Wydanie II

Python
Automatyzacja zadań

Jak efektywnie pracować z danymi, arkuszami Excela, raportami i e-mailami

Jaime Buelta

Helion

Packt

Tytuł oryginału: Python Automation Cookbook: 75 Python automation ideas for web scraping, data wrangling, and processing Excel, reports, emails, and more, 2nd Edition

Tłumaczenie: Tomasz Walczak

ISBN: 978-83-8322-566-1

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/pyau2v
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
https://ftp.helion.pl/przyklady/pyau2v.zip

Printed in Poland.

Spis treści

3

ROZDZIAŁ

Tworzenie pierwszej aplikacji do pobierania informacji ze stron WWW

Internet (**sieć WWW**) jest obecnie zapewne najważniejszym źródłem informacji. Większość informacji można pobrać za pomocą protokołu HTTP. Pierwotnie został on opracowany w celu udostępniania stron hipertekstu (stąd nazwa — **HyperText Transfer Protocol**), co zapoczątkowało rozwój internetu.

Ten proces zachodzi za każdym razem, gdy użytkownik żąda strony WWW, dlatego powinien być znany prawie każdemu. Jednak te same operacje można wykonywać w kodzie, aby automatycznie pobierać i przetwarzać informacje. Python w standardowej bibliotece udostępnia klienta HTTP, jednak fantastyczny moduł `requests` sprawia, że pobieranie stron WWW jest bardzo łatwe. W tym rozdziale zobaczysz, jak to robić.

W tym rozdziale opisane są następujące receptury:

- Pobieranie stron WWW
- Parsowanie kodu HTML
- Crawling w sieci WWW
- Subskrybowanie kanałów informacyjnych
- Dostęp do internetowych API
- Interakcje z formularzami
- Używanie pakietu Selenium do obsługi zaawansowanych interakcji
- Dostęp do stron chronionych hasłem
- Przyspieszanie pobierania informacji ze stron WWW

Zacznij od podstaw programowego pobierania istniejących stron WWW.

Pobieranie stron WWW

Podstawową techniką pobierania stron WWW jest zgłaszanie żądań HTTP `GET` dotyczących adresu URL. Jest to podstawowa operacja w każdej przeglądarce internetowej.

Oto krótki przegląd różnych elementów tej operacji. Obejmuje ona trzy różne aspekty:

1. Użycie protokołu HTTP. Ten aspekt dotyczy struktury żądania.
2. Użycie metody `GET`. Jest to najczęściej stosowana metoda protokołu HTTP. Więcej dowiesz się z receptury „Dostęp do internetowych API".
3. Kompletny adres URL. Określa on adres strony, w tym serwer (na przykład `mojastrona.pl`) i ścieżkę (na przykład `/strona`).

Żądanie jest przekazywane na serwer przez internet i przetwarzane na serwerze. Następnie odsyłana jest odpowiedź. Zawiera ona **kod statusu** (zwykle 200, jeśli nie wystąpiły problemy) i treść z wynikiem, którym zwykle jest tekst strony HTML.

Większość tych operacji automatycznie wykonuje klient HTTP używany do zgłaszania żądania. W tej recepturze zobaczysz, jak utworzyć proste żądanie w celu pobrania strony WWW.

> Żądania i odpowiedzi HTTP mogą też obejmować nagłówki. Nagłówki zawierają ważne informacje na temat samego żądania, na przykład jego łączną wielkość, format treści, datę zgłoszenia oraz zastosowaną przeglądarkę i używany serwer.

Przygotowania

Dzięki świetnemu modułowi `requests` pobieranie stron WWW jest niezwykle łatwe. Zainstaluj ten moduł:

```
$ echo "requests==2.23.0" >> requirements.txt
$ source .venv/bin/activate
(.venv) $ pip install -r requirements.txt
```

Pobierzesz teraz stronę *http://www.columbia.edu/~fdc/sample.html*, ponieważ jest to prosta strona HTML łatwa do wczytania w trybie tekstowym.

Jak to zrobić?

1. Zaimportuj moduł `requests`:

```
>>> import requests
```

2. Przygotuj żądanie dla serwera, używając poniższego adresu URL. Przetwarzanie żądania zajmie sekundę lub dwie:

```
>>> url = 'http://www.columbia.edu/~fdc/sample.html'
>>> response = requests.get(url)
```

3. Sprawdź kod statusu w zwróconym obiekcie:

```
>>> response.status_code
200
```

4. Sprawdź treść wyniku:

```
>>> response.text
'<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">\
n<html>\n<head>\n
...
KOMPLETNA TREŚĆ
...
<!-- close the <html> begun above -->\n'
```

5. Sprawdź wysłane i otrzymane nagłówki:

```
>>> response.request.headers
{'User-Agent': 'python-requests/2.22.0', 'Accept-Encoding': 'gzip,
deflate', 'Accept': '*/*', 'Connection': 'keep-alive'}
>>> response.headers
{'Date': 'Fri, 24 Jan 2020 19:04:12 GMT', 'Server': 'Apache',
'Last-Modified': 'Wed, 11 Dec 2019 12:46:44 GMT', 'Accept-Ranges':
'bytes', 'Vary': 'Accept-Encoding,User-Agent', 'Content-Encoding':
'gzip', 'Content-Length': '10127', 'Keep-Alive': 'timeout=15,
max=100', 'Connection': 'Keep-Alive', 'Content-Type': 'text/
html', 'Set-Cookie': 'BIGipServer~CUIT~www.columbia.edu-80-
pool=1311259520.20480.0000; expires=Sat, 25-Jan-2020 01:04:12 GMT;
path=/; Httponly'}
```

Jak to działa?

Moduł `requests` działa w bardzo prosty sposób — wykonuje żądanie, używając (w tym przykładzie) metody `GET` i adresu URL. Zwracany jest obiekt `result`, który można przeanalizować. Główne elementy tego obiektu to kod statusu (`status_code`) i treść odpowiedzi (`text`).

Kompletne żądanie można sprawdzić za pomocą atrybutu `request`:

```
>>> response.request
<PreparedRequest [GET]>
>>> response.request.url
http://www.columbia.edu/~fdc/sample.html'
```

Pełną dokumentację modułu `requests` znajdziesz na stronie *https://requests.readthedocs.io/en/master/*.

W tym rozdziale poznasz więcej mechanizmów biblioteki `requests`.

Dodatkowe informacje

Wszystkie kody statusu z protokołu HTTP znajdziesz na stronie *https://httpstatuses.com/*. A także w wyliczeniu `http.HTTPStatus` zawierającym wygodne nazwy stałych (na przykład `OK`, `NOT_FOUND` lub `FORBIDDEN`).

> Najbardziej znanym kodem statusu informującym o błędzie jest 404. Jest on zwracany, gdy nie można znaleźć zasobu opisywanego przez podany adres URL. Przekonaj się o tym, zgłaszając wywołanie `requests.get(http://www.columbia.edu/invalid)`.

Ogólna struktura kodów statusu wygląda tak:

1XX — informacje o protokole.

2XX — powodzenie.

3XX — przekierowanie, na przykład wtedy, gdy adres URL nie jest już poprawny, a zasób jest dostępny gdzie indziej. Należy wtedy użyć nowego adresu URL.

4XX — błąd klienta. Wystąpił błąd w informacjach przesłanych na serwer (na przykład zastosowano niewłaściwy format danych) lub w kliencie (dostęp do danego adresu URL może wymagać uwierzytelnienia).

5XX — błąd serwera. Wystąpił błąd po stronie serwera, na przykład serwer jest niedostępny lub pojawił się problem w trakcie przetwarzania żądania.

W żądaniu można użyć protokołu **HTTPS** (**HTTP Secure**, czyli zabezpieczony HTTP). Działa on tak samo jak HTTP, ale zapewnia prywatność treści żądania i odpowiedzi. Moduł `requests` automatycznie obsługuje ten protokół.

> W każdej witrynie, w której przetwarzane są prywatne dane, należy stosować protokół HTTPS, aby zapewnić ochronę przed wyciekiem informacji. Protokół HTTP jest podatny na podsłuchiwanie. Jeśli protokół HTTPS jest dostępny, powinieneś z niego korzystać.

Zobacz także

- Receptura „Instalowanie niezależnych pakietów” w rozdziale 1., „Rozpoczęcie przygody z automatyzacją”. Poznasz w niej podstawy instalowania zewnętrznych modułów.
- Receptura „Parsowanie kodu HTML” w dalszej części rozdziału. Dzięki niej dowiesz się, jak przetwarzać informacje zwrócone przez serwer.

Parsowanie kodu HTML

Pobranie nieprzetworzonego tekstu lub pliku binarnego to dobry wstęp, jednak głównym językiem sieci WWW jest HTML.

Jest to język ustrukturyzowany, definiujący różne fragmenty dokumentu, na przykład nagłówki i akapity. HTML jest też hierarchiczny, ponieważ definiowane są w nim elementy podrzędne. Możliwość przekształcenia nieprzetworzonego tekstu na ustrukturyzowany dokument oznacza, że możliwe jest też automatyczne pobieranie informacji ze stron WWW. Na przykład tekst może być istotny, jeśli znajduje się w określonych elementach HTML-a, takich jak `div` o określonej klasie lub znacznik nagłówka `h3`.

Przygotowania

Użyjesz tu znakomitego modułu `BeautifulSoup`, aby przetworzyć tekst z kodu HTML na obiekt w pamięci, który można analizować. Aby zachować zgodność z Pythonem 3, musisz użyć najnowszej wersji pakietu — `beautifulsoup4`. Dodaj ten pakiet do pliku *requirements.txt* i zainstaluj zależności w środowisku wirtualnym:

```
$ echo "beautifulsoup4==4.8.2" >> requirements.txt
$ pip install -r requirements.txt
```

Jak to zrobić?

