Computation, Optimization, and Intellectual Exploration

L03: Programming Revision 3

Faculty of Engineering, Khon Kaen University

Submission: https://autolab.en.kku.ac.th

==========

Instructions

* Submit an answer to a question with a file with `txt` extension. E.g., an answer for Q1 should be submitted in a text file "`Q1.txt`".

* Submit a program to a (programming) problem with a file with a proper extension. E.g., a python program for P2 should be named "`P2.py`".

* Autograder is run on a docker image with Python 3.5.

* Grader tests student's answers with tolerance 0.001.

==========

P1. Cosine similarity is a similarity measure, widely used in document retrieval. Write a function named `cos_sim` to take 2 arguments each as a list of numbers, calculate cosine, and return the number.

Cosine similarity:

$$s = \frac{\sum_{i=1}^{N} a_i \cdot b_i}{\sqrt{\sum_{i=1}^{N} a_i^2} \cdot \sqrt{\sum_{i=1}^{N} b_i^2}},$$

where s is a similarity score; a_i and b_i are components of vectors representing documents A and B, respectively; and N is a vector length.

Hint: `math.sqrt` may be handy.

Invocation example

When the function is invoked as follows:

```
cs = cos_sim([1, 0], [5, 5])
print(cs)
```

We will see the outcome:

```
0.7071067811865475
```

When the function is invoked as follows:

```
cs = cos_sim([14, 0, 5], [5, 8, 4])
print(cs)
```

We will see the outcome:

```
0.5908152858583454
```

P2. In photography, aperture, shutter speed, and ISO are variables to control image exposure. A larger aperture allows more exposure. So does a slower shutter speed. A larger ISO also does a similar effect, but with resulting a grainier image. In addition, they all affect other aspects, e.g., depth of field (how farther away from camera an object still appear sharp in the image), overall image sharpness, sharpness of moving objects, overall brightness of an image, etc.

Apertures may come in f/1.4, f/2, f/2.8, f/4, f/5.6, f/8, f/11, f/16, f/22. Since the number comes with f/, it is called *f-number*. Long story short: smaller f-number, larger aperture. For each step of the f-number goes up, the size of an aperture area goes down by half. Notice that f-number goes up by a squared root of 2. See the illustration below. Shutter speed may come in 1/4, 1/8, 1/15, 1/30,

1/60, 1/125, 1/250, 1/500, 1/1000, 1/2000, 1/4000. The numbers go from the slowest to the fastest, 1/4 is the slowest and 1/4000 is the fastest. Each step gets roughly twice as fast. ISO is the sensitivity of the sensor. The higher number, the more sensitive (picking up light better). ISO may come in 100, 200, 400, 800, 1600, 3200. ISO 100 is the least sensitive. ISO 200 is double sensitive of ISO 100, and so on.

Given Aperture in f/1.4, f/2, f/2.8, f/4, f/5.6, f/8, f/11, f/16, f/22; Shutter speed in 1/4, 1/8, 1/15, 1/30, 1/60, 1/125, 1/250, 1/500, 1/1000, 1/2000, 1/4000, and ISO in 100, 200, 400, 800, 1600, 3200, write a function named `cam_expos` to take a "neutral setting" as a list of [f-number, shutter speed, and ISO] and a photographer's setting, also as a list of [f-number, shutter speed, and ISO], find out the difference in each variable, as well as the overall exposure, and return the finding in tuple. For example, camera may suggest [f/2.8, 1/500, 400]. A photographer chooses [f/5.6, 1/125, 200]. Then, the photographer choice is –2 +2 –1 = –1 (or 1 stop under neutral exposure).

Hint: check out list methods; they could be handy.

Invocation example

When the function is invoked as follows:

```
res = cam_expos(['f/2.8', '1/500', '400'], ['f/5.6', '1/125', '200'])
print(res)
```

We will see the outcome:

```
(-2, 2, -1, -1)
```

When the function is invoked as follows:

```
res = cam_expos(['f/2.8', '1/500', '400'], ['f/1.4', '1/60', '100'])
print(res)
```

