PYTHON PROGRAMLAMA DİLİ

MUSTAFA KURAL

PROGRAMLAMA NEDİR?

- Programlama, bilgisayarların belirli görevleri yerine getirebilmesi için yazılım oluşturma sürecidir.
- Bilgisayara, hangi işlemleri nasıl yapması gerektiğini anlatan komutların yazılması anlamına gelir.

PROGRAMLAMA DİLİ NEDİR?

- **Programlama dili**, bilgisayarların anlayabileceği ve yerine getirebileceği komutları yazmak için kullanılan bir dil türüdür.
- Bir programlama dili, belirli bir **sözdizimi** (syntax) ve **kurallar** (semantics) içerir. Bu kurallar, geliştiricilerin yazdığı komutların doğru bir şekilde çalışmasını sağlar.

PROGRAMLAMA DİLLERİNİN KATEGORİLERİ

- **Yüksek Seviye Diller:** İnsanlar tarafından daha kolay anlaşılabilir ve yazılabilir. Bu diller, bilgisayarın donanımından bağımsızdır. Örnekler: Python, Java, C#, Ruby.
- **Düşük Seviye Diller:** Bilgisayarın donanımına daha yakın olan ve doğrudan makine diline (ikili kod) çevrilebilen diller. Örnekler: Assembly, C.
- **Script Dilleri:** Genellikle daha basit ve hızlı yazılım geliştirme için kullanılır. Web geliştirmede yaygın olarak kullanılırlar. Örnekler: JavaScript, PHP, Python.
- **Veritabanı Dilleri:** Veri manipülasyonu ve sorgulama için kullanılan diller.

  Örnek: SQL.

PROGRAMLAMA DİLİ KULLANIM ALANLARI

- **Web Geliştirme:** HTML, CSS, JavaScript, PHP gibi dillerle web siteleri ve uygulamaları oluşturulabilir.
- **Mobil Uygulama Geliştirme:** Java, Kotlin, Swift gibi dillerle Android ve iOS uygulamaları yazılabilir.
- **Oyun Geliştirme:** C++, C#, Python gibi diller oyun motorları ve oyunlar için kullanılır.
- **Veri Bilimi ve Yapay Zeka:** Python, R gibi diller veri analizi, yapay zeka ve makine öğrenimi projelerinde yaygın olarak kullanılır.

PYTHON PROGRAMLAMA DİLİ

- **Python**, yüksek seviyeli, nesne yönelimli, güçlü ve oldukça okunabilir bir programlama dilidir. 1991 yılında **Guido van Rossum** tarafından geliştirilen Python, sade sözdizimi ve geniş kütüphane desteği ile programcıların hızlı ve verimli bir şekilde yazılım geliştirmelerini sağlar.

PYTHON'UN TEMEL ÖZELLİKLERİ

- Okunabilirlik ve Basitlik: Python, diğer birçok programlama diline göre daha sade bir sözdizimine sahip olup, kod yazarken geliştiricinin daha az karmaşıklıkla iş yapmasını sağlar. Python'un "daha az kod, daha fazla iş" felsefesi, programcıların hızlıca geliştirme yapabilmelerini sağlar.
- Dinamik Tür Desteği: Python, statik tür denetimi gerektirmeyen bir dil olup, değişken türleri çalışma zamanında belirlenir. Bu, programcıların daha esnek ve hızlı kod yazmalarına olanak tanır.
- Çapraz Platform Desteği: Python, bir kez yazıldığında farklı işletim sistemlerinde çalışabilir. Bu da onu platform bağımsız hale getirir.

PYTHON'UN TEMEL ÖZELLİKLERİ

- Zengin Kütüphane Desteği: Python, gelişmiş projeler için pek çok hazır kütüphane ve modül sunar. Bu kütüphaneler, veri analizi, web geliştirme, yapay zeka, oyun geliştirme gibi çeşitli alanlarda kullanılabilir.
- Nesne Yönelimli Programlama (OOP): Python, nesne yönelimli programlamayı destekler. Bu, yazılımı daha modüler ve sürdürülebilir hale getirir.
- Geniş Kullanım Alanı: Python, web geliştirme, veri bilimi, yapay zeka, makine öğrenimi, otomasyon, oyun geliştirme, masaüstü uygulamaları gibi çok geniş bir kullanım alanına sahiptir.

PYTHON'UN AVANTAJLARI

- Kolay Öğrenilebilir: Sade ve anlaşılır sözdizimi sayesinde, yeni başlayanlar için oldukça uygun bir dil olarak kabul edilir.
- Büyük Topluluk ve Destek: Python'un büyük bir kullanıcı topluluğu vardır, bu da sorunlar için kolayca çözüm bulmayı sağlar.
- Ücretsiz ve Açık Kaynak: Python, açık kaynaklı bir dildir, yani herkes tarafından ücretsiz olarak kullanılabilir, dağıtılabilir ve geliştirilebilir.

PYTHON'UN DEZAVANTAJLARI

- Yavaş Çalışma: Python, derlenen diller kadar hızlı değildir. Bu, yüksek performans gerektiren bazı uygulamalar için sınırlayıcı olabilir.
- Mobil Uygulama Geliştirme: Python, mobil uygulama geliştirme konusunda diğer diller kadar yaygın bir seçenek değildir.

İLK PYTHON KODUMUZ

```
print("Hello, World!")
```

- Bu kod parçası ekrana tırnak içindeki ifadeyi yazdırır.

PYTHON'DA AÇIKLAMA SATIRLARI

```
#5 ve 3 değerlerinin toplamını yazdırır
print(5 + 3)
```

- Açıklama satırları kodun çalışması hakkında programcıya bilgi verir
- Açıklama satırları yorumlanmaz, yani programın çalışmasına etkisi yoktur.

PYTHON'DA AÇIKLAMA SATIRLARI

```
"""
Bu bir açıklama satırıdır
ve
birden fazla satır içerir
"""
print("Python çok kolay.")
```

- Çok satırlı açıklamalar yapmak da mümkündür.

PYTHON'DA DEĞİŞKENLER

```
x = 5              # x is of type int
y = "Python"       # y is of type str and it equals 'Python'
z = 3.5            # z is of type float
```

- Bu kod parçasında x ,y ve z adında üç değişken tanımlanmıştır.
- X bir tam sayı, Y ise metinsel, Z ise ondalıklı sayı bir ifadedir.

PYTHON'DA DEĞİŞKEN TANIMLAMA KURALLARI

- Değişken adı **harf** (a-z, A-Z) veya **alt çizgi** (_) **ile** başlayabilir, değişken adı **rakamlarla** (0-9) ile başlayamaz.
- Değişken adı **harfler (a-z, A-Z)**, **rakamlar** (0-9) ve **alt çizgi** (_) karakterlerinden oluşabilir, özel karakterler ve boşluk içeremez.
- Python'da önceden belirlenmiş bazı anahtar kelimeler (rezerv kelimeler) vardır. Bu kelimeler değişken adı olarak kullanılamaz.
- Python'da değişken adları büyük/küçük harf duyarlıdır. Yani, age ile Age farklı değişkenlerdir.

PYTHON'DA DEĞİŞKEN TANIMLAMA KURALLARI

- Değişkenlerin anlamlı ve anlaşılır isimlerle tanımlanması önemlidir. Bu, kodun okunabilirliğini artırır.
- Zorunlu olmasa da değişken isimlerinde Türkçe karakterler kullanmayın.

PYTHON'DA DEĞİŞKEN TANIMLAMA KURALLARI

Özet:

Python'da değişken tanımlarken şu kurallara dikkat etmeniz gerekir:

- Değişken adı bir harf veya alt çizgi ile başlamalıdır.
- Değişken adı yalnızca harf, rakam ve alt çizgi içerebilir.
- Pyhton anahtar kelimeleri kullanılmamalıdır.
- Büyük/küçük harf duyarlılığı vardır.
- Değişken isimleri anlamlı ve okunabilir olmalıdır.

PYTHON'DA DEĞİŞKENLERE İSİM VERME

- Değişkenleri küçük harf ile isim verelim ve anlamlı isimler verelim.
- Birden fazla kelime bulunan değişken isimlerinde altçizgi(_) ile ayırabiliriz yada ilk kelimeden sonra ilk harfleri Büyük harf ile başlatabiliriz. tel_no, telNo
- Değişken isimlerinde Türkçe karakterler kullanmayalım.

  (ç, Ç, ş, Ş, ğ, Ğ, ö, Ö, ü, Ü, ı, İ)

PYTHON'DA DEĞİŞKENLERE DEĞER ATAMA

```
ad = "Ali"
a, b, c = 3, 6, 10
x = y = z = 1
```

PYTHON'DA DEĞİŞKENLERİ YAZDIRMA

```
x = 5
y = "Python"
print("x")     # ekrana x yazar
print("y")     # ekrana y yazar
print(x) # ekrana 5 yazar
print(y) # ekrana Python yazar
```

PYTHON'DA DEĞİŞKENLERİ YAZDIRMA

```
x = "Python"
y = "is"
z = "awesome"
print(x + y + z)
```

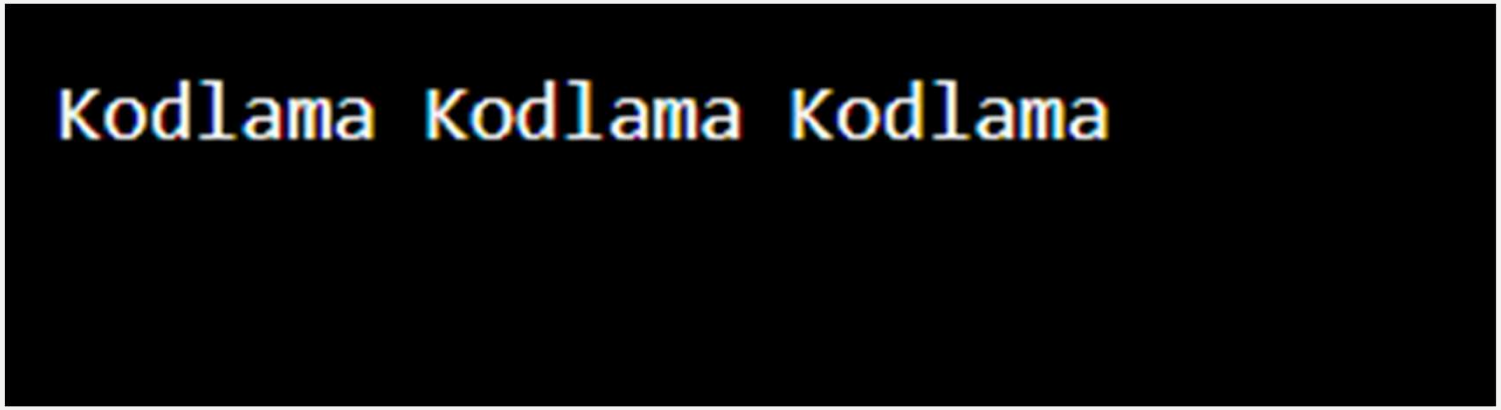
```
Pythonisawesome
```

```
x = "Python"
y = "is"
z = "awesome"
print(x, y, z)
```

```
Python is awesome
```

PYTHON'DA METİNLERİ İSTENEN SAYIDA YAZDIRMA

```
print("Kodlama " * 3)
```



PYTHON'DA DEĞİŞKENLERİN TİPLERİNİ YAZDIRMA

```
x = 5
y = "John"
z = 3.5
print(type(x))
print(type(y))
print(type(z))
```

Çıktı

```
<class 'int'>
<class 'str'>
<class 'float'>
```

PYTHON'DA VERİ TİPLERİ

Bir değişkene atanan veri değişkenin veri tipini belirler.