1. Zaimportuj moduły `BeautifulSoup` i `requests`:

```
>>> import requests
>>> from bs4 import BeautifulSoup
```

2. Podaj adres strony URL, aby ją pobrać:

```
>>> URL = 'http://www.columbia.edu/~fdc/sample.html'
>>> response = requests.get(URL)
>>> response
<Response [200]>
```

3. Przetwórz pobraną stronę:

```
>>> page = BeautifulSoup(response.text, 'html.parser')
```

4. Pobierz tytuł strony. Sprawdź, czy jest taki sam jak wyświetlony w przeglądarce:

```
>>> page.title
<title>Sample Web Page</title>
>>> page.title.string
'Sample Web Page'
```

5. Znajdź wszystkie elementy `h3` na stronie, aby ustalić, jakie sekcje się na niej znajdują:

```
>>> page.find_all('h3')
[<h3><a name="contents">CONTENTS</a></h3>, <h3><a name="basics">1.
Creating a Web Page</a></h3>, <h3><a name="syntax">2. HTML
```

```
Syntax</a></h3>, <h3><a name="chars">3. Special Characters</a></
h3>, <h3><a name="convert">4. Converting Plain Text to HTML</
a></h3>, <h3><a name="effects">5. Effects</a></h3>, <h3><a
name="lists">6. Lists</a></h3>, <h3><a name="links">7. Links</
a></h3>, <h3><a name="tables">8. Tables</a></h3>, <h3><a
name="install">9. Installing Your Web Page on the Internet</a></
h3>, <h3><a name="more">10. Where to go from here</a></h3>]
```

6. Pobierz tekst z sekcji *Special Characters*. Zakończ po dotarciu do następnego znacznika <h3>:

```
>>> link_section = page.find('h3', attrs={'id': 'chars'})
>>> section = []
>>> for element in link_section.next_elements:
...     if element.name == 'h3':
...         break
...     section.append(element.string or '')
...
>>> result = ''.join(section)
>>> result
'3. Special Characters\n\nHTML special "character entities"
start with ampersand (&&) and\nend with semicolon (;;), like
"&euro;&euro;" = "€". The\never-popular "no-break space" is
  . There are special\nentity names for accented Latin
letters and other West European special\ncharacters such as:\
n\n\n\n\n\n&auml;&auml;\na-umlaut\n\xa0ä\xa0\n\n\n&Auml;&Auml;\
nA-umlaut \n\xa0Ä\xa0\n\n\n&aacute;&aacute;\na-acute \n\xa0á\xa0\
n\n\n&agrave;&agrave;\na-grave \n\xa0à\xa0\n\n\n&ntilde;&ntilde;\
nn-tilde \n\xa0ñ\xa0\n\n\n&szlig;&szlig;\nGerman double-s\n\
xa0ß\xa0\n\n\n&thorn;&thorn;\nIcelandic thorn \n\xa0þ\xa0\n\xa0þ\
xa0\n\n\n\n\nExamples:\n\n\nFor SpanishSpanish you would need:\
n&Aacute;&Aacute; (Á),\n&aacute;&aacute; (á),\n&Eacute;&Eacute;
(É),\n&eacute;&eacute; (é),\n&Iacute;&Iacute; (Í),\
n&iacute;&iacute; (í),\n&Oacute;&Oacute; (Ó),\n&oacute;&oacute;
(ó),\n&Uacute;&Uacute; (ú),\n&uacute;&uacute; (ú),\
n&Ntilde;&Ntilde; (Ñ),\n&ntilde;&ntilde; (ñ);\n&iquest;&iquest;
(¿);\n&iexcl;&iexcl; (¡).\nExample: Añorarán = A&ntilde;ora
r&aacute;nA&ntilde;orar&aacute;n.\n\n\nFor GermanGerman you
would need:\n&Auml;&Auml; (Ä),\n&auml;&auml; (ä),\n&Ouml;&Ouml;
(Ö),\n&ouml;&ouml; (ö),\n&Uuml;&Uuml; (ü),\n&uuml;&uuml; (ü),\
n&szlig;&szlig; (ß).\nExample: Grüße aus Köln = Gr&uuml;&szlig;e
aus K&ouml;lnGr&uuml;&szlig;e aus K&ouml;ln.\n\n\n\nCLICK
HERECLICK HERE\nfor a complete list. When the page encoding
is\nUTF-8UTF-8, which is\nrecommended, you can also enter any
character at all, Roman,\nCyrillic, Arabic, Hebrew, Greek.
Japanese,\netc, either as numeric entities or (if you have a way
to type them) directly\nfrom the keyboard.\n\n\n\nAnd remember:
if you want to\ninclude <<, &&,\nor >> literally in text to be
displayed, you have\nto write &lt;&lt;,\n&amp;&amp;, &gt;&gt;,
respectively.\n\n\n\n\n'
```

Zauważ, że wyświetlany jest sam tekst, bez zawierających go znaczników HTML-owych.

Jak to działa?

Pierwszy etap polega na pobraniu strony. Następnie można przetworzyć surowy tekst, tak jak w kroku 3. Wynikowy obiekt `page` zawiera przetworzone informacje.

> Domyślnie używany jest parser `html.parser`, który jednak może sprawiać problemy przy wykonywaniu niektórych operacji. Na przykład działa powoli, jeśli strona jest długa, i może błędnie przetwarzać wysoce dynamiczne strony WWW. Możesz używać także innych parserów takich jak `lxml`, który funkcjonuje znacznie szybciej, lub `html5lib`, bardziej zbliżony w działaniu do przeglądarki. Mają one postać modułów zewnętrznych i trzeba je dodać do pliku *requirements.txt*.

Moduł `BeautifulSoup` umożliwia wyszukiwanie elementów HTML-owych. Metoda `.find()` znajduje pierwszy określony element, a metoda `.find_all()` zwraca listę wszystkich elementów danego rodzaju. W kroku 5. kod szukał określonego znacznika, `<a>`, o podanym atrybucie, `id=chars`. Dalej za pomocą wywołań `.next_elements` iteracyjnie pobierane były kolejne elementy do czasu znalezienia następnego znacznika `h3`, oznaczającego koniec sekcji.

Kod pobiera tekst z każdego elementu i łączy fragmenty w jeden blok. Zwróć uwagę na wyrażenie `or`, które pozwala uniknąć zapisywania tekstu `None` zwracanego, gdy element nie zawiera tekstu.

> HTML to bardzo wszechstronny język, a napisany w nim kod może mieć różną strukturę. W tej recepturze opisany jest typowy scenariusz, jednak sekcje mogą być dzielone także w inny sposób — przez grupowanie powiązanych sekcji w rozbudowanym znaczniku `<div>`, za pomocą innych elementów, a nawet w surowym tekście. Opracowanie procesu pobierania potrzebnych fragmentów ze strony WWW wymaga eksperymentów. Nie wahaj się próbować różnych rozwiązań!

Dodatkowe informacje

Jako dane wejściowe w metodach `.find()` i `.find_all()` można stosować wyrażenia regularne. Na przykład poniższe wyszukiwanie dotyczy znaczników `h2` i `h3`:

```
>>> page.find_all(re.compile('^h(2|3)'))
[<h2>Sample Web Page</h2>, <h3 id="contents">CONTENTS</h3>, <h3
id="basics">1. Creating a Web Page</h3>, <h3 id="syntax">2. HTML Syntax</
h3>, <h3 id="chars">3. Special Characters</h3>, <h3 id="convert">4.
Converting Plain Text to HTML</h3>, <h3 id="effects">5. Effects</
h3>, <h3 id="lists">6. Lists</h3>, <h3 id="links">7. Links</h3>, <h3
id="tables">8. Tables</h3>, <h3 id="viewing">9. Viewing Your Web Page</
h3>, <h3 id="install">10. Installing Your Web Page on the Internet</h3>,
<h3 id="more">11. Where to go from here</h3>]
```

Innym przydatnym parametrem metody `find` jest klasa CSS, podawana za pomocą parametru `class_`. Technikę tę przedstawiam w dalszej części książki.

Pełną dokumentację modułu Beautiful Soup znajdziesz na stronie *https://www.crummy.com/software/BeautifulSoup/bs4/doc/*.

Zobacz także

- Receptura „Instalowanie niezależnych pakietów" w rozdziale 1., „Rozpoczęcie przygody z automatyzacją". Dowiesz się tam, jak instalować zewnętrzne moduły.
- Receptura „Pobieranie stron WWW" wcześniej w tym rozdziale. Nauczysz się z niej podstaw zgłaszania żądań dotyczących stron WWW.

Crawling w sieci WWW

Z powodu natury stron z hiperłączami rozpoczynanie od znanego miejsca i podążanie za odsyłaczami do innych stron jest bardzo ważną techniką, jeśli chcesz pobierać informacje ze stron WWW.

Tu poszukasz na stronach krótkiego tekstu i wyświetlisz wszystkie akapity, które go zawierają. Przeszukiwane będą tylko strony z jednej witryny, na przykład tylko adresy URL rozpoczynające się członem www.jakaswitryna.pl. Nie będziesz podążać za odsyłaczami do zewnętrznych witryn.

Przygotowania

W tej recepturze wykorzystasz już zdobytą wiedzę. Będziesz pobierać i przetwarzać strony w poszukiwaniu odsyłaczy, a następnie kontynuować wczytywanie kolejnych stron.

> Pamiętaj, aby w trakcie crawlingu w sieci WWW określić limit liczby pobieranych stron. Bardzo łatwo jest zacząć przetwarzać zbyt dużą ich liczbę. Internet jest niemal nieograniczony, co potwierdzi każdy, kto przeglądał zasoby Wikipedii.

Używane są tu specjalnie przygotowane przykładowe strony, dostępne w witrynie wydawnictwa Helion i w repozytorium w serwisie GitHub — *https://github.com/PacktPublishing/Python-Automation-Cookbook-Second-Edition/tree/master/Chapter03/test_site*. Pobierz całą tę witrynę i uruchom dołączony skrypt:

```
$ python simple_delay_server.py
```

Będzie on udostępniał strony witryny pod adresem *http://localhost:8000*. Możesz otworzyć tę witrynę w przeglądarce. Zawiera ona prosty blog z trzema wpisami.

Większość tej witryny nie jest ciekawa (zobacz rysunek 3.1), dodałem jednak kilka akapitów zawierających słowo kluczowe `python`.

Rysunek 3.1. Zrzut przedstawiający blog

Jak to zrobić?

