CRC

PROGRAMMING COMPETITION

PROBLEM BOOKLET

BLOCK A

A FEW NOTES

- The complete rules are in section 4 of the rulebook.
- You have until **19h59** to submit your solutions.
- Feel free to ask organizers questions and discuss the problem with your team members. It is not an exam!
- **We are giving you quick and easy-to-use template files. Please use them!**

USING THE TEMPLATE FILE

- All the files contain a function called solve() where you have to put your code. **DO NOT CHANGE THE NAME OF THAT FUNCTION!**
- To run the tests you will have to open a terminal (we strongly recommend to use vscode) and run the command `pytest.` If the command is ran in the folder containing all the problems, all tests are going to be ran
- To run a specific test, run it in the folder containing the test you want. You can do the same for sections, just navigate in the folder before executing the command `pytest`

STRUCTURE

Every problem contains a small introduction like this about the basics of the problem and what is required to solve it. Points distribution is also given here for the preliminary problems.

Input and output specification:

In these two sections, we specify what the inputs will be and what form they will take, and we also say what outputs are required for the code to produce and in what format they shall be.

Sample input and output:

In these two sections, you will find a sample input (that often has multiple entries itself) in the sample input and what your program should produce for such an input in the sample output.

First output explanation:

Sometimes, the problem might still be hard to understand after those sections, which is why there will also be a usually brief explanation of the logic that was used to reach the first output from the first input.

Table of content

2D PROBLEMS (2D) **4**
- 2D1: Euclid and Manhattan (25 points) 4
- 2D2: Staircase (40 points) 6
- 2D3: Arrows (25 points) 8
- 2D4: Smoke and mirrors (70 points) 10

DATA STRUCTURES (DS) **12**
- DS1: 2 Stacks makes a Queue (25 points) 12
- DS2: Strange sorting (30 points) 14
- DS3: In the matrix (30 points) 16

SCIENTIFIC CALCULATIONS (SC) **18**
- SC1: Ratio of the cylinders (10 points) 18
- SC2: Pi or almost (25 points) 20
- SC3: Averaging in circles (-350%360 points) 21
- SC4: Paint is really expensive! (30 points) 22

TEXT PROCESSING (TP) **25**
- TP1: Encrypted punctuation (40 points) 25
- TP2: Scrabble scramble (25 points) 27
- TP3: Literal reflection (25 points) 30

2D PROBLEMS (2D)

2D1: Euclid and Manhattan (25 points)

There are several ways to calculate the distance between two points. Two of the most popular are the Euclidean distance and the Manhattan distance. The Euclidean distance is simple; it is the shortest straight line between two points, using the Pythagorean theorem. Here is the formula for the Euclidean distance in 2D, with **x** representing the distance between the two points along the x-axis and **y** representing the distance along the y-axis.

$$D_{Euclid} = \sqrt{\Delta x^2 + \Delta y^2}$$

For the Manhattan distance, the concept is similar to a taxi traveling from point A to point B. You cannot move diagonally; you must move along the edges between points. In short, the Manhattan distance can be calculated using the formula:

$$D_{Manhattan} = \Delta x + \Delta y$$

As the final result, you will need to provide the sum of the Manhattan distance and the Euclidean distance. The two objects for which you must find the distance are represented by a and b in an array of strings forming a 2D grid. The distance between each letter in a string is 1, and the distance between each row is also 1.

Input Specification:

You will receive:

- ***grid:*** an *array* of *strings* that form the grid and where the markers are located.

Output Specification:

You will need to output:

- a *float* of the sum of the Manhattan and Euclidean distances with a precision of 2 decimals.

Sample input:

```
############
#    a     #
#          #
#      b   #
############


############
```

```
#a           #
#            #
#            #
#            #
#            #
#  b         #
#            #
#############

########
#      #
#     b#
#      #
#      #
#    a #
#      #
########
```

Sample output:

```
4 2.83
7 5.39
4 3.16
```

First output explanation:

In the first example, we have a grid where we can locate our two points. The first point is at (5,1), and the second point is at (7,3). This gives us a Manhattan distance between the two points, calculated using the following formula:

$$D_{Manhattan} = \Delta x + \Delta y$$

$$D_{Manhattan} = abs(5 - 7) + abs(1 - 3)$$

$$D_{Manhattan} = 2 + 2 = 4$$

Here's the calculation for the Euclidean distance between the same points:

$$D_{Euclid} = \sqrt{\Delta x^2 + \Delta y^2}$$

$$D_{Euclid} = \sqrt{(5 - 7)^2 + (1 - 3)^2}$$

$$D_{Euclid} = \sqrt{4 + 4}$$

$$D_{Euclid} = 2.82842712475 \approx 2.83$$

2D2: Staircase (40 points)

Building a staircase requires preparation and logic. You can't start building without knowing what you're doing. You will visually represent your staircase to help guide the building process. Your staircase will always start at the bottom left and ascend to the top right. It will have four customizable features:

- **Step Length (step_l)**: This affects how wide the top surface of the step is.
- **Step Height (step_h)**: This changes the height between steps.
- **Number of Steps (nbr_steps)**: You can specify how many steps the staircase should have.
- **Solid or Hollow Stairs (is_full)**: If the stairs are hollow, there is space between the steps. If solid, the area between two steps is filled.

You will display a side view of the staircase using the symbols | and _.

