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
Sessions 2022

DISCIPLINE	SECTION(S)	ÉPREUVE ÉCRITE	
Informatique	CB	*Date de l'épreuve :*	22.09.22
		Durée de l'épreuve :	08:15 - 11:25
		Numéro du candidat :	

- Dans votre répertoire de travail (à définir par chaque lycée), vous trouverez un dossier nommé **EXAMEN_CB**. Renommez ce dossier par votre numéro de candidat (p.ex. : **LXY_CB1_07**).
- Tous vos fichiers devront être sauvegardés à l'intérieur de ce dossier.
- Les seules importations permises dans cet examen sont les librairies **sys**, **pygame**, **pygame.locals**, ainsi que les fonctions **random** et **randint** de la librairie **random**.
- Ajoutez votre numéro de candidat en tant que commentaire en haut de chaque code-source.

Partie au choix. Choisissez une question parmi les deux suivantes et indiquez votre choix avec un X.			
Question	**Nb points**	**Sujet**	**Choix du candidat**
Question 1	11 points	dérivée	
Question 2	11 points	find_first	
Partie obligatoire			
Question	**Nb points**	**Sujet**	**Obligatoire**
Question 3	49 points	breakout	**X**

Question 1 : dérivation de fonctions simples (11 points) :

On se propose de dériver une fonction de la forme $f(x) = ax^n \ avec\ a, n \in \mathbb{Z}$.

On suppose une notation mathématique naturelle: `"2x"` au lieu de `"2x^1"`, `"-x"` au lieu de `"-1x^1"`, `"3"` au lieu de `"3x^0"` etc.

$f(x)$ sera implémenté sous forme d'une chaîne de caractères, p. ex.: `"2x"`, `"-x^-1"`, `"12x^-3"`

Dans le programme **q1_diff.py** écrivez la fonction **differentiate** au paramètre **f** de type **str** (dont le format est supposé correct) qui doit retourner une chaîne contenant f'(x). On demande que les résultats soient formatés comme dans les exemples du tableau ci-contre.

Exemples d'exécution:

```
"3x^2"   => "6x"
"-5x^3"  => "-15x^2"
"x^-2"   => "-2x^-3"
"5x"     => "5"
"-x"     => "-1"
"42"     => "0"
```

Validez votre code en calculant et affichant la dérivée de $-1/x$ dans le programme principal (affichage de `x^-2`).

Veillez à traiter correctement les cas particuliers des coefficients 1 ou –1 et de l'exposant 1 au niveau du résultat. (p. ex. : `"-1x^1"` → `"-x"`)

Question 2 : find avec caractère(s) générique(s) (11 points) :

On se propose de rechercher une clé dans une chaîne. Cette clé peut comporter un ou plusieurs caractères génériques "**?**". Le caractère générique "**?**"remplace un et un seul caractère quelconque.

Ainsi une recherche de `"j??e?"` trouverait aussi bien "**joies**" que "**jouer**", "**jaden**" ou "**jx§e4**".