Example	Data Type
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = range(6)	range
x = {"name" : "John", "age" : 36}	dict
x = {"apple", "banana", "cherry"}	set
x = frozenset({"apple", "banana", "cherry"})	frozenset
x = True	bool
x = b"Hello"	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview
x = None	NoneType

PYTHON'DA VERİ TİPLERİNİN DÖNÜŞÜMÜ

```
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3

a = str("s1") # x will be 's1'
b = str(2)    # y will be '2'
c = str(3.0)  # z will be '3.0'
```

Bir veri tipini hedef veri tipine dönüştürürken hedef tipin adı ve parantez içine veri yazılır.

PYTHON'DA KLAVYEDEN VERİ ALMA

```
ad = input("Lütfen Adınızı Girin:")
print("Merhaba: " + ad)
```

Klavyeden bir string(metin) okumak için input fonksiyonu kullanılır.

input fonksiyonu geriye string türünde veri döner.

PYTHON'DA KLAVYEDEN VERİ ALMA

```
sayi = int(input("Bir sayı girin:"))
print("Sayının karesi: " + (sayi**2))
```

Klavyeden okunan değer bir string(metin) olduğu için sayısal değişkenlere aktarmak için tip dönüşümü yapılmalıdır.

Çünkü aritmetiksel işlemlerde yalnızca sayılar kullanılabilir.

ÖRNEK SORULAR;

1. Klavyeden girilen iki adet sayının toplamını ekrana yazdıran python kodlarını yazınız.
2. Klavyeden iki kenar uzunluğu girilen dörtgenin alanını ve çevresini ekrana yazdıran python kodlarını yazınız.

PYTHON'DA KARAR YAPILARI

PYTHON'DA KARAR YAPILARI

```
if True:
  print("Bu doğru bir ifade.")
```

Karar yapılarında if anahtar kelimesi kullanılır.
Eğer ifade True ise bloktaki kodlar çalışır.

*Blok başlatmak için : (iki nokta üst üste)kullanılır ve sonrasındaki satır içeriden (girintili) başlar.

PYTHON'DA KARAR YAPILARI

```
if False:
  print("Bu blok asla çalışmaz.")
```

Karar yapılarında if anahtar kelimesi kullanılır.
Eğer ifade False ise bloktaki kodlar çalışmaz.

PYTHON'DA KARAR YAPILARI

```
if a > b:
  print("a değeri b'den büyüktür.")
```

Daha önce anlatıldığı gibi karşılaştırma ifadeleri True yada False sonucunu üretir.

Bu sonuca göre if bloğu çalışır yada çalışmaz.

PYTHON'DA KARAR YAPILARI

```
if a > b:
  print("a değeri b'den büyüktür.")
else:
  print("a değeri b'den büyük değildir.")
```

Karşılaştırmanın True olmadığı durumları çalıştırmak için else anahtar kelimesi kullanılır.

PYTHON'DA KARAR YAPILARI

```
if a > b:
  print("a değeri b'den büyüktür.")
elif a < b:
  print("a değeri b'den küçüktür.")
else:
  print("a değeri b'ye eşittir.")
```

Çoklu ve birbirine bağlı koşullar için elif anahtar kelimesi kullanılır.

PYTHON'DA KARAR YAPILARI

```
if a > b and a > c:
  print("a değeri b'den ve c'den büyüktür.")
```

```
if a > b or a > c:
  print("a değeri b'den veya c'den büyüktür.")
```

```
if not a > b:
  print("a değeri b'den büyük değildir.")
```

Karar yapılarında birden fazla koşula bakmak için mantıksal operatörler kullanılır. (and, or, not)

PYTHON'DA KARAR YAPILARI(ÖRNEK)

Öğrencinin ortalaması 50 ve üzerinde ise ekrana GEÇTİ ifadesini yazdıran, aksi durumda ise KALDI ifadesini yazdıran python uygulamasını yazınız.

```
ort = 60
if ort >= 50:
    print("GEÇTİ")
else:
    print("KALDI")
```

PYTHON'DA KARAR YAPILARI(ÖRNEK)

Üç kenar uzunluğu verilen bir üçgenin Eşkenar, İkizkenar veya Çeşitkenar olma durumunu yazdıran python kodlarını yazınız.

```
a, b, c = 6, 3, 6
if a==b and a==c:
    print("Eşkenar Üçgen")
elif a==b or a==c or b==c:
    print("İkizkenar Üçgen")
else:
    print("Çeşitkenar Üçgen")
```

ÖRNEK SORULAR;

1. Klavyeden girilen iki adet sayıdan büyük olanını ve küçük olanını bulup yazdıran python kodlarını yazınız.
2. Klavyeden girilen bir sayının Pozitif, Negatif veya Sıfır olma durumunu yazdıran python kodlarını yazınız.
3. Klavyeden girilen yaşa göre; 0-12 yaş ise "Çocuk", 13-18 yaş ise "Genç", 19-64 yaş ise "Yetişkin", 65 ve üzeri yaşlarda ise "Yaşlı" yazdırın.
4. Klavyeden ortalaması girilen ve devamsızlık yaptığı gün sayısı girilen öğrencinin Geçme/Kalma durumunu sebebi ile yazan python kodlarını yazınız.
5. Girilen boy ve kiloya göre vücut kitle indeksini hesaplayıp; kişinin zayıf, normal, kilolu, aşırı kilolu olma durumunu yazdıran python kodunu yazınız.

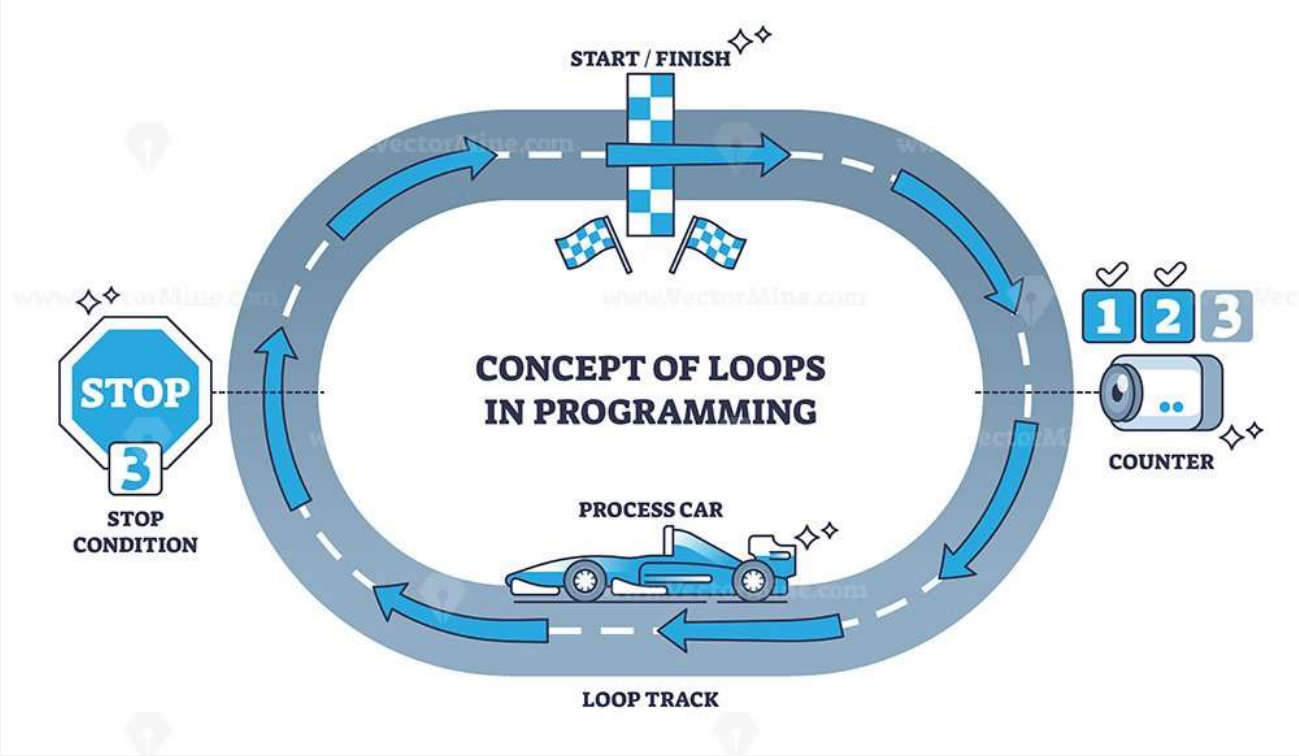
   vki = kilo/boy^2,

   < 18.5 Zayıf, 18.5 → 25 Normal, 25 → 30 Kilolu, > 30 Aşırı Kilolu

PYTHON'DA DÖNGÜLER

PYTHON'DA DÖNGÜLER

Python'da döngüler, belirli bir koşul sağlandığı sürece bir işlemi tekrar tekrar yapmamızı sağlar.

