API en Python

1 Wat is een API?

? Leerdoelen

- Je weet wat een API is en waarom het wordt gebruikt.
- Je begrijpt wat de termen **endpoint**, **request** en **response** betekenen.
- Je kunt uitleggen wat het verschil is tussen een API en een database.
- Je kunt een eenvoudig voorbeeld van een API-aanvraag in Python uitvoeren.

? Uitleg

Een **API** (Application Programming Interface) is een soort “tussenpersoon” tussen twee programma’s. Met een API kan jouw programma data opvragen of versturen naar een andere dienst — zonder dat je hoeft te weten hoe die dienst precies werkt.

Vergelijk het met een **ober in een restaurant**: jij (de klant) zegt wat je wilt, de ober geeft het door aan de keuken, en brengt het resultaat terug. De ober is hier de API.

Voorbeeld

Stel: je wilt in je eigen app het actuele weer laten zien. Je hoeft niet zelf alle meetstations te bouwen — je vraagt het gewoon op via de OpenWeatherMap-API.

Als jij bijvoorbeeld deze URL opent in je browser:

```
https://api.open-meteo.com/v1/forecast?latitude=52.37&longitude=4.90¤t_weather=true
```

Dan krijg je (ongeveer) dit te zien:

```
{
  "latitude": 52.37,
  "longitude": 4.90,
  "generationtime_ms": 0.392,
```

```
  "current_weather": {
    "temperature": 14.3,
    "windspeed": 8.1,
    "weathercode": 3
  }
}
```

De API geeft gegevens terug in het formaat **JSON** (JavaScript Object Notation). Dat is de “taal” waarin programma’s gegevens met elkaar uitwisselen. Python kan JSON direct begrijpen.

Hoe werkt het technisch?

1. Je stuurt een **request** (verzoek) naar de API met een bepaalde URL (het **endpoint**).
2. De server voert iets uit (zoals data ophalen uit een database).
3. De server stuurt een **response** (antwoord) terug in JSON-vorm.

Een API is dus eigenlijk een manier om met een ander systeem te “praten” via het internet.

? Voorbeeld in Python

Met Python kun je heel makkelijk gegevens van een API ophalen met de `requests`-bibliotheek.

```
import requests

# Stuur een GET-verzoek naar de API
url = "https://api.open-meteo.com/v1/forecast?latitude=52.37&longitude=4.90¤t_weather=true"
response = requests.get(url)

# Zet de JSON-tekst om in een Python-object (dictionary)
data = response.json()

# Lees de temperatuur uit
temperatuur = data["current_weather"]["temperature"]

print(f"De temperatuur in Amsterdam is {temperatuur}°C")
```

□□ **Opmerking:** als je de foutmelding krijgt dat `requests` niet bestaat, installeer het pakket via de terminal met:

```
pip install requests
```

?? Opdracht

1. Installeer de `requests`-bibliotheek (indien nodig).
2. Maak een nieuw Python-bestand `weer.py`.
3. Kopieer de bovenstaande code en voer het programma uit.
4. Pas het script aan zodat ook de `windspeed` wordt afgedrukt.
5. Voeg een eigen stad toe door de coördinaten aan te passen (zoek op via Google Maps).

? Reflectie

- Wat is het voordeel van een API ten opzichte van zelf een database bijhouden?
- Hoe weet jouw Python-script wat de temperatuur is, zonder dat jij iets invoert?
- Wat zou er gebeuren als de URL fout is of de server offline?

? Inleveren

- Lever het Python-bestand in (`weer-<jouwnaam>.py`).
- Voeg een screenshot toe van de uitvoer in je terminal waarin de temperatuur en windsnelheid worden getoond.

? Verdieping

Voor studenten die sneller klaar zijn:

- Lees de JSON-data volledig uit en toon alle waarden van `current_weather`.
- Laat het programma elke 10 seconden opnieuw de temperatuur ophalen en bijwerken (gebruik `time.sleep(10)`).
- Probeer een andere API, bijvoorbeeld de PokéAPI of de JSONPlaceholder.

2 JSON begrijpen en gebruiken

? Leerdoelen

- Je begrijpt wat JSON is en waarom het wordt gebruikt bij API’s.
- Je kunt JSON-data lezen en omzetten naar Python-objecten (dictionaries en lijsten).
- Je kunt gegevens uit JSON afdrukken met een Python-script.
- Je kunt een JSON-bestand aanpassen en uitbreiden.