1. Kompletny skrypt *crawling_web_step1.py* jest dostępny w witrynie wydawnictwa Helion i w serwisie GitHub pod adresem *https://github.com/PacktPublishing/PythonAutomation-Cookbook-Second-Edition/blob/master/Chapter03/crawling_web_step1.py*. Oto jego najważniejsze fragmenty:

```
...
def process_link(source_link, text):
    logging.info(f'Pobieranie odsyłaczy ze strony {source_link}')
    parsed_source = urlparse(source_link)
    result = requests.get(source_link)
    # Obsługa błędów. Szczegółowe informacje znajdziesz w kompletnym skrypcie.
    ...
    page = BeautifulSoup(result.text, 'html.parser')
    search_text(source_link, page, text)
    return get_links(parsed_source, page)

def get_links(parsed_source, page):
    '''Pobieranie odsyłaczy ze strony'''
    links = []
    for element in page.find_all('a'):
```

```
            link = element.get('href')
            # Sprawdzanie poprawności odsyłacza. Szczegółowe informacje znajdziesz w kompletnym skrypcie.
            ...
            links.append(link)
    return links
```

2. Poszukaj wystąpień słowa `python`, aby uzyskać listę zawierających je adresów URL i akapitów. Zwróć uwagę na kilka błędów spowodowanych niedziałającymi odsyłaczami:

```
$ python crawling_web_step1.py http://localhost:8000/ -p python
Odsyłacz http://localhost:8000/: --> Krótszy artykuł z informacjami o Pythonie
Odsyłacz http://localhost:8000/files/5eabef23f63024c20389c34b94d
ee593-1.html: --> Krótszy artykuł z informacjami o Pythonie
Odsyłacz http://localhost:8000/files/33714fc865e02aeda2dabb9a42a78
7b2-0.html: --> Tu właśnie znajdują się ciekawe informacje o Pythonie
Odsyłacz http://localhost:8000/files/archive-september-2018.html: -->
Krótszy artykuł z informacjami o Pythonie
Odsyłacz http://localhost:8000/index.html: --> Krótszy artykuł z informacjami
o Pythonie
```

3. Innym ciekawym słowem, którego możesz tu poszukać, jest `krokodyl`. Sprawdź to:

```
$ python crawling_web_step1.py http://localhost:8000/ -p krokodyl
```

Jak to działa?

Przyjrzyj się każdemu komponentowi tego skryptu:

1. Oto pętla przetwarzająca wszystkie znalezione odsyłacze. Znajduje się ona w funkcji `main`:

```
def main(base_url, to_search):
    checked_links = set()
    to_check = [base_url]
    max_checks = 10
    while to_check and max_checks:
        link = to_check.pop(0)
        links = process_link(link, text=to_search)
        checked_links.add(link)
        for link in links:
            if link not in checked_links:
                checked_links.add(link)
                to_check.append(link)

        max_checks -= 1
```

Zwróć uwagę na limit pobierania, 10 stron, i kod sprawdzający, czy nowy odsyłacz, który ma zostać dodany, nie znajduje się jeszcze na liście.

> Zauważ, że te dwa elementy ograniczają zakres działania skryptu. Kod nie pobiera dwukrotnie tego samego odsyłacza i w pewnym momencie kończy pracę.

2. Ten kod z funkcji `proces_link` pobiera i przetwarza odsyłacz:

```
def process_link(source_link, text):
    logging.info(f'Pobieranie odsyłaczy ze strony {source_link}')
    parsed_source = urlparse(source_link)
    result = requests.get(source_link)
    if result.status_code != http.client.OK:
        logging.error(f'Błąd pobierania {source_link}: {result}')
        return []

    if 'html' not in result.headers['Content-type']:
        logging.info(f'Odsyłacz {source_link} nie prowadzi do strony HTML')
        return []

    page = BeautifulSoup(result.text, 'html.parser')
    search_text(source_link, page, text)

    return get_links(parsed_source, page)
```

Ten kod pobiera plik i sprawdza, czy status odpowiedzi jest poprawny (pozwala to pominąć błędy wynikające na przykład z niedziałających odsyłaczy). Ten kod sprawdza także, czy typ dokumentu (określony w nagłówku `Content-Type`) wskazuje na stronę HTML. Dokumenty PDF i w innych formatach są pomijane. Ponadto kod przetwarza surowy kod HTML na obiekt typu `BeautifulSoup`.

Kod przetwarza też źródłowy odsyłacz za pomocą wywołania `urlparse`, dlatego później, w kroku 4., można pominąć wszystkie referencje do zewnętrznych źródeł. Wywołanie `urlparse` dzieli adres URL na elementy:

```
>>> from urllib.parse import urlparse
>>> urlparse('http://localhost:8000/files/
b93bec5d9681df87e6e8d5703ed7cd81-2.html')
ParseResult(scheme='http', netloc='localhost:8000', path='/files/
b93bec5d9681df87e6e8d5703ed7cd81-2.html', params='', query='',
fragment='')
```

3. W funkcji `search_text` kod pobiera szukany tekst:

```
def search_text(source_link, page, text):
    '''Znajdowanie i wyświetlanie elementu z szukanym tekstem'''
    for element in page.find_all(text=re.compile(text, flags=re.IGNORECASE)):
        print(f'Odsyłacz {source_link}: --> {element}')
```

Ten fragment szuka podanego tekstu w przetworzonym obiekcie. Zauważ, że wyszukiwanie odbywa się z użyciem wyrażenia regularnego i uwzględniany jest tylko tekst strony. Wyświetlane są wyniki dopasowania, w tym odsyłacz ze zmiennej `source_link` określający adres URL strony z dopasowanym tekstem:

```
for element in page.find_all(text=re.compile(text)):
    print(f'Odsyłacz {source_link}: --> {element}')
```

4. Funkcja `get_links` pobiera wszystkie odsyłacze ze strony:

```
def get_links(parsed_source, page):
    '''Pobieranie odsyłaczy ze strony'''
    links = []
    for element in page.find_all('a'):
```

```
            link = element.get('href')
            if not link:
                continue

            # Unikanie odsyłaczy wewnętrznych, prowadzących do tej samej strony.
            if link.startswith('#'):
                continue

            if link.startswith('mailto:'):
                # Ignorowanie innych odsyłaczy takich jak mailto.
                # Możesz tu dodać także inne rodzaje odsyłaczy, na przykład ftp.
                continue

            # Należy zawsze akceptować lokalne odsyłacze.
            if not link.startswith('http'):
                netloc = parsed_source.netloc
                scheme = parsed_source.scheme
                path = urljoin(parsed_source.path, link)
                link = f'{scheme}://{netloc}{path}'

            # Przetwarzane są tylko odsyłacze z tej samej domeny.
            if parsed_source.netloc not in link:
                continue

            links.append(link)

        return links
```

Ten kod szuka na przetwarzanej stronie wszystkich elementów `<a>` i pobiera je, ale tylko wtedy, jeśli mają element `href` i są kompletnym adresem URL (rozpoczynającym się członem `http`) lub odsyłaczem lokalnym. Pomijane są odsyłacze, które nie są adresami URL, na przykład odsyłacze `'#'` i odsyłacze do wewnętrznych elementów strony.

> Pamiętaj, że niektóre odsyłacze, na przykład `mailto:`, działają w specjalny sposób. Kod sprawdza odsyłacze, aby uniknąć `mailto:`, ale możesz napotkać także odsyłacze `ftp` lub `irc`, przy czym są one stosowane rzadko.

Kod dodatkowo sprawdza, czy odsyłacze mają to samo źródło co pierwotny odsyłacz. Tylko wtedy są uznawane za prawidłowe. Atrybut `netloc` pozwala wykryć, czy odsyłacz pochodzi z tej samej domeny co przetwarzany adres URL uzyskany w kroku 2.

> Kod nie podąża za odsyłaczami prowadzącymi do innych adresów, na przykład *http://www.google.com*.

Na końcu odsyłacze są zwracane i przekazywane do pętli opisanej w kroku 1.

Dodatkowe informacje

Możesz zastosować dodatkowe filtry, aby na przykład odrzucać wszystkie odsyłacze z rozszerzeniem *.pdf*, ponieważ prawdopodobnie prowadzą do plików PDF:

```
# W funkcji get_links.
if link.endswith('pdf'):
    continue
```

Możesz też użyć nagłówka `Content_Type`, aby przetwarzać zwrócone obiekty w różny sposób. Aby sprawdzić ten nagłówek, trzeba zgłosić żądanie. To oznacza, że jeśli zastosujesz tę technikę, nie będzie można pominąć odsyłaczy bez zażądania strony. Dla otrzymanego dokumentu PDF (`Content-Type: application/pdf`) nie będzie dostępny poprawny obiekt `response.text`, który można byłoby przetworzyć. Jednak dokumenty w formacie PDF można przetwarzać w inny sposób. To samo dotyczy też dokumentów innego typu, na przykład plików CSV (`Content-Type: text/csv`) lub plików ZIP (`Content-Type: application/zip`), które mogą wymagać dekompresji. Przetwarzanie plików tego rodzaju omawiam dalej.

Zobacz także

- Receptura „Pobieranie stron WWW" we wcześniejszej części rozdziału. Nauczysz się z niej podstaw żądania stron WWW.
- Receptura „Parsowanie kodu HTML" we wcześniejszej części rozdziału. Dowiesz się z niej, jak przetwarzać elementy w kodzie HTML.

Subskrybowanie kanałów informacyjnych

Kanały RSS są prawdopodobnie największą zagadką internetu. Lata ich świetności przypadły na pierwsze dziesięciolecie naszego wieku. Kanały RSS umożliwiają łatwe subskrybowanie informacji z witryn. Są dostępne w wielu witrynach i niezwykle przydatne.

RSS jest sposobem udostępniania serii uporządkowanych materiałów (zwykle artykułów, ale też odcinków podcastów lub nagrań z serwisy YouTube) wraz z czasem ich publikacji. Pozwala to łatwo stwierdzić, które artykuły pojawiły się od momentu ostatniego sprawdzania ich dostępności, a także publikować ustrukturyzowane dane na temat materiałów, na przykład tytuły i streszczenia.

W tej recepturze przedstawiam moduł `feedparser` i pokazuję, jak pobierać dane z kanałów RSS.

> RSS nie jest jedynym dostępnym formatem informacyjnym. Dostępny jest też format Atom, który jest jednak podobny do RSS. Moduł `feedparser` potrafi przetwarzać także dane w formacie Atom, dlatego dane w obu formatach można traktować tak samo.

Przygotowania

Dodaj moduł `feedparser` do pliku *requirements.txt* i ponownie zainstaluj zależności:

```
$ echo "feedparser==5.2.1" >> requirements.txt
$ pip install -r requirements.txt
```

Adresy URL kanałów informacyjnych znajdziesz na prawie wszystkich stronach z publikacjami, w tym na blogach, w serwisach informacyjnych, na stronach podcastów itd. Nieraz znalezienie takich adresów jest bardzo łatwe, jednak czasem są one nieco schowane. Szukaj tekstu *feed* lub *RSS*.

