

EĞİTİM PLANI

Hafta	Konu Başlıkları
1. Hafta: Python'a Giriş	<ul style="list-style-type: none">- Python Nedir? Kullanım Alanları- Python Kurulumu ve IDE'ler- Python Çalışma Mantığı: İfadeler, Yorumlayıcı- <code>print()</code> ve <code>input()</code> Fonksiyonları- Veri Türleri (<code>int</code>, <code>float</code>, <code>string</code>, <code>bool</code>)- Yorum Satırları- Basit Matematiksel İşlemler- Casting (Tür Dönüşümü)- Koşullu İfadeler (<code>if</code> , <code>else</code> , <code>elif</code>)
2. Hafta: Döngüler ve Veri Yapıları	<ul style="list-style-type: none">- Döngüler: <code>for</code> , <code>while</code>- Döngülerde <code>break</code> , <code>continue</code> , <code>range()</code>- Listeler (<code>list</code>)- Demetler (<code>tuple</code>)- Sözlükler (<code>dictionary</code>)
3. Hafta: Fonksiyonlar ve Değişkenler	<ul style="list-style-type: none">- Fonksiyonlar ve Fonksiyon Tanımlama- <code>return</code> İfadesi- Global ve Yerel Değişkenler- Fonksiyonlarda Parametreler
4. Hafta: Dosya İşlemleri	<ul style="list-style-type: none">- Dosya Açma, Okuma, Yazma- Dosya Kapatma- Dosya Modları (<code>r</code> , <code>w</code> , <code>a</code>)- Hata Yönetimi ve Dosya İşlemleri

Koleksiyon Veri Türleri

Python'da listeler ve setler, veri koleksiyonlarını depolamak için kullanılan önemli veri yapılarıdır.

Listeler

Listeler, sıralı ve değiştirilebilir (mutable) veri koleksiyonlarıdır. İçinde farklı veri türlerini barındırabilirler ve aynı elemanı birden fazla kez içerebilirler.

Listelerin Özellikleri:

- Sıralıdır: Elemanların eklenme sırasına göre indekslenir.
- Değiştirilebilir: Elemanlar sonradan değiştirilebilir.
- Tekrarlı elemanları destekler.

Setler

Setler, sırasız ve değiştirilebilir veri koleksiyonlarıdır. Setler, yalnızca benzersiz elemanları saklar; yani aynı eleman set içinde birden fazla kez bulunamaz.

Setlerin Özellikleri:

- Sırasızdır: Elemanların sırası garanti edilmez.
- Değiştirilebilir: Elemanlar sonradan değiştirilebilir.
- Tekrar eden elemanları desteklemez.



```
# Liste oluşturma
meyveler = ["elma", "muz", "portakal", "elma"]

# Eleman ekleme
meyveler.append("çilek")

# Eleman silme
meyveler.remove("muz")

# Listeyi yazdırma
print(meyveler) # Çıktı: ['elma', 'portakal', 'elma', 'çilek']

# İndeks ile eleman erişimi
print(meyveler[0]) # Çıktı: 'elma'
```



```
# Set oluşturma
meyve_seti = {"elma", "muz", "portakal", "elma"} # 'elma' yalnızca bir kez alınır

# Eleman ekleme
meyve_seti.add("çilek")

# Eleman silme
meyve_seti.discard("muz") # 'muz' varsa siler, yoksa hata vermez

# Seti yazdırma
print(meyve_seti) # Çıktı: {'elma', 'portakal', 'çilek'} (sıra değişebilir)

# Set işlemleri
diğer_set = {"portakal", "armut"}
birlesim = meyve_seti.union(diğer_set) # Birleşim
print(birlesim) # Çıktı: {'elma', 'portakal', 'çilek', 'armut'}
```

Tuple'lar

Python'da tuple, sıralı ve değiştirilemez (immutable) bir veri yapısıdır. Listelere benzerler, ancak bir kez oluşturulduktan sonra elemanları değiştirilemez. Tuple'lar, birden fazla veriyi bir arada saklamak için kullanışlıdır ve genellikle bir nesnenin birden fazla özelliğini gruplamak için tercih edilir.

