Requests

December 13, 2021

1 Requests

1.1 Inleiding

De `requests`-bibliotheek is een elegante en simpele HTTP bibliotheek voor Python. Met `requests` kunnen heel eenvoudig HTTP/1.1-verzoeken verstuurd worden. Het is niet nodig om handmatig queryreeksen aan de URL's toe te voegen of om POST-gegevens te coderen. `Keep-alive` en HTTP-verbindingspooling zijn 100% automatisch, dankzij `urllib3`.

Het ophalen en analyseren van online data wordt overzichtelijk gefaciliteerd dankzij deze bibliotheek:

```
r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
r.status_code
>>> 200
r.headers['content-type']
>>> 'application/json; charset=utf8'
r.encoding
>>> 'utf-8'
r.text
>>> '{"type":"User"...'
r.json()
>>> {'private_gists': 419, 'total_private_repos': 77, ...}
```

> Het verschil tussen `requests` en `urllib2` wordt hier geïllustreerd.

De volgende zaken worden ondersteund: - Keep-Alive & Connection Pooling - International Domains and URLs - Sessions with Cookie Persistence - Browser-style SSL Verification - Automatic Content Decoding - Basic/Digest Authentication - Elegant Key/Value Cookies - Automatic Decompression - Unicode Response Bodies - HTTP(S) Proxy Support - Multipart File Uploads - Streaming Downloads - Connection Timeouts - Chunked Requests - `.netrc` Support

De `requests`-bibliotheek wordt ondersteund door Python 2.7 en Python 3.6+.

Deze tutorial is een vertaling van de getting started guide op https://docs.python-requests.org/

1.2 Installatie

De `requests`-bibliotheek is niet direct beschikbaar na een standaard installatie van Python. Het is daarom vereist om deze bibliotheek achteraf te downloaden en te installeren. Vervolgens wordt de installatie gecontroleerd.

1.2.1 PyPy

De eerste manier om `requests` te installeren is door simpelweg het volgende commando uit te voeren in de opdrachtprompt of terminal:

```
python -m pip install requests
```

1.2.2 Conda

Als gebruik gemaakt wordt door een virtuele omgevingsbeheerder, zoals Anaconda of Miniconda, is het gebruik van het volgende commando aangewezen. Besteed hierbij de nodige aandacht voor het voorbereiden en activeren van de gewenste *environment*:

```
conda install -c anaconda requests
```

1.2.3 Broncode

`requests` wordt nog steeds actief ontwikkeld op Github, waar de meest recente broncode terug te vinden is. Deze code kan gekloond worden uit de publieke *repository*:

```
git clone git://github.com/psf/requests.git
```

De tarball (of zipball, voor Windows) kan ook gedownload worden:

```
curl -OL https://github.com/psf/requests/tarball/main
```

Deze code kan toegevoerd worden aan bestaande Python pakketten, maar kan ook als *site-packages easily* geïnstalleerd worden:

```
cd requests python -m pip install .
```

1.2.4 Installatie en versie contoleren

Voer de volgende code uit om de installatie van `requests` te controleren:

```
python -c "import requests; print(requests.__version__)"
```

Opdracht: Installeer requests en controleer dat de bibliotheek correct geïnstalleerd is.

[ ]:
```
## UW CODE HIER ##
```

1.3 Aan de slag

In de volgende sectie van het gebruik van `requests` geïllustreerd worden aan de hand van enkele voorbeelden. Hierbij wordt dieper ingegaan op verschillende functies, argumenten en output. Binnen de GeoICT worden dergelijke zaken meer en meer gebruikt om online databronnen aan te spreken.

Vanaf nu worden de volgende zaken verondersteld: * De `requests`-bibliotheek is successvol geïnstalleerd; * De `requests`-bibliotheek is up-to-date.

We starten met enkele eenvoudige voorbeelden.

1.3.1 Een *request* plaatsen

Een *request* plaatsen met `requests` is heel eenvoudig. Hiervoor begin we met het importeren van de module `requests`:

[ ]:
```
import requests
```

Laten we nu proberen een webpagina te krijgen. We illustreren dit door een `get`-*request* uit te voeren op de *endpoint* van een simpele HTTP *request* & *response* Service op https://httpbin.org/.

[ ]:
```
r = requests.get('https://httpbin.org')
```

Nu hebben we een `response`-object genaamd `r`. We kunnen alle informatie die we nodig hebben uit dit object halen. De status van de *request* kunnen we eenvoudig bekijken door het object rechtstreeks aan te spreken:

[ ]:
```
r
```

De eenvoud van de API van `requests` zit hem in het feit dat alle vormen van HTTP-*requests* even vanzelfsprekend zijn. Een HTTP `POST`-*request* voeren we bijvoorbeeld als volgt uit:

[ ]:
```
r = requests.post('https://httpbin.org/post', data={'key': 'value'})
```

De andere typen HTTP-*requests* (`PUT`, `DELETE`, `HEAD` en `OPTIONS`) werken op een soortgelijke manier:

[ ]:
```
r = requests.put('https://httpbin.org/put', data={'key': 'value'})
r = requests.delete('https://httpbin.org/delete')
r = requests.head('https://httpbin.org/get')
r = requests.options('https://httpbin.org/get')
```

Opdracht: Bestudeer deze website om inzicht te krijgen in het verschil tussen de verschillende HTTP-requests. Eventueel kan deze website ook al bekeken worden voor meer informatie over de verschillende HTTP-response-codes, maar deze komen later nog aan bod.

