Poniższy rozdział pochodzi z książki:

Charles R. Severance - "Python dla wszystkich: Odkrywanie danych z Python 3"

Pełna wersja podręcznika znajduje się na stronie https://py4e.pl/book

Rozdział 13

Korzystanie z usług sieciowych

Kiedy pozyskiwanie dokumentów przez HTTP i ich przetwarzanie stało się łatwe, ludzie szybko opracowali sposoby na tworzenie dokumentów specjalnie zaprojektowanych do wykorzystania przez inne programy (w odróżnieniu od kodu HTML, który ma być wyświetlany w przeglądarce).

Istnieją dwa popularne formaty, których używamy przy wymianie danych w sieci. Format XML (eXtensible Markup Language, z ang. rozszerzalny język znaczników) jest używany od bardzo dawna i najlepiej nadaje się do wymiany danych w formie dokumentów. Gdy programy chcą tylko wymieniać ze sobą słowniki, listy lub inne wewnętrzne informacje, używają formatu JSON (JavaScript Object Notation, z ang. notacja obiektowa JavaScript, patrz https://www.json.org). Przyjrzymy się obu formatom.

13.1. XML

Kod XML wygląda bardzo podobnie do kodu HTML, ale jest od niego nieco bardziej uporządkowany. Oto fragment dokumentu XML:

```
<osoba>
  <imie>Chuck</imie>
  <telefon typ="miedzynar">
    +1 734 303 4456
  </telefon>
  <email ukryty="tak" />
</osoba>
```

Każda para znaczników: otwierający (np. `<osoba>`) i zamykający (np. `</osoba>`), reprezentuje *element* lub *węzeł* (ang. *node*) o tej samej nazwie co znacznik (np. `osoba`). Każdy element może zawierać jakiś tekst, pewne atrybuty (np. `ukryty`) i inne zagnieżdżone elementy. Jeśli element XML jest pusty (tzn. nie ma treści), to może być przedstawiony za pomocą samozamykającego się znacznika (np. `<email />`).

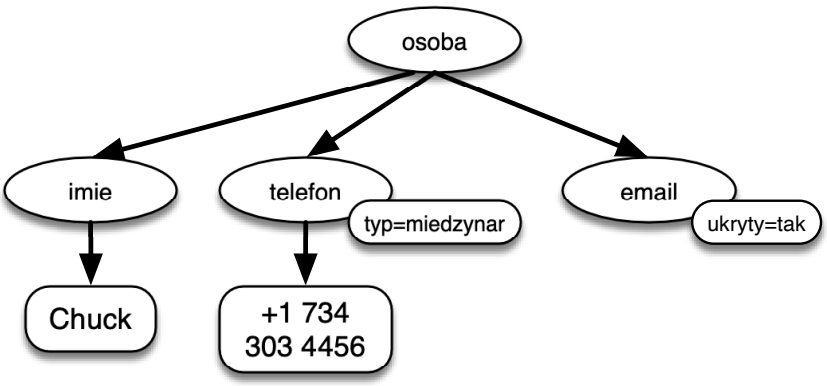
Często pomocne jest myślenie o dokumencie XML jako o strukturze drzewa, w którym znajduje się najwyższy element (tutaj: `osoba`), a inne znaczniki (np. `telefon`) są oznaczane jako *dzieci* elementów będących ich *rodzicami*.

13.2. Parsowanie XML

Oto prosta aplikacja, która przetwarza część kodu XML i wyodrębnia z niego niektóre elementy danych:

```
import xml.etree.ElementTree as ET

data = '''
```



Rysunek 13.1. Drzewiasta reprezentacja XMLa

```
<osoba>
  <imie>Chuck</imie>
  <telefon typ="miedzynar">
    +1 734 303 4456
  </telefon>
  <email ukryty="tak" />
</osoba>'''

tree = ET.fromstring(data)
print('Imię:', tree.find('imie').text)
print('Attr:', tree.find('email').get('ukryty'))

# Kod źródłowy: https://py4e.pl/code3/xml1.py
```

Potrójny apostrof (`'''`), jak również potrójny cudzysłów (`"""`) pozwalają na tworzenie napisów, które obejmują wiele linii.

Wywołanie `fromstring()` zamienia reprezentację kodu XML z napisu na „drzewo" elementów XML. Gdy element XML znajduje się w drzewie, mamy wiele metod, które możemy wywołać po to, by wyodrębnić interesujące nas dane z tekstu w kodzie XML. Funkcja `find()` przeszukuje drzewo XML i wyciąga element, który pasuje do określonego znacznika.

```
Imię: Chuck
Attr: tak
```

Użycie parsera XML takiego jak `ElementTree` ma tę zaletę, że choć XML w tym przykładzie jest dość prosty, to okazuje się, że istnieje wiele reguł dotyczących poprawności składni XML, a użycie `ElementTree` pozwala nam na wydobycie danych z XML bez przejmowania się tymi regułami.

