



**Sébastien Hoarau**

**Thierry Massart**

## Table des matières

UpyLaB 2.1.....	4
UpyLaB 2.2.....	4
UpyLaB 2.3.....	4
UpyLaB 2.4.....	5
UpyLab 2.5.....	5
UpyLab 2.6.....	7
UpyLab 2.7.....	7
UpyLab 2.8.....	8
UpyLab 3.1.....	8
UpyLab 3.2.....	8
UpyLab 3.3.....	9
UpyLab 3.4.....	9
UpyLab 3.5.....	9
UpyLab 3.6.....	10
UpyLab 3.7.....	11
UpyLab 3.8.....	11
UpyLab 3.9.....	12
UpyLab 3.10.....	12
UpyLab 3.11.....	14
UpyLab 3.12.....	14
UpyLab 3.13.....	15
UpyLab 3.14.....	15
UpyLab 4.1.....	16
UpyLab 4.2.....	16
UpyLab 4.3.....	18
UpyLab 4.5.....	19
UpyLab 4.6.....	19
UpyLab 4.7.....	20
UpyLab 4.8.....	20
UpyLab 5.1.....	21
UpyLaB 5.2.....	21
UpyLab 5.3.....	21
UpyLab 5.4.....	22
UpyLab 5.5.....	22
UpyLab 5.6.....	22
UpyLab 5.7.....	23
UpyLab 5.8.....	23
UpyLab 5.9.....	23
UpyLab 5.10.....	23
UpyLab 5.11.....	24
UpyLab 5.12.....	24
UpyLab 5.13.....	24
UpyLab 5.14.....	25
UpyLab 5.15.....	25
UpyLab 5.16.....	25
UpyLab 5.17.....	25
UpyLab 5.18.....	26
UpyLab 5.19.....	26
UpyLab 5.20.....	26
UpyLab 5.21.....	27

UpyLab 5.22.....	27
UpyLab 5.23.....	27
UpyLab 5.24.....	28
UpyLab 5.25.....	28
UpyLab 5.26.....	28
UpyLab 5.27.....	28
UpyLab 5.28.....	29
UpyLab 5.29.....	29
UpyLab 5.30.....	29
UpyLab 6.1.....	30
UpyLab 6.2.....	30
UpyLab 6.3.....	31
UpyLab 6.4.....	31
UpyLab 6.5.....	31
UpyLab 6.6.....	32
UpyLab 6.7.....	32
UpyLab 6.8.....	32
UpyLab 6.9.....	32
UpyLab 6.10.....	33
UpyLab 6.11.....	33
UpyLaB 6.12.....	34
UpyLaB 6.13.....	35
UpyLaB 6.14.....	36
UpyLaB 6.15.....	37

## UpyLaB 2.1

Assignez la valeur

- entière 36 dans la variable x;
- entière 36 fois 36 dans la variable produit;
- entière avec le résultat de la division entière de 36 par 5 dans la variable div\_entiere;
- entière 15 exposant 15 à la variable expo;
- float 3.14159 à la variable pi
- "Bonjour" à la variable chaîne de caractères mon\_texte

Attention : aucun print() n'est demandé dans cet exercice !

## UpyLaB 2.2

Écrire un programme qui imprime la moyenne arithmétique  $\frac{a+b}{2}$  de deux nombres de type float lus sur input.

Votre code à tester par UpyLaB ne doit pas avoir de paramètre dans les input.

Exemple: `a = float(input())` plutôt que `a = float(input("a = "))`

Notez que lorsque vous testez votre code Python, dans la colonne Résultats de la fenêtre UpyLaB, le caractère "↵" représente le caractère "passage à la ligne" appelée également "retour chariot" ("return" ou "carriage return" en anglais).

## UpyLaB 2.3

[La règle de trois](#) est une méthode pour trouver le quatrième terme parmi quatre termes ayant un même rapport de proportion  $\frac{a}{b} = \frac{c}{d}$ , lorsque trois de ces termes sont connus.

Par exemple si chacun mange autant de chocolat et que pour 4 personnes il en faut 100 grammes, pour 7 personnes il en faudra donc  $\frac{7}{4} \cdot 100$  grammes = 175 grammes.

Dans cet exemple  $a=4$ ,  $b=100$ ,  $c=7$  et  $d=175$  : la règle est que l'on veut que chacun ai la même quantité de chocolat; donc il faut respecter le rapport  $\frac{4}{100} = \frac{7}{175}$

Elle utilise le fait que le produit des premier et quatrième termes est égal au produit du second et du troisième:  $a \cdot d = b \cdot c$  et donc  $d = \frac{cb}{a}$

- Écrire un programme qui lit les valeurs de type float pour  $a$ ,  $b$  et  $c$  et qui affiche la valeur de  $d$
- correspondant à la règle de trois.
- Votre code à tester par UpyLaB ne doit pas avoir de paramètre dans les input. Exemple: `a = float(input())` plutôt que `a = float(input("a = "))`
- De même, votre code à tester par UpyLaB ne doit imprimer que le résultat numérique de  $d$
- , pas de texte supplémentaire. Exemple: `print(d)` plutôt que `print("Résultat : ", d)`

- Le but de cet exercice est de vous familiariser avec la lecture (input()) de données et l'impression (print()) de résultats.

## UpyLaB 2.4

Écrire un programme qui lit deux nombres a et b de type float sur input et qui calcule et affiche le nombre c tel que b soit la moyenne arithmétique de a et c.

Votre code à tester par UpyLaB ne doit pas avoir de paramètre dans les input. Exemple: `a = float(input())` plutôt que `a = float(input("a = "))`

De même, votre code à tester par UpyLaB ne doit imprimer que le résultat numérique de c, pas de texte supplémentaire. Exemple: `print(c)` plutôt que `print("Résultat : ", c)`

**Conseil si vous ne trouvez pas la solution:**

Ici on a l'équation simple  $b = \frac{a+c}{2}$

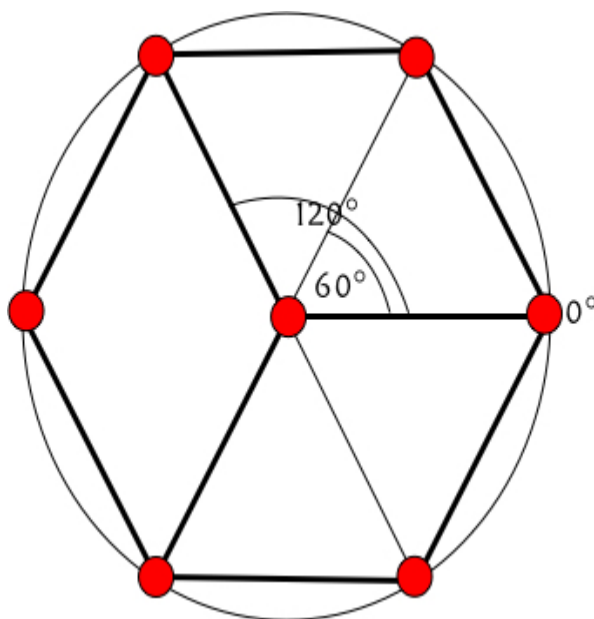
où a et b sont connus par le programme et c est l'inconnue. Avant d'écrire du code, vous devez donc d'abord faire un peu de mathématiques pour déterminer ce que vaut c en fonction de a et de b.

Si nécessaire, pour vous rafraîchir la mémoire, faites une recherche sur internet: "techniques résolution equation simple".

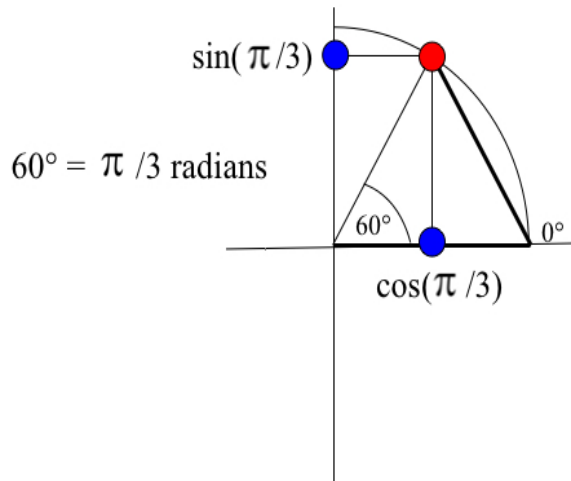
## UpyLab 2.5

Écrire un programme, qui lit une longueur long de type float, strictement positive et qui affiche les valeurs x y des coordonnées (x, y) des sommets de l'hexagone de centre (0,0) et de rayon long, dans l'ordre donné par le point à 0° ensuite 60°, 120° jusqu'au 6ème point). Chaque couple de coordonnées sera affiché sur une ligne différente.

**Notez que votre code ne doit pas tester si la valeur lue est bien strictement positive.**



Pour les calculs utilisez la valeur pi et les fonctions sin et cos du module math, comme indiqué sur la figure ci-dessous.



### Conseils:

- Il ne vous est **pas** demandé de tracer l'hexagone, mais uniquement d'afficher des valeurs. Notez qu'UpyLaB ne supporte pas le module turtle et donc ne l'importez pas dans votre code pour cet exercice !
- Avant de le tester dans UpyLaB, développez et testez votre code d'abord dans un environnement de développement : PyCharm ou un autre. Pour un rayon de longueur 100.0
- , la réponse vaut environ:

100.0 0.0

50.0 86.602

-50.0 86.602

-100.0 0.0

-50.0 -86.602

50.0 -86.602

- Vous constaterez que l'utilisation des fonctions `sin()` et `cos()` et les erreurs d'arrondis et de troncature durant les calculs induiront que les résultats peuvent être un peu différents du résultat théorique; UpyLaB laisse une certaine latitude dans les résultats (ici de l'ordre de  $10^{-5}$  ).
- **Votre code à tester par UpyLaB ne doit pas avoir de paramètre dans les input. Exemple: `x = int(input())` plutôt que `x = int(input("x = "))`**

## UpyLab 2.6

Écrire un programme qui imprime le volume d'une sphère de rayon  $r$ ; le rayon étant lu sur input.