1.3.2 Parameters doorgeven aan URL's

Vaak willen we een gegevens verzenden in de queryreeks van de URL, bijvoorbeeld om een specifiek adres op te halen uit de API van Vlaamse Basisregisters. Als we de URL manueel zouden opstellen, zouden alle gegevens als sleutel/waarde-paren in de URL ingegeven moeten worden na een vraagteken, bijvoorbeeld `httpbin.org/get?key=val`. Met `requests` kunnen we u deze argumenten opgeven als een `dict` van strings, met behulp van het params-sleutelwoordargument.

Als we bijvoorbeeld de Valentin Vaerwyckweg 1 in Gent willen opzoeken, gebruiken zullen we bijvoorbeeld `gemeentenaam=gent`, `straatnaam=valentin vaerwyckweg` en `huisnummer=1`door moeten geven aan `https://api.basisregisters.vlaanderen.be/v1/adresmatch`. Deze operatie kunnen we als volgt uitvoeren:

[ ]:
```
import requests
payload = {'gemeentenaam': 'gent', 'straatnaam': 'valentin vaerwyckweg',␣
 ↪'huisnummer': '1'}
```

```
r = requests.get('https://api.basisregisters.vlaanderen.be/v1/adresmatch',␣
↪params=payload)
```

We kunnen zien dat de URL correct is gecodeerd door de URL af te drukken:

[ ]:
```
print(r.url)
```

Houd er rekening mee dat een sleutel in de dict-object met een waarde None niet zal worden toegevoerd aan de queryreeks van de URL.

Opdracht: bekijk het resultaat van bovenstaande voorbeeld (opzoeking van de Valentin Vaerwyckweg 1 in Gent met behulp van https://api.basisregisters.vlaanderen.be/v1/adresmatch wanneer we de verschillende HTTP-requests uitvoeren. Om de status te evalueren gebruiken we de status_code-operator.

[ ]:
```
## UW CODE HIER ##
```

We kunnen ook meerdere waarden met een bepaalde sleutel doorgeven. Combinaties van meerdere waarden kunnen worden aangemaakt door van gegevens een `list` met `tuples` te maken. Ook kunnen we een `dict`-object voorzien van lijsten met de bijbehorende waarden. We illustreren dit concept door de verwijzing naar de Valentin Vaerwyckweg en de Voskenslaan in Gent op te halen van de API van Vlaamse Basisregisters:

[ ]:
```
payload_tuples = [('gemeentenaam', 'gent'), ('straatnaam', 'valentin␣
↪vaerwyckweg'), ('straatnaam', 'Voskenslaan')]
r_tupples = requests.get('https://api.basisregisters.vlaanderen.be/v1/
↪adresmatch', params=payload_tuples)
payload_dict = {'gemeentenaam': 'gent', 'straatnaam': ['valentin vaerwyckweg',␣
↪'Voskenslaan']}
r_dict = requests.get('https://api.basisregisters.vlaanderen.be/v1/adresmatch',␣
↪params=payload_dict)
r_tupples.text == r_dict.text
```

Opdracht: op de website van de NMBS kunnen we via een formulier aanbevolen routes opvragen om op een gegeven datum en tijd van het ene naar het andere station in België te gaan. iRail is een open-source platform dat eveneens allerhande data van de NMBS beschikbaar maakt via een API. De API beschikt over een uitgebreide documentatie, waarin verschillende functies besproken worden, zoals stations, connections of occumancy. Per functie wordt aangegeven of de serive wordt aangesproken met een GET of POST request. Daarnaast worden de verschillende parameters besproken en wordt telkens een voorbeeld gegeven.

Gebruik connections om de aanbevolen routes te bepalen van Neerpelt naar Gent-Sint-Pieters op zondag 12 december 2021 om 1900h. Maak hiervoor een verwijzing naar de endpoint als een string-object en een dict-object om de verschillende parameters te verwerken. Het volstaat nu om voor het resultaat van de request enkel de URL terug te geven met de url-operator.

[ ]:
```
## UW CODE HIER ##
```

1.3.3 Inhoud van de *response*

Vanzelfsprekend wordt het gebruik van de `requests`-bibliotheek helemaal zinvol als het mogelijk is om de inhoud van de *response* te lezen. We bekijken dit aan de hand van het resultaat voor de Valentin Vaerwyckweg 1 te Gent volgens de API van Vlaamse Basisregisters:

[ ]:
```
import requests
payload = {'gemeentenaam': 'gent', 'straatnaam': 'valentin vaerwyckweg',␣
 ↪'huisnummer': '1'}
r = requests.get('https://api.basisregisters.vlaanderen.be/v1/adresmatch',␣
 ↪params=payload)
r.text
```

De inhoud van de server zal automatisch gedecodeerd worden door de `requests`-bibliotheek. De meeste Unicode-tekensets zullen zonder problemen gedecodeerd kunnen worden. Wanneer we een *request* uitvoeren, maakt `requests` een redelijk goede inschatting over de juiste codering van het antwoord op basis van de *HTTP-headers*. De tekstcodering die wordt ingeschat door `requests` wordt gebruikt wanneer we de code `r.text` uitvoeren. We kunnen achterhalen welke coderingsverzoeken worden gebruikt (en deze eventueel wijzigen):

[ ]:
```
r.encoding
r.encoding = 'ISO-8859-1'
```

Zodra we de codering wijzigen, zal `requests` de nieuwe waarde van `r.encoding` gebruiken wanneer we `r.text` opnieuw aanroepen. Dit is mogelijkst nuttig in situaties waarin we speciale karaktersets moeten toepassen om de inhoud van de *response* correct te verwerken. HTML en XML hebben bijvoorbeeld de mogelijkheid om de codering expliciet te specificeren in hun body. In dergelijke situaties moeten we `r.content` gebruiken om de oorspronkelijke codering te vinden en vervolgens `r.encoding` op deze opnieuw in te stellen. Tot slot gebruiken we `r.text` om het resultaat te controleren.