13.3. Przechodzenie w pętli po węzłach

Często XML ma wiele węzłów i żeby je wszystkie przetworzyć musimy napisać pętlę. Przyjmijmy, że mamy do przeanalizowania prostą bazę użytkowników, która jest zapisana w postaci dokumentu XML. W poniższym programie przechodzimy w pętli przez wszystkie elementy o nazwie `user`:

```
import xml.etree.ElementTree as ET

input = '''
<stuff>
  <users>
    <user x="2">
      <id>001</id>
```

```
      <name>Chuck</name>
    </user>
    <user x="7">
      <id>009</id>
      <name>Brent</name>
    </user>
  </users>
</stuff>'''

stuff = ET.fromstring(input)
lst = stuff.findall('users/user')
print('Liczba użytkowników:', len(lst))

for item in lst:
    print('Name', item.find('name').text)
    print('Id', item.find('id').text)
    print('Attribute', item.get('x'))

# Kod źródłowy: https://py4e.pl/code3/xml2.py
```

Metoda `findall()` zwraca Pythonową listę poddrzew, które reprezentują struktury `user` w drzewie XML. Następnie możemy napisać pętlę `for`, która analizuje każdy węzeł o nazwie `user` i wypisuje jego elementy tekstowe `name` i `id` oraz atrybut `x`.

```
Liczba użytkowników: 2
Name Chuck
Id 001
Attribute 2
Name Brent
Id 009
Attribute 7
```

W funkcji `findall()` należy uwzględnić wszystkie elementy poziomu nadrzędnego z wyjątkiem elementu najwyższego poziomu (np. `users/user`). W przeciwnym razie Python nie znajdzie żadnych szukanych przez nas węzłów.

```
import xml.etree.ElementTree as ET

input = '''
<stuff>
  <users>
    <user x="2">
      <id>001</id>
      <name>Chuck</name>
    </user>
    <user x="7">
      <id>009</id>
      <name>Brent</name>
    </user>
  </users>
</stuff>'''

stuff = ET.fromstring(input)

lst = stuff.findall('users/user')
print('Liczba użytkowników:', len(lst))

lst2 = stuff.findall('user')
print('Liczba użytkowników:', len(lst2))
```

Zmienna `lst` przechowuje wszystkie elementy `user`, które są zagnieżdżone w ich rodzicu, tj. w `users`. Zmienna `lst2` szuka elementów `user`, które miałyby być zagnieżdżone w obrębie najwyższego poziomu, czyli `staff`, ale tam nie ma ani jednego takiego węzła.

```
Liczba użytkowników: 2
Liczba użytkowników: 0
```

13.4. JSON

Format JSON został zainspirowany obiektowym i tablicowym formatem używanym w języku JavaScript. Ponieważ Python został wymyślony przed JavaScriptem, składnia Pythona dla słowników i list wpłynęła na składnię JSON. Format JSON jest więc prawie identyczny z połączeniem list i słowników Pythona.

Oto przykładowy kod JSON, który w przybliżeniu odpowiada prostemu, widzianemu przez nas wcześniej dokumentowi XML:

```
{
  "imie" : "Chuck",
  "telefon" : {
    "typ" : "miedzynar",
    "numer" : "+1 734 303 4456"
   },
   "email" : {
     "ukryty" : "tak"
   }
}
```

Zapewne zauważysz kilka różnic. Przede wszystkim w kodzie XML możemy dodać do znacznika `telefon` atrybuty takie jak `miedzynar`. W JSON mamy po prostu pary klucz-wartość. Znacznik XML `osoba` również zniknął i został zastąpiony zestawem zewnętrznych nawiasów klamrowych.

Ogólnie rzecz biorąc, struktury JSON są prostsze niż XML, ponieważ JSON ma mniej możliwości niż XML. Ale ma tę zaletę, że mapuje *bezpośrednio* do jakiegoś połączenia słowników i list. A ponieważ prawie wszystkie języki programowania mają coś w rodzaju słowników i list Pythona, JSON jest bardzo naturalnym formatem wymiany danych pomiędzy dwoma współpracującymi ze sobą programami.

JSON szybko staje się najczęściej wybieranym formatem dla prawie wszystkich danych wymienianych między aplikacjami, właśnie ze względu na swoją relatywną prostotę w porównaniu do kodu XML.

