Casse-brique 1ère partie

Objectifs :

- Utiliser tkinter pour réaliser une application comportant une interface graphique ;
- Voir comment utiliser Chatgpt (ou tout autre AI) pour progresser dans son apprentissage de la programmation ;
- Utiliser les structures vues dans les premiers Tps sur le langage Python.
- Prendre conscience de ce que vous avez appris depuis le début de l'année !

Table des matières

1. Création de la zone de jeu : casse-brique_0001.py....2
2. dessiner la balle : casse-brique_0002.py....3
   a. Prompt chatgpt pour comprendre la méthode zone_de_jeu.create_oval():....3
   b. Réponse de chatgpt :....3
      ✅ Ligne 1 :....3
      ✅ Ligne 2 (commentaire) :....4
      ✅ Ligne 3 :....4
      🧠 Visualisation (simplifiée)....4
      🧩 En résumé....4
3. Faire bouger la balle : casse-brique_0003.py....5
   a. Prompt....5
   b. Réponse....6
      Ligne 1 :....6
      Début de la fonction :....6
      Ligne suivante :....6
      Déplacement de la balle :....6
      Animation continue :....7
      Résumé global....7
4. C'est le moment de faire le point ….....7
5. Questions et exercices....7
   a. Exercice 1 : casse-brique_0004.py....7
   b. Exercice 2 : casse-brique_0004.py....8
   c. Exercice 3 : casse-brique_0004.py....8
   d. Exercice 4 : casse-brique_0005.py....8
   e. Question....8

1. Création de la zone de jeu : casse-brique_0001.py



casse-brique_0001.py

```
# on importe tout le contenu du module tkinter
# pour raccourcir la syntaxe as tk permet de l'appeler tk dans la suite
import tkinter as tk

# Fenêtre principale (root)
# Appel de la fonction Tk() du module tk pour créer une fenêtre nommée root
root = tk.Tk()
#appel de la méthode title de l'objet root pour définir le titre de la fenêtre
root.title("Mini casse-brique (3 murs)")

# Canvas de jeu (Zone de dessin graphique = constituée de pixels)
LARGEUR = 400 # définition d'une constante
HAUTEUR = 300 # définition d'une constante
zone_de_jeu = tk.Canvas(root, width=LARGEUR, height=HAUTEUR, bg='black')
zone_de_jeu.pack() # méthode pack pour placer la zone de jeu dans root

# boucle principale
root.mainloop()
```

casse-brique_0001.py

Souvenez-vous bien de la syntaxe :

<nom_du_module_ou_alias>.<fonction> pour appeler une fonction d'un module

exemples : `root = tk.Tk()` , `zone_de_jeu = tk.Canvas(...)`

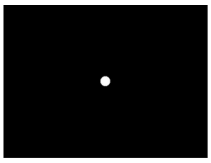
<objet>.<méthode> pour appeler une méthode associée à un objet

exemples : `root.title("…")` , `zone_de_jeu.pack()` , `root.mainloop()`

Avez vous l'impression d'avoir parfaitement compris le programme casse-brique_0001.py ?

- Si votre réponse est oui ; commencez par ajouter une fiche dans votre memento_Python puis passez à la section suivante. Prenez le temps de la réflexion. C'est aussi à cela que sert la rédaction de fiche dans Memento_python.
  Pour valider votre compréhension, n'hésitez pas à tester quelques modifications du programme sans le rendre plus complexe , juste pour valider la compréhension de cette partie.
- Si la réponse est non, n'hésitez pas à poser quelques questions complémentaires à chatgpt et/ou à votre enseignant

2. dessiner la balle : casse-brique_0002.py



casse-brique_0002.py

```
import tkinter as tk

# Fenêtre principale
root = tk.Tk()
root.title("Mini casse-brique (3 murs)")

# Canvas de jeu
LARGEUR, HAUTEUR = 400, 300 # une façon de fixer deux constantes en une ligne
zone_de_jeu = tk.Canvas(root, width=LARGEUR, height=HAUTEUR, bg='black')
zone_de_jeu.pack()

ball_radius = 10
# Création d'un objet ball dans la zone de jeu
# x_gauche , y_haut, x_droit, y_bas
ball = zone_de_jeu.create_oval(190, 140, 210, 160, fill='white')

root.mainloop()
```

casse-brique_0002.py

a. Prompt chatgpt pour comprendre la méthode zone_de_jeu.create_oval():

Mon programme en Python utilise tkinter (import tkinter as tk). Peux-tu m'expliquer ces lignes de code en Python?
ball_radius = 10
 # Création d'un objet ball dans le canvas
 # x_gauche , y_haut, x_droit, y_bas
ball = zone_de_jeu.create_oval(190, 140, 210, 160, fill='white')

b. Réponse de chatgpt :

Bien sûr ! Analysons ligne par ligne ce code Python utilisant `tkinter`, qui est une bibliothèque pour créer des interfaces graphiques (GUI).

