notu)
```

Görüldüğü üzere harf notları büyükten küçüğe göre sıralanarak verilmiştir. Bu sadece işlem kolaylığı içindir. Belirli aralıklar olarak yazılması durumda daha fazla kod yazımı söz konusu olacaktır.

```
if (puan > y0 ) & (puan <= 80): karne_notu = "B"
if (puan > 60 ) & (puan <= 70): karne_notu = "C"
if (puan > 50 ) & (puan <= 60): karne_notu = "D"
```

İç-içe geçmiş (nested) if-else-elif yapıları

"iç içe if-else-elif yapıları", bir koşulun içinde başka bir koşulu kontrol etmek için kullanılır. Bu yapılar, daha karmaşık karar verme süreçlerini yönetmek veya koşullara bağlı olarak adım adım işlemler gerçekleştirmek gerektiğinde tercih edilir. İç içe koşul yapıları, dış koşul doğruysa, iç koşulların da kontrol edilerek programın belirli bir yöne yönlendirilmesini sağlar. Bu iç-içe yapıların kullanım amaçları şöyle özetlenebilir:

1. **Karmaşık Karar Verme Süreçleri:** İç içe yapılar, bir ana koşul sağlandığında başka koşulları da kontrol ederek, daha ince ayrıntılara dayalı kararlar almayı sağlar.
2. **Adım Adım Kontrol:** Bazen bir durumu aşama aşama kontrol etmek gerekebilir. Bu aşamalı kontrol iç içe if yapıları ile kolayca sağlanabilir.
3. **Daha Detaylı Kontroller:** Ana koşulun doğru olup olmadığını kontrol ettikten sonra, alt koşullarla daha hassas kontroller yapabilirsiniz.

İç-içe yapılar karşılaştırılan koşulların sayısına bağlı olarak oldukça karmaşık yapılarda olabilirler. Bunun en basit hali olan 3 koşullu bir durumu dikkate alırsak iç-içe yapı şöyle görünecektir.

```
if koşul_1:
    işlemler_1_1

    if koşul_2:
        işlemler_2_1
    else:
        işlemler_2_2

else:

    if koşul_3:
        işlemler_3_1
    else:
        işlemler_3_2
```

Dikkat Edilmesi Gerekenler

- **Kodun okunabilirliği**: İç içe yapılar mantıksal olarak faydalı olsa da, aşırı kullanımı kodu karmaşık hale getirebilir. Bu nedenle, mümkün olduğunca kodun anlaşılır ve temiz olmasına dikkat edilmelidir.
- **Girintiler:** Python'da bloklar girintilerle tanımlandığı için, iç içe if yapılarında girintilere dikkat etmek çok önemlidir. Hatalı girintiler, beklenmedik sonuçlara yol açabilir.

İç-içe if-else-elif yapısı örneği

Bir öğrencinin sınıf geçme notuna göre başarı durumunu değerlendirmek istediğimizi varsayalım. Notu 50'nin üzerinde olan öğrencilere **"geçer not"** ile, notu 80'ın üzerinde olan öğrencilere **"teşekkür belgesi"** ile, notu 90'ın üzerinde olan öğrencileri **"onur belgesi"** ile mezun olarak değerlendirmek istiyoruz.

```
geçme_notu = 95
if geçme_notu >= 50:
    print("Sınıfı Geçtiniz.")
    if geçme_notu >= 90:
        print("Onur Belgesi kazandınız.")
    elif geçme_notu >= 80:
        print("Teşekkür Belgesi kazandınız.")
    else:
        print("Tebrikler.")

