LE GOUVERNEMENT
DU GRAND-DUCHÉ DE LUXEMBOURG
Ministère de l'Éducation nationale,
de l'Enfance et de la Jeunesse

EXAMEN DE FIN D'ÉTUDES SECONDAIRES CLASSIQUES
2019

BRANCHE	SECTION(S)	ÉPREUVE ÉCRITE
Informatique	**B**	*Durée de l'épreuve :* 3h *Date de l'épreuve :*

Vous sauvegarderez votre travail dans un dossier nommé suivant les instructions du professeur. Chaque question devra être sauvegardée dans un nouveau fichier Python. En haut de chaque fiche vous indiquerez en tant que commentaire votre numéro de candidat. A la fin de l'épreuve vous imprimerez et signerez chaque fiche.

Attention : Les seules importations permises sont **`randrange`** et **`randint`** du module **`random`**, **`sqrt`** du module **`math`** ainsi que **`pygame`**, **`pygame.locals`** et **`sys`**.

Question 1 — 10 (=5+5) points

Cette question sera sauvegardée dans un fichier nommé **`question1.py`**.

(1) Implémenter la classe **`Stack`** qui permet de manipuler des piles du type LIFO (« Last In First Out »). Cette classe possède deux ***attributs*** :

- **`nb`** : le nombre d'éléments de la pile ;
- **`items`** : la liste des éléments de la pile.

Les ***méthodes*** de la classe sont :

- **`__init__`** : permet de créer une pile vide correctement initialisée ;
- **`isEmpty`** : retourne **`True`** si la pile est vide, **`False`** sinon ;
- **`push`** : permet d'ajouter un élément au sommet de la pile ;
- **`pop`** : enlève et retourne l'élément au sommet de la pile, si elle n'est pas vide.

(2) ***Application*** : En utilisant la classe **`Stack`** et toutes ses méthodes, écrire une fonction **`reverse_order(s)`** qui, pour une chaîne de caractères **`s`** donnée, retourne la chaîne dans laquelle l'ordre des lettres de **`s`** est inversé. Par exemple : **`reverse_order('python')`** retourne **`'nohtyp'`**. Tester la fonction sur cet exemple dans le programme principal. (Il est évidemment interdit d'utiliser la méthode **`reverse()`** des listes ou la fonction **`reversed()`**).

Question 2 — 10 (=3+1+5+1) points

Cette question sera sauvegardée dans un fichier nommé **`question2.py`**.

(1) Ecrire une fonction **`random_matrix(rows,cols,maxi)`** qui retourne une matrice à **`rows`** lignes et **`cols`** colonnes dont les éléments sont des ***entiers aléatoires*** de l'intervalle [0, **`maxi`**]. Les matrices seront représentées par des listes de listes. Par exemple :

`random_matrix(3,4,20)` pourrait retourner la matrice :

```
[[15, 12, 16, 0], [3, 15, 10, 11], [3, 20, 16, 8]]
```

(2) Ecrire une fonction **`display(m)`** qui affiche ligne par ligne une matrice **`m`** d'entiers. Voici par exemple la sortie obtenue par l'appel **`display(my_matrix)`** où **`my_matrix`** est la matrice de la question précédente :

```
[15, 12, 16, 0]
[3, 15, 10, 11]
[3, 20, 16, 8]
```

(3) Imaginons maintenant le jeu suivant : chaque entier dans une matrice du type précédent représente un nombre de pièces de 1 €. On parcourt la matrice en partant de l'élément (0,0) (en haut à gauche) et en se déplaçant toujours soit d'une colonne vers la droite, soit d'une ligne vers le bas, jusqu'à ce que l'on atteigne le dernier élément de la matrice (en bas à droite). On empoche toutes les pièces rencontrées durant ce parcours.

