Modüller ve Paketler

Table of contents

Bölüm 5: Modüller ve Paketler - Python'un Hazineleri

Modüller, Python'da kodun tekrar kullanılabilirliğini ve organizasyonunu artırmak için kullanılan güçlü araçlardır. Bir modül, fonksiyonlar, sınıflar ve değişkenler gibi Python kodlarını içeren bir dosyadır. Paketler ise, birden fazla modülü bir araya getiren dizinlerdir. Bu bölümde, modülleri ve paketleri nasıl kullanacağımızı, Python'un bazı popüler modüllerini ve paket yönetimini öğreneceğiz.

5.1 Modül Nedir? Neden Kullanılır?

Modüller, Python kodunuzu düzenlemenize ve tekrar kullanılabilir hale getirmenize yardımcı olur. Aynı kodu farklı projelerde tekrar tekrar yazmak yerine, bir modülde bir kez tanımlayabilir ve istediğiniz kadar projede kullanabilirsiniz.

Modül kullanmanın bazı avantajları:

- **Kodun organize edilmesi:** Modüller, kodunuzu mantıklı parçalara ayırmanıza olanak tanır, böylece daha okunaklı ve anlaşılır hale gelir.
- **Kod tekrarı:** Bir modüldeki fonksiyonları ve sınıfları farklı programlarda tekrar tekrar kullanabilirsiniz, bu da zamandan ve emekten tasarruf sağlar.
- **Ad çakışmalarını önleme:** Farklı modüllerde aynı ada sahip fonksiyonlar veya sınıflar tanımlayabilirsiniz, çünkü modüller kendi ad alanlarını oluşturur.

5.2 Modülleri İçe Aktarma (Import)

Bir modülü kullanmak için öncelikle onu `import` anahtar kelimesiyle içe aktarmanız gerekir. İçe aktarma işlemi için üç farklı yöntem kullanabilirsiniz.

5.2.1 Temel İçe Aktarma

```
import math

print(math.pi)  # Çıktı: 3.141592653589793
```

Bu yöntemle, modüldeki tüm fonksiyonlar, sınıflar ve değişkenler kullanılabilir hale gelir. Ancak, bunlara erişmek için `modul_adi.nitelik` şeklinde modül adını kullanmanız gerekir.

5.2.2 Belirli Bir Fonksiyonu veya Değişkeni İçe Aktarma

```
from math import pi

print(pi)  # Çıktı: 3.141592653589793

from math import sqrt

print(sqrt(25))  # Çıktı: 5.0
```

Bu yöntemle, sadece istediğiniz fonksiyon veya değişkenleri içe aktarırsınız. Bu, gereksiz yere tüm modülü belleğe yüklemekten kaçınmanızı sağlar.

5.2.3 Bir Modülü Takma Adla İçe Aktarma

```python
import math as m

print(m.pi)  # Çıktı: 3.141592653589793
```

Bu yöntemle, modüle bir takma ad verirsiniz ve bu takma ad ile modüldeki niteliklere erişebilirsiniz. Bu, özellikle uzun modül adlarıyla çalışırken kodun okunabilirliğini artırır.

5.3 Popüler Python Modülleri

Python'da birçok yerleşik modül bulunur. Bu modüller, çeşitli görevler için hazır fonksiyonlar ve sınıflar sağlar.

5.3.1 `math` Modülü: Matematiksel İşlemler

`math` modülü, matematiksel fonksiyonlar ve sabitler içerir.

```python
import math

print(math.sin(math.pi / 2))  # Çıktı: 1.0 -> sin(90)
print(math.sqrt(25))        # Çıktı: 5.0 -> 25'in karekökü
print(math.log10(100))       # Çıktı: 2.0 -> 100'ün 10 tabanında logaritması
print(math.pow(2, 3))        # Çıktı: 8.0 -> 2 üzeri 3
```

