**Bölüm 4 Sıralı ve Liste Veri Yapısı**

Bu bölümde, bir veri koleksiyonunu yapılandırmanın ve bunlara erişmenin bir yolunu inceleyeceğiz. Özellikle, genellikle bir liste olarak adlandırılan doğrusal bir sırayla verileri düzenlemenin bir yoluna bakıyoruz.

---2

**Liste özellikleri**

* Aynı tip, ya da farklı tiplerde elemanlardan oluşabilir
* Sabit, ya da değişken uzunlukta olabilir.
* Elemanların değerini değiştirmek mümkün olabilir ya da olmayabilir.

**---3**

**Liste Yapıları**

* Liste kavramı, günlük liste kavramımıza benzer. Yapılacaklar listemizdeki öğeleri okuruz (erişiriz), öğeler ekleriz, öğelerin üzerini çizeriz (sileriz) vb. Daha sonra listelerin kullanımına bakacağız.

**Kullanışlı olması için gereken işlem türleri:**

Tipik olarak listenin boş veya dolu olduğunu kolayca anlayabilmeli ve eleman sayısını bulabilmeliyiz.

Ayrıca şu işlemler yaygın olarak gerekir:

* Listedeki maddeye sıra numarasından erişmek.
* Erişilen maddenin değerini değiştirmek.
* Listenin istenen yerine yeni eleman eklemek.
* Listenin sonuna yeni eleman eklemek.
* Listeden bir maddeyi silmek.

---4

**Liste nedir**

Liste, doğrusal bir veri yapısıdır, yani öğelerinin doğrusal bir sıralaması vardır. Her öğe sıfırdan başlayıp birer artarak giden dizin (index) ile adreslenir.

---5

Örneğin, aşağıdaki belirli bir hafta için günlük sıcaklıklar listesi gibi:

Dizin Sıcaklık

0: 22.1

1: 20.6

2: 19.8

3: 21.5

4: 20.9

5: 20.0

6: 21.2

Bu listede dizin-0 satırı Pazar gününün, dizin-1 satırı Pazartesi gününün sıcaklığını, daha sonrakiler de sırayla diğer günlerin sıcaklığını veriyor.

Programlama dillerinde dizinler genellikle 1 den değil 0 dan başlar. Bu yüzden, ***sıfır tabanlı dizin*** olarak adlandırılır**.**

**---6**

**Ortak Liste İşlemleri**

Listelerde yaygın olarak gerçekleştirilen işlemler şunları içerir:

* Erişim (dizinden öğeye ulaşma)
* Güncelleme (öğenin değerini değiştirme)
* Araya eklemek (insert)
* Sona eklemek (append)
* Silmek (remove delete kaldırmak)

---10

**Liste Geçişi**

Listedeki öğelere sırayla birer birer ulaşıp listenin tümünün üzerinden geçmeye **liste geçişi** denir.

Liste geçişi, bir listenin her bir öğesine tek tek erişme yoludur. Örneğin şu amaçlarla kullanılabilir:

* bir listede belirli bir öğeyi ararken
* bir listenin tüm öğelerini toplarken gerekir.

Örnek: Toplam değişkenini sıfır olarak başlatıp listedeki sayıları dizin sırasıyla toplayarak gidersek listenin tümünü geçtiğimizde toplamı elde ederiz

---11

 Toplam

 0

Dizin Sayılar

0: 5 5

1: 15 20

2: 45 65

3: 20 85

4: 30 115

—12

Örnek: Listede 65 ararsak dizin 0 dan başlayarak sırayla tüm değerlere bakmamız gerekir

 Bulundu

 False

Dizin Sayılar

0: 5 False

1: 15 False

2: 45 False

3: 20 True

4: 30 …

Amaç ilk 65 değerini bulmak ise, arama işlemini sayıyı bulduğumuzda sonlandırabiliriz. Amaç bütün 65 leri bulmak ise listenin üzerinden sonuna kadar geçmeliyiz.

