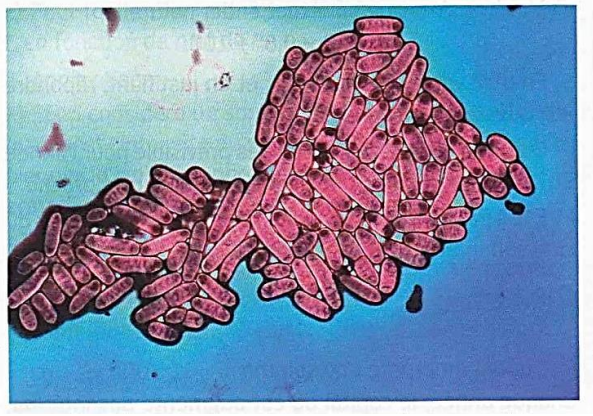
OBJECTIFS : L'objectif de ce TP est de réaliser des codes pythons utilisant la structure *while* vue en cours.

DOCUMENT A RENDRE : Ce travail est évalué. Vous en rédigerez un compte-rendu sous le nom « *tp7_nomfamille.doc* » ou « *tp7_nomfamille.odt* » et vous le transférerez en fin d'activité **par l'intermédiaire de l'onglet transfert** du site en utilisant le code : **tp7** . Ce compte-rendu contiendra :

- les réponses aux différentes questions posées,
- les captures d'écran **des morceaux de codes** écrits **et** celles **des résultats des exécutions** données dans le shell. Pour faire ces captures, utiliser *l'Outil Capture d'écran* de Windows.

1. Boucle *while* pour répondre à une question de biologie :



On a étudié le nombre de bactéries du type coliforme dans 1 litre de culture liquide.

Au départ, il y avait $n = 50\ 000$ bactéries. Après ajout d'un antibiotique, on constate que toutes les minutes, 10 % de ces bactéries meurent et que dans le même temps 100 nouvelles bactéries naissent.

Questions :

1- Le nombre de bactéries au bout d'1 mn sera : $n = 50000 \times 0.9 + 100 = 45100$
Calculer à la main le nombre n de bactéries au bout de 3 mn.
$n = 45100 \times 0.9 + 100 = 41059$ au bout de 2 mn
$n = 41059 \times 0.9 + 100 = 37053{,}1 \approx 37053$ au bout de 2 mn

2- Ecrire un code python utilisant une boucle *while* et qui permet de calculer le nombre de minutes nécessaires pour que n devienne inférieur à 1500.

```
1 n = 50000
2 mn = 0
3 while n > 1500 :
4     n = 0.9*n + 100
5     mn = mn + 1
6 print(f"{n:.0f} bactéries au bout de {mn} mn")
```

```
>>> (executing file "exercice_1.py")
1475 bactéries au bout de 44 mn
```

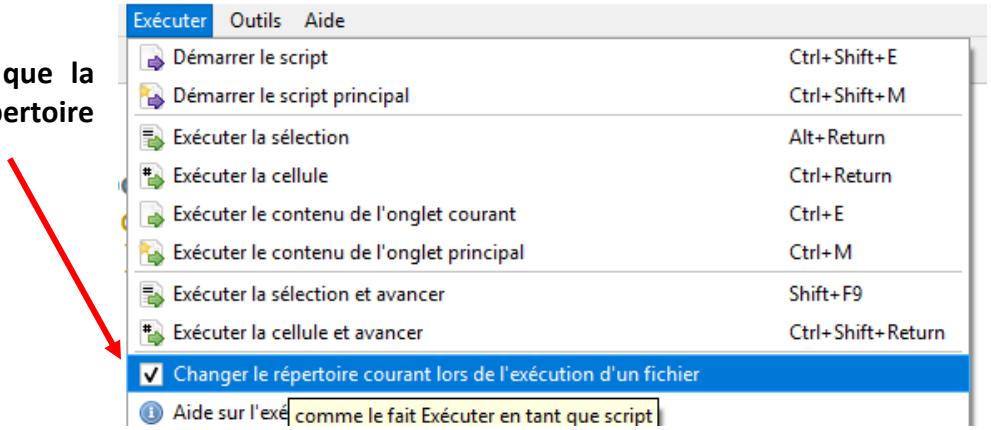
3- Utiliser ce code pour calculer le temps nécessaire pour que n devienne inférieur à 500. …….. En l'exécutant, on constate que la boucle *while* tourne à l'infini. Pourquoi le code fonctionne avec 1500 et pas avec 500 ? Justifier en affichant avec des *print()* le contenu de la variable n au cours de l'exécution.

La condition n > 500 est toujours vraie car le nombre de bactéries ne baisse pas sous le seuil de 1000. Ainsi la boucle *while* boucle à l'infini.

