Paralel Hesaplama

Öğr. Gör. Zafer SERİN

ZİP CLASSI

- Python programlama dilinde zip() classı kullanılarak veri yapıları birleştirilebilir. zip() classı kendisine verilen veri yapılarından en kısa olan bittiğinde durur.

```
meyveler = ["elma", "armut", "ayva"]
kaloriler = [150, 250, 200]
gunler = ["pazartesi", "sali", "carsamba"]
birlestirilmis_liste = list(zip(meyveler, kaloriler, gunler))
print(birlestirilmis_liste)
```

FONKSİYONLAR

- Fonksiyon, belirli bir görevi yerine getiren ve tekrar kullanılabilir kod bloklarıdır. Fonksiyonlar, kodun daha modüler ve okunabilir olmasını sağlar.
- Python'da fonksiyon 'def' anahtar kelimesi ile tanımlanır. Fonksiyonun adı ve parantez içinde parametreleri (varsa) belirtilir. Fonksiyonun gövdesi, iki nokta üst üste (:) sonrasında girintili olarak yazılır.

```
def selamla():
    print("Merhaba")
```

FONKSİYONLAR

- Tanımlanan bir fonksiyonu çağırmak için, fonksiyonun adını ve parantezleri kullanırız.

```
def selamla():
    print("Merhaba")
selamla() # Çıktı Merhaba
```

FONKSİYONLAR

- **Parametreler:** Fonksiyon tanımlanırken parantez içinde belirtilen değişkenlerdir.
- **Argümanlar:** Fonksiyon çağrılırken parantez içinde verilen değerlerdir.

FONKSİYONLAR

- 4 tip fonksiyon tanımlamasından bahsedilebilir:

1. Parametre almayan ve değer döndürmeyen fonksiyon.
2. Parametre alan ve değer döndürmeyen fonksiyon.
3. Parametre almayan ve değer döndüren fonksiyon.
4. Parametre alan ve değer döndüren fonksiyon.

FONKSİYONLAR

- Parametre almayan ve değer döndürmeyen fonksiyon.

```
def metinyaz():
    print("Merhaba, nasılsın")
metinyaz()
```

Merhaba, nasılsın

FONKSİYONLAR

- Parametre alan ve değer döndürmeyen fonksiyon.

```
def toplamyaz(sayi1, sayi2):
    print(sayi1 + sayi2)
toplamyaz(3, 5)
```

8

FONKSİYONLAR

- Parametre almayan ve değer döndüren(return ile) fonksiyon.

```
def besdondur():
    return 5
donusDegeri = besdondur()
print(donusDegeri)
print(besdondur())
```

```
5
5
```

FONKSİYONLAR

- Parametre alan ve değer döndüren(return ile) fonksiyon.

```
def carp(sayi1, sayi2):
    return sayi1 * sayi2
carpim = carp(4, 7)
print(carpim)
print(carp(8, 9))
```

```
28
72
```

FONKSİYONLAR

- Fonksiyon tanımlanırken varsayılan parametreler verilebilir. Eğer fonksiyon çağrılırken bu argüman gönderilmezse varsayılan değer kabul edilir.

```
def kisiselamla(isim="Zafer"):
    print(f"Merhaba {isim}")

kisiselamla()
kisiselamla("Ahmet")
```

FONKSİYONLAR

- Fonksiyon çağrılırken, argümanları parametre adlarıyla birlikte belirtebiliriz. Bu durumda, argümanların sırası önemli değildir.

```
def dortislem(sayi1, sayi2):
    print(f"Toplam = {sayi1 + sayi2}")
    print(f"Çarpım = {sayi1 * sayi2}")
    print(f"Bolum = {sayi1 / sayi2}")
    print(f"Çıkarma = {sayi1 - sayi2}")

dortislem(3, 4)
dortislem(sayi2=6, sayi1=2)
```

FONKSİYONLAR

- Fonksiyon çağrılırken, argümanları parametre adlarıyla birlikte belirtebiliriz. Bu durumda, argümanların sırası önemli değildir.

```
Toplam = 7
Çarpım = 12
Bolum = 0.75
Çıkarma = -1
Toplam = 8
Çarpım = 12
Bolum =
0.3333333333333333
Çıkarma = -4
```

FONKSİYONLAR

- Bazı durumlarda, fonksiyona değişken sayıda argüman göndermek isteyebiliriz. Bu durumda *args ve **kwargs kullanılır.

