Simple APIs

Random User and Fruitvice API Examples

Estimated time needed: **25** minutes

Objectives

After completing this lab you will be able to:

- Load and use RandomUser API, using `RandomUser()` Python library
- Load and use Fruitvice API, using `requests` Python library

The purpose of this notebook is to provide more examples on how to use simple APIs. As you have already learned from previous videos and notebooks, API stands for Application Programming Interface and is a software intermediary that allows two applications to talk to each other.

The advantages of using APIs:

- **Automation**. Less human effort is required and workflows can be easily updated to become faster and more productive.
- **Efficiency**. It allows to use the capabilities of one of the already developed APIs than to try to independently implement some functionality from scratch.

The disadvantage of using APIs:

- **Security**. If the API is poorly integrated, it means it will be vulnerable to attacks, resulting in data breeches or losses having financial or reputation implications.

One of the applications we will use in this notebook is Random User Generator. RandomUser is an open-source, free API providing developers with randomly generated users to be used as placeholders for testing purposes. This makes the tool similar to Lorem Ipsum, but is a placeholder for people instead of text. The API can return multiple results, as well as specify generated user details such as gender, email, image, username, address, title, first and last name, and more. More information on RandomUser can be found here.

Another example of simple API we will use in this notebook is Fruitvice application. The Fruitvice API webservice which provides data for all kinds of fruit! You can use Fruityvice to find out interesting information about fruit and educate yourself. The webservice is completely free to use and contribute to.

Example 1: RandomUser API

Bellow are Get Methods parameters that we can generate. For more information on the parameters, please visit this documentation page.

Get Methods

- get_cell()
- get_city()
- get_dob()
- get_email()
- get_first_name()
- get_full_name()
- get_gender()
- get_id()
- get_id_number()
- get_id_type()
- get_info()
- get_last_name()
- get_login_md5()
- get_login_salt()
- get_login_sha1()
- get_login_sha256()
- get_nat()
- get_password()
- get_phone()
- get_picture()
- get_postcode()
- get_registered()
- get_state()
- get_street()
- get_username()
- get_zipcode()

To start using the API you can install the `randomuser` library running the `pip install` command.

In [1]:
```
!pip install randomuser
```

```
Collecting randomuser
  Downloading randomuser-1.6.tar.gz (5.0 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: randomuser
  Building wheel for randomuser (setup.py) ... done
  Created wheel for randomuser: filename=randomuser-1.6-py3-none-any.whl size=5066 s
ha256=48553ae6000abdd04c9a01c644c638075f1d6c4902d0eaa8356fef3e0a5eac2c
  Stored in directory: /home/jupyterlab/.cache/pip/wheels/21/10/7b/c13bc3e24a3c1000a
34554802ad8aa0ff27df56366998a0473
Successfully built randomuser
Installing collected packages: randomuser
Successfully installed randomuser-1.6
```

Then, we will load the necessary libraries.

In [2]:
```
from randomuser import RandomUser
import pandas as pd
```

First, we will create a random user object, r.

In [3]:
```
r = RandomUser()
```

Then, using `generate_users()` function, we get a list of random 10 users.

In [4]:
```
some_list = r.generate_users(10)
```

In [5]:
```
some_list
```

Out[5]:
```
[<randomuser.RandomUser at 0x7fc30d3b8650>,
 <randomuser.RandomUser at 0x7fc30d3b8690>,
 <randomuser.RandomUser at 0x7fc30d3b86d0>,
 <randomuser.RandomUser at 0x7fc30d3b8710>,
 <randomuser.RandomUser at 0x7fc30d3b8750>,
 <randomuser.RandomUser at 0x7fc30d3b87d0>,
 <randomuser.RandomUser at 0x7fc30d3b8810>,
 <randomuser.RandomUser at 0x7fc30d3b8850>,
 <randomuser.RandomUser at 0x7fc30d3b8890>,
 <randomuser.RandomUser at 0x7fc30d3b8790>]
```

The **"Get Methods"** functions mentioned at the beginning of this notebook, can generate the required parameters to construct a dataset. For example, to get full name, we call `get_full_name()` function.

In [6]:
```
name = r.get_full_name()
```

Let's say we only need 10 users with full names and their email addresses. We can write a "for-loop" to print these 10 users.