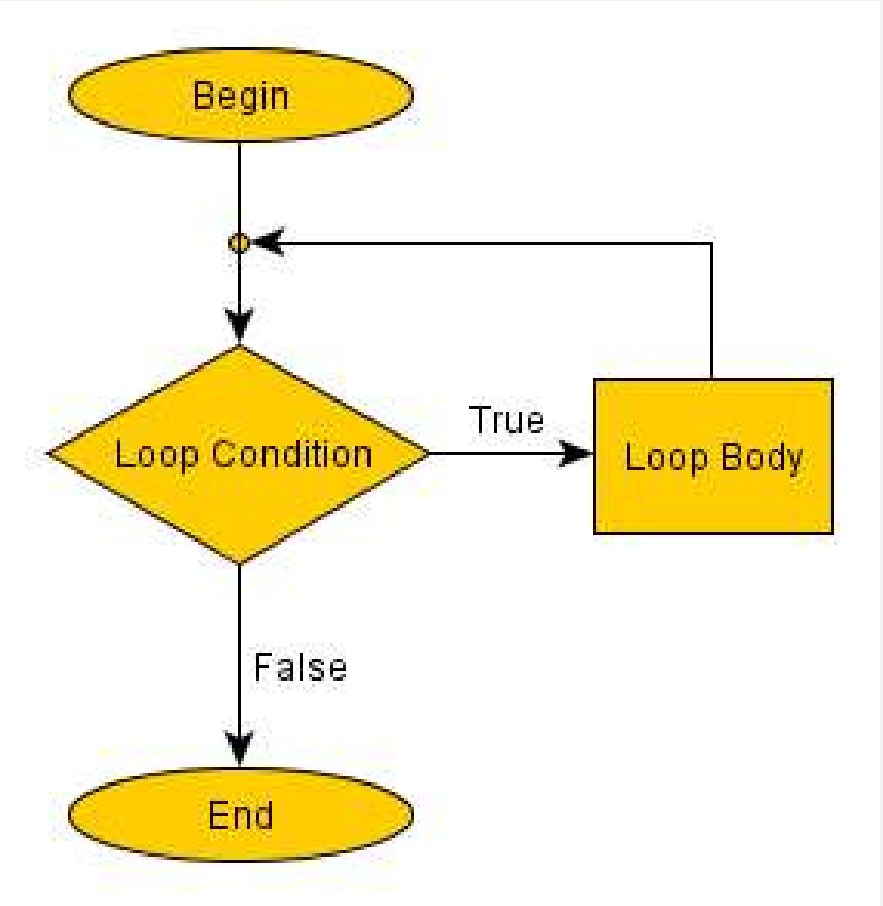


PYTHON'DA DÖNGÜLER

1-Belirli bir koşul sağlandığı sürece bir işlemi tekrar tekrar yapmamızı sağlar.

2-Döngünün bir koşulu ve birde gövdesi vardır.

3-Döngünün her çevriminde bazı değerler değişir ve değişen değerlere göre koşul yeniden kontrol edilir.



PYTHON'DA DÖNGÜLER

İki temel döngü türü vardır:

for döngüsü ve while döngüsü.

Her iki döngü de farklı senaryolarda kullanılır.

```
 3  print(1)
 4  print(2)
 5  print(3)
 6  print(4)
 7  print(5)
 8  print(6)
 9  print(7)
10  print(8)
11  print(9)
12  print(10)
13  print("Ready or not, here I come!")
```

FOR DÖNGÜSÜ

for döngüsü, belirli bir sayıdaki eleman üzerinde işlem yapmak için kullanılır. Çoğunlukla listeler, demetler, diziler veya aralıklar gibi yinelenebilir (iterable) veri yapılarıyla birlikte kullanılır.

[1, 2, 3, 4, 5, 6, ...] bu bir sayı listesidir.

["Ankara", "İstanbul", "İzmir", ...] bu bir metin listesidir.

FOR DÖNGÜSÜ

for döngüsü kullanarak listedeki her bir elemanı ilerleyerek dolaşabiliriz.

```
for sayi in [1, 2, 3, 4, 5, 6]:
    print(sayi)
```

Döngüler bir blok başlatırlar. Blok için python'da : (ikinokta) kullanılır ve sonraki satır ise içeriden başlatılır.

Yukarıdaki döngünün ilk turunda sayi = 1 olacaktır, döngünün son turunda ise sayi=6 olacaktır.

FOR DÖNGÜSÜ

for döngüsü kullanarak listedeki her bir elemanı ilerleyerek dolaşabiliriz.

```
for sehir in ["Ankara", "İstanbul", "İzmir"]:
    print(sehir)
```

Döngüler bir blok başlatırlar. Blok için python'da : (ikinokta) kullanılır ve sonraki satır ise içeriden başlatılır.

Yukarıdaki döngünün ilk turunda sehir = "Ankara" olacaktır, döngünün son turunda ise sehir = "İzmir" olacaktır.

RANGE DEYİMİ

range deyimi bir sayıdan başlayarak, belirli bir artış ile belirli bir sona ilerleyen bir sayı listesi oluşturmak için kullanılır. Bu liste daha sonra for döngüsü ile ilerletilebilir.

```
range( başlangıç, bitiş, artış )
```

- Başlangıç: Liste bu sayıdan başlar. Varsayılan olarak 0 (sıfır)'dır.
- Bitiş: Liste bu sayıya kadardır ve sayı dahil değildir.
- Artış: Listedeki her bir eleman bu sayı kadar artarak ilerler. Varsayılan olarak 1 (bir)'dir.

RANGE DEYİMİ

range deyimi farklı parametrelerle kullanılabilir.

```
range( bitiş )
```

```
range( başlangıç, bitiş )
```

```
range( başlangıç, bitiş, artış )
```

RANGE DEYİMİ ÖRNEKLER

```
range( 5 )          # [ 0, 1, 2, 3, 4]
range( 3 )          # [0, 1, 2]
range( 2, 5 )       # [ 2, 3, 4]
range( 1, 6, 2 )    # [ 1, 3, 5]
range( 6, 1 )       # [ ]
```

Büyük sayıdan küçük sayıya doğru ilerletmek için artış değeri negatif olmalı!

```
range( 6, 1, -1 )   # [ 6, 5, 4, 3, 2 ]
```

Dikkat: Sonuncu dahil değil...

FOR VE RANGE KULLANIMI

```
for sayi in range(10):
    print(sayi)
```

```
for sayi in range(5):
    print("Merhaba")
```

Bazı durumlarda döngü değişkeni sadece döngüyü saydırmak için kullanılır.

```
for sayi in range(1, 10):
    print(sayi)
```

```
for sayi in range(1, 10, 3):
    print(sayi)
```

FOR VE RANGE KULLANIMI

```
bas, son = 5, 15
for sayi in range(bas, son):
    print(sayi)
```

```
bas, son, art = 15, 5, -2
for sayi in range(bas, son, art):
    print(sayi)
```

ÖRNEK SORULAR;

1. 1-100 arasındaki sayıları ekrana yazdıran python kodlarını yazınız.
2. 0-100 arasındaki çift sayıları ekrana yazdıran python kodlarını yazınız.
3. 100'den başlayarak 1 e kadar olan sayıları 3'er azaltarak ekrana yazdıran python kodlarını yazınız.
4. Klavyeden girilen 10 adet sayının toplamını ekrana yazdıran python kodlarını yazınız.
5. Klavyeden girilen adınızı ve yaşınızı isteyen ve adınızı yaşınız kadar for döngüsü ile ekrana yazdıran python kodlarını yazınız.
6. Klavyeden girilen 10 adet sayıdan kaç adet pozitif, kaç adet negatif ve kaç adet sıfır olduğunu bulan ve ekrana yazdıran python kodlarını yazınız.
7. Klavyeden girilen 10 adet sayıdan en küçüğünü ve en büyüğünü bulan ve ekrana yazdıran python kodlarını yazınız.

ÖRNEK SORULAR;

8. Bir sayı dizisinin maksimum ve minimum değerlerini bulan python kodunu for döngüsü kullanarak yazın.
9. Bir dizideki tek sayıların adedini bulan python kodunu for döngüsü kullanarak yazın.
10. Fibonacci dizisini 10. terimine kadar yazdıran python kodunu for döngüsü kullanarak yazın.
11. Bir sayının tam kare olup olmadığını kontrol eden python kodunu for döngüsü kullanarak yazın.
12. 1-100 arasındaki asal sayıları yazdıran python kodunu for döngüsü kullanarak yazın.

WHILE DÖNGÜSÜ

while döngüsü, belirli bir koşul doğru olduğu sürece sürekli olarak bir kod bloğunu çalıştırmak için kullanılır.

- Yani, koşul sağlandığı sürece döngü devam eder.
- Eğer koşul yanlış olursa, döngü sonlanır.

Temel yapısı şu şekildedir:

```
while koşul:
    # Koşul doğru olduğu sürece çalışacak kod
```

WHILE DÖNGÜSÜ

Nasıl çalıştığını daha iyi anlamak için bir örnek üzerinden açıklayalım:

Örneğin, 1'den 5'e kadar olan sayıları yazdırmak için bir while döngüsü kullanabiliriz:

```
sayi = 1
while sayi <= 5:
    print(sayi)
    sayi += 1  # sayıyı 1 artır
```

WHILE DÖNGÜSÜ

Eğer döngü koşulu True olursa döngü sonsuza kadar çalışır.

```
while True:
    # sonsuza kadar çalışır
```

Eğer döngü koşulu False olursa döngü hiç çalışmaz.

```
while False:
    # hiçbir zaman çalışmaz
```

ÖRNEK SORULAR;

1. 1-100 arasındaki tek sayıları ekrana yazan python uygulamasını while döngüsünü kullanarak yazınız.
2. Klavyeden pin kodunu soran ve yapılan giriş "1234" olana kadar sormaya devam eden, pin kodu doğru girilirse ekrana "Tebrikler Giriş Başarılı" mesajını yazan python uygulamasını while döngüsünü kullanarak yazınız.
3. Klavyeden 0 (sıfır) girilene kadar girilen sayıların toplamını ekrana yazan python uygulamasını while döngüsünü kullanarak yazınız.
4. Klavyeden not ortalamasını soran ve girilen ortalama 0-100 arasında değilse tekrar soran, sonrasında ortalama 50 ve üzerinde ise ekrana "BAŞARILI", değilse "BAŞARISIZ" yazan python uygulamasını while döngüsünü kullanarak yazınız.
5. Klavyeden girilen bir sayının faktoryelini hesaplayan python uygulamasını while döngüsünü kullanarak yazınız.