Opdracht: We gaan verder met de vorige oefening. Gebruik opnieuw connections om nu de aanbevolen routes te bepalen van Appelterre naar Gent-Sint-Pieters op zondag 12 december 2021 om 1900h. Maak hiervoor een verwijzing naar de endpoint als een string-object en een dict-object om de verschillende parameters te verwerken. Geef het resultaat van request weer door de text-operator te gebruiken.

[ ]:
```
## UW CODE HIER ##
```

1.3.4 Inhoud van JSON-response

Er is ook een ingebouwde JSON-decoder binnen de `requests` bibliotheek, voor het geval dat we te maken hebben met JSON-gegevens. Een groot aantal services leveren hun data standaard aan in het JSON-formaat. We illustreren deze veelgebruikte methode aan de hand van het eerdere voorbeeld voor de Valentin Vaerwyckweg 1 te Gent:

[ ]:
```
import requests
payload = {'gemeentenaam': 'gent', 'straatnaam': 'valentin vaerwyckweg',␣
 ↪'huisnummer': '1'}
```

```
r = requests.get('https://api.basisregisters.vlaanderen.be/v1/adresmatch',␣
 ↪params=payload)
r.json()
```

Het ophalen van specifieke waarden verloopt zoals we dat gewoon zijn bij het werken met JSON-bestanden in Python. De coördinaten van bovenstaande *request* halen we bijvoorbeeld als volg op:

[ ]:
```
coordinates = r.
 ↪json()['adresMatches'][0]['adresPositie']['point']['coordinates']
x = coordinates[0]
y = coordinates[1]
print('De coördinaten zijn x: %.2f en y: %.2f' % (x, y))
```

Indien de JSON-decodering mislukt, genereert `r.json()` een uitzondering. Als het antwoord bijvoorbeeld een HTTP status 204 (No Content) is, of als het antwoord ongeldige JSON-code bevat, zal `r.json()` een `requests.exceptions.JSONDecodeError` opwerpen. Dit specifiek type wrapper-uitzondering maakt interoperabiliteit van meerdere uitzonderingen mogelijk die kunnen worden gegenereerd door verschillende python-versies en json-serialisatiebibliotheken. Deze melding krijgen we bijvoorbeeld wanneer we de zojuist opgehaalde afbeelding proberen om te zetten naar JSON:

[ ]:
```
import requests
endpoint = "https://tile.informatievlaanderen.be/ws/raadpleegdiensten/wmts"
payload = {"SERVICE":"WMTS", "VERSION":"1.0.0", "REQUEST":"GetTile",\
           "LAYER":"grb_bsk", "STYLE": "", "FORMAT":"image/png",\
           "TILEMATRIXSET":"GoogleMapsVL", "TILEMATRIX":"15",\
           "TILEROW":"10965", "TILECOL":"16721"}
r = requests.get(endpoint, params=payload)
r.json()
```

Merk op dat het al dan niet succesvol zijn van `r.json()` niets zegt over het daadwerkelijke succes van de *response*. Sommige servers zullen bijvoorbeeld ook een JSON-object terug geven bij een foute *response* (bijvoorbeeld bij een HTTP status 500 (Internal Server Error). Een dergelijke *response* kan wel degelijk omgezet worden naar een geldig JSON-object. Om te controleren of een *request* succesvol is, en dus resulteert in een HTTP status 200(OK), kan de correcte HTTP-status geëvalueerd worden met `r.raise_for_status()` of meer algemeen met `r.status_code`:

[ ]:
```
import requests
payload = {'onzin': 'nonsense'}
r = requests.get('https://api.basisregisters.vlaanderen.be/v1/adresmatch',␣
 ↪params=payload)
print("HTTP status: %s" % r.status_code)
```

Het resultaat van een bepaalde *request* kunnen we gebruiken om in bepaalde gevallen een geheel nieuwe URL aan te maken. Dit illustreren we in onderstaande code waarbij we een geojson van het Vlaamse GRB gebruiken als input voor een website die de visualisatie van geojson-bestanden mogelijk maakt:

[ ]:
```
from requests.utils import requote_uri
def getGBG(fid):
    url = 'https://geoservices.informatievlaanderen.be/overdrachtdiensten/GRB/
 ↪wfs'
    payload = {'SERVICE':'WFS','REQUEST':'GetFeature','VERSION':'2.0.0',
               'TYPENAMES':'GRB:GBG','featureID':'GBG.' + str(fid),
               'outputFormat':'application/json','srsName':'EPSG:4326'}
    return requests.get(url, params=payload)

url = 'http://geojson.io'
url = url + '#data=data:application/json,' + requote_uri(getGBG(1929586).text)

r = requests.get(url)
print(r.url)
```

