İçindekiler

Yazar: Emrecan Güldoğan

Python Sözdizimi

Bu doküman, temel Python sözdizimini açıklamaktadır. Kod örnekleri, diğer metinlerden ayırt edilebilmesi için özel olarak biçimlendirilmiştir.

1. Temel İşlemler ve Veri Tipleri

Python'da sayılar ve metinler gibi temel veri tipleri üzerinde işlemler yapabilirsiniz.

- **Aritmetik İşlemler:** Sayılar üzerinde toplama, çarpma gibi temel matematiksel işlemler. İşlem önceliği parantezlerle değiştirilebilir.

```
>>> 2 + 3 * 6
>>> (2 + 3) * 6
```

```
# Output
20
30
```

- **String Birleştirme ve Çoğaltma:** Metin (string) ifadeleri birleştirmek veya tekrarlamak için kullanılır.

```
>>> 'Alice' 'Bob'
>>> 'Alice' * 5
```

```
# Output
'AliceBob'
'AliceAliceAliceAliceAlice'
```

- **Değişken Tanımlama:** Değerleri saklamak için değişkenler kullanılır. Atama operatörü (`=`) kullanılır.

```
>>> spam = 'Hello'
a = 1 # initialization
```

- **Yorum Satırları:** Kodun okunurluğunu artırmak için açıklama ekleme. Tek satır için `#`, çoklu satır için `'''` veya `"""`.

```
# Bu tek satırlık bir yorum
""" Bu da çok
satırlı bir yorum """
```

- **Ekrana Yazdırma (`print`) ve Girdi Alma (`input`):** Kullanıcıyla etkileşim kurma.

```
>>> print('Hello world!')
>>>
>>> a = 1
>>> print('Hello world!', a)
```

```
# Output
Hello world!
Hello world! 1
```

```
>>> print('What is your name?') # ask for their name
>>>
>>> myName = input() # Kullanıcıdan girdi alır
>>> print('It is good to meet you, {}'.format(myName))
```

```
# Output
What is your name?
Ali
It is good to meet you, Al
```

- **Uzunluk Bulma (`len`):** Bir veri yapısının (string, liste vb.) eleman sayısını bulma.

```
>>> len('hello')
```

```
# Output
5
```

- **Tip Dönüşümleri:** Bir veri tipini başka bir veri tipine dönüştürmek için kullanılır. Örnek: `str()` .

```
>>> str(29)
```

```
# Output
29
```

2. Karşılaştırma ve Mantıksal Operatörler

- **Karşılaştırma Operatörleri:** İki değeri karşılaştırma (eşitlik, büyüklük vb.).

```
Operator Meaning
==   Equal to
!=   Not equal to
<    Less than
>    Greater Than
<=   Less than or Equal to
>=   Greater than or Equal to
```

```
>>> 'hello' == 'Hello'
>>> 42 == 42.0
>>> 'dog' != 'cat'
```

```
# Output
False
True
True
```

- **Kimlik Operatörleri (`is` , `is not`):** İki değişkenin aynı nesneye mi referans verdiğini kontrol etme.

```
>>> True is True
>>> True is not False
```

```
# Output
True
True
```

- **Mantıksal Operatörler (`and` , `or` , `not`):** Birden fazla koşulu birleştirme.

```
>>> (4 < 5) and (5 < 6)
>>> (1 == 2) or (2 == 2)
>>> 2 + 2 == 4 and not 2 + 2 == 5 and 2 * 2 == 2 + 2
```

```
# Output
True
True
True
```

3. Akış Kontrolü

Kodun hangi sırayla çalışacağını belirler.

- **Koşullu İfadeler (`if` , `elif` , `else`):** Belirli koşullara göre farklı kod bloklarını çalıştırma.

```
name = 'Bob'
age = 30
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
else: # Yukarıdaki koşulların hiçbiri doğru değilse
    print('You are neither Alice nor a little kid.')
```

- **`while` Döngüsü:** Bir koşul doğru olduğu sürece belirli bir kod bloğunu tekrarlamak için kullanılır. Döngünün içinde koşulu değiştirecek bir ifade (genellikle sayaç artırma/azaltma veya bir durum değişikliği) bulunmalıdır, aksi takdirde sonsuz döngü oluşabilir.

```
spam = 0
while spam < 5:
    print('Hello, world.')
    spam = spam + 1 # Döngüyü sonlandırmak için koşulu değiştiren ifade
```

```
# Output
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
```