DÖNGÜ KONTROL DEYİMLERİ

Bir döngünün gövdesinde döngü işleyişini değiştirmek için kullanılır.

break deyimi döngüyü yarıda kesmek için kullanılır.

```
for i in range(10):
    if i == 5:
        break  # i 5 olduğunda döngü sona erer
    print(i)
```

DÖNGÜ KONTROL DEYİMLERİ

Bir döngünün gövdesinde döngü işleyişini değiştirmek için kullanılır.

break deyimi döngüyü yarıda kesmek için kullanılır.

```
while True:
    sayi = int(input("Bir sayı girin:"))
    if sayi == 0:
        break  # sayi 0 girildiğinde döngü sona erer
```

DÖNGÜ KONTROL DEYİMLERİ

Bir döngünün gövdesinde döngü işleyişini değiştirmek için kullanılır.

continue deyimi döngüyü ilerisini çalıştırmadan sonraki turuna ilerletmek için kullanılır.

```
for i in range(10):
    if i == 5:
        continue  # i 5 olduğunda döngü sonraki tura geçer
    print(i)
```

DÖNGÜ KONTROL DEYİMLERİ

Bir döngünün gövdesinde döngü işleyişini değiştirmek için kullanılır.

continue deyimi döngüyü ilerisini çalıştırmadan sonraki turuna ilerletmek için kullanılır.

```
while True:
    sayi = int(input("Bir sayı girin:"))
    if sayi < 0:
        continue  # sayi negatif girildiğinde döngü sonraki tura geçer
```

DÖNGÜ KONTROL DEYİMLERİ

Bir döngünün gövdesinde döngü işleyişini değiştirmek için kullanılır.

for else deyimi döngü kırılmadan çalıştı ise bittiğinde else bloğunu çalıştırır.

```
for i in range(10):
    print(i)
else:
    print("bitti") # döngü bloğu çalıştı ve döngü sona erdi
```

DÖNGÜ KONTROL DEYİMLERİ

Bir döngünün gövdesinde döngü işleyişini değiştirmek için kullanılır.

while else deyimi döngü kırılmadan çalıştı ise bittiğinde else bloğunu çalıştırır.

```
i = 0
while i < 5:
    print(i)
    i+=1
else:
    print("bitti") # döngü bloğu çalıştı ve döngü sona erdi
```

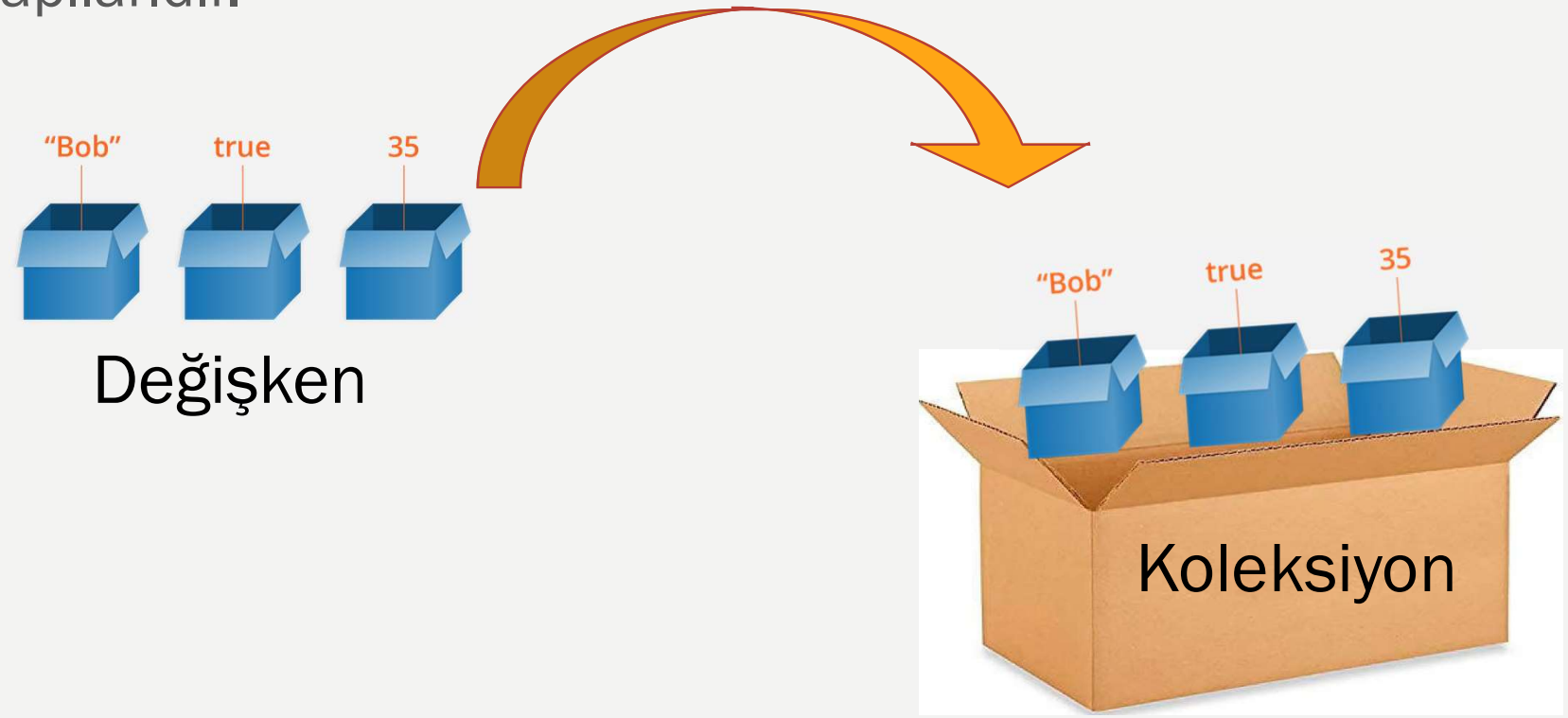
ÖRNEK SORULAR;

1. Klavyeden sıfır girilinceye kadar girilen sayıların toplamını hesaplayan ve sıfır girildiğinde döngüden çıkan python kodunu break deyimini kullanarak yazınız.
2. Bir listedeki yalnızca çift sayıları ekrana yazdıran python kodunu continue deyimini kullanarak yazınız.
3. Kullanıcıdan pin kodunu isteyen ve 3 defa yanlış girildiğinde "3 defa yanlış girdiniz, sistem kilitlendi" yazan, doğru girildiğinde ise "tebrikler giriş başarılı" yazan python kodunu yazınız.
4. 1-100 aralığındaki asal sayıları bulan ve ekrana yazan python kodunu yazınız.
5. Makinenin tuttuğu rasgele sayıyı en fazla 5 tahminde kullanıcının tahmin etmesini, eğer yanlış tahmin ise makinenin aşağı veya yukarı diyerek yardımcı olmasını sağlayacak oyun için python kodlarını yazınız

PYTHON'DA KOLEKSİYONLAR

KOLEKSİYON NEDİR?

Python'da koleksiyonlar, birden fazla öğeyi saklayabilen veri yapılarıdır.



KOLEKSİYON TÜRLERİ

Python'da dört temel koleksiyon türü bulunmaktadır.

1. Listeler (list)
2. Demetler (tuple)
3. Kümeler (set)
4. Sözlükler (dict)

LİSTELER

Listeler sıralı, değiştirilebilir (mutable) ve indekslenebilir veri yapılarıdır. İçerisinde farklı türdeki elemanları tutabilirler.

Özellikler:

- Elemanlar sıralıdır, yani her bir elemanın bir indeksi vardır.
- Elemanlar değiştirilebilir, yani listeyi güncelleyebilirsiniz.
- Aynı eleman birden fazla kez listede bulunabilir.

LİSTELER

Listeler tanımlanırken köşeli parantezler '[ ]' kullanılır. Her eleman virgül ile ayrılır.

```
my_list = [1, 2, 3, 'hello', True]
print (my_list)    # [1, 5, 3, 'hello', True]
```

LİSTE ELEMANLARINA ERİŞMEK

Listenin bir elemanına indeks numarası ile erişelebilir.

```
my_list = [1, 2, 3, 'hello', True]
print (my_list [0])      # ilk eleman : 1
print (my_list [3])      # 4.sıradaki eleman : 'hello'
print (my_list [-1])     # son eleman : True
```

- Soldan başlandığında ilk elemanın indeksi 0' dır ve artarak gider.
- Sağdan başlandığında ilk elemanın indeksi -1'dir ve azalarak gider.

LİSTE ELEMANLARINI DEĞİŞTİRMEK

Listenin bir elemanına indeks numarası ile erişip değiştirilebilir.

```
my_list = [1, 2, 3, 'hello', True]
my_list [0] = 5
my_list [3] = 'Bob'
my_list [-1] = 3.14
print (my_list [0])      # ilk eleman : 5
print (my_list [3])      # 4.sıradaki eleman : 'Bob'
print (my_list [-1])     # son eleman : 3.14
```

LİSTELERİ DİLİMLEME (SLICING)

Listelerde dilimleme yaparak belirli bir aralıktaki elemanları seçebilirsiniz. Yani listenin bir parçasını alabilirsiniz.

```
my_list = [1, 2, 3, 'hello', True]
print (my_list [1 : 3])          # yeni liste : [2, 3]
#İndeks 1 ile 3 arasındaki elemanları al (indeks 3 dahil değildir)
print (my_list [0 : 4 : 2])      # yeni liste : [1, 3]
#İndeks 0 ile 4 arasındaki elemanları 2 atlayarak al (indeks 4 dahil değildir)
```

LİSTELERİ DİLİMLEME (SLICING)

Farklı kombinasyonlarda değerler vererek listenin bir parçasını alabilirsiniz.

[baslangıç : son : atlama] *#başlangıçtan son'a atlayarak al*

[: son : atlama] *#en baştan atlayarak al*

[: : atlama] *#en baştan en sona kadar atlayarak al*

[: :] *#en baştan en sona tümünü al*

Sağdan değerler verilmez ise varsayılan değerler kullanılır.

LİSTELERİ DİLİMLEME (SLICING)

Farklı kombinasyonlarda değerler vererek listenin bir parçasını alabilirsiniz.

[baslangıç : son] *#başlangıçtan son'a tümünü al*

[başlangıç : : atlama] *#başlangıçtan en son'a tümünü atlayarak al*

LİSTELERİ DİLİMLEME (SLICING)

Örnekler :

```
my_list = [3, 6, -8, 2, 5, 4, 7]

print (my_list [3])          # 2
print (my_list [ : 4])       # [3, 6, -8, 2]
print (my_list [2 : 5])      # [-8, 2, 5]
print (my_list [ : : 3])     # [3, 2, 7]
print (my_list [1 : : 2])    # [6, 2, 4]
print (my_list [ : : ])      # [3, 6, -8, 2, 5, 4, 7]
```

LİSTE METOTLARI

Listeler üzerinde işlem yapmak için tasarlanmış birçok metot vardır.

