Robotique
CRC
Robotics

CRC

COMPÉTITION DE PROGRAMMATION

TAK TIK 2025

LIVRET DE PROBLÈMES BLOC A

QUELQUES NOTES

- Les règles complètes sont dans la section 4 du livret des règlements
- Vous avez jusqu'à **19h59** pour remettre vos solutions
- N'hésitez pas à nous poser des questions et à communiquer avec les membres de votre équipe de programmation. Ce n'est pas un examen!
- **On vous donne des fichiers modèles faciles à utiliser pour votre code. Merci de les utiliser.**

UTILISATION DU FICHIER MODÈLE

- Tous les fichiers possèdent une fonction solve() dans lequel vous devez résoudre le problème. **NE CHANGEZ PAS LE NOM DE CETTE FONCTION!**
- Pour rouler vos tests, dans un terminal (nous vous conseillons l'utilisation de vscode) écrivez la ligne de commande: `pytest` pour rouler les tests. Si vous faites la commande `pytest` dans le répertoire contenant tous les fichiers, tous les tests seront exécutés.
- Pour rouler les tests d'un problème spécifique, naviguez dans le répertoire de la question et exécutez la commande `pytest` pour rouler les tests du problème. Vous pouvez faire la même chose pour une section entière!

STRUCTURE

Chaque problème contient une petite mise en situation comme celle-ci expliquant les fondements du problème et donnant les bases nécessaires pour résoudre celui-ci. Chaque problème préparatoire contient aussi la répartition des points pour le problème.

Spécifications d'entrée et de sortie:

Dans cette section, on retrouve les caractéristiques des entrées qui peuvent être fournies au code en question ainsi que les critères attendus pour les sorties du programme.

Exemple d'entrée et de sortie:

Dans cette section se trouve un exemple d'entrée (parfois constitué lui-même de plusieurs sous-exemples) pour que vous puissiez tester votre code. L'exemple de sortie donne donc la réponse attendue pour cette entrée.

Explication de la première sortie:

Si le problème n'est toujours pas clair après la mise en situation, l'explication de la première sortie sert parfois à démêler le tout en expliquant comment la première entrée est traitée et en montrant le chemin menant à cette réponse.

Table des matières

PROBLÈMES 2D (2D)

2D1: Euclid et Manhattan (25 points)

Il y a plusieurs moyens de calculer des distances entre deux points. Deux des plus populaires sont la distance euclidienne et la distance de Manhattan. La distance Euclidienne est simple, c'est la plus petite ligne droite entre deux points. Elle se sert du théorème de Pythagore. Voici la formule pour la distance Euclidienne en 2D avec Δx la distance entre les deux points selon l'axe des x et Δy la distance entre les deux points selon l'axe des y.

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

Pour la distance Manhattan, le concept est un peu comme si un taxi devait parcourir un trajet pour se rendre du point A au point B. On ne peut pas passer en diagonale, on doit se déplacer sur les arêtes entre les points. Bref, la distance Manhattan peut être obtenue avec la formule:

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

Comme résultat final, vous devrez nous donner la somme des distances entre la distance Manhattan et la distance Euclidienne. Les deux objets dont vous devrez trouver la distance sont a et b dans un array composé de string qui forment une grille en 2D. La distance entre chaque lettre de string est de 1 et la distance entre chaque rangée est également de 1.

Spécifications d'entrée:

En entrée vous recevrez:

- ***grid:*** un array contenant des strings qui forment la grille ou sont les objets à trouver la distance

Spécifications de sortie:

En sortie vous devrez fournir:

- un *float* avec 2 décimales de précision de la somme des distances Manhattan et Euclidiennes

Exemple d'entrée:

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

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

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

Exemple de sortie:

```
6.83
12.39
7.16
```

Explication de la première sortie:

Dans le premier exemple, nous avons une grille dans laquelle on peut aller chercher nos deux points. Nous avons le premier point qui est en (5,1) et le second point qui est en (7, 3). Ceci nous fais une distance de manhattan entre les deux points calculée avec la formule suivante:

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

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

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

Voici le calcul pour la distance euclidienne entre ces deux même 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: L'escalier (40 points)

Construire un escalier demande de la préparation et de la logique. Il ne faut pas commencer à construire sans savoir ce que l'on fait. Vous allez donc représenter visuellement votre escalier pour vous aider à le construire par la suite. **Votre escalier va toujours commencer en bas à gauche et monter vers en haut à droite.** Votre escalier aura 4 caractéristiques sur lesquelles on peut jouer.

- On peut changer, la longueur de la marche (step_l) qui impacte à quelle point la surface du dessus de la marche est grande.
- On peut changer la hauteur de la marche (step_h) qui va changer la hauteur entre les marches.
- On peut aussi choisir le nombre de marches à afficher (nbr_steps).
- Finalement, le dernier aspect sur lequel on peut travailler est de savoir si les escaliers sont pleins ou vides. Si les escaliers sont vides, on a un espace entre les différentes marches, sinon on peut voir la montée entre deux marches.

