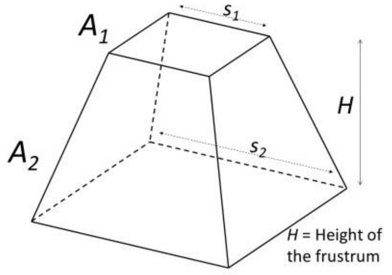
Midterm Q3

A frustum is a parallel truncation of a right pyramid. A piece of metal (shown in the Figure below) is in the shape of a frustum with a square base. The side length of the top square is s_1 and the side length of the bottom square is s_2. The height of the frustum is H.



The Volume of the frustum is given by the formula:

$$Volume = \frac{H}{3}\left(A1 + A2 + \sqrt{A1 \times A2}\right)$$

where A1 is the area of the upper square, A2 is the area of the lower square, and H is the height of the frustum.

a) Write a python function `area_square(s)` that takes the side of a square as an input argument s, and returns the area of the square.
b) Write a python function `vol_frustum(top_area, bottom_area, height)` that takes three arguments, a top area, a bottom area and a height in that order, and returns the volume of the frustum.
c) Write a python function `get_volume(s1, s2, height)` that takes three arguments, a top side length, a bottom side length and a height and returns the volume of a frustum based on those dimensions. This function should first call area_square to obtain the two needed areas, and then call vol_frustum to evaluate the volume.

All input arguments and return values are floats. Please round only your **final** output of `get_volume` to three decimal places. Please use `math.sqrt()` to compute the square root in your python code. Note that you get only full marks if your `get_volume` function makes use of the other two functions.

Test Code:	**Output:**

`print('{:.3f}'.format(area_square(2)))`
`print('{:.3f}'.format(area_square(3)))`
`print('{:.3f}'.format(vol_frustum(1,4,2)))`
`print('{:.3f}'.format(vol_frustum(2,2,2)))`
	`4.000 9.000`
	`4.667`
	`4.000`

`print('{:.3f}'.format(get_volume(1,2,2)))`
`print('{:.3f}'.format(get_volume(1.5,3.3,5.0)))`
`print('{:.3f}'.format(get_volume(3.6,6.4,4.0)))`
	`4.667`
	`30.150`
	`102.613`

Midterm Q4

A **square matrix** is a matrix of size *n* x *n*, i.e. *n* rows and *n* columns.

Given a square matrix **M** of size $1 \leqslant n \leqslant 3$, the **determinant** of **M**, denoted **det(M)**, is calculated using the following algorithm:

(1) If *n* = 1, and

$$M = [a]$$

Then **det(M)** = *a*.

(2) If *n* = 2, and

$$M = \begin{bmatrix} \mathrm{a} & \mathrm{b} \\ \mathrm{c} & \mathrm{d} \end{bmatrix}$$

Then **det(M)** = *ad – bc*.

(3) If *n = 3*, and

$$M = \begin{bmatrix} \mathrm{a} & \mathrm{b} & \mathrm{c} \\ \mathrm{d} & \mathrm{e} & \mathrm{f} \\ \mathrm{g} & \mathrm{h} & \mathrm{i} \end{bmatrix}$$

Then:

$$\det(M) = a \cdot \det\left(\begin{bmatrix} \mathrm{e} & \mathrm{f} \\ \mathrm{h} & \mathrm{i} \end{bmatrix}\right) - b \cdot \det\left(\begin{bmatrix} \mathrm{d} & \mathrm{f} \\ \mathrm{g} & \mathrm{i} \end{bmatrix}\right) + c \cdot \det\left(\begin{bmatrix} \mathrm{d} & \mathrm{e} \\ \mathrm{g} & \mathrm{h} \end{bmatrix}\right)$$

Task: Implement a function `determinant(matrix)` that takes a `matrix` as input (represented as a **nested list**) and returns its **determinant** as output. The function should satisfy the following requirements:

- If the input `matrix` is not of dimension *n* x *n* (for $1 \leqslant n \leqslant 3$), the function should return **`None`**;

- The function is to be implemented without importing any libraries (e.g. numpy), otherwise no marks will be awarded.