Opdracht: Op Provincies in Cijfers worden een groot aantal kerncijfers aangeboden met betrekking tot allerhande variabelen op verschillende ruimtelijke niveaus in Vlaanderen. De meeste gebruikers zullen deze website gebruiken om inzicht te krijgen in hun wijk, gemeente, provincie, … door automatische rapporten te bestuderen, of door tabellen aan te maken via een interactieve omgeving. Het is echter ook mogelijk om specifieke data op te vragen door parameters mee te geven met de URL. Bekijk de handleiding voor meer informatie.

```
<p>Maak gebruik van <code>https://provincies.incijfers.be/jive/selectiontableasjson.ashx</code
    <ul>
        <li>Totaal aantal inwoners volgens rijksregister [aantal]</li>
        <li>Eigenaars (t.o.v. huishoudens met gekende eigendomstitel) [%]</li>
        <li>Huurders (t.o.v. huishoudens met gekende eigendomstitel) [%]</li>
</ul>
<p>Zoek hiervoor de verschillende NIS-codes op van de deelgemeenten, alsook de codes van de ver
```

[ ]:
```
## UW CODE HIER ##
```

Opdracht: voor een internationale veldcampagne buiten Europa hebben we een vliegtuig te nemen vanaf Schiphol (Nederland). Op woensdag 15 december 2021 dienen we ten laatste om 1600h op deze luchthaven aanwezig. Gebruik opnieuw de connections-functie van iRail om de aanbevolen verbindingen vanuit Gent Sint-Pieters te bepalen, en geef op overzichtelijke wijze de volgende zaken weer:

```
<ul>
    <li>Vertrektijd vanuit Gent</li>
    <li>Eventuele stations waar we moeten overstappen, inclusief de aankomst- en vertrektijden
    <li>Aankomsttijd op Schiphol</li>
</ul>
```

[ ]:
```
## UW CODE HIER ##
```

1.3.5 Binaire *response*-inhoud

We kunnen ook toegang vragen tot de *response* als bytes voor niet-tekstverzoeken. Dit is nuttig als we bijvoorbeeld afbeeldingen willen opvragen, zoals in onderstaande voorbeeld een snede uit het

Vlaamse GRB via WMTS:

```
[ ]: import requests
endpoint = "https://tile.informatievlaanderen.be/ws/raadpleegdiensten/wmts"
payload = {"SERVICE":"WMTS", "VERSION":"1.0.0", "REQUEST":"GetTile",\
           "LAYER":"grb_bsk", "STYLE": "", "FORMAT":"image/png",\
           "TILEMATRIXSET":"GoogleMapsVL", "TILEMATRIX":"15",\
           "TILEROW":"10965", "TILECOL":"16721"}

r = requests.get(endpoint, params=payload)
print(type(r.content))
```

De *response* bevat binaire data (type `byte`) en omvat meer bepaald een afbeelding in het PNG-formaat. Deze afbeelding kunnen we opslaan voor verdere verwerking, maar ook gewoon visualiseren:

```
[ ]: from PIL import Image
from io import BytesIO
with open('data/GRB-GRB_BSK.png', mode='wb') as grb_bsk:
    grb_bsk.write(r.content)
i = Image.open(BytesIO(r.content))
display(i)
```

Opdracht: Historisch-cartografisch materiaal van Vlaanderen wordt onder meer aangeboden via deze WMS. Schrijf Python-code waarmee voor een gegeven adrespunt een bounding box van +/- 500 meter wordt berekend, en waarbij deze rechthoek vervolgens wordt gebruikt om voor dit gebied zowel de Ferraris-kaart als de Vandermaelen-kaart te downloaden PNG.

```
[ ]: ## UW CODE HIER ##
```

1.3.6 Onbewerkte response-inhoud

> Deze sectie heeft de auteur van deze *Notebook* niet kunnen testen in een praktische toepassing.

In het zeldzame geval dat we het raw socket-antwoord van de server willen ontvangen, kunnen we toegang krijgen tot `r.raw`. Indien dit gewenst is, dient `stream=True` in te worden gesteld bij het eerste verzoek. Hieronder wordt een *request* uitgevoerd en worden de eerste 10 bytes gelezen:

```
[ ]: import requests
r = requests.get('https://api.github.com/events', stream=True)
r.raw.read(10)
```

Over het algemeen moeten we echter een patroon gebruiken zoals in onderstaande voorbeeld om de data doe gestreamd op te slaan in een bestand:

```
[ ]: r = requests.get('https://api.github.com/events', stream=True)
with open("RawResponseContent.txt", 'wb') as fd:
    for chunk in r.iter_content(chunk_size=128):
        fd.write(chunk)
```

Het gebruik van `response.iter_content` zal veel afhandelen van wat we anders zelf zouden moeten doen als we `response.raw` rechtstreeks zouden gebruiken. Het wordt daarom aanbevolen om de bovenstaande methode te gebruiken voor het streamen van downloads en het ophalen van de inhoud. Merk op dat de variabele `chunk_size` vrij kan worden aangepast in functie van specifieke toepassingen.

