Python introductory pre-class assignment

March 8, 2021

Student Name: Huynh Truong Tu

Below are my assignment to calculate the folded-43-time paper's thickness.

0.1 What I know:

- The distance to the moon is 384400 km

[23]:
```
MOON_DISTANCE = 384400 #kilometers
```

- The original thickness of the paper before folding is 0.00008m.

[24]:
```
O_THICKNESS_M = 0.00008 #Original Thickness in Meters
```

- [Problem 2] I'll use the "Unit Conversion Sample code" to converse it from m into km right from the start. Thus, we don't have to worry about it in later calculations.

[25]:
```
# Convert meters to kilometers
O_THICKNESS_KM = O_THICKNESS_M/1000
print("Thickness: {:} kilometers".format(O_THICKNESS_KM)) #Original Thickness␣
↪in Kilometers.
```

```
Thickness: 8e-08 kilometers
```

- Formular:

$$t_{43} = t_0 x 2^{43}$$

0.2 What to do:

- Thickness calculation by two methods: Exponentiation arithmetic operators & For statement.
- Comparison of calculation time.
- Saving the value after each fold to a list.
- Displaying a line graph & Customizing it.

0.3 Execution:

[Problem 1] Create using using exponentiation arithmetic operator.

[26]:
```
"""
Code to calculate the thickness when the paper is folded 43 times
"""
folded_thickness_p1 = 0_THICKNESS_KM*2**43
print("Thickness: {:.2f} kilometers".format(folded_thickness_p1))
```

```
Thickness: 703687.44 kilometers
```

[Problem 3] Create using a for statement

[27]:
```
folded_thickness_p3 = 0_THICKNESS_KM #initial folded_thickness of problem 3
for fold_time in range(1,44):
    folded_thickness_p3 *=2
print("Thickness: {:.2f} kilometers".format(folded_thickness_p3))
```

```
Thickness: 703687.44 kilometers
```

=> In comparation, the result of the two methods above is the same, and the thickness of the folded-43-time paper is more than enough to reach the moon

[Problem 4] Comparison of calculation time

[28]:
```
import time
start = time.time()
#####
# Paste the code you want to compare here
folded_thickness_p1 = 0_THICKNESS_KM*2**43
#####
elapsed_time = time.time() - start
print("time : {}[s]".format(elapsed_time))
```

```
time : 0.0[s]
```

[29]:
```
import time
start = time.time()
#####
# Paste the code you want to compare here
folded_thickness_p3 = 0_THICKNESS_KM
for fold_time in range(1,44): #it goes from
    folded_thickness_p3 *=2
#####
elapsed_time = time.time() - start
print("time : {}[s]".format(elapsed_time))
```

```
time : 0.0[s]
```

Within a few loop, the difference in time is hard to notice.

- <Development: Increase the number of repetitions>

[30]:
```
import time
start = time.time()
```

```
#####
# Paste the code you want to compare here
folded_thickness_p1 = 0_THICKNESS_KM*2**500
#####
elapsed_time = time.time() - start
print("time : {}[s]".format(elapsed_time))
```

```
time : 0.0[s]
```

[31]:
```
import time
start = time.time()
#####
# Paste the code you want to compare here
folded_thickness_p3 = 0_THICKNESS_KM
for fold_time in range(1,501):
    folded_thickness_p3 *=2
#####
elapsed_time = time.time() - start
print("time : {}[s]".format(elapsed_time))
```

```
time : 0.0[s]
```

I tried with 500 times, but the results are still hard to compare.