We will see the outcome:

```
(2, 3, -2, 3)
```

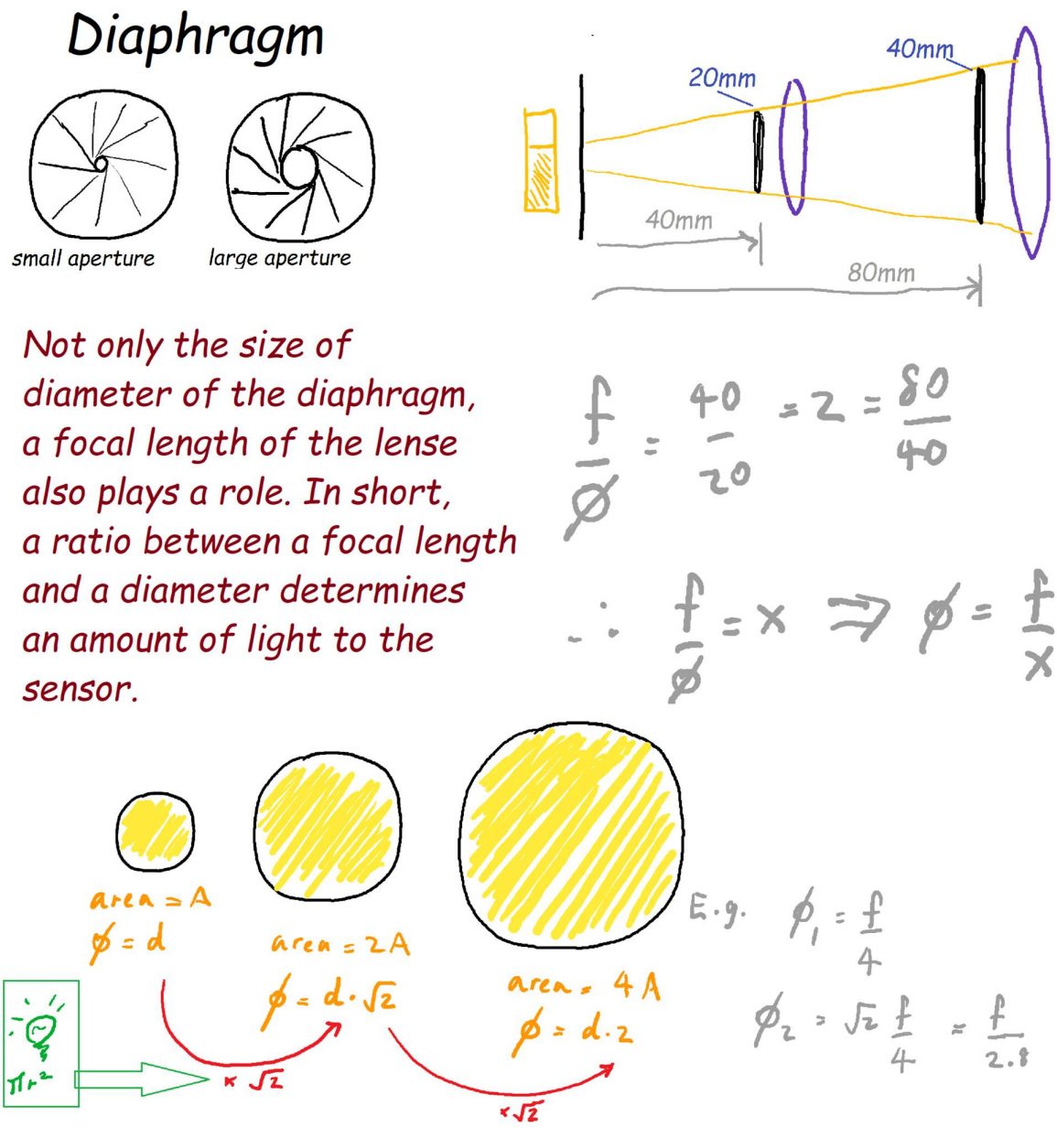


Figure 1

P3. Write a function named `word_freq` to take a string as its argument, count occurrences of each word, and return the count as a Python dict. Discard any word of length 1 or less, e.g., *"Our class is a programming class."* should be counted to: `{'Our': 1, 'class': 2, 'is': 1, 'programming': 1}`. There will be no count for `'a'` (length of '*a*' is 1).

Invocation example

When the function is called as follows:

```
txt = "Evil is done by oneself; " + \
        "by oneself is one defiled. \n " + \
        "Evil is left undone by oneself; " + \
        "by oneself is one cleansed. \n " + \
        "Purity and impurity are one's own doing. \n" + \
        "No one purifies another. \n" + \
        "No other purifies one."
        # excerpt from Attavagga: Self, www.accesstoinsight.org
wf = word_freq(txt)
print('Count:')
print(wf)
```

We will see the outcome:

```
Count:
{'doing': 1, 'another': 1, 'purifies': 2, 'own': 1, 'undone': 1, 'is':
4, 'other': 1, 'impurity': 1, 'left': 1, 'by': 4, 'oneself': 4, 'No': 2,
'Purity': 1, 'one': 4, 'cleansed': 1, 'and': 1, 'are': 1, 'done': 1,
"one's": 1, 'Evil': 2, 'defiled': 1}
```

Note: punctuation has been cleaned off the keys, i.e., no key has a period, comma, or semicolon. But upper case, plurality, and tense stay.

Use `P3_template.py`. (The template is only to allow smooth autograding. You should edit and rename it properly.)

P4. Baleen whales, e.g., gray whales, humpback whales, blue whales, feed on krill, plankton, and tiny marine organisms. A blue whale can eat as much as 40 million krill a day, approximately 3600 kg of krill. Krill are tiny shrimps found in oceans.

Given a dictionary of baleen whales and their average daily diet on krill during feeding season and a dictionary of whale counts observed during a field trip, write a function named `krill_consumption` to estimate the total amount of krill consumed daily by the observed whales during feeding season.

Invocation example 1

When the function is called as follows:

```
feeding = {'Humpback whale': 2000, 'Gray whale': 1500, 'Bowhead whale':
2500, 'Blue whale': 3600}
```

```
whales = {'Humpback whale': 8, 'Gray whale': 3, 'Bowhead whale': 1,
'Blue whale': 12}

total_consum = krill_consumption(feeding, whales)
print('Estimate daily consumption: %d kg of krill'%total_consum)
```

We will see the outcome:

```
Estimate daily consumption: 66200 kg of krill
```

Invocation example 2

When the function is called as follows:

```
feeding = {'Humpback whale': 2000, 'Gray whale': 1500, 'Bowhead whale':
2500, 'Blue whale': 3600}
whales = {'Humpback whale': 8, 'Gray whale': 3}

total_consum = krill_consumption(feeding, whales)
print('Estimate daily consumption: %d kg of krill'%total_consum)
```

We will see the outcome:

```
Estimate daily consumption: 20500 kg of krill
```

Invocation example 3

When the function is called as follows:

```
feeding = {'Humpback whale': 2000, 'Gray whale': 1500, 'Bowhead whale':
2500, 'Blue whale': 3600}
whales = {'Bowhead whale': 80000}

total_consum = krill_consumption(feeding, whales)
print('Estimate daily consumption: %d kg of krill'%total_consum)
```

We will see the outcome:

```
Estimate daily consumption: 200000000 kg of krill
```

Note: given a shorter dictionary of whale counts, the function still works fine.

P5. A kinetic power of flowing water can be estimated from

$$P = \frac{1}{2A^2}\,\rho Q^3,$$

where P is a kinetic power (in w), A is an estimate cross-section area (in m^2), ρ is a water density (in kg/m^3), and Q is a flow rate (in m^3/s).

Write a function named `mighty_river` to take in an argument: a dictionary of river information (having a key being a river name and a value being a list of a cross-section area, water density, and a flow rate), calculate the estimated power, and return the result in another dictionary using a river name as a key and calculated power as a value.