---13

**Python'da Listeler (Sıralı Yapılar)**

Python’daki **liste** (list), uzunluğu ve içeriği programın çalışması sırasında değiştirilebilen doğrusal bir veri yapısıdır ve karışık tip öğeler barındırmaya izin verir.

Python listelerinde sıfır tabanlı dizinleme (indeksleme) kullanır. Bu çeşit dizinlemede eleman sayısı n olan, bir listede indeks değerleri 0 dan başlar n-1 de sonlanır.

---14

**Liste oluşturulması**

Listeler, köşeli parantezler içinde virgülle ayrılmış bir öğe listesiyle gösterilir, örneğin:

lst=[1, 2, 3], ya da

Ls=['bir', 'iki', 'üç'], veya

Kr=['elma', 50, True] gibi.

Bunlardan lst tamsayı (integer) tipte listedir. Ls dize (string) tipidir. Kr ise karma tiptir.

**Boş Liste**

İçi boş köşeli parantez çifti [] boş listeyi gösterir.

---15

**Öğeye erişmek:** Bir listenin elemanlarına köşeli parantez içindeki bir dizin değeri kullanılarak erişilir,

lst = [1, 2, 3] için,

lst[0] İlk öğenin değeri olan 1 e erişir.

Örneğin, print(lst[0]) komutu, lst=[1,2,3] listesindeki ilk öğe olan 1 sayısını yazdırır,

lst[1] ikinci öğenin değeri olan 2 e erişir.

lst[2] üçüncü öğenin değeri olan 1 e erişir.

toplam = lst[0] + lst[1] + lst[2]

üçünün toplamını hesaplar.

**Güncelleme** yaparken, lst[2] = 4 komutu listede indeks-2'deki 3 ü 4 e günceller.

Başta lst = [1, 2, 3] olsun. lst[2] = 4 komutundan sonra lst = [1, 2, 4] olur.

**Araya ekleme**de, dizin-1'e 3 eklemek üzere lst.insert(1,3)komutu kullanılır. Öğe sayısı bir arttığından dizinler de bir artar. Dizin-2 deki eski değer dizin-3 e, dizin-1 deki eski değer, dizin-2 ye kayar. Önceden lst=[1,2,3] iken lst.insert(1,3) komutundan sonra lst=[1,3,2,3] olur.

**Sona eklemede**, lst.append(4) listenin sonuna 4 ekler. Başlangıçta lst=[1,2,3] ise komut sonrası lst=[1,2,3,4] olur.

**Silerken**, dizin-2'deki 4 ü silmek üzere del lst[2] komutu kullanılır. Önceden lst=[1,2,3] iken del lst[2] komutu sonrası lst=[1,2]olur.

---16

Karma ya da dize tipi listelerde de aynı komutlar işler.

Örneğin meyve=[‘muz’, ‘elma’, ‘kiraz’] listesi üzerinde komutları ve listenin son durumunu görelim:

meyve[1]=’armut’ sonrası

meyve==[‘muz’, ‘armut’, ‘kiraz’] olur

del meyve[0] sonrası

meyve==[‘elma’, ‘kiraz’] olur

meyve.insert(1,’üzüm’) sonrası

meyve=[‘muz’, ‘üzüm’, ‘elma’, ‘kiraz’] olur

---17

**Deneyelim**

>>> lst=[10, 20, 30]

>>> lst

???

>>> lst[0]

???

>>> lst[0]=5

>>> lst

???

>>> del lst[2]

>>> lst

???

>>> lst.insert(1,15)

>>> lst

???

>>> lst.append(40)

>>> lst

???

---18

Tuple (Demet)

Bir demet (tuple), liste benzeri, ancak değişmez bir doğrusal veri yapısıdır. Listeden farkı, bir kez tanımlandıktan sonra değiştirilemez. Diğer yönlerden demetler ve listeler temelde aynıdır.

