1.10: Python Assignment 1

Getting Molecular Properties through PUG-REST

Downloadable Files

Lecture01_Basics.ipynb

- Download and run the above file in your Jupyter notebook.
  - You can use the notebook you created in section 1.5 or the Jupyter hub at LibreText: https://jupyter.libretexts.org (see your instructor if you do not have access to the hub).
- This page is an html version of the above file.
  - If you have questions on this assignment you should use this web page and the hypothes.is annotation to post a question (or comment) to the 2019OLCCStu class group. If you are not on the discussion group you should contact your instructor for the link to join.

Objectives

- Learn the basic approach to getting data from PubChem through PUG-REST
- Retrieve a single property of a single compound.
- Retrieve a single property of multiple compounds
- Retrieve multiple properties of multiple compounds.
- Write a `for` loop to make the same kind of requests.
- Process a large amount of data by splitting them into smaller chunks

The Shortest Code to Get PubChem Data

Let's suppose that we want to get the molecular formula of water from PubChem through PUG-REST. You can get this data from your web browsers (Chrome, Safari, Internet Explorer, etc) via the following URL:
https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/water/property/MolecularFormula/txt
Getting the same data using a computer program is not very difficult. This task can be with a three-line code.

Line 1: First, the "**requests**" **python library** (https://3.python-requests.org/) is imported. The "requests" library contains a set of pre-written codes that allows you to access information on the web.

In [1]:

```
1 import requests
```

Note: if you receive an error indicating that you do not have the requests library, you should go back to your anaconda prompt and type

```
pip install requests
```

Line 2: Get the desired information using **the function get( )** in the requests library. The PUG-REST request URL (enclosed within a pair of quotes(") is provided within the parentheses. The result will be stored in **a variable** called **res** .

In [2]:

```
1 res =
  requests.get('https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/water/prop
```

Line 3: The **res** variable (which means "result" or "response") contains not only the requested data but also some information about the request. To view the returned data, you need to get the data from **res** and **print** it out.

In [3]:

```
1 print(res.text)
```

As another example, the following code retrieves the number of heavy (non-hydrogen) atoms of butadiene.

In [4]:

```
1 res =
  requests.get('https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/butadiene/
2 print(res.text)
```

Note that in this example, we did not import the **requests** library because it has already been imported (in the very fist example for getting the molecular formula of water).

LibreText Reading:

Review Section 1.6.2 Rest Architecture before doing this assignment, and reference back to the Compound Properties Table as needed.

Exercise 1a: Retrieve the molecular weight of ethanol in a "text" format.

In [7]:

```
# Write your code in this cell:
```

Exercise 1b: Retrieve the number of hydrogen-bond acceptors of aspirin in a "text" format.

In [8]:

```
# Write your code in this cell:
```

Formulating PUG-REST request URLs using variables

In the previous examples, the PUG-REST request URLs were directly provided to the **requests.get( )**, by explicitly typing the URL within the parentheses. However, it is also possible to provide the URL using a variable. The following example shows how to formulate the PUG-REST request URL using variables and pass it to **requests.get( )**.

In [9]:

```
1 pugrest = "https://pubchem.ncbi.nlm.nih.gov/rest/pug"
2 pugin   = "compound/name/water"
3 pugoper = "property/MolecularFormula"
4 pugout  = "txt"
5
6 url     = pugrest + '/' + pugin + '/' + pugoper + '/' + pugout
7 print(url)
```

https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/water/property/MolecularFormula/txt

A PUG-REST request URL encodes three pieces of information (input, operation, output), preceded by the prologue commont to all requests. In the above code cell, these pieces of information are stored in four different variables (**pugrest**, **pugin**, **pugoper**, **pugout**) and combined into a new variable **url**.

One can also generate the same URL using the **join( )** function, available for a string.

In [10]:

```
url = "/".join( [pugrest, pugin, pugoper, pugout] )
print(url)
```

```
https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/water/property/Molecular
```

Here, the strings stored in the four variables are joined by the "/" character as a separator. Note that the four variables are enclosed within the square bracket ([]), meaning that a list containing them as elements is provided to **join( )**.

