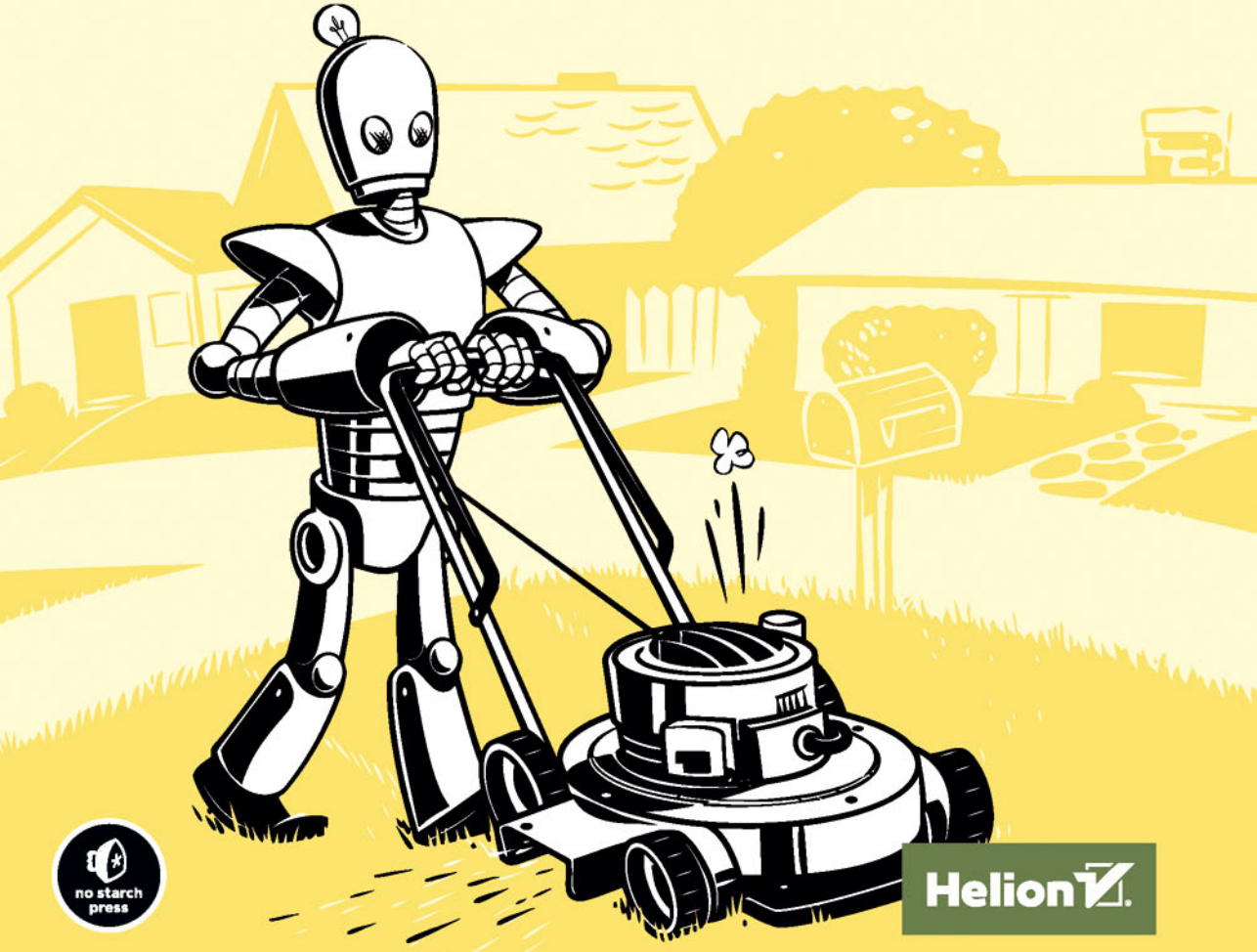
AUTOMATYZACJA NUDNYCH ZADAŃ Z PYTHONEM

NAUKA PROGRAMOWANIA

ALBERT SWEIGART



no starch press

Helion

Tytuł oryginału: Automate the Boring Stuff with Python: Practical Programming
for Total Beginners

Tłumaczenie: Robert Górczyński

ISBN: 978-83-283-3260-7

Title of English-language original: Automate the Boring Stuff with Python,
ISBN: 978-1-59327-599-0, published by No Starch Press.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

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

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

Printed in Poland.

Spis treści

11

Pobieranie danych z internetu

W TYCH RZADKICH, KŁOPOTLIWYCH MOMENTACH, KIEDY JESTEM POZBAWIONY DOSTĘPU DO WI-FI, ZDAJĘ SOBIE SPRAWĘ Z TEGO, JAK WIELE CZYNNOŚCI, KTÓRE WYKONUJĘ W MOIM KOMPUTERZE, robię w internecie. Z czystego przyzwyczajenia sprawdzam pocztę elektroniczną, opublikowane przez moich przyjaciół komunikaty w serwisie Twitter lub po prostu szukam odpowiedzi na pytania typu: „Czy Kurtwood Smith zagrał jakieś główne role, zanim w roku 1987 wcielił się w postać Robocopa?[1]".

Ponieważ duża część pracy wykonywanej w komputerze wymaga dostępu do internetu, więc byłoby dobrze, gdyby tworzone programy miały dostęp do sieci WWW. *Pobieranie danych z internetu* oznacza umożliwienie programowi pobrania i przetwarzania treści pochodzącej z sieci WWW. Przykładowo Google używa wielu tego rodzaju programów w celu indeksowania stron internetowych dla zbudowanej przez tę firmę wyszukiwarki internetowej. W tym rozdziale poznasz kilka modułów ułatwiających pobieranie danych z internetu w programach Pythona.

[1] Odpowiedź na to pytanie brzmi nie.

- Moduł **`webbrowser`** jest dostarczany wraz z Pythonen i pozwala na wyświetlenie wskazanej strony w przeglądarce WWW.
- Moduł **`requests`** pozwala na pobieranie z internetu plików i stron internetowych.
- Moduł **`BeautifulShop`** umożliwia przetwarzanie kodu HTML, za pomocą którego są tworzone strony internetowe.
- Moduł **`selenium`** uruchamia i kontroluje przeglądarkę WWW. Moduł `selenium` ma możliwość wypełniania formularzy oraz symulacji kliknięć myszą w przeglądarce WWW.

Projekt — mapIt.py z użyciem modułu webbrowser

Zdefiniowana w module `webbrowser` funkcja `open()` może uruchomić przeglądarkę WWW i przejść pod wskazany adres URL. W powłoce interaktywnej wprowadź przedstawione poniżej polecenia.

```
>>> import webbrowser
>>> webbrowser.open('http://inventwithpython.com/')
```

W przeglądarce WWW zostanie utworzona nowa karta, w której będzie wyświetlona witryna znajdująca się pod adresem URL *http://inventwithpython.com/*. To jest jedyne zadanie, jakie może wykonać moduł `webbrowser`. Mimo to funkcja `open()` otwiera przed nami interesujące możliwości. Przykładowo do żmudnych zadań należy kopiowanie do schowka adresu zawierającego nazwę ulicy, numer i miejscowość, aby później wyświetlić podane miejsce w aplikacji sieciowej Mapy Google. Całą operację można zredukować o kilka kroków przez utworzenie prostego skryptu, który na podstawie adresu umieszczonego w schowku automatycznie wyświetli mapę w przeglądarce WWW. W ten sposób musisz jedynie skopiować adres do schowka, a następnie uruchomić skrypt. W przeglądarce WWW otrzymasz wyświetloną mapę.

Poniżej wymieniłem w punktach sposób działania programu.

- Pobranie adresu z argumentów wiersza poleceń lub schowka.
- W przeglądarce WWW uruchomienie aplikacji sieciowej Mapy Google i wyświetlenie mapy dla podanego adresu.

Oznacza to, że kod będzie musiał wykonywać przedstawione poniżej zadania.

- Odczyt argumentów wiersza poleceń ze zmiennej `sys.argv`.
- Odczyt zawartości schowka.
- Wywołanie funkcji `webbrowser.open()` w celu otworzenia strony w przeglądarce WWW.

Otwórz nowe okno edytora pliku i zapisz plik pod nazwą *mapIt.py*.

Etap 1. Ustalenie adresu URL

Opierając się na informacjach przedstawionych w dodatku B, skonfiguruj skrypt *mapIt.py* w taki sposób, aby po uruchomieniu go z poziomu wiersza poleceń, na przykład tak:

```
C:\> mapit 870 Valencia St, San Francisco, CA 94110
```

wykorzystał argumenty wiersza poleceń, a nie używał schowka. Jeżeli nie podano żadnych argumentów wiersza poleceń, program powinien automatycznie wykorzystać zawartość schowka.

Przede wszystkim trzeba ustalić, jaki adres URL powinien być użyty dla podanego adresu wskazującego fizyczne miejsce. Po wczytaniu w przeglądarce WWW aplikacji sieciowej Mapy Google (*http://mapy.google.pl/*) i wyszukaniu wymienionego powyżej adresu pasek adresu URL będzie miał postać podobną do *https://www.google.pl/maps/place/870+Valencia+St/@37.7590311,-122.4215096,17z/data=!3m1!4b1!4m2!3m1!1s0x808f7e3dadc07a37:0xc86b0b2bb93b73d8*.

Wprawdzie adres został podany w URL, ale jednocześnie zawiera dużą ilość dodatkowego tekstu. W witrynach internetowych w adresie URL często znajdują się dane dodatkowe, które mają pomagać w śledzeniu odwiedzających lub w dostosowaniu witryny do własnych potrzeb. Jeżeli jednak wpiszesz adres URL w postaci *https://www.google.pl/maps/place/870+Valencia+St+San+Francisco+CA/*, wówczas zostaniesz przekierowany na właściwą stronę. Dlatego też nasz program może po prostu otworzyć nową kartę w przeglądarce WWW i przejść pod adres `'https://www.google.pl/maps/place/ciąg_tekstowy_adresu'` (gdzie *`ciąg_tekstowy_adresu`* oznacza adres, który ma zostać pokazany na mapie).

Etap 2. Obsługa argumentów wiersza poleceń

W pliku programu wprowadź przedstawiony poniżej fragment kodu.

```
#! python3
# mapIt.py — Wyświetla w przeglądarce WWW mapę na podstawie adresu
# podanego w wierszu poleceń lub w schowku.

import webbrowser, sys
if len(sys.argv) > 1:
    # Pobranie adresu z wiersza poleceń.
    address = ' '.join(sys.argv[1:])

# TODO: Pobranie adresu ze schowka.
```

Po wierszu shebang (`#!`) omawianego programu importujemy moduł `webbrow` ➥`ser`, aby mieć możliwość uruchomienia przeglądarki. Ponadto importujemy moduł `sys`, który pozwoli na odczyt potencjalnych argumentów wiersza poleceń.

Zmienna `sys.argv` przechowuje listę składającą się z nazwy pliku programu oraz argumentów podanych w wierszu poleceń. Jeżeli ta lista będzie zawierała coś więcej niż tylko nazwę pliku skryptu, wówczas wartością zwrotną wywołania `len(sys.argv)` będzie liczba całkowita większa niż 1. To oznacza, że użytkownik podał argumenty w wierszu poleceń.

Argumenty w wierszu poleceń są zwykle rozdzielone spacjami, choć w omawianym przypadku chcemy zinterpretować je wszystkie jako pojedynczy ciąg tekstowy. Ponieważ zmienna `sys.argv` to lista ciągów tekstowych, można ją przekazać metodzie `join()`, której wartością zwrotną jest pojedynczy ciąg tekstowy. Nie chcemy nazwy programu w tym ciągu tekstowym, więc zamiast po prostu `sys.argv` przekazujemy `sys.argv[1:]` i tym samym pozbywamy się pierwszego elementu listy. Ostateczny ciąg tekstowy będzie przechowywany w zmiennej o nazwie `address`.

Jeżeli teraz uruchomisz program za pomocą poniższego polecenia:

```
mapit 870 Valencia St, San Francisco, CA 94110
```

wówczas lista przechowywana w zmiennej `sys.argv` będzie miała następującą postać:

```
['mapIt.py', '870', 'Valencia', 'St, ', 'San', 'Francisco, ', 'CA', '94110']
```

Z kolei wartością zmiennej `address` będzie ciąg tekstowy `'870 Valenci St, San Francisco, CA 94110'`.

