

Python Eğitimi

Cosmios Akademi - Fatih Kuşçu

Python'a Giriş ve Temeller

Python, Guido van Rossum tarafından 1991 yılında geliştirilen, yüksek seviyeli, yorumlanabilir, nesne yönelimli bir programlama dilidir. Python'un temel felsefesi okunabilirliği ve basitliği üzerine kuruludur. Bu yüzden, Python kodları hem yazması kolaydır hem de okunabilirliği yüksektir. Python, web geliştirme, veri bilimi, yapay zeka, bilimsel hesaplamalar, otomasyon ve daha birçok alanda kullanılmaktadır.



Kurulum ve Çalışma Ortamı Hazırlama

Python'ı kullanmaya başlamadan önce bilgisayarınıza Python'ı kurmanız gerekmektedir. Python'ın resmi web sitesi olan python.org üzerinden işletim sisteminize uygun Python sürümünü indirip kurabilirsiniz. Kurulum sırasında "Add Python to PATH" seçeneğini işaretlemeniz, komut satırından Python'ı direkt olarak çalıştırabilmeniz için önemlidir.



Temel Veri Tipleri

String (str): Metinsel veriler için kullanılır. Örneğin: "Merhaba Dünya"

Integer (int): Tam sayılar için kullanılır. Örneğin: 5

Float (float): Ondalıklı sayılar için kullanılır. Örneğin: 3.14

Boolean (bool): Mantıksal değerler için kullanılır. True veya False



Değişkenler ve Değişken Tanımlama Kuralları

Değişkenler, verileri saklamak için kullanılır. Python'da bir değişkene değer atamak için eşittir (=) işaretini kullanırız. Örneğin:

```
isim = "Fatih"
```

```
yas = 25
```



Matematiksel İşlemler

Toplama (+), çıkarma (-), çarpma (*), bölme (/), tam sayı bölmesi (//), kalan (%), üs alma (**) gibi işlemleri yapabilirsiniz. Örneğin:

$$\text{toplam} = 5 + 3$$

$$\text{fark} = 10 - 2$$

$$\text{carpim} = 4 * 2$$

$$\text{bolme} = 8 / 2$$



Koşullu ifadeler

Python'da koşullu ifadeler, belirli koşulların doğruluğuna bağlı olarak farklı kod bloklarının çalıştırılmasını sağlar. Temel koşullu ifadeler if, elif (else if'in kısaltması), ve else'dir.

if İfadesi: Eğer belirtilen koşul doğruysa (True), if bloğu içindeki kod çalıştırılır.

elif İfadesi: Önceki koşullar yanlışsa (False) ve bu koşul doğruysa, elif bloğu içindeki kod çalıştırılır.

else İfadesi: Tüm koşullar yanlışsa, else bloğu içindeki kod çalıştırılır.



Koşullu ifadeler

yas = 20

if yas < 18:

 print("Reşit değilsiniz.")

elif yas >= 18 and yas < 65:

 print("Yetişkin bir bireysiniz.")

else:

 print("Yaşlı bir bireysiniz.")



Alıştırma

1. `isim` adlı bir değişken oluştur, içine kendi adını koy, ekrana yazdır.
2. `yas` değişkenine bir tam sayı ata, ekrana “Ben X yaşımdayım” yazdır (f-string ile).
3. `pi` değişkenine 3.14 değerini koy. Yarıçapı 5 olan bir dairenin alanını hesapla ve yazdır (alan = πr^2).
4. Kullanıcıdan yaşını `input()` ile al. Onu **tam sayıya** çevirip, 10 yıl sonra kaç yaşında olacağını hesapla.
5. “3” ve “4” (string) değerlerini toplayınca ne olur? Peki bu değerleri tam sayıya çevirip toplarsan ne olur?

Alıştırma

`x = "12.5"` değişkenini önce float'a, sonra int'e çevirip sonucu yazdır.

`a = 5` ve `b = 2.5` için toplama, çıkarma, çarpma ve bölme işlemlerinin tiplerini (`type()`) ekrana yazdır.

`sayi = 0` ve `metin = ""` için `bool()` değerlerini test et (True mu False mu?).

`None` tipinde bir değişken oluştur (`x = None`) ve `type()` ile tipini ekrana yazdır.

`isim = "Python"` için `isim[0]`, `isim[-1]` ve `isim[1:4]` çıktıları nedir? Tahmin et, sonra dene.

`5 > 3 and 2 < 1` ifadesinin sonucu nedir? `True` mu `False` mu?

`not (3 == 3)` sonucu nedir?

Kullanıcıdan iki sayı al, büyük olanı ekrana yazdır (`if` kullan).



Alıştırma

`5 > 3 and 2 < 1` ifadesinin sonucu nedir? `True` mu `False` mu?

`not (3 == 3)` sonucu nedir?

Kullanıcıdan iki sayı al, büyük olanı ekrana yazdır (`if` kullan).



Döngüler

Döngüler, belirli bir kod bloğunun tekrar tekrar çalıştırılmasını sağlar. Python'da iki temel döngü türü vardır: for ve while.

for Döngüsü: Belirli bir dizide (liste, demet, string vb.) dolaşmak için kullanılır.

while Döngüsü: Bir koşul doğru olduğu sürece döngü devam eder.



for Döngüsü Örneği:

```
isimler = ["Ahmet", "Mehmet", "Ayşe", "Fatma"]
```

```
for isim in isimler:
```

```
    print("Merhaba", isim)
```



Soru

- 1) Belirli aralıktaki sayıları toplayan programı yaz
- 2) Kullanıcıdan bir kelime al ve bu kelimenin her harfini **for** döngüsü ile alt alta yazdır.
- 3) Kullanıcıdan kaç sayı gireceğini sor ve bu sayılar arasında bir **for** döngüsü ile gezerek ortalamasını hesapla.



while Döngüsü Örneği:

```
sayi = 5
```

```
while sayi > 0:
```

```
    print(sayi)
```

```
    sayi -= 1 # sayi = sayi - 1
```



Soru

- 1) Bir sayıyı 0'a kadar azaltarak konsola her adımda ki değerini yazan programı yaz.
- 2) Kullanıcıdan tekrar tekrar sayı al ve bu sayıları topla. Kullanıcı, toplama işlemini bitirmek istediğinde 0 girmelidir. En sonunda, girilen sayıların toplamını ekrana yazdır.
- 3)Kullanıcıdan sürekli sayı al ve 0 girene kadar `while` döngüsü kullanarak kaç tane çift sayı girdiğini say.



Döngü Kontrol İfadeleri

`break`: Döngüyü sonlandırır.

`continue`: Döngünün mevcut iterasyonunu sonlandırır ve sonraki iterasyona geçer.

`pass`: Hiçbir işlem yapmadan geçer. Genellikle döngü veya fonksiyonların henüz tamamlanmadığı yerlerde geçici bir yer tutucu olarak kullanılır.



Soru 1

```
for i in range(5):
```

```
    if i == 3:
```

```
        break
```

```
    print(i)
```

bu kod ne yapar ?



Soru 2

Bir liste içindeki sayıların toplamını hesaplayan ve sonucu ekrana yazdıran bir Python programı yazın. Listede sadece sayılar olacak ve liste şu şekilde:

[1, 2, 3, 4, 5]. Bu programı yazarken bir for döngüsü kullanın.



Soru 3

Kullanıcıdan alınan bir sayının çift mi yoksa tek mi olduğunu kontrol eden bir Python programı yazın. Programınız, kullanıcıdan bir sayı girmesini istemeli ve sayı çift ise ekrana "Sayı çift.", tek ise "Sayı tek." mesajını yazdırmalı. Bu programı yazarken if-else koşullu ifadelerini kullanın.



Fonksiyonlar ve Modüller

Fonksiyonlar, belirli bir görevi yerine getiren, tekrar tekrar kullanılabilen kod bloklarıdır. Python'da `def` anahtar kelimesi ile fonksiyon tanımlanır. Fonksiyonlar, kodun okunabilirliğini ve yeniden kullanılabilirliğini artırır.



Fonksiyon Tanımlama ve Çağırma:

```
def selamla():
```

```
    print("Merhaba, Fatih!")
```

```
# Fonksiyonu çağır
```

```
selamla()
```




Parametreler ve Argümanlar:

```
def selamla(isim):
```

```
    print("Merhaba, " + isim + "!")
```

```
# Fonksiyonu argüman ile çağır
```

```
selamla("Fatih")
```



Fonksiyon dönüş değerleri

```
def toplama(a, b):
```

```
    return a + b
```

```
sonuc = toplama(5, 3)
```

```
print(sonuc) # 8
```



Modül içe aktarma

```
import math
```

```
sonuc = math.sqrt(16) # Karekök hesaplama
```

```
print(sonuc) # 4.0
```



Soru 4

Bir liste içindeki en büyük sayıyı bulan bir fonksiyon yaz. Bu fonksiyon, liste olarak bir argüman alacak ve listedeki en büyük sayıyı dönecek.



Soru 6

Kullanıcıdan alınan 2 sayının çarpımını yazan fonksiyonu yaz.



Soru 7

Bir liste ve bir sayı alan, ve verilen sayının listede kaç kez geçtiğini döndüren bir fonksiyon yaz?

Örneğin, liste = [1, 4, 3, 7, 4, 8, 4] ve sayı = 4 için fonksiyon 3 döndürmelidir, çünkü 4 sayısı listede 3 kez geçmektedir.



Veri Yapıları

Python'da dört temel yerleşik veri yapısı vardır: Listeler, Demetler (Tuples), Sözlükler (Dictionaries) ve Kümeler (Sets). Her birinin kullanım amacı ve özellikleri farklıdır.



Listeler

Listeler, sıralı eleman koleksiyonlarını saklamak için kullanılır. Elemanlar köşeli parantezler [] içinde tanımlanır ve her bir eleman virgülle ayrılır. Listeler değiştirilebilir (mutable), yani listeye eleman eklenebilir, eleman çıkarılabilir veya elemanlar değiştirilebilir.



