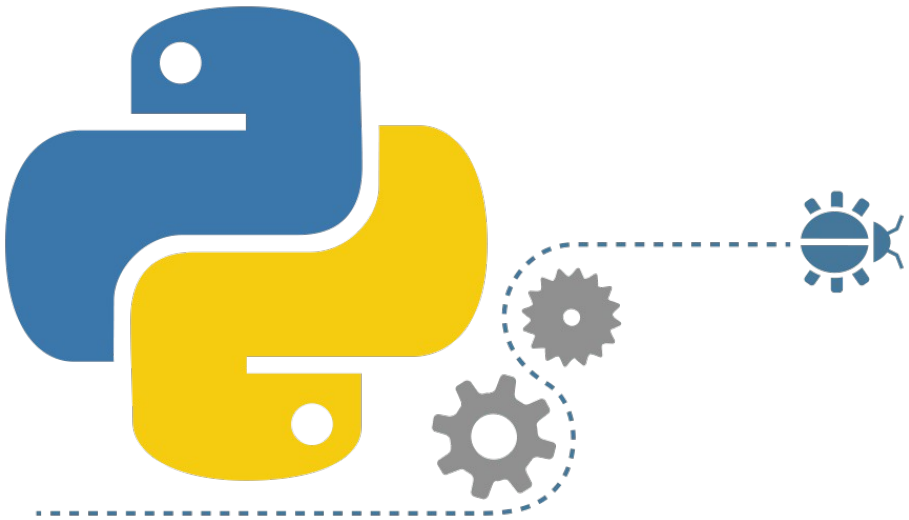
Programmation avec Python3

Exercices



D.Carrera

Sommaire

Notes

Tous les exercices seront construits selon le même schéma :

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# NOM Prénom Classe

# fichier.py – date jj.mm.aaaa

# Votre code source ...
```

Entrée utilisateur

Dans les exemples d'exécution de script, les entrées seront notées en italique et entre chevrons.

Exemple

```
nom = input("Donner votre prénom : ")
```

Nous afficherons :

```
Donner votre prénom : <Gérard>
```

Les chevrons « < » et « > » servent simplement à délimiter l'information tapée au clavier. Dans cet exemple, c'est la chaîne de caractères `"Gérard"` qui sera enregistrée dans la variable `nom`.

Il s'agit d'une présentation de l'information entrée par l'utilisateur. Lorsque vous testerez votre code, vous n'écrirez pas les chevrons !

Exercices

Introduction

Consignes

Créez un dossier **`introduction`** dans votre dossier **`programmation`**. Tous les scripts de cette section y seront sauvegardés.

Enregistrez chaque fichier de cette section sous le nom **`introXX.py`** où **`XX`** sera remplacé par le numéro de l'exercice écrit sur deux chiffres.

Par exemple, l'exercice 7 sera enregistré sous le nom **`intro07.py.`**

Exercices

1. Créer un script qui demande à l'utilisateur d'entrer son nom. Le script affichera un message de bienvenue à l'utilisateur.

Exemple

```
Donner votre prénom : <Gérard>
Bonjour Gérard !
Et votre nom ? <Manvusa>
Bonne journée ! Au revoir Gérard Manvusa.
```

2. Créer un script qui demande à l'utilisateur d'entrer un nombre quelconque. Le script affichera le carré ainsi que le cube de ce nombre.

Exemple

```
Entrer un nombre : <12>
Le carré de 12 vaut 144.
Le cube de 12 vaut 1728.
```

3. Créer un script qui demande deux nombres quelconques à l'utilisateur. Le script affichera le résultat des quatre opérations fondamentales et du modulo.

Exemple

```
Donner un premier nombre : <23>
Donner un deuxième nombre : <14>
Addition : 23 + 14 = 37
Soustraction : 23 - 14 = 9
Multiplication : 23 x 14 = 322
Division : 23 / 14 = 1.6428571428571428
Division entière : 23 // 14 = 1
Modulo : 23 % 14 = 9
```

4. Créer un script capable de transformer un nombre de secondes en jour(s) heure(s) minute(s) seconde(s).

Exemple

```
Donner un nombre (en secondes) : <7531>
7531 secondes = 0 jour(s) 2 heure(s) 5 minute(s) 31 seconde(s).
```

5. Créer un script capable de calculer le volume d'une sphère à partir du rayon donné par l'utilisateur. Le rayon pourra être fourni sous forme d'un nombre décimal.

La variable **pi** se trouve dans le module **math**. Importez-la.

Exemple d'importation

```
from math import pi
# Calcul de la surface
aire = pi * rayon ** 2
```

Ensuite, utiliser la variable **pi** comme une variable classique. La formule du calcul du volume de la sphère est $4/3.\prod.r^3$.

Exemple

```
Entrez le rayon de la sphère : <1.24>
Le volume de la sphère est de 7.986447935410647.
```

6. Écrire un programme qui demande à l'utilisateur de taper la largeur et la longueur d'un champ et qui en affiche le périmètre et la surface.

Exemple

```
Entrez la longueur du champs (en m): <1.4>
Entrez la largeur du champs (en m): <2.54>
Le périmètre du champs est de 7.88
La superficie du champs est de 3.5559999999999996
```

7. Écrire un programme qui demande à l'utilisateur de taper cinq entiers et qui affiche leur moyenne. Le programme ne devra utiliser que deux variables.

Exemple

```
Entrez un nombre : <12>
Entrez un nombre : <65>
Entrez un nombre : <78>
Entrez un nombre : <49>
Entrez un nombre : <34>
La moyenne des 5 nombres entrés est de 47.6.
```

8. Écrire un programme qui demande à l'utilisateur de saisir deux entiers **a** et **b**, qui échange le contenu des variables **a** et **b** puis qui affiche **a** et **b**.

Exemple

```
Entrez le nombre A : <4>
Entrez le nombre B : <9>
AVANT échange, A : 4; B : 9
APRÈS échange, A : 9; B : 4
```

9. Écrire un programme qui demande à l'utilisateur de taper le prix HT d'un kilo de tomates, le nombre de kilos de tomates achetés, le taux de TVA (Exemple 5.5, 19.6,...). Le programme affiche alors le prix TTC des marchandises.

Exemple

```
Donnez le prix hors taxe : <2.99>
Donnez le nombre de kilos achetés : <1.2>
Donnez le montant de la T.V.A. (en %): <6>
Le prix TTC est de 3.80328.
```

10. Écrire un programme qui demande à l'utilisateur de saisir les coordonnées de deux points du plan **A** et **B** et qui affiche la distance entre **A** et **B**.

N.B. 1 : la distance entre deux points du plan est calculée par la formule suivante :

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

N.B. 2 : utiliser le module **math** pour importer la fonction **sqrt()** qui calcule la racine carrée (*square root*, en anglais).

Exemple d'importation

```
from math import sqrt
# Calcul de la racine carrée de y
x = sqrt(y)
```

Le résultat renvoyé par la fonction **sqrt()** est de type 'flottant'.

Exemple

```
Donnez l'abscisse du point A : <1>
Donnez l'ordonnée du point A : <5>
Donnez l'abscisse du point B : <7>
Donnez l'ordonnée du point B : <9>
La distance entre les points (1,5) et (7,9) est de 7.21.
```

Alternative

Consignes

Créer un dossier **`alternative`** dans votre dossier **`programmation`**. Tous les scripts de cette section y seront sauvegardés.

Tous les scripts de cette section y seront sauvegardés sous le nom **`alterXX.py`** où **`XX`** est le numéro de l'exercice écrit sur deux chiffres.

Par exemple, l'exercice 7 sera enregistrer sous le nom **`alter07.py.`**

Exercices

1. Créer un script qui vérifie que le nombre entré par l'utilisateur est bien compris entre 0 et 19 inclus. Afficher un message comme dans l'exemple ci-dessous.

   Exemples

```
Donner un nombre compris entre 0 et 19 : <23>
ERREUR : le nombre devait être compris entre 0 et 19 inclus !
```

```
Donner un nombre compris entre 0 et 19 : <5>
OK. Merci !
```

2. Créer un script qui indiquera si le nombre entré par l'utilisateur est une nombre pair ou impair.

   Exemples

```
Donner un nombre : <23>
Le nombre 23 est impair.
```

```
Donner un nombre : <256>
Le nombre 256 est pair.
```

3. Créer un script qui déterminera si le nombre entré par l'utilisateur est multiple de 2, de 3 ou de 2 et de 3 ou encore s'il n'est ni un multiple de 2 ni un multiple de 3.

   Afficher également la décomposition du nombre.

   Exemples

```
Donner un nombre entier : <27>
Le nombre 27 est multiple de 3 (27 = 3 x 9).
```

```
Donner un nombre entier : 144
Le nombre 144 est multiple de 2 et de 3 (144 = 2 x 72 et
144 = 3 x 48).
```

```
Donner un nombre entier : 58
Le nombre 58 est multiple de 2 (58 = 2 x 29).
```

```
Donner un nombre entier : 13
Le nombre 13 n'est ni multiple de 2 ni multiple de 3.
```

4. Signaler à l'utilisateur que le nombre qu'il a entré est soit positif, soit négatif, soit nul.

   Exemples

```
Entrer un nombre quelconque : <23>
Le nombre 23 est positif.
```

```
Entrer un nombre quelconque : <-3>
Le nombre -3 est négatif.
```

```
Entrer un nombre quelconque : <0>
Vous avez entré zéro!
```

5. Écrivez un programme qui lit trois valeurs entières (A, B et C) au clavier et qui affiche la plus grande des trois valeurs.

   Exemple

```
Entrez la valeur associée à A : <13>
Entrez la valeur associée à B : <43>
Entrez la valeur associée à C : <12>
La valeur la plus grande se trouve dans B.
```

6. Écrivez un programme qui lit trois valeurs entières (A, B et C) au clavier. Triez les valeurs A, B et C.

   Échangez les valeurs de A, B et C de manière à ce que A contienne la valeur la plus petite, B la valeur moyenne et C la plus grande des trois valeurs.

   Exemple

```
Entrez la valeur associée à A : <13>
Entrez la valeur associée à B : <43>
Entrez la valeur associée à C : <12>
Les valeurs non triées : A(=13), B(=43), C(=12)
Les valeurs triées par ordre croissant : A(=12), B(=13), C(=43)
```

7. Écrivez un programme qui lit deux valeurs entières (A et B) au clavier et qui affiche le signe du produit de A et B sans faire la multiplication.

   Exemple

```
Entrez la valeur associée à A : <4>
Entrez la valeur associée à B : <6>
Le signe de la multiplication est positif.
```

```
Entrez la valeur associée à A : <-3>
Entrez la valeur associée à B : <7>
Le signe de la multiplication est négatif.
```

```
Entrez la valeur associée à A : <-8>
Entrez la valeur associée à B : <-1>
Le signe de la multiplication est positif.
```

8. x

Répétitive

Consignes

Créer un dossier **`repetitive`** dans votre dossier **`programmation`**.

Tous les scripts de cette section y seront sauvegardés sous le nom **`repetXX.py`** où **`XX`** est le numéro de l'exercice écrit sur deux chiffres.

Par exemple, l'exercice 7 sera enregistré sous le nom **`repet07.py`**.

Exercices

1. (*) Créer un script qui affichera les 20 premiers termes de la table de multiplication d'un nombre quelconque. Ce nombre sera préalablement entré par l'utilisateur.

Exemple

```
Quelle table de multiplication désirez-vous afficher ? <7>
Table de multiplication par 7
1 x 7 = 7
2 x 7 = 14
[…]
20 x 7 = 140
Termine !
```

2. (**) Créer un script qui affichera une suite de 12 nombres dont chaque terme soit égal au triple du terme précédent. Demander à l'utilisateur le nombre de départ.

Attention, tous les nombres doivent être affichés sur la même ligne !

Exemple

```
Donner le nombre de départ : <11>
11 33 99 297 891 2673 8019 24057 72171 216513 649539 1948617
```

3. (*) Créer un script qui affichera une table de conversion de sommes d'argent exprimées en euros, en dollars canadiens. La progression des sommes de la table sera « géométrique », comme dans l'exemple ci-dessous.

Afficher les euros sur 8 caractères, les dollars canadiens sur 8 caractères dont deux derrières la virgule. Utiliser les chaînes de format !

Exemple

```
1 EUR (euro(s)) =  1.54 CAD (dollar(s) canadien(s))
2 EUR (euro(s)) =  3.08 CAD (dollar(s) canadien(s))
```

```
4 EUR (euro(s)) =  6.16 CAD (dollar(s) canadien(s))
8 EUR (euro(s)) = 12.32 CAD (dollar(s) canadien(s))
etc. (S'arrêter à 16384 EUR.)
```

4. (**) Créer un script qui simulera le jeu suivant : un nombre est tiré aléatoirement entre 1 et 6. À l'utilisateur de le deviner en utilisant le moins de coups possibles.

Pour tirer un nombre aléatoire, il faut faire appel à la fonction **`randrange()`** de la bibliothèque **`random`** de Python. Importez-la.

Exemple d'importation

```
from random import randrange
# Tirage d'un nombre aléatoire compris entre 1 et 6.
aleat = randrange(1,7)
```

Exemple

```
Devinez le nombre tiré : <4>
Ce nombre ne correspond pas, devinez encore : <1>
Ce nombre ne correspond pas, devinez encore : <5>
Vous avez deviné le nombre 5 en 3 coups !
```

5. (*) Construire un script capable d'additionner un ensemble de notes données au fur et à mesure. Le nombre de notes à encoder sera défini dès le départ par l'utilisateur. Une fois la dernière note entrée, le programme affichera la somme ainsi que la moyenne arithmétique de toutes les notes.

Exemple

```
Combien de notes à entrer ? <4>
Entrez la note no 1 : <7>
Entrez la note no 2 : <4>
Entrez la note no 3 : <9>
Entrez la note no 4 : <2>
La somme de ces notes est de 22. La moyenne est de 5.5.
```

6. (*) Calculer la **factorielle** **`n! = 1.2.3. ... .(n-1).n`** d'un entier naturel **`n`** en respectant le fait que 0 ! = 1.

Par exemple, la factorielle de 7 s'écrit 7! et vaut 1.2.3.4.5.6.7 = 24.

Exemple

```
Calculer la factorielle de quel nombre ? <14>
14! = 1.2.3.4.5.6.7.8.9.10.11.12.13.14 = 87178291200
```

N.B. : pour afficher la suite de multiplications, construisez une chaîne de caractères. L'utilisation seule de l'instruction **`print()`** n'est pas possible sans l'utilisation de l'instruction alternative (qui n'a pas encore été vue!).

7. (*) Calculer le énième terme U_n de la **suite de Fibonacci** qui est donnée par la relation de récurrence:

$$U_1 = 1 ; U_2 = 1 ; U_n=U_{n-1} + U_{n-2} \text{ (pour } n > 2)$$

Les deux premiers termes de la suite valent 1. Les suivants sont calculés en additionnant les deux termes précédents. L'utilisateur entrera au minimum le nombre 2.

Exemple

```
Suite de Fibonacci, combien de termes voulez-vous afficher ? <10>
1 1 2 3 5 8 13 21 34 55
```

8. (**) Afficher un **triangle rectangle** formé d'étoiles et un triangle isocèle formé d'étoiles à sa suite. La hauteur du triangle (c'est à dire le nombre de lignes) sera fourni par l'utilisateur, comme dans l'exemple ci-dessous.

Exemple

```
Combien de lignes ? <5>
Triangle Rectangle
*
***
*****
*******
*********
Triangle Isocèle
    *
   ***
  *****
 *******
*********
```

N.B. : Dans cet exercice, nous jouons avec les caractères **" "** (espace) et **"*"** (étoile).

9. (**) Calculez **par des soustractions successives** le quotient entier et le reste de la division entière de deux nombres entiers entrés au clavier.

Exemple

```
Donner le dividende : <47>
Donner le diviseur : <9>
47 = 9 x 5 + 2
```

N.B. : Dans l'exemple ci-dessus, nous devons soustraire 9 à 47 un certain nombre de fois (dans ce cas-ci, 5x) jusqu'à ce que le résultat de la soustraction soit plus petit que le diviseur.

10. (*) Calculez par multiplications successives X^N de deux entiers naturels X et N entrés au clavier.

Exemple

```
Donner un nombre : <11>
Donner une puissance : <5>
Le résultat de 11^5 = 161051
```

N.B. : Le résultat obtenu, **161051**, est calculé comme **11.11.11.11.11**.

Instruction `for`

1. Demander un nombre compris entre 0 et 20 non inclus à l'utilisateur. Afficher les vingt premiers termes de la table de multiplication du nombre donné.

   Exemple

```
Entrer un nombre compris entre 0 et 20 non inclus : <11>
1 x 11 = 11
2 x 11 = 22
3 x 11 = 33
[ … ]
19 x 11 = 209
20 x 11 = 220
```

2. Affichez, en ligne, les entiers de 0 à 45 compris et multiple de trois, en utilisant uniquement une boucle **`for`** et la fonction **`range()`**.

   Exemple

```
0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45
```

3. x

Entraînement

Consignes

Créer un dossier **`entrainement`** dans votre dossier **`programmation`**. Tous les scripts de cette section y seront sauvegardés.

Tous les scripts de cette section y seront sauvegardés sous le nom **`trainingXX.py`** où **`XX`** est le numéro de l'exercice écrit sur deux chiffres.

Par exemple, l'exercice 7 sera enregistré sous le nom **`training07.py`**.

Exercices

1. Écrire un programme qui trouve la plus grande et la plus petite valeur d'une succession de notes (nombres entiers compris entre 0 et 20) fournies par l'utilisateur, ainsi que le nombre de fois où ce maximum et ce minimum ont été attribués.

   On supposera que les notes, en nombre non connu à l'avance, seront terminées par une valeur négative.

   Exemple

```
Donnez une note (-1 pour terminer) : <8>
Donnez une note (-1 pour terminer) : <13>
Donnez une note (-1 pour terminer) : <7>
Donnez une note (-1 pour terminer) : <12>
Donnez une note (-1 pour terminer) : <7>
Donnez une note (-1 pour terminer) : <-1>
Note maximale : 13 attribuée 1 fois
Note minimale : 7 attribuée 2 fois
```

2. Afficher tous les nombres premiers compris entre 2 et 100 inclus.

3. Afficher les nombres de 1 à 30 inclus, en ajoutant deux étoiles « ** » à côté des nombres multiples de 2, trois étoiles « *** » à côté des nombres multiples de 3 et cinq étoiles « ** *** » à côté des nombres multiples de 2 et de 3.

   Exemple

```
1
2 **
3 ***
4 **
etc.
29
30 ** ***
```

4. Écrire un script qui permet d'afficher la racine carrée des nombres compris entre 16 et 225 inclus qui sont des carrés parfaits.

   Exemple

```
Carré parfait 16; racine carrée 4
Carré parfait 25; racine carrée 5
Carré parfait 36; racine carrée 6
Carré parfait 49; racine carrée 7
Carré parfait 64; racine carrée 8
Carré parfait 81; racine carrée 9
Carré parfait 100; racine carrée 10
Carré parfait 121; racine carrée 11
Carré parfait 144; racine carrée 12
Carré parfait 169; racine carrée 13
Carré parfait 196; racine carrée 14
Carré parfait 225; racine carrée 15
```

5. Écrire un script qui permet de tirer 100 nombres au hasard, compris entre 0 et 99 inclus, et qui affiche le **pourcentage de nombres impairs** tirés.

   De plus, vous afficherez également le pourcentage de nombres impairs multiple de 3 et de 5 ainsi que le pourcentage restant.

   Attention, les nombres impairs peuvent être à la fois multiples de 3 et de 5, dans ce cas n'oubliez pas de les compter comme multiple de 3 et comme multiple de 5.

   Exemple

```
45.0% des 100 nombres tirés au hasard entre 0 et 99 inclus sont
impairs.

Statistiques
------------
* 33.3% des nombres tirés sont multiples de 3 ;
* 26.7% des nombres tirés sont multiples de 5 ;
* 40.0% des nombres ne sont ni multiples de 3 ni multiples de 5.
```

6. Calculez le P.G.C.D. de deux entiers naturels entrés au clavier en utilisant l'algorithme d'EUCLIDE.

   Algorithme d'Euclide

   Vérifier d'abord que le premier entier entré (appelons-le *a*) est plus grand ou égal au deuxième (appelons-le *b*). Si ce n'est pas le cas, inverser les valeurs.

   On commence par calculer le reste de la division de *a* par *b*, qu'on note *r* ; puis on remplace *a* par *b*, puis *b* par *r*, et on réapplique le procédé depuis le début.

   On obtient ainsi une suite, qui vaut 0 à un certain rang ; le PGCD cherché est le terme précédent de la suite.

   Exemple

```
Entrez un premier nombre : <1071>
Entrez un deuxième nombre : <1029>
Le PGCD de 1071 et de 1029 à l'aide de l'algorithme d'Euclide :
1071 = 1029 ×  1 + 42
1029 =   42 × 24 + 21
  42 =   21 ×  2 +  0
PGCD(1071 ; 1029) = 21.
```

Chaîne de caractères

Consignes

Créer un dossier **`string`** dans lequel vous sauvegarderez tous les exercices qui suivront.

Tous les scripts de cette section y seront sauvegardés sous le nom **`stringXX.py`** où **`XX`** est le numéro de l'exercice écrit sur deux chiffres.

Par exemple, l'exercice 7 sera enregistrer sous le nom **`string07.py.`**

Les chaînes de caractères que vous utiliserez pour tester vos script ne devront pas contenir de caractères spéciaux ni de caractères accentués, sauf mention explicite.

Exercices

1. Réaliser un script Python qui affiche chacune des lettres d'un mot sur une ligne.
   Exemple

```
Entrez une chaîne : <Paul>
P
a
u
l
```

2. Même chose que le script précédent. Cependant, le script affichera les lettres les unes à côté des autres et *en commençant par la fin*, elles seront séparées par un espace.
   Exemple

```
Entrez une chaîne : <Paul>
l u a P
```

3. Écrire un programme qui compte le nombre de lettre « e » dans une phrase donnée.

4. Réaliser un programme qui compte les voyelles dans une phrase donnée.

5. Réaliser un programme qui permettra de supprimer les espaces blancs dans une phrase.
   Exemple

```
Entrez une phrase : <La vie est belle>
Sans espace : Lavieestbelle.
```

6. Réaliser un programme qui permet de tester les messages SMS destinés à différentes filiales d'une grande société. Si le SMS commence par les lettres **`lg`** il est destiné à la filiale liégeoise, si **`bx`** à la bruxelloise, si **`nm`** à la namuroise et **`lu`** à la luxembourgeoise. Sinon le programme signalera une erreur mentionnant que l'extension de la filiale est inconnue.

7. Écrire un script qui comptera le nombre de consonnes doubles présentes dans un mot donné.
   Exemple

```
Entrez un mot : <cannelle>
Le mot "cannelle" contient 2 consonnes doubles.
```

8. Écrire un script qui affichera le nombre de syllabes présentes dans un mot donné.
   Pour l'exercice, nous simplifierons quelque peu les règles de division des mots en syllabe :
   - ✔ Une consonne placée après une voyelle introduit une nouvelle syllabe. *Exemple : in/for/mer → 3 syllabes.*
   - ✔ Deux voyelles consécutives comptent pour la même syllabe,
     Exemple : mai/rie → 2 syllabes, chan/de/lier → 3 syllabes, poi/son/ne/rie → 4 syllabes...
     1. SAUF si la deuxième voyelle est un « o » ou un « a » ;
        Exemple : li/on → 2 syllabes, bi/o/lo/gie → 4 syllabes, co/a/sser → 3 syllabes…
     2. ATTENTION, les terminaisons en « -sion » ou « -tion » ne comptent que pour une syllabe.
        Exemple : at/ten/tion → 3 syllabes, mi/ssion → 2 syllabes...

9. Écrire un script qui mélange les lettres d'un mot donné. Le mot sera composé uniquement de lettres différentes !
   Afficher le mot donné au départ et le mot mélangé.
   Pour changer la place des lettres, importer la fonction **`randrange()`** du module **`random.`**

```
from random import randrange
```

   Exemple

```
Entrez un mot : <olibrius>
Le mot donné est "olibrius".
Le mot mélangé est "libusoir".
```

10. Écrire un script qui affichera uniquement les caractères d'index impair d'une chaîne donnée si la longueur de la chaîne est impair, les caractères d'indice pair si sa longueur de la chaîne est pair.
    Exemples

```
Entrez une chaîne : <Honolulu>
La chaîne a une taille pair(8).
Les caractères d'indice pair sont : H, n, l, l.
```

```
Entrez une chaîne : Pamukkale
La chaîne a une taille impair (9).
Les caractères d'indice impair sont : a, u, k, l.
```

11. Écrire un script qui calcule la position d'un caractère donné dans l'alphabet. Le script doit vérifier préalablement que le caractère donné est bien une lettre de l'alphabet.

    Exemple

```
Entrez une lettre de l'alphabet : n
La lettre 'n' est la 14e lettre de l'alphabet.
```

12. Écrire un script qui crypte une chaîne de caractères donnée en décalant chacune des lettres du rang donné.

    L'utilisateur entrera une chaîne de caractères et un nombre.

    Utiliser les fonctions **ord()** et **chr()** qui utilisent la table Unicode pour coder/décoder les caractères.

    - **ord()** : retourne le code du caractère donné en argument.

```
nbre = ord('a') # attribuera la valeur 97 à nbre.
```

    - **chr()** : retourne le caractère correspondant au code donné.

```
carac = chr(99) # attribuera le caractère 'c' à carac.
```

13. Écrire un script qui demandera à l'utilisateur d'entrer une chaîne de caractères quelconque. Le script se chargera de supprimer toutes les occurrences successives d'une même lettre.

    Exemple : la chaîne « **aaaaattttddddjjfyyfioo** » deviendra « **atdjfyfio** ».

Liste

Consignes

Créer un dossier **`listes`** dans votre dossier **`programmation`**.

Tous les scripts de cette section y seront sauvegardés sous le nom **`listXX.py`** où **`XX`** est le numéro de l'exercice écrit sur deux chiffres.

Par exemple, l'exercice 7 sera enregistré sous le nom **`list07.py`**.

Les seules fonctions/méthodes autorisées dans les scripts sont :

- **`len()`**
- **`append()`**
- **`extend()`**

Utilisez l'instruction **`del`** pour effacer un élément d'une liste.

Exercices

1. définir la liste :
   `liste = [17, 38, 10, 25, 72]`

   puis effectuez les actions suivantes :

   – ajoutez l'élément 12 à la liste et affichez la liste ;

   – affichez l'indice de l'élément 17 ;

   – enlevez l'élément 38 et affichez la liste ;

   – affichez la sous-liste du 2e au 3e élément ;

   – affichez la sous-liste du début au 2e élément ;

   – affichez la sous-liste du 3e élément à la fin de la liste ;

   – affichez la sous-liste complète de la liste ;

   – affichez le dernier élément en utilisant un indiçage négatif.

2. Écrire un programme qui calcule le produit scalaire de deux vecteurs d'entiers **`u`** et **`v`** (de même dimension).

$$U.V = u_1.v_1 + u_2.v_2 + u_3.v_3$$

   Exemple 1

```
U = [3, 6, 9] ; V = [1, 9, 0]
>>> [3, 6, 9] . [1, 9, 0] = 57
```

Exemple 2

```
U = [1, 7, 2] ; V = [3, 2, 1]
>>> [1, 7, 2] . [3, 2, 1] = 19
```

Exemple 3

```
U = [1, 5, 4] ; V = [7, 8, 9]
>>> [1, 5, 4] . [7, 8, 9] = 83
```

3. On dispose de deux tableaux **tab1** et **tab2** (de dimensions respectives **n** et **m**), triés par ordre croissant. Fusionner les éléments de **tab1** et **tab2** dans un troisième tableau **fusion** trié par ordre croissant.

   Exemple

```
tab1 = [3, 7, 9, 15, 18]
tab2 = [1, 2, 8, 12, 14, 24, 36]
>>> fusion = [1, 2, 3, 7, 8, 9, 12, 14, 15, 18, 24, 36]
```

   Méthode: Utiliser trois indices **itab1**, **itab2** et **ifus**.

   (a) Comparer **tab1[itab1]** et **tab2[itab2]** ;

   (b) remplacer **fusion[ifus]** par le plus petit des deux éléments;

   (c) avancer dans le tableau fusion et dans le tableau qui a contribué son élément.

   (d) Lorsque l'un des deux tableaux **tab1** ou **tab2** est épuisé, il suffit de recopier les éléments restants de l'autre tableau dans le tableau **fusion**.

4. Écrire un programme qui lit les points (sur 60) de **n** élèves d'une classe dans un devoir et les mémorise dans un tableau **points** de dimension **n**.

   (a) Générer la liste des points :

   ➔ Demander **n**, le nombre d'élèves présents dans la classe (nombre strictement compris entre 10 et 30).

   ➔ Créer **points**, une liste de nombres aléatoires tirés entre 0 et 60 inclus. Il y aura autant de nombres que d'élèves.

   (b) Rechercher dans la liste **points** et afficher :

   ➔ la note maximale, la note minimale et la moyenne des notes.

   (c) À partir des **points** des élèves, établir un tableau **notes** de dimension 7 qui est composé de la façon suivante:

- ➔ **notes[0]** contient le nombre de notes comprises entre 0 et 9.
- ➔ **notes[1]** contient le nombre de notes comprises entre 10 et 19.
- ➔ **notes[2]** contient le nombre de notes comprises entre 20 et 29.
- ➔ **notes[3]** contient le nombre de notes comprises entre 30 et 39.
- ➔ **notes[4]** contient le nombre de notes comprises entre 40 et 49.
- ➔ **notes[5]** contient le nombre de notes comprises entre 50 et 59.
- ➔ **notes[6]** contient le nombre de notes égales à 60.

(d) Établir un graphique de barreaux représentant le tableau **notes**. Utilisez les symboles **#######** pour la représentation des barreaux et affichez le domaine des notes en dessous du graphique.

Idée: Déterminer la valeur maximale **maxn** dans le tableau **notes** et afficher autant de lignes sur l'écran. (Dans l'exemple ci-dessous, **maxn** = 6).

Exemple

```
La note maximale est 58.
La note minimale est 13.
La moyenne des notes est 37.250000.

6  >                              #######
5  >                      ####### #######
4  >              ####### ####### #######
3  >              ####### ####### ####### #######
2  >       ####### ####### ####### ####### #######
1  >       ####### ####### ####### ####### #######
    +-------+-------+-------+-------+-------+-------+-------+
    | 0 - 9 | 10-19 | 20-29 | 30-39 | 40-49 | 50-59 |  60   |
```

5. Soient les listes suivantes :

```
t1 = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
t2 = ['Janvier', 'Février', 'Mars', 'Avril', 'Mai', 'Juin', 'Juillet', 'Août',
'Septembre', 'Octobre', 'Novembre', 'Décembre']
```

Écrire un script qui crée une nouvelle liste **t3**. Celle-ci devra contenir tous les éléments des deux listes en les alternant, de telle manière que chaque nom de mois soit suivi du nombre de jours correspondant :

```
['Janvier', 31, 'Février', 28, 'Mars', 31, ... ]
```

6. À partir d'une série donnée, construire un programme qui cherchera l'élément le plus grand et l'élément le plus petit. Le programme affichera les deux éléments suivis de leur index respectif.

Exemple

```
serie = [2, 5, 4, 56, 14, 6, 0]
Max : 56 [3]
Min :  0 [6]
```

7. Construire un script qui, à partir d'une série et d'une valeur entière, affichera les mots dont la taille excédera la valeur fournie par l'utilisateur. Si aucun mot n'est assez grand, le script le signalera.

   Exemple 1

```
serie = ['crotale', 'python', 'boa', 'couleuvre', 'cobra']
valeur = 5
Les mots de plus de 5 caractères sont : crotale, python, couleuvre.
```

   Exemple 2

```
serie = ['crotale', 'python', 'boa', 'couleuvre', 'cobra']
valeur = 9
Il n'existe pas de mots de plus de 9 caractères.
```

8. Construire un script qui, à partir d'une liste, retournera un entier. Cet entier sera le résultat de l'addition de tous les éléments de la liste de départ.

   Exemple

```
liste = [5, 8, 9, 47, 36, 123, 5, 3, 1]
La somme de tous les éléments de la liste est 237.
```

9. Construire un script qui, à partir d'un caractère et d'une liste de chaînes de caractères, affichera les mots de la liste contenant le caractère fourni ainsi que le nombre d'occurrences de celui-ci dans ce mot.

   Exemple

```
ptit_dej = ['biscottes', 'chocolat', 'cafe', 'tartines', 'the']
caract = 'c'
Le mot 'biscottes' contient 1x le caractère 'c'.
Le mot 'chocolat' contient 2x le caractère 'c'.
Le mot 'cafe' contient 1x le caractère 'c'.
```

10. Écrire un script qui compare deux séries de nombres. La comparaison se basera sur la somme des nombres de la série.

    Exemple

```
s1 = [2,6,8,9,4,63] # somme = 92
s2 = [5,6,74,3,1,0] # somme = 89
La somme des nombres de la série [2,6,8,9,4,63] est plus grande que la somme
des nombres de la série [5,6,74,3,1,0].
```

11. Même chose que l'exercice précédent mais avec des séries de chaînes de caractères.

    Exemple

```
s1 = ['chat', 'chien', 'condor', 'serpent'] # [4, 5, 6, 7] => 22
s2 = ['rat', 'souris', 'cochon', 'dragon'] # [3, 6, 6, 6] => 21
La série ['chat, 'chien', 'condor', 'serpent'] est plus grande que la série
['rat', 'souris', 'cochon', 'dragon'].
```

12. Construire un programme qui, à partir d'une liste et d'un entier, permettra d'afficher une liste composée des mêmes éléments que la liste de départ tout en omettant l'élément d'index donné.

    Si l'index n'est pas légal, le script affichera la liste de départ. Si la liste de départ est vide, le script affichera une liste vide !

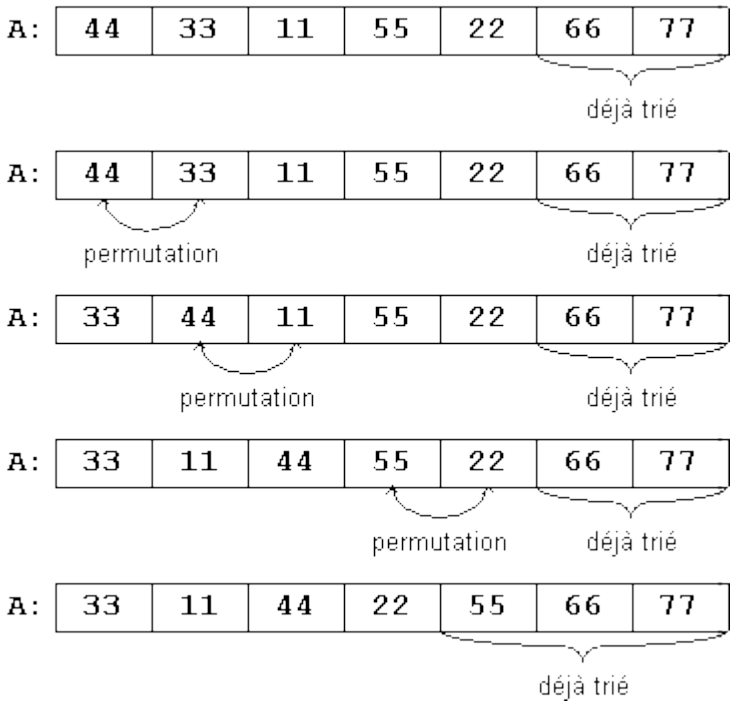
    Exemple

```
s1 = ["hello", 40.25, 57, "world", 4, "!"]
i2r = 2
L'élément [2]:'57' a été enlevé : ["hello", 40.25, "world", 4, "!"].
```

13. Tri par propagation (**bubblesort**). Classer les éléments d'un tableau A par ordre croissant.

    Méthode: En recommençant chaque fois au début du tableau, on effectue à plusieurs reprises le traitement suivant: on propage, par permutations successives, le plus grand élément du tableau vers la fin du tableau (comme une bulle qui remonte à la surface d'un liquide).

    Exemple



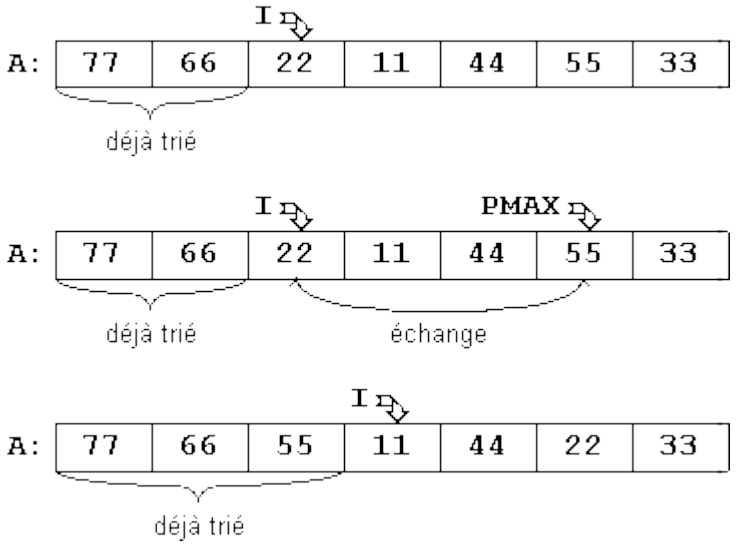
Implémenter l'algorithme en considérant que:

- La partie du tableau (à droite) où il n'y a pas eu de permutations est triée.
- Si aucune permutation n'a eu lieu, le tableau est trié.

14. Tri par **sélection du maximum**. Classer les éléments d'un tableau **`tab`** par ordre décroissant.

    Méthode: Parcourir le tableau de gauche à droite à l'aide de l'indice **`i`**. Pour chaque élément **`tab[i]`** du tableau, déterminer la position **`pmax`** du (premier) maximum à droite de **`tab[i]`** et échanger **`tab[i]`** et **`tab[pmax]`**.

    Exemple



Liste de listes : matrice

Consignes

Créer un dossier **`matrix`** dans votre dossier **`programmation`**.

Tous les scripts de cette section y seront sauvegardés sous le nom **`matrixXX.py`** où **`XX`** est le numéro de l'exercice écrit sur deux chiffres.

Par exemple, l'exercice 5 sera enregistré sous le nom **`matrix05.py`**.

Exercices

0. **Création d'une matrice contenant des entiers tirés aléatoirement.** Écrire un script qui construit une matrice d'entiers tirés au hasard. Appeler la matrice **`matrix`**. La taille de la matrice (**`lignes`** x **`colonnes`**) est demandée à l'utilisateur. La matrice peut être carrée (**`lignes == colonnes`**) ou rectangulaire (**`lignes != colonnes`**).

   Pour les exercices suivants, copier-coller le code de l'exercice 0. Le code copié-collé ne peut pas être modifié ! Écrire après le code de l'exercice 0, le code source de l'exercice donné.

   La variable matrix, celle qui contient la matrice créée, peut être modifiée pour les besoins de l'exercice !

1. **Mise à zéro de la diagonale principale d'une matrice.** Écrire un programme qui met à zéro les éléments de la diagonale principale d'une matrice ***carrée*** **`matrix`** donnée.

2. **Matrice identité.** Écrire un programme qui construit et affiche une matrice ***carrée identité*** **`unite`** de dimension **`n`**. Demandez la taille de la matrice à l'utilisateur.
   Une matrice identité est une matrice, telle que :

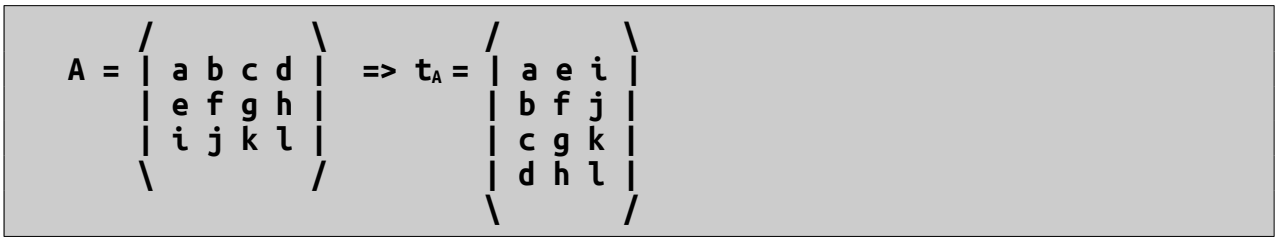
   $$U_{ij} = \begin{cases} 1 \text{ si } i == j \\ 0 \text{ si } i != j \end{cases}$$

   Il s'agit du seul exercice qui ne nécessite pas la matrice de l'exercice 0 !

3. **Transposition d'une matrice.** Écrire un programme qui effectue la transposition **`t_matrix`** d'une matrice **`matrix`** de dimensions (**`n,m`**) en une matrice de dimensions (**`m,n`**).
   (a) La matrice transposée sera mémorisée dans une deuxième matrice **`t_matrix`** qui sera ensuite affichée.

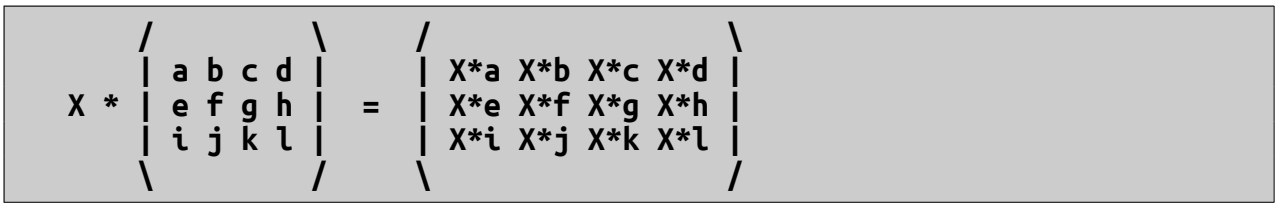
   (b) La matrice **`matrix`** sera transposée par permutation des éléments.

Rappel



4. **Multiplication d'une matrice par un réel.** Écrire un programme qui réalise la multiplication d'une matrice **matrix** par un réel **x**.

   Rappel

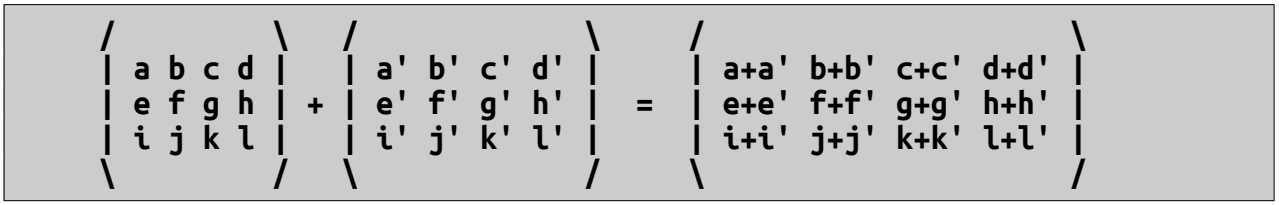


   (a) Le résultat de la multiplication sera mémorisé dans une deuxième matrice **matrix** qui sera ensuite affichée.

   (b) Les éléments de la matrice **matrix** seront multipliés par **x**.

5. **Addition de deux matrices.** Écrire un programme qui réalise l'addition de deux matrices **mata** et **matb** de mêmes dimensions **(n,m)**.

   Rappel



   (a) Le résultat de l'addition sera mémorisé dans une troisième matrice **matc** qui sera ensuite affichée.

   (b) La matrice **matb** est ajoutée à **mata**.

6. En multipliant une matrice **mata** de dimensions **(n,m)** avec une matrice **matb** de dimensions **(m,p)** on obtient une matrice **matc** de dimensions **(n,p)** :

   mata(n,m) * matb(m,p) = matc(n,p)

   $$c_{ij} = \sum_{k=1}^{k=M} (a_{ik} * b_{kj})$$

   La multiplication de deux matrices se fait en multipliant les composantes des deux matrices lignes par colonnes :

   Rappel

```
/       \   /     \   /                                   \
| a b c |   | p q |   | a*p + b*r + c*t    a*q + b*s + c*u |
| e f g | * | r s | = | e*p + f*r + g*t    e*q + f*s + g*u |
| h i j |   | t u |   | h*p + i*r + j*t    h*q + i*s + j*u |
| k l m |   \     /   | k*p + l*r + m*t    k*q + l*s + m*u |
\       /             \                                   /
```

   Écrire un programme qui effectue la multiplication de deux matrices **mata** et **matb**. Le résultat de la multiplication sera mémorisé dans une troisième matrice **matc** qui sera ensuite affichée.

Dictionnaire

Consignes

Créer un dossier **`dictionnaire`** dans votre dossier **`programmation`**. Tous les scripts de cette section y seront sauvegardés.

Enregistrer chaque fichier de cette section sous le nom **`dictXX.py`** où **`XX`** sera remplacé par un nombre écrit sur deux chiffres.

Par exemple, l'exercice 7 sera enregistré sous le nom **`dict07.py`**.

Exercices

1. On dispose d'un dictionnaire associant à des noms de commerciaux d'une société le nombre de ventes qu'ils ont réalisées. Par exemple :

```
ventes = {"Dupont": 14, "Hervy": 19, "Geoffroy": 15, "Layec": 21}
```

   Écrivez un script qui

   1. affiche le nombre total de ventes dans la société ;
   2. affiche le nom du vendeur ayant réalisé le plus de ventes. Si plusieurs vendeurs sont *ex-aequo* sur ce critère, le script affichera la liste de tous les vendeurs répondant au critère.

2. Sur base d'une chaîne de caractères comprenant, sur chaque ligne, trois champs séparés par des caractères **`';'`** (un numéro d'étudiant, un nom et un prénom), écrivez un script qui construit un dictionnaire dont les clés sont les numéros d'étudiants lus et les valeurs sont, pour chaque numéro d'étudiant, une chaîne correspondant à la concaténation des prénom et nom de la personne.

   On pourra tester la fonction avec la chaîne suivante :

```
chaine_etudiants = """213615200;BESNIER;JEAN
213565488;DUPOND;MARC
214665555;DURAND;JULIE"""
```

3. Écrivez un script qui prend en entrée un dictionnaire associant à un nom une liste de notes et qui retourne la liste des noms des personnes qui ont la moyenne la plus élevée (s'il y a des *ex-aequo*, cette liste contiendra plusieurs éléments, sinon, elle n'en contiendra qu'un) et la moyenne correspondante. On pourra utiliser le dictionnaire suivant pour tester la fonction ainsi écrite :

```
notes = {"Tom": [8, 10, 12], "Mila": [10, 9], "Alex": [], "Lina": [12, 10, 8]}
```

4. Écrivez un script qui prend crée un dictionnaire fusionnant les informations de trois autres dictionnaires de la manière suivante : pour chaque clef présente dans au moins un dictionnaire, la valeur associée sera la liste des valeurs associées à cette clef dans les dictionnaires donnés.

   Par exemple, pour les dictionnaires suivants :

```
dict1 = {"a": 1, "d": 4, "g": 7}
dict2 = {"a": 1, "b": 2, "h": 8}
dict3 = {"a": 2, "c": 3, "h": 9}
```

   Le script devra afficher :

```
{'a': [1, 1, 2], 'd': [4], 'g': [7], 'b': [2], 'h': [[8], 9], 'c': [3]}
```

5. x

Fonction

Consignes

Créez un dossier **`fonctions`** dans lequel vous enregistrerez tous les exercices suivants dans un fichier nommé **`fonctions.py`**.
Insérez un commentaire avant la définition de chaque fonction.

```
# Exercice X
def maSuperFonctionDeLaMortQuiTue():
    # code source de la fonction
```

Utilisez le fichier **`test_fonctions.py`** pour tester vos fonctions !

Exercices

1. Construire une fonction **`somme(liste)`** qui, à partir d'une liste, retournera un entier. Cet entier sera le résultat de l'addition de tous les éléments de la liste de départ.

```
l = [5, 8, 9, 47, 36, 123, 5, 3, 1]
somme(l) # retournera 237.
```

2. Construire une fonction **`pair(nbre)`** qui, à partir d'un entier, renverra le booléen **`True`** si ce nombre est pair, **`False`** sinon.

```
pair(3) # retournera 0
pair(4) # retournera 1
```

3. Définir une fonction **`maximum(n1, n2, n3)`** qui renvoie le plus grand des 3 nombres **`n1`**, **`n2`**, **`n3`** fournis en arguments.

```
maximum(2, 5, 4) # retournera 5
```

4. Définir une fonction **`indexMax(liste)`** qui renvoie l'index de l'élément ayant la valeur la plus élevée dans la liste transmise en argument.

```
serie = [5, 8, 2, 1, 9, 3, 6, 7]
indexMax(serie) # retournera 4
```

5. Définir une fonction **`nomMois(n)`** qui renvoie le nom du nième mois de l'année. Si le paramètre **`n`** donné n'est pas compris entre 1 et 12, alors la fonction retournera l'élément **`None`**.

```
nomMois(4) # retournera "Avril"
```

6. Définir une fonction **inverse(chaine)** qui permette d'inverser l'ordre des caractères d'une chaîne quelconque. La chaîne inversée sera renvoyée au programme appelant.

7. Définir une fonction **compteMots(phrase)** qui renvoie le nombre de mots contenus dans la phrase **phrase**. On considère comme mots les ensembles de caractères inclus entre des espaces.

8. Construire une fonction **occurrence(chaine, car)** qui, à partir d'une chaîne de caractères et d'un caractère, renverra un entier. Cet entier sera le nombre de fois que le caractère apparaîtra dans la chaîne.

   Exemple

```
occurrence("une belle vache", "e") # retournera 4.
```

9. Construire une fonction **str2list(chaine)** qui, à partir d'une chaîne, renverra une série. Chaque élément de la série sera composé d'une lettre de la chaîne.

```
str2list("hello !") # retournera ["h","e","l","l","o"," ","!"].
```

10. Construire une fonction **find_num(matrix, nbre)** qui, à partir d'une matrice d'entiers et d'une valeur entière quelconque, retournera la coordonnée du premier entier rencontré **(X, Y)** du paramètre **nbre** dans la matrice **matrix**.

    Si le paramètre **nbre** n'est pas présent dans la matrice **matrix**, la fonction retournera le mot-clé **None**.

```
matrix = [[15, 23, 45],
          [39, 63, 78],
          [56, 45, 12]]
find_num(matrix, 45)  # retournera (0, 2)
find_num(matrix, 17)  # retournera None
```

11. Construire une fonction **grands_mots(liste, nbre)** qui, à partir d'une série et d'une valeur entière, renverra un liste. Cette nouvelle liste contiendra les mots dont la taille excédera la valeur fournie en argument. Si aucun mot n'est assez grand, la fonction renverra **None**.

```
l = ['crotale', 'python', 'boa', 'couleuvre', 'cobra']
grands_mots(l, 5) # retournera ['crotale', 'python', 'couleuvre']
grands_mots(l, 9) # retournera None.
```

12. Construire une fonction **liste_paire(liste)** qui, à partir d'une liste d'entier, renverra une nouvelle liste. Celle-ci sera composée des éléments de la liste précédente avec les changements suivants :
    - tous les éléments d'index pair se verront diviser par 2 (division entière);
    - tous les éléments d'index impair se verront multiplié par 2 ;

```
serie = [4, 52, 13, 9, 16, 49, 756, 7]
```

```
liste_paire(serie) # retournera [2, 104, 6, 18, 8, 98, 378, 14]
```

13. Construire une fonction **occ_liste(liste, car)** qui, à partir d'un caractère et d'une liste de chaînes de caractères, renverra une liste d'entier. Chaque élément de celle-ci contiendra le nombre de fois qu'apparaîtra le caractère dans le mot.

    Utilisez une fonction précédente pour vous faciliter la tâche !

```
ptit_dej = ['biscottes', 'chocolat', 'cafe', 'tartines', 'the']
occ_liste(ptit_dej, 'c') # retournera [1, 2, 1, 0, 0]
```

14. Construire une fonction **stat_chiffres(nbre)** qui, à partir d'un entier (positif ou négatif), renverra un dictionnaire contenant pour chaque chiffre présent son occurrence dans le nombre.

    Utilisez une fonction précédente pour vous faciliter la tâche !

```
nbre = 452475
stat_chiffres(nbre) # retournera {4: 2, 5: 2, 2: 1, 7: 1}
nbre = 11122544456666
stat_chiffres(nbre) # retournera {1: 3, 2: 2, 5: 2, 4: 3, 6: 4}
```

15. Construire une fonction **fibonacci(n)** qui, à partir d'un entier, retournera une liste contenant les **n** premiers entiers faisant partie de la suite de Fibonacci, **n** étant l'entier passé à la fonction.

    Les deux premiers termes de la suite sont toujours donnés. Il s'agit de 0 et 1.

```
fibonacci(9) # retournera [0, 1, 1, 2, 3, 5, 8, 13, 21]
```

16. Construire une fonction **diff_list(liste1, liste2)** qui, à partir de deux listes, permet de calculer une nouvelle liste qui est le résultat de la différence, élément par élément, des deux listes précédentes. Ces deux listes doivent posséder le même nombre d'éléments ! Si ce n'est pas le cas, la fonction retournera **None**.

    Exemple

```
s1 = [4, 5, 8, 89, 2, 14]
s2 = [2, 3, 8, 65, 45, 6]
diff_list(s1, s2) # retournera [2, 2, 0, 24, -43, 8]
```

17. Écrire une fonction **distance(xa, ya, ab, yb)** comportant quatre paramètres : **xa**, **ya**, **xb** et **yb** qui représentent les coordonnées de deux points A et B dans le plan et qui renvoie la distance AB.

18. Écrire une fonction **eratosthène(n)** ayant en paramètre un entier **n** et qui renvoie une liste de tous les nombres premiers compris entre 0 et **n**.

Le code de votre fonction reprendra le crible d'ératosthène.

19. Écrire une fonction **`premier(n)`** ayant en paramètre un entier et qui renvoie un booléen **`True`** si l'entier est premier et **`False`** sinon.

    Utilisez la fonction précédente pour réaliser cette fonction.

20. Écrire une fonction **`npremier(n)`** ayant comme paramètre un entier **`n`** et qui renvoie le n-ième nombre premier.

```
npremier(7)  # retournera 17
```

21. Écrire une fonction **`swap(a, b)`** ayant en paramètres 2 entiers **`a`** et **`b`** et qui échange les contenus de **`a`** et de **`b`**.

22. Écrire une fonction **`checkindex(t, n)`** ayant comme paramètres une liste d'entiers **`t`** et un entier **`n`**. La fonction doit retourner l'index de l'élément **`n`** de la liste. Si l'entier **`n`** n'existe pas dans la liste **`t`**, alors la fonction retournera **`None`**.

```
t = [12, 65, 23, 14, 85, 11, 8, 41, 9, 0, 55, 3]
Si n = 85, alors checkindex(t, n) == 4
Si n = 13, alors checkindex(t, n) == False
```

23. Écrire une fonction **`check0to10(t, n)`** ayant en paramètres une liste **`t`** de taille quelconque et un entier **`n`**. La fonction doit renvoyer un booléen indiquant s'il existe une valeur comprise entre 0 et 10 dans les **`n`** premiers éléments de la liste **`t`**. La fonction doit retourner **`False`** si la valeur de **`n`** est plus grande que la taille de la liste.

```
t = [12, 65, 23, 14, 85, 11, 8, 41, 9, 0, 55, 3]
Si n = 5, alors check0to10(t, n) == False
Si n = 8, alors check0to10(t, n) == True
```

24. Écrire la fonction **`nchiffres(n)`** qui prend une valeur entière **`n`** (positive ou négative) comme paramètre et qui retourne le nombre de chiffres différents composant le nombre **`n`**.

```
nchiffres(45)  # retournera 2
nchiffres(6457392)  # retournera 7
nchiffres(4555463)  # retournera 4
```

25. Écrire la fonction **`dec2bin(n, size=8)`** qui prend deux entiers **`n`** et **`size`** en paramètres et qui retourne, sous forme d'une chaîne de caractères, la représentation binaire du nombre **`n`** donné codé sur **`size`** bits.

    Le paramètre **`size=8`** signifie que ce paramètre est optionnel lors de l'appel de la fonction. En effet, s'il est omis, la fonction considérera que le paramètre **`size`** à la valeur de 8. S'il est donné, il prendre la valeur passée à la fonction.

Si la taille donnée (**size**) est trop petite pour contenir le nombre demandé, la fonction ignorera le paramètre **size**.

```
dec2bin(42)  # retournera "00101010"
dec2bin(42, 4)  # retournera "101010"
dec2bin(35, 5)  # retournera "100011"
```

26. Écrire la fonction **dec2base(x, base)** qui prend deux paramètres : un entier **x** et un entier **base**. La fonction retournera, sous forme d'une chaîne de caractères, la représentation du nombre **x** en base **base**.

    La base **base** devra être comprise entre 2 et 26 inclus. Si ce n'est pas le cas, la fonction retournera le mot-clé **None**.

```
dec2base(43, 7) # retournera "61"
```

    Réécrire la fonction **dec2bin_()** à l'aide de la fonction **dec2base()**. Ajoutez le caractère « underscore _ » à la fin du nom de la fonction pour ne pas créer un conflit avec celle qui existe déjà.

27. Écrire la fonction **bin_compla2(x)** qui prend en paramètre un nombre entier signé **x** représenté sous forme binaire (codé sous forme d'une chaîne de caractères). La fonction retournera le complément à 2 du nombre binaire **x**.

    N'oubliez pas que **bin_compla2(bin_compla2(x)) == x**.

```
bin_compla2("0101010") # retournera "1010110"
```

28. Écrire la fonction **convert_mode(perm)** qui prend comme paramètre les permissions d'un fichier sous forme d'une chaîne d'entier. Ces permissions sont écrites sous forme de code octal. L'objectif de la fonction est d'afficher ces permissions sous forme d'un triplet symbolique.

    Vérifiez au préalable la validité du code octal.

    Pour rappel, **r = 4**, **w = 2**, **x = 1** et **rw- = 4+2 = 6**

```
convert_mode(755)  # retournera "rwxr-xr-x"
convert_mode(021)  # retournera "----w---x"
```