Bu metotlar liste örneğinizin sonunda nokta karakterinden sonra erişilerek kullanılır.

Örneğin:

my_list.metot_adi(*varsa parametreler*)

LİSTEYE ELEMAN EKLEME

Listeye eleman eklemek için append(), insert() veya extend() metodlarını kullanabilirsiniz.

append(): Listeye bir eleman ekler.

insert(): Belirtilen indeks konumuna bir eleman ekler.

extend(): Bir listeyi başka bir listeyle birleştirir.

LİSTEYE ELEMAN EKLEME

Örnekler:

append

```
my_list = [1, 2, 3]
my_list.append(4)      # Listeye 4'ü ekler
print(my_list)         # [1, 2, 3, 4]
```

LİSTEYE ELEMAN EKLEME

Örnekler:

insert

```
my_list = [1, 2, 3]
my_list.insert(1, 10)  # 1.indekse 10 ekler
print(my_list)         # [1, 10, 2, 3]
```

LİSTEYE ELEMAN EKLEME

Örnekler:

extend

```
my_list = [1, 2, 3]
my_list.extend(4, 5, 6)        # listeye [4, 5, 6] listesini ekler
print(my_list)                 # [1, 2, 3, 4, 5, 6]
```

LİSTELERİ BİRLEŞTİRME

Örnekler:

+ operatörü

```
my_list1 = [1, 2, 3]
my_list2 = ['a', 'b', 'c']
my_list = my_list1 + my_list2      # iki listeyi birleştir
print(my_list)                     # [1, 2, 3, 'a', 'b', 'c']
```

LİSTEDEN ELEMAN ÇIKARMA

Listeye eleman silmek için remove(), pop() veya clear() metodlarını kullanabilirsiniz.

remove(): Belirtilen değeri listeden çıkarır. Eğer öğe birden fazla kez varsa, sadece ilk karşılaşılan öğe çıkarılır.

pop(): Belirtilen indeks numarasındaki elemanı çıkarır. Eğer indeks belirtilmezse, son eleman çıkarılır.

clear(): Listenin tüm elemanlarını temizler.

LİSTEDEN ELEMAN ÇIKARMA

Örnekler:

remove

```
my_list = [1, 2, 3, 2, 4]
my_list.remove(2)          # listeden ilk 2 yi çıkarır
print(my_list)             # [1, 3, 2, 4]
```

LİSTEDEN ELEMAN ÇIKARMA

Örnekler:

pop

```
my_list = [1, 2, 3, 2, 4]
my_list.pop(2)                # indeks 2 deki elemanı çıkarır
print(my_list)                # [1, 2, 2, 4]
```

LİSTEDEN ELEMAN ÇIKARMA

Örnekler:

clear

```
my_list = [1, 2, 3, 2, 4]
my_list.clear()                 # listeyi boşaltır
print(my_list)                  # []
```

LİSTEYİ SIRALAM VE TERS ÇEVİRME

Listedeki elemanları sıralamak için sort(),tersine çevirmek için reverse() metodlarını kullanabilirsiniz.

sort(): Listeyi sıralar. Varsayılan olarak küçükten büyüğe sıralar.

reverse(): Listeyi tersine çevirir.

LİSTEYİ SIRALAMA

Örnekler:

sort

```
my_list = [4, 1, 2, 3]
my_list.sort()
print(my_list)          # [1, 2, 3, 4]
# ters sıralama my_list.sort(reverse=True) → [4, 3, 2, 1]
```

LİSTEYİ TERSİNE ÇEVİRME

Örnekler:

reverse

```
my_list = [1, 2, 3, 4]
my_list.reverse()
print(my_list)          # [4, 3, 2, 1]
```

LİSTEYİ KOPYALAMA

Listeyi kopyalamak için copy() metodunu kullanabilirsiniz.

copy(): Listeyi kopyalar. Bu, orijinal liste ile yeni bir liste oluşturur.

```
my_list = [1, 2, 3, 4]
copied_list = my_list.copy()
print(copied_list)        # [1, 2, 3, 4]
```

LİSTEDE ELEMAN ARAMA

Listeyi belirtilen elemanın indeksini bulmak için index(), belirtilen elemanın sayısını bulmak için count() metodunu kullanabilirsiniz.

Ayrıca in anahtar kelimesi ile de bir eleman listede var mı yok mu kontrol edebilirsiniz.

index(): Belirtilen elemanın indeksini döner. Eleman yok ise hata döner.

count(): Belirtilen değerin listede kaç kere geçtiğini döndürür.

LİSTEDE ELEMAN ARAMA

Örnekler:

index

```
my_list = [1, 5, 8, 2]
print(my_list.index(8))        # 2
```

LİSTEDE ELEMAN ARAMA

Örnekler:

count

```
my_list = [1, 5, 8, 5, 2, 5]
print(my_list.count(5))       # 3
```

LİSTEDE ELEMAN ARAMA

Örnekler:

in

```
my_list = [1, 5, 8, 5, 2, 5]
print(3 in my_list)      # False
print(2 in my_list)      # True
```

LİSTELERDE KULLANILABİLECEK HARİCİ METOTLAR

len(list) : Liste uzunluğunu döndürür.

max(list) : Listede en büyük elemanı döndürür.

min(list) : Listede en küçük elemanı döndürür.

sum(list) : Liste elemanlarının toplamını döndürür.

ÖRNEK SORULAR;

1- Bir listeye klavyeden okunan 5 adet sayıyı saklayın ve sonrasında bu sayıları küçükten büyüğe doğru sıralayarak ekrana yazdıran python kodunu yazın.

2- Sınıftaki öğrencilerin olduğu bir liste tanımlayın ve rasgele bu kişilerden birini seçtirin, daha sonra devam etmek isteyip istemediğinizi soran ve evet cevabını alınca tekrar listeden bir öğrenciyi seçen python kodunu yazın.

3- Bir metin içindeki Türkçe karakterleri bulan ve bunları İngiliz alfabesindeki karşılıkları ile değiştiren ve sonucu ekrana yazan python kodunu yazın.

4- 52'lik kart destesini rasgele karıştırın ve iki oyuncunun sırayla birer kart çekmesini sağlayın. Ayrıca oyuncular PAS diyerek o eli kart çekmeden geçebilir. Bir oyuncu üst üste 2 defa PAS geçer ise yada çekilen kart değerlerine göre 21' değerini geçerse oyuncu için 'KAYBETTİ', diğer oyuncu için ise 'KAZANDI' şeklinde çıktı oluşturan oyun için python kodlarını yazınız.

TUPLE (DEMET) NEDİR?

Tuple, birden fazla öğeyi tek bir değişkende saklamak için kullanılan, sıralı (ordered) ve değiştirilemez (immutable) bir veri yapısıdır.

```
my_tupple = (1, 2, 3)
```

TUPLE NEDEN KULLANILIR?

- Verilerin değişmemesi isteniyorsa tuple tercih edilir.
- Daha güvenlidir çünkü veriler değiştirilmez.
- Listelerden daha performanslıdır, özellikle büyük veri kümelerinde.
- Anahtar (key) olarak kullanılabilir çünkü hashlenebilir (*listeler hashlenemez*).

TUPLE VE LİSTE ARASINDAKİ FARKLAR

Özellik	Liste (`list`)	Tuple (`tuple`)
Değiştirilebilir	✅ Evet	❌ Hayır (immutable)
Parantez	Köşeli `[]`	Normal `()`
Performans	Daha yavaş	Daha hızlı
Kullanım Amacı	Değişken veri	Sabit, güvenli veri
Hashlenebilirlik	❌ Hayır	✅ Evet (içerik uygunsa)

TUPLE NASIL OLUŞTURULUR?

```
# Boş tuple
empty = ()
# Tek elemanlı tuple (virgül gerekli!)
single = (5,)
# Çok elemanlı tuple
numbers = (1, 2, 3)
# Karışık veri türleri
mixed = (1, 'hello', True)
```

TUPLE ÜZERİNDE İŞLEMLER

- Listelerden tek farkı değiştirilemez olmasıdır.

```
t = (10, 20, 30, 40)    # Tanımlama
print(t[0])             # 10
print(t[-1])            # 40
print(t[1:3])           # (20, 30)

# tuple'lar değiştirilemez ama yeniden atanabilir:
t = t + (50,)           # (10, 20, 30, 40, 50)
```

TUPLE METOTLARI

- Tuple'lar çok fazla metoda sahip değildir, çünkü değiştirilemezler. Ancak bazı temel metotları vardır.
- Listelerde değişiklik yapan metotlar burada kullanılamazlar.
- Bunun yanında indeksli erişim, dilimleme ve for döngüsü ile gezinme kullanılabilir

TUPLE METOTLARI

Demet içinde belirtilen elemanın indeksini bulmak için index(), belirtilen elemanın sayısını bulmak için count() metodunu kullanabilirsiniz. Ayrıca in anahtar kelimesi ile de bir eleman demet içinde var mı yok mu kontrol edebilirsiniz.

index(): Belirtilen elemanın indeksini döner. Eleman yok ise hata döner.

count(): Belirtilen değerin listede kaç kere geçtiğini döndürür.

TUPLE METOTLARI

Bunun yanında birde gömülü bazı fonksiyonlar da kullanılabilir.

len(list) : Demet uzunluğunu döndürür.

max(list) : Demet içindeki en büyük elemanı döndürür.

min(list) : Demet içindeki en küçük elemanı döndürür.

sum(list) : Demet içindeki elemanlarının toplamını döndürür.

LİSTEDEN FARKLILIK OLARAK

Özellik/Metot	Liste (`list`)	Tuple (`tuple`)
`append()`	✅ Kullanılır	❌ Yok
`extend()`	✅ Kullanılır	❌ Yok
`insert()`	✅ Kullanılır	❌ Yok
`remove()`	✅ Kullanılır	❌ Yok
`pop()`	✅ Kullanılır	❌ Yok
`reverse()`	✅ Kullanılır	❌ Yok
`sort()`	✅ Kullanılır	❌ Yok

DEMETİ LİSTEYE DÖNÜŞTÜRME

```
demet = (1, 2, 3, 4)
liste = list (demet)      # Liste'ye dönüştür
# Liste üzerinde ekleme, çıkarma, sıralama yapabilirsin
liste.append(5)
liste.reverse()
liste.pop()
# v.b. ...
demet = tuple (liste) # Tekrar demet'e  dönüştür
```

ALIŞTIRMA

```
meyveler = ("elma" , "armut", "muz")
```

Görevler:

1. meyveler adlı tuple'ı listeye çevir.
2. Listenin sonuna "çilek" ekle.
3. Listenin başına "kiraz" ekle.
4. Listenin son halini tekrar tuple'a çevir ve ekrana yazdır.

