Casse-brique 1ère partie - correction

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

a. Exercice 1 : casse-brique_0004.py....1
Correction :....1
b. Exercice 2 : casse-brique_0004.py....2
Correction :....2
c. Exercice 3 : casse-brique_0004.py....2
Correction :....2
d. Exercice 4 : casse-brique_0005.py....2
e. Question : que fait exactement l'instruction root.mainloop() ?....2
Réponse courte :....2
Réponse plus longue de chatgpt :....3

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.

Correction :

Il y a un piège… si le programme était bien écrit, il suffirait de changer la valeur de ball_radius pour la fixer à 8 et le tour serait joué. Mais le programme n'est pas bien écrit ; la valeur de ball_radius n'est pas prise en compte lors de la création de l'objet ball. En résumé, ball_radius a été défini mais ne sert à rien, sauf à induire en erreur le programmeur qui souhaite modifier le programme initial. Ci-dessous une programmation plus propre

```
# Création de la balle
BALL_RADIUS = 8

# positionner ball dans un rectangle englobant (x1,y1,x2,y2)
# x1=gauche=190 y1=haut=140 x2=droit=210 y2=bas=160 couleur = blanc
ball = zone_de_jeu.create_oval(200 - BALL_RADIUS//2, 150 - BALL_RADIUS//2, 200
+ BALL_RADIUS//2 , 150 + BALL_RADIUS//2, fill='white')
```

b. Exercice 2 : casse-brique_0004.py

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

Correction :

Il y a plusieurs solutions. La plus simple est d'appeler moins souvent la fonction move_ball() ; par exemple toutes les 40 millisecondes au lieu des 20 millisecondes. root.after(40, move_ball)
La seconde consiste à modifier dx et dy pour avoir un vecteur vitesse de plus faible module ; par exemple dx = 2 et dy = 2.

On rappelle que la valeur du module du vecteur vitesse est donnée par $||vitesse|| = \sqrt{dx^2 + dy^2}$

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 ?

Correction :

Pour que la balle se déplace vers le haut de l'écran il faut que dy soit négatif. Pour que la balle se déplace vers la droite il faut que dx soit positif. Le couple dx = 3 et dy = -3 constitue une solution possible.

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 compilateurs. 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 : que fait exactement l'instruction root.mainloop() ?

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

Dans un programme en python, en supposant que l'on a créé une fenêtre principale root = tkinter.Tk() ,que fait exactement l'instruction root.mainloop() ?

Réponse courte :

Dans un programme classique l'instruction while True : précède le bloc d'instructions répété en permanence pendant le programme ; c'est ce qu'on appelle la boucle principale (main loop).

Ici cette boucle principale est contenue dans l'instruction root.mainloop(). mainloop() scrute en boucle tous les évènements possibles associés à la fenêtre root et appelle la fonction associée à un évènement lorsque l'un d'eux se produit.

Réponse plus longue de chatgpt :

Excellente question 👋 — c'est **une des plus importantes** pour bien comprendre le fonctionnement d'un programme graphique avec Tkinter (et plus généralement de toute interface graphique).
Voyons cela en détail 👇

🧱 1. Rappel du contexte

Quand tu écris :

```
import tkinter as tk

root = tk.Tk()       # Création de la fenêtre principale
root.mainloop()      # Boucle principale de l'application
```

le programme Python **crée d'abord la fenêtre** (`root = Tk()`), puis **lance sa boucle d'exécution graphique** avec `root.mainloop()`.

⚙️ 2. Que fait `mainloop()` exactement ?

f. ➤ C'est la boucle d'événements (event loop)

Elle **attend et gère en continu les événements** qui se produisent dans la fenêtre :

- mouvement ou clic de souris 🖱️
- pression d'une touche du clavier ⌨️
- redimensionnement ou fermeture de la fenêtre 🪟
- mise à jour des widgets (boutons, labels, entrées, etc.)

Tkinter doit écouter ces événements **tant que la fenêtre est ouverte** — c'est exactement le rôle de `root.mainloop()`.

🧭 3. Que se passe-t-il avant et après `mainloop()`

g. Avant :

Tout ce que tu écris avant `root.mainloop()` sert à **préparer** l'interface :

- création de widgets (boutons, labels, frames…)
- configuration (couleurs, taille, polices…)
- positionnement (via `.pack()`, `.grid()`, `.place()`)

Pendant :