Invocation example 3

When the function is called as follows:

```
mighties = {'Amazon': [1.2e6, 1100, 210000],
                  'Congo':  [2e6, 1150, 41200],
                  'Yangtze': [800e3, 1200, 30000]}

river_power = mighty_river(mighties)
print(river_power)
```

We will see the outcome:

```
{'Amazon': 3537187.5, 'Congo': 10053.0884, 'Yangtze': 25312.5}
```

Use `P5_template.py`. (The template is only to allow smooth autograding. You should edit and rename it properly.)

P6. Game theory is a study of social interaction among rational decision makers in such a way that a decision of any decision maker affects the others.

For example (Prisoner's Dilemma), Lobha and Raga were arrested for theft. The police tries to get them to confess so each of them is offered a deal. If both do not confess, both will get 3-year sentence. However, if Lobha confesses and Raga does not, Lobha will get 1-year sentence and Raga will get 10-year sentence. Vice versa, if Lobha does not confess, but Raga does, Lobha will serve 10 years in prison while Raga will only have to do 1 year. If both confess, both will get 5-year sentence. Table below summarizes the 4 possible scenarios

Lobha \ Raga	Not confess	Confess
Not confess	3/3	10/1
Confess	1/10	5/5

It is clearly that the mutual interest is maximized when both do not confess. However, Lobha and Raga cannot trust one another. The best choice of Lobha

regardless of Raga decision is to confess and so is Raga's. These rational decisions of both players can be settled and this state is called Nash equilibrium.

Write a function named `rational_decision`. The function takes 2 arguments: (1) information table of the 4 scenarios as a dictionary and (2) a name of a person the function finding the decision for. The function returns a rational decision of the given person regardless of the other's decision or `None` if there is no such an option. The information is represented as a dictionary with format:

`{'Lobha': [[3, 10], [1, 5]], 'Raga': [[3, 10], [1, 5]]}` each dict value is a list of 4 entries each of a number of years serving in prison for the corresponding scenario, i.e, in order, a person does not confess and the other does not confess, a person not confess but the other does, a person confesses but the other does not, and a person confesses and the other does too.

Invocation example

When the function is called as follows:

```
choices = ['not confess', 'confess']

s = {'Lobha': [[3, 10], [1, 5]], 'Raga': [[3, 10], [1, 5]]}

p = 'Lobha'
r = rational_decision(s, p)
print(p, ':', choices[r])
```

We will see the outcome:

```
Lobha : confess
```

P7. Around 2nd century BCE, (1) Greek astronomer Hipparchus uses geometry to estimate a distance to the Moon. (2) Given the distance to the moon, on the day of half-lit moon another Greek astronomer Aristarchus also uses geometry and the observing angle to the sun to derive the distance to the sun.

Src: Morgan Rehnberg, How did we find the distance to the sun? in Universe Today, Jan 5, 2015. https://phys.org/news/2015-01-distance-sun.html

Given the distance between 2 cities d and the moon angle θ, Hipparchus can use simple geometry to compute the distance to the moon. Hipparchus uses noticeable moon features to help estimate the angle.

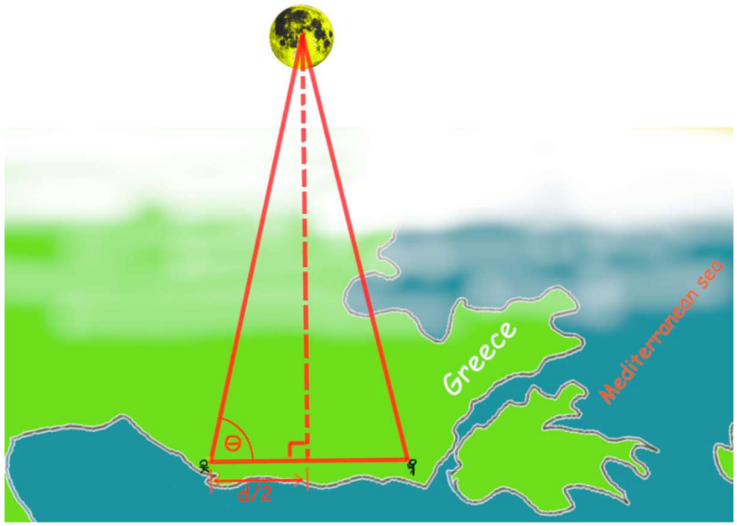


Figure 2. Hipparchus measuring the distance to the moon.

Distance to the moon, $D = d/2 \cdot \tan \theta$.

E.g., suppose d = 400 km and θ = 89.97°, then the computed distance D = 381,971.8 km.

Note the current acceptable number is around 384,400 million km. The actual number estimated by Hipparchus is little far off than what is illustrated here. This technique is handy. It is called parallax and is still in use to estimate a distance of a celestial object.

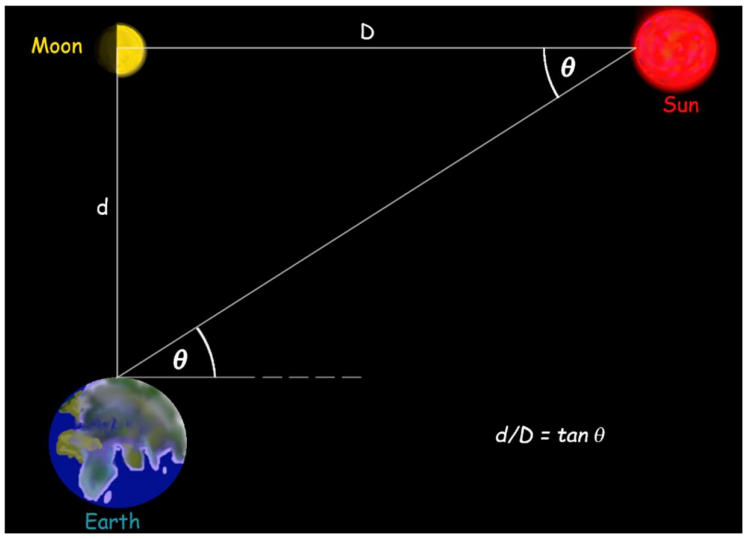


Figure 3. At half-lit moon, Aristarchus computes the distance to the sun.