- La formule de calcul du volume d'une sphère de rayon  $r$  est donné par :

$$\text{volume} = \frac{4}{3}\pi r^3$$

**Notez que votre code ne doit pas tester si la valeur lue est bien positive ou nulle.**

- Votre code à tester par UpyLaB ne doit pas avoir de paramètre dans les input. Exemple, écrivez :

```
r = float(input())
```

plutôt que

```
r = float(input("rayon = "))
```

- Pour avoir une valeur assez proche de la vraie valeur de  $\pi$ , nous vous conseillons, en début de code, d'importer la valeur  $\pi$  de la librairie mathématique (avec : `from math import pi`)

## UpyLab 2.7

Écrire un programme, qui lit deux valeurs entières  $x$  et  $y$  strictement positives suivies de deux valeurs réelles (float)  $z$  et  $t$  et qui affiche (on dit aussi imprime) les valeurs des expressions suivantes (où ici  $x.z$  par exemple, indique le produit de  $x$  et  $z$ ), chacune sur une nouvelle ligne :

$x-y$

$x+z$

$z+t$

$x.z$

$\frac{x}{2}$

$\frac{x}{y+1}$

$\frac{(x+y).z}{4.x}$

$x^{-\left(\frac{1}{2}\right)}$  (c'est-à-dire  $x$  exposant  $-12$ )

**Notez que votre code ne doit pas tester si les valeurs de  $x$  et  $y$  sont bien strictement positives.**

Le but de cet exercice est de vous familiariser avec la syntaxe Python pour écrire des expressions arithmétiques simples et avec l'instruction `print` qui affiche (on dit aussi imprime) des valeurs à l'écran.

Votre code à tester par UpyLaB ne doit pas avoir de paramètre dans les input. Exemple: `x = int(input())` plutôt que `x = int(input("x = "))` et ne doit imprimer que les valeurs résultats.

## UpyLab 2.8

Écrire un programme affichant les **quatre** lignes suivantes (sans les espaces en début de chaque ligne) :

```
Hello World
Aujourd'hui
C'est "Dommage!"
Hum \o/
```

Sans lignes vides supplémentaires ou espaces supplémentaires en fin de ligne.

Le but de cet exercice est de vous familiariser avec l'impression de certains caractères particuliers comme les simples et double quotes, la "backslash" (la barre oblique inverse), ....

- Attention aux détails: les simples et doubles quotes (apostrophes) sont droites, le nombre d'espaces entre les mots, ....

## UpyLab 3.1

Écrire un programme qui lit 3 nombres entiers, et qui, si au moins deux d'entre eux ont la même valeur, imprime cette valeur (le programme n'imprime rien dans le cas contraire).  
Comme d'habitude, votre code à tester par UpyLaB

- ne doit pas avoir de paramètre dans les input.  
Exemple: `a = float(input())` plutôt que `a = float(input("a = "))`
- ne doit pas dans les print, imprimer autre chose que le résultat.  
Exemple: `print(a)` plutôt que `print("Résultat : ", a)`

## UpyLab 3.2

Écrire le morceau de code qui si, a (entier lu sur input) est supérieur à 0, teste si a vaut 1, auquel cas il imprime le texte :

"a vaut 1"

et qui, si a n'est pas supérieur à 0, imprime le texte :

"a est inférieur ou égal à 0"

Dans les autres cas, votre code **n'imprime rien**

**Notes:**

- Lors de l'affichage des résultats, en cas d'erreur dans certains tests, UpyLaB pourra marquer

**"Le résultat attendu était :**

**aucun résultat"**

Cela voudra bien dire qu'il ne faut rien imprimer dans ce cas.

- Votre code à tester par upylab ne doit pas avoir de paramètre dans les input. Exemple:

```
a = int(input())
```



plutôt que

```
a = int(input("a = "))
```

## UpyLab 3.3

Écrire le programme qui lit en input trois entiers  $a, b$  et  $c$ . Si l'entier  $c$  est égal à 1, alors le programme affiche sur output (imprime) la valeur de  $a+b$  ; si  $c$  vaut 2 alors le programme affiche la valeur de  $a-b$  ; si  $c$  est égal à 3 alors l'output sera la valeur de  $a.b$  (produit de  $a$  par  $b$ ). Enfin, si la valeur 4 est assignée à  $c$ , alors le programme affiche la valeur de  $a^2+a.b$  ( $a$  au carré plus le produit de  $a$  et  $b$ ). Si  $c$  contient une autre valeur, le programme affiche le message "Erreur".

Votre code à tester par upylab ne doit pas avoir de paramètre dans les input. Exemple: `a = int(input())` plutôt que `a = int(input("a = "))`

## UpyLab 3.4

Ecrivez un programme qui teste la parité d'un nombre entier lu sur input et imprime True si le nombre est pair, False dans le cas contraire.

## UpyLab 3.5

Ecrivez un programme qui teste si deux nombres entiers,  $a$  et  $b$  strictement supérieurs à 0, lus sur input sont tels que :

- $a$  ne divise pas entièrement  $b$  sans reste et
- $a$  n'est pas entièrement divisé par  $b$  sans reste.

Dans ce cas le programme imprime True ;

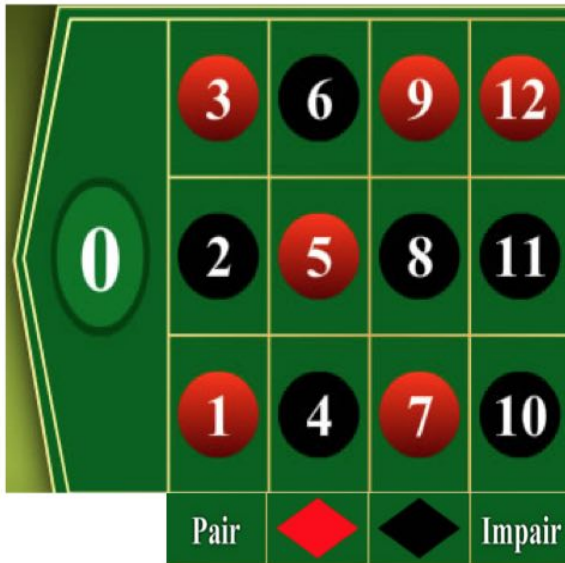
sinon il imprime False.

- Une définition de la division entière appelée également division euclidienne est donnée en cliquant [sur le lien ici](#)

**Notez que votre code ne doit pas tester si les valeurs  $a$  et  $b$  sont bien strictement positives.**

## UpyLab 3.6

Dans mon casino, ma roulette comporte 13 valeurs de 0 à 12 telles que montrées par la figure.



Les mises possibles et les retours correspondants quand le pari est gagné sont donnés ci-dessous :

Mise	retour si gagné
Numéro exact (0 à 12)	12 fois la mise
Pair ou Impair	2 fois la mise
Rouge ou Noir	2 fois la mise
Pour simplifier, on suppose que le zéro est ni rouge ni noir, mais est pair.	

Écrire un programme qui lit ce que mise le joueur parmi :

- 0, 1, ..., 12
- 13 pour "Pair"
- 14 pour "Impair"
- 15 pour "Rouge"
- 16 pour "Noir"

et qui, après avoir actionné la roulette et reçu un nombre  $x$  entre 0 et 12, imprime le retour correspondant, soit:

- 12 si le numéro exact est trouvé,
- 2 si Pair/Impair ou Rouge/Noir est gagné,
- 0 si le pari est perdu.

Par exemple si la mise est 14 ("Impair") et que 7 est sorti par la roulette, le résultat est : 2

Comme malheureusement nous n'avons pas de roulette à notre disposition, pour tester votre code, celui-ci débutera par les deux premières lignes suivantes :

```
a = int(input()) # reçoit la mise du joueur entre 0 et 16
```

```
x = int(input()) # reçoit la valeur du tirage entre 0 et 12
```






## UpyLab 3.7

Écrire un programme qui imprime la [moyenne géométrique](#)  $\sqrt{a \cdot b}$  (la racine carrée du produit de  $a$  par  $b$ ) de deux nombres positifs de type float lus sur input. Si au moins un des deux nombres est négatif, imprime "Erreur". **Votre code à tester par upylab ne doit pas avoir de paramètre dans les input. Exemple: `a = float(input())` plutôt que `a = float(input("a = "))`**

**Notez que votre code ne doit pas tester si les valeurs  $a$  et  $b$  sont bien positives ou nulles.**

## UpyLab 3.8

Les cinq [polyèdres réguliers de Platon](#) sont les suivants avec la formule de leur volume correspondant.

nom	volume	image
Tétraèdre	$\frac{\sqrt{2}}{12} a^3$	
Cube	$a^3$	
Octaèdre	$\frac{\sqrt{2}}{3} a^3$	
Dodecaèdre	$\frac{(15+7\sqrt{5})}{4} a^3$	
Icosaèdre	$\frac{5(3+\sqrt{5})}{12} a^3$	

Écrire un programme qui lit

- la première lettre en majuscule ("T", "C", "O", "D" ou "I") du polyèdre
- la taille d'une arête

et qui imprime le volume du polyèdre correspondant.

Si la lettre lue n'est pas une des 5 lettres, votre programme imprime "Polyèdre non connu".

**Votre code à tester par upylab ne doit pas avoir de paramètre dans les input. Exemple: `a = float(input())` plutôt que `a = float(input("a = "))`**

## UpyLab 3.9

Écrire un programme qui calcule la taille moyenne (en nombre de personnes) des Petites et Moyennes Entreprises de la région, les tailles étant données en input chaque valeur sur une ligne différente; la fin des données étant signalée par la "valeur sentinelle" -1 **qui ne fait pas partie des éléments à comptabiliser** mais indique que l'ensemble des valeurs a été donné.

On suppose aussi que la suite des tailles contient toujours au moins un élément avant la valeur sentinelle -1 et que toutes ces valeurs sont positives ou nulles. Après l'entrée de la valeur sentinelle, le programme affiche sur la ligne suivante la valeur de la moyenne arithmétique.

Exemple d'exécution : avec les valeurs lues suivantes :

- 11
- 8
- 14
- 5
- -1

le résultat que doit donner votre code est :

9.5

Votre code à tester par upylab ne doit pas avoir de paramètre dans les input. Exemple : `a = int(input())` plutôt que `a = int(input("a = "))`

## UpyLab 3.10

Écrivez un mini jeu : le programme génère de manière (pseudo-) aléatoire un nombre naturel (nombre secret) dans l'intervalle entre 0 et 100.

