Algorithmique et Informatique

Durée : 45 minutes

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 `Python`, la fonction `randint` (importée de la bibliotèque `random`) prend deux arguments entiers a et b et renvoie un **nombre entier** choisi aléatoirement dans l'ensemble $[\![a, b]\!]$ des entiers compris entre a et b (bornes comprises). L'instruction `randint(a,b)` peut être vue comme la simulation d'une variable aléatoire suivant la loi uniforme sur $[\![a, b]\!]$.

  Exemple : `randint(4, 8)` *peut renvoyer* `4`, `7` *ou* `8` *mais pas* `9`.

  On considère cette fonction préalablement définie et on pourra l'utiliser sans condition dans le sujet.
- en `Python`, la fonction `input` permet d'interagir avec un utilisateur en l'invitant à entrer des caractères au clavier.

  Cette fonction renvoie une **chaîne de caractères**.

  Exemple : l'instruction `ch = input("Entrer un nombre : ")` *affiche (en interrompant le programme) la chaîne de caractères* `"Entrer un nombre : "`*. Si l'utilisateur entre* `1234` *(sans guillemets), la variable* `ch` *contient la chaîne de caractères* `"1234"` *(et non l'entier 1234).*

1. Questions préliminaires

1. On considère la chaîne de caractères `ch = "1234"`.

a. Quel est le type de `ch[0]` ? Quelle est sa valeur ?

b. Quel est le type de `int(ch[0])` ? Quelle est sa valeur ?

2. Écrire une fonction `stringToList` qui prend comme argument d'entrée une chaîne de caractères représentant un nombre entier positif (de longueur quelconque) et qui renvoie la liste des chiffres qui composent l'entier représenté par la chaîne de caractères.

Par exemple, l'instruction `stringToListe("1234")` *renverra la liste d'entiers* `[1, 2, 3, 4]`.

3. Quel est le type de `"1" + str(2)` ? Quelle est sa valeur ?

4. Écrire une fonction `listToString` réciproque de la fonction précédente, qui, à partir d'une liste de chiffres (compris entre 0 et 9) renvoie la chaîne de caractères correspondante.

Par exemple, l'instruction `listToString([1, 2, 3, 4])` *renverra la chaîne de caractères* `"1234"`.

2. Création d'un jeu numérique

On considère un jeu à un seul joueur dont le but est de déterminer une combinaison (générée aléatoirement par l'ordinateur) de `n` chiffres (compris entre 0 et 9).

- Au démarrage, le jeu génère une liste de `n` entiers (compris entre 0 et 9), inconnue du joueur, qu'on appellera `solution`.
- À chaque tour de jeu, le jeu demande au joueur d'entrer un entier à `n` chiffres (chacun compris entre 0 et 9). Après conversion de cette combinaison en une liste de `n` chiffres, qu'on appellera `proposition`, un algorithme renvoie au joueur le nombre de chiffres de `proposition` à la bonne place, puis le nombre de chiffres corrects mais mal placés **parmi les chiffres restants**.
- Le jeu s'arrête dès que le joueur a trouvé la combinaison `solution` ou que son nombre d'essais atteint le nombre maximal de tours `nbEssaisMax` fixé initialement par le jeu.

L'exemple ci-dessous illustre une partie entre l'utilisateur et la machine :

- Le jeu génère la liste `[4, 7, 7, 1]` comme `solution` (`n = 4`).
- Au premier tour, le joueur propose `1234` (converti en la liste `[1, 2, 3, 4]`). Le jeu affiche alors :

```
Nombre de chiffres bien placés : 0
Nombre de chiffres mal placés : 2
```

- Au second tour, le joueur propose `4477` (converti en la liste `[4, 4, 7, 7]`). Le jeu affiche alors :

```
Nombre de chiffres bien placés : 2
Nombre de chiffres mal placés : 1
```

- Au troisième tour, le joueur (inspiré !) propose 4771 (converti en la liste `[4, 7, 7, 1]`). Le jeu affiche alors :

```
Nombre de chiffres bien placés : 4
Nombre de chiffres mal placés : 0
GAGNÉ !
```

1) Construction des outils

1. À l'aide de la fonction `randint`, écrire une fonction `generateurSolution` qui génère aléatoirement (de manière indépendante), à partir d'un entier `n` passé en argument, une liste de `n` chiffres (chacun compris entre 0 et 9).

La fonction devra renvoyer la liste ainsi créée.