Etap 3. Obsługa zawartości schowka i uruchomienie przeglądarki WWW

Wprowadź w kodzie programu przedstawione poniżej zmiany.

```
#! python3
# mapIt.py — Wyświetla w przeglądarce WWW mapę na podstawie adresu
# podanego w wierszu poleceń lub w schowku.

import webbrowser, sys, pyperclip
if len(sys.argv) > 1:
    # Pobranie adresu z wiersza poleceń.
    address = ' '.join(sys.argv[1:])
else:
    # Pobranie adresu ze schowka.
    address = pyperclip.paste()

webbrowser.open('https://www.google.pl/maps/place/' + address)
```

Jeżeli w wierszu poleceń nie zostały podane żadne argumenty, program przyjmie założenie, że odpowiedni adres znajduje się w schowku. Zawartość schowka można pobrać za pomocą wywołania `pyperclip.paste()` i przechowywać ją

w zmiennej o nazwie `address`. Na końcu używamy wywołania `webbrowser.open()` do uruchomienia przeglądarki WWW wraz z adresem URL dla aplikacji Mapy Google.

Choć niektóre tworzone programy będą wykonywały ogromne zadania i pozwolą Ci uniknąć wielu godzin pracy, to równie satysfakcjonujące jest korzystanie z programu pozwalającego zaoszczędzić kilka sekund czasu podczas wykonywania często powtarzającego się zadania, takiego jak wyświetlenie danego adresu na mapie. W tabeli 11.1 wymieniłem kroki niezbędne do wykonania, aby wspomniany adres na mapie wyświetlić bez użycia programu *mapIt.py*.

Tabela 11.1. Wyświetlenie adresu na mapie z użyciem maplt.py i bez zastosowania tego programu

Ręczne wyświetlenie adresu na mapie	Użycie maplt.py
Zaznaczenie adresu.	Zaznaczenie adresu.
Skopiowanie adresu.	Skopiowanie adresu.
Otworzenie przeglądarki WWW.	Uruchomienie programu *maplt.py*.
Przejście pod adres URL *http://maps.google.pl/*.	
Kliknięcie pola tekstowego adresu.	
Wklejenie adresu.	
Naciśnięcie klawisza *Enter*.	

Czy dostrzegasz już, jak program *mapIt.py* pomaga podczas wykonywania omawianego zadania?

Pomysły na podobne programy

Kiedy masz adres URL, moduł `webbrowser` pozwala wyeliminować krok, jakim jest otworzenie przeglądarki WWW i przejście pod wskazany adres URL. Poniżej wymieniłem inne programy, które mogą wykorzystać tego rodzaju funkcjonalność.

- Otworzenie w oddzielnych kartach przeglądarki WWW wszystkich łączy znalezionych na stronie.
- Otworzenie przeglądarki WWW i przejście na stronę z prognozą pogody.
- Otworzenie kilku witryn serwisów społecznościowych, z których najczęściej korzystasz.

Pobieranie plików z internetu za pomocą modułu requests

Moduł `requests` pozwala na łatwe pobieranie plików z internetu bez konieczności zajmowania się bardziej skomplikowanymi kwestiami, takimi jak błędy sieciowe, problemy z połączeniem i kompresja danych. Moduł `requests` nie jest dostarczany

wraz z Pythonem, więc najpierw trzeba go zainstalować. Z poziomu wiersza poleceń wydaj polecenie **`pip install requests`**. (W dodatku A znajdziesz więcej informacji na temat instalacji modułów opracowanych przez firmy trzecie).

Moduł `requests` powstał, ponieważ oferowana przez Pythona biblioteka `urllib2` jest zbyt skomplikowana w użyciu. Myślę, że powinieneś wziąć czarny mazak i zamazać ten cały akapit. Zapomnij, że kiedykolwiek wspomniałem o `urllib2`. Jeżeli w programie Pythona trzeba pobrać dane z internetu, po prostu skorzystaj z modułu `requests`.

Teraz przeprowadzimy prosty test w celu sprawdzenia, czy moduł `requests` został prawidłowo zainstalowany. W powłoce interaktywnej wprowadź przedstawione poniżej polecenie.

```
>>> import requests
```

Jeżeli nie zostanie wyświetlony żaden komunikat błędu, to oznacza, że moduł `request` jest zainstalowany poprawnie.

Pobieranie strony internetowej za pomocą funkcji requests.get()

Funkcja `requests.get()` pobiera ciąg tekstowy zawierający adres URL przeznaczony do pobrania. Gdy wywołane zostaje `type()` na wartości zwrotnej funkcji `requests.get()`, możesz sprawdzić, czy otrzymałeś obiekt `Response`. Obiekt zawiera odpowiedź udzieloną przez serwer WWW na wykonane do niego żądanie. Szczegółowe omówienie obiektu `Response` znajdziesz dalej w tym rozdziale. Natomiast teraz, skoro Twój komputer ma połączenie z internetem, w powłoce interaktywnej wprowadź przedstawione poniżej polecenia.

```
>>> import requests
>>> res = requests.get('http://www.gutenberg.org/files/27062/27062-0.txt') ❶
>>> type(res)
<class 'requests.models.Response'>
>>> res.status_code == requests.codes.ok ❷
True
>>> len(res.text)
167343
>>> print(res.text[:250])
The Project Gutenberg EBook of Romeo i Julia, by William Shakespeare

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Project
```

Podany adres URL prowadzi do tekstowej wersji sztuki *Romeo i Julia* dostępnej w witrynie projektu Gutenberg ❶. Jeżeli chcesz dowiedzieć się, czy żądanie internetowe do wskazanej strony zakończyło się powodzeniem, sprawdź atrybut `status_code` obiektu `Response`. Jeżeli wartością będzie `requests.codes.ok`, wówczas masz pewność, że wszystko przebiegło dobrze ❷. (W protokole HTTP kod

oznaczający zakończenie operacji powodzeniem to 200. Prawdopodobnie znasz także kod 404, który oznacza, że wskazany zasób nie został znaleziony).

Jeżeli żądanie zakończy się powodzeniem, pobrana strona będzie przechowywana w postaci ciągu tekstowego w zmiennej `text` obiektu `Response`. Wymieniona zmienna przechowuje ogromny ciąg tekstowy zawierający tekst całej sztuki. Wywołanie funkcji `len(res.text)` wskazuje, że ciąg zawiera prawie 170000 znaków. Na końcu wywołanie `print(Res.text[:250])` wyświetla jedynie pierwsze 250 znaków tekstu.

Sprawdzenie pod kątem błędów

Jak wcześniej widziałeś, obiekt `Response` ma atrybut `status_code`, którego wartość można sprawdzić. Jeżeli jest nią `requests.codes.ok`, oznacza to, że żądanie zakończyło się powodzeniem. Prostszym sposobem sprawdzenia sukcesu jest wywołanie metody `raise_for_status()` obiektu `Response`. Metoda zgłosi wyjątek, jeśli nastąpi błąd podczas pobierania pliku. Gdy pobieranie zakończy się powodzeniem, nic nie zostanie zgłoszone. W powłoce interaktywnej wprowadź przedstawione poniżej polecenia.

```
>>> res = requests.get('http://inventwithpython.com/strona_ktora_nie_istnieje')
>>> res.raise_for_status()
Traceback (most recent call last):
  File "<pyshell#138>", line 1, in <module>
    res.raise_for_status()
  File "C:\Python34\lib\site-packages\requests\models.py", line 773,
  ↪in raise_for_status
    raise HTTPError(http_error_msg, response=self)
requests.exceptions.HTTPError: 404 Client Error: Not Found
```

Metoda `raise_for_status()` to dobry sposób na zagwarantowanie zatrzymania działania programu w przypadku nieprawidłowego pobrania danych. To dobre rozwiązanie, bp program powinien zatrzymać swoje działanie w przypadku wystąpienia pewnego nieoczekiwanego błędu. Jeżeli pobranie nieprawidłowych danych *nie stanowi* problemu dla programu, zawsze możesz opakować wywołanie `raise_for_status()` poleceniami `try` i `except` w celu obsługi tego rodzaju błędu bez powodowania awarii całego programu. Spójrz na poniższy fragment kodu.

```
import requests
res = requests.get('http://inventwithpython.com/strona_ktora_nie_istnieje')
try:
    res.raise_for_status()
except Exception as exc:
    print('Wystąpił następujący problem: %s' % (exc))
```

Powyższe wywołanie metody `raise_for_status()` spowoduje wygenerowanie przez program przedstawionych poniżej danych wyjściowych.

```
Wystąpił następujący problem: błąd po stronie klienta 404: nie znaleziono zasobu.
```

Zawsze stosuj wywołania `raise_for_status()` po wywołaniu funkcji `requests.get()`. Powinieneś mieć pewność o prawidłowym pobraniu danych, zanim program będzie kontynuował pracę.

Zapis pobranych plików na dysku twardym

Od tego miejsca, za pomocą metod `open()` i `write()` możesz zapisywać strony internetowe do plików umieszczonych na dysku twardym komputera. Jednak istnieją pewne drobne różnice. Przede wszystkim konieczne jest otworzenie pliku w *trybie binarnym*, co wymaga przekazania ciągu tekstowego `'wb'` jako drugiego argumentu metody `open()`. Jeśli nawet strona jest w postaci zwykłego tekstu (na przykład pobranego wcześniej pliku ze sztuką *Romeo i Julia*), to i tak konieczne jest zapisanie danych binarnych zamiast tekstowych, aby zachować *kodowanie Unicode* tego tekstu.

KODOWANIE UNICODE

Omówienie kodowania Unicode wykracza poza zakres tematyczny tej książki. Więcej informacji na ten temat znajdziesz w wymienionych poniżej artykułach.

- *Joel on Software: The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)* na stronie *http://www.joelonsoftware.com/articles/Unicode.html*.
- *Pragmatic Unicode* na stronie *http://nedbatchelder.com/text/unipain.html*.

W celu zapisania strony internetowej do pliku można użyć pętli `for` wraz z metodą `iter_content()` obiektu `Response`, tak jak pokazałem poniżej.

```
>>> import requests
>>> res = requests.get('http://www.gutenberg.org/files/27062/27062-0.txt')
>>> res.raise_for_status()
>>> playFile = open('Romeo-i-Julia.txt', 'wb')
>>> for chunk in res.iter_content(100000):
        playFile.write(chunk)

100000
67343
>>> playFile.close()
```

Wartością zwrotną metody `iter_content()` są „fragmenty” treści przetwarzane podczas każdej iteracji pętli. Każdy fragment to po prostu dane typu *bajty*, metoda podaje ilość bajtów znajdujących się w poszczególnych fragmentach.

Sto tysięcy bajtów to rozsądna wielkość, więc przekazujemy wartość `100000` jako argument funkcji `iter_content()`.

Plik *Romeo-i-Julia.txt* teraz już istnieje w bieżącym katalogu roboczym. Zwróć uwagę, że nazwa pliku na stronie internetowej to *27062-0.txt*, natomiast zapisany na dysku twardym ma zupełnie inną nazwę. Obsługą pobierania zawartości stron internetowych zajmuje się moduł `requests`. Strona po pobraniu stanowi po prostu dane w programie. Jeśli nawet utracisz połączenie z internetem po pobraniu strony internetowej, wszystkie dane strony nadal pozostaną w komputerze.

Wartością zwrotną metody `write()` jest liczba bajtów zapisanych w pliku. W przedstawionym powyżej przykładzie pierwszy fragment składał się ze 100000 bajtów, natomiast pozostała część wymagała jedynie 67343 bajtów.

Poniżej przedstawiam pełny proces pobierania i zapisu pliku.

1. Wywołanie funkcji `requests.get()` w celu pobrania pliku.
2. Wywołanie funkcji `open()` wraz z argumentem `'wb'` w celu utworzenia nowego pliku w trybie binarnym.
3. Iteracja przez obiekt `Response` za pomocą metody `iter_content()`.
4. Wywołanie `write()` w trakcie każdej iteracji, aby umieścić treść w pliku.
5. Wywołanie funkcji `close()` w celu zamknięcia pliku.

To już wszystko, co dotyczy modułu `requests`! Pętla `for` i metoda `iter_content()` mogą wydawać się skomplikowane w porównaniu z opartym na funkcjach `open()`, `write()` i `close()` rozwiązaniem, którego używamy podczas pracy z plikami tekstowymi. Jednak przedstawione podejście gwarantuje, że moduł `requests` nie będzie zużywał zbyt dużej ilości pamięci nawet podczas pobierania ogromnych plików. Więcej informacji na temat innych funkcji modułu `requests` znajdziesz na stronie *http://requests.readthedocs.io/en/master/*.