Liste kullanımı

```
meyveler = ["elma", "muz", "kiraz"]  
print(meyveler)  
# Listeye eleman ekleme  
meyveler.append("portakal")  
print(meyveler)  
# Listedden eleman çıkarma  
meyveler.remove("muz")  
print(meyveler)  
# Liste elemanına erişim  
print(meyveler[0]) # "elma" çıktısını verir
```



Liste Özellikleri:

Negatif İndeksleme

`print(my_list[-1])` # Son öğeyi yazdırır

Dilimleme

`print(my_list[1:3])` # İkinci öğeden üçüncü öğeye kadar olan öğeleri alır (3 dahil değil)

Eleman Ekleme

`my_list.append(4)` # Listenin sonuna 4 ekler `my_list.insert(1, 5)` # Listenin birinci indeksine 5 ekler

Eleman Silme

`my_list.remove(4)` # 4 değerini siler

`popped = my_list.pop(1)` # İkinci öğeyi siler ve döndürür

`del my_list[0]` # İlk öğeyi siler



Listeler Üzerinde Diğer İşlemler

`len(my_list)` - Listenin uzunluğunu döndürür.

`my_list + my_list2` - İki listeyi birleştirir.

`my_list * 2` - Listeyi tekrarlar.

`my_list.reverse()` - Listeyi ters çevirir.

`my_list.sort()` - Listeyi sıralar.

`max(my_list), min(my_list)` - Listede en büyük ve en küçük öğeyi bulur.

`my_list.index(value)` - Verilen değerın indeksini döndürür.

`my_list.count(value)` - Verilen değerin listede kaç kez geçtiğini döndürür.

`list(sequence)` - Bir diziyi listeye dönüştürür.



Soru 5

Bir liste üzerinde arama yapan ve aranan değerin o listede olup olmadığı söyleyen fonksiyonu yaz.



Soru

Bir üniversite sınıfında öğrencilerin notlarını tutan bir liste var. Bu listedeki notlar sırasıyla şu şekilde:

notlar = [88, 75, 96, 55, 83, 65, 72, 89, 90, 99]

Sınıfın not ortalamasını hesaplayın.

En yüksek ve en düşük notu bulun.

Notları büyükten küçüğe doğru sıralayın.

Not listesine yeni gelen bir öğrencinin notu olan 85'i ekleyin.

En düşük notu silin ve yerine 74 ekleyin.

90 ve üzeri not alan öğrenci sayısını bulun.



Demetler (Tuples)

Demetler, değiştirilemeyen (immutable) sıralı eleman koleksiyonlarıdır. Demet elemanları, normal parantez () içinde tanımlanır. Değiştirilemeyen yapısı nedeniyle, demetler listelere göre daha az yer kaplar ve bazı durumlarda programınızı hızlandırabilir.



Demet Kullanımı

```
renkler = ("kırmızı", "yeşil", "mavi")
```

```
print(renkler)
```

```
# Demet elemanına erişim
```

```
print(renkler[1]) # "yeşil" çıktısını verir
```



Demet İşlemleri

`single_element_tuple = (1,)` # tek eleman tanımlandığında , mutlaka koyulmalı

`print(my_tuple[0])` # İlk elemanı yazdırır

`print(my_tuple[-1])` # Son elemanı yazdırır

`print(my_tuple[1:3])` # İkinci elemandan üçüncü elemana kadar olan elemanları alır

`nested_tuple = (1, 2, (3, 4))` # demet içinde demet



Demet İşlemleri

`len(my_tuple)` - Demetin uzunluğunu döndürür.

`my_tuple + my_tuple2` - İki demeti birleştirir.

`my_tuple * 2` - Demeti tekrarlar.

`2 in my_tuple` - Bir elemanın demette olup olmadığını kontrol eder.



Soru 1

Bir kitaplık yönetim sistemi için kitap bilgilerini tutan bir demet listesi düşünün. Her bir demet, bir kitabın ISBN numarası, adı, yazarı ve yayın yılı gibi bilgileri içersin. Kitap bilgileri şu şekilde tanımlansın:

kitaplar = [

("978-1-60309-452-8", "Watchmen", "Alan Moore", 1987),

("978-1-891830-85-3", "The League of Extraordinary Gentlemen", "Alan Moore", 1999),

("978-1-78116-593-1", "V for Vendetta", "Alan Moore", 1988),

("978-0-316-76948-9", "The Road", "Cormac McCarthy", 2006)

]



Soru 1.1

Bu kitap listesi üzerinde aşağıdaki işlemleri gerçekleştirin:

Yazarı "Alan Moore" olan tüm kitapların bilgilerini yazdırın.

Yayın yılı 2000'den önce olan kitapların isimlerini ve yayın yıllarını yazdırın.

ISBN numarası "978-1-60309-452-8" olan kitabın adını yazdırın.



Sözlükler (Dictionaries)

Sözlükler, anahtar-değer çiftleri koleksiyonunu saklar. Her eleman, bir anahtar ile bir değer arasında eşleştirilir. Sözlükler, köşeli parantez {} içinde tanımlanır ve elemanlar virgülle ayrılır. Sözlükler değiştirilebilir.



Sözlükler kullanımı

```
kisi = {"isim": "Ahmet", "yas": 30, "sehir":  
"İstanbul"}
```

```
print(kisi)
```

```
# Sözlüğe yeni bir anahtar-değer çifti  
ekleme
```

```
kisi["meslek"] = "Mühendis"
```

```
print(kisi)
```

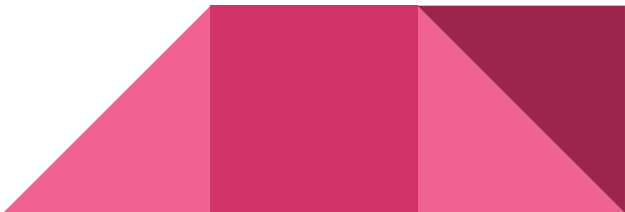
```
# Sözlükten bir anahtar-değer çifti çıkarma
```

```
del kisi["yas"]
```

```
print(kisi)
```

```
# Anahtara göre değere erişim
```

```
print(kisi["isim"]) # "Ahmet" çıktısını verir
```



Sözlükler Kullanımı

pop() ile eleman silme ve değeri döndürme

```
year = book.pop("year")
```

```
print(year)
```

del ile eleman silme

```
del book["pages"]
```

popitem() ile son eklenen elemanı silme ve döndürme

```
item = book.popitem()
```

```
print(item)
```



Soru

```
books = {  
    "1984": False,  
    "Kaşağı": True,  
    "denemeler": False,  
    "Moby Dick": True  
}
```



Soru

books adında, kitap adlarını anahtar olarak ve onların ödün alınma durumunu (True ödün alındı, False ödün alınmadı) değeri olarak içeren bir sözlük.

Görev:

Kütüphaneye yeni bir kitap ekleyin. Bu kitap ödün alınmamış olmalı.

Herhangi bir kitabın ödün alınma durumunu güncelleyin.

Ödün alınmış tüm kitapların listesini yazdırın.

Ödün alınmamış kitap sayısını hesaplayın ve yazdırın.



SORU

Telefon Rehberi Oluşturma

Adımlar:

1. Boş bir sözlük oluştur (`rehber = {}`).
2. Kullanıcıdan **isim** ve **telefon numarası** al.
3. Kullanıcı "q" yazarsa döngüden çık.
4. Sözlüğe `rehber[isim] = numara` şeklinde ekle.
5. Döngü bitince sözlüğü ekrana yazdır.



SORU

Sınıf Not Ortalaması

Adımlar:

1. Boş bir sözlük oluştur (`notlar = {}`).
2. Kullanıcıdan öğrenci adı ve notunu al.
3. "q" girilince veri alma işlemini bitir.
4. Tüm notların toplamını ve sayısını bul.
5. Ortalama = toplam / öğrenci sayısı şeklinde hesapla.
6. Ekrana yazdır.

Kümeler (Sets)

Kümeler, sırasız ve benzersiz eleman koleksiyonlarını saklamak için kullanılır. Küme elemanları, küme parantezleri $\{$ içinde tanımlanır ve virgülle ayrılır. Kümeler, matematikteki kümeler gibi işlemleri destekler (birleşim, kesişim, fark vb.).



Küme Kullanımı

```
renkler = {"kırmızı", "yeşil", "mavi"}
```

```
print(renkler)
```

```
# Küme elemanı ekleme
```

```
renkler.add("sarı")
```

```
print(renkler)
```

```
# Küme elemanı çıkarma
```

```
renkler.remove("yeşil")
```

```
print(renkler)
```



Küme Kullanımı

Birleşim

```
union_set = set1.union(set2)
```

```
print(union_set)
```

Kesişim

```
intersection_set = set1.intersection(set2)
```

```
print(intersection_set)
```

```
original_set = {1, 2, 3}
```

```
copied_set = original_set.copy()
```

```
print(copied_set)
```

Fark

```
difference_set =  
set1.difference(set2)
```

```
print(difference_set)
```

Simetrik fark

```
symmetric_difference_set =  
set1.symmetric_difference(set2)
```

```
print(symmetric_difference_set)
```



Soru 1

Bir üniversitede üç farklı öğrenci kulübü var: Kitap Kulübü, Sinema Kulübü ve Doğa Yürüyüşü Kulübü.

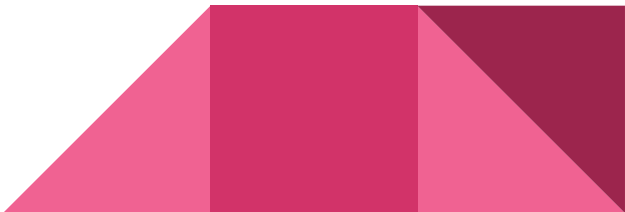
Kitap Kulübü: {"Ahmet", "Beril", "Cem", "Deniz", "Elif"}

Sinema Kulübü: {"Ahmet", "Fatma", "Cem", "Gizem", "Elif", "Hakan"}

Doğa Yürüyüşü Kulübü: {"Cem", "Deniz", "İrem", "Elif", "Kerem"}



Soru 1.1

- Üniversitede toplam kaç farklı öğrenci kulüplere üye olmuştur?
 - Tüm kulüplere üye olan öğrenciler kimlerdir?
 - Yalnızca Kitap Kulübüne üye olan öğrenciler kimlerdir?
 - Kitap ve Doğa Yürüyüşü Kulüplerine üye olup Sinema Kulübünde olmayan öğrenciler kimlerdir?
 - Doğa Yürüyüşü Kulübünden rastgele bir öğrencinin çıkarılması durumunda kalan öğrenciler kimlerdir?
- 

SORU

Tekrar Edenleri Temizleme

Soru: Kullanıcıdan alınan sayıları küme kullanarak tekrar edenleri temizleyin.