Tuple'ın Özellikleri:

- Sıralıdır: Elemanlar, eklenme sırasına göre indekslenir.
- Değiştirilemez: Oluşturulduktan sonra elemanlar üzerinde değişiklik yapılamaz.
- Tekrar eden elemanları destekler: Aynı eleman birden fazla kez yer alabilir.
- Heterojen: Farklı veri türlerini içerebilir.

```
# Tuple oluşturma
nokta = (3, 5) # İki elemanlı bir tuple
renk = ("kırmızı", "yeşil", "mavi") # String elemanlardan oluşan bir tuple

# Eleman erişimi
print(nokta[0]) # Çıktı: 3
print(renk[1]) # Çıktı: yeşil

# Tuple içindeki elemanların sayısını öğrenme
print(len(renk)) # Çıktı: 3

# Tekrar eden elemanlar
tekrar_tuple = (1, 2, 2, 3)
print(tekrar_tuple) # Çıktı: (1, 2, 2, 3)

# Tuple birleştirme
birlesim = nokta + renk
print(birlesim) # Çıktı: (3, 5, 'kırmızı', 'yeşil', 'mavi')

# Tuple'dan eleman çıkarma (hata verir)
# nokta[0] = 10 # TypeError: 'tuple' object does not support item assignment
```

Dictionary'ler

Python'da dictionary (sözlük), anahtar-değer çiftleri içeren ve sırasız bir veri yapısıdır. Dictionary'ler, belirli bir anahtara karşılık gelen bir değeri hızlı bir şekilde bulmak için kullanılır. Bu yapı, verileri etiketleyerek organize etmenin etkili bir yolunu sağlar.

Dictionary'nin Özellikleri:

- **Anahtar-Değer Çiftleri:** Her bir değer, benzersiz bir anahtar ile ilişkilendirilir.
- **Sırasız:** Elemanların sırası garanti edilmez.
- **Değiştirilebilir:** Elemanlar sonradan eklenebilir, güncellenebilir veya silinebilir.
- **Benzersiz Anahtarlar:** Aynı anahtar birden fazla kez kullanılamaz.

```
# Dictionary oluşturma
notlar = {
    "Ali": 85,
    "Ayşe": 92,
    "Mehmet": 78
}

# Eleman ekleme
notlar["Zeynep"] = 90

# Eleman güncelleme
notlar["Ali"] = 88

# Eleman silme
del notlar["Mehmet"]

# Dictionary'i yazdırma
print(notlar) # Çıktı: {'Ali': 88, 'Ayşe': 92, 'Zeynep': 90}

# Anahtar ile değer erişimi
print(notlar["Ayşe"]) # Çıktı: 92

# Anahtarların listesi
print(notlar.keys()) # Çıktı: dict_keys(['Ali', 'Ayşe', 'Zeynep'])

# Değerlerin listesi
print(notlar.values()) # Çıktı: dict_values([88, 92, 90])

# Anahtar-Değer çiftlerini listeleme
for anahtar, deger in notlar.items():
    print(f"{anahtar}: {deger}")

# Çıktı:
# Ali: 88
# Ayşe: 92
# Zeynep: 90
```

DÖNGÜLER NEDİR?

Döngüler, belirli bir koşul sağlandığı sürece bir kod bloğunu tekrar tekrar çalıştırmak için kullanılır. Python'da iki temel döngü türü vardır: for ve while.

Döngüler programlamada tekrar eden işleri daha basit hale getirmek için kullanılır. Aynı işlemi birçok kez manuel yazmaktan kurtarır.

FOR DÖNGÜSÜ

for döngüsü, belirli bir dizi eleman (list, tuple, string gibi) üzerinde yineleme yapmak için kullanılır.

```
for eleman in koleksiyon:  
    # işlem
```

```
liste = [1, 2, 3, 4, 5]  
  
for eleman in liste:  
    print(eleman)
```

STRING ÜZERİNDE FOR DÖNGÜSÜ

Bir string üzerindeki karakterleri
teker teker yazdırmak için for
döngüsü kullanılabilir.



```
kelime = "Python"  
  
for harf in kelime:  
    print(harf)
```


FOR DÖNGÜSÜ İLE LİSTE OLUŞTURMA

For döngüsü ile elemanları bir listede toplama işlemi.

```
sayilar = []

for i in range(5):
    sayilar.append(i ** 2) # i'nin karesini alıp
    listeye ekler
print(sayilar)

# Çıktı : [0, 1, 4, 9, 16]
```

