Accessing Databases via Web APIs: Lecture Code

In [2]:
```
# Import required libraries
import requests
# from urllib3 import quote_plus
import json
from __future__ import division
import math
```

1. Constructing API GET Request

In the first place, we know that every call will require us to provide a) a base URL for the API, b) some authorization code or key, and c) a format for the response. So let's put store those in some variables.

In [3]:
```
# set key
key="6e23901ee0fc07f0f6cee3a45b566bc5:13:73313103"

# set base url
base_url="http://api.nytimes.com/svc/search/v2/articlesearch"

# set response format
response_format=".json"
```

You often want to send some sort of data in the URL’s query string. This data tells the API what information you want. In our case, we want articles about Duke Ellington. Requests allows you to provide these arguments as a dictionary, using the `params` keyword argument. In addition to the search term `q`, we have to put in the `api-key` term.

In [4]:
```
# set search parameters
search_params = {"q":"Duke Ellington",
                 "api-key":key}
```

Now we're ready to make the request. We use the `.get` method from the `requests` library to make an HTTP GET Request.

```
In [5]: # make request
        r = requests.get(base_url+response_format, params=search_params)
```

Now, we have a response (http://docs.python-requests.org/en/latest/api/#requests.Response) object called `r`. We can get all the information we need from this object. For instance, we can see that the URL has been correctly encoded by printing the URL. Click on the link to see what happens.

```
In [6]: print(r.url)

        http://api.nytimes.com/svc/search/v2/articlesearch.json?api-key=6e23901
        ee0fc07f0f6cee3a45b566bc5%3A13%3A73313103&q=Duke+Ellington
```

Click on that link to see it returns!

Challenge 1: Adding a date range

What if we only want to search within a particular date range? The NYT Article Api allows us to specify start and end dates.

Alter the `search_params` code above so that the request only searches for articles in the year 2005.

You're gonna need to look at the documentation here (http://developer.nytimes.com/docs/read/article_search_api_v2) to see how to do this.

```
In [7]: #YOUR CODE HERE



        search_params = {"q":"Duke Ellington",
                          "api-key":key,
                         "begin_date": 20050101,
                         "end_date": 20051231,
                         }
        # Uncomment to test
        r = requests.get(base_url+response_format, params=search_params)
        print(r.url)

        http://api.nytimes.com/svc/search/v2/articlesearch.json?end_date=200512
        31&api-key=6e23901ee0fc07f0f6cee3a45b566bc5%3A13%3A73313103&q=Duke+Elli
        ngton&begin_date=20050101
```

Challenge 2: Specifying a results page

The above will return the first 10 results. To get the next ten, you need to add a "page" parameter. Change the search parameters above to get the second 10 resuls.

```
In [8]: #YOUR CODE HERE

        # Uncomment to test
        # r = requests.get(base_url+response_format, params=search_params)
        # r.url
```

2. Parsing the response text

We can read the content of the server's response using `.text`

```
In [9]: # Inspect the content of the response, parsing the result as text
        response_text= r.text
        print(response_text[:1000])
```

```
{"response":{"meta":{"hits":77,"time":36,"offset":0},"docs":[{"web_ur
l":"http:\/\/www.nytimes.com\/2005\/10\/02\/nyregion\/02bookshelf.htm
l","snippet":"A WIDOW'S WALK:.","lead_paragraph":"A WIDOW'S WALK: A Mem
oir of 9\/11 By Marian Fontana Simon & Schuster ($24, hardcover) Theres
a and I walk into the Blue Ribbon, an expensive, trendy restaurant on F
ifth Avenue in Park Slope. We sit at a banquette in the middle of the r
oom and read the eclectic menu, my eyes instinctively scanning the pric
es for the least expensive item.","abstract":null,"print_page":"9","blo
g":[],"source":"The New York Times","multimedia":[],"headline":{"mai
n":"NEW YORK BOOKSHELF\/NONFICTION","kicker":"New York Bookshelf"},"key
words":[{"name":"persons","value":"ELLINGTON, DUKE"},{"name":"person
s","value":"HARRIS, DANIEL"}],"pub_date":"2005-10-02T00:00:00Z","docume
nt_type":"article","news_desk":"The City Weekly Desk","section_name":"N
ew York and Region","subsection_name":null,"byline":{"person":[{"firstn
ame":"N.","middl
```