In [7]:
```
for user in some_list:
    print (user.get_full_name()," ",user.get_email())
```

```
Juho Ranta   juho.ranta@example.com
Nicolas Gagnon   nicolas.gagnon@example.com
Michail Van der Staaij   michail.vanderstaaij@example.com
Emma Slawa   emma.slawa@example.com
Eve Pierre   eve.pierre@example.com
Monica Soler   monica.soler@example.com
Heinz-Dieter Schönemann   heinz-dieter.schonemann@example.com
Hugo Castro   hugo.castro@example.com
Ella Arnold   ella.arnold@example.com
Elias Andersen   elias.andersen@example.com
```

Exercise 1

In this Exercise, generate photos of the random 5 users.

In [10]:
```
## Write your code here
photo = r.get_picture()

for user in some_list:
    print (user.get_full_name()," ", photo)
```

```
Juho Ranta   https://randomuser.me/api/portraits/women/79.jpg
Nicolas Gagnon   https://randomuser.me/api/portraits/women/79.jpg
Michail Van der Staaij   https://randomuser.me/api/portraits/women/79.jpg
Emma Slawa   https://randomuser.me/api/portraits/women/79.jpg
Eve Pierre   https://randomuser.me/api/portraits/women/79.jpg
Monica Soler   https://randomuser.me/api/portraits/women/79.jpg
Heinz-Dieter Schönemann   https://randomuser.me/api/portraits/women/79.jpg
Hugo Castro   https://randomuser.me/api/portraits/women/79.jpg
Ella Arnold   https://randomuser.me/api/portraits/women/79.jpg
Elias Andersen   https://randomuser.me/api/portraits/women/79.jpg
```

▶ Click here for the solution

To generate a table with information about the users, we can write a function containing all desirable parameters. For example, name, gender, city, etc. The parameters will depend on the requirements of the test to be performed. We call the Get Methods, listed at the beginning of this notebook. Then, we return pandas dataframe with the users.

In [11]:
```
def get_users():
    users =[]

    for user in RandomUser.generate_users(10):
        users.append({"Name":user.get_full_name(),"Gender":user.get_gender(),"City"

    return pd.DataFrame(users)
```

In [12]:
```
get_users()
```

Out[12]:

	Name	Gender	City	State	Email	
0	Wallace Fisher	male	Peterborough	Kent	wallace.fisher@example.com	1 01T07:03:
1	Julia Herrero	female	Granada	Castilla la Mancha	julia.herrero@example.com	1 25T23:48:
2	Amber Morris	female	Tauranga	Northland	amber.morris@example.com	1 03T02:13:
3	Guy Lacroix	male	Reutigen	Vaud	guy.lacroix@example.com	1 14T08:07:
4	Jacob Christensen	male	Sønder Stenderup	Hovedstaden	jacob.christensen@example.com	1 14T14:53:
5	Jens Georgsen	male	Aksdal	Oslo	jens.georgsen@example.com	1 21T23:45:
6	Vedat Solmaz	male	Gümüşhane	Aydın	vedat.solmaz@example.com	1 17T15:11:
7	Lucia Pastor	female	Alcobendas	La Rioja	lucia.pastor@example.com	1 25T13:16:
8	Kurt Hanson	male	Stoke-on-Trent	Wiltshire	kurt.hanson@example.com	1 27T16:03:
9	Gonca Atakol	female	Eskişehir	Kars	gonca.atakol@example.com	1 09T19:39:

In [13]:
```
df1 = pd.DataFrame(get_users())
```

Now we have a *pandas* dataframe that can be used for any testing purposes that the tester might have.

Example 2: Fruitvice API

Another, more common way to use APIs, is through `requests` library. The next lab, Requests and HTTP, will contain more information about requests.

We will start by importing all required libraries.

In [1]:
```
import requests
import json
```

We will obtain the fruitvice API data using `requests.get("url")` function. The data is in a json format.

In [15]:
```
data = requests.get("https://fruityvice.com/api/fruit/all")
```

We will retrieve results using `json.loads()` function.

In [16]:
```
results = json.loads(data.text)
```

We will convert our json data into *pandas* data frame.