Tuple'ları listelerden ayırt etmek için, tuple'lar aşağıda gösterildiği gibi köşeli parantez yerine parantez içinde gösterilir,

nums = (10, 20, 30)

student = ('John Smith', 48, 'Computer Science', 3.42)

---19

Tuple ve listelerin diğer bir farkı, tek elemanlı tuple’ları gösterirken elemandan sonra bir virgül konulmasıdır. Tek elemanın sonuna virgül konmazsa parantez sanki aritmetik ifade gibi işlem gördüğünden demet oluşturmaz. Çok elemanlı tuple veya listede de son elemandan sonra virgül varsa hata oluşmaz.

>>> (1,)

(1,)

>>> (1,2)

(1,2)

>>> (1,2,)

(1,2)

Aşağıdaki örnekler aritmetik ifade gibi algılandığından tuple oluşturmuyor.

>>> (1)

1

>>> (1+2)

3

---20

Içi boş parantez çifti “()”, boş bir demet oluşturur.

>>> bosT= ()

>>> bosT

???

Tuple öğelerine, köşeli parantezler ile listelerde olduğu gibi erişilir,

>>> t = (10, 20, 30)

>>> t[0]

???

Ancak öğeler hiçbir yolla değiştirilemez. Değiştirme çabası hata ile sonuçlanır.

>>> t[2]= 50

???

>>> del t[2]

???

>>> t.insert(1, 15)

???

>>> t.append(40)

???

---21

**Veri Dizileri**

Python'da dizin numarasıyla erişilen doğrusal olarak sıralanmış bir öğe kümesi bir veri dizisi oluşturur.

Listeler, demetler ve karakter dizeleri Python’daki veri dizileridir. Demetler gibi dizeler de değişmez (immutable) dir, dizenin karakterleri değiştirilemezler.

---22

| **İşlem**  | **örnek** | **String** s='hello' w= '!' | **Tuple** s=(1,2,3,4) w=(5,6) | **List**s=[1,2,3,4] w= [5,6] |
| --- | --- | --- | --- | --- |
| **Uzunluk****Length**  | len(s) | 5 | 4 | 4 |
| **Erişim****Select**  | s[0] | 'h' | 1 | 1 |
| **Kesit****Slice**  | s[1:4]s[1:] | 'ell''ello' | (2,3,4)(2,3,4) | [2,3,4][2,3,4] |
| **Sayım****Count**  | s.count('e')s.count(4)  | 1Hata verir | 01 | 01 |
| **Dizin****Index**  | s.index('e') s.index(3)  | 1-- | --2 | --2 |
| **üyelik****Membership**  | 'h' in s | True | False | False |
| **Ulama****Concatenation**  | s+w | 'hello!' | (1,2,3,4,5,6) | [1,2,3,4,5,6] |
| **Endüşük değer****Minimum Value** | min(s) | 'e' | 1 | 1 |
| **Enyüksek değer****Maximum Value** | max(s) | 'o' | 4 | 4 |
| **Toplam****Sum**  | sum(s)  | Hata verirerror | 10 | 10 |

---23

**Deneyelim**

>>> s = 'coconut'

>>> s[4:7]

???

>>> s.count('o')

???

>>> s.index('o')

???

>>> s+' juice'

???

>> s = (10, 30, 20, 10)

>>> s[1:3]

???

>>> s.count(10)

???

>>> s.index(10)

???

>>> s.index(15)

???

>>> s+(40, 50)

???

>>> s = [10, 30, 20, 10]

>>> s[1:3]

???

>>> s.count(10)

???

>>> s.index(10)

???

>>> s+[40, 50]

???

Aynı tipte olmayan listelerle ulama yapılamaz

>>> [1, 2]+[3, 5]

[1, 2, 3, 5]

>>> ‘1, 2’+’3, 5’

‘1235’

Şunlar hata verir.

>>> [1, 2]+(3, 5)

>>> [1, 2]+”35”

---24

**İç içe listeler**

Listeler ve demetler, diğer diziler de dahil herhangi bir türden öğe içerebilir.