HTML

Zanim będziesz mógł zająć się poważniej stronami internetowymi, najpierw powinieneś poznać podstawy języka HTML. Zobaczysz również, jak uzyskać dostęp do oferowanych przez przeglądarkę WWW użytecznych narzędzi programistycznych, z pomocą których pobieranie danych z internetu stanie się jeszcze łatwiejsze.

Zasoby pomagające w poznawaniu języka HTML

Hipertekstowy język znaczników (ang. *hypertext markup language*, czyli **HTML**) to format, w którym są zapisywane strony internetowe. W tym rozdziale przyjąłem założenie, że znasz HTML przynajmniej w zakresie podstawowym. Jeżeli mimo wszystko potrzebujesz pewnego wprowadzenia, sugeruję zapoznanie się z materiałem przedstawionym na jednej z wymienionych poniżej stron internetowych.

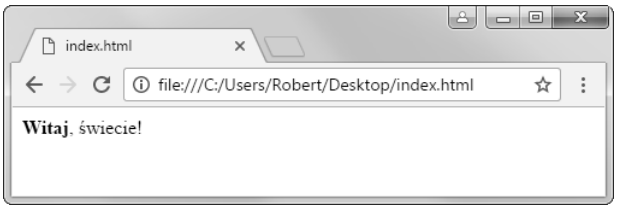
- *http://htmldog.com/guides/html/beginner/*
- *https://www.codecademy.com/learn/web*
- *https://developer.mozilla.org/en-US/docs/Learn/HTML*

Krótkie wprowadzenie

Kiedy minęło już sporo czasu od chwili, gdy miałeś okazję analizować jakikolwiek kod HTML, tutaj znajdziesz krótkie omówienie podstaw. Plik w formacie HTML jest plikiem zwykłego tekstu wraz z rozszerzeniem *.html*. Tekst w tego rodzaju pliku jest ujęty w tak zwane *znaczniki*, którymi są po prostu słowa umieszczone w nawiasach ostrych. Znaczniki wskazują przeglądarce WWW sposób formatowania strony internetowej. Znaczniki otwierający i zamykający mogą oblewać pewien tekst i tym samym tworzą tak zwany *element*. Z kolei *tekst* (lub *wewnętrzny HTML*) to zawartość umieszczona między znacznikami otwierającym i zamykającym. Przykładowo poniższy fragment kodu HTML powoduje wyświetlenie w przeglądarce WWW komunikatu *Witaj, świecie!*, przy czym słowo *Witaj* będzie pogrubione.

```
<strong>Witaj</strong>, świecie!
```

Po wygenerowaniu przez przeglądarkę WWW tekst ten będzie wyglądał tak, jak pokazałem na rysunku 11.1.



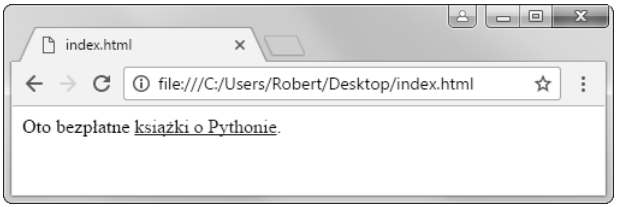
Rysunek 11.1. Komunikat „Witaj, świecie!" wyświetlony przez przeglądarkę WWW

Znacznik otwierający `<strong>` informuje, że obejmowany przez niego tekst ma zostać pogrubiony. Z kolei znacznik zamykający `</strong>` wskazuje przeglądarce WWW, gdzie kończy się pogrubiony tekst.

W języku HTML mamy wiele różnych znaczników. Część z nich obsługuje dodatkowe właściwości w postaci *atrybutów* definiowanych wewnątrz nawiasu ostrego. Przykładowo znacznik `<a>` zawiera tekst, który powinien być wyświetlony w postaci łącza. Adres URL dla tego łącza jest określany przez atrybut `href`. Poniżej przedstawiłem przykład.

```
Oto bezpłatne <a href="http://inventwithpython.com">książki o Pythonie</a>.
```

Po wygenerowaniu przez przeglądarkę WWW tekst ten będzie wyglądał tak, jak pokazałem na rysunku 11.2.



Rysunek 11.2. Łącze wygenerowane przez przeglądarkę WWW

Niektóre elementy mają atrybut `id` używany do unikatowej identyfikacji elementu na stronie. Bardzo często będziesz nakazywać programowi wyszukiwanie elementów na podstawie ich atrybutów `id`. Dlatego też ustalenie tego atrybutu za pomocą wbudowanych w przeglądarkę WWW narzędzi programistycznych jest dość często spotykanym zadaniem podczas tworzenia programów pobierających dane z internetu.

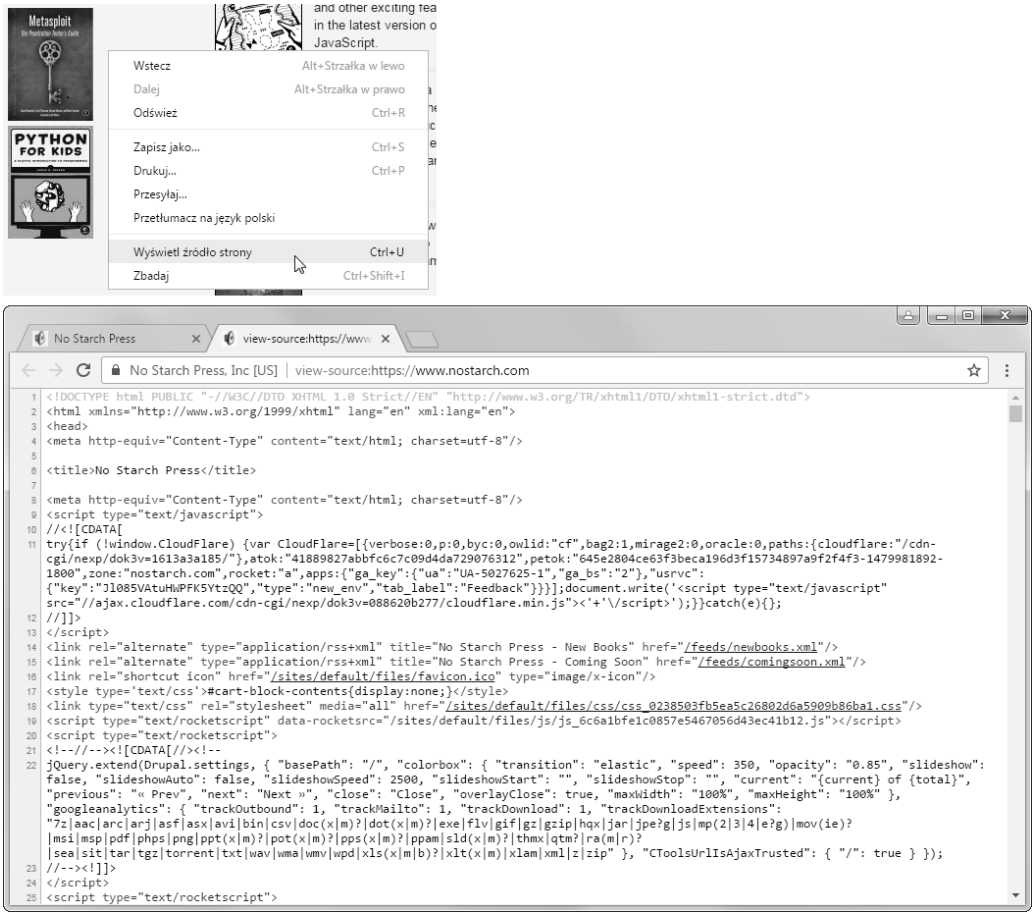
Wyświetlenie kodu źródłowego HTML strony internetowej

Nierzadko będzie występowała konieczność przeanalizowania kodu źródłowego HTML stron internetowych, z którymi współpracują tworzone przez Ciebie programy. W tym celu kliknij prawym przyciskiem myszy dowolną stronę internetową w przeglądarce WWW, a następnie wybierz opcję *Wyświetl źródło* lub *Wyświetl źródło strony*, aby faktycznie zobaczyć kod HTML tej strony (patrz rysunek 11.3). To jest tekst w rzeczywistości otrzymywany przez przeglądarkę WWW. Trzeba w tym miejscu dodać, że przeglądarka WWW „wie", w jaki sposób wyświetlić, inaczej *wygenerować*, stronę internetową na podstawie kodu HTML.

Gorąco zachęcam do wyświetlenia kodu źródłowego HTML jednej z Twoich ulubionych stron internetowych. Nie przejmuj się, jeśli nie zrozumiesz w pełni tego, co zobaczysz, patrząc na kod źródłowy. Nie musisz być mistrzem w tworzeniu kodu HTML, aby pisać proste programy pobierające dane z internetu — przecież nie zamierzasz samodzielnie budować witryn internetowych. Musisz jedynie wiedzieć, jak pobrać dane z istniejącej witryny.

Wyświetlenie oferowanych przez przeglądarkę WWW narzędzi programistycznych

Kod źródłowy strony możesz wyświetlić również za pomocą narzędzi programistycznych wbudowanych w przeglądarkę WWW. Przykładowo w przeglądarkach Chrome i Internet Explorer dla Windows narzędzia programistyczne są zainstalowane. Wystarczy nacisnąć klawisz *F12*, aby zostały wyświetlone na ekranie (patrz rysunek 11.4). Ponowne naciśnięcie klawisza *F12* zamyka te narzędzia. W przeglądarce Chrome narzędzia programistyczne można wyświetlić za pomocą opcji menu *Więcej narzędzi/Narzędzia dla programistów*. Z kolei w systemie macOS trzeba nacisnąć klawisze *Command+Option+I*, aby wyświetlić narzędzia programistyczne wbudowane w przeglądarkę Chrome.



Rysunek 11.3. Wyświetlenie kodu źródłowego strony internetowej

Jeżeli używasz przeglądarki Firefox, narzędzia programistyczne zostaną wyświetlone po naciśnięciu klawiszy *Ctrl+Shift+C* (platformy Windows i Linux) lub *Command+Option+C* (platforma macOS). Układ tych narzędzi jest niemal identyczny z układem narzędzi wbudowanych w przeglądarkę Chrome.

W przeglądarce Safari przejdź do karty *Zaawansowane* okna preferencji, a następnie zaznacz pole wyboru *Pokazuj menu Programowanie na pasku menu*. Po włączeniu wymienionego menu narzędzia programistyczne będziesz mógł wyświetlić po naciśnięciu klawiszy *Command+Option+I*.

Po włączeniu lub zainstalowaniu narzędzi programistycznych w przeglądarce WWW możesz kliknąć dowolny element strony internetowej, a następnie z menu kontekstowego wybrać opcję *Skontroluj element*. W ten sposób przejdziesz do fragmentu kodu HTML odpowiedzialnego za wygenerowanie klikniętego elementu strony. Ta możliwość okaże się użyteczna, gdy rozpoczniemy przetwarzanie kodu HTML w programach pobierających dane z internetu.

Rysunek 11.4. Okno narzędzi programistycznych w przeglądarce Chrome

NIE UŻYWAJ WYRAŻEŃ REGULARNYCH DO PRZETWARZANIA KODU HTML

Odszukanie określonego fragmentu kodu HTML w ciągu tekstowym wydaje się idealnym zadaniem dla wyrażeń regularnych. Jednak odradzam takie podejście. Istnieje znacznie więcej różnych sposobów, na jakie może być formatowany kod HTML, który nadal będzie uznany za prawidłowy. Próba dopasowania tych wszystkich możliwych odmian za pomocą wyrażenia regularnego może być żmudna i podatna na błędy. Moduł opracowany specjalnie do przetwarzania kodu HTML nosi nazwę `BeautifulSoup` i otrzymane za jego pomocą wyniki będą prawdopodobnie miały znacznie mniej błędów.