Een belangrijke opmerking over het gebruik van response.iter_content versus response.raw is dat response.iter_content automatisch de gzip zal decoderen en transfer-coderingen leeg zal laten lopen. Aan de andere kant werkt response.raw met een onbewerkte stroom van bytes. Het transformeert de inhoud van de response bijgevolg niet. Indien we echt toegang nodig hebben tot de bytes zoals ze zijn geretourneerd, gebruiken we dus best response.raw.

1.3.7 Aangepaste *headers*

Als we HTTP-*headers* aan een verzoek willen toevoegen, dienen we eenvoudigweg een `dict` door aan de `headers`-parameter. Dit is bijvoorbeeld interessant wanneer we de *user-agent* mee willen geven, of zoals in onderstaande voorbeeld, om de *response* op te vragen in XML:

[ ]:
```
import requests
payload = {'gemeentenaam': 'gent', 'straatnaam': 'valentin vaerwyckweg',␣
 ↪'huisnummer': '1'}
headers = {"accept": "application/xml"}
r = requests.get('https://api.basisregisters.vlaanderen.be/v1/adresmatch',␣
 ↪params=payload, headers=headers)
r.text
```

Het moet worden opgemerkt dat eigen *headers* een lagere prioriteit krijgen dan meer specifieke informatiebronnen. Bijvoorbeeld: - Autorisatie-*headers* die zijn ingesteld in de `dict`-object met `headers=...` worden overschreven als referenties zijn opgegeven in .netrc, die op hun beurt worden overschreven door de parameter `auth=`. `requests` naar naar het netrc-bestand op `~/.netrc`, `~/_netrc` of op het pad dat is opgegeven in de NETRC-omgevingsvariabele; - Autorisatie-*headers* worden verwijderd als we *off-host* worden omgeleid; - Proxy-autorisatie-headers worden overschreven door proxy-inloggegevens die in de URL zijn opgegeven; - *Content-Length headers* worden overschreven wanneer we de lengte van de inhoud kunnen bepalen.

Tot slot verandert het gedrag van `requests` helemaal wanneer aangepaste headers zijn opgegeven. De *headers* worden simpelweg doorgegeven in de uiteindelijke *request*.

Opmerking: alle koptekstwaarden moeten een string, bytestring of unicode zijn. Het wordt geadviseerd om te voorkomen dat Unicode-headerwaarden worden doorgegeven, maar dit is niet vereist.

Opdracht: met de routerplanner van De Lijn kunnen we de aanbevolen routes berekenen tussen twee punten in Vlaanderen. Dergelijke data zijn ook beschikbaar via de API van De Lijn, naast een grote hoeveelheid andere data. Informatie over het gebruik van deze data is hier beschikbaar. Het gebruik van deze service vereist registratie, waarna de gebruiker een API-key toegewezen krijgt. Deze key dient bij iedere request meegestuurd te worden in de header, gekoppeld aan de Ocp-Apim-Subscription-Key-sleutel.

Gebruik (onder andere) de methodes geefHaltesIndebuurt en geefRouteplan om een overzicht te krijgen in de aanbevole route van Gent Sint Pieters naar de Bijloke-site in Gent op maandag 13

december 2021, met voorzien vertrek om 0800h. Geef het resutaat op een overzichtelijke manier weer.

[ ]:
```
## UW CODE HIER ##
```

1.3.8 Meer complexe POST-requests

Meestal willen we gegevens van de *client* naar de *server* gecodeerd verzenden, zoals bij het opsturen van de inhoud van een HTML-formulier of voor het doorsturen van parameters via AJAX.

Om dit te doen, geven we eenvoudigweg een `dict`-object door aan het `data`-argument. Dit `dict`-object wordt automatisch in de juiste vorm gecodeerd wanneer de *request* wordt uitgevoerd. In onderstaand voorbeeld gebruiken we een API voor het ophalen van data over telramen in Vlaanderen:

Opmerking: voor het gebruik van https://telraam.net is een API key vereist. Deze kan gratis verkregen worden na registratie op de website.

[ ]:
```
import requests
url = 'https://telraam-api.net/v1/reports/traffic'
headers = {'X-Api-Key': '<HIER MOET EEN API KEY KOMEN>, 'content-type':␣
 ↪'application/json'}
payload = {'level': 'segments', 'format': 'per-hour','id':␣
 ↪'155073','time_start': '2021-12-01 06:00:00Z','time_end': '2021-12-01 09:00:
 ↪00Z'}
r = requests.post(url, data=payload, headers=headers)
r.json()
```

Opmerking: zoals bij het params-argument bij een GET-request, kan het data-argument ook meerdere waarden hebben voor een individuele key. Dit is met name handig wanneer het formulier meerdere elementen bevat die dezelfde sleutel gebruiken.