What you see here is JSON text, encoded as unicode text. JSON stands for "Javascript object notation." It has a very similar structure to a python dictionary -- both are built on key/value pairs. This makes it easy to convert JSON response to a python dictionary.

```
In [10]: # Convert JSON response to a dictionary
         data=json.loads(response_text)
         # data
```

That looks intimidating! But it's really just a big dictionary. Let's see what keys we got in there.

```
In [17]: #data
In [18]: data.keys()

Out[18]: dict_keys(['response', 'copyright', 'status'])
```

```
In [19]: # this is boring
         data['status']

Out[19]: 'OK'
```

```
In [20]: # so is this
         data['copyright']

Out[20]: 'Copyright (c) 2013 The New York Times Company.  All Rights Reserved.'
```

```
In [21]: # this is what we want!
         #data['response']
```

```
In [22]: data['response'].keys()

Out[22]: dict_keys(['meta', 'docs'])
```

```
In [23]: data['response']['meta']

Out[23]: {'hits': 77, 'offset': 0, 'time': 36}
```

```
In [24]: # data['response']['docs']
         type(data['response']['docs'])

Out[24]: list
```

That looks what we want! Let's put that in it's own variable.

```
In [25]: docs = data['response']['docs']
In [26]: docs[0]
```

```
Out[26]: {'_id': '4fd2872b8eb7c8105d858553',
          'abstract': None,
          'blog': [],
          'byline': {'original': 'By N.R. Kleinfield',
           'person': [{'firstname': 'N.',
             'lastname': 'Kleinfield',
             'middlename': 'R.',
             'organization': '',
             'rank': 1,
             'role': 'reported'}]},
          'document_type': 'article',
          'headline': {'kicker': 'New York Bookshelf',
           'main': 'NEW YORK BOOKSHELF/NONFICTION'},
          'keywords': [{'name': 'persons', 'value': 'ELLINGTON, DUKE'},
           {'name': 'persons', 'value': 'HARRIS, DANIEL'}],
          'lead_paragraph': "A WIDOW'S WALK: A Memoir of 9/11 By Marian Fontana
         Simon & Schuster ($24, hardcover) Theresa and I walk into the Blue Ribb
         on, an expensive, trendy restaurant on Fifth Avenue in Park Slope. We s
         it at a banquette in the middle of the room and read the eclectic menu,
         my eyes instinctively scanning the prices for the least expensive ite
         m.",
          'multimedia': [],
          'news_desk': 'The City Weekly Desk',
          'print_page': '9',
          'pub_date': '2005-10-02T00:00:00Z',
          'section_name': 'New York and Region',
          'slideshow_credits': None,
          'snippet': "A WIDOW'S WALK:.",
          'source': 'The New York Times',
          'subsection_name': None,
          'type_of_material': 'News',
          'web_url': 'http://www.nytimes.com/2005/10/02/nyregion/02bookshelf.htm
         l',
          'word_count': 629}
```

3. Putting everything together to get all the articles.

That's great. But we only have 10 items. The original response said we had 171 hits! Which means we have to make 171 /10, or 18 requests to get them all. Sounds like a job for a loop!

But first, let's review what we've done so far.

In [27]:
```
# set key
key="6e23901ee0fc07f0f6cee3a45b566bc5:13:73313103"
# set base url
base_url="http://api.nytimes.com/svc/search/v2/articlesearch"
# set response format
response_format=".json"
# set search parameters
search_params = {"q":"Duke Ellington",
                 "api-key":key,
                 "begin_date":"20050101", # date must be in YYYYMMDD for
mat
                 "end_date":"20051231"}
# make request
r = requests.get(base_url+response_format, params=search_params)
# convert to a dictionary
data=json.loads(r.text)
# get number of hits
hits = data['response']['meta']['hits']
print("number of hits: " + str(hits))
# get number of pages
pages = int(math.ceil(hits/10))
# make an empty list where we'll hold all of our docs for every page
all_docs = []
# now we're ready to loop through the pages
for i in range(pages):
    print("collecting page " + str(i))
    # set the page parameter
    search_params['page'] = i
    # make request
    r = requests.get(base_url+response_format, params=search_params)
    # get text and convert to a dictionary
    data=json.loads(r.text)
    # get just the docs
    docs = data['response']['docs']
    # add those docs to the big list
    all_docs = all_docs + docs
```

```
number of hits: 77
collecting page 0
collecting page 1
collecting page 2
collecting page 3
collecting page 4
collecting page 5
collecting page 6
collecting page 7
```

```
In [28]: len(all_docs)

Out[28]: 77
```