Adımlar:

1. Kullanıcıdan virgülle ayrılmış sayılar al.
2. `split()` ile listeye çevir.
3. Listeyi kümeye çevirerek tekrarları kaldır.
4. Sonucu ekrana yazdır.



`*args **kwargs`

Python'da `*args` ve `**kwargs` kullanımı, fonksiyonlara esnek bir şekilde argüman ve anahtar kelime argümanları geçirmeyi sağlayan bir özelliktir. Bu özellikler sayesinde, fonksiyonlar sabit sayıda argüman almak zorunda kalmaksızın, istenilen sayıda pozisyonel argüman veya anahtar kelime argümanı alabilir.



*args Kullanımı

```
def fonksiyon(*args):
```

```
    for arg in args:
```

```
        print(arg)
```

```
fonksiyon(1, 2, 3, 4) # 1, 2, 3 ve 4'ü yazdırır
```



****kwargs Kullanımı**

```
def fonksiyon(**kwargs):
```

```
    for key, value in kwargs.items():
```

```
        print(f"{key} = {value}")
```

```
fonksiyon(ad="Ali", soyad="Yılmaz") # ad = Ali ve soyad = Yılmaz'ı yazdırır
```



Hem *args Hem **kwargs Kullanımı

```
def fonksiyon(*args, **kwargs):  
    for arg in args:  
        print(arg)  
    for key, value in kwargs.items():  
        print(f"{key} = {value}")
```

```
fonksiyon(1, 2, ad="Ali", soyad="Yılmaz")
```



Argümanları *args ve **kwargs Kullanarak Geçirme

```
def fonksiyon(a, b, c, d):  
    print(a, b, c, d)
```

```
liste = [1, 2]
```

```
sozluk = {"d": 4, "c": 3}
```

```
fonksiyon(*liste, **sozluk) # 1 2 3 4 yazdırır
```



Soru

Aşağıdaki fonksiyon ****kwargs** ile istenen kadar anahtar-değer çifti almalı

ve her birini "anahtar: değer" formatında ekrana yazdırmalıdır.

def ogrenci_bilgileri(****kwargs**):

Beklenen çıktı:

isim: Ali

sınıf: 10A

okul_no: 123

ogrenci_bilgileri(isim="Ali", sınıf="10A", okul_no=123)



Soru

Aşağıdaki fonksiyon, verilen listeye *args ile gelen elemanları eklemelidir.

```
def listeye_ekle(liste, *args):
```

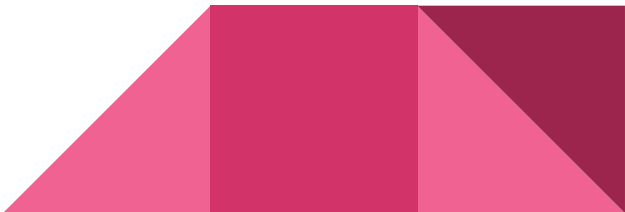
```
    # Eksik kod
```

```
    return ...
```

Beklenen çıktı: ['elma', 'armut', 'muz']

```
meyveler = ['elma']
```

```
print(listeye_ekle(meyveler, 'armut', 'muz'))
```



Soru

Aşağıdaki fonksiyon `**kwargs` ile gelen verilerden `islem` parametresine göre işlem yapacaktır.

islem parametresi: "topla", "carp", "ortalama" olabilir.

Örnek kullanım: `hesapla(islem="topla", a=5, b=10, c=15)`

```
def hesapla(**kwargs):
```

```
    # islem = kwargs.get("islem")
```

```
    # Diğer anahtarlar sayı olacak: a, b, c, ...
```

```
    # İlgili işlemi yapıp sonucu döndürmelisiniz
```

```
    return ...
```

```
print(hesapla(islem="topla", a=5, b=10, c=15)) # 30
```

```
print(hesapla(islem="carp", x=2, y=3, z=4))    # 24
```

```
print(hesapla(islem="ortalama", p=10, q=20))  # 15.0
```

Recursive (Özyinelemeli) Fonksiyonlar

Temel Durum (Base Case): Rekürsif çağrıların sona ermesi için bir veya birden fazla temel durum belirlenmelidir. Temel durum, fonksiyonun daha fazla kendini çağırmadan bir sonuç döndürdüğü durumdur.

Rekürsif Durum (Recursive Case): Fonksiyonun kendini daha küçük bir alt problemle tekrar çağırdığı durumdur. Bu, problemin parçalanıp küçültülmesini sağlar.

Sonlanma: Her rekürsif çağrı, problemi sürekli küçülterek en sonunda temel duruma ulaşmalıdır; aksi takdirde fonksiyon sonsuz bir döngüye girer.



Recursive Kullanımı

```
def fibonacci(n):  
    if n == 0:  
        return 0  
    elif n == 1:  
        return 1  
    else:  
        return fibonacci(n-1) + fibonacci(n-2)  
print(fibonacci(7)) # Çıktı: 13
```

Fibonacci dizisi, her sayının kendisinden önce gelen iki sayının toplamı olduğu bir sayı dizisidir. İlk iki sayı genellikle 0 ve 1 olarak kabul edilir. Bu dizinin ilk birkaç elemanı şöyle sıralanır: 0, 1, 1, 2, 3, 5, 8, 13, ...



Recursive Avantajları / Dez Avantajları

Kod Sadeliği: Karmaşık algoritmalar, rekürsif fonksiyonlar kullanılarak daha anlaşılır ve sade bir şekilde yazılabilir.

Problem Çözme: Bazı problemler (örneğin, ağaç ve graf traversalleri, Fibonacci sayıları, faktöriyel hesaplama) rekürsif olarak daha doğal bir şekilde ifade edilir.

Performans ve Bellek Kullanımı: Her rekürsif çağrı için fonksiyon çağrı yığnında bir katman eklenir, bu da bellek kullanımını artırır. Ayrıca, bazı durumlarda rekürsif çözümler iteratif çözümlere göre daha az performanslı olabilir.

Anlaşılması Zor Olabilir: Rekürsif fonksiyonlar bazen, özellikle derinlikleri fazla olduğunda, anlaması ve hata ayıklaması zor olabilir.



Soru

Faktöriyel hesaplama, $n! = n * (n-1) * (n-2) * \dots * 1$ formülüyle hesaplanır ve $0! = 1$ olarak tanımlanır. recursive fonksiyon kullanarak faktöriyel hesaplama kodunu yazın.




Lambda Expression

Python'da lambda ifadeleri, küçük ve anonim fonksiyonları tanımlamak için kullanılır. Bu ifadeler, lambda anahtar kelimesi ile başlar ve genellikle tek bir satırda yazılırlar. Lambda ifadelerinin temel özellikleri ve kullanımları hakkında detaylı bilgi vereyim:

lambda arguments: expression # şeklinde tanımlanır

arguments: Lambda fonksiyonuna geçirilecek argümanlar. Birden fazla argüman kullanılabilir ve virgülle ayrılır.

expression: Lambda fonksiyonunun döndüreceği değer. Bu ifade, argümanları kullanarak hesaplanır.



Lambda Expression

```
kare = lambda x: x * x
```

```
print(kare(5)) # Çıktı: 25
```

```
maks = lambda x, y: x if x > y else y
```

```
print(maks(5, 8)) # Çıktı: 8
```

```
tuples = [(1, 'bir'), (3, 'üç'), (4, 'dört'), (2, 'iki')]
```

```
tuples.sort(key=lambda x: x[1])
```


```
print(tuples) # Çıktı: [(1, 'bir'), (4, 'dört'), (2, 'iki'), (3, 'üç')]
```



Soru

Bir öğrenci listesi veriliyor. Her öğrenci bir sözlük olarak temsil ediliyor ve bu sözlükler isim ve not anahtarları içeriyor. Bu listeyi öğrencilerin aldığı notlara göre azalan sıralamada düzenleyin. Sıralamayı yerinde yapmak için `.sort()` metodunu ve bir lambda ifadesini kullanın.

```
ogrenciler = [  
    {'isim': 'Elif', 'not': 88},  
    {'isim': 'Ahmet', 'not': 92},  
    {'isim': 'Zeynep', 'not': 74},  
    {'isim': 'Mehmet', 'not': 85}  
]
```



Filter Fonksiyonu

Python'da filter fonksiyonu, belirli bir koşula göre bir diziden (liste, demet vb.) elemanları süzmenize olanak tanır. Bu fonksiyon, bir dizi üzerinde yineleme yapar ve her bir elemanı belirtilen bir fonksiyona uygular. Bu fonksiyonun döndürdüğü değer True ise, o eleman yeni bir diziye eklenir. Böylece, filter fonksiyonu, belirli bir koşulu sağlayan tüm elemanları içeren yeni bir dizi oluşturur.

```
filter(filterFunction, iterable)
```



Filter Fonksiyon

```
def is_even(num):
```

```
    return num % 2 == 0
```

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
filtered_numbers = filter(is_even, numbers)
```

```
print(list(filtered_numbers))
```



Filter Fonskiyonu

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
filtered_numbers = filter(lambda x: x % 2 == 0, numbers)
```

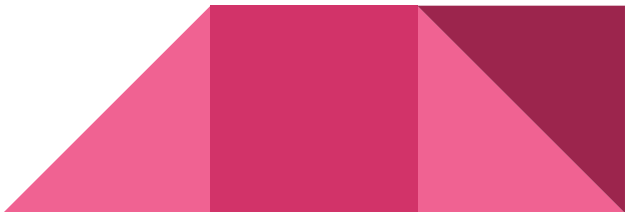
```
print(list(filtered_numbers))
```



Soru

Bir spor salonuna üye kişilerin listesi, her kişi bir sözlük olarak temsil ediliyor. Bu sözlükler isim, yas ve abonelik_suresi (ay olarak) anahtarlarını içeriyor. filter() fonksiyonunu ve lambda ifadesini kullanarak, abonelik süresi 12 aydan uzun olan üyeleri filtreleyip yeni bir liste oluşturun.

```
uyeler = [  
    {'isim': 'Ahmet', 'yas': 30, 'abonelik_suresi': 14},  
    {'isim': 'Mehmet', 'yas': 25, 'abonelik_suresi': 8},  
    {'isim': 'Ayşe', 'yas': 22, 'abonelik_suresi': 18},  
    {'isim': 'Elif', 'yas': 29, 'abonelik_suresi': 24},  
    {'isim': 'Osman', 'yas': 35, 'abonelik_suresi': 10}  
]
```



