1. HAFTA

PYTHON TEMELLERİ VE KONTROL AKIŞI

1) Python Nedir? Nerelerde Kullanılır?

Python, Guido van Rossum tarafından geliştirilmiş, yüksek seviyeli, yorumlanmış, genel amaçlı bir programlama dilidir. Okunabilirliği yüksek sözdizimi sayesinde yeni başlayanlar için idealdir ve hızlı prototipleme imkanı sunar. "Yorumlanmış" olması, yazdığınız kodun doğrudan çalıştırıldığı anlamına gelir; derleme aşamasına ihtiyaç duymaz.

Python'ın kullanım alanları oldukça geniştir:

- **Web Geliştirme:** Django, Flask gibi güçlü framework'ler ile dinamik web siteleri ve web uygulamaları geliştirmek için kullanılır.
- **Veri Bilimi ve Yapay Zeka:** NumPy, Pandas, Scikit-learn, TensorFlow, Keras ve PyTorch gibi kütüphaneler sayesinde veri analizi, makine öğrenimi ve derin öğrenme projelerinde yaygın olarak kullanılır.
- **Otomasyon ve Scripting:** Tekrarlayan görevleri otomatikleştirmek, sistem yönetimi scriptleri yazmak ve dosya işlemleri yapmak için sıkça tercih edilir.
- **Oyun Geliştirme:** Pygame gibi kütüphanelerle basit oyunlar geliştirilebilir.
- **Masaüstü Uygulamaları:** PyQt, Tkinter gibi araçlarla grafiksel kullanıcı arayüzüne sahip masaüstü uygulamaları oluşturulabilir.
- **Ağ Programlama:** Soket programlama ile ağ üzerinde çalışan uygulamalar yazılabilir.

2) Python Kurulumu ve Temel Sözdizimi

Python'ı kullanmaya başlamak için öncelikle bilgisayarınıza kurmanız gerekir. Resmi Python web sitesi olan python.org adresinden en güncel sürümü indirebilirsiniz. Kurulum sırasında "Add Python to PATH" seçeneğini işaretlemek, komut satırından Python'ı kolayca çalıştırmanızı sağlar.

Python'ın temel sözdizimi oldukça sezgiseldir:

- **Girintileme (Indentation):** Python'da kod blokları süslü parantezler yerine **girintileme** ile belirlenir. Bu, kodun okunabilirliğini artırır ancak yanlış girintileme hatalarına yol açabilir.
- **Yorum Satırları:** Kodunuza açıklama eklemek için # karakterini kullanırsınız. # sonrasındaki her şey o satırda yorum olarak kabul edilir ve Python tarafından göz ardı edilir. Çoklu satır yorumları için üç tırnak (""" veya ''') kullanabilirsiniz.
- **Büyük/Küçük Harf Duyarlılığı:** Python büyük/küçük harf duyarlıdır. Yani myVariable ve myvariable iki farklı değişken olarak kabul edilir.
- **Noktalı Virgül:** Çoğu programlama dilinin aksine, Python'da her satırın sonuna noktalı virgül koymak zorunlu değildir. Ancak aynı satıra birden fazla komut yazacaksanız aralarına noktalı virgül koymanız gerekir (bu yaygın bir pratik değildir).

Örnek:

```
# Bu tek satırlık bir yorumdur
"""
Bu
çoklu
satırlı
bir yorumdur.
"""
print("Merhaba, Python!") # Ekrana çıktı veren temel bir komut
```

3) Değişkenler ve Veri Tipleri

Değişkenler, programlama sırasında verileri depolamak için kullanılan isimlendirilmiş bellek konumlarıdır. Python'da bir değişken tanımlamak için herhangi bir anahtar kelimeye ihtiyacınız yoktur; doğrudan bir değer atayarak değişken oluşturabilirsiniz. Değişkenin tipi, atadığınız değere göre otomatik olarak belirlenir.