- **`break` ve `continue` İfadeleri:** Döngü akışını değiştirme. `break` döngüyü tamamen sonlandırır, `continue` mevcut iterasyonu atlar.

```
while True:
    print('Who are you?')
    name = input()
    if name != 'Joe':
        continue # Koşul sağlanmazsa döngünün başına dön
    print('Hello, Joe. What is the password? (It is a fish.)')
    password = input()
    if password == 'swordfish': # Koşul sağlanmazsa döngüye devam et
        break
print('Access granted.')
```

```
# Input/Output
Who are you?
>>> Ali
Who are you?
>>> Joe
Hello, Joe. What is the password? (It is a fish.)
>>> cat
Who are you?
>>> Joe
Hello, Joe. What is the password? (It is a fish.)
>>> swordfish
Access granted.
```

- `for` **Döngüsü:** Bir dizi (liste, aralık vb.) üzerinde yineleme yapma. `range()` belirli bir sayı aralığı oluşturur.

```
print('My name is')
for i in range(5): # 0'dan 4'e kadar döner
    print('Jimmy Five Times ({})'.format(str(i)))
```

```
# Output
My name is
Jimmy Five Times (0)
Jimmy Five Times (1)
Jimmy Five Times (2)
Jimmy Five Times (3)
Jimmy Five Times (4)
```

```
for i in range(0, 10, 2): # 0'dan 10'a (10 hariç) 2'şer artarak döner
    print(i)
```

```
# Output
0
2
4
6
8
```

`for` döngüsünün `else` bloğu, döngü `break` ile sonlanmadığında çalışır.

```
for i in [1, 2, 3, 4, 5]:
    if i == 3:
        break
```

```
else: # Döngü break ile sonlanmazsa çalışır
    print("only executed when no item of the list is equal to 3")
```

```
# Bu örnekte 3 listede olduğu için break çalışır ve else bloğu çalışmaz.
# Eğer liste [1, 2, 4, 5] olsaydı, else bloğu çalışırdı.
```

- **Modül İçe Aktarma (`import`):** Başka Python dosyalarındaki veya kütüphanelerdeki kodları kullanma.

```
import random # random modülünü içe aktar
for i in range(3):
    print(random.randint(1, 10)) # random modülünden fonksiyon kullan
```

```
# Output (Örnek)
7
3
9
```

```
from random import * # random modülündeki her şeyi içe aktar
# Artık random. ön eki olmadan randint() gibi fonksiyonları kullanabilirsiniz.
```

```
import sys # sys modülünü içe aktar
while True:
    print('Type exit to exit.')
    response = input()
    if response == 'exit':
        sys.exit() # sys modülünden fonksiyon kullan (programı sonlandırır)
    print('You typed {}'.format(response))
```

```
# Input/Output (Örnek)
Type exit to exit.
>>> hello
You typed hello
Type exit to exit.
>>> exit
## Program sonlanır.
```

4. Fonksiyonlar

Belirli bir görevi yerine getiren yeniden kullanılabilir kod blokları.

- **Fonksiyon Tanımlama (`def`):** Fonksiyon oluşturma.

```
>>> def hello(name): # 'hello' adında, 'name' argümanı alan fonksiyon
>>> print('Hello {}'.format(name))
>>> hello('Alice')
>>> hello('Bob')
```

```
# Output
Hello Alice
Hello Bob
```

```
import random
def getAnswer(answerNumber): # Değer döndüren fonksiyon
    if answerNumber == 1:
        return 'It is certain' # Değer döndürme
    elif answerNumber == 2:
        return 'It is decidedly so'
    elif answerNumber == 3:
        return 'Yes'
    elif answerNumber == 4:
        return 'Reply hazy try again'
    elif answerNumber == 5:
        return 'Ask again later'
    elif answerNumber == 6:
        return 'Concentrate and ask again'
    elif answerNumber == 7:
        return 'My reply is no'
    elif answerNumber == 8:
        return 'Outlook not so good'
    elif answerNumber == 9:
        return 'Very doubtful'
r = random.randint(1, 9) # 1-9 arasında rastgele bir sayı al
fortune = getAnswer(r) # Fonksiyonu çağır ve değeri değişkene ata
print(fortune) # Değeri yazdır
```

```
# Output (Örnek)
Yes
```

- `print()` **Fonksiyon Argümanları:** Çıktı formatını ayarlama (`end` , `sep`).

```
>>> print('Hello', end='') # Sonuna varsayılan yeni satır yerine boşluk koy
>>> print('World') # Çıktı: HelloWorld
```

```
# Output
HelloWorld
```

```
>>> print('cats', 'dogs', 'mice') # Varsayılan ayraç boşluk
```

```
# Output
cats dogs mice
```

```
>>> print('cats', 'dogs', 'mice', sep=',') # Öğeleri virgülle ayır
```

```
# Output
cats,dogs,mice
```

- `global` **Anahtar Kelimesi:** Fonksiyon içinde global bir değişkeni değiştirme.

```
def spam():
    global eggs # Global 'eggs' değişkenini kullanacağını belirt
    eggs = 'spam' # Global değişkeni değiştir

eggs = 'global' # Global değişkenin başlangıç değeri
spam() # Fonksiyonu çağır
print(eggs) # Global değişkenin yeni değeri
```

```
# Output
spam
```

5. Hata Yönetimi

Program çalışırken oluşabilecek hataları ele alma.