Given the distance to the moon d and the angle to the sun from the horizon θ, Aristarchus uses geometry to estimate the distance to the sun D.

That is, $D = d / \tan \theta$.

E.g., suppose 381,971.8 km was a distance to the moon Hipparchus concluded and the observing angle was measured to be 0.15°, then the computed distance D = 381,971.8 / tan (0.15°) ≈ 145,902,147 km.

Note the current acceptable number is around 147.28 million km. The actual number estimated by Aristarchus is far off than what is illustrated here.

Write functions `Hipparchus` and `Aristarchus`. Function `Hipparchus` takes in distance between two observing points d_o (in km) and the moon observing angle θ_m (in degree) then returns an estimate distance to the moon D_m (in km) along with its possible range if his measure is off by 0.01 degree, all in a tuple of three numbers: D_m, D_{ml}, and D_{mu}. The range bound is presented such that $D_{ml} < D_{mu}$.

Function `Aristarchus` takes in the estimated distance to the moon d_m (in km) and the sun observing angle θ_s (in degree), then returns and returns an estimate distance to the sun D_s (in km) along with its possible range if his measure if off by 0.01 degree, all in a tuple of three numbers: D_s, D_{sl}, and D_{su}. The range bound is presented such that $D_{sl} < D_{su}$.

Invocation example

When the function is called as follows:

```
res = Hipparchus(400, 89.97)
print('Hipparchus:{:,.2f} km; [{:.0f} , {:.0f}] if 0.01 deg off'.format(*res))

res = Aristarchus(381972, 0.15)
print('Aristarchus: {:,.2f} km; [{:.0f} , {:.0f}] if 0.01 deg off'.format(*res))
```

We will see the outcome:

```
Hipparchus: 381,971.83 km; [286479 , 572958] if 0.01 deg off
Aristarchus: 145,902,223.28 km; [136783291 , 156323857] if 0.01 deg off
```

Notice that a range `[136783291 , 156323857]` is derived by computing at 0.14 and 0.16 degrees, then sort them such that a shorter distance comes first.

P8. Our gene information contains in every of our cells. Genes are coded with DNA. With DNA transcription and translation processes, our genetic code is used to guide the protein synthesis. Proteins are macromolecules serving various functions within organisms, including being parts of cells, catalyzing metabolisms as enzymes, being reactants in the metabolism, etc. Protein is

essential to living organisms and it is a sequence of amino acids. Different sequences of amino acids define different types of proteins. DNA is a molecule composed of two strands of complementary pairs of nucleotide bases. There are 4 types of nucleotide bases, i.e., cytosine (C), guanine (G), adenine (A), or thymine (T). A sequence of the bases carries genetic information for protein synthesis, such that 3 bases are translated to 1 of 20 amino acids composing to be a protein (or a stop signal, indicating the end of sequence). A 3-base set is called a codon. Therefore, "given a codon table", protein---a sequence of amino acids---- can be synthesized from genetic DNA code.

Note: codons are read in succession and not overlap each other. (See https://www.nature.com/scitable/definition/codon-155 for details)

Write a function named codon. The function takes 2 arguments: a file name of a codon table (as string) and a file name of a DNA sequence, reads the two files, uses a codon table to decipher the DNA sequence to a protein, and returns a list of the deciphered sequence of amino acids. The codon table file is a simple text file, written in a simple format: each line contains one codon and its corresponding amino acid with "=" in between. The gene sequence is a text file with the first line being a source URL and gene sequence is written from the second line on. Each line contains no more than 70 bases. (Note: fun gene data can be downloaded from NCBI GenBank, looking for download "FASTA")

Hint: Codon is a non-overlapping string and it goes by 3 bases, i.e., `range(0, N, 3)` may be handy.

Invocation example

When the function is called as follows:

```
res = codon('codons.txt', 'homo_sapiens_mitochondrion.txt')
print(res)
print(len(res))
```

We will see the outcome:

```
['Lysine', 'Glycine', 'Leucine', 'Alanine', 'stop', 'Leucine', 'Lysine',
'Tryptophan', 'Leucine', 'Isoleucine', 'Cysteine', 'Valine', 'Glutamine',
'Leucine', 'Methionine', 'Glutamine', 'Serine', 'Glycine', 'Valine',
'Leucine', 'Glutamine', 'Serine', 'Leucine']
23
```

See `codons.txt` and `homo_sapiens_mitochondrion.txt` for their content, c.f. the results shown. It is recommended that you also test your code with `homo_sapiens_insulin.txt`, which is longer and has multiple lines.

Use `P8_template.py`. (The template is only to allow smooth autograding. You should edit and rename it properly.)

P9. Sweetness is often considered to be the most desirable taste. Sweetness is measured as perception comparable to sucrose (table sugar). For example, sucrose has sweetness of 1, fructose (found in fruits) is sweeter than sucrose has sweetness of 1.7. This is measured by perception test where tasters taste the solution compared to sucrose solution and a 10% fructose solution (10g of fructose filled up with water to 100mL) is on average perceived as sweet as a 17% sucrose solution (17g of sucrose filled up with water to 100mL). Soft drink is made up to sweetness of 10% sucrose solution. Getting to around 15% sucrose solution makes the drink tasted more like syrup.

Write a function named `sweet` to take 2 filenames: one is for a sweetness table and another one is for sweeteners in a drink. The function calculates estimated sweetness of the drink comparable to 10% sucrose solution, and appends its calculation to the end of the second file---the drink-sweetener file. The sweetness file has its first line as a header. Then from the second line on, each line contains a substance name, its sweetness level, and its calories (we don't use calories here). The drink-sweetener file does not have a header. Each line in the drink-sweetener file contains a substance name and its weight (in gram). All substance weights are in solution of 100mL. Note: the calculation is similar to weighted sum. For example, given a cup of 100mL drink contains 5g sucrose and 5g fructose. For sucrose sweetness of 1 and fructose of 1.7, this solution is approximately equivalent to $5 \cdot 1 + 5 \cdot 1.7 = 13.5$. That is, this drink is as sweet as 13.5% sucrose solution and the word "Sweet as 13.5% sucrose solution" should be appended to the end of the drink-sweetener file.