Böylece listeler ve demetler iç içe yerleştirilerek istenilen karmaşık veri yapıları oluşturulabilir.

---25

Örnek:

Sınfnotları=[[85, 91, 89],

[78, 81, 86],

[62, 75, 77],

[50, 60, 70] ]

Bu liste dört öğrencilik sınıfta her öğrencinin üç sınav notunu gösteriyor.

Sınıfnotları[0] yazınca [85, 91, 89] olan ilk öğrencinin not listesine,

Sınıfnotları[1] yazınca [78, 81, 86] olan ikinci öğrencinin not listesine erişilir.

İkinci öğrencinin birinci sınav notuna erişmek için

Notlar= Sınıfnotları[1]

Not=Notlar[0]

yazabiliriz. Ya da

Sınıfnotları[1][0] yazmak yeterlidir.

---26

İlk sınavın ortalamasını hesaplamak için, her öğrencinin not listesinin birinci sınavını toplayacak bir while döngüsü oluşturulabilir:

Sınıfnotları = [ [85, 91, 88],

 [78, 81, 87],

 [62, 75, 79],

 [79, 75, 74] ]

toplam = 0

k = 0

while k < len(class\_grades):

 toplam = toplam + Sınıfnotları[k][0]

 k = k + 1

sınavortalaması1 = ( sum

/ len(Sınıfnotları) )

---27

Sınıftaki her öğrencinin sınav ortalamasını Sınavortalaması adında yeni bir listeye yazmak için:

Sınavortalaması = []

k = 0

while k < len(Sınıfnotları):

 avg = (Sınıfnotları[k][0] + \
 Sınıfnotları [k][1] + \
 Sınıfnotları [k][2]) / 3

  Sınavortalaması.append(avg)

 k = k + 1

**Deneyelim**

Python Shell'den aşağıdakini girin ve sonuçları gözlemleyin.

>>> lst = [[1, 2 ,3], [4, 5, 6], [7, 8, 9]]

>>> lst[0]

???

>>> lst[0][1]

???

>>> for k in lst:

 print(k)

???

>>> for k in lst[0]:

 print(k)

???

Listeler ve demetler, karmaşık veri yapıları oluşturmak için iç içe yerleştirilebilir.

---29

**Python'da Listeler (veri dizileri) Üzerinde Yineleme**

Python'da “for” ifadesi, listeler (ve diğer diziler) üzerinde yineleme yapmak için uygun bir yol sağlar.

---30

**For komutları**

For deyimi, belirli bir öğe dizisindeki her öğe için bir kez yineleme sağlayan bir yinelemeli kontrol deyimidir.

For döngüleri, “sayısı belirli” döngüler oluşturmak için kullanılır.

| For deyimi | Kullanma Örneği |
| --- | --- |
| For k in *dizi*: *suite* | nums=[10, 20, 30, 40] for k in nums : print(k) |

For komutundaki k değişkeni, **döngü değişkeni** olarak adlandırılır. İşlenen listede dört öğe olduğundan, for döngüsü bloku k nın değişik değerleriyle tam dört kez döner.

---32

Liste geçişinde for ile while döngüsü kullanımını karşılaştırmak üzere, aşağıdaki aynı geçişi sağlayan while döngüsünü verelim:

i = 0

while i<len(nums):

 print(nums[i])

 i = i + 1

while döngüsünde, döngü değişkeni i listeyi dizin aracılığıyla geçtiğinden, i ye 0 atanarak başlatılır ve döngü boyunca her seferinde bir artırılır. For döngüsünde ise listeyi geçerken döngü değişkeni k, doğrudan listedeki değerleri alır. For ifadesi, dizeler de dahil dizi tiplerinin tümünün üzerinden geçmek (traverse) için kullanılabilir.

---34

**Deneyelim**

Python Shell'den aşağıdakini girin ve sonuçları gözlemleyin.

>>> for k in range(0, 11):

 print(k)

???

