Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

1. Algorithmes et premières instructions (exercices de TD)

Exercice 1.1

1. Indiquer précisément les actions engendrées par les instructions Python suivantes :

```
x = 3.5
print(x)
print("x")
print("x=", x)
v = float(input("Saisir au clavier une valeur réelle"))
print("x=", v)
```

2. Combien vaut la variable `x` à la fin de l'exécution du programme ?

Exercice 1.2

Écrire un algorithme en pseudo-code permettant de lire une valeur entière, d'incrémenter cette valeur de 1, puis de l'afficher.

Exercice 1.3

Écrire un algorithme en pseudo-code permettant de calculer l'image d'un nombre x (saisi au clavier par l'utilisateur) par la fonction

$$f(x) = 3x + 1$$

et d'afficher le résultat.

Exercice 1.4

Écrire un algorithme en pseudo-code permettant de lire deux nombres et de les stocker dans deux variables, puis d'échanger les valeurs de ces deux variables.

Exercice 1.5

Traduire en expressions booléennes les phrases suivantes :

1. « x est compris au sens large entre 4,7 et 8. »
   Donner la table de vérité du ET.
2. « y est divisible par 2 ou z n'est pas divisible par 3 (OU inclusif). »
   Donner la table de vérité du OU.
3. « Une et une seule des conditions suivantes est vraie : 1) u n'est pas nul, 2) v a la même valeur que u. »
   A cet effet, l'opérateur OU eXclusif, connu sous le nom XOR, renvoie, étant donnés deux opérandes, le résultat VRAI si et seulement si les deux opérandes ont des valeurs distinctes.
   (a) Donner la table de vérité de A XOR B.
   (b) Les opérateurs OU et ET sont traduits en Python respectivement par `or` et `and`. Par contre l'opérateur XOR n'est pas prédéfini en Python. Comment peut-on modéliser A XOR B à l'aide uniquement des opérateurs NON, ET et OU ? Le vérifier en établissant les tables de vérité respectives.

Exercice 1.6

On donne le programme suivant :

```
1. a = float(input())
2. b = int(input())
3. if (a > b): b = b+1
4. if (b < 4):
5.     if (a < 0):
           a = a-b
       else:
           b += int(a)
       print("moins de 4")
   else:
       print("plus de 4")
       b -= 4
6. print("a=", a)
7. print("b=", b)
```

Donner l'affichage produit par le programme précédent si l'utilisateur tape :

1. 2.3 puis -2
2. 4 puis 3
3. -1 puis 1

Exercice 1.7

Écrire en pseudo-code un algorithme qui lit trois nombres a, b et c, résout l'équation $ax^2 + bx + c = 0$ et affiche les solutions réelles (on distinguera 3 cas : $\Delta < 0$, $\Delta = 0$ et $\Delta > 0$).

Exercice 1.8

Écrire un algorithme en pseudo-code qui lit un entier n et affiche "Ajourné" si $n < 10$, "Passable" si $10 \leq n < 12$, "AB" si $12 \leq n < 14$, "B" si $14 \leq n < 16$ et "TB" si $n \geq 16$.

Exercice 1.9

Dans le programme Python ci-dessous, pour quelles valeurs des différents paramètres verra-t-on s'afficher chacun des messages ?

```
if (niveau <= 2):
    if not(section == 1):
        if (discipline > 1):
            print("Lycéen en section générale non littéraire")
        elif (discipline == 1):
            print("Lycéen en section générale littéraire")
    else: print("Lycéen qui n'est pas en section générale")
elif (niveau <= 6):
    print("c'est un collégien")
else: print("il est trop petit")
```

Exercice 1.10

Écrire en pseudo-code un algorithme qui lit 3 nombres et affiche "oui" si l'un des 3 est égal à la somme des 2 autres et "non" sinon.

Exercice 1.11

Écrire en pseudo-code, un algorithme qui lit 6 nombres a, b, c, d, e et f et affiche la résolution du système d'équations :

$$\begin{aligned} ax + by &= c \\ dx + ey &= f \end{aligned}$$

Quelles sont les différentes issues de cette résolution ?

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

1. Algorithmes et premières instructions (exercices de TP)

But des TP : mettre en pratique les différentes notions algorithmiques vues en cours en écrivant de petits programmes qui vont s'exécuter sur les machines.

Pour chaque exercice donnant lieu à l'écriture d'un programme, ce dernier devra être tapé et sauvegardé sur le disque de manière indépendante des autres afin d'être conservé. Il faut donc bien penser à ouvrir un nouveau fichier à chaque début d'exercice.

Exercice 1.12

— `int(X)` permet de transformer une valeur réelle (de type `float`) ou une chaîne (représentant un entier) en une valeur de type entier.
— `float(X)` permet de transformer une valeur entière ou une chaîne en valeur réelle de type `float`.
— `str(X)` permet de transformer une valeur numérique en chaîne de caractères.

Tester les commandes suivantes et interpréter les résultats :

```
type(3)
type(3.14)
type((3,14))
type("3")
type("3.14")
type(3+4)
type(3*4.0)
6/4
8/4
6//4
8//4
6.0//4
2+"3"
2+int(3.5)
2+int("3")
2+int("3.5")
2+float("3")
2+float("3.5")
str(2)+"3"
str(2.5)+"3.5"
```

Exercice 1.13

Écrire un programme en Python qui demande à l'utilisateur d'entrer une valeur pour une variable x, puis :

1. en utilisant une unique instruction `print()` affiche les valeurs des éléments suivants :

   ```
   x
   2*x
   3*x
   FIN
   ```

2. en utilisant 4 fois l'instruction `print()` affiche les éléments suivants (en remplaçant les '`?`' par la valeur correspondante) :

   ```
   x = ?, 2*x = ?, 3*x = ?, FIN
   ```

Exercice 1.14

Nous vous rappelons qu'une année est dite bissextile :

— si l'année est divisible par 4 et non divisible par 100 ;

— ou si l'année est divisible par 400.

Écrire un programme en Python qui demande à l'utilisateur de saisir une année (on supposera que l'utilisateur saisira correctement l'année) et affichera si l'année en question est bissextile ou pas.

Exercice 1.15

Écrire un programme en Python qui demande à l'utilisateur d'entrer son nom, son prénom, son sexe, sa ville de naissance et son département de naissance, puis affiche à l'écran les informations saisies au clavier par l'utilisateur.

Si l'utilisateur saisi 'Lagaffe', 'Gaston', 'masculin', 'Angoulême' puis '16' alors les informations doivent être affichées à l'écran sous le format suivant :

```
Vous vous appelez Gaston Lagaffe
Vous etes un homme né a Angouleme (16)
```

Si l'utilisateur saisi 'Blanc-Sec', 'Adèle', 'Feminin', 'Paris' puis '75' alors les informations doivent être affichées à l'écran sous le format suivant :

```
Vous vous appelez Adele Blanc-Sec
Vous etes une femme nee a Paris (75)
```

Modifier votre programme pour qu'il teste la validité du numéro de département : les numéros de département vont de 1 à 95, de 971 à 974 et 976. Ainsi si le numéro de département saisi est 203 alors il faut afficher le message suivant :

```
Votre numero de département est errone !
```

Exercice 1.16

Écrire un programme, appelé `image_f.py`, permettant de lire une valeur saisie au clavier par un utilisateur, de calculer et d'afficher l'image de ce nombre x par la fonction

$$f(x) = \frac{1}{3}x + 1$$

Vous prendrez soin d'arrondir le résultat au dixième près à l'aide de la fonction python `round()`. Par exemple, `round(5.6757,1)` renvoie la valeur `5.6`.

Exercice 1.17

Étant donnés la taille (en cm) et l'âge d'un individu, on donne la formule suivante du poids idéal : $0,9(taille - 100 + age/10)$. Écrire et tester un programme en Python qui lit la taille et l'âge d'un utilisateur et affiche son poids idéal. Ce programme doit ensuite lire le poids actuel de l'utilisateur et afficher le nombre de kilos à prendre ou à perdre pour atteindre ce poids idéal. On distinguera trois résultats possibles :

```
Vous devez perdre x kilos.
Votre poids est parfait.
Vous devez prendre x kilos.
```

Exercice 1.18 *(Mise en œuvre de l'exercice 1.6)*

Créer un fichier `ex5.py` contenant le programme suivant :

```
a = float(input())
b = int(input())
if (a > b): b=b+1
if (b < 4):
    if (a < 0):
        a = a-b
    else:
        b += int(a)
    print("moins de 4")
else:
    print("plus de 4")
    b -= 4
print("a=",a)
print("b=",b)
```

Exécuter votre programme pour différentes valeurs saisies par l'utilisateur et analyser les différentes exécutions :

a) 2.3 puis -2

b) 4 puis 3

c) -1 puis 1

Exercice 1.19 *(Mise en œuvre de l'exercice 1.7)*

Écrire un programme en Python qui lit trois nombres réels a, b et c et affiche la résolution de l'équation $ax^2 + bx + c = 0$. Ici, on affichera les solutions réelles ou complexes.
Afin de pouvoir calculer la racine carrée, on ajoutera la ligne

```
from math import sqrt
```

en tête de fichier et on utilisera ensuite la fonction `sqrt()` *(square root)*.

Tester votre programme en vous assurant que vous obtenez bien les solutions suivantes (si tel n'est pas le cas, corrigez votre programme !) :

Test n°	a	b	c	Solutions
1	1	1	-2	$x_1 = -2, x_2 = 1$
2	4	4	1	racine double $x = -0.5$
3	-1	2	-3	$x_1 = 1 + i * 1.41$; $x_2 = 1 - i * 1.41$
4	0	1	-2	$x = 2$
5	0	0	-2	pas de solution
6	0	0	0	infinité de solution

Exercice 1.20 *(Mise en œuvre de l'exercice 1.8)*

Écrire un programme en Python qui saisit un entier n et affiche "Ajourné" si $n < 10$, "Passable" si $10 \leq n < 12$, "AB" si $12 \leq n < 14$, "B" si $14 \leq n < 16$ et "TB" si $n \geq 16$.

Exercice 1.21

On souhaite caractériser des triangles.
Écrire un programme **Python** qui :

1. demande à l'utilisateur de saisir trois longueurs réelles a, b, c ;
2. détermine s'il est possible de construire un triangle de côtés a, b et c (pour cela, la somme des deux côtés les plus courts doit toujours être supérieure ou égale au côté le plus long) ;
3. et si c'est le cas, détermine si ce triangle est rectangle (le carré d'un des côtés doit être égal à la somme des carrés des deux autres), équilatéral (les trois côtés sont égaux) ou ni l'un ni l'autre (on dira alors qu'il est quelconque) ;
4. et termine en affichant les résultats (affiche s'il est possible ou non de construire un tel triangle, ses propriétés ou s'il est quelconque).

Exercice 1.22 *(Mise en œuvre de l'exercice 1.10)*

Écrire un programme en Python qui lit 3 nombres et affiche "oui" si l'un des 3 est égal à la somme des 2 autres et "non" sinon.

Exercice 1.23 *(Mise en œuvre de l'exercice 1.11)*

Écrire en Python, un programme qui saisit 6 nombres a, b, c, d, e et f et affiche la résolution du système d'équations :

$$\begin{aligned} ax + by &= c \\ dx + ey &= f \end{aligned}$$

Tester votre programme en vous assurant que vous obtenez bien les solutions suivantes (si tel n'est pas le cas, corriger votre programme !) :

Tests	a	b	c	d	e	f	Solutions
1	1	2	1	2	1	2	$x = 1, y = 0$
2	1	2	1	0	1	2	$x = -3, y = 2$
3	1	2	1	1	0	2	$x = 2, y = -0.5$
4	1	0	-1	2	1	2	$x = -1, y = 4$
5	2	1	1	4	2	6	pas de solution
6	1	0	-1	1	0	1	pas de solution
7	-3	1	2	12	-4	-8	infinité de solutions

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

2. Boucle répétitive (exercices de TD)

Exercice 2.1

Écrire un algorithme en pseudo-code qui lit un réel $x \geq 0$ et un entier $n \geq 0$, et affiche la valeur de x^n (sans utiliser l'opérateur puissance).

Exercice 2.2

Écrire un algorithme en pseudo-code qui affiche la somme de tous les entiers impairs positifs, inférieurs à 100.

Exercice 2.3

Écrire un algorithme en pseudo-code qui lit un entier (supposé strictement positif), affiche la somme de tous ses diviseurs stricts (c'est-à-dire différents du nombre) et précise si ce nombre est premier (c'est-à-dire qu'il n'a que 1 comme diviseur strict). Par exemple, si l'on saisit 8, il affiche 7 (car somme de ses diviseurs stricts qui sont 1, 2 et 4).

Exercice 2.4

Un entier est dit parfait s'il est égal à la somme de ses diviseurs stricts (6 par exemple est parfait). Écrire un algorithme qui lit un entier n (supposé strictement positif) et affiche tous les nombres parfaits inférieurs ou égaux à n.

Exercice 2.5

Soit le programme Python suivant :

```
i = int(input("debut : "))
j = int(input("fin : "))
while (i != j):
    if (i%7 == 0):
        print(i, " est multiple de 7")
    i += 1
```

1. Que fait ce programme dans le cas général ?
2. Que se passe-t-il si on entre une valeur de `i` strictement plus grande que `j` ? Comment pourrait-on améliorer ce programme ?
3. Donner l'algorithme associé en pseudo-code.

Exercice 2.6

À l'aide de 2 boucles `TANT_QUE` emboîtées, écrire un algorithme en pseudo-code qui affiche :

```
1   2   3   4   5
1   3   5   7   9
1   4   7   10  13
1   5   9   13  17
```

Exercice 2.7

Écrire en pseudo-code un algorithme qui lit un entier strictement positif n, puis n entiers, et qui vérifie si la suite des n entiers saisis est triée de façon croissante.

Par exemple, si la valeur saisie pour n est 7 et les 7 valeurs saisies sont : -1, 3, 7, 8, 11, 234, 300, le programme affichera "les 7 valeurs sont triées de façon croissante", alors que le programme affichera "les 7 valeurs ne sont pas triées" si les entiers saisis sont : 1, 2, 7, -2, 4, 5, 200.

Exercice 2.8

Écrire en pseudo-code un algorithme qui lit un entier strictement positif n, puis n entiers strictement positifs, et qui vérifie si parmi les n entiers saisis, la somme des entiers pairs est égale à la somme des entiers impairs. Le programme doit refuser les valeurs négatives ou nulles.

Par exemple, si la valeur saisie pour n est 5 et les 5 valeurs saisies sont 2, 3, 2, 3, 2, l'algorithme affichera "la somme des nombres pairs est égale à la somme des nombre impairs", alors que l'algorithme affichera "la somme des nombres pairs n'est pas égale à la somme des nombres impairs" si les valeurs saisies sont 2, 3, 4, 3, 1. Si la valeur saisie pour n (ou une des n valeurs) est par exemple -2, l'algorithme demandera à nouveau d'insérer une valeur, jusqu'à ce que celle-ci soit strictement positive.

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

2. Boucle répétitive while (TP)

Exercice 2.9 *(Debugage d'une solution de l'exercice 1.7)*

L'équipe Python a écrit une solution pour l'exercice 1.7 sur l'équation du second degré. Malheureusement le programme ne fonctionne pas ! Vous allez devoir le corriger et le faire fonctionner.

Pour récupérer le fichier, aller sur le répertoire `ETUD/DEMI2E_L1_AlgoProg/` et copier, dans votre répertoire personnel, le fichier `Debug_Equation2ndDegre.py`.

1. A quoi servent les lignes commençant par # ? Vous aident-elles à comprendre ce que fait ce programme ?
2. Exécuter le programme ci-dessus tel quel. Quel est le message d'erreur ? Que faut-il corriger dans le programme pour résoudre le problème ?
3. L'instruction `print("la valeur de delta est",d)` est-elle nécessaire dans le programme ? Si non, pourquoi l'ajouter ?
4. Exécuter le programme en donnant 2 pour a, -8 pour b et 6 pour c. Quel est le message d'erreur ? Que faut-il corriger dans le programme pour lever ce message ?
5. Exécuter le programme en donnant 2 pour a, -8 pour b et 6 pour c. Le programme s'exécute-t-il ? Comment s'assurer que la solution proposée est la bonne ? Si ce n'est pas la bonne que faut-il corriger ? Où dans le programme ? Faire les corrections correspondantes.
6. Exécuter le programme en donnant 2 pour a, 4 pour b et 4 pour c. Quel est le message d'erreur ? Que faut-il corriger dans le programme pour résoudre le problème ?
7. Le cas où l'équation admet une unique solution réelle a-t-il été testé ? Proposer un jeu de valeur permettant de l'exécuter.
8. Exécuter le programme en donnant 0 pour a, 2 pour b et 6 pour c. Quel est le message d'erreur ? Que faut-il corriger dans le programme pour résoudre le problème ?

Exercice 2.10 *(Mise en œuvre de l'exercice 2.1)*

Écrire un programme Python qui lit un réel $x \geq 0$ et un entier $n \geq 0$, et affiche la valeur de x^n (sans utiliser l'opérateur puissance).

Exercice 2.11 *(Mise en œuvre de l'exercice 2.2)*

Écrire un programme Python qui affiche la somme de tous les entiers impairs positifs, inférieurs à 100.

Exercice 2.12 *(Mise en œuvre de l'exercice 2.3)*

Écrire un programme Python qui lit un entier (supposé strictement positif), affiche la somme de tous ses diviseurs stricts (c'est-à-dire différents du nombre) et précise si ce nombre est premier (c'est-à-dire qu'il n'a que 1 comme diviseur strict). Par exemple, si l'on saisit 8, il affiche 7 (car somme de ses diviseurs stricts qui sont 1, 2 et 4).

Exercice 2.13 *(Mise en œuvre de l'exercice 2.4)*

Un entier est dit parfait s'il est égal à la somme de ses diviseurs stricts (6 par exemple est parfait). Écrire un programme Python qui lit un entier n et affiche tous les nombres parfaits inférieurs ou égaux à n.

Exercice 2.14

Le Plus Petit Commun Multiple (PPCM) de deux entiers positifs a et b est le plus petit entier p tel que p soit à la fois multiple de a et de b.

Afin de calculer le PPCM de a et b, on applique l'algorithme suivant qui consiste à calculer les multiples successifs de a et b, notés m_a et m_b, jusqu'à arriver à l'obtention du PPCM.

(i) initialiser m_a et m_b ;

(ii) tant que le PPCM n'est pas trouvé, on augmente la plus petite des deux valeurs m_a et m_b ;

(iii) afficher le PPCM.

1. Comment initialiser m_a et m_b ?
2. Comment identifier que le PPCM est trouvé ?
3. De combien doit-on augmenter à chaque itération m_a ? et m_b ?
4. Écrire en Python le programme complet : il doit lire deux nombres saisis par l'utilisateur (on les supposera entiers et positifs), calculer leur PPCM et l'afficher.

Exercice 2.15

Étant donné un livret bancaire rémunéré au taux annuel de 1,5 %, sur lequel 6 000 euros auront été déposés le 1er janvier 2017, écrire un programme en Python permettant de savoir au bout de combien d'années la somme disponible sur ce livret aura dépassé 7 000 euros.

Nous vous rappelons que si l'on place une somme s le 1er janvier d'une année n sur un livret bancaire rémunéré au taux annuel de 1,5 %, la somme disponible le 1er janvier de l'année $(n+1)$ sera $s + s * 0,015$. *(La fonction logarithme n'est pas utilisable.)*

Exercice 2.16 *(Mise en œuvre de l'exercice 2.6)*

A l'aide de 2 boucles `while` emboîtées, écrire un programme Python qui affiche (la gestion précise de l'espacement n'est pas demandée) :

```
1   2   3   4   5
1   3   5   7   9
1   4   7   10  13
1   5   9   13  17
```

Exercice 2.17 *(Mise en œuvre de l'exercice 2.7)*

Écrire un programme Python qui lit un entier positif n, puis n entiers, et qui vérifie si la suite des n entiers saisis est triée de façon croissante.

Par exemple, si la valeur saisie pour n est 7 et les 7 valeurs saisies sont : -1, 3, 7, 8, 11, 234, 300, le programme affichera "les 7 valeurs sont triées de façon croissante", alors que le programme affichera "les 7 valeurs ne sont pas triées" si les entiers saisis sont : 1, 2, 7, -2, 4, 5, 200.

Exercice 2.18 *(Mise en œuvre de l'exercice 2.8)*

Écrire un programme Python qui lit un entier positif n, puis n entiers positifs, et qui vérifie si parmi les n entiers saisis la somme des entiers pairs est égale à la somme des entiers impairs. Le programme doit refuser les valeurs nulles ou négatives.

Par exemple, si la valeur saisie pour n est 5 et les 5 valeurs saisies sont 2, 3, 2, 3, 2, le programme affichera "la somme des nombres pairs est égale à la somme des nombre impairs", alors que le programme affichera "la somme des nombres pairs n'est pas égale à la somme des nombres impair" si les valeurs saisies sont 2, 3, 4, 3, 1. Si la valeur saisie pour n (ou une des n valeurs) est par exemple -2, le programme demandera à nouveau d'insérer une valeur, jusqu'à ce que celle-ci soit positive.

Exercice 2.19

On joue à lancer 3 dés : un dé rouge à 6 faces (numérotées de 1 à 6), un dé vert à 8 faces (numérotées de 1 à 8) et un dé bleu à 10 faces (numérotées de 1 à 10). On veut déterminer combien il y a de façons d'obtenir un nombre n donné en additionnant les trois faces des dés. Par exemple, il y a une seule façon d'obtenir 3 qui est de faire 1 avec le dé vert, 1 avec le dé rouge et 1 avec le dé bleu. Il y a 3 façons d'obtenir 4 :

— 1 avec le dé vert, 1 avec le dé rouge et 2 avec le dé bleu, ou bien,
— 1 avec le dé vert, 2 avec le dé rouge et 1 avec le dé bleu, ou bien,
— 2 avec le dé vert, 1 avec le dé rouge et 1 avec le dé bleu.

Écrire le programme **Python** qui implémente l'algorithme suivant :

1. demander à l'utilisateur de saisir une valeur entière ;
2. à l'aide de boucles imbriquées, compter le nombre de possibilités permettant d'obtenir ce nombre comme somme des dés rouge, vert et bleu ;
3. afficher le résultat.

Exemple d'exécution :

affichage Entrez un nombre entier :

saisie 5

affichage Il y a 6 façon(s) de faire 5 avec ces trois dés.

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

3. Fonctions (exercices de TD)

Exercice 3.1

Écrire en pseudo-code une fonction `maximum()` prenant 2 arguments de type NOMBRE et renvoyant le maximum des deux. Écrire un algorithme qui appelle cette fonction pour afficher le maximum de deux valeurs saisies par l'utilisateur.

Exercice 3.2

Écrire en pseudo-code :

1. une fonction `cube()` qui calcule le cube d'un entier n passé en argument ;
2. une fonction `volume()` qui calcule le volume d'une sphère de rayon r passé en argument en appelant la fonction `cube()` précédente. Pour rappel : le volume d'une sphère se calcule par la formule : $V = (4/3) \times \pi \times r^3$;
3. écrire un algorithme qui permet de tester la fonction précédente en affichant le volume d'une sphère dont le rayon est saisi par l'utilisateur.

Exercice 3.3

Écrire en pseudo-code une fonction `tableMulti()` permettant d'afficher une table de multiplication avec 3 arguments : `base`, `debut` et `fin`, la fonction affichant le résultat de la multiplication de `base` par tous les entiers situés entre `debut` et `fin` inclus. Par exemple l'appel de `tableMulti(8, 13, 17)` devra afficher :

```
Fragment de la table de multiplication par 8 :
13 x 8 = 104
14 x 8 = 112
15 x 8 = 120
16 x 8 = 128
17 x 8 = 136
```

Écrire un algorithme utilisant la fonction précédente pour afficher une table de multiplication dont les paramètres auront été donnés par l'utilisateur. [1]

Exercice 3.4

Écrire 3 fonctions qui permettent de calculer le Plus Grand Commun Diviseur (PGCD) de deux nombres a et b (avec $a \geq b$) et l'algorithme permettant de les tester. [2]

1. repris de `http://python.developpez.com/cours/TutoSwinnen/?page=page_9`
2. inspiré de `http://www.mathematiquesfaciles.com/calculer-le-pgcd-d-un-nombre-avec-cours_2_75657.htm`

1. La première fonction `pgcdParDiviseurs(a,b)`, avec $a \geq b$, calculera les diviseurs communs à chacun des deux nombres et renverra le plus grand diviseur commun (on pourra ne calculer que les diviseurs nécessaires).

   Par exemple, si on appelle `pgcdParDiviseurs(63,42)`, les diviseurs de `63` étant `1;3;7;9;21;63` et ceux de `42` étant `1;2;3;6;7;14;21;42`, on doit pouvoir afficher, à l'aide de la valeur renvoyée :

   ```
   Le plus grand diviseur commun de 63 et 42 est : 21
   ```

2. La deuxième fonction `pgcdParDifferences(a,b)`, avec $a \geq b$, utilisera l'algorithme des différences pour renvoyer le PGCD. Si un nombre est un diviseur de 2 autres nombres a et b, alors il est aussi un diviseur de leur différence $a - b$. Le PGCD de a et b est donc aussi celui de b et $a - b$.

