

Séance 4 : GRAPHISMES, OBJETS ET VARIABLES GLOBALES

L1 – Université Côte d’Azur

Dans tout le TD, on suppose que l’on travaille dans une fenêtre tk créée à partir du code suivant.

```

1 import tkinter as tk
2 from math import *
3 root = tk.Tk()
4 root.title("TD 4")
5 Largeur=500
6 Hauteur=500
7 Dessin = tk.Canvas(root,height=Hauteur,width=Largeur)
8 Dessin.pack()
    
```

On rappelle les principales commandes pour tracer des figures

```

1 Dessin.create_line(point_1,point_2) # option fill='blue' pour la couleur
2 Dessin.create_oval(point_1,point_2)
3 Dessin.create_rectangle(point_1,point_2)
    
```

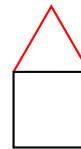
Exercice 1 – Variables globales et crayon (★)

On cherche à définir un crayon qui possède une position et une couleur. L’état du crayon sera définie par des variables globales.

1. Quelles sont les variables nécessaires pour définir l’état du crayon ?
2. Écrire une fonction **déplacer**(x, y) qui modifie les coordonnées du crayon sans rien tracer à l’écran.
3. Écrire une fonction **tracer**(x, y) qui trace un segment à partir de la position courante du crayon jusqu’au point de coordonnée (x, y). La position du crayon sera modifiée par cette fonction.
4. Écrire une fonction **changer_couleur**(c) qui modifie la couleur du crayon.

Exercice 2 – Maison (★)

En utilisant les trois fonctions précédentes, écrire une fonction **dessine_maison**(x, y, c) qui prend en argument un entier c et qui dessine une maison de côté c (tous les traits ont pour longueur c) et dont le coin en bas en gauche a pour coordonnée (x, y). La maison devra être noire avec un toit rouge. *Bonus : ne levez jamais le crayon, et ne repassez pas deux fois sur le même trait!*



Exercice 3 – Les cercles et les disques (★★)

Il existe en tk une fonction pour tracer un cercle `Dessin.create_oval(point_1,point_2)`, mais la question que tout le monde se pose est de savoir comment l’ordinateur procède¹. Bonne nouvelle, c’est l’objectif de l’exercice !

1. On se donne deux points p et q (rappel : un point est un couple (x, y) de coordonnées). Écrire une fonction **distance**(p, q) qui calcule la distance euclidienne entre p et q .

¹ si vous ne vous êtes jamais posé la question, vous manquez grandement de curiosité!

```

1 def distance(p,q):
2     (x1,y1)=p
3     (x2,y2)=q
4     return ((x1-x2)**2 + (y1-y2)**2)**.5
    
```

2. En mathématiques, donner une équation d’un cercle de centre (x_0, y_0) et de rayon r ? Même question pour un disque.

Notons o le centre du cercle et p le point de coordonnée (x, y) .

– Pour un cercle : $(x - x_0)^2 + (y - y_0)^2 = r$ ou dit autrement $distance(o, p)^2 = r$

– Pour un disque : $(x - x_0)^2 + (y - y_0)^2 \leq r$ ou dit autrement $distance(o, p)^2 = r$

On remarque que l’on utilise jamais directement la distance, mais surtout la distance au carré. Inutile donc de calculer une racine carré!

```

1 def dist2(p,q):
2     (x1,y1)=p
3     (x2,y2)=q
4     return (x1-x2)**2 + (y1-y2)**2
    
```

3. En parcourant tous les pixels de l’écrans, et en utilisant une fonction `affiche_pixel(x,y,couleur)` similaire à celle vue en cours, écrire une fonction `disque(x0,y0,r,couleur)`, qui affiche tous les pixels à l’intérieurs du disque de centre (x_0, y_0) et de rayon r .

```

1 def disque(x0,y0,r,couleur):
2     centre = (x0,y0)
3     for x in range(Largeur):
4         for y in range(Hauteur):
5             p=(x,y)
6             if dist2(centre,p)<=r**2:
7                 affiche_pixel(x,y,couleur)
    
```

4. Il n’est pas nécessaire de parcourir tous les pixels de l’écrans, mais simplement ceux d’un petit carré qui contient le cercle. Modifier votre code pour tenir compte de cette astuce.

```

1 def disque(x0,y0,r,couleur):
2     centre = (x0,y0)
3     for x in range(x0-r, x0+r+1):
4         for y in range(y0-r, y0+r+1):
5             p=(x,y)
6             if dist2(centre,p)<=r**2:
7                 affiche_pixel(x,y,couleur)
    
```

5. Écrire une fonction `cercle(x,y,r,couleur)` qui affiche le cercle de centre (x_0, y_0) et de rayon r .

```

1 def cercle(x0,y0,r,couleur):
2     centre = (x0,y0)
3     for x in range(x0-r, x0+r+1):
4         for y in range(y0-r, y0+r+1):
5             p=(x,y)
6             d2=dist2(centre,p)
7             if (r-0.5)**2 <= d2 and d2 <= (r+0.5)**2:
8                 affiche_pixel(x,y,couleur)
    
```

6. À votre avis cette fonction est elle plus ou moins efficace que la méthode vue en cours (tracer un cercle revient à tracer un polygone à 40 côtés.)