Invocation example

Given `sweetness1.txt`

sweetness1.txt

```
substance sweetness calories/gram
lactose 0.16 4
glucose 0.7 4
sucrose 1 4
fructose 1.7 4
saccharin 300 0
```

and `CocaPanda.txt`

CocaPanda.txt

```
glucose 3
sucrose 3
fructose 3
```

When the function is called as follows:

```
sweet('sweetness1.txt', 'CocaPanda.txt')
```

The outcome is that the `CocaPanda.txt` will be updated. The updated content will look as follows.

CocaPanda.txt (updated after function call)

```
glucose 3
sucrose 3
fructose 3
Sweet as 10.2% sucrose solution
```

Use `P9_template.py`. (The template is only to allow smooth autograding. You should edit and rename it properly.)

P10. "Fire is a rapid oxidation of a material in the exothermic chemical process of combustion, releasing heat, light, and various reaction products." from *Wikipedia: Fire*. Combustion is a chemical reaction between fuel and an oxidant, like oxygen. It takes an activation energy to break the covalence bonds of the reactants and then after the products are formed, new covalence bonds are made and energy are released. The entire process releases energy more than it took, therefore it is exothermic. For example, burning methane

$$CH_4 + 2\ O_2 \rightarrow CO_2 + 2\ H_2O$$

requires an activation energy to break 4 `C-H` bonds and 2 `O=O` bonds and it releases energy from forming 2 `C=O` bonds and 4 `O-H` bonds in the products. Given `C-H` bond energy of 416 kJ, `O=O` bond energy of 498 kJ, `C=O` bond energy of 803 kJ, and `O-H` bond energy of 467 kJ, the activation energy is $4 \times 416 + 2 \times 498 = 2660$ kJ and the energy released is $2 \times 803 + 4 \times 467 = 3474$ kJ, which gives the difference of -814kJ.

Write a function named `fire`. The function takes 2 filenames: one is for covalence bond energies and another is for the reaction specifying amounts of reactant and product bonds. The function reads both files for necessary information, then calculates activation energy, releasing energy, and the different energy and appends the result at the end of the second file---the reaction-bond file. The bond-energy file contains bonds and their corresponding energies: each line has bond symbol and its energy (in kJ/mol), separated by a space. The reaction file has the first line describing the reaction (we can ignore it). The second line specifies numbers of reactant bonds (required to break) in format: a number of bonds, bond symbol, '+', a number of bonds, bond symbol, and so on. The '+' sign is to separate numbers and bond symbols into pairs. The third line specifies numbers of reactant bonds (required to form) in a similar format to the second line. Note that a number of bonds can be floating point, e.g., octane burning $C_8H_{18} + 12.5\ O_2 \rightarrow 8\ CO_2 + 9\ H_2O$.

Invocation example

Given `bond_energy.txt`

bond_energy.txt

```
C-C 346
C=C 835
C-O 358
C=O 803
C-H 416
O-O 142
O=O 498
O-H 467
H-H 432
```

and `methane.txt`

methane.txt

```
CH4 + 2 O2 -> CO2 + 2 H2O
4 C-H + 2 O=O
2 C=O + 4 O-H
```

When the function is called as follows:

```
fire('bond_energy.txt', 'methane.txt')
```

The outcome is that the `methane.txt` will be updated. The updated content will look as follows.

methane.txt (updated after function call)

```
CH4 + 2 O2 -> CO2 + 2 H2O
```

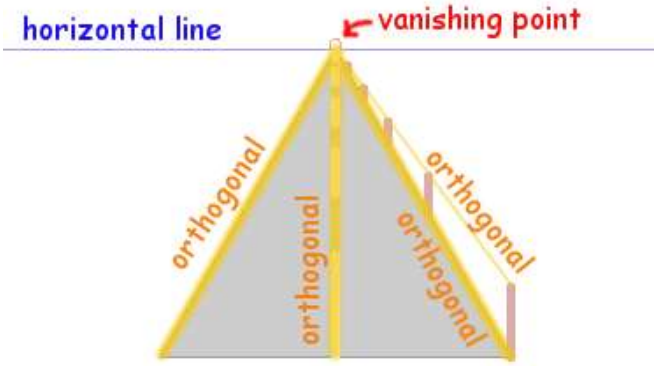
```
4 C-H + 2 O=O
2 C=O + 4 O-H
Ea = 2,660.0 kJ, Er = 3,474.0 kJ, E = -814.0 kJ
```

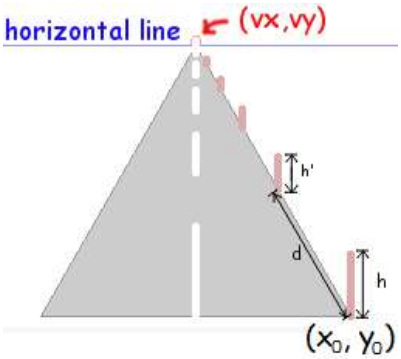
Use `P10_template.py`. (The template is only to allow smooth autograding. You should edit and rename it properly.) It is recommend that you also try your code on `octane.txt`.

P11. [Difficulty: 5 stars] Linear perspective is a visual technique to create an illusion of depth on a 2D screen. The technique is believed to be developed by a renaissance architect and artist Filippo Brunelleschi. The technique has main 3 components: parallel lines (called *orthogonals*), the horizontal line, and a vanishing point. In short, as an object appears smaller as it locates further away, parallel lines---like a rail way--- seem to converge at one point as they go further away from a viewer. That point to where lines seem to converge is called a *vanishing point*. Any parallel lines going toward a vanishing point are then drawn tilted toward the vanishing point. A length of any line perpendicular to the orthogonals is shorten accordingly. Note that the vanishing point is usually the point toward where artists intentionally lead viewer's eyes.