   Par exemple, calculons le PGCD de `60` et `36`.

   `60 - 36 = 24`, donc le PGCD de `60` et `36` est un diviseur de `24`. On recommence en effectuant une nouvelle soustraction entre le résultat obtenu à la soustraction précédente (24) et le plus petit des deux termes de la soustraction précédente (36) : on obtient `36 - 24 = 12`. Par conséquent, le PGCD de `60` et `36` est un diviseur de `12` et on peut ré-appliquer la soustraction. On arrête l'algorithme dès que la soustraction donne un résultat nul. Le PGCD correspond au résultat obtenu à l'étape précédente. Voici l'illustration des étapes :

   ```
   60 - 36 = 24
   36 - 24 = 12
   24 - 12 = 12
   12 - 12 = 0
   ```

   Ainsi, l'appel de `pgcdParDifferences(60,36)` doit renvoyer 12.

3. La troisième fonction `pgcdParEuclide(a,b)`, avec $a \geq b$, utilisera l'algorithme d'Euclide pour renvoyer le PGCD. L'algorithme d'Euclide pour calculer le PGCD de deux entiers a et b (avec $a \geq b$) consiste à effectuer une suite de divisions euclidiennes :
   — étape 1 : effectuer la division euclidienne de a par b ; on obtient r le reste ; si r est égal à 0, alors fin et le PGCD est égal à b, sinon aller à l'étape 2 ;
   — étape 2 : a reçoit alors la valeur de b, b reçoit la valeur de r et aller à l'étape 1.

   Cette méthode en général est plus rapide.

   Voici l'illustration des étapes :

   ```
   561 divisé par 357 donne 1 en quotient et 204 en reste
   donc 561 = 357 x 1 + 204
   357 divisé par 204 donne 1 en quotient et 153 en reste
   donc 357 = 204x1 + 153
   204 divisé par 153 donne 1 en quotient et 51 en reste
   donc 204 = 153x1 + 51
   153 divisé par 51 donne 3 en quotient et 0 en reste
   donc 153 = 51x3+ 0
   Le plus grand diviseur commun de 561 et 357 est : 51
   ```

   Ainsi, l'appel de `pgcdParEuclide(561,357)` doit renvoyer 51.

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

3. Fonctions (TP)

Exercice 3.5

La liste des fonctions Python existantes (comme par exemple `print()` et `input()` que vous avez déjà utilisées en TP) est disponible à l'adresse suivante :

`https://docs.Python.org/fr/3/library/functions.html`

Dans l'interpréteur de commandes, taper les instructions suivantes :

```
abs(-5)
help(abs)
max(5,12)
help(max)
min(5,12)
help(min)
```

Exercice 3.6 *(Mise en œuvre de l'exercice 3.1)*

Écrire en Python une fonction `maximum()` prenant 2 arguments de type entier et renvoyant le maximum des deux. Tester cette fonction dans un programme Python.

NB : Comme le montre l'exercice précédent, il existe une fonction `max` en Python. Vous devez donc, dans cet exercice, écrire une fonction similaire portant un autre nom que la fonction Python existante.

Exercice 3.7 *(Mise en œuvre de l'exercice 3.2)*

Écrire en Python [3] :

1. une fonction `cube()` qui calcule le cube d'un entier n passé en paramètre ;
2. une fonction `volume()` qui calcule le volume d'une sphère de rayon r passé en paramètre en appelant la fonction `cube()` précédente. Pour rappel : le volume d'une sphère se calcule par la formule : $V = (4/3) \times \pi \times r^3$;
3. un programme qui permet de tester la fonction précédente en affichant le volume d'une sphère dont le rayon est saisi par l'utilisateur.

Exercice 3.8 *(Mise en œuvre de l'exercice 3.3)*

Écrire en Python [4] une fonction `tableMulti()` permettant d'afficher une table de multiplication avec 3 arguments : `base`, `debut` et `fin`, la fonction affichant le résultat de la multiplication de `base` par tous les entiers situés entre `debut` et `fin` inclus. Par exemple l'appel de `tableMulti(8, 13, 17)` devra afficher :

3. Exercice repris de *Apprendre à programmer avec Python* de Gérard Swinnen, 2009
4. repris de `http://python.developpez.com/cours/TutoSwinnen/?page=page_9`

```
Fragment de la table de multiplication par 8 :
13 x 8 = 104
14 x 8 = 112
15 x 8 = 120
16 x 8 = 128
17 x 8 = 136
```

Écrire un programme utilisant la fonction précédente pour afficher une table de multiplication dont les paramètres auront été donnés par l'utilisateur.

Exercice 3.9 *(Généralisation de l'exercice 2.6)*

Dans l'exercice 2.16, nous avions écrit un programme Python qui affichait :

```
1   2   3   4   5
1   3   5   7   9
1   4   7   10  13
1   5   9   13  17
```

Ce programme ne fonctionne que pour 4 lignes et 5 colonnes. Écrire une fonction qui réalise un affichage similaire mais pour un nombre de lignes et de colonnes passés en paramètre. Écrire un programme qui permet de tester votre fonction en demandant à l'utilisateur de saisir le nombre de lignes et de colonnes voulues.

Exercice 3.10 *(Mise en œuvre de l'exercice 3.4)*

Écrire 3 fonctions [5] qui permettent de calculer le Plus Grand Commun Diviseur (PGCD) de deux nombres a et b (avec $a \geq b$).

1. La première fonction `pgcdParDiviseurs(a,b)`, avec $a \geq b$, calculera les diviseurs communs à chacun des deux nombres et renverra le plus grand diviseur commun (on pourra ne calculer que les diviseurs nécessaires).

   Par exemple, si on appelle `pgcdParDiviseurs(63,42)`, les diviseurs de `63` étant `1;3;7;9;21;63` et ceux de `42` étant `1;2;3;6;7;14;21;42`, on doit pouvoir afficher, à l'aide de la valeur renvoyée :

   ```
   Le plus grand diviseur commun de 63 et 42 est: 21
   ```

2. La deuxième fonction `pgcdParDifferences(a,b)`, avec $a \geq b$, utilisera l'algorithme des différences pour renvoyer le PGCD. Si un nombre est un diviseur de 2 autres nombres a et b, alors il est aussi un diviseur de leur différence $a - b$. Le PGCD de a et b est donc aussi celui de b et $a - b$.

   Par exemple, calculons le PGCD de `60` et `36`.

   `60 - 36 = 24`, donc le PGCD de `60` et `36` est un diviseur de `24`. On recommence en effectuant une nouvelle soustraction entre le résultat obtenu à la soustraction précédente (24) et le plus petit des deux termes de la soustraction précédente (36) : on obtient `36 - 24 = 12`. Par conséquent, le PGCD de `60` et `36` est un diviseur de `12` et on peut ré-appliquer la soustraction. On arrête l'algorithme dès que la soustraction donne un résultat nul. Le PGCD correspond au résultat obtenu à l'étape précédente.. Voici l'illustration des étapes :

5. inspiré de http://www.mathematiquesfaciles.com/calculer-le-pgcd-d-un-nombre-avec-cours_2_75657.htm

```
60 - 36 = 24
36 - 24 = 12
24 - 12 = 12
12 - 12 = 0
```

Ainsi, l'appel de `pgcdParDifferences(60,36)` doit renvoyer 12.

Vérifier que votre programme rend le bon résultat pour les couples $(60, 32)$; $(6, 2)$; $(36, 15)$.

3. La troisième fonction `pgcdParEuclide(a,b)`, avec $a \geq b$, utilisera l'algorithme d'Euclide pour renvoyer le PGCD. L'algorithme d'Euclide pour calculer le PGCD de deux entiers a et b (avec $a \geq b$) consiste à effectuer une suite de divisions euclidiennes :
   — étape 1 : effectuer la division euclidienne de a par b ; on obtient r le reste ; si r est égal à 0, alors fin et le PGCD est égal à b, sinon aller à l'étape 2 ;
   — étape 2 : a reçoit alors la valeur de b et b reçoit la valeur de r et aller à l'étape 1.

   Cette méthode en général est plus rapide.

   Voici l'illustration des étapes :

```
561 divisé par 357 donne 1 en quotient et 204 en reste
donc 561 = 357 x 1 + 204
357 divisé par 204 donne 1 en quotient et 153 en reste
donc 357 = 204x1 + 153
204 divisé par 153 donne 1 en quotient et 51 en reste
donc 204 = 153x1 + 51
153 divisé par 51 donne 3 en quotient et 0 en reste
donc 153 = 51x3+ 0
Le plus grand diviseur commun de 561 et 357 est : 51
```

   Ainsi, l'appel de `pgcdParEuclide(561,357)` doit renvoyer 51.

Exercice 3.11

1. Écrire en Python une fonction `AleatoireAvecRepetitions(n)` qui prend en paramètre un entier strictement positif n et effectue le traitement suivant :
   — la fonction génère au hasard un nombre entier qui sera 0 ou 1. Elle répète ces tirages aléatoires jusqu'à avoir obtenu n fois 1 de façon consécutive ;
   — ensuite la fonction réalise un dernier tirage aléatoire entre 0 et 1 ;
   — la fonction doit alors renvoyer la valeur de ce dernier tirage aléatoire ainsi que le nombre total de tirages aléatoires qu'elle aura dû effectuer au préalable.
2. Écrire en Python le programme principal qui réalise les tâches suivantes :
   — le programme demande à l'utilisateur un nombre entier n strictement positif, correspondant au nombre de répétitions ;
   — le programme affiche ensuite le nombre de 1, renvoyés par 1000 appels à la fonction `AleatoireAvecRepetitions(n)`, et la valeur moyenne du nombre total de tirages.
3. Tester votre programme (après avoir testé votre fonction en demandant des affichages intermédiaires si nécessaires) pour des valeurs de n allant de 1 à 7. Que constatez-vous ? Comment l'expliquez-vous ?

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

4. Séquences, tableaux et instruction POUR (exercices de TD)

Exercice 4.1

Écrire un algorithme qui déclare un tableau de 10 notes, dont les valeurs sont saisies par l'utilisateur. L'algorithme détermine et affiche la moyenne de ces notes, puis calcule et affiche le nombre de notes au-dessus de cette moyenne.

Modifier l'algorithme pour réaliser les mêmes calculs et affichages sur un nombre n de notes, ce nombre n étant saisi par l'utilisateur.

Compléter l'algorithme pour qu'il affiche (en dernier) la note la plus élevée ainsi que sa position dans le tableau (la position de la première occurrence si la note la plus élevée apparait plusieurs fois).

Exercice 4.2

1. Écrire un algorithme qui lit un tableau `tab` de taille `long` (entrée par l'utilisateur) dont les éléments sont des nombres, puis affiche `tab` sans ses éléments négatifs.
2. Écrire un nouvel algorithme qui permet, non pas de simplement d'afficher, mais de modifier `tab` en enlevant tous les nombres négatifs. On peut alors s'aider d'une variable intermédiaire `tab2` de type tableau également.
3. Refaire le même exercice mais sans utiliser cette fois de variable intermédiaire de type tableau : il faudra afficher le tableau initial, puis le tableau modifié.

Exercice 4.3

1. Écrire une fonction `TabAlea(n,a,b)` qui prend trois arguments - 3 valeurs entières n, a et b - et qui renvoie un tableau de n entiers aléatoirement choisis entre a et b inclus. On supposera qu'il existe une fonction `nombreAleatoire(a,b)` qui permet d'obtenir un nombre entier aléatoire compris entre a et b inclus.

2. Puis, écrire une fonction `TabProduit(T)` qui prend un tableau `T` de valeurs entières en argument et qui renvoie le produit de tous les éléments du tableau.

3. Écrire un algorithme qui lit trois entiers n, a et b et qui appelle successivement les fonctions `TabAlea()` et `TabProduit()` pour afficher le tableau de valeurs entières ainsi que le produit de ces valeurs.

Exercice 4.4

Quel résultat produira l'algorithme suivant ?

```
ALGO
VARIABLES
    i, j TYPE NOMBRE
    tab TYPE TABLEAU DE NOMBRE
DEBUT
    tab <- CREER_TABLEAU(4, 2)
    POUR i ALLANT DE 0 A 3
        DEBUT
            POUR j ALLANT DE 0 A 1
                DEBUT
                    tab[i][j] <- 2*i + j
                FIN
        FIN
    POUR i ALLANT DE 0 A 3
        DEBUT
            POUR j ALLANT DE 0 A 1
                DEBUT
                    ECRIRE tab[i][j]
                FIN
        FIN
FIN
```

Exercice 4.5

Soit un tableau T à deux dimensions, 12 par 8, de valeurs numériques (à remplir par l'utilisateur). Écrire un algorithme qui recherche la plus grande valeur au sein de ce tableau.

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

4. Modules externes - Casino (TP)

Exercice 4.6 *(Debugage d'une solution de l'exercice 3.10)*

L'équipe Python a écrit une solution pour l'exercice 3.10 sur le calcul du pdcg. Malheureusement le programme ne fonctionne pas ! Vous allez devoir le corriger et le faire fonctionner.

Pour récupérer le fichier, aller sur le répertoire `ETUD/DEMI2E_L1_AlgoProg/` et copier, dans votre répertoire personnel, le fichier `Debug_pgcd.py`.

1. Exécuter le programme et corriger les erreurs de syntaxe annoncées par l'interpréteur.
2. Pourquoi le mot-clé `None` s'affiche-t-il ? Modifier le programme pour supprimer ces affichages inutiles et améliorer la lisibilité de l'affichage général.
3. Après avoir ré-exécuté le programme, que remarquez-vous ? Quel est le bon résultat ? Corriger les erreurs de conception algorithmique dans chacune des fonctions en ajoutant les affichages intermédiaires utiles.
4. Modifier le programme pour tester le cas $a = 40$, $b = 20$ et corriger si nécessaire.
5. Modifier le programme pour tester le cas $a = 4$, $b = 0$ et corriger si nécessaire.

Exercice 4.7

1. Écrire un programme Python qui permet de calculer et d'afficher le cosinus de $\pi/2$ (module `math`).
2. Écrire un programme Python qui permet d'afficher une série de 10 nombres réels aléatoires compris entre 10 et 50 (inclus) avec 1 seconde d'intervalle entre chaque écriture (modules `random` et `time`).

Exercice 4.8 *(Évaluation de π par la méthode de Monte-Carlo)*

Soit un cercle, de centre $(0, 0)$ et de rayon $r = 1$, inscrit dans un carré de coté $l = 2$. L'aire du carré vaut 4 et l'aire du cercle vaut π. En choisissant N points aléatoires (à l'aide d'une distribution uniforme) à l'intérieur du carré, chaque point a une probabilité

$$p = \frac{\text{aire du cercle}}{\text{aire du carré}} = \frac{\pi}{4}$$

de se trouver également dans le cercle.

Soit n le nombre de points tirés aléatoirement se trouvant effectivement dans le cercle, on a :

$$p = \frac{n}{N} \approx \frac{\pi}{4}$$

d'où

$$\pi \approx 4 \times \frac{n}{N}$$

Déterminer une approximation de π par cette méthode. Pour cela, on procède en N itérations : à chaque itération, choisir aléatoirement les coordonnées d'un point entre -1 et 1 (fonction `uniform()` du module `random`), calculer la distance entre ce point et le centre du cercle, déterminer si cette distance est inférieure au rayon du cercle égal à 1, et si c'est le cas, incrémenter le compteur n de 1. Quelle est la qualité de l'approximation de π pour $N = 50$, $N = 500$, $N = 5\,000$ et $N = 50\,000$?

Exercice 4.9 *(À vous de jouer !)*

Écrire un programme de jeu de roulette (très simplifié) dans lequel le joueur peut miser une certaine somme sur un numéro et gagner ou perdre de l'argent si ce numéro est tiré aléatoirement. Le joueur peut enchainer plusieurs parties mais, il est contraint de s'arrêter s'il ne lui reste plus d'argent !

Voici la règle du jeu :

Le joueur mise une certaine somme sur un numéro compris entre 0 et 49 (50 numéros en tout) en déposant cette somme sur le numéro choisi sur la table de jeu : il donne donc sa mise au croupier. La roulette est constituée de 50 cases allant de 0 à 49. Les numéros pairs sont de couleur noire, les numéros impairs sont de couleur rouge. Le croupier lance la roulette, lâche la bille et quand la roulette s'arrête, relève le numéro de la case dans laquelle la bille s'est arrêtée. Le numéro sur lequel s'est arrêtée la bille est, naturellement, le numéro gagnant. Si le numéro gagnant est celui sur lequel le joueur a misé (probabilité de 1/50, plutôt faible), le croupier lui remet 3 fois la somme misée. Dans le cas contraire, si le numéro gagnant a la même couleur que celui sur lequel le joueur a misé, le croupier lui remet 1,5 fois la somme misée ; sinon, lorsque le joueur a misé sur le mauvais numéro de mauvaise couleur, le joueur perd définitivement sa mise.

Écrire dans un premier temps en pseudo-code l'algorithme conforme à la règle du jeu. Ensuite, traduire cet algorithme en Python. Dans votre programme, il faut lire différentes valeurs (et vérifier leur cohérence vis-à-vis des valeurs attendues) : le budget initial du joueur, le numéro choisi par le joueur, le fait qu'il souhaite ou non continuer de jouer.

Remarque : avec cette règle du jeu, si le joueur mise sur la bonne couleur mais pas le bon numéro, son bénéfice est de 50% de la somme misée. Au bout de nombreuses parties, on risque

d'arriver à des nombres flottants avec beaucoup de chiffres après la virgule. Alors autant arrondir au nombre supérieur. Ainsi, si le joueur mise 3 euros sur la bonne couleur mais pas le bon numéro, il gagne 2 euros et le croupier lui rend $3 + 2 = 5$ euros. Pour cela, on va utiliser une fonction du module `math` nommée `ceil()`.

5. Chaînes de caractères en Python (exercices de TD)

Exercice 5.1

Décrire ce que fait le programme Python ci-dessous :

```
print("Entrer une chaine de caracteres :")
ch = input()
i = 0
lgr = len(ch)
chNew = ""
while i < lgr:
    if (ch[i] == "," or  ch[i] == ";" or  ch[i] == "."):
        chNew = chNew+"p"
    else:
        chNew = chNew+ch[i]
    i += 1
print(chNew)
```

Lorsque l'utilisateur saisit au clavier :

```
En Python, la ponctuation ;,. est facile à manipuler.
```

indiquer ce que va faire ce programme.

Écrire une nouvelle version beaucoup plus simple de ce programme Python en utilisant des méthodes de chaînes.

Exercice 5.2

Écrire une fonction Python, appelée `palindrome()`, qui prend une chaîne de caractères `ch` en argument et renvoie `True` si `ch` est un palindrome et, `False` sinon. Un palindrome est une chaîne de caractères dont l'ordre des lettres reste le même qu'on la lise de la gauche vers la droite ou de la droite vers la gauche. C'est par exemple le cas de la chaîne `"kayak"`. On supposera que la chaîne de caractères lue ne contient que des caractères minuscules non accentués, et qu'elle ne contient ni espace, ni caractère de ponctuation. Écrire un programme Python qui lit une chaîne de caractères saisie par l'utilisateur, appelle la fonction `palindrome` pour afficher si oui ou non la chaîne saisie est un palindrome.

Exercice 5.3

Écrire un programme Python qui lit une chaîne de caractères `ch` et qui affiche la chaîne de caractères obtenue en ignorant les mots de longueur inférieure ou égale à 3 dans `ch`. Par exemple, si la chaîne lue est `"bonjour les amies !"` l'algorithme affichera `"bonjour amies"`. On définira un mot comme toute suite de caractères quelconques ne contenant pas d'espace.

Exercice 5.4

Écrire en Python un programme qui lit une chaîne de caractères (supposée sans espace et de longueur au moins égale à 3) et l'affiche sous la forme d'un N. Si l'on saisit par exemple "`abcde`", alors le programme doit afficher :

```
a   a
bb  b
c c c
d  dd
e   e
```

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

5. Chaînes de caractères en Python (TP)

Exercice 5.5

1. Écrire un programme Python qui lit une chaîne de caractères `s`, affiche sa longueur, puis affiche `s` avec tous les espaces du début, intermédiaires et de la fin, supprimées.
2. Écrire un nouveau programme Python qui permet, non pas simplement d'afficher `s`, mais de modifier `s` en enlevant tous les espaces du début, intermédiaires et de la fin.

Exercice 5.6

Écrire un programme Python qui lit une chaîne de caractères, puis lit une ou plusieurs valeurs entières (tant que l'utilisateur le souhaite) et qui, pour chacune de ces valeurs `v`, remplace le caractère situé à l'indice `v` par le caractère `'?'`. Ce programme se termine par l'affichage de la nouvelle chaîne. Attention : il faut bien vérifier que les valeurs `v` correspondent à des indices valides pour la chaîne de caractères lue.

Exercice 5.7

Écrire un programme Python qui lit une chaîne de caractères et qui affiche cette chaîne avec les suites de caractères identiques remplacées par un seul caractère. Par exemple, on donnera la chaîne `"abbcdebbb"` et votre programme devra afficher `"abcdeb."`

Exercice 5.8

Écrire une fonction Python, appelée `test()`, qui prend 3 arguments : un entier `n` positif, et deux chaînes de caractères (ne contenant pas d'espace) `ch1` et `ch2`. Cette fonction renvoie `True` si les deux conditions suivantes sont vérifiées : `ch1` contient `n` fois `ch2`, et dans `ch2` tout caractère est utilisé au plus une fois ; sinon la valeur booléenne `False` est renvoyée.

Écrire un programme Python qui lit un entier `n` positif, puis deux chaînes de caractères (ne contenant pas d'espace) `ch1` et `ch2`, et appelle la fonction `test()` sur ces variables pour afficher `"oui"` si la valeur de retour de `test()` est `True` et `"non"` sinon. Le tableau ci-dessous donne des exemples de valeurs que le programme doit afficher.

ch1	ch2	n	valeur affichée
"habhabab"	"abh"	1	oui
"habhabab"	"abh"	2	non
"habhabab"	"ab"	3	oui
"habhabab"	"ab"	4	non
"baaabaa"	"aa"	2	non
"baaabaa"	"aa"	3	non
"ababaaaba"	"ab"	3	oui

Exercice 5.9

Écrire une fonction Python, appelée `test2()`, qui prend deux chaînes de caractères `ch1` et `ch2` en argument. Cette fonction renvoie un entier `N` tel que `ch1` est égale à `N` concaténations successives de `ch2` avec elle-même (ou 0 si ce n'est pas le cas).

Écrire un programme Python qui lit deux chaînes de caractères `ch1` et `ch2`, et appelle la fonction `test2()` pour afficher `N` s'il existe ; dans le cas contraire, le programme affichera `"non"`. On suppose que l'utilisateur saisira toujours des chaînes de caractères `ch1` et `ch2` non vides ne contenant pas d'espace. Le tableau ci-dessous donne des exemples de valeurs que le programme doit afficher.

ch1	ch2	valeur affichée
"ababab"	"ab"	3
"ab"	"ab"	1
"abhabab"	"ab"	non
"aaaa"	"aa"	2
"aaa"	"aa"	non
"aaa"	"a"	3

Exercice 5.10

Écrire un programme Python qui lit une chaîne de caractères et l'affiche sous la forme d'un `X`.
Si l'on saisit par exemple "`abcde`", alors il doit afficher :

```
a   e
 b d
  c
 b d
a   e
```

Si l'on saisit par exemple "`abcd`", alors il doit afficher :

```
a  d
 bc
 bc
a  d
```

Tester votre programme, entre autre avec les chaînes de caractères `MAGIQUE` et `PYTHON`, et avec la chaîne obtenue en concaténant les deux précédentes.

Exercice 5.11

Écrire un programme Python qui prend en entrée deux chaînes de caractères `s1` et `s2` (supposées sans espace), et affiche `"OK"` s'il est possible, en enlevant certains caractères de `s2`, de retrouver `s1`. Dans le cas contrainte, le programme affiche `"impossible"`.

Par exemple, si `s1='bd'` et `s2='abcde'`, le programme affiche `"OK"`.

Avec `s1='bd'` et `s2='bd'`, le programme affiche également `"OK"`.

Par contre, il affiche `"impossible"` lorsque `s1='db'` et `s2='abcde'`.

Exercice 5.12

Il pîaart que puor la lctreue, l'orrde des lrttees à l'iétunreir des mots n'a acnuue itnpocmare. La sulee chose qui cptmoe est que la pemièrre et la dneèirre lttree seonit à leur pclae.

L'objectif de cet exercice est de concevoir un programme en **Python** permettant de tester cette théorie. Étant donnée une chaîne de caractères saisie par l'utilisateur, il mélangera aléatoirement les lettres à l'intérieur des mots et affichera la phrase modifiée. On supposera que la chaîne de caractères ne comporte pas de signe de ponctuation et que deux mots sont séparés par exactement une espace.

Attention : L'utilisation du type `list` est interdite dans cet exercice.