Większość witryn gazet i agencji informacyjnych udostępnia kanały RSS uporządkowane tematycznie. Tu będziesz przetwarzać kanał z głównej strony dziennika **New York Times** — *https://rss.nytimes.com/services/xml/rss/nyt/HomePage.xml*. Na stronie głównej kanałów informacyjnych (*https://archive.nytimes.com/www.nytimes.com/services/xml/rss/index.html*) dostępnych jest ich więcej.

> Pamiętaj, że kanały informacyjne mogą podlegać warunkom użytkowania. W witrynie dziennika New York Times te warunki są opisane na końcu strony głównej kanałów informacyjnych.

Używany tu kanał informacyjny zmienia się często, co oznacza, że wpisy, jakie zobaczysz, będą inne niż w przykładach z tej książki.

Jak to zrobić?

1. Zaimportuj moduł `feedparser`, a także moduły `datetime`, `delorean` i `requests`:

```
>>> import feedparser
>>> import datetime
>>> import delorean
>>> import requests
```

2. Uruchom przetwarzanie kanału (informacje będą pobierane automatycznie) i sprawdź czas ostatniej aktualizacji. Informacje o kanale, na przykład jego nazwę, można pobrać z atrybutu `feed`:

```
>>> rss = feedparser.parse('http://rss.nytimes.com/services/xml/
rss/nyt/HomePage.xml')
>>> rss.channel.updated
Friday, 24 Jan 2020 19:42:27 +0000'
```

3. Pobierz wpisy opublikowane nie dawniej niż sześć godzin temu:

```
>>> time_limit = delorean.parse(rss.channel.updated) - datetime.
timedelta(hours=6)
>>> entries = [entry for entry in rss.entries if delorean.
parse(entry.published) > time_limit]
```

4. Niektóre ze zwróconych wpisów będą starsze niż sześć godzin:

```
>>> len(entries)
28
>>> len(rss.entries)
54
```

5. Pobierz informacje o wpisach, na przykład tytuły (`title`). Kompletny adres URL wpisu jest dostępny jako `link`. Sprawdź informacje dostępne dla danego wpisu:

```
>>> entries[18]['title']
'These People Really Care About Fonts'
>>> entries[18]['link']
'https://www.nytimes.com/2020/01/24/style/typography-font-design.
html?emc=rss&partner=rss'
>>> requests.get(entries[18].link)
<Response [200]>
```

Jak to działa?

Przetworzony obiekt `feed` zawiera informacje o wpisach, a także ogólne dane na temat samego kanału, na przykład czas aktualizacji. Informacje o kanale można pobrać za pomocą atrybutu `feed`:

```
>>> rss.feed.title
'NYT > Top Stories'
```

Każdy wpis działa jak słownik, dlatego możesz łatwo pobrać dostępne pola. Możesz ich używać także za pomocą atrybutów, jednak traktowanie ich jak kluczy pozwala pobrać listę wszystkich dostępnych pól:

```
>>> entries[5].keys()
dict_keys(['title', 'title_detail', 'links', 'link', 'id', 'guidislink',
'media_content', 'summary', 'summary_detail', 'media_credit', 'credit',
'content', 'authors', 'author', 'author_detail', 'published', 'published_
parsed', 'tags'])
```

Podstawowa strategia pracy z kanałami informacyjnymi polega na tym, aby je przetwarzać i sprawdzać wpisy, szybko ustalając (na przykład na podstawie *opisu* lub *streszczenia*), czy są warte uwagi. Jeśli wpis wydaje się interesujący, można go pobrać w całości za pomocą pola `link`. Aby uniknąć ponownego sprawdzania wpisów, zapisz datę ostatniej publikacji i następnym razem zbadaj tylko nowsze wpisy.

Dodatkowe informacje

Kompletną dokumentację modułu `feedparser` znajdziesz na stronie *https://pythonhosted.org/feedparser/*.

W różnych kanałach dostępne mogą być inne informacje. W kanale dziennika *New York Times* znajduje się pole `tag` z informacjami o tagach, jednak nie wszędzie tak jest. Wpisy zawsze mają przynajmniej tytuł, opis i odsyłacz.

Kanały RSS są też doskonałym narzędziem do zarządzania własnym zestawem źródeł informacji. Dostępne są świetne czytniki kanałów informacyjnych, które w tym pomagają.

Zobacz także

- Receptura „Instalowanie niezależnych pakietów" z rozdziału 1., „Rozpoczęcie przygody z automatyzacją". Poznasz w niej podstawy instalowania modułów zewnętrznych.
- Receptura „Pobieranie stron WWW" we wcześniejszej części rozdziału. Dowiesz się z niej, jak zgłaszać żądania i pobierać strony ze zdalnych serwerów.

Dostęp do internetowych API

W internecie można tworzyć rozbudowane interfejsy, umożliwiające zaawansowane interakcje z użyciem protokołu HTTP. Najczęściej używanym interfejsem jest API REST wykorzystujący format JSON. Tego rodzaju tekstowe interfejsy są łatwe do zrozumienia i zaprogramowania, a także **niezależne od języka**, co oznacza, że są dostępne w każdym języku programowania z modułem *klienckim* protokołu HTTP, w tym oczywiście w Pythonie.

Używane są też formaty inne niż JSON, na przykład XML. Jednak format JSON jest bardzo prosty i czytelny oraz łatwo jest go przekształcić na słowniki z Pythona (lub z innych języków programowania). Obecnie JSON jest zdecydowanie najczęściej używanym formatem API REST. Więcej na temat tego formatu dowiesz się ze strony *https://www.json.org/*.

Ścisła definicja modelu REST wymaga specyficznego opisu, jednak nieformalnie można stwierdzić, że system REST opisuje zasoby za pomocą adresów URL zgodnych z protokołem HTTP. To oznacza, że każdy adres URL reprezentuje określony zasób, na przykład artykuł w gazecie lub nieruchomość w witrynie biura nieruchomości. Takimi zasobami można operować za pomocą metod protokołu HTTP (metoda `GET` wyświetla zasób, `POST` tworzy go, `PUT` i `PATCH` służą do edycji, a `DELETE` do usuwania zasobów).

Kompletne interfejsy REST muszą mieć określone cechy. Istnieją sposoby tworzenia interfejsów, które nie ograniczają się do korzystania z protokołu HTTP. Więcej na ten temat dowiesz się ze strony *https://codewords.recurse.com/issues/five/what-restful-actually-means*.

Używanie modułu `requests` do interfejsów REST jest bardzo proste, ponieważ zapewniają one wbudowaną obsługę formatu JSON.

Przygotowania

Do pokazania, jak korzystać z API REST, posłuży tu przykładowa witryna *https://jsonplaceholder.typicode.com/*. Symuluje ona standardową witrynę z wpisami, komentarzami i innymi zasobami. W kodzie uwzględnisz wpisy i komentarze. Oto używane adresy URL:

```
# Zbiór wszystkich wpisów.
/posts
# Jeden wpis. X oznacza identyfikator wpisu.
/posts/X
# Komentarze dotyczące wpisu X.
/posts/X/comments
```

Witryna zwraca poprawny wynik dla każdego z tych adresów. To naprawdę wygodne!

> Ponieważ jest to witryna testowa, nie utworzysz w niej danych, ale otrzymasz poprawne odpowiedzi.

Jak to zrobić?

1. Zaimportuj moduł `requests`:

```
>>> import requests
```

2. Pobierz listę wszystkich wpisów i wyświetl najnowszy z nich:

```
>>> result = requests.get('https://jsonplaceholder.typicode.com/posts')
>>> result
<Response [200]>
>>> result.json()
# Lista 100 wpisów, TU POMINIĘTA.
>>> result.json()[-1]
{'userId': 10, 'id': 100, 'title': 'at nam consequatur ea labore
ea harum', 'body': 'cupiditate quo est a modi nesciunt soluta\
nipsa voluptas error itaque dicta in\nautem qui minus magnam et
distinctio eum\naccusamus ratione error aut'}
```

3. Utwórz nowy wpis. Sprawdź adres URL nowo utworzonego zasobu. Pokazane wywołanie zwraca też sam zasób:

```
>>> new_post = {'userId': 10, 'title': 'Tytuł', 'body':
                'Jakieś informacje'}
>>> result = requests.post('https://jsonplaceholder.typicode.com/posts',
json=new_post)
>>> result
<Response [201]>
>>> result.json()
{'userId': 10, 'title': 'Tytuł', 'body': 'Jakieś informacje', 'id': 101}
>>> result.headers['Location']
'http://jsonplaceholder.typicode.com/posts/101'
```

Żądanie POST służące do utworzenia zasobu zwraca kod 201. Jest to poprawny status dla tworzenia zasobów.

4. Pobierz istniejący wpis za pomocą metody GET:

```
>>> result = requests.get('https://jsonplaceholder.typicode.com/posts/2')
>>> result
<Response [200]>
>>> result.json()
{'userId': 1, 'id': 2, 'title': 'qui est esse', 'body': 'est
rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae
ea dolores neque\nfugiat blanditiis voluptate porro vel nihil
molestiae ut reiciendis\nqui aperiam non debitis possimus qui
neque nisi nulla'}
```

5. Użyj metody PATCH do zaktualizowania wartości. Sprawdź zwrócony zasób:

```
>>> update = {'body': 'Nowa treść'}
>>> result = requests.patch('https://jsonplaceholder.typicode.com/posts/2',
                            json=update)
>>> result
<Response [200]>
>>> result.json()
{'userId': 1, 'id': 2, 'title': 'qui est esse', 'body': 'Nowa treść'}
```

Jak to działa?

Zwykle używane są dwa rodzaje zasobów — pojedyncze (*https://jsonplaceholder.typicode.com/posts/X*) i kolekcje (*https://jsonplaceholder.typicode.com/posts*).

- Dla kolekcji można używać metod GET, aby pobrać wszystkie elementy kolekcji, i POST, by tworzyć nowe zasoby.
- Pojedyncze elementy akceptują żądania GET (pobieranie elementu), PUT i PATCH (edycja) oraz DELETE (usuwanie elementu).