Then, the url can be passed to **requests.get( )**.

In [11]:

```
res = requests.get(url)
print(res.text)
```

Warning: Avoid using **in** or **input** as a variable name in python. In python, in is a reserved keyword and input is the name of a built-in function. In the example above, the variables are prefixed with "pug" to avoid this naming conflict.

Making multiple requests using a for loop

The approach in the previous section (that use variables to construct a request URL) looks very inconvenient, compared to the three-line code shown at the beginning, where the request URL is directly provided to **requests.get( )**. If you are making only one request, it would be simpler to provide the URL directly to **requests.get( )**, rather than assign the pieces to variables, constructing the URL from them, and passing it to the function.
However, if you are making a large number of requests, it would be very time consuming to type the respective request URLs for all requests. In that case, you want to store common parts as variables and use them in a loop. For example, suppose that you want to retrieve the SMILES strings of 5 chemicals.

In [12]:

```
names = [ 'cytosine', 'benzene', 'motrin', 'aspirin', 'zolpidem' ]
```

Now the chemical names are stored in a list called **names** . Using a **for** loop, you can loop over each chemical name, formulating the request URL and retrieving the desired data, as shown below.

In [13]:

```
pugrest = "https://pubchem.ncbi.nlm.nih.gov/rest/pug"
pugoper = "property/CanonicalSMILES"
pugout  = "txt"

for myname in names:    # loop over each element in the "names" list

    pugin = "compound/name/" + myname

    url = "/".join( [pugrest, pugin, pugoper, pugout] )
    res = requests.get(url)
    print(myname, ":", res.text)
```

Warning: When you make a lot of programmatic access requests using a loop, you should limit your request rate to or below **five requests per second**. Please read the following document to learn more about PubChem's usage

policies:https://pubchemdocs.ncbi.nlm.nih.gov/programmatic-access$_RequestVolumeLimitations
Violation of usage policies may result in the user being **temporarily blocked** from accessing PubChem (or NCBI) resources**

In the for-loop example above, we have only five input chemical names to process, so it is not likely to violate the five-requests-per-second limit. However, if you have thousands of names to process, the above code will exceed the limit (considering that this kind of requests usually finish very quickly). Therefore, the request rate should be adjusted by using the **sleep( )** function in the **time** module. For simplicity, let's suppose that you have 12 chemical names to process (in reality, you could have much more to process).

In [14]:

```
names = [ 'water', 'benzene', 'methanol', 'ethene', 'ethanol', \
          'propene','1-propanol', '2-propanol', 'butadiene', '1-butanol', \
          '2-butanol', 'tert-butanol']
```

LibreText Reading:

In analyzing the code of the following example you should reference Code Example 9.1.1 of Section 9.1.2 of Appendix 9.1

In [15]:

```
01 import time
02
03 pugrest = "https://pubchem.ncbi.nlm.nih.gov/rest/pug"
04 pugoper = "property/CanonicalSMILES"
05 pugout  = "txt"
06
07 for i in range(len(names)):    # loop over each index (position) in the
   "names" list
08
09     pugin = "compound/name/" + names[i]    # names[i] = the ith element in the
   names list.
10
11     url = "/".join( [pugrest, pugin, pugoper, pugout] )
12     res = requests.get(url)
13     print(names[i], ":", res.text)
14
15     if ( i % 5 == 4 ) :  # the % is the modulo operator and returns the
   remainder of a calculation (if i = 4, 9, ...)
16         time.sleep(1)
```

There are three things noteworthy in the above example (compared to the previous examples with the five chemical name queries).

- First, the for loop iterates from 0 to [**len(names)** − 1], that is, [0, 1, 2, 3, ...,11].
- The variable **i** is used (in **names(i)** to generate the input part (**pugin**) of the PUG-REST request URL.
- The variable **i** is used (in the **if** if sentence) to stop the program for one second for every five requests.

