+1/1/60+

Programmation impérative en Python — SPUF21

Année 2021-2022 — Seconde session

Nom : .....

Prénom : .....

Numéro d'étudiant : .....

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9
☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

Durée : 2 heures.

Aucun document n'est autorisé. L'usage de la calculatrice ou de tout autre appareil électronique est interdit.

Les exercices sont indépendants. Au sein d'un même exercice, vous pouvez utiliser les variables et fonctions des questions précédentes, même si vous n'avez pas su les faire ; chaque question est donc indépendante.

À part les méthodes et fonctions de base, vous n'avez pas le droit d'utiliser les fonctions et les méthodes « avancées », sauf si l'énoncé vous conseille l'utilisation de certaines d'entre elles.

```
# Fonctions autorisées
len(…)
range(…)
print(…)

# Méthodes autorisées
L.append(x) (ainsi que son équivalent pour les ensembles)
```

```
# Par exemple les méthodes et fonctions suivantes sont entre autres interdites
max(…) min(…) sum(…)
s.split(…) s.index(…) L.extend(…)

# Vous n'avez pas le droit d'utiliser des compréhensions ou des slices.
# À la place vous devez utiliser des boucles.
[ x for x in range(L) ]
chaine[début:fin:pas]
```

Exercice 1 Questions rapides . . . . . 3 points

☐ 0 ☐ 0,5 ☐ 1 ☐ 1,5 ☐ 2 ☐ 2,5 ☐ 3

1. Construire un ensemble `E` contenant trois éléments : `"Oui"`, `"Non"` et `"Bof"`.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

2. Écrire une fonction `nombre_s(c)` qui renvoie le nombre de lettres « `s` » minuscules dans la chaîne `c`. Par exemple `nombre_s("Grosses saucisses")` renverra 7.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

3. On définit ci-dessous une fonction `matrice(n)` qui renvoie une matrice de `n` lignes et `n` colonnes.

```
def matrice(n):
    ligne = [0] * n
    M = []
    for i in range(n):
        M.append(ligne)
    return M
```

En utilisant cette fonction `matrice(n)` on initialise une matrice `M` et on la modifie. Que vaut `M` après la modification ? Justifier.

```
>>> M = construire_matrice(3)
>>> M
[[0, 0, 0], [0, 0, 0], [0, 0, 0]]
>>> M[1][1] = 5
```

. . . . . .

. . . . . .

. . . . . .

. . . . . .

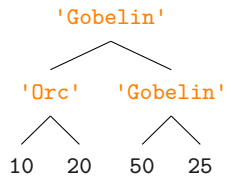
L'objectif de ce sujet sera de développer un jeu dans lequel le héros devra affronter de terribles monstres. Nous allons nous concentrer sur quelques aspects du jeu : tracé du logo, gestion de l'aléatoire, des personnages et des combats. L'écriture du jeu complet sera laissée en exercice facultatif à faire pendant les vacances.

Exercice 2 Arbres et labyrinthe . . . . . 2,5 points

☐ 0 ☐ 0,5 ☐ 1 ☐ 1,5 ☐ 2 ☐ 2,5

Dans cet exercice, vous ne pouvez utiliser que les fonctions suivantes :
— `arbre(r,Ag,Ad)` renvoie un arbre de racine `r` et de fils `Ag` (gauche) et `Ad` (droit) ;
— `est_feuille(A)` renvoie `True` si `A` est une feuille, et `False` sinon ;
— `racine(A)` renvoie la racine de l'arbre `A` ;
— `fg(A)` (ou `fd(A)`) renvoie le fils gauche (ou droit) de `A`.

L'arbre représente un labyrinthe, les feuilles sont des entiers représentant des sommes de pièces d'or et les racines (nœuds internes) représentent des monstres.



1. Écrire une fonction `somme_pièces_d_or(A)` qui prend un arbre et renvoie la somme des pièces d'or (c'est-à-dire la somme des valeurs comprises dans les feuilles).

. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

2. Écrire une fonction `nombre_gobelins(A)` qui prend un arbre et renvoie le nombre de gobelins.

. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

Exercice 3 Tracer le logo de notre jeu . . . . . 3 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3

Pour cet exercice, il vous est conseillé d'utiliser, si nécessaire, les trois fonctions vues en cours : `Dessin.create_line(p,q)`, `disque(centre,rayon)` et `cercle(centre,rayon)`. Par la suite on appellera point un couple de flottants `(x,y)`.

