PYTHON LİSTELER

```
listem = ["elma", "muz", "kiraz"]
```

Liste

Listeler, birden çok öğeyi tek bir değişkende saklamak için kullanılır.

Listeler, Python'da veri koleksiyonlarını depolamak için kullanılan 4 yerleşik veri türünden biridir, diğer 3'ü Tuple, Set ve Dictionary'dir ve tümü farklı niteliklere ve kullanıma sahiptir.

Listeler köşeli parantezler kullanılarak oluşturulur:

Örnek

Liste oluştur:

```
listem = ["elma", "muz", "kiraz"]
print(listem)
```

Öğeleri Listele

Liste öğeleri sıralıdır, değiştirilebilir ve yinelenen değerlere izin verir.

Liste öğeleri indekslenir, ilk öğenin indeksi [0], ikinci öğenin indeksi [1] vardır vb.

Sipariş Edildi

Listeler sıralanır dediğimizde, öğelerin belirli bir sıraya sahip olduğu ve bu sıranın değişmeyeceği anlamına gelir.

Bir listeye yeni öğeler eklerseniz, yeni öğeler listenin sonuna yerleştirilir.

NOT: Sıralamayı değiştirecek bazı liste yöntemleri vardır, ancak genel olarak: öğelerin sırası değişmez.

Değiştirilebilirlik

Liste değiştirilebilir, yani bir liste oluşturulduktan sonra listedeki öğeleri değiştirebilir, ekleyebilir ve kaldırabiliriz.

Çoğaltmaya İzin Ver

Listeler indekslendiğinden, listeler aynı değere sahip öğelere sahip olabilir:

Örnek

Listeler, yinelenen değerlere izin verir:

```
listem = ["elma", "muz", "kiraz", "elma", "kiraz"]
print(listem)
```

Liste Uzunluğu

Bir listenin kaç öğeye sahip olduğunu belirlemek için len() işlevini kullanın:

Örnek

Listedeki öğelerin sayısını yazdırın:

```
listem = ["elma", "muz", "kiraz"]
print(len(listem))
```

Liste Öğeleri - Veri Türleri

Liste öğeleri herhangi bir veri türünde olabilir:

Örnek

Dize, int ve boolean veri türleri:

```
liste1 = ["elma", "muz", "kiraz"]
liste2 = [1, 5, 7, 9, 3]
liste3 = [True, False, False]
```

Bir liste farklı veri türleri içerebilir:

Örnek

Dizeler, tam sayılar ve boole değerleri içeren bir liste:

```
liste1 = ["abc", 28, True, 35, "erkek"]
```

type()

Python'un bakış açısından, listeler 'liste' veri tipine sahip nesneler olarak tanımlanır:

```
<class 'list'>
```

Örnek

Bir listenin veri türü nedir?

```
listem = ["elma", "muz", "kiraz"]
print(type(listem))
```

list() Yapıcısı/Kurucusu

Yeni bir liste oluştururken list() kurucusunu kullanmak da mümkündür.

Örnek

Liste yapmak için list() kurucusunu kullanma:

```
buListe = list(("elma", "muz", "kiraz")) # çift standart parantezlere
dikkat edin
print(buListe)
```

Python Koleksiyonları (Diziler)

Python programlama dilinde dört toplama veri türü vardır:

- Liste, sıralı ve değiştirilebilir bir koleksiyondur. Yinelenen üyelere izin verir.
- Tuple, sıralı ve değişmez bir koleksiyondur. Yinelenen üyelere izin verir.
- Set, sırasız, değiştirilemez* ve indekslenmemiş bir koleksiyondur. Çift üye yok.
- Sözlük, sıralı** ve değiştirilebilir bir koleksiyondur. Çift üye yok.

*Set öğeleri değiştirilemez, ancak istediğiniz zaman öğeleri kaldırabilir ve/veya ekleyebilirsiniz.

**Python sürüm 3.7'den itibaren sözlükler sıralanmıştır. Python 3.6 ve önceki sürümlerde sözlükler sırasızdır.

Bir koleksiyon türü seçerken, o türün özelliklerini anlamakta fayda var. Belirli bir veri seti için doğru türün seçilmesi, anlamın korunması anlamına gelebilir ve verimlilik veya güvenlikte bir artış anlamına gelebilir.

Elemanlara Erişim

Liste öğeleri indekslenir ve bunlara indeks numarasına başvurarak erişebilirsiniz:

Örnek

Listenin ikinci öğesini yazdırın:

```
listem = ["elma", "muz", "kiraz"]
print(listem[1])
```

NOT: İlk öğenin indeksi 0'dır.

Negatif İndeksleme

Negatif indeksleme, sondan başlamak anlamına gelir

-1 son öğeyi, -2 sondan ikinci öğeyi vb. belirtir.