else:
    print("Maalesef sınıfta kaldınız...")
```

- İlk if bloğu: "geçme_notu >= 50" koşulu kontrol edilir. Eğer doğruysa (ki burada 95 olduğu için doğru), öğrenci geçmiştir.

- İkinci if bloğu (iç içe if): " geçme_ notu >= 90" koşulu tekrar kontrol edilir. Eğer doğruysa (bu örnekte doğru), öğrenci onur belgesi kazanmıştır. Eğer bu koşul doğru değilse, "elif" ve "else" blokları kontrol edilir.

- Eğer ilk if bloğundaki koşul sağlanmamış olsaydı, yani notu 50'nin altında olsaydı, program direk "else" bloğuna geçer ve "Maalesef sınıfta kaldınız" mesajını yazdırırdı.

İç-içe if-else-elif yapısı diğer bir örnek

Girilen her hangi bir tamsayının 2'ye ve 3'e bölünebilme koşullarını kontrol ederek sayının her ikisine ya da sadece birine bölünebildiği durumları kontrol edelim. Daha bu durumları belirten mesajları hazırlayalım. İlk adımda kullanıcıdan bir sayı girişi yapıldığını varsayıyoruz.

```
sayi = int(input("Bir sayı giriniz: "))
if sayi%2 == 0:
    if sayi%3 == 0:
         print("{} sayısı hem 2'ye hem de 3'e bölünmektedir.".format(sayi))
    else:
          print("{} sayısı sadece 2'e bölünmektedir.".format(sayi))
else:
    if sayi%3 == 0:
         print("{} sayısı sadece 3'ye bölünmektedir.".format(sayi))
    else:
          print("{} sayısı 2'ye ve 3'e bölünmeMEktedir.".format(sayi))
```

Programı çalıştırıp farklı sayı girişleri ile test edelim. İlk olarak 8 sayısını daha sonra sırasıyla 15, 12 ve 5 sayılarını girdiğimiz yandaki çıktıları alacağız.

```
Bir sayı giriniz: 8
    8 sayısı sadece 2'e bölünmektedir.
Bir sayı giriniz: 15
    15 sayısı sadece 3'e bölünmektedir.
Bir sayı giriniz: 12
    12 sayısı hem 2'ye hem de 3'e bölünmektedir.
Bir sayı giriniz: 5
    5 sayısı 2'ye ve 3'e bölünmeMEktedir.
```

Tek satır if-else-elif yapısı kullanımı

Bazı "if" koşul durumlarında işlemler çok kısa olup sadece bazı değişkenlerin belirli değerler alması gerekebilir. Bu durumda "if-else-elif" yapısını alt alta satırlar şeklinde yazmak yerine tek satıra yazılabilir. Ancak bu yazım düzeni 2'den fazla koşulların olduğu durumlarda kodun okunabilirliğini ve anlaşılmasını zorlaştırabilmektedir.

Örneğin a ve b gibi iki sayının büyüklükleri kıyas eden tek koşullu bir durum için;

```
if a > b:
    print("{}, {}'den büyüktür.".format(a,b))
```

yerine aşağıdaki ifadeyi kullanmak kod kısaltma açısından daha uygun olur.

```
if a > b: print("{}, {}'den büyüktür.".format(a,b))
```

2 koşullu bir örnek olarak, bir sayının çift ya da tek sayı olduğunu belirtecek durumu tek satır "if" yapısı ile ifade etmek daha kolay olabilir. Burada "if" koşulunun doğru sonucu "if" kelimesinden önce yanlış sonucu ise "else" den sonra yazılır.

```
a = 10
durum = "Çift sayı" if a%2 == 0 else "Tek sayı"
print("{} sayısı {}'dır.".format(a, durum))
```

normal yazım düzeni

```
if a%2 == 0: durum = "Çift sayı"
else: durum = "Tek sayı"
```

Tek satır if-else-elif yapısı kullanımı

"if" koşul sayısının 2'den fazla olduğu durumlarda kod yazımı tek satıra sığdırılsa bile kodu anlaşılması oldukça zor olabilmektedir. Örneğin verilen bir yaş kriterine göre kişinin "bebek", "genç" ya da "çocuk" olduğu bir durumu dikkate alalım.

normal yazım düzeni

```
if yas >= 18: durum = "Genç"
elif yas < 5: durum = "Bebek"
else: durum = "Çocuk"
```