5.3.2 `random` Modülü: Rastgele Sayı Üretimi

`random` modülü, rastgele sayılar üretmek için kullanılır.

```python
import random

print(random.random())       # Çıktı: 0.725238067631986 (0 ile 1 arasında rastgele bir ondali
print(random.randint(1, 10))    # Çıktı: 5 (1 ile 10 arasında (1 ve 10 dahil) rastgele bir ta
```

- **`random.choice(dizi)`:** Verilen diziden (liste, demet, string vb.) rastgele bir eleman seçer.

```python
import random

liste = ["elma", "armut", "muz"]
print(random.choice(liste))  # Çıktı: armut (Rastgele bir meyve)
```

```
metin = "Merhaba"
print(random.choice(metin)) # Çıktı: a (rastgele harf)
```

- **`random.shuffle(dizi)`:** Verilen diziyi (liste) **yerinde** rastgele karıştırır.

```
import random

liste = [1, 2, 3, 4, 5]
random.shuffle(liste)
print(liste)  # Çıktı: [3, 2, 5, 1, 4] (Rastgele karıştırılmış liste)
```

- **`random.seed(değer)`:** Rastgele sayı üretecini başlatmak için kullanılır. Aynı seed değeri ile başlatılan rastgele sayı üreteci, her zaman aynı sayı dizisini üretir. Eğer `seed()` fonksiyonuna bir değer verilmezse, rastgele sayı üreteci işletim sisteminden aldığı rastgele bir değerle başlatılır.

```
import random

random.seed(42) # seed değerini 42 olarak ayarlar
print(random.random()) # Çıktı: 0.639426...

random.seed(42) # seed değerini tekrar 42 olarak ayarlar
print(random.random()) # Çıktı: 0.639426... (aynı çıktı)


random.seed(10)
print(random.choice(liste))  # Çıktı: elma
random.seed(10)
print(random.choice(liste)) # Çıktı: elma (aynı çıktı)
```

5.3.3 `datetime` Modülü: Tarih ve Saat İşlemleri

- Tarih ve saatlerle çalışmak için kullanılır.

```
import datetime

bugun = datetime.date.today()
print(bugun)                    # Çıktı: 2024-04-03 (Bugünün tarihi)

su_an = datetime.datetime.now()
print(su_an)                  # Çıktı: 2024-04-03 18:15:30.123456 (Şu anki tarih ve saat)
```

```
dogum_gunu = datetime.date(1990, 1, 1)
print(dogum_gunu)                  # Çıktı: 1990-01-01

# Tarih formatlama
tarih_str = bugun.strftime("%d/%m/%Y") # 03/04/2024
print(tarih_str)

zaman_str = su_an.strftime("%H:%M:%S") # 18:15:30
print(zaman_str)
```

5.3.4 `os` Modülü: Dosya ve Dizin İşlemleri

- İşletim sistemiyle etkileşim kurmamızı sağlayan fonksiyonlar içerir.

```
import os

print(os.getcwd())      # Çıktı: /home/kullanici/proje (Geçerli çalışma dizini)

os.chdir("/home/kullanici/Belgeler") # Geçerli dizini değiştirir.
print(os.getcwd())      # Çıktı: /home/kullanici/Belgeler

print(os.listdir("."))  # Çıktı: ['dosya1.txt', 'dosya2.txt', 'dizin1'] (Geçerli dizindeki d

os.mkdir("yeni_dizin")  # yeni_dizin adında bir dizin oluşturur
os.rename("eski.txt", "yeni.txt")  # Dosya adını değiştirir

print(os.path.exists("dosya.txt")) # Çıktı: True (Dosya varsa) / False
print(os.path.isfile("dosya.txt")) # Çıktı: True (Dosya ise) / False
print(os.path.isdir("dizin"))  # Çıktı: True (Dizin ise) / False
print(os.path.getsize("dosya.txt")) # Çıktı: 1024 (Dosya boyutu, bayt cinsinden)


os.remove("yeni.txt") # yeni.txt adlı dosyayı siler.
# os.rmdir("klasör_adi") # Belirtilen klasörü siler
# os.makedirs("dizin1/dizin2") # Birden fazla iç içe klasör oluşturur.
```

5.3.5 Ortam Değişkenlerine Erişme

- `os` modülü, ortam değişkenlerine erişmek için de kullanılabilir.
- `os.environ`: Ortam değişkenlerini içeren bir sözlük.

