1. Ders - Python

2 Eylül 2025 Salı 13:50

Upper Snake Case :
Değişmeyecek değeri ifade eder.

Kod dizaynı iki çeşittir :
1. Event Bazlı : İşin tamamı ile ilgilidir.
2.Domain Bazlı : İşin bir kısmı ile ilgilidir.

Kod yazma standartlarına bak.

!! ISO 8601

`değişken_adı:str` ile string boyutunda yeni bir değişken tanımlamış oluyoruz.

int .. , double .. boolean .., string .. diyerek değişken niteliğini tanımlayabiliriz. Noktalı yerler nitelik ismi olarak atanabilir.

End Point (Uç Nokta): Bir iletişimin bittiği, bir hizmetin sunulduğu adres veya cihaz.

- API'de: Bir işlemin yapıldığı URL.
- Ağda: Ağa bağlı olan son cihaz (telefon, bilgisayar).

Tuple (demet): Python'da bir sıralı ve değiştirilemez (immutable) veri koleksiyonudur. Parantezler () içine yazılır.

_ (alt tire) kullanılmayan alanı ifade eder. Örnek kod :

```
_, longitude, latitude = ("abc123", 29.012, 41.856)
#_ ifadesi çöp alan oluşturur
```

Star Unpacking (yıldız açma), Python'da bir dizi (sequence) içindeki elemanları değişkenlere atarken veya fonksiyonlara parametre gönderirken kullanılan kısa ve pratik bir yöntemdir.

- Yıldız (*) operatörü ile kullanılır.
- Birden fazla elemanı tek bir değişkene (liste olarak) atamaya veya fonksiyona parametre olarak göndermeye yarar.
- **Log Parse :** log dosyalarını işlerken veya metin tabanlı verileri ayrıştırırken (parsing) star unpacking'in sağladığı pratik atama ve paketleme özelliğinden kaynaklanır.

```
[ ]  ts, level, *message = ["2025-09-02T10:22:06Z", "WARN", "High", "latency", "on", "pod-7"]
```

```python
numbers = (1, 2, 3, 4, 5)
first, *middle, last = numbers

print(first)   # 1
print(middle)  # [2, 3, 4] (liste olarak)
print(last)    # 5
```

Değişken atamada kullanımı

```python
def sum(a, b, c):
    return a + b + c

values = (1, 2, 3)
result = sum(*values)  # sum(1, 2, 3) ile aynı
print(result)  # 6
```

Fonksiyon çağırırken kullanımı

```python
list1 = [1, 2, 3]
list2 = [4, 5, 6]
merged = [*list1, *list2]  # [1, 2, 3, 4, 5, 6]
```

Listeleri birleştirirken kullanımı

OS kütüphanesi : İşletim sistemi ile etkileşim kurarak dosya ve dizin işlemleri, ortam değişkenlerine erişim ve sistem komutları çalıştırma gibi görevleri gerçekleştirmeye yarar.

- cwd = bulunduğun klasörü gösterir
  - cwd = os.getcwd()
- Mkdir = yeni bir klasör oluşturur
  - os.mkdir()

PATH Kütüphanesi : Dosya ve dizin yollarını nesne tabanlı, güvenli ve platform bağımsız bir şekilde oluşturmak, erişmek ve işlemek için kullanılan modern bir Python kütüphanesidir.

```
[2]  from pathlib import Path
     RAW_DIR = Path(C://Downloads/data/raw) ; Curated_DIR = Path("data/curated")
```

FONKSİYONLAR

- Belirli bir görevi gerçekleştirmek için tekrar kullanılabilir kod blokları oluşturmayı sağlar ve **def** anahtar kelimesi ile tanımlanır.

```
def selam(isim="misafir"):
  print(f"Merhaba {isim}...")
```

```
[ ]  def topla(a: int, b: int) -> int: #-> int bu yapı sayı döndürüleceğini belirtiyor
       return a+b
```

- Tek yıldız (*) argüman [**args**], Çift yıldız (**) key wargs [**kwargs**] olur.
- Args : Gelen argümanları bir tuple içinde toplar.
- Kwargs : Gelen argümanları bir sözlük (dictionary) içinde toplar.

```
def siparis (*urunler, **detaylar):
  print("Ürünler" , urunler)
  print("Detaylar", detaylar)
```

```python
def bilgileri_goster(isim, *args, **kwargs):
    print(f"İsim: {isim}")
    print(f"Ek bilgiler: {args}")
    print(f"İletişim: {kwargs}")

bilgileri_goster("Ayşe", "Mühendis", "Python", email="ayse@mail.com", tel="0555")
# Çıktı:
# İsim: Ayşe
# Ek bilgiler: ('Mühendis', 'Python')
# İletişim: {'email': 'ayse@mail.com', 'tel': '0555'}
```