1. Écrire une fonction `permuteMot()` qui prend en argument un mot et permute aléatoirement les lettres à l'intérieur du mot en gardant les première et dernière lettres inchangées.
2. Écrire une fonction `permute()` qui prend en argument une phrase et applique la première fonction sur chacun des mots.
3. Écrire le programme demandé en utilisant les fonctions précédentes.

Exemple : L'utilisation saisit la phrase `"Je vais avoir une bonne note"`, alors le programme peut afficher `"Je vias avior une bnone ntoe"`.

Exercice 5.13

Écrire un programme Python qui lit une chaîne de caractères `ch` et qui affiche la liste des mots de `ch` ainsi que les indices de début de chacun des mots dans `ch` (chaque mot pouvant se répéter plusieurs fois).
Par exemple, pour `ch=" ab cda fg ab  fg cda h cda  "`, le programme doit afficher :

```
"ab" 1, 11
"cda" 4, 19, 25
"fg" 8, 16
"h" 23
```

Attention, plusieurs espaces peuvent se trouver entre deux mots consécutifs, ainsi qu'avant le premier mot et après le dernier mot.

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

6. Listes en Python et boucle FOR (exercices de TD)

Exercice 6.1

Écrire un programme Python qui fait saisir à l'utilisateur un entier n puis une liste de nombres et les met dans une liste avant de l'afficher.

Exercice 6.2

Écrire un programme Python qui lit une liste de n notes, dont les valeurs sont saisies par l'utilisateur. Ce programme doit calculer et afficher la moyenne de ces notes, puis calculer et afficher le nombre de notes au-dessus de la moyenne. Il doit également afficher la note la plus élevée ainsi que sa position dans la liste.

Exercice 6.3

Soit la suite U définie comme suit : $U_n = 5U_{n-1} + 10U_{n-2}$ avec $U_0 = 1$ et $U_1 = 2$.
Écrire le programme qui affiche la liste $[U_0, U_1, ..., U_{n-1}, U_n]$ étant donné un $n \geq 2$ saisi par l'utilisateur.

Exercice 6.4

Soient deux listes d'entiers `L1` et `L2`. Donner le programme qui permet de construire la liste `LRES` qui contient la somme des éléments de même rang de la liste `L1` et de la liste `L2` auxquels on ajoute les derniers éléments de la liste la plus longue.

Écrire et simuler votre programme sur la somme de [1,2,3,4,5] et [2,15,18] qui devrait donner [3,17,21,4,5].

Exercice 6.5

Écrire un programme Python qui fait saisir à l'utilisateur une liste de listes de nombres.

Exercice 6.6

Étant donnée la liste de listes suivante :

```
Résultats = [ ["Bob","Python",11],
              ["Zoe","Python",14],
              ["Bob","algebre",10.5] ]
```

Proposer un programme en Python permettant de calculer la moyenne d'un élève donné, par exemple `Bob`.

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

6. Listes en Python et boucle `FOR` (TP)

Exercice 6.7

On veut calculer la somme des entiers de 1 à n, sans utiliser la formule $\frac{n(n+1)}{2}$ mais à l'aide d'une boucle `for`.

1. Écrire le programme Python correspondant. Dans un premier temps, écrire une boucle `for` pour calculer la somme des entiers de 1 à 10. La somme sera stockée dans une variable nommée `somme`. Afficher cette somme à l'écran en fin de programme.
2. Modifier votre programme pour obliger l'utilisateur à saisir un nombre n strictement positif, puis calculer la somme des entiers de 1 à n, et l'afficher à l'écran en fin de programme.
3. Proposer un programme utilisant la fonction `sum()` permettant de répondre à la question précédente. Remarque : en tapant `help(sum)` dans l'interpréteur Python, vous obtiendrez l'aide concernant cette fonction.

Exercice 6.8 *(Mise en œuvre de l'exercice 4.3)*

Écrire une fonction `TabAlea(n,a,b)` qui prend trois arguments - 3 valeurs entières n, a et b - et qui renvoie une liste de n entiers aléatoirement choisis entre a et b inclus.

Puis, écrire une fonction `TabProduit(T)` qui prend une liste `T` de valeurs entières en argument et qui renvoie le produit de tous les éléments de la liste.

Écrire un programme Python qui lit trois entiers n, a et b et qui appelle successivement les fonctions `TabAlea()` et `TabProduit()` pour afficher la liste de valeurs entières ainsi que le produit de ces valeurs.

Exercice 6.9

1. Écrire un programme qui crée une liste contenant les 20 premiers multiples de 7 strictement positifs.
2. Compléter votre programme pour qu'il affiche les 20 premiers multiples de 7 strictement positifs, séparés par un point-virgule sur une même ligne, et aller à la ligne après chaque multiple de 3.
3. Modifier ce programme pour qu'il :
   - fasse saisir par l'utilisateur un nombre n et un nombre p strictement positifs et crée une liste contenant les n premiers multiples de 7 ;
   - affiche cette liste selon le format suivant : les n premiers multiples de 7, séparés par un point-virgule sur une même ligne, mais en allant à la ligne après chaque multiple de p.
4. Modifier ce programme pour que, en plus, il compte les multiples de p qui auront été affichés et affiche ce nombre à la fin.

Exemple d'exécution :

```
  Combien de multiples de 7 voulez-vous afficher ? 20
  Vous irez à la ligne après chaque multiple de ...
  (nombre strictement positif svp) : 3
7 ; 14 ; 21
28 ; 35 ; 42
49 ; 56 ; 63
70 ; 77 ; 84
91 ; 98 ; 105
112 ; 119 ; 126
133 ; 140 ;
Vous avez affiché 6 multiples de 3
```

Exercice 6.10

Écrire un programme Python qui fait saisir à l'utilisateur une liste de listes de nombres.

Exercice 6.11

Écrire un programme en Python qui, étant donnée, une liste de listes de nombres (initialisée dans le programme), affiche la somme de la liste de nombres la plus longue (s'il y en a plusieurs, on prendra la première).
Par exemple, pour la liste $[[1, 2, 3], [], [2, 3, 0, 4], [2, 5], [6, 0, 2, 4]]$, le programme affichera 9.

Exercice 6.12

Après un examen, les notes des étudiants sont les suivantes : Florian : 2, Antoine : 12, Charles1 : 0, Robert : 0, Charles2 : 8, Erwan : 7, Jean : 20, Xavier : 11, Didier : 20, Alain : 0, Hadi : 12.

1. Utiliser deux listes pour stocker les noms des étudiants et les notes associées.
2. Calculer et afficher la moyenne de l'examen.
3. Afficher le nom et la note de chacun des élèves, avec la mention « admis » si la note est supérieure ou égale a 10, ou bien « recalé » sinon.
4. Stocker dans des listes séparées les élèves admis et les élèves recalés.
5. Les personnes ayant eu 0 à l'examen ne se sont jamais présentées. Modifier votre programme pour supprimer les informations relatives à ces élèves.
6. Modifier votre programme pour permettre à l'utilisateur de saisir de nouvelles notes.

Exercice 6.13

Soit la matrice des distances (en km) entre les villes suivantes :

Auxerre	0	45	35	25	20
Avallon	45	0	30	70	100
Clamecy	35	30	0	60	55
Joigny	25	70	60	0	10
Migennes	20	100	55	10	0
	Auxerre	Avallon	Clamecy	Joigny	Migennes

On indice les villes de 0 à 4 dans l'ordre alphabétique (0 pour Auxerre, 1 pour Avallon, ... et 4 pour Migennes).

1. Écrire une fonction `lireDistance()` qui lit les distances entre ces 5 villes, saisies au clavier par l'utilisateur, et renvoie le tableau à deux dimensions des distances entre ces villes. On veillera à ce que pour chaque couple de ville i et j, la distance ne soit saisie qu'une seule fois par l'utilisateur (car pour tout i, j, on a `distance[i][j]= distance[j][i]`). En Python, ce tableau correspond à une variable `distance` de type `list` de `list` telle que `distance[i][j]` contient la distance entre la ville indicée par `i` et la ville indicée par `j`.
2. Écrire une fonction qui vérifie l'inégalité triangulaire, c'est-à-dire que pour toutes les valeurs i, j et k, on a : `distance[i][j]` $\leq$ `distance[i][k]+distance[k][j]`. Cette fonction doit renvoyer la listes des triplets i,j,k qui ne respectent pas l'inégalité triangulaire.
3. Écrire le programme principal qui appelle la fonction `lireDistance()`, construit le tableau des distances et teste si les inégalités triangulaires sont respectées. Si elles sont respectées, un message l'indiquant doit être affiché, sinon, pour chacun des triplets ne vérifiant pas l'inégalité triangulaire, il faut afficher les trois villes concernées.

7. Notions avancées sur les listes (exercices de TD)

Exercice 7.1

Simuler l'exécution dans l'interpréteur des instructions suivantes :

```
L = [1,2,3,'toto',8,True]
L[0]
L[4]
len(L)
L[len(L)]
L[-1]
L[1:-2]
L[1:5:2]
L[:5:-1]
L[:4:-1]
L[5::-1]
L[3][1]
```

Exercice 7.2

Quel est l'affichage des deux programmes ci-dessous ?
Qu'est-ce qui les différencie concernant `L` ?

Programme 1 :

```
L = []
L1 = L
for i in range(5):
    L = L+[i]
print(L)
print(L1)
```

Programme 2 :

```
L = []
L1 = L
for i in range(5):
    L.append(i)
print(L)
print(L1)
```

Exercice 7.3

Considérer le programme Python ci-dessous :

```
def f(k, taille, Liste):
    for j in range(k+1, taille):
        if Liste[k] > Liste[j]:
            x = Liste[k]
            Liste[k] = Liste[j]
            Liste[j] = x

L = []
n = int(input())
for i in range(n):
    L.append(int(input()))
for i in range(n-1):
    f(i,n,L)
print(L)
```

1. Dans la fonction `f()`, à quoi sert `x` ? Quels sont les types des arguments de `f()` ?
2. Décrire ce que réalise la fonction `f()`.
3. Que permet de faire le bloc d'instructions suivant :

```
for i in range(n):
    L.append(int(input()))
```

4. Que permet de faire le bloc d'instructions suivant :

```
for i in range(n-1):
    f(i, n, L)
```

   Pourquoi la dernière valeur à considérer pour `i` dans cette boucle doit-elle être `n-2` ?
5. Que va afficher le programme si l'utilisateur saisit d'abord 6 puis 14 3 54 2 1 5 ?

Exercice 7.4

On cherche à écrire une fonction permettant de supprimer d'une liste donnée en argument toutes les occurrences d'un élément donné également en argument.

1. On vous propose la fonction `supp_1()` ci-dessous :

```
def supp_1(L, elem):
    i = 0
    while (i < len(L)):
        if (L[i] == elem) :
            L.pop(i)
        i += 1
```

   Simuler l'exécution de cette fonction sur la liste `[10, 20, 30]` et l'élément `20`.

2. On vous propose maintenant la fonction `supp_2()` ci-dessous :

```
def supp_2(L, elem):
    for i in range(len(L)):
        if (L[i] == elem) :
            L.pop(i)
```

   Simuler l'exécution de cette fonction sur la liste `[10, 20, 30]` et l'élément `20`. D'où vient le problème ?

3. On vous propose maintenant la fonction `supp_3()` ci-dessous :

```
def supp_3(L, elem):
    for e in L:
        if (e == elem) :
            L.remove(e)
```

   Simuler l'exécution de cette fonction sur la liste `[10, 20, 30]` et l'élément `20`.
   Simuler l'exécution de cette fonction sur la liste `[10, 20, 10]` et l'élément `10`.
   Simuler l'exécution de cette fonction sur la liste `[10, 10, 20]` et l'élément `10`. D'où vient le problème ?

4. Reprendre la fonction `supp_1()` et simuler son exécution sur la liste `[10, 20, 10]` et l'élément `10`.
   Simuler l'exécution de cette fonction sur la liste `[10, 10, 20]` et l'élément `10`.
   Proposer une version correcte pour la fonction.

7. Notions avancées sur les listes (exercices de TP)

Exercice 7.5 *(Debugage d'une solution de l'exercice 6.9)*

L'équipe Python a écrit une solution pour l'exercice 6.9 sur l'affichage des multiples de 7. Malheureusement le programme ne fonctionne pas ! Vous allez devoir le corriger et le faire fonctionner.

Pour récupérer le fichier, aller sur le répertoire `ETUD/DEMI2E_L1_AlgoProg/` et copier, dans votre répertoire personnel, le fichier `Debug_multiples.py`.

1. Exécuter le programme. Que vous indique l'erreur de syntaxe ? Corriger cette erreur.
2. Tester le programme en affichant 20 multiples de 7 en allant à la ligne après les multiples de 3. Que vous indiquent les deux erreurs sur `multi` ? Corriger-les.
3. Tester le programme en affichant 20 multiples de 7 en allant à la ligne après les multiples de 3. Corriger l'erreur de l'algorithme, corriger l'affichage.
4. Tester le programme en affichant 20 multiples de 7 en allant à la ligne après les multiples de 3. Corriger l'affichage final.
5. Tester le programme en affichant 10 multiples de 7 en allant à la ligne après les multiples de 4. Que faut-il corriger pour que l'algorithme soit correct ? Tester et corriger l'erreur de syntaxe. Vérifier que votre code fonctionne aussi avec 12 multiples de 7 en allant à la ligne après les multiples de 5 et corriger-le si nécessaire.

Exercice 7.6

Exécuter dans l'interpréteur les instructions suivantes et analyser le résultat :

```
L = [1,2,3,'toto',8,True]
L[0]
L[4]
len(L)
L[len(L)]
L[-1]
L[1:-2]
L[1:5:2]
L[:5:-1]
L[:4:-1]
L[5::-1]
L[3][1]
L.append(3)
L.append([3])
L += 4
L += [4]
L.count('toto')
L.count(3)
L.index('toto')
L.index(3)
'toto' in L
[1,2,3] in L
[1,3,2] in L
list(range(10))
list(range(1,11))
list(range(1,11,2))
L = list(range(101,1,-1))
```

Exercice 7.7

Étant donnés une liste L et un élément x, donner le programme qui permet de supprimer toutes les occurrences de l'élément x dans la liste L.
Écrire et tester votre programme pour l'élément 0 dans la liste $L = [0, 1, 2, 0, 0, 4]$.

Exercice 7.8

1. Écrire une fonction, appelée `filtrage()`, qui prend une liste `Lr` de réels en argument, et enlève de `Lr` tous les éléments négatifs.
2. Appeler cette fonction sur la liste [-2.3,5,-9,0,12.5,-6] et afficher les informations suivantes :

   ```
   Liste avant filtrage : [-2.3,5,-9,0,12.5,-6]
   Liste après filtrage : [5,0,12.5]
   ```

   Après l'appel de la fonction `filtrage()`, la liste initiale n'existe plus, elle est remplacée par la liste filtrée. Modifier la fonction `filtrage()` pour conserver en mémoire à la fois la liste initiale et la liste filtrée.

Exercice 7.9

1. Dans le module `random` se trouve une fonction nommée `randint()`. Utilisez-la pour écrire une fonction, appelée `listeAlea()`, qui prend en argument 3 entiers a, b et n et qui renvoie une liste de n entiers tirés au hasard entre a et b.
2. Écrire une fonction, appelée `absent()`, qui prend en argument une liste `L` comprenant 100 nombres tirés au hasard entre 0 et 99, créée avec la fonction `listeAlea()`, et renvoie le nombre d'entiers compris entre 0 et 99 qui n'appartiennent pas à `L`.
3. Dans le programme principal, recommencer ce tirage aléatoire 1000 fois, calculer et afficher la moyenne du nombre d'absents.
4. Comparer à la valeur théorique.

Exercice 7.10

On considère une séquence de n entiers relatifs $S = (a_1, \ldots, a_n)$. On appelle coupe de S toute sous-séquence non vide. Ainsi, pour tout i, j avec $1 \leq i \leq j \leq n$, $(a_i, \ldots, a_j)$ est une coupe qu'on notera $S_{i,j}$. La valeur d'une coupe $S_{i,j}$ est définie par la somme des éléments qui la compose :

$$v_{i,j} = \sum_{k=i}^{j} a_k$$

Il s'agit de déterminer un algorithme efficace pour déterminer la coupe de valeur minimale.

Par exemple dans la séquence $S = (2, -6, 4, 5, -10, -3, 2)$, $S_{2,4} = (-6, 4, 5)$ de valeur $v_{2,4} = -6 + 4 + 5 = 3$. La coupe de valeur minimale est $S_{5,6} = (-10, -3)$ de valeur $v_{5,6} = -10 - 3 = -13$.

1. **Algorithme naïf**
   (a) Écrire en Python une fonction `somme()` prenant en paramètre une coupe et renvoyant sa valeur.
   (b) Écrire en Python une fonction `coupeMin1()` prenant en paramètre une séquence `S`, et qui, à l'aide de la fonction `somme()`, renvoie la valeur de la coupe minimale.
2. **Algorithme plus rapide**
   (a) Écrire en Python une fonction `coupei()`, prenant en paramètre une séquence $(a_i, \ldots, a_n)$ et renvoyant la valeur minimale d'une coupe dont le premier élément est a_i. Cette fonction ne doit parcourir la séquence à partir de a_i qu'une seule fois. (Attention : cette fonction ne doit pas appeler la fonction `somme()`).
   (b) Écrire en Python une fonction `coupeMin2()` prenant en paramètre une séquence `S`, et qui, à l'aide de la fonction `coupei()`, renvoie la valeur de la coupe minimale dans `S`.

3. **Algorithme très rapide.**
   Étant donnée la séquence $S_i = (a_1, \ldots, a_i)$, on note v_i la valeur de la coupe minimale dans S_i et m_i la valeur minimale d'une coupe dans S_i se terminant par a_i. On a :

$$m_{i+1} = \min(m_i + a_{i+1}, a_{i+1}) \text{ et } v_{i+1} = \min(v_i, m_{i+1})$$

   Proposer une fonction Python, appelée `coupeMin3()`, prenant en paramètre une séquence `S` et renvoyant la coupe de valeur minimale dans `S`. Cette fonction démarre avec $v = a_1$ et $m = v$, puis à chaque itération i, utilise la relation précédente pour calculer la coupe minimale.

4. L'objectif est de comparer les temps de calculs des trois fonctions `coupeMin1()`, `coupeMin2()` et `coupeMin3()`. À cet effet, pour une séquence donnée, récupérer les temps d'exécution de chacune des trois fonctions à l'aide d'une fonction du module `time`. Pour identifier l'évolution du temps d'exécution en fonction de la taille de la séquence, il faut utiliser la fonction `listeAlea()` de l'exercice 7.9 pour générer aléatoirement 10 séquences de nombres compris entre -100 et 100 de taille 100, de taille 600, de taille 1100 et enfin de taille 1600. Pour une taille donnée, il faut alors calculer une moyenne du temps d'exécution de chaque fonction sur les 10 séquences aléatoires. Votre affichage devra adopter le format suivant :

	Algorithme naif	Algorithme optimisé	Algorithme linéaire
100	tps moyen en s	tps moyen en s	tps moyen en s
600	tps moyen en s	tps moyen en s	tps moyen en s
1100	tps moyen en s	tps moyen en s	tps moyen en s
1600	tps moyen en s	tps moyen en s	tps moyen en s

   Que constatez-vous ?

Exercice 7.11

Exercice repris et adapté de « Google Code Jam », Africa 2010, Qualification Round.

Écrire un programme Python faisant saisir une chaîne de caractères à l'utilisateur, inversant l'ordre des mots de cette chaîne et l'affichant. On supposera qu'il y a exactement une espace entre chaque mot. Par exemple, si l'utilisateur saisit la chaîne `"exercice"`, le programme affichera `"exercice"`.

Si l'utilisateur saisit :
`"Ceci est un exercice de Python"`,
le programme affichera :
`"Python de exercice un est Ceci"`.

1. Écrire une version du programme sans utiliser les méthodes `split()` et `join()`.
2. Écrire une version du programme avec ces méthodes.

Exercice 7.12

Pour organiser le TP noté en salle machine des étudiants de L1, il faut affecter chaque étudiant à une salle. Pour ce faire, on dispose de deux listes initiales que l'on supposera données :

— la liste des étudiants est stockée dans une variable `E` : chaque élément de cette liste est donc une chaîne de caractères représentant le nom d'un étudiant suivi d'un espace et de son prénom ;
— la liste des capacités des salles est stockée dans une variable `S` : chaque élément de cette liste est donc un entier.

Nous supposerons que la somme des capacités des salles est supérieure ou égale au nombre total d'étudiants. L'objectif est de créer les listes des étudiants affectés aléatoirement à chaque salle (une liste par salle dont la capacité est stockée dans la liste `S`). Chaque étudiant devra être affecté une et une seule fois à une unique salle. Ces listes seront elles-mêmes stockées dans une variable de type liste.

1. Écrire en Python une fonction `salle()` prenant en argument une liste `E` et un entier `c` (inférieur ou égal à la taille de la liste `E`) et qui renvoie une liste constituée de `c` éléments distincts de `E` choisis aléatoirement.
2. Écrire en Python une fonction `affectation()` prenant en argument les listes `E` et `S` et qui renvoie une liste dont chaque élément est une liste d'étudiants affectés à une salle. Cette fonction devra faire appel à la fonction `salle()`.
3. Écrire en Python une fonction `emargement()` qui prend en argument une liste de listes et ne renvoie rien. Cette fonction permet de créer pour chaque salle un fichier qui contient les nom et prénom de chacun des étudiants affectés à cette salle. Sur la première ligne de chaque fichier sera écrit `Feuille d'émargement`, puis un nom et un prénom d'étudiant par ligne, et enfin sur la dernière ligne le nombre d'étudiants affectés à cette salle. Le nom de chaque fichier devra être de la forme `emargement_i.txt`, où `i` est l'indice de la sous-liste associée à la salle `i`.
4. À l'issue de l'affectation, il est possible que les premières salles soient pleines et la dernière presque vide (et même éventuellement d'autres complètement vides ensuite). Pour y remédier, proposer une nouvelle version de la fonction `affectation()` qui permet d'avoir une répartition homogène des places vides : entre deux salles quelconques les nombres restants de places vides diffèrent au plus d'une unité.

   Par exemple, s'il y a 10 étudiants à placer dans trois salles de capacités 5, 8 et 4, au lieu de remplir les deux premières salles de 5 et 5 étudiants (et de laisser la dernière vide), on les remplit respectivement de 2, 6 et 2 étudiants (laissant respectivement 3, 2 et 2 places vides).
5. Écrire le programme principal qui :
   — crée la liste `E` en récupérant les données se trouvant dans un fichier `"L1.txt"` dont chaque ligne correspond au nom et prénom d'un étudiant ;
   — demande à l'utilisateur les capacités de chacune des salles pour créer la liste `S` ;
   — crée les fichiers correspondant aux feuilles d'émargement des salles.

8. Portée des variables en Python (TD)

Exercice 8.1

Que font ces deux programmes et qu'affichent-ils ?

```
# Programme 1

def f(n):
    global c, p
    c = n+1
    return p*2

c = 0
p = 1

for i in range(10):
    n = f(i)
    print(c, n)
```

```
# Programme 2
def f(n):
    global c, p
    c = n+1
    p = p*2
    return p

c = 0
p = 1

for i in range(10):
    n = f(i)
    print(c, n)
```

Exercice 8.2

Que font ces deux programmes et qu'affichent-ils ?

```
# Programme 1
def gp():
    global p
    return p

def f(n):
    global c
    c = n+1
    p = gp()*2
    return p

c = 0
p = 1

for i in range(10):
    n = f(i)
    print(c, n)
```

```
# Programme 2
def gp():
    global p
    return p

def f(n):
    global c, p
    c = n+1
    p = gp()*2
    return p

c = 0
p = 1

for i in range(10):
    n = f(i)
    print(c, n)
```

Exercice 8.3 *(MASTERMIND)*

Le but de cet exercice est d'étudier un algorithme pour le jeu « Mastermind » afin de le programmer en TP.

Principes du jeu

Le Mastermind est un jeu à deux joueurs dans lequel un des deux joueurs doit deviner une combinaison secrète choisie par son adversaire. Pour ce faire, il va proposer lui-même des combinaisons et recevoir des indications de son adversaire suivant que cette combinaison est plus ou moins proche de la combinaison secrète.

- Le programme choisit une combinaison secrète formée de n chiffres compris entre 1 et b inclus.
- Le joueur peut alors à chaque tour proposer lui-même une combinaison de n chiffres.
- L'ordinateur indique alors à chaque fois au joueur combien de chiffres sont « bien placés » (c'est-à-dire que le même chiffre est situé au même endroit dans les deux combinaisons) et combien de chiffres sont « mal placés » (le même chiffre est alors placé à deux endroits différents dans la combinaison secrète et celle testée).

L'objectif du joueur est bien sûr de trouver la combinaison secrète (c'est-à-dire d'avoir n chiffres bien placés) en un minimum d'essais (ou moins qu'un nombre maximum d'essais possibles).