It should be noted that the request volumn limit can be lowered through the dynamic traffic control at times of excessive load (https://pubchemdocs.ncbi.nlm.nih.gov/dynamic-request-throttling). Throttling information is provided in the HTTP header response, indicating the system-load state and the per-user limits. Based on this throttling information, the user should moderate the speed at which requests are sent to PubChem. We will cover this topic later in this course.

Exercise 3a: Retrieve the XlogP values of linear alkanes with 1 ~ 12 carbons.

- Use the chemical names as inputs
- Use a for loop to retrieve the XlogP value for each alkane.
- Use the sleep() function to stop the program for one second for every **five** requests.

In [16]:

```
# Write your code in this cell: (The solution code below will be removed later)
```

Exercise 3b Retrieve the **isomeric** SMILES of the 20 common amino acids.

- Use the chemical names as inputs. Because the 20 common amino acids in living organisms predominantly exist as one chrial form (the L-form), the names should be prefixed with "**L-**" (e.g., "L-alanine", rather than "alanine"), except for "glycine" (which does not have a chiral center).
- Use a for loop to retrieve the isomeric SMILES for each alkane.
- Use the sleep() function to stop the program for one second for every **five** requests.

In [17]:

```
# Write your code in this cell (The solution code below will be removed later)
```

Getting multiple molecular properties

All the examples we have seen in this notebook retrieved a single molecular property for a single compound (although we were able to get a desired property for a group of compounds using a for loop). However, it is possible to get multiple properties for multiple compounds with a single request.

The following example retrieves the hydrogen-bond donor count, hydrogen-bond acceptor count, XLogP, TPSA for 5 compounds (represented by PubChem Compound IDs (CIDs) in a comma-separated values (CSV) format.

In [18]:

```
pugrest = "https://pubchem.ncbi.nlm.nih.gov/rest/pug"
pugin   = "compound/cid/4485,4499,5026,5734,8082"
pugoper = "property/HBondDonorCount,HBondDonorCount,XLogP,TPSA"
pugout  = "csv"

url = "/".join( [pugrest, pugin, pugoper, pugout] )   # Construct the URL
print(url)
print("-" * 30)   # Print "-" 30 times (to print a line for readability)

res = requests.get(url)
print(res.text)
```

```
https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/4485,4499,5026,5734,8082/prope
------------------------------
"CID","HBondDonorCount","HBondDonorCount","XLogP","TPSA"
4485,1,1,2.200,110.0
4499,1,1,3.300,110.0
5026,1,1,4.300,123.0
5734,1,1,0.2,94.6
8082,1,1,0.800,12.0
```

In [19]:

```
res.text.rstrip()
```

Out[19]:

```
'"CID","HBondDonorCount","HBondDonorCount","XLogP","TPSA"\n4485,1,1,2.200,110.0\n4499
```

PubChem has a standard time limit of **30 seconds per request**. When you try to retrieve too many properties for too many compounds with a single request, it can take longer than the 30-second limit and a time-out error will be returned. Therefore, you may need to split the compound list into smaller chunks and process one chunk at a time.

In [20]:

```
cids = [ 443422,  72301,   8082,    4485,    5353740, 5282230, 5282138, 1547484, 9413
         5494,    5422,    5417,    5290,    5245,    5026,    4746,    4507,    4499
         4494,    4474,    4418,    4386,    4009,    4008,    3949,    3926,    3878
         3698,    3547,    3546,    3336,    3333,    3236,    3076,    2585,    2520
         2312,    2162,    1236,    1234,    292331,  275182,  235244,  108144,  1049
         5942250, 5311217, 4564402, 4715169, 5311501]
```

In [21]:

```
chunk_size = 10

if ( len(cids) % chunk_size == 0 ) : # check if total number of cids is divisible by
    num_chunks = len(cids) // chunk_size # sets number of chunks
else : # if divide by 10 results in remainder
    num_chunks = len(cids) // chunk_size + 1 # add one more chunk

print("# Number of CIDs:", len(cids) )
print("# Number of chunks:", num_chunks )
```

```
# Number of CIDs: 55
# Number of chunks: 6
```