- `try`, `except` **Blokları:** Hata oluşabilecek kodu deneme ve hatayı yakalama.

```
def spam(divideBy):
    try: # Hata oluşabilecek kod bloğu
        return 42 / divideBy
    except ZeroDivisionError as e: # Sıfıra bölme hatasını yakala
        print('Error: Invalid argument: {}'.format(e))
        return None # Hata durumunda None döndür

print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

```
# Output
21.0
3.5
Error: Invalid argument: division by zero
None
42.0
```

- `finally` **Bloğu:** Hata olsa da olmasa da her zaman çalışan kod bloğu.

```
def spam(divideBy):
    try:
        return 42 / divideBy
    except ZeroDivisionError as e:
        print('Error: Invalid argument: {}'.format(e))
        return None
    finally: # Try veya except bloğundan sonra her zaman çalışır
        print("-- division finished --")

print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

```
# Output
-- division finished --
21.0
-- division finished --
3.5
Error: Invalid Argument division by zero
-- division finished --
None
-- division finished --
42.0
```

- `raise` **İfadesi:** Kendi hatalarınızı fırlatma.

```
>>> raise Exception('This is the error message.') # Genel bir hata fırlat
```

```
# Output
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
Exception: This is the error message.
```

Belirli koşullarda hata fırlatmak için kullanılır.

```python
def check_positive(number):
    """
    Sayı pozitif değilse ValueError fırlatır.
    """
    if number <= 0:
        raise ValueError("Sayı pozitif olmalıdır!")
        # Koşul sağlanmazsa hata fırlat
    return f"{number} sayısı pozitif."

# Fonksiyonu farklı değerlerle test etme
test_values = [10, 0, -5]

for value in test_values:
    try:
        result = check_positive(value)
        print(f"Sonuç: {result}")
    except ValueError as e:
        print(f"'{value}' için hata: {e}")
```

```
# Output
Sonuç: 10 sayısı pozitif.
'0' için hata: Sayı pozitif olmalıdır!
'-5' için hata: Sayı pozitif olmalıdır!
```

6. Veri Yapıları: Listeler, Tuple'lar, Sözlükler ve Kümeler

Verileri düzenli bir şekilde saklama.

- **Listeler (`list`):** Sıralı, değiştirilebilir koleksiyonlar. Köşeli parantez `[]` ile tanımlanır. İndeksleme ve dilimleme desteklenir.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam
```

```
# Output
['cat', 'bat', 'rat', 'elephant']
```

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0]     # Çıktı: 'cat' (İlk eleman)
```

```
# Output
'cat'
```

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[-1]    # Çıktı: 'elephant' (Son eleman)
```

```
# Output
'elephant'
```

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0:4]
>>> spam[1:3]
>>> spam[0:-1]
```

```
# Output
['cat', 'bat', 'rat', 'elephant']
['bat', 'rat']
['cat', 'bat', 'rat']
```

Liste kopyalama (slicing ile), eleman ekleme (`append`), eleman değiştirme, listeleri birleştirme (`+`), listeleri çoğaltma (`*`), eleman silme (`del`) işlemleri yapılabilir.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam2 = spam[:] # Liste kopyalama (yeni bir liste oluşturur)
>>> spam2
>>> spam.append('dog') # Orijinal listeye eleman ekleme
>>> spam
>>> spam2 # Kopyalanan liste değişmedi
```

```
# Output
['cat', 'bat', 'rat', 'elephant']
['cat', 'bat', 'rat', 'elephant', 'dog']
['cat', 'bat', 'rat', 'elephant']
```

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[1] = 'aardvark' # 1. indeksteki elemanı değiştirme
>>> spam
>>> spam[2] = spam[1] # 2. indeksteki elemanı 1. indeksteki değer ile değiştir
>>> spam
```

```
# Output
['cat', 'aardvark', 'rat', 'elephant']
['cat', 'aardvark', 'aardvark', 'elephant']
```

```
>>> [1, 2, 3] + ['A', 'B', 'C'] # İki listeyi birleştirme (yeni liste oluşturur)
>>> ['X', 'Y', 'Z'] * 3 # Listeyi 3 kez çoğaltma (yeni liste oluşturur)
>>> spam = [1, 2, 3]
>>> spam = spam + ['A', 'B', 'C'] # Atama ile birleştirme
>>> spam
```

```
# Output
[1, 2, 3, 'A', 'B', 'C']
['X', 'Y', 'Z', 'X', 'Y', 'Z', 'X', 'Y', 'Z']
[1, 2, 3, 'A', 'B', 'C']
```

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> del spam[2] # 2. indeksteki elemanı silme
>>> spam
```

```
# Output
['cat', 'bat', 'elephant']
```

`enumerate()` ile indeks ve elemanlara aynı anda erişilebilir. `zip()` ile birden fazla liste üzerinde paralel olarak yineleme yapılabilir.