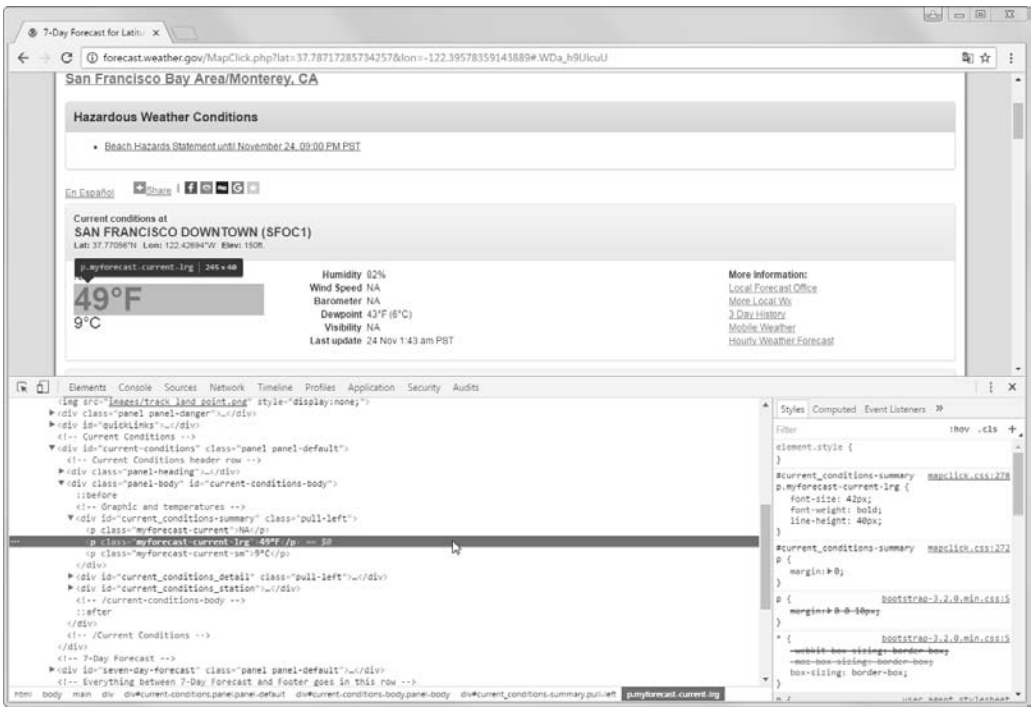
Więcej dokładniejszych informacji na temat tego, dlaczego nie powinieneś przetwarzać kodu HTML za pomocą wyrażeń regularnych znajdziesz na stronie *http://stackoverflow.com/questions/1732348/regex-match-open-tags-except-xhtml-self-contained-tags/1732454#1732454*.

Użycie narzędzi programistycznych do wyszukiwania elementów HTML

Kiedy Twój program pobierze stronę internetową za pomocą modułu `requests`, treść tej strony otrzymasz w postaci jednego dużego ciągu tekstowego. Teraz musisz ustalić, które fragmenty kodu HTML odpowiadają interesującym Cię informacjom na pobranej stronie internetowej.

Tutaj z pomocą mogą przyjść wbudowane w przeglądarkę WWW narzędzia programistyczne. Przyjmujemy założenie, że chcesz utworzyć program pobierający z witryny *http://www.weather.gov/* dane dotyczące prognozy pogody. Przed napisaniem jakiegokolwiek kodu musisz przeprowadzić małe badania. Jeżeli odwiedziesz wymienioną witrynę i podasz kod pocztowy 94105, przejdziesz na stronę zawierającą prognozę pogody dla wymienionego regionu.

Co zrobić w sytuacji, gdy jesteś zainteresowany pobraniem informacji o temperaturze dla regionu o podanym kodzie pocztowym? Kliknij prawym przyciskiem myszy tę informację na stronie, a następnie z wyświetlonego menu kontekstowego wybierz opcję *Zbadaj*. Na ekranie pojawi się okno narzędzi programistycznych, które będzie wyświetlało kod HTML odpowiedzialny za wygenerowanie kliknięte-go fragmentu strony. Na rysunku 11.5 pokazałem narzędzia programistyczne wraz z kodem HTML generującym fragment strony wyświetlający temperaturę.



Rysunek 11.5. Użycie narzędzi programistycznych do analizy elementu wyświetlającego temperaturę

W oknie narzędzi programistycznych możesz zobaczyć, że kod HTML odpowiedzialny za wyświetlenie temperatury na tej stronie internetowej to `<p class="myforecast-current -lrg">49°F</p>`. To jest dokładnie to, czego szukamy! Wydaje się, że informacje o temperaturze są umieszczone w elemencie `<p>` o klasie `myforecast-current-lrg`. Skoro już wiesz, czego szukasz, moduł Beautiful Soup pomoże Ci w odszukaniu właściwego ciągu tekstowego.

Przetwarzanie kodu HTML za pomocą modułu Beautiful Soup

`BeautifulSoup` to moduł przeznaczony do wyodrębniania informacji ze strony HTML (sprawdza się w tym znacznie lepiej niż wyrażenia regularne). W kodzie Pythona nazwą tego modułu jest `bs4` (skrót od Beautiful Soup, version 4). W celu zainstalowania modułu należy z poziomu wiersza poleceń wydać polecenie `pip install beautifulsoup4`. (W dodatku A znajdziesz więcej informacji na temat instalacji modułów opracowanych przez firmy trzecie). Wprawdzie `beautifousoup4` to nazwa używana podczas instalacji, ale w kodzie Pythona stosowana jest nazwa `bs4`, dlatego też, aby zaimportować moduł, w kodzie należy wydać polecenie `import bs4`.

W przykładach omawianych w tym rozdziale za pomocą modułu `BeautifulSoup` będziemy *przetwarzać* (to znaczy analizować i identyfikować) fragmenty pliku HTML znajdującego się na dysku twardym. Otwórz nowe okno edytora pliku i wprowadź poniższy fragment kodu. Następnie zapisz plik pod nazwą *example.html*. Ewentualnie możesz pobrać ten plik ze strony *ftp://ftp.helion.pl/przyklady/autopy.zip*.

```
<!-- To jest przykładowy plik example.html. -->

<html><head><title>Tytuł mojej witryny internetowej</title></head>
<body>
<p>Pobierz książkę o języku <strong>Python</strong> z <a href="http://
inventwithpython.com">mojej witryny internetowej</a>.</p>
<p class="slogan">Poznaj Pythona w łatwy sposób!</p>
<p>Autor <span id="author">Al Sweigart</span></p>
</body></html>
```

Jak możesz zobaczyć, nawet prosty plik HTML zawiera wiele różnych znaczników i atrybutów. W skomplikowanych witrynach internetowych kod bardzo szybko może stać się zagmatwany. Na szczęście moduł `BeautifulSoup` niezwykle ułatwia pracę z kodem HTML.

Utworzenie obiektu BeautifulSoup na podstawie kodu HTML

Funkcja `bs4.BeautifouSoup()` musi być wywołana wraz z ciągiem tekstowym zawierającym kod HTML przeznaczony do przetworzenia. Wartością zwrotną tej funkcji jest obiekt `BeautifulSoup`. W powłoce interaktywnej komputera, który ma połączenie z internetem, wprowadź przedstawione poniżej polecenia.

```
>>> import requests, bs4
>>> res = requests.get('http://nostarch.com')
>>> res.raise_for_status()
>>> noStarchSoup = bs4.BeautifulSoup(res.text)
>>> type(noStarchSoup)
<class 'bs4.BeautifulSoup'>
```

W powyższym fragmencie kodu użyliśmy wywołania `requests.get()` w celu pobrania strony głównej z witryny No Starch Press, a następnie przekazaliśmy funkcji `bs4.BeautifulSoup()` atrybut `text` otrzymanej odpowiedzi. Zwrócony obiekt `BeautifulSoup` jest przechowywany w zmiennej o nazwie `noStarchSoup`.

Istnieje również możliwość wczytania kodu HTML z pliku znajdującego się na dysku twardym komputera. To wymaga przekazania obiektu `File` do wywołania `bs4.BeautifulSoup()`.W powłoce interaktywnej wprowadź przedstawione poniżej polecenia (upewnij się, że bieżący katalog roboczy zawiera plik o nazwie *example.html*).

```
>>> exampleFile = open('example.html')
>>> exampleSoup = bs4.BeautifulSoup(exampleFile)
>>> type(exampleSoup)
<class 'bs4.BeautifulSoup'>
```

Po otrzymaniu obiektu `BeautifulSoup` można wykorzystać jego metody w celu wyszukiwania konkretnych fragmentów dokumentu HTML.

Wyszukiwanie elementu za pomocą metody select()

Element strony internetowej można pobrać z obiektu `BeautifulSoup` za pomocą wywołania metody `select()` i przekazania jej ciągu tekstowego w postaci *selektora* CSS szukanego elementu. Pod pewnymi względami selektory przypominają wyrażenia regularne — określają szukany wzorzec, w tym przypadku na stronach HTML, a nie w ogólnych ciągach tekstowych.

Pełne omówienie składni selektorów CSS wykracza poza zakres tematyczny tej książki[2]. Poniżej przedstawiłem tylko kilka krótkich informacji o selektorach. W tabeli 11.2 wymieniłem najczęściej stosowane wzorce selektorów CSS.

Różne wzorce selektorów można ze sobą łączyć i tym samym zdefiniować znacznie bardziej skomplikowane dopasowania. Przykładowo `soup.select('p #author')` spowoduje dopasowanie dowolnego elementu, który ma atrybut `id` o wartości `author`, o ile znajduje się wewnątrz elementu `<p>`.

Metoda `select()` zwróci listę obiektów `Tag`, za pomocą których moduł `Beauti`
`↪fulSoup` przedstawia element HTML. Lista będzie zawierała po jednym obiekcie

[2] Dobre wprowadzenie do tematu selektorów znajdziesz na stronach *https://www.w3.org/TR/CSS2/selector.html*, *http://ryanstutorials.net/css-tutorial/css-selectors.php* i *http://htmldog.com/guides/css/beginner/selectors/*.

Tabela 11.2. Przykłady selektorów CSS

Selektor przekazany metodzie select()	Dopasowuje…
`soup.select('div')`	Wszystkie elementy o nazwie `<div>`.
`soup.select('#author')`	Element, którego atrybut `id` ma wartość `author`.
`soup.select('.notice')`	Wszystkie elementy używające atrybutu CSS `class` o wartości `notice`.
`soup.select('div span')`	Wszystkie elementy o nazwie `<span>`, które zostały umieszczone w elemencie o nazwie `<div>`.
`soup.select('div > span')`	Wszystkie elementy o nazwie `<span>`, które są *bezpośrednio* w elemencie o nazwie `<div>` i między nimi nie istnieje żaden inny element.
`soup.select('input[name]')`	Wszystkie elementy o nazwie `<input>`, które mają atrybut `name` o dowolnej wartości.
`soup.select('input[type="button"]')`	Wszystkie elementy o nazwie `<input>`, które mają atrybut `type` o wartości `button`.

`Tag` dla każdego dopasowania w obiekcie `BeautifulSoup`. Wartości obiektów `Tag` mogą być przekazywane funkcji `str()` w celu wyświetlenia przedstawianych przez nie znaczników HTML. Wartości obiektów `Tag` mogą mieć również atrybut `attrs` pokazujący podane w postaci słownika wszystkie atrybuty HTML danego znacznika. Używając przygotowanego wcześniej pliku o nazwie *example.html*, w powłoce interaktywnej wprowadź przedstawione poniżej polecenia.

```
>>> import bs4
>>> exampleFile = open('example.html')
>>> exampleSoup = bs4.BeautifulSoup(exampleFile.read())
>>> elems = exampleSoup.select('#author')
>>> type(elems)
<class 'list'>
>>> len(elems)
1
>>> type(elems[0])
<class 'bs4.element.Tag'>
>>> elems[0].getText()
'Al Sweigart'
>>> str(elems[0])
'<span id="author">Al Sweigart</span>'
>>> elems[0].attrs
{'id': 'author'}
```

Powyższy fragment kodu wyciągnie z naszego przykładowego pliku HTML element o atrybucie `id="author"`. Wywołania `select('#author')` używamy do zwrócenia listy wszystkich elementów wraz z atrybutem `id="author"`. Listę obiektów `Tag` przechowujemy w zmiennej o nazwie `elems`. Wywołanie `len(elems)` pokazuje, że mamy tylko jeden obiekt `Tag` na liście, czyli było tylko jedno dopasowanie. Wywołanie `getText()` w elemencie zwraca jego tekst lub wewnętrzny kod HTML tego elementu. Tekstem elementu jest treść znajdująca się między znacznikami otwierającym i zamykającym. W omawianym przykładzie to będzie `'Al Sweigart'`.

W wyniku przekazania elementu do funkcji `str()` zwracany jest ciąg tekstowy zawierający znaczniki otwierający i zamykający oraz tekst elementu. Na końcu `attrs` daje słownik wraz z atrybutem elementu, tutaj `'id'`, oraz jego wartości, czyli `'author'`.