- <Development: Using magic commands>

[32]:
```
%%timeit
#####
# Paste the code you want to compare here
folded_thickness_p1 = 0_THICKNESS_KM*2**43
#####
```

```
69.4 ns ± 1.54 ns per loop (mean ± std. dev. of 7 runs, 10000000 loops each)
```

[33]:
```
%%timeit
#####
# Paste the code you want to compare here
folded_thickness_p3 = 0_THICKNESS_KM
for fold_time in range(1,44):
    folded_thickness_p3 *=2
#####
```

```
1.75 µs ± 97.7 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

With this, we can see the run time per loop of the for loop is longer than exponentiation arithmetic operator.

[Problem 5] Saving to a list

[43]:
```
#initial list
folded_values = []
```

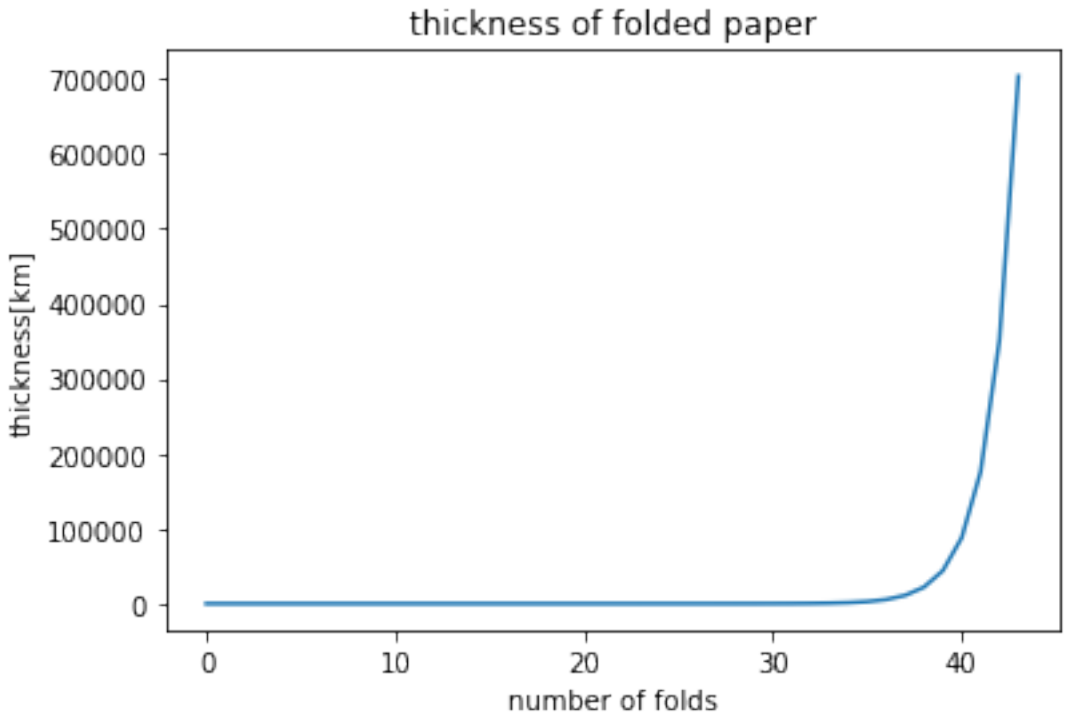
```
folded_values.append(0_THICKNESS_KM)
folded_thickness_p3 = 0_THICKNESS_KM
for fold_time in range(1,44):
    folded_thickness_p3 *=2
    folded_values.append(folded_thickness_p3)
print("Number of values in the list is:",len(folded_values)) # to confirm that␣
 ↪the list contains 44 values using the len function.
```

