TP 4 : un nouveau type d'objets en Pyhton : les tuples.

1. Découverte

On considère la fonction suivante (fonction que nous avons vu la semaine dernière et qui permet de déterminer le vol, la hauteur et la longueur du vol d'un nombre entier selon l'algorihtme de Syracuse).

```
def vol(n):
    retour=str(n) # description du vol
    hauteur=n # hauteur max du vol fixée à n au départ
    long=1 # longueur du vol
    boucle=True # variable booléenne pour déterminer quand on met fin à
l'algorihtme
    nb=n # variable locale qui prend la valeur n de départ pour éviter des
effets de bords
    while boucle:
        if nb>hauteur : # on vérifie si le nombre dépasse la hauteur
            hauteur=nb # si oui, on modifie la hauteur
        long+=1 # on ajoute 1 étape, donc 1 à la longueur du vol
        if nb%2==0: # on applique l'algorihtme de syracuse
            nb=nb//2
        else :
            nb=nb*3+1
        retour+=","+str(nb) # on ajoute la valeur au vol
        if nb==1 : # on s'arrête lorsque nb==1
            boucle=False
    return retour,long,hauteur # on retourne les 3 valeurs attendues
```

On a ainsi par exemple, après exécution de cette fonction :

```
>>>vol(5)
('5,16,8,4,2,1', 6, 16)
```

La fonction renvoie un triplet de valeurs mises entre parenthèses.

Il s'agit d'une collection d'objets particulière à Python : les tuples.

```
>>>A=vol(5)
>>> type(A)
<class 'tuple'>
```

Comme une chaîne de caractères est une collection de caractères avec des méthodes propres, les tuples sont des collections d'objets avec leurs propres méthodes.

Ces méthodes sont communes pour beaucoup avec celles de la classe str.

```
>>> A[0]
'5,16,8,4,2,1'
>>> A[1]
6
>>> A[2]
16
```

Définir un tuple :

```
>>> A=1,2,3 # les () ne sont pas obligatoires
>>> A
(1, 2, 3)
>>> A=(1,2,3)
>>> A
(1, 2, 3)
>>> B=2, # définir un tuple ne contenant qu'un seul élément
>>> B
(2,)
>>> type(B)
<class 'tuple'>
>>> len(B)
1
>>> C=() # définir un tuple vide
>>> type(C)
<class 'tuple'>
>>> len(C)
0
```

Récupérer un élément composant un tuple : on procède comme avec les chaînes de caractères.

```
>>> A="az",12,"12"
>>> A
('az', 12, '12')
>>> A[0]
'az'
>>> A[-1]
'12'
>>A[1]
12
```

Les tuples peuvent contenir des tuples :

```
>>> A=((1,2,3),(4,5,6))
>>> A
((1, 2, 3), (4, 5, 6))
>>> len(A)
2
>>> A[0]
(1, 2, 3)
>>> A[1]
(4, 5, 6)
```

On peut explorer le contenu d'un tuple en profondeur :

```
>>> A=((1,2,3),(4,5,6))
>>> A[0][1] # on demande le second élément du premier élément de A
2
```

On peut ajouter un tuple à un tuple :

```
>>> A=1,2
>>> A
(1, 2)
>>> A=A+"az"
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: can only concatenate tuple (not "str") to tuple
>>> A=A+3
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: can only concatenate tuple (not "int") to tuple
>>> A=A+(3,)
>>> A
(1, 2, 3)
```

Les tuples sont des collections d'objets particulières : ils sont immutables, c'est à dire que l'on ne peut modifier leurs valeurs.

```
>>> A=(1,2)
>>> A
(1, 2)
>>> type(A)
<class 'tuple'>
>>> A[0]=2
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

On utilise le plus souvent des tuples pour manipuler des données dont on veut interdire la modification, en général des relevés statistiques de valeurs.

Quelques méthodes utiles sur les tuples

```
>>> A=(1,2,3,2,7)
>>> len(A) # renvoie la longueur du tuple cad son nombre d'éléments.
5
>>> max(A) # renvoie la valeur maximale
7
>>> min(A) # renvoie la plus petite valeur
1
>>> A.count(2) # compte le nombre de fois où la valeur 2 est présente dans le
tuple
2
>>> A.count(1) # compte le nombre de 1
1
>>> A.index(2) # renvoie le plus petit indice correspondant à la valeur 2
1
>>> A.index(3)
2
```

2. Retour au TP sur la conjecture de Syracuse.

Modifier la fonction vol pour la simplifier en utilisant les méthodes propres aux tuples.

```
def vol(n):
    retour=(n,) # description du vol, retour est un tuple
    boucle=True # variable booléenne pour déterminer quand on met fin à
l'algorihtme
    nb=n # variable locale qui prend la valeur n de départ pour éviter des
effets de bords
    while boucle:
        if nb%2==0: # on applique l'algorihtme de syracuse
            nb=nb//2
        else :
            nb=nb*3+1
        ....
        if nb==1 : # on s'arrête lorsque nb==1
            boucle=False
    return retour,...,... # on retourne les 3 valeurs attendues
```

1. Ajouter une fonction hauteur_max(n) qui déterminera quel est l'entier k qui possède le vol le plus haut

pour $1 \leq k \leq n$. La fonction retournera la valeur de k et la hauteur du vol associée.

2. Ajouter une fonction longueur_max(n) qui déterminera quel est l'entier k qui possède le vol

le plus long

pour $1 \leq k \leq n$. La fonction retournera la valeur de k et la longueur du vol associée.