13.5. Parsowanie JSONa

Budujemy nasz kod JSON poprzez dowolne zagnieżdżanie słowników i list. W poniższym przykładzie przedstawiamy listę użytkowników, gdzie każdy użytkownik jest zestawem par klucz-wartość (czyli słownikiem). Mamy więc listę słowników.

W poniższym programie wykorzystujemy wbudowany w Pythona moduł `json` do parsowania kodu JSON i odczytywania z niego danych. Porównaj kod z wcześniejszym przykładem pracującym na kodzie XML. JSON ma mniej detali, więc musimy z góry wiedzieć, że otrzymujemy listę, i że lista ta składa się z użytkowników, a każdy użytkownik jest zestawem par klucz-wartość. JSON jest bardziej zwięzły (zaleta), ale mniej samoopisowy (wada).

```
import json

data = '''
[
  { "id" : "001",
```

```
    "x" : "2",
    "name" : "Chuck"
  } ,
  { "id" : "009",
    "x" : "7",
    "name" : "Brent"
  }
]'''

info = json.loads(data)
print('Liczba użytkowników:', len(info))

for item in info:
    print('Name', item['name'])
    print('Id', item['id'])
    print('Attribute', item['x'])

# Kod źródłowy: https://py4e.pl/code3/json2.py
```

Jeśli porównasz kod wyodrębniający dane z kodu JSON i kodu XML, to zauważysz w tym przykładzie, że to, co otrzymujemy z `json.loads()`, jest listą Pythona, po której przechodzimy pętlą `for`, a każda pozycja na tej liście jest słownikiem Pythona. Kiedy już JSON zostanie przeparsowany, możemy użyć operatora indeksu, tak by wyodrębnić różne fragmenty danych związane z użytkownikiem. Nie musimy korzystać z biblioteki JSON, by przekopać się przez parsowany kod JSON, ponieważ zwracane dane są po prostu natywnymi strukturami Pythona.

Wynik powyższego programu jest dokładnie taki sam jak wersji z dokumentem XML.

```
Liczba użytkowników: 2
Name Chuck
Id 001
Attribute 2
Name Brent
Id 009
Attribute 7
```

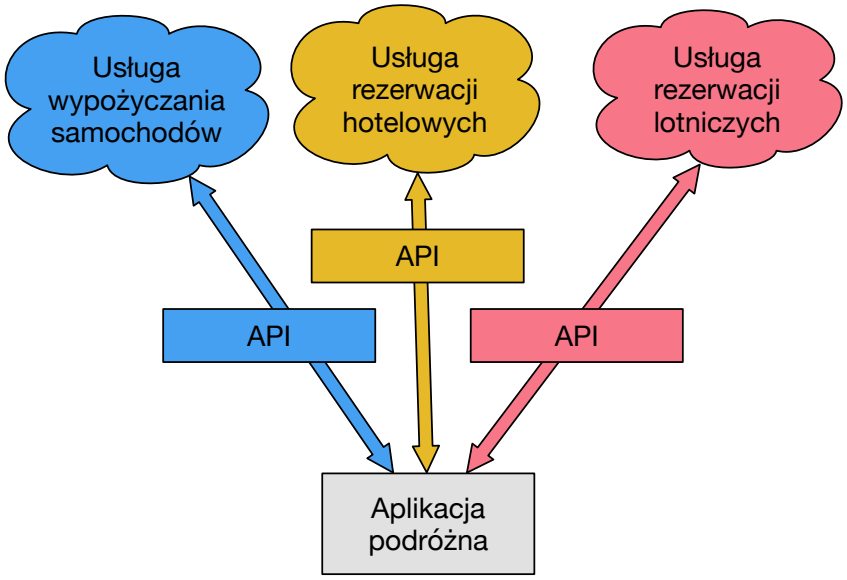
Ogólnie rzecz biorąc, w obrębie usług sieciowych istnieje branżowy trend polegający na odchodzeniu od dokumentów XML i zmierzaniu w kierunku kodu JSON. Jest on prostszy i bardziej bezpośrednio mapuje do natywnych struktur danych, które mamy już w językach programowania, więc kod parsujący JSON i wyodrębniający dane jest w takim przypadku zazwyczaj prostszy i bardziej czytelny. Z drugiej strony XML jest bardziej samoopisowy niż JSON, a więc są pewne zastosowania, w których to XML ma przewagę. Na przykład większość edytorów tekstu przechowuje dokumenty wewnętrznie przy użyciu kodu XML, a nie JSON.

