sözlükler

- koleksiyonlar: ardışık ve eşleştirme
- ardışık: dizgi, liste, tuple (çok öğeliler)
- ardışık: erişim indisle yapılır
- eşleştirme: sözlük
- eşleştirme: anahtar/değer
- PB: 6593

merhaba

→ türkçe - ispanyolca sözlük tasarımı

```
>>> tr2sp = {}
>>> tr2sp['bir'] = 'uno'
>>> tr2sp['iki'] = 'dos'
```

→ {}: boş sözlük

→ anahtar-değer çiftleri

devam

→ böyle yazdır

```
>>> print tr2sp
{'iki': 'dos', 'bir': 'uno'}
```

→ istersen böyle ata, çiftler virgülle ayrılıyor

```
>>> tr2sp = {'bir':'uno', 'iki':'dos', 'uc':'tres'}
```

→ eleman eleman böyle eriş

```
>>> print tr2sp['iki']
dos
```

sözlük işlemleri

→ del ifadesi anahtar-değer çiftini siler

```
>>> stok = {'elma': 430, 'muz': 312, 'portakal': 525, 'erik':
>>> print stok
{'elma': 430, 'erik': 217, 'portakal': 525, 'muz': 312}
>>>
>>> del stok['erik']
>>> print stok
{'elma': 430, 'portakal': 525, 'muz': 312}
```

→ değer güncellerken

```
>>> stok['erik'] = 0
>>> print stok
{'elma': 430, 'erik': 0, 'portakal': 525, 'muz': 312}
```

→ sözlükte kaç çift var

```
>>> len(stok)
4
```

sözlük metodları

→ tüm anahtarlar, değerler ve çiftler (tuple'ı)

```
>>> tr2sp.keys()
['iki', 'bir', 'uc']
>>>
>>> tr2sp.values()
['dos', 'uno', 'tres']
>>>
>>> tr2sp.items()
[('iki', 'dos'), ('bir', 'uno'), ('uc', 'tres')]
```

nesne yönelimlilik paradigması

- → dizgi, listelerde olduğu gibi sözlük metodlarında da
- → nokta-. gösterilimi
- → noktanın sağına metod ismi,
- → noktanın solundaki değişkene ilgili metod uygulanır
- → parantez içerisinde bir şeylerin yazılmamış olması, parametresiz olduğunu gösterir
- → metod çağrısı -> **çağırma (invocation)**
- → tr2sp.keys(): tr2sp nesnesinde keys metodunu çağırdık
- → tasarıma ait: bu metodun ilk argümanı, nesnenin kendisidir (self)

```
1  def keys(self):
2      ...
```

sözlük metodları

→ böyle bir anahtar var mı?

```
>>> tr2sp.has_key('bir')
True
>>> tr2sp.has_key('deux')
False
```

→ bunu doğrudan yapar ve olmayan anahtar için değer sorarsanız

```
>>> tr2sp['dog']
Traceback (most recent call last):
  File "<input>", line 1, in <module>
KeyError: 'dog'
```

rumuz ve kopyalama X deep-shallow copy

→ shallow copy X deep copy

```
>>> karsitlar = {'up': 'down', 'right': 'wrong', 'true': 'fals
>>> rumuz = karsitlar
>>> kopya = karsitlar.copy()
```

→ shallow (rumuz) olandaki değişiklik orjinali de etkiler

```
>>> rumuz['right'] = 'left'
>>> karsitlar['right']
'left'
```

→ deep'deki değişiklik ise etkilemez

```
>>> kopya['right'] = 'privilege'
>>> karsitlar['right']
'left'
```

dağınık matrisler (sparse matrix)

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \end{bmatrix}$$

→ matris için liste gösterilimi bol sıfır içerir

```
matris = [[0, 0, 0, 1, 0],
          [0, 0, 0, 0, 0],
          [0, 2, 0, 0, 0],
          [0, 0, 0, 0, 0],
          [0, 0, 0, 3, 0]]
```

→ alternatif: sözlük kullanımı

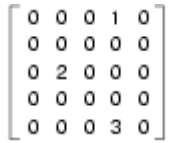
```
>>> matris = {(0, 3): 1, (2, 1): 2, (4, 3): 3}
```

→ kaç elemanı var? anahtarları nedir?

→ nasıl erişiriz

```
>>> matris[(0, 3)]
1
```

dağınık matrisler (sparse matrix)



→ matrisin boş olan bölgesine nasıl erişeceğiz?