2. Écrire une fonction `distribution` qui, à partir d'une liste d'entiers compris entre 0 et 9, renvoie une liste `L` telle que, pour tout $i \in [\![0, 9]\!]$, `L[i]` contienne le nombre d'entiers `i` dans la liste passée en argument.

Par exemple, `distribution([4, 7, 7, 1])` *renverra la liste* `[0, 1, 0, 0, 1, 0, 0, 2, 0, 0]`. *En effet, la liste* `[4, 7, 7, 1]` *ne contient pas d'autres entiers que les entiers 1, 4 et 7 (qui apparaissent une fois pour les entiers 1 et 4, et deux fois pour 7). De même,* `distribution([4, 4, 7, 7])` *renverra la liste* `[0, 0, 0, 0, 2, 0, 0, 2, 0, 0]`.

3. On considère une fonction `nbEntiersCommuns` qui renvoie le nombre d'entiers en commun (mais pas nécessairement bien placés) dans deux listes d'entiers (compris entre 0 et 9) passées en argument.

Par exemple, `nbEntiersCommuns([4, 7, 7, 1], [4, 4, 7, 7])` *renverra 3.* En effet, les listes `[4, 7, 7, 1]` *et* `[4, 4, 7, 7]` *ont trois chiffres en commun : 4 et 7, qui apparaissent respectivement une fois et deux fois dans chacune des deux listes.*

Parmi les fonctions suivantes, indiquer (sans justifier) l'unique fonction qui correspond à celle décrite ci-dessus.

```
def nbEntiersCommuns1(liste1,liste2):
  s = 0
  for k in range(len(liste1)):
    s += min(liste1[k], liste2[k])
  return s
```

```
def nbEntiersCommuns2(liste1,liste2):
  nb = 0
  distrib1 = distribution(liste1)
  distrib2 = distribution(liste2)
  for k in range(10):
    nb += max(distrib1[k], distrib2[k])
  return nb
```

```
def nbEntiersCommuns3(liste1,liste2):
  nb = 0
  for k in range(len(liste1)):
    i = 0
    while i < len(liste2):
      if liste1[k] == liste2[i]:
        nb = nb + 1
  return nb
```

```
def nbEntiersCommuns4(liste1,liste2):
  s = 0
  distrib1 = distribution(liste1)
  distrib2 = distribution(liste2)
  for k in range(10):
    s += min(distrib1[k], distrib2[k])
  return s
```

4. Écrire une fonction `bienPlaces` qui, à partir de deux listes de même longueur qu'on appellera `proposition` et `solution`, renvoie le nombre d'éléments de la liste `proposition` qui sont bien placés, i.e. qui se trouvent aux mêmes indices dans les listes `proposition` et `solution`.

Par exemple, l'instruction `bienPlaces([1, 2, 3, 4], [4, 7, 7, 1])` *renverra 0 tandis que l'instruction* `bienPlaces([4, 4, 1, 7], [4, 7, 7, 1])` *renverra* 1.

On ne demande pas de vérifier que les deux listes passées en arguments ont la même longueur.

2) Programme principal

On souhaite alors coder le programme principal du jeu dans le cas où le nombre de chiffres de la combinaison est égal à 4 et où le nombre maximal d'essais est égal à 12.

Recopier, sans les commentaires, et compléter le code ci-dessous

Si le joueur perd, le programme devra l'indiquer à l'utilisateur et afficher la chaîne de caractères que le joueur devait déterminer.

```
# INITIALISATION
nbChiffres = ...
nbEssaisMax = ...
nbEssais = ...
nbBienPlaces = ...
# génération (aléatoire d'une combinaisaon à deviner)
solution = ...
# DÉBUT DU JEU
# affichage du nombre total d'essais
print("Vous avez au plus ", ..., " essais.")
# BOUCLE DE JEU
while ... < ... and ... < ...:
  # affichage du nombre d'essais restants
  print("Nombre d'essais restants : ", ...)
  # entrée d'un essai :
  ... = ... (input("Entrez une proposition : "))
  # détermination du nombre de chiffres bien/mal placés.
  nbBienPlaces = ...
  nbMalPlaces = ...
  # affichage du nombre de chiffres bien placés
  ...
  # affichage du nombre de chiffres mal placés
  ...
  # mise-à-jour
  nbEssais = ...
# FIN DU JEU
if ... :
  print("GAGNÉ !")
else:
  ...
```