Project blok 6 (ingekorte versie)

Inleiding

In dit blok maak je een *web crawler*. Dit is een tool die automatisch webpagina's bezoekt en er informatie uit haalt. Zoekmachines gebruiken web crawlers om content te vinden en te indexeren, machine learning tools gebruiken ze (al dan niet met respect voor de juridische beperkingen...) om training data te verzamelen.

(Technisch is *scraping* het ophalen van de data en *crawling* het bezoeken van URL's, maar de twee gaan samen.)

In essentie komt de werking van een web crawler neer op het volgende:

1. Vraag een webpagina op
2. Verzamel de interessante content op die pagina
3. Verzamel verdere links op de pagina
4. Bezoek die links en doe daar hetzelfde, tot een maximaal aantal links gebruikt is of tot een zeker tijdbudget op is

We zullen data van de websites halen met reguliere expressies, die in een eerder project reeds aan bod kwamen. We zullen deze combineren met `beautifulsoup`, een HTML-parser, dus een tool die aansluit bij het eerdere `libcst` maar voor een andere taal. We zullen ook een handige command line interface bouwen met `click`. In plaats van het eerdere requests gebruiken we `aiohttp`, omdat we *asynchroon* zullen werken.

Indien er tijd overblijft, bekijken we ook een extra veelzijdige aanpak, waarbij je een klassieke browser gebruikt, met Selenium.

generatorfuncties

We doorlopen in de les deze uitleg tot en met "Understanding the Python `yield` statement".

Oefening

Maak een generatorfunctie `fibonacci(n)`, die de eerste n getallen van de Fibonacci-reeks genereert. De Fibonacci-reeks is een reeks getallen waarbij elk getal de som is van de twee voorgaande getallen, te beginnen met 0 en 1. Test ze uit door de eerste 10 Fibonaccigetallen te printen. Gebruik in dit geval een **iteratieve** implementatie, dus een implementatie met een lus, géén recursieve implementatie.

asynchroon programmeren

We doorlopen in de les deze uitleg. We slaan "Using a queue" en "Async IO's Roots in Generators" over. Voor we het grote voorbeeld "A Full Program: Asynchronous Requests" behandelen, maken we onderstaande oefeningen. Na het grote voorbeeld kan je stap 1 van het project maken.

Oefeningen

Asynchrone hello, world!

Maak een asynchrone functie async_hello() die simpelweg "Hello, World!" naar de console print na een wachttijd van 3 seconden. Gebruik `asyncio.sleep(3)` om de wachttijd te implementeren.

Asynchrone Fibonacci

Maak een asynchrone versie van je Fibonaccigenerator. Zorg dat je generator telkens minstens vijf seconden "rust" neemt na het genereren van een getal. Zet ook drie andere taken in de event loop, die telkens hetzelfde bericht (bijvoorbeeld "taak 1", "taak 2",...) tonen na verschillende delays tussen de 1 en de 5 seconden. Zorg dat je begrijpt wat er op het scherm verschijnt. Schrijf een testfunctie die eindeloos lang Fibonaccigetallen genereert.

Referenties

- https://realpython.com/async-io-python/

beautifulsoup

We baseren ons op de quickstart van BeautifulSoup. Hier zijn geen aparte oefeningen rond, we maken meteen stappen 2 en 3 van het project.

project

Stap 1

Vraag via een asynchroon request `www.ap.be` op. Toon de HTML-code op het scherm.

Stap 2

Verzamel met BeautifulSoup alle woorden die voorkomen in een tekstelement op de homepagina van AP. Gebruik een reguliere expressie om de woorden van elkaar te scheiden. Print het aantal voorkomens van elk woord op de console. Tip: een `Counter` is hier handig voor.

Verzamel ook alle URL's in links. Print achteraf de lijst van alle links onderaan af.

Stap 3

Laat je programma nu elke verzamelde link volgen en ook op die pagina's hetzelfde doen. Zorg dat je programma stopt door nooit verder dan twee links van de startpagina te gaan en door ook nooit een

link te bezoeken die reeds bezocht is.

Stap 4

In plaats van het aantal voorkomens van elk woord te printen, verzamel je dit in een JSON-file. Hier is de key telkens de URL en de value het aantal voorkomens van elk woord. De naam van de JSON file is verplicht mee te geven via de command line interface. Het object in JSON-notatie heeft ook een property `total` waarin de som van alle voorkomens van alle woorden over alle pagina's heen staat.

Epiloog

In de praktijk zou je, zeker wanneer JavaScript geen probleem vormt, eerder gebruik van de bibliotheek Scrapy. Scrapy handelt heel veel zaken automatisch goed af, maar je leert er niet zo veel mee bij over Python.