Örnek

Listenin son öğesini yazdırın:

```
buListe = ["elma", "muz", "kiraz"]
print(buListe[-1])
```

Dizin Aralığı

Aralığın nereden başlayacağını ve nerede biteceğini belirterek bir dizin aralığı belirtebilirsiniz.

Bir aralık belirtirken, dönüş değeri, belirtilen öğeleri içeren yeni bir liste olacaktır.

Örnek

Üçüncü, dördüncü ve beşinci öğeyi döndürün:

```
buListe =
["elma", "muz", "kiraz", "portakal", "kivi", "kavun", "mango"]
print(buListe[2:5])
```

NOT: Arama, dizin 2'de (dahil) başlar ve dizin 5'te (dahil değildir) biter.

İlk öğenin 0 indeksine sahip olduğunu unutmayın.

Başlangıç değerini dışarıda bırakarak, aralık ilk öğeden başlayacaktır:

Örnek

Bu örnek, öğeleri baştan sona döndürür, ancak "kivi" dahil DEĞİLDİR:

```
buListe =
["elma", "muz", "kiraz", "portakal", "kivi", "kavun", "mango"]
print(buListe[:4])
```

Bitiş değerini dışarıda bırakarak, aralık listenin sonuna kadar devam edecektir:

Örnek

Bu örnek, "kiraz" dan sona kadar olan öğeleri döndürür:

```
buListe =
["elma", "muz", "kiraz", "portakal", "kivi", "kavun", "mango"]
print(buListe [2:])
```

Negatif Endeks Aralığı

Aramayı listenin sonundan başlatmak istiyorsanız, negatif dizinleri belirtin:

Örnek

Bu örnek, "portakal"dan (-4)'e kadar olan öğeleri döndürür, ancak "mango" (-1) dahil DEĞİLDİR:

```
buListe =
["elma", "muz", "kiraz", "portakal", "kivi", "kavun", "mango"]
print(buListe [-4:-1])
```

Öğenin Var olup olmadığını kontrol edin

Listede belirtilen bir öğenin olup olmadığını belirlemek için in anahtar sözcüğünü kullanın:

Örnek

Listede "elma" olup olmadığını kontrol edin:

```
buListe = ["elma", "muz", "kiraz"]
if "elma" in buListe:
  print("Evet, 'elma' meyveler listesinde mevcuttur")
```

Liste Öğelerini Değiştir

Belirli bir öğenin değerini değiştirmek için dizin numarasına bakın:

Örnek

İkinci öğeyi değiştirin:

```
buListe = ["elma", "muz", "kiraz"]
buListe[1] = "frenk üzümü"
print(buListe)
```

Öğe Değerleri Aralığını Değiştirin

Belirli bir aralıktaki öğelerin değerini değiştirmek için, yeni değerlerle bir liste tanımlayın ve yeni değerleri eklemek istediğiniz dizin numarası aralığına bakın:

Örnek

"Muz" ve "kiraz" değerlerini "frenk üzümü" ve "karpuz" değerleriyle değiştirin:

```
buListe = ["elma", "muz", "kiraz", "portakal", "kivi", "mango"]
buListe[1:3] = ["frenk üzümü", "karpuz"]
print(buListe)
```

Değiştirdiğinizden daha fazla öğe eklerseniz, yeni öğeler belirttiğiniz yere eklenir ve kalan öğeler buna göre (sağa/sola ötelenerek) hareket eder:

Örnek

İkinci değeri (muz) yerine iki yeni değer (frenk üzümü ve karpuz) koyarak değiştirin:

```
buListe = ["elma", "muz", "kiraz"]
buListe[1:2] = ["frenk üzümü", "karpuz"]
print(buListe)
```

NOT: Eklenen öğelerin sayısı değiştirilen öğelerin sayısıyla eşleşmediğinde listenin uzunluğu değişecektir.

Değiştirdiğinizden daha az öğe eklerseniz, yeni öğeler belirttiğiniz yere eklenir ve kalan öğeler buna göre (sağa/sola ötelenerek) hareket eder:

Örnek

İkinci ve üçüncü değeri bir değerle değiştirerek değiştirin:

```
buListe = ["elma", "muz", "kiraz"]
buListe[1:3] = ["karpuz"]
print(buListe)
```

Öğe Ekleme

Mevcut değerlerden herhangi birini değiştirmeden yeni bir liste öğesi eklemek için insert() yöntemini kullanabiliriz.

insert() yöntemi, belirtilen dizine bir öğe ekler:

Örnek

Üçüncü öğe olarak "karpuz" ekleyin:

```
buListe = ["elma", "muz", "kiraz"]
buListe.insert(2, "karpuz")
print(buListe)
```

NOT: Yukarıdaki örneğin bir sonucu olarak, liste artık 4 öğe (önceden 3'tü) içerecektir.