```
>>> supplies = ['pens', 'staplers', 'flame-throwers', 'binders']
>>> for i, supply in enumerate(supplies): # İndeks ve elemanları alarak döngü
>>> print('Index {} in supplies is: {}'.format(str(i), supply))
```

```
# Output
Index 0 in supplies is: pens
Index 1 in supplies is: staplers
Index 2 in supplies is: flame-throwers
Index 3 in supplies is: binders
```

```
>>> name = ['Pete', 'John', 'Elizabeth']
>>> age = [6, 23, 44]
>>> for n, a in zip(name, age): # İki listeyi paralel gezme
>>> print('{} is {} years old'.format(n, a))
```

```
# Output
Pete is 6 years old
John is 23 years old
Elizabeth is 44 years old
```

Bir elemanın listede olup olmadığını kontrol etmek için `in` ve `not in` kullanılır.

```
>>> 'howdy' in ['hello', 'hi', 'howdy', 'heyas' ] # Eleman listede mi?
>>> spam = ['hello', 'hi', 'howdy', 'heyas']
>>> 'cat' in spam
>>> 'howdy' not in spam # Eleman listede değil mi?
```

```
# Output
True
False
False
```

```
>>> a = [1, 2, 3, 4]
>>> 5 in a
>>> 2 in a
```

```
# Output
False
True
```

- Çoklu değişken atama (unpacking) ve artırılmış atama operatörleri (+=, *=) kullanılabilir.

```
>>> cat = ['fat', 'orange', 'loud']
>>> size, color, disposition = cat # Liste elemanlarını değişkenlere atama
```

```
# Artık;
# size = 'fat', color = 'orange', disposition = 'loud'
```

```
>>> spam = 'Hello'
>>> spam += ' world!' # Stringe ekleme (spam = spam + ' world!')
>>> spam
>>> bacon = [ 'Zophie' ]
>>> bacon *= 3 # Listeyi çoğaltma ve atama (bacon = bacon * 3)
>>> bacon
```

```
# Output
'Hello world!'
['Zophie', 'Zophie', 'Zophie']
```

- `index()` ile bir elemanın ilk geçtiği indeks bulunur.

- `insert()` ile belirli bir indekse eleman eklenir.
- `sort()` ile listeyi sıralaması değiştirilir.
- `sorted()` fonksiyonu ise listenin sıralanmış bir kopyasını döndürür.

```
>>> spam = ['Zophie', 'Pooka', 'Fat-tail', 'Pooka']
>>> spam.index('Pooka') # 'Pooka' elemanının ilk geçtiği indeksi
```

```
# Output
1
```

```
>>> spam = ['cat', 'dog', 'bat']
>>> spam.insert(1, 'chicken') # 1. indekse 'chicken' ekle
>>> spam
```

```
# Output
['cat', 'chicken', 'dog', 'bat']
```

```
>>> spam = [2, 5, 3.14, 1, -7]
>>> spam.sort() # Listeyi küçükten büyüğe sırala
>>> spam
>>>
>>> spam = ['ants', 'cats', 'dogs', 'badgers', 'elephants']
>>> spam.sort() # Alfabetik olarak sırala
>>> spam
>>>
>>> spam.sort(reverse=True) # Tersten sırala
>>> spam
```

```
# Output
[-7, 1, 2, 3.14, 5]
['ants', 'badgers', 'cats', 'dogs', 'elephants']
['elephants', 'dogs', 'cats', 'badgers', 'ants']
```

```
>>> spam = ['ants', 'cats', 'dogs', 'badgers', 'elephants']
>>> sorted(spam) # Listeyi sıralar ve yeni bir liste döndürür (orijinal değişmez)
```

```
# Output
['ants', 'badgers', 'cats', 'dogs', 'elephants']
```

- **Tuple'lar (`tuple`):** Sıralı ve değiştirilemez eleman koleksiyonlarıdır. Parantez `()` ile tanımlanır. Listelere benzer şekilde indeksleme ve dilimleme yapılabilir, ancak elemanları değiştirilemez veya silinemez. Listeden tuple'a veya tuple'dan listeye dönüşüm yapılabilir.