Hoewel we in bovenstaande voorbeeld onze parameters verwerkt hebben in een `dict`, hebben we niet het gewenste resultaat gekregen (HTTP 400 Bad Request). Er zijn namelijk situaties, zoals bij dit voorbeeld, waarbij we gegevens moeten (of willen) verzenden die nog niet in de finale vorm zijn gecodeerd. Als we een `string` doorgeven in plaats van een `dict`, worden die gegevens direct gepost:

[ ]:
```
url = 'https://telraam-api.net/v1/reports/traffic'
headers = {'X-Api-Key': '<HIER MOET EEN API KEY KOMEN>', 'content-type':␣
 ↪'application/json'}
payload = {'level': 'segments', 'format': 'per-hour','id': '155073',
           'time_start': '2021-12-01 06:00:00Z','time_end': '2021-12-01 09:00:
 ↪00Z'}
r = requests.post(url, data=str(payload), headers=headers)
r.json()
```

Het `dict`-object kunnen we ook omzetten in een json-achtige string met behulp van `json.dumps`. Hiermee krijgen we dezelfde resultaten als zojuist:

```
[ ]: import json
url = 'https://telraam-api.net/v1/reports/traffic'
headers = {'X-Api-Key': '<HIER MOET EEN API KEY KOMEN>', 'content-type':␣
 ↪'application/json'}
payload = {'level': 'segments', 'format': 'per-hour','id': '155073',
           'time_start': '2021-12-01 06:00:00Z','time_end': '2021-12-01 09:00:
 ↪00Z'}
r_dumps = requests.post(url, data=json.dumps(payload), headers=headers)
r_str = requests.post(url, data=str(payload), headers=headers)
r_dumps.text == r_str.text
```

In plaats van de inhoud van het `dict`-object zelf te coderen, kunnen we deze ook direct doorgeven met behulp van de `json`-parameter, dat de inhoud van het `dict`-object automatisch zal coderen:

```
[ ]: url = 'https://telraam-api.net/v1/reports/traffic'
headers = {'X-Api-Key': '<HIER MOET EEN API KEY KOMEN>', 'content-type':␣
 ↪'application/json'}
payload = {'level': 'segments', 'format': 'per-hour','id': '155073',
           'time_start': '2021-12-01 06:00:00Z','time_end': '2021-12-01 09:00:
 ↪00Z'}
r_json = requests.post(url, json=payload, headers=headers)
r_str = requests.post(url, data=str(payload), headers=headers)
r_json.text == r_str.text
```

Opmerking: de json-parameter wordt genegeerd zodra er gegevens of bestanden worden doorgegeven. Als we de json-parameter in de aanvraag gebruiken, wordt het inhoudstype in de koptekst gewijzigd naar application/json.

Opdracht: de Openrouteservice-API biedt wereldwijde ruimtelijke diensten door gebruik te maken van OpenStreetMap. Het is zeer performant en flexibel te gebruiken met behulp van onder andere de requests-bibliotheek. Onder meer de volgende services zijn beschikbaar:

```
<ul>
    <li>Routebeschrijving: retourneert een route tussen twee of meer locaties voor een geselec
    <li>Isochronen: verkrijgt bereikbaarheidsgebieden van bepaalde locaties;</li>
    <li>Matrix: berekent een-op-veel-, veel-op-een- of veel-op-veel-routes voor elk vervoermid
</ul>
<p>De service vereist registratie en een API-<i>key</i> bij iedere <i>request</i>. Maak een ac
```

```
[ ]: ## UW CODE HIER ##
```

1.3.9 POST een uit meerdere delen gecodeerd bestand

Met `requests` is het eenvoudig om *Multipart*-gecodeerde bestanden te uploaden. *Multipart requests* combineren een of meer verzamelingen van data in een enkele *body*, zoals binaire bestanden. In onderstaand voorbeeld gebruiken we `https://httpbin.org` om een Excel-bestand op te laden met Belgische bevolkingsstatistieken. De data worden eerst gedownlaod en vervolgens in een *request* meegestuurd:

```
[ ]: import requests
url = 'https://statbel.fgov.be/sites/default/files/files/documents/bevolking/5.
 ↪1%20Structuur%20van%20de%20bevolking/Bevolking_per_gemeente.xlsx'
r = requests.get(url)
with open('data/statbel.xlsx', mode='wb') as grb_bsk:
    grb_bsk.write(r.content)
url = 'https://httpbin.org/post'
files = {'file': open('data/statbel.xlsx', 'rb')}

r = requests.post(url, files=files)
r.text
```

We kunnen de bestandsnaam, *content_type* en aanvullende *headers* expliciet instellen in een 4-*tupple*, zoals omschreven in de documentatie. In onderstaande voorbeeld wordt dit geïllustreerd door een snede uit het Vlaamse GRB op te halen en mee te geven in een `POST`-*request*. Met onderstaande code tonen we aan dat het niet vereist is om te verwijzen naar een bestand dat ergens op de PC aanwezig is, maar dat we ook gebruik kunnen maken van een datastream:

```
[ ]: def getGRB_WMS():
    url = 'https://geoservices.informatievlaanderen.be/raadpleegdiensten/
 ↪GRB-basiskaart/wms'
    payload = {'SERVICE':'WMS','VERSION':'1.3.0','REQUEST':'GetMap',
               'BBOX':'51.0327,3.7003,51.0362,3.7038', 'CRS':'EPSG:4326',
               'WIDTH':'512','HEIGHT':'512','LAYERS':'GRB_BSK','FORMAT':'image/
 ↪geotiff',
               'DPI':'72','TRANSPARENT':'TRUE'}
    r = requests.get(url, params=payload)
    return r

grb = getGRB_WMS()
url = 'https://httpbin.org/post'
files = {'file': ('data/GRB_BSK.tif', grb.content, 'image/geotiff', {'Expires':␣
 ↪'0'})}

r = requests.post(url, files=files)
r.text
```