Challenge 3: Make a function

Turn the code above into a function that inputs a search term, and returns all the documents containing that search term in 2014.

```
In [29]: #YOUR CODE HERE

def search_nyt_2014(term, year = 2014):
    # set key
    key="6e23901ee0fc07f0f6cee3a45b566bc5:13:73313103"
    # set base url
    base_url="http://api.nytimes.com/svc/search/v2/articlesearch"
    # set response format
    response_format=".json"
    # set search parameters
    search_params = {"q":term,
                     "api-key":key,
                     "begin_date":str(year)+"0101", # date must be in YY
YYMMDD format
                     "end_date":str(year)+"1231"}
    # make request
    r = requests.get(base_url+response_format, params=search_params)
    # convert to a dictionary
    data=json.loads(r.text)
    # get number of hits
    hits = data['response']['meta']['hits']
    print("number of hits: " + str(hits))
    # get number of pages
    pages = int(math.ceil(hits/10))
    # make an empty list where we'll hold all of our docs for every page
    all_docs = []
    # now we're ready to loop through the pages
    for i in range(pages):
        print("collecting page " + str(i))
        # set the page parameter
        search_params['page'] = i
        # make request
        r = requests.get(base_url+response_format, params=search_params)
        # get text and convert to a dictionary
        data=json.loads(r.text)
        # get just the docs
        docs = data['response']['docs']
        # add those docs to the big list
        all_docs = all_docs + docs
    return all_docs
```

```
In [30]: stuff = search_nyt_2014("marco rubio")
```

```
number of hits: 572
collecting page 0
collecting page 1
collecting page 2
collecting page 3
collecting page 4
collecting page 5
collecting page 6
collecting page 7
collecting page 8
collecting page 9
collecting page 10
collecting page 11
collecting page 12
collecting page 13
collecting page 14
collecting page 15
collecting page 16
collecting page 17
collecting page 18
collecting page 19
collecting page 20
collecting page 21
collecting page 22
collecting page 23
collecting page 24
collecting page 25
collecting page 26
collecting page 27
collecting page 28
collecting page 29
collecting page 30
collecting page 31
collecting page 32
collecting page 33
collecting page 34
collecting page 35
collecting page 36
collecting page 37
collecting page 38
collecting page 39
collecting page 40
collecting page 41
collecting page 42
collecting page 43
collecting page 44
collecting page 45
collecting page 46
collecting page 47
collecting page 48
collecting page 49
collecting page 50
collecting page 51
```

```
            collecting page 52
            collecting page 53
            collecting page 54
            collecting page 55
            collecting page 56
            collecting page 57
```

4. Formatting and Exporting

Let's take another look at one of these documents.