Wszystkie metody HTTP można stosować za pomocą modułu `requests`. W poprzednich recepturach używane było wywołanie `.get()`, jednak dostępne są też wywołania `.post()`, `.patch()`, `.put()` i `.delete()`.

Zwrócony obiekt odpowiedzi ma metodę `.json()`, która dekoduje wynik z formatu JSON.

Jeśli chcesz wysłać informację, dostępny jest argument `json`. W tej technice słownik jest kodowany do formatu JSON i przesyłany na serwer. Dane muszą być zgodne z formatem zasobu. W przeciwnym razie może zostać zgłoszony błąd.

> Metody GET i DELETE nie wymagają podawania danych, ale gdy używasz metod PATCH, PUT i POST, musisz przekazać dane w treści żądania.

Zwracany jest wskazany zasób, a jego adres URL jest dostępny w nagłówku. Jest to przydatne, gdy chcesz utworzyć nowy zasób, ponieważ wcześniej jego adres URL nie jest określony.

> Różnica między metodami `PATCH` i `PUT` polega na tym, że ta druga zastępuje cały zasób, natomiast pierwsza wykonuje tylko częściową aktualizację.

Dodatkowe informacje

API REST dają bardzo duże możliwości, ale też są wysoce zróżnicowane. Aby poznać szczegóły, zajrzyj do dokumentacji określonego API.

Zobacz także

- Receptura „Pobieranie stron WWW" we wcześniejszej części rozdziału. Poznasz w niej podstawy żądania stron WWW.
- Receptura „Instalowanie niezależnych pakietów" w rozdziale 1., „Rozpoczęcie przygody z automatyzacją". Poznasz w niej podstawy instalowania zewnętrznych modułów.

Interakcje z formularzami

Standardowym elementem na stronach WWW są formularze. Umożliwiają one przesyłanie wartości na strony WWW, na przykład w celu dodania nowego komentarza do wpisu z bloga lub dokonania zakupu.

Przeglądarki wyświetlają formularze w taki sposób, że możesz wprowadzić w nich wartości, a następnie za pomocą jednej czynności je przesłać, wciskając przycisk z napisem *Prześlij* lub podobnym. W tej recepturze zobaczysz, jak wykonać taką czynność programowo.

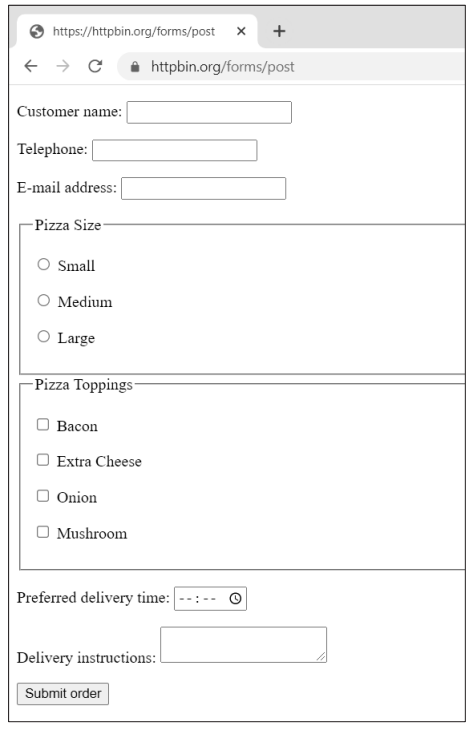
> Pamiętaj, że przesyłanie danych do witryny jest zwykle bardziej delikatną operacją niż pobieranie informacji. Na przykład wysyłanie automatycznych komentarzy do witryny jest **spamowaniem**. To oznacza, że automatyzacja przesyłania danych jest trudniejsza, ponieważ trzeba uwzględnić zabezpieczenia. Dobrze się zastanów, czy to, co chcesz zrobić, jest dozwolone i etyczne.

Przygotowania

W tej recepturze wykorzystasz serwer testowy *https://httpbin.org/forms/post*. Umożliwia on wysłanie formularza testowego i zwraca przekazane informacje.

> Zauważ, że adres URL *https://httpbin.org/forms/post* powoduje wyświetlenie formularza, jednak wewnętrznie do przesyłania informacji używany jest adres *https://httpbin.org/post*. W tej recepturze stosowane są oba podane adresy.

Na rysunku 3.2 pokazany jest przykładowy formularz do zamawiania pizzy.



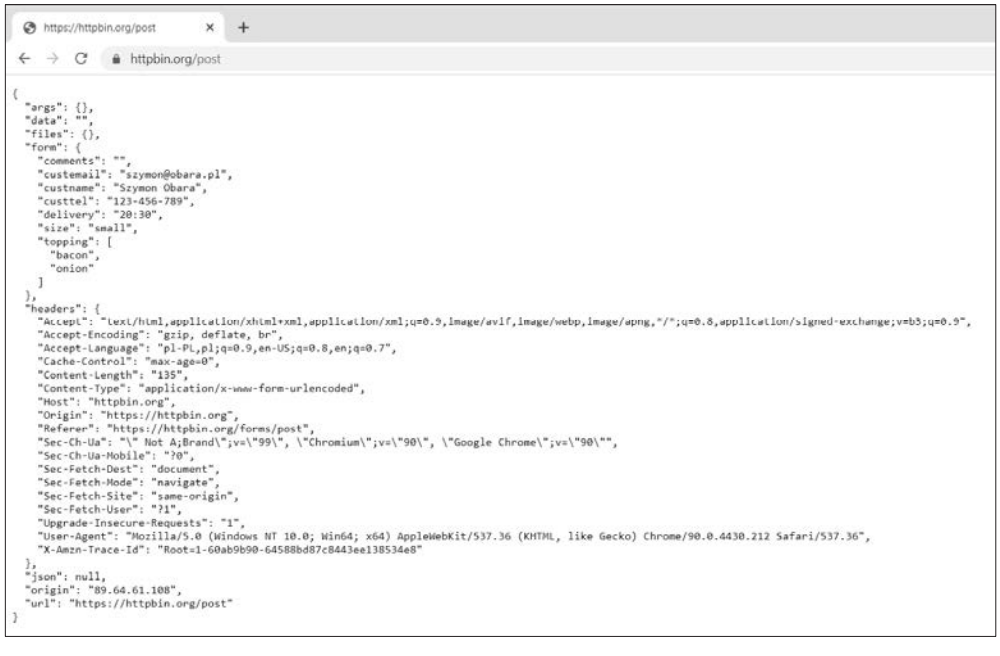
Rysunek 3.2. Wyświetlony formularz

Możesz ręcznie uzupełnić formularz i zobaczyć, że zwrócone zostaną informacje w formacie JSON. Obejmują one także dodatkowe informacje, na przykład o tym, jaka przeglądarka jest używana.

Na rysunku 3.3 pokazany jest generowany frontend formularza internetowego.

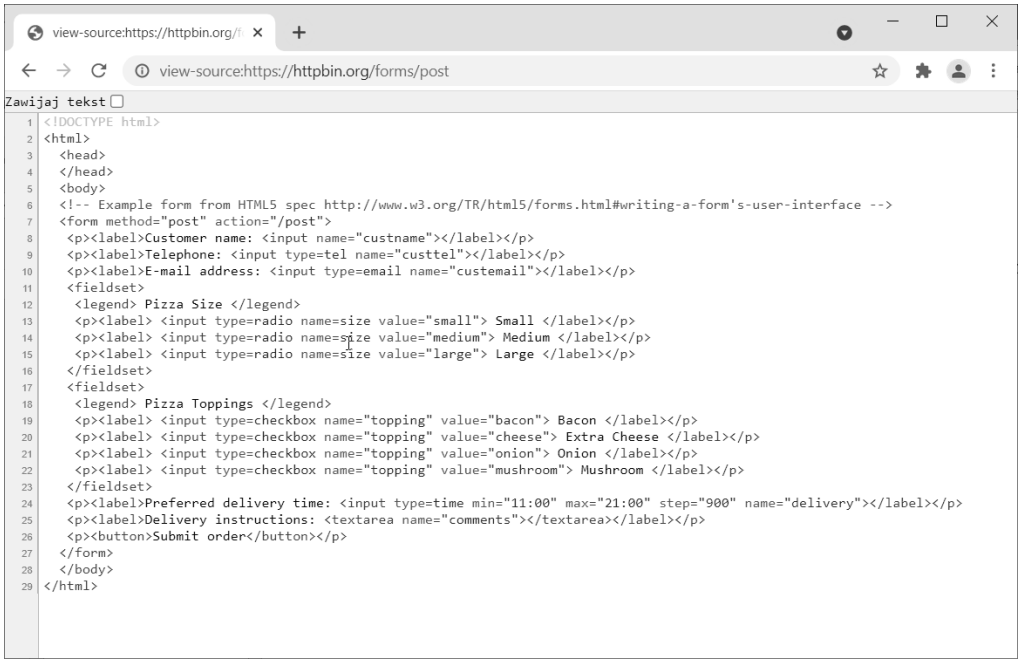
Zrzut z rysunku 3.4 przedstawia backend generowanego formularza internetowego.

Rysunek 3.3. Wypełniony formularz



Rysunek 3.4. Zwrócone dane w formacie JSON

Trzeba przeanalizować kod HTML, aby zobaczyć, jakie dane można wprowadzać w formularzu. Na rysunku 3.5 pokazany jest kod źródłowy strony.



Rysunek 3.5. Kod źródłowy

Przejrzyj nazwy elementów na wprowadzane dane: `custname`, `custtel`, `custemail`, `size` (przycisk opcji), `topping` (pola wyboru), `delivery` (czas) i `comments`.

Jak to zrobić?

1. Zaimportuj moduły `requests`, `BeautifulSoup` i `re`:

```
>>> import requests
>>> from bs4 import BeautifulSoup
>>> import re
```

2. Pobierz stronę z formularzem, przetwórz jej zawartość i wyświetl pola na dane wejściowe. Zauważ, że adres URL do przesyłania danych to `/post` (a nie `/forms/post`):

```
>>> response = requests.get('https://httpbin.org/forms/post')
>>> page = BeautifulSoup(response.text)
>>> form = page.find('form')
>>> {field.get('name') for field in form.find_all(re.
compile('input|textarea'))}
{'delivery', 'topping', 'size', 'custemail', 'comments',
'custtel', 'custname'}
```

Element `textarea` to poprawne pole na dane wejściowe zdefiniowane w formacie HTML.