Indien gewenst kunnen we ook *strings* verzenden die als bestanden te worden ontvangen:

```
[ ]: def getGBG_WFS(fid):
    url = 'https://geoservices.informatievlaanderen.be/overdrachtdiensten/GRB/
 ↪wfs'
    payload = {'SERVICE':'WFS','REQUEST':'GetFeature','VERSION':'2.0.0',
              'TYPENAMES':'GRB:GBG','featureID':'GBG.' + str(fid),
              'outputFormat':'application/json','srsName':'EPSG:4326'}
    r = requests.get(url, params=payload)
    return r
```

```
grb_gbg = getGBG_WFS(1929586)
url = 'https://httpbin.org/post'
files = {'file': ('grb_gbg.geojson', grb_gbg.text)}
r = requests.post(url, files=files)
r.text
```

Wanneer we een zeer groot bestand meesturen als `multipart/form-data`, zouden we onze data misschien beter streamen. Standaard wordt dit niet ondersteunt door de `requests`-bibliotheek. Er si echter een apart pakket dat dit wel doet, namelijk de `requests-toolbelt`-bibiotheek. We verwijzen naar de *toolbelt*-documentatie voor meer informatie. Voor het verzenden van meerdere bestanden in een *request* verwijzen we naar de oorspronkelijke documentatie van `requests`.

Raadpleeg de geavanceerde sectie voor het verzenden van meerdere bestanden in één verzoek.

Waarschuwing: Het wordt sterk aanbevolen om bestanden in binaire modus te openen. Dit komt omdat requests zal proberen om de Content-Length-header voor de data zal berekenen, die wordt deze ingesteld op basis van het aantal bytes in het bestand. Er kunnen echter fouten optreden als we het bestand in tekstmodus openen.

1.3.10 HTTP *response* status codes

Zoals eerder vermeld kunnen we de reactiestatuscode controleren, ook wel HTTP response status codes genoemd:

[ ]:
```
import requests
payload = {'gemeentenaam': 'gent', 'straatnaam': 'valentin vaerwyckweg',␣
↪'huisnummer': '1'}
r = requests.get('https://api.basisregisters.vlaanderen.be/v1/adresmatch',␣
↪params=payload)
r.reason
```

Verzoeken worden ook geleverd met een ingebouwd statuscode-opzoekobject voor eenvoudige referentie:

[ ]:
```
r.status_code == requests.codes.ok
```

Als we een slechte *request* verzonden hebben, resulterend in een 4XX-clientfout of een 5XX-serverfoutreactie, kunnen we de resulterende fout opvangen met `response.raise_for_status()`:

[ ]:
```
import requests
payload = {'onzinsleutel': 'onzinwaarde'}
r_bad = requests.get('https://api.basisregisters.vlaanderen.be/v1/adresmatch',␣
↪params=payload)
r_bad.status_code
```

[ ]:
```
print(r_bad.raise_for_status())
```

Maar aangezien onze `status_code` voor `r` gelijk is aan `200` was krijgen we niet terug als we `raise_for_status()` aanroepen:

[ ]:
```
print(r.raise_for_status())
```

Opdracht: tot hier hebben we al een aantal services gezien die we kunnen gebruiken voor een grote verscheidenheid aan vraagstukken. In de documentatie hebben we ook gezien dat er expliciet een onderscheid gemaakt wordt tussen functies die we kunnen gebruiken in een GET-request en andere in een POST-request. Tracht met deze services een aantal verschillende HTTP-status codes te genereren, en print het resultaat.

[ ]:
```
## UW CODE HIER ##
```

1.3.11 Response Headers

We kunnen de *headers* van de *response* van de server bekijken met behulp van een Python-`dict`. We illustreren dit opnieuw aan de hand van de Valentin Vaerwyckweg 1 te Gent volgens de API van Vlaamse Basisregisters:

[ ]:
```
import requests
payload = {'gemeentenaam': 'gent', 'straatnaam': 'valentin vaerwyckweg',␣
 ↪'huisnummer': '1'}
r = requests.get('https://api.basisregisters.vlaanderen.be/v1/adresmatch',␣
 ↪params=payload)
r.headers
```

Dit `dict`-object is echter speciaal: het is alleen gemaakt voor HTTP-*headers*. Volgens RFC 7230 zijn de sleutels van HTTP-headers niet hoofdlettergevoelig. We hebben dus toegang tot de headers met elk hoofdlettergebruik dat we willen:

[ ]:
```
print(r.headers['Content-Type'])
print(r.headers['CONTENT-TYPE'])
print(r.headers.get('Content-Type'))
print(r.headers.get('content-type'))
```

Het is ook bijzonder dat de server dezelfde header meerdere keren met verschillende waarden zou kunnen terugsturen. De `requests`-bibliotheek combineert ze echter, zodat ze in het `dict`-object kunnen worden weergegeven binnen een enkele toewijzing. RFC 7230 zegt hierover:

> *"A recipient may combine multiple header fields with the same field name into one "field-name: field-value" pair, without changing the semantics of the message, by appending each subsequent field value to the combined field value in order, separated by a comma."* (RFC 7230, §3.2.2.
>
> *"RFC: request for romment, a document circulated on the internet which forms the basis of a technical standard." (Oxford Dictionaries).

Opdracht: epsg.io is een open-source web service met een databank voor coördinaatreferentiesystemen (CRS). Naast een overzicht van een groot aantal systemen, zoals het Belgische Lambert 2008 of WGS84, laat de service ook toe om coördinaten te transformeren van het ene naar het andere systeem.

