

Jul 10 Notes/HW (PS4)

Hello students it is I



Import Statement

Python's namespace has a lot of stuff. We've covered `{'key': 'value'}`, `{'a', 'b'}`, `'ab'`, `1`, `1.2`, `[1, 2, 3]`, `[[1], [2]]`, `(1, 2)` – a bunch of classes (`'str'`, `'dict'`). A lot of functions, too! `len()`, `enumerate()`, `range()`. Beyond these, the Python distribution has 100000s of *more* functions and classes organized in **libraries** or **modules**. These can be accessed and loaded using the **import** statement.

Let's revisit this problem: What is the smallest positive number that is evenly divisible by all the numbers from 1 to 20?

We are really asking: what's the least common multiple of 1-20. That's just

$$\text{lcm}(a, b) = \frac{|a * b|}{\text{gcd}(a, b)}$$

where *gcd* is the “greatest common denominator” and *lcm* is “least common multiple”. The *lcm* of 5 and 6 is 30, the *gcd* of 8 and 12 is 4.

To code this problem, we then just need to loop through 1-20, then take `lcm(1, 2)`, `lcm(3, lcm(1, 2))`, so on and so forth.

We could write it this way:

```
def gcd(x, y):
    if x > y: small = y
    else: small = x
    for i in range(1, small+1):
        if (x%i==0) and (y%i==0):
            gcd = i
    return gcd

def f():
    b = 1
    for i in range(1, 21):
        (b *= i) //gcd(i, b)
    return str(b)

print(f())

232792560
```

But coming up with how to compute a greatest common denominator is a real hassle... thank god we can import it!

```
import math

def f():
    b = 1
    for i in range(1, 21):
        (b *= i) // math.gcd(i, b)
    return str(b)

print(f())

232792560
```

If we didn't want to say "math.gcd", we could write

```
from math import gcd
```

and voila, the function "gcd" is added to the current namespace, so we can write:

```
def f():
    b = 1
    for i in range(1, 21):
        (b *= i) // gcd(i, b)
    return str(b)
```

or, if we wanted to import *everything* from the "math" module into our current namespace:

```
from math import *

x = pi #Works!
```



Active solve: What is the smallest number evenly divisible by the numbers 1-20? Can you try to solve it using least common multiple.

Output should be 232792560

We could even create our own modules and import them using *relative imports*. For instance, if I had the below **file structure**:

__init__.py	Jul 4, 2021 at 4:18 PM
> data	Jun 7, 2021 at 5:30 PM
> models	Today at 1:58 PM
> static	Jul 5, 2021 at 1:15 AM
> templates	Jul 5, 2021 at 1:19 AM
test.py	Jul 7, 2021 at 9:39 PM
▼ views	Today at 1:58 PM
home_bp.py	Jul 4, 2021 at 4:18 PM
liamgg_bp.py	Yesterday at 12:35 PM
liamslab_bp.py	Jul 4, 2021 at 4:18 PM
movies_bp.py	Jul 4, 2021 at 4:18 PM

And wanted to import into `__init__.py` some stuff from `home_bp.py`, I can write

```
#this code is in __init__.py
from views.home_bp import home_bp
```

This means there is something in the file `views/home_bp` called `home_bp`!

There are some other helpful modules other than `math`, like `os` (for file system interaction), `random` (random number generation), `re` (for regex matching), `sys` (interact w/ Python interpreter), and `time` (delay or measure time)

```
#check python version
import sys; print(sys.version)

#generate number 1 to 100
import random; print(random.randint(1,100))

import re
```

```

a = "ThisFileHasDigitsPleaseRemoveThem1203912491823019230"
readable_filename = ''.join(re.findall("[a-zA-Z]", a))
print(readable_filename)

#delay by 1 second
import time; time.sleep(1)

#check the location of this file
import os; print(os.path.dirname(__file__))

```

You *may* feel **bored** ☹ in this next section about looping thru. Files in a folder

