Séance 4 : Graphismes, objets et variables globales

L1 – Université Côte d'Azur

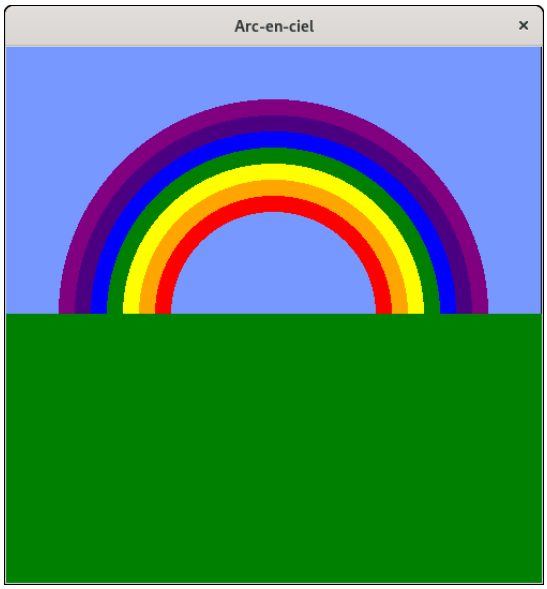
Dans le répertoire `Python` que vous avez créé pour les premières séances, créez un repertoire TP 4. On utilisera un fichier par exercice. Chaque fichier devra commencer par les lignes suivantes qui permettent de créer une fenêtre graphique. La hauteur et la largeur du *canvas* (partie de la fenêtre où l'on dessine) pourront être changées si nécessaire.

```
import tkinter as tk

# On crée une fenêtre
root = tk.Tk()
root.title("Mon programme Tk")

# On crée un canvas (zone de dessin)
Hauteur = 500
Largeur = 500
Dessin=tk.Canvas(root,height=Hauteur,width=Largeur,bg="white")
Dessin.pack()
```

Les fonctions pour tracer des cercles en `Tk` ne sont pas très pratiques. Vous êtes libre de réutiliser les fonctions `cercle` et `disque` définies dans le transparent du cours *« Écrire sa propre fonction de tracer de cercle »*

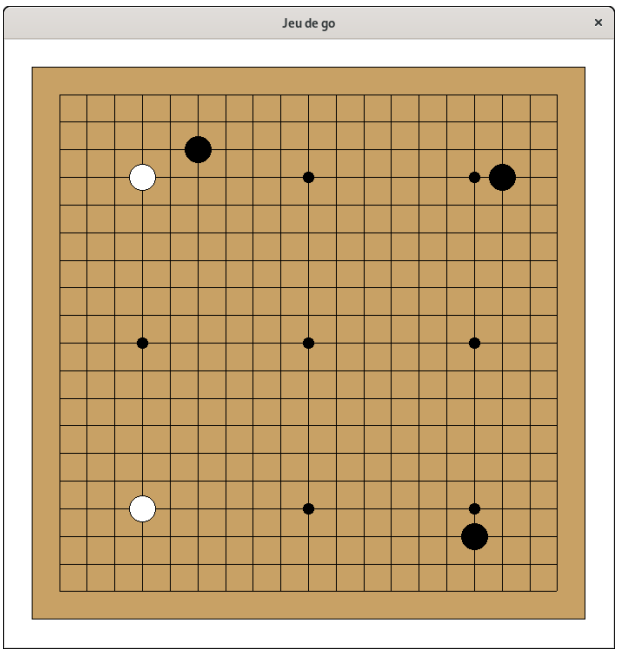
Exercice 1 — L'arc-en-ciel (⋆)

1. Tracer deux rectangles correspondant au ciel en bleu (on pourra utiliser la couleur *#7799FF*) et au sol en vert.
2. Tracer un arc-en-ciel. On pourra utiliser les couleurs pré-définies en `Tk` : *purple*, *indigo*, *blue*, *green*, *yellow*, *orange*, *red*. Si vous ne voyez pas comment faire, je vous laisse regarder le transparent intitulé « Exemple 2 : tracer une cible ».