À titre indicatif, on donne ci-dessous le code à mettre au début du programme. Lisez bien les deux dernières lignes.

```
import tkinter as tk
import math
root = tk.Tk() # On crée une fenêtre
root.title("Logo du jeu")
# On crée un canvas (zone de dessin)
Dessin=tk.Canvas(root,height=500,width=500)
Dessin.pack()
(p0,p1,p2,p3,p4) = … # Ces cinq points sont définis par l'enseignant
L = [p0,p1,p2,p3,p4]
```

1. Que vaut la variable `choucroute` après exécution du code ci-dessous ?

```
choucroute = []
saucisse = 0
bière = 0
while bière < len(L): # On rappelle que L = [p0,p1,p2,p3,p4]
    choucroute.append(L[saucisse])
    saucisse = (saucisse+2)%5
    bière = bière+1
```

. . . . . .

. . . . . .

2. Écrire une fonction `tracer_polygone(liste_points)` qui prend une liste de points `q1`, `q2`, … , `qn`, et relie graphiquement en Tk les points correspondants. On n'oubliera pas de relier `qn` avec `q1`.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

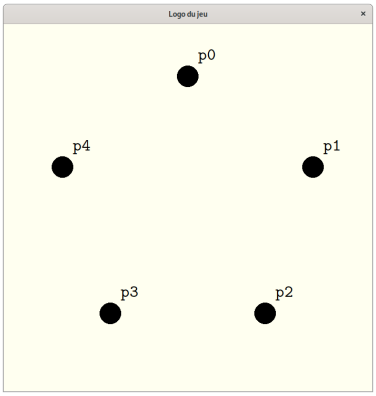
. . . . . .

. . . . . .

. . . . . .

. . . . . .

3. Que trace l'appel de `tracer_polygone(choucroute)` (`choucroute` étant la variable définie à la question 1) ? On tracera l'image obtenue sur la fenêtre ci-dessous en s'aidant des points `p0` à `p4` qui vous sont donnés.



Exercice 4 Aléatoire . . . . . 5 points

☐ 0 ☐ 0,5 ☐ 1 ☐ 1,5 ☐ 2 ☐ 2,5 ☐ 3 ☐ 3,5 ☐ 4 ☐ 4,5 ☐ 5

L'objectif de cet exercice est d'implémenter des petites fonctions pour gérer l'aléatoire dans notre jeu. Cela ne veut pas dire que vous pouvez répondre au hasard ! Vous utiliserez dans cette partie la fonction `randint(a,b)` qui renvoie un entier aléatoire dans l'intervalle $[a, b]$.

1. Écrire une fonction `lancer_dés(k,n)` qui simule `k` lancers de dés à `n` faces et renvoie la valeur obtenue. À titre d'illustration, `lancer_dés(3,10)` va simuler 3 lancers de dés à 10 faces, on obtient ainsi trois nombres aléatoires entre 1 et 10 — par exemple 2, 10 et 4 — et le résultat final sera la somme de ces valeurs, ici 16.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

2. On cherche maintenant à sélectionner `k` ennemis au hasard dans une liste L. On ne peut pas choisir deux fois le même ennemi. L'algorithme utilisé sera le suivant : on choisit un élément au hasard dans la liste L (avec la fonction `randint`) et on l'ajoute à un ensemble E. On continue jusqu'à ce que E contienne `k` éléments (forcément distincts car E est un ensemble). Quel est l'inconvénient d'un tel algorithme ?

. . . . . .

. . . . . .

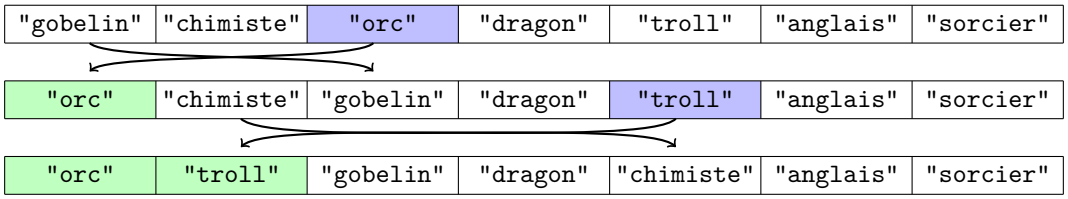
. . . . . .

3. Écrire la fonction `choix_non_efficace(k,L)` qui renvoie un ensemble de `k` éléments aléatoires et distincts d'une liste `L`. La fonction devra utiliser la méthode décrite à la question précédente.