Map Fonksiyonu

Python'da map fonksiyonu, bir fonksiyonu bir dizi veya dizi benzeri bir veri yapısının (örneğin, listeler, demetler) tüm elemanlarına uygulamak için kullanılır. Bu işlem, her bir elemana belirtilen fonksiyonu uygulayarak yeni bir dizi oluşturur. map fonksiyonunun temel kullanımı şu şekildedir:

```
map(fonksiyon, dizi)
```



Soru

Bir e-ticaret platformunuz var ve şu anda satışta olan ürünlerin fiyatlarını güncellemek istiyorsunuz. Platformdaki her ürün için bir fiyat bilgisi ve ürünün indirimde olup olmadığı bilgisi bulunmaktadır. Amacınız, indirimde olan ürünlerin fiyatını %10 azaltmak ve sonrasında fiyatı 100 TL'nin üzerinde olan ürünleri seçmek.

urunler = [

{"ad": "Ürün A", "fiyat": 150, "indirim": True},

{"ad": "Ürün B", "fiyat": 80, "indirim": False},

{"ad": "Ürün C", "fiyat": 120, "indirim": True},

{"ad": "Ürün D", "fiyat": 200, "indirim": False},

{"ad": "Ürün E", "fiyat": 50, "indirim": True},

]



Soru

```
siparisler = [  
    {"siparis_no": 1, "musteri": "Ahmet", "yemekler": [("Hamburger", 25), ("Kola", 5), ("Tatlı", 15)]},  
    {"siparis_no": 2, "musteri": "Mehmet", "yemekler": [("Pizza", 20), ("Ayran", 3)]},  
    {"siparis_no": 3, "musteri": "Derya", "yemekler": [("Salata", 18), ("Su", 1), ("Çay", 3)]},  
    {"siparis_no": 4, "musteri": "Burcu", "yemekler": [("Makarna", 18), ("Tiramisu", 10), ("Limonata", 7)]},  
    {"siparis_no": 5, "musteri": "Cem", "yemekler": [("Steak", 50), ("Şarap", 45)]}  
]
```

Soru

Görevler

Her müşterinin ödemesi gereken toplam tutarı

Toplamı 30 TL'yi geçen siparişleri

Her müşterinin sadece içecekler için ödediği tutarı

Sadece pizza siparişi veren müşterilerin listelenmesi

ortalama sipariş tutarı



Comprehension

Comprehension (anlama) yapıları, Python'da koleksiyonları (listeler, sözlükler, kümeler) döngülerle daha okunaklı ve etkin bir şekilde oluşturmanıza olanak tanıyan bir özelliktir. Bu yapılar, genellikle bir döngü içindeki koşullu ifadeler ve işlemleri tek bir satırda ifade etmenizi sağlar. Comprehensionlar, yazılım geliştirme sürecini hızlandırır ve kodun okunabilirliğini artırır.



Liste Comprehensions

List comprehensionlar, bir liste oluşturma'nın kısa ve açık bir yolunu sağlar. Genel sözdizimi şöyledir:

[ifade for öğe in iterable if koşul]

ifade: Her öğe için uygulanacak işlem.

öğ: Geçici değişken, iterable içindeki her öğe için kullanılır.

iterable: Üzerinde döngü kurulacak olan koleksiyon (liste, demet, sözlük vb.).

koşul: (Opsiyonel) Yalnızca bu koşulu sağlayan öğeler için ifadenin uygulanacağı koşul.



Sözlük Comprehensions

Sözlük comprehensionları, anahtar/değer çiftleri içeren bir sözlük oluşturmanın kısa yoludur. Sözdizimi liste comprehensionlarına benzer:

{anahtar_ifadesi: değer_ifadesi for öge in iterable if koşul}



Küme Comprehensions

Küme comprehensionları, benzersiz öğeler içeren bir küme oluşturmanın kısa yoludur. Sözdizimi liste ve sözlük comprehensionlarına benzer:

`{ifade for öğe in iterable if koşul}`




Koşullu Comprehensions

Comprehensionlar içinde koşullar kullanarak, belirli koşulları sağlayan öğelerle sınırlı koleksiyonlar oluşturabilirsiniz. Ayrıca, if-else yapılarını kullanarak öğeler üzerinde daha karmaşık işlemler gerçekleştirebilirsiniz.




Sorular

- Bir liste içindeki tüm sayıların 4 katını hesaplayan bir liste comprehension yazınız.
 - 0'dan 20'ye kadar olan sayılardan sadece çift olanları içeren bir liste oluşturun.
 - numbers = [10, 23, 45, 68, 70, 93] bu dizide çift ve tek sayı ayrımı yapın
 - words = ["hello", "world", "list", "comprehension"] her kelimesini ve uzunluklarını içeren bir sözlük oluşturun.
 - 0'dan 50'ye kadar olan sayılardan hem 2'ye hem de 3'e bölünebilen sayıları içeren bir küme oluşturun.
- 

Modüller

Python'da modüller, Python kodlarını organize etmenin bir yoludur. Bir modül, işlevler, sınıflar ve değişkenler ile Python kodlarını içeren bir dosyadır. Modüller, kodun tekrar kullanılabilirliğini artırır ve büyük projeleri daha yönetilebilir parçalara ayırmayı sağlar. Python, birçok standart modülle birlikte gelir ve üçüncü taraf modülleri de kullanılabilir.

Bir modülü kullanmak için, `import` anahtar kelimesi ile modülü kodunuza dahil edersiniz. Modül içindeki belirli öğeleri doğrudan `import` etmek için `from ... import ...` sözdizimini kullanabilirsiniz.



Math Modülü

Python'un math modülü, matematiksel işlemler için birçok fonksiyon ve sabit sunar. Bu modül, yuvarlama işlemleri, trigonometrik fonksiyonlar, logaritmik fonksiyonlar, üstel fonksiyonlar ve daha fazlasını içerir.

`math.pi`: Pi sayısı (~ 3.14159)

`math.e`: Euler sayısı (~ 2.71828)



Sabitler

`math.pi`: Pi sayısının değeri.

`math.e`: Euler sayısının (Doğal logaritma tabanı) değeri.

`math.tau`: Tau değeri (2π).

`math.inf`: Pozitif sonsuzluk.

`math.nan`: "Not a Number" değeri.

Trigonometrik Fonksiyonlar

`math.sin(x)`: x'in sinüsünü hesaplar (x radyan cinsindendir).

`math.cos(x)`: x'in kosinüsünü hesaplar.

`math.tan(x)`: x'in tanjantını hesaplar.

`math.asin(x)`: x'in ark sinüsünü hesaplar.

`math.acos(x)`: x'in ark kosinüsünü hesaplar.

`math.atan(x)`: x'in ark tanjantını hesaplar.

`math.atan2(y, x)`: Y/X'in ark tanjantını döndürür; iki parametre alır.

Logaritmik ve Üstel Fonksiyonlar

`math.exp(x)`: e üzeri x'i hesaplar.

`math.log(x)`: x'in doğal logaritmasını hesaplar. İkinci bir argüman olarak taban alabilir.

`math.log10(x)`: x'in 10 tabanında logaritmasını hesaplar.

`math.log2(x)`: x'in 2 tabanında logaritmasını hesaplar.

`math.pow(x, y)`: x üzeri y'yi hesaplar.

`math.sqrt(x)`: x'in karekökünü hesaplar.

Yuvarlama ve Mutlak Değer Fonksiyonları

`math.ceil(x)`: x'ten büyük veya ona eşit en küçük tam sayıyı döndürür.

`math.floor(x)`: x'ten küçük veya ona eşit en büyük tam sayıyı döndürür.

`math.trunc(x)`: x'in tam sayıya dönüştürülmesi (kesirler atılır).

`math.fabs(x)`: x'in mutlak değerini döndürür.

Hiperbolik Fonksiyonlar

`math.sinh(x)`: x'in hiperbolik sinüsünü hesaplar.

`math.cosh(x)`: x'in hiperbolik kosinüsünü hesaplar.

`math.tanh(x)`: x'in hiperbolik tanjantını hesaplar.

`math.asinh(x)`: x'in ters hiperbolik sinüsünü hesaplar.

`math.acosh(x)`: x'in ters hiperbolik kosinüsünü hesaplar.

`math.atanh(x)`: x'in ters hiperbolik tanjantını hesaplar.

Diğer Fonksiyonlar

`math.fmod(x, y)`: x'in y'ye bölümünden kalanı döndürür (float için).

`math.factorial(x)`: x'in faktöriyelini hesaplar.

`math.gcd(x, y)`: x ve y'nin en büyük ortak bölenini hesaplar.

`math.isfinite(x)`: x'in sonlu bir sayı olup olmadığını kontrol eder.

`math.isinf(x)`: x'in sonsuz olup olmadığını kontrol eder.

`math.isnan(x)`: x'in NaN (Not a Number) olup olmadığını kontrol eder.

`math.copysign(x, y)`: y'nin işareti ile x'in büyüklüğünü birleştirir.

Kullanım Alanları

Alan Hesaplamaları

```
radius = 5 # yarıçap  
area = math.pi * radius ** 2  
print("Dairenin alanı:", area)
```

İpotenüs Hesaplama

```
a = 3  
b = 4  
c = math.sqrt(a**2 + b**2)  
print("İpotenüs uzunluğu:", c)
```

Kullanım Alanları

Maliyet ve Fiyat Hesaplamaları

```
principal = 1000 # başlangıç miktarı  
rate = 0.05 # yıllık faiz oranı  
years = 10  
final_amount = principal * math.pow((1 + rate),  
years)  
print("10 yıl sonra yatırımın değeri:", final_amount)
```

Açı Dönüşümleri

```
degrees = 90  
radians = math.radians(degrees)  
print("90 derecenin radyan cinsinden  
değeri:", radians)
```

Kullanım Alanları

Hacim Hesaplamaları

```
radius = 3
```

```
height = 5
```

```
volume = math.pi * radius**2 * height
```

```
print("Silindirin hacmi:", volume)
```

math modülü

Yükseklik ve Mesafe Ölçümleri

10 metre yükseklikte bir çubuğun gölgesi 5 metre ise güneşin açısı nedir?

```
shadow_length = 5
```

```
pole_height = 10
```

```
angle = math.atan(pole_height / shadow_length)
```

```
angle_degrees = math.degrees(angle)
```

```
print("Güneşin açısı (derece):", angle_degrees)
```



Random Modülü

Python'un random modülü, rastgele sayılar üretmek için çeşitli fonksiyonlar sunar. Bu modül, basit rastgele sayı üretiminden daha karmaşık dağılımlara kadar birçok işlevi içerir.