Exercice 2 — Jeu de go (⋆⋆)

En 2016, pour la première fois un ordinateur a battu le numéro 1 mondial au jeu de go. L'objectif de l'exercice est de dessiner un goban (le plateau de jeu) puis de représenter sur ce goban les premiers coups de cette partie historique.



1. Créer une fenêtre de dimensions 700 × 700.

```
root = tk.Tk()
root.title("Jeu de go")
Hauteur = 700
Largeur = 700
Dessin=tk.Canvas(root,height=Hauteur,width=Largeur,bg="white")
Dessin.pack()
```

Un goban est constitué de 19 lignes horizontales et 19 lignes verticales (ce qui donne 18 colonnes et 18 rangées). Pour cet exercice, on fera en sorte que la marge sur le goban ainsi que la marge blanche dans le *canvas* aient la même taille que les colonnes. Nous nommerons cette taille `delta`.

2. Initialiser une variable `delta` qui correspond à l'écart entre deux lignes.

```
delta=Hauteur/22 # Hauteur=Largeur
```

3. Tracer le goban. On pourra choisir comme couleur pour le goban : *#C8A165*

```
Dessin.create_rectangle(delta,delta,Largeur-delta, Hauteur-delta,fill='#C8A165')

for i in range(2,21):
    Dessin.create_line((2*delta,delta*i) , (20*delta, delta*i))
    Dessin.create_line((delta*i,2*delta) , (delta*i, 20*delta))
```

Sur le goban, on ne joue pas dans les cases, mais sur les intersections. L'intersection en bas à gauche a pour coordonnées (1,1) et celle en haute à droite (19,19).

4. Placer les neufs étoiles (ce sont les petits cercles noirs) qui sont aux coordonnées (4,4), (4,10), (4,16), (10,4), (10,10), (10,16), (16,4), (16,10), (16,16). Les neuf étoiles sont visibles dans le dessin ci-dessus (sauf pour les deux cachées sous les pierres blanches)

```
def place_étoile(x,y):
    disque((x+1)*delta,(y+1)*delta,delta/5,'black')

for i in range(4,17,6):
    for j in range(4,17,6):
        place_étoile(i,j)
```

5. Créer une procédure `place_pierre(x,y,couleur)` qui place des pierres (les jetons noirs ou blanc) sur le goban. Chaque pierre aura une largeur de `delta`

```
def place_pierre(x,y,couleur):
    disque((x+1)*dx,(19-y+2)*dy,dx//2,couleur)
```

6. Tester votre procédure en jouant les cinq premiers coups de la célèbre partie de 2016. Vous devez obtenir la même image que celle du dessus.

```
place_pierre(17,16,'black')
place_pierre(4,16,'white')
place_pierre(16,3,'black')
place_pierre(4,4,'white')
place_pierre(6,17,'black')
```

7. Les couleurs alternant à chaque tour, créez une variable globale `couleur` qui change de valeur à chaque appel de la fonction `place_pierre` de telle sorte que l'on puisse jouer les premiers coups avec le code ci-dessous.

```
place_pierre(17,16)
place_pierre(4,16)
place_pierre(16,3)
place_pierre(4,4)
place_pierre(6,17)
```

```
couleur='black'

def place_pierre(x,y):
    global couleur
    disque((x+1)*delta,(19-y+2)*delta,delta//2,couleur)
    if couleur=='black':
        couleur='white'
    else:
        couleur='black'
```

Exercice 3 — Tracer des droites (⋆⋆)

```
def affiche_pixel(x,y,couleur):
    Dessin.create_rectangle(x,y,x,y,fill=couleur,outline='')
```

