ANNEXE
RESSOURCE : JOURNÉE DE L'EUROPE

PROBLÉMATIQUE :

L'objectif final de cet ensemble d'activités est de répondre à la question suivante : **déterminer le chemin le plus court passant par l'ensemble des capitales et pour lequel la capitale de départ et d'arrivée est identique.**

DESCRIPTIF :

Les étapes qui permettent d'y arriver ne sont pas simples. C'est la raison pour laquelle quatre activités progressives sont proposées.

Toutes les activités font appel à des boucles, ainsi qu'à des listes pour enregistrer les différentes données nécessaires.

L'activité 4 rassemble les étapes précédentes en plus d'une recherche d'un minimum de liste et d'une étape de mise en forme de restitution des résultats obtenus.

Les méthodes de listes .append(), .pop() et .insert() ont été utilisées. Si on peut se passer des deux dernières, la première est presque indispensable.

La méthode .copy() a aussi été utilisée pour travailler sur une liste sans modifier l'exemplaire original.

La compréhension de liste a été utilisée dans le corrigé de l'activité 1, elle aurait pu être remplacée par l'utilisation d'une boucle pour créer les listes vides ou remplies de 0 suivant le cas.

ACTIVITÉ 1 :

Énoncé

Créer un programme en Python, ou une fonction, qui permette à l'élève de préciser les villes choisies, puis d'enregistrer les distances qui séparent les villes dans des listes.

Les villes seront enregistrées dans une liste *villes*.

Les distances seront enregistrées dans une liste de listes *distances* : la liste *distances[i]* contiendra les distances j qui séparent la ville i des villes j.

Si par exemple la deuxième ville envisagée est Berlin (donc si *villes[1]= « Berlin » (*)*), et que *ville[3] = « Paris »*, *distances[1]* contiendra les distances qui séparent Berlin des autres villes choisies, et *distances[1][3]* représentera la distance qui sépare Berlin de Paris.

(*) il ne faut pas oublier que le premier élément d'une liste est numéroté 0.

Pistes pour l'enseignant

- On peut proposer l'aide suivante sur la structure du programme :
  « Après avoir créé les variables nécessaires, on commence par faire compléter la liste *villes*.
  Puis on fait compléter des listes *distance[i]* pour i de 0 au nombre de villes choisies, remplies d'autant de 0 qu'il y a de villes.
  À l'aide de deux boucles imbriquées, on peut alors remplacer chacun des éléments des listes par les bonnes distances. La question posée à l'utilisateur peut utiliser les éléments de la liste *villes* pour simplifier son travail. »
- Les boucles imbriquées pour l'enregistrement des distances entre les villes peuvent être conçues pour éviter de répondre deux fois à la même question.
- On pourrait partir d'une version CSV du tableau Excel pour y récupérer automatiquement les valeurs utiles, après le choix des villes fait.

Programme corrigé :

```
#on demande quel va être le nombre de villes
n = int(input("Avec combien de capitales travaille-t-on? "))
#on crée une liste vide pour les noms des villes
villes=[]
#ici la liste distances est créée par compréhension, mais on peut la créer à l'aide d'une boucle
distances = [[] for i in range(n)]
#on crée la liste des villes à l'aide d'une boucle, et on en profite pour remplir de 0 la liste des distances qui séparent cette ville des autres
for i in range(n):
    ville= input("Quelle est la ville " +str(i+1)+ " de la liste?")
    villes.append(ville)
```

```
    #distances[i] est ici aussi créé par compréhension, on aurait pu écrire une boucle
    distances[i]=[0 for j in range(n)]
#on utilise des boucles imbriquées pour compléter les listes de distances
for i in range(n):
    #et la deuxième boucle commence à i pour ne pas poser deux fois les mêmes questions à l'ordre près
    for j in range(i,n):
        if i != j:
            distance = int(input("Quelle distance sépare "+villes[i]+" de " +villes[j])+"? ")
            distances[i][j]=distance
            distances[j][i]= distance
```

ACTIVITÉ 2 :

Énoncé

Écrire un programme en Python, ou une fonction, qui génère la liste des chemins qui permettent de relier un certain nombre de points numérotés en passant une fois (et une seule !) par chaque point.

Ces listes seront des listes de nombres entiers.

Cela revient presque à compter les permutations de ces points, mais sans tenir compte de l'ordre (les chemins parcourus dans un sens ou dans l'autre sont les mêmes), et en ne gardant que les permutations qui commencent du même point (les chemins sont les mêmes, quel que soit le point de départ choisi).

Cela revient aussi à compter le nombre de polygones différents que l'on peut tracer entre ces points (si trois points de cet ensemble ne sont jamais alignés).