Vous devrez faire le visuel de l'escalier vu de côté en utilisant les symboles: | _

Pour mieux voir l'escalier il faut l'encadrer à gauche et à droite du symbole: #

Le bas de l'escalier doit avoir l'espacement de la hauteur d'une marche peu importe si l'escalier est plein ou vide.

Spécifications d'entrée:

En entrée vous recevrez:

- ***step_l:*** un *int* de la longueur de la marche
- ***step_h:*** un *int* de la hauteur entre les marches
- ***nbr_steps:*** un *int* de le nombre de marche de l'escalier
- **is_full:** un *bool* qui indique si l'escalier est plein

Spécifications de sortie:

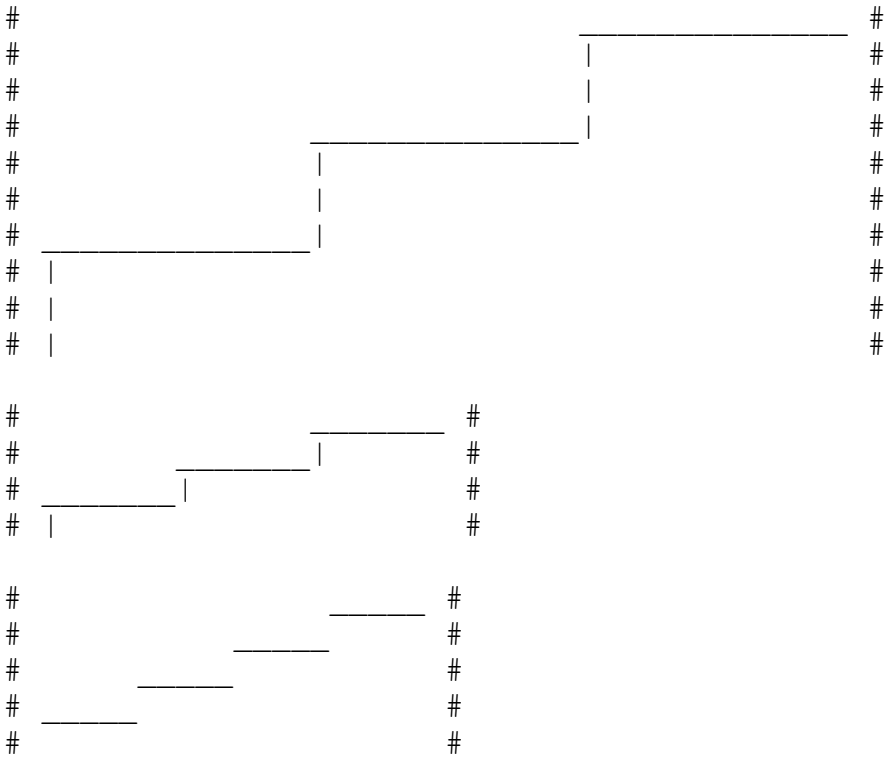
En sortie vous devrez fournir:

- un array de *string* qui représente le visuel de l'escalier.

Exemple d'entrée:

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

Exemple de sortie:



Explication de la première sortie:

Dans le premier exemple, nous avons une longueur de marche de 14, une hauteur de marche de 3 et un nombre de marche de 3 pour un escalier plein. Ainsi, il nous faut faire le bas des marches, car l'escalier est un escalier plein. Même si l'escalier n'est pas plein comme dans le troisième exemple, on doit laisser un dégagement avec le sol selon la hauteur de la marche.

2D3: Flèches (25 points)

Vous commencez une compagnie de flèches artisanales de décoration. Vous souhaitez faire des petits aperçus rapide à vos clients de quoi auront l'air leur flèches une fois produites. Une flèche se divise en trois parties simples qui sont la hampe, l'empennage et la pointe. L'hampe est la longue section en bois (et maintenant pour des flèches modernes en carbone), l'empennage est les plumes qui aident à stabiliser le vol et finalement nous avons la pointe qui est souvent faite de métal dans les flèches modernes. Vous devrez donc afficher des flèches sur mesure selon le nombre de plumes demandées, la longueur de la hampe et selon le nombre de pointes. Les plumes sont représentées de part et d'autre de la flèche comme des \ et /. **Il est important de noter que pour afficher le symbole \ vous devez le doubler en python car c'est un symbole d'échappement de string.** Vous aurez aussi la hampe qui commence avec un dégagement d'un espace pour les plumes et qui est composé de symbole = selon la longueur demandée. et nous avons finalement la pointe qui est composée d'un symbole >. Le symbole peut être doublé ou triplé selon le nombre de pointes demandé. Vous devez vous assurer que chacune des lignes de string affichent la même longueur dans votre sortie. Pour ce faire, vous devrez terminer chacune des flèches par un mur de # dans lesquels la flèche est plantée.