Write a function named `brunelleschi_lamp` to calculate a height of a lamp post on a road-lamp-post scene. The function takes vanishing point coordinates (*vx, vy*), a reference lamp post coordinates (x_0, y_0), height of the reference lamp post *h*, a projected distance *d* between the reference lamp post and the lamp post under question, and a log filename, then calculates the height of the lamp post under question *h'* according to linear perspective and return it out as well as **append** the calculation to a log file.

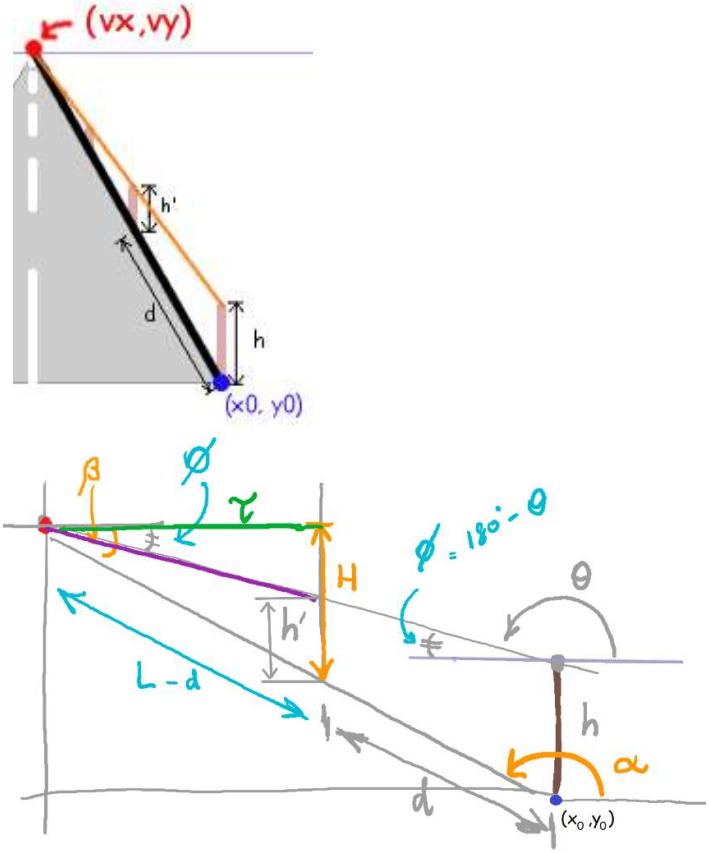
A road-lamp-post scene for this function is intended may look like the following illustrations.





When we consider (vx, vy), (x_0, y_0), d, and h, the height under question h' can be deduced from geometry. The angle α pointing from reference (x_0, y_0) toward the vanishing point (vx, vy) and the length L between two points can be easily obtained: function `find_pol` of module `mama_turtle` takes care of this.

Then, with the angle β (= π – α) and length L (actually $L - d$), H and τ can be deduced: $H = (L - d)\ sin\ \beta$ and $\tau = (L - d)\ cos\ \beta$. Similarly, an angle θ pointing from top of the reference post (x_0, y_0+h) toward the vanishing point can be obtained (function `find_pol` again). Given H, τ, and angle ϕ = π – θ, h' can be obtained from $(H - h')/\tau = \tan\ \phi$.



Invocation example 1

When the function is called as follows:

```
    vx, vy = 0, 75
    x, y = 100, -100
    height = 40
    d = 0
    res = brunelleschi_lamp(vx, vy, x, y, height, d)
    print('Ans =', res)
```

We will see the outcome:

```
Ans = 40.00000000000014
```

Invocation example 2

When the function is called as follows:

```
from mama_turtle import road
if __name__ == '__main__':
    r = road()
    r.draw_road()
    r.draw_lamp_posts(brunelleschi_lamp)
    input('enter to exit')
```

Notice that `brunelleschi_lamp` function is passed to `draw_lamp_posts` method of `road` object imported from `mama_turtle` module (auxiliary file). The resulting `log.txt` (freshly created) may look like

```
Input: vx=0.00, vy=75.00, x=100.00, y=-100.00, height=40.00, d=0.00.
Calc: L = 201.56, beta = 1.05, H = 175.00, tau = 100.00, phi = 0.93, h' = 40.00.
Input: vx=0.00, vy=75.00, x=100.00, y=-100.00, height=40.00, d=94.06.
Calc: L = 201.56, beta = 1.05, H = 93.33, tau = 53.33, phi = 0.93, h' = 21.33.
Input: vx=0.00, vy=75.00, x=100.00, y=-100.00, height=40.00, d=141.09.
Calc: L = 201.56, beta = 1.05, H = 52.50, tau = 30.00, phi = 0.93, h' = 12.00.
Input: vx=0.00, vy=75.00, x=100.00, y=-100.00, height=40.00, d=169.31.
Calc: L = 201.56, beta = 1.05, H = 28.00, tau = 16.00, phi = 0.93, h' = 6.40.
Input: vx=0.00, vy=75.00, x=100.00, y=-100.00, height=40.00, d=188.12.
Calc: L = 201.56, beta = 1.05, H = 11.67, tau = 6.67, phi = 0.93, h' = 2.67.
```

This is due to that `draw_lamp_posts` method calls `brunelleschi_lamp` function for 5 times. (Recall: it is opening a file in an append mode.)

Use `P11_template.py` and auxiliary file `mama_turtle.py` (The template and auxiliary files are only to allow smooth autograding and better visualization. You should use them properly.)

Note: this linear perspective is more like a “Rube Goldberg machine” way of doing linear perspective. Later in the program, you will learn ray tracing in

computer graphics course, which is a much more general and less complicated approach.

