Exercices sur la programmation objet

Exercice 1.

a) Écrire la définition d'une classe nommée `Carre` admettant la mesure des côtés d'un carré en attribut, avec deux méthodes :
- une méthode `perimetre` permettant de retourner le périmètre du carré ;
- une méthode `aire` permettant de retourner son aire.

b) Écrire la définition d'une classe nommée `Triangle` représentant un triangle et admettant la mesure des trois côtés comme attributs, avec deux méthodes :
- une méthode `aire` renvoyant l'aire du triangle à l'aide de la formule de Héron :

$A = \sqrt{p\,(p-a)(p-b)(p-c)}$,

où a, b et c sont les longueurs des trois côtés et où $p = (a + b + c) / 2$;
- une méthode `rect` qui renvoie True si le triangle est rectangle et False sinon.

c) Puis définir un carré de côté 5, afficher son périmètre et son aire.
d) Définir un triangle de côtés 5,4 ; 9 et 7,2 pour afficher son aire et regarder s'il est rectangle.

Exercice 2.

Définir une classe Point donnant la définition d'un point dans le plan.
L'affichage de point.__doc__ doit afficher " Définition d'un point dans le plan "
Cette classe admet deux arguments a et b qui sont les coordonnées du point, initialisées par défaut à 0.
Écrire les méthodes suivantes :
- la méthode distance qui renvoie la distance entre ce point et l'origine du repère (0, 0).
- la méthode repre qui renvoie une chaîne de caractère et permet un affichage correct dans l'interpréteur , comme (5.6 ; 8).
- la méthode egal avec en argument (autre que self) un point p, et qui renvoie True si les coordonnées d'un point p1 sont égaux aux coordonnées d'un point p2 avec le contrôle suivant :

    self.x == p.x and self.y == p.y
- la méthode distAB qui prend en argument (en plus de self) un point p et renvoie un nombre, la distance entre les deux points (self et p).

Exercice 3.

Écrire la définition d'une classe `Cercle_magique`. C'est une classe pour représenter des cercles magiques.
Elle prend en argument deux valeurs, r pour le rayon et c pour la couleur.
Elle possède trois méthodes :
- gonfler qui fait grandir le rayon d'une valeur dr
- aire qui retourne l'aire du disque de rayon r,
- pondre qui diminue le rayon r du cercle de 1 et retourne un nouveau cercle de rayon 1 et de la même couleur que le cercle de départ (r et c).

Après la définition de la classe,
- instancier deux nouveaux cercles, c_v (rayon = 3, couleur = vert) et c_r (rayon = 7, couleur = rouge).

Puis,
- accéder au rayon de c_v
- augmenter le rayon de c_v de 2
- vérifier que le rayon s'est modifié
- accéder à l'aire de c_r.

Enfin,
- créer un cercle c_x à partir de c_v,
- accéder à sa couleur,
- et à son aire.

Exercice 4.

La bibliothèque tkinter

Nous utilisons cette bibliothèque pour construire une GUI (graphical user interface), c'est-à-dire une fenêtre que nous complétons avec différents éléments, des widgets.
Plus précisément, tkinter (pour tool kit interface) est une interface Python d'une boîte à outils nommé Tk. Tkinter a été écrit par Steen Lunholt et Guido van Rossum. Tk est une bibliothèque d'interfaces graphiques multiplateforme conçue par John Ousterhout vers 1990.

```
import tkinter as tk

fen = tk.Tk()                          #la fenêtre principale
fen.title(" ma fenêtre ")
fen.geometry("300x70")

texte = tk.Label(fen, text = "bonjour ", fg= "red")     # fg = foreground
texte.pack()                           #une méthode pour placer texte

bouton = tk.Button(fen, text= "quitter ",  command = fen.destroy)
bouton.pack()

fen.mainloop()
```

Objet
Classe
Méthode
Paramètre

Tester ce code.

Description du code.