? Uitleg

Bij het werken met een API krijg je bijna altijd data terug in het formaat **JSON** – dat staat voor *JavaScript Object Notation*. JSON is een gestandaardiseerde manier om gegevens te structureren, zodat computers ze makkelijk kunnen lezen en versturen.

JSON lijkt een beetje op Python-code, want het gebruikt accolades `{ }` en rechte haken `[ ]` voor lijsten en objecten.

Voorbeeld van JSON

```
{
  "personen": [
    {
      "naam": "Alice",
      "leeftijd": 25,
      "stad": "Amsterdam"
    },
    {
      "naam": "Bob",
      "leeftijd": 32,
      "stad": "Rotterdam"
    },
    {
      "naam": "Charlie",
      "leeftijd": 42,
      "stad": "Utrecht"
    }
  ]
}
```

We zien hier een JSON-object met één sleutel (`personen`) die een lijst bevat met drie personen. Elke persoon heeft drie eigenschappen: `naam`, `leeftijd` en `stad`.

JSON in Python lezen

Python heeft standaard de bibliotheek `json` waarmee je JSON-data kunt inlezen of wegschrijven.

```
import json

# JSON-bestand lezen
with open("data.json") as bestand:
    data = json.load(bestand)

# Gegevens verwerken
personen = data["personen"]

for persoon in personen:
    naam = persoon["naam"]
    leeftijd = persoon["leeftijd"]
    stad = persoon["stad"]
    print(f"{naam} is {leeftijd} jaar oud en woont in {stad}.")
```

In deze code:

- `import json` haalt de JSON-bibliotheek binnen.
- `json.load(bestand)` zet de tekst uit het bestand om in een Python-object (dictionary).
- Je kunt daarna gewoon met Python-code door de data heen lopen met een `for`-loop.

JSON en Python-dictionaries

De JSON-structuur lijkt sterk op de manier waarop Python met **dictionaries** werkt:

```
persoon = {
  "naam": "Ahmed",
  "leeftijd": 19,
  "stad": "Amstelveen"
}

print(persoon["naam"])  # Geeft 'Ahmed'
```

Daardoor kun je JSON en Python eenvoudig combineren.

?? Opdracht

1. Maak een nieuw bestand `data.json` en plak daarin de JSON-gegevens uit het voorbeeld hierboven.
2. Maak een Python-bestand `lees_json.py` met de voorbeeldcode.
3. Voer het script uit en controleer of alle namen, leeftijden en steden worden getoond.
4. Voeg een vierde persoon toe aan het JSON-bestand en laat het script opnieuw draaien.
5. Verander de uitvoer zodat het script zinnen afdrukt als:
   “Alice woont in Amsterdam en is 25 jaar oud.”

? Reflectie

- Wat is het voordeel van JSON tegenover bijvoorbeeld een gewoon tekstbestand?
- Waarom is JSON een goed formaat voor data die via internet wordt verstuurd?
- Hoe verschilt JSON van een Python-dictionary?

? Inleveren

- Lever je Python-bestand in: `lees_json-<jouwnaam>.py`.
- Lever ook het aangepaste `data.json`-bestand in.
- Maak een screenshot van je terminal waarin de uitvoer zichtbaar is.

? Verdieping

Voor studenten die sneller klaar zijn of extra willen oefenen:

- Voeg een nieuw veld `"telefoon"` toe aan elk persoon in het JSON-bestand en pas je Python-script aan om dit ook af te drukken.
- Maak een Python-functie `zoek_persoon(naam)` die de gegevens van een specifieke persoon uitprint.
- Laat je programma het aantal personen tellen en dit onderaan afdrukken.
- Extra uitdaging: schrijf de gewijzigde JSON terug naar het bestand met `json.dump()`.

3 Data ophalen van een API (GET)

? Leerdoelen

- Je weet wat een GET-request is en waar het voor dient.
- Je kunt met Python en de `requests`-bibliotheek data ophalen van een echte API.
- Je kunt de JSON-response van een API omzetten naar bruikbare Python-data.
- Je kunt zelf specifieke gegevens uit de response selecteren en tonen.

? Uitleg

In de vorige les heb je geleerd hoe JSON eruitziet en hoe je het kunt lezen in Python. In deze les gaan we een stap verder: we halen live data op van een **echte API** op het internet.

Wat is een GET-request?

Een **GET-request** is het meest gebruikte type verzoek op het web. Wanneer je bijvoorbeeld een website bezoekt, stuur je onbewust een GET-request naar de server om de pagina op te vragen. Bij een API vraag je met zo'n verzoek om gegevens, zoals een lijst met gebruikers, producten of berichten.