P12. [Difficulty: 5+ stars] Travel is about learning other cultures, other places, other perspectives. Society and humanity are always at the center of this learning. To process this kind of non-structure data (i.e., it is a description, not a tabular data; some calls this natural language processing or NLP), various schemes have been invented to transform this non-structure-data message into some kind of structure data, preferably computable as well. Among those, TF-IDF is one of the most widely used. TF-IDF (term frequency-inverse document frequency) is a measure of word importance in documents. It is designed to signify specific words uniquely appeared in some documents as well as tone down common words commonly appeared in all documents.

Write a function named `culture_tf_idf`. The function takes an argument: a list of filenames each describing a culture. The function goes through all files in the specific path, computes TD-IDF for every file and every word in it, and return a nested dictionary of the results. The resulting dictionary is nested in a way that the first level key is a document name and the second level key is a word. Note: use the following TF-IDF formulation for this problem:

$$tfidf = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \cdot log\left(\frac{N}{1+N_t}\right),$$

where *tfidf* is a TF-IDF measure of term *t* in document *d*; $f_{t,d}$ is a frequency term *t* appeared in document *d*; *N* is a number of all documents; and N_t is a number of documents having term *t*. The first half of the product, $\frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$ is term frequency or *tf*. It is to quantify how often a term appears in the document. The second half of the product $log\left(\frac{N}{1+N_t}\right)$ is inverse document frequency or *idf*, which discounts the term if it is common in many document. For example, given 3 documents: A, B, C, where A has 'stupid is stupid does'; B has 'real strength is within' and C has 'wisdom is power', we will have

$f_{stupid,\ A} = 2$; $f_{is,\ A} = 1$, $f_{does,\ A} = 1$.

$f_{real,\ B} = 1$; $f_{strength,\ B} = 1$, $f_{is,B} = 1$, $f_{within,\ B} = 1$.

$f_{wisdom,\ C} = 1$, $f_{is,C} = 1$, $f_{power,C} = 1$.

N = 3 (for A, B, and C).

N_t: $N_{stupid} = 1$ (for A having stupid), $N_{is} = 3$ (for A, B, C all having it), $N_{does} = 1$, $N_{real} = 1$, ..., $N_{power} = 1$.

Therefore, $\text{tf-idf}_{stupid,A} = 2/(2 + 1 + 1) \cdot \log(\ 3/(1 + 1)\ ) = 0.2027$;

$\text{tf-idf}_{is,A} = 1/4 \cdot \log(\ 3/(1 + 3)\ ) = -0.0719$;

... $\text{tf-idf}_{power,C} = 1/3 \cdot \log(\ 3/(1 + 1)\ ) = 0.1352$

Hint:

(1) find $f_{t,d}$ first (using a nested dictionaries for $f_{t,d}$ could make things easier);
(2) find N and N_t (using a dictionary with keys being words for N_t could ease the task);
(3) find $\sum_{t' \in d} f_{t',d}$ before calculating TF.

Invocation example

Given `chinese.txt`

chinese.txt

```
China
Chinese speaks loudly.
They associate loudly speaking with good health.
They value good health and longevity highly.
```

`thai.txt`

thai.txt

```
Thai
Thai speaks softly.
They associate soft speaking with good manner.
They value fun and good food.
```

`japanese.txt`

japanese.txt

```
Japan
Japanese is polite.
They speaks formally.
They value responsibility and politeness.
```

`western.txt`

western.txt

```
Western
Westerner is individualized.
They have relatively large personal space.
They value freedom of speech and individualism.
```

When the function is called as follows:

```
def nice_print2Ddict(d):
    dkeys = list(d.keys())
    dkeys.sort()
    for k in dkeys:
        print('\n', k)
        d2 = d[k]
        k2s = list(d2.keys())
        k2s.sort()
        print('*', end=' ')
        for k2 in k2s:
            print("{}:{:.3f}".format(k2, d2[k2]), end='; ')

if __name__ == '__main__':
    cultures = ['chinese.txt', 'thai.txt', 'japanese.txt', 'westerner.txt']
    res = culture_tf_idf(cultures)
    nice_print2Ddict(res)
```

We will see the outcome:

```
chinese.txt
* and:-0.013; associate:0.017; chinese:0.041; good:0.034; health:0.082;
highly:0.041;     longevity:0.041;     loudly:0.082;     speaking:0.017;
speaks:0.000; they:-0.026; value:-0.013; with:0.017;
japanese.txt
*  and:-0.020;  formally:0.063;  is:0.026;  japanese:0.063;  polite:0.063;
politeness:0.063;   responsibility:0.063;   speaks:0.000;   they:-0.041;
value:-0.020;
thai.txt
*   and:-0.014;   associate:0.018;   food:0.043;   fun:0.043;   good:0.036;
manner:0.043;  soft:0.043;  softly:0.043;  speaking:0.018;  speaks:0.000;
thai:0.043; they:-0.028; value:-0.014; with:0.018;
westerner.txt
*    and:-0.014;    freedom:0.043;    have:0.043;    individualism:0.043;
individualized:0.043;  is:0.018;  large:0.043;  of:0.043;  personal:0.043;
relatively:0.043;  space:0.043;  speech:0.043;  they:-0.028;  value:-0.014;
westerner:0.043;
```

Use `P12_template.py`. (The template is only to allow smooth autograding. You should edit and rename it properly.)

P13. Write a function named `simple_poet` taking a file name as an argument, read a content in the file, convert it to a list of strings for each line as an item, and return the list.

Invocation example

Given `poem.txt`

poem.txt

```
Where the departed goes, by Student

Where herons go
is a place for a departed soul.
What the choirs say
is how soul not just go away.
Like an origami,
it just shifts to a new story.
Transform the end of an old
to the beginning of a new fold.
```

When the function is called as follows:

```
res = simple_poet('poem.txt')
print(res)
```

We will see the outcome:

```
['Where the departed goes, by Student', '', 'Where herons go', 'is a
place for a departed soul.', 'What the choirs say', 'is how soul not
just go away.', 'Like an origami,', 'it just shifts to a new story.',
'Transform the end of an old', 'to the beginning of a new fold.']
```

P14. Write a program to calculate energy per volume (in MJ/bbl) from fuel density (in kg/L) and its calorific value (in MJ/kg).