1. En utilisant la fonction `affiche_pixel` et des boucles `for`, tracer un carré de 200 pixels de côtés dont le sommet en haut à gauche a pour coordonnée (100,100).

```
def afficher_carré():
    xmin = 100
    ymin = 100
    xmax = xmin+200
    ymax = ymin+200
    # On trace les lignes horizontales
    for x in range(xmin,xmax+1):
        affiche_pixel(x,ymin,'black')
        affiche_pixel(x,ymax,'black')
    # On trace les lignes verticales
    for y in range(ymin,ymax+1):
        affiche_pixel(xmin,y,'black')
        affiche_pixel(xmax,y,'black')

afficher_carré()
```

Pour tracer des lignes autre que des verticales et des horizontales, nous aurons besoin d'un algorithme.
On se donne deux points : $p_1 = (x_1, y_1)$ et $p_2 = (x_2, y_2)$.
Ces deux points définisse une droite d'équation $y = ax + b$ où $a = \dfrac{y_2 - y_1}{x_2 - x_1}$ et $b = y_1 - ax_1$.
Pour chaque x compris en x_1 et x_2 exécuter les deux instructions suivantes :

(a) Calculer $y = ax + b$ puis remplacer y par l'entier le plus proche
(b) tracer le pixel de coordonnées (x, y)

2. On commencera par écrire une fonction `entier_le_plus_proche(q)` tel que :

```
>>> entier_le_plus_proche(17.4)
17
>>> entier_le_plus_proche(17.5)
18
>>> entier_le_plus_proche(17.6)
18
```

```
def entier_le_plus_proche(q):
    n = floor(q)
    if q<n+0.5:
        return n
    else:
        return n+1
```

3. Programmer une fonction `affiche_segment(p1,p2)` qui implémente l'algorithme précédant (on fera appel à la fonction `entier_le_plus_proche`)

```python
def affiche_segment_naïf(p1,p2):
    (x1,y1)=p1
    (x2,y2)=p2
    a = (y2-y1)/(x2-x1)
    b = y1-a*x1
    for x in range(x1,x2+1):
        y = entier_le_plus_proche(a*x+b)
        affiche_pixel(x,y,'black')
```

4. La tester avec les segments (p_1, p_2) et (p_1, p_3) avec $p_1 = (100, 100)$, $p_2 = (200, 150)$ et $p_3 = (150, 400)$. Que remarquez-vous ?

```python
p1=(100,100)
p2=(200,150)
p3=(150,300)
affiche_segment_naïf(p1,p2)
affiche_segment_naïf(p1,p3)
```

5. Corriger la fonction pour qu'elle fonctionne même dans le cas où $|y2 - y1| > |x2 - x1|$.

```python
def affiche_segment(p1,p2):
    (x1,y1)=p1
    (x2,y2)=p2
    if abs(y2-y1) >= abs(x1-x2):
        a = (x2-x1)/(y2-y1)
        b = x1-a*y1
        for y in range(y1,y2+1):
            x = entier_le_plus_proche(a*y+b)
            affiche_pixel(x,y,'black')
    else:
        a = (y2-y1)/(x2-x1)
        b = y1-a*x1
        for x in range(x1,x2+1):
            y = entier_le_plus_proche(a*x+b)
            affiche_pixel(x,y,'black')

p1=(100,100)
p2=(200,150)
p3=(125,200)
affiche_segment(p1,p2)
affiche_segment(p1,p3)
```

Exercice 4 — Tracer un emploi du temps (⋆⋆)