- **Append**, listeye yeni eleman ekler.
- **Hata Yönetimi :** Kodun çalışması sırasında oluşabilecek istisnaları (exceptions) yakalayarak programın hatada bile kontrollü şekilde çalışmasını sağlar.

```
def sayiya_cevir(metin):
  try :
    return int(metin)
  except ValueError:
    return None
```

- **Generator ile Akış Yönetimi :** Bellek verimliliği sağlayan, talep üzerine değer üreten (**yield**) ve büyük veri kümeleri veya sonsuz akışlarla çalışmada etkili olan fonksiyonlardır.

```python
#nums = [1 2 3 ... 999]
def only_even(nums : Iterable[int]):
  for n in nums:
    if n %2 == 0:
      yield n
```

- **Dekoratör :** Mevcut bir fonksiyonu veya sınıfı, onun kodunu değiştirmeden, sarmalayarak (wrap) yeni davranışlar eklemeye yarayan bir tasarım desenidir.

```python
def logla(fonk): #ön işlem ve son işlem yapısı
    def sarmal (*a, **k):
        print("Çalışıyor...")
        return fonk(*a, **k)
      return sarmal

#Bir fonksiyon çalıştırılmadan önce "Çalışıyor..." mesajı yazdıran basit bir loglama (kayıt tutma) dekoratörü oluşturur.
```

- Performans için veri yolunu input/output noktalarını minimumda tutun.
- Sık sık çağırılan bir fonksiyon varsa **functools.lru_cache** kullanımı faydalı olur.
- Büyük listeleri kopyalamak yerine iterasyon ve dekoratör kullanımı faydalı olur.

Wrap ve Metadata ilişkisi : wrap, orijinal fonksiyonu sarmalayıp korurken; metadata (isim, docstring gibi fonksiyon kimlik bilgileri, işlem sırasında kullanılmayan bilgiler) bu işlem sırasında kaybolabilir, bu nedenle *functools.wraps* dekoratörü ile metadata'nın korunması sağlanır.

```python
from functools import wraps

def logged (func) :
#dekoratör fonksiyon haline gelir. İçerisine başka bir fonksiyon aldığı için
  @wraps(func) #bunun altına wrap uygulanır, logged func. ilgilendirmiyor.
  def wrapper(*a, **kw):
    print(f"[CALL] {func.__name__} a={a} kw = {kw}")
    #fonksiyon cağırıldığında önce log basar. örn : add(3,4) çağrısında şu yazılır :
    return func(*a, **kw) # asıl fonksiyonu çağırır
  return wrapper # log fonksiyonu süslenmiş haliyle (wrapper) geri döndürür.

  @logged
  def add(a,b) : return a + b
```

Bu kod, `logged` adında bir dekoratör fonksiyonu tanımlamaktadır. Dekoratörler, Python'da fonksiyonları veya metodları dinamik olarak değiştirmek için kullanılan bir tasarım desenidir. `logged` dekoratörü, kendisine parametre olarak verilen fonksiyonun çağrıldığı bilgisini ve aldığı argümanları konsola yazdırarak bir loglama işlevi gerçekleştirir.

Dekoratörün içinde `functools.wraps` decorator'ı kullanılmıştır; bu, sarmalanan fonksiyonun meta verilerinin (isim, docstring gibi) korunmasını sağlar ve hata ayıklamayı kolaylaştırır. `wrapper` fonksiyonu, dekoratörün uygulandığı fonksiyonun argümanlarını alır (`*args` ve `**kwargs`), fonksiyon çağrılmadan önce bir log mesajı yazdırır ve ardından orijinal fonksiyonu çağırarak sonucu döndürür.

Kodun son kısmında, `@logged` dekoratörü `add` fonksiyonuna uygulanmıştır. Bu sayede `add` fonksiyonu her çağrıldığında, fonksiyonun ismi ve aldığı parametreler konsola yazdırılacak, ardından toplama işlemi gerçekleştirilecektir. Örneğin, `add(3, 4)` çağrısı `[CALL] add a=(3, 4) kw = {}` çıktısını üretecek ve ardından 7 sonucunu döndürecektir.