De JSONPlaceholder API

Om te oefenen gebruiken we een gratis en veilige test-API: https://jsonplaceholder.typicode.com/

Deze API simuleert een echte webservice met data zoals:

- `/users` → gebruikers
- `/posts` → blogberichten
- `/todos` → takenlijsten

Voorbeeld

```
import requests
import json

# 1□□ Stuur een GET-request naar de API
response = requests.get("https://jsonplaceholder.typicode.com/users")

# 2□□ Controleer of het verzoek is gelukt
if response.status_code == 200:
```

```
    print("□ Data succesvol opgehaald!\n")
else:
    print("□ Er ging iets mis:", response.status_code)

# 3□□ Zet de JSON-response om naar een Python-object
data = response.json()

# 4□□ Toon de naam en e-mail van elke gebruiker
for gebruiker in data:
    naam = gebruiker["name"]
    email = gebruiker["email"]
    print(f"{naam} – {email}")
```

□□ **Opmerking:** Als je de foutmelding krijgt dat `requests` niet is geïnstalleerd, doe dan: `pip install requests`

?? Opdracht

1. Maak een nieuw Python-bestand `api_get.py`.
2. Kopieer de bovenstaande code en voer het programma uit.
3. Controleer of de namen en e-mails van alle gebruikers netjes worden weergegeven.
4. Pas het script aan zodat het ook het `address` toont (straat + stad).
5. Druk aan het eind af hoeveel gebruikers er in totaal zijn opgehaald.

? Reflectie

- Wat is het verschil tussen data ophalen van een lokaal JSON-bestand en via een API?
- Waarom is het belangrijk om de `status_code` van een response te controleren?
- Wat zou er gebeuren als je een verkeerde URL gebruikt of de server offline is?

? Inleveren

- Lever je Python-bestand in: `api_get-<jouwnaam>.py`.
- Voeg een screenshot toe van je terminal waarin de uitvoer zichtbaar is (namen en e-mails).

? Verdieping

Voor studenten die sneller klaar zijn of meer willen begrijpen:

- Laat het programma alleen gebruikers tonen waarvan de `website` eindigt op `.org`.
- Toon de data in een net geformatteerde tabel met `print(f"{naam:25} {email:30} {stad}")`.
- Probeer in plaats van `/users` eens de endpoint `/todos` en toon de eerste 10 taken.
- Gebruik `json.dumps(data, indent=2)` om de volledige response mooi leesbaar af te drukken.

?? Bonusopdracht (extra uitdaging)

Maak een programma dat de langste gebruikersnaam zoekt:

```
langste = ""
for gebruiker in data:
    if len(gebruiker["username"]) > len(langste):
        langste = gebruiker["username"]

print(f"De langste gebruikersnaam is: {langste}")
```

Voeg dit toe aan je script en test of het werkt.

4 filteren, bewerken en combineren via de API

? Leerdoelen

- Je kunt opgehaalde JSON-data filteren op bepaalde voorwaarden.
- Je kunt specifieke onderdelen van de data bewerken of samenvatten.
- Je begrijpt hoe je data uit een API kunt combineren met lokale gegevens.
- Je kunt resultaten op een overzichtelijke manier tonen in de terminal.

? Uitleg

In de vorige les heb je geleerd hoe je gegevens kunt ophalen via een API. Vaak wil je echter niet *alle* gegevens gebruiken, maar alleen de stukken die relevant zijn. Daarvoor kun je de data filteren en bewerken met Python.

Filteren van data

Stel dat we opnieuw data ophalen van de **JSONPlaceholder API**, maar nu willen we alleen de gebruikers zien die in de stad `South Christy` wonen.

```
import requests

response = requests.get("https://jsonplaceholder.typicode.com/users")
data = response.json()

print("Gebruikers uit South Christy:\n")

for gebruiker in data:
    if gebruiker["address"]["city"] == "South Christy":
        print(f"- {gebruiker['name']} ({gebruiker['email']})")
```

In dit voorbeeld gebruiken we een eenvoudige `if`-voorwaarde binnen een `for`-loop om te bepalen wie aan onze voorwaarde voldoet.