On demande d'écrire une fonction récursive **`s(m,x,y)`** qui retourne la somme maximale possible qu'on peut obtenir dans ce jeu pour une matrice donnée **`m`**, les paramètres **`x`** et **`y`** étant respectivement les indices de la dernière ligne et de la dernière colonne de **`m`**. Pour la matrice **`my_matrix`** ci-dessus, la valeur de retour de la fonction sera :

```
15 + 12 + 15 + 20 + 16 + 8 = 86
```

A titre d'information, le parcours à suivre (pas demandé) est indiqué en rouge ci-dessous :

```
[15, 12, 16, 0]
[3, 15, 10, 11]
[3, 20, 16, 8]
```

Indication : En notant **`s(m,i,j)`** la somme maximale qu'on peut obtenir dans le jeu en parcourant la matrice **`m`** jusqu'à l'élément **`(i,j)`**, on a la formule de récurrence suivante, valable pour **`i`** ≥ 1 et **`j`** ≥ 1 :

```
s(m,i,j) = m[i][j] + max(s(m,i-1,j),s(m,i,j-1))
```

En effet, pour arriver en **`(i,j)`**, il faut obligatoirement passer par **`(i-1,j)`** ou **`(i,j-1)`**.
Il appartient au candidat de calculer la somme maximale dans les cas de base :

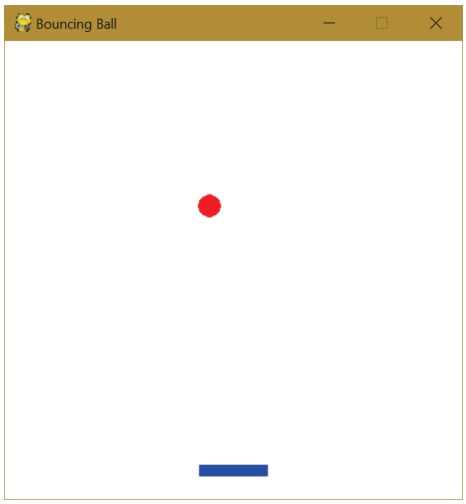
a) **`i=j=0`** b) **`i=0`** et **`j>0`** c) **`i>0`** et **`j=0`**

(4) Tester la fonction **`s`** sur une matrice aléatoire à 4 lignes et 6 colonnes avec des entiers dans [0,10], engendrée par la fonction **`random_matrix`**. On demande aussi d'afficher cette matrice à l'aide de la fonction **`display`**.

Question 3 — 40 (=3+2+4+(4+(2+5+9))+8+1+2) points

Cette question sera sauvegardée dans un fichier nommé **`question3.py`**.

Le but de la question est de programmer une version **`pygame`** du jeu « Bouncing Ball ». Le jeu consiste à faire rebondir autant de fois que possible une balle en mouvement sur une palette rectangulaire pour éviter qu'elle ne sorte par le bord inférieur de l'écran. Le joueur peut déplacer la palette vers la droite ou vers la gauche en utilisant les flèches de direction correspondantes du clavier. La balle rebondit automatiquement sur les bords de gauche, de droite et le bord supérieur de l'écran.



(1) Initialiser l'écran **`pygame`** avec une largeur et une hauteur égales à **`SIZE = 400`**. On utilisera la constante **`SIZE`** tout au long du programme, même dans les classes, le programme devant continuer à fonctionner correctement si l'on change cette constante. Le titre de la fenêtre est **`Bouncing Ball`**. Le jeu devra tourner avec une fréquence de rafraîchissement **`FPS = 60`**. Un clic sur la croix de fermeture permettra de quitter l'environnement **`pygame`** et de terminer l'application.

(2) Ecrire une fonction **`distance(x1,y1,x2,y2)`** qui calcule la distance entre deux points **`(x1,y1)`** et **`(x2,y2)`**.

(3) Définir la classe **`Paddle`** (la palette rectangulaire en bas de l'image).

a) Le constructeur initialise les attributs suivants sur les valeurs passées aux paramètres :

- **`x`** et **`y`** : les coordonnées du coin supérieur gauche de la palette ;
- **`width`** : la largeur de la palette ;
- **`height`** : la hauteur de la palette (valeur par défaut : 10 pixels) ;

b) Les méthodes de la classe **`Paddle`** sont :

- **`draw(self, screen)`** : permet de dessiner en bleu la palette sur l'écran **`screen`**.
- **`move(self, dx)`** : permet de déplacer horizontalement la palette de **`dx`** pixels, le sens de déplacement dépendant du signe de **`dx`**. (On accepte que la palette puisse être déplacée en dehors de l'écran.)

