Générer des listes pour le 421

Enoncé

On lance 3 dés à 6 faces parfaitement équilibrés et on se pose la question de la probabilité de réaliser un 421.

1. A l'aide du langage Python, définir la fonction **Lancer3des** qui simule le lancer de 3 dés à 6 faces et renvoie sous la forme d'un entier à 3 chiffres le résultat du lancer sans classement. Définir ensuite la fonction **ListeLancer** qui renvoie la liste des résultats obtenus pour les n lancers où n est passé en paramètre.
2. A l'aide du langage Python, définir la fonction **Frequence421** qui renvoie la fréquence f de réussite de l'expérience (obtenir un 421 au premier lancer de 3 dés). Cette fonction prend en paramètre la liste des expériences réalisées. Donner la fréquence en pourcentage obtenue pour 1000 expériences.
3. Et en théorie, quel pourcentage devrait-on avoir ?

1. Définir la fonction **Lancer3des**

L'exercice peut sembler proche dans sa forme de celui proposé sur le lancer de 3 dés où l'on s'interroge sur la probabilité de faire au moins un 6. Ici, on souhaite conserver la mémoire des différents résultats obtenus à chaque lancer et on va voir comment les listes en Python vont nous y aider. Cela nous permettra d'aborder différentes modalités de construction des listes.

On crée un script **LNCER421** de type **Simulation Aléatoire**. On importe ensuite la librairie **math** à l'aide de l'onglet Fns… puis Menu **Modul** et enfin **math…(**la librairie **random** est déjà importée).

On définit ensuite la fonction **Lancer3des** selon le script ci-contre. On utilise la fonction **randint(1,6)** trois fois pour générer trois nombres aléatoires entre 1 et 6 qui, combinés par multiplication de multiple de dix, renvoie un nombre entre 111 et 666. Dans notre simulation, 123 est un résultat différent de 321. 123 signifie que le dé n°1 a renvoyé le chiffre 1 tandis que 321 signifie que le dé 1 a renvoyé le chiffre 3.

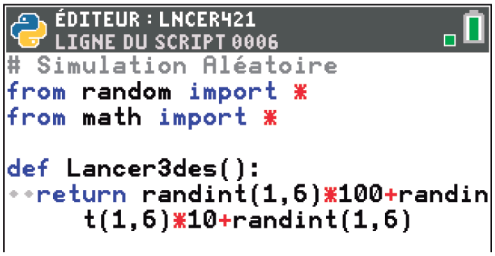
Intéressons nous à la définition de la fonction **ListeLancer**. Nous allons faire 3 propositions de rédaction pour un même résultat mais par des techniques de définition de listes différentes.

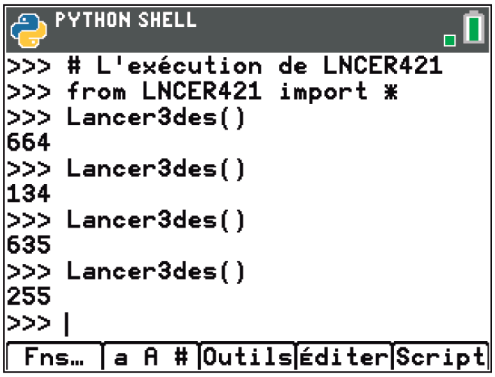
Commençons par saisir **ListeLancer1**. Il s'agit d'initialiser la liste par une liste de la taille du nombre **n** de lancers qui vont être réalisés et où chaque élément vaut 0. Ils seront ensuite remplacés par les **n** résultats obtenus à l'aide de la fonction **Lancer3des** précédente.

On teste alors la fonction **ListeLancer1** en console et il est intéressant de voir que nous sommes limités pour le nombre de lancers par la taille maximale d'éléments que peut contenir une liste. A mettre en regard avec un exercice précédent de lancers de dés où nous utilisions des compteurs et des accumulateurs pour notre travail et parvenions à réaliser 100 000 lancers.

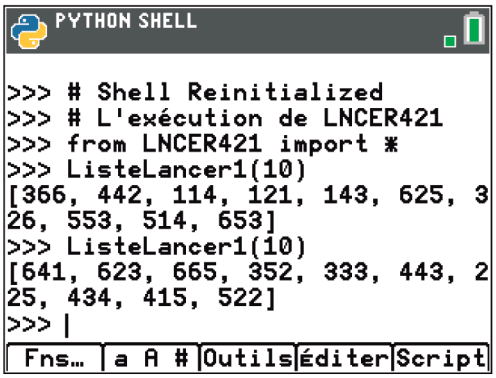
Il existe d'autres façons de construire des listes en Python.