```
yas = 10
durum = "Genç" if yas >= 18 else "Bebek" if yas < 5 else "Çocuk"
print("Kişi ", durum)
```

Görüldüğü üzere tek satır "if" yazım düzeninin ilk bakışta anlaşılması, normal "if" yazım düzenine göre biraz zordur. Bu bakımdan dikkatli bir şekilde ayarlanması gereklidir. Aksi takdirde istenilen sonuçlar doğru bir şekilde ifade edilmeyecektir.

Çoklu koşulların tek satır halinde ifade edilmesi

Bazı şartlarda "if-else-elif" yapılarındaki koşul sayısı 3-4'den fazla olabilmektedir. Bu durumda koşullar normal parantezler ile ifade edilerek mantık operatörleri ile birbirlerine istenilen sayıda bağlanabilir. Bu durumda mantık operatörleri **"and"** yerine **"&"** sembolü **"or"** yerine **"|"** sembolü kullanılabilir.

```
durum_a = (koşul_1) & (koşul_2) ya da
durum_b = ((koşul_1) & (koşul_2)) | (koşul_3) ya da
durum_a |= (koşul_3)

if durum_b: işlemler_1
else: işlemler_2
```

Match-case yapısı

Python'daki **"match-case"** yapısı, belirli bir değere veya duruma göre farklı kod bloklarının çalıştırılmasını sağlayan bir akış kontrol mekanizmasıdır. Python 3.10 sürümüyle tanıtılan bu yapı, diğer dillerdeki **"switch-case"** yapısına benzer ve daha karmaşık karşılaştırmalar yapmak için kullanılabilir. **"match-case"**, bir değişkenin veya ifadenin değerini çeşitli olasılıklar ile karşılaştırır ve eşleşen durumda ilgili kod bloğunu çalıştırır.

Genel yazım düzeni

```
match değişken_ismi:
   case 'durum 1':
       ifade 1
    case 'durum 2':
       ifade 2
    ...
    case 'durum n':
       ifade n
    case _ :
       ifade genel
```

Örnek kullanım

```
def haftagunu(gun):

   match gun:
      case 1: return "Pazartesi"
      case 2: return "Salı"
      case 3: return "Çarşamba"
      case 4: return "Perşembe"
      case 5: return "Cuma"
      case 6: return "Cumartesi"
      case 7: return "Pazar"
      case _: return "Yanlış gün numarası"


print (haftagunu(3))  // Çarşamba
print (haftagunu(6))  // Cumartesi
print (haftagunu(0))  // Yanlış gün numarası
```

"case" içinde çoklu elemanlar VE, VEYA yapıları ile birlikte "case a & b" yada "case x | y" şeklinde kullanılabilir. Ayrıca list ve tupel yapıları da "case" ile birlikte kullanılabilir.

Match-case örnekleri

Aşağıdaki kod **"match"** deyiminde durumların nasıl birleştirileceğini gösterir. Diğer kod ise **"list"** yapısının nasıl parametre olarak verileceği göstermektedir.

```
def erisim(kisi):                # çoklu parametre işleme

   match kisi:
      case "yönetici" | "idareci": return "Tam erişim"
      case "misafir": return "Sınırlı erişim"
       case _: return "Erişim yok"


print (erisim("idareci"))     // Tam erişim
print (erisim("misafir"))// Sınırlı erişim
print (erisim("Gelen"))       // Erişim yok
```

```
def merhaba(girdi):            # list kullanımı

   match girdi:
      case [durum, kişi]:
          return f"İyi {durum} {kişi}!"

       case [durum, *kişi]:
         msg = ""
         for isim in kisi:
            msg += f"Good {durum} {isim}!\n"
         return msg

print (merhaba(["Sabahlar", "Ahmet"]))          // İyi Sabahlar Ahmet!
print (merhaba(["Akşamlar", "Ayşe", "Cemil"]))    // İyi Akşamlar Ayşe!
                                                  // İyi Akşamlar Cemil!
```

Döngüler

Belirli işlemlerin tekrarlanmasını sağlayan yapılardır. Python'da **"for"** ve **"while"** döngüsü olarak iki temel döngü türü vardır.

"for" döngüsü, genellikle bir dizideki (örneğin, bir sayılabilir liste veya karakter dizisi) her elemanı tek tek işlemek için kullanılır. Örneğin, bir listenin her elemanı üzerinde işlemek yapılmak istenildiğinde **"for"** döngüsü kullanılabilir. Genel yazım düzeni şöyledir.