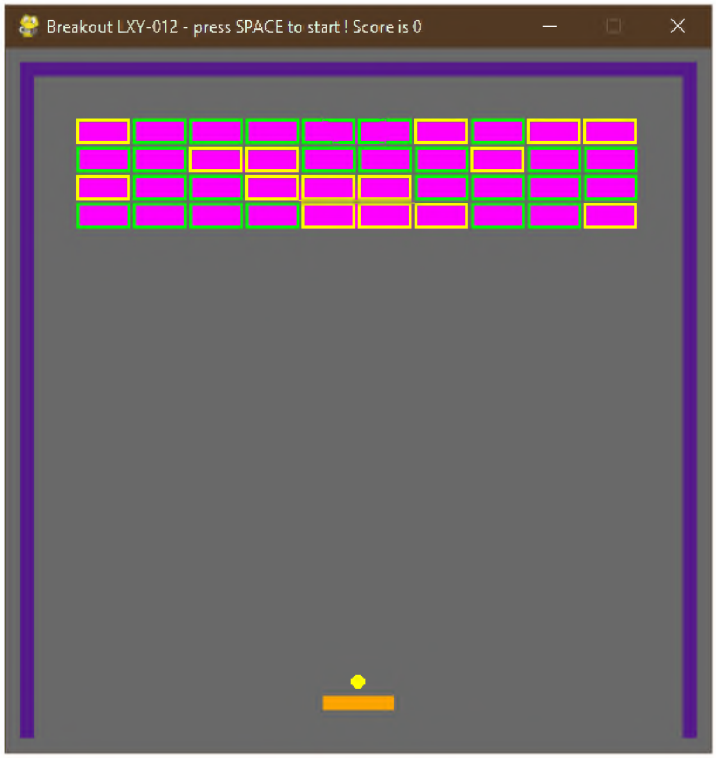
Dans le programme **q2_find.py** écrivez la fonction **find_first** au paramètres **needle** et **haystack** de type **str** retournant la position de la première occurrence de la clé **needle** dans **haystack**, -1 s'il n'y en a pas. L'utilisation des fonctions prédéfinies **find** ou **rfind** de Python est interdite.

Validez votre code en affichant dans le programme principal le résultat de la recherche de :

`"g??b"` dans `"That's the good thing about being grabbed as president."` (affichage de 20)

(Idée: utiliser une fonction auxiliaire **match** qui vérifie l'égalité de deux mots passés par paramètre, dont le 1er peut contenir un ou plusieurs caractères génériques "?".)

Question 3 : Création du jeu « Breakout » (49 points) :



Breakout est un jeu de type arcade qui consiste à éliminer une série de briques à l'aide d'une balle et d'une raquette.

La balle, initialement lancée du milieu de la base inférieure, rebondit de chaque objet (murs, briques, raquette). Chaque brique dispose d'une résistance variable qui diminue à chaque fois qu'elle est touchée par la balle. Une brique à résistance nulle est retirée du plateau de jeu. Le jeu est gagné lorsque toutes les briques ont été détruites et il est perdu lorsque la balle passe en-dessous du niveau de la raquette.

Créez l'application **q3_breakout**, basée sur les classes **Brick**, **Ball** et **Game** et faisant appel à la librairie **pygame**.

Afin de vous familiariser avec le fonctionnement du programme, vous trouverez une version exécutable dans votre dossier. Référez-vous à cet exécutable et aux instructions ci-dessous pour réaliser votre programme.

I) Chargement des bibliothèques et variables globales (1 p.)

Déclarez ensuite les constantes globales suivantes:

- **SIZE**, initialisée à 500, indique la taille de la fenêtre carrée de l'application
- **FPS**, initialisée sur 100, indique la cadence de répétition de la boucle **pygame**.

II) La classe Brick : (1 + 1 + 2 + 2 + 2 = 8 p.)

La classe **Brick** modélise les briques, les murs et la raquette. Elle décrit un rectangle avec les coordonnées (**x**, **y**) du coin supérieur gauche, les dimensions **width** et **height**, la vitesse de déplacement horizontale **speed**, la couleur **color** et la résistance **resistance**. La résistance d'une brique peut prendre les valeurs 0, 1 ou 2.

Brick
x, y, width, height, speed, resistance : int
color : Color
__init__(x : int, y : int , w : int, h : int, c : Color)
bash() : int
move(left : int, right : int)
contains(x : int, y : int) : bool
draw()

La classe **Brick** dispose des méthodes suivantes :

1) le **constructeur** initialise tous les attributs sur les valeurs passées aux paramètres respectifs, sauf la vitesse qui est initialisée à zéro. A défaut, la résistance est aussi initialisée à zéro ; (1 p.)
2) **bash** réduit la résistance d'une unité, puis retourne cette nouvelle valeur ; (1 p.)
3) **move** déplace la brique de **speed**, mais sans jamais sortir ni entièrement, ni partiellement de l'intervalle [left, right] passé par paramètre ; (2 p.)
4) **contains** retourne **True** si le point aux coordonnées x et y, fournis comme paramètres est à l'intérieur de la brique (bords compris), **False** sinon ; (2 p.)
5) **draw** dessine la brique sur la toile **pygame** sous forme de rectangle rempli de couleur **color**. Lorsque la résistance vaut 1 (respectivement 2) on ajoute un cadre vert (respectivement jaune) de mêmes dimensions et d'épaisseur 2px. (2 p.)