Listeye Eleman Ekleme

Öğe İliştirme

Listenin sonuna bir öğe eklemek için append() yöntemini kullanın:

Örnek

Bir öğe eklemek için append() yöntemini kullanma:

```
buListe = ["elma", "muz", "kiraz"]
buListe.append("portakal")
print(buListe)
```

Öğe Ekleme

Belirtilen dizine bir liste öğesi eklemek için insert() yöntemini kullanın.

insert() yöntemi, belirtilen dizine bir öğe ekler:

Örnek

İkinci konum olarak bir öğe ekleyin:

```
buListe = ["elma", "muz", "kiraz"]
buListe.insert(1, "portakal")
print(buListe)
```

Listeyi Genişletme

Geçerli listeye başka bir listeden öğeler eklemek için extension() yöntemini kullanın.

Örnek

Tropikal unsurları bu listeye ekleyin:

```
buListe = ["elma", "muz", "kiraz"]
tropikal = ["mango", "ananas", "papaya"]
buListe.extend(tropikal)
print(buListe)
```

Öğeler listenin sonuna eklenecektir.

Yinelenebilir Herhangi Bir Ekle

Extend() yönteminin listeler eklemesi gerekmez, yinelenebilir herhangi bir nesne (tuple'lar, kümeler, sözlükler vb.) ekleyebilirsiniz.

Örnek

Bir listeye bir demetin öğelerini ekleyin:

```
buListe = ["elma", "muz", "kiraz"]
buTuple = ["kivi", "portakal"]
buListe.extend(buTuple)
print(buListe)
```

Listede Belirtilen Öğelerini Kaldır

Remove() yöntemi, belirtilen öğeyi kaldırır.

Örnek

"Muz"u kaldırın:

```
buListe = ["elma", "muz", "kiraz"]
buListe.remove("muz")
print(buListe)
```

Belirtilen Dizini Kaldır

pop() yöntemi, belirtilen dizini kaldırır.

Örnek

İkinci öğeyi kaldırın:

```
buListe = ["elma", "muz", "kiraz"]
buListe.pop(1)
print(buListe)
```

Dizini belirtmezseniz, pop() yöntemi son öğeyi kaldırır.

Örnek

Son öğeyi kaldırın:

```
buListe = ["elma", "muz", "kiraz"]
buListe.pop()
print(buListe)
```

del anahtar sözcüğü ayrıca belirtilen dizini de kaldırır:

Örnek

İlk öğeyi kaldırın:

```
buListe = ["elma", "muz", "kiraz"]
del buListe[0]
print(buListe)
```

del anahtar sözcüğü ayrıca listeyi tamamen silebilir.

Örnek

Tüm listeyi silin:

```
buListe = ["elma", "muz", "kiraz"]
del buListe
```

Listeyi Temizle

clear() yöntemi listeyi boşaltır.

Liste hala duruyor, ancak içeriği yok.

Örnek

Liste içeriğini temizle:

```
buListe = ["elma", "muz", "kiraz"]
buListe.clear()
print(buListe)
```

Listelerde Döngü

Bir for döngüsü kullanarak liste öğeleri arasında dolaşabilirsiniz:

Örnek

Listedeki tüm öğeleri tek tek yazdırın:

```
buListe = ["elma", "muz", "kiraz"]
for x in buListe:
  print(x)
```

Python For Loops Bölümümüzde for döngüleri hakkında daha fazla bilgi edinin.

Dizin Numaralarında Döngü

Ayrıca, dizin numaralarına başvurarak liste öğeleri arasında dolaşabilirsiniz.

Uygun bir yinelenebilir oluşturmak için range() ve len() işlevlerini kullanın.

Örnek

Dizin numaralarına bakarak tüm öğeleri yazdırın:

```
buListe = ["elma", "muz", "kiraz"]
for i in range(len(buListe)):
  print(buListe[i])
```

Yukarıdaki örnekte oluşturulan yinelenebilir [0, 1, 2] şeklindedir.

Bir While Döngüsü Kullanmak

Bir while döngüsü kullanarak liste öğeleri arasında dolaşabilirsiniz.

Listenin uzunluğunu belirlemek için len() işlevini kullanın, ardından 0'dan başlayın ve dizinlerine başvurarak liste öğeleri arasında dolaşın.

Her yinelemeden sonra dizini 1 artırmayı unutmayın.