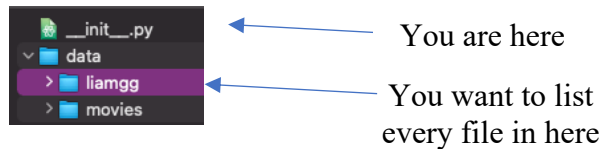
... but I *promise* it is going to help you out, knowing your comp. like a Map, a 地圖, knowing how the *hierarchies* work in any society is important .. you can look @ the hierarchy std. [here](#)

In fact, the study of file system is so old that the memes are like, from '06



Let's say I'm in `__init__.py` again, and want to print out every file in a folder that, *relative to my current file*, is in the data folder, then the liamgg folder:

Just to give you a visual on what our directory looks like:



```
import os
```

```
#__file__ is set to filepath by Python, __stuff__ in python are called  
#"dunders", short for double underscore ...
```

```
dirname = os.path.dirname(__file__)  
folder = os.path.join(dirname, 'data/liamgg')
```

```
def bytes_to_mb(bytes): return bytes*(10e-7)
```

```
for file in os.listdir(folder):  
    fpath = os.path.join(folder, file)  
    size_in_bytes = os.path.getsize(fpath)  
  
    #Round MB size to 2 significant figures  
    size_in_mb = round(bytes_to_mb(size_in_bytes, 2))  
  
    print(f'{file} is {size_in_mb} MB large!')
```

#Output:

```
SILVERI.csv is 0.32 MB large!  
SILVERIV.csv is 0.31 MB large!  
IRONIII_MatchData.csv is 2.14 MB large!  
PLATINUMI.csv is 0.33 MB large!  
PLATINUMI_MatchData.csv is 2.2 MB large!  
IRONIII.csv is 0.32 MB large!  
SILVERIV_MatchData.csv is 2.19 MB large!  
PLATINUMII_MatchData.csv is 2.22 MB large!  
IRONII.csv is 0.31 MB large!  
DIAMONDIV_MatchData.csv is 2.16 MB large!  
SILVERIII_MatchData.csv is 2.22 MB large!  
GOLDII_MatchData.csv is 2.23 MB large!  
GOLDI.csv is 0.32 MB large!  
IRONIV_MatchData.csv is 1.29 MB large!  
PLATINUMIV_MatchData.csv is 2.21 MB large!  
DIAMONDII_MatchData.csv is 2.17 MB large!  
IRONI.csv is 0.32 MB large!  
GOLDIII.csv is 0.31 MB large!  
CHALLENGERI.csv is 0.06 MB large!  
DIAMONDII.csv is 0.32 MB large!  
GOLDIV_MatchData.csv is 2.21 MB large!  
PLATINUMII.csv is 0.32 MB large!  
IRONII_MatchData.csv is 1.6 MB large!  
SILVERI_MatchData.csv is 2.22 MB large!
```

Is there a *faster* way to do this? Is there a *way to do this?*



Try this on your computer. Open Terminal. On Windows, you need a linux bash shell. Mac OS X is built on [BSD](#), from the '70s Research Unix rifts in C, and uses Bash (unless you're M1, then you have Z shell), and the commands, when compared w/ linux bash shell, are pretty alike, so that's kind of how we get you PC users and Mac users on the same page.

Go to a folder you want to get to by typing

```
cd path_to_file
```

for example,

```
cd Desktop/personalwebsite/data/liamgg
```

then type, *in Terminal*, yes, Shell scripting in this syntax looks like waves on the ocean, rocking back and forth, endless and infinite (I use talk about "the infinite" so much ...)

```
for file in *; do          hit enter
    echo $file             hit enter again
done
```

Do you see what spits out? All da files :3 in da foldersh...

Bash scripting can be *extremely useful* in certain contexts where you're manipulating the operating system a lot, particularly like issuing commands. It's not like a *this* or *that* sort of thing, it's just there are many paths to a given goal, always (hint: that is the *meaning* of my use of the word "the infinite"). Your journey does not end with Python, it just happens to start with it

Pandas

The pandas module is a very important data tool. In fact, Pandas was written on top of NumPy designed to be vectorized and designed by Jack Black for Kung Fu Panda 3's release in 2016