`random.random()`: 0.0 ile 1.0 arasında rastgele bir float sayı döndürür.

`random.randint(a, b)`: a ve b arasında (her iki sınır dahil) rastgele bir tam sayı döndürür.

`random.randrange(start, stop[, step])`: Belirtilen aralıkta rastgele bir tam sayı döndürür. step parametresi seçmeli olarak belirlenebilir.



Temel Fonksiyonlar

`random.seed(a=None, version=2)`: Rastgele sayı üreticinin başlangıç noktasını (seed) ayarlar.

`random.random()`: 0 ile 1 arasında rastgele bir float sayı döndürür.

`random.uniform(a, b)`: Belirtilen aralıkta rastgele bir float sayı döndürür.

Tam Sayı Üretimi

`random.randint(a, b)`: `[a, b]` aralığında rastgele bir tam sayı döndürür (b dahil).

`random.randrange(start, stop[, step])`: Belirtilen aralık ve adımla rastgele bir tam sayı döndürür.

Dizi İşlemleri

`random.choice(seq)`: Bir diziden rastgele bir eleman seçer.

`random.choices(population, weights=None, *, cum_weights=None, k=1)`: Popülasyondan ağırlıklara göre bir veya daha fazla eleman seçer.

`random.shuffle(x[, random])`: Bir dizi listeyi yerinde karıştırır.

`random.sample(population, k)`: Popülasyondan benzersiz elemanlar içeren bir liste döndürür.

İstatistiksel Dağılımlar

`random.normalvariate(mu, sigma)`: Normal (Gauss) dağılımından bir sayı döndürür.

`random.expovariate(lambd)`: Üstel bir dağılımdan bir sayı döndürür.

`random.lognormvariate(mu, sigma)`: Log normal dağılımdan bir sayı döndürür.

`random.vonmisesvariate(mu, kappa)`: Von Mises dağılımından bir sayı döndürür.

`random.gammavariate(alpha, beta)`: Gama dağılımından bir sayı döndürür.

`random.betavariate(alpha, beta)`: Beta dağılımından bir sayı döndürür.

`random.paretovariate(alpha)`: Pareto dağılımından bir sayı döndürür.

`random.weibullvariate(alpha, beta)`: Weibull dağılımından bir sayı döndürür.

Rastgele Sayı Üreteçleri

`random.getrandbits(k)`: Rastgele bir tam sayı döndürür (k bitlik).

`random.getstate()`: Üretecin mevcut durumunu döndürür.

`random.setstate(state)`: Üreteci belirtilen duruma ayarlar.

random modülü kullanımları

Rastgele bir float üretme

```
print(random.random())
```

Belirli bir aralıkta rastgele bir tam sayı üretme

```
print(random.randint(1, 10))
```

Bir listeden rastgele bir eleman seçme

```
items = ['apple', 'banana', 'cherry']
```

```
print(random.choice(items))
```

Bir listenin elemanlarını karıştırma

```
random.shuffle(items)
```

```
print(items)
```

Belirli bir aralıkta rastgele bir tam sayı üretme

```
print(random.randint(1, 10))
```

Yemek Menüsü Seçimi

Haftalık yemek planlamasında, belirli bir yemek listesinden rastgele yemekler seçmek

```
yemekler = ['Pizza', 'Hamburger', 'Salata', 'Tavuk', 'Balık']
```

```
haftalik_menu = random.sample(yemekler, 5) # Haftanın 5 günü için 5 yemek seç
```

```
print(haftalik_menu)
```

Soru : Piyango Çekilişi

Kullanıcıdan numara seçmelerini isteyin.

Rastgele bir piyango çekilişi yapın.

Kullanıcının seçtiği numaralarla çekiliş numaralarını karşılaştırın.

Kazançları hesaplayın ve sonucu gösterin.

```
def kullanici_numaralari_al(): # 1-49 arasında 6 adet benzersiz numara girmeli.  
def piyango_cikis(): # Program, 1'den 49'a kadar olan sayılardan rastgele 6 adet seçer.  
def karsilastir(): # Kullanıcı numaraları ile çekiliş numaraları karşılaştırılır.  
def odul_hesapla(): # Eşleşen sayı miktarına göre kazanılan ödül gösterilmeli  
def main():
```

random modülü



Soru : Zar Atma Olasılık Hesabı

6 yüzlü bir zar düşünün ve bu zarın herhangi bir yüzeyinin (örneğin 6 gelme) kaç defa geleceğini simüle ederek, belirli bir sayının gelme olasılığını hesaplayalım.

```
def zar_at(): # belirtilen sayıda zar atar ve her atışta '6' sayısının kaç kez geldiğini sayar.
```

```
def olasilik_hesapla(): # atış sayısına göre '6' sayısının gelme olasılığını yüzdelik olarak hesaplar.
```

```
def main():
```

random modülü



Time Modülü

Python'un time modülü, zamanla ilgili işlevler sunar. Bu modül, zamanı ölçmek, zaman damgaları ile çalışmak, programı belirli bir süre için duraklatmak ve sistem saatine erişmek gibi işlemleri gerçekleştirmenize olanak tanır.

time modülü, zamanla ilgili basit ve karmaşık işlemleri gerçekleştirmek için geniş bir araç seti sunar. Özellikle, zaman damgaları ile çalışmak, zamanı biçimlendirmek ve parse etmek, programı duraklatmak ve sistem saati ile etkileşimde bulunmak için kullanışlıdır.



Time

`time.time():`

Mevcut zamanı, Epoch (1 Ocak 1970) üzerinden geçen saniye cinsinden döndürür.

Örnek kullanım:

```
saniye = time.time()
```

```
print("Geçen zaman (saniye):", saniye)
```

Sleep

time.sleep(seconds):

Programı belirtilen saniye kadar durdurur.

Örnek kullanım:

```
print("3 saniye uyuyor...")
```

```
time.sleep(3)
```

```
print("Uyanıldı!")
```

time modülü



Ctime

`time.ctime(seconds=None):`

Bir zaman damgasını insanın okuyabileceği bir tarih ve saat biçimine dönüştürür.

Örnek kullanım:

```
zaman_damgasi = time.time()
```

```
okunabilir_zaman = time.ctime(zaman_damgasi)
```

```
print("Okunabilir zaman:", okunabilir_zaman)
```

GMtime ve Localtime

`time.gmtime(seconds=None)` ve `time.localtime(seconds=None)`:

Sistem saati yerel zaman dilimine (`localtime`) veya koordinatlı evrensel zamana (`gmtime`) çevrilir.

Örnek kullanım:

```
gmtime = time.gmtime()
```

```
localtime = time.localtime()
```

```
print("GM Zaman:", gmtime)
```

```
print("Yerel Zaman:", localtime)
```

mktime

`time.mktime(t):`

Yapılandırılmış zamanı (yıl, ay, gün, saat, dakika, saniye) Epoch'tan bu yana saniye olarak döndürür.

Örnek kullanım:

```
t = (2022, 9, 29, 10, 30, 0, 3, 272, 0) # yıl, ay, gün, saat, dakika, saniye, gün_no, yılın_günü, yaz_saati_uyg
```

```
zaman_damgasi = time.mktime(t)
```

```
print("Zaman damgası:", zaman_damgasi)
```

strftime ve strptime

`time.strftime(format, t=None)` ve `time.strptime(string, format)`:

Zamanı belirtilen formata göre dizeye dönüştürür (`strftime`) veya bir zaman dizisini belirtilen formata göre analiz eder (`strptime`).

Örnek kullanım:

```
t = time.localtime()
```

```
formatli_zaman = time.strftime("%Y-%m-%d %H:%M:%S", t)
```

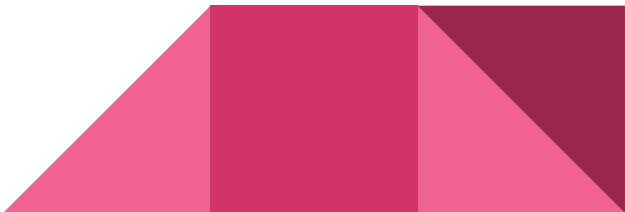
```
print("Formatlı zaman:", formatli_zaman)
```

```
dizi_zaman = "2022-09-29 10:30:00"
```

```
t = time.strptime(dizi_zaman, "%Y-%m-%d %H:%M:%S")
```

```
print("Yapılandırılmış zaman:", t)
```

time modülü



datetime Modülü

Python'un `datetime` modülü tarih ve zaman ile ilgili işlemler için oldukça güçlü ve esnek işlevler sağlar.



Ana Bileşenler

datetime.date: Yalnızca tarih bilgisi içerir (yıl, ay, gün).

datetime.time: Yalnızca zaman bilgisi içerir (saat, dakika, saniye, mikrosaniye).

datetime.datetime: Hem tarih hem de zaman bilgisini içerir.

datetime.timedelta: İki tarih veya zaman arasındaki farkı temsil eder.

datetime.tzinfo: Zaman dilimi bilgisini içeren soyut bir sınıftır.



datetime.date

Tarih oluşturma ve işlemleri:

Belirli bir tarih oluşturma

```
tarih = datetime.date(2022, 9, 29)
print("Tarih:", tarih)
```

Bugünün tarihi

```
bugun = datetime.date.today()
print("Bugünün tarihi:", bugun)
```

Tarihin yıl, ay ve gün olarak ayrıştırılması

```
print("Yıl:", tarih.year)
print("Ay:", tarih.month)
print("Gün:", tarih.day)
```



datetime.time

Zaman oluşturma ve işlemleri:

Belirli bir tarih oluşturma

```
zaman = datetime.time(12, 30, 45, 123456)
```

```
print("Zaman:", zaman)
```


Zamanın saat, dakika, saniye ve mikrosaniye olarak ayrıştırılması

```
print("Saat:", zaman.hour)
```

```
print("Dakika:", zaman.minute)
```

```
print("Saniye:", zaman.second)
```

```
print("Mikrosaniye:", zaman.microsecond)
```



datetime.datetime

Hem tarih hem zaman oluşturma ve işlemleri:

Belirli bir tarih oluşturma

```
dt = datetime.datetime(2022, 9, 29, 12, 30, 45)
```

```
print("Tarih ve Zaman:", dt)
```