`<p>Herneem de code waarmee de coördinaten van de Valentin Vaerwyckweg 1 in Gent zijn bepaald m`

[ ]:
```
## UW CODE HIER ##
```

1.3.12 Cookies

Als een *response* cookies bevat, kunnen we deze afzonderlijk bekijken:

[ ]:
```
import requests
headers = {'Ocp-Apim-Subscription-Key': '4ffa80ce87a3430184c9963eb46d0fae'}
url = 'https://api.delijn.be/DLZoekOpenData/v1/zoek/haltes/Gent Hogeschool'
r = requests.get(url, headers=headers)
r.cookies
```

Om onze eigen cookies naar de server te sturen, kunnen we de `cookies`-parameter gebruiken:

[ ]:
```
cookies = r.cookies
url = 'https://api.delijn.be/DLZoekOpenData/v1/zoek/haltes/Gent Hogeschool'
headers = {'Ocp-Apim-Subscription-Key': '4ffa80ce87a3430184c9963eb46d0fae'}
r = requests.get(url, headers=headers, cookies=cookies)
r.text
```

Cookies worden geretourneerd in een `RequestsCookieJar`, die werkt als een dict maar daarnaast een completere interface biedt, geschikt voor gebruik over meerdere domeinen of paden. `RequestsCookieJar`-objecten kunnen ook worden doorgegeven aan `requests`:

[ ]:
```
jar = requests.cookies.RequestsCookieJar()
jar.set('tasty_cookie', 'yum', domain='httpbin.org', path='/cookies')
jar.set('gross_cookie', 'blech', domain='httpbin.org', path='/elsewhere')
url = 'https://httpbin.org/cookies'
r = requests.get(url,  cookies=jar)
r.text
```

1.3.13 Omleiding en geschiedenis

Met uitzondering van `HEAD` voert `requests` standaard locatie-omleiding uit voor alle HTTP *request*-methoden. We kunnen de `history`-eigenschap van het `response`-object gebruiken om de omleidingen bij te houden. De lijst `response.history` bevat de `response`-objecten die zijn gemaakt om de *request* te voltooien. De lijst is gesorteerd van de oudste tot de meest recente reactie.

De server van geo.hogent.be is bijvoorbeeld zo ingesteld dat een *request* naar `http://geo.hogent.be` (HTTP) zal worden omgeleid naar `https://geo.hogent.be` (HTTPS). Dit resulteert in een HTTP 302 Found-melding.

[ ]:
```
import requests
r = requests.get('http://geo.hogent.be')
print('url: %s' % r.url)
print('status code: %s' % r.status_code)
print('history: %s' % r.history)
```

Als we `GET`, `OPTIONS`, `POST`, `PUT`, `PATCH` of `DELETE` gebruiken, kunnen we de afhandeling van omleidingen uitschakelen met de parameter `allow_redirects`:

[ ]:
```
import requests
r = requests.get('http://geo.hogent.be', allow_redirects=False)
print('url: %s' % r.url)
print('status code: %s' % r.status_code)
print('history: %s' % r.history)
```

Als we `HEAD` gebruikt, kunt we omleidingen ook inschakelen:

[ ]:
```
import requests
r = requests.head('http://geo.hogent.be', allow_redirects=True)
print('url: %s' % r.url)
print('status code: %s' % r.status_code)
print('history: %s' % r.history)
```

1.3.14 *Time-outs*

We kunnen *requests* na een bepaalde tijd laten stoppen met wachten als er geen reactie komt van de server. De wachttijd stellen we vast met de `timeout`-parameter. Het wordt aanbevolen om alle code in operationele omgevingen te voorzien van een dergelijke parameter, en deze dus te gebruiken bij iedere *request*. Als we dit niet doen, kan onze code voor onbepaalde tijd vastlopen:

[ ]:
```
import requests
try:
    requests.get('https://github.com/', timeout=0.001)
except requests.exceptions.Timeout as e:
    print(e)
```

Opmerking: de timeout-parameter is geen tijdslimiet voor de volledige download van de response. Er wordt echter een uitzondering opgeworpen wanneer de server geen reactie heeft gegeven voor *time-out*-seconden, of met andere woorden, wanneer er geen *bytes* zijn ontvangen op de onderliggende socket gedurende *time-out*-seconden. Als er geen timeout-parameter expliciet wordt opgegeven, is er ook geen time-out voor aanvragen.

1.3.15 Fouten en uitzonderingen

We sluiten deze *notebook* af met enkele voorbeelden van fouten en uitzonderingen: * In het geval van een netwerkprobleem (bijvoorbeeld bij een DNS-fout, een geweigerde verbinding, enz.), zal `requests` een `ConnectionError`-uitzondering opwerpen. * Er wordt een `HTTPError` gegenereerd wanneer `response.raise_for_status()` een mislukte statuscode retourneerd. * Als er een *time-out* optreedt voor een *request* wordt een `Timeout`-uitzondering gegenereerd. * Als een aanvraag het geconfigureerde aantal maximale omleidingen overschrijdt, wordt een `TooManyRedirects`-uitzondering gegenereerd. * Alle uitzonderingen die `requests` expliciet oproept, worden overgeorven van `requests.exceptions.RequestException`.

1.4 Meer informatie

Bekijk de originele gids of de geavanceerde sectie voor meer informatie.