Excel is awesome but it is kind of slow when you have Big, 100k even (which is small) data, and you want to say count duplicates? Highlight then conditional formatting, color them red, filter col. By color then count, but this will lag. gSheets is even worse in terms of big stuff. There's uses for these apps, like Excel's pivot tables and stuff can be *very* fast in terms of generating graphs, but the operations themselves on data can be slow (you can learn VBA, which is Excel's programming language, but it is extremely annoying to use). In gSheets, you might want to connect to a gMail API to send emails with data in them automatically or something like that, too (that will all be in Google Apps Script, which is like javascript in terms of programming languages)

The python library **pandas** can be much faster at doing this, and it's much more readable (makes more sense to someone else) code-wise. This is because it uses NumPy, whose ND-arrays are *homogenous* (containing only one type of data), a restriction that allows NumPy to delegate mathematical operations on the array's contents to optimized, compiled C code (this is what "vectorization" is)

We import pandas like this

```
import pandas
```

and can give whatever alias we want to the module using *as*, e.g.:

```
import pandas as pd
import pandas as KungFuPanda
```

How convenient can this be? Let's try *reading in* some data. Files can be read and write remember, yes. So let's try grabbing a `.csv` (the file "extension" here is csv, comma-separated values). In Python writing this ourselves is going to be a real drag.

We could use python's [csv library](#) to try to do that, but that's more for just reading in rows and stuff – what if we wanted to *store* that data somehow?

Think .. what are the 5 data types we've spent so much time on? We can store as dict, tuple, set, string, list, but trying to instantiate a dict will be very annoying, and at that point we are just going from dict to [JSON](#)

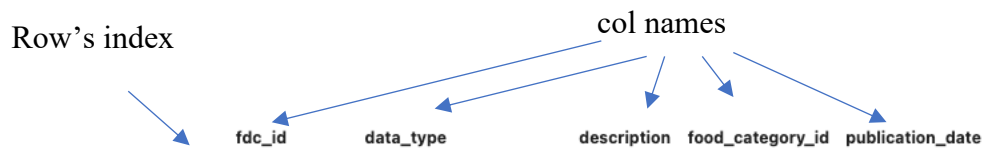
We can use pandas to do this as

```
import pandas as pd

filepath = '/Users/liamisaacs/Desktop/useless_vegetable_information/vegetables.csv'

vegetables = pd.read_csv(filepath)
```

The output of this will basically look like a table, it will look *tabular* is a way of saying it. It will look like this in a jupyter notebook:



	fdc_id	data_type	description	food_category_id	publication_date
0	319874	sample_food	HUMMUS, SABRA CLASSIC	16.0	2019-04-01
1	319875	market_acquisition	HUMMUS, SABRA CLASSIC	16.0	2019-04-01
2	319876	market_acquisition	HUMMUS, SABRA CLASSIC	16.0	2019-04-01
3	319877	sub_sample_food	Hummus	16.0	2019-04-01
4	319878	sub_sample_food	Hummus	16.0	2019-04-01
...
27588	1757386	sub_sample_food	NaN	16.0	2021-04-23
27589	1757387	sub_sample_food	NaN	16.0	2021-04-23
27590	1757388	sub_sample_food	NaN	16.0	2021-04-23
27591	1757389	sub_sample_food	NaN	16.0	2021-04-23
27592	1757390	sub_sample_food	NaN	16.0	2021-04-23

27593 rows x 5 columns

2D tabular data has rows and columns, and (since it comes from Matrix math stuff) we always say rows *by* columns

What do you think I'm gonna get when I try to guess the type of that table?

Pandas data types: DataFrames and Series

Hmmm, well let's say we wanted to make our own, we would input this sort of information:

```
import pandas as pd

vegetables = {
    'vegetable': ['Banana', 'Broccoli'],
    'liam_like_it': ['Yes', 'very much so Monsieur -- oui!']
}

table = pd.DataFrame(data=vegetables)

print(table)
```

```
vegetable      liam_like_it
0    Banana             Yes
1  Broccoli  very much so Monsieur -- oui!
[Finished in 0.695s]
```

So, we call it a *DataFrame*, because if it's a picture it sort of looks like the frame where you put the picture – boom, DataFrame. So, we tend to call stuff that's a DataFrame, *df*

```
df = pd.DataFrame(data=vegetables)

print(type(df))
```

```
<class 'pandas.core.frame.DataFrame'>
```