Örnek

Tüm dizin numaralarından geçmek için bir while döngüsü kullanarak tüm öğeleri yazdırın

```
buListe = ["elma", "muz", "kiraz"]
i = 0
while i < len(buListe):
  print(buListe[i])
  i = i + 1
```

Liste Kapsamını Kullanarak Döngü Yapma

Liste Anlama, listeler arasında dolaşmak için en kısa sözdizimini sunar:

Örnek

Bir listedeki tüm öğeleri yazdıracak kısa bir gösterimli for döngüsü:

```
buListe = ["elma", "muz", "kiraz"]
[print(x) for x in thislist]
```

Liste Kapsamı

Liste anlama, mevcut bir listenin değerlerine dayalı olarak yeni bir liste oluşturmak istediğinizde daha kısa bir sözdizimi sunar.

Örnek:

Bir meyve listesine dayanarak, sadece adında "a" harfi olan meyveleri içeren yeni bir liste istiyorsunuz.

Liste kapsam olmadan, içinde koşullu bir test içeren bir for ifadesi yazmanız gerekecek:

Örnek

```
meyveler = ["elma", "muz", "kiraz", "kivi", "mango"]
yeniListe = []

for x in meyveler:
  if "a" in x:
    yeniListe.append(x)

print(yeniListe)
```

Liste kapsama ile tüm bunları yalnızca bir kod satırıyla yapabilirsiniz:

Örnek

```
meyveler = ["elma", "muz", "kiraz", "kivi", "mango"]

yeniListe = [x for x in meyveler if "a" in x]

print(yeniListe)
```

Sözdizimi

```
yeniListe = [ifade for eleman in yinelenebilir if şart == True]
```

Dönüş değeri, eski listeyi değiştirmeden bırakan yeni bir listedir.

Koşul

Koşul, yalnızca True olarak değerlenen öğeleri kabul eden bir filtre gibidir.

Örnek

Yalnızca "elma" olmayan öğeleri kabul edin:

```
yeniListe = [x for x in meyveler if x != "elma"]
```

`if x != "elma"` koşulu, "elma" dışındaki tüm öğeler için True değerini döndürür ve yeni listenin "elma" dışındaki tüm meyveleri içermesini sağlar.

Koşul isteğe bağlıdır ve atlanabilir:

Örnek

Hayır if deyimi ile:

```
yeniListe = [x for x in meyveler]
```

Yinelenebilir

Yinelenebilir, bir liste, grup, küme vb. gibi yinelenebilir herhangi bir nesne olabilir.

Örnek

Yinelenebilir bir öğe oluşturmak için range() işlevini kullanabilirsiniz:

```
yeniListe = [x for x in range(10)]
```

Aynı örnek, ancak bir koşulla:

Örnek

Yalnızca 5'ten küçük sayıları kabul edin:

```
yeniListe = [x for x in range(10) if x < 5]
```

İfade

İfade, yinelemedeki geçerli öğedir, ancak aynı zamanda, yeni listedeki bir liste öğesi gibi sona ermeden önce değiştirebileceğiniz sonuçtur:

Örnek

Yeni listedeki değerleri büyük harfe ayarlayın:

```
yeniListe = [x.upper() for x in meyveler]
```

Sonucu istediğiniz gibi ayarlayabilirsiniz:

Örnek

Yeni listedeki tüm değerleri 'merhaba' olarak ayarlayın:

```
yeniListe = ['merhaba' for x in meyveler]
```

İfade, bir filtre gibi değil, sonucu değiştirmenin bir yolu olarak koşulları da içerebilir:

Misal

"muz" yerine "portakal" döndürün:

```
yeniListe = [x if x != "muz" else "portakal" for x in meyveler]
```

Yukarıdaki örnekteki ifade şöyle diyor:

"Muz değilse öğeyi iade edin, muz ise portakalı iade edin".

Listelerde Sıralama

Listeyi Alfanümerik Olarak Sırala

Liste nesnelerinin, listeyi varsayılan olarak artan, alfasayısal olarak sıralayacak bir sort() yöntemi vardır:

Örnek

Listeyi alfabetik olarak sıralayın:.

```
buListe = ["portakal", "mango", "kivi", "ananas", "muz"]
buListe.sort()
print(buListe)
```

Örnek

Listeyi sayısal olarak sıralayın:

```
buListe = [100, 50, 65, 82, 23]
buListe.sort()
print(buListe)
```

Azalan şekilde sırala

Azalan sıralama için, reverse = True anahtar kelime bağımsız değişkenini kullanın:

Örnek

Listeyi azalan şekilde sıralayın:

```
buListe = ["portakal", "mango", "kivi", "ananas", "muz"]
buListe.sort(reverse = True)
print(buListe)
```

Örnek

Listeyi azalan şekilde sıralayın:

```
buListe = [100, 50, 65, 82, 23]
buListe.sort(reverse = True)
print(buListe)
```

Sıralama İşlevini Özelleştir

Ayrıca, anahtar = işlev anahtar sözcüğünü kullanarak kendi işlevinizi özelleştirebilirsiniz.