To enhance visibility, the staircase should be framed on the left and right with the # symbol.

The bottom of the staircase should always have spacing equivalent to the height of one step, regardless of whether the staircase is solid or hollow.

Input Specification:

You will receive:

- ***step_l:*** an *int* of the length of each stair
- ***step_h:*** an *int* of the height between stairs
- ***nbr_steps:*** an *int* of the number of stairs
- **is_full:** a *bool* that indicates if the staircase is full or not

Output Specification:

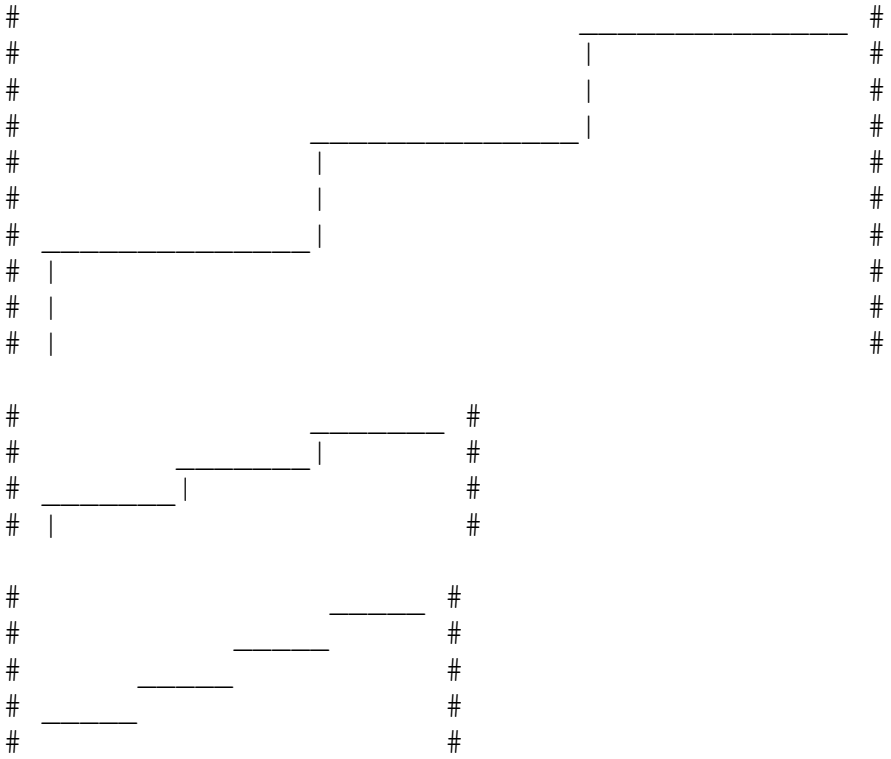
You will need to output:

- an array of *string* that are the visual representation of the staircase

Sample input:

```
14, 3, 3, True
7, 1, 3, True
5, 1, 4, False
```

Sample output:



First output explanation:

In the first example, we have a step length of 14, a step height of 3, and a number of steps set to 3 for a solid staircase. Therefore, we need to display the bottom of the steps since the staircase is solid. Even if the staircase is hollow, as in the third example, there must still be clearance from the ground according to the height of the steps.

2D3: Arrows (25 points)

You are starting a company that makes handcrafted decorative arrows. You want to give your customers quick previews of what their arrows will look like once produced. An arrow is divided into three simple parts: the shaft, the fletching, and the tip. The shaft is the long wooden section (and now often made of carbon for modern arrows), the fletching consists of the feathers that help stabilize the flight, and finally, the tip is usually made of metal in modern arrows. You will need to display custom arrows based on the number of feathers requested, the shaft length, and the number of arrow tips. The feathers are represented on either side of the arrow as \ and /. It is important to note that to display the \ symbol in Python, you must double it because it is a string escape character. The shaft starts with one space for feather clearance and is made up of the = symbol based on the requested length. Finally, the tip is represented by the > symbol, which can be doubled or tripled depending on the number of tips requested. You must ensure that each line of the string has the same length in your output. To do this, you should end each arrow with a wall of # symbols, where the arrow is embedded.

Input Specification:

You will receive:

- ***feathers***: an *int* of the number of feathers on the arrow
- ***length***: an *int* of the length of the body of the arrow
- ***point***: an *int* of the number of tips on the arrow

Output Specification:

You will need to output:

- an *array* of *strings* that represent the arrow stuck in the wall

Sample input:

```
feathers=3 length=11 point=2
feathers=2 length=10 point=1
feathers=2 length=12 point=2
```

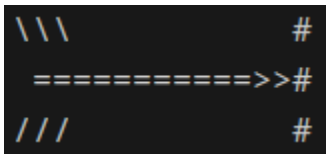
Sample output:

```
'\\\\\\            #',
' ==========>>#',
'///            #'

'\\\\           #',
' ==========>#',
'//           #'
'\\\\               #',
' ============>>#',
'//              #'
```

First output explanation:

In the first example, we have an arrow with 3 feathers, a shaft of length 11, and 2 tips. We need to double the \ symbol as it is an escape character. We also need to ensure the wall is properly shifted accordingly. For the arrow's shaft, we must leave an initial space for the feathers, followed by 11 units of length with the wall at the end. Here is the final visual if the lines are printed to the console:



2D4: Smoke and mirrors (70 points)

The thing that Narcissus loves the most in the world is to be able to look at himself all day. A daemon, wanting to have fun, offers to show him a room filled with mirrors. Without hesitating, Narcissus agrees. Suddenly, he is transported to a room and finds himself unable to move. As promised, the room is filled with mirrors, but none of them can give him a direct reflection. Undeterred by this, our hero tries to see if he can see himself by using multiple mirrors. Your task will be to determine whether or not Narcissus can see his reflection by using the mirrors that are in the room with him.The room that he was transported to measures a x b. Narcissus' position will be given as a vector (x_n, y_n). The extremities of the mirrors will be given in a list of vectors $[x_1, y_1, x_2, y_2]$.