SET (KÜME) NEDİR?

Python'da küme (set), sırasız (*unordered*), değiştirilebilir (*mutable*), tekrarsız (*unique*) elemanlardan oluşan bir veri yapısıdır.

```
my_set = {"elma", "armut", "muz"}
```

- Aynı matematikteki kümeler gibi çalışır.

NEDEN KULLANILIR?

- Tekrarlı verileri otomatik olarak filtrelemek için.
- İki küme arasındaki fark, kesişim, birleşim gibi işlemleri kolayca yapmak için.
- Verilerde hızlı üyelik sorguları (in) yapmak için.
- Veri temizleme ve karşılaştırma işlemlerinde.

KÜME NASIL TANIMLANIR?

```
# Boş küme (dikkat: { } boş sözlüktür, başka anlama gelir)
bos_kume = set()
# elemanlı küme
renkler = {"kırmızı", "mavi", "yeşil"}
# Bir lisden küme elde etmek
sayilar = set([1, 2, 2, 3, 4])
print(sayilar)      # {1, 2, 3, 4} → dikkat 2 tek
```

KÜME VE LİSTE ARASINDAKİ FARKLAR

Özellik	Liste (`list`)	Küme (`set`)
Sıralı mı?	✅ Evet	❌ Hayır
Elemanlar tekrar eder mi?	✅ Evet	❌ Hayır (benzersiz)
Değiştirilebilir mi?	✅ Evet	✅ Evet
İndeksleme var mı?	✅ Var	❌ Yok
Performans (arama)	Daha yavaş	Daha hızlı (hash tabanlı)

KÜME VE LİSTE ARASINDAKİ FARKLAR

- Kümeler **sırasız** olduğu için indeksleme yapılamaz.

```
my_set = {1, 2, 3}
my_set[0]  # ❌ HATA
```

- Aynı eleman birden fazla eklenirse yalnızca bir tanesi saklanır.

KÜME METOTLARI

Listelerde kullanılan metotlar kümelerde de kullanılabilir. Bunun yanında bazı kümeye özgü metotlar da tanımlanmıştır.

KÜME METOTLARI

- Küme'ye Özgü Metotlar

Metot/Fonksiyon	Açıklama
`add(elem)`	Eleman ekler
`remove(elem)`	Eleman siler, yoksa hata verir
`discard(elem)`	Eleman siler, yoksa hata vermez
`pop()`	Rastgele bir elemanı çıkarır
`clear()`	Tüm elemanları siler
`update(iterable)`	Birden fazla eleman ekler

KÜME OPERASYONLARI (MATEMATİKSEL)

a = {1, 2, 3}

b = {3, 4, 5}

İşlem	Açıklama	Kod Örneği
Birleşim	Tüm elemanlar	`a.union(b)` veya `a|b`
Kesişim	Ortak elemanlar	`a.intersection(b)` veya `a & b`
Fark	Sadece `a` da olanlar	`a.difference(b)` veya `a - b`
Simetrik Fark	Her ikisinde olmayanlar	`a.symmetric_difference(b)` veya `a ^ b`

GÖMÜLÜ FONKSİYONLAR

Fonksiyon	Açıklama
`len(set)`	Eleman sayısını verir
`max(set)`	En büyük eleman
`min(set)`	En küçük eleman
`sorted(set)`	Sıralı liste döner
`sum(set)`	Elemanların toplamı

ALIŞTIRMA

```
# Öğrenci listeleri
ogrenciler = {"Ayşe", "Mehmet", "Zeynep", "Ali", "Burak", "Ahmet",
"Fatma", "Cem", "Deniz"}
python_kursu = {"Ayşe", "Mehmet", "Zeynep", "Ali", "Burak"}
veri_bilimi_kursu = {"Ali", "Zeynep", "Ahmet", "Fatma"}
```

Görevler:

1. Her iki kursa da katılan öğrencileri listeleyin.
2. Sadece python kursuna katılan öğrencileri listeleyin.
3. Herhangi bir kursa katılan öğrencileri listeleyin.
4. Bir kursa katılmayan öğrencileri listeleyin.

ÖRNEKLER;

1. Bir okulda 3 kurs var: İngilizce, Almanca, Fransızca. Bu kurslar demet(tuple) olarak saklanıyor. Okuldaki kursları ekrana yanlarına sıra numarası ile yazdırması, Kullanıcıya okul numarasını ,adını sorması ve bu kurslardan katılmak istediklerini seçmesi istenmektedir. Kullanıcı aralarına virgül ekleyerek istediği kurs numaralarını seçebilir. Her seçtiği kurs için kullanıcının **numarası + adı** ilgili kursun kümesine eklenecektir. Bu işlem toplam 5 öğrenci için gerçekleşecektir.
    1. Sonuç olarak Kurslara katılan öğrenci listelerini yazdırın.
    2. Yalnızca bir kurs için başvuran kaç öğrenci var?
    3. Hem İngilizce ve Almanca kursunu seçen öğrenciler kimler?
    4. Her üç kursa başvuran kaç öğrenci var?

DICTIONARY (SÖZLÜK) NEDİR?

Sözlük (dictionary), anahtar-değer (key-value) çiftleriyle veri saklamaya yarar. Her anahtar (key) benzersizdir ve her anahtar bir değere (value) karşılık gelir.

```
ogrenci = {
    "ad" : "Ali",
    "soyad" : "Yılmaz",
    "yas" : 21,
    "bolum" : "Bilgisayar Mühendisliği"
}
```

Burada "ad", "soyad", "yas" ve "bolum" anahtarlardır. Her birinin bir değeri vardır.

SÖZLÜK NASIL TANIMLANIR?

1- Kırlangıç (süslü) parantezlerle:

```
bilgi = { "renk" : "mavi", "sayi" : 10 }
```

2- dict() fonksiyonuyla:

```
bilgi = dict (renk="mavi", sayi=10)
```

LİSTE VE SÖZLÜK ARASINDAKİ FARKLAR

Özellik	Liste (`list`)	Sözlük (`dict`)
Erişim	Sıra numarasıyla (indeks)	Anahtar ile
Veri yapısı	Sıralı	Sırasız (Python 3.7+ sürümünden itibaren sıralıdır)
Kullanım amacı	Sıralı veri tutmak	Anahtar-değer ilişkisiyle veri tutmak
Örnek erişim	`liste[0]`	`sozluk["ad"]`

SÖZLÜK METOTLARI

Python sözlükleri için kullanılabilecek bazı yaygın metotlar şunlardır:

keys() – Tüm anahtarları döner:

```
ogrenci.keys()  # dict_keys(['ad', 'soyad', 'yas', 'bolum'])
```

values() – Tüm değerleri döner:

```
ogrenci.values()  # dict_values(['Ali', 'Yılmaz', 21, 'Bilgisayar Mühendisliği'])
```

items() – Anahtar-Değer çiftlerini döner:

```
ogrenci.items()
# dict_items([('ad', 'Ali'), ('soyad', 'Yılmaz'), ('yas', 21), ('bolum', 'Bilgisayar Mühendisliği')])
```

SÖZLÜK METOTLARI

Python sözlükleri için kullanılabilecek bazı yaygın metotlar şunlardır:

get(key) – Anahtara göre değeri getirir (hata vermez):

```
ogrenci.get("ad")     # 'Ali'
ogrenci.get("adres")  # None (adres yoksa hata vermez)
```

update() – Sözlüğü günceller:

```
ogrenci.update({"yas": 22, "adres": "Ankara"})
```

SÖZLÜK METOTLARI

Python sözlükleri için kullanılabilecek bazı yaygın metotlar şunlardır:

pop(key) – Anahtara göre elemanı siler ve değerini döner:

```
yas = ogrenci.pop("yas")
```

clear() – Tüm öğeleri siler:

```
ogrenci.clear()
```

SÖZLÜK ELEMANLARI

Python sözlükleri için her bir eleman (key,value) şeklindedir:

```
kisiler = {}
# Sözlüğe eleman ekleme
kisiler["ali"] = "05001234567"
kisiler["ayse"] = "05007654321"
```

Sözlüğe ait elemanları for döngüsü ile dolaşabilirsiniz:

```
# Bilgileri listeleme
for k, v in kisiler.items():
    print(f"{k} => {v}")
```

ALIŞTIRMALAR

1. Aşağıdaki sözlüğe bir "yas" anahtarı ve değeri ekleyiniz:

```
kisi = {"ad": "Ahmet", "soyad": "Demir"}
```

2. Aşağıdaki sözlükten "soyad" anahtarını silen kodu yazınız:

```
kisi = {"ad": "Ahmet", "soyad": "Demir", "yas": 25}
```

3. Aşağıdaki sözlükte, "yas" değerini 30 olarak güncelleyiniz:

```
kisi = {"ad": "Ahmet", "yas": 25}
```

4. ogrenci adlı sözlükteki tüm anahtarları ve değerleri alt alta yazdırınız:

```
ogrenci = {"ad": "Elif", "bolum": "Matematik", "not": 85}
```

ALIŞTIRMALAR

5. Aşağıdaki sözlükte kaç tane anahtar-değer çifti vardır bulunuz?

```
veri = {"x": 1, "y": 2, "z": 3}
```

6. Aşağıdaki sözlükte tüm değerlerin ortalamasını bulan kodu yazınız:

```
notlar = {"Ali": 80, "Ayşe": 90, "Mehmet": 70}
```

7. Kullanıcıdan alınan isim ve telefon numarasını bir sözlüğe ekleyin. Kullanıcı "q" girene kadar devam etsin. Sonrasında tüm isimleri ve telefon numaralarınız yazdırın.

8. Aşağıdaki sözlükte "not" değerini alıp harf notuna çeviren bir if-else bloğu yazınız:


```
ogrenci = {"ad": "Zeynep", "not": 87}
#Örneğin: 90+ → A, 80-89 → B, 70-79 → C, 60-69 → D, 50-59 → E ...
```

MİNİ PROJE – TELEFON REHBERİ

Şimdi aşağıdaki örneği yapalım.

Bu uygulamada:

- Kişi ekleyebilir,
- Numara sorgulayabilir,
- Kişi silebilir,
- Tüm kişileri listeleyebilirsin.

Sözlük veri yapısı, kişi adlarını **anahtar**, telefon numaralarını ise **değer** olarak tutacak.

