17.7. Extracting from Nested Data

A common problem, especially when dealing with data returned from a web site, is to extract certain elements from deep inside a nested data structure. In principle, there's nothing more difficult about pulling something out from deep inside a nested data structure: with lists, you use [] to index or a for loop to get them them all; with dictionaries, you get the value associated with a particular key using [] or iterate through all the keys, accessing the value for each. But it's easy to get lost in the process and think you've extracted something different than you really have. Because of this, we have created a usable technique to help you during the debugging process.

Follow the system described below and you will have success with extracting nested data. The process involves the following steps:

1. Understand the nested data object.
2. Extract one object at the next level down.
3. Repeat the process with the extracted object

Understand. Extract. Repeat.

To illustrate this, we will walk through extracting information from data formatted in a way that it's return by the Twitter API. This nested dictionary results from querying Twitter, asking for three tweets matching "University of Michigan". As you'll see, it's quite a daunting data structure, even when printed with nice indentation as it's shown below.

Save & Run Original - 1 of 1 Show in CodeLens

```
res = {
  "search_metadata": {
    "count": 3,
    "completed_in": 0.015,
    "max_id_str": "536624519285583872",
    "since_id_str": "0",
    "next_results": "?max_id=536623674942439424&q=University%20of%20Michigan&count
    "refresh_url": "?since_id=536624519285583872&q=University%20of%20Michigan&incl
    "since_id": 0,
    "query": "University+of+Michigan",
    "max_id": 536624519285583872
  },
  "statuses": [
    {
      "contributors": None,
```

Activity: 1 -- ActiveCode (ac300_1_1)

17.7.1. Understand

At any level of the extraction process, the first task is to make sure you understand the current object you have extracted. There are few options here.

1. Print the entire object. If it's small enough, you may be able to make sense of the printout directly. If it's a little bit larger, you may find it helpful to "pretty-print" it, with indentation showing the level of nesting of the data. We don't have a way to pretty-print in our online browser-based environment, but if you're running code with a full Python interpreter, you can use the dumps function in the json module. For example:

```
import json
print(json.dumps(res, indent=2))
```

2. If printing the entire object gives you something that's too unwieldy, you have other options for making sense of it.
    - Copy and paste it to a site like https://jsoneditoronline.org/ which will let you explore and collapse levels
    - Print the type of the object.
    - **If it's a dictionary:**
        - print the keys
    - **If it's a list:**
        - print its length
        - print the type of the first item
        - print the first item if it's of manageable size

Save & Run Original - 1 of 1 Show in CodeLens

```
import json
print(json.dumps(res, indent=2)[:100])
print("----------")
print(type(res))
print(res.keys())
```

```
{"search_metadata":{"count":3,"completed_in":0.015,"max_id_str":"536624519285583872","since_id_st
r":
----------
<class 'dict'>
['search_metadata', 'statuses']
```

Activity: 2 -- ActiveCode (ac300_1_2)

17.7.2. Extract

In the extraction phase, you will be diving one level deeper into the nested data.

1. If it's a dictionary, figure out which key has the value you're looking for, and get its value. For example: `res2 = res['statuses']`
2. If it's a list, you will typically be wanting to do something with each of the items (e.g., extracting something from each, and accumulating them in a list). For that you'll want a for loop, such as `for res2 in res`. During your exploration phase, however, it will be easier to debug things if you work with just one item. One trick for doing that is to iterate over a slice of the list containing just one item. For example, `for res2 in res[:1]`.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
1 print(type(res))
2 print(res.keys())
3 res2 = res['statuses']
4
```

```
<class 'dict'>
['search_metadata', 'statuses']
```

Activity: 3 -- ActiveCode (ac300_1_3)

17.7.3. Repeat

Now you'll repeat the Understand and Extract processes at the next level.

17.7.3.1. Level 2

First understand.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
1 print(type(res))
2 print(res.keys())
3 res2 = res['statuses']
4 print("----Level 2-----")
5 print(type(res2)) # it's a list!
6 print(len(res2))
7
```

```
<class 'dict'>
['search_metadata', 'statuses']
----Level 2-----
<class 'list'>
3
```

Activity: 4 -- ActiveCode (ac300_1_4)

It's a list, with three items, so it's a good guess that each item represents one tweet.