```
>>> eggs = ('hello', 42, 0.5)
>>> eggs[0] # İndeksleme
>>> eggs[1:3] # Dilimleme
>>> len(eggs) # Uzunluk
```

```
# Output
'hello'
(42, 0.5)
3
```

```
>>> tuple(['cat', 'dog', 5]) # Listeden tuple'a dönüştürme
>>> list(('cat', 'dog', 5)) # Tuple'dan listeye dönüştürme
```

```
# Output
('cat', 'dog', 5)
['cat', 'dog', 5]
```

- **Sözlükler (`dict`):** Anahtar-değer çiftleri halinde veri depolayan değiştirilebilir ve sırasız koleksiyonlardır. Süslü parantez `{}` ile tanımlanır. Anahtarlara göre değerlere erişilir. `keys()` , `values()` , `items()` metotları ile anahtarlara, değerlere veya çiftlere erişilebilir.

```
myCat = {'size': 'fat', 'color': 'gray', 'disposition': 'loud'}
```

```
# 'size' anahtarına karşılık gelen değer 'fat', .....
```

```
spam = {'color': 'red', 'age': 42}
for v in spam.values(): # Sözlüğün değerleri üzerinde döngü
    print(v)
```

```
# Output
red
42
```

```
for k in spam.keys(): # Sözlüğün anahtarları üzerinde döngü
    print(k)
```

```
# Output
color
age
```

```
for i in spam.items(): # Sözlüğün anahtar-değer çiftleri üzerinde döngü
    print(i)
```

```
# Output
('color', 'red')
('age', 42)
```

```
spam = {'color': 'red', 'age': 42}

for k, v in spam.items(): # Anahtar ve değerlere ayrı ayrı erişim
    print('Key: {} Value: {}'.format(k, str(v)))
```

```
# Output

Key: age Value: 42
Key: color Value: red

# Sözlükler sırasız olabilir, çıktı sırası farklılık gösterebilir
```

Bir anahtarın veya değerin sözlükte olup olmadığını kontrol etmek için `in` kullanılır.

```
spam = {'name': 'Zophie', 'age': 7}
>>> 'name' in spam.keys() # 'name' anahtarı var mı?
>>> 'Zophie' in spam.values() # 'Zophie' değeri var mı?
>>> # You can omit the call to keys() when checking for a key
>>> 'color' in spam # Varsayılan olarak anahtarlarda arama yapar
```

```
# Output
True
True
False
```

`get()` metodu, bir anahtarın değeri yoksa varsayılan bir değer döndürmek için kullanılır. `setdefault()` metodu, bir anahtar yoksa eklemek ve değer atamak için kullanılır. Sözlükleri birleştirmek için `**` operatörü kullanılabilir.

```
>>> picnic_items = {'apples': 5, 'cups': 2}
>>> 'I am bringing {} cups.'.format(str(picnic_items.get('cups', 0)))
>>>  # 'cups' var değerini al, yoksa 0
>>> 'I am bringing {} eggs.'.format(str(picnic_items.get('eggs', 0)))
>>>  # 'eggs' yok, varsayılan 0 döner
```

```
# Output
'I am bringing 2 cups.'
'I am bringing 0 eggs.'
```

```
spam = { 'name' : 'Pooka', 'age' : 5}
if 'color' not in spam: # 'color' anahtarı yoksa ekle
    spam['color'] = 'black'
# spam şimdi {'name': 'Pooka', 'age': 5, 'color': 'black'} oldu

spam = { 'name' : 'Pooka', 'age' : 5}
spam.setdefault('color', 'black') # 'color' yoksa ekle
# ve 'black' değerini ata, eklenen değeri döndürür

>>> spam.setdefault('name', 'Zophie') # 'name' anahtarı zaten var
# mevcut değeri döndürür
```

```
# Output
'black'
'Pooka'
```

```
>>> x = {'a': 1, 'b': 2}
>>> y = {'b': 3, 'c': 4}
>>> z = {**x, **y} # Sözlükleri birleştirme (Python 3.5+)
>>> z # Ortak anahtarın ('b') değeri y'den gelir
```

```
# Output
{'a': 1, 'b': 3, 'c': 4}
```

- **Sözlüğün Anahtarlarını Listeye Çevirme:** Sözlüğün anahtarlarından oluşan bir liste elde etmek için `list()` fonksiyonunu kullanın.

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
keys_list = list(my_dict)
print(keys_list)
```

```
# Output
['a', 'b', 'c']
```

- **Sözlüğün Değerlerini Listeye Çevirme:** Sözlüğün değerlerinden oluşan bir liste elde etmek için `values()` metodu ile birlikte `list()` fonksiyonunu kullanın.