```
Number of values in the list is: 44
```

I've tested the results by using "print()" function, I'm confident that the folded values in the list are correct.

[Problem 6] Displaying a line graph

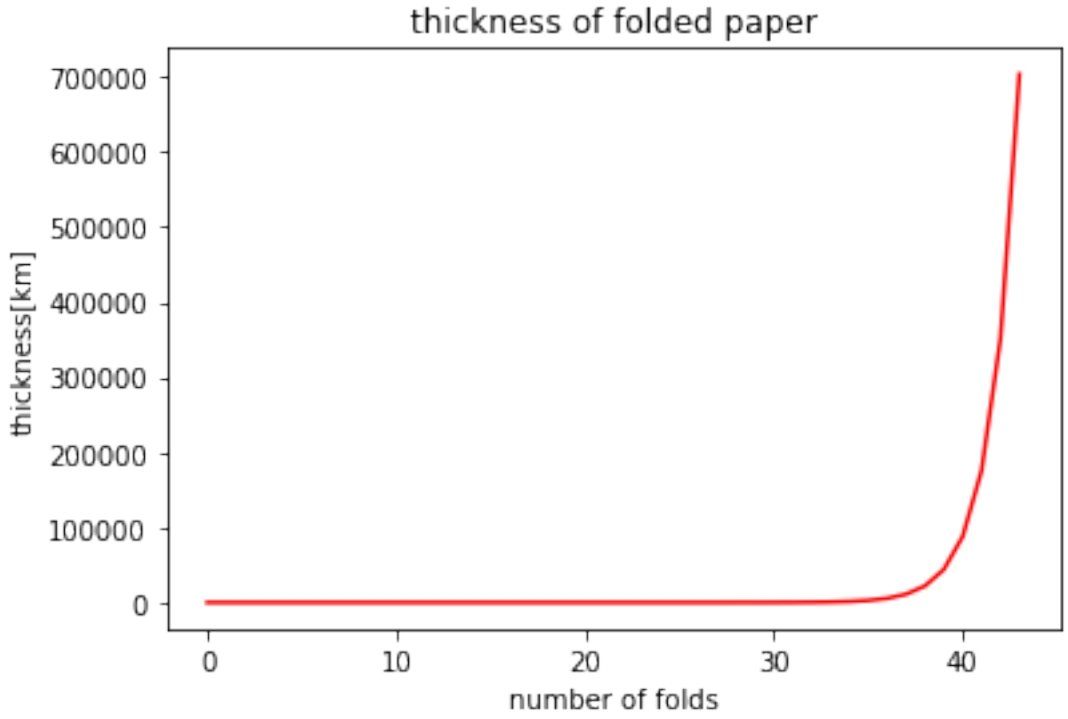
[35]:
```
"""
Display the graph. Title and axis label name.
"""
import matplotlib.pyplot as plt
%matplotlib inline
plt.title("thickness of folded paper")
plt.xlabel("number of folds")
plt.ylabel("thickness[km]")
plt.plot(folded_values) # Enter the variable name of the list in "List name"
plt.show()
```



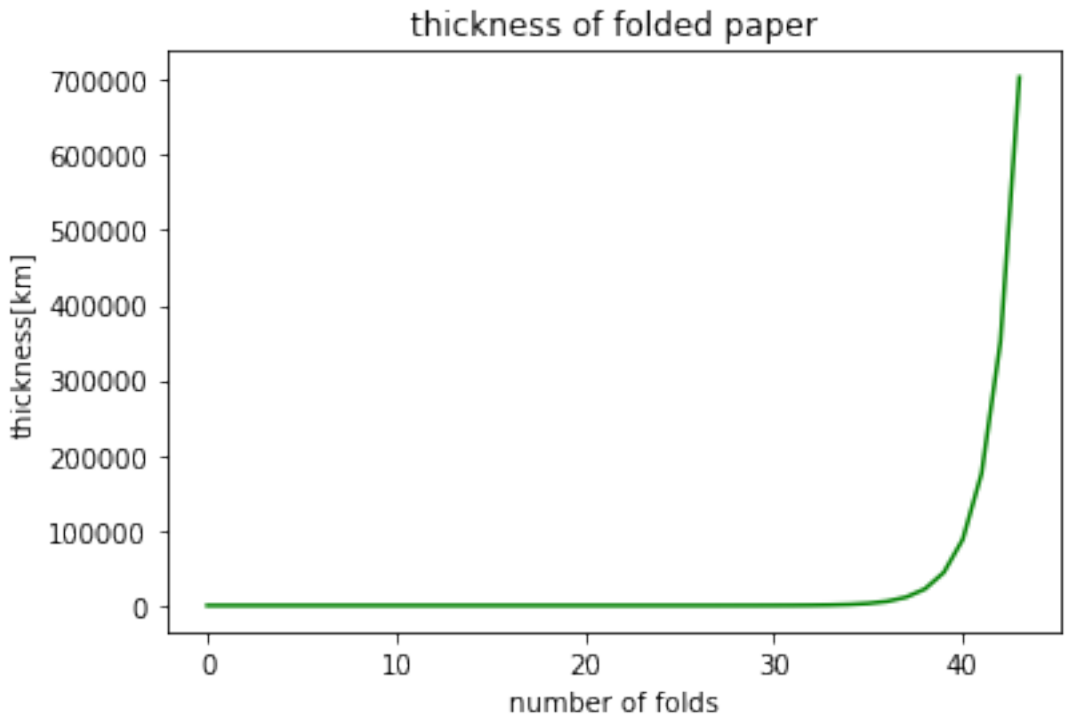
The value of the first 35 folds is barely noticable, but, as expected from exponential factor, it scales really quickly afterwards.