- `os.getenv(isim, varsayılan_değer)`: Belirtilen isimdeki ortam değişkeninin değerini döndürür. Değişken yoksa, `varsayılan_değer`'i döndürür.

```
import os

# Tüm ortam değişkenlerini yazdır
print(os.environ)

# USER ortam değişkeninin değerini al
kullanici_adi = os.getenv("USER") # Linux/macOS için
#kullanici_adi = os.getenv("USERNAME") # Windows için
print(f"Kullanıcı adı: {kullanici_adi}")

# TEMP ortam değişkeninin değerini al, yoksa varsayılan değeri kullan
temp_dizin = os.getenv("TEMP", "/tmp")
print(f"Geçici dizin: {temp_dizin}")
```

5.4 Her Python Dosyası Bir Modüldür!

- Kendi Python dosyalarınızı oluşturarak modül olarak kullanabilirsiniz.
- Modül dosyanızın adı, `modul_adi.py` gibi olmalıdır.
- Modülünüzü `import modul_adi` şeklinde içe aktarabilirsiniz.

Örnek:

`modul_ornegi.py` dosyasına aşağıdaki kodları yazın:

```
def selamla(isim):
    print(f"Merhaba, {isim}!")

def topla(a, b):
    return a + b
```

Başka bir Python dosyasında (örneğin, `ana_program.py`) bu modülü içe aktarın ve fonksiyonları kullanın:

```
import modul_ornegi

modul_ornegi.selamla("Ali") # Çıktı: Merhaba, Ali!
sonuc = modul_ornegi.topla(5, 3)
print(sonuc) # Çıktı: 8
```

5.5 `__name__` Değişkeni ve Önemi

- Her Python dosyasının özel bir `__name__` değişkeni vardır.
- Dosya doğrudan çalıştırılırsa, `__name__` değişkeninin değeri `"__main__"` olur.
- Dosya bir modül olarak içe aktarılırsa, `__name__` değişkeninin değeri modülün adı olur.

> **Note**
>
> Bu özellik, test kodlarını sadece modül doğrudan çalıştırıldığında çalıştırmak, modül içe aktarıldığında ise çalıştırılmamasını sağlamak için kullanılabilir.

Örnek:

```
# dosya_adi.py

def fonksiyon():
    print("Fonksiyon çalıştı.")

if __name__ == "__main__":
    print("Dosya doğrudan çalıştırıldı.")
    fonksiyon()
```

Çıktı (Doğrudan çalıştırıldığında):

```
Dosya doğrudan çalıştırıldı.
Fonksiyon çalıştı.
```

Başka bir dosyadan içe aktarıldığında:

```
import dosya_adi

print(dosya_adi.__name__) # Çıktı: dosya_adi
```

5.6 Modülleri Keşfetmek: `dir()` Fonksiyonu

- Bir nesnenin (modül, sınıf vb.) sahip olduğu tüm nitelikleri (attributes) ve metotları listeler.

```
import math

print(dir(math)) # math modülündeki tüm nitelikleri ve metotları listeler. Örnek çıktı: ['__
```

```
liste = [1, 2, 3]
print(dir(liste)) # liste nesnesinin tüm niteliklerini ve metotlarını listeler. Örnek çıktı:
```

5.7 Paketler (Packages)

Paketler, birlikte kullanılan birden fazla modülü bir arada tutan dizinlerdir. Paketler, kodun daha da organize edilmesini sağlar. Bir paketin içinde, başka paketler ve modüller de bulunabilir.