Now extract. Since it's a list, we'll want to work with each item, but to keep things manageable for now, let's use the trick for just looking at the first item. Later we'll switch to processing all the items.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
1 import json
2 print(type(res))
3 print(res.keys())
```

```
res2 = res['statuses']
print("----Level 2: a list of tweets-----")
print(type(res2)) # it's a list!
print(len(res2))  # looks like one item representing each of the three tweets
for res3 in res2[:1]:
    print("----Level 3: a tweet----")
    print(json.dumps(res3, indent=2)[:30])
```

```
<class 'dict'>
['search_metadata', 'statuses']
----Level 2: a list of tweets-----
<class 'list'>
3
----Level 3: a tweet----
{"id":536624519285583872","id
```

Activity: 5 -- ActiveCode (ac300_1_5)

17.7.3.2. Level 3

First understand.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
import json
print(type(res))
print(res.keys())
res2 = res['statuses']
print("----Level 2: a list of tweets-----")
print(type(res2)) # it's a list!
print(len(res2))  # looks like one item representing each of the three tweets
for res3 in res2[:1]:
    print("----Level 3: a tweet----")
    print(json.dumps(res3, indent=2)[:30])
    print(type(res3)) # it's a dictionary
    print(res3.keys())
```

```
<class 'dict'>
['search_metadata', 'statuses']
----Level 2: a list of tweets-----
<class 'list'>
3
----Level 3: a tweet----
{"id":536624519285583872","id
<class 'dict'>
['id', 'id_str', 'entities', 'lang', 'created_at', 'contributors', 'truncated', 'text', 'in_reply_
to_status_id', 'favorite_count', 'source', 'retweeted', 'coordinates', 'in_reply_to_screen_name',
'in_reply_to_user_id', 'retweet_count', 'favorited', 'user', 'geo', 'in_reply_to_user_id_str', 'in
_reply_to_status_id_str', 'place', 'metadata', 'retweeted_status']
```

Activity: 6 -- ActiveCode (ac300_1_6)

Then extract. Let's pull out the information about who sent each of the tweets. Probably that's the value associated with the 'user' key.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
import json
print(type(res))
print(res.keys())
res2 = res['statuses']
print("----Level 2: a list of tweets-----")
print(type(res2)) # it's a list!
print(len(res2))  # looks like one item representing each of the three tweets
for res3 in res2[:1]:
    print("----Level 3: a tweet----")
    print(json.dumps(res3, indent=2)[:30])
    res4 = res3['user']
```

```
<class 'dict'>
['search_metadata', 'statuses']
----Level 2: a list of tweets-----
<class 'list'>
3
----Level 3: a tweet----
{"id":536624519285583872","id
```

Activity: 7 -- ActiveCode (ac300_1_7)

Now repeat.

17.7.3.3. Level 4

Understand.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
import json
print(type(res))
print(res.keys())
res2 = res['statuses']
print("----Level 2: a list of tweets-----")
print(type(res2)) # it's a list!
print(len(res2))  # looks like one item representing each of the three tweets
for res3 in res2[:1]:
    print("----Level 3: a tweet----")
    print(json.dumps(res3, indent=2)[:30])
    res4 = res3['user']
    print("----Level 4: the user who wrote the tweet----")
    print(type(res4)) # it's a dictionary
    print(res4.keys())