In [17]:
```
pd.DataFrame(results)
```

Out[17]:

	name	id	family	order	genus	nutritions
0	Persimmon	52	Ebenaceae	Rosales	Diospyros	{'calories': 81, 'fat': 0.0, 'sugar': 18.0, 'c...
1	Strawberry	3	Rosaceae	Rosales	Fragaria	{'calories': 29, 'fat': 0.4, 'sugar': 5.4, 'ca...
2	Banana	1	Musaceae	Zingiberales	Musa	{'calories': 96, 'fat': 0.2, 'sugar': 17.2, 'c...
3	Tomato	5	Solanaceae	Solanales	Solanum	{'calories': 74, 'fat': 0.2, 'sugar': 2.6, 'ca...
4	Pear	4	Rosaceae	Rosales	Pyrus	{'calories': 57, 'fat': 0.1, 'sugar': 10.0, 'c...
5	Durian	60	Malvaceae	Malvales	Durio	{'calories': 147, 'fat': 5.3, 'sugar': 6.75, '...
6	Blackberry	64	Rosaceae	Rosales	Rubus	{'calories': 40, 'fat': 0.4, 'sugar': 4.5, 'ca...
7	Lingonberry	65	Ericaceae	Ericales	Vaccinium	{'calories': 50, 'fat': 0.34, 'sugar': 5.74, '...
8	Kiwi	66	Actinidiaceae	Struthioniformes	Apteryx	{'calories': 61, 'fat': 0.5, 'sugar': 9.0, 'ca...
9	Lychee	67	Sapindaceae	Sapindales	Litchi	{'calories': 66, 'fat': 0.44, 'sugar': 15.0, '...
10	Pineapple	10	Bromeliaceae	Poales	Ananas	{'calories': 50, 'fat': 0.12, 'sugar': 9.85, '...
11	Fig	68	Moraceae	Rosales	Ficus	{'calories': 74, 'fat': 0.3, 'sugar': 16.0, 'c...
12	Gooseberry	69	Grossulariaceae	Saxifragales	Ribes	{'calories': 44, 'fat': 0.6, 'sugar': 0.0, 'ca...
13	Passionfruit	70	Passifloraceae	Malpighiales	Passiflora	{'calories': 97, 'fat': 0.7, 'sugar': 11.2, 'c...
14	Plum	71	Rosaceae	Rosales	Prunus	{'calories': 46, 'fat': 0.28, 'sugar': 9.92, '...
15	Orange	2	Rutaceae	Sapindales	Citrus	{'calories': 43, 'fat': 0.2, 'sugar': 8.2, 'ca...
16	GreenApple	72	Rosaceae	Rosales	Malus	{'calories': 21, 'fat': 0.1, 'sugar': 6.4, 'ca...
17	Raspberry	23	Rosaceae	Rosales	Rubus	{'calories': 53, 'fat': 0.7, 'sugar': 4.4, 'ca...
18	Watermelon	25	Cucurbitaceae	Cucurbitales	Citrullus	{'calories': 30, 'fat': 0.2, 'sugar': 6.0, 'ca...