2. Boucle *while* pour réaliser une simulation de shoot au basket :

⇨ Télécharger les 2 images *.gif* et les fichiers *bibli_basket.py* et *exercice_2.py* se trouvant dans le zip associé au TP7, à télécharger à partir du site. Après les avoir dézippés, copier ces fichiers **dans votre répertoire** de travail.

⇨ **Dans Pyzo, s'assurer que la case « Changer le répertoire courant …. » est cochée** :

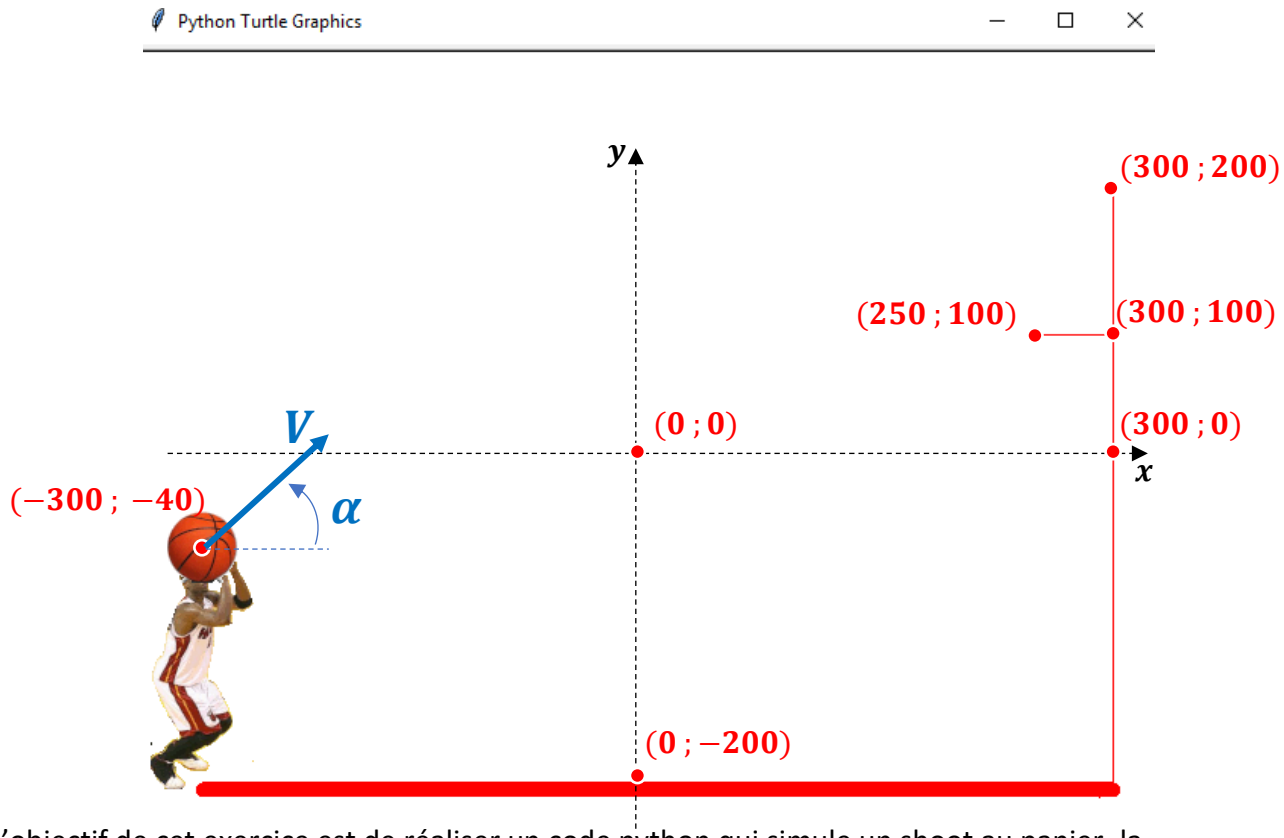


⇨ Ouvrir et exécuter le fichier *exercice_2.py* .

Ce fichier permet l'importation des bibliothèques *turtle* pour pouvoir faire du graphisme, *math* pour pouvoir utiliser la fonction *sinus* et *bibli_basket* qui permet, à partir de la fonction *decor()* de créer la fenêtre ci-dessous et de renvoyer un objet *Turtle* nommé ballon et les cordonnées $(xb\ ,\ yb)$ du centre du ballon au début du shoot. Vous pouvez jeter un coup d'œil dans ce fichier *bibli_basket.py* .

```
1  # importation des bibliothèques
2  from turtle import *
3  from math import *
4  from bibli_basket import decor
5
6  # construction du décor par l'appel de la fonction decor()
7  ballon,xb,yb = decor()
8
9
10                    ⇦ Zone que vous allez compléter dans cet exercice
11
12
13 # à laisser
14 exitonclick()
15 mainloop()
```

Les coordonnées des différents points ci-dessous sont en pixels. On supposera qu'elles sont en cm.