- Ensuite, le joueur doit deviner ce nombre en utilisant le moins d'essais possible.
- A chaque tour le joueur peut faire un essai et le programme doit donner une parmi les réponses suivantes:
  - "Trop grand" : Si le nombre secret est plus petit et qu'on n'est pas au maximum d'essais
  - "Trop petit" : Si le nombre secret est plus grand et qu'on n'est pas au maximum d'essais
  - "Gagné en n essais !" : Si le nombre secret est trouvé
  - "Perdu ! Le secret était", nombre : Si vous avez fait 6 essais sans trouver le nombre secret
- Le joueur a maximum 6 essais ; s'il ne devine pas le secret après 6 essais, le programme s'arrête après avoir écrit "Perdu ! Le secret était", nombre"
- Exemple d'exécution gagnante (après la génération du nombre à deviner):
  - 50
  - Trop grand
  - 8
  - Trop petit
  - 20
  - Trop petit
  - 27
  - Gagné en 4 essais !

- Exemple d'exécution perdante (après la génération du nombre à deviner):
- 50
- Trop grand
- 24
- Trop petit
- "37
- Trop petit
- 43
- Trop grand 40
- Trop petit
- 41
- Perdu ! Le secret était 42

**Attention: Au dernier essai, votre programme ne doit ni afficher "Trop petit" ni "Trop grand", mais le verdict comme expliqué plus haut !**

- Pour qu'Upylab puisse tester si votre solution est correcte, il faut que vous respectiez strictement cette séquence. Si par exemple, vous n'affichez pas Trop petit ou Trop grand, le nombre suivant ne sera pas fourni par le système et votre solution sera considérée comme incorrecte
- En pratique, vous devez débiter votre code comme suit :
- `import random`
- `NB_ESSAIS_MAX = 6`
- `secret = random.randint(0,100)`

et ne pas faire d'autre appel à `randint` ou à une autre fonction de `random`

Conseil pour le débogage de votre code: le programme test d'UpyLaB utilise l'argument entier affiché en sortie, comme seed (graine) du module `random`. En clair, cela signifie qu'après avoir importé `random`, l'instruction :

```
random.seed(argument)
```

est réalisé par UpyLaB avant d'exécuter votre propre code :

```
...
```

```
secret = random.randint(0,100)
```

qui fournit le nombre à deviner.

**Attention: l'argument n'est donc pas le nombre à deviner, mais bien ce qui permet au programme de le générer.**

En faisant de même, vous pouvez répliquer une exécution test.

## UpyLab 3.11

On peut calculer approximativement le sinus de  $x$  ([voir définition du sinus](#)) en effectuant la sommation des  $n$  premiers termes de la série (c'est-à-dire la somme infinie) :

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

où  $x$  est exprimé en radians.

On vous demande d'écrire un code qui lit une valeur flottante (float)  $x$  en entrée et qui imprime une approximation de  $\sin(x)$

.

Votre code additionne les termes successifs dans la série jusqu'à ce que la valeur d'un terme devienne inférieure (en valeur absolue) à une constante  $\epsilon$

(prenez  $\epsilon = 10^{-6}$

). Affichez (imprimé) ensuite l'approximation ainsi obtenue.

**Attention :** Calculer explicitement la valeur des factorielles peut poser des soucis lorsque vous utilisez les valeurs pour des calculs avec des float. Si c'est le cas, pensez à une autre façon de faire !

Votre code à tester par UpyLaB ne doit pas avoir de paramètre dans les input. Exemple : `x = float(input())` plutôt que `x = float(input("x = "))`

## UpyLab 3.12

Écrire un programme qui lit sur input une valeur naturelle  $n$  et qui affiche à l'écran un carré de  $n$  caractères X (majuscule) de côté, comme suit (pour  $n = 6$ ):

```
XXXXXX
XXXXXX
XXXXXX
XXXXXX
XXXXXX
XXXXXX
```

**Votre code ne doit pas tester si la valeur  $n$  est bien positive ou nulle.**

Votre code à tester par upylab ne doit pas avoir de paramètre dans les input. Exemple : `n = int(input())` plutôt que `n = int(input("n = "))`

## UpyLab 3.13

Variante de l'exercice "Carré de 'X'", afficher le triangle supérieur droit, comme suit (pour  $n = 6$ ):

```
XXXXXX
XXXXX
XXXX
XXX
XX
X
```

**Notez que votre code ne doit pas tester si la valeur  $n$  est bien positive ou nulle.**

Votre code à tester par upylab ne doit pas avoir de paramètre dans les input. Exemple :  $n = \text{int}(\text{input}())$  plutôt que  $n = \text{int}(\text{input}("n = "))$

## UpyLab 3.14

Il est demandé d'additionner des valeurs naturelles lues sur input et d'imprimer le résultat.

La première donnée lue ne fait pas partie des valeurs à sommer. Elle détermine si la liste contient un nombre déterminé à l'avance de valeurs à lire ou non.

- si cette première donnée vaut un nombre positif ou nul, ce nombre donne le nombre de valeurs à lire et à sommer
- si la donnée est négative, cela signifie que la liste des données à lire sera terminée par un caractère "F" signifiant que la liste est terminée

**Exemple 1:** les données lues suivantes:

```
4
1
3
5
7
```

indiquent qu'il y a 4 données à sommer :  $1 + 3 + 5 + 7$ .

Le résultat à imprimer vaudra donc 16

**Exemple 2:** les données lues suivantes:

```
-1
1
3
5
7
21
F
```

indiquent qu'il faut sommer :  $1 + 3 + 5 + 7 + 21$ . Le résultat à imprimer vaudra donc 37.

**Exemple 3:** La donnée 0 indique qu'il faut sommer 0 nombre.

Le résultat à imprimer vaudra donc 0.

### Conseils:

- Dans la cas où la liste est terminée par le caractère 'F', vous pouvez d'une part faire l'input (ex : `data = input()`), ensuite tester si `data == 'F'` et sinon additionner la valeur `int(data)` à ce qui a déjà été sommé;
- Utilisez un `for` dans le cas où le nombre de valeurs à sommer est connu, un `while` dans le cas contraire.

### Consignes:

- Ne mettez pas de paramètre texte dans les input: `data = input()` et pas `data = input("Donnée suivante :")` par exemple;
- Le résultat doit juste faire l'objet d'un `print(res)` sans texte supplémentaire (pas de `print("résultat =", res)` par exemple).

## UpyLab 4.1

Écrire une fonction `deux_egaux(a, b, c)` qui reçoit 3 nombres entiers ou float en paramètre, et qui, renvoie **la valeur booléenne** `True` si au moins deux d'entre eux ont la même valeur et **la valeur booléenne** `False` sinon.

Par ailleurs, votre programme lira 3 données de type `int` `x`, `y` et `z` et imprimera le résultat de l'exécution de `deux_egaux(x, y, z)`

- Votre code à tester par UpyLaB
- ne doit pas avoir de paramètre dans les input.  
Exemple: `x = int(input())` plutôt que `x = int(input("x = "))`
- ne doit pas dans les print imprimer autre chose que le résultat (dans ce cas-ci `True` ou `False`)  
Exemple: `print(deux_egaux(x, y, z))` plutôt que `print("Résultat : ", deux_egaux(x, y, z))`
- dans la fonction `deux_egaux` ne doit pas tester le type des paramètres reçus

## UpyLab 4.2

De Wikipedia (5 février 2019) : En mathématiques, et plus particulièrement en combinatoire, les **nombres de Catalan** forment une suite d'entiers naturels utilisée dans divers problèmes de dénombrement. Ils sont nommés ainsi en l'honneur du mathématicien belge Eugène Charles Catalan (1814-1894). Les dix premiers nombres de Catalan (pour `n` de 0 à 9) sont

1,1,2,5,14,42,132,429,1430,4862

Le nombre de Catalan d'indice `n`, appelé `n`-ième nombre de Catalan, est défini par :

$$C_n = \frac{(2n)!}{(n+1)!n!}$$

où `n!` est la factorielle de la valeur entière `n` :

$$n! = n.(n-1).(n-2)....1$$

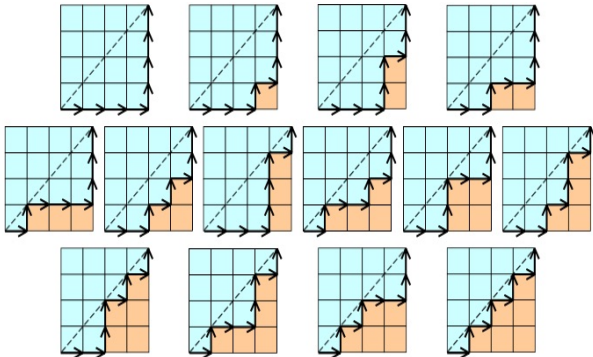
Par exemple : `5!=5.4.3.2.1` c'est-à-dire 120



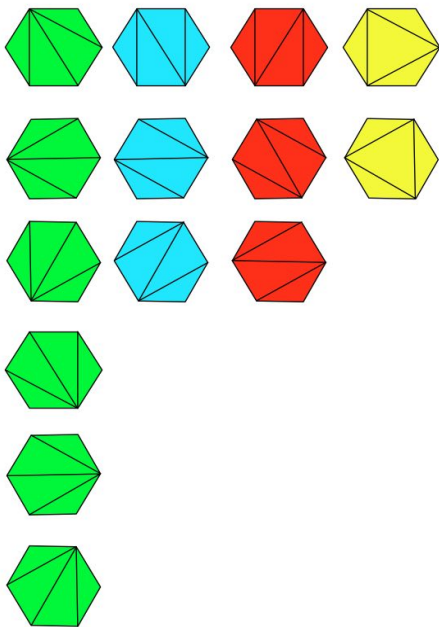
Le nombre de chemins les plus courts dans une grille de taille  $n \times n$ , le nombre de configurations de triangles dans un polygône convexe à  $n+2$  côtés ou encore le nombre de configurations possibles d'expressions avec  $n$  paires de parenthèses, appelé également mot de Dyck de longueur  $2n$ , sont des exemples dont le résultat est donné par le nombre de Catalan  $C_n$

### Exemples d'applications au calcul de $C_n$ (ici $n = 4$ )

nombre de chemins les plus courts dans une grille de taille  $n \times n$



Configurations de triangles dans un polygône convexe de taille  $n+2$



Mots de Dyck (parenthésages possibles)

```
((((( ))) (( ( ( )) ) (( ( ) ) ( ) ( ( )) ) ( ( ( ( )) )
( ( )) ) (( )) ) ( ( )) ( ( )) ( ( )) ( ( )) ( ( )) ( ( ))
( ( )) ( ( )) ( ( )) ( ( )) ( ( )) ( ( )) ( ( )) ( ( ))
```

Ecrivez une fonction `catalan(n)` où  $n$

est positif ou nul, qui renvoie la valeur du  $i$ -ème nombre de Catalan.