13.6. API – Interfejsy programowania aplikacji

Mamy teraz możliwość wymiany danych pomiędzy aplikacjami przy użyciu protokołu HTTP oraz sposób na reprezentowanie skomplikowanych danych, które wysyłamy tam i z powrotem między aplikacjami przy użyciu kodu XML lub JSON.

Następnym krokiem jest rozpoczęcie definiowania i dokumentowania „umów" pomiędzy aplikacjami wykorzystującymi powyższe techniki. Ogólna nazwa takiej umowy to *interfejsy programowania aplikacji* (ang. *Application Program Interfaces*, API). Kiedy używamy API, zazwyczaj jeden z programów udostępnia zestaw *usług* do wykorzystania przez inne aplikacje i publikuje API (tzn. „zasady"), których należy przestrzegać, by uzyskać dostęp do tych usług.

Gdy zaczynamy budować nasze programy, których działanie obejmuje dostęp do usług świadczonych przez inne programy, nazywamy to podejście *architekturą usługową* (ang. *Service-Oriented Architecture*, SOA). Podejście SOA to takie, w którym cała nasza aplikacja korzysta z usług innych aplikacji. Podejście inne niż SOA to



Rysunek 13.2. SOA – architektura usługowa

takie, w którym aplikacja jest pojedyncza i samodzielna – zawiera cały kod niezbędny do jej implementacji i wdrożenia.

Podczas korzystania z sieci widzimy wiele przykładów SOA. Możemy wejść na stronę internetową i zarezerwować podróże lotnicze, hotele i wypożyczyć samochody, a wszystko to z poziomu jednej strony. Dane dotyczące hoteli nie są przechowywane na serwerach linii lotniczych. Zamiast tego serwery linii lotniczych kontaktują się z usługami znajdującymi się na serwerach rezerwacji hotelowych, pobierają dane dotyczące hoteli i przedstawiają je użytkownikowi. Kiedy użytkownik wyraża zgodę na dokonanie rezerwacji hotelowej za pomocą strony internetowej linii lotniczych, strona ta korzysta z innej usługi internetowej w systemach hotelowych, tak aby faktycznie dokonać tej rezerwacji. A gdy przychodzi czas, by obciążyć kartę płatniczą za całą transakcję, w proces ten włączają się jeszcze inne serwery.

Architektura usługowa ma wiele zalet, w tym: (1) zawsze przechowujemy tylko jedną kopię danych (jest to szczególnie ważne w przypadku rezerwacji hotelowych, w których nie chcemy wykonywać nadmiernych i niewykonalnych zobowiązań) oraz (2) właściciele danych mogą ustalić zasady korzystania z nich. Z tego powodu system SOA musi być starannie zaprojektowany, tak aby działał wydajnie i spełniał potrzeby użytkownika.

Kiedy aplikacja udostępnia przez internet zestaw usług w swoim API, nazywamy to *usługą sieciową* (ang. *web service*).

13.7. Bezpieczeństwo i korzystanie z API

Dość często zdarza się, że chcąc skorzystać z API jakiegoś dostawcy, musisz posiadać tzw. klucz API. Ogólna idea jest taka, że dostawcy chcą wiedzieć, kto korzysta z ich usług i w jakim stopniu ich używa. Być może oferują darmowe i płatne wersje swoich usług lub mają politykę ograniczającą liczbę zapytań, które pojedyncza osoba może wysłać w danym odcinku czasu.

Czasami jest tak, że po otrzymaniu klucza API po prostu dołączasz go jako część danych w zapytaniu POST lub jako parametr w adresie URL podczas wywoływania API.

Inni dostawcy usług chcą mieć większą pewność co do źródła zapytań i z tego powodu oczekują, że będziesz wysyłał kryptograficznie podpisane wiadomości z wykorzystaniem tzw. współdzielonego klucza i sekretu. Bardzo popularną technologią używaną do podpisywania tego rodzaju zapytań jest *OAuth*. Więcej o tym protokole możesz przeczytać na stronie https://www.oauth.net.

Na szczęście istnieje wiele wygodnych i darmowych bibliotek OAuth, dzięki czemu możesz uniknąć implementowania OAuth od zera. Biblioteki mają różny stopień skomplikowania i możliwości, a informacje o nich znajdują się na stronie internetowej OAuth.

13.8. Słowniczek

API Interfejs programowania aplikacji. Umowa pomiędzy aplikacjami, która określa schematy interakcji pomiędzy dwoma komponentami tych aplikacji.
ElementTree Wbudowana biblioteka Pythona służąca do parsowania danych XML.
JSON Notacja obiektowa JavaScript. Format, który pozwala na oznakowanie ustrukturyzowanych danych w oparciu o składnię obiektów JavaScript.
SOA Architektura usługowa. Sytuacja, w której aplikacja składa się z połączonych w sieci komponentów.
XML Rozszerzalny język znaczników. Format, który pozwala na oznakowanie danych strukturalnych.