Input and output specification:

For input, you will receive:

- (a, b): the dimensions of the room
- (x_n, y_n): Narcissus' position
- a list of vectors $[x_1, y_1, x_2, y_2]$: (x_1, y_1) and (x_2, y_2) are the coordinates of the edges of a mirror

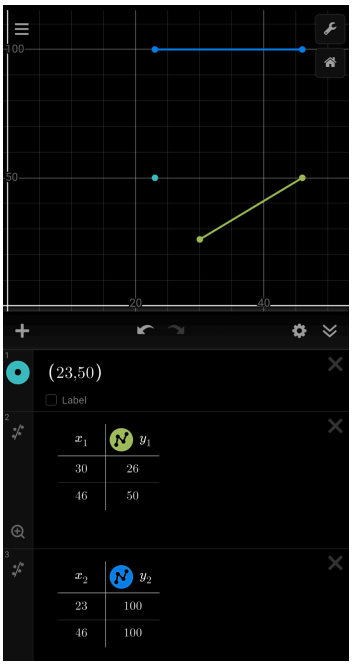
For output, you will return a boolean indicating whether or not Narcissus can see himself in the mirrors of the room.

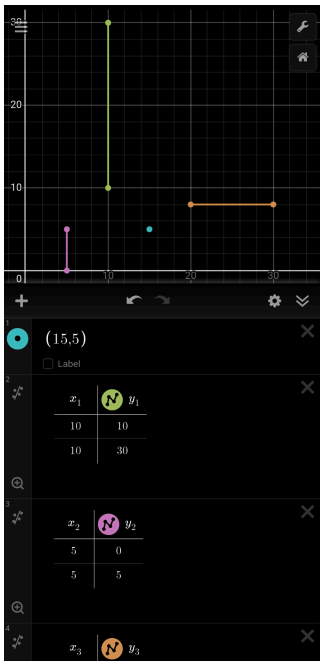
Sample input:

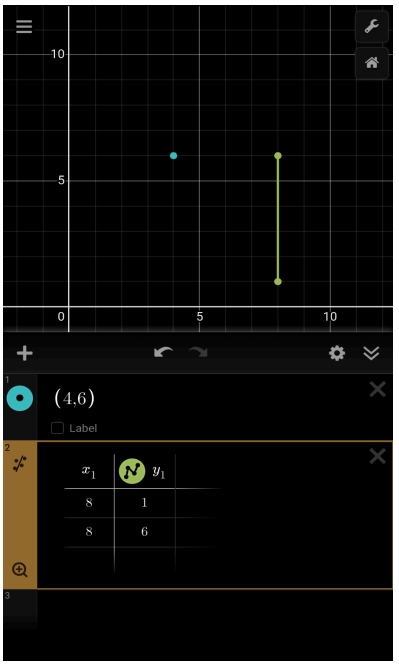
(46, 100)
(23, 50)
[(30, 26, 46, 50),
(23, 100, 46, 100)]

(30, 30)
(15, 5)
[(10, 10, 10, 30),
(5, 0, 5, 5),
(20, 8, 30, 8)]

(10, 10)
(4, 56)
[(8, 1, 8, 6)]







Sample output:

True
False
True

DATA STRUCTURES (DS)

DS1: 2 Stacks makes a Queue (25 points)

You want to implement a queue using only two stacks. You will add elements to the first stack and remove elements from the second stack. If you attempt to remove an element and the second stack is empty, you will need to transfer all elements from the first stack to the second stack.

You will have a series of operations to perform, and you must provide the list of removed elements. To add an element, the letter "A" will be followed by the number to add to the first stack, and to remove an element, the letter "R" will be used. Both stacks always start empty and may end with unsorted elements. You must provide the final state of both stacks at the end of the series of operations.

Input Specification:

You will receive:

- ***operations***: an array of *strings* that are the operations to do in order.

Output Specification:

You will need to output:

- an *array* containing 2 *arrays* of *int* that are the states of the 2 stacks at the end of the operations.

Sample input:

```
['A32', 'A42', 'R', 'A7', 'R', 'A21', 'A20', 'R', 'A53']
['A3', 'A5', 'A2', 'R', 'A7', 'R', 'A4', 'R', 'R', 'A9']
['A61', 'A80', 'A7', 'A9', 'A20', 'R', 'A53']
```

Sample output:

```
[[53], [20, 21]]
[[9], [4]]
[[53], [20, 9, 7, 80]]
```

First output explanation:

In the first example, both stacks start empty as usual. We add 32, then 42, and then we want to perform the first removal. We transfer 42, then 32, and remove 32. This leaves the first stack empty, and the second stack contains 42.