Z obiektu `BeautifulSoup` można wyodrębnić również elementy `<p>`. W powłoce interaktywnej wprowadź przedstawione poniżej polecenia.

```
>>> pElems = exampleSoup.select('p')
>>> str(pElems[0])
'<p>Pobierz książkę o języku <strong>Python</strong> z <a href="http://
inventwithpython.com">mojej witryny internetowej</a>.</p>'
>>> pElems[0].getText()
'Pobierz książkę o języku Python z mojej witryny internetowej.'
>>> str(pElems[1])
'<p class="slogan">Poznaj Pythona w łatwy sposób!</p>'
>>> pElems[1].getText()
'Poznaj Pythona w łatwy sposób!'
>>> str(pElems[2])
'<p>Autor <span id="author">Al Sweigart</span></p>'
>>> pElems[2].getText()
'Autor Al Sweigart'
```

Tym razem wywołanie `select()` daje listę trzech dopasowań, które są przechowywane w zmiennej `pElems`. Za pomocą funkcji `str()` użytej wraz z `pElems[0]`, `pElems[1]` i `pElems[2]` możemy wyświetlić każdy element jako ciąg tekstowy. Natomiast wywołanie `getText()` w poszczególnych elementach wyświetla znajdujący się w nich tekst.

Pobieranie danych z atrybutów elementu

Metoda `get()` w obiektach `Tag` znacznie ułatwia uzyskanie dostępu do wartości atrybutów w danym elemencie. Wymienionej metodzie przekazujemy ciąg tekstowy nazwy atrybutu, która z kolei zwraca wartość tego atrybutu. Używając przygotowanego wcześniej pliku o nazwie *example.html*, w powłoce interaktywnej wprowadź przedstawione poniżej polecenia.

```
>>> import bs4
>>> soup = bs4.BeautifulSoup(open('example.html'))
>>> spanElem = soup.select('span')[0]
>>> str(spanElem)
'<span id="author">Al Sweigart</span>'
>>> spanElem.get('id')
'author'
>>> spanElem.get('some_nonexistent_addr') == None
True
>>> spanElem.attrs
{'id': 'author'}
```

W powyższym fragmencie kodu metoda `select()` wyszukała wszystkie elementy `<span>`, a następnie pierwszy dopasowany umieściła w zmiennej `spanElem`. Przekazanie funkcji `get()` nazwy atrybutu `'id'` powoduje zwrot jego wartości, czyli `'author'`.

Projekt — wyszukiwanie typu „szczęśliwy traf” w Google

Kiedy szukam informacji na dany temat za pomocą wyszukiwarki internetowej Google, nie przeglądam po kolei otrzymanych wyników wyszukiwania. Klikając łącza prawym przyciskiem myszy (lub z naciśniętym klawiszem *Ctrl*), otwieram kilka pierwszych wyników w nowych kartach, aby później się z nimi zapoznać. Z wyszukiwarki internetowej Google korzystam na tyle często, że podejście typu: otworzenie przeglądarki WWW, wpisanie szukanego tematu i kolejne otwieranie wyników w nowych kartach — stało się żmudne. Byłoby znacznie lepiej, gdybym mógł wpisać w wierszu poleceń szukany temat, a komputer automatycznie uruchomiłby przeglądarkę i kilka pierwszych wyników otworzył w nowych kartach. Dlatego też teraz zajmiemy się przygotowaniem skryptu, który wykona tego rodzaju zadanie.

Poniżej wymieniłem w punktach sposób działania programu.

- Pobranie słów kluczowych wyszukiwania z argumentów wiersza poleceń.
- Pobranie strony wyników wyszukiwania.
- Otworzenie nowej karty dla każdego znalezionego wyniku.

Oznacza to, że kod będzie musiał wykonywać przedstawione poniżej zadania.

- Odczyt argumentów wiersza poleceń z listy `sys.argv`.
- Pobranie za pomocą modułu `requests` strony wyników wyszukiwania.
- Wyszukanie łączy dla każdego wyniku wyszukiwania.
- Wywołanie funkcji `webbrowser.open()` w celu przejścia do przeglądarki WWW.

Otwórz nowe okno edytora pliku i zapisz plik pod nazwą *lucky.py*.

Etap 1. Pobranie argumentów wiersza poleceń i żądanie strony wyszukiwarki

Zanim w ogóle przejdziemy dalej, najpierw musisz ustalić adres URL strony wyświetlającej wyniki wyszukiwania. Patrząc na pasek adresu przeglądarki WWW po przeprowadzeniu jakiegokolwiek wyszukiwania w Google, możesz ustalić, że interesujący nas adres URL ma postać *http://www.google.pl/search?q=SZUKANE_WYRAŻENIE*. Moduł `requests` może pobrać tę stronę, a następnie możemy wykorzystać moduł `BeautifulSoup` do wyodrębnienia z otrzymanego dokumentu HTML

łączy wyników wyszukiwania. Na koniec użyjemy modułu `webbrowser` do otworzenia wspomnianych łączy na kartach przeglądarki WWW.

W pliku wprowadź przedstawiony poniżej kod.

```
#! python3
# lucky.py — Otwiera kilka wyników wyszukiwania Google.

import requests, sys, webbrowser, bs4

print('Wyszukiwanie...')      # Komunikat wyświetlany podczas pobierania strony Google.
res = requests.get('http://google.pl/search?q=' + ' '.join(sys.argv[1:]))
res.raise_for_status()

# TODO: Pobranie łączy z kilkoma pierwszymi wynikami wyszukiwania.

# TODO: Otworzenie karty przeglądarki WWW dla każdego wyniku wyszukiwania.
```

Użytkownik dostarczy szukane wyrażenia za pomocą argumentów wiersza poleceń podczas uruchamiania programu. Argumenty będą przechowywane na liście `sys.argv` w postaci ciągów tekstowych.

Etap 2. Wyszukanie wszystkich wyników

Teraz będziemy musieli użyć modułu `BeautifulSoup` do wyodrębnienia pierwszych kilku wyników wyszukiwania z pobranego dokumentu HTML. W tym miejscu rodzi się pytanie, jak wybrać odpowiedni selektor do tego zadania? Nie można po prostu wyszukać wszystkich znaczników `<a>`, ponieważ na pobranej stronie znajduje się wiele łączy, które nas nie interesują. Dlatego też konieczne jest przeanalizowanie strony wyników wyszukiwania za pomocą wbudowanych w przeglądarkę WWW narzędzi programistycznych i w ten sposób podjęcie próby ustalenia selektora, który pozwoli na wyodrębnienie jedynie interesujących nas łączy.

Po przeprowadzeniu wyszukiwania w Google wyrażenia *Beautiful Soup* przejdź do narzędzi programistycznych i zajrzyj do dowolnie wybranego elementu łącza na stronie. Tego rodzaju elementy wyglądają na niezwykle skomplikowane i mają postać podobną do przedstawionej `Beautiful Soup: We called him Tortoise because he taught us.`.

Nie ma absolutnie żadnego znaczenia, że element wygląda na bardzo skomplikowany. Musimy po prostu odnaleźć wzorzec, który mają wszystkie łącza przedstawiające wyniki wyszukiwania. Jednak ten element `<a>` nie ma niczego, co łatwo odróżniałoby go od nieinteresujących nas elementów `<a>` na stronie.

Wprowadź w kodzie przedstawione poniżej zmiany.

```
#! python3
# lucky.py — Otwiera kilka wyników wyszukiwania Google.

import requests, sys, webbrowser, bs4

--cięcie--

# Pobranie łączy z kilkoma pierwszymi wynikami wyszukiwania.
soup = bs4.BeautifulSoup(res.text)

# Otworzenie karty przeglądarki WWW dla każdego wyniku wyszukiwania.
linkElems = soup.select('.r a')
```

Gdy rozejrzysz się nieco w okolicach analizowanego wcześniej elementu `<a>`, zauważysz element, taki jak `<h3 class="r">`. Po przejrzeniu pozostałej części kodu źródłowego HTML można dojść do wniosku, że klasa `r` jest używana jedynie dla łączy zawierających wyniki wyszukiwania. Nie musisz wiedzieć, czym jest klasa CSS o nazwie `r` lub na czym polega jej działanie. Po prostu użyjemy jej do odnalezienia interesujących nas elementów `<a>`. Możemy utworzyć obiekt `Beautiful` ↪`Soup` na podstawie kodu HTML pobranej strony, a następnie wykorzystać selektor `'.r a'` do odszukania wszystkich elementów `<a>` znajdujących się wewnątrz elementów o przypisanej klasie CSS `r`.

Etap 3. Otworzenie kart przeglądarki WWW dla poszczególnych wyników

Na koniec nakazujemy programowi otworzenie kart przeglądarki WWW dla poszczególnych wyników. Na końcu programu wprowadź poniższy fragment kodu.

```
#! python3
# lucky.py — Otwiera kilka wyników wyszukiwania Google.

import requests, sys, webbrowser, bs4

--cięcie--

# Otworzenie karty przeglądarki WWW dla każdego wyniku wyszukiwania.
linkElems = soup.select('.r a')
numOpen = min(5, len(linkElems))
for i in range(numOpen):
    webbrowser.open('http://google.pl' + linkElems[i].get('href'))
```

Domyślnie w nowych kartach otwieramy jedynie pięć pierwszych wyników wyszukiwania, używając modułu `webbrowser`. Jednak użytkownik mógł szukać czegoś, co pojawiło się w mniejszej liczbie wyników wyszukiwania. Wywołanie `soup.select()` zwraca listę wszystkich elementów dopasowanych za pomocą

selektora `'.r a'`. Dlatego też liczba kart przeznaczonych do otworzenia wynosi 5 lub odpowiada wielkości liczby (pod uwagę brana jest mniejsza wartość).

Wbudowana w Pythonie funkcja `min()` zwraca argument o mniejszej wartości z przekazanych liczb całkowitych lub zmiennoprzecinkowych. (Istnieje również wbudowana funkcja `max()`, która zwraca większy z przekazanych argumentów). Funkcję `min()` można wykorzystać do ustalenia, czy na liście znajduje się mniej niż pięć łączy. Liczbę łączy przeznaczonych do otworzenia w nowych kartach przechowujemy w zmiennej `numOpen`. Następnie za pomocą pętli `for` przeprowadzamy iterację przez te łącza, wywołując `range(numOpen)`.

W trakcie każdej iteracji pętli wywołanie funkcji `webbrowser.open()` otwiera nową kartę w przeglądarce WWW. Zwróć uwagę, że wartość atrybutu `href` w zwracanych elementach `<a>` nie zawiera początkowego fragmentu `http://google.pl`, więc trzeba przeprowadzić konkatenację podanego fragmentu i ciągu tekstowego będącego wartością atrybutu `href`.

Teraz możemy natychmiast otworzyć pięć pierwszych wyników wyszukiwania w Google, na przykład wyrażenia *Python programming tutorials*, przez wywołanie w wierszu poleceń `lucky python programming tutorials`! (W dodatku A znajdziesz informacje o tym, jak można łatwo uruchamiać programy Pythona w różnych systemach operacyjnych).

Pomysły na podobne programy

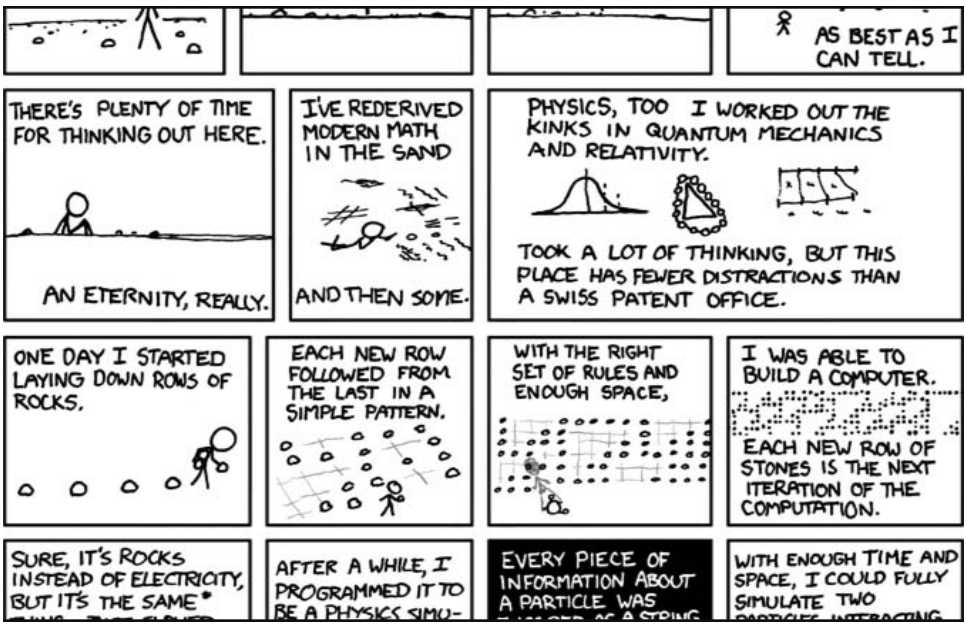
Zaletą kart w przeglądarce WWW jest możliwość łatwego otwierania łączy w nowych kartach, z którymi będzie można zapoznać się później. Program automatycznie otwierający jednocześnie kilka łączy może być wygodnym skrótem dla niektórych zadań. Oto one.

- Otworzenie wszystkich stron produktu po przeprowadzeniu wyszukiwania w witrynie, takiej jak Amazon.
- Otworzenie wszystkich łączy prowadzących do opinii o danym produkcie.
- Otworzenie łączy wyników do zdjęcia po przeprowadzeniu wyszukiwania zdjęcia w witrynach, takich jak Flickr lub Imgur.

Projekt — pobranie wszystkich komiksów z witryny XKCD

Blogi i inne regularnie uaktualniane witryny internetowe zwykle mają stronę główną wraz z aktualnym postem oraz przycisk typu *Poprzedni* pozwalający na przejście do poprzedniego postu. Wcześniejszy post również ma przycisk *Poprzedni* i tak dalej. W ten sposób powstaje ścieżka od najnowszego do najstarszego postu opublikowanego w danej witrynie internetowej. Jeżeli chcesz zapoznać się z treścią strony, gdy nie masz połączenia z internetem, musisz ręcznie przechodzić pomiędzy poszczególnymi stronami i zapisywać każdą z nich. Jednak to niezwykle nużące zadanie i dlatego warto opracować program, który będzie to robił automatycznie.

Pokazana na rysunku 11.6 XKCD to publikująca komiksy popularna witryna internetowa doskonale wpisująca się w przedstawiony powyżej schemat. Na stronie głównej witryny XKCD pod adresem *http://xkcd.com/* znajduje się przycisk *Prev* pozwalający na przejście do wcześniejszych komiksów. Ręczne pobranie każdego komiksu zajmie wieki, ale za pomocą odpowiedniego skryptu będzie można zrobić to w ciągu zaledwie kilku minut.



Rysunek 11.6. XKCD to witryna internetowa publikująca komiksy o różnej tematyce, np. przykład romantyzm, sarkazm, matematyka, język i tak dalej

Poniżej wymieniłem w punktach sposób działania programu.

- Wczytanie strony głównej XKCD.
- Zapisanie obrazu zawierającego komiks opublikowany na danej stronie.
- Przejście na stronę wskazywaną przez łącze *Previous Comic*.
- Powtarzanie operacji, dopóki nie zostanie pobrany najstarszy komiks.

Oznacza to, że kod będzie musiał wykonywać przedstawione poniżej zadania.

- Pobieranie stron za pomocą modułu `requests`.
- Odszukanie za pomocą modułu `BeautifulSoup` adresu URL obrazu komiksu.
- Pobranie i zapisanie na dysku twardym obrazu komiksu za pomocą metody `iter_content()`.
- Odszukanie adresu URL łącza *Previous Comic* i powtórzenie całej procedury od początku.

Otwórz nowe okno edytora pliku i zapisz plik pod nazwą *downloadXkcd.py*.

Etap 1. Projekt programu

Jeżeli po wyświetleniu strony komiksu otworzysz wbudowane w przeglądarkę narzędzia dla programistów i przeanalizujesz elementy na stronie, będziesz mógł poczynić następujące ustalenia.

- Adres URL pliku obrazu komiksu jest zdefiniowany w atrybucie `href` elementu `<img>`.
- Element `<img>` znajduje się wewnątrz elementu `<div id="comic">`.
- Przycisk *Prev* ma atrybut HTML `rel` o wartości `prev`.
- Przycisk *Prev* najstarszego komiksu ma adres URL w postaci *http://xkcd.com/#*, co wskazuje na brak wcześniejszych stron.

Wprowadź w pliku przedstawiony poniżej fragment kodu.

```
#! python3
# downloadXkcd.py — Pobiera wszystkie komiksy opublikowane w witrynie XKCD.

import requests, os, bs4

url = 'http://xkcd.com'                 # Początkowy adres URL.
os.makedirs('xkcd', exist_ok=True)      # Komiksy są przechowywane w katalogu ./xkcd.
while not url.endswith('#'):
    # TODO: Pobranie strony.

    # TODO: Ustalenie adresu URL pliku obrazu komiksu.

    # TODO: Pobranie obrazu.

    # TODO: Zapis obrazu w katalogu ./xkcd.

    # TODO: Pobranie adresu URL w przycisku Prev.

print('Gotowe!')
```

W powyższym fragmencie kodu mamy zmienną o nazwie `url` rozpoczynającą się od ciągu tekstowego `'http://xkcd.com'`. Wartość tej zmiennej jest nieustannie uaktualniana (w pętli `while`) adresem URL łącza *Prev* znajdującego się na bieżącej stronie. W trakcie każdej iteracji pętli jest pobierany plik obrazu komiksu znajdujący się pod adresem wskazywanym przez zmienną `url`. Gdy adres URL będzie się kończył znakiem `#`, nastąpi opuszczenie pętli `while`.

Plik obrazu zostanie pobrany do podkatalogu o nazwie *xkcd* umieszczonego w bieżącym katalogu roboczym. Wywołanie `os.makedirs()` gwarantuje istnienie wymienionego katalogu. Argument w postaci `exist_ok=True` uniemożliwia funkcji zgłoszenie wyjątku, jeśli ten katalog już istnieje. Pozostała część kodu to po prostu komentarze przedstawiające resztę programu.

Etap 2. Pobranie strony internetowej

Przechodzimy do implementacji kodu odpowiedzialnego za pobranie strony internetowej. Wprowadź w kodzie przedstawione poniżej zmiany.

```
#! python3
# downloadXkcd.py — Pobiera wszystkie komiksy opublikowane w witrynie XKCD.

import requests, os, bs4

url = 'http://xkcd.com'                 # Początkowy adres URL.
os.makedirs('xkcd', exist_ok=True)      # Komiksy są przechowywane w katalogu ./xkcd.
while not url.endswith('#'):
    # Pobranie strony.
    print('Pobieranie strony %s...' % url)
    res = requests.get(url)
    res.raise_for_status()

    soup = bs4.BeautifulSoup(res.text)

    # TODO: Ustalenie adresu URL pliku obrazu komiksu.

    # TODO: Pobranie obrazu.

    # TODO: Zapis obrazu w katalogu ./xkcd.

    # TODO: Pobranie adresu URL w przycisku Prev.

print('Gotowe!')
```

Najpierw wyświetlana jest wartość zmiennej `url`, aby użytkownik wiedział, z jakiego adresu URL program pobiera dane. Następnie za pomocą funkcji `requests.get()` modułu `requests` faktycznie pobieramy stronę. Jak zwykle, kolejnym wywołaniem po pobraniu danych jest metoda `raise_for_status()` obiektu `Response` w celu zgłoszenia wyjątku i zakończenia działania programu, jeśli wystąpił jakikolwiek problem podczas pobierania danych. Gdy wszystko przebiegło bez zakłóceń, na podstawie tekstu pobranej strony tworzymy obiekt `BeautifulSoup`.

Etap 3. Odszukanie i pobranie obrazu komiksu

W kodzie programu wprowadź przedstawione poniżej zmiany.

```
#! python3
# downloadXkcd.py — Pobiera wszystkie komiksy opublikowane w witrynie XKCD.

import requests, os, bs4

--cięcie--

    # Ustalenie adresu URL pliku obrazu komiksu.
    comicElem = soup.select('#comic img')
    if comicElem == []:
```

```
        print('Nie udało się odnaleźć pliku obrazu komiksu.')
    else:
        comicUrl = 'http:' + comicElem[0].get('src')
        # Pobranie obrazu.
        print('Pobieranie obrazu %s...' % (comicUrl))
        res = requests.get(comicUrl)
        res.raise_for_status()

    # TODO: Zapis obrazu w katalogu ./xkcd.

    # TODO: Pobranie adresu URL w przycisku Prev.

print('Gotowe!')
```

Po przeprowadzonej za pomocą narzędzi dla programistów analizie strony głównej XKCD wiesz, że element `<img>` pliku obrazu komiksu znajduje się wewnątrz elementu `<div>` zawierającego atrybut `id` o wartości `comic`. Dlatego też selektor `'#comic img'` pozwoli na wyodrębnienie właściwego elementu `<img>` z obiektu `BeautifulSoup`.

Kilka stron XKCD ma treść specjalną, która nie jest po prostu plikiem obrazu. Nie stanowi to żadnego problemu, po prostu je pominiemy. Jeżeli selektor nie dopasuje żadnych elementów, wówczas wartością zwrotną `soup.select('#comic img')` będzie pusta lista. W takim przypadku program może po prostu wyświetlić komunikat błędu i przejść dalej bez pobierania obrazu.

W przeciwnym razie selektor zwróci listę, na której znajduje się jeden element `<img>`. Teraz wystarczy wartość jego atrybutu `src` przekazać do wywołania `requests.get()`, aby tym samym pobrać plik obrazu komiksu.

Etap 4. Zapis obrazu i odszukanie poprzedniego komiksu

W kodzie programu wprowadź przedstawione poniżej zmiany.

```
#! python3
# downloadXkcd.py — Pobiera wszystkie komiksy opublikowane w witrynie XKCD.

import requests, os, bs4

--cięcie--

        # Zapis obrazu w katalogu ./xkcd.
        imageFile = open(os.path.join('xkcd', os.path.basename(comicUrl)), 'wb')
        for chunk in res.iter_content(100000):
            imageFile.write(chunk)
        imageFile.close()

    # Pobranie adresu URL w przycisku Prev.
    prevLink = soup.select('a[rel="prev"]')[0]
    url = 'http://xkcd.com' + prevLink.get('href')

print('Gotowe!')
```

Na tym etapie plik obrazu komiksu jest przechowywany w zmiennej `res`. Trzeba więc zapisać obraz do pliku na dysku twardym.

Wywołaniu funkcji `open()` należy przekazać nazwę pliku. Zmienna `comicUrl` będzie miała wartość, taką jak `'http://imgs.xkcd.com/comics/heartbleed _ex ↪planation.png'`. Jak możesz zauważyć, przypomina ona ścieżkę dostępu do pliku. Wartość tej zmiennej można przekazać wywołaniu `os.path.basename()`, aby w wyniku otrzymać ostatnią część adresu URL, czyli `'heartbleed _explanation.png'`. Następnie tę część możemy wykorzystać w charakterze nazwy pliku dla obrazu zapisywanego na dysku twardym. Za pomocą wywołania `os.path.join()` łączymy tę nazwę wraz z nazwą katalogu *xkcd*, ponieważ dzięki wymienionej funkcji program użyje lewych ukośników (\) w systemie Windows oraz prawych ukośników (/) w systemach macOS i Linux. Gdy mamy określoną nazwę pliku, można już wywołać funkcję `open()` wraz z atrybutem `'wb'`, aby otworzyć nowy plik w trybie binarnym.

Powinieneś już pamiętać, bo pisałem o tym na początku tego rozdziału, że w celu zapisania plików pobieranych za pomocą modułu `requests` konieczne jest przeprowadzenie iteracji przez wartość zwrotną metody `iter_content()`. Kod w pętli `for` zapisuje dane w pliku obrazu we fragmentach o maksymalnej wielkości 100000 bajtów, a następnie zamyka plik. W tym momencie plik obrazu znajduje się już na dysku twardym Twojego komputera.

Dalej selektor `'a[rel="prev"]'` identyfikuje element `<a>` wraz z atrybutem `rel` o wartości `prev`. Wartość atrybutu `href` tego elementu zawiera adres URL wcześniejszego komiksu, który zapisujemy w zmiennej `url`. Następnie pętla `while` ponownie rozpoczyna proces pobierania danych dla wskazanego komiksu.

Dane wyjściowe wygenerowane przez ten program będą przedstawiały się podobnie do pokazanych poniżej.

```
Pobieranie strony http://xkcd.com...
Pobieranie obrazu http://imgs.xkcd.com/comics/phone_alarm.png...
Pobieranie strony http://xkcd.com/1358/...
Pobieranie obrazu http://imgs.xkcd.com/comics/nro.png...
Pobieranie strony http://xkcd.com/1357/...
Pobieranie obrazu http://imgs.xkcd.com/comics/free_speech.png...
Pobieranie strony http://xkcd.com/1356/...
Pobieranie obrazu http://imgs.xkcd.com/comics/orbital_mechanics.png...
Pobieranie strony http://xkcd.com/1355/...
Pobieranie obrazu http://imgs.xkcd.com/comics/airplane_message.png...
Pobieranie strony http://xkcd.com/1354/...
Pobieranie obrazu http://imgs.xkcd.com/comics/heartbleed_explanation.png...
--cięcie--
```

Ten projekt jest dobrym przykładem programu, który może automatycznie podążać za łączami w celu pobrania ogromnej ilości danych z internetu. Więcej informacji na temat pozostałych funkcji modułu `BeautifulSoup` znajdziesz w dokumentacji dostępnej na stronie *https://www.crummy.com/software/BeautifulSoup/bs4/doc/*.

Pomysły na podobne programy

Pobieranie stron i podążanie za łączami to podstawowe zadania wielu programów pobierających dane z internetu. Podobne programy mogą wykonywać przedstawione poniżej operacje.

- Utworzenie kopii zapasowej całej witryny internetowej przez podążanie za jej wszystkimi łączami.
- Skopiowanie wszystkich postów opublikowanych na forum internetowym.
- Powielenie katalogu produktów na wyprzedaży w sklepie internetowym.

Moduły `requests` i `BeautifulSoup` sprawdzają się doskonale, o ile jesteś w stanie ustalić adres URL, który należy przekazać wywołaniu `requests.get()`. Jednak czasami określenie takiego adresu nie jest łatwe. Ewentualnie witryna internetowa, po której ma się poruszać Twój program, wymaga najpierw zalogowania użytkownika. Moduł o nazwie `selenium` daje Twoim programom potężne możliwości w zakresie wykonywania tak skomplikowanych zadań.

Kontrolowanie przeglądarki WWW za pomocą modułu selenium

Moduł `selenium` pozwala Pythonowi na bezpośrednie, programowe kontrolowanie przeglądarki WWW przez klikanie łączy i wypełnianie formularzy sieciowych. To wszystko odbywa się w prawie taki sam sposób, w jaki ze strony internetowej korzysta człowiek. Moduł `selenium` pozwala na współdziałanie ze stronami internetowymi w znacznie bardziej zaawansowany sposób niż oferowany przez moduły `requests` i `BeautifulSoup`. Ponieważ uruchamia przeglądarkę WWW, więc działa nieco wolniej. Ponadto trudno nie zauważyć jego działania w tle, jeśli trzeba na przykład pobrać pewne pliki z internetu.

Więcej informacji oraz dokładne omówienie instalacji modułów opracowanych przez firmy trzecie znajdziesz w dodatku A.

Uruchomienie przeglądarki WWW kontrolowanej przez moduł selenium

W omawianym tutaj przykładzie wykorzystamy przeglądarkę WWW o nazwie Firefox. Tę właśnie przeglądarkę będziemy kontrolować. Jeżeli jeszcze nie zainstalowałeś przeglądarki Firefox, możesz ją pobrać z witryny *http://getfirefox.com/*.

Import modułu `selenium` przebiega nieco inaczej niż modułów wcześniejszych. Zamiast polecenia `import selenium` trzeba wydać polecenie `from selenium import webdriver`. (Dokładne wyjaśnienie powodu, dla którego moduł `selenium` został skonfigurowany w taki właśnie sposób wykracza poza zakres tematyczny książki).

Teraz będzie już można uruchomić przeglądarkę Firefox kontrolowaną przez moduł `selenium`[3]. W powłoce interaktywnej wprowadź przedstawione poniżej polecenia.

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> type(browser)
<class 'selenium.webdriver.firefox.webdriver.WebDriver'>
>>> browser.get('http://inventwithpython.com')
```

Zauważysz, że po wywołaniu `webdriver.Firefox()` następuje uruchomienie wskazanej przeglądarki WWW (tutaj to Firefox). Wywołanie `type()` z wartością udostępnioną przez `webdriver.Firefox()` ujawnia typ danych `WebDriver`. Natomiast wywołanie `browser.get('http://inventwithpython.com')` powoduje przekierowanie przeglądarki WWW do witryny *http://inventwithpython.com/*. W tym momencie okno przeglądarki Firefox powinno wyglądać tak, jak pokazałem na rysunku 11.7.

Wyszukanie elementów na stronie

Obiekty typu `WebDriver` mają całkiem sporą ilość metod przeznaczonych do wyszukiwania elementów na stronie. Zostały podzielone na metody typu `find_ele ➥ment_*` i `find_elements_*`. Metody typu `find_element_*` zwracają pojedynczy obiekt `WebElement` przedstawiający pierwszy element na stronie, który został dopasowany do zapytania. Z kolei metody typu `find_elements_*` zwracają listę obiektów `WebElement_*` dla *każdego* dopasowanego elementu na stronie.

W tabeli 11.3 wymieniłem kilka przykładów metod typu `find_element_*` i `find_elements_*` wywoływanych w obiekcie `WebDriver`, który jest przechowywany w zmiennej o nazwie `browser`.

Poza metodami `*_by_tag_name()`, argumenty wszystkich pozostałych metod uwzględniają wielkość znaków. Jeżeli na stronie nie istnieją elementy możliwe do dopasowania przez metodę, moduł `selenium` zwróci wyjątek `NoSuchElement`. Jeśli nie chcesz, aby ten wyjątek doprowadził do awarii programu, w kodzie powinieneś zastosować polecenia `try` i `except`.

Więcej informacji o obiekcie `WebElement` można zdobyć przez odczyt atrybutów lub wywołanie metod wymienionych w tabeli 11.4.

[3] Dla używanej przeglądarki WWW (tutaj Firefox) niezbędny jest jeszcze sterownik. W omawianym przykładzie skorzystamy ze sterownika Geckodriver, który możesz pobrać ze strony *https://github.com/mozilla/geckodriver/releases*. Po rozpakowaniu archiwum otrzymasz plik *geckodriver.exe*. Ponieważ musi znajdować się w katalogu podanym w zmiennej systemowej `path`, więc najlepiej skopiuj go w katalogu, w którym zainstalowałeś Pythona — *przyp. tłum.*

Rysunek 11.7. Po wywołaniach webdriver.Firefox() i get() w środowisku IDLE na ekranie pojawia się okno wskazanej przeglądarki WWW (tutaj to Firefox)

Otwórz nowe okno edytora pliku i wpisz w nim poniższy fragment kodu.

```
from selenium import webdriver
browser = webdriver.Firefox()
browser.get('http://inventwithpython.com')
try:
    elem = browser.find_element_by_class_name('bookcover')
    print('Znaleziono element <%s> wraz z taką nazwą klasy!' % (elem.tag_name))
except:
    print('Nie udało się znaleźć elementu wraz z podaną nazwą klasy.')
```

W powyższym fragmencie kodu uruchamiamy przeglądarkę WWW o nazwie Firefox i przekierowujemy ją do podanego adresu URL. Na tej stronie próbujemy wyszukać elementy o nazwie klasy `'bookcover'`. Jeżeli tego rodzaju element istnieje, wyświetlamy jego nazwę za pomocą atrybutu `tag_name`. Natomiast w przypadku nieznalezienia elementu wyświetlony będzie zupełnie inny komunikat.

Tabela 11.3. Oferowane przez moduł selenium metody obiektu WebDriver przeznaczone do wyszukiwania elementów

Nazwa metody	Zwrócony obiekt lub lista obiektów WebDriver
`browser.find_element_by_class_name(`*`nazwa`*`)` `browser.find_elements_by_class_name(`*`nazwa`*`)`	Elementy używające klasy CSS o podanej *`nazwie`*.
`browser.find_element_by_css_selector(`*`selektor`*`)` `browser.find_elements_by_css_selector(`*`selektor`*`)`	Elementy dopasowane przez *`selektor`* CSS.
`browser.find_element_by_id(`*`id`*`)` `browser.find_elements_by_id(`*`id`*`)`	Elementy o dopasowanej wartości atrybutu `id`.
`browser.find_element_by_link_text(`*`tekst`*`)` `browser.find_elements_by_link_text(`*`tekst`*`)`	Elementy <a>, które zawierają całkowicie dopasowany podany *`tekst`*.
`browser.find_element_by_partial_link_text(`*`tekst`*`)` `browser.find_elements_by_partial_link_text(`*`tekst`*`)`	Elementy <a>, które zawierają podany *`tekst`*.
`browser.find_element_by_name(`*`nazwa`*`)` `browser.find_elements_by_name(`*`nazwa`*`)`	Elementy o dopasowanej wartości atrybutu *`nazwa`*.
`browser.find_element_by_tag_name(`*`nazwa`*`)` `browser.find_elements_by_tag_name(`*`nazwa`*`)`	Elementy o dopasowanej *`nazwie`* znacznika (wielkość liter nie ma znaczenia, element <a> będzie dopasowany zarówno przez `'a'`, jak i `'A'`).

Tabela 11.4. Atrybuty i metody obiektu WebElement

Atrybut lub metoda	Opis
`tag_name`	Nazwa znacznika, na przykład `'a'` dla elementu <a>.
`get_attribute(`*`nazwa`*`)`	Wartość atrybutu o podanej *`nazwie`* w elemencie.
`text`	Tekst wewnątrz elementu, na przykład `'witaj'` w elemencie witaj.
`clear()`	W przypadku elementów pola lub obszaru tekstowego ta metoda powoduje usunięcie tekstu wyświetlanego przez element.
`is_displayed()`	Zwraca wartość `True`, jeśli element jest widoczny, w przeciwnym razie wartością zwrotną jest `False`.
`is_enabled()`	W przypadku elementów danych wejściowych zwraca wartość `True`, jeśli element jest włączony. W przeciwnym razie wartością zwrotną jest `False`.
`is_selected()`	W przypadku elementów, takich jak pole wyboru lub pole opcji, zwraca wartość `True`, jeśli element jest włączony. W przeciwnym razie wartością zwrotną jest `False`.
`location`	Słownik wraz z kluczami `'x'` i `'y'` wskazującymi położenie elementu na stronie.

Program wygeneruje następujące dane wyjściowe.

```
Znaleziono element <img> wraz z taką nazwą klasy!
```

Udało nam się znaleźć element o nazwie klasy `'bookcover'` oraz nazwie znacznika `'img'`.

Kliknięcie na stronie

Obiekty WebElement zwracane przez metody typu `element_*` i `find_elements_*` zawierają metodę `click()` symulującą kliknięcie myszą tego elementu. Metoda ta może być używana w celu podążania za łączem, dokonywania wyboru za pomocą przycisków opcji, kliknięcia przycisku wysyłającego formularz lub wywołania każdej innej akcji, która następuje po kliknięciu elementu myszą. W powłoce interaktywnej wprowadź przedstawione poniżej polecenia.

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> browser.get('http://inventwithpython.com')
>>> linkElem = browser.find_element_by_link_text('Read It Online')
>>> type(linkElem)
<class 'selenium.webdriver.remote.webelement.WebElement'>
>>> linkElem.click()       # Podążanie za łączem zatytułowanym "Read It Online".
```

W powyższym fragmencie kodu przeglądarka Firefox przechodzi do witryny *http://inventwithpython.com/*, pobiera obiekt `WebElement` dla elementu `<a>` zawierającego tekst *Read It Online*, a następnie symuluje kliknięcie tego elementu `<a>`. To przypomina kliknięcie łącza przez człowieka, a przeglądarka WWW podąża za tym łączem.

Wypełnianie i wysyłanie formularzy sieciowych

Symulowanie naciśnięcia klawiszy w polu tekstowym na stronie internetowej sprowadza się do odszukania elementu `<input>` lub `<textarea>` dla danego pola tekstowego, a następnie do wywołania metody `send_keys()`. W powłoce interaktywnej wprowadź przedstawione poniżej polecenia.