Ecrivez un code principal qui lit la valeur de  $n$  (int) sur input et affiche le résultat de votre fonction appelée avec le paramètre  $n$  : `print(catalan(n))`.

**Votre fonction ne doit pas tester si la valeur du paramètre n est bien positive ou nulle.**

Attention: le code de votre fonction catalan ne peut contenir d'input() (ceux-ci interféreraient avec les test d'UpyLaB)

Votre code à tester par UpyLaB ne doit pas avoir de paramètre dans l'es 'input. Exemple: `n = int(input())` plutôt que `n = int(input("n = "))`

De même, votre code à tester par UpyLaB ne doit imprimer que le résultat numérique de `catalan(n)`, pas de texte supplémentaire. Exemple: `print(catalan(n))` plutôt que `print("Résultat : ", catalan(n))`

## UpyLab 4.3

Écrivez une fonction `fibonacci(n)` qui reçoit un nombre entier :  $n$  en paramètre, et qui, renvoie le  $n$ -ième nombre de Fibonacci  $F_n$  avec

- $F_0$  valant 0
- $F_1$  valant 1
- $F_{i+1}$  valant  $F_i + F_{i-1}$
- $F_n$  valant None si  $n < 0$
- Par ailleurs, écrivez le code principal: votre programme lira une donnée entière  $x$  de type int et imprimera le résultat de l'exécution de `fibonacci(i)` pour  $i$  allant de 0 compris à  $x$  non compris avec chaque valeur sur une ligne séparée.

Votre code à tester par UpyLaB :

- ne doit pas avoir de paramètre dans les input.  
Exemple: `x = int(input())` plutôt que `x = int(input("x = "))`
- ne doit pas dans les print imprimer autre chose que les résultats.

Exemple:

`print(fibonacci(i))`

plutôt que

`print("Résultat : ", fibonacci(i))`

## UpyLab 4.5

Soit l'équation du second degré

$$ax^2+bx+c=0$$

paramétrée par  $a$ ,  $b$  et  $c$ . Écrivez une fonction `rac_eq_2nd_deg` prenant 3 paramètres qui calcule et renvoie la ou les solutions s'il y en a. Le résultat sera un tuple avec les racines :

- Si il n'y a pas de solution réelle, renvoyez un tuple vide :

```
return tuple()
```

- Si il y a une seule racine  $r_1$ , terminez votre code en renvoyant le résultat avec

```
return (r1,)
```

- Dans le cas où il y a deux solutions réelles  $r_1$  et  $r_2$ , la plus petite devra être dans la première composante du tuple renvoyé (composante d'indice 0). Vous pouvez utiliser

```
return (min(r1,r2), max(r1,r2)).
```

Attention: nous ne vous demandons pas le code qui lit des données, appelle la fonction, imprime des résultats, ....

## UpyLab 4.6

Dans le module `random`, la fonction `randint(a, b)` renvoie un nombre entier aléatoire compris entre  $a$  et  $b$  (inclus). Écrivez une fonction `alea_dice` qui génère 3 nombres aléatoires représentant 3 dés à jouer (à six faces avec les valeurs 1 à 6) et qui renvoie **la valeur booléenne** `True` si les dés forment un 421, et **la valeur booléenne** `False` sinon.

Prenez garde que toute combinaison des trois dés avec un 4, un 2 et un 1 forme 421 quelque soit l'ordre de la combinaison (par exemple 2, 1 et 4 forme bien un 421).

- En pratique, pour tester votre fonction sur UpyLaB, celui-ci recevra comme **paramètre**, un nombre entier  $s$ , et générera, grâce à la méthode `seed` de la librairie `random`, les nombres secrets.

Les premières lignes de votre codes seront donc :

```
def alea_dice(s):
```

```
    random.seed(s)
```

```
    d1 = random.randint(1,6)
```

```
    d2 = random.randint(1,6)
```

```
    d3 = random.randint(1,6)
```

- Attention: nous ne vous demandons pas le code qui lit des données, appelle la fonction, imprime des résultats, ....

## UpyLab 4.7

Considérons les billets et pièces de valeurs suivantes : 20 euros, 10 euros, 5 euros, 2 euros, 1 euros. Écrivez une fonction `rendre_monnaie` qui prend en paramètres un entier prix et cinq valeurs entières `x20`, `x10`, `x5`, `x2`, `x1` représentant le nombre de billets et de pièces de chaque sorte que donne un client pour payer l'objet dont le prix est mentionné. La fonction doit renvoyer cinq valeurs représentant la somme qu'il faut rendre au client, décomposée en billets et pièces (dans le même ordre que précédemment). La décomposition doit être faite en rendant le plus possible de billets et pièces de grosses valeurs (éventuellement avec d'autres billets que ceux apportés par le client; on suppose qu'il y a toujours assez de billets chez le vendeur).

Pour renvoyer les cinq valeurs, vous utilisez l'instruction:

```
return res20, res10, res5, res2, res1
```

où les cinq variables `res20` à `res1` contiennent les cinq valeurs à renvoyer (`res20` contient le nombre de billets de 20 à remettre, ..., `res1` le nombre de pièces de 1 à remettre).

S'il manque de l'argent, la fonction renverra cinq valeurs `None`.

**Attention:** nous ne vous demandons pas le code qui lit des données, appelle la fonction, imprime des résultats, ....

**Remarque:** au chapitre suivant, nous verrons la notion de tuple. Au niveau du langage Python, nous verrons que `return res20, res10, res5, res2, res1` renvoie en réalité un tuple de cinq valeurs. Pour l'instant, ne vous préoccupez pas de ceci,

## UpyLab 4.8

Écrivez une fonction `sum(a, b)` qui prend 2 valeurs entières et renvoie la somme de  $a$  et de  $b$ . Par défaut, la valeur de  $a$  est 0 et la valeur de  $b$  est 1.

- **Attention:** nous ne vous demandons pas un programme complet; en particulier vous ne devez pas donner le code qui lit des données, appelle la fonction, imprime des résultats, ....

## UpyLab 5.1

Écrivez une fonction `distance_points()` qui, étant donnés deux points  $(x_1, y_1)$  et  $(x_2, y_2)$  reçus en paramètres sous forme de tuples, calcule et renvoie la distance euclidienne entre ces deux points. Pour rappel, la distance entre deux points  $(x_1, y_1)$  et  $(x_2, y_2)$  se calcule comme suit:

$$dist = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

où  $\sqrt{a}$  est la racine carrée de  $a$  et correspond à  $a^{1/2}$  ( $a$  exposant un demi).

## UpyLab 5.2

Écrivez une fonction `longueur(*points)` qui prend en paramètres un nombre arbitraire de points de coordonnées  $(x, y)$  et calcule la longueur du trait correspondant. Pour calculer la longueur, il faut effectuer la somme de la longueur des segments qui composent le trait. Un segment de droite est composé de deux points consécutifs passés en paramètre. Pour rappel, la distance entre deux points  $(x_1, y_1)$  et  $(x_2, y_2)$  se calcule comme suit:

$$dist = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Pour simplifier, on suppose qu'il y a au moins 2 points passés en paramètre lors des appels à la fonction `longueur`

Conseil: utilisez la fonction `distances_points(p1, p2)` définie avant.

## UpyLab 5.3

On vous demande d'écrire une fonction `plus_grand_bord(w)` qui, étant donné un mot  $w$ , retourne le plus grand bord de ce mot. On dit qu'un mot  $u$  est un bord de  $w$ , si  $u$  est à la fois un préfixe strict (c'est-à-dire un texte avec la première partie du texte  $w$  non vide et non égal à  $w$  lui-même) de  $w$  et un suffixe strict (c'est-à-dire un texte avec la dernière partie du texte  $w$  non vide et non égal à  $w$  lui-même) de  $w$ . Si  $w$  n'a pas de bord, la fonction retourne une chaîne de caractères vide.

Exemple : 'a' et 'abda' sont des bords de 'abdabda'

En effet:

'**a**bdabda' à la fois commence et se termine par 'a'

De même 'abdabda' à la fois commence et se termine par 'abda'('ab**da**bdabda' et 'abdab**da**')

Le plus grand bord est 'abda'.

## UpyLab 5.4

Ecrire une fonction `trans(text, replaceA, replaceB)`, qui reçoit trois paramètres:

- `text` : une chaîne de caractères
- `replaceA` : un couple (`oldA`, `newA`) où `oldA` est un caractère et `newA` un texte
- `replaceB` : un couple (`oldB`, `newB`) où `oldB` est un caractère et `newB` un texte et qui renvoie le résultat de la transformation suivante : chaque occurrence du symbole "`oldA`" dans la chaîne `text` est remplacée par la chaîne "`newA`", et simultanément chaque occurrence du symbole "`oldB`" est remplacée par la chaîne "`newB`".

Pour simplifier, vous pouvez supposer que "`oldA`" est différent de "`oldB`"

### Exemple:

```
>>> print(trans('ABBAB', ('A', 'AB'), ('B', 'BA')))  
>>> 'ABBABAABBA'
```

## UpyLab 5.5

Écrivez une fonction `duree` qui prend deux paramètres `debut` et `fin`. Ces derniers sont des couples (tuples de 2 composantes) dont la première composante représente une heure et la seconde composante représente les minutes. Cette fonction doit calculer le nombre d'heures et de minutes qu'il faut pour passer de `debut` à `fin`.

- Exemple : un appel à `duree` (`((14, 39), (18, 45))`) renvoie `(4, 6)` pour 4 heures et 6 minutes.
- Notez qu'un appel à `duree` (`((6, 0), (5, 15))`) renvoie `(23, 15)` pour 23 heures et 15 minutes, et non `(0, 45)` !

## UpyLab 5.6

Voici le début d'une suite logique inventée par John Horton Conway (et donc appelée [suite de Conway](#)) :

```
1  
11  
21  
1211  
111221  
312211  
...
```

Chaque ligne à partir de la deuxième décrit la précédente...

- La première ligne : 1,
- on l'a décrit "Un 1" ce qui donne la deuxième : 11,
- la troisième décrit la deuxième : "Deux 1" : 21
- la quatrième décrit la troisième : "Un 2 suivi de Un 1" : 1211
- et ainsi de suite.

On vous demande d'écrire une fonction `next_line(line)` qui prend une liste d'entiers décrivant une ligne de cette suite et qui calcule et retourne la liste correspondant à la ligne suivante. Par exemple :

```
next_line([1,2,1,1])
```

retourne la liste `[1,1,1,2,2,1]`

**Par convention, `next_line([])` retourne la liste `[1]`**

## UpyLab 5.7

Ecrivez une fonction `my_pow` qui prend comme paramètres un nombre entier  $m$  et un nombre flottant  $b$  et qui renvoie une liste contenant les  $m$  premières puissances de  $b$ , c'est-à-dire une liste contenant les éléments allant de  $b^0$  à  $b^{m-1}$ .

Si le type des paramètres n'est pas celui attendu, votre fonction renverra la valeur `None`

## UpyLab 5.8

- Veuillez écrire une fonction `anagrammes(v, w)` qui renvoie la valeur booléenne `True` si les mots  $v$  et  $w$  sont des anagrammes, c'est-à-dire des mots qui comprennent les mêmes lettres, en même quantité, mais pas nécessairement dans le même ordre. Par exemple, 'marion' et 'romina' sont des anagrammes.

La fonction renvoie la valeur booléenne `False` sinon

**Notes :**

- Votre code ne doit pas tester si les mots existent bien dans un quelconque dictionnaire
- On suppose que tout mot est anagramme avec lui même (par exemple: `anagrammes('jour', 'jour')` renvoie `True`)

## UpyLab 5.9

Nous vous demandons d'écrire une fonction `intersection(v, w)` qui calcule l'intersection entre deux chaînes de caractères  $v$  et  $w$ .

On définit l'intersection de deux mots comme étant la plus grande partie commune à ces deux mots. Par exemple, l'intersection de "programme" et "grammaire" est "gramm", et `intersection("salut", "rien")` renvoie le string vide "", puisqu'aucun caractère n'est commun.

Si plusieurs solutions sont possibles, prenez celle qui est d'indices les plus petits dans  $v$  (par exemple `intersection("bbaacc", "aabb")` renvoie "bb").

## UpyLab 5.10

Écrivez une fonction `parallelogramme` qui reçoit quatre points du plan en paramètres et calcule le périmètre du parallélogramme correspondant. Les points  $(x_1, y_1)$ ,  $(x_2, x_2)$ ,  $(x_3, x_3)$  et  $(x_4, x_4)$  correspondent au coin supérieur gauche, au coin supérieur droit, au coin inférieur droit et au coin inférieur gauche. La fonction renverra le résultat du périmètre si les côtés sont bien parallèles et de même taille deux à deux (avec une tolérance `EPSILON = 1.0e-7`), et renverra `None` si ce n'est pas le cas.

## UpyLab 5.11

Ecrivez une fonction `prime_numbers` qui prend comme paramètre un nombre entier `nb` et qui renverra une liste contenant les `nb` premiers nombres premiers.

Nous vous demandons de penser aux différents cas pouvant intervenir dans l'exécution de la fonction. Si le paramètre est du mauvais type ou ne représente pas un entier positif, votre fonction doit renvoyer la valeur `None`

Pour rappel, un nombre premier  $b$  est un entier naturel qui admet que deux diviseurs distincts entiers et positifs :  $b$  et 1. En d'autres termes,  $b$  est premier si il n'existe pas d'entiers naturels  $c$  et  $d$  différents de  $b$  et de 1 tel que  $b = c \times d$ .

### Note :

Mathématiquement, on peut définir un nombre premier  $b$  comme étant un nombre tel que :

$$(b \geq 2) \wedge (\nexists a \cdot (1 < a < b) \wedge (b \bmod a = 0))$$

qui en français signifie :  $b$  est un nombre (entier) plus grand ou égal à 2 et il n'existe pas de nombre (entier)  $a$  strictement plus grand que 1 dont le reste de la division de  $b$  par  $a$  soit nul.

## UpyLab 5.12

[Trier](#) une collection d'objets est l'opération qui consiste à les ranger selon l'ordre donné. Un tel rangement revient à effectuer une permutation des éléments pour les ordonner.

Écrivez une fonction `my_insert` qui prend une liste d'entiers triée (par ordre UTF-8) `sorted_list` et un entier `n` en paramètres. La fonction devra renvoyer une liste correspondant à la liste reçue triée par ordre croissant, où `n` a été insérée tout en la maintenant triée. Vous pouvez supposer que la liste est bien formée de valeurs entières triées. Essayez de tester les autres cas d'erreur et dans ce cas de renvoyer `None`.

## UpyLab 5.13

Même exercice que le précédent, mais ici, si les paramètres ont le bon type, la fonction modifie la liste donnée en paramètre et ne renvoie rien (c'est-à-dire renvoie `None`), sinon une erreur se produit à l'exécution.

Pour tester les paramètres, vous pouvez utiliser l'instruction `assert` et les verbes `type(v)` ou `isinstance(v, typ)`.

### Notes:

- L'instruction `assert condition` laisse continuer l'exécution du code si la condition évaluée est vraie (`True`) et provoque une erreur (exception Python), si la condition est évaluée à Faux (`False`)
- `type(v)` donne le type de `v`
- `isinstance(v, typ)` teste si `v` est de type `typ`.



## UpyLab 5.14

Écrivez une fonction `my_count` qui prend une liste `lst` et un élément `e` en paramètres. La fonction doit renvoyer le nombre de fois que l'élément `e` apparaît dans la liste. 0 est retourné si `lst` n'a pas le bon type.

## UpyLab 5.15

Ecrivez une fonction `my_remove` qui prend une liste `lst` et un élément `e` en paramètre et qui effacera la première apparition de l'élément `e` dans la liste et ne renvoie rien (c'est-à-dire `None`). Si `lst` n'est pas une liste, `my_remove` ne fait rien.

## UpyLab 5.16

Nous pouvons définir la distance entre deux mots `mot_1` et `mot_2` de même longueur (même nombre de lettres) comme étant le nombre minimum de fois qu'il faut modifier une lettre du `mot_1` pour obtenir `mot_2`. Par exemple, le mot "lire" et le mot "bise" sont à une distance de 2 puisqu'il faut changer le 'l' et le 'r' du mot "lire" pour arriver à "bise".

- Nous vous demandons d'écrire une fonction `distance_mots(mot_1, mot_2)` qui retourne la distance entre deux mots. Nous supposons que `len(mot_1) == len(mot_2)` et que les mots n'ont pas d'accents.
- (Fait par Trinh Jacky 11/02/2018)

## UpyLab 5.17

Joao vient d'arriver dans notre pays depuis le Portugal. Il a encore du mal avec la langue française. Malgré ses efforts considérables, il fait une faute d'orthographe quasi à chaque mot. Son soucis est qu'il n'arrive pas toujours à écrire un mot correctement sans se tromper à une lettre près. Ainsi pour écrire "bonjour", il écrit "binjour". Pour remédier à ce problème, Joao utilise un correcteur orthographique. Malheureusement, Joao a un examen aujourd'hui et il a oublié son petit correcteur.

Afin de l'aider, nous vous demandons d'écrire une fonction `correcteur(mot, liste_mots)` où `mot` est le mot que Joao écrit et `liste_mots` est une liste qui contient les mots (ayant la bonne orthographe) que Joao est susceptible d'utiliser. Cette fonction doit retourner le mot dont l'orthographe a été corrigée. Pour cet exercice, vous devrez utiliser la fonction `distance_mots(mot_1, mot_2)` que vous avez précédemment codée et qui donne la distance entre deux mots de **même longueur**. N'oubliez pas de mettre aussi le code de la fonction `distance_mots(mot_1, mot_2)` dans votre solution

### Notes:

- le correcteur orthographique demandé est une version simple, c'est-à-dire que les mots en paramètre auront au maximum une seule lettre de différent par rapport à la bonne orthographe. Nous ne prenons pas en compte les mots avec accents, ni les mots composés de tiret, d'apostrophes, d'espace,... `liste_mots` ne contient pas de mots qui se ressemblent, c'est-à-dire que si Joao écrit le mot "liee", il se peut que cela représente le mot "lire" ou "lier". Afin d'éviter cette confusion, deux mots de même longueur de la liste sont au moins à une distance de 3.
- Vous pouvez supposer que Joao soit arrive à écrire des mots sans fautes, soit fait au plus 1 faute.

(Ecrit par Jacky Trinh le 11/02/2018)

## UpyLab 5.18

On se donne une liste qui encode une séquence  $t$ . Chaque élément de cette liste est un tuple de deux éléments: le nombre de répétitions successives de l'élément  $x$  dans  $t$ , et l'élément  $x$  lui-même. Les éléments successifs répétés forment la séquence  $t$

Ecrivez une fonction `decompresse`, qui reçoit une telle liste en paramètre et renvoie la séquence sous forme d'une nouvelle liste. Exemple:

```
>>> liste_comp = [(4, 1), (2, 2), (2, 'test'), (3, 3), (1, 'bonjour')] >>> decompresse (liste_comp)
[1, 1, 1, 1, 2, 2, 'test', 'test', 3, 3, 3, 'bonjour']
```

## UpyLab 5.19

Ecrivez une fonction `my_filter` qui prend une liste `lst` et une fonction booléenne `f` en paramètres.

Cette fonction renverra une nouvelle liste constituée des éléments de la liste `lst` pour lesquels la fonction `f` renvoie `True`.

### Note:

Pour les tests, UpyLaB utilise les fonctions Python `lambda`. Cela permet de définir au moment même une fonction anonyme avec un ou plusieurs paramètres

Ainsi :

```
lambda e: isinstance(e, int)
```

donne une fonction à un paramètre, qui teste si ce paramètre est de type entier.

## UpyLab 5.20

Écrivez une fonction `wc(nomFichier)` qui doit ouvrir le fichier en question et renvoyer un tuple avec trois valeurs :

- le nombre de caractères (y compris les caractères de retour à la ligne),
- le nombre de mots et
- le nombre des lignes du fichier.

Nous définissons ici un mot comme étant une chaîne de caractères (maximale, c'est-à-dire entourée d'espaces ou de séparateurs ou de caractères de fin de phrase) répondant `True` à la méthode `isalnum()`.

### Notes :

- Les fichiers utilisés pour tester le code sont accessibles en :

<https://upylab.ulb.ac.be/pub/data/wc1.txt>

<https://upylab.ulb.ac.be/pub/data/Zola.txt>

<https://upylab.ulb.ac.be/pub/data/le-petit-prince.txt>

- Pour vous aider : le premier fichier contient les caractères (sans les guillemets)  
`"a2x!&t5\n"` (avec `"\n"` qui désigne un seul caractère de fin de ligne),  
 assurez-vous bien que votre code vous donne bien 8 caractères, 2 mots et une ligne; les mots étant : `"a2x"` et `"t5"`
- Le fichier `le-petit-prince.txt` est libre de droit d'auteur sauf en France (voir [https://fr.wikipedia.org/wiki/Le\\_Petit\\_Prince](https://fr.wikipedia.org/wiki/Le_Petit_Prince)); si vous habitez en France, légalement vous ne pouvez donc pas le télécharger; ailleurs dans le monde pas de soucis !

## UpyLab 5.21

Écrivez une fonction `replace(in_path, out_path, pattern, subst)` qui doit ouvrir le fichier dont le chemin est `in_path` et remplacer dans ce dernier toutes les occurrences **sans chevauchement** du string `pattern` par `subst`. Le résultat de cette modification devra être écrite dans le fichier de chemin `out_path`.

- On peut supposer que `pattern` ne contient pas le caractère fin de ligne.
- Ainsi le remplacement sans chevauchement de `'aa'` par `'aaa'` dans le texte `'aaa'` donnera le texte `'aaaa'` (le string `'aa'` initialement en positions 0 et 1 dans le texte étant remplacé par `'aaa'`)

### Fichiers utilisés lors des tests:

- <http://upylab.ulb.ac.be/pub/data/le-petit-prince.txt>
- <http://upylab.ulb.ac.be/pub/data/Zola.txt>

## UpyLab 5.22

Écrivez une fonction `liste_des_mots`,

- qui reçoit en paramètre, le nom d'un fichier texte, que votre fonction doit ouvrir (vous pouvez-supposer que le fichier est présent et dans le bon format en encodage UTF-8),
- et qui renvoie la liste des mots non vides contenus dans le fichier. Les mots dans la liste seront mis en minuscules et triés par l'ordre donné par la codification UTF-8 (en clair simplement via la méthode `sort`), les accents n'étant pas gérés de façon spécifique. Un même mot ne peut pas être deux fois dans la liste. Les mots peuvent être séparés par des espaces habituels (caractère espace, tabulation, passage à la ligne), ou par n'importe quel parmi les caractères suivants:

`- ' " ? ! : ; . , * = ( ) 1 2 3 4 5 6 7 8 9 0`

## UpyLab 5.23

Écrivez une fonction `init_mat(m,n)` qui construit et renvoie une matrice d'entiers initialisée à la matrice nulle et de dimension  $m \times n$ .

## UpyLab 5.24

Écrivez une fonction `print_mat(m)` qui prend une matrice en paramètre et affiche son contenu.

Affichez les éléments de chaque ligne de la matrice séparés par un ou plusieurs espaces mais sans passer de ligne et passez une ligne entre chaque nouvelle ligne de la matrice (ici un ou plusieurs espaces sont aussi permis en fin de lignes)

La fonction ne renvoie pas de valeur particulière (renvoie `None`)

Votre code contiendra également, après le code de la définition de la fonction `print_mat(m)`, le code suivant qui initialisera le paramètre effectif `ma_matrice`:

```
ma_matrice = eval(input())  
print_mat(ma_matrice)
```

La matrice vide est la matrice `[[[]]]` (une liste avec une liste vide); dans ce cas le résultat est une ligne vide

## UpyLab 5.25

Écrivez une fonction `trace(M)` qui prend en paramètre une matrice  $M$  de taille  $n \times n$  contenant des valeurs numériques (int ou float) et qui renvoie sa trace de la manière suivante:

$$trace(A) = \sum_{i=1}^n A_{ii}$$

c'est-à-dire la somme des valeurs de la première diagonale de la matrice. Par exemple avec la matrice :

123

456

789

la valeur renvoyée est 15 (somme des valeurs 1 5 et 9 qui forment la première diagonale)

## UpyLab 5.26

Une matrice  $M = \{m_{ij}\}$  de taille  $n \times n$  est dite antisymétrique lorsque pour toute paire d'indices  $i, j$ , on a  $m_{ij} = -m_{ji}$ . Écrire une fonction booléenne `antisymetrique` qui teste si une matrice est antisymétrique

## UpyLab 5.27

Écrivez une fonction `symetrie_horizontale(A)` qui prend en paramètre une matrice  $A$  et qui renvoie l'image de  $A$  par symétrie horizontale par rapport à la ligne du milieu : la dernière ligne devenant la première, la seconde, l'avant dernière, etc.

## UpyLab 5.28

Écrivez une fonction `symetrie_diagonale_2(A)` qui prend en paramètre une matrice  $A$  et qui renvoie l'image de  $A$  par symétrie par rapport à la seconde diagonale.

## UpyLab 5.29

Puissance 4 est un jeu de stratégie combinatoire abstrait dont le but est d'aligner une suite de 4 pions de même couleur sur une grille comptant 6 rangées et 7 colonnes. Chaque joueur possède 21 pions d'une couleur (jaune ou rouge). À chaque tour, le joueur suivant doit placer un pion dans la colonne de son choix. Le pion tombe dans la position la plus basse possible. Le vainqueur est le premier qui a réussi à aligner horizontalement, verticalement ou diagonalement de manière consécutive 4 de ses pions.

Nous vous demandons d'écrire une fonction `placer_pion(couleur, colonne, grille)` où

- `couleur` est la couleur du pion qui peut être soit "R" (rouge) soit "J" (jaune),
- `colonne` est la colonne où on aimerait placer le pion (allant de 0 jusqu'à 6 inclus) et
- `grille` est la grille du jeu sous forme de matrice. Cette matrice contient donc 6 listes de lignes contenant chacune 7 éléments. La ligne à l'indice 0 représente le bas du jeu. Cette grille contiendra soit "R", soit "J" ou soit "V" pour vide. Cette fonction placera un pion dans la grille du jeu et renverra un couple de valeurs :
- la valeur booléenne `True` si le placement est possible et `False` sinon ;
- la grille modifiée si le placement a été effectuée, non modifiée sinon. (Fait par Jacky Trinh le 15/02/2018)

## UpyLab 5.30

Puissance 4 est un jeu de stratégie combinatoire abstrait dont le but est d'aligner une suite de 4 pions de même couleur sur une grille comptant 6 rangées et 7 colonnes. Chaque joueur possède 21 pions d'une couleur (jaune ou rouge). À chaque tour, le joueur suivant doit placer un pion dans la colonne de son choix. Le pion tombe dans la position la plus basse possible. Le vainqueur est le premier qui a réussi à aligner horizontalement, verticalement ou diagonalement de manière consécutive 4 de ses pions.

Nous vous demandons d'écrire une fonction `gagnant(grille)` où `grille` est la grille du jeu sous forme de matrice. Cette matrice contient donc 6 listes de lignes contenant chacune 7 éléments. La ligne à l'indice 0 représente le bas du jeu. Cette grille contiendra soit "R", soit "J" ou soit "V" pour vide. Cette fonction renverra soit "R" si c'est le rouge qui a gagné, soit "J" si c'est le jaune ou bien `None` si personne a gagné.

Note: On peut supposer que dans une grille donnée à la fonction, il n'y a pas à la fois une configuration jaune et une configuration rouge gagnante. (Fait par Jacky Trinh le 15/02/2018)

## UpyLab 6.1

Monsieur Germain est une personne très âgée et aimerait préparer une liste de courses à faire à l'avance. Ayant un budget assez serré, il voudrait que sa liste de courses soit dans ses capacités. Son seul petit soucis est qu'il a une très mauvaise vue et n'arrive donc pas à voir le prix associé à chaque produit contenu dans le catalogue de courses.

Nous vous demandons d'écrire une fonction `calcul_prix(produits,catalogue)` où

- `produits` est un dictionnaire contenant, en clé, les produits souhaités par Monsieur Germain et, en valeur, leur quantité et
- `catalogue` est un dictionnaire contenant tous les produits du magasin ainsi que leur prix associé. Cette fonction retournera le total que Monsieur Germain devra payer à la caisse pour qu'il achète toutes les courses prévues.

Exemple de `produits` = {"brocoli":2, "mouchoirs":5, "bouteilles d'eau":6} Exemple de `catalogue` = {"brocoli":1.50, "mouchoirs":0.80, "bouteilles d'eau":1, "bière":2, "savon":2.50}

(Fait par Jacky Trinh le 19/02/18)

## UpyLab 6.2

Un jury doit attribuer le prix du "Codeur de l'année".

Afin de récompenser les trois candidats ayant la meilleure moyenne, nous vous demandons d'écrire une fonction `top_3_candidats(moyennes)` qui reçoit un dictionnaire contenant en clé des noms de candidats et en valeur la moyenne qui lui est attribuée.

Cette fonction doit retourner un tuple des trois meilleurs candidats. Le premier meilleur candidat doit se trouver à l'indice 0 du tuple (le troisième meilleur candidat se trouvera donc à l'indice 2 du tuple).

### Notes:

- Les candidats ont des moyennes différentes. Il y aura toujours plus de 3 candidats dans le dictionnaire.
- Comme l'ordre des entrées d'un dictionnaire n'est pas déterminé, le jeu de test d'UpyLaB utilise 5 fois le même dictionnaire, mais avec les éléments créés dans un ordre différent.

(Fait par Trinh Jacky le 24/02/18)

## UpyLab 6.3

Dans un texte, il nous arrive souvent de remplacer des mots par des abréviations (exemple : bonjour par bjr ). Nous allons utiliser un dictionnaire pour contenir les abréviations et leur signification. Nous vous demandons d'écrire une fonction `substitue(message,abreviation)` qui va renvoyer une copie de la chaîne de caractères `message` dans laquelle les mots qui figurent dans le dictionnaire `abreviation` (comme clé) sont remplacés par leur signification (valeur). Exemple d'utilisation :

```
abrev = {'C.' : 'Chuck',
        'N.' : 'Norris',
        'cpt' : 'counted',
        '2' : '2 times',
        'inf' : 'infinity'}
print(substitue('C. N. cpt 2 to inf', abrev))
```

Pour simplifier, on suppose que chaque mot est séparé par un espace (' ').

## UpyLab 6.4

Nous pouvons construire un dictionnaire dont les clés sont les prénoms des personnes nommées (pour simplifier on suppose que deux personnes n'ont pas le même prénom) et la valeur de chaque entrée est l'ensemble des amis de la personne. Ecrivez une fonction `construction_dict_amis` qui reçoit une liste de couples (`prénom1`, `prénom2`) voulant dire que `prénom1` déclare `prénom2` comme étant son ami et qui construit et renvoie le dictionnaire tel que décrit plus haut.

**Si dans la liste de couples reçue nous avons (`prénom1`, `prénom2`), cela n'induit pas que `prénom2` soit également ami de `prénom1`. Si le couple (`prénom2`, `prénom1`) n'est pas dans cette liste, nous aurons donc une amitié à sens unique !**

## UpyLab 6.5

Ecrivez une fonction `symetrise_amis` qui reçoit un dictionnaire `d` d'amis où les clés sont des prénoms et les valeurs des ensembles de prénoms parmi les clés qui représentent les ensembles des amis des personnes. Cette fonction modifie le dictionnaire `d` pour que si une entrée de clé `a` contient `b` dans l'ensemble de ses amis, l'inverse soit vrai aussi.

Votre fonction aura un second paramètre: `englobe`. Si `englobe` est vrai, votre fonction ajoutera les éléments nécessaires aux ensembles pour symétriser le dictionnaire `d`. Sinon, votre fonction enlèvera les éléments nécessaires pour symétriser `d`.

Par exemple avec `d = {'Thierry': {'Michelle', 'Bernadette'}, 'Michelle': {'Thierry'}, 'Bernadette': set()}`

- Après l'appel à `symetrise_amis(d, True)`, `d` vaudra `{'Thierry': {'Michelle', 'Bernadette'}, 'Michelle': {'Thierry'}, 'Bernadette': {'Thierry'}}`
- Par contre après l'appel à `symetrise_amis(d, False)`, `d` vaudra `{'Thierry': {'Michelle'}, 'Michelle': {'Thierry'}, 'Bernadette': set()}`

**Note :** la fonction `symetrise_amis` ne renverra rien (return None)

## UpyLab 6.6

Ecrivez une fonction `valeurs(dico)` qui doit fournir, à partir du dictionnaire donné en paramètre, une liste des valeurs du dictionnaire telles qu'elles soient triées (par ordre UTF-8) sur base de la clef du dictionnaire. Vous pouvez supposer que les clefs du dictionnaire sont des strings et vous pouvez utiliser la méthode `sort()` sur la liste des clés pour vous aider.

**Par exemple :**

```
valeurs({'three': 'trois', 'two': 'deux', 'one': 'un'})
```

renvoie la liste :

```
['un', 'trois', 'deux']
```

étant donné que :

pour les clés correspondantes sont dans cet ordre :

```
'one' < 'three' < 'two'
```

## UpyLab 6.7

Ecrivez une fonction `primes_odds_numbers(numbers)` qui reçoit une liste de nombres et qui renvoie un couple d'ensembles contenant respectivement les nombres premiers de cette liste ainsi qu'ensuite les nombres impairs. Votre fonction doit faire appel à deux autres fonctions

- `even(max)` qui renvoie l'ensemble (set) des entiers pairs inférieurs à `max`
- `prime_numbers(max)` qui renvoie l'ensemble (set) des nombres premiers inférieurs à `max`

que vous devez également coder.

## UpyLab 6.8

Ecrivez une fonction `store_email(liste_mails)` qui reçoit en paramètre une liste d'adresses e-mail et qui renvoie un dictionnaire avec comme clefs le domaine des adresses e-mail et comme valeurs les listes correspondantes **triées par ordre (UTF-8) croissant** des utilisateurs. Par exemple, la liste d'adresse suivante:

```
[ "ludo@prof.ur", "andre.colon@stud.ulb", "thierry@profs.ulb",  
  "sébastien@prof.ur", "eric.ramzi@stud.ur", "bernard@profs.ulb",  
  "jean@profs.ulb" ]
```

donnerait le dictionnaire suivant :

```
{ 'prof.ur' : ['ludo', 'sébastien']  
  'stud.ulb' : ['andre.colon']  
  'profs.ulb' : ['bernard', 'jean', 'thierry']  
  'stud.ur' : ['eric.ramzi'] }
```

## UpyLab 6.9

Ecrivez une fonction `belongs_to_file` prenant en paramètre une chaîne de caractères, qui vérifie si elle représente un mot qui figure dans la liste des mots contenus dans le fichier `/pub/data/words.txt`, auquel cas elle renverra `True`. Sinon, la fonction renvoie `False`. Attention : les mots du fichier `words.txt` seront stockés ligne par ligne à raison de un mot par ligne dans le fichier.



## UpyLab 6.10

Nous vous demandons d'écrire une fonction `compteur_lettres(texte)` qui renverra un dictionnaire contenant **toutes** les lettres de l'alphabet en tant que clés ainsi que leur nombre d'occurrence dans texte.

Note: Les clés du dictionnaire devront être en minuscule. Aucun accent ou cédille, ... ne sera présent dans texte.

Par contre :

- il peut y avoir des majuscules; dans ce cas, la lettre doit être comptabilisée comme étant la lettre minuscule correspondante ;
- les espaces et autres caractères de ponctuation (', ", -, ...) doivent être ignorés.

(Fait par Trinh Jacky le 25/02/18)

## UpyLab 6.11

À l'aide d'un dictionnaire, écrivez les fonctions suivantes :

- `file_histogram(fileName)` qui prend en paramètre le nom, sous forme d'une chaîne de caractères, d'un fichier texte et qui renvoie un dictionnaire contenant comme clé le caractère et comme valeur la fréquence absolue (c'est-à-dire le nombre d'occurrences) de ce caractère dans le texte contenu dans le fichier.
- `vowels_histogram(fileName)` qui prend en paramètre le nom, sous forme d'une chaîne de caractères, d'un fichier texte et qui renvoie un dictionnaire contenant la fréquence absolue de chaque suite de voyelles (par exemple : "i", "e", "oe", "oui", "eui", "you", "ee", etc.). Notez que si on trouve "oe" par exemple, il ne faut pas compter une occurrence de "o" ni de "e".  
**Vous ne comptabiliserez pas les voyelles avec accent, ... (uniquement les suites de a, e, i, o, u, y en minuscules ou majuscules).**
- `words_by_length(fileName)` qui prend en paramètre le nom, sous forme d'une chaîne de caractères, d'un fichier texte et qui renvoie un dictionnaire associant à une valeur entière ℓ

, la liste **triée par ordre (UTF-8) croissant** des mots de longueur ℓ

- dans le fichier. **Ici vous supposerez qu'un mot est une séquence de caractères alphabétiques (utilisez la méthode `isalpha()`). De plus mettez vos résultats en minuscules grâce à la méthode `lower`)**
  - Notez que l'on considère 'cat4dog' comme formé de 3 parties dont 2 sont des mots alphabétiques
  - Veillez à ne pas avoir plusieurs fois les mêmes mots dans chacune des listes et de construire les listes en respectant l'ordre d'apparition des mots
  - Pour le tri des listes, utilisez simplement la méthode `sort()` sans paramètres.

**Dans vos fonctions, prenez en compte tous les caractères lus: n'utilisez par exemple pas la méthode `strip()` pour enlever certains caractères en début et fin de ligne.**

**Fichiers utilisés lors des tests:**

- <http://upylab.ulb.ac.be/pub/data/words.txt>
- <http://upylab.ulb.ac.be/pub/data/le-petit-prince.txt>

- <http://upylab.ulb.ac.be/pub/data/Zola.txt>

## UpyLaB 6.12

Le sudoku est un jeu de logique dans lequel le joueur reçoit une grille de 9 X 9 cases, chacune pouvant contenir un chiffre de 1 à 9 ou bien être vide. La grille est divisée en 9 lignes, 9 colonnes ainsi qu'en 9 « sous-grilles » appelées *régions*, formées de 3 X 3 boîtes contigües. Le but du jeu est de remplir les cases vides avec des chiffres de 1 à 9, de telle sorte que, dans chaque ligne, colonne et région soient présents tous les chiffres de 1 à 9, sans doublons.

Nous vous demandons d'écrire une fonction `check_sudoku` capable de vérifier si une affectation possible des toutes les cases est valide, c'est-à-dire est une solution au problème du sudoku. La fonction recevra une affectation via une liste de string `grille`. Cette liste contient une description de la solution sous forme d'une matrice 9x9. Elle donnera la réponse (vrai ou faux) sur l'écran.

Exemple d'input :

```
grille = ["9 6 3 1 7 4 2 5 8\n",
          "1 7 8 3 2 5 6 4 9\n",
          "2 5 4 6 8 9 7 3 1\n",
          "8 2 1 4 3 7 5 9 6\n",
          "4 9 6 8 5 2 3 1 7\n",
          "7 3 5 9 6 1 8 2 4\n",
          "5 8 9 7 1 3 4 6 2\n",
          "3 1 7 2 4 6 9 8 5\n",
          "6 4 2 5 9 8 1 7 3\n"]
```

Cette fonction devra utiliser et implanter les cinq fonctions suivantes :

- `check_cols(grid)` qui prend en paramètre la grille sous forme de matrice à deux dimensions et vérifie si toutes les colonnes sont valides (c'est-à-dire que sur chaque colonne, chaque chiffre apparaît une et une seule fois).
- `check_rows(grid)` qui prend en paramètre la grille sous forme de matrice à deux dimensions et vérifie si toutes les lignes sont valides (c'est-à-dire que sur chaque ligne, chaque chiffre apparaît une et une seule fois).
- `check_regions(grid)` qui prend en paramètre la grille sous forme de matrice à deux dimensions et vérifie si toutes les régions sont valides (c'est-à-dire que dans chaque région, chaque chiffre apparaît une et une seule fois).
- `parse_solution(grille)` qui prend en paramètre une liste de String et renvoie la grille du sudoku correspondante sous forme de matrice à deux dimensions.
- `check_sudoku(grille)`, une fonction "principale" qui prend en paramètre `grille` une liste de String, à l'aide d'appels aux quatre fonctions précédentes vérifie si la solution donnée est une solution valide selon les règles du sudoku.

## UpyLaB 6.13

Le sudoku est un jeu de logique composé d'une grille de 9 lignes et 9 colonnes. La grille principale est subdivisée en 9 sous-grilles contigües de 3\*3 appelées régions. Au début du jeu, certaines cases sont dévoilées et comportent un nombre entre 1 et 9. Le but du jeu est de placer un chiffre dans chaque case vide avec la contrainte que chaque ligne, chaque colonne et chaque région comporte une et une seule fois chaque nombre de 1 à 9.

Une méthode de résolution passe par l'analyse des candidats uniques. Un candidat est un nombre permis pour une case car on ne le retrouve pas ailleurs dans la ligne, la colonne et la région de cette case.

Pour déterminer pour une configuration donnée s'il existe un candidat unique, la technique du *naked single* peut être utilisée. Le *naked single* est une case pour laquelle il n'y a qu'un seul candidat étant donné la ligne colonne et région où cette case se trouve.

Nous vous demandons d'écrire une fonction *naked\_single(grille)* qui, recevant en paramètre une matrice d'entiers de 9\*9 représentant une grille, renvoie un couple de valeurs :

- un booléen ok qui détermine si le problème peut être résolu en utilisant exclusivement la technique du *naked single*.
- si ok est vrai, la grille complétée, sinon None

L'encodage de la matrice (liste de listes d'entiers), prend la valeur 0 pour les cases vides.

Des deux grilles suivantes, seule la première peut être résolue de cette manière: par exemple, en indiquant le tableau à partir de 0 (0..8×0..8

), 0,0) étant en haut à gauche, à la case d'indice (4,7)

, seule la valeur 3 est possible. On peut ensuite faire de même jusqu'à la résolution complète du sudoku.

4		3		9	6		1	
		2	8		1			3
	1							7
	4		7				2	6
5		7		1		4		9
1	2				3		8	
2							7	
7			2		9	8		
	6		1	5		3		2

		6		4		1		
	5			9			6	
8								5
			3		4			
3	1						4	8
			8		7			
6								9
	2			3			5	
		1		5		7		

### Attention à l'encodage du jeu :

Notez que contrairement à l'exercice précédent, la grille du sudoku est ici encodée sous forme d'une matrice 9 x 9 d'entiers valant entre 0 et 9 (un zéro désignant une case non dévoilée).

## UpyLaB 6.14

Un dictionnaire peut nous permettre de stocker un tableau partiel, c'est-à-dire uniquement les cases remplies du tableau avec comme valeur le contenu de la case en question. Voici un exemple :

	X		O		
			O		
		O			
					O

X : trésor  
O : piège

Figure 1: Exemple de tableau. Le tableau ci-dessus peut être implémenté de la manière suivante :

```
MY_PRECIOUS = 1
TRAP = -1
map = {}
map[(1,1)] = MY_PRECIOUS
map[(2,3)] = TRAP
map[(4,5)] = TRAP
map[(1,3)] = TRAP
map[(3,2)] = TRAP
```

On vous demande d'implémenter les fonctions suivantes :

- `create_map(size, trapsNbr)` qui reçoit en paramètres deux nombres naturels `size` (de valeur entre deux et 100) et `trapsNbr` (de valeur strictement inférieure à `size2`) et qui renvoie un dictionnaire représentant une carte de taille `size×size`
  - dans laquelle on place `trapsNbr` pièges (valeur : -1) et un trésor (valeur: 1) de manière aléatoire (utilisez le module `random`).
  - `play_game(mapSize, map)` qui reçoit une carte de taille `mapSize×mapSize`
    - sous la forme d'un dictionnaire tel que décrit précédemment dans laquelle un certain nombre de pièges et un trésor ont été placés de manière aléatoire. La fonction demande à l'utilisateur d'entrer les coordonnées de cases (chaque case via deux appels à `input()` successifs). Les inputs continuent jusqu'au premier qui correspond à une case occupée. La fonction renvoie `True` si l'utilisateur a trouvé le trésor (et donc `False` si la case trouvée est un piège).
- En pratique, pour tester en particulier la fonction `play_game`, celle-ci ne fera aucun print et n'écrira rien lors des input (utilisez uniquement `input()` sans paramètre)

## UpyLaB 6.15

Arthur nous a envoyé des statistiques produites par UpyLaB contenant des informations sur des exercices et sur des apprenants. Nous avons décidé de ne retenir que les "meilleurs" exercices en les classant (triant dans le jargon informatique).

Chaque jeux de données est formé de deux fichiers de type csv (produit par un tableur du type excel ou LibreOffice).

Un fichier 'result-pass-fail.csv'

Un fichier 'result-count.csv'

### La structure d'un fichier 'result-pass-fail-i.csv'

est la suivante :

- en première ligne les intitulés des exercices chacun séparé de suivant par un caractère point-virgule ';' ;
- chacune des lignes suivantes correspondent aux résultats d'un apprenant : chaque ligne a le même nombre d'éléments que le nombre d'intitulés, et chaque élément vaut soit le texte 'VRAI', soit le texte 'FAUX', soit est vide (respectivement suivant que l'apprenant a validé tous les tests UpyLaB pour l'exercice correspondant, a demandé la validation par UpyLaB, mais malheureusement ce dernier a répondu par la négative, ou n'a pas encore testé l'exercice dans UpyLaB.

A nouveau sur une même ligne, chaque valeur est séparée de la suivante par un caractère point-virgule ';'.

### La structure du fichier 'result-count-i.csv'

est similaire à celle de l'autre fichier, à la différence près que ici les textes 'VRAI' ou 'FAUX' sont remplacés par des (représentations de) nombres naturels strictement positifs donnant le nombre de fois que l'apprenant a testé son code. A nouveau l'entrée est vide si aucune validation n'a été demandée par l'apprenant.

De façon précise la structure d'un fichier 'result-count-i.csv' est la suivante :

- en première ligne les intitulés des exercices chacun séparé de suivant par un caractère point-virgule ';' ;
- chacune des lignes suivantes correspondent aux résultats d'un apprenant : chaque ligne le même nombre d'éléments que le nombre d'intitulés, et chaque élément représente soit un nombre entier strictement positif, soit est vide (respectivement suivant que l'apprenant a (essayé de) valider les tests UpyLaB pour l'exercice correspondant ou n'a pas encore testé l'exercice dans UpyLaB.

A nouveau sur une même ligne, chaque valeur est séparée de la suivante par un caractère point-virgule ';'.

## Nous vous demandons

de réaliser un programme qui lit deux chaînes de caractères, qui représentent les noms des deux fichiers décrits ci-dessus (dans l'ordre le fichier de type 'result-pass-fail.csv' suivit du fichier du type

'result-count.csv'), et qui imprime la liste des intitulés, un par ligne, dans l'ordre (UTF-8) décroissant des "valeurs" calculées (le plus grand devant), comme suit.

La "valeur" d'un intitulé est le nombre des "apprenants fiables" ayant réussi l'exercice correspondant.

Un "apprenant fiable" est un apprenant qui n'a jamais testé plus de 10 fois chacun des codes qu'il a essayé de valider (10 peut être vue comme une constante globale de votre programme : MAX = 10).

Par exemple

si un apprenant a testé les trois premiers exercices avec 1 2 et 10 tests, il est réputé "apprenant fiable" ;

si un apprenant a testé tous les exercices mais que malheureusement il a testé 11 fois un exercice, il n'est **plus** réputé "apprenant fiable".

Si 2 intitulés d'exercices ont la même valeur, l'affichage les imprime dans l'ordre décroissant habituel pour Python : par exemple, "1.1. Premier exercice" est inférieur à "1.5 Cinquième exercice" et donc "1.5 Cinquième exercice" sera écrit avant "1.1. Premier exercice" sur output

## Exemple

### Fichier result-pass-fail-0.csv

```
ex1;ex2;ex3
VRAI;VRAI;VRAI
VRAI;VRAI;VRAI
VRAI;FAUX;FAUX
VRAI;VRAI;VRAI
VRAI;VRAI;
FAUX;VRAI;VRAI
```

### Fichier result-count-0.csv

```
ex1;ex2;ex3
2;3;5
1;2;4
4;2;666
11;6;3
1;1;
7;7;7
```

## Solution

```
ex2
ex3
ex1
```

En effet, ex2 vaut 4 (car les 3ème et 4ème apprenants ne sont pas des "apprenants fiables"; les autres valent 3 mais 'ex3' > 'ex1')

## **Fichiers utilisés pour les validations**

<http://upylab.ulb.ac.be/pub/data/result-pass-fail-0.csv>

<http://upylab.ulb.ac.be/pub/data/result-count-0.csv>

<http://upylab.ulb.ac.be/pub/data/result-pass-fail-1.csv>

<http://upylab.ulb.ac.be/pub/data/result-count-1.csv>