- *args: Değişken sayıda sıralı argümanları alır (tuple olarak).

- **kwargs: Değişken sayıda anahtar kelime argümanlarını alır (dictionary olarak).

FONKSİYONLAR

```
def yaz(*args):
    print(args[0])

yaz(1, "selam", "deneme", 3.14, False)
```

1

FONKSİYONLAR

```
def bilgi_goster(**kwargs):
    print(kwargs["isim"])

bilgi_goster(isim = "Zafer", yas = 28, meslek = "akademisyen")
```

Zafer

FONKSİYONLAR

- Lambda fonksiyonları, tek satırlık, anonim fonksiyonlardır. Genellikle basit işlemler için kullanılır.

```
def topla(*args):
    return sum(args)
topla_lambda = lambda *args : sum(args)
t1 = topla(2,3,4)
t2 = topla_lambda(5,6,7)
print(t1)
print(t2)
```

FONKSİYONLAR

- Fonksiyon içinde fonksiyon tanımlanabilir.

```
def dis_fonksiyon():
    print("Dış fonksiyon çalıştı.")
    def ic_fonksiyon():
        print("İç fonksiyon çalıştı.")
    ic_fonksiyon()
dis_fonksiyon()
```

```
Dış fonksiyon
çalıştı.
İç fonksiyon
çalıştı.
```

FONKSİYONLAR

- Local Scope: Fonksiyon içinde tanımlanan değişkenler sadece o fonksiyon içinde erişilebilir.
- Global Scope: Fonksiyon dışında tanımlanan değişkenler tüm programda erişilebilir.
- Eğer bir fonksiyon içinde global bir değişken değiştirilmek istenirse global anahtar kelimesi kullanılır.

FONKSİYONLAR

```
bakiye = 100
def degistir():
    global bakiye
    bakiye = 250
print(bakiye)
degistir()
print(bakiye)
```

```
100
250
```

FONKSİYONLAR

- Bir fonksiyon, kendisini çağırarak bir problemi çözmeye çalışıyorsa, bu duruma özyinelemeli (recursion) fonksiyon denir.

```
def us_al(taban, us):
    if us == 0:
        return 1
    else:
        return taban * us_al(taban, us-1)
print(us_al(2, 3))
```

MAP CLASSI

- Python programlama dilinde map() classı kullanılarak bir fonksiyon bir veri yapısının tamamına uygulanabilir.

```
# f(x, y) = X^2 + y^3 + 5
def f(x, y):
    return (x**2)+(y**3)+5
liste1 = [2,3,11,21]
liste2 = [3,5,6,7]
mlist = list(map(f, liste1, liste2))
print(mlist)
```

[36, 139, 342, 789]

FİLTER CLASSI

- Python programlama dilinde filter() classı kullanılarak bir fonksiyon bir veri yapısının filtrelenmesi için kullanılabilir.

```
isimler = ["Ahmet", "Mehmet", "Ali", "Veli", "Ayse", "Zeynep", "Zafer"]
def a_bul(isim):
    return "a" in isim.lower()

print(list(filter(a_bul, isimler)))
```

['Ahmet', 'Ali', 'Ayse', 'Zafer']

KAPSAM(SCOPE) DETAYLARI

- Python programlama dilinde tanımlanan bir değişken sırası ile şu alanlarda aranır: Local, Enclosing, Global, Built-In. Burada Local tanımlandığı faaliyet alanında (indent-tab boşlukları) aranmasını, Enclosing kendi faaliyet alanında değil ama için de bulunduğu bir üst faaliyet alanında aranmasını(fonksiyon içinde fonksiyon gibi), Global en dışta hiçbir faaliyet alanına dahil olmadığında aranmasını ve Built-In ise Python'un kendi kütüphaneleri arasında aranmasını ifade eder. Bunlar arasındaki sıralama asla değişmez! Daima bu sıralama izlenecektir.