L'objectif de cet exercice est de réaliser un code python qui simule un shoot au panier, la vitesse v et l'angle α du tir étant définis par l'utilisateur.

Le code devra réaliser les opérations suivantes :

Etape 1 :

⇨ Ecrire les lignes suivantes qui demandent une valeur de la vitesse v en cm/s et une valeur de l'angle α en degrés. Pour que la saisie se fasse dans la fenêtre graphique, on utilise la fonction *numinput()* de python qui permet de dialoguer à partir d'une fenêtre graphique *(équivalent de input() qui permet de dialoguer à partir du shell).*

```
v = int(numinput("", "vitesse ?",minval = 0, maxval = 500))
angle = int(numinput("", "angle ?",minval = 0, maxval = 90))
```

La valeur saisie de l'angle étant en degré, il faudra la convertir en radians par la ligne :

```
angle = angle * pi / 180
```

⇨ Insérer un *print(v , angle)* dans le code afin de vérifier que les valeurs saisies sont bien les bonnes.

Etape 2 :

Comme dit précédemment, la ligne `ballon,xb,yb = decor()` permet d'exécuter la fonction *decor()* qui renvoie un objet *Turtle* nommé ballon et les cordonnées $(xb\ ,\ yb)$ du centre du ballon **au début du shoot**.

L'objectif ici est de déplacer cet objet *Turtle* nommé ballon. Pour cela, on utilisera la méthode *Turtle goto()* appliquée à ce ballon.

⇨ Tester ce déplacement de ballon en insérant dans le code par exemple : `ballon.goto(10,30)`

On utilise à présent la méthode `ballon.goto(x,y)` que l'on répètera pour déplacer le ballon de pixel en pixel en modifiant à chaque fois les coordonnées (x, y) du déplacement.

⇨ déplacer le ballon en répétant `ballon.goto(x,y)` dans une boucle *while* , les valeurs des coordonnées (x, y) du centre du ballon étant égales à :

$$x = \mathrm{v}\cos(\alpha)\ t + xb \qquad \text{et} \qquad y = -\frac{9.81}{2}\ t^2 + v\sin(\alpha)\ t + yb$$

Sur le code cela donnera :

```
x = v*cos(angle)*t + xb
y = -9.81/2*t**2 + v*sin(angle)*t + yb
```

t est le temps en secondes avec $t = 0$ au début du lancer, $(xb\ ,\ yb)$ sont les coordonnées du centre du ballon au début du shoot (*ces relations sont obtenues en appliquant les lois de la physique sur les corps en mouvement*).
A chaque tour de boucle, le temps sera augmenté de 1 s : `t = t + 1`
Le **bouclage s'arrêtera** lorsque le ballon sort de la fenêtre, c'est-à-dire dès que $x > 300$ ou $y < -200$.
Pour visualiser l'emplacement du point on pourra rajouter un cercle de diamètre 5px avec la commande : `ballon.dot(5)`

Etape 3 :

⇨ repérer si un panier est marqué, ce qui est fait si $y = 100$ et $250 < x < 300$. A ce moment écrire dans la fenêtre graphique le message GAGNE en écrivant :

```
write("GAGNE",font = ("Arial", 30, "bold"))
```

Comme l'ordonnée des points calculés ne sera JAMAIS exactement égale à 100 px , on peut calculer plus de points, par exemple toutes les 0,1 s (`t = t + 0.1`)

et/ou remplacer la condition $y = 100$ par $95 < y < 105$ …. En tout cas ces paramètres sont à optimiser en réessayant plusieurs fois.

```
# importation des bibliothèques
from turtle import *
from math import *
from bibli_basket import decor

# construction du décor par l'appel de la fonction decor()
ballon,xb,yb = decor()

# saisi vitesse et angle
v = int(numinput("", "vitesse ?",minval = 0, maxval = 500))
angle = int(numinput("", "angle ?",minval = 0, maxval = 90))
angle = angle * pi / 180
# lancer
x , y = xb , yb
t = 0
while x < 300 and y > -200 :
    x = v*cos(angle)*t + xb
    y = -9.81/2*t**2 + v*sin(angle)*t + yb
    ballon.goto(x,y)
    ballon.dot(5)
    t = t + 0.1
    if x > 250 and x < 300 and y > 95 and y < 105 :
        write("GAGNE",font = ("Arial", 30, "bold"))

# à laisser
exitonclick()
mainloop()
```

3. Utilisation du code de l'exercice 2 pour en faire un jeu :

Reprendre le code de l'exercice 2 pour y apporter les modifications suivantes :