Şu anki tarih ve zaman

```
simdi = datetime.datetime.now()
```

```
print("Şu anki tarih ve zaman:", simdi)
```

datetime'dan tarih ve zaman öğelerine erişim

```
print("Yıl:", dt.year)
```

```
print("Ay:", dt.month)
```

```
print("Gün:", dt.day)
```

```
print("Saat:", dt.hour)
```

```
print("Dakika:", dt.minute)
```

```
print("Saniye:", dt.second)
```



datetime.timedelta

Zaman farkı hesaplamaları:

Bir hafta sonraki tarih

```
bir_hafta = datetime.timedelta(weeks=1)
bugun = datetime.date.today()
bir_hafta_sonra = bugün + bir_hafta
print("Bir hafta sonra:", bir_hafta_sonra)
```

İki tarih arasındaki fark

```
tarih1 = datetime.date(2022, 9, 29)
tarih2 = datetime.date(2022, 10, 10)
fark = tarih2 - tarih1
print("Fark:", fark.days, "gün")
```



strftime ve strptime

Tarih ve zamanları string formatına çevirme (strftime) ve string'ten tarih/zaman çözümleme (strptime):

Tarih ve zamanı string'e dönüştürme

```
# Tarih ve zamanı string'e dönüştürme
```

```
dt = datetime.datetime.now()
```

```
formatli = dt.strftime("%Y-%m-%d %H:%M:%S")
```

```
print("Formatlı tarih ve zaman:", formatli)
```

String'i datetime'a çözümleme

```
dt_cozumlenmis =
```

```
datetime.datetime.strptime(formatli,  
"%Y-%m-%d %H:%M:%S")
```

```
print("Çözümlemiş tarih ve zaman:",  
dt_cozumlenmis)
```



Soru : Yaş Hesaplama

Kullanıcıdan alınan doğum tarihi bilgisine göre yaş hesaplayan programı yazın.

```
def yas_hesapla(): # Mevcut tarihi alır ve kullanıcının doğum tarihinden yaşını hesaplar.
```

```
def dogum_tarihi_al(): # Kullanıcıdan doğum tarihini YYYY-AA-GG formatında alır.
```

```
def main():
```



Soru : Randevu Zamanlayıcı

Kullanıcıdan bir randevu tarihi ve saati alarak, randevuya ne kadar kaldığını hatırlatan basit bir randevu zamanlayıcı program yazmanızı istiyorum.

```
def randevu_al(): # Kullanıcıdan, "YYYY-AA-GG SS" formatında randevu tarih ve saati  
    alır.datetime.datetime.strptime ile girdiyi doğrulayarak bir datetime objesine dönüştürür. Format  
    hatası olursa uygun mesaj verir ve tekrar girdi ister.
```

```
def kalan_sure_hesapla(): # Geçerli tarih ve saat (datetime.datetime.now()) ile randevu tarihi  
    arasındaki farkı hesaplar.Randevu geçmişse uygun bir mesaj döndürür, değilse kalan süreyi gün, saat  
    ve dakika olarak hesaplar ve geri döndürür.
```

```
def main():
```



OS Modülü

Python'da os modülü, işletim sistemi ile etkileşimde bulunmak için çeşitli fonksiyonlar sağlar. Bu modül, dosya işlemleri, izin işlemleri, işletim sistemi bilgilerini sorgulama ve işlem yönetimi gibi geniş bir yelpazede işlevsellik sunar.



os modülü

```
import os
```

```
#Geçerli çalışma dizinini döndürür.
```

```
print(os.getcwd())
```

```
#Çalışma dizinini değiştirir.
```

```
os.chdir('/path/to/directory')
```

```
print(os.getcwd())
```



os modülü

#Belirtilen dizindeki dosya ve dizinlerin listesini döndürür.

```
print(os.listdir('/path/to/directory')) # ['file1.txt', 'file2.txt', 'dir1']
```

#Yeni bir dizin oluşturur.

```
os.mkdir('new_directory')
```

#Gerekli tüm ara dizinlerle birlikte yeni bir dizin oluşturur.

```
os.makedirs('parent_dir/child_dir')
```



os modülü

#Boş bir dizini kaldırır.

```
os.rmdir('empty_directory')
```

#Boş olan tüm ara dizinlerle birlikte belirtilen dizini kaldırır.

```
os.removedirs('parent_dir/child_dir')
```

#Belirtilen dosyayı kaldırır.

```
os.remove('file.txt')
```



os modülü

#Dosya veya dizini yeniden adlandırır veya taşır.

```
os.rename('old_name.txt', 'new_name.txt')
```

```
os.rename('old_name.txt', 'path/to/new_name.txt')
```

#Yolları birleştirir.

```
full_path = os.path.join('/path', 'to', 'directory')
```

```
print(full_path) # /path/to/directory
```

#Belirtilen yolun var olup olmadığını kontrol eder.

```
print(os.path.exists('/path/to/directory')) # True veya False
```



os modülü

#Belirtilen yolun bir dosya olup olmadığını kontrol eder.

```
print(os.path.isfile('/path/to/file.txt')) # True veya False
```

#Belirtilen yolun bir dizin olup olmadığını kontrol eder.

```
print(os.path.isdir('/path/to/directory')) # True veya False
```



os modülü

#Belirtilen yolun mutlak yolunu döner.

```
absolute_path = os.path.abspath('file.txt')
```

```
print(absolute_path) # /home/kullanıcı/file.txt
```

#Belirtilen yolun dizin adını döner.

```
print(os.path.dirname('/path/to/file.txt')) # /path/to
```



Hata Yönetimi

Python'da hata yönetimi, programınızın beklenmedik hatalarla karşılaştığında çökmesini önlemek ve program akışını kontrol altında tutmak için önemlidir. Python, hataları yakalamak ve işlemek için **try**, **except**, **else**, **finally** ifadelerini kullanır. Bu yapılar sayesinde, hata yönetimi daha okunabilir, anlaşılır ve etkili bir şekilde gerçekleştirilebilir.



try-except ile Hata Yakalama

try:

```
# Hata oluşabilecek kod bloğu
```

```
sonuc = 10 / 0
```

except ZeroDivisionError:

```
# Hata yakalandığında çalışacak kod
```

```
print("Bir sayı sıfıra bölünemez!")
```



try-except ile Tüm Hataları Yakalama

try:

```
# Hata oluşabilecek kod
```

```
# ...
```

except Exception as e:

```
print(f"Bir hata oluştu: {e}")
```



Alınabilecek Hatalar

SyntaxError: Python sözdizimine uygun olmayan kod yazdığınızda ortaya çıkar. Genellikle yazım hataları, yanlış noktalama işaretleri veya blok yapılarında hatalar nedeniyle meydana gelir.

NameError: Tanımlanmamış bir değişkeni kullanmaya çalıştığınızda bu hata ile karşılaşabilirsiniz.

TypeError: Veri türleri arasında uyumsuz işlem veya çağrı yapıldığında bu hata alınır. Örneğin, bir sayıyı bir dizgeye eklemeye çalışmak.


IndexError: Bir listeye, demete veya başka bir dizinle erişilebilen veri türüne, var olmayan bir indeks ile erişmeye çalıştığınızda bu hatayı alırsınız.

KeyError: Sözlükte olmayan bir anahtara erişmeye çalıştığınızda meydana gelir.

AttributeError: Nesnenin sahip olmadığı bir özelliğe veya metoda erişmeye çalıştığınızda bu hatayla karşılaşabilirsiniz.

ValueError: Bir işlev veya operasyon için doğru türde ama uygun olmayan değer verdiğinizde ortaya çıkar.

ZeroDivisionError: Herhangi bir sayıyı sıfıra bölmeye çalıştığınızda bu hata meydana gelir.



Alınabilecek Hatalar

IOError: Giriş/çıkış işlemleri sırasında dosya açma, okuma veya yazma işlemleri sırasında bir hata oluştuğunda bu hatayla karşılaşabilirsiniz (Python 3.3 ve sonrasında **OSError** olarak da bilinir).

ImportError: Python, istenilen modülü veya modül içindeki belirli bir bileşeni bulamadığında bu hata ortaya çıkar.

ModuleNotFoundError: İstenen modül bulunamadığında Python 3.6'dan itibaren bu spesifik hata türü kullanılmaya başlandı.

IndentationError: Kodunuzda girintileme hataları varsa (örneğin, beklenmeyen bir girinti veya girintilerin uyumsuz olması), bu hata meydana gelir.

MemoryError: Programınız yeterli bellek alamadığında bu hata ortaya çıkar.

OverflowError: Aritmetik bir işlem sonucu çok büyük bir sayı (taşma) üretildiğinde bu hata meydana gelir.

RecursionError: Maksimum özyineleme derinliği aşıldığında (genellikle çok derin özyinelemeli çağrılar yüzünden) bu hata oluşur.

StopIteration: İteratörün sonuna ulaşıldığında ve daha fazla öge yokken **next()** fonksiyonu çağrıldığında bu hata fırlatılır.

UnicodeError: Unicode ile ilgili kodlama/çözme işlemleri sırasında bir sorun yaşandığında ortaya çıkar.

RuntimeError: Diğer kategorilere girmeyen hatalar için genel bir hata türüdür.



else Kullanımı

else bloğu, try bloğu içinde herhangi bir hata oluşmazsa çalıştırılacak kodları içerir:

try:

```
print("Merhaba")
```

except Exception as e:

```
print(f"Bir hata oluştu: {e}")
```

else:

```
print("Hiç hata oluşmadı!")
```



finally Kullanımı

try:

```
# Hata oluşabilecek kod
```

```
# ...
```

except Exception as e:

```
print(f"Bir hata oluştu: {e}")
```

finally:

```
print("Bu blok her durumda çalışır.")
```



Hata Fırlatma: raise

Programınızda belirli koşullarda kendi hatalarınızı fırlatmak için raise anahtar kelimesini kullanabilirsiniz:

```
x = -1
```

```
if x < 0:
```

```
    raise ValueError("x 0'dan küçük olamaz")
```



Soru

Bir matematik testi uygulaması oluşturacağız. Bu uygulama, kullanıcıdan iki sayı ve bu sayılar üzerinde gerçekleştirilecek bir matematik işlemi (toplama, çıkarma, çarpma, bölme) isteyecek. Kullanıcıdan alınan bilgilere göre işlemi gerçekleştirecek ve sonucu ekrana yazdıracaksınız. Aşağıdaki durumları ele almanız gerekmektedir:



Soru

Kullanıcıdan alınan girdilerin sayı olup olmadığını kontrol edin. Eğer sayı değilse, bir hata mesajı yazdırın.