KAPSAM(SCOPE) DETAYLARI

```
# Global
isim = "Zafer 1"
def fonksiyon1():
    # Enclosing
    isim = "Zafer 2"
    def fonksiyon2():
        # Local
        isim = "Zafer 3"
        print(isim)
    fonksiyon2()

fonksiyon1()
```

PYTHON İLE NESNE YÖNELİMLİ PROGRAMLAMA(OOP)

- OOP, gerçek dünyayı temel alan ve dünyadaki her şeyin bir nesne olması durumunu programlamaya aktaran bir yaklaşımdır.
- OOP, birbirleriyle ilişkili veri(data) ve davranışları(behaviour) nesneler(objects) adı verilen birimlerde birleştirir.
- OOP'nin amacı daha modüler, esnek ve yeniden kullanılabilir kod yazmaktır.

CLASS(SINIF) VE NESNE(OBJECT/INSTANCE) NEDİR?

- OOP’de class bir şablon görevi görürken object ise o şablon kullanılarak oluşturulan sınırsız(RAM bellek dolana kadar) yapıyı ifade eder.
- Bir evin projesi class olarak ifade edilebilirken o proje kullanılırak aynı temel özellikleri sahip binlerce ev üretilmesi ise nesne olarak ifade edilebilir.

CLASS TANIMLANMASI

- Python programlama dilinde aşağıdaki gibi class tanımlanabilir.
- Bir class içerisinde özellik(attribute): isim, yas, kilo vb. tanımlanabilir.
- Bir class içerisinde metot(method): nesnenin yapabildiği eylemler tanımlanabi

```
class Ogrenci():
    isim = ""
    yas = 0
    def selamla(self):
        print("selam")
```

SELF VE __İNİT__ () İFADELERİ

- Bir class içerisinde o an işlem yapılan nesnenin kendisini temsil etmek için self anahtar sözcüğü kullanılır. Class içerisindeki metotların ilk parametresi her zaman self olmalıdır.
- Bir classtan nesne üretildiği anda çalışan ilk metot __init__() olarak isimlendirilen constructor(yapıcı metot)'tur.

OOP’NİN 4 TEMEL İLKESİ

- OOP 4 temel ilke etrafında şekillenir. Bunlar:
- Encapsulation(Kapsülleme/Sarmalama)
- Inheritance(Kalıtım/Miras Alma)
- Polymorphism(Çok biçimlilik)
- Abstraction(Soyutlama)

ENCAPSULATİON(KAPSÜLLEME/SARMALAMA)

- Kapsülleme sınıf içerisindeki verilere kontrollü bir şekilde erişilmesi ve bunlara dışarıdan kontrollü bir şekilde değer atanması temeline dayanır.
- Python programlama dilinde bir attribute veya method “__” ile sadece class içerisinden erişilebilir bir hale getirilebilir. “_” ile ise sadece class içerisinden erişilmesi tavsiye edilir hale getirilebilir. Burada “_” nin sadece bir öneri niteliği taşıdığı unutulmamalıdır.