```

```
<class 'dict'>
['search_metadata', 'statuses']
----Level 2: a list of tweets-----
<class 'list'>
3
----Level 3: a tweet----
{"id":536624519285583872","id
----Level 4: the user who wrote the tweet----
<class 'dict'>
['id', 'id_str', 'screen_name', 'name', 'description', 'follow_request_sent', 'profile_use_backgro
und_image', 'profile_text_color', 'default_profile_image', 'profile_background_image_url_https',
'verified', 'profile_location', 'profile_image_url_https', 'profile_sidebar_fill_color', 'entitie
s', 'followers_count', 'profile_sidebar_border_color', 'profile_background_color', 'listed_count',
'is_translation_enabled', 'utc_offset', 'statuses_count', 'friends_count', 'location', 'profile_li
nk_color', 'profile_image_url', 'following', 'geo_enabled', 'profile_banner_url', 'profile_backgro
und_image_url', 'lang', 'profile_background_tile', 'favourites_count', 'notifications', 'url', 'cr
eated_at', 'contributors_enabled', 'time_zone', 'protected', 'default_profile', 'is_translator']
```

Activity: 8 -- ActiveCode (ac300_1_8)

Extract. Let's print out the user's screen name and when their account was created.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
import json
# print(type(res))
# print(res.keys())
res2 = res['statuses']
# print("----Level 2: a list of tweets-----")
# print(type(res2)) # it's a list!
# print(len(res2))  # looks like one item representing each of the three tweets
for res3 in res2[:1]:
    print("----Level 3: a tweet----")
    # print(json.dumps(res3, indent=2)[:30])
    res4 = res3['user']
    print("----Level 4: the user who wrote the tweet----")
    # print(type(res4)) # it's a dictionary
    # print(res4.keys())
    print(res4['screen name'], res4['created at'])
```

```
----Level 3: a tweet----
----Level 4: the user who wrote the tweet----
31brooks_ Wed Apr 09 14:34:41 +0000 2014
```

Activity: 9 -- ActiveCode (ac300_1_9)

Now, we may want to go back have it extract for all the items rather than only the first item in res2.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
import json
# print(type(res))
# print(res.keys())
res2 = res['statuses']
#print("----Level 2: a list of tweets-----")
#print(type(res2)) # it's a list!
#print(len(res2))  # looks like one item representing each of the three tweets
for res3 in res2:
    #print("----Level 3: a tweet----")
    #print(json.dumps(res3, indent=2)[:30])
    res4 = res3['user']
    #print("----Level 4: the user who wrote the tweet----")
    #print(type(res4)) # it's a dictionary
    #print(res4.keys())
    print(res4['screen name'], res4['created at'])
```

```
31brooks_ Wed Apr 09 14:34:41 +0000 2014
froyoho Thu Jan 14 21:37:54 +0000 2010
```

```
MDuncan95814 Tue Sep 11 21:02:09 +0000 2012
```

Activity: 10 -- ActiveCode (ac300_1_10)

17.7.3.4. Reflections

Notice that each time we descend a level in a dictionary, we have a [] picking out a key. Each time we look inside a list, we will have a for loop. If there are lists at multiple levels, we will have nested for loops.

Once you've figured out how to extract everything you want, you *may* choose to collapse things with multiple extractions in a single expression. For example, we could have this shorter version.

Save & Run | Original - 1 of 1 | Show in CodeLens

```
1 for res3 in res['statuses']:
2     print(res3['user']['screen_name'], res3['user']['created_at'])
3
```

```
31brooks_ Wed Apr 09 14:34:41 +0000 2014
froyoho Thu Jan 14 21:37:54 +0000 2010
MDuncan95814 Tue Sep 11 21:02:09 +0000 2012
```

Activity: 11 -- ActiveCode (ac300_1_11)

Even with this compact code, we can still count off how many levels of nesting we have extracted from, in this case four. res['statuses'] says we have descended one level (in a dictionary). for res3 in… says we are have descended another level (in a list). ['user'] is descending one more level, and ['screen_name'] is descending one more level.

You have attempted 12 of 11 activities on this page

17.6. Deep and Shallow Copies">
ep and Shallow Copies">
✔ Completed. Well Done!
17.8. Exercises">
17.8. Exercises">Next Section - 17.8. Exercises

| Back to top