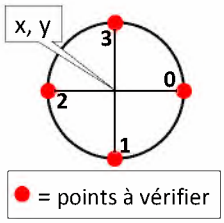
III) La classe Ball : (1 + 3 + 8 + 1 = 13 p.)

La classe **Ball** décrit la balle de jeu sous forme de disque jaune avec les coordonnées (**x**, **y**) de son centre, son rayon **radius** et les composantes **vx** et **vy** de son vecteur-vitesse.

Ball
x, y, vx, vy : float
radius : int
__init__(x : int, y : int , radius : int)
check_hit(brick : Brick) : int
move(brick_list : [Brick]) : int
draw()

La classe **Ball** dispose des méthodes suivantes :

1) le **constructeur** initialise les attributs sur les valeurs passées aux paramètres respectifs et les composantes **vx** et **vy** du vecteur vitesse respectivement sur 2 et -2 ; (1 p.)
2) **check_hit** vérifie une intersection entre la balle et un obstacle. Afin de garder simple la logique de rebondissement, on se limite à vérifier si l'un des 4 points extrêmes de la balle est inclus dans l'objet à vérifier (bords inclus).
La méthode retourne le numéro du point de contact, -1 s'il n'y en a pas. (3 p.)
(Idée: créer une liste temporaire avec les 4 points à vérifier.)



3) **move** effectue un déplacement et, le cas échéant, un rebondissement de la balle. Le rebondissement se fait sur chaque objet (brique / mur / raquette) de la liste fournie en paramètre, mais s'arrête au premier objet touché. La balle rebondit sur chaque surface d'objet: le rebondissement s'obtient par un changement de signe de la composante du vecteur-vitesse concernée au moment du contact. L'indice de liste du premier objet touché (-1 à défaut de collision) est retourné.

   Pour pimenter le jeu, le rebondissement sur la raquette et uniquement sur la raquette fixe la composante verticale du vecteur vitesse sur une valeur décimale aléatoire ∈]-3, -1] (8 p.)

   Notons que la liste **brick_list** contiendra toujours :
   - la raquette à l'indice 0,
   - les murs aux indices 1, 2 et 3 et
   - les briques aux indices >= 4 (voir IV-1-a) ;
4) **draw** dessine la balle sur la toile **pygame** sous forme de disque jaune. (1 p.)

IV) La classe Game : (11 + 6 + 1 = 18 p.)

La classe **Game** décrit et gère le jeu. A cet effet, elle se sert de la liste **bricks**, qui contient non seulement les briques à éliminer, mais aussi – au début de la liste et dans cet ordre – la raquette et les trois murs.

La classe **Game** dispose des attributs suivants :

Game
bricks : [Brick]
paddle : Brick
score, state : int
messages: [str]
ball : Ball
__init__()
do_step()
draw()

a. la liste **bricks** de tous les éléments de jeu (raquette, murs, briques) ;

b. la balle de jeu **ball** de type **Ball** ;

c. la raquette **paddle** de type **Brick** ;

d. le **score** qui contient les points obtenus ;

e. **status** mémorise l'état actuel du jeu et est une des valeurs du tableau repris sous f. ci-dessous ;

f. la liste **messages** décrivant les différents états du jeu selon l'état **status** :

status	message
0	"Breakout LXY-012 - press SPACE to start ! Score is <score>"
1	"Breakout LXY-012 - running, your score is <score>"
2	"Breakout LXY-012 - game paused, your score is <score>"
3	"Breakout LXY-012 - game lost with a score of <score>"
4	"Breakout LXY-012 - GAME WON with a score of <score>"

en remplaçant `<score>` par le **score** effectif et «`LXY-012`» par **votre code d'examen**. Ces messages seront inscrits dans l'entête de la fenêtre d'application.