Voici un exemple de partie, la combinaison secrète est formée de 4 chiffres entre 1 et 5 inclus.

```
Saisir un nombre de 4 chiffres :
1234
Bien places : 0
 Mal places : 3
Il vous reste 9 essais.
Saisir un nombre de 4 chiffres :
5423
Bien places : 2
 Mal places : 1
Il vous reste 8 essais.
Saisir un nombre de 4 chiffres :
5412
Bien places : 0
 Mal places : 2
Il vous reste 7 essais.
Saisir un nombre de 4 chiffres :
4323
Bien places : 4
 Mal places : 0
Bravo !
```

La combinaison secrète et celle testée peuvent contenir des doubles. Dans ce cas, chaque chiffre ne peut servir que pour une seule correspondance, les chiffres bien placés étant prioritaires. Voici quelques exemples :

`2122` `1333`	1 mal placé
`1122` `1333`	1 bien placé
`1212` `3133`	1 mal placé
`1222` `1133`	1 bien placé
`1222` `3113`	1 mal placé
`1122` `3113`	1 bien placé et 1 mal placé

Algorithmique

Les parties « génération de la combinaison secrète » et « récupération de la combinaison à tester » seront traitées directement en TP. Nous allons ici nous intéresser à l'algorithme permettant de calculer les réponses à donner au joueur.

On supposera dans la suite que l'on dispose de deux tableaux `Secr[]` et `Test[]` de taille n contenant respectivement les chiffres de la combinaison secrète et ceux de la combinaison proposée par le joueur.

1. Écrire en pseudo-code une fonction `bienPlaces()` qui prend les deux tableaux en paramètre et renvoie le nombre d'éléments bien placés.
2. Proposer une fonction en pseudo-code qui prend les deux tableaux en paramètre et renvoie le nombre d'éléments mal placés.

   Pensez-vous que votre code fonctionne en présence de doubles ? Donne-t-il le bon résultat si la combinaison est 1231 et le test 4115 ?

   Pour éviter de compter un chiffre plusieurs fois, une fois qu'une correspondance est établie, il faut « éliminer » le chiffre en le transformant en une valeur ne pouvant pas donner de correspondance. Écrire une nouvelle fonction `malPlaces()` en pseudo-code qui prend les deux tableaux en paramètre et renvoie le nombre d'éléments mal placés. Cela fonctionne-t-il avec les combinaisons 1234 et 2516 ?
3. Suffit-il d'appeler dans l'ordre les deux fonctions `bienPlaces()` et `malPlaces()` pour avoir les deux résultats à afficher ? Que se passe-t-il avec les combinaisons 1234 et et 5226 ?

8. Portée des variables en Python (TP)

Exercice 8.4 *(Portée des variables)*

1. Écrire en Python une fonction `echange(a,b)` qui échange les valeurs des 2 paramètres `a` et `b`. Cette fonction ne renvoie aucune valeur.
2. Tester votre fonction avec les instructions suivantes et expliquer ce qu'il se passe :

```
a = 2
b = 3
echange(a, b)
print("a=", a)
print("b=", b)
```

3. Modifier votre fonction afin qu'elle renvoie les 2 valeurs de `a` et `b` après les avoir échangées. Tester alors les instructions suivantes :

```
a = 2
b = 3
a, b = echange(a, b)
print("a=", a)
print("b=", b)
```

4. Écrire une fonction `echangeListe(L)` qui prend en argument une liste contenant deux éléments et échange leur position. Cette fonction ne renvoie rien. Tester les instructions suivantes :

```
L = [2,3]
echangeListe(L)
print(L)
```

5. Tester le programme suivant :

```
def echangeAB():
    global a
    global b
    tmp = a
    a = b
    b = tmp
a = 2
b = 3
echangeAB()
print("a=", a)
print("b=", b)
```

Exercice 8.5 *(Mise en œuvre de l'exercice 8.3)*

Cet exercice a pour objectif de vous faire réaliser un petit « Mastermind » à l'aide de fonctions en Python.

1. Écrire une fonction `genere(n,b)` qui prend en paramètre une taille `n` et une borne maximale `b` et génère `n` chiffres aléatoires **compris entre 1 et `b` (inclus).** Les chiffres seront stockés dans une liste renvoyée par la fonction.
2. Écrire une fonction `enTableau(s)` qui transforme une chaîne de caractères `s` (composée de chiffres sans espace) en une liste d'entiers, en prenant chaque caractère de la chaîne comme élément de la liste. Cette fonction renvoie la liste d'entiers.

   Votre fonction devra par exemple permettre, lors de l'exécution des instructions suivantes :

   ```
   print("Saisir un nombre de", n, " chiffres : ")
   s = input()
   print(enTableau(s))
   ```

   d' afficher :

   ```
   Saisir un nombre de 5 chiffres : 34678
   [3, 4, 6, 7, 8]
   ```

3. Écrire une fonction `copie(source)` qui renvoie une nouvelle liste contenant les mêmes éléments que la liste source.

   Remarque : attention, simplement exécuter l'instruction `y=x`, quand les deux variables sont des listes, implique que toute modification sur `x` sera répercutée sur `y` ; `x` et `y` correspondent en fait physiquement à la même liste. Par exemple :

   ```
   >>> x = [1,2,3]
   >>> y = x
   >>> print(x)
    [1, 2, 3]
   >>> print(y)
    [1, 2, 3]
   >>> x.append(4)
   >>> print(y)
    [1, 2, 3, 4]
   ```

4. En vous aidant éventuellement des deux fonctions vues en TD (`bienPlaces()` et `malPlaces()`), écrire une fonction `verification(sol, proposition)` qui affiche combien il y a d'éléments bien placés et combien il y a d'éléments mal placés dans la liste `proposition` par rapport à la liste `sol`. La fonction devra, en plus d'afficher ces nombres, les renvoyer.

   Par exemple, l'exécution `verification([1,2,3,4],[2,8,3,6])` devra afficher :

   ```
   Nombre d'éléments bien placés : 1
   Nombre d'éléments mal placés : 1
   ```

   Attention de ne pas compter plusieurs fois le même élément, car chaque chiffre de la proposition et de la solution ne peut être utilisé que pour une seule correspondance. Pour cela, vous pouvez, comme vu dans l'algorithme de TD, remplacer l'élément testé

par une valeur spéciale dans la liste `sol` et par une autre dans la liste `proposition`, en ayant préalablement fait une copie des listes grâce à la fonction `copie()` définie précédemment.

5. Écrire une fonction `mastermind(n,b,nb_essai)`, qui prend en paramètre une taille `n`, une borne maximale `b` et un nombre maximum d'essais `nb_essai` et appelle les trois fonctions précédentes pour exécuter un Mastermind.
   Par exemple l'appel de cette fonction `mastermind(5,6,5)` pourrait afficher :

```
Saisir un nombre de 5 chiffres (entre 1 et 6) : 11111
Nombre d'éléments bien placés : 2
Nombre d'éléments mal placés : 0
Il vous reste 4 essais.

Saisir un nombre de 5 chiffres (entre 1 et 6) : 11222
Nombre d'éléments bien placés : 0
Nombre d'éléments mal placés : 4
Il vous reste 3 essais.

Saisir un nombre de 5 chiffres (entre 1 et 6) : 11223
Nombre d'éléments bien placés : 0
Nombre d'éléments mal placés : 4
Il vous reste 2 essais.

Saisir un nombre de 5 chiffres (entre 1 et 6) : 41212
Nombre d'éléments bien placés : 1
Nombre d'éléments mal placés : 4
Attention dernier essai!!

Saisir un nombre de 5 chiffres (entre 1 et 6) : 21142
Nombre d'éléments bien placés : 2
Nombre d'éléments mal placés : 3

Dommage!
La solution était : [2, 2, 1, 1, 4]
```

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

9. Récursivité (exercices de TD)

Exercice 9.1

1. Écrire la fonction récursive qui permet d'obtenir la valeur de la suite U_n, pour un n donné en argument, et définie par :

$$\begin{cases} U_0 &= 10 \\ U_n &= 0.2U_{n-1} + 3 \end{cases}$$

2. Écrire cette même fonction mais de façon itérative.

Exercice 9.2

Simuler l'exécution de la fonction récursive `mystere` ci-dessous en lui mettant comme paramètre 36 et 15.

```
def mystere(a,b):
    if (a == 0 or b == 0):
        return 0
    elif a == b:
        return a
    elif a > b:
        return mystere(a-b, b)
    else:
        return mystere(a, b-a)
```

Exercice 9.3

1. On considère la fonction :

```
def concatene(L):
    if len(L) > 0:
        return L[0] + " " + concatene(L[1:])
    else:
        return "."
```

qui est appelée de la manière suivante :

```
concatene(["j'","apprends","la","programmation","récursive"])
```

Dessiner la suite des appels récursifs et indiquer la valeur renvoyée par la fonction suite à cet appel.

2. Que se passe-t-il si l'on appelle la fonction suivante avec la même liste en argument ?

```
def concatene2(L):
    if len(L) > 0:
        return concatene2(L[1:]) + L[0] + " "
    else:
        return "."
```

Exercice 9.4

Nous allons créer une fonction `Produit(a,b)` qui effectuera le produit (la multiplication) de deux nombres entiers naturels (c'est-à-dire positifs) a et b. Facile me direz-vous. Sauf que nous ajoutons une difficulté : il sera interdit d'utiliser le symbole *. Vous devrez donc implémenter cette fonction uniquement avec des additions et de manière récursive.

Ecrire une fonction **récursive** en Python `Produit(a,b)` qui prend en paramètre deux entiers a et b et renvoie comme résultat la multiplication de ces deux nombres. Nous supposerons que l'opérateur $*$ n'est pas défini : la fonction ne pourra calculer ce résultat qu'à l'aide d'additions.

Exercice 9.5

Ci-dessous une autre version de la fonction récursive qui renvoie la liste dans laquelle toutes les occurrences de l'éléments x auront été supprimées.

```
def sansOccu2(L,x):
    if L != []:
        if L[0] == x:
            L = sansOccu2(L[1:],x)
        else:
            L = [L[0]] + sansOccu2(L[1:],x)
    return L
```

1. Que renvoie cette fonction pour $L = []$?
2. Simuler son appel pour `L=[1,3,3,7,3,5]` et `x=3`, et sur `L=[1,3,5,7]` et `x=5`. Si on exécute cette fonction sur L, initialisée dans le programme principal, à la fin de son exécution, L est-elle modifiée ?

Exercice 9.6

1. Sans utiliser de méthode de chaîne, écrire une fonction récursive qui permet de remplacer toutes les occurrences du caractère x dans une chaîne ch par le caractère y. Cette fonction `remplaceOccu(x,ch,y)` aura pour paramètres les chaînes de caractères `x`, `ch` et `y` et renverra la chaîne modifiée.
2. Simuler l'appel de votre fonction avec `ch='papa'`, `x='a'` et `y='e'`.

Exercice 9.7

Les n premiers nombres de Fibonacci sont définis par :

$$Fib(0) = Fib(1) = 1,\ Fib(i) = Fib(i-1) + Fib(i-2)\ \texttt{si}\ \ i \geq 2$$

1. Écrire en pseudo-code une fonction itérative renvoyant le n^{e} nombre de Fibonacci.
2. Écrire en pseudo-code une fonction récursive renvoyant le n^{e} nombre de Fibonacci.
3. Dessiner l'arbre des appels récursifs de la fonction précédente appelée avec l'argument 5.

Exercice 9.8

L'algorithme d'Euclide pour calculer le Plus Grand Commun Diviseur (PGCD) de deux entiers a et b (avec $a \geq b$) consiste à effectuer une suite de divisions euclidiennes :

— étape 1 : effectuer la division euclidienne de a par b ; on obtient r le reste ; si r est égal à 0, alors fin et le PGCD est égal à b, sinon aller à l'étape 2,

— étape 2 : a reçoit alors la valeur de b, b reçoit la valeur de r et aller à l'étape 1.

Écrire en pseudo-code une fonction récursive appelée `pgcd()`, qui prend deux arguments `a` et `b` (avec `a`$\geq$`b`) et qui renvoie le PGCD de `a` et `b`.

Exercice 9.9

1. Écrire en pseudo-code une fonction prenant un entier a et un entier positif n et qui renvoie a^n de manière itérative.
2. Écrire une fonction récursive calculant cette même puissance.
3. Sachant que $a^0 = 1$ et

$$a^n = \begin{cases} (a^{(n/2)})^2 & \text{si } n \text{ est pair} \\ a \times (a^{(n-1)/2)})^2 & \text{si } n \text{ est impair} \end{cases}$$

écrire une deuxième fonction récursive plus maligne calculant cette même puissance.

Exercice 9.10

Étant donné le programme Python ci-dessous :

```
def fac_rec(n):
    global nbPassagesFonction
    nbPassagesFonction = nbPassagesFonction+1
    print("n =", n)
    if (n == 1 or n == 0):
       return 1
    else:
       return n*fac_rec(n-1)

nbPassagesFonction = 0
i = int(input("i="))
print("Résultat de " + str(i) + "! =", fac_rec(i))
print("Nbr de passages dans la fonction :",nbPassagesFonction)
```

1. Qu'affiche le programme si l'utilisateur saisit 5 comme valeur pour `i` ?
2. Écrire la version itérative de la fonction `fac_rec(n)`.

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

9. Récursivité (TP)

Exercice 9.11 *(Debugage lié à l'interpréteur)*

1. Récupérer le fichier `Debug_reinit_1.py` se trouvant dans le répertoire `ETUD/DEMI2E_L1_AlgoProg/` et copier-le dans votre répertoire personnel.
   Ce programme est supposé afficher une liste de 5 nombres aléatoires pris entre 0 et 10 inclus.
   Exécuter le programme plusieurs fois ; vous semble-t-il correct ? Corriger si nécessaire.
2. On décide maintenant de mettre le code dans une fonction et on écrit le programme suivant :
   Récupérer le fichier `Debug_reinit_2.py` se trouvant dans le répertoire `ETUD/DEMI2E_L1_AlgoProg/` et copier-le dans votre répertoire personnel.
   Exécuter le programme une fois ; vous semble-t-il fonctionner ?
   Exécuter le programme plusieurs fois. Que remarquez-vous ?
   Redémarrer l'interpréteur et relancer le programme. Pourquoi y a–t-il cette erreur seulement maintenant ? Corrigez-là.

Exercice 9.12

1. Écrire une fonction récursive, appelée `sommeRec()`, qui prend comme argument une valeur entière n, et qui calcule la somme des n premiers entiers. Appeler cette fonction pour qu'elle calcule la somme des entiers allant de 0 à 100.
2. Transformer la fonction `sommeRec()` afin qu'elle admette 2 arguments `debut` et `fin`, et qu'elle calcule la somme des entiers variant de la valeur entière `debut` à la valeur entière `fin`. Appeler cette fonction pour calculer la somme des entiers variant de 50 à 100.

Exercice 9.13 *(Mise en œuvre de l'exercice 9.7)*

Les n premiers nombres de Fibonacci sont définis par :

$$Fib(0) = Fib(1) = 1,\ Fib(i) = Fib(i-1) + Fib(i-2)\ \texttt{si}\ i \geq 2$$

1. Programmer la fonction itérative `FibIt()` renvoyant le n^{e} nombre de Fibonacci.
2. Programmer la fonction récursive `FibRec()` renvoyant le n^{e} nombre de Fibonacci.
3. A l'aide de la bibliothèque `time` et de la fonction `clock()`, comparer les temps de calcul de `FibIt(n)` et `FibRec(n)` pour `n=35` puis `n=70`. Que constatez-vous ?

Exercice 9.14 *(Mise en œuvre de l'exercice 9.9)*

1. Écrire une fonction prenant un entier a et un entier positif n et qui renvoie a^n de manière itérative.
2. Écrire une fonction récursive calculant cette même puissance.
3. Sachant que $a^0 = 1$ et

$$a^n = \begin{cases} (a^{(n/2)})^2 & \text{si } n \text{ est pair} \\ a \times (a^{(n-1)/2)})^2 & \text{si } n \text{ est impair} \end{cases}$$

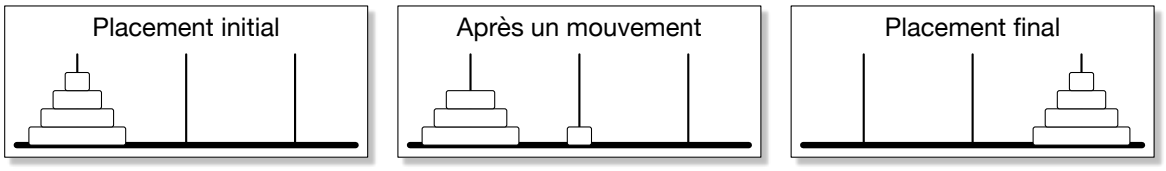
   écrire une deuxième fonction récursive plus maligne calculant cette même puissance.
4. Afficher les puissances de 2 de 2^0 à 2^{30} en appelant les deux fonctions récursives ci-dessus. On affichera également pour chaque puissance le nombre d'appels à chaque fonction, comptabilisés à l'aide d'une variable globale.

Exercice 9.15

Un exemple classique : les tours de Hanoï

On dispose de trois tiges et une "tour" de n disques de diamètres différents est enfilée sur une des tiges. Il faut déplacer la tour de la première à la troisième tige en respectant deux règles :

1. un seul disque peut être déplacé à la fois ;
2. un disque ne peut être posé que sur une tige vide ou un autre disque de diamètre plus grand.



Il existe des algorithmes de résolution itératifs mais une solution récursive est plus naturelle.

Réflexion

Pour déplacer le plus grand disque (qui se trouve tout en dessous) de la tige 1 à la tige 3, il faut :

— que l'ensemble des autres disques ait été déplacé (pour libérer le plus grand) ;

— que la tige 3 soit vide (pour pouvoir l'y mettre).

Autrement dit, il faut que la tour des $n-1$ autres disques ait été déplacée de la tige 1 à 2. Cela nous montre que la solution pour déplacer une tour de n disques passe par le déplacement de la tour des $n-1$ plus petits disques.

Algorithme de résolution

L'algorithme récursif pour déplacer une tour de n disques d'une tige source à une tige destination est donc le suivant (voir une illustration figure 1) :

1. déplacer la tour formée des $n-1$ disques supérieurs de la tige source à la tige intermédiaire ;
2. déplacer le plus grand disque de la tige source à la tige destination ;
3. déplacer la tour des $n-1$ disques de la tige intermédiaire à la tige destination.

L'objectif de cet exercice est de programmer la résolution des Tours de Hanoï à l'aide d'une fonction récursive.

Les trois tiges sont représentées par trois listes `t1`, `t2` et `t3`, et les disques par des entiers donnant leur diamètre. Ainsi, la liste `[4,3,2,1]` correspond à une tige de 4 disques, tandis qu'une liste vide correspond à une tige vide.

1. À quelle méthode Python correspond l'action d'enlever le premier disque d'une tige ?
2. À quelle méthode Python correspond la pose d'un disque sur une tige ?
3. Écrire une fonction à 4 paramètres, un entier `n` et 3 listes `src`, `dest` et `inter`, qui déplace une tour de `n` disques se trouvant sur la tige `src` vers la tige `dest` en utilisant la tige `inter`. Elle le fera en implémentant l'algorithme récursif vu en cours.
4. Écrire le programme principal qui définit les trois tiges d'un puzzle de Hanoï à 4 disques et affiche l'état des trois tiges avant et après appelle de la fonction précédente.

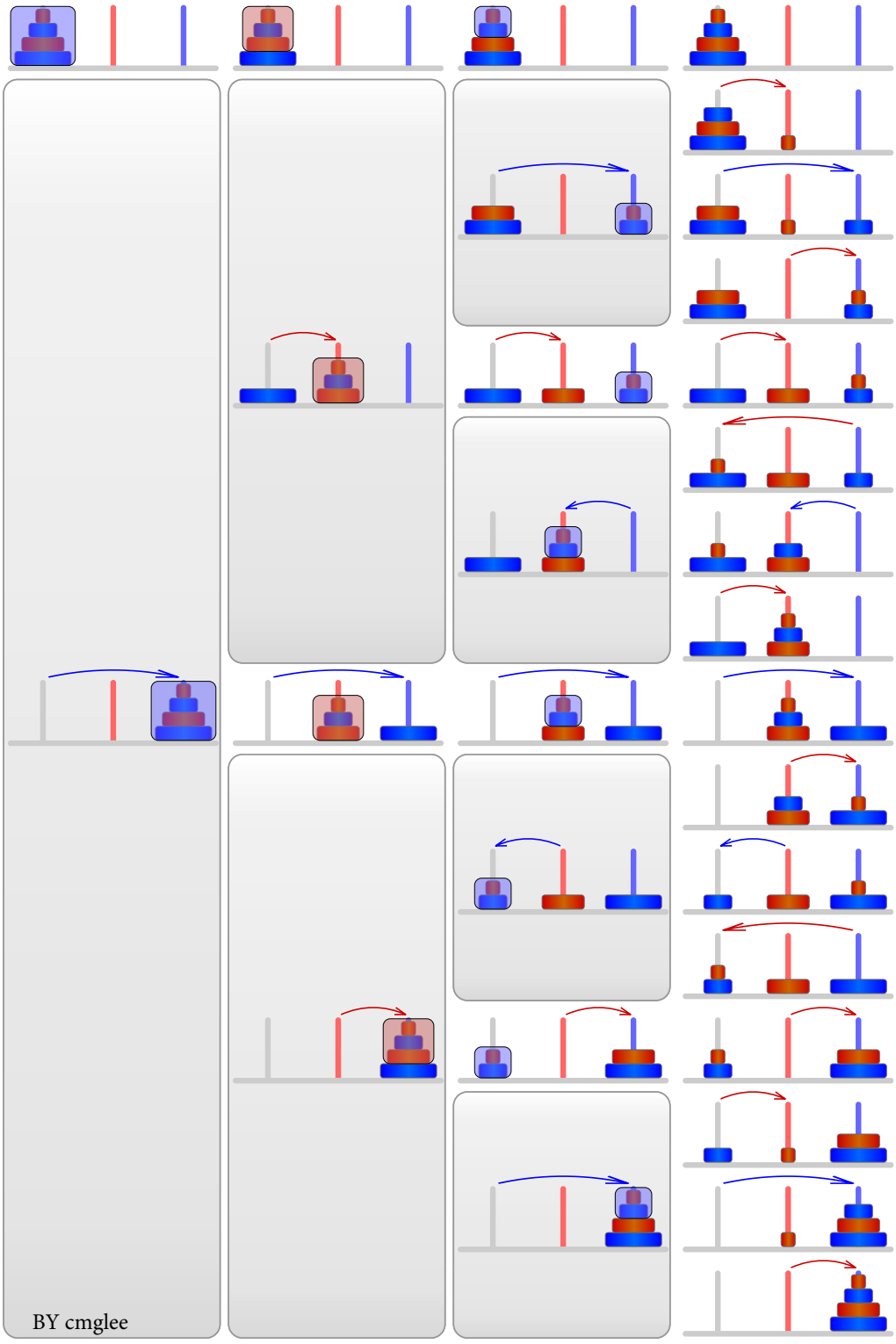


FIGURE 1 – Décomposition récursive des tours de Hanoi

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

10. Tuples et dictionnaires (exercices de TD)

Exercice 10.1

Expliquer ce que fait le programme suivant :

```
def mystere(a,b):
    return b,a

x = 5
y = 12
print("Avant appel à mystere() : x=", x, " y=", y)
x, y = mystere(x,y)
print("Apres appel à mystere() : x=", x, " y=", y)
```

Exercice 10.2

Soit[6] le dictionnaire suivant :

```
d = {'nom'   : 'Dupuis',
     'prenom': 'Jacque',
     'age'   : 30}
```

Écrire en Python les instructions permettant de :

1. corriger l'erreur dans le prénom, la bonne valeur devant être `'Jacques'` ;
2. afficher les différentes clés du dictionnaire ;
3. afficher les différentes valeurs du dictionnaire ;
4. afficher les différentes paires clé/valeur du dictionnaire ;
5. afficher la phrase `"Jacques Dupuis a 30 ans"` à l'aide des éléments du dictionnaire.

Exercice 10.3

Que va afficher le programme suivant :

```
dico = {"dog"   : "chien",
        "cat"   : "chat",
        "mouse" : "souris"}
dico2 = {}
for k in dico:
    dico2[dico[k]] = k

print(dico2)
```

6. Exercice repris et adapté de `https://www.boureliou.com/cnam/xhtml/python2-15-exercices.html`

Exercice 10.4

On souhaite inverser un dictionnaire, c'est-à-dire permuter la clé et la valeur pour chaque couple.

1. On considère un dictionnaire où chaque clé et chaque valeur sont des chaînes. On supposera dans cette question que toutes les clés sont distinctes (obligatoire pour un dictionnaire) mais également que toutes les valeurs sont distinctes.

   Écrire en **Python** une fonction `inverse()` qui prend un dictionnaire en paramètre et renvoie un nouveau dictionnaire dans lequel tous les couples (clé,valeur) auront été inversés.

   Par exemple `inverse({'a':'zz', 'b':'yy' })` doit renvoyer `{'zz': 'a', 'yy': 'b'}`

2. Toujours avec un dictionnaire de chaînes, on suppose maintenant que les valeurs ne sont pas forcément distinctes. Les valeurs du nouveau dictionnaire (associées à une clé `V`) doivent alors être une liste de toutes les clés du dictionnaire initial qui avaient `V` comme valeur. Écrire en **Python** une fonction `inverse2()` qui prend un dictionnaire en paramètre et renvoie un nouveau dictionnaire correspondant.