Kullanıcı bölme işlemi seçtiğinde ve ikinci sayı 0 ise, bir hata mesajı yazdırarak sifıra bölme hatasını ele alın.

İşlem başarıyla gerçekleştirildiğinde, işlem sonucunu ekrana yazdırın.

İşlemler sırasında bir hata oluşursa (try içindeki kod bloğu), hatayı yakalayın ve kullanıcıya anlaşılır bir hata mesajı gösterin

İşlem sonucunu başarıyla hesapladıktan sonra, başka bir işlem yapmak isteyip istemediğini kullanıcıya sorun

Programın sonunda, başarılı veya başarısız olması fark etmeksizin, kullanıcıya teşekkür eden bir mesaj yazdırın



Dosya İşlemleri

Python'da dosya işlemleri, dosyaları okuma, yazma, güncelleme gibi işlemleri gerçekleştirmek için kullanılan işlemlerdir. Bu işlemler için `open()` fonksiyonu kullanılır ve bu fonksiyon ile dosya modları ve çeşitli parametreler üzerinden dosya işlemleri yönetilir.

```
open(file, mode='r', encoding=None)
```



Dosya İşlemleri : With Kullanımı

with ifadesi Python'da yalnızca dosya işlemleriyle sınırlı değildir; genel olarak kaynak yönetimi için kullanılır. Context Manager protokolünü destekleyen herhangi bir nesne ile kullanılabilir. Bu protokol, bir kaynağın kullanımını başlatmadan önce ve kullanımı bittikten sonra yapılması gereken işlemleri tanımlar. Bu, dosya işlemleri, ağ bağlantıları, veritabanı bağlantıları gibi kaynakların yönetimi için oldukça yararlıdır.

with open('ornek.txt', 'r', encoding='utf-8') as dosya:



Dosya İşlemleri: Modlar

Dosya Modları

r: Okuma modu. Dosya mevcut değilse hata verir.

w: Yazma modu. Dosya mevcut ise üzerine yazar; yoksa dosyayı oluşturur.

a: Ekleme modu. Dosya mevcutsa, dosyanın sonuna ekler; yoksa dosyayı oluşturur.

x: Özel oluşturma modu. Dosya zaten varsa hata verir.

b: İkili (binary) mod. Dosyayı ikili modda açar. Örneğin, rb, wb olarak kullanılır.

t: Metin (text) modu. Varsayılan moddur. Örneğin, rt (aslında sadece r ile aynı).

+: Güncelleme (okuma ve yazma) modu. Örneğin, r+, dosyayı hem okuma hem de yazma modunda açar.



Dosya İşlemleri : Yazma Fonksiyonları

write : Verilen string'i dosyaya yazar ve yazılan karakter (metin modunda) veya byte (ikili modda) sayısını döndürür.

```
dosya.write("Merhaba Python\n")
```

writelines : Verilen bir dizi/string listesini dosyaya yazar. writelines metoduna verilen her bir string, dosyaya arka arkaya yazılır; satır sonları otomatik olarak eklenmez.

```
dosya.writelines(["Merhaba", "Python"])
```



Dosya İşlemleri : Konum Fonksiyonları

tell()

Dosya içerisindeki mevcut konumu (imleç pozisyonunu) bir tamsayı olarak döndürür.

Kullanımı: `konum = dosya.tell()`

seek()

Dosyanın imleç konumunu değiştirir. offset, başlangıç noktasına göre pozisyon değişikliğidir. whence ise başlangıç noktasını belirler: 0 (dosyanın başı), 1 (mevcut konum), 2 (dosyanın sonu).

Kullanımı: `dosya.seek(0)` # Dosyanın başına git



Dosya İşlemleri : Okuma Fonksiyonları

read()


Dosyadan en fazla size karakter (metin modunda) veya byte (ikili modda) okur. size belirtilmezse veya negatif ise, dosyanın kalan tüm içeriğini okur.

Kullanımı: `icerik = dosya.read()`

readline()

Dosyadan bir satır okur. size ile en fazla kaç karakter veya byte okunacağını belirleyebilirsiniz. size belirtilmezse veya negatif ise, tam bir satır okur.

Kullanımı: `satir = dosya.readline()`



Dosya İşlemleri : Okuma Fonksiyonları

readlines()

Dosyanın kalan kısmındaki tüm satırları bir liste olarak okur. hint parametresi, okunacak maksimum karakter (metin modunda) veya byte (ikili modda) sayısını belirtir. hint belirtilirse, en fazla bu kadar karakter/byte kadarını içeren satırlar okunur.

Kullanımı: `satirlar = dosya.readlines()`



TXT Dosyalarında işlemler

Kullanacağımız temel modlar:

- `r(Read)`: Dosyayı **okumak** için açar. Dosya yoksa hata verir.
- `w(Write)`: Dosyayı **yazmak** için açar. Dosya varsa içeriğini siler, yoksa oluşturur.
- `a(Append)`: Dosyanın **sonuna ekleme** yapar. Dosya yoksa oluşturur.
- `r+`: Hem okuma hem yazma modu. Dosya yoksa hata verir.



TXT Dosyası Uygulama

- 1) Öğrenci isimlerinin listesini ogrenciler.txt dosyasına kayıt eden ve okumayı sağlayan programı yaz. Öğrenci isimleri satır satır kayıt edilmeli.
- 2) ogrenciler.txt dosyasından a harfı ile başlayan öğrencileri a_ogrenciler.txt dosyasına kayıt eden programı yaz.



Soru:günlük tutma uygulaması.

Bu uygulama, kullanıcıdan alınan günlük girdilerini bir dosyaya kaydedecek ve istenildiğinde bu girdileri okuyabilecek. Uygulamanın aşağıdaki özellikleri desteklemesi gerekmektedir:

Kullanıcıdan günlük için bir girdi alın ve bu girdiyi, tarih bilgisiyle birlikte "gunluk.txt" adlı bir dosyaya kaydedin. Her girdinin başına otomatik olarak güncel tarih eklenmelidir.

Kullanıcının mevcut günlük girdilerini okuyabilmesini sağlayın. Bu işlem için, kullanıcıdan bir komut alın ve eğer kullanıcı "okuma" modunu seçerse, "gunluk.txt" dosyasındaki tüm girdileri ekrana yazdırın.

Eğer "gunluk.txt" dosyası mevcut değilse ve kullanıcı okuma modunu seçerse, kullanıcıya dosyanın bulunamadığına dair bir mesaj gösterin ve yeni bir girdi eklemesini teklif edin.

Kullanıcı herhangi bir hata ile karşılaştığında (örneğin, dosyaya yazma sırasında), kullanıcıya anlaşılır bir hata mesajı gösterin ve programın düzgün bir şekilde sonlandırılmasını sağlayın.

JSON Dosyalarında işlemler

python'da json dosyalarda dönüşümleri yapabilmek için python içerisinde gelen json modülü bulunur. Bu modülü kullanarak yapacağımız işlemler;

- `json.dump()`: Python nesnesini JSON dosyasına yazar.
- `json.load()`: JSON dosyasını Python nesnesine çevirir.
- `json.dumps()`: Python nesnesini JSON string'ine dönüştürür.
- `json.loads()`: JSON string'ini Python nesnesine çevirir.

Dosya açma modları:

- "w": JSON dosyasını yazmak için açar (eski verileri siler).
- "r": JSON dosyasını okumak için açar.
- "a": **JSON için uygun değil!** (JSON yapısı eklemeye uygun değildir).



JSON dosyalarında işlem

```
# JSON'a Kaydetme
veri = {"ad": "Ahmet", "yas": 30, "şehir": "İstanbul"}
with open("kullanici.json", "w", encoding="utf-8") as dosya:
    json.dump(kullanici, dosya, ensure_ascii=False, indent=4)
```

json.dump() : Python nesnesini (sözlük, liste vb.) **doğrudan bir JSON dosyasına yazmak için kullanılır.**

Parametreler:

- obj: Kaydedilecek Python nesnesi (örn: sözlük).
- fp: Dosya pointer'ı (open() ile açılan dosya).
- ensure_ascii: Türkçe karakterler için False yapın.
- indent: Okunabilir format için girinti sayısı (4).

```
# JSON'u Okuma
with open("kullanici.json", "r", encoding="utf-8") as dosya:
    veri = json.load(dosya)
    print("Kullanıcı Adı:", veri["ad"])
```

Soru 1

Bir "ogrenciler.json" dosyası oluşturun ve içine 3 öğrencinin ad, soyad ve numara bilgilerini kaydedin.

Girilen numaraya göre öğrenci bilgilerini jsondan getiren fonksiyonu yazın.

```
ogrenciler = [
```

```
  {"ad": "Ali", "soyad": "Yılmaz", "numara": "101"},
```

```
  {"ad": "Zeynep", "soyad": "Kaya", "numara": "102"},
```

```
  {"ad": "Mehmet", "soyad": "Demir", "numara": "103"}]
```

Soru 2

Bir not uygulaması geliştireceğiz. Uygulama aşağıdaki işlevlere sahip olmalıdır:

1. **Not Ekleme:**

- Kullanıcıdan başlık ve içerik alınacak.
- Her not otomatik bir ID ile kaydedilecek (ID'ler 1, 2, 3 şeklinde artacak).
- Tarih, notun oluşturulduğu anın tarihi olarak (datetime modülü ile) otomatik eklenecek.
- Tüm notlar notlar.json dosyasında saklanacak.

2. **Not Silme:**

- Kullanıcıdan bir ID alınacak ve bu ID'ye ait not dosyadan silinecek.

3. **Notları Listeleme:**

- Tüm notlar okunup kullanıcıya gösterilecek.



Soru 2

```
{  
  "1": {  
    "tarih": "2023-10-30 14:30",  
    "baslik": "Alışveriş Listesi",  
    "icerik": "Süt, ekmek, yumurta"  
  },  
  "2": {  
    "tarih": "2023-10-31 09:15",  
    "baslik": "Toplantı Notları",  
    "icerik": "Proje sonuçlarını tartış"  
  }  
}
```



Soru 2

İstenen Fonksiyonlar:

- `not_ekle(baslik, icerik)`: JSON dosyasına yeni not ekler.
- `not_sil(id)`: Belirtilen ID'ye ait notu siler.
- `notlari_listele()`: Tüm notları okur ve ekrana yazdırır.