Bewerken van data

Soms wil je data herschrijven of samenvatten. Bijvoorbeeld: alle e-mailadressen in hoofdletters tonen.

```
for gebruiker in data:
    email = gebruiker["email"].upper()
    print(f"{gebruiker['name']} – {email}")
```

Combineren van data

Je kunt ook data uit verschillende bronnen combineren. Bijvoorbeeld: we halen een lijst met **posts** op van dezelfde API en koppelen die aan de gebruikers.

```
# Haal gebruikers op
users = requests.get("https://jsonplaceholder.typicode.com/users").json()

# Haal posts op
posts = requests.get("https://jsonplaceholder.typicode.com/posts").json()
```

```
# Combineer: toon hoeveel berichten elke gebruiker heeft geplaatst
for user in users:
    user_id = user["id"]
    user_posts = [p for p in posts if p["userId"] == user_id]
    print(f"{user['name']} heeft {len(user_posts)} berichten geplaatst.")
```

Hier gebruiken we een **list comprehension** om alle berichten te selecteren die bij de juiste gebruiker horen. Dit is een veelgebruikte techniek bij het analyseren van API-data.

?? Opdracht

1. Maak een nieuw Python-bestand `api_filter.py`.
2. Haal met `requests` de gebruikers op via de JSONPlaceholder API.
3. Filter de lijst zodat alleen gebruikers met een e-mailadres dat eindigt op `.org` worden getoond.
4. Toon hun naam, e-mailadres en stad in één nette regel.
5. Print onderaan hoeveel gebruikers aan de voorwaarde voldeden.

? Reflectie

- Wat is het verschil tussen data ophalen en data filteren?
- Welke Python-functies kun je gebruiken om data te bewerken of te tellen?
- Wat gebeurt er als een bepaalde sleutel (zoals `city`) niet bestaat in de data? Hoe zou je dat kunnen opvangen?

? Inleveren

- Lever je Python-bestand in: `api_filter-<jouwnaam>.py`.
- Voeg een screenshot toe van je terminal waarin de gefilterde uitvoer zichtbaar is.

? Verdieping

Voor studenten die extra willen oefenen of hun kennis willen uitbreiden:

- Gebruik de endpoint `/todos` en toon hoeveel taken per gebruiker zijn voltooid.
- Maak een lijst van gebruikers die minder dan 5 taken hebben afgerond.
- Schrijf de resultaten weg naar een nieuw JSON-bestand met `json.dump()`.
- Experimenteer met sorteren: toon gebruikers op alfabetische volgorde of op basis van aantal taken.

?? Bonusopdracht – Data combineren

Combineer de gegevens van twee endpoints: `/users` en `/posts`. Laat per gebruiker zien:

- De naam van de gebruiker.
- Het aantal posts dat hij/zij heeft geschreven.

```
for user in users:
    count = sum(1 for p in posts if p["userId"] == user["id"])
    print(f"{user['name']} – {count} posts")
```

□□ Tip: gebruik `sum()` met een generator in plaats van een hele lijst voor meer efficiëntie.

5 Data verzenden naar een API (POST)

? Leerdoelen

- Je begrijpt het verschil tussen een `GET`- en een `POST`-verzoek.
- Je kunt met Python en de `requests`-library data versturen naar een API.
- Je weet hoe je JSON-data correct opbouwt en verstuurt in een POST-request.
- Je kunt de serverrespons uitlezen en interpreteren.

? Uitleg

Tot nu toe hebben we alleen data opgehaald met **GET**-requests. In deze les leren we hoe we data kunnen *versturen* met een **POST**-request. Dit gebruik je bijvoorbeeld als je een formulier indient, een nieuw record toevoegt aan een database of een bericht plaatst via een webservice.

GET vs POST

Kenmerk	GET	POST
Doel	Data **ophalen**	Data **versturen** of aanmaken
Verzending	Data gaat via de URL	Data gaat via de request-body
Geschikt voor	Lezen van data	Invoeren, updaten of verwijderen
Voorbeeld	`GET /users`	`POST /users`

Een eenvoudige POST-request

We gebruiken opnieuw de test-API **JSONPlaceholder**. Daar kun je zonder risico POST-verzoeken naartoe sturen — de server doet alsof er iets wordt toegevoegd, maar bewaart het niet echt.

```
import requests

# De URL van de API
url = "https://jsonplaceholder.typicode.com/posts"

# De data die we willen verzenden
nieuwe_post = {
    "title": "Mijn eerste API-bericht",
    "body": "Dit is een testbericht dat via Python is verstuurd.",
    "userId": 1
}

# Verstuur de data als JSON
response = requests.post(url, json=nieuwe_post)

# Controleer het resultaat
print("Statuscode:", response.status_code)
print("Serverantwoord:")
print(response.json())
```

 Als alles goed gaat, geeft de server een statuscode `201 Created` terug en toont hij de nieuwe data met een automatisch toegevoegd ID.