Örnek Senaryo

Bir oyun geliştirdiğinizi düşünün. Oyununuzun grafikleri, sesleri, oynanışı ve yapay zekası için farklı modüller kullanabilirsiniz. Bu modülleri bir `oyun` paketi altında toplayabilirsiniz. `oyun` paketi içinde, `grafik`, `ses`, `oynanis` ve `yapay_zeka` gibi alt paketler olabilir.

5.7.1 Paket Oluşturma

- Bir dizin oluşturun ve içine `__init__.py` adında (boş da olabilir) bir dosya koyun.
- Bu dizin artık bir paket olarak kabul edilir.
- Pakete modüller eklemek için, `.py` uzantılı dosyaları bu dizine ekleyin.

Örnek Paket Yapısı:

```
geometri/
  __init__.py
  daire.py
  dikdortgen.py
```

```
# geometri/daire.py
def alan(yaricap):
    # ... Dairenin alanını hesaplayan kodlar ...

# geometri/dikdortgen.py
def alan(kisa_kenar, uzun_kenar):
    # ... Dikdörtgenin alanını hesaplayan kodlar ...
```

```
# ana_program.py

from geometri import daire

daire_alani = daire.alan(5)
```

```
# veya

from geometri.dikdortgen import alan as dikdortgen_alanı

dikdortgen = dikdortgen_alanı(4, 6)

# veya

import geometri.daire

daire_alani = geometri.daire.alan(5)
```

5.8 Paket Yönetimi: `pip`

- `pip`, Python paketlerini yüklemek, yönetmek ve kaldırmak için kullanılan bir araçtır.

```
pip install requests
pip uninstall requests
pip list  # Kurulu paketleri listeler
pip show requests # request paketinin bilgilerini gösterir.
pip freeze > requirements.txt # Projenin bağımlılıklarını dosyaya kaydeder
```

5.8.1 Requirements.txt kullanımı

- Bir projede kullanılması gerekli paketlerin listesini `requirements.txt` dosyasına kaydedin.
- Bu dosyayı kullanarak, proje bağımlılıklarınızı kolayca yönetebilir ve farklı ortamlarda aynı paketleri kurabilirsiniz.

```
pip install -r requirements.txt
```

requirements.txt örneği:

```
Django==3.2.9
requests==2.26.0
numpy>=1.21.0
Flask>=2.0,<3.0
pandas==1.3.3
gunicorn==20.1.0
```

5.9 Sanal Ortamlar (Virtual Environments)

- Farklı projeler için izole Python ortamları oluşturmamızı sağlar.
- Paket çakışmalarını önler.

Senaryo:

Diyelim ki iki farklı web projesi üzerinde çalışıyorsunuz: Proje A ve Proje B. Proje A, Django 2.2 sürümünü kullanırken, Proje B, Django 3.1 sürümünü kullanıyor. Her iki projeyi aynı anda geliştirmek istiyorsanız ve sisteminizde tek bir Python sürümü varsa, iki farklı sanal ortam oluşturmanız gerekir. Böylece her proje, kendi bağımlılıklarına (kütüphane sürümleri) sahip olur ve birbirini etkilemez.

Oluşturma:

```
python3 -m venv .venv  # .venv adında bir sanal ortam oluşturur
```

Aktifleştirme:

- **Linux/macOS:** `source .venv/bin/activate`
- **Windows:** `.venv\Scripts\activate`

Devre Dışı Bırakma:

```
deactivate
```

5.10 `requests` Kütüphanesi

- HTTP istekleri göndermek için kullanılan popüler bir kütüphanedir. Web sitelerinden veri almak veya web servisleriyle etkileşim kurmak için kullanılır.

Kurulum:

```
pip install requests
```

Örnek:

```
import requests

response = requests.get("https://www.example.com")
print(response.status_code)  # Çıktı: 200 (Başarılı istek)
print(response.text) # Çıktının uzunluğuna göre değişir. Web sitesinin kaynak kodunu döndürü

response = requests.get("https://jsonplaceholder.typicode.com/todos/1")
print(response.json()) # {'userId': 1, 'id': 1, 'title': 'delectus aut autem', 'completed': F
```