Cette méthode oblige à parcourir tout les points du carré contenant le cercle. Si c’est pratique pour afficher le disque, on parcourt beaucoup trop de pixels par rapport à ceux du cercle.

Exercice 4 – Des cercles qui ont la classe (*)

Créer une classe `Cercle` avec quatre attribus : les coordonnées, le rayon et la couleur et deux méthodes `afficher_disque`

et `afficher_cercle`. On fera appel aux deux fonctions `cercle` et `disque` définies précédemment.

```

1 class Cercle:
2     def __init__(self,x,y,r,c):
3         self.x = x
4         self.y = y
5         self.rayon = r
6         self.couleur = c
7
8
9     def afficher_cercle(self):
10        cercle(self.x,self.y,self.couleur)
11
12    def afficher_disque(self):
13        disque(self.x,self.y,self.couleur)

```

Exercice 5 – Couleur RGB (★)

En tk les couleurs peuvent être définies en hexadécimal, par exemple `'#FF12E4'` où FF, 12 et E4 correspondent aux trois composantes de rouge, de vert et de bleu. Écrire une fonction `couleur_rgb` qui prend trois entiers en paramètres compris entre 0 et 255 et qui **renvoie** la chaîne de caractère correspondante de la forme `'#RRVVBB'` ou chaque composante est codée sur un nombre à deux chiffres hexadécimaux.

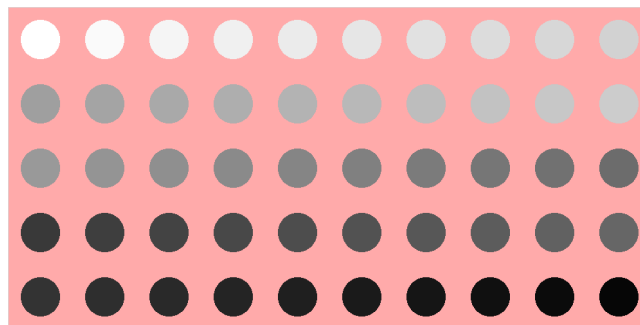
```

1 def couleur_rgb(r,g,b):
2     d = '0123456789ABCDEF'
3     rr = d[r//16]+d[r%16]
4     gg = d[g//16]+d[g%16]
5     bb = d[b//16]+d[b%16]
6     return '#' + rr + gg + bb

```

Exercice 6 – Cinquante nuances de gris (en boustrophédon) (★ ★ ★)

On cherche à obtenir la figure ci-dessous. On suppose le canvas déjà créé, en rose, et avec les bonnes dimensions.



1. Créer une fonction `gris(i)` qui prend un entier `i` entre 0 et 49 et qui renvoie du gris avec une proportion de `i/49` de noir. En particulier, on aura : `gris(0)=='#FFFFFF'` (0% de noir) et `gris(49)=='#000000'` (100% de noir).

```

1 def gris(i):
2     g = (49-i)*255//49
3     return couleur_rgb(g,g,g)

```

2. Écrire le code qui affiche 50 disques noirs (5 lignes et 10 colonnes). Le centre du premier disque (en haut à gauche) a pour coordonnées (50,50) et la distance entre le centre des disques est 100 pixels et leurs rayons 25 pixels.

```

1 def une_nuance():
2     for i in range(5):
3         for j in range(10):
4             g="#000000"
5             disque(100*j+50,100*i+50,30,g)

```

3. Écrire le code qui affiche les 50 disques mais cette fois-ci avec nuance de gris et en boustrophédon : la première ligne sera parcourue de gauche à droite, la suivante de droite à gauche, la troisième de nouveau de gauche à droite et ainsi de suite.

```

1 def cinquante_nuances():
2     n=0
3     for i in range(5):
4         for j in range(10):
5             g=gris(n)
6             if i%2==0:
7                 disque(100*j+50,100*i+50,30,g)
8             else:
9                 disque(100*(10-j-1)+50,100*i+50,30,g)
10            n=n+1

```

Exercice 7 – Chaîne de montagnes (*)

Écrivez une fonction `dessine_montagnes(p0,n,h,l)` qui dessine n montagnes de hauteur h sur une longueur totale de l en partant de p_0 . Par exemple, `dessine_montagnes((20,20),6,1,12)` produira le dessin ci-dessous.



où le pied gauche de la première montagne est en $p_0=(x_0, y_0)$, le pied droit de la dernière montagne sera aux coordonnées $(x_0 + l, y_0)$.

```

1 def dessine_montagnes(p0,n,h,l):
2     (x,y)=p0
3     dx = .5*l/n
4     for i in range(1,n+1) :
5         p=(x,y)
6         (x,y) = (x+dx, y-h)
7         Dessin.create_line(p, (x,y))
8         p=(x,y)
9         (x,y) = (x+dx, y+h)
10        Dessin.create_line(p, (x,y))

```