Tip: gebruik van headers

Sommige API’s eisen dat je expliciet vermeldt welk datatype je verstuurt. Dat doe je met HTTP-headers:

```
headers = {"Content-Type": "application/json"}
response = requests.post(url, json=nieuwe_post, headers=headers)
```

Verwerking van de response

De API-response is meestal ook JSON. Je kunt die omzetten naar een Python-dict:

```
data = response.json()
print("Nieuwe post-ID:", data["id"])
```

?? Opdracht

1. Maak een nieuw Python-bestand `api_post.py`.
2. Gebruik de URL `https://jsonplaceholder.typicode.com/posts`.
3. Verstuur een nieuw “bericht” met een zelfgekozen titel, tekst en userId.
4. Toon de statuscode en de ID die je van de server terugkrijgt.
5. Controleer of de server de data correct terugstuurt.

? Reflectie

- Wat is het belangrijkste verschil tussen GET en POST?
- Waarom is een statuscode 201 een goed teken?
- Wat zou er kunnen misgaan als je geen headers meestuurt?
- Welke situaties in echte projecten vereisen een POST-request?

? Inleveren

- Lever het Python-bestand in: `api_post-<jouwnaam>.py`
- Voeg een screenshot toe van de terminal waarin de JSON-response zichtbaar is.

? Verdieping

Voor studenten die verder willen experimenteren:

- Probeer ook een `PUT`-verzoek om een bestaande post te *wijzigen*:

```
requests.put("https://jsonplaceholder.typicode.com/posts/1", json={"title":
"Aangepast"})
```

- Probeer een `DELETE`-verzoek en controleer de statuscode (verwacht: 200 of 204).
- Voeg foutafhandeling toe:

```
if response.status_code != 201:
    print("Er ging iets mis:", response.text)
```

- Gebruik `input()` om de gebruiker zelf een titel en bericht te laten invoeren en verstuur dat via de API.

6 Authenticatie en beveiligde API's

? Leerdoelen

- Je begrijpt waarom veel API's beveiligd zijn en authenticatie vereisen.
- Je weet wat een **API-key** en een **Bearer Token** zijn.
- Je kunt met Python een beveiligde API aanroepen met behulp van headers.
- Je kunt foutmeldingen (zoals 401 Unauthorized) herkennen en oplossen.

? Uitleg

Tot nu toe hebben we gewerkt met openbare test-API's zoals **JSONPlaceholder**. In de praktijk zijn de meeste API's echter **afgeschermd**: je hebt een *sleutel* of *token* nodig om toegang te krijgen. Dit voorkomt misbruik en zorgt ervoor dat alleen geautoriseerde gebruikers data kunnen lezen of wijzigen.

Wat is een API-key?

Een API-key is een unieke code (meestal een lange reeks letters en cijfers) die je van de aanbieder van de API ontvangt. De key identificeert jou als gebruiker. Je voegt deze toe aan je request, meestal via de `headers`.

```
import requests
```

```
url = "https://api.openweathermap.org/data/2.5/weather"
params = {"q": "Amsterdam", "appid": "JOUW_API_KEY"}

response = requests.get(url, params=params)
print(response.json())
```

Hier gebruik je de `appid`-parameter die vereist is door de OpenWeatherMap-API. Als de key ongeldig is, krijg je een foutmelding zoals:

```
{
  "cod": 401,
  "message": "Invalid API key"
}
```

Wat is een Bearer Token?

Een **Bearer Token** wordt vaak gebruikt bij modernere API's (zoals van Google, GitHub of Rentman). Je krijgt dit token na het inloggen of registreren, en voegt het toe aan de `Authorization`-header:

```
import requests

url = "https://api.github.com/user"
headers = {
    "Authorization": "Bearer JOUW_ACCESS_TOKEN"
}

response = requests.get(url, headers=headers)
print(response.status_code)
print(response.json())
```

De header vertelt de server: "deze gebruiker is geverifieerd". Als de token niet klopt of verlopen is, krijg je een `401 Unauthorized` of `403 Forbidden` statuscode.

Headers en veiligheid

Headers zijn extra stukjes informatie die met elk verzoek worden meegestuurd. Behalve `Authorization` worden ook vaak gebruikt:

- `Content-Type` – geeft aan welk formaat de body heeft, meestal `application/json`.
- `Accept` – geeft aan welk formaat je als antwoord verwacht.

```
headers = {
    "Authorization": "Bearer JOUW_TOKEN",
    "Content-Type": "application/json",
    "Accept": "application/json"
}
```

Belangrijk: Deel nooit je API-key of token in code die je publiceert. Gebruik in echte projecten bij voorkeur een *.env-bestand* om sleutels veilig op te slaan.