>>> for k in range[0, 11]:

 print(k)

???

>>> for k in ['Apple', 'Banana', 'Pear']:

 print(k)

???

>>> for k in 'Apple':

 print(k)

???

---35

**Python’daki aralık (built-in range) işlevi**

Python, bir for döngüsünün yineleyebileceği bir tamsayı dizisi oluşturmak için kullanılabilecek yerleşik bir aralık (**range**) işlevi sağlar.

toplam = 0

for k in range(2, 11):

toplam = toplam + k

—36

Oluşturulan dizideki değerler, bitiş değerine kadar ancak başlangıç değerini içerir.

Örneğin, range(2, 11) yazarak [2, 3, 4, 5, 6, 7, 8, 9, 10] dizisi üretilir. Böylece, bu for döngüsü 2 den 10 a kadarki tamsayı değerlerini toplar.

**range** işlevi, uzun tamsayı dizileri gerektiğinde kullanışlıdır.

—37

(Aslında, range gerçek bir tamsayı dizisi oluşturmaz. Dizinin sonraki her öğesini üretebilen bir üreteç işlevi oluşturur. Böylece, özellikle uzun listeler için bellek tasarrufu sağlanır. Python kabuğuna range(0, 9) yazmak da beklendiği gibi bir liste oluşturmaz - yalnızca range çağrısını "tekrarlar".)

—38

Varsayılan olarak, aralık işlevi bir dizi ardışık tamsayı oluşturur. Bir “adım” değeri sağlanabilir,

range(0, 11, 2) → [0, 2, 4, 6, 8, 10], (2şer adımlı)

range(5, 0, -1) → [5, 4, 3, 2, 1]. (-1 ile geriye adımlar).

—39

**Liste Öğeleri ve Liste Dizin Değerleriyle Yineleme**

For döngüsü bir listenin üzerinden geçerken döngü değişkeni ya doğrudan öğeleri temsil edebilir, ya da öğelere erişecek dizini temsil edebilir.

—40

| Doğrudan öğelerle geçiş |
| --- |
| L=[5, 10, 15]toplam=0for k in L: toplam += kprint(toplam) |

| Dizin aracılığı ile geçiş |
| --- |
| L=[5, 10, 15]toplam=0for i in range(0,len(L)): toplam += L(i)print(toplam) |

Doğrudan öğelerle geçiş, döngüde listenin öğeleri kullanılıyor, ama değiştirilmiyor ise daha kullanışlıdır.

Ancak, bazı durumlarda, örneğin geçilen öğelerin güncellenmesi amaçlanmış ise öğeler üzerinden geçmek işleri zorlaştırır. Aşağıda, liste öğelerini dizin kadar artıran programa bakalım. Örneğin L=[5,10,15] dizini sonunda [5,11,17] olacak.

| Doğrudan öğelerle geçiş |
| --- |
| L=[5, 10, 15]for k in L: L[L.index(k)] = k+L.index(k)print(L) |

| Dizin aracılığı ile geçiş |
| --- |
| L=[5, 10, 15]for i in range(0,len(L)): L[i]= L[i]+iprint(L) |

Görüldüğü gibi bu kez döngüde liste üzerinden dizin aracılığıyla geçiş daha kullanışlı oldu.

Görevin tipine göre doğru tür liste geçişi kullanırsak programımız daha anlaşılır olur.

Bu iki tür geçişi while kullanarak yapmak da mümkündür, ancak öğelere for ile hiç dizin kullanmadan erişim mümkün iken while ile ancak dizin aracılığıyla erişir.

