Uygulama Programı Arayüzleri (API - Application Program Interfaces) hakkında konuşuyoruz. API, iki yazılım parçasının birbirleriyle iletişim kurmasını sağlar. Örneğin, kendi programınız var, bazı verileriniz var ve diğer yazılım bileşenleriyle iletişim kurmak için API'yi kullanıyorsunuz. API, bir işlev gibi çalışır; nasıl çalıştığını bilmenize gerek yok, sadece girdi (input) ve çıktıları (output) bilmeniz yeterlidir.

Bir örnek vermek gerekirse, Pandas kütüphanesi (library) bir dizi yazılım bileşeni içerir. Verilerinizi işlemek için Pandas API'sini kullanarak diğer yazılım bileşenleriyle iletişim kurarsınız. Örneğin, bir sözlük (dictionary) oluşturduğunuzda ve ardından bu sözlüğü Pandas'ın DataFrame yapıcısı (constructor) ile bir Pandas nesnesine dönüştürdüğünüzde, bu API dilinde bir "örnek" (instance) oluşturmuş olursunuz. DataFrame ile API'yi kullanarak verilerinizi işleyebilir ve analiz edebilirsiniz.

Bu videoda HTTP protokolü hakkında konuşuyoruz. HTTP, web üzerinden bilgi transferi için genel bir protokoldür. Bu, birçok türde REST API'yi (Representational State Transfer Application Programming Interface) içerir. REST API'ler, bir istemcinin (client) sunucuya (server) bir istek (request) göndermesiyle çalışır. İstek, genellikle bir HTTP mesajı (HTTP message) aracılığıyla iletilir ve bu mesaj genellikle bir JSON dosyası (JSON file) içerir.

Örnek vermek gerekirse, bir web sayfasını ziyaret ettiğinizde, tarayıcınız sunucuya bir HTTP isteği gönderir. Sunucu, varsayılan olarak "index.html" dosyasını bulmaya çalışır. Eğer istek başarılı olursa, sunucu istemciye bir HTTP yanıtı (HTTP response) gönderir. Bu yanıt, kaynağın türü (type of resource), uzunluğu (length of resource) ve diğer bilgileri içerir.

HTTP Protokolü ve Requests Kütüphanesi: Basit Bir Açıklama

Bu videoda, HTTP (Hypertext Transfer Protocol) protokolünü ve Python'daki requests kütüphanesini öğrendik. Requests, HTTP/1.1 isteklerini kolayca göndermemizi sağlayan bir Python kütüphanesidir. Örneğin, bir GET isteği (GET request) yaparak bir web sitesinden veri alabiliriz. Bu isteği yaparken, bir yanıt nesnesi (response object) alırız. Bu nesne, isteğin durumu (status) gibi bilgileri içerir. Eğer istek başarılıysa, durum kodu (status code) 200 olur, bu da "OK" anlamına gelir.

Örnek: GET İsteği

Bir GET isteği yapmak için şu şekilde bir kod yazabiliriz:

```
import requests


response = requests.get('http://www.ibm.com')

print(response.status_code)  # Durum kodunu yazdırır
```

Bu kodda, requests.get() fonksiyonu ile belirtilen URL'ye bir GET isteği gönderiyoruz. Yanıt nesnesi response içinde, isteğin durumu ve yanıt başlıkları (headers) gibi bilgilere ulaşabiliriz. Örneğin, yanıt başlıklarını görmek için response.headers kullanabiliriz.

HTML (Hypertext Markup Language) ile web scraping (web'den veri çekme) konusunu ele alıyoruz. Web sayfalarında, emlak fiyatları veya kodlama sorularının çözümleri gibi birçok faydalı veri bulunmaktadır. Örneğin, Wikipedia, dünyanın bilgilerini barındıran bir kaynaktır. HTML'i anladığınızda, Python kullanarak bu bilgileri çıkarabilirsiniz.

HTML, bir web sayfasının yapısını belirleyen bir dizi "tag" (etiket) içerir. Örneğin, bir web sayfasında "h3" etiketi, başlıkları belirtirken, "p" etiketi paragrafları temsil eder. HTML ağaçları (HTML Trees) kavramı, bu etiketlerin nasıl hiyerarşik bir yapı oluşturduğunu anlamamıza yardımcı olur. Örneğin, bir "html" etiketi, "head" ve "body" etiketlerini içerir. "Head" etiketi, sayfanın meta bilgilerini barındırırken, "body" etiketi sayfanın görünen içeriğini tutar.

Web scraping, bir web sitesinden otomatik olarak bilgi çıkarmak için kullanılan bir süreçtir. Bu sayede, yüzlerce veri noktasını analiz etmek için saatlerce manuel olarak bilgi kopyalamak yerine, birkaç dakikada gerekli verileri elde edebilirsiniz.

Web scraping'e başlamak için biraz **Python** kodu ve iki modül olan **Requests** ve **Beautiful Soup**'a ihtiyacımız var. Örneğin, bir **National Basketball League** (NBA) oyuncusunun adını ve maaşını bulmak istiyorsanız, önce BeautifulSoup'u içe aktarıyoruz. Web sayfasının HTML'sini bir dize olarak saklayıp, bunu Beautiful Soup yapıcısına geçiriyoruz. Bu işlem, HTML'yi ağaç benzeri bir veri yapısı olarak temsil eden bir Beautiful Soup nesnesi (soup) oluşturur. Bu nesne ile HTML'yi daha kolay bir şekilde analiz edebiliriz.

Bu videoda, farklı dosya formatlarını tanımlamayı, basit programlar yazarak verileri okumayı ve çıkarmak için gereken Python kütüphanelerini listelemeyi öğreneceksiniz. Veri toplarken, CSV, XML ve JSON gibi birçok farklı dosya formatıyla karşılaşacaksınız. Bu dosyaları okumak için Python'un önceden tanımlanmış kütüphanelerini kullanmak süreci kolaylaştırır.

Örneğin, bir dosya adı gördüğünüzde, sonundaki uzantı (extension) dosyanın türünü belirtir. "FileExample.csv" gibi bir dosya adı gördüğünüzde, bunun bir "csv" dosyası olduğunu bilirsiniz. Python'da verileri okumak için ilk olarak "Pandas" kütüphanesini kullanmayı öğreniyoruz. Bu kütüphaneyi kodun başında içe aktararak (import) farklı dosya türlerini kolayca okuyabiliriz. Örneğin, bir CSV dosyasını okumak için şu şekilde bir kod yazabiliriz:

```
import pandas as pd

df = pd.read_csv('FileExample.csv')

print(df)
```

Bu kodda, "pd" kısaltması Pandas kütüphanesini temsil eder. "read_csv" fonksiyonu, CSV dosyasını okuyarak verileri bir DataFrame (veri çerçevesi) olarak döndürür. Eğer dosyada başlık (header) yoksa, ilk satır başlık olarak alınır. Bunu düzeltmek için "columns" özelliğini ekleyerek verileri daha düzenli bir şekilde görüntüleyebiliriz.

Başka bir dosya formatı olan JSON, dil bağımsız bir veri formatıdır ve Python sözlüklerine (dictionary) benzer. JSON dosyasını okumak için "json" kütüphanesini içe aktarıp "load" fonksiyonunu kullanarak dosyayı açabiliriz. XML dosyaları ise "Extensible Markup Language" olarak bilinir ve Pandas bu dosyaları doğrudan okuyamaz. Bunun için "xml" kütüphanesini kullanarak dosyayı ayrıştırmamız (parse) gerekir.