- Placer une partie du code dans une autre boucle *while* afin de pouvoir shooter plusieurs fois de suite. On pourra utiliser les instructions suivantes :

```
rep = 1
while rep == 1 :
```

```
ballon.clear()
ballon.up()
ballon.goto(xb,yb)
ballon.down()
```

```
rep = int(numinput("", "On rejoue ? 1:oui, 0:non",minval = 0, maxval = 1))
```

```
write("GAGNE",font = ("Arial", 30, "bold"))
```

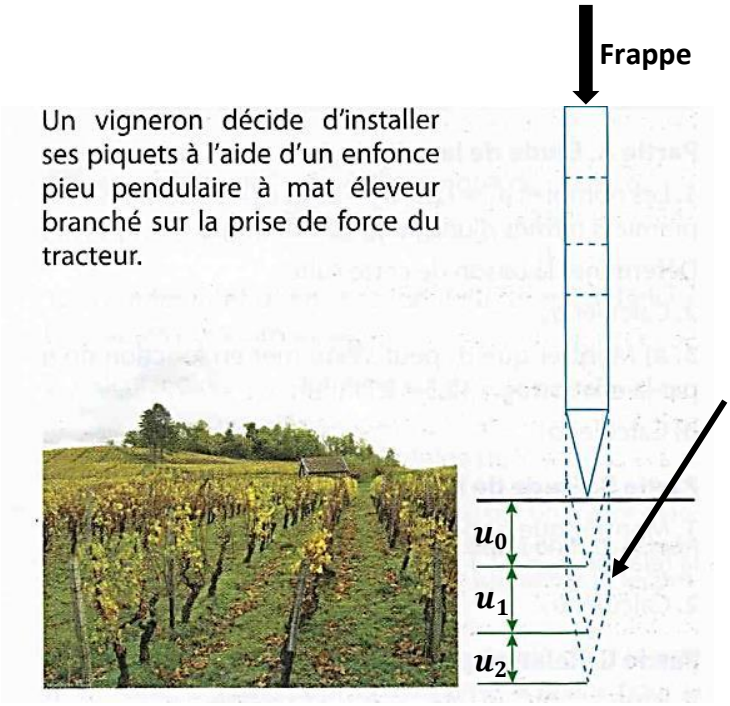
- Inclure dans le code un script qui comptabilise le nombre de paniers tirés.

```
from turtle import *
from math import *
from bibli_basket import decor
# construction du décor par l'appel de la fonction decor()
ballon,xb,yb = decor()
rep = 1
# positionnement texte
hideturtle()
up()
goto(-200,0)

# début jeu
nb_paniers = 0
while rep == 1 :
    # initialisation
    clear()
    ballon.clear()
    ballon.up()
    ballon.goto(xb,yb)
    ballon.down()
    ballon.dot()
    # saisi vitesse et angle
    v = int(numinput("", "vitesse ?",minval = 0, maxval = 500))
    angle = int(numinput("", "angle ?",minval = 0, maxval = 90))
    angle = angle * pi / 180
    # lancer
    x , y = xb , yb
    t = 0
    while x < 300 and y > -200 :
        x = v*cos(angle)*t + xb
        y = -9.81/2*t**2 + v*sin(angle)*t + yb
        ballon.goto(x,y)
        ballon.dot(5)
        t = t + 0.1
        if x > 250 and x < 300 and y > 95 and y < 105 :
            nb_paniers = nb_paniers + 1
            message = "SHOOT : "+str(nb_paniers)+" shoots réussis"
            write(message,font = ("Times", 20, "bold"))

    rep = int(numinput("", "Rejoue ? 1:oui, 0:non",minval = 0, maxval = 1))
clear()
write("FIN",font = ("Arial", 30, "bold"))
```

4. Boucle *while* pour calculer un enfoncement



Le pieu est introduit initialement dans un trou de 10 cm de profondeur. A chaque frappe, il s'enfonce un peu plus.

$u_0 = 10$,

$u_1 =$ **déplacement** vertical du pieu après la 1ère frappe,

$u_2 =$ **déplacement** vertical du pieu après la 2nde frappe (voir figure ci-contre),

…….

$u_n =$ déplacement vertical du pieu après la nième frappe.

Le sol étant de plus en plus dense, lors de l'enfoncement, entre 2 frappes successives, le déplacement vertical diminue à chaque fois de 15 %. **Ainsi u_1 est 15 % inférieur à u_0 , u_2 est 15 % inférieur à u_1, etc …. u_{n+1} est 15 % inférieur à u_n**

L'enfoncement total du pieu après n frappe est : $E_n = u_0 + u_1 + u_2 + \cdots .. + u_n$

⇨ Ecrire un code qui permette de calculer le nombre n de frappes pour que l'enfoncement total soit de 30 cm.

```
u = 10
enfonc = 10
n = 0
while enfonc < 65 :
    n = n + 1
    u = u * 0.85
    enfonc = enfonc + u

print(f"Enfoncement de {enfonc:.1f} obtenu après {n} coups")
```