Hint: in the n = 3 case, it may be helpful to assign the elements of the matrix to variables, and then use those variables in computing the determinant.

Finals Q3

Book-keeping: In book-keeping, one form of asset is **Accounts Receivable** and one form of liability is **Accounts Payable**.

- **Accounts receivable** is money owed to your company by your clients.
- **Accounts payable** is money that your company owes to your suppliers.

The **Net equity** = **Total Accounts Receivable** – **Total Accounts Payable**. Write a function `equity(f)` that:

- Takes as input a dictionary `f` that models the accounting journal. The keys are the company names and the corresponding values are a list with two values. The zeroth value is the accounts receivable and the first value is the accounts payable. For example, given the following dictionary:

  {'Y': [100,20],'Z': [0,90]}

  It says that with company Y, the accounts receivable is 100 units and the accounts payable is 20 units. With company Z, there is no accounts receivable and the accounts payable is 90 units.
- Uses the information in the dictionary returned by ledger to calculate the net equity
- Returns a tuple with total receivable, total payable and net equity (see test code below)

Test Code:

```
f1={'A': [100, 0], 'B': [100, 0],
'C': [100, 0]}
f2={'M': [30, 20], 'N': [50, 70],
'O': [60, 80]}
f3={'J': [0, 30], 'K': [0, 20],
'L': [0, 40]}
 print(equity(f1))
print(equity(f2))
print(equity(f3))
```

Output:

```
(300, 0, 300)
(140, 170, -30)
(0, 90, -90)
```

Finals Q4

Write a function called `wordcount(s)` that takes as input, a string `s`, and returns as output, a list of counts of <u>**distinct, unique**</u> **words** that appear in each **line** contained in `s`.

A **line** is defined as a set of characters terminated by the newline character `'\n'`. Examples:

- `s = 'Hello hello'`, `s` contains one line
- `s = 'Hello \nWorld'`, `s` contains two lines. The individual lines are: o `'Hello '` o `'World'`
- `s = 'Hello \n\nWorld'`, `s` contains three lines. The individual lines are: o `'Hello '`

  o `''` (empty string) o `'World'`

The input string is made up of **ASCII** characters, and valid words contain only the letters of the English alphabet. Two words are distinct and unique only if they are the same word, case included. The words need not mean anything. If a line contains no words return `None` in place of the word count for that line.

Examples:

- For `s = 'As as'`, the word count is `2`
- For `s = 'as as '`, the word count is `1`.
- For `s = 'abcde ggg '`, the word count is `2`.
- For `s = ''`, the word count is `None`.

Punctuation characters from the set (`':', ';', ',', '.'` and `'-'`) may appear in the string and <u>**are to be ignored**</u>. For example, the string “hello; hello- hello.” will give the same count as the string “hello hello hello”, which is one (1) unique word. You may assume that words are separated by white space, even if there is punctuation in between them. Thus substrings of the type `'abc;def'` will not be present in the input.

Test Code:	**Output:**
print(wordcount('Tom Dick Harry'))	[3]
print(wordcount("Hello world\nHello Hello"))	[2, 1]
print(wordcount ("Hello hello"))	[2]
print(wordcount ("Hello World"))	[2]
print(wordcount ("Hello, Hello World"))	[2]
print(wordcount ("Hello \nWorld"))	[1,1]
print(wordcount ("Hello \nWorld\n"))	[1,1,None]
print(wordcount ("Hello \n\nWorld"))	[1, None, 1]
print(wordcount ("asdasf \n \n \n\nasdasfd;; asfdasd\n Hello hello hello hello world world world"))	[1, None, None, None, 2, 3]
print(wordcount ("Hello, world\nHello. Hello\n.”))	[2,1,None]