3. Przygotuj dane, które zostaną przesłane jako słownik. Upewnij się, że wartości są zdefiniowane w następującym formacie:

```
>>> data = {'custname': "Szymon Obara", 'custtel': '123-456-
789', 'custemail': 'szymon@obara.pl', 'size': 'small', 'topping':
['bacon', 'onion'], 'delivery': '20:30', 'comments': ''}
```

4. Prześlij wartości i sprawdź, czy odpowiedź jest identyczna jak w przeglądarce:

```
>>> response = requests.post('https://httpbin.org/post', data)
>>> response
<Response [200]>
>>> response.json()
{'args': {}, 'data': '', 'files': {}, 'form': {'comments': '',
'custemail': 'szymon@obara.pl', 'custname': "Szymon Obara",
'custtel': '123-456-789', 'delivery': '20:30', 'size': 'small',
'topping': ['bacon', 'onion']}, 'headers': {'Accept': '*/*',
'Accept-Encoding': 'gzip, deflate', 'Connection': 'close',
'Content-Length': '140', 'Content-Type': 'application/x-wwwform-urlencoded', 'Host':
'httpbin.org', 'User-Agent': 'pythonrequests/2.22.0'}, 'json': None, 'origin':
'89.100.17.159', 'url':
'https://httpbin.org/post'}
```

Jak to działa?

Moduł `requests` koduje i przesyła dane w skonfigurowanym formacie. Domyślnie przesyła dane z żądania `POST` w formacie `application/x-www-form-urlencoded`.

> Porównaj działanie modułu requests z rozwiązaniem z receptury „Dostęp do internetowych API", gdzie dane są wysyłane bezpośrednio w formacie JSON z użyciem argumentu `json`. Oznacza to użycie nagłówka `Content-Type application/json` zamiast `application/x-www-form-urlencoded`.

Najważniejsze, aby uwzględnić format formularza i dozwolone wartości. Jeśli prześlesz błędne dane, wystąpi błąd — zwykle 400 — informujący o problemie po stronie klienta.

Dodatkowe informacje

Oprócz zapewnienia zgodności z formatem formularza i przesłania poprawnych wartości głównym problemem są różne sposoby blokowania spamu i nadużyć.

Przed przesłaniem formularza często musisz potwierdzić jego pobranie. Ma to chronić przed przesyłaniem wielu formularzy i atakami **CSRF** (ang. *Cross-Site Request Forgery*).

Ataki CSRF polegają na przesyłaniu szkodliwych żądań z jednej strony do innej. Wykorzystywany jest przy tym fakt, że przeglądarka jest uwierzytelniona w tej innej witrynie. Tego typu ataki stanowią poważny problem. Może Ci się wydawać, że wchodzisz na stronę poświęconą uroczym szczeniaczkom, a w rzeczywistości witryna wykorzystuje to, że jesteś zalogowany na stronie banku, i wykonuje operacje finansowe w Twoim imieniu, na przykład przesyła Twoje oszczędności na inne konto. Dobry opis ataków CSRF znajdziesz na stronie *https://stackoverflow.com/a/33829607*. Nowe rozwiązania w przeglądarkach pomagają automatycznie chronić się przed takimi atakami.

Aby pobrać token, musisz najpierw wczytać formularz (tak jak w recepturze). Następnie należy znaleźć wartość tokenu CSRF i odesłać ją. Ten token może być dostępny pod różnymi nazwami; tu używana jest tylko przykładowa nazwa:

```
>>> form.find(attrs={'name': 'token'}).get('value')
'ABCEDF12345'
```

Zobacz także

- Receptura „Pobieranie stron WWW" we wcześniejszej części rozdziału. Poznasz w niej podstawy żądania stron WWW.
- Receptura „Parsowanie kodu HTML" we wcześniejszej części rozdziału. Dowiesz się z niej, jaką strukturę mają informacje zwracane przez serwer.

Używanie pakietu Selenium do obsługi zaawansowanych interakcji

Czasem potrzebne są zaawansowane rozwiązania. Pakiet Selenium jest narzędziem umożliwiającym automatyzację w przeglądarkach internetowych. Został opracowany jako moduł do obsługi testów automatycznych, jednak umożliwia także automatyczne interakcje z witryną.

Selenium potrafi kontrolować przeglądarki Safari, Chrome, Firefox, Internet Explorer i Microsoft Edge, jednak każda z nich wymaga zainstalowania specjalnego sterownika. W tym podrozdziale używana jest przeglądarka Chrome.

Przygotowania

Trzeba zainstalować sterownik odpowiedni dla przeglądarki Chrome — `chromedriver`. Jest on dostępny na stronie *https://sites.google.com/a/chromium.org/chromedriver/*. Istnieją wersje dostosowane do większości systemów operacyjnych. Konieczne jest też zainstalowanie przeglądarki Chrome — *https://www.google.com/chrome/*.

Dodaj moduł `selenium` do pliku *requirements.txt* i zainstaluj potrzebne moduły:

```
$ echo "selenium==3.141.0" >> requirements.txt
$ pip install -r requirements.txt
```

Jak to zrobić?

1. Zaimportuj pakiet Selenium, uruchom przeglądarkę i wczytaj stronę z formularzem. Otworzy się strona z informacjami o tych operacjach:

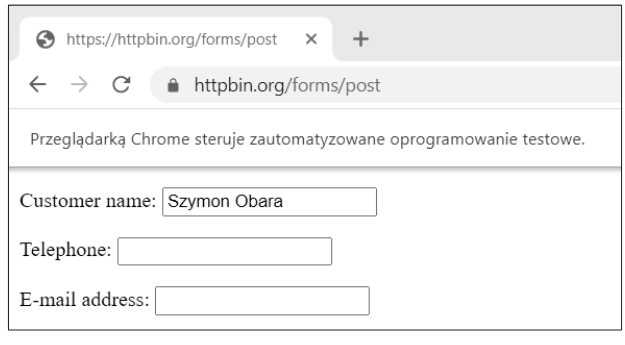
```
>>> from selenium import webdriver
>>> browser = webdriver.Chrome()
>>> browser.get('https://httpbin.org/forms/post')
```

> W Chrome wyświetlana jest wtedy informacja o kontrolowaniu przeglądarki przez zautomatyzowane oprogramowanie testowe.

2. Dodaj wartość w polu *Customer name*. Pamiętaj, że nazwa tego pola to `custname`:

```
>>> custname = browser.find_element_by_name("custname")
>>> custname.clear()
>>> custname.send_keys("Szymon Obara")
```

Formularz zostanie zaktualizowany, co ilustruje rysunek 3.6.



Rysunek 3.6. Formularz jest wypełniany automatycznie

3. Ustaw wielkość pizzy na `medium`:

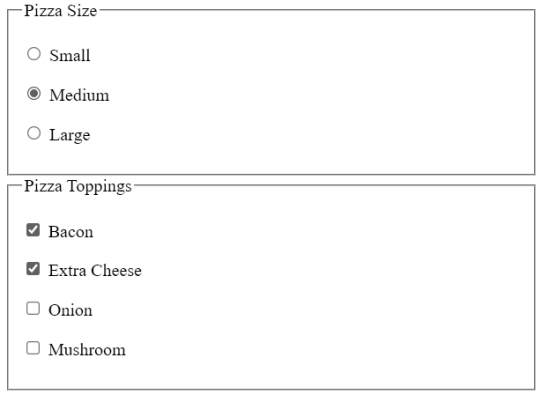
```
>>> for size_element in browser.find_elements_by_name("size"):
...     if size_element.get_attribute('value') == 'medium':
...         size_element.click()
...
>>>
```

To spowoduje zaznaczenie przycisku opcji *Pizza Size*.

4. Wybierz dodatki `bacon` i `cheese`:

```
>>> for topping in browser.find_elements_by_name('topping'):
...     if topping.get_attribute('value') in ['bacon', 'cheese']:
...         topping.click()
...
>>>
```

Te pola wyboru będą widoczne jako zaznaczone, co ilustruje rysunek 3.7.

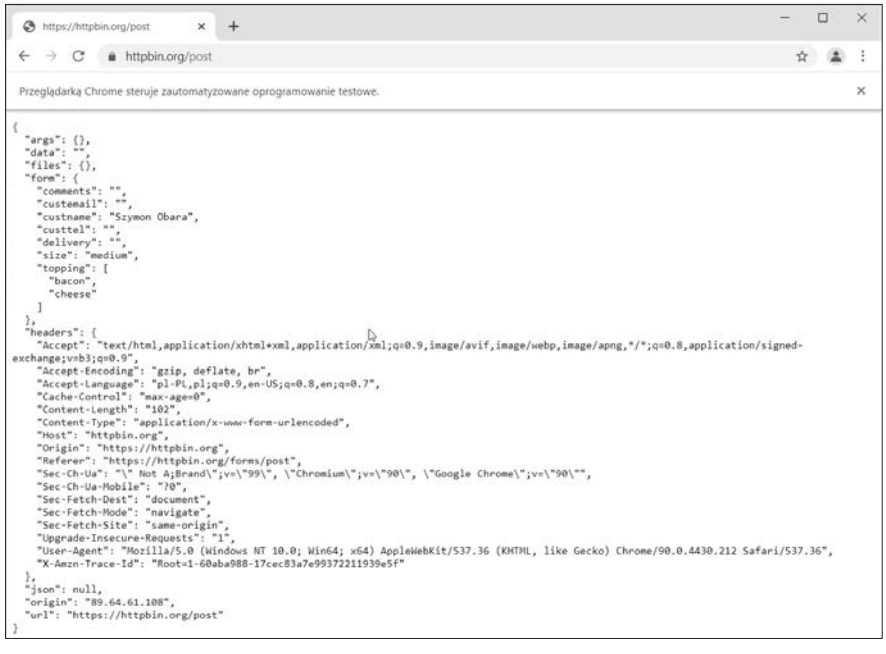


Rysunek 3.7. Formularz z zaznaczonymi polami

5. Prześlij formularz. Strona zostanie wysłana i przeglądarka wyświetli wynik:

```
>>> browser.find_element_by_tag_name('form').submit()
```

Formularz zostanie przesłany i wyświetlony zostanie wynik odesłany przez serwer (zobacz rysunek 3.8).



Rysunek 3.8. Zwrócone informacje w formacie JSON

6. Zamknij przeglądarkę:

```
>>> browser.quit()
```

Jak to działa?