13.9. Aplikacja nr 1: usługa sieciowa OpenStreetMap do geokodowania

OpenStreetMap oferuje usługę internetową Nominatim, która pozwala nam korzystać z ich dużej bazy danych geograficznych. Możemy przesłać do ich geokodującego API tekst opisujący jakieś miejsce geograficzne, np. „Ann Arbor, MI", i sprawić, że OpenStreetMap zwróci najbardziej prawdopodobne przypuszczenie dotyczące tego, gdzie na mapie możemy odnaleźć wprowadzoną przez nas lokalizację.

Usługa geokodowania jest bezpłatna, ale ograniczona ilościowo, więc w aplikacji komercyjnej nie możesz korzystać z API w sposób nieograniczony. Ale jeśli np. masz trochę danych z ankiety, w której użytkownicy wprowadzili w polu danych jakąś lokalizację w dowolnym formacie, to możesz użyć tego API, tak by całkiem nieźle oczyścić swoje dane.

Używając bezpłatnego API, takiego jak na przykład API geokodowania OpenStreetMap, korzystaj z zasobów z poszanowaniem zasad. Jeśli zbyt wiele osób będzie nadużywać takiej usługi, to jej dostawca może zrezygnować z udostępniania bezpłatnej wersji lub znacznie ją ograniczyć.

Możesz zapoznać się z dokumentacją online dotyczącą tej usługi[1], ale jest ona na tyle prosta, że możesz ją przetestować nawet za pomocą przeglądarki, wpisując następujący adres URL:

https://nominatim.openstreetmap.org/search.php?q=Ann%20Arbor%2C%20MI&format=geojson&limit=1

Usługa zwraca nam dane w formacie GeoJSON. Jest to otwarty standard formatu danych przeznaczony do przedstawiania prostych obiektów geograficznych wraz z ich atrybutami nieprzestrzennymi i bazuje on na formacie JSON.

Poniżej znajduje się prosta aplikacja wyszukująca informacje geograficzne na temat wprowadzonej przez użytkownika nazwy miejsca. Aplikacja wywołuje API geokodowania OpenStreetMap i pobiera informacje ze zwróconego kodu JSON.

```
import urllib.request
import json
import ssl

serviceurl = 'https://nominatim.openstreetmap.org/search.php?'

# Ignoruj błędy związane z certyfikatami SSL
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

while True:
    address = input('Podaj nazwę miejsca: ')
    if len(address) < 1: break

    parms = dict()
    parms['q'] = address
    parms['format'] = 'geojson'
```

[1] https://nominatim.org/release-docs/develop/api/Search/

```
    parms['limit'] = 1

    url = serviceurl + urllib.parse.urlencode(parms)

    print('Pobieranie', url)
    uh = urllib.request.urlopen(url, context=ctx)
    data = uh.read().decode()
    print('Pobrano:', len(data))

    try:
        js = json.loads(data)
    except:
        js = None

    if not js or 'features' not in js or len(js['features']) == 0:
        print('==== Błąd pobierania ====')
        print(data)
        continue

    print(json.dumps(js, indent=4))

    lng = js['features'][0]['geometry']['coordinates'][0]
    lat = js['features'][0]['geometry']['coordinates'][1]
    print('szer. geogr.', lat, 'dł. geogr.', lng)
    location = js['features'][0]['properties']['display_name']
    print(location)

# Kod źródłowy: https://py4e.pl/code3/geojson.py
```

Program odczytuje tekst podany przez użytkownika i konstruuje adres URL zawierający wprowadzoną lokalizację do wyszukania jako prawidłowo zakodowany parametr adresu URL, a następnie używa `urllib` do pobrania tekstu z API geokodowania OpenStreetMap. W przeciwieństwie do stałej/niezmiennej strony internetowej, otrzymywane dane zależą od parametrów, które wysyłamy, oraz od danych geograficznych przechowywanych na serwerach OpenStreetMap.

Kiedy pobierzemy dane JSON, parsujemy je za pomocą modułu `json` i sprawdzamy kilka rzeczy, tak aby upewnić się, że otrzymaliśmy dobre dane, a następnie pobieramy informacje, których szukamy.