La classe **Game** dispose des méthodes suivantes :

1) Le **constructeur** effectue les opérations ci-dessous : (11 p.)

a) il initialise la liste **bricks** à une liste vide, puis ajoute à cette liste - dans cet ordre - les éléments suivants :

- la raquette **paddle**, de taille 50 x 10 px, au milieu de l'écran à 40 px du bord inférieur et de couleur **orange** ;
- les trois murs, tous de longueur **SIZE** - 20 px et d'épaisseur 10 px, positionnés à une distance de 10 px des bords de la fenêtre et de couleur **purple4** (il n'y a pas de mur en bas) ;
- 4 rangées de 10 briques de taille 38 x 18 px, de couleur **magenta**, de résistance aléatoire (soit 1, soit 2), espacées entre elles de 2 px, la première brique se trouvant au point (50, 50) ;

b) il initialise la liste **messages** avec les différents états du jeu (voir ci-dessus) ;

c) il met la balle **ball**, de rayon 4px, au milieu de l'écran à 50 px du bord inférieur ;

d) il initialise le **score** et l'état **status** à zéro ;

2) **do_step** effectue une étape élémentaire du jeu : (6 p.)

   a) un déplacement de la balle. Si la balle a touché une brique(donc ni mur, ni raquette), le score est incrémenté d'une unité et la résistance de la brique est réduite d'une unité. Si, après, cette résistance est nulle, alors la brique en question est détruite ;

   b) un déplacement de la raquette, les limites étant les faces intérieures des murs gauche et droite ;

   c) la mise à jour de l'état du jeu :

   - le jeu est perdu si le centre de la balle descend en dessous du niveau supérieur de la raquette ;
   - le jeu est gagné si toutes les briques ont été détruites ;

3) **draw** dessine tous les éléments du jeu sur la toile. (1 p.)

IV) Le programme principal: (1 + 2 + 6 = 9 p.)

1) Effectuez les initialisations nécessaires pour créer un environnement **pygame** aux caractéristiques suivantes : (1 p.)

   a. la fenêtre **pygame** est carrée de taille **SIZE** et est rafraîchie 100 fois par seconde sur un fond de couleur " **dimgray**" ;

   b. l'entête de la fenêtre est à garder à jour pendant toute la durée du jeu selon les indications fournies ci-dessus ;

   c. créez une instance **game** de la classe **Game** et un témoin de mode automatique **auto_mode**, initialisé sur **False**.

2) Tant que l'application n'est pas quittée, la boucle principale réagit correctement aux commandes de l'utilisateur et effectue les calculs et mises-à-jour nécessaires.
Le jeu n'est mis à jour que dans l'état 1. L'entête de la fenêtre est toujours mis à jour.
A la fin du programme, l'environnement **pygame** et l'application sont clôturés et quittés correctement. (2 p.).

3) Les commandes de clavier gérées sont les suivantes : (6 p.)

   a. la touche **K_SPACE** démarre le jeu. Pendant le jeu elle fait basculer le jeu entre les états 1 (*running*) et 2 (*paused*) ;

   b. la touche **K_LEFT** fixe la vitesse de la raquette sur -4 ;

   c. la touche **K_RIGHT** fixe la vitesse de la raquette sur 4 ;

   d. la touche **K_n** crée un nouveau jeu ;

   e. la touche **K_a** active/désactive le mode automatique et cela indépendamment de l'état du jeu.
   En mode automatique, pendant le jeu, la raquette est déplacée horizontalement de façon à rester centrée en-dessous de la balle pour autant que possible, les limites de déplacement restant toujours les faces intérieures des murs gauche et droite ;

   f. tout relâchement d'une touche du clavier met la vitesse de la raquette à zéro.