W kroku 1. w punkcie „Jak to zrobić?" pokazuję, jak uruchomić przeglądarkę za pomocą Selenium i przejść pod określony adres URL.

Selenium działa podobnie jak Beautiful Soup — należy wybrać element i wykonać na nim operacje. Metody wyboru elementów w Selenium funkcjonują w zbliżony sposób jak w Beautiful Soup. Najczęściej używane z nich to `find_element_by_id`, `find_element_by_class_name`, `find_element_by_name`, `find_element_by_tag_name` i `find_element_by_css_selector`.

Istnieją analogiczne funkcje `find_elements_by_X` (na przykład `find_elements_by_tag_name`, `find_elements_by_name` i inne), które na podstawie atrybutów zwracają listy zamiast tylko pierwszego znalezionego elementu. Są one przydatne także do sprawdzania, czy dany element istnieje. Jeśli nie istnieje, wywołanie `find_element` zgłosi błąd, a `find_elements` zwróci pustą listę.

Dane dotyczące elementów można pobrać za pomocą wywołań `.get_attribute()` (w przypadku atrybutów HTML, na przykład wartości elementów formularza) i `.text`.

Operacje na elementach można wykonywać, symulując wciśnięcia klawiszy odpowiadające wprowadzaniu tekstu (metoda `.send_keys()`), kliknięcia myszą (metoda `.click()`) i przesyłanie formularza (metoda `.submit()`). Zauważ, że metoda `.click()` powoduje dodanie lub usunięcie zaznaczenia w taki sam sposób jak kliknięcie myszą.

W kroku 6. przeglądarka jest zamykana.

Dodatkowe informacje

Dokumentacja modułu Selenium dla Pythona jest dostępna na stronie *http://selenium-python.readthedocs.io/*.

Na temat każdego elementu można pobrać dodatkowe informacje, na przykład za pomocą metod `.is_displayed()` i `.is_selected()`. Tekst można przeszukiwać za pomocą metod `.find_element_by_link_text()` i `.find_element_by_partial_link_text()`.

Czasem otwieranie przeglądarki jest niewygodne. Zamiast tego można uruchomić ją w trybie bezobsługowym i operować nią z poziomu kodu:

```
>>> from selenium.webdriver.chrome.options import Options
>>> chrome_options = Options()
>>> chrome_options.add_argument("--headless")
>>> browser = webdriver.Chrome(chrome_options=chrome_options)
>>> browser.get('https://httpbin.org/forms/post')
```

Strona nie zostanie wyświetlona, ale za pomocą poniższego wiersza można zapisać jej zrzut:

```
>>> browser.save_screenshot('screenshot.png')
```

Zobacz także

- Receptura „Parsowanie kodu HTML" we wcześniejszej części rozdziału. Opisuję tam, jak przetwarzać elementy z kodu HTML.
- Receptura „Interakcje z formularzami" we wcześniejszej części rozdziału. Poznasz tam inne sposoby pracy z formularzami.

Dostęp do stron chronionych hasłem

Niektóre strony WWW nie są publicznie dostępne, ponieważ są w jakiś sposób zabezpieczone. Najprostsza technika ochrony polega na zastosowaniu podstawowego uwierzytelniania HTTP wbudowanego w prawie każdy serwer WWW i bazującego na modelu użytkownik-hasło.

Przygotowania

Uwierzytelnianie tego rodzaju można przetestować w witrynie *https://httpbin.org*.

Dostępna jest w niej ścieżka `/basic-auth/{użytkownik}/{hasło}`, która wymusza uwierzytelnienie przez podanie nazwy użytkownika i hasła. Dzięki przykładowi lepiej zrozumiesz działanie uwierzytelniania.

Jak to zrobić?

1. Zaimportuj moduł `requests`:

```
>>> import requests
```

2. Prześlij na adres URL żądanie `GET` z prawidłowymi danymi uwierzytelniającymi. Tu dane uwierzytelniające to `user` i `psswd`:

```
>>> requests.get('https://httpbin.org/basic-auth/user/psswd',
                 auth=('user', 'psswd'))
<Response [200]>
```

3. Teraz użyj nieprawidłowych danych uwierzytelniających, aby witryna zwróciła kod statusu 401 (`unauthorized`):

```
>>> requests.get('https://httpbin.org/basic-auth/user/psswd',
                 auth=('user', 'niepoprawne'))
<Response [401]>
```

4. Dane uwierzytelniające można też przesyłać bezpośrednio w adresie URL. Nazwę użytkownika i hasło należy rozdzielić dwukropkiem, a przed adresem trzeba dodać symbol `@`:

```
>>> requests.get('https://user:psswd@httpbin.org/basic-auth/user/psswd')
<Response [200]>
>>> requests.get('https://user:niepoprawne@httpbin.org/basic-auth/user/psswd')
<Response [401]>
```

Jak to działa?

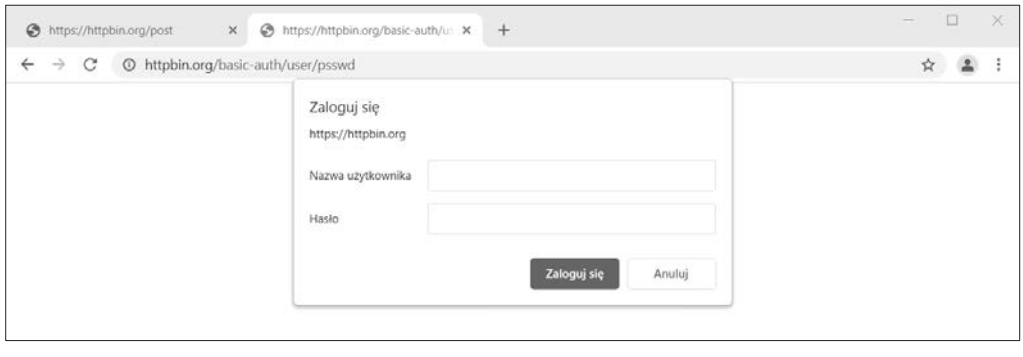
Podstawowe uwierzytelnianie HTTP jest obsługiwane wszędzie, a korzystanie z niego za pomocą modułu `requests` jest proste.

W krokach 2. i 4. z punktu „Jak to zrobić?" pokazuję, jak przekazać poprawne hasło. W kroku 3. ilustruję, co się dzieje, gdy hasło jest nieprawidłowe.

> Pamiętaj, aby zawsze używać protokołu HTTPS, co pozwoli zapewnić, że przesyłane hasło pozostanie prywatne. Gdy używasz HTTP, hasło jest przesyłane w otwartej komunikacji przez internet i może zostać przechwycone przez nasłuchujące jednostki.

Dodatkowe informacje

Dodanie nazwy użytkownika i hasła do adresu URL działa także w przeglądarce. Spróbuj bezpośrednio uzyskać dostęp do strony, aby wyświetlić okienko z prośba o nazwę użytkownika i hasło (zobacz rysunek 3.9).



Rysunek 3.9. Okienko na dane uwierzytelniające użytkownika

Gdy adres URL zawiera nazwę użytkownika i hasło, na przykład *https://user:psswd@httpbin.org/basic-auth/user/psswd*, okienko się nie pojawia i użytkownik jest uwierzytelniany automatycznie.

Jeśli chcesz uzyskać dostęp do wielu stron, możesz utworzyć sesję w module `requests` i zapisać dane uwierzytelniające, aby nie musieć ich podawać dla każdej strony:

```
>>> s = requests.Session()
>>> s.auth = ('user', 'psswd')
>>> s.get('https://httpbin.org/basic-auth/user/psswd')
<Response [200]>
```

Zobacz także

- Receptura „Pobieranie stron WWW" we wcześniejszej części rozdziału. Poznasz w niej podstawy żądania stron WWW.
- Receptura „Dostęp do internetowych API" we wcześniejszej części rozdziału. Dowiesz się z niej, jak używać API niewymagających uwierzytelniania.

Przyspieszanie pobierania informacji ze stron WWW

Większość czasu w trakcie pobierania informacji ze stron WWW zwykle zajmuje czekanie. Żądanie w celu przetworzenia jest przesyłane z komputera na zdalny serwer, a do czasu wygenerowania odpowiedzi i jej zwrócenia nie możesz zrobić nic więcej.

W trakcie wykonywania receptur z tej książki zauważysz, że wywołania z modułu `requests` zwykle wymagają jedno- lub dwusekundowego oczekiwania. Jednak w tym czasie komputery mogą wykonywać inne zadania, na przykład zgłaszać więcej żądań. W tej recepturze zobaczysz, jak równolegle pobierać listę stron i oczekiwać, aż wszystkie staną się dostępne. Celowo używany jest tu wolny serwer, aby pokazać, dlaczego warto odpowiednio przygotować skrypt.

Przygotowania

Kod będzie skanował strony witryny i szukał na nich słów kluczowych. Wykorzystasz tu możliwości obiektów `Future` z Pythona 3 do jednoczesnego pobierania wielu stron.

Obiekt `Future` reprezentuje obietnicę wartości. To oznacza, że natychmiast otrzymujesz obiekt, a kod jest wykonywany w tle. Dopiero gdy za pomocą wywołania `.result()` bezpośrednio zażądasz wyniku, kod zacznie na niego oczekiwać.

> Jeśli wynik w momencie wspomnianego wywołania jest już dostępny, przyspiesza to pracę kodu. Wyobraź sobie, że wkładasz pranie do pralki i zabierasz się za inne prace. Możliwe, że zanim je skończysz, ukończone zostanie także pranie.

Aby wygenerować obiekt `Future`, potrzebny jest **wykonawca**, czyli kod działający w tle. Po utworzeniu wykonawcy przekaż do niego funkcję i parametry, aby otrzymać obiekt `Future`. Pobieranie wyniku można odkładać tak długo, jak to konieczne, co pozwala wygenerować kilka obiektów `Future` jeden po drugim. Następnie można poczekać na ukończenie powiązanych z nimi prac i wykonywać je równolegle. Jest to alternatywa do tworzenia jednego zadania, czekania na jego ukończenie, tworzenia kolejnego zadania itd.

Wykonawcę można wygenerować na kilka sposobów. W tej recepturze zastosowany jest używający wątków wykonawca typu `ThreadPoolExecutor`.