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
values_list = list(my_dict.values())
print(values_list)
```

```
# Output
[1, 2, 3]
```

- **Sözlüğün Anahtar-Değer Çiftlerini Listeye Çevirme:** Her biri bir anahtar ve değer içeren tuple'lardan oluşan bir liste elde etmek için `items()` metodu ile birlikte `list()` fonksiyonunu kullanın.

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
items_list = list(my_dict.items())
print(items_list)
```

```
# Output
[('a', 1), ('b', 2), ('c', 3)]
```

- **Kümeler (`set`):** Sırasız ve benzersiz eleman koleksiyonlarıdır. Süslü parantez `{}` veya `set()` fonksiyonu ile tanımlanır. Tekrar eden elemanlar otomatik olarak kaldırılır. İndeksleme desteklenmez.

```
>>> s = {1, 2, 3}
>>> s = set([1, 2, 3]) # Listeden küme oluşturma
```

```
>>> s = {1, 2, 3, 2, 3, 4} # Tekrar eden elemanlar kaldırılır
>>> s
```

```
# Output (Kümeler sırasızdır)
{1, 2, 3, 4}
```

```
>>> s = {1, 2, 3}
```

```
>>> s[0] # Kümeler indekslenemez
```

```
# Output
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object does not support indexing
```

Eleman ekleme (`add` , `update`), eleman silme (`remove` , `discard`) işlemleri yapılabilir. `remove` metodu olmayan elemanı silmeye çalışınca hata verirken, `discard` hata vermez.

```
>>> s = {1, 2, 3}
>>> s.add(4) # Tek eleman ekleme
>>> s
```

```
# Output
{1, 2, 3, 4}
```

```
>>> s = {1, 2, 3}
>>> s.update([2, 3, 4, 5, 6]) # Birden fazla eleman ekleme (iterable kullanarak)
>>> s
```

```
# Output
{1, 2, 3, 4, 5, 6}
```

```
>>> s = {1, 2, 3}
>>> s.remove(3) # Eleman silme
>>> s
```

```
# Output
{1, 2}
```

```
>>> s.remove(3) # Olmayan elemanı silmeye çalışmak hata verir
```

```
# Output
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 3
```

```
>>> s = {1, 2, 3}
>>> s.discard(3) # Eleman silme (olmayan eleman hata vermez)
>>> s
```

```
# Output
{1, 2}
```

```
>>> s.discard(3) # Olmayan elemanı silmeye çalışmak hata vermez
```

```
# Output
# Hata yok
```

Küme işlemleri: birleşim (`union` veya `|`), kesişim (`intersection` veya `&`), fark (`difference` veya `-`), simetrik fark (`symmetric_difference` veya `^`).

```
>>> s1 = {1, 2, 3}
>>> s2 = {3, 4, 5}
>>> s1.union(s2) # veya 's1 | s2' (Birleşim)
```

```
# Output
{1, 2, 3, 4, 5}
```

```
>>> s1 = {1, 2, 3}
>>> s2 = {2, 3, 4}
>>> s3 = {3, 4, 5}
>>> s1.intersection(s2, s3) # veya 's1 & s2 & s3' (Kesişim)
```

```
# Output
{3}
```

```
>>> s1 = {1, 2, 3}
>>> s2 = {2, 3, 4}
>>> s1.difference(s2) # veya 's1 - s2' (Fark: s1'de olup s2'de olmayanlar)
>>> s2.difference(s1) # veya 's2 - s1' (Fark: s2'de olup s1'de olmayanlar)
```

```
# Output
{1}
```

```
{4}
```

```
>>> s1 = {1, 2, 3}
>>> s2 = {2, 3, 4}
>>> s1.symmetric_difference(s2) # veya s1^s2 (Her kümeden sadece birinde olanlar)
```

```
# Output
{1, 4}
```

7. Comprehensions (Anlayıcılar)

Listeler, kümeler ve sözlükler gibi veri yapılarını daha kısa sözdizimi ile oluşturma.

- **Liste Anlayıcı:**

```
>>> a = [1, 3, 5, 7, 9, 11]
>>> [i - 1 for i in a] # a'daki her elemanın 1 eksiğini içeren yeni liste
```

```
# Output
[0, 2, 4, 6, 8, 10]
```

- **Küme Anlayıcı:**

```
>>> b = {"abc", "def"}
>>> {s.upper() for s in b} # b'deki her stringin büyük harf halini içeren küme
```

```
# Output (Kümeler sırasızdır)
{"ABC", "DEF"}
```

- **Sözlük Anlayıcı:**

```
>>> c = {'name': 'Pooka', 'age': 5}
>>> {v: k for k, v in c.items()} # c'nin anahtar, değerlerini ters çeviren sözlük
```

```
# Output (Sözlükler sırasızdır)
{'Pooka': 'name', 5: 'age'}
```

8. String Manipülasyonu

String'ler üzerinde çeşitli işlemler yapma.