ENCAPSULATİON(KAPSÜLLEME/SARMALAMA)

```
class Ogrenci:
    def __init__(self, yas):
        self.__yas = yas # '__' ile gizlendi

    def yas_goster(self):
        return self.__yas

ogr = Ogrenci(20)
# print(ogr.__yas)  # HATA!
print(ogr.yas_goster()) # Çıktı: 20
```

INHERITANCE(KALITIM/MİRAS ALMA)

- Bir classın özelliklerini bir diğer classa aktarmak için kalıtım kullanılır. Bu sayede hiyerarşik bir yapı kurulur ve üst classın izin verilen yapılarına da erişilebilir.
- Alt sınıfın üst sınıfın özelliklerini miras alması şeklinde de ifade edilebilir.

INHERITANCE(KALITIM/MİRAS ALMA)

```
class Kus:
    def uc(self):
        print("Uçuyorum...")

class Serce(Kus): # Kus'tan miras aldı
    pass

serce = Serce()
serce.uc() # Miras alınan metodu kullandı
```

POLYMORPHISM(ÇOK BİÇİMLİLİK)

- Polymorphism tanım olarak farklı nesnelerin, aynı isimdeki metoda farklı cevap vermesi olarak ifade edilebilir.

POLYMORPHISM(ÇOK BİÇİMLİLİK)

```
class Kedi:
    def ses(self): print("Miyav")

class Kopek:
    def ses(self): print("Hav hav")

def konustur(canli):
    canli.ses() # Hangi tip olduğunu bilmemize gerek yok

konustur(Kedi())  # Çıktı: Miyav
konustur(Kopek()) # Çıktı: Hav hav
```

ABSTRACTION(SOYUTLAMA)

- Abstraction alt sınıfları zorunlu kılan bir şablon (soyut sınıf) oluşturmak için kullanılır.

ABSTRACTION(SOYUTLAMA)

```
from abc import ABC, abstractmethod
class Arac(ABC): # Soyut şablon
    @abstractmethod
    def calistir(self): pass # Zorunlu metot

class Araba(Arac):
    def calistir(self): # Zorunlu olduğu için ezildi
        print("Vroom!")
araba = Araba()
araba.calistir() # Çıktı: Vroom!
```

ÖNEMLİ ÖZEL METOTLAR

- __str__(): Bulunduğu classtan bir nesne üretilip doğrudan yazdırıldığında çalışan metottur.
- __len__(): Bulunduğu classtan üretilen nesne için len() fonksiyonu çağrıldığında çalışan metottur.
- __getitem__(): Bulunduğu classtan üretilen nesne için bir anahtar(key) çağrımı yapılırsa çalışan metottur.

HATALARI ELE ALMAK

- Python programlama dilinde hataları ele almak ve yönetmek için try-except-else-finally blokları kullanılır.
- try: Hata vermesi muhtemel kod bu bloğun içerisine yazılır.
- except: try bloğu içerisindeki kod hata verirse yapılacak işlemler bu bloğa yazılır.
- else: try bloğu başarılı bir şekilde çalışır ve ondan sonra yapılması istenen işlemler bu bloğun içerisine yazılır.
- finally: Kod çalışsa da hata verse de yapılacak işlemler bu bloğun içerisine yazılır.

DOSYALARLA ÇALIŞMAK

- Python programlama dilinde open() fonksiyonu ile dosyalar açılabilir. read() metodu ile dosya içeriği okunabilir ama 1 kez okuma yaptıktan sonra tekrar okuma gerçekleşmesi için dosyanın başına seek() metodu ile dönmek gerekebilir.
- Bu şekilde okuma verimli bir yol olmadığı içi with anahtar sözcüğü ile dosya açılması tavsiye edilir. open() fonksiyonunun “mode” argümanına w-> write, r->read, a->append komutlarını verir.

TYPE ANNOTATION

- Python programlama dilinde değişkenlere istenirse “:” işareti ile tip belirtmesi yapılabilir. Bu sadece bir öneri mahiyetindedir bu şekilde int olarak belirtilen bir değişkene string değer atanabilir herhangi bir hata ile karşılaşılmaz!
- Fonksiyonlarda da geri dönüş değeri “->” işareti ile belirtilebilir.