L=[11,12,13]

i=0

k=L[0]

while k in L:

 print(k)

 i +=1

 if i==len(L): break

 k = L[i]

— 41

Örneğin

Sınıfnotları = [ [85, 91, 88],

 [78, 81, 87],

 [62, 75, 79],

 [79, 75, 74] ]

verilirse birinci öğrencinin not ortalamasını hesaplamak üzere şöyle yazarız.

toplam=0

for k in Sınıfnotları[0]:

 toplam=toplam+k

print(toplam/len(Sınıfnotları[0]))

Aynı veri ile birinci sınav not ortalamasını da “for” döngüsünde şöyle hesaplarız.

toplam=0

for k in Sınıfnotları:

 toplam=toplam+k[0]

print(toplam/len(Sınıfnotları))

“for” döngüsüyle dizin (index) aracılığıyla geçiş için range() fonksiyonu ile dizin üretmemiz gerekir. range(nf) fonksiyonu 0 1 2 … nf-1 sayılarını üretir. L listesinde kullanılan tüm dizinleri üretmek için range(len(L)) yazarız. Böylece ilk öğrencinin not ortalamasını hesaplamak için şu kodu yazarız:

toplam=0

for i in range(len(Sınıfnotları[0])):

 toplam = toplam + Sınıfnotları[0][i]

print( toplam/len(Sınıfnotları[0]))

Dizin aracılığıyla ilk sınav ortalamasını ise şöyle kodlarız.

toplam=0

for i in range(len(Sınıfnotları)):

 toplam = toplam + Sınıfnotları[i][0]

print( toplam/len(Sınıfnotları))

—43

range() fonksiyonu iki ya da üç argümanla da çağırılabilir. range(ni, nf, ns) üretilecek sayıların ilk değerini (ni), son değerini (nf-1) ve değerler arasındaki aralığı (ns) belirler. Örneğin range(2,10,3) ile { 2, 5, 8 } sayıları üretilir.

range() sayıları liste olarak üretmek yerine sayıları üretecek bir fonksiyon oluşturur. Bu nedenle aşağıdaki gibi komut olarak yazıldığında üretilen sayıları göremeyiz.

>>> range(2,8,3)

range(2,8,3)

Sayıları listeye çevirmek üzere list() fonksiyonu kullanılır.

>>> list(range(2,10,3))

[2, 5, 8]

—44

**Deneyelim**

>>> nums = [10, 20, 30]

???

>>> for k in range(len(nums)):

 print(nums[k])

???

>>> for k in range(len(nums)-1, -1, -1):

 print(k)

???

—45

**Deneyelim**

>>> k=0

>>> sum = 0

>>> nums = range(100)

>>> while k < len(nums) and sum < 100:

... sum = sum + nums[k]

... k = k + 1

>>> print('The first', k, 'integers sum to 100 or greater')

—46

While Döngüsü ve Listeler (Diziler)

Bir dizinin öğelerinin belirli bir koşul doğru olana kadar geçilmesi gereken durumlar vardır. Bu gibi durumlarda while döngüsü for dan daha kullanışlıdır.

—47

sayılar=[10, 20, 30]

k=0

aranan=40

Bulundu=False

while k < len(sayılar) and not Bulundu:

 if sayılar[k]==aranan:

 Bulundu=True

 else:

 k=k+1

if Bulundu :

 print("sayı bulundu")

else:

 print("sayı bulunamadı")

Döngü, öğe bulunana veya listenin tümü geçilene kadar sürer. Son if ifadesi, 'sayı bulundu' veya 'sayı bulunamadı' görüntüleyerek, döngüyü sonlandırmanın iki olasılığından hangisinin gerçekleştiğini belirler.

—48

**Deneyelim**

k=0

Sum=0

maxsum=100

nums=range(100)

while k<len(nums) and Sum<maxsum:

 Sum=Sum + k

 k=k+1

print('The first', k,

 'integers sum to', maxsum,

 'or greater')

???

k=0

Sum=0

maxsum=150

maxnum=100

while k<maxnum and Sum<maxsum:

 Sum=Sum + k

 k=k+1

print('The first', k,

 'integers sum to', maxsum,

 'or greater')

???

—49

**Python Listelerine özel ayrıntılar**

—50

1. **Listelerin Atanması ve Kopyalanması**

Listelerin Python'da tutulma biçimi nedeniyle, bir değişkene

liste2 = liste1

gibi bir liste değişkeni atandığında, her iki değişken, bellekteki aynı liste örneğini temsil eder.