- **Kaçış Karakterleri:** String içinde özel karakterleri kullanma.

```
>>> print("Hello there!\nHow are you?\nI\'m doing fine.")
>>> # \n yeni satır, \' tek tırnak
```

```
# Output
Hello there!
How are you?
I'm doing fine.
```

- **Raw String'ler (`r`):** Kaçış karakterlerinin etkisiz hale getirilmesi.

```
>>> print(r'That is Carol\'s cat.') # \ karakteri olduğu gibi yazılır
```

```
# Output
That is Carol\'s cat.
```

- **Çok Satırlı String'ler:** Birden fazla satıra yayılan stringler. `textwrap.dedent` ile girintileri temizleme.

```
print('''Dear Alice,

Eve's cat has been arrested for catnapping, cat burglary, and extortion.

Sincerely,
Bob''')
```

```
# Output
Dear Alice,

Eve's cat has been arrested for catnapping, cat burglary, and extortion.

Sincerely,
Bob
```

```
from textwrap import dedent

def my_function():
```

```
    print(dedent('''
           Dear Alice,

        Eve's cat has been arrested for catnapping, cat burglary, and extortion

        Sincerely,
        Bob
    ''').strip())

my_function()
```

```
# Output
Dear Alice,

Eve's cat has been arrested for catnapping, cat burglary, and extortion

Sincerely,
Bob
```

- **String İndeksleme ve Dilimleme:** String içindeki karakterlere veya bölümlere erişme.

```
>>> spam = 'Hello world!'
>>> spam[0]     # Çıktı: 'H' (İlk karakter)
```

```
# Output
'H'
```

```
>>> spam = 'Hello world!'
>>> spam[0:5]   # Çıktı: 'Hello' (0. indeksten 5. indekse kadar - 5 hariç)
>>> spam[:5]    # Baştan 5. karaktere kadar (0. indeksten başlar)
>>> spam[6:]    # 6. karakterden sona kadar
```

```
# Output
'Hello'
'Hello'
'world!'
```

- **String'de Arama (`in`, `not in`):** Bir alt stringin varlığını kontrol etme.

```
>>> 'Hello' in 'Hello World' # 'Hello', 'Hello World' içinde mi?
>>> 'HELLO' in 'Hello World' # Büyük/küçük harf duyarlı
>>> 'cats' not in 'cats and dogs' # 'cats', 'cats and dogs' içinde değil mi?
```

```
# Output
True
False
False
```

```
>>> a = [1, 2, 3, 4]
>>> 5 in a
>>> 2 in a
```

```
# Output
False
True
```

- **String Metotları:** String'leri dönüştürme, kontrol etme, birleştirme, bölme, hizalama, boşlukları temizleme gibi işlemler.

```
>>> spam = 'Hello world!'
>>> spam = spam.upper()      # Tüm harfleri büyütme
>>> spam
>>> spam = spam.lower()      # Tüm harfleri küçültme
>>> spam
```

```
# Output
'HELLO WORLD!'
'hello world!'
```

```
>>> spam = 'Hello world!'
>>> spam.islower() # Tamamı küçük harf mi?
>>> spam.isupper() # Tamamı büyük harf mi?
>>> 'HELLO'.isupper()
```

```
# Output
False
False
True
```

```
>>> 'Hello world!'.startswith('Hello') # 'Hello' ile başlıyor mu?
>>> 'Hello world!'.endswith('world!') # 'world!' ile bitiyor mu?
```

```
# Output
True
True
```

```
>>> ', '.join(['cats', 'rats', 'bats']) # Listeyi ', ' ile birleştirme
>>> ' '.join(['My', 'name', 'is', 'Simon']) # Listeyi boşluk ile birleştirme
```

```
# Output
'cats, rats, bats'
'My name is Simon'
```

```
>>> 'My name is Simon'.split() # Boşluklardan bölerek liste oluşturma (varsayılan)
>>> 'MyABCnameABCisABCSimon'.split('ABC') # 'ABC' ayraçından bölme
```

```
# Output
['My', 'name', 'is', 'Simon']
['My', 'name', 'is', 'Simon']
```

```
>>> 'Hello'.rjust(10) # Sağa hizalama, toplam genişlik 10 (boşluklarla doldurur)
>>> 'Hello'.rjust(20) # Sağa hizalama, toplam genişlik 20
```

```
# Output
'     Hello'
'               Hello'
```

```
>>> 'Hello'.rjust(20, '*') # Sağa hizalama, boşlukları '*' ile doldurma
>>> 'Hello'.ljust(20, '-') # Sola hizalama, boşlukları '-' ile doldurma
```

```
# Output
'***************Hello'
'Hello---------------'
```

```
>>> 'Hello'.center(20) # Ortaya hizalama, toplam genişlik 20 (boşluklarla doldurur
>>> 'Hello'.center(20, '=') # Ortaya hizalama, boşlukları '=' ile doldurma
```

```
# Output
'      Hello       '
'========Hello========'
```

```
>>> spam = ' Hello World '
>>> spam.strip() # Baştaki ve sondaki boşlukları silme
>>> spam.rstrip() # Sondaki boşlukları silme
>>> spam.lstrip() # Baştaki boşlukları silme (OCR'da görünmüyordu, ekledim)
```

```
# Output
'Hello World'
' Hello World'
'Hello World '
```