So, like how'd we'd usually see a 'str' print out the Python data class 'str' when we run the code `type('abc')`, we see that *df* is a DataFrame – the prefixes `pandas.core.frame` indicate the breadcrumb trail of where DataFrame is defined in the pandas library, like how we say a Book in a library is in the East Wing, Romantic Novels section

What about our 'vegetable' column, which data type is that, huh?

```
print(type(df.vegetable))
print(type(df['vegetable']))
```

```
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
```

It's a pandas *Series*! Not a *list*! Our DataFrame just looks like a dict (in fact, it's dict-like) and our Series look like lists

Let's say I'm looking at some agricultural data from the USDA, and my csv looks like this:

	fdc_id	data_type	description	food_category_id	publication_date
0	319874	sample_food	HUMMUS, SABRA CLASSIC	16.0	2019-04-01
1	319875	market_acquisition	HUMMUS, SABRA CLASSIC	16.0	2019-04-01
2	319876	market_acquisition	HUMMUS, SABRA CLASSIC	16.0	2019-04-01
3	319877	sub_sample_food	Hummus	16.0	2019-04-01
4	319878	sub_sample_food	Hummus	16.0	2019-04-01
...
27588	1757386	sub_sample_food	NaN	16.0	2021-04-23
27589	1757387	sub_sample_food	NaN	16.0	2021-04-23
27590	1757388	sub_sample_food	NaN	16.0	2021-04-23
27591	1757389	sub_sample_food	NaN	16.0	2021-04-23
27592	1757390	sub_sample_food	NaN	16.0	2021-04-23

27593 rows x 5 columns

If someone asks me – how many unique food_category_id are there? I can just

```
[62]: import os; import pandas as pd

dirname = os.path.abspath('/Users/liamisaacs/Downloads/FoodData_Central_foundation_food_csv_2021-04-28')
df = pd.read_csv(os.path.join(dirname, 'food.csv'))
df.nunique()

[62]: fdc_id          27593
      data_type        5
      description    11367
      food_category_id  18
      publication_date  6
      dtype: int64
```

What about how many rows and columns?

```
[65]: import os; import pandas as pd

dirname = os.path.abspath('/Users/liam
df = pd.read_csv(os.path.join(dirname,
df.shape

[65]: (27593, 5)
```

How many non-null in cols/rows?

```
[66]: import os; import pandas as pd

dirname = os.path.abspath('/Users/liamisaacs/Downloads/FoodData_Central_foundation
df = pd.read_csv(os.path.join(dirname, 'food.csv'))
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27593 entries, 0 to 27592
Data columns (total 5 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   fdc_id                 27593 non-null  int64
1   data_type              27593 non-null  object
2   description            27585 non-null  object
3   food_category_id       27571 non-null  float64
4   publication_date        27593 non-null  object
dtypes: float64(1), int64(1), object(3)
memory usage: 1.1+ MB
```

Homework
PS5, Due Thurs, Jul 15 12:00 AM

- (1) Read up through Merge, join, concatenate and compare on the [pandas official guide](#)
- (2) Let's say I wanted to look at data from the US Department of Agriculture. Their database is FoodData Central, you can see it [here](#). Download the "Foundation Foods" data zip folder and unzip it.
 - (i) In a Jupyter notebook and using the os module, cd to the downloaded zip folder (where ever you want to put it on your computer) and print out all files in the zip folder. (Bonus: try to write a bash script and screenshot it!) [5 Pts]
 - (ii) Take a look at "food.csv". There's an interesting "data_type" column which notes the origin of each fdc_id, for example 319874 is of the data_type "sample_food", because it's found in the sample_food spreadsheet. Can you write code that will create a column in foods.csv called "origin_spreadsheet" and provide the same functionality? [20 Pts, this is just a technical question]
 - (iii) Take a look at "market_acquisition.csv". These are foods obtained by the USDA via random sampling for chemical analysis. The store_state is the state from which the food was acquired. Let's see if it's a good balance of states. Do you think the random sampling was random? How can you use data to support your conclusion that the random sampling by the USDA was truly random? [20 Pts, this is more of a "construct a narrative" question]