	name	id	family	order	genus	nutritions
19	Lemon	26	Rutaceae	Sapindales	Citrus	{'calories': 29, 'fat': 0.3, 'sugar': 2.5, 'ca...
20	Mango	27	Anacardiaceae	Sapindales	Mangifera	{'calories': 60, 'fat': 0.38, 'sugar': 13.7, '...
21	Blueberry	33	Rosaceae	Rosales	Fragaria	{'calories': 29, 'fat': 0.4, 'sugar': 5.4, 'ca...
22	Apple	6	Rosaceae	Rosales	Malus	{'calories': 52, 'fat': 0.4, 'sugar': 10.3, 'c...
23	Guava	37	Myrtaceae	Myrtales	Psidium	{'calories': 68, 'fat': 1.0, 'sugar': 9.0, 'ca...
24	Apricot	35	Rosaceae	Rosales	Prunus	{'calories': 15, 'fat': 0.1, 'sugar': 3.2, 'ca...
25	Papaya	42	Caricaceae	Caricacea	Carica	{'calories': 43, 'fat': 0.4, 'sugar': 1.0, 'ca...
26	Melon	41	Cucurbitaceae	Cucurbitaceae	Cucumis	{'calories': 34, 'fat': 0.0, 'sugar': 8.0, 'ca...
27	Tangerine	77	Rutaceae	Sapindales	Citrus	{'calories': 45, 'fat': 0.4, 'sugar': 9.1, 'ca...
28	Pitahaya	78	Cactaceae	Caryophyllales	Cactaceae	{'calories': 36, 'fat': 0.4, 'sugar': 3.0, 'ca...
29	Lime	44	Rutaceae	Sapindales	Citrus	{'calories': 25, 'fat': 0.1, 'sugar': 1.7, 'ca...
30	Pomegranate	79	Lythraceae	Myrtales	Punica	{'calories': 83, 'fat': 1.2, 'sugar': 13.7, 'c...
31	Dragonfruit	80	Cactaceae	Caryophyllales	Selenicereus	{'calories': 60, 'fat': 1.5, 'sugar': 8.0, 'ca...
32	Grape	81	Vitaceae	Vitales	Vitis	{'calories': 69, 'fat': 0.16, 'sugar': 16.0, '...
33	Morus	82	Moraceae	Rosales	Morus	{'calories': 43, 'fat': 0.39, 'sugar': 8.1, 'c...
34	Feijoa	76	Myrtaceae	Myrtoideae	Sellowiana	{'calories': 44, 'fat': 0.4, 'sugar': 3.0, 'ca...
35	Avocado	84	Lauraceae	Laurales	Persea	{'calories': 160, 'fat': 14.66, 'sugar': 0.66,...
36	Kiwifruit	85	Actinidiaceae	Ericales	Actinidia	{'calories': 61, 'fat': 0.5, 'sugar': 8.9, 'ca...
37	Cranberry	87	Ericaceae	Ericales	Vaccinium	{'calories': 46, 'fat': 0.1, 'sugar': 4.0, 'ca...

	name	id	family	order	genus	nutritions
38	Cherry	9	Rosaceae	Rosales	Prunus	{'calories': 50, 'fat': 0.3, 'sugar': 8.0, 'ca...
39	Peach	86	Rosaceae	Rosales	Prunus	{'calories': 39, 'fat': 0.25, 'sugar': 8.4, 'c...
40	Jackfruit	94	Moraceae	Rosales	Artocarpus	{'calories': 95, 'fat': 0.0, 'sugar': 19.1, 'c...
41	Horned Melon	95	Cucurbitaceae	Cucurbitales	Cucumis	{'calories': 44, 'fat': 1.26, 'sugar': 0.5, 'c...

The result is in a nested json format. The 'nutrition' column contains multiple subcolumns, so the data needs to be 'flattened' or normalized.

In [18]:
```
df2 = pd.json_normalize(results)
```

In [19]:
```
df2
```

Out[19]:

	name	id	family	order	genus	nutritions.calories	nutri
0	Persimmon	52	Ebenaceae	Rosales	Diospyros	81	
1	Strawberry	3	Rosaceae	Rosales	Fragaria	29	
2	Banana	1	Musaceae	Zingiberales	Musa	96	
3	Tomato	5	Solanaceae	Solanales	Solanum	74	
4	Pear	4	Rosaceae	Rosales	Pyrus	57	
5	Durian	60	Malvaceae	Malvales	Durio	147	
6	Blackberry	64	Rosaceae	Rosales	Rubus	40	
7	Lingonberry	65	Ericaceae	Ericales	Vaccinium	50	
8	Kiwi	66	Actinidiaceae	Struthioniformes	Apteryx	61	
9	Lychee	67	Sapindaceae	Sapindales	Litchi	66	
10	Pineapple	10	Bromeliaceae	Poales	Ananas	50	
11	Fig	68	Moraceae	Rosales	Ficus	74	
12	Gooseberry	69	Grossulariaceae	Saxifragales	Ribes	44	
13	Passionfruit	70	Passifloraceae	Malpighiales	Passiflora	97	
14	Plum	71	Rosaceae	Rosales	Prunus	46	
15	Orange	2	Rutaceae	Sapindales	Citrus	43	
16	GreenApple	72	Rosaceae	Rosales	Malus	21	
17	Raspberry	23	Rosaceae	Rosales	Rubus	53	
18	Watermelon	25	Cucurbitaceae	Cucurbitales	Citrullus	30	
19	Lemon	26	Rutaceae	Sapindales	Citrus	29	
20	Mango	27	Anacardiaceae	Sapindales	Mangifera	60	
21	Blueberry	33	Rosaceae	Rosales	Fragaria	29	
22	Apple	6	Rosaceae	Rosales	Malus	52	
23	Guava	37	Myrtaceae	Myrtales	Psidium	68	
24	Apricot	35	Rosaceae	Rosales	Prunus	15	
25	Papaya	42	Caricaceae	Caricacea	Carica	43	
26	Melon	41	Cucurbitaceae	Cucurbitaceae	Cucumis	34	
27	Tangerine	77	Rutaceae	Sapindales	Citrus	45	
28	Pitahaya	78	Cactaceae	Caryophyllales	Cactaceae	36	
29	Lime	44	Rutaceae	Sapindales	Citrus	25	