✅Ligne 1 :

`ball_radius = 10`

- Tu définis une **variable** appelée `ball_radius` qui représente le **rayon de la balle** .
- Elle est ici fixée à **10 pixels**.

✅ *Ligne 2 (commentaire) :*

```
# Création d'un objet ball dans la zone de jeu
# x_gauche , y_haut, x_droit, y_bas
```

- Commentaire pour expliquer que la ligne suivante crée une balle dans le `zone de jeu`.
- Il précise aussi la signification des 4 coordonnées qui vont être utilisées :
  - **x_gauche** , **y_haut** : coordonnée du coin supérieur gauche de l'ovale.
  - **x_droit, y_bas** : coordonnée du coin inférieur droit.

✅ *Ligne 3 :*

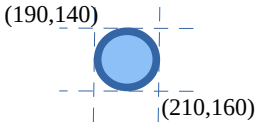
```
ball = zone_de_jeu.create_oval(190, 140, 210, 160, fill='white')
```

- `zone_de_jeu` est un objet de la classe `tk .Canvas ;` `zone_de_jeu = tk.Canvas(..)`
- `create_oval(...)` est une méthode de la classe `tk.Canvas` qui **dessine un ovale** (ou cercle) à l'intérieur d'un **rectangle délimité par deux points**
  - `(190, 140)` est le **coin supérieur gauche** du rectangle.
  - `(210, 160)` est le **coin inférieur droit**.
- Ces deux points forment un **rectangle de 20x20 pixels**, donc un **cercle de rayon 10 pixels** :
  - Largeur = 210 - 190 = **20 px**
  - Hauteur = 160 - 140 = **20 px**
- L'ovale (ou cercle ici, car le rectangle est carré) est **rempli en blanc** grâce à `fill='white'`.
- **`ball`** est l'**identifiant** de l'objet ovale créé dans la zone de jeu. Tu peux l'utiliser plus tard pour le déplacer, le supprimer, changer sa couleur, etc. syntaxe : *ball.<méthode(...)>*

🧠 *Visualisation (simplifiée)*

Imagine que tu as un canevas, et tu dessines un cercle blanc centré autour du point (200, 150), avec un rayon de 10 :

```
coin haut gauche : (190, 140)
coin bas droit   : (210, 160)
donc centre      : (200, 150)
```



🧩 *En résumé*

Élément	Signification
`ball_radius = 10`	rayon du cercle
`zone_de_jeu.create_oval(...)`	Appel de la méthode create_oval pour dessiner un ovale ou cercle dans l'objet zone de jeu
`(190, 140, 210, 160)`	rectangle délimitant le cercle
`fill='white'`	couleur de remplissage du cercle
`ball = ...`	identifiant de l'objet pour le manipuler ensuite

Avez vous l'impression d'avoir parfaitement compris le programme casse-brique_0002.py ?

- Si votre réponse est oui ; ajouter au moins une fiche dans votre memento_Python puis passez à la section suivante. Pour valider votre compréhension, n'hésitez pas à tester quelques modifications du programme pour valider la compréhension de cette partie.
- Si la réponse est non, n'hésitez pas à poser quelques questions complémentaires à chatgpt et/ou à votre enseignant

3. Faire bouger la balle : casse-brique_0003.py

```
import tkinter as tk

# Fenêtre principale
root = tk.Tk()
root.title("Mini casse-brique (3 murs)")

# zone de jeu de type Canvas pour une gestion pixel par pixel de la zone
LARGEUR, HAUTEUR = 400, 300
zone_de_jeu = tk.Canvas(root, width=LARGEUR, height=HAUTEUR, bg='black')
zone_de_jeu.pack()


ball_radius = 10
# Création d'un objet ball dans la zone de jeu
# x_gauche , y_haut, x_droit, y_bas
ball = zone_de_jeu.create_oval(190, 140, 210, 160, fill='white')

# Vitesse initiale
dx, dy = 3, 3

def move_ball():
    # on récupère deux variables dx et dy du programme principale
    global dx, dy

    # Déplacement d'un objet dans la zone de jeu
    zone_de_jeu.move(ball, dx, dy)

    root.after(20, move_ball)

# Lancer le mouvement
move_ball()
root.mainloop()
```

casse-brique_0003.py

a. Prompt

mon programme contient les lignes suivantes :

```
# Vitesse initiale
dx, dy = 3, 3

def move_ball():
    # on récupère deux variables dx et dy du programme principale
    global dx, dy

    # Déplacement d'un objet dans la zone de jeu
    zone_de_jeu.move(ball, dx, dy)
```

```
    root.after(20, move_ball)
Peux-tu m’expliquer la signification de ces lignes ?
```

b. Réponse

Bien sûr ! Tu es en train de programmer un petit jeu ou une animation avec **Tkinter**, et ce bloc de code permet de **faire bouger une balle** sur la zone de jeu.
Analysons ligne par ligne :

Ligne 1 :

```
# Vitesse initiale
dx, dy = 3, 3
```

- Tu définis la **vitesse initiale** de déplacement de la balle.
- `dx = 3` : déplacement de **3 pixels vers la droite** à chaque "frame".
- `dy = 3` : déplacement de **3 pixels vers le bas** à chaque "frame".
- Autrement dit, à chaque appel de `move_ball()`, la balle se déplace de `(3, 3)` pixels.