In [31]:
```
stuff[0]
```

```
Out[31]:
```

```
{'_id': '53755d9c79881068df7c36a4',
 'abstract': "Paul Krugman Op-Ed column criticizes Republican Sen Marco
Rubio for declaring overwhelming scientific consensus on climate change
to be false; contends Republicans have reached a point where allegiance
to false doctrines has become crucial badge of identity; compares part
y's resistance to science about climate change to its insistence that r
unaway inflation is a problem.",
 'blog': [],
 'byline': {'contributor': '',
  'original': 'By PAUL KRUGMAN',
  'person': [{'firstname': 'Paul',
    'lastname': 'KRUGMAN',
    'organization': '',
    'rank': 1,
    'role': 'reported'}]},
 'document_type': 'article',
 'headline': {'content_kicker': 'Op-Ed Columnist',
  'kicker': 'Op-Ed Columnist',
  'main': 'Points of No Return',
  'print_headline': 'Points of No Return'},
 'keywords': [{'is_major': 'Y',
   'name': 'subject',
   'rank': '1',
   'value': 'Global Warming'},
  {'is_major': 'Y',
   'name': 'subject',
   'rank': '2',
   'value': 'United States Politics and Government'},
  {'is_major': 'N',
   'name': 'organizations',
   'rank': '3',
   'value': 'Republican Party'},
  {'is_major': 'Y', 'name': 'persons', 'rank': '4', 'value': 'Rubio, Ma
rco'},
  {'is_major': 'Y',
   'name': 'subject',
   'rank': '5',
   'value': 'Inflation (Economics)'},
  {'is_major': 'N',
   'name': 'glocations',
   'rank': '6',
   'value': 'United States'}],
 'lead_paragraph': 'False doctrines on climate science have become badg
es of identity for Republicans, and that&#8217;s more frightening than
some of the environmental change underway.',
 'multimedia': [{'height': 126,
   'legacy': {'wide': 'images/2014/10/30/opinion/krugman-new-1114/krugm
an-new-1114-thumbWide.jpg',
    'wideheight': '126',
    'widewidth': '190'},
   'subtype': 'wide',
   'type': 'image',
   'url': 'images/2014/10/30/opinion/krugman-new-1114/krugman-new-1114-
thumbWide.jpg',
   'width': 190},
  {'height': 900,
   'legacy': {'xlarge': 'images/2014/10/30/opinion/krugman-new-1114/kru
gman-new-1114-articleLarge.jpg',
    'xlargeheight': '900',
    'xlargewidth': '600'},
   'subtype': 'xlarge',
   'type': 'image',
   'url': 'images/2014/10/30/opinion/krugman-new-1114/krugman-new-1114-
articleLarge.jpg',
   'width': 600},
  {'height': 75,
   'legacy': {'thumbnail': 'images/2014/10/30/opinion/krugman-new-1114/
krugman-new-1114-thumbStandard.jpg',
    'thumbnailheight': '75',
    'thumbnailwidth': '75'},
   'subtype': 'thumbnail',
   'type': 'image',
   'url': 'images/2014/10/30/opinion/krugman-new-1114/krugman-new-1114-
thumbStandard.jpg',
   'width': 75}],
 'news_desk': 'Editorial',
 'print_page': '27',
 'pub_date': '2014-05-16T00:00:00Z',
 'section_name': 'Opinion',
 'slideshow_credits': None,
 'snippet': 'False doctrines on climate science have become badges of i
dentity for Republicans, and that&#8217;s more frightening than some of
the environmental change underway.',
 'source': 'The New York Times',
 'subsection_name': None,
 'type_of_material': 'Op-Ed',
 'web_url': 'http://www.nytimes.com/2014/05/16/opinion/krugman-points-o
f-no-return.html',
 'word_count': '786'}
```

This is all great, but it's pretty messy. What we'd really like to to have, eventually, is a CSV, with each row representing an article, and each column representing something about that article (header, date, etc). As we saw before, the best way to do this is to make a lsit of dictionaries, with each dictionary representing an article and each dictionary representing a field of metadata from that article (e.g. headline, date, etc.) We can do this with a custom function:

In [32]:
```
def format_articles(unformatted_docs):
    '''
    This function takes in a list of documents returned by the NYT api
    and parses the documents into a list of dictionaries,
    with 'id', 'header', and 'date' keys
    '''
    formatted = []
    for i in unformatted_docs:
        dic = {}
        dic['id'] = i['_id']
        dic['headline'] = i['headline']['main'].encode("utf8")
        dic['date'] = i['pub_date'][0:10] # cutting time of day.
        formatted.append(dic)
    return(formatted)
```

In [33]:
```
all_formatted = format_articles(stuff)
```

In [34]:
```
all_formatted[:5]
```

Out[34]:
```
[{'date': '2014-05-16',
  'headline': b'Points of No Return',
  'id': '53755d9c79881068df7c36a4'},
 {'date': '2014-12-18',
  'headline': b'2014: Rubio Criticizes Obama on Cuba',
  'id': '5493863779881048d26b30e3'},
 {'date': '2014-11-11',
  'headline': b'A Well-Timed Book Tour for Rubio',
  'id': '5462503079881072f4f7304b'},
 {'date': '2014-05-12',
  'headline': b'Rubio on a Presidential Bid, and Climate Change',
  'id': '536fc5c2798810420b81d1ee'},
 {'date': '2014-03-02',
  'headline': b'Rubio Proposes Steps U.S. Should Take With Russia',
  'id': '531288fb79881022a8e2e718'}]
```

Challenge 4: Export the data to a CSV.

In [35]:
```
import csv
stuff = search_nyt_2014("berkeley police california", 2015)
```

```
number of hits: 87
collecting page 0
collecting page 1
collecting page 2
collecting page 3
collecting page 4
collecting page 5
collecting page 6
collecting page 7
collecting page 8
```

In [36]:
```
testfile = open('test.csv','w')
f = csv.writer(testfile)
f.writerow(["date", "headline", "id"])
for x in format_articles(stuff):
    f.writerow([x["date"],x["headline"],x["id"]])
testfile.close()
```