Hint: 1 barrel (bbl) = 158.987 liters (L). If the problem is too big, break it down to smaller tasks.

Interaction example

```
Fuel density (in kg/L): 0.8
Calorific value (in MJ/kg): 44
This fuel has energy per volume of 35.20 MJ/L.
That is 5596.34 MJ/bbl.
```

P15. PM2.5. Given the level of PM2.5 (in µg/m^3) as shown in **Error! Reference source not found.**, write a program to ask a user for a level of PM2.5 (in µg/m^3) and report the AQI category, according to EPA AQI Factsheet 2012 (**Error! Reference source not found.**, the first and last columns).

AQI Category	Index Values	Previous Breakpoints (1999 AQI) (µg/m^3, 24-hour average)	Revised Breakpoints (µg/m^3, 24-hour average) PM2.5
Good	0 - 50	0.0 - 15.0	0.0 – 12.0
Moderate	51 - 100	>15.0 – 40.0 OBSOLETE	12.1 – 35.4
Unhealthy for Sensitive Groups	101 – 150	>40 – 150 OBSOLETE	35.5 – 55.4
Unhealthy	151 – 200	> 150	55.5 – 150.4
Very Unhealthy	201 – 300	> 150	150.5 – 250.4
Hazardous	301 – 400	> 250 –	250.5 – 350.4
	401 – 500	> 350 – 500	350.5 – 500

https://www.epa.gov/sites/production/files/2016-04/documents/2012_aqi_factsheet.pdf, retrieved on Feb 6th, 2021.

Interaction example

Outcome when inputting the level of PM2.5 as 11

```
PM2.5: 11
AQI: Good
```

Outcome when inputting the level of PM2.5 as 35

```
PM2.5: 35
AQI: Moderate
```

Outcome when inputting the level of PM2.5 as 50

```
PM2.5: 50
AQI: Unhealthy for Sensitive Groups
```

Outcome when inputting the level of PM2.5 as 60

```
PM2.5: 60
AQI: Unhealthy
```

Outcome when inputting the level of PM2.5 as 250

```
PM2.5: 250
AQI: Very Unhealthy
```

Outcome when inputting the level of PM2.5 as 300

```
PM2.5: 300
AQI: Hazardous
```

P16. Power series of exponential function. Given a power series of an exponential function $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$, write a program to calculate a sequence

$$e^x = \sum_{n=0}^{M} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \cdots + \frac{x^M}{M!}$$

where *x* and *M* are taken from a user.

Interaction example

Outcome when inputting *x* and *M* as 1 and 10

```
x: 1
M: 10
s = 2.71828
```

P17. Carbon-14 dating. Living organism has a relatively constant ratio of carbon-12 to carbon-14, but unstable carbon-14 will decay (and turn to nitrogen) at a constant rate after an organism dies, while carbon-12 remains constant. The drop in ^{14}C-to-^{12}C ratio can then be used to determine the last time the organism has lived.

Given the half-life of carbon-14 is 5730 years and the ^{14}C-to-^{12}C starting ratio is about 1 to 10^{12} (src: https://www.canadianarchaeology.ca/dating), write a program to take a starting ratio and an instrument sensitivity (the smallest ratio it can detect) and print out the estimated ^{14}C-to-^{12}C ratio at every 5730 year until the ratio goes below the sensitivity.

Interaction example 1

Outcome when inputting ratio (carbon-14 to 10^{12} carbon-12) and sensitivity (carbon-14 to 10^{12} carbon-12) as 1 and 0.001

```
ratio: 1
sensitivity: 0.001
Year 0: ratio = 1
Year 5730: ratio = 0.5
Year 11460: ratio = 0.25
Year 17190: ratio = 0.125
Year 22920: ratio = 0.0625
Year 28650: ratio = 0.03125
Year 34380: ratio = 0.015625
Year 40110: ratio = 0.0078125
```

```
Year 45840: ratio = 0.00390625
Year 51570: ratio = 0.00195312
```

Interaction example 2

Outcome when inputting ratio (carbon-14 to 10^{12} carbon-12) and sensitivity (carbon-14 to 10^{12} carbon-12) as 1.2 and 0.004

```
ratio: 1.2
sensitivity: 0.004
Year 0: ratio = 1.2
Year 5730: ratio = 0.6
Year 11460: ratio = 0.3
Year 17190: ratio = 0.15
Year 22920: ratio = 0.075
Year 28650: ratio = 0.0375
Year 34380: ratio = 0.01875
Year 40110: ratio = 0.009375
Year 45840: ratio = 0.0046875
```

P18. Garbage and landfill. Write a program to estimate a land size (in rai) needed for landfill.

The program asks an average daily waste (in kg/day) produced by a person and holding capacity of a land (in kg/m^2). Then the program calculates the daily total waste (in kg, using **70 million** as a population size), as well as computing a land size (in rai) needed for landfill of daily waste and a land size (in rai) needed for landfill of yearly waste (using **365 days**/year).

Hint: when a problem is too big, break it down to smaller tasks, e.g.,

(1) total waste = average waste * population size;

(2) land size (in m^2) = total waste (in kg)/holding capacity (in kg/m^2)

and (3) conversion: 1 rai = 1600 m^2.

Interaction example 1

Outcome when inputting 1.13 as a daily waste producing per person and 53900 as a holding capacity of a square meter of land.

```
Waste: 1.13
Cap: 53900
Total waste= 79100000.00
Landfill= 0.92
Annual land= 334.78
```

Noted that, to have a holding capacity of 53900 kg/m^2, we may have to fill a land as high as 70 meter on average to achieve such a capacity. However, landfill is generally the last report for handling the waste. Anyhow, ! CONSUME CONSCIOUSLY!

Interaction example 2

Outcome when inputting 1.14 as a daily waste producing per person and 50000 as a holding capacity of a square meter of land.

```
Waste: 1.14
Cap: 50000
Total waste= 79800000.00
Landfill= 1.00
Annual land= 364.09
```