Wynik działania programu jest następujący (część zwróconych danych JSON została usunięta):

```
$ python3 geojson.py
Podaj nazwę miejsca: Ann Arbor, MI
Pobieranie https://nominatim.openstreetmap.org/search.php?q=...
Pobrano: 594
```

```
{
    "type": "FeatureCollection",
    "licence": "Data \u00a9 OpenStreetMap contributors, ODbL 1.0. ...",
    "features": [
        {
            "type": "Feature",
            "properties": {
                "place_id": 255697953,
                "osm_type": "relation",
                "osm_id": 135130,
                "display_name": "Ann Arbor, Washtenaw County, ...",
                "place_rank": 16,
                "category": "boundary",
                "type": "administrative",
                "importance": 0.837069344370284,
                "icon": "https://nominatim.openstreetmap.org/..."
```

```
            },
            "bbox": [
                -83.799572,
                42.222668,
                -83.675807,
                42.3238941
            ],
            "geometry": {
                "type": "Point",
                "coordinates": [
                    -83.7312291,
                    42.2681569
                ]
            }
        }
    ]
}
```

```
szer. geogr. 42.2681569 dł. geogr. -83.7312291
Ann Arbor, Washtenaw County, Michigan, 48104, United States of America
Podaj nazwę miejsca:
```

Możesz pobrać https://py4e.pl/code3/geoxml.py, aby sprawdzić wariant XML w API geokodowania OpenStreetMap.

Ćwiczenie 1. Zmień https://py4e.pl/code3/geojson.py lub https://py4e.pl/code3/geoxml.py tak, żeby wypisać nazwę stanu z pobranych danych (jeżeli podane miejsce jest w USA). Dodaj parametr zapytania `parms['accept-language'] = 'pl'`, tak aby zwracane wyniki o lokalizacji były w języku polskim. Dodaj sprawdzanie błędów, aby Twój program nie wyświetlał danych z mechanizmu *traceback*, gdy w danych nie ma nazwy stanu. Przetestuj działanie na „Ann Arbor, MI", „Washington, WI" oraz „Warszawa". Gdy kod już zacznie działać, wyszukaj „Ocean Atlantycki" oraz „Atlantic Ocean" i upewnij się, że program obsługuje też lokalizacje, które nie znajdują się w żadnym kraju.

13.10. Aplikacja nr 2: Twitter

Od czasu, gdy API Twittera stało się bardziej wartościowe, Twitter przeszedł z otwartego i publicznego API do takiego, które przy każdym zapytaniu wymaga użycia OAuth.

Oba nasze przykłady opierają się na plikach `twurl.py`, `hidden.py`, `oauth.py`, `twitter1.py` i `twitter2.py`, które możesz pobrać z https://py4e.pl/code3. Po pobraniu plików umieść je wszystkie w jednym katalogu.

Aby móc korzystać z tych programów, musisz mieć konto na Twitterze i autoryzować swój kod Pythona jako aplikację, ustawić klucz, sekret, token i sekret tokena. Wyedytuj plik `hidden.py` i umieść w nim cztery specjalne ciągi znaków w odpowiednich miejscach:

```
# Przechowuj ten plik oddzielnie

# 1. Zaloguj się lub załóż konto na https://twitter.com
# 2. Wejdź na stronę https://developer.twitter.com/en/apps
#    i spróbuj utworzyć nową aplikację ("Create an app"):
#    a) zostaniesz poproszony o założenie konta deweloperskiego
#       (Please apply for a Twitter developer account);
#    b) kliknij na "Apply"
#    c) wybierz "Hobbyist" > "Exploring the API" i wybierz "Next"
#    d) uzupełnij formularz (może być wymagana weryfikacja
#       numeru telefonu)
#    e) w kolejnym kroku opisz po angielsku, jak zamierzasz
```

```
#        wykorzystać dane Twittera (wspomnij, że uczysz się
#        Pythona, chcesz dowiedzieć się, jak korzystać z API
#        Twittera, zamierzasz napisać dwa mini-projekty, które
#        w interaktywny sposób pobierają nazwę konta Twittera
#        i dla podanego konta w projekcie nr 1 pobierzesz i
#        wyświetlisz oś czasu użytkownika zwróconą w formacie
#        JSON, a w projekcie nr 2 pobierzesz listę jego znajomych
#        w formacie JSON i przeparsujesz ją w celu wyciągnięcia
#        informacji o znajomych; opisz tę sekcję dość szczegółowo,
#        gdyż wpływa ona na akceptację Twojego zgłoszenia);
#        w sekcji "Specifics" zaznacz wszystko na "No"
#    f) zweryfikuj zgłoszenie, przeczytaj warunki umowy
#        i wyślij zgłoszenie
#    g) potwierdź zgłoszenie, klikając na link w wiadomości
#        mailowej, którą otrzymasz po wysłaniu zgłoszenia
# 3. Gdy Twoje zgłoszenie zostanie zaakceptowane, wejdź jeszcze
#    raz na https://developer.twitter.com/en/apps i ponownie
#    spróbuj utworzyć nową aplikację ("Standalone Apps" >
#    "Create App")
# 4. Utwórz aplikację, w karcie "Keys and tokens" wygeneruj
#    klucze  i podmień poniższe cztery ciągi znaków dotyczące
#    consumer_key, consumer_secret, token_key i token_secret
#    (podmień wszystko, co znajduje się między cudzysłowami,
#    łącznie z nawiasami ostrokątnymi)

def oauth():
    return {# Consumer Keys
            "consumer_key":    "<tutaj umieść API key>",
            "consumer_secret": "<tutaj umieść API key secret>",
            # Authentication Tokens - Access Token & Secret
            "token_key":       "<tutaj umieść Access token>",
            "token_secret":    "<tutaj umieść Access token secret>"}

# Kod źródłowy: https://py4e.pl/code3/hidden.py
```