We add 7 to the first stack and remove 42 from the second stack. We add 21 and 20 to the first stack before attempting a removal, but the second stack is empty. Therefore, we transfer 20, 21, and 7, then remove 7. Finally, we add 53!

DS2: Strange sorting (30 points)

You want to do a slightly strange sorting of the people you work with. You want to organize them by age, but you also want to make the sorting a bit random. Therefore, after sorting them by age, you will swap the positions of those with names further in the alphabet one position toward the end of the list. You will make these changes for each neighboring pair only once, starting from the left to the right of the array.

So, if the first name comes earlier in the alphabet than the second, you don't change anything and move to the next comparison. However, if the first name comes later in the alphabet than the second, you swap their positions and then compare the swapped name with the next one. You repeat this comparison operation only once for each pair of names, and you stop with a final, half-sorted list.

Input Specification:

You will receive:

- ***people***: an *array* of people containing a *tuple* of their name and age

Output Specification:

You will need to output:

- an *array* of the *tuples* of persons sorted by the sorting rules given.

Sample input:

```
[(Justin, 18), (Marie, 19), (Judith, 22), (Georges, 14),
(Zachary, 21)]
[(Marc, 17), (Samuel, 20), (Ava, 22), (Greg, 15)]
[(Felix, 25), (Raphael, 24), (Gabriel, 22), (Francois, 23)]
```

Sample output:

```
[(Georges, 14), (Marie, 19), (Justin, 18), (Judith, 22),
(Zachary, 21)]
[(Greg, 15), (Marc, 17), (Ava, 22), (Samuel, 20)]
[(Francois, 23), (Gabriel, 22), (Felix, 25), (Raphael, 24)]
```

First output explanation:

In the first example, the input is:
[(Justin, 18), (Marie, 19), (Judith, 22), (Georges, 14), (Zachary, 21)]

After the sort by ages, we have:
[(Georges, 14), (Justin, 18), (Marie, 19), (Zachary, 21), (Judith, 22)]

We now do a pass of bubble sort using the names
step 1: (We compare Georges and Justin and they are in the right order alphabetically so we don't change anything)
[(Georges, 14), (Justin, 18), (Marie, 19), (Zachary, 21), (Judith, 22)]

step 2: (We compare Justin and Marie and swap them)
[(Georges, 14), (Marie, 19), (Justin, 18), (Zachary, 21), (Judith, 22)]

step 3: (We compare Justin and Zachary and don't need to swap them)
[(Georges, 14), (Marie, 19), (Justin, 18), (Zachary, 21), (Judith, 22)]

step 4: (We compare Zachary and Judith and swap them)
[(Georges, 14), (Marie, 19), (Justin, 18), (Judith, 22), (Zachary, 21)]

We have completed a full pass of bubble sort and we now have our final answer:
[(Georges, 14), (Marie, 19), (Justin, 18), (Judith, 22), (Zachary, 21)]

DS3: In the matrix (30 points)

You are doing matrix multiplication exercises and want to verify your results. You will need to program an algorithm that will first allow you to check if the multiplication between two matrices is possible, and then calculate the product of these two matrices if it is possible. The multiplication of two matrices is defined as follows:

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} r & s \\ t & u \\ v & w \end{bmatrix} = \begin{bmatrix} a*r+b*t+c*v & a*s+b*u+c*w \\ d*r+e*t+f*v & d*s+e*u+f*w \end{bmatrix}$$

Thus, we always multiply a row of the first matrix by a column of the second matrix. The coordinates of the resulting value are always given by which row was multiplied by which column. That is, if we multiply the 2nd row by the 1st column, we see that we get the element in the 2nd row, 1st column of the resulting matrix. **However, for this to work, the rows of the first matrix and the columns of the second matrix must have the same number of elements**. If this condition is not met, you must say that multiplication is not possible. Be careful, the order of operations is important!

Input Specification:

You will receive:

- ***m1***: a 2D *array* containing either *float* or *int*
- ***m2***: a 2D *array* containing either *float* or *int*

Output Specification:

You will need to output:

- a 2D *array* of *floats* or *int* that is the product of **m1*m2**

OR

- "Impossible" if the multiplication is not allowed

Sample input:

```
[[2,3],[7,1]] [[-1,-2],[3,5]]
[[1.1],[-0.7],[2.3]] [[5.4, -9.5]]
[[5,8]] [[4,6]]
[[1,-1],[-1,0],[1,1]] [[-1,-1,0],[1,0,-1]]
```

Sample output:

```
[[7,11],[-4,-9]]
[[5.94,-10.45],[-3.78,6.65],[12.42,-21.85]]
Impossible
[[-2,-1,1],[1,1,0],[0,-1,-1]]
```

First output explanation:

We first consider that the rows of m1 have 2 elements and that the columns of m2 also have 2 elements, which makes the product possible. We multiply the 1st row of m1 by the 1st column of m2, which gives 2 * -1 + 3 * 3 = -2 + 9 = 7. Continuing in this way, we get 2 * -2 + 3 * 5 = 11, 7 * -1 + 1 * 3 = -4, and 7 * -2 + 1 * 5 = -9.