İşlev, listeyi sıralamak için kullanılacak bir sayı döndürür (önce en düşük sayı):

Örnek

Sayının 50'ye ne kadar yakın olduğuna göre listeyi sıralayın:

```
def fonk(n):
  return abs(n - 50)

buListe = [100, 50, 65, 82, 23]
buListe.sort(key = fonk)
print(buListe)
```

Büyük/Küçük Harfe Duyarsız Sıralama

Varsayılan olarak sort() yöntemi büyük/küçük harfe duyarlıdır ve tüm büyük harflerin küçük harflerden önce sıralanmasına neden olur:

Örnek

Büyük/küçük harfe duyarlı sıralama beklenmedik bir sonuç verebilir:

```
buListe = ["muz", "Portakal", "Kivi", "kiraz"]
buListe.sort()
print(buListe)
```

Neyse ki, bir listeyi sıralarken yerleşik işlevleri temel işlevler olarak kullanabiliriz.

Bu nedenle, büyük/küçük harfe duyarlı olmayan bir sıralama işlevi istiyorsanız, anahtar işlev olarak str.lower öğesini kullanın:

Örnek

Listenin büyük/küçük harfe duyarlı olmayan bir sıralamasını gerçekleştirin:

```
buListe = ["muz", "Portakal", "Kivi", "kiraz"]
buListe.sort(key = str.lower)
print(buListe)
```

Ters Sıralama

Alfabeden bağımsız olarak bir listenin sırasını tersine çevirmek isterseniz ne olur?

reverse() yöntemi, öğelerin geçerli sıralama düzenini tersine çevirir.

Örnek

Liste öğelerinin sırasını tersine çevirin:

```
buListe = ["muz", "Portakal", "Kivi", "kiraz"]
buListe.reverse()
print(buListe)
```

Liste Kopyalama

liste2 = liste1 yazarak bir listeyi kopyalayamazsınız çünkü: liste2 sadece liste1'e referans olacaktır ve liste1'de yapılan değişiklikler liste2'de de otomatik olarak yapılacaktır.

Bir kopya oluşturmanın yolları vardır, bir yol yerleşik List yöntemini copy() kullanmaktır.

Örnek

copy() yöntemiyle bir listenin bir kopyasını oluşturun:

```
buListe = ["elma", "muz", "kiraz"]
listem = buListe.copy()
print(listem)
```

Bir kopya oluşturmanın başka bir yolu da yerleşik yöntem list()'i kullanmaktır.

Örnek

list() yöntemiyle bir listenin bir kopyasını oluşturun:

```
buListe = ["elma", "muz", "kiraz"]
listem = list(buListe)
print(listem)
```

Listeleri Birleştirme

Python'da iki daha fazla listeyi birleştirmenin veya birleştirmenin birkaç yolu vardır.

En kolay yollardan biri + operatörünü kullanmaktır.

Örnek

İki listeyi birleştirin:

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```

İki listeyi birleştirmenin başka bir yolu da list2'deki tüm öğeleri liste1'e tek tek eklemektir:

Örnek

list2'yi list1'e ekleyin:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

print(list1)
```

Veya bir listeden başka bir listeye öğe eklemek olan extension() yöntemini kullanabilirsiniz:

Örnek

list1'in sonuna list2 eklemek için extension() yöntemini kullanın:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```

Liste Metotları

Python, listelerde kullanabileceğiniz bir dizi yerleşik yönteme sahiptir.

Metot	Tanımlama
append()	Listenin sonuna bir öğe ekler
clear()	Tüm öğeleri listeden kaldırır
copy()	Listenin bir kopyasını döndürür
count()	Belirtilen değere sahip öğelerin sayısını döndürür
extend()	Geçerli listenin sonuna bir listenin öğelerini (veya herhangi bir yinelenebilir) ekleyin
index()	Belirtilen değere sahip ilk öğenin dizinini döndürür
insert()	Belirtilen konuma bir eleman ekler
pop()	Belirtilen konumdaki öğeyi kaldırır
remove()	Belirtilen değere sahip öğeyi kaldırır
reverse()	Listenin sırasını tersine çevirir
sort()	Listeyi sıralar

PYTHON TUPLES (DEMETLER)

Tuple'lar, birden çok öğeyi tek bir değişkende saklamak için kullanılır.

Tuple, Python'da veri koleksiyonlarını depolamak için kullanılan 4 yerleşik veri türünden biridir, diğer 3'ü Liste, Küme ve Sözlük'tür ve tümü farklı niteliklere ve kullanıma sahiptir.

Tuple, sıralı ve değiştirilemez bir koleksiyondur.

Tuples standart parantez ile yazılır.

Örnek

Bir Tuple oluşturun:

```
buTuple = ("elma", "muz", "kiraz")
print(thistuple)
```

Tuple Öğeleri

Tuple öğeleri sıralanır, değiştirilemez ve yinelenen değerlere izin verir.

Tuple öğeleri indekslenir, ilk öğenin indeksi [0], ikinci öğenin indeksi [1] vardır vb.

Sıralılık (İlk tanımlandığı sıralamayı koruma)

Tuple'ların sıralı olduğunu söylediğimizde, öğelerin belirli bir sıraya sahip olduğu ve bu sıranın değişmeyeceği anlamına gelir.

Değiştirilemez

Tuple'lar değiştirilemez, yani tuple oluşturulduktan sonra öğeleri değiştiremez, ekleyemez veya çıkaramayız.

Kopyalara İzin Verir

Tuple'lar indekslendiğinden, aynı değere sahip öğelere sahip olabilirler:

Örnek

Tuple'lar, yinelenen değerlere izin verir:

```
buTuple = ("elma", "muz", "kiraz", "elma", "kiraz")
print(buTuple)
```

Tuple Uzunluğu

Bir demetin kaç öğeye sahip olduğunu belirlemek için len() işlevini kullanın:

Örnek

Tuple'daki öğe sayısını yazdırın:

```
buTuple = ("elma", "muz", "kiraz")
print(len(buTuple))
```

Bir Eleman ile Tuple Oluşturun

Yalnızca bir öğe içeren bir demet oluşturmak için öğeden sonra virgül eklemeniz gerekir, aksi takdirde Python onu bir demet olarak tanımaz.

Örnek

Bir öğe demeti, virgül unutmayın:

```
buTuple = ("elma",)
print(type(buTuple))

#NOT a tuple
buTuple = ("elma ")
print(type(buTuple))
```

Tuple Öğeleri - Veri Türleri

Tuple öğeleri herhangi bir veri türünde olabilir:

Örnek

Dize, int ve boolean veri türleri:

```
tuple1 = ("elma", "muz", "kiraz")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)
```

Bir demet, farklı veri türleri içerebilir:

Örnek

Dizeler, tamsayılar ve boole değerleri içeren bir demet:

```
tuple1 = ("abc", 28, True, 35, "erkek")
```

type()

Python'un bakış açısından, tanımlama grupları, veri türü "tuple" olan nesneler olarak tanımlanır:

```
<class 'tuple'>
```

Örnek

Bir demetin veri türü nedir?

```
tuplem = ("elma", "muz", "kiraz")
print(type(tuplem))
```

Tuple() Yapıcısı/Kurucusu

Bir demet oluşturmak için tuple() yapıcısını kullanmak da mümkündür.

Örnek

Bir demet oluşturmak için tuple() yöntemini kullanma:

```
buTuple = tuple(("apple", "banana", "cherry")) # İki tane parantez olduğuna dikkat ediniz
print(buTuple)
```

Python Koleksiyonları (Diziler)

Python programlama dilinde dört toplama veri türü vardır:

- Liste, sıralı ve değiştirilebilir bir koleksiyondur. Yinelenen üyelere izin verir.
- Tuple, sıralı ve değiştirilemez bir koleksiyondur. Yinelenen üyelere izin verir.
- Set, sıralanmamış, değiştirilemez* ve indekslenmemiş bir koleksiyondur. Çift üye yok.
- Sözlük, sıralı** ve değiştirilebilir bir koleksiyondur. Çift üye yok.

*Set öğeleri değiştirilemez, ancak istediğiniz zaman öğeleri kaldırabilir ve/veya ekleyebilirsiniz.

**Python sürüm 3.7'den itibaren sözlükler sıralanmıştır. Python 3.6 ve önceki sürümlerde sözlükler sırasızdır.

Bir koleksiyon türü seçerken, o türün özelliklerini anlamakta fayda var. Belirli bir veri seti için doğru türün seçilmesi, anlamın korunması anlamına gelebilir ve verimlilik veya güvenlikte bir artış anlamına gelebilir.

Tuple'a Erişim

Tuple Öğelerine Erişim

Tuple öğelerine, köşeli parantez içindeki dizin numarasına bakarak erişebilirsiniz:

Örnek

Tuple'daki ikinci öğeyi yazdırın:

```
buTuple = ("elma", "muz", "kiraz")
print(buTuple[1])
```

NOT: İlk öğenin indeksi 0'dır.

Negatif İndeksleme

Negatif indeksleme, sondan başlamak anlamına gelir.

-1 son öğeyi, -2 ikinci son öğeyi vb. belirtir.

Örnek

Tuple'ın son öğesini yazdırın:

```
buTuple = ("elma", "muz", "kiraz")
print(buTuple[-1])
```

Dizin/İndis Aralığı

Aralığın nereden başlayacağını ve nerede biteceğini belirterek bir dizin aralığı belirtebilirsiniz.

Bir aralık belirtirken, dönüş değeri, belirtilen öğelerle yeni bir demet olacaktır.

Örnek

Üçüncü, dördüncü ve beşinci öğeleri döndürün:

```
buTuple =("elma", "muz", "kiraz", "portakal", "kivi", "kavun", "mango")
print(buTuple[2:5])
```

NOT: Arama, dizin 2'de (dahil) başlar ve dizin 5'te (dahil değildir) biter.

İlk öğenin 0 indeksine sahip olduğunu unutmayın.

Başlangıç değerini dışarıda bırakarak, aralık ilk öğeden başlayacaktır:

Örnek

Bu örnek, öğeleri baştan "kivi"ye kadar döndürür, ancak buna dahil DEĞİLDİR:

```
buTuple = ("elma", "muz", "kiraz", "portakal", "kivi", "kavun", "mango")
print(buTuple[:4])
```

Bitiş değerini dışarıda bırakarak, aralık listenin sonuna kadar devam edecektir:

Örnek

Bu örnek, "kiraz" dan ve sonuna kadar olan öğeleri döndürür:

Negatif Endeks Aralığı

Aramayı demetin sonundan başlatmak istiyorsanız, negatif dizinleri belirtin:

Örnek

Bu örnek, dizin -4'ten (dahil) dizin -1'e (hariç tutulan) öğeleri döndürür

```
buTuple = ("elma", "muz", "kiraz", "portakal", "kivi", "kavun", "mango")
print(buTuple[-4:-1])
```

Öğenin Var olup olmadığını kontrol edin

Bir demette belirli bir öğenin bulunup bulunmadığını belirlemek için in anahtar sözcüğünü kullanın:

Örnak

Tuple'da "elma" olup olmadığını kontrol edin:

```
buTuple = ("elma", "muz", "kiraz")
if "elma" in buTuple:
  print("Evet, 'elma' meyveler tuple'ında vardır.")
```

Tuple Güncelleştirme

Tanımlama grupları değiştirilemez, yani tanımlama grubu oluşturulduktan sonra öğeleri değiştiremez, ekleyemez veya kaldıramazsınız.

Ama bazı geçici çözümler var.

Grup Değerlerini Değiştir

Bir demet oluşturulduktan sonra değerlerini değiştiremezsiniz. Tuple'lar değiştirilemez veya değiştirilemez olarak da adlandırılır.

Ama bir geçici çözüm var. Tuple'ı bir listeye dönüştürebilir, listeyi değiştirebilir ve listeyi tekrar bir Tuple'a dönüştürebilirsiniz.

Örnek

Değiştirebilmek için demeti bir listeye dönüştürün:

```
x = ("elma", "muz", "kiraz")
y = list(x)
y[1] = "kivi"
x = tuple(y)
print(x)
```

Öğe/Eleman Ekleme

Gruplar değişmez olduklarından, yerleşik bir append() yöntemine sahip değildirler, ancak bir demete öğe eklemenin başka yolları da vardır.

1. Listeye dönüştürün: Bir demeti değiştirmeye yönelik geçici çözüm gibi, onu bir listeye dönüştürebilir, öğe(ler)inizi ekleyebilir ve tekrar bir demete dönüştürebilirsiniz.

Örnek

Tuple'ı bir listeye dönüştürün, "turuncu" ekleyin ve onu tekrar bir Tuple'a dönüştürün:

```
buTuple = ("elma", "muz", "kiraz")
y = list(buTuple)
y.append("portakal")
buTuple = tuple(y)
```

2. Bir demete demet ekleyin. Tuple'a demet eklemenize izin verilir, bu nedenle bir veya daha fazla öğe eklemek için, öğe(ler) ile yeni bir demet oluşturun ve onu mevcut demete ekleyin:

Örnek

"Orange" değerine sahip yeni bir demet oluşturun ve bu demeti ekleyin:

```
buTuple = ("elma", "muz", "kiraz")
y = ("portakal",)
buTuple += y
print(buTuple)
```

NOT: Yalnızca bir öğe içeren bir demet oluştururken, öğeden sonra bir virgül eklemeyi unutmayın, aksi takdirde bir demet olarak tanımlanmayacaktır.

Öğeleri Silme/Kaldırma

NOT: Bir tanımlama grubundaki öğeleri kaldıramazsınız.

Tuple'lar değiştirilemez, bu nedenle ondan öğeleri kaldıramazsınız, ancak tuple öğelerini değiştirmek ve eklemek için kullandığımızla aynı geçici çözümü kullanabilirsiniz:

Örnek

Tuple'ı bir listeye dönüştürün, "elmayı" kaldırın ve tekrar bir Tuple'a dönüştürün:

```
buTuple = ("elma", "muz", "kiraz")
y = list(buTuple)
y.remove("elma")
buTuple = tuple(y)
```

Veya demeti tamamen silebilirsiniz:

Örnek

del anahtar sözcüğü, demeti tamamen silebilir:

```
buTuple = ("elma", "muz", "kiraz")
del buTuple
print(buTuple) #bu bir hata döndürecektir, çünkü tuple kalıcı olarak
silinmiştir.
```

Bir Tuple'ı Açmak

Bir demet oluşturduğumuzda, normalde ona değerler atarız. Buna bir demet "paketleme" denir:

Örnek

Bir demet paketleme:

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

Ancak Python'da değerleri değişkenlere geri almamıza da izin verilir. Buna "paket açma" denir:

Örnek

Bir demeti açma:

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

(yeşil, sari, kirmizi) = meyveler

print(yesil)
print(sari)
print(kirmizi)
```

NOT: Değişkenlerin sayısı, tanımlama grubundaki değerlerin sayısıyla eşleşmelidir, değilse, kalan değerleri bir liste olarak toplamak için bir yıldız işareti kullanmanız gerekir.

Yıldız kullanma*

Değişken sayısı değer sayısından az ise, değişken adına * ekleyebilirsiniz ve değerler liste olarak değişkene atanacaktır:

Örnek

Değerlerin geri kalanını "kırmızı" adlı bir liste olarak atayın:

```
meyveler = ("elma", "muz", "kiraz", "çilek", "böğürtlen")

(yesil, sari, *kirmizi) = meyveler

print(yesil)
print(sari)
print(kirmizi)
```

Yıldız işareti sondan başka bir değişken adına eklenirse, Python, kalan değerlerin sayısı kalan değişkenlerin sayısıyla eşleşene kadar değişkene değerler atar.

Örnek

"Tropik" değişkene bir değerler listesi ekleyin:

```
meyveler = ("elma", "mango", "papaya", "ananas", "kiraz")

(yesil, *tropik, kirmizi) = meyveler

print(yesil)
print(tropik)
print(kirmizi)
```

Bir Tuple Üzerinden Döngü

Bir for döngüsü kullanarak demet öğeleri arasında döngü yapabilirsiniz.

Örnek

Öğeleri yineleyin ve değerleri yazdırın:

```
buTuple = ("elma", "muz", "kiraz")
for x in buTuple:
  print(x)
```

Dizin Numaralarında Döngü

Ayrıca dizin numaralarına bakarak tanımlama grubu öğeleri arasında geçiş yapabilirsiniz.

Uygun bir yinelenebilir oluşturmak için range() ve len() işlevlerini kullanın.

Örnek

Dizin numaralarına bakarak tüm öğeleri yazdırın:

```
buTuple = ("elma", "muz", "kiraz")
for i in range(len(buTuple)):
  print(buTuple[i])
```

Bir While Döngüsü Kullanmak

Bir while döngüsü kullanarak liste öğeleri arasında dolaşabilirsiniz.

Demetin uzunluğunu belirlemek için len() işlevini kullanın, ardından 0'dan başlayın ve dizinlerine başvurarak demet öğeleri arasında dolaşın.

Her yinelemeden sonra dizini 1 artırmayı unutmayın.

Örnek

Tüm dizin numaralarını gözden geçirmek için bir while döngüsü kullanarak tüm öğeleri yazdırın:

```
buTuple = ("elma", "muz", "kiraz")
i = 0
while i < len(buTuple):
  print(buTuple[i])
  i = i + 1
```

İki Tuple'ı Birleştirme

İki veya daha fazla demeti birleştirmek için + operatörünü kullanabilirsiniz:

Örnek

İki tuple'a katılın:

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

Tuples'ları Çarpma

Bir demetin içeriğini belirli bir sayıda çarpmak istiyorsanız, * operatörünü kullanabilirsiniz:

Örnek

Meyve demetini 2 ile çarpın:

```
meyveler = ("elma", "muz", "kiraz")
tuplem = meyveler * 2

print(tuplem)
```

Tuple Metotları

Python, tuple'larda kullanabileceğiniz iki yerleşik yönteme sahiptir.

Metot	Tanımlama
count()	Bir tanımlama grubunda belirtilen bir değerin kaç kez oluştuğunu döndürür
index()	Tuple'ı belirtilen bir değer için arar ve bulunduğu yerin konumunu döndürür