```
for değişken in dizi:
    .
    . işlemler
    .
```

"while" döngüsü ise belirli bir koşul sağlandığı sürece kodun tekrarlanmasını sağlar. Koşul doğru olduğu sürece döngü devam eder, koşul yanlış olduğunda döngü sonlanır. Ayrıca "else" ifadesiyle de kullanılabilir. Genel yazım düzeni şöyledir.

```
while(koşul):
    .
    . işlemler
    .
else:
    .
    . işlemler
    .
```

"for" ve **"while"** döngüleri içerisinde belirli koşulların durumuna göre **"break"**, **"continue"** ve **"pass"** komutları program akışının kontrolü kullanılırlar.

For döngüsü kullanımı

Belirli bir koleksiyon (liste, string, tuple, set veya dictionary) gibi yineleyebilen bir yapının elemanları üzerinde sırasıyla dolaşmak için kullanılır. "for" döngüsü, belirli bir kod bloğunu, bu yapıdaki her bir eleman için bir kez çalıştırır.

Örnek 1: Liste üzerinde for döngüsü

Bir "meyveler" listesi içindeki elemanları sırayla yazdıralım.

```
meyveler = ["elma", "armut", "muz"]
for meyve in meyveler:
    print(meyve)
```

çıktısı

```
elma
armut
muz
```

Örnek 2: Metin üzerinde for döngüsü

Bir "kelime" metninin içindeki harfleri sırayla yazdıralım.

```
kelime = "Günaydın"
for harf in kelime:
    print(harf)
```

çıktısı

```
G
ü
n
a
y
d
ı
n
```

For döngüsü kullanımı

Örnek 3: "range()" fonksiyonu ile for döngüsü

Python'da **"range()"** fonksiyonu, bir sayış dizisi üretir ve bu dizinin tek tek sayılar üzerinde belirli işlemler yapmak için "for" döngüsü kullanılabilir.

```
for sayi in range(5):

    print(sayi)
```

çıktısı

```
0
1
2
3
4
```

Bu örnekte, "range(5)" ifadesi, 0'dan başlayıp 5'e kadar (5 hariç) sayılar üretir ve "for" döngüsü her adımda bu sayıları ekrana yazdırır.

Örnek 4: For döngüsü ile sözlük kullanımı

Python'da "for" döngüsü, sözlüklerde anahtarlar (keys) ve değerler (values) üzerinde de kullanılabilir.

```
ogrenci = {"isim":"Ali", "yas":21, "bolum":"Bilgisayar Mühendisliği"}

for anahtar, deger in ogrenci.items():

    print("{}:{}".format(anahtar, deger))
```

çıktısı

```
isim: Ali
yas: 21
bolum: Bilgisayar Mühendisliği
```

Bu örnekte, öğrenci sözlüğündeki her anahtar ve ona karşılık gelen değer sırasıyla alınarak yazdırılır.

For döngüsü ve "continue" ve "break" ifadeleri

Döngüyü belirli bir koşul sağlandığında sonlandırmak için **"break"** komutu, döngüdeki belirli bir adımı atlayıp bir sonraki adıma geçmek için ise **"continue"** komutu kullanılır.

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

for sayi in sayi_listesi:

    if sayi == 3:

        break    # sayı 3 olunca döngüden çık

    print(sayi)
```

çıktısı

```
1
2
```

Bu örnekte, sayi 3'e eşit olduğunda **"break"** komutu döngüyü sonlandırır ve artık geri kalan elemanlar işlenmez.

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

for sayi in sayi_listesi:

    if sayi == 3:

        continue  # 3 bulununca bu adımı atlar

    print(sayi)
```

çıktısı

```
1
2
4
5
```

Bu örnekte, sayi 3'e eşit olduğunda "**continue**" komutu döngünün normal adımını atlar ve döngünün bir sonraki elemanı 4 ile devam eder.

While döngüsü kullanımı

"while" döngüsü, **"for"** döngüsünden farklı olarak bir sayı dizisi veya liste üzerinden değil, bir koşulun sağlanıp sağlanmadığına bakarak çalışır. Koşul yanlış (False) olduğunda döngü sona erer. Bu nedenle, **"while"** döngüsü genellikle koşula dayalı işlemler için kullanılır.

Örnek 1: Basit Bir While Döngüsü

Bir koşul doğru olduğu sürece tekrar tekrar işlem yapılır. Örneğin, bir sayacı 1'den 5'e kadar arttıralım.