?? Opdracht

1. Maak een bestand `api_secure.py`.
2. Kies een gratis API die authenticatie vereist (bijv. OpenWeatherMap, TMDB of NewsAPI) en vraag een eigen key aan.
3. Maak een GET-verzoek met je API-key of token.
4. Print enkele relevante velden uit het JSON-antwoord (bijv. temperatuur, titel, of naam).
5. Vang foutmeldingen af met `if response.status_code != 200:` en geef een duidelijke melding weer.

? Reflectie

- Waarom werken de meeste professionele API's met authenticatie?
- Wat zijn de risico's als je een API-key in je code laat staan?
- Hoe kun je sleutels veiliger opslaan in een project?
- Wat betekent statuscode 401 versus 403?

? Inleveren

- Lever het bestand in: `api_secure-<jouwnaam>.py`.
- Voeg een screenshot toe van je werkende API-response.

? Verdieping

- Lees in over **OAuth 2.0** – de standaard voor moderne authenticatie bij o.a. Google en Microsoft-API's.
- Gebruik de `requests.Session()`-methode om meerdere beveiligde verzoeken efficiënt te doen met dezelfde token.
- Experimenteer met foutafhandeling: wat gebeurt er als je token verloopt of verkeerd is?
- Probeer in plaats van een GET-request een beveiligde `POST`-aanvraag uit te voeren.

?? Bonusopdracht – Sleutelbeheer

Maak een extra bestand `.env` in dezelfde map met daarin:

```
API_KEY=JOUW_API_KEY_HIER
```

Installeer daarna de library `python-dotenv` en gebruik deze code:

```
from dotenv import load_dotenv
import os, requests

load_dotenv()
api_key = os.getenv("API_KEY")

response =
requests.get(f"https://api.openweathermap.org/data/2.5/weather?q=Haarlem&appid={api_key}")
print(response.json())
```

Zo leer je hoe professionele ontwikkelaars veilig met API-sleutels omgaan.

7 Data visualiseren uit een API (duo-opdracht)

? Leerdoelen

- Je kunt data uit een API ophalen, verwerken en begrijpelijk weergeven.
- Je leert samenwerken aan een klein Python-project in duo's.
- Je gebruikt libraries zoals `requests` en `matplotlib` om data visueel te maken.

- Je kunt uitleggen hoe ruwe API-data wordt omgezet naar bruikbare informatie.

? Uitleg

Tot nu toe hebben we geleerd hoe je gegevens uit een API kunt ophalen en verwerken. Maar data krijgt pas echt waarde als je het **visueel** maakt. Denk aan grafieken, diagrammen of lijsten die trends of verbanden laten zien.

In deze les gaan jullie in duo's werken aan een kleine toepassing die data ophaalt uit een openbare API, zoals:

- Open-Meteo API – weergegevens
- ExchangeRate API – wisselkoersen
- SpaceX API – lanceringen
- DataUSA API – bevolkings- en inkomensdata

Samen ontwerpen jullie een klein Python-script dat deze data ophaalt, verwerkt en visualiseert. Je gebruikt hiervoor **matplotlib** of een eenvoudige `print()`-weergave in tabelvorm.

Voorbeeld

Hieronder een voorbeeld dat de temperatuur van de komende dagen ophaalt uit de Open-Meteo API en weergeeft in een lijngrafiek:

```
import requests
import matplotlib.pyplot as plt

url = "https://api.open-meteo.com/v1/forecast"
params = {"latitude": 52.37, "longitude": 4.9, "daily": "temperature_2m_max", "timezone":
"Europe/Amsterdam"}

response = requests.get(url, params=params)
data = response.json()

dagen = data["daily"]["time"]
temperaturen = data["daily"]["temperature_2m_max"]

plt.plot(dagen, temperaturen, marker='o')
plt.title("Maximale temperatuur in Amsterdam (komende week)")
```

```
plt.xlabel("Datum")
plt.ylabel("Temperatuur (°C)")
plt.grid(True)
plt.show()
```

Dit voorbeeld toont de kern van deze les: data ophalen, selecteren, en weergeven in een grafiek. Jullie mogen zelf een andere dataset of onderwerp kiezen.