9. String Formatlama

Değişkenleri veya ifadeleri string içine yerleştirme.

- `format()` **Metodu:**

```
>>> name = 'John'
>>> age = 20
>>> "Hello I'm {}, my age is {}".format(name, age)
```

```
# Output
"Hello I'm John, my age is 20"
```

- **f-string'ler (Formatted String Literals):** Daha modern ve okunaklı string formatlama yöntemi (Python 3.6+).

```
>>> name = 'Elizabeth'
>>> f'Hello {name}!' # Değişkeni doğrudan string içine yerleştirme
```

```
# Output
'Hello Elizabeth!'
```

```
>>> a = 5
>>> b = 10
>>> f'Five plus ten is {a + b} and not {2 * (a + b)}.'
>>> # İfadeleri doğrudan string içine yerleştirme
```

```
# Output
'Five plus ten is 15 and not 30.'
```

10. Lambda Fonksiyonları

- Tek bir ifade içeren küçük, anonim fonksiyonlar.

```
def add(x, y):
    return x + y
add(5, 3)
add = lambda x, y: x + y # İki argümanı toplayan lambda fonksiyonu
add(5, 3)
```

11. Ternary Koşullu Operatör

- Tek satırda basit bir if-else ifadesi.

```
>>> age = 15
>>> print('kid' if age < 18 else 'adult')
>>> # Koşul doğruysa 'kid', değilse 'adult' yazdır
```

```
# Output
kid
```

12. `*args` ve `**kwargs`

Fonksiyonlara değişken sayıda argüman gönderme.

- `*args` : Değişken sayıda pozisyonel argümanı bir tuple olarak alır.

```
def fruits(*args): # Ne kadar argüman gönderileceği bilinmiyorsa kullanılır
     for fruit in args:
        print(fruit)
fruits("apples", "bananas", "grapes")
```

```
# Output
apples
bananas
grapes
```

- `**kwargs`: Değişken sayıda anahtar kelime argümanını bir sözlük olarak alır.

```
def fruit(**kwargs): # Anahtar=değer şeklinde gönderilen argümanları alır
    for key, value in kwargs.items():
        print("{0}: {1}".format(key, value))
fruit(name = "apple", color = "red")
```

```
# Output
name: apple
color: red
```

13. `if __name__ == "__main__":`

- Bir dosyanın doğrudan çalıştırıldığında mı yoksa içe aktarıldığında mı farklı kod çalıştığını gösterme.

```
# dosya_adi.py olarak kaydedin

print("Bu satır her zaman çalışır.")

if __name__ == "__main__":
    print("Bu satır sadece dosya doğrudan çalıştırıldığında çalışır.")

def hello():
    print("Bu bir fonksiyondur ve import edildiğinde kullanılabilir.")
```

```
# Bu dosyayı doğrudan çalıştırırsanız (`python dosya_adi.py`),
# her iki `print` satırını da görürsünüz.
# Bu dosyayı başka bir Python dosyasından `import dosya_adi`
# şeklinde içe aktarırsanız, sadece ilk `print` satırını görürsünüz.
# `if __name__ == "__main__":` bloğunun içindeki satır çalışmaz.
# `hello()` fonksiyonu ise import eden dosyadan çağrılabilir.
```

- Bu yapı, bir dosyanın hem çalıştırılabilir bir betik hem de yeniden kullanılabilir bir modül olmasını sağlar.

```
# Python program to execute function directly
def add(a, b):
    return a+b

add(10, 20) # we can test it by calling the function save it as calculate.py
```

```
# Output
30
```

```
# Now if we want to use add module in calculate.py

# main.py
if __name__ == "__main__":
    add(3, 5)

# calculate.py
import calculate # Başka bir dosyadan içe aktarma
calculate.add(3, 5)
```

```
# Output
8
```