SCIENTIFIC CALCULATIONS (SC)

SC1: Ratio of the cylinders (10 points)

You will need to calculate the ratio between the surface area and the volume of a cylinder. The dimensions of the cylinder given to you will be the diameter (d) and the height (h). To find the surface area of the cylinder, it can be broken down into two parts. The first part is the circles at the top and bottom of the cylinder, and the second part is the side of the cylinder.

$$A_{cercle} = \pi * r^2$$

$$A_{côté} = 2 * \pi * r * h$$

$$A_{totale} = 2 * A_{cercle} + A_{côté}$$

Here's the formula to get the volume of a cylinder:

$$V = \pi * r^2 * h$$

After performing these two calculations, you will need to return the ratio between the area and the volume, that is, the area divided by the volume with a precision of 2 decimal places

Input Specification:

You will receive:

- ***d***: an *int* that represent the diameter of the cylinder
- **h**: an *int* that represent the height of the cylinder

Output Specification:

You will need to output:

- a *float* with a 2 decimals precision of the ratio between the area and the volume of the cylinder

Sample input:

```
d=12, h=8
d=10, h=10
d=120, h=80
```

Sample output:

```
0.58
0.6
0.06
```

First output explanation:

For the first example, we have a diameter (d) of 12 and a height (h) of 8. We can therefore calculate the area of the circle, the area of the sides, and the total area.

$$A_{cercle} = \pi * r^2 = \pi * 6^2 \approx 113.09733552923255$$

$$A_{côté} = 2 * \pi * r * h = 2 * \pi * 6 * 8 \approx 301.59289474462014$$

$$A_{totale} = 2 * A_{cercle} + A_{côté} \approx 527.7875658030853$$

We can now compute the volume and the ratio between the total area and the volume:

$$V = \pi * r^2 * h = \pi * 6^2 * 8 \approx 904.7786842338604$$

$$ratio = \frac{A_{tot}}{V} = 0.5833333333333334 \approx 0.58$$

SC2: Pi or almost (25 points)

Many prefer the exact answers, but sometimes it is unfortunately not possible to get them. So, we will explore this topic by approximating the value of pi by repeating a simple operation 1,000,000 times. This operation is simply to generate two values between -1 and 1 that together give us a Cartesian coordinate in the square [(-1, -1), (-1, 1), (1, -1), (1, 1)]. In short, a square with a side length of 2, centered at (0,0). So, all the generated points will be within this square.

What we are really interested in is whether the random coordinate lies inside a circle of radius 1 centered at (0,0). To do this, we will check if

$$x^2 + y^2 < rayon$$
$$x^2 + y^2 < 1$$

We will be able to count how many times our random coordinate pairs fall inside the circle. Since we know the ratio of the areas between a square and a circumscribed circle, we can estimate, with enough trials, that the ratio of coordinates within the circle to the total number of generated coordinates should be approximately the same. Here is the formula for the ratio and how to isolate the value of pi.

$$\frac{nbr\ dans\ le\ cercle}{nbr\ coordonnées\ total} = ratio = \frac{\pi}{4}$$
$$\pi = 4 \cdot \frac{nbr\ dans\ le\ cercle}{nbr\ coordonnées\ total}$$

P.S. your code will be analyzed to make sure you have followed the prescribed method

Input Specification:

You will receive no input

Output Specification:

You will need to output:

- a float of the approximation of pi using 1,000,000 sample points

Sample output:

```
3.141952
3.142008
3.142476
```

First output explanation:

For each example, we generate x and y coordinates between -1 and 1. We check if the value is inside the circumscribed circle; if so, we increase the count. After performing 1,000,000 examples, we use the ratio to approximately calculate pi.

SC3: Averaging in circles (25 points)

You've known for many years how to calculate the average between 2 points, or even 3, or even n. It's easy, just sum the values and divide by the number of values. In 2D, averaging points is also relatively simple; just average along one axis, then average along the other axis. But what happens when we want to average 2 angles around a circle? We have to consider that 5 degrees is the same as 365 degrees and 350 degrees is the same as -10 degrees. You will need to create an algorithm that calculates the average angle of 2 angles and gives the result rounded to the nearest degree between 0 and 360. You need to find the average angle that is the closest on the circle. For that reason, we WON'T give you a problem with angles 180 degrees apart.

Input Specification:

You will receive:

- ***angles:*** an *array* of the angles in degree that you will need to find the average of. The angles are *int*, but can be negative values or even values over 360.

Output Specification:

You will need to output:

- an *int* of the average of the angles rounded to the nearest integer

Sample input:

```
[355, 735]
[0, -90]
[-365, 395]
```

Sample output:

```
10
315
20
```

First output explanation:

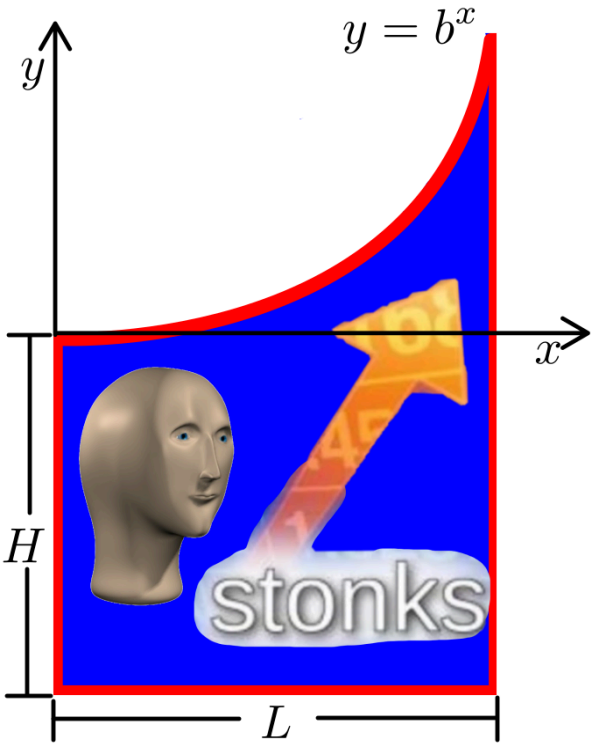
In the first example, we have 355 and 735 which can be interpreted as -5 and 15 degrees. The average of -5 and 15 degrees is 10 degrees.