MİNİ PROJE – TELEFON REHBERİ

1-Öncelikle rehber adında boş bir sözlük tanımlayın.

```
rehber = {}
```

2-Ardından ana program döngüsünü bir menü ile yazın.

```
while True:
  print("\n--- Telefon Rehberi ---")
  print("1. Kişi Ekle")
  print("2. Kişi Ara")
  print("3. Kişi Sil")
  print("4. Tüm Rehberi Göster")
  print("5. Çıkış")
```

MİNİ PROJE – TELEFON REHBERİ

3-Kullanıcıdan seçimini alın.

```
secim = input("Seçiminiz (1-5): ")
```

Kullanıcının seçimine göre iş ve işlemler yapılacak.

MİNİ PROJE – TELEFON REHBERİ

4-Ardından kullanıcının seçimine karşılık gelecek işlemleri yazın

```
if secim == "1":
  #Rehbere yeni kişi eklenecek
elif secim == "2":
  #Rehberden kişi aranacak
elif secim == "3":
  #Rehberden kişi silinecek
elif secim == "4":
  #Rehberdeki kişiler listelenecek
elif secim == "5":
  #Çıkış yapılacak
else:
  print("Lütfen 1 ile 5 arasında bir seçim yapınız.")
```

MİNİ PROJE – TELEFON REHBERİ

5-Yeni kişi eklemek için gerekli kodları yazın.

```
if secim == "1":
  #Rehbere yeni kişi eklenecek
  ad = input("Kişi adı: ").strip().upper()
  if ad in rehber:
      print(f"{ad} zaten rehberde kayıtlı.")
  else:
      numara = input("Telefon numarası: ").strip()
      rehber[ad] = numara
      print(f"{ad} rehbere eklendi.")
```


Dikkat: Kişilerin isimleri büyük harf ve başında ve sonunda boşluksuz olarak eklenecek.

MİNİ PROJE – TELEFON REHBERİ

6-Kişi arama için gerekli kodları yazın.

```
elif secim == "2":
  #Rehberden kişi aranacak
  ad = input("Aranan kişi adı: ").strip().upper()
  if ad in rehber:
      print(f"{ad} => {rehber[ad]}")
  else:
      print(f"{ad} rehberde bulunamadı.")
```


Dikkat: Daha önceden hatırlayın: Kişilerin isimleri büyük harf ve başında ve sonunda boşluksuz olarak eklenmişti.

MİNİ PROJE – TELEFON REHBERİ

7-Kişi silmek için gerekli kodları yazın.

```
elif secim == "3":
  #Rehberden kişi silinecek
  ad = input("Silinecek kişi adı: ").strip().upper()
  if ad in rehber:
      rehber.pop(ad)
      print(f"{ad} rehberden silindi.")
  else:
      print(f"{ad} rehberde yok.")
```


Dikkat: Daha önceden hatırlayın: Kişilerin isimleri büyük harf ve başında ve sonunda boşluksuz olarak eklenmişti.

MİNİ PROJE – TELEFON REHBERİ

8-Kişileri listelemek için gerekli kodları yazın.

```
elif secim == "4":
  #Rehberdeki kişiler listelenecek
  if not rehber: # yada rehber == {}
      print("Rehber boş.")
  else:
      print("\n--- Rehber Listesi ---")
      for ad, numara in rehber.items():
          print(f"{ad}: {numara}")
```

MİNİ PROJE – TELEFON REHBERİ

9-Çıkış için gerekli kodları yazın.

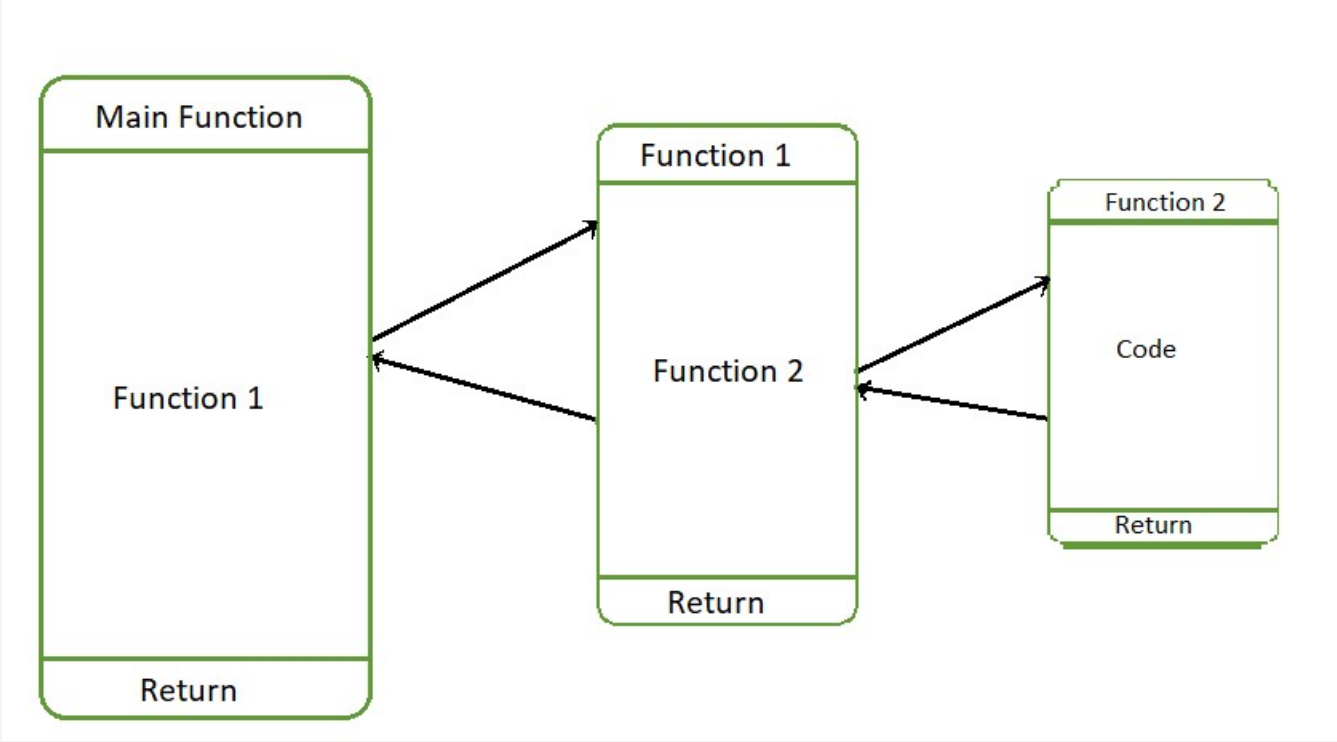
```
elif secim == "5":
    #Çıkış yapılacak
    print("Çıkılıyor... Görüşmek üzere!")
    break
```

Harika! Şimdi uygulamanızı test edebilirsiniz.

PYTHON'DA FONKSİYONLAR

FONKSİYON NEDİR?

Fonksiyon, belirli bir işi yapan kod bloklarına verilen isimdir. Önceden tanımlanmış kodlar ismi ile çağırılarak çalıştırılır.



NEDEN KULLANILIR?

- Kod tekrarını önlemek,
- Programı daha düzenli ve okunabilir yapmak,
- Büyük programları küçük parçalara bölerek yönetmek

NASIL TANIMLANIR?

Python'da fonksiyonlar def anahtar kelimesiyle tanımlanır.

Dikkat edin yeni bir blok içerisine kodlar yazılır

```
def fonksiyon_adi():
    # Yapılacak işlemler
```

PARAMETRESİZ FONKSİYONLAR

Çağırılırken herhangi bir değer almayan fonksiyonlardır.

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

Çağırmak için ise fonksiyonun adı yazılır.

```
# Fonksiyon çağrılır:
selamla()
```

PARAMETRELİ FONKSİYONLAR

Çağırılırken fonksiyona bir değer göndermemizi sağlar.

Parameteye fonsiyon tanımlanırken bir isim verilir.

```
def selamla_isimle(isim):
    print(f"Merhaba, {isim}!")
```

Çağırmak için ise;

```
# Fonksiyon çağrılır:
selamla_isimle("Ali")
```

GERİYE DEĞER DÖNEN FONKSİYONLAR

Çağırılırken fonksiyon işlem sonunda geriye bir değer döndürebilir. Bunun için return anahtar kelimesi kullanılır.

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

Çağırmak ve dönen değeri almak için;

```
sonuc = topla(3, 5)
print(sonuc)
```

GERİYE DEĞER DÖNDÜRMEYEN FONKSİYONLAR

Eğer return anahtar kelimesi kullanılmazsa fonksiyon sadece bir şeyler yapar ama geriye değer dönmez.

```
def yazdir_toplam(a, b):
    print(a + b)
```

Çağırmak için;

```
yazdir_toplam(2, 4)
```

FONKSİYONLARLA İLGİLİ KÜÇÜK ÖZET TABLOM

Konu	Açıklama
Fonksiyon	Belirli işi yapan kod bloğu
Tanımlama	`def` ile yapılır
Parametreli	Dışarıdan bilgi alır
Parametresiz	Dışarıdan bilgi almaz
Geriye dönen	`return` kullanır
Geriye dönmeyen	Sadece işlem yapar

FONKSİYONLU ÖRNEK SORULAR

1. İki sayının çarpımını bulan bir fonksiyon yaz.

İpucu:

- Fonksiyon 2 parametre alacak.
- Çarpım sonucunu ekrana yazdıracak.

2. Bir sayının karesini hesaplayan bir fonksiyon yaz.

İpucu:

- 1 parametre alacak.
- Karesini döndürüp print() ile yazdıracaksın.

FONKSİYONLU ÖRNEK SORULAR

3. İsmi parametre olarak alıp ekrana "Merhaba, [isim]!" yazdıran bir fonksiyon yaz.

İpucu:

- Sadece yazdıracak (return yok).

4. Kullanıcının doğum yılına göre yaşını hesaplayan bir fonksiyon yaz.

İpucu:

- Parametre: doğum yılı
- Şu anki yılı (örneğin 2025) kullanarak yaş hesapla ve ekrana yazdır.

FONKSİYONLU ÖRNEK SORULAR

5. Bir listenin içindeki sayıların toplamını bulan bir fonksiyon yaz.
İpucu:
- Parametre: bir liste
- Listedeki tüm sayıları toplayıp sonucu döndür.

6. Bir sayının pozitif mi negatif mi olduğunu kontrol eden bir fonksiyon yaz.
İpucu:
- Parametre: 1 sayı
- Sayıya göre "Pozitif", "Negatif" veya "Sıfır" yazdıracak.