1. Commencer par tracer le quadrillage avec les jours en haut et les horaires à gauche.

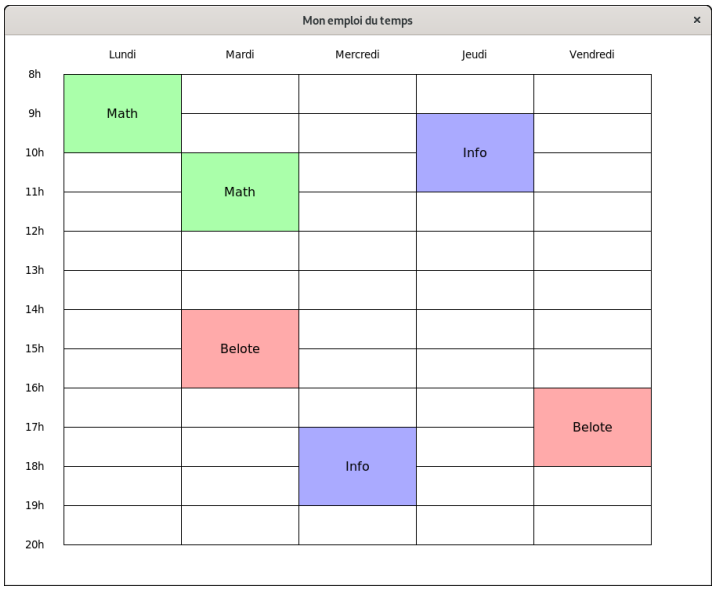
```
dx = Largeur/6
dy = Hauteur/14

def jour(j):
        if j==1:
                return "Lundi"
        elif j==2:
                return "Mardi"
        elif j==3:
                return "Mercredi"
        elif j==4:
                return "Jeudi"
        else:
                return "Vendredi"


Dessin.create_rectangle(dx/2,dy,Largeur-dx/2, Hauteur-dy)


for i in range(8,21):
        Dessin.create_text(dx/4,dy*(i-8+1),text=str(i)+"h")
        Dessin.create_line((dx/2,dy*(i-8+1)) , (Largeur-dx/2, dy*(i-8+1)))


for i in range(1,6):
        Dessin.create_text(i*dx ,dy/2,text=jour(i))
        Dessin.create_line(i*dx+dx/2 , dy, i*dx+dx/2,Hauteur-dy)
```

2. Créer une classe `UE` avec comme attributs le nom de l'UE et une couleur. Cette classe devra définir une méthode `séance(self,joueur,début)` qui affiche le créneau de la séance sur l'empoi du temps. On supposera que toutes les séances durent deux heures.

```
class UE:

        def __init__(self,nom,couleur):
                self.nom=nom
                self.couleur=couleur


        def séance(self,jour,début):
                (x1,y1)= (dx*(jour-.5),dy*(début-8+1))
                p1=(x1,y1)
                p2=(x1+dx,y1+2*dy)
                Dessin.create_rectangle(p1,p2,fill=self.couleur)
                Dessin.create_text(x1+.5*dx,y1+dy,text=self.nom,font="20pt")
```

3. Générez votre propre emploi du temps. On améliorera la classe et les méthodes pour afficher les salles et le type de cours (CM, TD, TP) et gérer la durée des séances (2h ou 3h).

```python
# Le code suivant doit
# permettre d'obtenir
# l'image de droite

math=UE("Math","#AAFFAA")
info=UE("Info","#AAAAFF")
belote=UE("Belote","#FFAAAA")

math.séance(1,8)
math.séance(2,10)
info.séance(3,17)
info.séance(4,9)
belote.séance(2,14)
belote.séance(5,16)
```



Exercice 5 — Courbe paramétrée (⋆⋆)

Écrivez une fonction `paramétrique(f,g,tmin,tmax,dt)` qui affiche la courbe paramétrée d'équation

$$\begin{cases} x = f(t) \\ y = g(t) \end{cases}$$

en échantillonnant les valeurs de t à partir de $t_{\min}$, jusqu'à $t_{\max}$, avec comme pas dt. Faites afficher la courbe obtenue en prenant $f(t) = 50\sin(t)$, $g(t) = 50\sin(2t)$, $t_{\min} = -\pi$, $t_{\max} = \pi$, et $dt = \frac{\pi}{1000}$.