SC4: Paint is really expensive! (30 points)

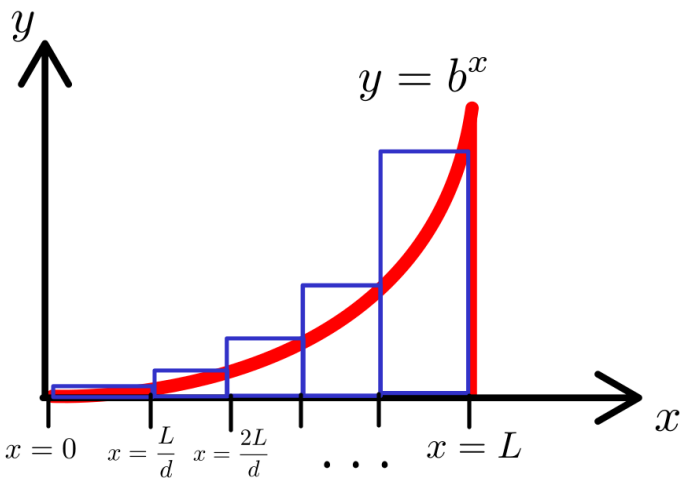
Your artist cousin has landed a big contract! In charge of creating a life-size logo for a stock market company, he's now at the painting stage. He calls on your services to determine the exact area to be painted.

The area to be painted is the following geometry:

To determine the area of the logo, the following dimensions are used. You need to calculate the sum of the area of the rectangle H x L and the area under the curve $y=b^x$, over the length L



To calculate the area under the curve, since you do not know the integral of the function $y=b^x$, you must use a method of approximation by rectangles, for which you can easily calculate the area as follows:



Your program needs to be able to compute the area under the curve using *d* rectangles to approximate the area.

Input Specification:

You will receive:

- H : a *float* of the height of the wall before the start of the curve in meters
- L : an *int* of the width of the wall in meters
- b : the *float* that is the basis of the exponential equation
- d : an *int* of the number of subdivisions to calculate the area under the curve

Output Specification:

You will need to output:

- a *float* of the total area to paint in square meters rounded up to the 3rd decimal.

Sample input:

```
H=2.23 L=3 b=2.1 d=5
H=4.1 L=6 b=3.2 d=10
H=3.4 L=4 b=1.2 d=18
```

Sample output:

```
20.486
1305.827
19.609
```

First output explanation:

We can separate the problem into 2 major sections. The base and the exponential section. The base is the easiest, since it is just a rectangle and the area of a rectangle is the multiplication between the 2 sides. So we have:

$$base = H * L$$

$$base = 2.23 * 3 = 6.69$$

For the exponential part, we can also use rectangle to our advantage! What we know is that we need to split the width of the wall (L) into d (5) equal bands. To make sure we have enough paint, we take the estimation of the height of the rectangle at the rightmost part of it. In order to get the height, we need to find the value of b^x where x is the rightmost point of the rectangle. For L=3 and d=5 the first rectangle end at $\frac{3}{5}$, for the second one it will be at $\frac{2*3}{5}$ then $\frac{3*3}{5}$, $\frac{4*3}{5}$ and finally $\frac{5*3}{5}$ which is 3 (the same thing as L!). for each rectangle we need to find the height so we have $y_1 = (2.1)^{\frac{3}{5}} = 1.560743651$ that is the height of the first rectangle. Since all of the 5 rectangles have a width of $\frac{3}{5}$, they area of the first rectangle is $A_1 = \frac{3}{5} * 1.560743651 = 0.93644619$

We can do the same thing for all of the 4 other rectangles that approximate the area under the curve to get:

$$A_2 = \frac{3}{5} * (2.1)^{\frac{2*3}{5}} = 1.461552446$$

$$A_3 = \frac{3}{5} * (2.1)^{\frac{3*3}{5}} = 2.2811087$$

$$A_4 = \frac{3}{5} * (2.1)^{\frac{4*3}{5}} = 3.5602259$$

$$A_5 = \frac{3}{5} * (2.1)^{\frac{5*3}{5}} = 5.5566$$

We can now get the approximated area by adding the base area and all the rectangles under the curve to get the total area.

$$A_{total} = base + A_1 + A_2 + A_3 + A_4 + A_5 = 20.4859332566295 \approx 20.486$$

TEXT PROCESSING (TP)

TP1: Encrypted punctuation (40 points)

Encryption is one of the classic problems in the CRC. However, this time the text is perfectly correct; it's the punctuation marks that you need to decode! To decode the punctuation, you will need to swap the punctuation marks until you find the correct placement for the text. The initial text will have punctuation marks in all the places that require one, but they may not necessarily be the correct ones. You will have to take the punctuation marks and shift them to the position of the next punctuation mark until you have a valid sentence. The provided text forms a loop, so the punctuation marks that should be moved to the end of the text will return to the first punctuation mark position after reaching the end.

