

Exercices d'informatique pour la 2^e B

STRUCTURE ALTERNATIVE

Exercice 1 : année bissextile

Ecrire un programme qui détermine si une année a introduite par l'utilisateur est bissextile ou non. Depuis l'instauration du calendrier, sont bissextiles les années soit divisibles par 4 mais non divisibles par 100, soit divisibles par 400. Par exemple : 2012, 2016, et 2000 sont bissextiles mais 2001, 2002 et 2100 ne le sont pas.

Exercice 2 : spectre de la lumière

La longueur d'onde de la lumière visible varie de 380 à 750 nanomètres (nm). Le spectre étant continu, il est souvent divisé en 6 couleurs comme indiqué ci-dessous:

Couleur	Longueurs d'onde (nm)
Violet	$\in [380, 450[$
Bleu	$\in [450, 495[$
Vert	$\in [495, 570[$
Jaune	$\in [570, 590[$
Orange	$\in [590, 620[$
Rouge	$\in [620, 750]$

Ecrire un script qui lit la longueur d'onde de l'utilisateur et affiche sa couleur. Afficher un message d'erreur approprié si la longueur d'onde entrée par l'utilisateur est en dehors du spectre visible.

Exercice 3 : nombre et signe des racines

Ecrire un programme qui détermine **le nombre** et **le signe** des racines d'une équation du 2^e degré $ax^2 + bx + c = 0$, **sans calculer les racines**. On pourra supposer que $a \neq 0$ et on discutera en fonction du discriminant Δ , du produit P et de la somme S des racines.

Exemple d'exécution :

```
Equation : a * x ** 2 + b * x + c == 0
=====
Enter the coefficients a, b and c :
a != 0 : 5
b : 6
c : 1
Your equation
5.0 * x ** 2 + 6.0 * x + 1.0 == 0 has
2 solutions x1 and x2 :
x1 < 0 and x2 < 0
```

Exercice 4 : équation du 1^{er} degré

Ecrire un programme qui permet de résoudre une équation du premier degré du type : $a \cdot x + b = c \cdot x + d$, d'inconnue x . Les coefficients réels **a**, **b**, **c** et **d** seront entrés au clavier par l'utilisateur et stockés dans des variables de même nom. Ensuite le programme devra analyser si l'équation a *une solution unique* et afficher cette solution le cas échéant. Dans le cas où l'équation n'a *pas de solution* ou une *infinité* de solutions, l'utilisateur en sera également informé.

Exercice 5 : triangles

Ecrire un programme **triangles** qui demande à l'utilisateur d'entrer les longueurs (réels positifs) des trois côtés d'un triangle.

- Si la longueur d'un côté est \geq la somme des longueurs deux autres côtés alors le triangle n'existe pas et le programme doit en informer l'utilisateur.
- Sinon le triangle existe le programme doit déterminer de quel type de triangle il s'agit : *équilatéral*, *rectangle*, *isocèle*, *rectangle isocèle* ou *scalène* (c.-à-d. dont les trois côtés ont des longueurs distinctes).

Exercice 6 : intersection de droites

Ecrire un programme qui demande à l'utilisateur d'étudier *la position relative de deux droites d1 et d2 du plan* rapporté à un repère cartésien. On suppose que les équations cartésiennes des droites sont de la forme $a1 \cdot x + b1 \cdot y = c1$ et $a2 \cdot x + b2 \cdot y = c2$. Il suffit donc de demander à l'utilisateur de préciser les coefficients (réels) $a1$, $b1$, $c1$, $a2$, $b2$ et $c2$ des deux équations.

Si $a1 = b1 = 0$ ou $a2 = b2 = 0$, le programme affichera le message : « Equation non valid » et se terminera.

Sinon le programme devra analyser si les deux droites $d1$ et $d2$ sont *sécantes* (et il affichera alors les coordonnées de leur point d'intersection) ou si les deux droites sont *strictement parallèles* ou alors si elles sont *confondues*. Utiliser la méthode des déterminants de Cramer, vue en 3^e.

BOUCLES FOR - WHILE

Exercice 7 : boucles et fonction range

En utilisant la fonction **range**, écrire les instructions qui affichent (*dans une ligne* avec comme séparateur un espace) :

- (1) les entiers pairs dans $[0, 60]$, a) par ordre croissant et b) par ordre décroissant.
- (2) les entiers impairs dans $[-50, 80]$, a) par ordre croissant et b) par ordre décroissant.
- (3) les multiples de 3 dans $[-40, 40]$, par ordre croissant
- (4) les entiers de la forme $9 \cdot n + 4$ dans $[0, 100]$, par ordre croissant ;
- (5) les puissances de 2 dans $[0, 1000]$, par ordre croissant ;
- (6) les carrés parfaits dans $[0, 500]$, par ordre croissant ;
- (7) les entiers divisibles par 11 ou par 19 dans $[0, 200]$, par ordre croissant ;
- (8) les noms de la liste **names** = ['Pol', 'Pit', 'Pir', 'Nick', 'Mick'] ;

Rappel : l'élément d'indice **i** de la liste est **names[i]**, par exemple :

names[0]= 'Pol', **names[1]**= 'Pit', ..., **names[4]**= 'Mick'

- (9) la somme des entiers naturels compris entre 1 et 100 ;
- (10) la somme des carrés des entiers naturels compris entre 1 et 100 ;
- (11) $20!$ (c.-à-d. la factorielle de 20 : $20! = 20 \cdot 19 \cdot 18 \cdot \dots \cdot 3 \cdot 2 \cdot 1$) ;
- (12) les factorielles des entiers de 1 à 15 (une factorielle par ligne) ;

Exercice 8 : nombre à composer

Ecrire un programme qui demande à l'utilisateur d'entrer l'un après l'autre 6 chiffres. Le programme devra ensuite afficher sur l'écran le nombre formé par ces chiffres. Par exemple, si l'utilisateur entre 3, 5, 0, 7, 4, 5 dans cet ordre, alors le nombre affiché sera 350745. On écrira deux versions, l'une utilisant les strings, l'autre utilisant uniquement les nombres entiers. Voici l'écran de sortie :

```
Enter the first digit : 3
Enter the next digit : 5
Enter the next digit : 0
Enter the next digit : 7
Enter the next digit : 4
Enter the next digit : 5
350745
```

Exercice 9 : nombre à décomposer

Ecrire un programme qui fait l'inverse du précédent : il demande à l'utilisateur d'entrer un nombre entier de 6 chiffres. Le programme doit ensuite afficher les différents chiffres composant cet entier, de la gauche vers la droite. On demande d'écrire deux versions : dans la première le nombre sera stocké sous forme de string, dans la deuxième version il est interdit d'utiliser les strings. Voici l'écran de sortie :

```
Enter a 6-digit number : 350745
3
5
0
7
4
5
```

Exercice 10 : voyelles à compter

Ecrire un programme qui compte le nombre de voyelles (i, u, e, o, a, y) dans une chaîne de caractères formée uniquement de lettres minuscules (et non accentuées).

Exercice 11 : couleurs

On définit la liste suivante avec des couleurs :

```
colors = ['red', 'green', 'blue', 'yellow']
```

Ecrire une instruction (ou un bloc d'instructions) qui :

- (1) affiche toutes les couleurs de la liste dans l'ordre, sans passer à la ligne.
- (2) affiche 1 fois la 1^{re} couleur, 2 fois la 2^e couleur, 3 fois la 3^e couleur etc. jusqu' à la fin de la liste. Le passage à la ligne ne doit être fait que si on affiche la couleur suivante.
- (3) demande à l'utilisateur d'entrer le nombre de fois que chaque couleur doit être affichée, à stocker dans une liste **frequencies**. Ensuite chaque couleur de **colors** est affichée autant de fois que spécifié dans la liste **frequencies**. Remarque : pour ajouter un élément à une liste, on utilise la méthode **append**. Documentez-vous sur internet !

Ecran de sortie :

```
Example 1
red green blue yellow
```

```
Example 2
red
green green
blue blue blue
yellow yellow yellow yellow
```

```

Example 3
How often would you like to print red : 5
How often would you like to print green : 3
How often would you like to print blue : 4
How often would you like to print yellow : 1
red red red red red
green green green
blue blue blue blue
yellow

```

Exercice 12 : triangles de chiffres

A l'aide de boucles **for**, écrire un programme qui affiche les lignes suivantes :

a)	b)	c)
1	1	999999999
22	12	88888888
333	123	7777777
4444	1234	666666
55555	12345	55555
666666	123456	4444
7777777	1234567	333
88888888	12345678	22
999999999	123456789	1

Exercice 13 : suites

A l'aide de boucles **for**, écrire un programme qui produit l'écran-sortie ci-dessous.

```

      1      2      3      4      5      6      7      8      9     10
      5      7      9     11     13     15     17     19     21     23
     59     53     47     41     35     29     23     17     11      5
*****
* * * * *
* * * * *
*****
:-)  :-)  :-)  :-)  :-)  (-:  (-:  (-:  (-:  (-:
  1    2    4    8   16   32   64  128  256  512
1024 -512  256 -128   64  -32   16   -8    4   -2
  1    1    2    3    5    8   13   21   34   55

```

Remarques :

- Pour représenter un entier **n** de sorte qu'il occupe 5 colonnes sur l'écran (à partir de la droite) on peut utiliser le f-string : **f'{n:5}'**.
- La suite de la dernière ligne est appelée suite de **Fibonacci** : chaque terme est la somme des deux termes qui le précèdent, les deux premiers termes de la suite sont initialisés à 1.

Exercice 14 : motifs étoilés

Ecrire des scripts permettant de représenter respectivement chacun des motifs étoilés ci-dessous.

a) ***Rectangle :***

```
Enter the number of rows : 5
Enter the number of columns : 10
*****
*****
*****
*****
*****
```

b) ***Bordure :***

```
Enter the number of rows : 5
Enter the number of columns : 10
*****
*           *
*           *
*           *
*****
```

c) ***Escalier descendant :***

```
Enter the number of rows : 10
*
**
***
****
*****
*****
*****
*****
*****
*****
```

d) ***Escalier ascendant :***

```
Enter the number of rows : 10
      *
     **
    ***
   ****
  *****
 *****
*****
*****
*****
*****
```

e) **Escalier double :**

Enter the number of rows : 8

```

      *
     ***
    *****
   *********
  ***********
 *****
*****
*****
*****
*****
*****

```

f) **Croix :**

Enter the number of rows : 10

```

*           *
*         *
*       *
*     *
*   *
* *
* *
*   *
*     *
*       *
*         *
*           *

```

g) **Damier 1 :**

Enter the number of rows : 5

Enter the number of columns : 10

```

* * * * *
* * * * *
* * * * *
* * * * *
* * * * *

```

h) **Damier 2 :**

Enter the number of squares in a row : 3

Enter the dimension of a square : 5

```

* * * * * * * * * * * * * * * *
*           *           *           *
*           *           *           *
*           *           *           *
*           *           *           *
* * * * * * * * * * * * * * * *
*           *           *           *
*           *           *           *
*           *           *           *
*           *           *           *
* * * * * * * * * * * * * * * *
*           *           *           *
*           *           *           *
*           *           *           *
*           *           *           *
* * * * * * * * * * * * * * * *

```

N.B. Pour que les cases aient un aspect plus « carré », on a affiché ici des suites de `* ` (avec un espace) au lieu de `*`.

i) ***Damier 3 :***

Enter the number of squares in a row : 4

Enter the dimension of a square : 5

```

* * * * * * * * * * * * * * * * * *
*          * 0 0 0 0 *          * 0 0 0 0 *
*          * 0 0 0 0 *          * 0 0 0 0 *
*          * 0 0 0 0 *          * 0 0 0 0 *
*          * 0 0 0 0 *          * 0 0 0 0 *
* * * * * * * * * * * * * * * * * *
* 0 0 0 0 *          * 0 0 0 0 *          *
* 0 0 0 0 *          * 0 0 0 0 *          *
* 0 0 0 0 *          * 0 0 0 0 *          *
* 0 0 0 0 *          * 0 0 0 0 *          *
* * * * * * * * * * * * * * * * * *
*          * 0 0 0 0 *          * 0 0 0 0 *
*          * 0 0 0 0 *          * 0 0 0 0 *
*          * 0 0 0 0 *          * 0 0 0 0 *
*          * 0 0 0 0 *          * 0 0 0 0 *
* * * * * * * * * * * * * * * * * *
* 0 0 0 0 *          * 0 0 0 0 *          *
* 0 0 0 0 *          * 0 0 0 0 *          *
* 0 0 0 0 *          * 0 0 0 0 *          *
* 0 0 0 0 *          * 0 0 0 0 *          *
* * * * * * * * * * * * * * * * * *

```

j) ***Diamant :*** seulement pour n impair !

Enter an odd number of rows : 11

```

      *
    * *
  *   *
*     *
*       *
*         *
*           *
*             *
*               *
*                 *
*                   *

```


Exercice 15 : suites – boucle while

Ecrire un programme qui affiche dans une ligne les suites d'entiers suivantes, aussi longtemps que le terme de la suite reste inférieur à un seuil donné, lu au clavier. Déterminer aussi le nombre de termes affichés de chaque suite. Utiliser chaque fois une boucle **while**.

Suite	Définition des termes
1, 3, 6, 8, 16, 18, 36, 38, 76, ...	+2, puis $\cdot 2$, ...
1, 6, 16, 36, 76, ...	+2, puis $\cdot 2$, mais en une étape
1, 1, 2, 3, 5, 8, 13, 21, ...	suite de Fibonacci
1, 11, 111, 1111, 11111, ...	suite des répunits (repeated units)
$u_n = 1 + 2 + 3 + \dots + n, n \geq 1$	la somme des n premiers entiers non nuls
$v_n = 1^2 + 2^2 + 3^2 + \dots + n^2, n \geq 1$	la somme des carrés des n premiers entiers non nuls

Exercice 16 : triangles revisités

Modifier l'exercice 5 pour que l'utilisateur puisse étudier la nature de plusieurs triangles sans devoir relancer l'exécution du programme à chaque fois.

Exercice 17 : composer un nombre, généralisation

Généraliser l'exercice 8 de façon à ce que l'utilisateur puisse entrer autant de chiffres qu'il veut : l'entrée est terminée lorsqu'il tape un . Voici un exemple d'exécution :

```
Enter the first digit : 5
Enter the next digit : 6
Enter the next digit : 2
Enter the next digit : 0
Enter the next digit : 3
Enter the next digit : .
56203
```

Exercice 18 : décomposer un nombre, généralisation

Généraliser l'exercice 9 de façon à ce que l'utilisateur puisse entrer un nombre avec autant de chiffres qu'il veut. Remarquer que la version utilisant uniquement des strings fonctionne sans modification. Il suffira donc écrire une nouvelle version utilisant uniquement des entiers.

Exercice 19 : maximum et minimum d'une suite de nombres

Ecrire un programme qui affiche le plus grand et le plus petit nombre d'une suite de nombres réels entrés au clavier. Il est interdit d'utiliser les fonctions prédéfinies min et max dans Python. Le programme se termine si l'utilisateur entre un point (.) au clavier.

Voici deux exemples d'exécution :

```
Maximum and minimum of a sequence :
-----
Enter a number (stop loop with '.'): 5
Enter a number (stop loop with '.'): -2.8
Enter a number (stop loop with '.'): 3.6
Enter a number (stop loop with '.'): 7
Enter a number (stop loop with '.'): 7
Enter a number (stop loop with '.'): 0
Enter a number (stop loop with '.'): -5.44
Enter a number (stop loop with '.'): .
Maximum : 7.0
Minimum : -5.44
```

```
Maximum and minimum of a sequence :
-----
Enter a number (stop loop with '.'): .
Empty sequence : min and max not defined
```

Exercice 20 : suite de Collatz (ou de Syracuse)

Ecrire un programme qui affiche la suite de Collatz pour un entier naturel non nul donnée. Cette suite est définie de la manière suivante :

On part d'un entier naturel non nul ; s'il est pair, on le divise par 2 ; s'il est impair, on le multiplie par 3 et on ajoute 1. En répétant l'opération, on obtient une suite récurrente, c.-à-d. une suite d'entiers positifs dont chacun ne dépend que de son prédécesseur. Par exemple, à partir de 14, on obtient la suite de Collatz suivante : 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1. La suite s'arrête si on tombe sur 1.

Le programme devra aussi afficher le nombre de termes (18 dans l'exemple ci-dessus) et le maximum (52 dans l'exemple ci-dessus) de la suite.

On conjecture mais on n'a pas démontré jusqu'aujourd'hui que la suite de Collatz se termine pour chaque entier naturel. Exemple d'exécution :

```
Enter the first term of the Collatz sequence : 14
14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
The Collatz sequence starting with 14 has 18 terms.
The maximum of the sequence is 52.
```

Exercice 21 : algorithme d'Héron

Ecrire un programme qui implémenter l'algorithme d'Héron pour calculer \sqrt{a} :

L'utilisateur doit fournir :

- la valeur de a , dont on veut calculer la racine carrée ;
- x_0 , un réel > 0 quelconque, le premier terme de la suite de Héron;
- une précision $\varepsilon > 0$ avec laquelle on souhaite calculer \sqrt{a} .

La suite de Héron est définie récursivement par :

$$x_n = \begin{cases} x_0 & \text{si } n = 0 \\ \frac{1}{2} \left(x_{n-1} + \frac{a}{x_{n-1}} \right) & \text{si } n > 0 \end{cases}$$

On demande de calculer et d'afficher tous les termes de cette suite tant que l'écart (différence en valeur absolue) entre a et x_n^2 soit $> \varepsilon$. Exemple : pour calculer $\sqrt{2}$ en partant de $x_0 = 1$, on obtient $x_1 = \frac{3}{2} = 1,5$, $x_2 = \frac{17}{12} = 1,41666\dots$, $x_3 = \frac{577}{409} = 1,414215686\dots$ etc.

Remarquer que la suite converge toujours très vite vers \sqrt{a} (la convergence est quadratique, le nombre de décimales exactes double à chaque itération). Vous trouverez une explication simple de la méthode à l'adresse suivante (approche géométrique) :

https://fr.wikipedia.org/wiki/Méthode_de_Héron.

Exercice 22 : limite d'une suite récurrente

On considère la suite récurrente $(u_n)_{n \geq 0}$ définie par :

$$\begin{cases} u_0 = 1 \\ u_{n+1} = \frac{9}{6 - u_n} \end{cases} \quad \text{si } n > 0$$

Afficher les 20 premiers termes de cette suite. Vers quel réel semble-t-elle converger ?

(Expliquer ce résultat en dessinant le graphe de la fonction $f : x \mapsto \frac{9}{6-x}$.)

Exercice 23 : série harmonique

Pour $n \in \mathbb{N}^*$, on définit : $u_n = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$.

Trouver le plus petit entier n tel que $u_n > 10$. (La suite (u_n) est appelée série harmonique : elle diverge, c.-à-d. la limite de u_n , lorsque $n \rightarrow +\infty$ est $+\infty$.)

Exercice 24 : nombre secret à deviner

Créer une simulation du jeu « Guess a number » : le programme choisit un nombre secret entre 1 et n , n étant lu au clavier. A chaque essai de l'utilisateur, le programme affiche le message adéquat : « Your guess is too low. », « Your guess is too high. » ou « Well done! You needed ... guess(es). ». L'utilisateur peut interrompre la partie à tout moment en introduisant 0. Dans ce cas le programme révèle le nombre secret. Exemples d'exécution :

```
*** Guess a number ! ***

Enter the upper limit for the secret number : 100
Please enter your guess : 50
Your guess is too high.
Please enter your guess : 20
Your guess is too low.
Please enter your guess : 30
Your guess is too low.
Please enter your guess : 40
Your guess is too high.
Please enter your guess : 35
Your guess is too low.
Please enter your guess : 37
Your guess is too low.
Please enter your guess : 38
Well done! You needed 7 guess(es).
```

```
*** Guess a number ! ***

Enter the upper limit for the secret number : 50
Please enter your guess : 25
Your guess is too low.
Please enter your guess : 40
Your guess is too low.
Please enter your guess : 45
Your guess is too high.
Please enter your guess : 0
The secret number was 41.
```

Exercice 25 : test de primalité

Ecrire un programme qui teste si un nombre entier naturel donné n est premier ou non, en utilisant l'algorithme suivant :

- si $n \leq 1$ alors n n'est pas premier ;
- si $n > 2$ et divisible par 2, alors n n'est pas premier ;
- si n est divisible par un entier impair dans $[3, \sqrt{n}]$, alors n n'est pas premier ;
- sinon n est premier.

Exercice 26 : dynamique des populations

Une grande ville de 300'000 habitants et un village de 60'000 sont à proximité. Chaque année, 3% des habitants du village partent s'installer dans la ville tandis que 1% des habitants de la ville vont s'installer dans le village. On note u_n le nombre d'habitants en ville la n -ième année et v_n le nombre d'habitants du village la n -ième année. Initialement, on a donc $u_0 = 300'000$ et $v_0 = 60'000$.

Écrire un programme qui permet de calculer et d'afficher le nombre d'habitants u_n de la ville et v_n du village pendant 10 ans. Conjecturer la limite de ces deux suites.

Exercice 27 : lancers d'un dé

Ecrire un script qui permet de simuler une suite de lancers d'un dé jusqu'au $n^{\text{ième}}$ 6, (n donné par l'utilisateur). Le script doit écrire les lancers à l'écran en passant à la ligne après chaque 6 rencontré et en ajoutant entre () en fin de chaque ligne le nombre de lancers jusqu'au 6 suivant. On calculera en outre le nombre moyen de lancers jusqu'au 6 suivant. Pour $n = 4$, on obtient par exemple :

```
Nombre de 6 : 4
3 5 4 4 5 3 2 4 6 (9)
2 4 1 6 (4)
3 2 3 6 (4)
5 4 6 (3)
Nombre moyens de lancers pour obtenir le 6 suivant : 5.0
```

Exercice 28 : lancers d'un dé

Ecrire un programme qui calcule le périmètre d'un polygone à partir des coordonnées (x, y) de ses sommets. Continuer à lire les couples (x, y) jusqu'à ce que l'utilisateur entre une ligne vide. Dans ce cas, le programme doit afficher le périmètre du polygone fermé formé par tous les points entrés, dans l'ordre.

Exercice 29 : somme des chiffres / nombre renversé

Ecrire trois fonctions qui permettent respectivement :

- de calculer la somme des chiffres d'un entier donné
- de calculer la somme alternée des chiffres d'un entier donné (les chiffres sont précédés alternativement du signe + et du signe -.)
- de renverser un entier donné, par exemple 1234 devient 4321.

LISTES

Exercice 30 : diagramme à barres

Ecrire un programme qui demande à l'utilisateur d'entrer une liste d'entiers positifs. Les éléments de cette liste seront alors visualisés graphiquement sous forme de barres verticales dirigées vers le bas (diagramme à barres ici). Le symbole pour représenter un élément de barre est `'|'`. On l'obtient en tapant : `Alt+124`. Les barres doivent occuper 3 colonnes de l'écran et seront alignées à droite.

Exemple d'exécution :

```
Enter a sequence of positive integers (data) : 2 5 0 6 1 8 4
```

0	1	2	3	4	5	6

Exercice 31 : enlever les doubles d'une liste

Ecrire un script qui permet d'*enlever les éléments répétés (doubles)* d'une liste donnée d'entiers. Pour chaque élément répété, on affichera le nombre d'occurrences dans la liste originale. **Consignes à respecter :** a) La 1^{re} occurrence d'un élément répété doit rester dans la liste, les autres exemplaires de cet élément doivent être enlevés. b) Il est interdit dans la programmation d'utiliser une deuxième liste, on modifiera donc la liste originale en utilisant par exemple **del** ! Voici un exemple d'exécution :

```
Entrez les éléments de votre liste. Terminez par un '.'
Element suivant : 1
Element suivant : 2
Element suivant : 1
Element suivant : 3
Element suivant : 1
Element suivant : 4
Element suivant : 4
Element suivant : 5
Element suivant : 4
Element suivant : 5
Element suivant : 4
Element suivant : .
Liste originale : [1, 2, 1, 3, 1, 4, 4, 5, 4, 5, 4]
    1 apparaît    3 fois
    4 apparaît    4 fois
    5 apparaît    2 fois
Liste sans doubles : [1, 2, 3, 4, 5]
```

Exercice 32 : renverser l'ordre des éléments d'une liste

Ecrire un script (ou une fonction) qui permet de *renverser l'ordre* des éléments d'une liste d'entiers entrée par l'utilisateur *sans utiliser la méthode reverse*. Il est interdit dans la programmation d'utiliser une deuxième liste, on modifiera donc la liste originale. *Exemple d'exécution* :

```
Entrez les éléments de votre liste. Terminez par un '.'
Element suivant : 5
Element suivant : 6
Element suivant : 7
Element suivant : 8
Element suivant : .
Liste originale : [5, 6, 7, 8]
Liste renversée : [8, 7, 6, 5]
```

Exercice 33 : sommes de palindromes

On rappelle qu'un nombre palindromique (ou palindrome) est un entier naturel qui reste le même qu'on le lise de droite à gauche ou de gauche à droite. Par exemple : 47374, 101, 6, 0 et 9449 sont des palindromes.

Le programme demande à l'utilisateur d'entrer un entier naturel N .

- (1) Créer la liste croissante de tous les palindromes dans $[0, N]$.
- (2) Créer la liste croissante de tous les entiers dans $[0, N]$ qui *ne peuvent pas* s'écrire comme la somme de 2 palindromes.
- (3) Créer la liste croissante et sans doubles de tous les entiers dans $[0, N]$ qui *peuvent s'écrire* comme la somme de 3 palindromes (au plus). Combien contient-elle d'éléments ? Formulez une conjecture !

Exercice 34 : liste tirette

A partir de deux listes d'entiers données, par exemple $\mathbf{li1} = [4, 7, 3, 8]$ et $\mathbf{li2} = [1, 5, 8, 0, 2, 4, 0]$, on demande de créer et d'imprimer sur l'écran la *liste* « *tirette* », à savoir : $\mathbf{li} = [4, 1, 7, 5, 3, 8, 8, 0, 2, 4, 0]$. Le script copiera donc dans \mathbf{li} alternativement les éléments de $\mathbf{li1}$ et de $\mathbf{li2}$ (en commençant toujours par $\mathbf{li1}$), jusqu'à ce que l'on ait atteint la fin de la liste la plus courte. Ensuite les éléments restants de la liste la plus longue sont ajoutés à \mathbf{li} .

Exercice 35 : entiers de 1 à n dans un ordre aléatoire

Ecrire un script qui demande à l'utilisateur d'entrer un entier naturel non nul n . Le script place alors tous les entiers de 1 à n dans une liste, dans un *ordre aléatoire*. *Attention* : Chaque entier devra figurer une seule fois dans la liste. Par exemple si n

= 5, la liste obtenue pourrait être **[3,1,4,2,5]**. On pourra utiliser la fonction **randint** du module **random**. La fonction **shuffle** de ce module est interdite !

Exercice 36 : insertion d'un élément dans une liste croissante

Ecrire un script dans lequel l'ordinateur choisit au hasard une liste *strictement croissante* de 3 entiers **a**, **b** et **c** compris entre 1 et 100, par exemple **[14,27,56]**. Le programme demande ensuite à l'utilisateur d'entrer successivement des entiers positifs, qui sont alors insérés dans la liste de façon à ce qu'elle reste *croissante*. La nouvelle liste sera affichée après l'initialisation et après chaque insertion. Lorsque l'utilisateur entre un entier < 0 , l'exécution s'arrête. *Exemple d'exécution :*

```
Liste originale : [46, 81, 87]
Entrez un entier positif, <0 pour arrêter : 47
Nouvelle liste : [46, 47, 81, 87]
Entrez un entier positif, <0 pour arrêter : 81
Nouvelle liste : [46, 47, 81, 81, 87]
Entrez un entier positif, <0 pour arrêter : 100
Nouvelle liste : [46, 47, 81, 81, 87, 100]
Entrez un entier positif, <0 pour arrêter : 32
Nouvelle liste : [32, 46, 47, 81, 81, 87, 100]
Entrez un entier positif, <0 pour arrêter : -1
```

Exercice 37 : crible d'Erathosthène

Ecrire un script qui permet de générer la liste de tous les nombres premiers jusqu'à un entier n donné par l'utilisateur. On utilisera une version très rapide de l'algorithme du crible d'Erathosthène :

- On initialise une liste **bools** contenant $n + 1$ fois **True** et une liste initialement vide **primes** qui va contenir tous les nombres premiers $\leq n$, **n** étant donné par l'utilisateur.
- Comme 0 et 1 ne sont pas premiers, on redéfinit **bools[0] = bools[1]=False**.
- On parcourt ensuite la liste **bools** à partir de l'indice 2 : lorsque l'élément d'indice **p** est **True**, cet indice **p** est un nombre premier et on l'ajoute à **primes**.
- On élimine les autres multiples **i** de **p** dans la liste **bools** en posant : **bools[i]=False**. (Indication : utiliser **range** de façon astucieuse !)

Les nombres non éliminés sont alors les nombres premiers $\leq n$.

Exercice 38 : sous-listes monotones

On donne une liste d'entiers, par exemple :

[1, 2, 3, 3, 4, 5, 4, 4, 3, 2, 6, 8, 9, 12, 5, 14].

Ecrire une fonction **split_list** qui scinde une telle liste en sous-listes monotones (donc croissantes ou décroissantes). Pour l'exemple ci-dessus, la valeur de retour de la fonction est :

[[1, 2, 3, 3, 4, 5], [5, 4, 4, 3, 2], [2, 6, 8, 9, 12], [12, 5], [5, 14]].

En déduire une fonction qui permet de déterminer la plus longue sous-suite monotone de la liste donnée d'entiers, sans utiliser max.

Exercice 39 : Problèmes du défi Turing

Voici deux problèmes du défi Turing

- a) **Problème n° 90** : 276 lampes sont numérotées de 1 à 276. Pour passer le temps, 25 enfants appuient sur les interrupteurs à tour de rôle. Le premier enfant presse chaque interrupteur. Le second presse les boutons 2, 4, 6, etc. (tous les boutons ayant un numéro multiple de 2), le troisième appuie sur les boutons 3, 6, 9, etc. Le quatrième presse tous les boutons ayant un numéro multiple de 4, et ainsi de suite jusqu'au 25^e enfant. Avant le passage du premier enfant, toutes les ampoules sont éteintes. Combien d'ampoules seront allumées après le passage des 25 enfants ?
- b) **Problème n° 60** : Les 2013 membres d'une secte ont décidé de se suicider. Pour effectuer le rituel funèbre, ils se mettent en cercle, puis se numérotent dans l'ordre de 1 à 2013. On commence à compter, à partir du numéro 1. Toutes les 7 positions, la personne désignée devra mourir. Ainsi, la première à mourir aura le n° 7, la deuxième le 14, la troisième le 21, etc. Vous faites partie de cette secte, mais vous n'avez aucune envie de mourir! Il s'agit donc de trouver la position sur le cercle qui vous permettra d'être désigné en dernier, et donc d'échapper à la mort.

<http://www.nymphomath.ch/turing/enonces.php>

Exercice 40 : tri par sélection et recherche dichotomique

- (1) Ecrire une fonction **tri_selection(liste)** qui trie les éléments de la liste (de nombres ou de strings) par ordre croissant à l'aide de l'algorithme de tri par sélection : on recherche d'abord le plus petit élément de la liste et on l'échange avec l'élément d'indice 0, on recherche le second plus petit élément dans les éléments restants (à partir de l'indice 1 donc) et on l'échange avec l'élément

d'indice 1, et ainsi de suite : on répète le procédé jusqu'à ce qu'on arrive à la fin de la liste.

- (2) Ecrire une fonction **rech_dicho(liste, cle)** qui retourne la position (c.-à-d. l'indice) d'un élément (= **cle**) dans une **liste triée par ordre croissant**. L'algorithme à mettre en œuvre, appelé algorithme de la recherche dichotomique (du grec διχοτομία, *dikhonomia* = « division en deux parties »), est le suivant : on compare l'élément au milieu de la liste à la clé cherchée ; s'il est égal à la clé, la fonction retourne l'indice de cet élément, s'il est plus grand que la clé, on recherche la clé dans la moitié gauche de la liste, s'il est plus petit que la clé, on recherche la clé dans la moitié droite de la liste. Si la clé ne se trouve pas dans la liste, la fonction retourne -1.

Exercice 41 : filtrage de listes

- (3) Ecrire une fonction **positive_numbers** qui prend en entrée une liste de nombres et qui retourne la liste avec tous les nombres positifs appartenant à cette liste.
- (1) Ecrire une fonction **numbers_with_digit** qui prend en entrée une liste de nombres et qui retourne les nombres contenant le chiffre **n** (paramètre). **Indication** : transformer chaque nombre de la liste en un string en utilisant **str()**.
- (2) Ecrire une fonction **inside_circle** qui prend en entrée une liste de points du plan (un point est un tuple de la forme **(x,y)**) et qui retourne la liste des points à l'intérieur du cercle de centre l'origine et de rayon **r** (paramètre).
- (3) Ecrire une fonction **inside_rectangle** qui prend en entrée une liste de points du plan et qui retourne la liste des points à l'intérieur du rectangle, déterminé par deux coins **(x1,y1)** et **(x2,y2)** n'appartenant pas au même côté.
- (4) Ecrire une fonction **words_with_greater_length** qui prend en entrée une phrase (de type **str**) et qui compte le nombre de mots dans cette phrase ayant une longueur $\geq n$ (paramètre). **Indication** : on peut créer une liste des mots de la phrase à l'aide de la méthode **split** d'un string.

Tester les fonctions à l'aide d'exemples.

Exercice 42 : suites palindromiques

Partant d'un entier naturel quelconque N , on obtient la **suite palindromique** associé à cet entier de la manière suivante :

- On inverse l'ordre des chiffres de N et on ajoute N à ce nombre inversé.

- On recommence l'opération avec le nouveau nombre jusqu'à ce qu'on obtienne un palindrome.

Exemples :

<i>N</i>	Suite palindromique
13	13, 44
29	29, 121
1048	1048, 9449
64	64, 110, 121
87	87, 165, 726, 1353, 4884
89	89, 187, 968, 1837, ... , 8813200023188 (24 étapes en tout !)
196	196, 887, 1675, ... (suite infinie ??)

On ne sait pas à ce jour si la suite palindromique de 196 se termine par un palindrome ou non, même après plus de 700 millions d'itérations de renversement-addition ...

Ecrire un script qui, à partir d'un entier naturel *N* donné détermine et affiche sa suite palindromique. On utilisera une fonction qui permet de calculer le terme suivant de la suite palindromique, pour un terme donné. Le programme devra s'arrêter lorsque le nombre d'éléments de cette suite dépasse une limite que l'utilisateur pourra fixer. Pour plus d'informations, voir <http://www.jasondoucette.com/worldrecords.html>.

Exercice 43

Soit une liste d'entiers contenant uniquement des 0 et des 1, placés au hasard. (On peut imaginer par exemple que c'est une suite de résultats du lancer d'une pièce de monnaie, 0 = pile, 1 = face). Une série de 0 (ou de 1) dans cette liste est une suite d'éléments consécutifs et égaux à 0 (resp. 1). Ecrire une fonction qui cherche la plus longue série de 0 ou de 1 dans une liste donnée, l'indice de son premier élément et sa longueur. Par exemple, si **liste = [0,1,1,1,0,1,0,0,0,0,1]**, la plus longue série de 1 commence à l'indice 1 et a comme longueur 3.

Exercice 44

Pour un entier naturel non nul *n* donné, écrire :

- Une fonction **premiers_avec(n)** qui retourne la liste des entiers *m* tels que $\text{pgcd}(m, n) = 1$, donc qui sont premiers avec *n*.
- Une fonction **diviseurs(n)** qui retourne, la liste avec tous les diviseurs de *n*.
- Une fonction **sigma(n)** qui calcule la somme de tous les diviseurs de *n*.

Tester les fonctions.

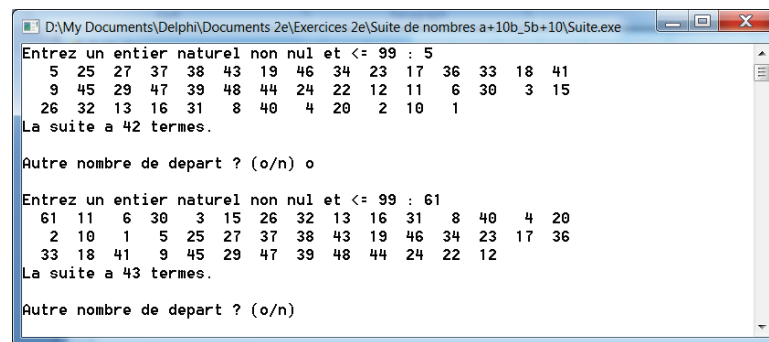
Exercice 45

On construit une suite de nombres entiers de la manière suivante :

- on choisit un nombre de départ a_1 parmi les entiers de 1 à 99 ;
- on obtient a_{n+1} en multipliant le chiffre des unités de a_n par 5 et en ajoutant au résultat le chiffre des dizaines de a_n .
- on arrête le procédé dès que $a_{n+1} \in \{a_1, a_2, \dots, a_n\}$, c.-à-d. on tombe sur un élément déjà obtenu auparavant. Le dernier terme de la suite est alors a_n .

Ecrire un programme, qui pour un entier de départ a_1 donné, écrit à l'écran tous les termes de la suite ainsi définie et compte le nombre de termes de cette suite.

- (1) Que se passe-t-il si on part a) d'un multiple de 7 ? b) d'un entier < 50 et non multiple de 7 ?
- (2) Pour quel entier de départ obtient-on la suite a) la plus courte b) la plus longue ?



Exercice 46

Leonhard Euler (1707-1783) a trouvé le polynôme quadratique remarquable suivant :

$$p(n) = n^2 + n + 41.$$

De façon surprenante, en remplaçant n par les valeurs successives de n allant de 0 à 39, on montre que $p(n)$ est un **nombre premier**. Toutefois, lorsque $n = 40$,

$$p(40) = 40^2 + 40 + 41 = 40 \cdot (40 + 1) + 41 = 40 \cdot 41 + 41 = 41^2$$

est divisible par 41 et n'est donc plus premier. A l'aide de l'ordinateur, un polynôme encore plus incroyable a été découvert :

$$p(n) = n^2 - 79n + 1601$$

Il fournit 80 nombres premiers pour les valeurs successives de n allant de 0 à 79.

Ecrire un programme qui permet à l'utilisateur d'entrer deux entiers relatifs a et b et d'afficher la liste des nombres premiers générée par le polynôme $p(n) = n^2 + an + b$.

On affichera donc la suite $p(0)$, $p(1)$, $p(2)$, ... jusqu'à ce que l'on rencontre pour la première fois un entier $p(i)$ non premier (qui ne sera plus affiché).

Le programme comprendra :

- a) la fonction **est_premier(n)** qui permet de déterminer si un entier naturel donné est premier ou non ;
- b) la fonction **liste_premiers(a,b)** qui retourne la liste **[p(0),p[1],p[2]...]** des nombres premiers générée par le polynôme $p(n) = n^2 + an + b$.

Exercice 47 : Polynômes

Ecrire un programme qui permet d'effectuer les opérations usuelles sur les polynômes à **coefficients réels** : a) addition et soustraction, b) multiplication, c) puissance d'un polynôme d) évaluation en un réel et e) dérivation. Pour chaque opération, on écrira une fonction adéquate.

Pour représenter un polynôme $p(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$ dans Python, on écrit dans une liste ses coefficients **suivant les puissances croissantes** de la variable :

$$\mathbf{p} = [\mathbf{a_0}, \mathbf{a_1}, \mathbf{a_2}, \dots, \mathbf{a_n}],$$

de sorte que l'élément d'indice k de la liste est toujours le coefficient de x^k . Par exemple le polynôme $p(x) = 3 + 2x - 5x^2 + 7x^4$ sera représenté par la liste

$$\mathbf{p} = [\mathbf{3}, \mathbf{2}, \mathbf{-5}, \mathbf{0}, \mathbf{7}] \text{ car :}$$

- 3 est le coefficient de x^0 ,
- 2 est le coefficient de x^1 ,
- -5 est le coefficient de x^2 ,
- 0 est le coefficient de x^3 ,
- 7 est le coefficient de x^4 .

Exercice 48 : opérations sur les matrices

Ecrire un script qui permet de calculer la somme et le produit de 2 matrices à coefficients **entiers**, de dimensions quelconques, à condition que les opérations ont un sens. Les matrices seront implémentées comme des listes de listes. Par exemple :

$$\mathbf{a} = [[\mathbf{1}, \mathbf{2}, \mathbf{4}, \mathbf{2}], [\mathbf{3}, \mathbf{8}, \mathbf{5}, \mathbf{3}], [\mathbf{8}, \mathbf{7}, \mathbf{7}, \mathbf{3}]]$$

définit la matrice à 3 lignes et 4 colonnes :

$$a = \begin{pmatrix} 1 & 2 & 4 & 2 \\ 3 & 8 & 5 & 3 \\ 8 & 7 & 7 & 3 \end{pmatrix}$$

et **a[i][j]** est l'élément à l'intersection de la **ligne** d'indice **i** et de la **colonne** d'indice **j**. Par exemple : **a[1][2]=5**. Le programme comportera :

- a) Une fonction **get_matrix(type='m')** permettant de définir les éléments d'une matrice, **ligne par ligne**. Le paramètre **type** permet de préciser le type de saisie, **'m'**

signifiant une entrée manuelle, '**r**' une entrée aléatoire avec des entiers compris entre -10 et 10. Voici un exemple d'exécution de cette fonction lorsque **type = 'm'** :

```
Entrée manuelle (m) ou aléatoire (r) ? m
Nombre de lignes : 3
Nombre de colonnes : 2
Ligne n°1 : 4 5
Ligne n°2 : -6 2 7
Ligne n°2 : -6 2
Ligne n°3 : -10 0
```

Remarquer que le programme redemande la ligne n° 2 puisque l'utilisateur a écrit un élément de trop la 1^{re} fois.

b) Une fonction **display_matrix(m)**, qui affiche une matrice sur l'écran, ligne par ligne. Chaque colonne de la matrice doit occuper 3 colonnes de l'écran (utiliser la méthode **format** d'un string.)

```
Matrice m1
  4    5
 -6    2
-10    0
```

c) Deux fonctions **sum_matrix(a,b)** et **product_matrix(a,b)**, retournant respectivement la somme et le produit de deux matrices, lorsque ceux-ci existent ; lorsque la somme ou le produit ne sont pas définis, un message d'erreur sera affiché par les fonctions et elles retournent une liste vide.

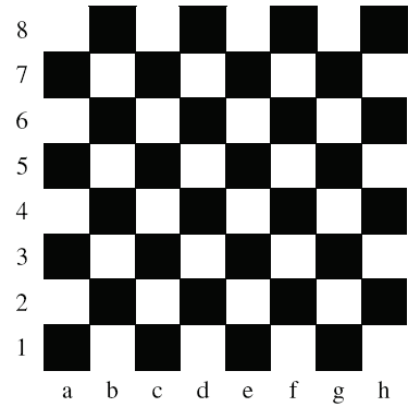
Des exemples d'exécution sont à faire pour tester le programme.

STRINGS

Exercice 49

Les positions des pièces sur un échiquier sont identifiées par des coordonnées formées d'une lettre (de A à H) suivie d'un chiffre (de 1 à 8). La lettre identifie la colonne, tandis que le nombre identifie la rangée (ou ligne), comme indiqué sur le schéma ci-contre.

Ecrire une fonction **case_color** qui prend en entrée une position, par exemple 'D5' (ou 'd5') et qui retourne la couleur ('black' ou 'white') de la case correspondante.



Exercice 50

Ecrire un programme qui affiche les lettres répétées d'un string de la plus fréquente à la moins fréquente. On peut supposer que le string est composé uniquement de lettres minuscules (et ne contient pas de caractères spéciaux). Par exemple, si l'utilisateur entre le string :

```
'thequickbrownfoxjumpsoverthelazydog',
```

le programme devra afficher :

```
o 4
e 3
u 2
h 2
r 2
t 2
```

Exercice 51

Ecrire une fonction **longest_word(phrase)** qui prend en entrée une phrase de type string (mots séparés par un espace) et qui retourne le mot le plus long de cette **phrase**. (Utiliser la méthode **split** d'un string pour transformer la phrase en une liste de mots.) Améliorer la fonction afin qu'elle ne tienne pas compte des caractères spéciaux `\,`, `\.`, `\:`, `\;`, `\?`, et `\!`.

Exercice 52

a) Ecrire une fonction **permute_first_last(str1)** qui prend en entrée un string **str1** et qui retourne le mot obtenu en échangeant la première et la dernière lettre de **str1**.

b) Ecrire une fonction **permute_letters(str1,i,j)** qui prend en entrée un string **str1** et qui retourne le mot obtenu en échangeant les lettres d'indices **i** et **j** de **str1**.

Exercice 53

Ecrire une fonction **permute_second_parts(s1,s2)** qui prend en entrée deux strings **s1** et **s2** contenant exactement un trait d'union et qui intervertit et écrit à l'écran les deux parties derrière les traits d'union. Donc par exemple : **croque-monsieur** et **porte-avions** deviennent respectivement **croque-avions** et **porte-monsieur**.

Exercice 54

Ecrire une fonction **count_different_char** qui compte le nombre de lettres distinctes dans une chaîne de caractères. Par exemple : **count_different_char('mathematiques')** retourne 9.

Exercice 55

Ecrire une fonction **all_char** qui teste si un string (formé uniquement de lettres minuscules et de caractères spéciaux), contient toutes les lettres de l'alphabet.

Exercice 56

Ecrire une fonction **with_spaces** qui insère entre deux lettres successives d'un mot donné un espace. Par exemple : **with_spaces('info')** retourne **'i n f o'**.

Exercice 57

Ecrire une fonction **random_order** qui prend en entrée un mot et qui retourne un mot formé exactement des mêmes lettres, mais dans un ordre aléatoire. Par exemple : **random_word('informatique') = 'utonirafqmei'**.

Exercice 58

a) Ecrire une fonction **to_decimal_hours** qui prend en entrée une durée de type **string** et dans le format **'...h...m...s'** (heures, minutes, secondes) et qui retourne cette durée en heures décimales. Par exemple : **to_decimal_hours ('5h30m')=5.5h**.

b) Ecrire la fonction réciproque **to_hms** de la fonction précédente.

Par exemple : **to_hms (5.5)=5h30m0.0s**

Exercice 59

Ecrire une fonction **alphabetic_string** qui prend en entrée un caractère majuscule et un entier **n** non nul et qui retourne le string formé de **n** lettres successives de l'alphabet et commençant avec la lettre fournie par l'utilisateur. Si on dépasse la lettre **'Z'**, on recommence cycliquement avec **'A'**. Par exemple :

```
alphabeticString('T',5)='TUVWX'  
alphabeticString('X',8)='XYZABCDE'  
alphabeticString('Z',28)='ZABCDEFGHJKLMNOPQRSTUVWXYZA'
```

Exercice 60

(1) Ecrire une fonction booléenne **est_palindrome** qui teste si une chaîne de caractères formée de lettres majuscules est un palindrome, c.-à-d. un texte dont l'ordre des lettres reste le même qu'on le lise de gauche à droite ou de droite à gauche comme dans les phrases «ZEUS A ETE A SUEZ» ou « EH ! CA VA LA VACHE ? ». On ne prend en considération que les lettres, pas les espaces ou autres caractères spéciaux.

(2) Ecrire une fonction booléenne **est_anagramme** qui teste si deux chaînes de caractères données sont des anagrammes l'une de l'autre. Une anagramme d'un mot est un mot différent formé avec exactement les mêmes lettres. Par exemple :

a) **anagramme(algorithme,logarithme)** retourne **true** et

b) **anagramme(test,tete)** retourne **false**.

Exercice 61 – Le code de César

Le code de César est la méthode de cryptographie la plus ancienne de l'histoire. César l'a utilisée pour envoyer des messages codés à ses généraux. L'idée est simple : toute lettre du message est remplacée par la lettre qui se trouve décalée de **n** places à droite dans l'alphabet. Les lettres non alphabétiques ne sont pas codées. Par exemple si **n =**

3, on remplace **A** par **D**, **B** par **E**, **C** par **F**, **D** par **G**, etc... Remarquer que **X** devient **A**, **Y** devient **B** et **Z** devient **C**.

Code de César avec un décalage de 3 lettres :

Texte clair : A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Texte codé : D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

Exemple avec n = 3 :

Texte clair :

**JE LEVAI LES YEUX VERS LE SOLEIL IL ETAIT BAS ; DANS MOINS D'UNE HEURE
IL ARRIVERAIT JUSTE AU-DESSUS DES BRANCHES SUPERIEURES DU VIEUX CHENE.**

Texte codé :

**MH OHYDL OHV BHXA YHUV OH VROHLO LO HWDLW EDV ; GDQV PRLQV G'XQH KHXUH
LO DUULYHUDLW MXVWH DX-GHVVXV GHV EUDQFKHV VXSHULHXUHV GX YLHXA FKHQH.**

- (1) Ecrire une fonction **encrypt(message, n)** qui permet de coder un texte (avec uniquement des majuscules ou des caractères spéciaux) en utilisant la méthode de César avec un décalage de n places.
- (2) Ecrire une fonction **decrypt(message)** qui essaie de décoder un texte codé avec la méthode de César en exploitant le fait que la lettre la plus fréquente est en général la lettre '**E**'. Bien sûr cette méthode ne saura être couronnée de succès que si le texte entré est assez long.

Exercice 62 : Calculatrice sur les fractions

```
Une petite calculatrice sur les fractions
=====
Entrez une fraction sous la forme a/b avec b <> 0: 35/-28
Choisissez une opération parmi +, -, *, /, = : =
Résultat final : -5/4
Autre calcul (o/n) ? : o

Entrez une fraction sous la forme a/b avec b <> 0: 2/3
Choisissez une opération parmi +, -, *, /, = : -
Entrez une fraction sous la forme a/b avec b <> 0: 1/5
Résultat intermédiaire : 7/15
Choisissez une opération parmi +, -, *, /, = : *
Entrez une fraction sous la forme a/b avec b <> 0: 5/-14
Résultat intermédiaire : -1/6
Choisissez une opération parmi +, -, *, /, = : =
Résultat final : -1/6
Autre calcul (o/n) ? : n
```

Réaliser un script qui permet de faire des opérations sur des fractions. Les fractions sont entrées sous la forme "**n/d**" où **n** est le numérateur et **d** le dénominateur. Le

programme devra aussi reconnaître des entiers, donc des inputs du type "**n**" (sans /). Les fractions seront implémentées sous la forme de tuples **(n,d)**. Le programme doit permettre : a) de simplifier des fractions et b) d'effectuer plusieurs opérations élémentaires successives (addition, soustraction, multiplication et division) comme dans l'exemple d'exécution ci-dessous. On ne demande pas que la calculatrice respecte les règles de priorité.

Exercice 63 : Polynômes

Reprendre l'exercice 34 sur les polynômes. Améliorer le programme de sorte que l'utilisateur pourra entrer les polynômes sous la forme d'un *string* et le programme affichera aussi les polynômes sous forme de string. Il faudra écrire donc deux fonctions **str2list** et **list2str** permettant respectivement de transformer un string du type ' $a_0 + a_1x + a_2x^2 + \dots + a_nx^n$ ' en la liste des coefficients et réciproquement. Un menu permettra de choisir l'opération à effectuer. Voir exemple d'exécution ci-dessous.

Opérations sur les polynômes

=====

- 1 : Ajouter un polynôme
- 2 : Effacer un polynôme
- 3 : Afficher les polynômes
- 4 : Additionner des polynômes
- 5 : Soustraire des polynômes
- 6 : Multiplier des polynômes
- 7 : Puissance d'un polynôme
- 8 : Dérivée d'un polynôme
- 9 : Evaluation d'un polynôme
- 10 : Quitter

Votre choix ? 1

Entrez le polynôme : $1+3x-5x^2$

Votre choix ? 1

Entrez le polynôme : $2x-4$

Votre choix ? 6

0 : $1.0+3.0x-5.0x^2$

1 : $-4.0+2.0x$

Entrez les numéros des polynômes à multiplier : 0 1

Résultat : $-4.0-10.0x+26.0x^2-10.0x^3$

Votre choix ? 3

0 : $1.0+3.0x-5.0x^2$

1 : $-4.0+2.0x$

2 : $-4.0-10.0x+26.0x^2-10.0x^3$

Votre choix ? 4

0 : $1.0+3.0x-5.0x^2$

1 : $-4.0+2.0x$

2 : $-4.0-10.0x+26.0x^2-10.0x^3$

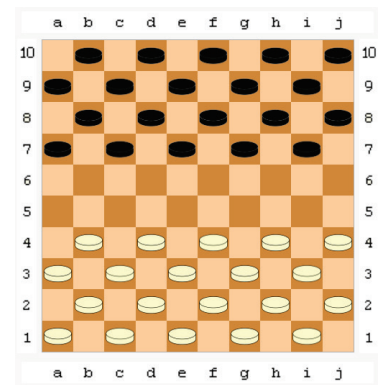
Entrez les numéros des polynômes à additionner : 1 2

Résultat : $-8.0-8.0x+26.0x^2-10.0x^3$

DICTIONNAIRES

Exercice 64 : Jeu de dames

Créer un dictionnaire **checkers** avec la position initiale au jeu de dames. Les clés sont des tuples du type `('A',1)` ou `('B',4)`, ou `('C',9)`, représentant les cases noires. Les valeurs sont `'b'` pour les pièces blanches, `'n'` pour les pièces noires et `' '` pour les cases noires non occupées. On ne mettra pas les cases blanches dans le dictionnaire car le jeu se déroule uniquement sur les cases noires.



Exercice 65 : Fréquences de lettres

Ecrire une fonction **char_frequency(my_str)** qui retourne un dictionnaire dont les clés sont les lettres (les caractères spéciaux sont permis) de **my_str** et les valeurs sont les fréquences (nombres d'occurrence) de ces lettres. Tester la fonction sur le mot `'google.com'`.

Exercice 66 : Dictionnaire combiné

Ecrire une fonction **combine_dico** qui combine deux dictionnaires donnés en additionnant les valeurs pour des clés communes. Par exemple, en prenant comme arguments :

```
d1 = {'a': 100, 'b': 200, 'c':300} et
```

```
d2 = {'a': 300, 'b': 200, 'd':400},
```

la fonction doit retourner : `{ 'a': 400, 'b': 400, 'd': 400, 'c': 300 }`

Exercice 67 : Mot aléatoire avec valeurs d'un dictionnaire

Ecrire une fonction **mot_alea(my_dico)** avec un paramètre **my_dico** de type dictionnaire, dont les clés sont des strings non vides distincts et les valeurs associées des entiers > 0 , par exemple :

```
my_dico = {'dic' :3, 'do' :4, 'da' :2}.
```

La fonction retourne un string aléatoire composé des strings clés du dictionnaire, chaque clé apparaissant avec la fréquence qui lui est associée dans le dictionnaire. Par exemple :

mot_alea(mydico) pourrait être égal à : `'dodicdodadicdicdadodo'`

Exercice 68 : Chiffres romains

Ecrire deux fonctions **rom2dec** et **dec2rom** et qui permettent de transformer respectivement un nombre écrit en chiffres romains en notation décimale habituelle et réciproquement. On peut se limiter aux entiers ≤ 5000 . Voir Wikipedia pour les règles.

On utilisera les dictionnaires réciproques :

SYMBOLS = {1:'I', 5:'V', 10:'X', 50:'L', 100:'C', 500:'D', 1000:'M'}

VALUES = {'I':1, 'V':5, 'X':10, 'L':50, 'C':100, 'D':500, 'M':1000}

(Une petite astuce : on peut définir **VALUES** en compréhension à partir de **SYMBOLS** !)

Exercice 69 : Polynômes

Reprendre l'exercice 34 : implémenter cette fois-ci les polynômes sous la forme d'un dictionnaire. Par exemple le polynôme $p(x) = 3 + 2x - 5x^2 + 7x^4$ sera représenté par le dictionnaire

p = {0:3, 1:2, 2:-5, 4:7}

L'avantage de cette représentation est qu'on n'a pas besoin d'inclure les termes de coefficient 0 dans le dictionnaire. Pour l'exemple ci-dessus, c'est l'élément **3:0** qui est superflu.

EXERCICES DE SYNTHÈSE - JEUX

Exercice 70 : Scrabble

Au Scrabble, il y a 102 jetons contenant soit une lettre soit un Joker. A chaque lettre on associe un certain score. Le score total d'un mot est la somme des scores de ses lettres. Le nombre de jetons de chaque lettre et le score associé à chaque lettre dans la version française sont donnés ci-dessous:



- 0 point : **Joker** ×2 (représenter le **Joker** par le caractère *)
- 1 point : **E** ×15, **A** ×9, **I** ×8, **N** ×6, **O** ×6, **R** ×6, **S** ×6, **T** ×6, **U** ×6, **L** ×5
- 2 points : **D** ×3, **M** ×3, **G** ×2
- 3 points : **B** ×2, **C** ×2, **P** ×2
- 4 points : **F** ×2, **H** ×2, **V** ×2
- 8 points : **J** ×1, **Q** ×1
- 10 points : **K** ×1, **W** ×1, **X** ×1, **Y** ×1, **Z** ×1

- (1) Créer un dictionnaire (constant) **SCORES** qui contient les scores des lettres. Les éléments du dictionnaire sont donc : **'*':0**, **'E':1**, **'A':1** ...

- (2) Créer un dictionnaire (constant) **TILES** représentant les 102 jetons. Ce dictionnaire contient des éléments du type **'*':2**, **'E':15** etc. Donc à chaque lettre, on associe comme valeur le nombre de jetons initialement dans le jeu. En déduire la liste **letter_bag**, qui représente le sac mélangé avec les 102 jetons individuels. Pour mélanger les lettres dans le sac, on pourra faire appel à la fonction **shuffle** du module **random**.
- (3) Télécharger la liste **'animals.txt'** qui se trouve dans le répertoire avec les exercices. Placer ce fichier dans le répertoire avec votre solution. Ouvrir le fichier dans Python avec **file = open('animals.txt')**. (La variable **file** est de type **file**. Consulter le site : <https://www.pythonforbeginners.com/files/reading-and-writing-files-in-python> pour de plus amples informations.) Créer une liste contenant tous les animaux du fichier **'animals.txt'** avec l'instruction **animals = list(file)**. Remarquer que chaque nom d'animal dans la liste (sauf le dernier) se termine par la séquence **'\n'**, qui traduit le passage à la ligne dans le fichier **'animals.txt'**. Trouver un moyen pour éliminer les **'\n'**. On aura donc : **animals = ['ABEILLE', 'AIGLE', 'ARAIGNEE', ...]**.
- (4) Ecrire une fonction **get_score** qui calcule le score d'un mot donné en paramètre. En déduire le dictionnaire **animal_scores** qui contient tous les animaux avec leur score comme valeur associée.
- (5) Ecrire une fonction **get_max_score** qui permet de déterminer un animal dans la liste ayant le score maximum. La fonction doit retourner le nom de l'animal et son score.
- (6) a) Ecrire une fonction **get_keys(dico, value)** qui retourne la liste de toutes les clés d'un dictionnaire **dico** dont la valeur associée est **value**. b) Déterminer à l'aide de cette fonction tous les animaux dont le score est 10. (C'est ce que l'on appelle une recherche inverse.) c) Est-ce qu'il y a un autre animal avec le score maximal (cf. question précédente) ?
- (7) Ecrire une fonction **get_tiles(n)** qui permet à un joueur de piocher **n** jetons dans le sac. Attention, les jetons que le joueur obtient doivent être retirés du sac : la variable **letter_bag** devra donc changer.
- (8) a) Ecrire une fonction **can_be_done(word, my_tiles)** qui retourne un tuple dont le 1^{er} élément est **True** ou **False** suivant que le mot **word** peut être composé avec les jetons contenus dans la liste **my_tiles** ou non. Veiller à ce que l'argument original **my_tiles** ne soit pas changé par la fonction au cours d'un appel. Le 2^e

élément du tuple est la liste des jetons utilisés pour composer le mot si c'est possible, la liste vide sinon. **Attention** : pensez aux **Jokers** !

b) Quels noms d'animaux peut-on composer avec les jetons contenus dans

`['A', 'E', 'R', '*', '*', 'U', 'V']` ?

Réponse :

FAUNE `['*', 'A', 'U', '*', 'E']`
OURS `['*', 'U', 'R', '*']`
RAT `['R', 'A', '*']`

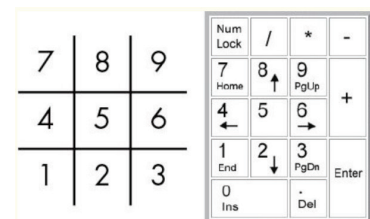
c) Maintenant un joueur pioche 12 jetons au hasard dans le sac avec les lettres. Afficher les mots qu'il peut former avec ces lettres, le score de chaque mot possible ainsi que les jetons à utiliser pour former ce mot. *Exemple :*

Mes lettres :
`['E', 'N', 'A', 'E', 'P', 'I', 'L', 'F', 'S', 'G', 'L', 'Q']`
AIGLE 6 `['A', 'I', 'G', 'L', 'E']`
LAPIN 7 `['L', 'A', 'P', 'I', 'N']`
SINGE 6 `['S', 'I', 'N', 'G', 'E']`

Exercice 71 (Tic-tac-toe)

Ecrire un programme qui permet à deux joueurs (1 et 2, variable **player**) de faire un match de Tic-tac-toe. Un match est composé de plusieurs parties, dont le nombre est fixé à l'avance par les adversaires. Les joueurs commencent alternativement à jouer en premier. A chaque joueur est associé un symbole représentant ses coups : **'X'** pour le joueur 1 et **'O'** pour le joueur 2. On introduira donc le dictionnaire constant **SYMBOL = {1:'X', 2:'O'}**. A chaque joueur est également associé son score, c.-à-d. le nombre de parties qu'il a gagnées. On utilisera à cette fin le dictionnaire **scores = {1:0, 2:0}**.

Les coups seront introduits à l'aide du clavier numérique : chaque touche représente la case avec la même position sur la grille du jeu, comme le montre le schéma ci-contre :



On utilisera un dictionnaire **board** (variable globale) à 9 éléments pour représenter la grille du jeu. Initialement les éléments de ce dictionnaire sont de la forme **i : ' '**, avec **i** variant de 1 à 9 (grille vide). Lorsqu'un joueur joue sur la case n° **i**, **board[i]** prendra la valeur correspondant à son symbole. Les joueurs jouent en introduisant à tour de rôle le numéro de la case qu'ils choisissent.

A la fin de chaque partie, le vainqueur sera affiché et les scores des deux joueurs seront actualisés. A la fin du match, les scores finaux et le vainqueur global seront affichés. Voir exemple d'exécution.

On utilisera des fonctions afin de rendre le code lisible :

- **draw_board()** : affiche le jeu sur l'écran. On utilisera la variable globale **board** dans cette fonction : voilà pourquoi la fonction n'a pas besoin de paramètres.
- **board_full()** : retourne **True** si le **board** est complètement rempli, **False** sinon.
- **is_won()** : retourne **True** si l'un des joueurs a gagné la partie, **False** sinon. (Il faudra tester les 3 types d'alignements.)
- **game_over()** : retourne **True** si la partie en cours est terminée, **False** sinon.
- **display_scores()** : affiche les scores des joueurs.
- **move(player)** : demande au joueur **player** d'entrer son coup et en vérifie la validité. Ensuite la grille de jeu sera redessinée et une décision sera prise quant à la partie en cours.

Exemple d'exécution (seulement la 1re partie et la décision finale) :

```
TICTACTOE
=====
Nombre de parties : 3
=====
Partie : 1
=====
  |  | 
---|---|---
  |  | 
---|---|---
  |  | 
Votre coup joueur 1 : 5
  |  | 
---|---|---
  | X | 
---|---|---
  |  | 
Votre coup joueur 2 : 1
  |  | 
---|---|---
  | X | 
---|---|---
 O |  | 
Votre coup joueur 1 : 7
 X |  | 
---|---|---
  | X | 
---|---|---
 O |  | 

Votre coup joueur 2 : 2
 X |  | 
---|---|---
  | X | 
---|---|---
 O | O | 
Votre coup joueur 1 : 3
 X |  | 
---|---|---
  | X | 
---|---|---
 O | O | X
Joueur 1 a gagné la partie 1
=====
Scores :
Joueur 1 : 1 Joueur 2 : 0
=====

# ... Les autres parties ne sont pas
# affichées ici...

=====
Scores :
Joueur 1 : 3 Joueur 2 : 0
=====
Joueur 1 a gagné le match ! BRAVO !
```

Exercice 72

Ecrire un programme qui permet de représenter graphiquement un chemin aléatoire reliant le coin supérieur gauche au coin inférieur droit d'une grille rectangulaire, dont le nombre de lignes et de colonnes pourra être choisi par l'utilisateur. Le chemin doit avoir la longueur minimale, c.-à-d. on pourra uniquement se promener d'une position vers la droite ou d'une position vers le bas dans la grille. Exemple :

```
Entrez le nombre de lignes : 10
Entrez le nombre de colonnes : 20
1  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
2  3  4  5  6  7  8  9  .  .  .  .  .  .  .  .  .  .  .  .  .
.  .  .  .  .  .  .  10 .  .  .  .  .  .  .  .  .  .  .  .
.  .  .  .  .  .  .  11 .  .  .  .  .  .  .  .  .  .  .  .
.  .  .  .  .  .  .  12 13 14 .  .  .  .  .  .  .  .  .  .
.  .  .  .  .  .  .  .  .  15 16 17 18 .  .  .  .  .  .  .
.  .  .  .  .  .  .  .  .  .  .  .  19 20 21 .  .  .  .  .
.  .  .  .  .  .  .  .  .  .  .  .  .  22 23 24 25 .  .
.  .  .  .  .  .  .  .  .  .  .  .  .  .  26 27 28
.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  29
```

La difficulté consiste à choisir un chemin au hasard : pour cela, on remarque que pour l'exemple d'exécution ci-dessus, parmi les $(10-1) + (20-1) = 28$ transitions à faire (vers la droite ou vers le bas), il suffit de choisir les $10-1=9$ endroits où on descend. Le chemin obtenu dans l'exemple d'exécution peut donc être représenté par la liste :

```
[1,0,0,0,0,0,0,0,1,1,1,0,0,1,0,0,0,1,0,0,1,0,0,0,1,0,0,1]
```

Chaque 1 représente une position où on descend, chaque 0 une position où on se déplace vers la droite.

Exercice 73 : Bataille navale

C'est l'exercice 8-3 du cours. Voici quelques précisions :

- (1) Chaque joueur (1 = humain ou 2 = ordinateur) possède 5 bateaux nommés 'A', 'B', 'C', 'D', et 'E', de tailles respectives 5, 4, 3, 3 et 2. On pourra par exemple utiliser le dictionnaire

```
SHIPS = { 'A' :5, 'B' :4, 'C' :3, 'D' :3, 'E' :2 }
```

pour stocker les longueurs de ces bateaux.

- (2) Pour représenter les mers adverses, on utilisera les dictionnaires :

opp_sea et my_sea. Les clés de ces dictionnaires seront des tuples (x,y) où x est l'abscisse et y l'ordonnée d'une case avec $0 \leq x \leq 9$ et $0 \leq y \leq 9$. La valeur associée à un tuple (x,y) est :

- '.' si la case n'a pas encore été attaquée. C'est la valeur à affecter à chaque tuple (x,y) lors de l'initialisation ;
 - la lettre du navire, si celui-ci occupe cette case et que cette case n'a pas encore été touchée ;
 - '~' si un joueur y a tiré mais raté ;
 - '#' si un joueur y a tiré et touché un navire ;
- (3) Voici l'écran de départ du jeu. Le joueur 1 est invité à entrer la position de ses navires. Pour simplifier, on autorise le joueur à placer ses navires comme il veut, avec la seule contrainte que les navires ne doivent pas se croiser !

B A T A I L L E N A V A L E
=====

	Opponent sea:	My sea:
	A B C D E F G H I J	A B C D E F G H I J
1:
2:
3:
4:
5:
6:
7:
8:
9:
10:

Entrez la position du navire A: A4H

	Opponent sea:	My sea:
	A B C D E F G H I J	A B C D E F G H I J
1:
2:
3:
4:	A A A A A
5:
6:
7:
8:
9:
10:

Le navire est donc placé horizontalement en A4. L'ordinateur redemande la position d'un navire aussi longtemps que le joueur entre une position impossible. Lorsque le joueur entre une position impossible, le message 'Impossible' est affiché. Par exemple :

Entrez la position du navire B: H1H
Impossible !
Entrez la position du navire B: H3V

Opponent sea:											My sea:										
A	B	C	D	E	F	G	H	I	J	A	B	C	D	E	F	G	H	I	J		
1:		
2:		
3:	B	.	.		
4:	A	A	A	A	A		
5:	B	.		
6:	B	.		
7:		
8:		
9:		
10:		

Lorsque tous les navires du joueur 1 (humain) sont positionnés, c'est le tour du joueur 2 (ordinateur = opposent). L'ordinateur place ses navires au hasard. Evidemment le programme ne doit pas montrer la position de ces navires !

- (4) Les joueurs tirent alternativement (un coup à la fois), le joueur 1 commençant et l'ordinateur plaçant ses tirs totalement au hasard. Si un joueur tire plusieurs fois au même endroit, il peut répéter son tir jusqu'à ce qu'il ait entré une position valide.

Entrez les coordonnées pour votre tir : A1
Raté !
Coordonnées du tir de l'ordinateur : D3
Raté !

Opponent sea:											My sea:										
A	B	C	D	E	F	G	H	I	J	A	B	C	D	E	F	G	H	I	J		
1:	~		
2:		
3:	~	.	.	B	.	.		
4:	A	A	A	A	A	.		
5:	B	.		
6:	B	.		
7:	C	C	C	.	.		
8:	D	D		
9:		
10:	E	E		

Après quelques tirs supplémentaires, le joueur 1 arrive à couler un navire de l'ordinateur. Il en est alors informé par le message supplémentaire 'Coulé !':

Entrez les coordonnées pour votre tir : E4
Touché !
Coordonnées du tir de l'ordinateur : E10
Raté !

Opponent sea:											My sea:										
A	B	C	D	E	F	G	H	I	J	A	B	C	D	E	F	G	H	I	J		
1:	~	.	~	~	.	~		
2:	.	~		
3:	~	.	~	~	~	.	~	.	#	.	~		
4:	.	~	.	#	#	A	A	A	A	A	.	.	B	.		
5:	~	.	~	~	~	.	.	B	.	~		
6:	.	~	~	B	.	.		

```

7: ~ . ~ . . . . . . . . . . ~ C C C . . . ~ .
8: . ~ . . . . . . . . . . . ~ . . . D D D ~
9: ~ . # # . . . . . . . . . . ~ . . . ~ ~ . . . .
10: . ~ . . . . . . . . . . E E . ~ ~ . . ~ . ~

```

Entrez les coordonnées pour votre tir : F4
 Touché !
 Coordonnées du tir de l'ordinateur : E2
 Raté !

	Opponent sea:	My sea:
	A B C D E F G H I J	A B C D E F G H I J
1:	~ . ~	~ . ~
2:	. ~ ~
3:	~ . ~ ~ ~ . ~ . # . ~
4:	. ~ . # # #	A A A A A . . B . .
5:	~ . ~ ~ ~ . . B . ~
6:	. ~	~ B . .
7:	~ . ~ ~ C C C . . . ~ .
8:	. ~ ~ . . . D D D ~
9:	~ . # # ~ . . ~ ~
10:	. ~	E E . ~ ~ . . ~ . ~

Entrez les coordonnées pour votre tir : G4
 Touché !
 Coulé !
 Coordonnées du tir de l'ordinateur : G3
 Raté !

	Opponent sea:	My sea:
	A B C D E F G H I J	A B C D E F G H I J
1:	~ . ~	~ . ~
2:	. ~ ~
3:	~ . ~ ~ ~ . ~ ~ # . ~
4:	. ~ . # # # #	A A A A A . . B . .
5:	~ . ~ ~ ~ . . B . ~
6:	. ~	~ B . .
7:	~ . ~ ~ C C C . . . ~ .
8:	. ~ ~ . . . D D D ~
9:	~ . # # ~ . . ~ ~
10:	. ~	E E . ~ ~ . . ~ . ~

A la fin de la partie le vainqueur est affiché. Lorsque le joueur humain a perdu, le programme doit lui montrer la mer adverse sans cacher cette fois les navires de l'ordinateur qui n'auraient pas encore été coulés ou touchés.

Entrez les coordonnées pour votre tir : I9
 Touché !
 Coulé !

	Opponent sea:										My sea:									
	A	B	C	D	E	F	G	H	I	J	A	B	C	D	E	F	G	H	I	J
1:	~	.	~	.	~	.	.	~	.	.	~	~	~	~	~
2:	.	~	.	~	.	.	~	.	.	.	~	.	~	.	~	.	.	~	.	~
3:	~	.	~	~	~	.	.	.	~	.	~	.	.	~	.	~	~	#	~	~
4:	.	~	~	#	#	#	#	~	.	.	A	A	#	A	#	~	.	B	~	~
5:	~	.	~	~	~	.	.	~	#	.	~	.	.	.	~	~	.	#	~	~
6:	.	~	.	.	~	.	.	.	#	.	~	.	.	C	C	.	.	B	~	.
7:	~	.	~	~	#	#	#	~	#	.	~	~	C	C	C	.	.	.	~	.
8:	.	~	.	#	#	#	~	~	#	.	.	.	~	~	.	~	D	D	D	~
9:	~	.	#	#	~	.	.	.	#	.	~	~	~	.	~	~	~	.	.	.
10:	.	~	.	.	.	~	E	#	.	~	~	.	.	~	.	~

Victoire pour le joueur 1

(5) **Programmation :**

- a) Ecrire la fonction

```
def print_seas(show=False):
```

qui affiche sur l'écran les deux mers adverses. Le paramètre `show` dont la valeur par défaut est `False` permet de contrôler si on veut **montrer** ou **cacher** les navires de l'ordinateur.

- b) Ecrire la fonction :

```
def check_position(symbol, x, y, orientation, sea):
```

qui vérifie s'il est possible de placer le navire dont le symbole est `symbol` en `(x,y)` dans la mer `sea` avec l'orientation donnée (`'H'` ou `'V'`). Si c'est possible cette fonction retourne `True`, sinon elle retourne `False`.

- c) Ecrire la fonction :

```
def set_ship(symbol, x, y, orientation, sea):
```

qui positionne le symbole du navire (`'A'`, `'B'`, ...) en `(x,y)` de la mer `sea` avec l'orientation donnée (`'H'` ou `'V'`) à condition que c'est possible. Sinon la fonction ne fait rien.

- d) Ecrire la fonction :

```
def set_ships(player):
```

qui permet au joueur `player` (1 ou 2) de positionner tous ses navires dans sa mer. Lorsque le joueur est 2 (ordinateur), la position des navires est laissée au hasard ! Lorsque le joueur est 1, les deux mers sont affichées après chaque positionnement d'un navire.

e) Ecrire la fonction :

```
def is_destroyed(ship, sea):
```

qui retourne True si le navire avec le symbole ship n'existe plus dans la mer sea.

f) Ecrire la fonction :

```
def fire(player):
```

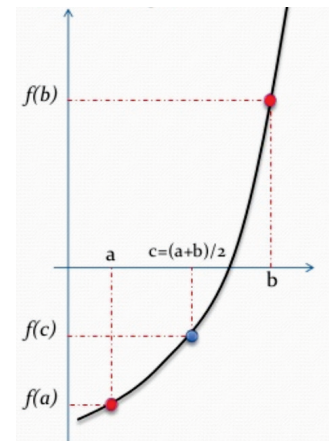
qui permet au joueur player de tirer. La mer correspondante doit être actualisée et le joueur obtient le message correct. L'ordinateur ne tire jamais au même endroit. Si le joueur essaie de tirer au même endroit, il en est averti et il peut rejouer.

g) Ecrire le programme principal du jeu. Inventer un mécanisme simple pour décider qui a gagné !

Exercice 74 : Méthode dichotomique

Ecrire un programme qui *recherche une racine* d'un polynôme par la *méthode dichotomique* :

- L'utilisateur entre deux réels $a < b$ tels que $p(a) \cdot p(b) < 0$, c.-à-d. tels que $p(a)$ et $p(b)$ soient de signes opposés. (Le programme devra vérifier si cette condition est bien remplie et afficher au cas contraire un message d'erreur).
- Le programme calcule $p\left(\frac{a+b}{2}\right)$: il y a alors 3 possibilités :
 - ou bien $p\left(\frac{a+b}{2}\right) = 0$, alors $\frac{a+b}{2}$ est une racine de p et le problème est résolu ;
 - ou bien $p\left(\frac{a+b}{2}\right) \cdot p(a) > 0$, alors le programme posera $a := \frac{a+b}{2}$ car il y a nécessairement une racine entre $\frac{a+b}{2}$ et b ;
 - ou bien $p\left(\frac{a+b}{2}\right) \cdot p(b) > 0$ alors le programme posera $b := \frac{a+b}{2}$ car il y a nécessairement une racine entre a et $\frac{a+b}{2}$;

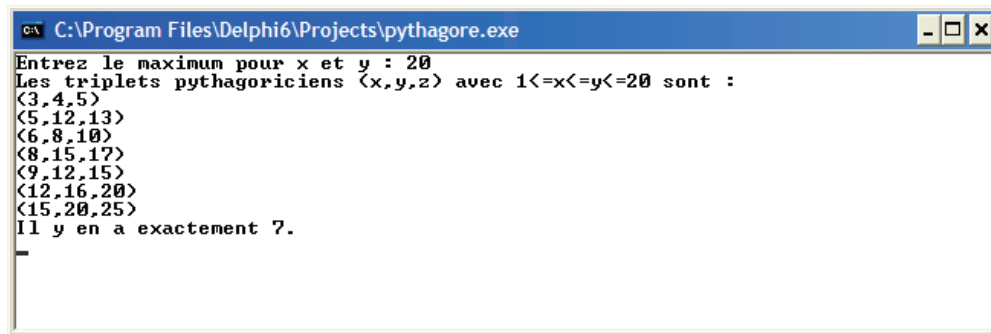


On répète cet algorithme jusqu'à ce que $b - a < \varepsilon$, où ε est l'erreur tolérée sur la racine (précision), fixée au préalable par l'utilisateur.

Exercice 75

Ecrire le programme qui recherche et compte tous les triplets pythagoriciens (x,y,z) tels que $1 \leq x \leq y \leq \max$, où \max est un entier entré par l'utilisateur. On rappelle qu'un triplet pythagoricien est un triplet d'entiers naturels (x,y,z) tels que $x^2 + y^2 = z^2$.

Exemple d'exécution :



Exercice 76

- (1) Ecrire un programme qui permet de simuler le tirage au sort de k boules dans une urne contenant a boules noires et b boules blanches. On distinguera les deux cas : tirages avec ou sans remise.
- (2) Ecrire un programme qui permet de simuler le tirage au sort d'une main de k cartes (sans remise) dans un jeu contenant 52 cartes.

Exercice 77 (Serpent vietnamien)

Voici un problème de maths repéré par le Guardian donné à des enfants de 8 ans au Vietnam.

Les 9 cases sont à remplir avec des chiffres de 1 à 9 (qu'il ne faut utiliser qu'une fois chacun). En ajoutant, multipliant, soustrayant et divisant au fur et à mesure (en suivant l'ordre des opérations – multiplications et divisions en priorité), on doit arriver à 66.

				66
+		×	−	=
13		12	11	10
×		+	+	−
:		+	×	:

Ecrire un programme qui trouve et affiche toutes les solutions. Essayer de simplifier l'équation et de limiter la recherche en éliminant certaines possibilités dès le départ !