	name	id	family	order	genus	nutritions.calories	nutri
30	Pomegranate	79	Lythraceae	Myrtales	Punica	83	
31	Dragonfruit	80	Cactaceae	Caryophyllales	Selenicereus	60	
32	Grape	81	Vitaceae	Vitales	Vitis	69	
33	Morus	82	Moraceae	Rosales	Morus	43	
34	Feijoa	76	Myrtaceae	Myrtoideae	Sellowiana	44	
35	Avocado	84	Lauraceae	Laurales	Persea	160	
36	Kiwifruit	85	Actinidiaceae	Ericales	Actinidia	61	
37	Cranberry	87	Ericaceae	Ericales	Vaccinium	46	
38	Cherry	9	Rosaceae	Rosales	Prunus	50	
39	Peach	86	Rosaceae	Rosales	Prunus	39	
40	Jackfruit	94	Moraceae	Rosales	Artocarpus	95	
41	Horned	95	Cucurbitaceae	Cucurbitales	Cucumis	44	

Let's see if we can extract some information from this dataframe. Perhaps, we need to know the family and genus of a cherry.

In [20]:
```
cherry = df2.loc[df2["name"] == 'Cherry']
(cherry.iloc[0]['family']) , (cherry.iloc[0]['genus'])
```

Out[20]:
```
('Rosaceae', 'Prunus')
```

Exercise 2

In this Exercise, find out how many calories are contained in a banana.

In [25]:
```
# Write your code here
banana = df2.loc[df2["name"] == 'Banana']
(banana.iloc[0]['nutritions.calories'])
```

Out[25]:
```
96
```

▶ Click here for the solution

Exercise 3

This page contains a list of free public APIs. Choose any API of your interest and use it to load/extract some information, as shown in the example above.

1. Using `requests.get("url")` function, load your data.

In [6]:
```
# Write your code here
url="https://www.petfinder.com/developers/"
data2= requests.get(url)
```

2. Retrieve results using `json.loads()` function.

In [7]:
```
result2=json.loads(data2.text)
```

```
---------------------------------------------------------------------------
JSONDecodeError                           Traceback (most recent call last)
/tmp/ipykernel_399/1138914910.py in <module>
----> 1 result2=json.loads(data2.text)

~/conda/envs/python/lib/python3.7/json/__init__.py in loads(s, encoding, cls, object
_hook, parse_float, parse_int, parse_constant, object_pairs_hook, **kw)
    346             parse_int is None and parse_float is None and
    347             parse_constant is None and object_pairs_hook is None and not k
w):
--> 348         return _default_decoder.decode(s)
    349     if cls is None:
    350         cls = JSONDecoder

~/conda/envs/python/lib/python3.7/json/decoder.py in decode(self, s, _w)
    335
    336         """
--> 337         obj, end = self.raw_decode(s, idx=_w(s, 0).end())
    338         end = _w(s, end).end()
    339         if end != len(s):

~/conda/envs/python/lib/python3.7/json/decoder.py in raw_decode(self, s, idx)
    353             obj, end = self.scan_once(s, idx)
    354         except StopIteration as err:
--> 355             raise JSONDecodeError("Expecting value", s, err.value) from None
    356         return obj, end

JSONDecodeError: Expecting value: line 1 column 1 (char 0)
```

▶ Click here for the solution

3. Convert json data into *pandas* data frame.

In [12]:
```
# Write your code here
pd.DataFrame(result2)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_434/525063997.py in <module>
      1 # Write your code here
----> 2 pd.DataFrame(result2)

NameError: name 'result2' is not defined
```

▶ Click here for the solution

Congratulations! - You have completed the lab

Author

Svitlana Kramar

Svitlana is a master's degree Data Science and Analytics student at University of Calgary, who enjoys travelling, learning new languages and cultures and loves spreading her passion for Data Science.