La **fenêtre**, le **texte** et le **bouton** sont des **objets**.
L'objet **fen** est une instance de la **classe** principale Tk().
Trois **méthodes** sont utilisées : **title** pour le titre, **geometry** pour la taille de la fenêtre et **mainloop** qui place la fenêtre en attente d'un événement.
Deux objets sont ajoutés à la fenêtre, une instance de la classe **Label** (une étiquette) et une instance de la classe **Button** (un bouton).
La création du bouton par exemple, utilise le constructeur de la classe **Button** qui prend plusieurs **paramètres**.
Trois **paramètres**, **fen**, **text** et **fg** sont donnés, les autres sont laissés à leur valeur par défaut.
Le premier paramètre représente la fenêtre dans laquelle se place le bouton.
Il est possible de construire d'autres fenêtres.
On applique ensuite la **méthode pack** dont le seul paramètre dans cet exemple est le bouton lui-même.

Pour bien comprendre le rôle de chaque instruction, il est possible de les écrire une à une dans l'interpréteur. On peut alors voir successivement la création de la fenêtre, l'ajout du titre, le redimensionnement, l'ajout de l'étiquette puis du bouton.

Les principaux widgets disponibles sont :
- Button : un bouton ;
- Canvas : un espace pour dessiner des formes ou des images ;
- Checkbutton: une case à cocher ;
- Entry : un champs où écrire du texte ;
- Frame : un cadre pour contenir d'autres widgets ;
- Label : un espace pour écrire du texte, décrit une Entry ;
- Listbox : une liste où l'utilisateur peut sélectionner un élément.

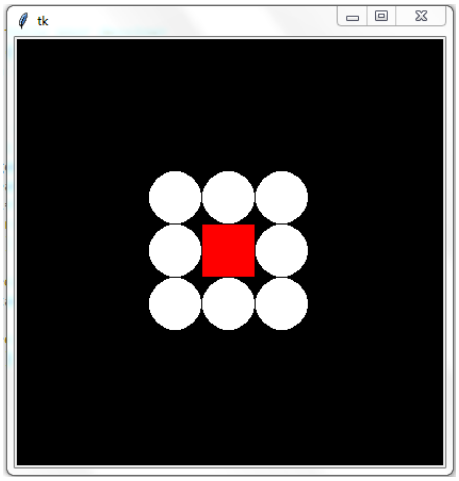
Le mémo Python apporte plus d'informations sur la bibliothèque tkinter.

Quelques dessins avec tkinter.

Dessin 1 :

Pour le premier palier recopier le code qui se trouve dans le mémo python au début de la partie '*Dessiner dans une fenêtre avec tkinter'* et exécuter le code.

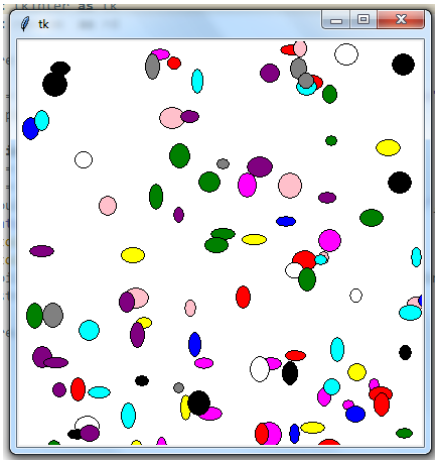
Dessin 2



Créer ce dessin dans une fenêtre 400x400 avec des cercles de diamètre 50 et un carré rouge le tout au centre de la fenêtre.

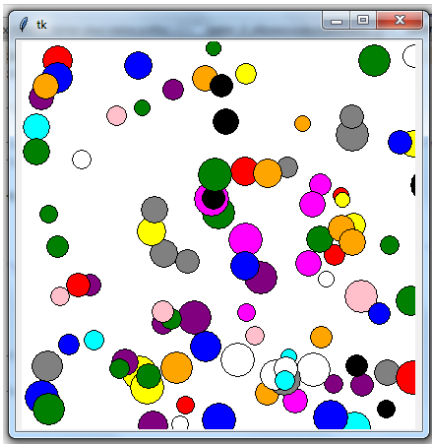
Dessin 3 :

3a - Créer un programme permettant d'afficher 100 ellipses au hasard à l'écran. Les deux axes de chaque ellipse seront fixés à 20 et 50. La couleur de chaque ellipse sera bleu.
3b – Créer le même programme 5a où les axes de chaque ellipse devront être aléatoires (mais compris entre 20 et 50). La couleur sera rouge.
3c - Créer le même programme 5b avec en plus pour chaque ellipse une couleur aléatoire.



100 ellipses avec des positions, des tailles et des couleurs aléatoires.

Dessin 4 :



Pour aller plus loin, refaire la même chose avec des disques.