Python'daki Temel Veri Tipleri:

- **Sayılar (Numbers):**
    - **Tam Sayılar (int):** Ondalıklı kısmı olmayan sayılar (örneğin, 5, -100, 0).
    - **Ondalıklı Sayılar (float):** Ondalıklı kısmı olan sayılar (örneğin, 3.14, -0.5, 2.0).
    - **Karmaşık Sayılar (complex):** Gerçek ve sanal kısımları olan sayılar (örneğin, 3 + 5j).
- **Metinler (Strings - str):** Tek tırnak (') veya çift tırnak (") arasına yazılan karakter dizileridir.
- **Boole (Boolean - bool):** Sadece iki değeri olabilen mantıksal veri tipidir: True (Doğru) veya False (Yanlış). Koşullu ifadelerde ve döngülerde karar verme mekanizmalarında kullanılır.
- **Listeler (Lists):** Farklı veri tiplerindeki öğeleri sıralı bir şekilde saklayabilen değiştirilebilir (mutable) koleksiyonlardır. Köşeli parantezler [] ile tanımlanır.
- **Demeler (Tuples):** Listelere benzer ancak değiştirilemez (immutable) koleksiyonlardır. Parantezler () ile tanımlanır.
- **Kümeler (Sets):** Sırasız ve tekrarlayan öğeleri barındırmayan koleksiyonlardır. Küme parantezleri {} ile tanımlanır veya set() fonksiyonu ile oluşturulur.
- **Sözlükler (Dictionaries):** Anahtar-değer (key-value) çiftleri halinde veri saklayan, sırasız ve değiştirilebilir koleksiyonlardır. Küme parantezleri {} ile tanımlanır ve anahtarlar ile değerler iki nokta : ile ayrılır.

Değişken Tanımlama Örnekleri:

isim = "Alice" **# String**

yas = 30 **# Integer**

boy = 1.75 **# Float**

evli_mi = True **# Boolean**

sehirler = ["Ankara", "İstanbul", "İzmir"] **# Liste**

koordinatlar = (40.7128, -74.0060) **# Tuple**

benzersiz_sayilar = {1, 2, 3, 3, 4} **# Küme (sonuç {1, 2, 3, 4} olur)**

kullanici_bilgisi = {"ad": "Mehmet", "soyad": "Yılmaz", "yas": 25} **# Sözlük**

print(type(isim)) **# <class 'str'>**

print(type(yas)) **# <class 'int'>**

4) Girdi/Çıktı İşlemleri

Programlarınızın kullanıcılarla etkileşime girmesi için girdi alma ve çıktı verme işlemleri hayati öneme sahiptir.

- **Çıktı Verme: print() Fonksiyonu:** print() fonksiyonu, ekrana veri yazdırmak için kullanılır. Birden fazla argüman alabilir ve bunları boşlukla ayırarak ekrana yazdırır.

Örnek:

- print("Merhaba dünya!")
- print("Benim adım", isim, "ve yaşım", yas)
- print(f"Benim adım {isim} ve yaşım {yas}.") # f-string ile daha modern bir çıktı

- **Girdi Alma: input() Fonksiyonu:** input() fonksiyonu, kullanıcıdan girdi almak için kullanılır. Kullanıcı bir değer girene kadar programın çalışmasını duraklatır ve girilen değeri bir string olarak döndürür.

Örnek:

- kullanici_adi = input("Lütfen adınızı girin: ")
- print("Merhaba,", kullanici_adi)
- # Not: input() her zaman string döndürdüğü için sayısal işlemler için dönüştürme gerekebilir.
- sayi1_str = input("Bir sayı girin: ")
- sayi1 = int(sayi1_str) # String'i tam sayıya dönüştürme
- sayi2_str = input("Başka bir sayı girin: ")
- sayi2 = float(sayi2_str) # String'i ondalıklı sayıya dönüştürme
- toplam = sayi1 + sayi2
- print("Toplam:", toplam)

5) Operatörler

Operatörler, değişkenler ve değerler üzerinde işlemler yapmak için kullanılan özel sembollerdir.