4. On modifie l'algorithme pour être plus efficace. Le principe est le suivant : on choisit un indice aléatoire : `aléa` (par exemple 2), puis on échange les valeurs de `L[aléa]` et `L[0]` (ici `orc` et `gobelin`). `L[0]` contient alors notre premier choix. On recommence avec une nouvelle valeur de `aléa` (par exemple 4), puis on échange les valeurs de `L[aléa]` et `L[1]` (cette fois, on échange `troll` et `chimiste`).

Ainsi, au bout de `k` étapes, dans les `k` premiers éléments de la liste (représentés ici en vert), on aura `k` méchants choisis aléatoirement. On fera attention qu'à chaque étape le choix aléatoire du méchant se fasse parmi ceux qui n'ont pas encore été choisis.

Écrire la fonction `choisir(k,L)` qui prend une liste `L` et renvoie une liste de `k` méchants choisis aléatoirement dans la liste `L`. On aura ici évidemment le droit de modifier la liste `L` en permutant ses éléments.



Exercice 5 À l'assaut des monstres ..... 5 points

☐0 ☐0,5 ☐1 ☐1,5 ☐2 ☐2,5 ☐3 ☐3,5 ☐4 ☐4,5 ☐5

Dans cet exercice et le suivant, un personnage est représenté par un triplet `(nom,vie,puissance)`.

1. Écrire une fonction `blesser(personnage,p)` qui à partir d'un triplet `personnage`, renvoie le nouveau triplet dans lequel le personnage a perdu `p` points de vie. Par exemple `blesser( ("Bob",120,5) ,20)` renverra `("Bob",100,5)`. Si les points de vie obtenus sont négatifs, on les mettra à 0.

.....

.....

.....

.....

.....

.....

.....

.....

2. On se donne une liste non vide d'adversaires (représentés par des triplets). Écrire une fonction `plus_puissant(adversaires)` qui renvoie celui qui a la puissance la plus grande.

.....

.....

.....

.....

.....

.....

.....

.....

3. Écrire une fonction `moyenne(liste)` qui renvoie la puissance moyenne des adversaires de la liste.

.....

.....

.....

.....

.....

.....

.....

4. Nous allons maintenant écrire la fonction qui simule un combat entre deux opposants. Lorsqu'un personnage attaque, il commence à lancer k fois un dé à n faces (on utilisera la fonction `lancer_dés(k,n)` de l'exercice 4).

   — k correspond aux points de vie divisés par 100. Par exemple un personnage ayant 1204 points de vie pourra lancer 12 fois le dé ;

   — n correspond à sa puissance.

Le nombre ainsi obtenu correspond à la blessure à infliger au défenseur (voir la question 1).

Un combat est organisé de manière relativement simple : les adversaires se tapent dessus l'un après l'autre (comme expliqué dans le paragraphe précédent) jusqu'à ce que l'un des deux meure. Écrire une fonction `combat(gentil,méchant)` qui simule un combat et renvoie le nom du vainqueur. Le gentil attaquera le premier.

Exercice 6 Créer un personnage ..... 1,5 points

☐0 ☐0,5 ☐1 ☐1,5

L'usage d'un triplet n'étant guère pratique, nous allons maintenant représenter les personnages avec une classe `Personnage`. Un personnage sera défini par quatre attributs : son nom, ses points de vie (qui peuvent changer durant la partie), son endurance (correspondant au nombre maximal de points de vie ; ce nombre reste le même durant toute la partie) et sa force.

`nom`	chaîne
`endurance`	entier
`vie`	entier
`force`	entier

1. Créez une classe Personnage avec sa méthode d'initialisation. Chaque objet de cette classe devra posséder quatre attributs : son nom, son endurance, sa vie et sa force. Lors de l'initialisation, la vie sera égale à l'endurance. Pour créer un personnage, il suffira alors de faire :

```
1 perso = Personnage("Conan le barbare",1664,20)
```

L'attribut `nom` vaudra alors `"Conan le barbare"`, les attributs `vie` et `endurance` vaudront 1664 et la `force` 20.

.....

.....

.....

.....

.....

.....

.....

.....

2. Écrivez une méthode `soigner` qui réinitialise les points de vie à leur valeur maximale (la valeur maximale étant comprise dans l'attribut `endurance`).

.....

.....

.....

.....

3. Créez une fonction (et non une méthode) `est_mort(perso)` qui prend un objet `perso` de la classe `Personnage` et renvoie `True` si le nombre de points de vie est nul ou négatif et `False` sinon.

.....

.....

.....

.....

4. Passez de bonnes vacances.