Wykorzystasz tu gotowy przykład dostępny w witrynie wydawnictwa Helion i w serwisie GitHub *https://github.com/PacktPublishing/Python-Automation-Cookbook-Second-Edition/tree/master/Chapter03/test_site*. Pobierz całą witrynę i uruchom powiązany skrypt:

```
$ python simple_delay_server.py -d 2
```

Skrypt udostępnia witrynę pod adresem URL *http://localhost:8000*. Możesz wyświetlić ją w przeglądarce. Jest to prosty blog z trzema wpisami. Większość tekstu nie jest interesująca, jednak dodałem kilka akapitów zawierających słowo kluczowe `python`. Parametr `-d 2` symuluje korzystanie z wolnego połączenia, co sprawia, że serwer działa powoli.

Jak to zrobić?

1. Zapisz poniższy skrypt, *speed_up_step1.py*. Kompletny kod znajdziesz w witrynie wydawnictwa Helion i w katalogu *Chapter03* w serwisie GitHub (*https://github.com/PacktPublishing/Python-Automation-Cookbook-Second-Edition/blob/master/Chapter03/speed_up_step1.py*). Tu przedstawiam tylko najważniejsze fragmenty. Ten kod bazuje na skrypcie *crawling_web_step1.py*:

```
...
def process_link(source_link, text):
    ...
    return source_link, get_links(parsed_source, page)
...

def main(base_url, to_search, workers):
    checked_links = set()
    to_check = [base_url]
    max_checks = 10

    with concurrent.futures.ThreadPoolExecutor(max_workers=workers) as executor:
        while to_check:
            futures = [executor.submit(process_link, url, to_search)
                       for url in to_check]
            to_check = []
            for data in concurrent.futures.as_completed(futures):
                link, new_links = data.result()

                checked_links.add(link)
                for link in new_links:
                    if link not in checked_links and link not in to_check:
                        to_check.append(link)

                max_checks -= 1
                if not max_checks:
                    return
```

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    ...
    parser.add_argument('-w', type=int, help='Liczba wątków roboczych,
                        default=4)
    args = parser.parse_args()

    main(args.u, args.p, args.w)
```

2. Zwróć uwagę na inny sposób wywoływania funkcji `main`. Znajduje się w niej dodatkowy parametr (liczba jednocześnie wykonywanych wątków roboczych), a funkcja `process_link` zwraca teraz odsyłacz źródłowy.
3. Uruchom skrypt *crawling_web_step1.py*, aby ustalić bazowy czas wykonywania kodu. Aby przykład był bardziej przejrzysty, dane wyjściowe zostały pominięte:

```
$ time python crawling_web_step1.py http://localhost:8000/
... POMINIĘTE DANE WYJŚCIOWE
real 0m12.221s
user 0m0.160s
sys 0m0.034s
```

4. Uruchom nowy skrypt z jednym wątkiem roboczym. Kod będzie działał wolniej niż pierwotna wersja:

```
$ time python speed_up_step1.py -w 1
... POMINIĘTE DANE WYJŚCIOWE
real 0m16.403s
user 0m0.181s
sys 0m0.068s
```

5. Zwiększ liczbę wątków roboczych:

```
$ time python speed_up_step1.py -w 2
... POMINIĘTE DANE WYJŚCIOWE
real 0m10.353s
user 0m0.199s
sys 0m0.068s
```

6. Dodanie kolejnych wątków roboczych skraca czas pracy:

```
$ time python speed_up_step1.py -w 5
... POMINIĘTE DANE WYJŚCIOWE
real 0m6.234s
user 0m0.171s
sys 0m0.040s
```

Jak to działa?

Głównym miejscem do zgłaszania jednoczesnych żądań jest funkcja `main`. Pozostały kod prawie nie wymagał zmian (oprócz zwracania odsyłacza źródłowego w funkcji `process_link`).

> Pokazana zmiana jest wprowadzana często, gdy kod jest dostosowywany do wykonywania współbieżnego. Współbieżne zadania muszą zwracać wszystkie potrzebne dane, ponieważ nie mogą być zależne od kolejności ich kończenia.

Oto fragment kodu odpowiedzialny za współbieżne wykonywanie zadań:

```
with concurrent.futures.ThreadPoolExecutor(max_workers=workers) as executor:
    while to_check:
        futures = [executor.submit(process_link, url, to_search)
                   for url in to_check]
        to_check = []
        for data in concurrent.futures.as_completed(futures):
            link, new_links = data.result()

            checked_links.add(link)
            for link in new_links:
                if link not in checked_links and link not in to_check:
                    to_check.append(link)

            max_checks -= 1
            if not max_checks:
                return
```

W wyrażeniu `with` tworzona jest pula z podaną liczbą wątków roboczych. W tym wyrażeniu generowana jest lista obiektów `Future` zawierających adresy URL wszystkich pobieranych stron. Funkcja `.as_completed()` zwraca ukończone obiekty `Future`, po czym trzeba pobrać nowe odsyłacze i sprawdzić, czy należy je dodać do listy stron do pobrania. Ten proces jest podobny do operacji wykonywanych w recepturze „Crawling w sieci WWW".

Następnie proces jest uruchamiany ponownie, chyba że pobrana została wystarczająca liczba odsyłaczy lub nie ma już więcej odsyłaczy do wczytania. Zauważ, że odsyłacze są pobierane w porcjach. W pierwszej iteracji przetwarzany jest odsyłacz bazowy i pobierane są wszystkie odsyłacze ze źródłowej strony. W drugiej iteracji wczytywane są wszystkie strony, do których prowadzą te odsyłacze. Po ich pobraniu przetworzona zostanie nowa porcja odsyłaczy.

> Gdy korzystasz z żądań współbieżnych, pamiętaj, że kolejność wykonywania kodu może się zmieniać. Jeśli przetwarzanie określonych danych zajmie trochę więcej lub nieco mniej czasu, kolejność pobieranych informacji może być inna. Ponieważ kod kończy pracę po wczytaniu 10 stron, oznacza to także, że za każdym razem możesz otrzymać 10 innych stron.

Dodatkowe informacje

Kompletna dokumentacja obiektów `Future` z Pythona jest dostępna na stronie *https://docs.python.org/3/library/concurrent.futures.html*.

> W krokach 4. i 5. w punkcie „Jak to zrobić?" ustalenie optymalnej liczby wątków roboczych wymaga wypróbowania różnych ustawień. Niektóre liczby mogą spowolnić pracę kodu z powodu dodatkowego zarządzania wątkami. Nie bój się eksperymentować!

W świecie Pythona istnieją też inne techniki zgłaszania współbieżnych żądań HTTP. Dostępny jest natywny moduł do zgłaszania żądań, `requests-futures`, obsługujący obiekty `Future` (*https://github.com/ross/requests-futures*).

Inna możliwość to użycie programowania asynchronicznego. Ten model ostatnio zyskuje sporo uwagi i bywa bardzo wydajny w sytuacjach wymagających wielu współbieżnych wywołań. Jednak sposób pisania kodu w tym modelu różni się od tradycyjnego podejścia i wymaga przyzwyczajenia. Python udostępnia moduł `asyncio`, który działa w ten sposób. Istnieje też przydatny moduł `aiohttp` do pracy z żądaniami HTTP. Więcej informacji na temat modułu `aiohttp` znajdziesz na stronie *https://aiohttp.readthedocs.io/en/stable/client_quickstart.html*.

Dobre wprowadzenie do programowania asynchronicznego zawiera artykuł *https://djangostars.com/blog/asynchronous-programming-in-python-asyncio/*.

Zobacz także

- Receptura „Crawling w sieci WWW” we wcześniejszej części rozdziału. Znajdziesz tam mniej wydajną wersję rozwiązania z tej receptury.
- Receptura „Pobieranie stron WWW” we wcześniejszej części rozdziału. Poznasz z niej podstawy żądania stron WWW.

Automatyzacja: monotonne zadania zostaw Pythonowi!

Ile czasu marnujesz na wykonywanie monotonnych, długotrwałych zadań? Mowa o przeglądaniu setek stron czy plików, ręcznym przekształcaniu danych, rozsyłaniu e-maili, tworzeniu wykresów i wielu innych nudnych czynnościach. A gdyby tak zlecić tę pracę komputerowi, a samemu zająć się bardziej odpowiedzialnymi i kreatywnymi zadaniami? To jest do zrobienia — trzeba tylko poświęcić nieco czasu i odpowiednio wykorzystać dostępne rozwiązania, takie jak Python i imponująca kolekcja opracowanych dla tego języka narzędzi, bibliotek i rozszerzeń.

Ta książka jest praktycznym zbiorem gotowych receptur, przeznaczonym dla początkujących użytkowników Pythona. Wydanie zostało dostosowane do wersji 3.8 języka, dodano też nowy materiał dotyczący automatyzowania testów, uczenia maszynowego i pracy z nieuporządkowanymi danymi. Dzięki lekturze zaczniesz automatyzować procesy biznesowe — napiszesz aplikację do pobierania informacji ze stron internetowych, tworzenia raportów z wykresami i diagramami na podstawie arkuszy kalkulacyjnych, a także automatycznego generowania e-maili. Będziesz również tworzyć zaawansowane grafiki z potrzebnymi informacjami, automatyzować kampanie marketingowe oraz stosować techniki testowania i debugowania.

W książce znajdziesz receptury, dzięki którym:

- przekształcisz dane na potrzeby data science za pomocą biblioteki pandas
- zautomatyzujesz klasyfikowanie tekstu, filtrowanie e-maili i pobieranie informacji ze stron WWW
- użyjesz biblioteki Matplotlib do generowania wykresów, diagramów i map
- zautomatyzujesz różne zadania związane z generowaniem raportów
- nauczysz się pracy z Beautiful Soup, programem cron, a także z dziennikami i wyrażeniami regularnymi
- napiszesz bot dla komunikatora Telegram, czytnik kanałów RSS i model uczenia maszynowego

Jaime Buelta

od ponad dekady jest programistą Pythona. Tworzył oprogramowanie dla różnych branż, w tym związanych z grami, finansami i edukacją. Stara się automatyzować wszystkie nudne zadania, aby zostawić sobie czas na ważniejsze sprawy, takie jak pisanie książek czy przygotowywanie wystąpień na PyCon Ireland. Mieszka w Dublinie w Irlandii.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-8322-566-1
9 788383 225661
Cena: 99,00 zł
Packt