En particulier, on ne connait pas toujours à l'avance le nombre d'éléments qu'occupera notre liste à la fin de l'algorithme.






```
def ListeLancer1(n):
••l = [0]*n
••for i in range(n):
••••l[i] = Lancer3des()
••return l
```



Générer des listes pour le 421

Aussi, saisissons maintenant **ListeLancer2**.

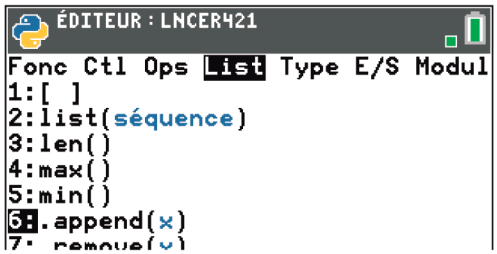
Il s'agit d'initialiser la liste avec la liste vide **[]**.

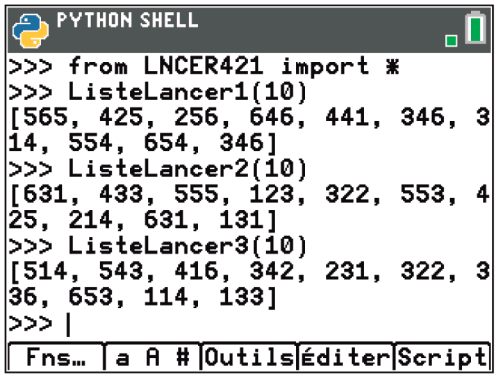
Cette fois-ci, chaque résultat obtenu avec la fonction **Lancer3des** est ajouté à la liste dont la taille grandit au fur et à mesure de la réalisation de l'algorithme.

Ces ajouts successifs d'éléments à la liste sont possibles à l'aide de la méthode **append** qui ajoute l'élément passé en paramètre à la liste depuis laquelle elle est appelée **l.append(Lancer3des())**.

Vous remarquerez dans le sous-menu **List** du menu Fns… le nombre de méthodes et fonctions disponibles pour travailler avec les listes. On en reparlera par la suite.

```
def ListeLancer2(n):
  l = []
  for i in range(n):
    l.append(Lancer3des())
  return l
```





Enfin, on souhaite présenter une dernière façon de générer une liste. Observer le code de la fonction **ListeLancer3** ci-contre.

Elle permet d'obtenir le même résultat que les fonctions précédentes en une seule ligne de code. On définit directement à l'interieur de la liste la description des éléments souhaités et leur nombre.

```
def ListeLancer3(n):
  return [Lancer3des() for i in
      range(n)]
```

2. Définir la fonction Frequence421

Il faut maintenant trier les résultats.

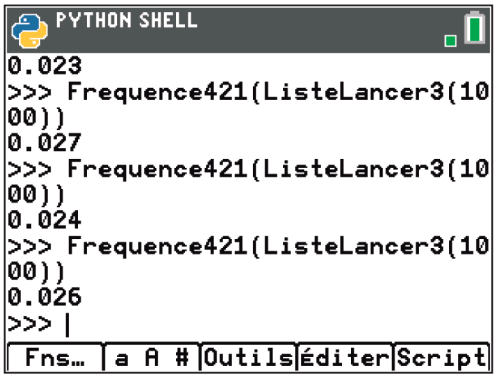
A l'aide de la méthode **count**, nous allons comptabiliser les occurrences de 124,142,214,241,412 et 421 dans la liste **l** passée en paramètre.

Nous utilisons un compteur pour sommer les différentes apparitions favorables et retournons la fréquence en divisant la valeur de **compteur** par le nombre d'éléments de la liste à l'aide de la fonction **len**.

Nous testons plusieurs fois en console notre fonction à l'aide de la commande **Frequence421(ListeLancer3(1000))**.

La fréquence obtenue fluctue et semble se situer aux alentours des 2,5 % dans notre courte série de tests.

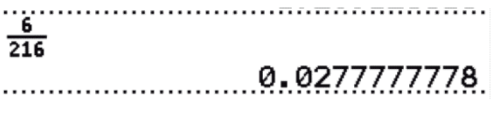
```
def Frequence421(l):
  resultat = [124,142,214,241,41
      2,421]
  compteur = 0
  for i in resultat:
    compteur += l.count(i)
  return compteur/len(l)
```



3. Et en théorie ?

A l'aide du dénombrement, on peut établir qu'il y a 6 possibilités qui nous sont favorables sur un total de $6 \times 6 \times 6 = 216$ possibilités. Ainsi, si on appelle P_{421} la probabilité de faire un 421 au premier lancer, on a :

$$P_{421} = \frac{6}{216} \approx 2{,}8\%.$$



Ce qui est proche des résultats de nos simulations précédentes.