Ce n'est pas un problème simple…

Piste pour l'enseignant :

- On peut rapidement donner l'aide suivante sur une manière simple d'aborder le problème :
  « On ne va créer que les chemins qui commencent au point 0. Ils se termineront aussi en 0.
  Pour créer les chemins de longueur 5, il faut déjà connaître les chemins de longueur 4 : on obtient un chemin de longueur 5 en insérant le point 4 dans les chemins précédents.
  On l'insère d'abord entre 0 et le point suivant, puis entre ce deuxième point et le troisième, entre le troisième et le quatrième et enfin à la fin du chemin (avant de revenir en 0).
  Le chemin [0;1;3;2;0] permet de cette manière de créer les chemins : [0,4,1,3,2,0], [0,1,4,3,2,0] , [0,1,3,4,2,0] ; [0,1,3,2,4,0].
  Il n'y a qu'un seul chemin de longueur 3, c'est un bon point de départ.»
- On trouve aussi les permutations dans le module itertools, qu'il faudrait installer/importer suivant votre version de python.

Programme corrigé :

```
#on définit le nombre de points qu'on veut atteindre
n=6


#on crée une liste de chemins, qui contient le chemin de longueur 1
chemins=[[0,1,2,0]]
#on entre dans les boucles:
for i in range(3,n):
    #on crée une copie de la liste pour ne pas travailler directement dessus
    anciens_chemins=chemins.copy()
```

```
    #on travaille sur chaque chemin de cette liste
    for chemin in anciens_chemins:
        #et on crée de nouvelles listes avec la valeur i à toutes les positions possibles sauf la première et la dernière
        for k in range(1,i+1):
            nouveau_chemin=chemin.copy()
            nouveau_chemin.insert(k,i)
            chemins.append(nouveau_chemin)
    #on supprime ensuite les chemins trop courts en partant de la fin de la liste
    for j in range(len(chemins)-1,-1,-1):
        if len(chemins[j])==i+1:
            chemins.pop(j)
```

ACTIVITÉ 3 :

Énoncé

Créer un programme en Python qui à partir d'une liste de chemins entre des villes et des distances séparant ces villes calcule les longueurs totales des chemins. Les longueurs seront enregistrées dans une liste nommée *longueurs*.

Les villes seront désignées par des nombres.

Un chemin sera donc une liste de nombres indiquant les points de passages. Les chemins sont dans une liste appelée *chemins*.

Les distances entre deux villes seront données par une liste de listes appelée *distances*. la liste *distances[1]* contiendra par exemple les distances qui séparent la ville 1 des autres villes (y compris elle-même), dans l'ordre. Le nombre *distances[1][3]* indiquera la distance entre la ville 1 et la ville 3.

On pourra utiliser ces données (Berlin, Vienne, Bruxelles, Belgrade) pour tester le programme :

chemins = [[0,1,3,2,0], [0,1,2,3,0] , [0,2,1,3,0]]

distances = [[0, 929, 793, 1242], [929, 0, 1225, 614], [793, 1225, 0, 1725], [1242, 614, 1725, 0]]

Pistes pour l'enseignant :

- On peut donner l'aide suivante
  « Il faut tenir compte de la longueur de chaque chemin pour lire les informations de la liste de listes *distances*.
  Deux valeurs successives de la liste chemins[i] donnent les indices à utiliser pour prélever la distance dans la liste de listes distances : pour calculer la longueur qui correspond à chemins[2] (attention, c'est [0,2,1,3,0] dans l'exemple) il faut additionner les quatre distances entre la ville 0 et la ville 2, entre la ville 2 et la ville 1, entre la ville 1 et la ville 3 et enfin entre la ville 3 et la ville 0.
  La distance entre la ville 0 et la ville 2 est *distances[0][2]* . »
- On trouve aussi les permutations dans le module itertools, qu'il faudrait installer/importer suivant votre version de python.
- On pourrait dès cette étape demander la détermination de la plus petite valeur de cette liste *longueurs*.