—51

Değişkenlerin tek ve aynı listeyi göstermesinin doğal sonucu olarak birinin içeriğindeki değişiklik diğerine de yansır.

—52

>>> list1 = [10, 20, 30, 40]

>>> list2 = list1

>>> list1[0] = 5

>>> list1

[5, 20, 30, 40] list1 de değişiklik oldu

>>> list2

[5, 20, 30, 40] list1 değişince list2 de değişti.

Bu özellik, değiştirilemedikleri için dizeler ve demetler için geçerli değildir.

—53

Listenin aynısından ayrı bir kopya üretmek için list() kullanılır.

>>> list1 = [10, 20, 30, 40]

>>> list3 = list(list1)

—54

>>> list1[0] = 10

>>> list1

[10, 20, 30, 40] list1 de değişiklik oldu

>>> list3

[5, 20, 30, 40] list1 değişince list3 değişmedi.

—55

Alt listeli listeleri kopyalarken, derin kopyalama adı verilen başka bir kopyalama yöntemine ihtiyaç duyulabilir.

**ListB=ListA**

listenin yalnızca adın kopyalar. Bu yüzden bir listedeki değişiklikler diğer listeye yansır,

copy kitaplığını koda aldıktan sonra:

**ListB=copy.copy(ListA)**

Ile ListB yalnızca öğelerin kopyasını sığ biçimde oluşturur. ListA içinde iç içe listeler olması durumunda, bir alt listenin öğelerindeki değişiklik diğer kopyaya yansır.

**ListB=copy.deepcopy(ListA**

listelerin tümüyle birbirinden bağımsız olduğu bir derin kopya oluşturur.

56

**Deneyelim**

>>> list1 = ['red',

 'blue', 'green']

>>> list2 = list1

>>> list1[2] = 'yellow'

>>> list1

???

>>> list2

???

>>> import copy

>>> list1 = [[10,20],[30,40]]

>>> list2 = copy.copy(list1)

>>> list1[0]=[15,25]

>>> list1[1][0] = 35

>>> list1

???

>>> list2

???

>>> list1 = ['red', 'blue', 'green']

>>> list2 = list(list1)

>>> list1[2] = 'yellow'

>>> list1

???

>>> list2

???

>>> import copy

>>> list1 = [[10,20],[30,40]]

>>> list2 = copy.deepcopy(list1)

>>> list1[0]=[15,25]

>>> list1[1][0] = 35

>>> list1

???

>>> list2

???

—57

1. **Kavramalı Liste Oluşturma**

Aralık (range) işlevi, sabit artışlarla tamsayı dizilerinin oluşturulmasına izin verir.

Python'daki kavramalı (comprehension) liste oluşturma yolu, çeşitli diziler oluşturmakta kullanılabilir.

—58

Örnekler ve oluşan listeler

>>> [x\*\*2 for x in [1, 2, 3]]

[1, 4, 9]

>>> [x\*\*2 for x in range(5)]

[0, 1, 4, 9, 16]

>>> nums =[-1,1,-2,2,-3,3,-4,4]

>>> [x for x in nums if x >= 0]

[1, 2, 3, 4]

>>> [ord(ch) for ch in 'Hello']

[72,101,108,108,111]

>>> vowels = ('a', 'e', 'i', 'o', 'u')

>>> w= 'Hello'

>>> [ch for ch in w if ch in vowels]

['e', 'o']

—59

Python'daki kavramalı liste oluşturma, aralık işlevi tarafından oluşturulabilecek dizilerden daha çeşitli diziler oluşturmanın özlü bir yolunu sağlar.

>>> temperatures = [88, 94, 97, 89, 101,

98, 102, 95, 100]

>>> [t for t in temperatures if t >= 100]

???

>>> [(t - 32) \* 5/9 for t in temperatures]

???