Banque « Agro- Veto »
A TB - 0315

Algorithmique et Informatique
Durée : 45 mn

Si au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.
L'usage d'une calculatrice est interdit pour cette épreuve.

En biologie, certains processus complexes se prêtent bien à une modélisation probabiliste. En génétique par exemple, on peut modéliser l'expérience de Mendel sur des plantes hétérozygotes à deux caractères (pois verts ou jaunes) par des répétitions de "pile ou face".

En `Python`, *la fonction* `random` *(importée de la bibliothèque du même nom) ne prend aucun argument et renvoie un* ***nombre flottant*** *aléatoire de l'intervalle* $[0,1[$. *La fonction* `random` *peut être vue comme une variable aléatoire suivant la loi uniforme sur* $[0,1[$.
Exemple : `random()` *peut renvoyer* `0.8794190882576618`.
On considère cette fonction préalablement définie et on pourra l'utiliser sans condition dans tout le sujet.

1 Lancers d'une pièce

1. *Reproduire et compléter sur la copie le code de la fonction suivante de façon à ce qu'elle simule la réalisation d'une variable aléatoire* **à valeurs dans** $\{0;1\}$ *suivant une loi de Bernoulli de paramètre de succès* $p \in [0,1]$.

```
def pile(p):
    if random() < ... :
        return ...
    else:
        return ...
```

   La fonction `pile` *pourra être utilisée dans la suite du sujet.*

2. *Écrire une fonction* `lancers_piece` *qui prend pour arguments d'entrée un entier naturel* `n` *et un nombre flottant* `p`$\in]0,1[$, *qui simule* `n` *lancers indépendants d'une pièce dont la probabilité d'obtenir "pile" est* `p` *et qui renvoie le nombre de "piles" obtenus lors de ces* `n` *lancers.*

3. *Écrire une fonction* `premier_face` *qui prend pour argument d'entrée un nombre flottant* `p`$\in]0,1[$ *et qui renvoie le nombre de lancers nécessaires pour obtenir le premier "face" (en supposant toujours que les lancers sont indépendants et qu'à chaque lancer, la probabilité d'obtenir "pile" est* `p`*).*

   Par exemple, si l'instruction `premier_face(0.5)` *renvoie 3, c'est qu'elle a simulé trois lancers d'une pièce équilibrée et qu'on a obtenu successivement deux fois "pile" puis "face".*

4. *On note* $\mathcal{E}$ *l'expérience où l'on lance une pièce jusqu'à obtention du premier "face". On souhaite simuler la répétition de* `n` *expériences* $\mathcal{E}$ *indépendantes.*

   Écrire pour cela une fonction `liste_nb_lancers` *qui prend pour arguments d'entrée un entier* `n` *et un nombre flottant* `p`$\in]0,1[$, *qui renvoie une liste (type* `list` *en* `Python`*) de* `n` *entiers naturels, chacun représentant le nombre de lancers nécessaires pour obtenir le premier face lors de l'expérience* $\mathcal{E}$.

   Par exemple, si l'instruction `list_nb_lancers(3, 0.4)` *renvoie* `[4,1,2]`, *c'est qu'on a réalisé trois expériences avec une pièce dont la probabilité d'obtenir "pile" est égale à 0.4. Le premier face a été obtenu au quatrième lancer lors de la première expérience, au premier lancer lors de la seconde et au second lancer lors de la dernière expérience.*

5. *Écrire une fonction* `nb_occurrences` *qui prend pour arguments une liste de nombres* `liste` *ainsi qu'un nombre* `k` *et qui renvoie le nombre de fois où le nombre* `k` *apparait dans* `liste`.

6. *Écrire une fonction* `frequence` *qui prend pour arguments deux entiers naturels* `n` *et* `k`*, et qui renvoie la fréquence des expériences pour lesquelles il a fallu exactement* `k` *lancers pour obtenir "face", lors de la répétition de* `n` *expériences $\mathcal{E}$ indépendantes.*

2 Nombre maximal de résultats consécutifs

Dans la suite du sujet, les questions sont à choix multiples (plusieurs réponses sont possibles). Les réponses devront être recopiées sur la copie. Aucune justification n'est attendue.