Spécifications d'entrée:

En entrée vous recevrez:

- ***feathers***: un *int* du nombre de plumes sur la flèche
- ***length***: un *int* de la longueur de la hampe (corps de la flèche)
- ***tips***: un *int* du nombre de pointes de la flèche

Spécifications de sortie:

En sortie vous devrez fournir:

- un *array* de *string* qui sont la représentation de la flèche plantée dans un mur.

Exemple d'entrée:

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

Exemple de sortie:

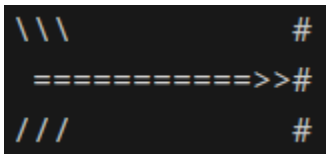
```
'\\\\\\            #',
' ==========>>#',
'///            #'

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

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

Explication de la première sortie:

Dans le premier exemple, nous avons une flèche avec 3 plumes, un corps de longueur 11 et 2 pointes. Nous devons mettre le double de \ comme c'est un symbole d'échappement. Nous devons aussi faire attention de décaler le mur en conséquence. Pour ce qui est du corps de la flèche, nous devons laisser un premier espacement pour les plumes, puis nous avons les 11 unités de longueur avec au bout le mur. Voici le visuel final si les lignes sont imprimées à la console:



2D4: Miroir mirroir (70 points)

Ce que Narcisse aime le plus au monde, c'est pouvoir se regarder toute la journée. Un démon, voulant s'amuser, lui propose de lui montrer une pièce remplie de miroirs. Sans hésiter, Narcisse accepte. Soudain, il est transporté dans une pièce et se retrouve incapable de bouger. Comme promis, la pièce est remplie de miroirs, mais aucun d'eux ne peut lui renvoyer une réflexion directe. Imperturbable, notre héros essaie de voir s'il peut se voir en utilisant plusieurs miroirs. Votre tâche consistera à déterminer si Narcisse peut ou non voir son reflet en utilisant les miroirs présents dans la pièce avec lui. La pièce dans laquelle il a été transporté mesure a x b. La position de Narcisse sera donnée sous forme de vecteur (xn, yn). Les extrémités des miroirs seront données dans une liste de vecteurs [x1, y1, x2, y2].

Spécifications d'entrée:

En entrée vous recevrez:

- (a, b): les dimensions de la pièce
- (x_n, y_n): La position de Narcisse
- un *array* de vecteur [x_1, y_1, x_2, y_2]: (x_1, y_1) et (x_2, y_2) qui sont les coordonnées des coins des miroirs

Spécifications de sortie:

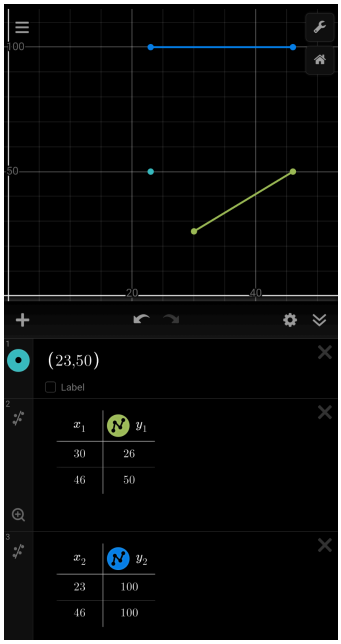
Comme sortie vous devrez retourner un *bool* de si Narcisse arrive à voir sa réflexion.

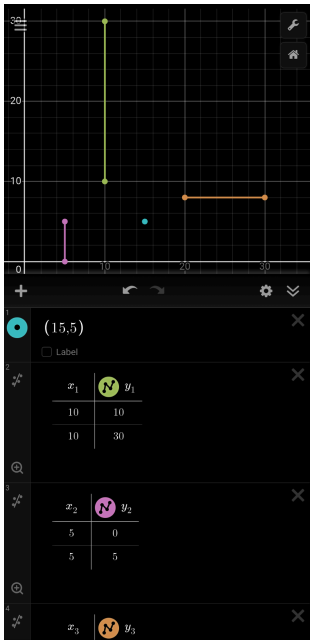
Exemple d’entrée:

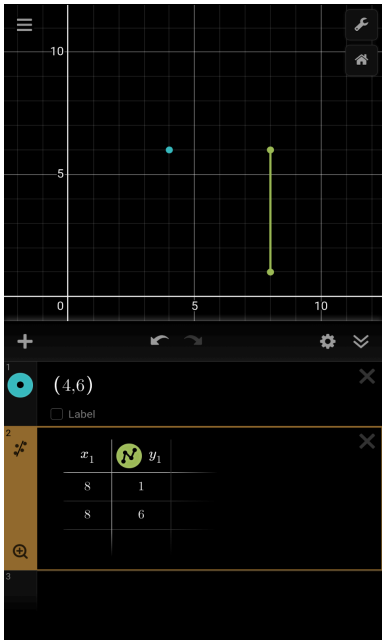
(46, 100)
(23, 50)
[(30, 26, 46, 50),
(23, 100, 46, 100)]

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

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







Exemple de sortie:

True
False
True

STRUCTURES DE DONNÉES (DS)

DS1: Deux piles font une file (25 points)

Vous voulez implémenter une file, mais vous n'avez accès qu' à deux piles. Vous mettrez les éléments ajoutés dans la première pile et vous allez retirer les éléments à partir de la seconde pile. Si vous essayez de retirer un élément et que la seconde pile est vide, il vous faudra transvider de la première vers la seconde pile.
Vous aurez donc une série d'opérations à effectuer et vous devrez donner la liste des éléments qui sont retirés. Pour faire un ajout nous aurons la lettre A suivi du nombre à ajouter à la première pile et pour retirer un nombre, nous aurons la lettre R. Les deux piles commencent toujours vides et peuvent terminer avec des éléments non sortis. Vous devrez donner l'état des deux piles à la fin de la série d'opérations.

Spécifications d'entrée:

En entrée vous recevrez:

- ***operations***: un array de *string* qui sont les opérations à effectuer en ordre

Spécifications de sortie:

En sortie vous devrez fournir:

- un tableau contenant deux tableaux de *int* qui sont les nombres dans les deux piles à la fin des opérations

Exemple d'entrée:

```
['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']
```

Exemple de sortie:

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

Explication de la première sortie:

Dans le premier exemple, nos deux piles commencent vides comme toujours. On ajoute 32, puis 42 et finalement on veut faire un premier retrait. On transfère 42 puis 32 et on retire 32. Ceci nous laisse la première pile vide et la seconde contient 42. On ajoute 7 à la première pile et on retire le 42 de la seconde pile. On ajoute 21 et 20 à la seconde pile avant de faire un retrait, mais la seconde pile est vide. Donc on transfère 20, 21 et 7 puis on retire 7. Finalement on ajoute 53!

DS2: Tri étrange (30 points)

Vous voulez faire un tri un peu étrange des gens avec qui vous travaillez. Vous souhaitez les organiser par âge, mais vous voulez rendre le tri un peu aléatoire aussi. Vous décidez donc qu'une fois trié par âge, vous allez échanger de place ceux qui ont un nom plus loin dans l'alphabet d'une place vers la fin de la liste. Vous allez faire ces changements pour chaque voisins une seule fois en partant de la gauche vers la droite du tableau.
Ainsi, si le premier nom est avant dans l'alphabet que le second, vous ne changez rien et passez à la prochaine comparaison. Par contre, si le premier nom est plus loin dans l'alphabet que le second, vous échangez les noms de place et vous faites une comparaison avec le nom qui est changé de place et le suivant. Vous répétez cette opération de comparaison des noms une seule fois pour chaque paire de noms puis vous arrêtez avec une liste finale à moitié triée.

Spécifications d'entrée:

En entrée vous recevrez:

- ***people***: un array de *tuple* qui contiennent un nom et un âge

Spécifications de sortie:

En sortie vous devrez fournir:

- un array contenant les *tuples* des gens triés selon les règles de tri

Exemple d'entrée:

```
[(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)]
```

Exemple de sortie:

```
[(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)]
```

Explication de la première sortie:

Dans le premier exemple, nous avons en entrée:
[(Justin, 18), (Marie, 19), (Judith, 22), (Georges, 14), (Zachary, 21)]

après le tri en âge on obtient:
[(Georges, 14), (Justin, 18), (Marie, 19), (Zachary, 21), (Judith, 22)]

On fait un pass complet de bubble sort par rapport aux noms
etape 1: (On compare Georges et Justin, pas besoin de changer c'est le bon ordre de nom)
[(Georges, 14), (Justin, 18), (Marie, 19), (Zachary, 21), (Judith, 22)]

etape 2: (On compare Justin et Marie, On les changent de place)
[(Georges, 14), (Marie, 19), (Justin, 18), (Zachary, 21), (Judith, 22)]

etape 3: (On compare Justin et Zachary, pas besoin de faire de changement)
[(Georges, 14), (Marie, 19), (Justin, 18), (Zachary, 21), (Judith, 22)]

etape 4: (On compare Zachary et Judith, on les changent de place)
[(Georges, 14), (Marie, 19), (Justin, 18), (Judith, 22), (Zachary, 21)]

On vient de finir un passage de tri par comparaison, on arrête. La réponse finale est:
[(Georges, 14), (Marie, 19), (Justin, 18), (Judith, 22), (Zachary, 21)]

DS3: Dans la matrice (30 points)

Vous faites des exercices de multiplication de matrices et souhaitez vérifier vos résultats. Vous devrez donc programmer un algorithme qui vous permettra d'abord de vérifier que la multiplication entre deux matrices est possible, puis de calculer le produit de ces deux matrices si c'est possible. On définit la multiplication de deux matrices comme suit:

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

On multiplie ainsi toujours une rangée de la première matrice par une colonne de la deuxième matrice. Les coordonnées du résultat obtenu sont toujours données par quelle rangée on a multiplié à quelle colonne, c'est-à-dire que si on multiplie la 2e rangée par la 1re colonne, on voit qu'on obtient l'élément dans la 2e rangée 1re colonne de la matrice résultante. **Pour que ceci fonctionne cependant, il faut que les rangées de la première matrice et que les colonnes de la deuxième matrice aient le même nombre d'éléments.** Si ce critère n'est pas respecté, vous devrez dire que la multiplication n'est pas possible. Attention, l'ordre d'opération est important!

Spécifications d'entrée:

En entrée vous recevrez:

- ***m1***: une liste en deux dimensions contenant des nombres entiers ou réels
- ***m2***: une liste en deux dimensions contenant des nombres entiers ou réels

Spécifications de sortie:

En sortie vous devrez fournir:

- une liste en deux dimensions d'entiers ou de réels (selon ce que vous avez reçu en entrée) correspondant au produit **m1*m2**

OU

- "Impossible" si la multiplication n'est pas possible

Exemple d'entrée:

```
[[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]]
```

Exemple de sortie:

```
[[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]]
```

Explication de la première sortie:

On considère d'abord que les rangées de m1 ont 2 éléments et que les colonnes de m2 ont 2 éléments aussi, ce qui rend le produit possible. On multiplie la 1re rangée de m1 par la 1re colonne de m2, ce qui donne 2*-1 + 3*3 = -2+9 = 7. Poursuivant ainsi, on obtient 2*-2 + 3*5 = 11, 7*-1 + 1*3 = -4 et 7*-2 + 1*5 = -9.

CALCULS SCIENTIFIQUES (SC)

SC1: Ratio des cylindres (10 points)

Vous devrez calculer le ratio entre l'aire de la surface et le volume d'un cylindre. Les dimensions du cylindre qui vous seront données sont le diamètre (d) et la hauteur (h). Pour trouver l'aire de la surface du cylindre, on peut la décomposer en 2 parties. La première est le cercle qui se trouve en haut et en bas du cylindre et la seconde est le côté du cylindre.

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

$$A_{côté} = 2 * \pi * r * h$$

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

Voici maintenant la formule pour calculer le volume d'un cylindre:

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

Après avoir fait ces deux calculs, vous devrez retourner le ratio entre l'aire et le volume donc l'aire divisée par le volume avec une précision de 2 décimales après la virgule.

Spécifications d'entrée:

En entrée vous recevrez:

- ***d***: un *int* qui représente le diamètre du cylindre
- **h**: un *int* qui représente la hauteur du cylindre

Spécifications de sortie:

En sortie vous devrez fournir:

- un float avec deux décimales de précision du ratio entre l'aire et le volume du cylindre

Exemple d'entrée:

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

Exemple de sortie:

```
0.58
0.6
0.06
```

Explication de la première sortie:

Pour le premier exemple on a d qui vaut 12 et h qui vaut 8. On peut donc aller chercher l'aire du cercle, l'aire des côtés et l'aire totale.

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

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

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

On peut maintenant calculer le volume et le ratio des aires et volumes.

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

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

SC2: Pi ou presque (25 points)

Plusieurs préfèrent les vraies réponses exactes, mais parfois ce n'est pas possible d'en avoir malheureusement. Nous allons donc explorer ce sujet en approximant la valeur de pi en répétant 1 000 000 de fois une opération simple. Cette opération est simplement de générer deux fois une valeur entre -1 et 1 qui vont nous donner ensemble une coordonnée cartésienne dans le carré [(-1, -1), (-1, 1), (1, -1), (1, 1)] Bref, un carré avec une longueur de côté de 2 qui est centré en (0,0). Donc tous les points générés seront dans ce carré.

Ce qui nous intéresse vraiment c'est de savoir si la coordonnée aléatoire se trouve dans un cercle de rayon 1 centré en (0,0). Pour ce faire, nous allons vérifier si

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

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

Nous allons pouvoir compter le nombre de fois que nos paires de coordonnées aléatoire sont dans le cercle. Comme nous connaissons le ratio des aires entre un carré et un cercle circonscrit, nous pouvons estimer avec assez d'expériences réalisées que le ratio des coordonnées dans le cercle par rapport au nombre total de de coordonnées générées devrait être approximativement le même. Voici la formule du ratio et comment isoler la valeur de pi.

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

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

N.B. votre code sera analysé pour s'assurer que vous faites la méthode demandée

Spécifications d'entrée:

En entrée vous ne recevrez rien

Spécifications de sortie:

En sortie vous devrez fournir:

- un float de l'approximation de pi avec 1 000 000 l'essai du test des coordonnées

Exemple de sortie:

```
3.141952
3.142008
3.142476
```

Explication de la première sortie:

Pour chacun des exemples, on génère des coordonnées de x et y entre -1 et 1.On regarde si la valeur est dans le cercle circonscrit, si oui on augmente le compte. Après avoir fait 1000000 exemples, on se sert du ratio pour calculer approximativement pi.

SC3: Moyennes circulaires (25 points)

Vous savez depuis plusieurs années comment faire la moyenne entre 2 points, ou même 3, ou même n. C'est facile, il suffit de faire la somme des valeurs et diviser par le nombre de valeurs. En 2D faire la moyenne de points aussi est relativement simple, il suffit de faire la moyenne selon un axe et puis la moyenne selon l'autre axe. Mais nous rendons les choses plus intéressantes en faisant la moyenne de deux angles. Nous allons mettre les valeurs en degré et n'oubliez pas que 725 degrés est équivalent à 5 degrés ou même -365 degrés. Les valeurs reviennent à 0 degrés après avoir passé 360. Vous devrez donc faire un algorithme qui calcule la moyenne d'angle et qui donne une valeur en degré qui se situe entre 0 et 360. Vous devrez avoir le centre du plus petit angle entre les deux angles fournis, donc nous ne vous donnerons PAS des angles qui ont 180 degrés d'écart.

Spécifications d'entrée:

En entrée vous ne recevrez:

- ***angles:*** un *array* des angles en degré dont vous devrez trouver la moyenne. Les angles données sont des *int*, mais peuvent être des valeurs négatives ou même des valeurs supérieures à 360.

Spécifications de sortie:

En sortie vous devrez fournir:

- un *int* de l'angle moyen arrondi à l'unité la plus près

Exemple d'entrée:

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

Exemple de sortie:

```
10
315
20
```

Explication de la première sortie:

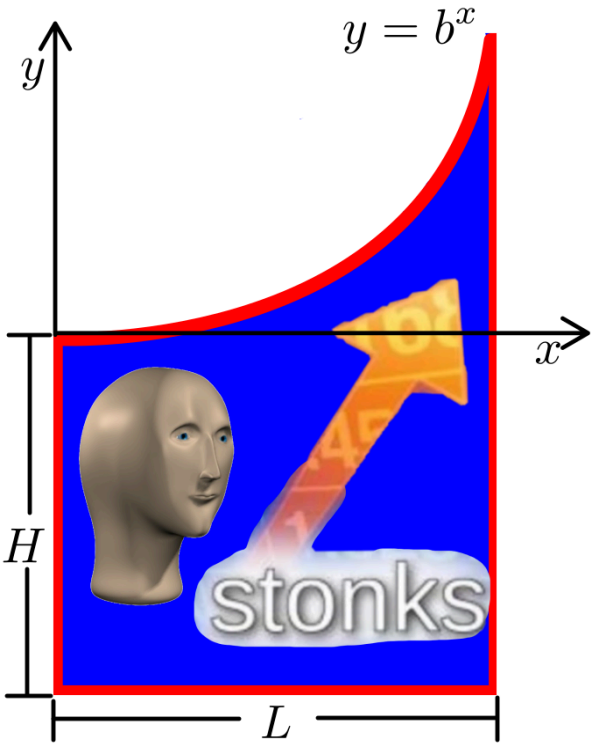
Dans le premier exemple, nous avons 355 et 735 qui peuvent aussi être interprété comme -5 et 15 degrés. L'angle moyen entre -5 et 15 degrés est 10 degrés.

SC4: Chère, chère la peinture (30 points)

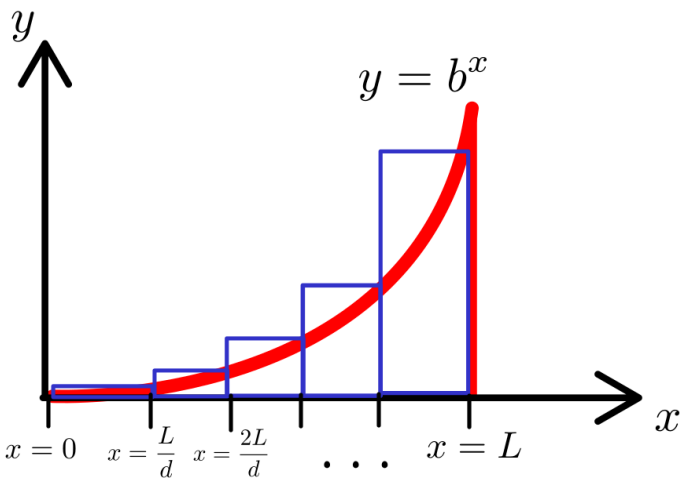
Votre cousin artiste a décroché un grand contrat ! Chargé de fabrication du logo grandeur nature d'une entreprise boursière, il est maintenant rendu à l'étape de la peinture. Il fait appel à vos services pour déterminer précisément la superficie à peindre.

La superficie à peindre est de la géométrie suivante :

Pour déterminer la superficie du logo, les dimensions suivantes sont utilisées. Vous devez déterminer la somme de l'aire du rectangle H x L et de l'aire sous la courbe $y=b^x$, sur la longueur L.



Pour calculer l'aire sous la courbe, comme vous ne connaissez pas l'intégrale de la fonction $y=b^x$, vous devez utilisez une méthode d'approximation par rectangles, dont vous pouvez calculer l'aire facilement, ainsi :



Votre programme doit pouvoir calculer l'aire en utilisant un nombre *d* donné de rectangles pour l'approximation de l'aire.

Spécifications d'entrée :

En entrée vous recevrez

- H : un *float* de la hauteur en mètres avant le début de la courbe
- L : un *float* de la largeur du mur en mètres
- b : un *float* qui est la base de la fonction exponentielle pour la courbe
- d : le nombre entier de subdivisions pour calculer l'aire sous la courbe

Spécifications de sortie :

En sortie vous devrez fournir:

- un *float* de la superficie totale à peindre en mètres carrés arrondi à la 3^{e} décimale

Exemple d'entrée :