→ sözlüğe eklenmemiş olanlara erişim

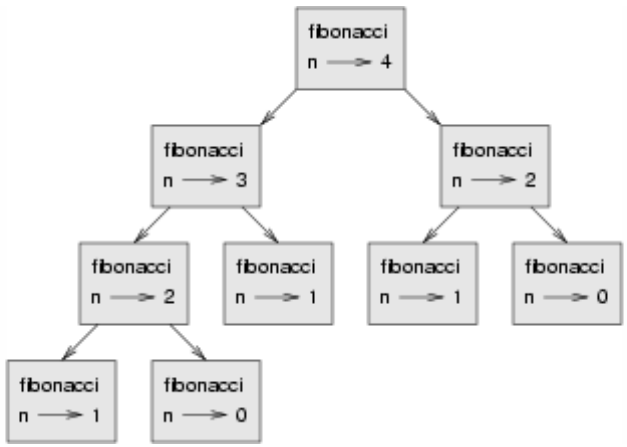
```
>>> matris[(1, 3)]
Traceback (most recent call last):
  File "<input>", line 1, in <module>
KeyError: (1, 3)
```

→ doğru yöntem get ve varsayılan değer yapısı

```
>>> matris.get((0, 3), 0)
1
>>> matris.get((1, 3), 0)
0
```

bellekleme (hint)

- → daha önce özyineli veya düz tasarladığınız fibonacci işlevi
- → büyük sayılarla sorun çıkartır
- → ör: fibonacci(20) anında, fibonacci(39) yaklaşık 1 sn,
- → fibonacci(40) ise neredeyse sonlanamamakta



- → bunun sebebi: tekrarlayan fazlalık çağrılardır
- → ör. fibonacci(4) için fibonacci(0) 2 kez,
- → fibonacci(1) 3 kez çağrılır/hesaplanır

bellekleme (hint)

→ daha önce hesaplananları hafızaya alalım

→ gerek duyulduğunda buradan verelim -> hint (ipucu)

```
onceki = {0: 0, 1: 1}

def fibonacci(n):
    if onceki.has_key(n):
        return onceki[n]
    else:
        yeni_deger = fibonacci(n-1) + fibonacci(n-2)
        onceki[n] = yeni_deger
        return yeni_deger
```

bellekleme

→ önce, başlangıç durumuyla onceki sözlüğünü ilkle

→ eğer sözlükte var olan isteniyorsa gönder

→ yoksa hesapla, sözlüğe ekle ve gönder

→ böylelikle göz açıp-kapayıncaya kadar kısa sürede hesapla

```
>>> from d12_fib import *
>>> fibonacci(100)
354224848179261915075L
```

→ L: long sayı anlamında

uzun sayılar

→ herhangi bir büyüklükteki sayıyı tutmak için uzun sayı türü-> long

```
>>> type(1L)
<type 'long'>
>>> long(7)
7L
>>> long(3.9)
3L
>>> long('59')
59L
```

harfleri saymak

→ daha öncede bunu yaptık (7. bölüm)

→ histogram dedik

→ VLC sıkıştırma tekniği

→ sözlük daha şık bir çözüm sunar

```
>>> harf_sayilari = {}
>>> for harf in "Mississippi":
...     harf_sayilari[harf] = harf_sayilari.get(harf, 0) + 1
...
>>> harf_sayilari
{'i': 4, 'p': 2, 's': 4, 'M': 1}
```

→ alfabetik sıraya göre sıralamakta mümkün

```
>>> harfler = harf_sayilari.items()
>>> harfler.sort()
>>> print harfler
[('M', 1), ('i', 4), ('p', 2), ('s', 4)]
```

sıra sizde

→ alıştırma 1

→ çıktı nedir?

```
>>> d = {'apples': 15, 'bananas': 35, 'grapes': 12}
>>> d['banana']
```

?	?	?

```
>>> d['oranges'] = 20
>>> len(d)
```

?	?	?

```
>>> d.has_key('grapes')
```

?	?	?

```
>>> d['pears']
```

?	?	?

```
>>> d.get('pears', 0)
```

?	?	?

```
>>> fruits = d.keys()
>>> fruits.sort()
>>> print fruits
```

?	?	?

```
>>> del d['apples']
>>> d.has_key('apples')
```

?	?	?

devam

→ aşağıdaki doctestten geçecek işlevi yazın

```
1  def add_fruit(inventory, fruit, quantity=0):
2      """
3      Adds quantity of fruit to inventory.
4
5        >>> new_inventory = {}
6        >>> add_fruit(new_inventory, 'strawberries', 10)
7        >>> new_inventory.has_key('strawberries')
8        True
9        >>> new_inventory['strawberries']
10       10
11       >>> add_fruit(new_inventory, 'strawberries', 25)
12       >>> new_inventory['strawberries']
13     """
```

devam

→ alıştırma 3: alice_words.py isminde bir program yazın, programınız alice_in_wonderland.txt dosyasındaki tüm kelimelerin alfabetik listesini her bir kelimenin kaç kere yer aldığıyla birlikte alice_words.txt adındaki metin dosyasına yazsın. Çıktınızın ilk 10 satırı aşağıdakine benzeyecektir:

```
Kelime                  Adet
======================
a                       631
a-piece                 1
abide                   1
able                    1
about                   94
above                   3
absence                 1
absurd                  2
```

→ alice kelimesi kitapta kaç kere yer almaktadır?

→ "Alice in Wonderland (Alice Harikalar Diyarında)"'ki en uzun kelime hangisidir? Kelime kaç karakter içermektedir??