Début de la fonction :

```
def move_ball():
```

- Tu déclares une fonction appelée **`move_ball`**.
- Elle sera appelée **récursivement** (de manière répétée) pour créer l’animation.

Ligne suivante :

```
    global dx, dy
```

- Cette ligne permet à la fonction `move_ball()` d’utiliser (et modifier) les **variables globales `dx` et `dy`** définies hors de la fonction.
- Sans `global`, tu ne pourrais pas les modifier dans la fonction (Python créerait alors de nouvelles variables locales `dx`, `dy`).

Déplacement de la balle :

```
    zone_de_jeu.move(ball, dx, dy)
```

- Cette ligne déplace l’objet `ball` sur la zone de jeu.
- `dx` : nombre de pixels à déplacer **horizontalement**.
- `dy` : nombre de pixels à déplacer **verticalement**.
- La balle se déplace donc de 3 px vers la droite et 3 px vers le bas à chaque appel de la fonction.
- Si tu changes le signe de `dx` ou `dy`, la balle changera de direction.

Animation continue :

```
root.after(20, move_ball)
```

- `root.after(ms, func)` est une méthode **Tkinter** qui dit :

  "Dans **20 millisecondes**, exécute la fonction `move_ball` à nouveau."
- Cela crée une **boucle d'animation** toutes les 20 ms (~50 images par seconde).
- C'est l'équivalent d'un **timer** ou d'une **boucle de jeu**.
- Cela permet à la balle de **bouger en continu**, en appelant `zone_de_jeu.move(...)` encore et encore.

Résumé global

À chaque 20 millisecondes :

1. La balle est déplacée de `(dx, dy)` pixels.
2. La fonction se **rappelle elle-même**, créant un mouvement fluide.
3. Tu peux modifier `dx` et `dy` à tout moment pour :
   - changer la vitesse
   - changer la direction
   - gérer les rebonds contre les bords de la zone_de_jeu

4. C'est le moment de faire le point …

Beaucoup de notions sont utilisées dans ces quelques lignes de code. Avez vous l'impression d'avoir parfaitement compris le programme casse-brique_0003.py ?

- Si votre réponse est oui ; commencez par ajouter quelques fiches dans votre memento_Python puis passez aux questions de la section suivante.
- Si la réponse est non, n'hésitez pas à poser quelques questions complémentaires à chatgpt et/ou à votre enseignant ?

5. Questions et exercices

Si vous êtes arrivé ici…. vous n'avez normalement plus besoin de votre enseignant (ni de chatgpt bien sur) pour répondre aux questions et pour faire les exercices ci-dessous. Si ce n'est pas le cas, c'est le signe que vous n'avez pas poussé assez loin le souci de compréhension du code qui précède cette section.

Vous avez le droit par contre d'utiliser Memento_python.

a. Exercice 1 : casse-brique_0004.py

Pensez-vous pouvoir réduire le rayon de la balle à 8 pixels au lieu de 10 ?
Modifiez le programme dans ce sens et vérifiez bien que la modification fonctionne avant de faire valider votre code par votre enseignant.

Souvenez-vous qu'un bon code est un code « propre » , c'est à dire concis, lisible et facilement modifiable.

b. Exercice 2 : casse-brique_0004.py

Pouvez-vous ralentir le fonctionnement du programme pour visualiser plus facilement les déplacements de la balle ?

c. Exercice 3 : casse-brique_0004.py

Pouvez-vous modifier la vitesse de départ de la balle pour qu'elle se déplace vers le haut et vers la droite de la fenêtre ?

d. Exercice 4 : casse-brique_0005.py

Dans les programmes il est conseillé d'utiliser des termes anglais car les caractères accentués sont généralement mal acceptés par les compilateur. Ce n'est pas le cas en python. On en profitera pour donner des noms français aux objets et variables que nous créons ; cela les distinguera des mots clé du langage.
Renommer donc les variables, objets et fonctions *balle_rayon, balle, zone_de_jeu, déplacer_balle*

e. Question

Pouvez-vous expliquer pourquoi en l'absence de while True : le programme ne s'arrête pas ?