Usługa sieciowa Twittera jest dostępna za pomocą podobnego do poniższego adresu URL:

https://api.twitter.com/1.1/statuses/user_timeline.json

Jednak po dodaniu wszystkich informacji dotyczących uwierzytelniania adres URL będzie wyglądał mniej więcej tak:

```
https://api.twitter.com/1.1/statuses/user_timeline.json?count=2
&oauth_version=1.0&oauth_token=101...SGI&screen_name=drchuck
&oauth_nonce=09239679&oauth_timestamp=1380395644
&oauth_signature=rLK...BoD&oauth_consumer_key=h7Lu...GNg
&oauth_signature_method=HMAC-SHA1
```

Jeśli chcesz dowiedzieć więcej o znaczeniu różnych parametrów, które są dodawane w celu spełnienia wymogów bezpieczeństwa OAuth, to możesz przeczytać specyfikację OAuth.

W naszych programach korzystających z Twittera wszystkie skomplikowane mechanizmy będą ukryte w plikach `oauth.py` i `twurl.py`. Po prostu ustawiamy sekrety w `hidden.py`, następnie wysyłamy żądany adres URL do funkcji `twurl.augment()`, a kod biblioteki dodaje za nas wszystkie niezbędne parametry do adresu URL.

Poniższy program pobiera oś czasu dla konkretnego użytkownika Twittera i zwraca go nam w formacie JSON w postaci tekstu. Po uzyskaniu danych, po prostu wypisujemy pierwsze 250 znaków:

```
import urllib.request
import twurl
import ssl
```

```
# https://developer.twitter.com/en/apps
# Utwórz aplikację i wstaw w hidden.py cztery ciągi znaków dotyczące OAuth

TWITTER_URL = 'https://api.twitter.com/1.1/statuses/user_timeline.json'

# Ignoruj błędy związane z certyfikatami SSL
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

while True:
    print('')
    acct = input('Podaj nazwę konta na Twitterze: ')
    if (len(acct) < 1): break
    url = twurl.augment(TWITTER_URL,
                        {'screen_name': acct, 'count': '2'})
    print('Pobieranie', url)
    connection = urllib.request.urlopen(url, context=ctx)
    data = connection.read().decode()
    print(data[:250])
    headers = dict(connection.getheaders())
    # wypisz nagłówki
    print('Pozostało', headers['x-rate-limit-remaining'])

# Kod źródłowy: https://py4e.pl/code3/twitter1.py
```

Po uruchomieniu programu uzyskamy mniej więcej następujący wynik:

```
Podaj nazwę konta na Twitterze: drchuck
Pobieranie https://api.twitter.com/1.1/ ...
[{"created_at":"Sat Sep 28 17:30:25 +0000 2013","
id":384007200990982144,"id_str":"384007200990982144",
"text":"RT @fixpert: See how the Dutch handle traffic
intersections: http:\/\/t.co\/tIiVWtEhj4\n#brilliant",
"source":"web","truncated":false,"in_rep
Pozostało 178

Podaj nazwę konta na Twitterze: fixpert
Pobieranie https://api.twitter.com/1.1/ ...
[{"created_at":"Sat Sep 28 18:03:56 +0000 2013",
"id":384015634108919808,"id_str":"384015634108919808",
"text":"3 months after my freak bocce ball accident,
my wedding ring fits again! :)\n\nhttps:\/\/t.co\/2XmHPx7kgX",
"source":"web","truncated":false,
Pozostało 177

Podaj nazwę konta na Twitterze:
```

Wraz z danymi zwróconymi z osi czasu, Twitter zwraca również w nagłówkach odpowiedzi HTTP metadane dotyczące żądania. Nagłówek `x-rate-limit-remaining` informuje nas o tym, ile jeszcze żądań możemy wysłać, zanim zostaniemy zablokowani na pewien krótki czas. Jak widać, nasze kolejne wywołania API powodują zmniejszenie tej wartości o jeden.