Programme corrigé :

```
#on initialise les données

chemins = [[0,1,3,2,0], [0,1,2,3,0] , [0,2,1,3,0]]

distances = [[0, 929, 793, 1242], [929, 0, 1225, 614], [793, 1225, 0, 1725], [1242, 614, 1725, 0]]

#on crée une liste pour stocker les résultats

longueurs =[]

#on doit considérer chaque chemin
```

```
for i in range(len(chemins)):
    #prélever le chemin, c'est juste plus pratique pour les notations ensuite
    chemin=chemins[i]
    #initialiser la longueur du chemin à 0
    longueur = 0
    #et découper chaque chemin en segment:
    for k in range(len(chemin)-1):
        depart=chemin[k]
        arrivee=chemin[k+1]
        #et cumuler les longueurs des segments
        longueur = longueur + distances[depart][arrivee]
    #on enregistre le total dans la liste avant de retourner dans la boucle...
    longueurs.append(longueur)
```

ACTIVITÉ 4 :

Énoncé

Assembler les programmes précédents (éventuellement transformés en fonctions) pour pouvoir déterminer la longueur du plus court chemin reliant un certain nombre de capitales européennes, ainsi que le chemin correspondant.

Le nombre de capitales sera au choix de l'utilisateur.

Les capitales seront au choix de l'utilisateur.

Le programme devra annoncer la longueur trouvée, et l'ordre des villes du chemin minimal.

Pistes pour l'enseignant :

- Contraindre à l'utilisation de fonctions rend le problème plus compliqué, notamment en raison du statut (global ou local) des variables qui interviennent suivant le découpage en fonctions proposé ou imposé.
- On pourrait utiliser le module *time*, et évaluer le temps mis par le programme pour trouver la solution au problème proposé…

Programme corrigé :

```
### programme qui détermine le plus court chemin reliant n villes à partir de la donnée des distances séparant ces villes deux à deux


#partie 1 : choix du nombre de villes, saisie des distances

#on demande quel va être le nombre de villes, cela pourrait être dans le corps du programme

n = int(input("Avec combien de capitales travaille-t-on? "))

#on crée une liste vide pour les noms des villes

villes=[]

#ici la liste distances est créée par compréhension, mais on peut la créer à l'aide d'une boucle. Elle est remplie de listes vides.

distances = [[] for i in range(n)]

#on crée la liste des villes à l'aide d'une boucle, et on en profite pour remplir de 0 la liste des distances qui séparent cette ville des autres

for i in range(n):

    ville= input("Quelle est la ville " +str(i+1)+ " de la liste?")

    villes.append(ville)

    #distances[i] est ici aussi créé par compréhension, on aurait pu écrire une boucle

    distances[i]=[0 for j in range(n)]

#on utilise des boucles imbriquées pour compléter les listes de distances

for i in range(n):
```

```
    #et la deuxième boucle commence à i pour ne pas poser deux fois les mêmes questions à l'ordre près
    for j in range(i+1,n):
        distance = int(input("Quelle distance sépare "+villes[i]+" de " +villes[j]+"? "))
        distances[i][j]=distance
        distances[j][i]= distance


#partie 2 : création des chemins
#on crée une liste de chemins, qui contient l'unique chemin de longueur 3
chemins=[[0,1,2,0]]
#on entre dans les boucles:
for i in range(3,n):
    #on crée une copie de la liste pour ne pas travailler directement dessus
    anciens_chemins=chemins.copy()
    #on travaille sur chaque chemin de cette liste
    for chemin in anciens_chemins:
        #et on crée de nouvelles listes avec la valeur i à toutes les positions possibles sauf la première et la dernière
        for k in range(1,i+1):
            nouveau_chemin=chemin.copy()
            nouveau_chemin.insert(k,i)
            chemins.append(nouveau_chemin)
    #on supprime ensuite les chemins trop courts en partant de la fin de la liste
    for j in range(len(chemins)-1,-1,-1):
        if len(chemins[j])==i+1:
            chemins.pop(j)


#partie 3 : calculs de longueurs
#on crée une liste pour stocker les résultats des calculs de longueurs
longueurs =[]


#on doit considérer chaque chemin
for i in range(len(chemins)):
    #prélever le chemin, c'est juste plus pratique pour les notations ensuite
    chemin=chemins[i]
```

```
    #initialiser la longueur du chemin à 0
    longueur = 0
    #et découper chaque chemin en segment:
    for k in range(len(chemin)-1):
        depart=chemin[k]
        arrivee=chemin[k+1]
        #et cumuler les longueurs des segments
        longueur = longueur + distances[depart][arrivee]
    #on enregistre le total dans la liste avant de retourner dans la boucle...
    longueurs.append(longueur)

#partie 4 : recherche du minimum
#on initialise une variable à une très grande valeur, et un indice à une valeur arbitraire
min = 10000000
indice=0
for i in range(len(longueurs)):
    if min > longueurs[i]:
        min = longueurs[i]
        indice=i
#et on annonce le résultat
print("Le chemin le plus court est:")
for i in range(len(chemins[indice])):
    print(str(i) + "-   " + villes[chemins[indice][i]])
print ("Pour une longueur totale de "+ str(min) + " kilomètres.")
```