(4) Définir la classe **`Ball`** (le disque rouge).

a) Le constructeur initialise les attributs suivants sur les valeurs passées aux paramètres :

- **`x`** et **`y`** : les coordonnées du centre de la balle ;
- **`radius`** : le rayon de la balle ;
- **`speed`** : la vitesse absolue de la balle (valeur par défaut : 4)

Les attributs ci-dessous sont également initialisés dans le constructeur, mais ce ne sont pas des paramètres :

- **`score`** : le score de la balle, initialisé à **`0`**. Par définition le score de la balle est le nombre de fois qu'elle va heurter la palette pendant le jeu.
- **`alive`** : indique si la balle est « en vie » ou « morte », initialisé à **`True`**.
- **`xspeed`** et **`yspeed`** : les composantes horizontales et verticales du vecteur vitesse de la balle. D'après le cours de physique, on a la relation suivante :

$$(*) \qquad \texttt{xspeed}^2 + \texttt{yspeed}^2 = \texttt{speed}^2$$

où **`speed`** est l'attribut vitesse ci-dessus. **`xspeed`** sera initialisé à un ***entier*** aléatoire dans l'intervalle [**`-speed+1`**, **`speed-1`**]. **`yspeed`** (**`float`**) sera calculé de façon à ce que la relation $(*)$ soit vérifiée et que la balle se déplace initialement vers le haut de l'écran (pour donner au joueur un temps de réaction suffisant au démarrage du jeu).

b) Les méthodes de la classe **`Ball`** sont les suivantes :

- **`draw(self, screen)`** : permet de dessiner en rouge la balle sur l'écran **`screen`**.
- **`hit_paddle(self, paddle)`** : retourne **`True`** si la balle touche la palette **`paddle`** en venant d'en haut. Pour simplifier, on admettra qu'une collision a lieu dans l'un des trois cas suivants :
  a) le ***point inférieur*** de la balle tombe au-dessous du ***bord supérieur*** de la palette ;
  b) la distance du ***centre de la balle*** au ***coin supérieur gauche*** de la palette devient plus petite que le rayon de la balle ;
  c) la distance du ***centre de la balle*** au ***coin supérieur droit*** de la palette devient plus petite que le rayon de la balle ;
  La méthode retourne **`False`** au cas contraire.
- **`move(self, paddle)`** : permet de déplacer la balle. Si la balle touche l'un des bords de gauche, de droite ou le bord supérieur, elle doit rebondir correctement, sans devenir plus lente. Si elle entre en collision avec la palette, son **`score`** sera incrémenté de 1 et son vecteur vitesse (**`xspeed`**, **`yspeed`**) sera réinitialisé aléatoirement, comme dans le constructeur. Dès que le ***centre de la balle*** tombe au-dessous ***de l'ordonnée du bord inférieur de la palette***, la balle « meurt ». La balle ne peut être déplacée que si elle est « en vie », auquel cas on incrémente les coordonnées **`x`** et **`y`** du centre de **`xspeed`** et **`yspeed`** respectivement.

(5) Dans le programme principal :
   a) Créer la palette **`paddle`** avec une largeur de 60 pixels, une hauteur de 10 pixels, de sorte qu'elle soit centrée horizontalement et que son bord supérieur se trouve à 30 pixels du bord inférieur de l'écran.
   b) Créer la balle **`ball`** au centre de l'écran avec un rayon de 10 pixels et une vitesse initiale égale à 4.
   c) Ecrire la boucle principale : A chaque fois qu'on appuie sur l'une des flèches de direction **`K_LEFT`** ou **`K_RIGHT`**, la palette se déplace de 5 pixels dans la direction correspondante. Le mouvement doit se poursuivre sans arrêt aussi longtemps que le joueur maintient l'une de ces touches enfoncée. La balle est déplacée une fois à chaque passage dans la boucle. La boucle tourne aussi longtemps que le joueur n'a pas cliqué sur la croix de fermeture et que la balle est « en vie ».

(6) Avant la terminaison du programme, le **`score`** de la balle (donc en fait du joueur) et le message **`GAME OVER`** sont affichés dans la console.

(7) Pour rendre le jeu plus difficile, on demande d'ajouter une instruction à la méthode **`move`** de la classe **`Ball`** pour qu'après chaque série de 5 collisions avec la palette, la vitesse absolue de la balle augmente de 1.