```
sayi = 1

while sayi <= 5:

    print(sayi)

    sayi += 1
```

çıktısı

```
1
2
3
4
5
```

Bu örnekte, "while" döngüsü, "sayi <= 5" koşulu doğru olduğu sürece çalışır. Her döngüde "sayi" 1 arttırılır ve ekrana yazdırılır. "sayi" 6 olduğunda koşul yanlış olur ve döngü sona erer.

Örnek 2: Sonsuz Döngü

Eğer **"while"** döngüsündeki koşul her zaman doğru kalırsa, döngü asla bitmez ve bu duruma sonsuz döngü denir. Sonsuz döngüden çıkmak için döngü içinde belirli bir koşulla döngüyü sonlandırmak gerekebilir.

```
while True:
    print("Bu bir sonsuz döngüdür.")
    break  # döngüyü durdurmak için kullanılır
```

While döngüsü kullanımı

Örnek 3: Kullanıcı Girişi ile While Döngüsü

"while" döngüsü, bir kullanıcı belirli bir koşulu sağladığında sonlanabilir. Örneğin, kullanıcı doğru bir şifre girdiğinde döngüyü sonlandıralım.

```
dogru_sifre = "12345"
giris = ""
while giris != dogru_sifre:
    giris = input("Lütfen şifrenizi girin: ")
print("Doğru şifreyi girdiniz!")
```

Bu örnekte, kullanıcı doğru şifreyi girene kadar her defasında şifre girilmesi istenir. Şifre doğru girildiğinde döngü biter ve "Doğru şifreyi girdiniz!" mesajı ekrana yazdırılır.

Örnek 4: While Döngüsü ile Faktöriyel Hesaplama

Bir sayının faktöriyelini hesaplamak için **"while"** döngüsü kullanabiliriz.

```
sayi = 5
faktoriyel = 1
while sayi > 0:
    faktoriyel *= sayi
    sayi -= 1
print("Faktöriyel: {}".format(faktoriyel))
```

çıktısı

```
Faktöriyel: 120
```

Bu örnekte, **"while"** döngüsü, sayıyı her adımda bir azaltarak faktöriyelini hesaplar. **"sayi"** sıfır olduğunda döngü sona erer ve faktöriyel sonucu ekrana yazdırılır.

While döngüsü ve "continue" ve "break" ifadeleri

Döngüyü belirli bir koşul sağlandığında sonlandırmak için **"break"** komutu, döngüdeki belirli bir adımı atlayıp bir sonraki adıma geçmek için ise **"continue"** komutu kullanılır.

```
while True:
    sayi = int(input("Bir sayı girin (çıkmak için 0 girin): "))
    if sayi == 0:
        print("Döngü sonlandırıldı.")
        break
    print("Girdiğiniz sayı: {}".format(sayi))
```

```
Bir sayı girin (çıkmak için 0 girin): 5
Girdiğiniz sayı: 5
Bir sayı girin (çıkmak için 0 girin): 3
Girdiğiniz sayı: 3
Bir sayı girin (çıkmak için 0 girin): 0
Döngü sonlandırıldı.
```

çıktısı

Bu örnekte, kullanıcı sıfır "0" sayısını girdiğinde **"break"** komutu ile döngü sona erer.

Örneğin, çift sayıları atlayalım ve sadece tek sayıları yazdıralım.

```
sayi = 0
while sayi < 10:
    sayi += 1
    if sayi%2 == 0:
        continue  # çift sayılar atlanır
    print(sayi)
```

çıktısı

```
1
3
5
7
9
```

Bu örnekte, **"sayi"** çift olduğunda **"continue"** komutu devreye girer ve o adım atlanır. Sadece tek sayılar ekrana yazdırılır.