?? Duo-opdracht – Bouw je eigen datavisualisatie

□□ Werkvorm

Werk in duo's. Verdeel de rollen als volgt:

- **Student A:** richt zich op het ophalen en verwerken van de data (API & JSON).
- **Student B:** richt zich op de presentatie en visualisatie (grafiek of tabel).

□□ Opdrachtstappen

1. Kies samen een API die jullie interessant vinden (zie de lijst hierboven).
2. Bestudeer de documentatie van de API om te ontdekken hoe je data kunt ophalen.
3. Maak een Python-script dat:
   - Een API-verzoek uitvoert met `requests`.
   - De ontvangen JSON-gegevens filtert tot de belangrijkste velden.
   - De resultaten weergeeft in een **grafiek** of **tabel**.
4. Gebruik `matplotlib` of een alternatief (zoals `prettytable` of gewoon `print()`).
5. Voeg jullie namen en datum bovenaan het script toe als commentaar.

□□ Suggesties voor onderwerpen

- Vergelijk de gemiddelde temperatuur van 2 steden.
- Toon het aantal SpaceX-lanceringen per jaar.
- Laat de wisselkoers Euro–Dollar van de laatste 7 dagen zien.
- Laat zien hoe de bevolkingsgroei van een land zich ontwikkelt.

? Reflectie

- Wat was het moeilijkste aan het ophalen of verwerken van de data?
- Welke rolverdeling werkte het beste binnen jullie duo?
- Hoe kun je deze aanpak (API + visualisatie) gebruiken in een webapplicatie?
- Wat zou je verbeteren als je meer tijd had?

? Inleveren

- Lever één gezamenlijke map in met:
  - `visualisatie-<jullie-namen>.py`
  - Een schermafbeelding van de uitvoer (grafiek of tabel).
  - Een kort verslag (`reflectie.txt`) van max. 200 woorden over jullie samenwerking.
- Vermeld duidelijk welke API jullie hebben gebruikt.

? Verdieping

- Gebruik meerdere API's tegelijk, bijvoorbeeld temperatuur en luchtvochtigheid, en combineer ze in één grafiek.
- Voeg labels, kleuren en een legenda toe aan de grafiek.
- Gebruik de `pandas`-library om de data te filteren of te sorteren.
- Maak een interactieve visualisatie met `plotly` (optioneel).

?? Technische tips

- Installeer matplotlib (indien nodig): `pip install matplotlib`.
- Gebruik `print(response.status_code)` om te controleren of de API werkt (200 = OK).
- Voeg foutafhandeling toe met `try/except` voor betrouwbaardere code.

□□ Samenvatting

In deze les leer je samenwerken aan een mini-project waarin je data uit een echte API omzet naar inzichtelijke visualisaties. Je combineert alles wat je hebt geleerd: API-verzoeken, JSON verwerken, data structureren, en presenteren.

8 Eindproject – API Dashboard (samenwerking)

? Leerdoelen

- Je kunt zelfstandig meerdere API's combineren in één Python-project.
- Je werkt effectief samen binnen een klein team (2 of 3 studenten).
- Je kunt data ophalen, verwerken en presenteren in een begrijpelijk dashboard.
- Je kunt uitleggen hoe jouw applicatie technisch werkt en waarom bepaalde keuzes zijn gemaakt.

? Inleiding

Je hebt in de vorige lessen geleerd hoe je gegevens kunt ophalen van één API en deze kunt visualiseren. In dit eindproject ga je die kennis uitbreiden: je combineert informatie van **meerdere API's** om een klein, nuttig **Python-dashboard** te maken. Dat kan bijvoorbeeld gaan over weergegevens, valutakoersen, nieuws, of sportresultaten.

Je werkt in een klein team (twee of drie studenten) en gebruikt jullie gecombineerde kennis om iets te maken dat meerwaarde heeft: een programma dat gegevens uit verschillende bronnen slim samenbrengt.

Voorbeelden van mogelijke projecten

- **Reisplanner-dashboard:** combineer een weer-API met een valuta-API om reizigers te helpen.
- **Crypto-monitor:** combineer de CoinGecko API met een nieuws-API en toon de trends van de dag.
- **Sportoverzicht:** gebruik een sportdata-API met een tijdzone-API om aanstaande wedstrijden te tonen.
- **Evenementenplanner:** haal evenementen op via Ticketmaster API en combineer met het weer.