- **Aritmetik Operatörler:** Sayısal değerler üzerinde matematiksel işlemler yapar.
    - + (Toplama)
    - - (Çıkarma)
    - * (Çarpma)
    - / (Bölme - sonuç her zaman float)
    - % (Modülüs - kalanı bulma)
    - ** (Üs alma)
    - // (Tam sayı bölme - sonuç tam sayı)

- **Örnek:**
- a = 10 , b=3
- b = 3
- print(a + b) # 13
- print(a / b) # 3.333...
- print(a // b) # 3
- print(a % b) # 1
- print(a ** b) # 1000

- **Atama Operatörleri:** Değişkenlere değer atamak için kullanılır.

= (Atama)
+= (Ekle ve ata: x = x + y yerine x += y)
-= (Çıkar ve ata)
***=** (Çarp ve ata)
/= (Böl ve ata)
%= (Mod al ve ata)
****=** (Üs al ve ata)
//= (Tam sayı böl ve ata)

- **Örnek:**
- x = 5
- x += 3 **# x şimdi 8**
- print(x)

Karşılaştırma Operatörleri: İki değeri karşılaştırır ve True veya False döndürür.

- **==** (Eşit mi?)
- **!=** (Eşit değil mi?)
- **>** (Büyük mü?)
- **<** (Küçük mü?)
- **>=** (Büyük veya eşit mi?)
- **<=** (Küçük veya eşit mi?)

Örnek:

- print(5 == 5) **# True**
- print(5 != 10) **# True**
- print(5 > 10) **# False**

Mantıksal Operatörler: Koşulları birleştirmek için kullanılır ve True veya False döndürür.

- **and:** Her iki koşul da True ise True döndürür.
- **or:** Koşullardan herhangi biri True ise True döndürür.
- **not**: Koşulun tersini döndürür.

Örnek:

- yas = 20
- ehliyet_var_mi = True
- print(yas >= 18 and ehliyet_var_mi) **# True**
- print(yas < 18 or not ehliyet_var_mi) **# False**

Kimlik Operatörleri: İki nesnenin aynı bellek konumunda olup olmadığını kontrol eder.

- **is:** Aynı nesne mi?
- **is not:** Aynı nesne değil mi?

Örnek:

- liste1 = [1, 2, 3]
- liste2 = [1, 2, 3]
- liste3 = liste1
- print(liste1 is liste2) **# False (farklı bellek konumları)**
- print(liste1 is liste3) **# True (aynı bellek konumu)**

Üyelik Operatörleri: Bir elemanın bir dizide (liste, string, demet vb.) bulunup bulunmadığını kontrol eder.

- **in:** İçinde mi?
- **not in:** İçinde değil mi?

Örnek:

- mesaj = "Merhaba dünya"
- print("dünya" in mesaj) **# True**
- print("python" not in mesaj) **# True**
- sayilar = [1, 2, 3, 4]
- print(5 in sayilar) **# False**

6) Koşullu İfadeler (if, elif, else)

Koşullu ifadeler, belirli koşullara göre farklı kod bloklarının çalışmasını sağlar.

- **if ifadesi:** Belirtilen koşul True ise, if bloğundaki kod çalışır.

Örnek:

- hava_durumu = "güneşli"
- if hava_durumu == "güneşli":
- print("Dışarı çıkıp yürüyüş yapabiliriz.")

- **else ifadesi:** if koşulu False ise, else bloğundaki kod çalışır. else her zaman bir if ifadesinden sonra gelir.

Örnek:

- yas = 17
- if yas >= 18:
- print("Oy kullanabilirsiniz.")
- else:
- print("Oy kullanmak için yaşınız küçük.")

- **elif (else if) ifadesi:** Birden fazla koşulu kontrol etmek için kullanılır. İlk if koşulu False olduğunda elif koşullarına sırayla bakılır. Herhangi bir elif koşulu True olursa o blok çalışır ve diğer elif veya else blokları atlanır.