MİNİ PROJE – ÖĞRENCİ NOT SİSTEMİ

Şimdi aşağıdaki örneği yapalım.

Bu uygulamada:

- Öğrenci ekleyebilir,
- Öğrenciye not ekleyebilir,
- Öğrencinin notunu güncelleyebilir,
- Tüm öğrencileri listeleyebilirsin.

Sözlük veri yapısı, kişi adlarını **anahtar**, notları ise **değer** olarak tutacak ve fonksiyonlar kullanılacak.

MİNİ PROJE – ÖĞRENCİ NOT SİSTEMİ

1 - İlk olarak boş bir sözlük tanımla

```
# Öğrencileri tutacağımız boş sözlük
ogrenciler = {}
```

MİNİ PROJE – ÖĞRENCİ NOT SİSTEMİ

2 – Öğrenci eklemek için fonksiyon tanımla

```
# Öğrenci ekleme fonksiyonu
def ogrenci_ekle(isim, notu):
    ogrenciler[isim] = notu
    print(f"{isim} adlı öğrenci {notu} notu ile eklendi.")
```

MİNİ PROJE – ÖĞRENCİ NOT SİSTEMİ

3 – Öğrenci notu güncellemek için fonksiyon tanımla

```
# Not güncelleme fonksiyonu
def not_guncelle(isim, yeni_not):
    if isim in ogrenciler:
        ogrenciler[isim] = yeni_not
        print(f"{isim} adlı öğrencinin notu {yeni_not} olarak güncellendi.")
    else:
        print(f"{isim} adlı öğrenci bulunamadı!")
```

MİNİ PROJE – ÖĞRENCİ NOT SİSTEMİ

4 – Öğrencileri listelemek için fonksiyon tanımla

```
# Tüm öğrencileri listeleme fonksiyonu
def tum_ogrencileri_listele():
    if len(ogrenciler) == 0:
        print("Henüz öğrenci eklenmedi.")
    else:
        print("Tüm Öğrenciler:")
        for isim, notu in ogrenciler.items():
            print(f"- {isim}: {notu}")
```

MİNİ PROJE – ÖĞRENCİ NOT SİSTEMİ

5 – Şimdide ana program akışını yaz, sonsuz döngü içinde bir menü olsun.

```
# Buradan itibaren program akışı
print("Öğrenci Not Sistemi Başladı!\n")

while True:
    print("\nSeçenekler:")
    print("1 - Öğrenci Ekle")
    print("2 - Not Güncelle")
    print("3 - Tüm Öğrencileri Listele")
    print("4 - Çıkış")
```

MİNİ PROJE – ÖĞRENCİ NOT SİSTEMİ

6 – Kullanıcıdan menü seçimini alın ve kontrol edin.

```
secim = input("Bir seçenek seç (1-4): ")


if secim == "1":
    ●
elif secim == "2":
    ●
    ●
```

MİNİ PROJE – ÖĞRENCİ NOT SİSTEMİ

7 – Şimdi ilk seçeneği yazalım, öğrenci ekle.

```
if secim == "1":
    isim = input("Öğrenci adı: ")
    notu = int(input("Öğrencinin notu: "))
    ogrenci_ekle(isim, notu)
```

MİNİ PROJE – ÖĞRENCİ NOT SİSTEMİ

8 – Şimdi ikinci seçeneği yazalım, öğrenci güncelle.

```
elif secim == "2":
    isim = input("Notunu güncellemek istediğin öğrenci adı: ")
    yeni_not = int(input("Yeni not: "))
    not_guncelle(isim, yeni_not)
```

MİNİ PROJE – ÖĞRENCİ NOT SİSTEMİ

9 – Şimdi sonraki, öğrenci listeleme ne kadar kısa değil mi ?

```
elif secim == "3":
    tum_ogrencileri_listele()
```

MİNİ PROJE – ÖĞRENCİ NOT SİSTEMİ

10 – Son seçenek, tabiki çıkış.

```
elif secim == "4":
    print("Çıkılıyor...")
    break

else:
    print("Geçersiz seçim! Lütfen 1-4 arasında bir rakam girin.")
```

ayrıca kullanıcı yanlış seçim yaptığında uyarmayı unutma!

VARSAYILAN PARAMETRELİ FONKSİYONLAR

Fonksiyon çağrılırken değer verilmez ise geçerli olacak varsayılan bir değer verilebilir.

```
def selamla(isim="Misafir"):
    print(f"Merhaba, {isim}!")

selamla()
selamla("Ahmet")
```

ANAHTAR KELİME VE KONUM ARGÜMANLAR

Fonksiyon çağrılırken parametreler sırasıyla veya karışık bir sırada parametre adı ile verilebilir.

```
def bilgi(isim, yas):
    print(f"{isim} - {yas}")

bilgi("Ali", 25)            # Konumsal
bilgi(yas=25, isim="Ali")   # Anahtar kelimeyle
```

*ARGS KONUMSAL PARAMETRELİ FONKSİYONLAR

Fonksiyona istenilen sayıda konumsal (sırayla verilen) argüman göndermek için kullanılır. *args bir tuple (demet) olur.

```
def topla(*args):
    print(args)
    return sum(args)

topla(1, 2, 3)  # Output: (1, 2, 3)
```

**KWARGS ANAHTAR KELİME PARAMETRELİ FONKSİYONLAR

Fonksiyona istenilen sayıda isimli (anahtar=değer şeklinde) argüman göndermek için kullanılır. **kwargs bir dict (sözlük) olur.

```
def ayarlar(**kwargs):
    print(kwargs)

ayarlar(renk="mavi", boyut=12)  # Output: {'renk': 'mavi', 'boyut': 12}
```

İÇ İÇE FONKSİYONLAR (NESTED FUNCTİONS)

Fonksiyon içinde bir fonksiyon tanımlanabilir. İç fonksiyon yalnızca tanımlandığı kapsamda (fonsiyon içinde) kullanılabilir.

```
def dis_fonksiyon():
    def ic_fonksiyon():
        print("İç fonksiyon")
    ic_fonksiyon()
```

BİRDEN FAZLA GERİYE DEĞER DÖNEN FONKSİYONLAR

Fonksiyon geriye tuple olarak birden fazla değer dönebilir.

return deger1, deger2,deger3, ...

```
def islemler(a, b):
    return a + b, a * b
```

REKÜRSİF (ÖZYİNELEMELİ) FONKSİYONLAR

Fonksiyon kendini çağırıyorsa bu fonksiyon özyinelemelidir.

Dikkat: Burada bir noktada bir şarta bağlı olarak kendini çağırması sonlandırılmalıdır.

```
def faktoriyel(n):
    if n == 0:
        return 1
    return n * faktoriyel(n - 1)
```

Kendini çağırmayı sonlandır

Kendini çağır

MİNİ PROJE – KAHVE DÜKKANI

Şimdi aşağıdaki örneği yapalım.

Bu uygulamayı bir kahve dükkanı gibi düşünebilirsin.

- Müşteriler sipariş veriyor
- Garson Siparişi alıyor
- Fiyat hesaplıyor
- İndirim uyguluyor (varsayılan : %5 yada girilen)
- Fatura oluşturuyor

MİNİ PROJE – KAHVE DÜKKANI

1- Öncelikle bir ürün-fiyat listemiz var. Yani bir sözlük!

```
menu = {
    "kahve":60,
    "çay":20,
    "pasta":90
}
```

MİNİ PROJE – KAHVE DÜKKANI

2- Müşteri bu fonksiyon yardımı ile siparişini verecek. İstediği ürünleri girer ve siparişi bittiğinde boş girer. Menüde olan ürünlerden girmek zorundadır.

```
def siparis_ver():
  liste = []
  while True:
    secim = input("Seçiminizi girin[kahve, çay, pasta,*bitirmek için boş giriş]:")
    if secim == "":
      return liste
    if secim in menu:
      liste.append(secim)
    else:
      print("Menüde olmayan ürün girdiniz!")
```

MİNİ PROJE – KAHVE DÜKKANI

3- Garson ise müşteriden gelen siparişi aşağıdaki fonksiyon yardımı ile oluşturacak.

```
def siparis_al(musteri_adi, urunler):
  return {"musteri": musteri_adi, "urunler": urunler}
```

MİNİ PROJE – KAHVE DÜKKANI

4- Sipariş edilen ürünlerin toplam tutarı hesaplanacak.

```
def hesapla_toplam(urunler):
  toplam = 0
  for urun in urunler:
    toplam += menu[urun]
  return toplam
```

MİNİ PROJE – KAHVE DÜKKANI

5- Müşteriye bir indirim uygulanacak. Oran girilmez ise %5 indirim uygulanır.

```
def indirim_uygula(toplam, oran = 5):
    return toplam * (100-oran) / 100
```

MİNİ PROJE – KAHVE DÜKKANI

6- Son olarak fatura yazdırılır.

```
def fatura_yaz(siparis, toplam, odenecek):
    print("---- FATURA ----")
    print("Müşteri:", siparis["musteri"])
    print("Ürünler:", ", ".join(siparis["urunler"]))
    print(f"Toplam: {toplam} TL")
    print(f"Ödenecek: {odenecek} TL")
```

MİNİ PROJE – KAHVE DÜKKANI

7- Şimdi artık tüm fonksiyonlarımızı ana program bölümünde mantıklı bir sırayla çağıralım..

```
isim = input("Merhaba, adınız nedir?")
print("Şimdi siparişinizi verebilirsiniz. Menümüz:", menu)
urunler = siparis_ver()

siparis = siparis_al(isim, urunler)

toplam = hesapla_toplam(urunler)
odenecek = indirim_uygula(toplam, 15)

fatura_yaz(siparis, toplam, odenecek)
```

MİNİ PROJE – KAHVE DÜKKANI

Uygulamanın çıktısı aşağıdaki gibi olacaktır.

```
Merhaba, adınız nedir?Mustafa
Şimdi siparişinizi verebilirsiniz. Menümüz: {'kahve': 60, 'çay': 20, 'pasta': 90}
Seçiminizi girin[kahve, çay, pasta,*bitirmek için boş giriş]:çay
Seçiminizi girin[kahve, çay, pasta,*bitirmek için boş giriş]:pasta
Seçiminizi girin[kahve, çay, pasta,*bitirmek için boş giriş]:
---- FATURA ----
Müşteri: Mustafa
Ürünler: çay, pasta
Toplam: 110 TL
Ödenecek: 93.5 TL
```