You will thus have a text containing several sentences with incorrect punctuation in the wrong positions. To decode this, you will need to shift the positions of the punctuation marks, maintaining their initial order. Here are the rules to follow to ensure that the punctuation marks are in the correct places in the text:

- Terminal punctuation marks must be followed by an uppercase letter.
- The last punctuation mark in the text must be a terminal punctuation mark.
- Other punctuation marks may or may not be followed by an uppercase letter.

You will need to return how many shifts the punctuation marks must undergo to arrive for the first time at a valid position. To help count the shifts, a shift corresponds to moving each punctuation mark to the end of the sentence, with the very last punctuation mark moving to the first position of punctuation.

Input Specification:

You will receive:

- ***text***: a *string* of the text containing the misplaced punctuation signs.

Output Specification:

You will need to output:

- an *int* of the minimal shift towards the end of the sentence that produce a valid punctuation.

Sample input:

```
Bizarre comme les virgules changent tout. Qu’est ce qu’on ferait
sans elles, Vraiment! je n’en sais rien!

I really believe the earth is flat! Furthermore! the moon is
also flat I can assure you,

I don’t think it was a good decision. Although, with careful
consideration, it wasn’t that bad of a choice either.
```

Sample output:

```
1
2
0
```

First output explanation:

In the first sentence, we have a period as the first punctuation mark followed by an uppercase letter, which is correct. Then, we have a comma followed by an uppercase letter, which is potentially correct. However, the next punctuation mark is an exclamation point, but it is followed by a lowercase letter. This is problematic, so we need to move the punctuation marks and check if all the rules are followed. We make a shift to the right and notice that now all the punctuation rules are valid. Therefore, a shift of 1 was required.

TP2: Scrabble scramble (25 points)

You travel all over the world and in your travels, you bring your favorite game, a game of Scrabble! However, since you're an expert in this game, you give your opponent the chance to play in their language of choice. It's a big challenge you're setting for yourself, but you'll make a program to help you find plausible words in languages that you may not be familiar with. To do this, you will be given 5 letters and you need to find all the potential words in any language using the given letters.

In all the languages you'll be playing in, there are a series of rules that must be followed to obtain words that have the potential to be valid:

- All words must contain at least one vowel.
- There can be at most 2 consecutive consonants (this is not true in all languages, I know).
- There can be at most 2 consecutive vowels (this is not true in all languages, I know).
- You can use each letter only once in the formation of a word.

You need to return a list of all words between 1 and 5 letters that have the potential to be valid words in some language. The words should be in alphabetical order, with shorter words first.

P.S. y are considered vowels and you are allowed to use itertools!

Input Specification:

You will receive:

- ***letters***: a *string* of 5 letters that you have to form words

Output Specification:

You will need to output:

- an *array* of *string* of all the potential word alphabetically ordered

Sample input:

```
edcba
tests
aeiou
```

Sample output:

```
['a', 'e', 'ab', 'ac', 'ad', 'ae', 'ba', 'be', 'ca', 'ce', 'da',
'de', 'ea', 'eb', 'ec', 'ed', 'abc', 'abd', 'abe', 'acb', 'acd',
```

'ace', 'adb', 'adc', 'ade', 'aeb', 'aec', 'aed', 'bac', 'bad',
'bae', 'bca', 'bce', 'bda', 'bde', 'bea', 'bec', 'bed', 'cab',
'cad', 'cae', 'cba', 'cbe', 'cda', 'cde', 'cea', 'ceb', 'ced',
'dab', 'dac', 'dae', 'dba', 'dbe', 'dca', 'dce', 'dea', 'deb',
'dec', 'eab', 'eac', 'ead', 'eba', 'ebc', 'ebd', 'eca', 'ecb',
'ecd', 'eda', 'edb', 'edc', 'abce', 'abde', 'abec', 'abed',
'acbe', 'acde', 'aceb', 'aced', 'adbe', 'adce', 'adeb', 'adec',
'aebc', 'aebd', 'aecb', 'aecd', 'aedb', 'aedc', 'bacd', 'bace',
'badc', 'bade', 'baec', 'baed', 'bcad', 'bcae', 'bcea', 'bced',
'bdac', 'bdae', 'bdea', 'bdec', 'beac', 'bead', 'beca', 'becd',
'beda', 'bedc', 'cabd', 'cabe', 'cadb', 'cade', 'caeb', 'caed',
'cbad', 'cbae', 'cbea', 'cbed', 'cdab', 'cdae', 'cdea', 'cdeb',
'ceab', 'cead', 'ceba', 'cebd', 'ceda', 'cedb', 'dabc', 'dabe',
'dacb', 'dace', 'daeb', 'daec', 'dbac', 'dbae', 'dbea', 'dbec',
'dcab', 'dcae', 'dcea', 'dceb', 'deab', 'deac', 'deba', 'debc',
'deca', 'decb', 'eabc', 'eabd', 'eacb', 'eacd', 'eadb', 'eadc',
'ebac', 'ebad', 'ebca', 'ebda', 'ecab', 'ecad', 'ecba', 'ecda',
'edab', 'edac', 'edba', 'edca', 'abced', 'abdec', 'abecd',
'abedc', 'acbed', 'acdeb', 'acebd', 'acedb', 'adbec', 'adceb',
'adebc', 'adecb', 'bacde', 'baced', 'badce', 'badec', 'baecd',
'baedc', 'bcade', 'bcaed', 'bcead', 'bceda', 'bdace', 'bdaec',
'bdeac', 'bdeca', 'beacd', 'beadc', 'becad', 'becda', 'bedac',
'bedca', 'cabde', 'cabed', 'cadbe', 'cadeb', 'caebd', 'caedb',
'cbade', 'cbaed', 'cbead', 'cbeda', 'cdabe', 'cdaeb', 'cdeab',
'cdeba', 'ceabd', 'ceadb', 'cebad', 'cebda', 'cedab', 'cedba',
'dabce', 'dabec', 'dacbe', 'daceb', 'daebc', 'daecb', 'dbace',
'dbaec', 'dbeac', 'dbeca', 'dcabe', 'dcaeb', 'dceab', 'dceba',
'deabc', 'deacb', 'debac', 'debca', 'decab', 'decba', 'ebacd',
'ebadc', 'ebcad', 'ebdac', 'ecabd', 'ecadb', 'ecbad', 'ecdab',
'edabc', 'edacb', 'edbac', 'edcab']