   Par exemple `inverse2({'a':'zz', 'b':'yy', 'c':'zz'})` doit renvoyer `{'zz': ['a', 'c'], 'yy': ['b']}`

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

10. Dictionnaires et tuples (TP)

Exercice 10.5

Écrire [7] en Python une fonction `calculFiboAvecTuple()` qui prend en paramètre 2 tuples (a, b), (u_0, u_1) et 1 entier n, et renvoie le terme u_n de la suite définie par :

$$u_n + au_{n-1} + bu_{n-2} = 0$$

Vérifier que l'appel à la fonction `calculFiboAvecTuple((-1,-1),(0,1),6)` renvoie bien la valeur 8.

Exercice 10.6

Écrire [8] en Python :

1. Une fonction qui prend en paramètre une chaîne de caractères et renvoie un dictionnaire mettant en correspondance chaque caractère apparaissant dans la chaîne avec le nombre de ses occurrences dans la chaîne.

   Par exemple, si la fonction prend en paramètre la chaîne `"toto"`, le dictionnaire renvoyé par la fonction aura pour valeur : `{'t': 2, 'o': 2}`.

2. Une fonction qui affiche le dictionnaire dans l'ordre alphabétique des lettres (on pourra utiliser la fonction `sorted()`).

   Par exemple, la fonction d'affichage, lorsqu'elle prend en paramètre le dictionnaire `{'t': 2, 'o': 2}`, va afficher :

   ```
   o 2
   t 2
   ```

7. Exercice repris et adapté de `http://yaspat.github.io/blog/posts/lecon-6-tuples-listes/`
8. Exercice repris et adapté de `www.i3s.unice.fr/~comet/SUPPORTS/Nice-MasterSVS-Python/TD3.pdf`

Exercice 10.7

Deux mots sont des anagrammes s'ils contiennent les mêmes lettres (avec le même nombre d'occurrences) dans un ordre qui peut être différent. Par exemple `"nacre"` et `"crane"` ou `"parisien"` et `"aspirine"` sont des anagrammes. On se propose de détecter les anagrammes de trois façons différentes.

1. (a) Écrire une fonction Python `compte()` qui prend en argument une chaîne de plusieurs caractères et une chaîne contenant un seul caractère et renvoie le nombre d'occurrences du caractère de la deuxième chaîne dans la première.
   (b) Écrire une fonction Python `estAnagramme()` prenant deux chaînes de caractères en argument et renvoyant `True` si ces deux chaînes sont des anagrammes ou `False` dans le cas contraire ; on utilisera la fonction précédente.
   (c) Écrire le programme principal permettant à l'utilisateur de saisir deux mots et lui indiquant, en utilisant la fonction précédente, s'ils sont des anagrammes.
2. (a) Écrire une fonction Python `Dico()` qui prend en argument une chaîne de caractères et renvoie un dictionnaire mettant en correspondance chaque caractère apparaissant dans la chaîne avec le nombre de ses occurrences dans la chaîne.

   Par exemple, si la fonction prend en paramètre la chaîne `"toto"`, le dictionnaire renvoyé par la fonction aura pour valeur : `{'t': 2, 'o': 2}`.
   (b) Écrire le programme principal permettant à l'utilisateur de saisir deux mots et lui indiquant, en comparant les dictionnaires, s'ils sont des anagrammes.
3. La fonction `sorted()` s'applique également à une chaîne : elle renvoie une nouvelle chaîne dans laquelle les caractères de la chaîne passée en argument sont triés par ordre alphabétique. Proposer une écriture de la fonction `estAnagramme()` de la question 1b utilisant `sorted()`.

Exercice 10.8

Soit[9] le dictionnaire suivant :

```
dicoAnglaisFrancais = {"langage"    : "language",
                       "un"         : "a",
                       "est"        : "is",
                       "formidable" : "wonderful"}
```

Écrire une fonction Python `traduire(ch, dico)`, qui prend en paramètre une chaîne `ch` et un dictionnaire `dico`, et qui renvoie une nouvelle phrase obtenue par une traduction mot à mot de `ch` en utilisant le dictionnaire `dico`. Quand un mot de `ch` n'est pas trouvé dans le dictionnaire, il n'est pas traduit. Ainsi :
`traduire(("Python est formidable", dicoAnglaisFrancais)`
doit renvoyer la chaîne `"Python is wonderful"`.

9. Exercice repris et adapté de `https://www.irif.fr/~sangnier/enseignement/Ip1-Python/IP1-Python-cours-td-10.pdf`

Exercice 10.9

Après un examen, les notes des étudiants sont les suivantes :
Florian : abs, Antoine : 12, Clara : 15, Robert : 0, Agathe : 8, Erwan : 7, Zoé : 20, Xavier : 11, Didier : 20, Alain : 0, Hadi : 12.

1. Créer un dictionnaire `notes2016` pour stocker les noms des étudiants (les clés) et les notes associées (les valeurs).
2. Écrire une fonction permettant de supprimer du dictionnaire les informations relatives aux élèves marqués absent.
3. Écrire une fonction qui permet de calculer et d'afficher la moyenne de l'examen.
4. Écrire une fonction qui permet d'afficher le nom et la note de chacun des élèves, avec la mention "admis" si la note est supérieure ou égale à 10, ou bien "recalé" sinon.
5. Modifier la fonction pour qu'elle stocke dans deux listes distinctes, les noms des élèves ayant réussi leur examen et ceux des élèves ayant raté leur examen.
6. Écrire une fonction permettant de saisir et d'intégrer dans le dictionnaire les notes de nouveaux étudiants.

Exercice 10.10

Cet exercice vise à gérer une liste d'ingrédients pour une recette de gateau. Dans un premier temps, on va utiliser un dictionnaire, appelé `recette`, dont les clés sont les ingrédients possibles (à savoir "oeufs", "farine", "lait", "beurre", "sucre", "levure", "chocolat"), et les valeurs sont les tuples `(quantite, unite)` de chacun des ingrédients dans la recette.

1. Écrire une fonction `creeRecette()` qui crée le dictionnaire `recette` et lit les valeurs correspondantes à la recette suivante :
   Recette du moelleux au chocolat (pour 4 personnes)
   3 œufs
   100 g farine
   25 cL de lait
   150 g de beurre
   1 sachet de levure
   200 g de chocolat
   75 g de sucre
   et renvoie le dictionnaire `recette`.
2. Écrire une fonction, appelé `afficheRecette()`, qui permet d'afficher la recette à l'écran en respectant le format de la question précédente. Attention à bien gérer le cas particulier des ingrédients dont la quantité vaut 0 : si un ingrédient a une quantité de 0, il ne doit pas être affiché.
3. Maintenant, on souhaite afficher les quantités nécessaires pour réaliser une recette pour un nombre de personnes saisi par l'utilisateur.
   Écrire une fonction qui lit le nombre de personnes, calcule la quantité de chacun des ingrédients et appelle la fonction `afficheRecette()`, pour afficher la recette correspondant au bon nombre d'invités. Ne pas oublier qu'il faut afficher des nombres entiers.
4. On souhaite maintenant saisir une nouvelle recette et stocker toutes les recettes dans une variable de type liste de recette, appelé `livreCuisine`. Modifier votre programme Python pour que : il demande à l'utilisateur le nombre r de recettes qu'il souhaite saisir, appelle la fonction `creeRecette()` r fois pour initialiser `livreCuisine`, et appelle r fois la fonction `afficheRecette()` pour afficher toutes les recettes.

Exercice 10.11 *(Mise en œuvre de l'exercice 10.4)*

On souhaite inverser un dictionnaire, c'est-à-dire permuter la clé et la valeur pour chaque couple.

1. On considère un dictionnaire où chaque clé et chaque valeur sont des chaînes. On supposera dans cette question que toutes les clés sont distinctes (obligatoire pour un dictionnaire) mais également que toutes les valeurs sont distinctes.

   Écrire en **Python** une fonction `inverse()` qui prend un dictionnaire en paramètre et renvoie un nouveau dictionnaire dans lequel tous les couples (clé,valeur) auront été inversés.

   Par exemple `inverse({'a':'zz', 'b':'yy' })` doit renvoyer `{'zz': 'a', 'yy': 'b'}`

2. Toujours avec un dictionnaire de chaînes, on suppose maintenant que les valeurs ne sont pas forcément distinctes. Les valeurs du nouveau dictionnaire (associées à une clé `V`) doivent alors être une liste de toutes les clés du dictionnaire initial qui avaient `V` comme valeur. Écrire en **Python** une fonction `inverse2()` qui prend un dictionnaire en paramètre et renvoie un nouveau dictionnaire correspondant.

   Par exemple `inverse2({'a':'zz', 'b':'yy', 'c':'zz'})` doit renvoyer `{'zz': ['a', 'c'], 'yy': ['b']}`

3. On revient au premier cas, avec un dictionnaire où toutes les clés et valeurs sont distinctes. On suppose que ce dictionnaire est stocké dans un fichier où chaque ligne donne un couple du dictionnaire. Ainsi, le dictionnaire `{'a':'zz', 'b':'yy', 'c':'xx'}` sera stocké dans le fichier sous la forme :

   ```
   a:zz
   b:yy
   c:xx
   ```

   Écrire une fonction **Python** `lireDico()` qui prend un nom de fichier en argument et renvoie un dictionnaire formé de tous les couples se trouvant dans le fichier.

4. Écrire une fonction **Python** `ecrireDico()` qui prend en argument un dictionnaire et un nom de fichier et écrit le dictionnaire dans un fichier de la manière indiquée à la question précédente.

5. Écrire le programme principal en **Python** qui :

   (a) lit le dictionnaire stocké dans le fichier `"dico.txt"` ;

   (b) l'inverse dans un nouveau dictionnaire ;

   (c) écrit ce nouveau dictionnaire dans le fichier `"sortie.txt"`.

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

A. Gestion de fichiers (TP) [10]

Exercice A.1 *(Debugage d'un exercice sur les chaînes)*

Nous cherchons à compter le nombre de mots d'une chaine et le nombre de caractères par mot. Pour ce faire, l'équipe Python a écrit une fonction mais malheureusement cette dernière ne fonctionne pas ! Vous allez devoir la corriger et la faire fonctionner.

Pour récupérer le fichier, aller sur le répertoire `ETUD/DEMI2E_L1_AlgoProg/` et copier, dans votre répertoire personnel, le fichier `Debug_Chaine.py`.

1. Exécuter le programme. Que vous indique l'erreur de syntaxe ? Corriger cette erreur.
2. Exécuter à nouveau le programme. Que vous indique l'erreur de syntaxe ? Corriger cette erreur.
3. Appeler la fonction en lui passant en paramètre la chaîne `"ananas poire kiwi"`. Quel est le message d'erreur ? Corriger cette erreur.
4. Appeler à nouveau la fonction (après avoir ré-exécuté le programme corrigé précédemment) en lui passant en paramètre la chaîne `"ananas poire kiwi"`. Quel est le message d'erreur ? Corriger cette erreur.
5. Appeler à nouveau la fonction (après avoir ré-exécuté le programme corrigé précédemment) en lui passant en paramètre la chaîne `"ananas poire kiwi"`. Pourquoi selon vous le programme tourne-t-il indéfiniment ? Quelle boucle pose problème selon vous ? Insérer des affichages pour le vérifier et corriger l'erreur.
6. Appeler à nouveau la fonction en lui passant en paramètre la chaîne `"ananas poire kiwi"`. Que signifie le message d'erreur ? Corriger le programme pour que ce message n'apparaisse plus.
7. Que se passe-t-il si vous inversez les conditions de la 2ème boucle `while`. Expliquer.

10. La partie « fichiers » de ce cours est largement reprise du livre de Gérard Swinnen, `fr.wikibooks.org/wiki/Apprendre_à_programmer_avec_Python/Les_fichiers`

8. Que faut-il modifier dans le programme pour que celui-ci affiche la description de tous les mots et pas uniquement du dernier ? Par exemple, l'appel doit afficher :

```
La chaine "ananas poire kiwi" contient : 3 mots
Le mot "ananas" contient :
  3 fois le caractère 'a'
  2 fois le caractère 'n'
  1 fois le caractère 's'
Le mot "poire" contient :
  1 fois le caractère 'p'
  1 fois le caractère 'o'
  1 fois le caractère 'i'
  1 fois le caractère 'r'
  1 fois le caractère 'e'
Le mot "kiwi" contient :
  1 fois le caractère 'k'
  2 fois le caractère 'i'
  1 fois le caractère 'w'
```

A.1 Travailler avec des fichiers

Jusqu'à présent, vous avez appris à lire des données en entrée d'un programme Python lorsqu'elles sont saisies au clavier par un utilisateur et à écrire les résultats d'un programme en les affichant à l'écran. Un programme Python peut également lire les données en entrée alors qu'elles sont stockées dans un fichier et écrire les résultats dans un autre fichier. Un fichier n'est autre qu'une suite d'informations enregistrée de manière permanente sur le disque. Ceci va nous permettre de garder les données en entrée et résultats de manière permanente.

Nous allons apprendre à lire et écrire dans des fichiers textes et ainsi séparer les données et les programmes qui les traitent dans des fichiers différents. Il faudra ajouter dans nos programmes des mécanismes permettant de créer des fichiers, d'y envoyer des données et de les récupérer par la suite. Quand les données deviennent trop importantes (en quantité) ou trop complexes (en structure), il devient nécessaire de structurer les relations entre ces données, on parle alors de bases de données (pas l'objet de ce cours, mais sachez que Python est capable de dialoguer avec des bases de données).

L'utilisation d'un fichier ressemble beaucoup à celle d'un livre : pour utiliser un livre il faut d'abord le trouver (à l'aide de son titre), puis l'ouvrir, et lorque vous avez terminé, vous le refermez. Tant qu'il est ouvert, vous pouvez y lire des info diverses et vous pouvez aussi éventuellement y écrire, mais en général pas les 2 à la fois. Dans tous les cas, on peut se situer dedans grâce aux numéros de page. Tout cela est un peu similaire pour les fichiers qui sont composées de données enregistrés sur n'importe quel support de mémorisation : disque dur, clé USB, CD/DVD...On y accède grâce à son nom et si on connaît sa place (cf section qui suit) et ensuite nous verrons comment accéder à son contenu.

A.2 Où sont mes fichiers ?

Il existe deux moyens de désigner un fichier par son nom : indiquer son « nom absolu » (c'est-à-dire la suite des répertoires formant le chemin depuis la racine de l'arborescence des fichiers jusqu'au fichier lui-même) ou son « chemin relatif » depuis le répertoire courant.

Faire un rappel sur l'organisation de la mémoire d'un ordi : on partitionne un espace (appelé répertoire) en plusieurs sous-répertoires et/ou fichiers et c'est une définition qui peut se faire de façon récursive. Pour représenter les différents endroits, on utilise l'arborescence associée.

Exemple de nom absolu : `/users/elazard/Documents/essai.py`
C'est un nom absolu car il commence par le symbole `/` qui indique la racine.

Exemple de chemin relatif : `Documents/essai.py` (on est déjà positionné dans le répertoire `/users/elazard/`)

Lors du lancement de l'interpréteur, le répertoire courant est normalement initialisé à la racine du dossier personnel de l'utilisateur (par exemple `/users/elazard/`). Cela signifie que tout fichier uniquement désigné par son nom sera créé ou ouvert dans ce répertoire. Si l'on souhaite regrouper ses fichiers Python dans un sous-dossier, il est plus simple de désigner ce sous-dossier comme le répertoire courant.

On peut le faire une bonne fois pour toute dans l'interpréteur de commandes à l'aide de la fonction `chdir()` (signifiant *CHange DIRectory)* du module `os` :

```
>>> from os import chdir
>>> chdir("/users/lazard/Documents/python/")
#ou chdir("Documents/python/")
```

La première ligne permet d'utiliser la fonction `chdir()` du module `os`. le module `os` contient toute une série de fonctions permettant de dialoguer avec le système d'exploitation (os = Operating System). La seconde (ou troisième) ligne provoque le changement de répertoire. On peut indiquer le nouveau répertoire courant soit par son chemin absolu (exemple 1), soit par son chemin relatif à partir du répertoire courant précédent (exemple 2). Notons que, si besoin, la fonction `getcwd()` *(GET Current Working Directory)* du module `os` permet de connaître le répertoire courant. En Windows, changer les caractères `/` par `\\` et éventuellement ajouter au début du chemin absolu la lettre (svt C ou D) qui désigne le périphérique de stockage.

Une fois cette commande `chdir()` exécutée dans l'interpréteur, le répertoire courant est positionné à sa nouvelle valeur pour tous les prochains fichiers que l'on serait amené à utiliser. Il sera alors très facile de ne plus s'en occuper et, dans le code, de désigner les fichiers simplement par leur nom. Bien évidemment, si l'interpréteur venait à être redémarrer (par exemple par le troisième bouton de la barre de commande), il faudrait saisir à nouveau le changement de répertoire courant.

On peut aussi faire le changement de répertoire courant directement depuis un programme en incluant les deux lignes au début du code. Cela permet de rendre le programme indépendant (il n'y a plus besoin de faire ce changement au lancement dans l'interpréteur) mais le fixe à un endroit précis de l'arborescence (car sinon le chemin absolu indiqué dans `chdir()` risque de ne plus fonctionner).

Dans les noms de fichiers, mieux vaut être concis et en tous les cas bânir les caractères accentués.

Exercice A.2

Dans l'interpréteur :

1. demander quel est le répertoire courant
2. changer de répertoire pour vous positionner dans votre répertoire `'Documents'`
3. vérifier que vous êtes bien dans ce répertoire

4. changer de répertoire courant pour vous positionner dans le répertoire où vous sauvegardez votre TP de la séance 5
5. vérifier que vous êtes bien dans ce répertoire

A.3 Manipuler des fichiers texte

Dans un fichier texte (la convention est de les nommer avec l'extension `.txt`), toutes les informations sont écrites sous la forme de caractères imprimables (c'est-à-dire lettres, chiffres, symboles de ponctuation...) séparés par des espaces, des tabulations (`'\t'`) et des caractères de fin de ligne (`'\n'`). L'avantage de ces fichiers est qu'il est possible de les ouvrir dans n'importe quel logiciel affichant les caractères bruts du fichier (par exemple un éditeur de texte) : comme tous ces caractères sont imprimables, le texte du fichier est directement lisible à l'écran [11].

A.3.1 Écriture séquentielle dans un fichier

Pour écrire dans un fichier texte en Python, on doit exécuter les instructions suivantes :

1. Il faut tout d'abord créer un nouveau fichier (vide) et le référencer par une variable Python. La fonction Python `open()` permet de réaliser ces deux actions. L'instruction

   ```
   fEcriture = open('nomFichier.txt','w')
   ```

   crée une variable Python, appelée `fEcriture`, qui permet d'accéder au nouveau fichier nommé `nomFichier.txt` en mode `w` pour *write* (ou écriture). La fonction `open()` attend toujours deux arguments, qui doivent être des chaînes de caractères ou des variables contenant une chaîne. Le premier argument est le nom du fichier à ouvrir, et le second est le mode d'ouverture.
2. La méthode de fichier `write()` permet d'écrire dans un fichier. L'instruction

   ```
   fEcriture.write(ch)
   ```

   permet d'écrire la chaîne de caractères référencée par la variable `ch` dans le fichier référencé par `fEcriture`. La méthode `write()` attend un argument de type `str`.
3. La méthode `close()` permet de « fermer proprement » le fichier (ceci signale au système que l'on a fini de l'utiliser et qu'il peut valider et reporter les éventuelles données modifiées sur le disque). Cette méthode sauvegarde sur le fichier toutes les modifications faites depuis son ouverture : tant qu'elle n'est pas exécutée, rien n'est enregistré. Cette méthode ne prend aucun argument.

Notons bien que si le fichier `nomFichier.txt` existe déjà avant son ouverture en écriture par `open()`, alors toutes les données qu'il contenait sont perdues pour « laisser la place » aux nouvelles que l'on souhaite écrire.

Exemple :

```
>>> f = open("test.txt", "w")
>>> f.write("Bonjour\n")
>>> f.write("Ceci est un test d'ecriture")
>>> f.close()
```

11. Ce n'est pas le cas d'un fichier word par exemple. Il contient, en plus du texte, de nombreuses informations supplémentaires utilisées par le logiciel Word : polices de caractères du document, taille, largeur de page, illustrations... Il n'est donc pas possible d'afficher directement le contenu brut d'un fichier word à l'écran.

— La première ligne crée la variable `f`, laquelle fait référence à un fichier dont le nom est `"test.txt"` qui va être créé dans le répertoire courant (ou celui indiqué au préalable avec la méthode `chdir` du module `os`).
— Le premier appel à la méthode `write()` réalise l'écriture proprement dite de la chaîne `"Bonjour"` suivi d'un passage à la ligne dans le fichier. Le deuxième appel ajoute la chaîne `"Ceci est un test d'ecriture"` à la suite.
— La méthode `close()` referme le fichier et sauvegarde toutes les modifications faites. Celui-ci est désormais disponible pour tout usage. Vous pouvez maintenant ouvrir ce fichier avec un éditeur de texte.

Vous noterez que l'on ne peut écrire dans un fichier, avec la méthode `write()`, que des valeurs de type `str`. En conséquence, pour écrire une valeur numérique dans un fichier, il faut faire une conversion explicite de type vers `str` (ainsi `f.write(34)` provoque une erreur alors que `f.write(str(34))` permet d'écrire la chaîne de caractères `'34'` dans le fichier référencé par `f`).

Il existe une seconde méthode permettant d'écrire dans un fichier : `writelines()` qui elle prend en argument une liste ou un tuple dont chaque élément est une chaîne de caractères à écrire, il ne faut alors pas oublier à la fin de chaque élément de la liste (qui est une chaîne de caratères) de rajouter `\n` pour pouvoir revenir à la ligne.

```
f=open("test1.txt", "w")
f.writelines(["premiere ligne\n", "seconde ligne"+"\n", "FIN"])
f.close()
```

Exercice A.3

Écrire un programme qui demande à l'utilisateur son nom, puis crée un fichier dont le nom sera `"TestNom.txt"` (où *nom* est celui de l'utilisateur) dans lequel vous écrirez sur une première ligne `"Ceci est un test"` puis sur une seconde ligne `"Je m'appelle "` en ajoutant le nom de l'utilisateur et sur une dernière ligne vous écrirez `"Test termine"`. Pour cela vous utiliserez dans un premier temps la méthode `write()`.

Reprendre l'exercice mais cette fois avec la méthode `writelines()` ; pour ne pas écraser le fichier précédent vous appellerez le fichier `"TestBisNom.txt"`.

A.3.2 Lecture depuis un fichier

La lecture des données stockées dans un fichier se fait en suivant les trois étapes suivantes :

1. il faut tout d'abord « ouvrir » le fichier en mode lecture avec la fonction `open()`, en lui donnant comme deuxième argument la chaîne de caractères `'r'` (pour *read*) ;
2. on peut ensuite lire le contenu du fichier ; plusieurs méthodes sont alors disponibles : `read()`, `readline()`, `readlines()` ;
3. enfin le fichier doit être « fermé » par la méthode `close()`.

Exemple :

```
>>> fLecture = open("test.txt", "r")
>>> s = fLecture.read()
>>> fLecture.close()
>>> print(s)
 Bonjour
 Ceci est un test d'ecriture
```

— La fonction `open()` a comme premier argument le nom du fichier que l'on souhaite lire, ici `"test.txt"`, et comme deuxième argument, le mode d'ouverture, ici `'r'`.
— La méthode `read()` lit les données présentes dans le fichier et renvoie ces données sous la forme d'une valeur de type `str`. La totalité des caractères contenus dans le fichier est renvoyée par la méthode `read`. Ici, la variable `s` référence donc la valeur renvoyée par la méthode `read()`. Lorsqu'on exécute l'instruction `print(s)`, on constate bien que les données du fichier ont été lues et sont maintenant référencées par `s`.La méthode `read()` peut également être utilisée avec un argument, celui-ci indiquera le nombre de caractères à lire, à partir de la position déjà atteinte dans le fichier. S'il reste moins de caractères à lire dans le fichier, alors s'arrête à la fin. Si la fin du fichier est déjà atteinte, la méthode `read()` renvoie une chaîne vide.

Remarque : si le fichier ouvert en mode lecture n'existe pas, alors Python affiche un message d'erreur.

En plus de la méthode `read()`, il existe d'autres méthodes de lecture d'un fichier :

— la méthode `readline()` lit seulement une ligne du fichier (une ligne se termine par la caractère `'\n'` et se positionne alors au début de la ligne suivante ;

```
>>> f = open("test.txt","r")
>>> t = f.readline()
>>> f.close()
>>> print(t)
Bonjour
```

— la méthode `readlines()` lit le fichier dans sa totalité mais renvoie les données sous la forme d'une liste de chaînes, chaque ligne correspondant à un élément de la liste :

```
>>> f = open("test.txt","r")
>>> l = f.readlines()
>>> f.close()
>>> print(l)
 ['Bonjour\n', 'Ceci est un test d'ecriture\n']
```

La méthode `readlines()` permet donc de lire l'intégralité d'un fichier en une seule instruction, toute comme la méthode `read()`. Ceci n'est possible que si le fichier n'est pas trop gros car il est copié intégralement dans une variable (c'est-à-dire dans la mémoire vive de l'ordinateur). Si on traite de gros fichiers, il vaut mieux utiliser la méthode `readline()` dans une boucle.

Exercice A.4

1. Utiliser la méthode `read()` et afficher le contenu récupéré pour vérifier les informations contenues dans le fichier `"TestNom.txt"`.
2. Recommencer la question précédente mais en utilisant la méthode `readlines()`. Que trouvez-vous à la fin de chaque élément de la liste ? Utiliser la méthode de chaînes `strip()` pour supprimer tous les passages à la ligne.
3. Reprendre la question précédente mais en utilisant la méthode `readline()` dans une boucle et en utilisant une liste pour mémoriser le contenu de chaque ligne. Afficher les éléments de cette liste et supprimer les passages à la ligne.

Les opérations de lecture/écriture se font de manière séquentielle, le « curseur » se positionnant toujours à la fin des données lues ou écrites. Il est cependant possible de naviguer à l'intérieur d'un fichier et de positionner le curseur à un endroit quelconque des données en spécifiant son décalage (en nombre de caractères) par rapport à l'origine du fichier à l'aide de la méthode `seek()`. La fonction `next(f)` permet également de passer à la ligne suivante lors de la lecture du fichier `f`.

Exercice A.5

1. Vérifier que votre fichier `"TestNom.txt"` a bien été créé et contient les informations souhaitées.
2. Utiliser la fonction `next()` pour lire et afficher à l'écran la seconde ligne de votre fichier créé précédemment sous le nom `"TestBisNom.txt"`.

Il existe également un autre mode d'ouverture d'un fichier avec la fonction `open()` : le mode ajout (`'a'`) qui permet d'écrire dans un fichier existant des données supplémentaires.

```
>>> fichier = open("test.txt", "a")
>>> fichier.write("\nFIN")
>>> fichier.close()
```

Le fichier `test.txt` contient alors :

```
Bonjour
Ceci est un test d'ecriture
FIN
```

A.4 Exercices sur les fichiers

Exercice A.6

Écrire un programme qui génère automatiquement un fichier texte contenant les tables de multiplication de 2 à 30 (chacune d'entre elles incluant 20 termes seulement).

Exercice A.7

Créer un fichier, appelé `premier.txt`, avec un éditeur de texte (par exemple *gedit*) contenant les données suivantes :

```
Voici un fichier texte cree en
2017
a l'Universite Paris-Dauphine
```

1. Écrire une fonction `lecture()` qui lit le contenu d'un fichier, dont le nom est donné en argument, et l'affiche à l'écran.
2. Appeler cette fonction pour lire le fichier `premier.txt`.
3. Écrire une fonction `copieFichier(source, destination)` qui recopie tout le contenu du fichier nommé `source` dans un fichier nommé `destination`.
4. Appeler cette fonction pour recopier le fichier `premier.txt` dans un nouveau fichier `premierCopie.txt`.
5. Écrire une fonction `copieFichierMiseAJour(source,destination)` qui recopie tout le contenu du fichier nommé `source` dans un fichier nommé `destination` après avoir remplacé toutes les occurrences de 2017 par 2018.
6. Appeler cette fonction pour recopier le fichier `premier.txt` dans un nouveau fichier `premierCopieMiseAJour.txt`.
7. Vérifier avec un éditeur de texte que `premierCopieMiseAJour.txt` existe et est conforme à ce qui a été demandé.
   Appeler la fonction `lecture()` pour lire le fichier `premierCopie.txt`.

Exercice A.8

Écrire un programme qui recopie le fichier `premier.txt` crée à l'exercice A.7 en triplant tous les espaces entre les mots.

Exercice A.9

Un fichier texte est mis à disposition dont chaque ligne est la représentation d'une valeur numérique de type réel (mais sans exposants). Par exemple :

```
14.896
7894.6
123.278
```

Écrire un programme qui recopie ces valeurs dans un autre fichier en les arrondissant au nombre entier le plus proche (on pourra utiliser la fonction `round()`).

Exercice A.10 *(À faire une fois les listes vues en cours)*

Créer un fichier nommé "`NotesGr1.txt`" qui contiendra les notes d'algorithmique des étudiants du groupe 1 de première année. Le format est le suivant : chaque ligne correspond à un étudiant ; elle commence par son nom, suivi d'une tabulation, puis son prénom, et une tabulation précédera la note de l'étudiant (valeur décimale entre 0 et 20). Vous créerez quelques étudiants dans ce fichier en ne respectant aucun ordre particulier, ni alphabétique ni en fonction de la note.

1. Écrire une fonction `lectureFichier(nom)` qui ouvre le fichier texte dont le nom est passé en argument et renvoie une liste composée de toutes les lignes du fichier.
2. Écrire une fonction `ecritureFichier(nom, liste)` qui ouvre le fichier dont le nom est passé en argument et y écrit la liste de chaînes passée en paramètre, un élément par ligne.
3. Écrire une fonction `triFichier()` qui lit le fichier nommé "`NotesGr1.txt`" et crée un nouveau fichier "`NotesGr1_trie.txt`" qui contient les notes de tous les étudiants suivant le même format mais trié par ordre alphabétique de nom. On pourra bien sûr utiliser les deux fonctions précédemment définies.
4. Généraliser la fonction précédente en écrivant une fonction `triNFichiers(N)` qui lit N fichiers "`NotesGr1.txt`" à "`NotesGrN.txt`" et écrit sur le disque un unique fichier "`NotesAmphi_trie.txt`" compilant toutes les notes et toujours trié par ordre alphabétique de nom.
5. On souhaite maintenant effectuer le même travail mais en triant le fichier final suivant un ordre croissant des notes. Proposer des modifications aux deux premières fonctions pour ce faire. Les problèmes à résoudre sont : comment extraire la note de la ligne ? comment composer la liste de tous les étudiants pour effectuer le tri ? comment réécrire les informations dans le fichier final ?

Exercice A.11 *(Création de modules externes)*

Nous avons pour l'instant mis toutes les fonctions d'un programme dans un même fichier et utilisé des fonctions provenant de modules externes fournis avec le langage (grâce à la commande `from xxx import yyy`). Nous pouvons nous aussi créer des modules externes en regroupant certaines fonctions dans un fichier Python.

1. Écrire trois fonctions :
   — `fact(n)` qui renvoie la factorielle de l'entier passé en paramètre (supposé positif ou nul) ;
   — `binome(n, p)` qui renvoie le coefficient binomial :
   $$\binom{n}{p} = \frac{n!}{p!\,(n-p)!}$$
   — `triangle(n)` qui affiche sur une même ligne les coefficients binomiaux de la n^e ligne du triangle de Pascal.
2. Mettre ces trois fonctions dans un même fichier appelé `myMath.py`
3. Créer, dans le même répertoire, un fichier `calculs.py` qui va permettre de tester ces fonctions en affichant la factorielle des nombres de 0 à 6 ainsi que les sept premières lignes du triangle de Pascal.
   — Il faut indiquer dans votre code où se trouvent ces fonctions à l'aide de la commande `from MyMath import *`.
   — Il faut également que l'interpréteur puisse trouver ce fichier ! Pour cela, le répertoire courant doit être initialisé avec le nom du répertoire de travail où se trouvent ces deux fichiers `myMath.py` et `calculs.py`.
4. Il est intéressant de pouvoir tester les fonctions définies dans un module. Cependant, si on ajoute directement dans le code du module des appels aux fonctions, les lignes correspondantes seront exécutées lors de l'importation, ce qui n'est pas souhaitable.
   — Ajouter, directement dans `myMath.py`, l'affichage de $5!$, de $\binom{5}{3}$ et de la 3^e ligne du triangle et exécuter à nouveau `calculs.py` en relançant l'interpréteur (pour forcer la relecture du module).
   — On voit que le résultat des trois tests s'affiche. Pour éviter cela, il faudrait que les tests ne soient exécutés que lorsque le module est exécuté et pas le fichier `calculs.py` (qui utilise ce module). Pour ce fait, il faut mettre les trois tests dans un bloc qui n'est exécuté que si le module est le fichier principal, condition qui peut être testée avec une ligne
   `if __name__ == "__main__":`
   — On peut maintenant vérifier que l'exécution directe du module provoque l'exécution des tests alors que son importation dans un autre fichier ne le fait pas.

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

B. Contrôle continu 2016-2017

Exercice B.1 *(Questions de cours)*

1. Donner la définition d'un algorithme.
2. Donner la différence entre un algorithme et un programme informatique.
3. Donner la définition d'un type en informatique.
4. Lors de l'évaluation d'une expression logique, quel est le type de la valeur renvoyée ? Quelles peuvent être ces différentes valeurs ?
5. Qu'est-ce qu'une itération d'une boucle `TANT_QUE` ?

Exercice B.2

Nous vous rappelons qu'une année est dite bissextile :
- si l'année est divisible par 4 et non divisible par 100 ;
- ou si l'année est divisible par 400.

Écrire un programme en Python qui demande à l'utilisateur de saisir une année (on supposera que l'utilisateur saisira correctement l'année) et affichera si l'année en question est bissextile ou pas.

Exercice B.3

Soit le programme Python ci-dessous :
(Les numéros à gauche vous permettent de faire plus facilement référence à une ligne).

```
1  nbLignes = input("Donner le nombre de lignes :")
2  caractere = input(Quel est le caractère à afficher ?)
3  compteur_1 == 0
4  while (compteur_1 < nbLignes):
5      compteur_2 = 0
6      while (compteur_2 <= compteur_1)
7          print("caractere", end='')
8      compteur_2 += 1
9      print()
10     compteur_1 = compteur_1+1
```

Ce programme contient 6 erreurs. Identifier ces 6 erreurs et proposer une correction pour chacune d'entre elles afin que l'exécution de ce programme génère les affichages suivants :

```
>>>Donner le nombre de lignes :5
>>>Quel est le caractère à afficher ?*
*
**
***
****
*****
```

Exercice B.4

Étant donné un livret bancaire rémunéré au taux annuel de 1,5 %, sur lequel 6 000 euros auront été déposés le 1er janvier 2017, écrire un algorithme en pseudo-code permettant de savoir au bout de combien d'années la somme disponible sur ce livret aura dépassé 7 000 euros.

Nous vous rappelons que si l'on place une somme s le 1er janvier d'une année n sur un livret bancaire rémunéré au taux annuel de 1,5 %, la somme disponible le 1er janvier de l'année $(n+1)$ sera $s + s * 0,015$. *(La fonction logarithme n'est pas utilisable.)*

Exercice B.5

Écrire un algorithme en pseudo-code qui demande à l'utilisateur de saisir un nombre a puis, si $a < 100$ et a est pair, il ne fait rien, et dans le cas contraire affiche `"c'est magique"`.

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

C. Contrôle continu 2017-2018

Exercice C.1 *(Questions de cours)*

1. Qu'est-ce qu'une erreur de syntaxe ?
2. Qu'est-ce qu'une erreur de conception ?
3. Que permet de faire une affectation ? Comment s'écrit-elle en pseudo-code ?
4. Quelle est la valeur de l'expression `(n<100 ou n%3==0)` pour $n = 20$?
   Quel est son type ?
5. Quels sont les 3 opérateurs logiques vus en Python ?
6. À quoi sert une boucle `TANT_QUE` ?

Exercice C.2

1. On considère le programme suivant :

```
1 n = int(input("Votre nombre ?"))
2 d = n-1
3 trouve = False
4 while (d>1) and not(trouve):
5     if (n%d == 0): trouve = True
6     else: d = d-1
7     print("  Pour d =", d, "trouve =", trouve)
8 print("Resultat =", d)
```

   On vous demande de simuler deux exécutions de ce programme et d'indiquer l'affichage généré. L'utilisateur entrera 3 pour la première exécution et 4 à la seconde.

   Pour ce faire, vous pouvez remplir les 2 tableaux suivants (attention, en raison du nombre variable de passages dans la boucle, toutes les lignes ne doivent pas forcément être remplies).

nº ligne exécutée	n	d	trouve	Affichage

nº ligne exécutée	n	d	trouve	Affichage

2. Que calcule ce programme ?

Exercice C.3

Le Plus Petit Commun Multiple (PPCM) de deux entiers positifs a et b est le plus petit entier p tel que p soit à la fois multiple de a et de b.

Afin de calculer le PPCM de a et b, on applique l'algorithme suivant qui consiste à calculer les multiples successifs de a et b, notés m_a et m_b, jusqu'à arriver à l'obtention du PPCM.

(i) initialiser m_a et m_b ;

(ii) tant que le PPCM n'est pas trouvé, on augmente la plus petite des deux valeurs m_a et m_b ;

(iii) afficher le PPCM.

1. Comment initialiser m_a et m_b ?
2. Comment identifier que le PPCM est trouvé ?
3. De combien doit-on augmenter à chaque itération m_a ? et m_b ?
4. Écrire en **pseudo-code** l'algorithme complet : il doit lire deux nombres saisis par l'utilisateur (on les supposera entiers et positifs), calculer leur PPCM et l'afficher.

Exercice C.4

On considère l'algorithme suivant :

```
ALGO
VARIABLES
X, U, N TYPE NOMBRE
DEBUT
    ECRIRE "Nombre ?"
    LIRE X
    U <- X
    N <- 0
    TANT_QUE U != 1 FAIRE
        DEBUT
            N <- N + 1
            SI U%2 == 0 ALORS
                DEBUT
                    U <- U / 2
                FIN
            SINON
                DEBUT
                    U <- 3*U + 1
                FIN
        FIN
    ECRIRE X, "arrive en 1 en ", N, " étapes."
FIN
```

Traduire cet algorithme en un programme **Python**.

Exercice C.5

On souhaite caractériser des triangles.
Écrire un programme **Python** qui :

1. demande à l'utilisateur de saisir trois longueurs réelles a, b, c ;
2. détermine s'il est possible de construire un triangle de côtés a, b et c (pour cela, la somme des deux côtés les plus courts doit toujours être supérieure ou égale au côté le plus long) ;
3. et si c'est le cas, détermine si ce triangle est rectangle (le carré d'un des côtés doit être égal à la somme des carrés des deux autres), équilatéral (les trois côtés sont égaux) ou ni l'un ni l'autre (on dira alors qu'il est quelconque) ;
4. et termine en affichant les résultats (affiche s'il est possible ou non de construire un tel triangle, ses propriétés ou s'il est quelconque).

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

D. Partiel 2016-2017

Exercice D.1 *(Questions de cours)*

1. Écrire une boucle `TANT_QUE` équivalente à

```
POUR i ALLANT_DE 1 A 10
    DEBUT
        BlocInstructions
    FIN
```

2. Étant donnés

```
ch1='C\'est un partiel'
ch2=' de Python'
```

décrire l'affichage produit par chacune des instructions suivantes

```
print(ch1 + ch2)
print(ch1[9:])
print(ch1 + ch2[:1] + "d'algo")
print('python' in ch2)
print(ch1.replace('partiel','examen'), ch1)
```

3. Comment initialiser une liste L contenant l'ensemble des entiers compris entre 4 et 100 inclus ?
4. Étant donnée

```
L=["Python", 4, -1]
```

décrire l'affichage produit par ce bloc d'instruction :

```
>>> L2=L.append("Algo")
>>> print(L, L2)
```

Exercice D.2

Étant donnée $f(x) = x^3 - 3x^2 + 1$, une fonction continue et strictement croissante sur l'intervalle $[2; 3]$, **écrire un algorithme** en pseudo-code permettant de déterminer la valeur approchée de x telle que $f(x) = 0$ par une méthode dichotomique avec une précision ϵ.

Explication de la méthode dichotomique

Étant donnée f une fonction continue et strictement croissante sur un intervalle $[a; b]$ telle que $f(a)$ et $f(b)$ soient de signes contraires, les étapes d'une telle méthode dichotomique sont :

— calculer m le milieu de $[a; b]$;
— si $f(m)$ et $f(b)$ sont de même signe, c'est que la solution se trouve dans $[a; m]$: on affecte à b la valeur de m afin de pouvoir continuer le processus ;
— dans le cas contraire, la solution se trouve dans $[m; b]$: on affecte à a la valeur de m afin de pouvoir continuer le processus ;
— on recommence le processus jusqu'à obtenir un encadrement de la solution avec la précision voulue (cette précision sera fixée par l'utilisateur).

Exercice D.3

1. **Écrire un programme Python** qui saisit un entier n puis une liste L de n entiers.
2. Étant donnée la liste L saisie précédemment, **écrire la suite du programme Python** qui permet de ne garder qu'une seule occurrence de chacun des éléments de L.

Exercice D.4

Écrire un programme Python qui prend en entrée deux chaînes de caractères `s1` et `s2` (supposées sans espace), et affiche `"OK"` s'il est possible, en enlevant certains caractères de `s2`, de retrouver `s1`. Dans le cas contrainte, le programme affiche `"impossible"`.

Par exemple, si `s1='bd'` et `s2='abcde'`, le programme affiche `"OK"`.

Avec `s1='bd'` et `s2='bd'`, le programme affiche également `"OK"`.

Par contre, il affiche `"impossible"` lorsque `s1='db'` et `s2='abcde'`.

Exercice D.5

Soit le programme Python suivant :

```
1  ch = input("Rentrez une chaine ")
2  deb = 0
3  fin = 0
4  while fin < len(ch) :
5      if ch[fin] != ' ' and fin < len(ch)-1 :
6          fin += 1
7      else :
8          if fin-deb > 2 :
9              cpt = deb
10             while cpt <= fin :
11                 print(ch[cpt], end='')
12                 cpt += 1
13         else :
14             cpt = deb
15             while cpt <= fin :
16                 print(ch[cpt]*2, end='')
17                 cpt += 1
18         fin += 1
19         deb = fin
```

1. Indiquer l'affichage produit lorsque la chaîne de caractères saisie par l'utilisateur est : `"Vive le Python!"`
2. Si on remplace maintenant la ligne 5 par
   `if ch[fin]!=' ' :`
   indiquer l'affichage produit lorsque la chaîne de caractères saisie par l'utilisateur est : `"Vive le Python!"`

Exercice D.6

Étant donnée une séquence $a_1, a_2, \ldots, a_n$ de n valeurs entières stockées dans un tableau `tab`, **écrire un algorithme** en pseudo-code qui permet de classer ces nombres de telle sorte que la première partie du tableau contienne les nombres pairs et la seconde partie les nombres impairs.

Par exemple, si l'utilisateur entre 5 valeurs : 4, 7, 0, 1, 5 alors le tableau `tab` devra contenir 4, 0, 7, 1, 5 à la fin de l'algorithme.

L'ordre respectif de tous les éléments pairs et tous les éléments impairs n'a pas d'importance ; il faut juste que tous les pairs soient avant tous les impairs.

L'algorithme devra comporter plusieurs parties :

1. demander à l'utilisateur de saisir le nombre d'éléments ainsi que la séquence initiale (on supposera que l'utilisateur ne saisit que des nombres) ;
2. ranger le tableau sans utiliser de variable de type tableau autre que `tab` ;
3. afficher le tableau modifié.

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

E. Partiel 2017-2018

Exercice E.1 *(Questions de cours)*

1. Soient les fonctions Python suivantes :

```
def calculPerimetre(largeur, longueur):
    perimetre = 2*(largeur+longueur)

def fonctionPerimetre(largeur, longueur):
    return 2*(largeur+longueur)

def trouverPerimetre(largeur, longueur):
    print(2*(largeur+longueur))
```

Quels vont être les affichages des instructions suivantes ?

(a) `print(calculPerimetre(4,5))`

(b) `print(fonctionPerimetre(4,5))`

(c) `print(trouverPerimetre(4,5))`

(d)
```
if (calculPerimetre(4,5) == 18):
    print("Le périmètre vaut 18")
```

2. Soit le programme Python suivant :

```
def maximum(x,y):
    if x > y:    return x
    elif x == y: return "Les deux nombres sont égaux"
    else:        return y

a = int(input("Saisir la valeur de a :"))
b = int(input("Saisir la valeur de b :"))
c = int(input("Saisir la valeur de c :"))
print("Le maximum est :", a+maximum(b,c))
```

(a) Qu'affiche le programme si l'utilisateur saisit `3` pour `a`, `2` pour `b` et `4` pour `c` ?

(b) Qu'affiche le programme si l'utilisateur saisit `3` pour `a`, `4` pour `b` et `4` pour `c` ?

3. Soient les fonctions suivantes :

```
def calculSuivant(n):
    if (n%2 == 0):
        return n//2
    else:
        return (3*n)+1

def syracuse(n):
    while (n != 1):
        n = calculSuivant(n)
        print(n)
```

Quels vont être les affichages des instructions suivantes :

(a) `syracuse(2)`

(b) `syracuse(8)`

Exercice E.2

On cherche à calculer e^x à l'aide des développements limités. On sait que lorsque K tend vers l'infini :

$$e^x = \sum_{n=0}^{K} \frac{x^n}{n!}$$

où $n! = 1 * 2 * \ldots * (n-1) * n$ et $0! = 1$.

Le programme ci-dessous lit un nombre saisi au clavier par l'utilisateur, et affiche la plus petite valeur de K qui permet de calculer l'exponentiel de ce nombre avec une précision de $0{,}01$. Mais ce programme comporte 11 erreurs ! Il s'agit de le corriger avec un stylo de couleur directement sur le programme ci-dessous : utiliser les lignes vides pour mettre une version corrigée de la ligne précédente comportant une (ou des) erreur(s) ou ajouter d'éventuelles instructions manquantes.

```
from math import exp

x = input(Donner x)

somme = 1

puissance = x

epsilon = '0.01'

n = 2

while abs(math.exp(x)-somme) < epsilon

    puissance = puissance*x

    fact = fact*n

    somme = puissance/fact

print('K='n)
```

Exercice E.3

On joue à lancer 3 dés : un dé rouge à 6 faces (numérotées de 1 à 6), un dé vert à 8 faces (numérotées de 1 à 8) et un dé bleu à 10 faces (numérotées de 1 à 10). On veut déterminer combien il y a de façons d'obtenir un nombre n donné en additionnant les trois faces des dés. Par exemple, il y a une seule façon d'obtenir 3 qui est de faire 1 avec le dé vert, 1 avec le dé rouge et 1 avec le dé bleu. Il y a 3 façons d'obtenir 4 :

— 1 avec le dé vert, 1 avec le dé rouge et 2 avec le dé bleu, ou bien,
— 1 avec le dé vert, 2 avec le dé rouge et 1 avec le dé bleu, ou bien,
— 2 avec le dé vert, 1 avec le dé rouge et 1 avec le dé bleu.

Écrire le programme **Python** qui implémente l'algorithme suivant :

1. demander à l'utilisateur de saisir une valeur entière ;
2. à l'aide de boucles imbriquées, compter le nombre de possibilités permettant d'obtenir ce nombre comme somme des dés rouge, vert et bleu ;
3. afficher le résultat.

Exemple d'exécution :

affichage Entrez un nombre entier :

saisie 5

affichage Il y a 6 façon(s) de faire 5 avec ces trois dés.

Exercice E.4

1. Étant donnée une séquence $a_1, a_2, \ldots, a_m$ de m nombres stockée dans un tableau `tab` de taille n, avec $m < n$, écrire en **pseudo-code une fonction** qui permet d'ajouter un nouvel élément x dans cette séquence. Cette fonction prendra en argument le tableau, l'élément x à insérer et la position `p` (cela signifiant qu'après l'insertion, l'élément se trouve en position `p`). Elle renverra le tableau modifié. **On pourra supposer que la position passée en argument est toujours valide.**
2. Écrire un algorithme permettant à l'utilisateur de saisir un nombre quelconque $m < n$ de valeurs qui initialiseront le tableau `tab`. Pour simplifier, nous supposerons que le tableau est de taille $n = 2m$. L'utilisateur doit d'abord saisir le nombre m de valeurs souhaitées puis saisir toutes ces valeurs. Ensuite l'algo demandera à l'utilisateur quel est l'élément `x` à ajouter et quelle est la position `p` à laquelle il faut insérer le nouvel élément et utilisera la fonction de la question précédente pour réaliser l'insertion. Une fois l'insertion réalisée, le programme affichera la nouvelle séquence et sa longueur. L'utilisateur doit pouvoir insérer autant d'éléments qu'il le souhaite tant que la taille de la séquence reste inférieure à $2m$ (dans le cas contraire, l'algorithme devra préciser à l'utilisateur que le tableau est plein).

Exercice E.5

Il pîaart que puor la lctreue, l'orrde des lrttees à l'iétunreir des mots n'a acnuue itnpocmare. La sulee chose qui cptmoe est que la pemièrre et la dneèirre lttree seonit à leur pclae.

L'objectif de cet exercice est de concevoir un programme en **Python** permettant de tester cette théorie. Étant donnée une chaîne de caractères saisie par l'utilisateur, il mélangera aléatoirement les lettres à l'intérieur des mots et affichera la phrase modifiée. On supposera que la chaîne de caractères ne comporte pas de signe de ponctuation et que deux mots sont séparés par exactement une espace.

Attention : L'utilisation du type `list` est interdite dans cet exercice.

1. Écrire une fonction `permuteMot()` qui prend en argument un mot et permute aléatoirement les lettres à l'intérieur du mot en gardant les première et dernière lettres inchangées.
2. Écrire une fonction `permute()` qui prend en argument une phrase et applique la première fonction sur chacun des mots.
3. Écrire le programme demandé en utilisant les fonctions précédentes.

Exemple : L'utilisation saisit la phrase `"Je vais avoir une bonne note"`, alors le programme peut afficher `"Je vias avior une bnone ntoe"`.

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

F. Partiel 2019-2019

Exercice F.1 *(Questions de cours)*

1. Transformer la boucle `POUR` ci-dessous en boucle `TANT_QUE` équivalente :

```
POUR i ALLANT DE 10 A 20
    DEBUT
        ECRIRE i
    FIN
```

2. Qu'affiche le programme suivant ?

```
x = 'Partiel algorithmique et Python'
if 'Python' in x: print("Langage")
elif 'Partiel' in x: print("Evaluation")
else: print("Math")
```

3. Dans la fonction `print()`, que permettent de faire les paramètres `sep` et `end` ? Quelle est leur valeur par défaut ?
4. Quelle est la différence entre un algorithme et un programme ?

Exercice F.2

On souhaite réaliser un **programme Python** permettant d'afficher une table d'addition de la forme :

```
Saisir un entier inférieur ou égal à 10 :7
 0+0 0+1 0+2 0+3 0+4 0+5 0+6 0+7
     1+0 1+1 1+2 1+3 1+4 1+5 1+6
         2+0 2+1 2+2 2+3 2+4 2+5
             3+0 3+1 3+2 3+3 3+4
                 4+0 4+1 4+2 4+3
                     5+0 5+1 5+2
                         6+0 6+1
                             7+0
```

Pour cela, on a écrit **en Python** le programme suivant :

```
1  n = 11
2  while n > 10:
3      n = input("Saisir un entier inferieur ou egal a 10: ")
4  while i <= n
5      print("    "*i, sep="")
6      j = i
7      ch = ""
8      while j <= n:
9          ch + " " + str(i) + "+" + (j-i)
10     print(ch)
11 i = i+1
```

Ce programme comporte **8 erreurs**. Corriger ces erreurs avec un stylo de couleur directement sur le programme ci-dessus : utiliser l'espacement entre les lignes pour mettre une version corrigée d'une ligne comportant une (ou des) erreur(s) et/ou ajouter d'éventuelles instructions manquantes.

Exercice F.3

Un administrateur d'un site web veut assurer un maximum de sécurité pour les utilisateurs du site. Pour cela, il décide de réaliser un programme Python qui évalue la force des mots de passe des différents utilisateurs du site, sachant qu'un mot de passe est une chaîne de caractères qui ne comporte ni espaces ni lettres accentuées.
La force d'un mot de passe varie, selon la valeur d'un score, de 'Très faible' jusqu'à 'Très fort' :

— si le score <20, la force du mot de passe est 'Très faible' ;
— sinon si le score<40, la force d'un mot de passe est 'Faible' ;
— sinon si le score <80, la force du mot de passe est 'Fort' ;
— sinon la force du mot de passe est 'Très fort'.

Le score se calcule en additionnant des bonus et en retranchant un malus.

Les bonus attribués sont :
— Nombre total de caractères * 4
— (Nombre total de caractères – nombre de lettres majuscules) * 2
— (Nombre total de caractères – nombre de lettres minuscules) * 3
— Nombre de caractères non alphabétiques * 5

Le malus se calcule en additionnant :
— La longueur de la plus longue séquence de lettres minuscules * 2 et
— La longueur de la plus longue séquence de lettres majuscules * 3

Pour cela l'administrateur a commencé par programmer 2 fonctions :

```
def minuscules():
    return "abcdefghijklmnopqrstuvwxyz"

def majuscules():
    return "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

Écrire **en Python** :

1. Une fonction, appelée `nbLettresMin()`, qui prend en argument une chaîne de caractères et renvoie le nombre total de lettres minuscules contenues dans la chaîne. Cette fonction doit faire appel à la fonction `minuscules()`.

   Par exemple, l'appel de `nbLettresMin("PL1_promo2018")` doit renvoyer `5`.

2. Une fonction, appelée `nbCaraNonAlpha()`, qui prend en argument une chaîne de caractères et renvoie le nombre total de caractères non alphabétiques (qui ne sont ni des lettres minuscules, ni des lettres majuscules) contenus dans la chaîne. Cette fonction doit faire appel aux fonctions `minuscules()` et `majuscules()`.

   Par exemple, l'appel de `nbCaraNonAlpha("PL1_promo2018")` doit renvoyer `6`.

3. Une fonction, appelée `bonus()`, qui prend en argument une chaîne de caractères et renvoie la valeur du bonus, calculé comme indiqué dans l'énoncé page 87. Cette fonction doit faire appel aux fonctions `nbLettresMin()` et `nbCaraNonAlpha()` et à une fonction `nbLettresMaj()` qu'on supposera existante qui prend en argument une chaîne de caractères et renvoie le nombre de lettres majuscules contenues dans la chaîne.

   Par exemple, l'appel de `bonus("PL1_promo2018")` doit renvoyer `128`, c'est-à-dire le résultat de $13 * 4 + (13 - 2) * 2 + (13 - 5) * 3 + 6 * 5$.

4. Une fonction, appelée `plusLongueSequenceMin()`, qui prend en argument une chaîne de caractères et renvoie le nombre de caractères de la plus longue séquence de minuscules. Cette fonction doit faire appel à la fonction `minuscules()`.

   Par exemple, l'appel de `plusLongueSequenceMin("ab2018toto")` doit renvoyer `4`.

5. Une fonction, appelée `malus()`, qui prend en argument une chaîne de caractères et renvoie la valeur du malus, calculé comme indiqué dans l'énoncé page 87. Cette fonction doit faire appel à la fonction `plusLongueSequenceMin()` et à une fonction qu'on supposera existante, `plusLongueSequenceMaj()`, qui prend en argument une chaîne de caractères et renvoie le nombre de caractères de la plus longue séquence de majuscules contenues dans la chaîne.

   Par exemple l'appel de `malus("PL1_promo2018")` doit renvoyer `16`, c'est-à-dire le résultat de $5 * 2 + 2 * 3$.

6. Un programme principal qui demande à l'utilisateur de saisir une chaîne de caractères et qui affiche la force du mot de passe saisi, en faisant appel aux fonctions `bonus()` et `malus()`.

   Par exemple, si l'utilisateur saisit la chaîne `"PL1_promo2018"`, le programme doit afficher :

   ```
   Le mot de passe a un score de 112 il est donc 'Tres fort'
   ```

Exercice F.4

Un entier naturel n est appelé un nombre **puissant** lorsque, pour tout diviseur premier p de n, p^2 divise n. L'objectif de cet exercice est d'écrire un algorithme en pseudo-code permettant de tester si une valeur n saisie au clavier par un utilisateur est un nombre puissant. Par exemple, le nombre 72 est un nombre puissant : ses deux diviseurs premiers sont 2 et 3 et, 72 est également divisible par 4 et 9. Par contre, 63 n'est pas un nombre puissant car 7 est un diviseur premier de 63 mais 63 n'est pas divisible par 49.

1. Écrire, **en pseudo-code**, une fonction, appelée `premier()`, qui prend un entier e en argument et renvoie un booléen qui vaut `Vrai` si e est premier et `Faux` sinon.
2. Écrire, **en pseudo-code**, une fonction, appelée `diviseurPremier()`, qui prend un entier n en argument et renvoie un tableau de booléens. Dans ce tableau, si la valeur booléenne stockée à l'indice i vaut `Vrai`, cela signifie que i est un diviseur premier de n ; à l'inverse, si la valeur booléenne stockée à l'indice i vaut `Faux`, cela signifie que i n'est pas un diviseur premier de n. Cette fonction fera appel à la fonction `premier()`.
3. Écrire, **en pseudo-code**, le programme principal qui lit une valeur n, appelle la fonction `diviseurPremier()` pour connaître les diviseurs premiers de n et indique si n est puissant en affichant le résultat à l'écran. Si n est puissant, il faut afficher la liste de ses diviseurs premiers. Si n n'est pas puissant, il faut afficher un diviseur premier d qui est tel que n n'est pas divisible par d^2.

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

G. Examen 2016-2017

Exercice G.1 *(Questions de cours)*

1. Qu'est-ce que la signature d'une fonction ?
2. Qu'est-ce qu'une variable locale ?
3. Comment modifier une variable globale dans une fonction ?
4. Donner les trois modes d'ouverture d'un fichier effectuée avec la fonction `open`.
5. Étant donnée la liste `L=[17, 38, 10, 25, 72]`, écrire les instructions permettant de :
   — trier et afficher la liste ;
   — ajouter l'élément 12 à la fin de `L` ;
   — afficher la liste renversée sans la modifier ;
   — enlever l'élément 38 ;
   — afficher la sous-liste du 2e au 3e élément (inclus), c'est-à-dire la sous-liste [38,10], en utilisant le slicing ;
   — afficher le dernier élément en utilisant un indice négatif.
6. À quoi sert un `return` dans une fonction ?
7. Lorsque l'on appelle une fonction en passant une variable en argument, quelle différence cela fait-il que cette variable contienne une valeur entière ou une liste ?

Exercice G.2

1. Écrire en **pseudo-code** une fonction `FUSION` qui prend en argument deux tableaux T et B de NOMBRES et qui renvoie un tableau de NOMBRES de même taille. On supposera que les tableaux T et B sont déjà triés par ordre croissant. La fonction `FUSION` doit, à l'aide d'UNE et UNE SEULE BOUCLE RÉPÉTITIVE, construire un tableau RES contenant les n plus petits éléments de T et B réunis. Le tableau RES renvoyé devra lui aussi être trié par ordre croissant.

   Par exemple, si $T = [1, 3, 3, 5, 6]$ et $B = [-2, 1, 2, 4, 6]$, alors `FUSION(T,B)` renvoie $[-2, 1, 1, 2, 3]$.
2. Écrire l'algorithme qui initialise les tableaux T et B avec les valeurs de la question précédente et affiche le tableau RES.

Exercice G.3

1. Représenter la simulation des différents appels récursifs de ce programme :

```
def mystere(s):
  if (s == ''):
      return 0
  else:
      return 1 + mystere(s[1:])

print(mystere('Algo'))
```

2. Que va afficher ce programme et que renvoie la fonction `mystere` lorsqu'une chaîne quelconque lui est passée en argument ?
3. Donner une version itérative en Python de la fonction `mystere` sans utiliser aucune fonction prédéfinie de Python.

Exercice G.4

Une technique de cryptographie consiste à décaler les lettres d'un message pour le rendre illisible. Dans cet exercice, pour chacune des trois approches ci-dessous, il vous est demandé d'écrire un programme en **Python** permettant de crypter un message donné. Pour chacun des trois programmes, il vous faudra :

— lire une phrase saisie par l'utilisateur ;
— crypter cette phrase en entier suivant le code indiqué ;
— afficher la phrase entièrement cryptée obtenue.

Nous supposerons que la chaîne saisie par l'utilisateur ne contient que des majuscules (non accentuées) et des espaces.

Dans tous les codages ci-dessous, les espaces ne seront pas modifiées, seules les lettres seront cryptées.

1. Une première approche consiste à décaler les lettres d'un caractère. Ainsi, les A deviennent des B, les B des C, ..., et les Z des A. Dans cette question, il est demandé d'utiliser le code ASCII *(American Standard Code for Information Interchange)* qui permet de représenter chaque caractère par une même valeur numérique. Ainsi A est codé par 65, B par 66, ..., et Z par 90.

   Exemple : si l'utilisateur entre « `ZORGLUB ET FANTASIO` », le programme devra afficher « `APSHMVC FU GBOUBTJP` ».

   Indication : la fonction `ord(car)` permet de récupérer la valeur numérique d'un caractère `car` et, à l'inverse, la fonction `chr(i)` permet de trouver le caractère correspondant à une valeur numérique `i`.

2. Une seconde approche, connue sous le nom de « chiffre de César », est une amélioration du premier procédé et consiste à décaler les lettres non pas d'un caractère mais d'un nombre quelconque de lettres. Par exemple, si le décalage choisi est de 10, les A deviennent des K, les B des L, ..., et les Z des J.

   Ici, le décalage doit être demandé à l'utilisateur.

   Exemple : si l'utilisateur entre « `ZORGLUB ET FANTASIO` » avec un décalage de 10, le programme devra afficher « `JYBQVEL OD PKXDKCSY` ».

Dans cette question, on n'utilisera pas le code ASCII (les fonctions `chr()` et `ord()` ne sont pas autorisées) mais on prendra une chaîne de caractères constituée des 26 lettres de l'alphabet (A à Z), dans l'ordre, comme point de référence.

3. Dans une troisième approche, une clé de codage est utilisée dans laquelle les 26 lettres sont rangées dans le désordre. Par exemple, une clé de codage possible serait « `CLUMNPHKOZYWVQBIFDJAXREGTS` ». Cette clé de codage sert ensuite à crypter le message : on substitue à chaque lettre du message celle située dans la clé à la même position alphabétique. Ainsi, avec notre exemple de clé, les A deviennent des C, les B des L, . . . , et les Z des S.

   Exemple : si l'utilisateur entre « `ZORGLUB ET FANTASIO` » avec la clé ci-dessus, le programme devra afficher « `SBDHWXL NA PCQACJOB` ».

   Ici, vous devrez d'abord générer aléatoirement la clé de codage avant de crypter votre message, en vous assurant que votre clé contienne bien les 26 lettres de l'alphabet, sans répétition. Là encore, les fonctions `chr()` et `ord()` ne sont pas autorisées.

Exercice G.5

Étant donné le programme Python ci-dessous :

```
def fonction1(a):
    n = []
    for i in range(a):
        n.append(int(input("Saisir une valeur: ")))
    return n

def fonction2(a):
    n = 0
    for i in a:
        n += i
    return n/len(a)

n = int(input())
a = fonction1(n)
n = fonction2(a)
c = 0
for i in a:
    if i > n:
        c += 1
print(c)
```

— Donner le type des variables locales de `fonction1`. Décrire ce que fait `fonction1`.
— Donner le type des variables locales de `fonction2`. Décrire ce que fait `fonction2`.
— Quel sera l'affichage produit par l'exécution de ce programme lorsque l'utilisateur va saisir au clavier les valeurs suivantes : 3 puis 10 puis 10 puis 4 ?

Exercice G.6

L'objectif du jeu dit du nombre mystère consiste à faire découvrir, par le joueur, la valeur (supposée de type entier) d'un nombre mystère, noté `Mys`, compris entre 0 (inclus) et 100 (inclus), généré par l'ordinateur. Pour cela le joueur dispose d'un nombre maximum d'essais, noté `NbEssai`. À chaque proposition du joueur, il lui est indiqué si le nombre recherché est plus petit, plus grand ou trouvé. Une proposition du joueur qui ne serait pas un entier de l'intervalle $[0, 100]$ n'est pas comptée comme un essai et invite à une nouvelle proposition. La partie se termine si le joueur a découvert le nombre mystère ou s'il a atteint le nombre maximum d'essais. Un message indique alors au joueur s'il a perdu ou gagné. Dans le cas où il a perdu, le programme devra afficher quel était le nombre mystère.

Écrire un algorithme en **pseudo-code** permettant à l'utilisateur de jouer contre l'ordinateur. Il sera demandé à l'utilisateur de fixer une valeur pour `NbEssai`. Le programme devra générer `Mys` puis inviter l'utilisateur à deviner le nombre en appliquant les règles proposées ci-dessus (on supposera l'existence d'une fonction `RAND(a,b)` renvoyant un nombre aléatoire dans l'intervalle [a,b]).

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

H. Examen 2017-2018

Exercice H.1 *(Questions de cours)*

1. Citer, en Python, un type modifiable et deux types non modifiables.
2. On définit une liste : `L=[12,42,'a',True]`
   Donner l'affichage de l'instruction suivante :
   `print(L[1:2], L[2:5])`
3. Soit le programme Python suivant :

```
def f(n):
    global x
    y = 5
    x += 1
    return n*x+y

x = 0
y = 2
print(y)
for i in range(2,4):
    print(f(i))
print(y)
```

   (a) Quelles sont les variables locales à la fonction `f()` ?
   (b) Quels sont les nombres affichés par le programme lors de son exécution ?
4. Quels vont être les affichages du programme suivant :

```
def fonction1(L):
  L = [4] + L[1:]

def fonction2(L):
  L[0] = 4

L = [1,2,3]
fonction1(L)
print(L)
fonction2(L)
print(L)
```

5. Comment obtient-on la représentation en complément à 2 sur 8 bits d'un entier négatif ?
6. Donner deux cas d'utilisation des tuples.

Exercice H.2

1. Écrire une fonction Python `minimum()` renvoyant l'indice de l'élément de plus petite valeur d'une liste de nombres, passée en paramètre. On pourra supposer que la liste passée en paramètre est non-vide.
2. Écrire (sans utiliser `sorted()` ou `sort()`) une fonction Python **récursive** `tri()` prenant en argument une liste `L` et renvoyant la liste triée, en appelant la fonction `minimum()`. Cette fonction ne doit pas modifier `L`.
3. Écrire en Python le programme principal faisant saisir un entier `n` à l'utilisateur, puis faisant saisir `n` nombres à l'utilisateur, les ajoutant dans une liste et affichant à la fin la liste triée en appelant la fonction récursive précédente.
4. Représenter les appels récursifs de votre fonction pour l'appel `tri([3,8,1,6])`.

Exercice H.3

Deux mots sont des anagrammes s'ils contiennent les mêmes lettres (avec le même nombre d'occurrences) dans un ordre qui peut être différent. Par exemple `"nacre"` et `"crane"` ou `"parisien"` et `"aspirine"` sont des anagrammes. On se propose de détecter les anagrammes de trois façons différentes.

1. (a) Écrire une fonction Python `compte()` qui prend en argument une chaîne de plusieurs caractères et une chaîne contenant un seul caractère et renvoie le nombre d'occurrences du caractère de la deuxième chaîne dans la première.

   (b) Écrire une fonction Python `estAnagramme()` prenant deux chaînes de caractères en argument et renvoyant `True` si ces deux chaînes sont des anagrammes ou `False` dans le cas contraire ; on utilisera la fonction précédente.

   (c) Écrire le programme principal permettant à l'utilisateur de saisir deux mots et lui indiquant, en utilisant la fonction précédente, s'ils sont des anagrammes.

2. (a) Écrire une fonction Python `Dico()` qui prend en argument une chaîne de caractères et renvoie un dictionnaire mettant en correspondance chaque caractère apparaissant dans la chaîne avec le nombre de ses occurrences dans la chaîne.

   Par exemple, si la fonction prend en paramètre la chaîne `"toto"`, le dictionnaire renvoyé par la fonction aura pour valeur : `{'t': 2, 'o': 2}`.

   (b) Écrire le programme principal permettant à l'utilisateur de saisir deux mots et lui indiquant, en comparant les dictionnaires, s'ils sont des anagrammes.

3. La fonction `sorted()` s'applique également à une chaîne : elle renvoie une nouvelle chaîne dans laquelle les caractères de la chaîne passée en argument sont triés par ordre alphabétique. Proposer une écriture de la fonction `estAnagramme()` de la question 1b utilisant `sorted()`.

Exercice H.4

Une séquence de n nombres strictement positifs est stockée dans un tableau de taille n. On souhaite créer un nouveau tableau dans lequel les répétitions successives d'un même nombre auront été supprimées. Ainsi, la séquence $(1, 1, 2, 3, 3, 1, 1, 4, 5, 6, 6, 2)$ devra donner la séquence $(1, 2, 3, 1, 4, 5, 6, 2)$ [on ne supprime pas toutes les répétitions de nombres mais on ne garde qu'un nombre pour chaque bloc de répétitions].

Écrire en **pseudo-code** une fonction prenant le tableau de départ en argument et renvoyant le nouveau tableau. Attention, ce dernier devra être exactement de la taille de la nouvelle séquence.

Exercice H.5

Pour organiser le TP noté en salle machine des étudiants de L1, il faut affecter chaque étudiant à une salle. Pour ce faire, on dispose de deux listes initiales que l'on supposera données :

— la liste des étudiants est stockée dans une variable `E` : chaque élément de cette liste est donc une chaîne de caractères représentant le nom d'un étudiant suivi d'un espace et de son prénom ;

— la liste des capacités des salles est stockée dans une variable `S` : chaque élément de cette liste est donc un entier.

Nous supposerons que la somme des capacités des salles est supérieure ou égale au nombre total d'étudiants. L'objectif est de créer les listes des étudiants affectés aléatoirement à chaque salle (une liste par salle dont la capacité est stockée dans la liste `S`). Chaque étudiant devra être affecté une et une seule fois à une unique salle. Ces listes seront elles-mêmes stockées dans une variable de type liste.

1. Écrire en Python une fonction `salle()` prenant en argument une liste `E` et un entier `c` (inférieur ou égal à la taille de la liste `E`) et qui renvoie une liste constituée de `c` éléments distincts de `E` choisis aléatoirement.
2. Écrire en Python une fonction `affectation()` prenant en argument les listes `E` et `S` et qui renvoie une liste dont chaque élément est une liste d'étudiants affectés à une salle. Cette fonction devra faire appel à la fonction `salle()`.
3. Écrire en Python une fonction `emargement()` qui prend en argument une liste de listes et ne renvoie rien. Cette fonction permet de créer pour chaque salle un fichier qui contient les nom et prénom de chacun des étudiants affectés à cette salle. Sur la première ligne de chaque fichier sera écrit `Feuille d'émargement`, puis un nom et un prénom d'étudiant par ligne, et enfin sur la dernière ligne le nombre d'étudiants affectés à cette salle. Le nom de chaque fichier devra être de la forme `emargement_i.txt`, où `i` est l'indice de la sous-liste associée à la salle `i`.
4. À l'issue de l'affectation, il est possible que les premières salles soient pleines et la dernière presque vide (et même éventuellement d'autres complètement vides ensuite). Pour y remédier, proposer une nouvelle version de la fonction `affectation()` qui permet d'avoir une répartition homogène des places vides : entre deux salles quelconques les nombres restants de places vides diffèrent au plus d'une unité.

   Par exemple, s'il y a 10 étudiants à placer dans trois salles de capacités 5, 8 et 4, au lieu de remplir les deux premières salles de 5 et 5 étudiants (et de laisser la dernière vide), on les remplit respectivement de 2, 6 et 2 étudiants (laissant respectivement 3, 2 et 2 places vides).
5. Écrire le programme principal qui :
   — crée la liste `E` en récupérant les données se trouvant dans un fichier `"L1.txt"` dont chaque ligne correspond au nom et prénom d'un étudiant ;
   — demande à l'utilisateur les capacités de chacune des salles pour créer la liste `S` ;
   — crée les fichiers correspondant aux feuilles d'émargement des salles.

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

I. Examen 2018-2019

Exercice I.1 *(Questions de cours)*

1. Citer 3 modules Python que vous avez utilisés lors des TP :
2. Soit la fonction Python `ajout()` suivante :

```
def ajout(t, i, v):
    t[i] = v
```

   (a) Quel est le type de D ? Que vaut D après l'exécution des 2 instructions suivantes ?

```
D = {}
ajout(D, 0, 'a')
```

   (b) Quel est le type de C ? Que vaut C après l'exécution des 2 instructions suivantes ?

```
C = 'A'
ajout(C, 0, 'a')
```

   (c) Quel est le type de L ? Que vaut L après l'exécution des 2 instructions suivantes ?

```
L = [0]
ajout(L, 0, 'a')
```

3. On définit une liste : `L = [[1,2,[3,4]], ['toto'], ['t', True]]`
   Que se passe-t-il lors de l'exécution du programme Python suivant ?

```
for i in L :
    i.pop()
print(L)
```

4. Que se passe-t-il lors de l'exécution du programme Python suivant ?

```
def test1(v, x):
    v = v+x

v = 10
test1(v,3)
print(v)
```

5. Que se passe-t-il lors de l'exécution du programme Python suivant ?

```
def test2():
    global x
    v = v+x

v = 10
x = 3
test2()
print(v)
```

6. Que se passe-t-il lors de l'exécution du programme Python suivant ?

```
def test3(x):
    global v
    v = v+x

v = 10
test3(3)
print(v)
```

7. Soit le programme suivant :

```
L = [10,54,3,20,1,6,0]
L = L[:2] + L[2:5].sort() + L[5:]
```

   Expliquer l'erreur résultant de son exécution.

Exercice I.2

Le programme ci-dessous permet de stocker un dictionnaire de synonymes. Lors de l'exécution de ce programme, l'utilisateur saisit au clavier des mots et, pour chacun de ces mots, une liste de mots synonymes. Ces valeurs sont stockées dans une variable de type dictionnaire dont les couples (cle, valeur) sont de type `(str, list de str)`. Malheureusement le programme contient **8** erreurs.

```
12  def lecture():
13      n = int(input('Nombre de synonymes : '))
14      valeur = []
15      for i in [0:n]:
16          print("Donner le synonyme numero", i+1, end=' : ')
17          valeur[i] = input()
18      print(valeur)
19  nbMot = int(input('Nombre de mots'))
20  for i in range(nbMot):
21      mot = input('Mot : ')
22      if mot in d.values():
23          print(mot, 'est deja dans le dictionnaire')
24      else:
25          existe = False
26          for cle in d.keys():
27              if mot in d[cle]:
28                  print(mot, "est un synonyme de cle")
29          if not existe:
30              v = lecture()
31              d[mot].append(v)
```

Corriger ces erreurs avec un stylo de couleur directement sur le programme ci-dessus : utiliser l'espacement entre les lignes pour mettre une version corrigée d'une ligne comportant une (ou des) erreur(s) et/ou ajouter d'éventuelles instructions manquantes. L'objectif est d'obtenir le résultat suivant à l'exécution (les valeurs saisies au clavier par l'utilisateur sont en gras) :

```
Nombre de mots 4
Mot :  manger
Nombre de synonymes :  2
Donner le synonyme numero 1 :  avaler
Donner le synonyme numero 2 :  devorer
Mot :  avaler
 avaler est un synonyme de  manger
Mot :  manger
 manger est deja dans le dictionnaire
Mot :  avoir
Nombre de synonyme :  1
Donner le synonyme numero 1 :  posseder
```

Exercice I.3

Soit la fonction récursive `mystere()` ci-dessous :

```
def mystere(ch, ch1, ch2):
    if not(ch1 in ch):
        return (ch)
    elif ch[0] == ch1:
        return (ch2+mystere(ch[1:], ch1, ch2))
    else:
        return (ch[0]+mystere(ch[1:], ch1, ch2))
```

1. Représenter les appels récursifs lorsqu'on exécute l'instruction `mystere('toto','o','a')`. Que permet de faire cette fonction ?
2. Qu'obtient-on si on exécute `print(mystere('toto','o','a'))` ?
3. Qu'obtient-on si on exécute `print(mystere([4,'a',True,4,'b'],4,3))` ?
4. Dans l'hypothèse où `ch`, `ch1` et `ch2` sont de type `str`, écrire une fonction Python `mystIter(ch, ch1, ch2)` qui soit une version itérative de la fonction `mystere()` ci-dessus (elle aura donc les mêmes arguments et renverra un résultat similaire). Attention, vous ne pouvez utiliser que les opérateurs vus sur les chaînes de caractères, vous ne pouvez utiliser aucune méthode de chaînes ou de listes.
5. Qu'obtient-on si on exécute :
   (a) `print(mystere('pomme','m',''))`
   (b) `print(mystere('pomme','o','****'))`
   (c) `print(mystere('pomme','mm','t'))`
6. Dans l'hypothèse où `ch`, `ch1` et `ch2` sont de type `str`, corriger la fonction récursive ci-dessus `mystere(ch, ch1, ch2)` pour qu'elle fonctionne dans tous les cas de figure incluant ceux de la question précédente. Attention, vous n'avez toujours pas le droit d'utiliser des méthodes de chaînes (ni des méthodes de listes).

Exercice I.4

Le mont Kelut est un volcan d'Indonésie. Du fait de sa fréquentation et de son activité volcanique fréquente, il présente un risque géologique majeur. Le fichier texte ci-dessous, nommé `"eruptionKelut.txt"`, rassemble les différentes années d'éruption de ce volcan depuis 1311. On supposera que ce fichier est stocké dans le répertoire de travail courant de Python.

```
1641;1901;1951;1752
1864;2007;1771;1450
1990;1851;1451;1411
1311;1548;1376;1826
1967;1835;2014;1385
1966;1395;1586;1716
1776;1825;1848;1334
1811;1920;1481;1968
1462;1785;1919
```

1. Écrire en Python une fonction `Extraction()` qui prend en argument le nom du fichier et renvoie deux listes : une liste contenant toutes les années rangées par ordre croissant et une liste contenant les durées entre deux éruptions consécutives.
2. Écrire une fonction Python `probaEcart()` qui prend en argument la liste des écarts et une valeur d'écart e. Cette fonction calcule et renvoie la probabilité d'avoir un écart de valeur e. Nous vous rappelons que la probabilité d'un événement est définie par le nombre de cas favorables sur le nombre total de cas.
3. On souhaite partitionner l'ensemble des valeurs des écarts en classes : la première classe contiendra toutes les valeurs appartenant à l'intervalle $[0, p[$, la deuxième classe les valeurs de l'intervalle $[p, 2p[$, ... La fréquence d'une classe est alors le nombre d'occurrences des valeurs appartenant à cette classe.
   Écrire une fonction `classeFrequence()` qui prend en argument la liste des écarts et le paramètre p, et renvoie la fréquence de chacune des classes.
4. Écrire le programme principal en Python qui :
   — affiche la fréquence de chacune des classes pour `p=5` selon le format suivant :

```
classe [0,5[ : 6
classe [5,10[ : 5
classe [10,15[ : 5
classe [15,20[ : 6
classe [20,25[ : 2
classe [25,30[ : 1
classe [30,35[ : 1
classe [35,40[ : 4
classe [40,45[ : 1
classe [45,50[ : 0
classe [50,55[ : 0
classe [55,60[ : 1
classe [60,65[ : 0
classe [65,70[ : 1
classe [70,75[ : 0
classe [75,80[ : 1
```

— calcule et affiche la probabilité qu'il y ait une éruption du mont Kelut en 2019 ;
— demande à l'utilisateur s'il y a eu une éruption en 2019, et si oui modifie le fichier `"eruptionKelut.txt"` pour ajouter cette nouvelle année ;
— et qui indique, toujours dans le cas d'une éruption en 2019, quelle classe a vu sa fréquence modifiée pour `p=5`.

Exercice I.5

Un voyageur doit se rendre dans n villes (indicées de 1 à n) et souhaite trouver un itinéraire limitant le nombre de kilomètres parcourus. Ce voyageur part de la ville 1 et, une fois toutes les villes visitées, doit revenir dans la ville 1. Son itinéraire doit lui permettre de passer par toutes les villes **une et une seule fois**. On supposera que quelles que soient deux villes i et j, la distance de i vers j est égale à la distance de j à i.

Considérons un exemple avec 5 villes (numérotées de 1 à 5) et le tableau des distances entre les villes ci-dessous (les numéros des lignes et des colonnes sont en italique et les distances sont en gras) :

	1	*2*	*3*	*4*	*5*
1	**0**	**15**	**10**	**30**	**40**
2	**15**	**0**	**10**	**20**	**35**
3	**10**	**10**	**0**	**20**	**35**
4	**30**	**20**	**20**	**0**	**25**
5	**40**	**35**	**35**	**25**	**0**

Dans ce tableau, l'élément situé ligne i colonne j représente la distance en kilomètres séparant la ville i de la ville j.

Pour déterminer un itinéraire satisfaisant, la stratégie, appelée « plus proche voisin », doit être utilisée : à partir d'une ville donnée, visiter la ville la plus proche non encore visitée et, quand toutes les villes ont été visitées, revenir à la ville de départ 1. Avec cette stratégie, on trouve dans l'exemple à 5 villes l'itinéraire suivant : 1-3-2-4-5-1 faisant 105 km au total.

1. Écrire en **pseudo-code** une fonction `lectureDonnees()` qui prend en paramètre le nombre de villes à visiter et lit les distances entre ces villes. Cette fonction renvoie le tableau des distances. Ces informations sont saisies au clavier par un utilisateur lors d'un dialogue de la forme suivante :

```
Distance entre la ville 1 et la ville 2 : 15
Distance entre la ville 1 et la ville 3 : 10
Distance entre la ville 1 et la ville 4 : 30
Distance entre la ville 1 et la ville 5 : 40
Distance entre la ville 2 et la ville 3 : 10
Distance entre la ville 2 et la ville 4 : 20
Distance entre la ville 2 et la ville 5 : 35
Distance entre la ville 3 et la ville 4 : 20
Distance entre la ville 3 et la ville 5 : 35
Distance entre la ville 4 et la ville 5 : 25
```

2. Écrire en **pseudo-code** une fonction `itineraire()` qui prend en argument le nombre de villes et le tableau des distances et qui applique la stratégie « plus proche voisin » pour renvoyer un itinéraire sous la forme d'un tableau donnant l'ordre de visite des villes. Par exemple, l'itinéraire 1-5-2-4-3-1 est représenté par le tableau [1,3,5,4,2] (la ville 1 est visitée en premier, la ville 2 est visitée en 3e, la ville 3 est visitée en 5e…).
3. Écrire un programme principal qui appelle ces fonctions pour :
   — lire les données saisies au clavier par un utilisateur ;
   — calculer et afficher l'itinéraire et la distance totale parcourue de cet itinéraire.

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

J. TP noté 2017-2018

Description générale

Le morpion est un jeu de réflexion se pratiquant à deux joueurs sur une grille à n lignes et n colonnes. Chaque joueur joue à tour de rôle. Durant son tour, un joueur inscrit son symbole dans une case libre de la grille. Le gagnant est celui qui, le premier, a aligné sur une même ligne, une même colonne ou une même diagonale principale son symbole sur toutes les cases.

Vous devez programmer ce jeu pour pouvoir faire jouer un joueur contre l'ordinateur : le joueur a le symbole X et l'ordinateur le symbole O.

À la fin du TP, votre programme devra :

- représenter le plateau de jeu ;
- demander au joueur de choisir une case libre pour y placer son symbole X sur le plateau et afficher le plateau ;
- terminer la partie si le joueur a réussi à aligner n symboles X en ligne, en colonne ou en diagonale : le joueur est alors déclaré gagnant ;
- sinon faire jouer l'ordinateur en choisissant une case libre et afficher le plateau ;
- si l'ordinateur a pu aligner n symboles O en ligne, en colonne ou en diagonale, terminer la partie en déclarant le joueur perdant ; sinon recommencer un nouveau tour tant qu'il reste des cases vides dans la grille ; si la grille est remplie alors que ni le joueur, ni l'ordinateur n'aient pu aligner n symboles, alors la partie se termine et est déclarée nulle.

Exemple de partie sur un plateau de taille 3 (les chiffres sont saisis au clavier par l'utilisateur) :

```
-------
| | | |
-------
| | | |
-------
| | | |
-------
A vous de jouer et de choisir une case :
Numero de ligne : 1
Numero de colonne : 1
-------
|X| | |
-------
| | | |
-------
| | | |
-------
```

```
Coup joué par l'ordinateur : (1,3)
-------
|X| |O|
-------
| | | |
-------
| | | |
-------
A vous de jouer et de choisir une case :
Numero de ligne : 2
Numero de colonne : 2
-------
|X| |O|
-------
| |X| |
-------
| | | |
-------
Coup joué par l'ordinateur : (3,1)
-------
|X| |O|
-------
| |X| |
-------
|O| | |
-------
A vous de jouer et de choisir une case :
Numero de ligne : 3
Numero de colonne : 3
-------
|X| |O|
-------
| |X| |
-------
|O| |X|
-------
Bravo ! Vous avez gagné !
```

Travail à réaliser

Commencez par créer un fichier `.py` dont le nom sera de la forme : `PrenomNomGr.py`
Par exemple, Raoul Michou du groupe de TD 2 nommera son fichier `RaoulMichou2.py`.
Vos réponses aux différentes questions ci-dessous seront toutes enregistrées dans ce seul fichier.

Dans tout le programme, le plateau `T` de jeu sera représenté par une liste de `n` sous-listes. Chaque sous-liste de taille `n` représente une rangée du plateau ; chaque élément de ces sous-listes étant `'O'`, `'X'` ou `' '` (espace).

1. Écrire une fonction, appelée `affichePlateau()`, qui prend en argument la taille `n` du plateau et le plateau `T`. Cette fonction ne renvoie rien, elle affiche le plateau. Le plateau vide doit être affiché sous le format suivant (voici un exemple pour $n = 3$) :

```
-------
| | | |
-------
| | | |
-------
| | | |
-------
```

2. Écrire la fonction `coupJoueur()` qui prend en argument `n` et `T` (correspondant au plateau). Cette fonction lit deux valeurs saisies au clavier par le joueur, la première étant le numéro de ligne et la seconde le numéro de colonne. Le programme devra vérifier que ces valeurs correspondent aux coordonnées d'une case libre du plateau. Puis, cette fonction modifie `T` pour y ajouter le symbole du joueur dans la case spécifiée, puis renvoie les indices de cette case. Par exemple, si un appel à `coupJoueur()` renvoie 0,0 alors le plateau obtenu sera :

```
-------
|X| | |
-------
| | | |
-------
| | | |
-------
```

3. Écrire la fonction `coupOrdinateurAlea()` qui prend en argument `n` et `T` et renvoie les indices dans `T` de la case « choisie » par l'ordinateur, c'est-à-dire deux valeurs choisies aléatoirement, la première étant l'indice de ligne et la seconde l'indice de colonne. Ces indices de ligne et de colonne doivent correspondre à une case libre du plateau. Par exemple, si un appel à `coupOrdinateurAlea()` renvoie 1,2, le plateau ensuite obtenu sera :

```
-------
|X| | |
-------
| | |O|
-------
| | | |
-------
```

4. Écrire la fonction `gagne()` qui prend en argument la taille `n` du plateau, le plateau `T`, les coordonnées du dernier coup joué et un symbole. Cette fonction renvoie un booléen qui vaut vrai si `n` symboles sont alignés en colonne, en ligne ou en diagonale, et faux sinon.

   Cette fonction devra faire appel à trois fonctions que vous devez écrire au préalable :
   — la fonction `gagneEnLigne()` qui prend en argument la taille `n` du plateau, le plateau `T`, un indice de ligne et un symbole, qui teste si la ligne contient `n` fois le symbole et qui renvoie une valeur booléenne ;

— la fonction `gagneEnColonne()` qui prend en argument la taille `n` du plateau, le plateau `T`, un indice de colonne et un symbole, qui teste si la colonne contient `n` fois le symbole et qui renvoie une valeur booléenne ;
— la fonction `gagneEnDiagonale()` qui prend en argument la taille `n` du plateau, le plateau `T` et un symbole, qui teste si au moins une des deux diagonales principales contient `n` fois le symbole et qui renvoie une valeur booléenne.

5. Écrire la fonction `partieAlea()` qui prend en argument la taille `n` du plateau et le plateau `T`. Cette fonction appelle les fonctions précédentes pour exécuter une partie de Morpion. À chaque tour, on prendra soin d'afficher le plateau obtenu. Cette fonction renvoie une chaîne de caractères qui conclut la partie : "Bravo ! Vous avez gagné" ou "Perdu ! L'ordinateur a gagné !" ou "Partie nulle !".
6. Écrire le programme principal qui permet de jouer une partie sur un plateau de taille 3 et d'en afficher l'issue.
7. Écrire une fonction `coupOrdinateurOptimise()` qui prend en argument `n` et `T` et renvoie les indices dans `T` de la case « choisie » par l'ordinateur à l'aide d'une stratégie "gagnante". En effet, pour décider du coup à jouer par l'ordinateur, on ne procèdera plus aléatoirement mais on suivra les étapes suivantes :
   — si l'ordinateur peut jouer un coup gagnant, alors jouer ce coup ;
   — sinon
      — si le joueur a la possibilité de jouer un coup gagnant au prochain tour, alors contrer ce coup et jouer un coup permettant d'empêcher le joueur de gagner par ce coup,
      — sinon, jouer un coup aléatoire.

   Cette fonction devra faire appel à trois fonctions que vous devez écrire au préalable :
   — la fonction `enLigne()` qui prend en argument la taille `n` du plateau, le plateau `T`, un indice de ligne et un symbole, et teste si la ligne contient `n-1` fois le symbole indiqué ainsi qu'une case vide ;
   — la fonction `enColonne()` qui prend en argument la taille `n` du plateau, le plateau `T`, un indice de colonne et un symbole, et teste si la colonne contient `n-1` fois le symbole indiqué ainsi qu'une case vide ;
   — la fonction `enDiagonale()` qui prend en argument la taille `n` du plateau, le plateau `T` et un symbole, et teste si au moins une des deux diagonales principales contient `n-1` fois le symbole indiqué ainsi qu'une case vide.

   Chacune de ces fonctions renvoie les coordonnées x, y de la case vide si elle existe, sinon elle renvoie $-1, -1$.

   Écrire la fonction `partieOptimisee()` qui prend en argument la taille `n` du plateau et le plateau `T`. Cette fonction appelle les fonctions précédentes pour exécuter une partie de Morpion avec la stratégie « gagnante » de l'ordinateur. À chaque tour, on prendra soin d'afficher le plateau obtenu. Cette fonction renvoie une chaîne de caractères qui conclut la partie : "Bravo ! Vous avez gagné" ou "Perdu ! L'ordinateur a gagné !" ou "Partie nulle !". Modifier votre programme principal pour exécuter une partie de morpion avec l'ordinateur et sa stratégie gagnante.

Rendu du TP

À la fin de la séance de TP noté, faire une copie de votre fichier `PrenomNomGr.py` contenant tout votre travail dans le répertoire `EXAM/DEMI2E_L1_AlgoProg/NumSalle` où *Num-*

Salle est le numéro de la salle dans laquelle vous travaillez. Ainsi, Raoul Michou du groupe de TD 2 actuellement en B042 devra coller son fichier `RaoulMichou2.py` dans le répertoire `EXAM/DEMI2E_L1_AlgoProg/B042`.

Attention, vous ne pourrez réaliser cette opération qu'une seule fois, car une fois votre fichier `.py` copié dans le repertoire `EXAM/DEMI2E_L1_AlgoProg/NumSalle`, vous ne pourrez plus le modifier, ni l'effacer de ce répertoire. C'est **donc UNIQUEMENT à la fin du TP** que vous devez coller la version finale (celle qui sera corrigée) de votre programme, dans le répertoire `EXAM/DEMI2E_L1_AlgoProg/NumSalle`.

Université Paris Dauphine **Licence MIE 1re année**
Algorithmique et programmation 1 **2019-2020**

K. TP noté 2018-2019

Description générale

Étant donné un réseau social, l'objectif est d'identifier un groupe d'amis. Pour cela, on dispose de la description suivante du réseau :

— la liste des n personnes du réseau (indicées de 0 à $n-1$) ;
— la liste des amis de chaque personne i, pour i allant de 0 à $n-1$.

À la fin du TP, votre programme devra :

— lire un réseau social ;
— déterminer les individus appartenant à un des plus grands groupes d'amis.

Voici un exemple de réseau social contenant 8 personnes :

Les relations d'amitiés sont :
— les amis de 0 : 2, 3 ;
— les amis de 1 : 3, 5 ;
— les amis de 2 : 0, 3, 6, 7 ;
— les amis de 3 : 0, 1, 2, 4, 5, 6 ;
— les amis de 4 : 3, 7 ;
— les amis de 5 : 1, 3, 6, 7 ;
— les amis de 6 : 2, 3, 5 ;
— les amis de 7 : 2, 4, 5.

Une représentation graphique de ce réseau est proposée dans la figure 2.

On suppose que la relation d'amitié liant deux personnes est **symétrique** : si i est l'ami de j, alors j est l'ami de i.

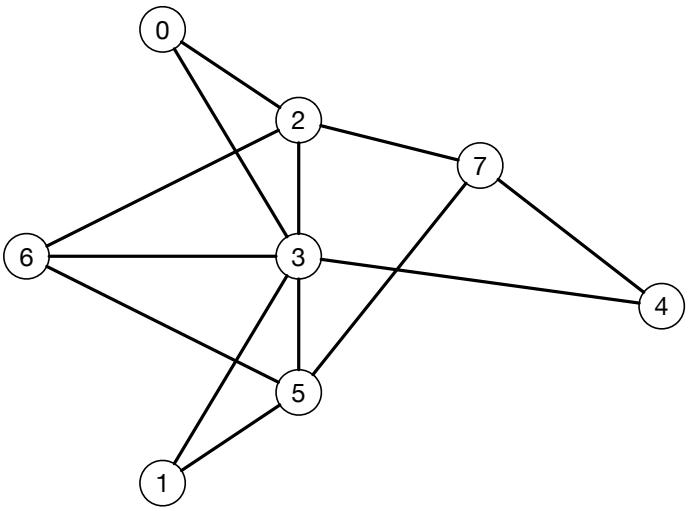


FIGURE 2 – Réseau à 8 personnes

Travail à réaliser

Commencer par créer un fichier `.py` dont le nom sera de la forme `PrenomNomSalle.py`, où `Salle` est le numéro de la salle dans laquelle vous réalisez le TP noté.

PENSEZ À SAUVEGARDER RÉGULIEREMENT VOTRE TRAVAIL PENDANT LA SÉANCE !

1. Écrire une fonction `saisirListeAmis()` qui prend le numéro d'une personne en argument, lit les numéros de ses amis saisis au clavier par l'utilisateur et renvoie la liste de ses amis.
2. Écrire une fonction `initReseau()` qui lit le nombre n de personnes dans le réseau (cette valeur est saisie au clavier par l'utilisateur) et renvoie deux valeurs : la taille n du réseau et une liste de n éléments, chaque élément d'indice i est la liste d'amis de la personne i. Cette fonction ne prend pas d'argument et doit faire appel à la fonction `saisirListeAmis()`. Pour le réseau décrit dans la figure 2, les valeurs renvoyées seraient :

   ```
   8, [[2, 3], [3, 5], [0, 3, 6, 7], [0, 1, 2, 4, 5, 6],
   [3, 7], [1, 3, 6, 7], [2, 3, 5], [2, 4, 5]]
   ```

   Par la suite, un réseau `r` sera toujours décrit sous la forme d'une liste de listes d'amis.
3. Écrire une fonction booléenne `estAmis()` qui prend un réseau `r` et deux numéros de personnes i et j en argument, et renvoie une valeur booléenne qui vaut vrai si i et j sont amis dans `r` et faux sinon.
4. Un groupe d'amis est un ensemble de personnes toutes amies 2 à 2. On veut déterminer un groupe d'amis contenant i. On utilise l'algorithme suivant :
   (a) À l'initialisation, le groupe ne contient que i.
   (b) Puis, on parcourt la liste des amis de i. Soit a l'ami considéré à l'itération courante : si a est ami avec toutes les personnes déjà incluses dans le groupe alors a est ajouté au groupe, sinon on passe à l'ami suivant dans la liste des amis de i.

En utilisant cet algorithme, écrire une fonction `groupeAmis()` qui prend en argument un réseau `r` et le numero i d'une personne et renvoie la liste d'un groupe d'amis auquel i appartient.

5. Écrire une fonction `grandGroupe()` qui prend en argument un réseau `r` et renvoie la liste des individus faisant partie d'un des plus grands groupes d'amis.
6. Écrire une fonction `lireFichier()` qui lit un fichier texte contenant la description d'un réseau sous la forme suivante :

```
Nombre de personnes : 8
Amis de 0 : 2 3
Amis de 1 : 3 5
Amis de 2 : 0 3 6 7
Amis de 3 : 0 1 2 4 5 6
Amis de 4 : 3 7
Amis de 5 : 1 3 6 7
Amis de 6 : 2 3 5
Amis de 7 : 2 4 5
```

   Cette fonction prend en argument un nom de fichier et renvoie le nombre de personnes dans le réseau et une liste de listes d'amis décrivant le réseau.
7. Écrire le programme principal qui propose à l'utilisateur deux modes d'exécution :
   (a) lire un réseau avec des saisies clavier ou
   (b) lire un réseau décrit dans un fichier texte.

   Une fois le réseau lu, ce programme devra tester la symétrie de la relation d'amitié : si la symétrie est vérifiée, il permettra d'afficher à l'écran les amis appartenant au plus grand groupe d'amis, sinon il affichera les couples des personnes pour lesquels la relation d'amitié n'est pas symétrique.
8. Effectuer, dans votre répertoire personnel, une copie du fichier `reseau100.txt` disponible dans `ETUD/DEMI2E_L1_AlgoProg`. Exécuter votre programme pour afficher les amis appartenant au plus grand groupe d'amis dans le réseau décrit dans le fichier `reseau100.txt`.

Rendu du TP

À la fin de la séance de TP noté, **faire une copie de votre fichier** `PrenomNomSalle.py` (par copier/coller et **pas** par glisser/déposer) contenant tout votre travail dans le répertoire nommé `EXAM/DEMI2E_L1_AlgoProg/NumSalle` où *NumSalle* est le numéro de la salle dans laquelle vous travaillez.

Ainsi, Raoul Michou actuellement en B042 devra copier son fichier `RaoulMichouB042.py` dans le répertoire `EXAM/DEMI2E_L1_AlgoProg/B042`.

Attention, vous ne pourrez réaliser cette opération qu'une seule fois, car une fois votre fichier `.py` copié dans le repertoire `EXAM/DEMI2E_L1_AlgoProg/NumSalle`, vous ne pourrez plus le modifier, ni l'effacer de ce répertoire. C'est **donc UNIQUEMENT à la fin du TP** que vous devez coller la version finale (celle qui sera corrigée) de votre programme dans le répertoire `EXAM/DEMI2E_L1_AlgoProg/NumSalle`.