In [22]:

```
pugrest = "https://pubchem.ncbi.nlm.nih.gov/rest/pug"
pugoper = "property/HBondDonorCount,HBondAcceptorCount,XLogP,TPSA"
pugout  = "csv"

csv = ""   #sets a variable called csv to save the comma separated output

for i in range(num_chunks) : # sets number of requests to number of data chunks as de

    idx1 = chunk_size * i        # sets a variable for a moving window of cids to sta
    idx2 = chunk_size * (i + 1)  # sets a variable for a moving window of cids to end

    pugin = "compound/cid/" + ",".join([ str(x) for x in cids[idx1:idx2] ]) # build p
    url = "/".join( [pugrest, pugin, pugoper, pugout] )   # Construct the URL

    res = requests.get(url)
```

```
    if ( i == 0 ) : # if this is the first request, store result in empty csv variabl
        csv = res.text
    else :          # if this is a subsequent request, add the request to the csv var
        csv = csv + "\n".join(res.text.split()[1:]) + "\n"

    if (i % 5 == 4):
        time.sleep(1)

print(csv)
```

```
"CID","HBondDonorCount","HBondAcceptorCount","XLogP","TPSA"
443422,0,5,3.1,40.2
72301,0,5,3.2,40.2
8082,1,1,0.800,12.0
4485,1,7,2.200,110.0
5353740,2,5,3.5,76.0
5282230,2,5,3.2,84.9
5282138,1,8,4.400,120.0
1547484,0,2,5.800,6.5
941361,0,4,6.000,6.5
5734,1,5,0.2,94.6
5494,0,6,5.0,57.2
5422,0,8,6.4,61.9
5417,0,5,3.2,40.2
5290,2,5,2.6,62.2
5245,5,8,-3.1,148.0
5026,1,8,4.300,123.0
4746,1,1,6.8,12.0
4507,1,7,2.900,110.0
4499,1,7,3.300,110.0
4497,1,8,3.100,120.0
4494,1,8,2.900,134.0
4474,1,8,3.800,114.0
4418,1,5,4.100,45.2
4386,2,3,4.400,49.3
4009,2,5,3.5,76.0
4008,1,9,5.600,117.0
3949,0,7,4.9,34.2
3926,1,5,6.0,35.6
3878,2,5,1.4,90.7
3784,1,8,4.300,104.0
3698,2,3,-0.2,68.0
3547,1,5,1.0,70.7
3546,3,5,-0.5,132.0
3336,1,1,5.5,12.0
3333,1,5,3.900,64.6
```

```
3236,0,2,3.8,20.3
3076,0,6,3.1,84.4
2585,3,5,4.200,75.7
2520,0,6,3.800,64.0
2351,0,3,5.3,15.7
2312,0,2,4.6,12.5
2162,2,7,3.000,99.9
1236,1,8,6.800,114.0
1234,0,7,3.800,73.2
292331,2,3,3.900,49.3
275182,1,8,6.1,72.9
235244,1,8,6.7,72.9
108144,2,5,3.9,117.0
104972,1,6,3.300,72.7
77157,1,4,3.2,49.8
5942250,2,5,3.5,76.0
5311217,1,7,4.500,90.9
4564402,0,4,4.1,45.5
4715169,2,3,-1.6,63.3
5311501,0,4,4.4,43.7
```

Exercise 4a: Below is the list of CIDs of known antiinflmatory agents (obtained from PubChem via the URL: https://www.ncbi.nlm.nih.gov/pccompound?LinkName=mesh_pccompound&from_uid=68000893). Download the following properties of those compounds in a comma-separated format: Heavy atom count, rotatable bond count, molecular weight, XLogP, hydrogen bond donor count, hydrogen bond acceptor count, TPSA, and isomeric SMILES.

- Split the input CID list into small chunks (with a chunk size of 100 CIDs).
- Process one chunk at a time using a for loop.
- Do not forget to add sleep() to comply the usage policy.

In [23]:

```
cids = [ 471, 1981, 2005, 2097, 2151, 2198, 2206, 2214, 2244, 2307, 2308, 2313, 2355,
len(cids)
```

Out[23]:

```
708
```

In [24]:

```
# Write your code in this cell.
```