['e', 'es', 'es', 'et', 'et', 'se', 'se', 'te', 'te', 'ess',
'est', 'est', 'ess', 'est', 'est', 'ets', 'ets', 'ett', 'ets',
'ets', 'ett', 'ses', 'set', 'set', 'sse', 'ste', 'ste', 'ses',
'set', 'set', 'sse', 'ste', 'ste', 'tes', 'tes', 'tet', 'tse',
'tse', 'tte', 'tes', 'tes', 'tet', 'tse', 'tse', 'tte', 'sest',
'sest', 'sets', 'sett', 'sets', 'sett', 'sset', 'sset', 'stes',
'stet', 'stes', 'stet', 'sest', 'sest', 'sets', 'sett', 'sets',
'sett', 'sset', 'sset', 'stes', 'stet', 'stes', 'stet', 'tess',
'test', 'tess', 'test', 'tets', 'tets', 'tses', 'tset', 'tses',
'tset', 'ttes', 'ttes', 'tess', 'test', 'tess', 'test', 'tets',

```
'tets', 'tses', 'tset', 'tses', 'tset', 'ttes', 'ttes', 'ssett',
'ssett', 'stest', 'stets', 'stest', 'stets', 'ssett', 'ssett',
'stest', 'stets', 'stest', 'stets', 'tsest', 'tsets', 'tsest',
'tsets', 'ttess', 'ttess', 'tsest', 'tsets', 'tsest', 'tsets',
'ttess', 'ttess']

['a', 'e', 'i', 'o', 'u', 'ae', 'ai', 'ao', 'au', 'ea', 'ei',
'eo', 'eu', 'ia', 'ie', 'io', 'iu', 'oa', 'oe', 'oi', 'ou',
'ua', 'ue', 'ui', 'uo']
```

First output explanation:

In the first example, when we have the letters edcba, we can form a lot of potential words. Indeed, we start with one-letter words, which are the available vowels. Then we form two-letter words, which can either be two vowels or one vowel and one consonant. We continue in this way, making sure not to have 3 consecutive consonants or 3 consecutive vowels. Therefore, combinations like abcde cannot be formed as they contain 3 consecutive consonants.

TP3: Literal reflection (25 points)

Has anyone ever told you to think before speaking? This solution is for you! Here, you need to perform two key reflections one after the other on a given character sequence. First, we will use a vertical axis centered in the middle of the sequence to reverse the placement of all the characters without changing their nature. For example, 'tourtour' becomes 'ruotruot'. Then, we will take a horizontal axis that perfectly splits all characters into two equal halves and change certain ones. The affected characters are only those that have a corresponding character after reflection, and these are **exclusively**:
'b' becomes 'p' and vice versa, 'd' becomes 'q' and vice versa, 'f' becomes 't' and vice versa, 'g' becomes '6' and vice versa, 'i' becomes '!' and vice versa, 'j' becomes '?' and vice versa, 'm' becomes 'w' and vice versa, 'n' becomes 'u' and vice versa, 's' becomes 'z' and vice versa, '2' becomes '5' and vice versa.
Uppercase letters and other characters not listed here are not affected by the second reflection.

Input Specification:

You will receive:

- a *string* that you will have to reflect

Output Specification:

You will need to output:

- the reflected *string*

Sample input:

```
tourtour
J’ai acheté 62 fleurs chez Bertrand!
je mange des pommes vertes avec 55 points roses dessus
CRC is absolute cinema!
```

Sample output:

```
rnofrnof
iquarfreB sehc zrnelt 5g éfehca !a’J
znzzeq zezor zfu!ob 22 ceva zefrev zewwob zeq e6uaw e?
iaweu!c efnlozpa z! CRC
```

First output explanation:

The vertical axis of symmetry requires the inversion of the string, which results in "ruotruot". Then, "o" and "r" are not in the list of characters to be swapped, so they will not be affected by the second reflection. As for the others, both "u"s become "n"s and both "t"s become "f"s, giving "rnofrnof".