?? Eindopdracht – Bouw een API Dashboard

?? Werkvorm

Werk samen in duo’s of in groepen van drie. Verdeel de verantwoordelijkheden duidelijk en noteer deze bovenaan in jullie code. Bijvoorbeeld:

```
# Team: Sarah & Mohammed
# Sarah: Data ophalen en JSON-verwerking
# Mohammed: Visualisatie en foutafhandeling
```

?? Opdrachtstappen

1. Kies samen een thema (weer, geld, sport, reizen, etc.).
2. Zoek twee of meer API’s die relevant zijn voor dat thema.
3. Lees de documentatie en bepaal welke data jullie nodig hebben.
4. Maak een Python-script dat:
   - Data ophaalt via `requests` uit minimaal twee API’s.
   - De gegevens combineert of vergelijkt (bijv. weer + locatie, koers + nieuws).
   - De resultaten weergeeft in een **duidelijke visualisatie** (grafiek of tabel).
5. Test jullie code en voeg foutafhandeling toe met `try/except`.
6. Schrijf een korte uitleg over hoe jullie project werkt en wie wat heeft gedaan.

?? Tip voor de opbouw van jullie script

```
import requests
import matplotlib.pyplot as plt

# --- Instellingen ---
API_1 = "https://api.exchangerate.host/latest"
API_2 = "https://api.coindesk.com/v1/bpi/currentprice.json"

# --- Data ophalen ---
valuta_data = requests.get(API_1).json()
bitcoin_data = requests.get(API_2).json()

# --- Gegevens verwerken ---
```

```
eur_usd = valuta_data["rates"]["USD"]
btc_usd = bitcoin_data["bpi"]["USD"]["rate_float"]

# --- Resultaten tonen ---
print(f"1 EUR = {eur_usd:.2f} USD")
print(f"1 Bitcoin = {btc_usd:.2f} USD")

plt.bar(["Euro", "Bitcoin"], [eur_usd, btc_usd])
plt.title("Vergelijking van valuta- en cryptokoersen")
plt.ylabel("Waarde in USD")
plt.show()
```

Dit voorbeeld combineert twee verschillende API’s (valuta en crypto) in één klein dashboard. Het toont hoe eenvoudig je meerdere databronnen kunt combineren in Python.

? Samenwerken

Goede samenwerking is cruciaal in softwareontwikkeling. Werk efficiënt door gebruik te maken van de volgende strategieën:

- Verdeel de code in logische onderdelen (API 1, API 2, visualisatie, foutafhandeling).
- Gebruik GitHub of deel regelmatig jullie code via Teams of een gezamenlijke map.
- Controleer elkaars code en geef constructieve feedback.
- Schrijf bij elke functie een korte comment die uitlegt wat die doet.

? Reflectie

- Hoe verliep de samenwerking binnen jullie team?
- Welke API’s hebben jullie gekozen en waarom?
- Wat was het lastigste aan het combineren van de data?
- Welke technieken uit eerdere lessen kwamen nu van pas?
- Hoe zou je dit project uitbreiden als je meer tijd had?

? Inleveren

- Lever één map in per groep met:

    - `dashboard-<teamnaam>.py`
    - Een schermafbeelding van de grafiek of console-output
    - `reflectie-<teamnaam>.txt` (max. 250 woorden)
- Vermeld bovenaan de code duidelijk de namen en taken van alle teamleden.
- Lever in via Canvas vóór de aangegeven deadline.

? Verdieping (voor snelle studenten)

- Gebruik drie of meer API's en combineer de resultaten in één visualisatie.
- Gebruik de `pandas`-library om data te sorteren of trends te berekenen.
- Maak een mini-dashboard met `streamlit` zodat de gebruiker zelf keuzes kan maken (stad, valuta, enz.).
- Voeg een cache-systeem toe zodat de API niet steeds opnieuw hoeft te laden.

?? Technische tips

- Gebruik `pip install matplotlib pandas` om extra libraries te installeren.
- Gebruik `print(response.status_code)` om te zien of een API werkt (200 = OK).
- Controleer of alle API's JSON teruggeven; zo niet, gebruik `response.text`.
- Voeg bij elke API een korte uitleg in commentaar (waarvoor dient deze API?).

□□ Samenvatting

In dit eindproject combineer je alles wat je geleerd hebt over API's, JSON, dataverwerking en visualisatie. Je leert niet alleen programmeren, maar ook **samenwerken, plannen en communiceren** zoals echte developers dat doen. Samen bouw je een klein maar krachtig Python-dashboard dat data uit de echte wereld tot leven brengt.

Revision #2
Created 2025-10-19 15:28:29 UTC by Max
Updated 2025-10-19 15:38:57 UTC by Max