Örnek:

- notu = 85
- if notu >= 90:
- print("AA")
- elif notu >= 80:
- print("BA")
- elif notu >= 70:
- print("BB")
- else:
- print("Kaldı")

- **İç İçe Koşullu İfadeler:** Bir koşul bloğunun içinde başka koşullu ifadeler de kullanabilirsiniz.
  Örnek:

```
hava_durumu = "yağmurlu"
sicaklik = 10
if hava_durumu == "yağmurlu":
    if sicaklik < 15:
        print("Şemsiye al ve kalın giyin.")
    else:
        print("Şemsiye al.")
elif hava_durumu == "güneşli":
    print("Güneş kremi sür.")
else:
 print("Hava durumunu kontrol edin.")
```

7) Döngüler (for, while)

Döngüler, belirli bir kod bloğunu tekrar tekrar çalıştırmak için kullanılır.

1) **for Döngüsü:** for döngüsü, bir dizi (liste, demet, string, sözlük veya küme gibi) üzerindeki her öğe için belirli bir kodu çalıştırmak için kullanılır.

- **Liste üzerinde döngü:**

  Örnek:

```
meyveler = ["elma", "muz", "kiraz"]
for meyve in meyveler:
    print(meyve)
```

- **String üzerinde döngü:**

  Örnek:

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

- **range() Fonksiyonu ile döngü:** range() fonksiyonu, belirli bir sayı aralığı oluşturur.

  Örnek:

```
range(stop):  0'dan stop-1'e kadar sayılar üretir.
range(start, stop):  start'tan stop-1'e kadar sayılar üretir.
range(start, stop, step):  start'tan stop-1'e kadar step adımlarıyla sayılar üretir.
for i in range(5):          # 0, 1, 2, 3, 4
    print(i)
for i in range(2, 7):       # 2, 3, 4, 5, 6
    print(i)
for i in range(0, 10, 2): # 0, 2, 4, 6, 8
print(i)
```

- **enumerate() ile index ve değer alma:**

Örnek:

```
sebzeler = ["havuç", "domates", "salatalık"]
for index, sebze in enumerate(sebzeler):
    print(f"{index}: {sebze}")
```

2) **while Döngüsü:** while döngüsü, belirli bir koşul True olduğu sürece bir kod bloğunu tekrar tekrar çalıştırmak için kullanılır.

Örnek:

```
sayac = 0
while sayac < 5:
    print("Sayaç:", sayac)
    sayac += 1 # Sayacı artırmayı unutmayın, yoksa sonsuz döngü olur!
# Kullanıcı "çıkış" yazana kadar döngü
while True:
    komut = input("Bir komut girin (çıkış için 'q'): ")
    if komut == 'q':
        break # Döngüden çıkar
    print("Girdiğiniz komut:", komut)
```

8) break, continue

Bu anahtar kelimeler, döngülerin ve koşullu ifadelerin akışını kontrol etmek için kullanılır.

- **break:** Döngüyü tamamen sonlandırmak için kullanılır. break ifadesi çalıştırıldığında, program döngüden çıkar ve döngüden sonraki ilk kodu çalıştırmaya başlar.

Örnek:

```
for i in range(10):
    if i == 5:
        print("5'e ulaşıldı, döngü sonlandırılıyor.")
        break
    print(i)
```

- **continue:** Mevcut döngü iterasyonunu atlamak ve bir sonraki iterasyona geçmek için kullanılır. continue ifadesi çalıştığında, döngünün o anki iterasyonunda kalan kodlar atlanır ve döngü bir sonraki iterasyondan devam eder.

Örnek:

```
for i in range(10):
    if i % 2 == 0:      # Eğer sayı çift ise
        continue        # Bu iterasyonu atla ve bir sonrakine geç
    print(i)            # Sadece tek sayılar yazdırılacak
```