İÇ İÇE (NESTED) FOR DÖNGÜSÜ

**For döngüleri iç içe kullanılabilir.
Bir örnek, bir matrisin tüm
elemanlarını yazdırma:**

```
matris = [  
    [1, 2, 3],  
    [4, 5, 6],  
    [7, 8, 9]  
]  
  
for satir in matris:  
    for eleman in satir:  
        print(eleman, end=" ")  
    print() # Satırları ayırmak için  
  
"""  
Çıktı:  
  
1 2 3  
4 5 6  
7 8 9  
  
"""
```

RANGE FONKSİYONU

range() fonksiyonu genellikle for döngüsünde kullanılır ve belirli bir aralıkta sayı üretir. 3 parametre alabilir:

- **range(baslangic, bitis, adım):**
 - **başlangıç:** Sayıların başlayacağı yer.
Varsayılan olarak 0'dır.
 - **bitiş:** Bitiş değeridir, bu dahil edilmez.
 - **adım:** Sayıların artış miktarıdır.
Varsayılan olarak 1'dir.



```
for i in range(5):  
    print(i) # 0, 1, 2, 3, 4
```




```
for i in range(2, 10, 2):  
    print(i)
```

WHILE DÖNGÜSÜ

while döngüsü, belirli bir koşul doğru olduğu sürece bir kod bloğunu çalıştırır.


```
while kosul:  
    # işlem
```



```
sayi = 0  
  
while sayi < 5:  
    print(sayi)  
    sayi += 1
```

İKİ DÖNGÜNÜN KARŞILAŞTIRILMASI

for ve while döngüsü ile aynı işlemi yapmak



```
# for döngüsü ile
for i in range(5):
    print(i)

# while döngüsü ile
i = 0
while i < 5:
    print(i)
    i += 1
```

DÖNGÜLERDE HATALI KULLANIMLAR

Sonsuz Döngüler: while döngüsünde koşul hiç yanlış olmazsa, döngü sonsuza kadar devam eder.



```
while True:  
    print("Bu bir sonsuz döngüdür!")
```

DÖNGÜ KONTROL DEYİMLERİ

Döngülerde kontrol deyimleri kullanılarak akış kontrol edilebilir.
En yaygın kullanılanlar **break** ve **continue** deyimleridir.

Break:

Döngüyü anında sonlandırmak için kullanılır.

Continue:

Döngünün o adımını atlar, sonraki adıma geçer.



```
for i in range(10):  
    if i == 5:  
        break # i 5 olduğunda döngü sona erer  
    print(i)
```



```
for i in range(5):  
    if i == 3:  
        continue # i 3 olduğunda o adımı atlar  
    print(i)
```

UYGULAMA: Liste (List) ve For Döngüsü

Kullanıcının girdiği 5 sayıyı bir listeye ekleyin ve bu sayılardan hangilerinin çift olduğunu bulun.

```
# Boş bir liste oluştur
sayilar = []

# 5 sayı al ve listeye ekle
for i in range(5):
    sayi = int(input("Bir sayı girin: "))
    sayilar.append(sayi)

# Çift sayıları bul
print("Çift sayılar:")
for sayi in sayilar:
    if sayi % 2 == 0:
        print(sayi)
```

UYGULAMA: İç İçe While ve For Döngüsü ile Basit Oyun

Sayı tahmin oyunu. Bilgisayar 1 ile 100 arasında bir sayı tutacak, kullanıcı bu sayıyı tahmin etmeye çalışacak.

```
import random

# 1 ile 100 arasında rastgele bir sayı tut
tutulan_sayi = random.randint(1, 100)

tahmin = None
deneme_sayisi = 0

# Tahmin doğru olana kadar devam et
while tahmin != tutulan_sayi:
    tahmin = int(input("Tahmininizi girin: "))
    deneme_sayisi += 1

    if tahmin < tutulan_sayi:
        print("Daha büyük bir sayı tahmin edin.")
    elif tahmin > tutulan_sayi:
        print("Daha küçük bir sayı tahmin edin.")
    else:
        print(f"Tebrikler! {deneme_sayisi} denemede doğru tahmin ettiniz!")
```