```
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
```

Exemple de sortie :

```
20.486
1305.827
19.609
```

Explication de la première sortie :

Nous pouvons diviser le problème en 2 sections principales : la base et la partie exponentielle. La base est la plus simple, car il s'agit simplement d'un rectangle et l'aire d'un rectangle est la multiplication des 2 côtés. Donc, nous avons : base = H * L base = 2.23 * 3 = 6.69

Pour la partie exponentielle, nous pouvons aussi utiliser le rectangle à notre avantage ! Ce que nous savons, c'est que nous devons diviser la largeur du mur (L) en d (5) bandes égales. Pour nous assurer d'avoir assez de peinture, nous prenons l'estimation de la hauteur du rectangle au point le plus à droite. Pour obtenir la hauteur, nous devons trouver la valeur de bx où x est le point le plus à droite du rectangle. Pour L=3 et d=5, le premier rectangle se termine à $\frac{3}{5}$, pour le deuxième il sera à $\frac{2*3}{5}$, puis $\frac{3*3}{5}$, $\frac{4*3}{5}$ et enfin $\frac{5*3}{5}$, ce qui donne 3 (la même chose que L !). Pour chaque rectangle, nous devons trouver la hauteur, donc nous avons y1 = $(2.1)^{\frac{3}{5}} = 1.560743651$, ce qui est la hauteur du premier rectangle. Étant donné que tous les 5 rectangles ont une largeur de $\frac{3}{5}$, l'aire du premier rectangle est:

$A_1 = \frac{3}{5} * 1.560743651 = 0.93644619$

On peut appliquer la même formule et même logique pour les 4 autres rectangles de sous-division pour estimer l'aire sous la courbe:

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

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

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

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

On peut additionner l'aire de la base avec tous les rectangles d'approximation pour obtenir notre approximation de l'aire à peindre.

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

TRAITEMENT DE TEXTE (TP)

TP1: Ponctuation cryptée (40 points)

Le cryptage est un des problèmes classiques de la CRC. Par contre, cette fois-ci le texte est tout à fait correct, c'est la ponctuation que vous devrez déchiffrer! Pour déchiffrer la ponctuation, vous aurez à échanger les signes de ponctuation de place jusqu'à trouver le positionnement valide pour le texte. Le texte initial aura les signes des signes de ponctuation à chacun des endroits qui doivent en contenir un, mais ça ne sera pas nécessairement le bon qui sera présent. Vous devrez prendre les signes de ponctuation, et les décaler vers la position du prochain signe de ponctuation jusqu'à avoir une phrase valide. Le texte fournit forme une boucle donc les signes de ponctuation qui doivent être déplacés vers la fin du texte qui arrivent à la fin du texte reviennent au premier positionnement de signe de ponctuation.

Vous aurez donc un texte contenant plusieurs phrases avec des signes de ponctuation qui ne sont pas les bons au bon endroit. Pour faire ce décryptage, vous devrez utiliser un décalage de la position des signes de ponctuation en respectant leur ordre initial. Voici les règles à suivre pour s'assurer que les signes de ponctuation sont au bon endroit dans le texte.

- Les signes de ponctuation terminaux doivent être suivis d'une majuscule
- Le dernier signe de ponctuation du texte doit être un signe de ponctuation terminal
- Les autres signes de ponctuation peuvent ou non être suivi d'une lettre majuscule

Vous devrez retourner combien de décalage les signes de ponctuation doivent subir pour arriver pour la première fois dans une position valide. Pour aider au compte de déplacement des signes de ponctuation, un déplacement correspond à décaler chacun des signes de ponctuation vers la fin de la phrase et le tout dernier à la position du tout premier signe de ponctuation.

Spécifications d'entrée:

En entrée vous recevrez:

- ***text***: le texte qui contient les signes de ponctuation mal placés

Spécifications de sortie:

En sortie vous devrez fournir:

- un *int* du nombre de décalage minimal vers la fin de la phrase pour obtenir une ponctuation valide.

Exemple d'entrée:

```
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.
```

Exemple de sortie:

```
1
2
0
```

Explication de la première sortie:

Dans la première phrase, nous avons comme premier signe de ponctuation un point suivi d'une majuscule ce qui est bon. Nous avons par la suite une virgule suivi d'une majuscule ce qui est potentiellement correct. Par contre, le prochain signe de ponctuation est un point d'exclamation, mais il est suivi d'une minuscule. Ceci est problématique donc il faut déplacer les signes de ponctuation et voir si toutes les règles sont respectées. On fait donc un déplacement vers la droite et on remarque que maintenant, toutes les règles de signe de ponctuation sont valides. Il fallait donc un déplacement de 1.

TP2: Scrabble scramble (25 points)

Vous voyagez un peu partout à travers le monde et dans vos voyages vous apportez votre jeu préféré, un jeu de Scrabble! Par contre, comme vous êtes expert dans ce jeu, vous donnez la chance à votre adversaire de jouer dans la langue de son choix. C'est un grand défi que vous vous donnez, mais vous allez vous faire un programme pour vous aider un peu à trouver des mots plausibles dans d'autres langues qui vous sont inconnues. Pour ce faire, vous aurez 5 lettres avec lesquelles vous devrez trouver tous les mots potentiels pour n'importe quelle langue avec les lettres fournies.

Dans toutes les langues dans lesquelles vous allez jouer, une série de règles sont impératives à suivre pour obtenir des mots qui ont un potentiel d'être valides.

- Tous les mots doivent contenir au moins une voyelle
- Il peut y avoir au maximum 2 consonnes de suite (Ce n'est pas vrai dans toutes les langues, je sais)
- Il peut y avoir au maximum 2 voyelles de suite (Ce n'est pas vrai dans toutes les langues, je sais)
- Vous pouvez utiliser chaque lettre une seule fois dans la formation d'un mot

Vous devrez retourner la liste de tous les mots entre une et 5 lettres qui ont un potentiel d'être des mots valides dans une langue quelconque. Les mots devront être en ordre alphabétique avec les mots plus courts mis en premier.

N.B. Les y sont considérés comme des voyelles et vous avez le droit d'utiliser itertools!

Spécifications d'entrée:

En entrée vous recevrez:

- ***letters***: une *string* des 5 lettres que vous avez à votre disposition

Spécifications de sortie:

En sortie vous devrez fournir:

- un *int* de *string* qui sont tous les mots potentiels classés en ordre alphabétique.

Exemple d'entrée:

```
edcba
tests
aeiou
```

Exemple de sortie:

```
['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']
```

Explication de la première sortie:

On peut voir dans le premier exemple lorsque nous avons les lettres edcba que nous pouvons faire énormément de potentiel mots. En effet, on débute avec les mots d'une lettre qui sont les voyelles disponibles. Puis on forme les mots de 2 lettres qui sont soit 2 voyelles ou 1 voyelle et une consonne. On poursuit ainsi en s'assurant de ne pas avoir 3 consonnes consécutives ou 3 voyelles consécutives. Il y a donc des combinaisons comme abcde qui ne peuvent pas être faites comme elle contient 3 consonnes de suite.

TP3: Réflexion littérale (25 points)

Vous a-t-on déjà dit de réfléchir avant de parler? Cette solution est pour vous! Vous devrez ici effectuer deux réflexions clés une suivie de l'autre sur toute séquence de caractères donnée. D'abord, nous utiliserons un axe de réflexion vertical centré au milieu de la séquence pour inverser le placement de tous les caractères sans changer leur nature. Par exemple, "tourtour" devient "ruotruot". Ensuite, nous prendrons un axe horizontal qui coupera parfaitement tous les caractères en 2 moitié égales afin d'en changer certains. Les caractères affectés sont seulement ceux qui ont un caractère correspondant après réflexion et sont **exclusivement** ceux-ci:
"b" devient "p" et vice-versa, "d" devient "q" et vice-versa, "f" devient "t" et vice-versa, "g" devient "6" et vice-versa, "i" devient "!" et vice-versa, "j" devient "?" et vice-versa, "m" devient "w" et vice-versa, "n" devient "u" et vice-versa, "s" devient "z" et vice-versa, "2" devient "5" et vice-versa.
Les majuscules et autres caractères non-listés ici ne sont pas affectés par la seconde réflexion.

Spécifications d'entrée:

En entrée, vous recevrez une *string* à réfléchir.

Spécifications de sortie:

En sortie vous devrez fournir:

- une string réfléchie

Exemple d'entrée:

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

Exemple de sortie:

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

Explication de la première sortie:

L'axe de symétrie vertical nécessite l'inversion de la *string*, ce qui donne "ruotruot". Ensuite, "o" et "r" ne figurent pas dans la liste des caractères à inverser et ne seront donc pas touchés par la deuxième réflexion. Pour ce qui est des autres, les deux "u" deviennent des "n" et les deux "t" deviennent des "f", donnant ainsi "rnofrnof".