[Problem 7] Customizing graphs

[36]:
```
"""
Display a red line graph.
"""
plt.title("thickness of folded paper")
plt.xlabel("number of folds")
plt.ylabel("thickness[km]")
plt.plot(folded_values, color='red')
plt.show()
```

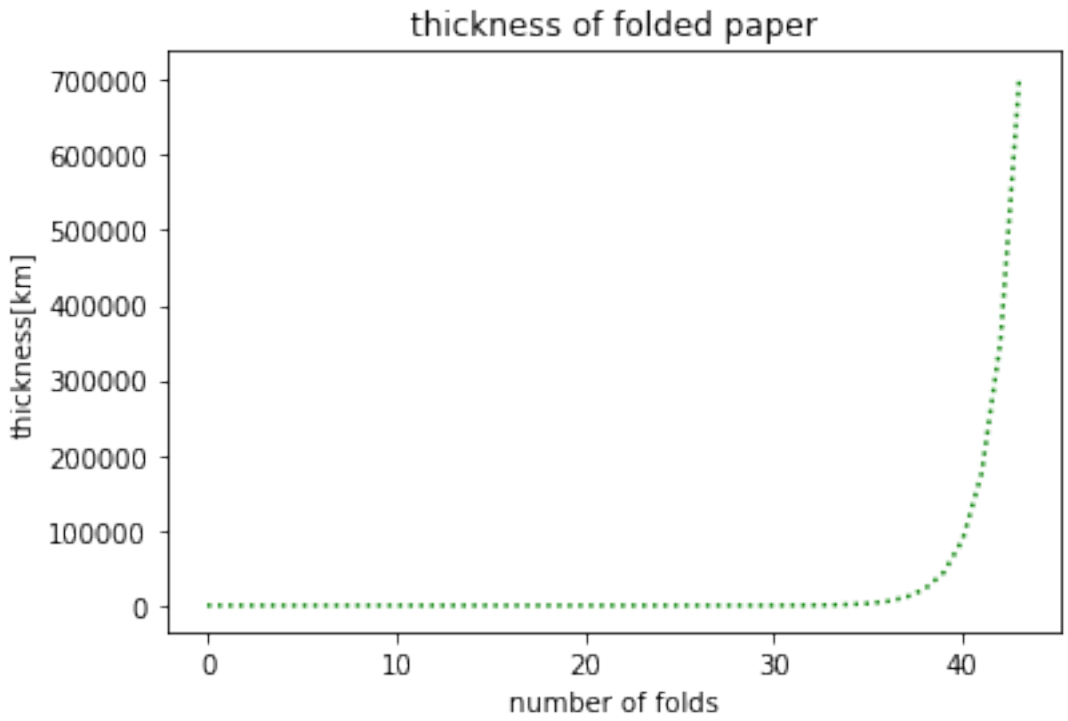


[37]:
```
"""
Display a green line graph.
"""
plt.title("thickness of folded paper")
plt.xlabel("number of folds")
plt.ylabel("thickness[km]")
plt.plot(folded_values,color='green')
plt.show()
```



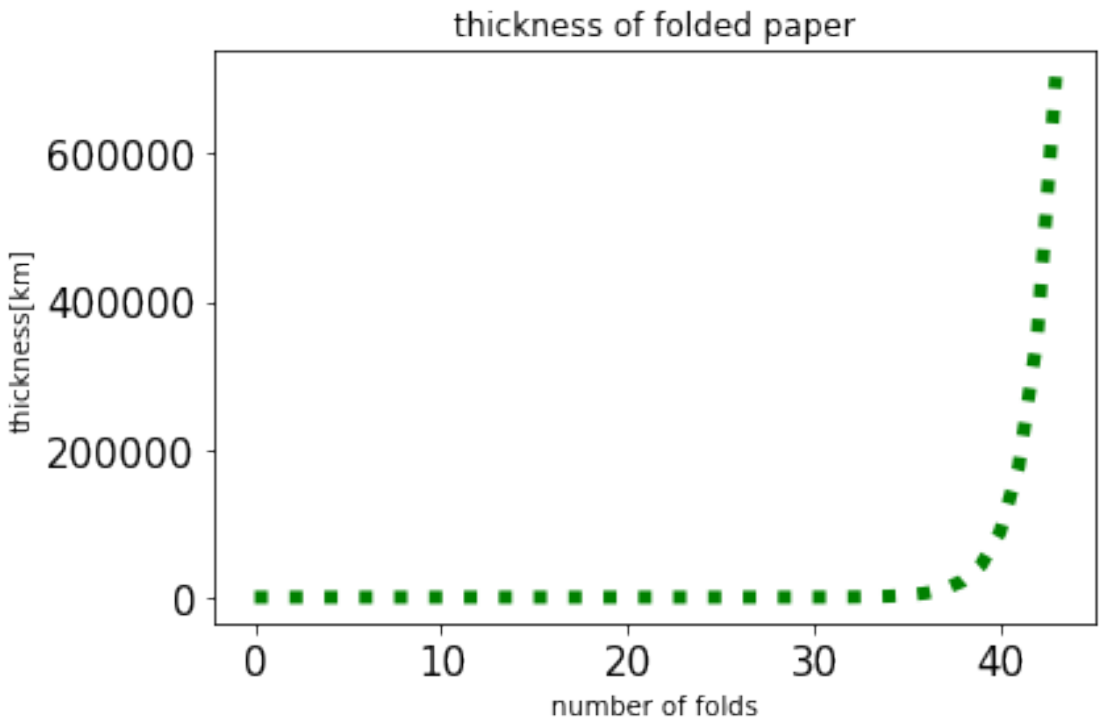
In the above graphs, I change the colour by adding specific color parameter in "plt.show()" function.

[38]:
```
"""
Display a green dotted line graph.
"""
plt.title("thickness of folded paper")
plt.xlabel("number of folds")
plt.ylabel("thickness[km]")
plt.plot(folded_values, ls=':',color='green')
plt.show()
```



In this graph, I change the line into dot by specify line style parameter in "plt.plot()" function.

[39]:
```
"""
Display a thicker green dotted line graph.
"""
plt.title("thickness of folded paper")
plt.xlabel("number of folds")
plt.ylabel("thickness[km]")
plt.tick_params(labelsize=15)
plt.plot(folded_values, ls=':',color='green', linewidth = 5)
plt.show()
```



Finaly, for this graph, I make the line thicker and incease the label font size.

==========

This is the end of my assignment, thank you for reading!

[ ]: