Pour les premiers pas, voici un cours miniature et brouillons : LesListes

Les signifient que la correction est disponible.

Exercice *L1/

L'exercice consiste à créer une fonction searchL qui a pour paramètres un élément et une liste et qui renvoie False si l'élément n'est pas dans la liste, ou l'indice de l'élément dans le cas contraire. (ici False est un booléen, pas une chaîne de caractères)

Correction

```
def searchL(i,L):                L0
    for k in range(len(L)):      L1
        if L[k]==i:              L2
            return(k)            L3
    return(False)                L4
```

-Explications :
Je nomme L la liste dans laquelle je vais chercher l'élément noté i. L et i sont donc les paramètres de la fonction **searchL.**

Boucle for : *on parcourt les indices de la liste L, pour cela :*

L1. On nomme k un élément qui commence avec la valeur 0, et dont la valeur augmentera de 1 à chaque boucle jusqu'à être égale à l'indice du dernier élément de L.
(c'est-à-dire la longueur de la liste moins 1, len(L)-1, car l'indice du premier élément de la liste est 0)

L2. On compare avec un " if " l'élément d'indice k dans L et l'élément i recherché. S'il y a égalité :

L3. On renvoie l'indice k grâce à la fonction return.

Fin Boucle for: *Si on arrive ici c'est qu'on n'a pas trouvé l'élément i dans L. L'élément i n'est donc pas dans L !*

L4. On renvoie le booléen False grâce à la fonction return.

ATTENTION!! L'indice du premier élément de la liste est 0.
ATTENTION!! Voici la syntaxe de la fonction range : range**(start,stop+1,step)**, avec par défaut start=0 et step=1.

Exercice *L2/

L'exercice consiste à créer une fonction maximum qui a pour paramètre une liste de nombres et qui renvoie la valeur maximale trouvée dans la liste et son indice.

Correction

```
def maximum(L):                 L0
    m=L[0]                      L1
    ind=0                       L2
    for k in range(len(L)-1):   L3
        if m<=L[k+1]:           L4
            m=L[k+1]            L5
            ind=k+1             L6
    return(m,ind)               L7
```

-Explications :

L1. On nomme m le 1er élément de la liste L.
(m deviendra le plus grand élément.)
L2. On note ind l'indice de m.

Boucle for : *On va parcourir les indices de la liste L. Dès que l'on trouve un élément plus grand que m, m prendra sa valeur.* Pour cela :
L3. Pour k allant de 0 à len(L)-2
L4. On regarde si m est inférieur ou égal au k+1ième élément de L (i.e. L[k+1]).
Si L[k+1] est plus grand que m :
L5. alors m=L[k+1] pour que m soit le plus grand élément à la fin.
L6. Puis on met à jour ind (l'indice de m dans L).

Fin boucle for : *On a fini de parcourir le liste*. m *est maintenant le plus grand élément.*
L7. A la fin on renvoie m et ind grâce à la fonction return.

ATTENTION!! à L3 k va de 0 à len(L)-1 car les indices sont repérés par k+1 ! Du coup on parcourt bien les indices de 1 à len(L)-1 avec la syntaxe range**(start,stop+1,step)**, avec par défaut start=0 et step=1.

Exercice *L3/

L'exercice consiste à créer une fonction trisel qui a pour paramètre une liste de nombres et qui renvoie la liste triée par sélection.

Correction

n°1:

```
1 import numpy as np
2
3 def trisel(L):
4     for i in range(len(L)-1):
5         i_min=np.argmin(L[i:])+i
6         L[i],L[i_min]=L[i_min],L[i]
```

n°2

```
def trisel(L):
    for i in range(len(L)-1):
        i_min=i
        for k in range(i,len(L)):
            if L[k]<=L[i_min]:
                i_min=k
```

L[i],L[i_min]=L[i_min],L[i]

n°1 Code copiable : [Afficher]

L1. On importe la bibliothèque numpy, alias np.
L3. On commence à définir la fonction **trisel** avec un paramètre L qui sera une liste.
Boucle for : **L4.** On parcourt tous les éléments de la liste L sauf le dernier (car au bout de len(L)-1 boucles, le plus grand élément est bien à la fin de la liste), en passant par chaque indice i.
L5. On obtient l'indice i_min du plus petit élément de la liste sans considérer les i premiers éléments (d'où le np.argmin(L[i:])). Pour que l'indice soit l'indice du plus petit élément dans la liste L (tous éléments considérés), on rajoute i à l'indice - car les i éléments anciennement considérés sont au début de liste.
L6. On permute le plus petit élément de la liste sans considérer les i premiers éléments (L[i_min]), avec le premier terme considéré (L[i]).
Fin Boucle for.

n°2: Code copiable : [Afficher]

On utilise le même raisonnement que la solution n°1, mais cette fois-ci on n'utilise pas la fonction argmin().
Du coup, il nous faut un moyen pour obtenir l'indice du plus petit élément de L[i:]. En voici un :
L3. On initialise l'indice i_min à i.
Boucle for : **L4.** On parcourt les éléments de L en partant du i-ième élément.
L5. Si on trouve un élément inférieur à l'élément d'indice i_min,
L6. alors i_min prend la valeur de k.
Fin Boucle for : Une fois sorti de cette boucle, i_min est l'indice du plus petit élément de la liste sans considérer les i premiers éléments.

REMARQUE!! Nous n'avons pas besoin de retourner la liste L à la fin de la définition de la fonction car cette liste a été directement modifiée ! Faites un print(L) après l'exécution de la fonction et vous retrouverez une liste triée !

Explication: [Afficher]

Exercice *L4/

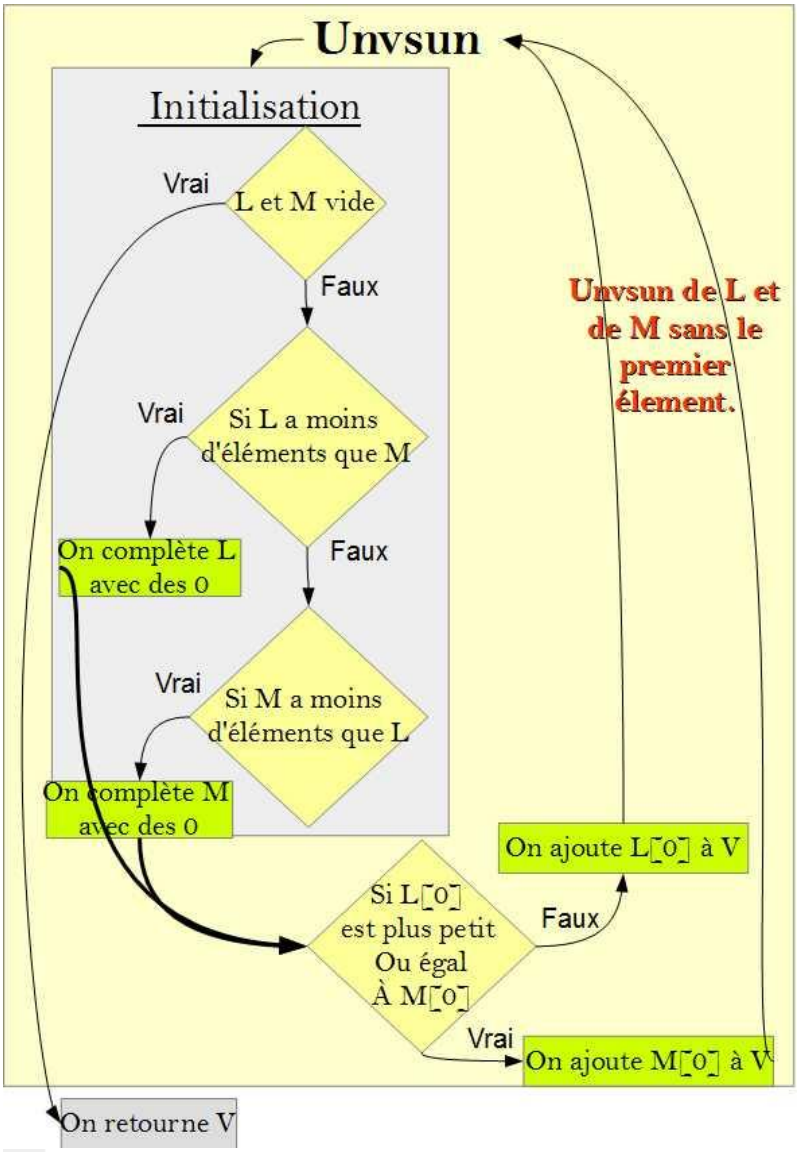
L'exercice consiste à créer une fonction récursive unvsun qui a pour paramètres trois listes L, M et V avec V une liste vide par défaut et qui renvoie une liste contenant les maximums élément par élément de L et M.

Correction

Exemple : M=[0,2,6], L=[1,2,3] => [1, 2, 6]

Indice : [Afficher]

-Une solution fonctionnant sous Python 2.7.5 et Python 3.4
Code copiable : [Afficher]




```
1  def unvsun(L,M,V=[]):
2      l=len(L)
3      m=len(M)
4      if l ==0 and m ==0:
5          return(V)
6      elif l < m:
7          L.append([0]*(m-l))
8      elif l > m:
9          M.append([0]*(l-m))
10
11     if L[0] <= M[0]:
12         V.append(M[0])
13     else:
14         V.append(L[0])
15
16     return(unvsun(L[1:l],M[1:m],V))
```

L1. On définit la fonction **unvsun** avec trois paramètres L, M et V. Le dernier est facultatif car il est initialisé à 0 par défaut. V sera la liste contenant les maximums terme à terme de L et de M.

L2. On pose l la taille de L grâce à la fonction len.
L3. On pose m la taille de M.
L4. Si la taille de L et de M sont nulles :
    L5. On retourne la liste vide V.
L6. Si L est de taille strictement inférieure à celle de M :
    L7. On complète L avec m-l éléments nuls à l'aide de la méthode append.
L8. Si L est de taille strictement supérieure à celle de M :
    L9 On complète M avec m-l éléments nuls.

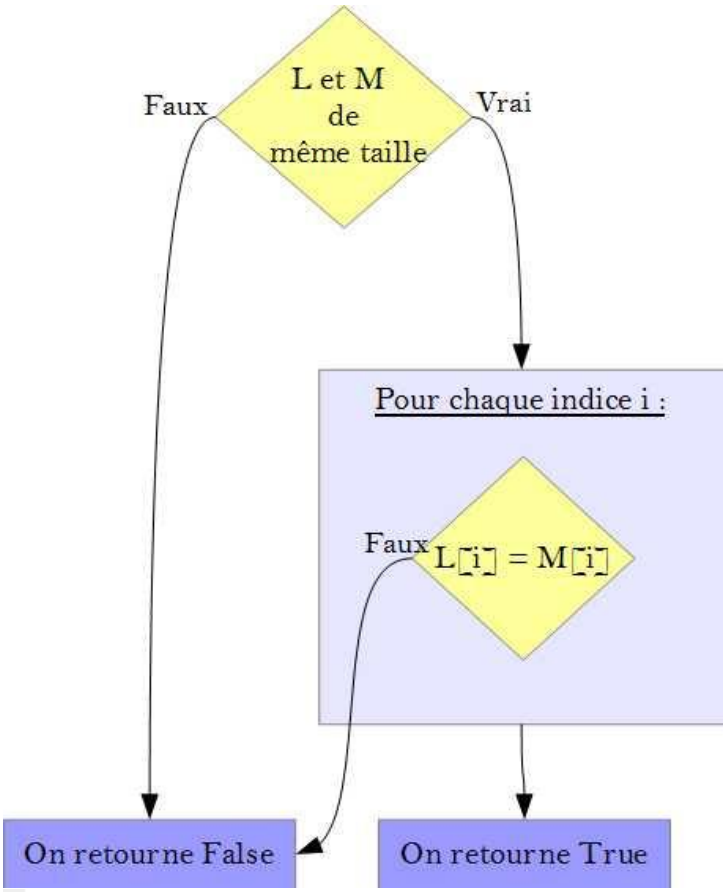
L11. Si le premier élément de L est inférieur ou égal à celui de M :
    L12. On rajoute celui de M dans V.
L13. Sinon
    L13. On rajoute celui de L dans V.
L14. On retourne **unsvun** de L, M et V en enlevant le premier terme de L et le premier terme de M.

Exercice *L5/

L'exercice consiste à créer une fonction itérative (i.e. non récursive) egal qui a pour paramètres deux listes L et M de nombres et qui renvoie True si elles sont égales terme à terme, et False sinon.

Correction

-Une solution fonctionnant sous Python 2.7.5 et Python 3.4




```
1 def egal(L,M):
2     l,m=len(L),len(M)
3     if l==m:
4         for i in range(len(L)):
5             if L[i] != M[i]:
6                 return(False)
7     else:
8         return(False)
9     return(True)
```

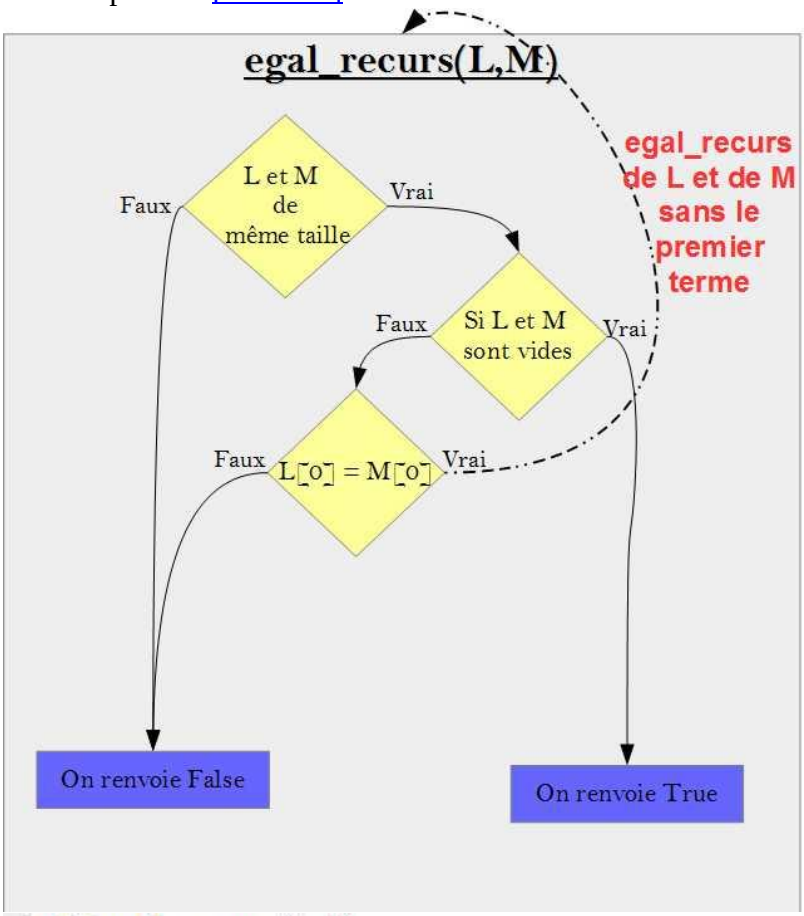
L1. On définit la fonction **egal** qui a deux paramètres L et M.
L2. La taille de L - obtenue grâce à la fonction len() - est égale à 1 et m est égal à la taille de M.
L3. Cas 1 : Si L et M ont la même taille, c'est-à-dire l et m sont égaux,
L4. Pour tout indice possible (i allant de 0 à len(L)-1),
L5. Si le i-ième éléments de L est différent du i-ième éléments de M,
L6. Alors L et M ne sont pas égaux terme à terme, on renvoie donc le booléen False.
L7. Cas 2 : Mais si L et M n'ont pas la même taille,
L8. Alors elles ne peuvent pas être égales. On renvoie donc False.
L9. Si on est là c'est que les listes sont de même taille et que tous leurs éléments vérifient l'égalité L[i]=M[i]. Donc on renvoie True.

Exercice *L6/

L'exercice consiste à créer une fonction récursive egal_recurs qui a pour paramètres deux listes L et M de nombres et qui renvoie True si elles sont égales terme à terme, et False sinon.

Correction

-Une solution fonctionnant sous Python 2.7.5 et Python 3.4
Code copiable : [Afficher]




```
1 def egal_recurs(L,M):
2     if len(L) == len(M):
3         if L==[]:
4             return(True)
5         else:
6             if L[0]==M[0]:
7                 return(egal_recurs(L[1:],M[1:]))
8             else:
9                 return(False)
10    else:
11        return(False)
```

L1. Création de la fonction **egal_recurs** possédant deux paramètres non facultatifs L et M.
L2. Cas 1 : Si L et M sont de même taille, i.e. si len(L) et len(M) sont égaux :
L3. Cas 1.1 : Si ces listes sont vides (ici on ne vérifie que pour une des listes car elles

sont de même taille)

L4. Alors elles sont égales terme à terme. Donc on renvoie True**.**

L5. Cas 1.2 : Sinon (si les listes ne sont pas vides),

L6. Cas 1.2.1 : On compare leurs premiers termes : s'ils sont égaux,

L7. On exécute les mêmes tests sur le reste des listes.

=> Pour cela on renvoie **egal_recurs** de L et de M sans prendre en compte le premier terme de chacune de ces listes.

L8. Cas 1.2.1 : Sinon (si les premiers termes des listes sont différents),

L9. Les deux listes ne sont pas égales terme à terme, on renvoie donc le booléen False.

L10. Cas 2 : Sinon (si L et M sont de tailles différentes)

L11. L et M ne peuvent pas être de même taille, donc ont renvoie False.

Exercice *L7/

Permutez les termes de la liste suivante à l'aide d'un script Python : [12,3,1,5,13,18,85,10,2,74,1,12,3]

Exemple : [1,2,3,4] serait permutée en [2,1,4,3]

Correction

Exemple : [1,2,3,4] serait permutée en [2,1,4,3]

Code copiable : [Afficher]

```
1 L = [12,3,1,5,13,18,85,10,2,74,1,12,3]
2 i = 1
3 while i < len(L) :
4     L[i-1],L[i] = L[i],L[i-1]
5     i += 2
6
7 print(L)
```

L1. On définit la liste L de l'énoncé.

L2. On initialise l'indice i du terme à permuter avec celle d'avant. i est initialisé à 1.

Boucle while : **L3.** Tant que le terme d'indice i n'est pas hors de la liste

L4. On permute le terme d'indice i avec l'indice d'avant.

L5. On incrémente i deux fois pour passer aux deux autres termes

Fin boucle while

Exercice L8/

On numérote des paquets de gâteau en commençant par 0.
La liste suivante correspond aux nombres de gâteaux contenus dans le paquet. L'indice de l'élément dans la liste correspond au numéro du paquet.

L = [10,5,6,4,8]

- Affichez le nombre de paquets de gâteaux référencés dans cette liste,
- Affichez le numéro du paquet contenant 5 gâteaux,
- Donnez le nombre total de gâteaux,

- Rajoutez le paquet numéro 5 contenant 2 gâteaux,
- Affichez le nombre de gâteaux que contient le paquet numéro 4,
- Affichez la liste.

Correction

Code copiable : [Afficher]

```
1 L = [ 10,5,6,4,8]
2
3 print("il y a ",len(L)," paquets de gâteaux")
4 print("le numéro du paquet contenant 5 gâteaux est ", L.index(5))
5 print("le nombre total de gâteaux est ", sum(L))
6 L.append(2)
7 print("le nombre de gâteaux que contient le paquet numéro 4 est ", L[4])
8 print(L)
```

L1. On crée la liste donnée dans l'énoncé, correspond au nombre de gâteaux contenu dans chaque paquet de gâteaux.

L3. On affiche le nombre de paquet référencés (i.e. le nombre d'éléments de la liste) à l'aide de la fonction print.
Pour obtenir le nombre d'éléments de la liste L on utilise la fonction len appliquée à L.
L4. On récupère l'indice de l'élément 5 dans la liste L avec L.index(5), puis on l'affiche avec la fonction print.
L5. On récupère le nombre total de gâteaux en sommant le nombre de gâteaux de chaque paquet, à l'aide de l'instruction sum(L).
L6. On ajoute 2 à la fin de L à l'aide de l'instruction L.append(2).
L7. On récupère l'élément d'indice 4 dans L avec l'instruction L[4].
L8. On affiche L.

Voilà le résultat obtenu :

```
il y a  5  paquets de gâteaux
le numéro du paquet contenant 5 gâteaux est  1
le nombre total de gâteaux est  33
le nombre de gâteaux que contient le paquet numéro 4 est  8
[10, 5, 6, 4, 8, 2]
```

Exercice *L9/

Voici la liste utilisée :
["T","O","A","p","t","p","l","o","e","s","t","t","r","s","t","t","t","u","m","m","p"]
A l'aide d'un script python, supprimez les éléments d'indices pairs puis supprimer toutes les lettres "t" restantes.
Voici le résultat : ["O","p","o","s","s","u","m"]

Correction

```
1 L = ["T","O","A","p","t","p","l","o","e","s","t","t","r","s","t","t","t","u","m","m","p"]
2 Copy = L
3 count = 0
4
5 for i in range(len(L)):
6     if i%2 == 0:
7         del Copy[i-count]
8         count += 1
9 while "t" in Copy:
10     Copy.remove("t")
11
12 print(L)
```

L1. La liste L est la liste donnée dans l'énoncé.
L2. Copy est une copie de la liste L. C'est cette liste qui sera modifié.
L3. count est initialisé à 0. Il s'agit du compteur des éléments supprimer dans la liste.

Boucle For : L5. Pour tous les indices de la liste L,
L6. Si l'indice est pair (l'indice i est congru à 0 modulo 2),
L7. On supprime l'élément d'indice i avec la fonction del.
L8. On incrémente le nombre d'élément supprimé.
Fin Boucle For : Une fois la boucle terminée, tous les éléments d'indice pair dans L ont été supprimé.
Boucle While : L9. et **L10.** Tant que la lettre "t" est présente dans la liste résultante, on supprime l'élément correspondant avec remove.
L12. On affiche la liste L, toujours égale à celle de l'énoncé. Mais la liste Copy est égale à ["O","p","o","s","s","u","m"]

Exercice *L10/

Voici la liste des commandes de petit pain d'une boulangère : "Amandine 6;Amélie 9;Cécile 5;Camille 10;Roland 6;Sebastien 3;David 9;Robin 20;Cédric 1;Kevin 10;Tim 1"
Créez la liste suivante : [[Amandine,6], [Amélie,9] ...] Puis calculez le nombre de petit pains commandés.

Correction

```
1 commandes = "Amandine 6;Amélie 9;Cécile 5;Camille 10;Roland 6;Sebastien 3;David 9;Robin 20;Cédric
2 liste = commandes.split(";")
3 for i in range(len(liste)):
4     liste[i] = liste[i].split(" ")
5 print(liste)
6
7 nbrPains = 0
8 for commande in liste:
9     nbrPains += int(commande[1])
10 print(nbrPains, " petits pains ont été commandé")
```

L1. La chaîne de caractères est définie avec la variable *commandes*.
L2. On crée la liste des commandes en découpant la chaîne de caractères sur les points-virgules. La liste obtenue est la suivante :
['Amandine 6', 'Amélie 9', 'Cécile 5', 'Camille 10', 'Roland 6', 'Sebastien 3', 'David 9', 'Robin 20', 'Cédric 1', 'Kevin 10', 'Tim 1']
Boucle For : L3. Pour toutes les commandes (i.e pour tous les éléments de la liste précédente),
L4. On redéfinit l'élément comme une liste en le découpant sur les espaces " ".
Fin Boucle For : L5. Maintenant la liste est la suivante : [['Amandine', '6'], ['Amélie', '9'], ['Cécile', '5'], ['Camille', '10'], ['Roland', '6'], ['Sebastien', '3'], ['David', '9'], ['Robin', '20'], ['Cédric', '1'], ['Kevin', '10'], ['Tim', '1']]

L7. On initialise le nombre de petits pains commandés à 0.
Boucle For : L8. Pour toutes les commandes (i.e pour tous les éléments de la liste),
L9. On récupère le nombre de petits pains commandés sans oublier de convertir en integer la chaîne de caractères correspondant à un nombre (à l'aide de int).
Fin Boucle For : L10. On affiche le nombre de petits pains.

Exercice *L11

En utilisant les listes de compréhension et la liste P et M suivantes, créez la liste [312*2, 312*41 ....,312*10, 413*2,413*41, ...,413*10].
P = [2, 4, 6, 8, 10]
M = [312, 413]

Correction

Le résultat obtenu est : [624, 1248, 1872, 2496, 3120, 826, 1652, 2478, 3304, 4130]

Voici le script :

```
P = [2,4,6,8,10]
M = [312,413]
print([ m*p for m in M for p in P ])
```