Nesne Tabanlı Programlama (OOP)

Nesne Yönelimli Programlama (OOP), günlük hayatta karşılaştığımız birçok durumla paralellik gösterir. OOP'nin temel kavramları olan sınıflar (classes) ve nesneler (objects) günlük hayattaki nesneler ve onların sınıflandırmalarıyla ilişkilendirilebilir.



Sınıf (Class) ve Nesne (Object)

Sınıf (Class): Günlük hayatta karşılaştığımız nesneleri düşünün; örneğin, araba, bisiklet, köpek gibi. Bu nesneleri tanımlayan genel özellikleri ve davranışları olan birer "sınıf" olarak düşünebiliriz. Yani, "Araba" bir sınıftır ve bu sınıf, arabaların genel özelliklerini (marka, model, renk) ve davranışlarını (sürmek, durmak) tanımlar.

Nesne (Object): Sınıfın somut bir örneğidir. Gerçek hayatta gördüğünüz her bir araba, "Araba" sınıfının bir örneğidir (nesnesidir). Örneğin, sizin mavi renkli, 2020 model Toyota Corolla arabanız, "Araba" sınıfının bir örneğidir.



Sınıf Oluşturma

Python'da bir sınıf, `class` anahtar kelimesi ile oluşturulur. Sınıf içerisinde metodlar (fonksiyonlar) ve özellikler (değişkenler) tanımlayabilirsiniz. Sınıfın yapıcı metodu `__init__` kullanılarak, nesne örneği oluşturulduğunda ilk çağrılan fonksiyon olur ve genellikle nesnenin başlangıç durumunu ayarlamak için kullanılır.



Sınıf Oluşturma

```
class Araba:
```

```
    def __init__(self, marka, model, yil):
```

```
        self.marka = marka
```

```
        self.model = model
```

```
        self.yil = yil
```



Nesne Oluşturma

Bir sınıftan nesne oluşturmak için, sınıf adı çağrılırken parantez içerisine sınıfın yapıcı metodunda (`__init__`) tanımlanan parametreler verilir.

```
arabam = Araba("Toyota", "Corolla", 2020)
```



Class'lar ve Fonksiyonlar

```
class Araba:
```

```
    def __init__(self, marka, model, yil):
```

```
        self.marka = marka
```

```
        self.model = model
```

```
        self.yil = yil
```

```
    def bilgi(self):
```

```
        return f"{self.marka} {self.model}, {self.yil}"
```




Soru

Bir Car sınıfı tanımlayın. Bu sınıfın brand ve model adında iki özelliği (attribute) olmalı. Ardından bu sınıftan bir nesne oluşturun ve brand ile model özelliklerine değer atayın.

Yukarıda tanımladığınız Car sınıfına `display_info()` adında bir metod ekleyin. Bu metod, arabanın markasını ve modelini bir cümle olarak yazdırsın. Örneğin, This car is a Mercedes-Benz C-Class.


Car sınıfına `total_cars` adında bir sınıf değişkeni ekleyin ve her yeni araba nesnesi oluşturulduğunda bu değeri bir artırın. Ayrıca, toplam araba sayısını döndüren bir sınıf metodunu (`get_total_cars()`) tanımlayın.



Sınıf Değişkenleri ve Nesne Değişkenleri

Class variables, bir sınıfa ait olan ve bu sınıftan türetilen tüm örnekler (instances) tarafından paylaşılan değişkenlerdir. Bu değişkenler, genellikle bir sınıfın tüm örnekleri için sabit değerler taşıyan ya da tüm örnekler tarafından ortak kullanılması gereken bilgiler için kullanılır. Class variables, sınıf tanımlandığı zaman sınıf tanımının içinde ve metodların dışında tanımlanır.

Instance variables ise bir sınıfın her bir örneğine özel olan değişkenlerdir. Her bir örnek, instance variables'ları kendi içinde saklar ve bu değişkenler yalnızca o örneğe özgüdür. Bu değişkenler genellikle, her bir örneğin kendi durumunu yansıtacak şekilde kullanılır.



Nesne, Sınıf ve Statik Fonksiyonlar

```
class MyClass:
```

```
    class_variable = 0
```

```
-
```

```
    def __init__(self, value):
```

```
        self.instance_variable = value
```

```
    def instance_method(self):
```

```
        # Instance method örneğin kendine ait verilere erişebilir
```

```
        return self.instance_variable, self.class_variable
```

```
@classmethod
```

```
def class_method(cls):
```

```
    # Class method sadece sınıf değişkenlerine erişebilir
```

```
    cls.class_variable += 1
```

```
    return cls.class_variable
```

```
@staticmethod
```

```
def static_method(value):
```

```
    # Static method ne sınıfa ne de örneğe erişim sağlar
```

```
    return value * 2
```



Kalıtım (Inheritance)

Kalıtım: Günlük hayatta, çocukların ebeveynlerinden bazı özellikleri ve davranışları miras alması gibi düşünebiliriz. Programlamada da benzer şekilde, bir sınıf başka bir sınıftan özelliklerini ve davranışlarını miras alabilir. Örneğin, "ElektrikliAraba" sınıfı, genel "Araba" sınıfından tüm özellikleri miras alır ve üzerine ekstra olarak pil kapasitesi gibi yeni özellikler ekleyebilir.



Kalıtım Oluşturma

```
class ElektrikliAraba(Araba):
```

```
    def __init__(self, marka, model, yil, pil_kapasitesi):
```

```
        super().__init__(marka, model, yil)
```

```
        self.pil_kapasitesi = pil_kapasitesi
```

```
    def pil_bilgisi(self):
```

```
        return f"Pil kapasitesi: {self.pil_kapasitesi} kWh"
```



Soru

Bir Hayvan sınıfı tanımlayın. Bu sınıf, hayvanın isim ve yaş özelliklerini barındırsın ve ses_cikar adında bir metoda sahip olsun ki bu metod varsayılan olarak "Sesim yok" mesajını döndürsün. Daha sonra, Hayvan sınıfından türeyen Köpek adında bir alt sınıf oluşturun ve ses_cikar metodunu "Hav hav" şeklinde özelleştirin. Her iki sınıftan da birer nesne oluşturarak, ses_cikar metodlarını çağırın



Polimorfizm

Polimorfizm: Farklı nesnelerin aynı eylemi farklı şekillerde gerçekleştirmesi durumudur. Örneğin, "konuşmak" eylemini düşünelim; insanlar konuşurken kelimeler kullanır, köpekler havlar, kediler miyavlar. Her biri "konuşmak" eylemini gerçekleştirir ancak yöntemleri farklıdır. Programlamada, farklı sınıflar aynı metod adını kullanabilir ancak her biri için farklı işlemler gerçekleştirebilir.



Polimorfizm

```
class Kus:
```

```
    def uc(self):
```

```
        print("Kuşlar uçar.")
```

```
class Penguen(Kus):
```

```
    def uc(self):
```

```
        print("Penguenler uçamaz.")
```

```
def kus_uca_bilir(kus):
```


```
    kus.uc()
```

```
kus = Kus()
```

```
penguen = Penguen()
```

```
kus_uca_bilir(kus)    # Çıktı: Kuşlar uçar.
```

```
kus_uca_bilir(penguen) # Çıktı:  
Penguenler uçamaz.
```



Soru

ÖdemeYöntemi adında bir temel sınıf tanımlayın. Bu sınıf, ödeme işlemini temsil eden öde adında bir metoda sahip olmalıdır. KrediKartı, PayPal ve BankaTransferi olmak üzere üç türetilmiş sınıf oluşturun. Her bir türetilmiş sınıf, öde metodunu kendine özgü bir şekilde uygulamalıdır:

KrediKartı sınıfı, kart numarası ve CVV bilgisi gerektirir ve bu bilgilerle ödeme işlemini simüle eder.

PayPal sınıfı, kullanıcı adı ve şifre gerektirir ve bu bilgilerle ödeme işlemini simüle eder.

BankaTransferi sınıfı, IBAN numarası gerektirir ve bu bilgiyle ödeme işlemini simüle eder.



Kapsülleme (Encapsulation)

Kapsülleme: Günlük hayatta, bir evin iç düzeni dışarıdan görülemez; yalnızca kapı, pencere gibi belirli noktalardan erişim sağlanabilir. Benzer şekilde, programlamada kapsülleme, bir nesnenin iç durumunun ve davranışlarının dış dünyadan gizlenmesi ve sadece belirli arayüzler üzerinden erişilebilir olmasıdır. Bu sayede, nesnenin iç yapısı hakkında bilgi sahibi olmadan da kullanılabilir.



kapsülleme

```
class Hesap:
```

```
    def __init__(self, sahip, bakiye=0):
```

```
        self.sahip = sahip
```

```
        self.__bakiye = bakiye # Bakiye özel bir değişken olarak  
tanımlanır
```

```
    def para_yatir(self, miktar):
```

```
        if miktar > 0:
```

```
            self.__bakiye += miktar
```

```
            print(f"{miktar} TL yatırıldı.")
```

```
        else:
```

```
            print("Geçersiz miktar.")
```

```
    def para_cek(self, miktar):
```

```
        if 0 < miktar <= self.__bakiye:
```

```
            self.__bakiye -= miktar
```

```
            print(f"{miktar} TL çekildi.")
```

```
        else:
```

```
            print("Yetersiz bakiye veya geçersiz miktar.")
```

```
    def bakiye_goruntule(self):
```

```
        return self.__bakiye
```



Soyutlama (Abstraction)

Python'da abstract kavramı, sınıflar ve metotlar üzerinden soyut sınıflar ve soyut metotlar yaratılmasını sağlayarak daha yapılandırılmış ve standartlara uygun kod yazımına olanak tanır. Soyut sınıflar (abstract classes), diğer sınıfların türetileceği temel yapıları tanımlar, ancak kendilerinden nesne yaratılmasına izin vermez. Soyut metotlar ise, türetilen sınıflarca kesinlikle uygulanması gereken metotları belirtir.



Soyutlama (Abstraction)

```
from abc import ABC, abstractmethod
```

```
class Sekil(ABC):
```

```
    @abstractmethod
```

```
    def alan_hesapla(self):
```

```
        pass
```

```
    @abstractmethod
```

```
    def cevre_hesapla(self):
```

```
        pass
```

```
class Dikdortgen(Sekil):
```

```
    def __init__(self, en, boy):
```

```
        self.en = en
```

```
        self.boy = boy
```

```
    def alan_hesapla(self):
```

```
        return self.en * self.boy
```

```
    def cevre_hesapla(self):
```

```
        return 2 * (self.en + self.boy)
```