1. *On considère le script suivant :*

```
suite = ""
for k in range(1,11):
    if random() > 0.7:
        suite = suite + "F"
    else:
        suite = suite + "P"
print(suite)
```

   a. *Quel est le type de la variable* `suite` *?*

   ☐ *entier* `int`
   ☐ *booléen* `bool`
   ☐ *liste* `list`
   ☐ *chaine de caractères* `str`.

   b. *Que peut afficher le script précédent ?*

   ☐ `"P"`
   ☐ `"FPPPPPPPPP"`
   ☐ `"FFFFFFFFFF"`
   ☐ `"FFPPFFPPFFP"`.

2. *Parmi les quatre fonctions suivantes, déterminer celle(s) qui permet(tent) de vérifier si une chaîne de caractères uniquement constituée des caractères F et P contient la sous-chaîne* `"FF"`*, i.e. deux caractères* `"F"` *consécutifs.*

   Par exemple, `double_facei("FPPFPF")` *renverra* `False` *et* `double_facei("FPFFPF")` *renverra* `True`.

☐

```
def double_face1(ch):
    for k in range(len(ch) - 1):
        if ch[k:k+2] == "FF":
            return True
    return False
```

☐

```
def double_face2(ch):
    for k in range(len(ch)):
        if ch[k:k+2] == "FF":
            return True
        else:
            return False
```

☐

```
def double_face3(ch):
    k = 0
    flag = False
    while k < len(ch) - 1 and not flag:
        if ch[k] == "F" and ch[k+1] == "F":
            flag = True
        k = k + 1
    return flag
```

☐

```
def double_face4(ch):
    compteur = 0
    for k in range(len(ch)):
        if ch[k] == "F":
            compteur = compteur + 1
    if compteur >= 2:
        return True
    else:
        return False
```

On rappelle que l'instruction `return` ***arrête le déroulement d'une fonction : tout code écrit après l'instruction*** `return`***, lorsqu'elle est exécutée, ne sera pas exécuté.***

On indiquera uniquement sur la copie le nom de la (ou des) fonction(s) qui répond(ent) à la question.

3. *On considère la fonction suivante :*

```
def fonction_mystere(ch):
    n = len(ch)
    compteur = 0
    for k in range(n):
        i = 0
        while k + i < n and ch[k + i] == "P":
            i = i + 1
        if i > compteur:
            compteur = i
    return compteur
```

Que renvoie chacune des instructions suivantes ?

a. *Instruction 1 :* `fonction_mystere("PFF")`

b. *Instruction 2 :* `fonction_mystere("PPPFFP")`

c. *Instruction 3 :* `fonction_mystere("PFFPPPPFFF")`

4. *Parmi toutes les sous-chaînes de caractères identiques et consécutifs d'une chaîne de caractères* `ch`*, on appelle* ***sous-chaîne maximale*** *toute sous-chaîne de caractères identiques et consécutifs dont la longueur est maximale.*

Exemple : dans la chaîne de caractères `ch = "PFFFPFPPPF"`*, il y a deux sous-chaînes maximales (de longueur 3) :* `"FFF"` *et* `"PPP"` *qui commencent respectivement aux indices 1 et 6 dans la chaîne* `ch`*.*

Modifier le code de la fonction précédente(`fonction_mystere`*) de façon à ce que la nouvelle fonction, qu'on nommera* `index_sous_chaine_max`*, détermine l'indice (dans la chaîne* `ch` *entrée en argument) du premier caractère de la plus grande sous-chaîne composée de caractères identiques. En cas d'existence de plusieurs sous-chaînes maximales, la fonction devra renvoyer l'indice de la dernière sous-chaîne maximale parcourue, i.e. le plus grand indice parmi ceux des premiers caractères des sous-chaînes maximales.*

Exemple : `index_sous_chaine_max("PFFFPFPPPF")` *devra renvoyer 6. En effet, la dernière sous-chaîne maximale,* `"PPP"`*, commence à l'indice 6 dans la chaîne de caractères* `"PFFFPFPPPF"`*.*

⋆⋆⋆ *FIN* ⋆⋆⋆