W poniższym przykładzie pobieramy znajomych użytkownika na Twitterze, analizujemy zwrócony JSON i wyodrębniamy niektóre informacje o znajomych. Po przeparsowaniu kodu JSON do postaci list i słowników, ponownie zamieniamy go do zwykłej postaci tekstowej, ale tym razem nadajemy mu po dwie spacje na wcięcie i „ładnie" wypisujemy na ekranie, tak aby umożliwić nam dalsze zgłębianie otrzymanych danych (co może być przydatne, jeśli będziemy chcieli wyodrębnić więcej pól).

```
import urllib.request
import twurl
```

```
import json
import ssl

# https://developer.twitter.com/en/apps
# Utwórz aplikację i wstaw w hidden.py cztery ciągi znaków dotyczące OAuth

TWITTER_URL = 'https://api.twitter.com/1.1/friends/list.json'

# Ignoruj błędy związane z certyfikatami SSL
ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

while True:
    print('')
    acct = input('Podaj nazwę konta na Twitterze: ')
    if (len(acct) < 1): break
    url = twurl.augment(TWITTER_URL,
                        {'screen_name': acct, 'count': '5'})
    print('Pobieranie', url)
    connection = urllib.request.urlopen(url, context=ctx)
    data = connection.read().decode()

    js = json.loads(data)
    print(json.dumps(js, indent=2))

    headers = dict(connection.getheaders())
    print('Pozostało', headers['x-rate-limit-remaining'])

    for u in js['users']:
        print(u['screen_name'])
        if 'status' not in u:
            print('   * Nie odnaleziono klucza "status"')
            continue
        s = u['status']['text']
        print('  ', s[:50])

# Kod źródłowy: https://py4e.pl/code3/twitter2.py
```

Ponieważ JSON staje się zestawem zagnieżdżonych list Pythona i słowników, to przy pomocy niewielkiej ilości kodu możemy użyć kombinacji operacji indeksowania i pętli `for` do przejścia po zwróconych strukturach danych.

Wynik programu wygląda następująco (niektóre elementy danych zostały skrócone, tak aby zmieścić wszystko na stronie):

```
Podaj nazwę konta na Twitterze: drchuck
Pobieranie https://api.twitter.com/1.1/friends ...
Pozostało 14
```

```
{
  "next_cursor": 1444171224491980205,
  "users": [
    {
      "id": 662433,
      "followers_count": 28725,
      "status": {
        "text": "@jazzychad I just bought one .__.",
        "created_at": "Fri Sep 20 08:36:34 +0000 2013",
        "retweeted": false,
      },
```

```
      "location": "San Francisco, California",
      "screen_name": "leahculver",
      "name": "Leah Culver",
    },
    {
      "id": 40426722,
      "followers_count": 2635,
      "status": {
        "text": "RT @WSJ: Big employers like Google ...",
        "created_at": "Sat Sep 28 19:36:37 +0000 2013",
      },
      "location": "Victoria Canada",
      "screen_name": "_valeriei",
      "name": "Valerie Irvine",
    }
  ],
 "next_cursor_str": "1444171224491980205"
}
```

```
leahculver
   @jazzychad I just bought one .__.
_valeriei
   RT @WSJ: Big employers like Google, AT&amp;amp;T are h
ericbollens
   RT @lukew: sneak peek: my LONG take on the good &a
halherzog
   Learning Objects is 10. We had a cake with the LO,
scweeker
   @DeviceLabDC love it! Now where so I get that "etc

Podaj nazwę konta na Twitterze:
```

W ostatnim fragmencie wyniku programu widzimy pętlę `for`, która odczytuje pięciu ostatnich „znajomych" konta *@drchuck* na Twitterze i wypisuje najnowszy status dla każdego z nich. W zwróconym kodzie JSON znajduje się dużo więcej danych. Jeśli spojrzysz na wynik programu, to zobaczysz również, że operacja „znalezienia znajomych" danego konta ma inny limit wywołań niż dozwolona liczba zapytań dotycząca osi czasu.

Korzystanie z bezpiecznych kluczy API daje Twitterowi pewność, kto i w jakim zakresie korzysta z jego API i danych. Podejście oparte na ustanawianiu limitów zapytań pozwala na proste wyszukiwanie danych osobowych, ale nie na zbudowanie produktu, który pobierałby przez API Twittera jakieś dane miliony razy dziennie.