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> browser.get('http://mail.yahoo.com')
>>> emailElem = browser.find_element_by_id('login-username')
>>> emailElem.send_keys('nieprawdziwy_adres_e-mail')
>>> passwordElem = browser.find_element_by_id('login-passwd')
>>> passwordElem.send_keys('12345')
>>> passwordElem.submit()
```

O ile Yahoo! nie zmieni identyfikatora pól tekstowych przeznaczonych do podania nazwy użytkownika i hasła, przedstawiony powyżej fragment kodu spowoduje wypełnienie tych pól podanym tekstem. (Za pomocą wbudowanych w przeglądarkę WWW narzędzi programistycznych zawsze możesz sprawdzić wspomniane identyfikatory). Wywołanie metody `submit()` w dowolnym elemencie ma dokładnie taki sam efekt jak kliknięcie przycisku *Wyślij* formularza zawierającego ten element. (Równie dobrze można użyć wywołania `emailElem.submit()`, a omawiany kod nadal będzie wykonywał to samo zadanie).

Symulacja naciśnięcia klawiszy specjalnych

Moduł `selenium` zapewnia obsługę klawiszy, których nie można zasymulować przez użycie pewnych wartości w postaci ciągu tekstowego. Działanie tego rodzaju klawiszy przypomina sekwencje sterujące. Wartości symulujące naciśnięcia tych klawiszy są przechowywane w module `selenium.webdriver.common.keys`. Ponieważ nazwa modułu jest bardzo długa, warto umieszczać na początku programu polecenie `from selenium.webdriver.common.keys import Keys`. Dzięki temu możesz później używać polecenia `Keys` wszędzie tam, gdzie normalnie musiałbyś pisać `selenium.webdriver.common.keys`. W tabeli 11.5 wymieniłem najczęściej używane zmienne `Keys`.

Tabela 11.5. Powszechnie używane zmienne zdefiniowane w module selenium.webdriver.common.keys

Atrybuty	Opis
`Keys.DOWN`, `Keys.UP`, `Keys.LEFT`, `Keys.RIGHT`	Klawisze kursora na klawiaturze.
`Keys.ENTER`, `Keys.RETURN`	Klawisze *Enter* i *Return*.
`Keys.HOME`, `Keys.END`, `Keys.PAGE_DOWN`, `Keys.PAGE_UP`	Klawisze *Home*, *End*, *Pagedown* i *Pageup*.
`Keys.ESCAPE`, `Keys.BACK_SPACE`, `Keys.DELETE`	Klawisze *Esc*, *Backspace* i *Delete*.
`Keys.F1`, `Keys.F2`, ..., `Keys.F12`	Klawisze od *F1* do *F12* znajdujące się na górze klawiatury.
`Keys.TAB`	Klawisz *Tab*.

Jeśli na przykład kursor aktualnie nie znajduje się w polu tekstowym, wówczas naciskając klawisze *Home* i *End*, można poruszać się między — odpowiednio — początkiem i końcem strony. W powłoce interaktywnej wprowadź przedstawione poniżej polecenia i zwróć uwagę, jak za pomocą wywołań `send_keys()` można przewijać zawartość strony.

```
>>> from selenium import webdriver
>>> from selenium.webdriver.common.keys import Keys
>>> browser = webdriver.Firefox()
>>> browser.get('http://nostarch.com')
>>> htmlElem = browser.find_element_by_tag_name('html')
>>> htmlElem.send_keys(Keys.END)     # Przewinięcie na koniec strony.
>>> htmlElem.send_keys(Keys.HOME)    # Przewinięcie na początek strony.
```

Znacznik `<html>` jest znacznikiem bazowym w plikach HTML. Cała zawartość dokumentu HTML jest umieszczona między znacznikami `<html>` i `</html>`. Wywołanie `browser.find_element_by_Tag_name('html')` to dobry początek na symulowanie naciśnięć klawiszy na stronie. Takie podejście będzie użyteczne, jeśli na przykład nowa zawartość strony jest wczytywana po jej przewinięciu do końca.

Klikanie przycisków przeglądarki WWW

Moduł `selenium` może symulować także klikanie różnych przycisków przeglądarki WWW. Do tego celu służą wymienione poniżej metody.

- `browser.back()`. Kliknięcie przycisku *Wstecz*.
- `browser.forward()`. Kliknięcie przycisku *Do przodu*.
- `browser.refresh()`. Kliknięcie przycisku *Odśwież*.
- `browser.quit()`. Kliknięcie przycisku *Zamknij okno*.

Więcej informacji na temat modułu selenium

Możliwości oferowane przez moduł `selenium` są znacznie większe niż przedstawione w rozdziale. Moduł pozwala na modyfikację plików cookie przeglądarki WWW, wykonywanie rzutów stron internetowych, a także na uruchamianie własnego kodu JavaScript. Jeżeli chcesz dowiedzieć się więcej na temat tych możliwości, zajrzyj do oficjalnej dokumentacji modułu `selenium`, którą znajdziesz na stronie *http://selenium-python.readthedocs.io/*.

Podsumowanie

Większość nudnych zajęć nie ogranicza się wyłącznie do związanych z plikami znajdującymi się na dysku twardym komputera lokalnego. Możliwość programowego pobierania stron internetowych pozwala na rozbudowę programów, które w ten sposób mogą się łączyć z internetem. Moduł `requests` znacznie ułatwia pobieranie danych. Mając nawet jedynie podstawową wiedzę o koncepcjach stosowanych w HTML i selektorach CSS, można wykorzystać moduł `BeautifulSoup` do przetwarzania pobieranych stron internetowych.

Jednak w celu przeprowadzenia pełnej automatyzacji zadań związanych z siecią WWW będziesz musiał przejąć bezpośrednią kontrolę nad przeglądarką WWW, na co pozwala moduł `selenium`. Moduł ten umożliwia automatyczne logowanie się w witrynach internetowych oraz wypełnianie formularzy sieciowych. Ponieważ przeglądarka WWW to obecnie narzędzie, za pomocą którego najczęściej wysyłamy i pobieramy informacje przez internet, warto mieć moduł `selenium` w arsenale dostępnych narzędzi.

Pytania kontrolne

1. Pokrótce omów różnice między modułami `webbrowser`, `requests`, `BeautifulSoup` i `selenium`.
2. Jaki typ obiektów jest zwracany przez funkcję `requests.get()`? Jak można uzyskać dostęp do pobranej treści jako ciągu tekstowego?

3. Jaka metoda modułu `requests` pozwala na sprawdzenie, czy dane zostały pobrane prawidłowo?
4. Jak można otrzymać kody stanu HTTP dla odpowiedzi udzielonej na żądania wysyłane za pomocą modułu `requests`?
5. Jak można zapisać w pliku odpowiedź udzieloną na żądanie wysłane za pomocą modułu `requests`?
6. Jaki jest skrót klawiszowy pozwalający na otworzenie wbudowanych w przeglądarkę WWW narzędzi dla programistów?
7. Jak można wyświetlić (za pomocą narzędzi dla programistów) kod HTML określonego elementu na stronie internetowej?
8. Jak przedstawia się ciąg tekstowy selektora CSS dopasowującego element o atrybucie `id` o wartości `main`?
9. Jak przedstawia się ciąg tekstowy selektora CSS dopasowującego elementy zawierające klasę CSS o nazwie `highlight`?
10. Jak przedstawia się ciąg tekstowy selektora CSS dopasowującego wszystkie elementy `<div>` znajdujące się wewnątrz innego elementu `<div>`?
11. Jak przedstawia się ciąg tekstowy selektora CSS dopasowującego element `<button>` wraz z atrybutem `value` o wartości `favorite`?
12. Przyjmujemy założenie, że masz obiekt `Tag` modułu `BeautifulSoup` przechowywany w zmiennej `spam` dla elementu `<div>Witaj, świecie!</div>`. Jak możesz pobrać ciąg tekstowy `'Witaj, świecie!'` z tego obiektu `Tag`?
13. Jak będziesz przechowywać wszystkie atrybuty obiektu `Tag` modułu `BeautifulSoup` w zmiennej o nazwie `linkElem`?
14. Wykonanie polecenia `import selenium` nie działa. Jak prawidłowo zaimportujesz moduł `selenium`?
15. Jaka jest różnica między metodami typu `find_element_*` i `find_elements_*`?
16. Jakie metody ma obiekt `WebElement` modułu `selenium` przeznaczony do symulowania kliknięć myszą i naciśnięć klawiszy na klawiaturze?
17. Masz możliwość wykonania wywołania `send_keys(Keys.ENTER)` w przycisku wysyłającym formularz sieciowy w obiekcie `WebElement`. Jaki jest jeszcze łatwiejszy sposób na wysłanie formularza sieciowego za pomocą modułu `selenium`?
18. Jak za pomocą modułu `selenium` można symulować kliknięcie przycisków przeglądarki WWW, takich jak wstecz, do przodu i odśwież.

Projekty praktyczne

W celu zdobycia doświadczenia utwórz programy wykonujące omówione poniżej zadania.

Klient poczty działający w wierszu poleceń

Utwórz program, który będzie pobierał argumenty wiersza poleceń w postaci adresu e-mail i ciągu tekstowego, a następnie za pomocą modułu `selenium` zaloguje się do Twojego konta e-mail i wyświetli wiadomość. Adresat wiadomości i jej treść są podawane jako argumenty programu. (Dla tego programu rozsądne może być utworzenie oddzielnego konta poczty elektronicznej).

Dobrze byłoby dodać funkcję powiadamiania. Możesz utworzyć również podobny program przeznaczony do wysyłania komunikatów z serwisów Facebook lub Twitter.

Pobieranie obrazów z witryny internetowej

Utwórz program, który będzie korzystać z serwisów przeznaczonych do dzielenia się zdjęciami, na przykład takich jak Flickr i Imgur. Zadaniem programu ma być wyszukiwanie pewnej kategorii zdjęć, a następnie pobranie wszystkich zwróconych w wyniku wyszukiwania. Możesz też utworzyć program, który będzie współdziałał z dowolnym serwisem dzielenia się zdjęciami oferującym funkcję wyszukiwania.

2048

2048 to prosta gra, w której zadaniem gracza jest łączenie pól przez ich przesuwanie w górę, w dół, w lewo i w prawo za pomocą klawiszy kursora. W tej grze można uzyskać bardzo wysoki wynik, nieustannie przesuwając pola zgodnie ze wzorcem w górę, w prawo, w dół, w lewo. Utwórz program otwierający stronę z grą 2048 dostępną pod adresem *https://gabrielecirulli.github.io/2048/*, a następnie przekazujący naciśnięcia klawiszy w górę, w prawo, w dół i w lewo, aby faktycznie symulować grę.

Weryfikacja łącza

Utwórz program, który po otrzymaniu adresu URL strony internetowej spróbuje pobrać wszystkie strony internetowe, do których prowadzą łącza z podanej. Program powinien oznaczać wszystkie strony generujące kod stanu HTTP 404 (czyli „nie znaleziono") i wyświetlać je na liście nieprawidłowych łączy.

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

W

Z

Ż

NIECH TWÓJ KOMPUTER CIĘ WYRĘCZY!

Komputer jest niezwykle wszechstronnym narzędziem, które może posłużyć do błyskawicznego wykonywania przeróżnych czynności. Dotyczy to również zadań, które zwykle zajmują mnóstwo czasu, a przy tym są męczące i nudne. Zamiast przez wiele godzin porównywać czy wprowadzać dane, lepiej dobrze zaprogramować komputer, który wykona takie zadania w ciągu kilku sekund. Aby to osiągnąć, wcale nie trzeba być profesjonalnym programistą!

Jeśli nie masz pojęcia o programowaniu, ale chcesz wykorzystać komputer do automatyzacji prostych, powtarzalnych operacji, trzymasz w ręku właściwą książkę. Podstawy programowania przedstawiono tu w sposób klarowny i zrozumiały, poszczególne działania wyjaśniono krok po kroku, a propozycje praktycznych projektów pozwalają na gruntowne przyswojenie materiału i poszerzenie wiedzy. Do nauki wykorzystano język Python, ponieważ jest on stosunkowo prosty, popularny, a przy tym wciąż rozwijany. Szybko przekonasz się, ile czasu i wysiłku można zaoszczędzić dzięki automatyzacji zadań za pomocą krótkich i łatwych programów!

W książce znajdziesz między innymi:

- podstawy programowania w Pythonie
- opis kontroli przepływu działania programu
- informacje o pracy na danych tekstowych i plikach
- wiadomości o pracy z plikami CSV i danymi JSON
- harmonogramy zadań i opis kontroli czasu uruchamiania programu
- możliwości kontrolowania klawiatury i myszy za pomocą automatyzacji GUI

Albert Sweigart ····· programista, projektant oprogramowania i nauczyciel kodowania. Autor licznych książek o Pythonie dla początkujących. Urodził się w Houston w stanie Teksas, a obecnie mieszka w San Francisco. Prowadzi blog dostępny pod adresem: *http://coffeeghost.net*.

ISBN 978-83-283-3260-7

9 788328 332607

cena: 89,00 zł

no starch press