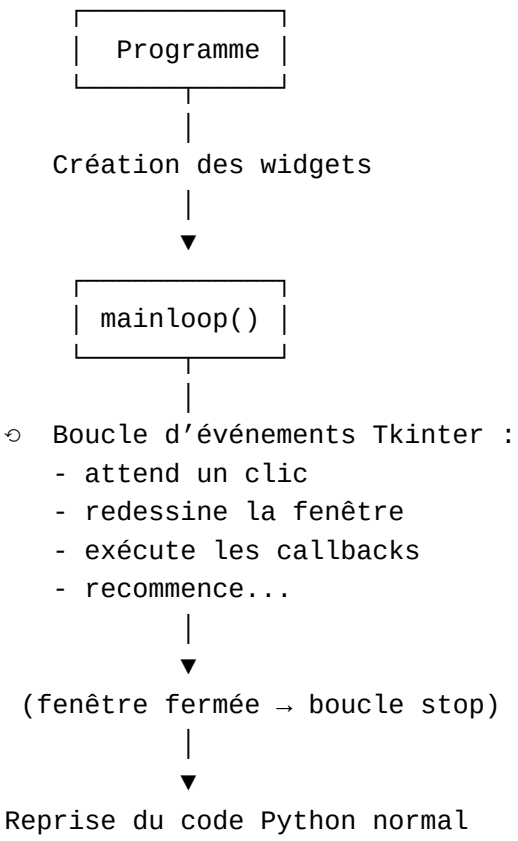
Une fois que tu appelles `root.mainloop()`,
le **contrôle du programme passe à Tkinter** :
➡️ Ton script ne continue plus à s'exécuter "normalement".
➡️ Tkinter tourne dans une boucle interne et attend les actions de l'utilisateur.
➡️ Tes fonctions (callback) sont exécutées quand un événement se produit (clic, saisie, etc.).

h. Après :

Quand la fenêtre est fermée (ex. via le bouton “Fermer”), la boucle `mainloop()` se termine, et ton script reprend son exécution **à la ligne suivante** (s’il y en a une). Exemple :

```
root.mainloop()
print("Fenêtre fermée !")   # s’exécute seulement après la fermeture
```

🔄 4. En résumé visuel



🧠 5. Détail technique (pour les curieux)

`root.mainloop()` appelle en réalité une méthode de la bibliothèque Tcl/Tk (le moteur sous-jacent à Tkinter).

C’est une **boucle infinie** gérée en C, qui ne sort que lorsque :

- la fenêtre principale (`root`) est détruite (`root.destroy()` ou fermeture manuelle),
- ou qu’une erreur stoppe le programme.

💡 6. Points importants à retenir

Ce que `mainloop()` fait	Ce qu’il ne fait pas
Gère les événements utilisateur	Créer ou modifier les widgets (tu le fais avant)
Rafraîchit automatiquement la fenêtre	Bloquer le programme (tu peux utiliser `after()` pour exécuter du code périodique)
Boucle jusqu’à la fermeture	Ne relance pas ton code après fermeture

---

i. Exemple illustratif :

```
import tkinter as tk

def dire_bonjour():
    print("Bonjour !")

root = tk.Tk()
root.title("Exemple mainloop")

bouton = tk.Button(root, text="Clique-moi", command=dire_bonjour)
bouton.pack(padx=20, pady=20)

root.mainloop()
print("La fenêtre a été fermée.")
```

Comportement :

- Tant que la fenêtre est ouverte, le programme “vit” dans la boucle Tkinter.
- Quand tu cliques, Tkinter exécute `dire_bonjour()`.
- Quand tu fermes la fenêtre, `print("La fenêtre a été fermée.")` s’exécute.

Souhaites-tu que je te montre **comment faire tourner une action périodique** (par exemple, une mise à jour toutes les 5 secondes) **sans bloquer le mainloop()** ?
✅ La bonne méthode : `root.after()`
Tkinter fournit une méthode spéciale : `root.after(delay_ms, fonction)`
➡️ Elle **planifie l’exécution différée** d’une fonction après un certain délai (en millisecondes).
➡️ Et surtout : **elle ne bloque pas le mainloop()**.

---

2. 💡 Exemple simple : compteur automatique

```
import tkinter as tk

compteur = 0

def mise_a_jour():
    global compteur
    compteur += 1
    label.config(text=f"Compteur : {compteur}")
    # planifie le prochain appel dans 1000 ms (1 seconde)
    root.after(1000, mise_a_jour)

root = tk.Tk()
root.title("Compteur avec after()")

label = tk.Label(root, text="Compteur : 0", font=("Arial", 18))
label.pack(padx=20, pady=20)

# Démarre la boucle d’actualisation
root.after(1000, mise_a_jour)

root.mainloop()
```

🟢 **Ce qu'il se passe :**

- `root.after(1000, mise_a_jour)` appelle `mise_a_jour()` après 1 seconde.
- `mise_a_jour()` met à jour le label **et se reprogramme elle-même**.
  ➡️ C'est une boucle "douce", intégrée à `mainloop()`.

---

3. ⚙️ Variante : arrêter la boucle

```
import tkinter as tk

running = True
compteur = 0

def mise_a_jour():
    global compteur
    if running:
        compteur += 1
        label.config(text=f"Compteur : {compteur}")
        root.after(1000, mise_a_jour)

def stop():
    global running
    running = False
    label.config(text="Arrêté.")

root = tk.Tk()
root.title("Boucle non bloquante")

label = tk.Label(root, text="Compteur : 0", font=("Arial", 18))
label.pack(pady=10)

btn = tk.Button(root, text="Arrêter", command=stop)
btn.pack(pady=10)

root.after(1000, mise_a_jour)
root.mainloop()
```

➡️ Ici, la fonction `stop()` met fin à la boucle périodique proprement, sans bloquer la fenêtre.
