



UNIVERSITÉ  
CÔTE D'AZUR

# Bases de l'informatique 1

## Cours 9. Animations avec Tk

---

Olivier Baldellon

Courriel : prénom.nom@univ-cotedazur.fr

Page professionnelle : <https://upinfo.univ-cotedazur.fr/~obaldellon/>

LICENCE I — FACULTÉ DES SCIENCES ET INGÉNIERIE DE NICE — UNIVERSITÉ CÔTE D'AZUR

 Partie I. Programmation événementielle

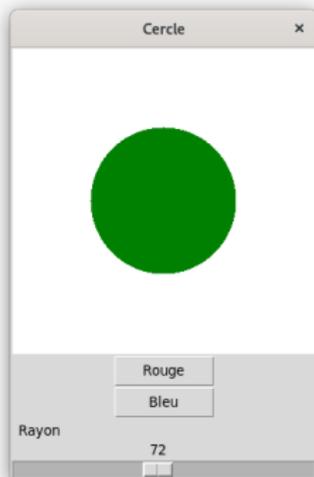
 Partie II. Animations

 Partie III. Projet

 Partie IV. Écrire du code propre

 Partie V. Table des matières

- ▶ On souhaite faire une petite animation qui affiche un cercle

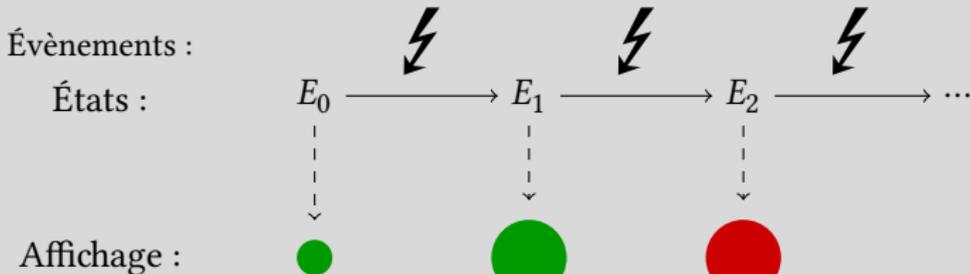


avec

- ▶ un bouton pour changer la couleur en rouge
  - ▶ un autre pour la changer en bleu
  - ▶ un curseur pour choisir le taille
- 
- ▶ Comment programmer une telle interaction avec l'utilisateur ?

- ▶ On ne va pas chercher à transformer une image en une autre.
- ▶ On va introduire une variable pour définir l'état du système
  - ▶ l'état correspond aux paramètres de l'image
  - ▶ à partir de l'état, on peut dessiner l'image correspondante
- ▶ Lorsque l'utilisateur déclenche un évènement (ex : clic sur bouton)
  - ▶ on modifie l'état
  - ▶ on efface l'ancienne image
  - ▶ on retrace l'image correspondante au nouvel état.

- ▶ Ici, l'image ne dépend que de deux **paramètres** :
  - ▶ La couleur du disque
  - ▶ La taille du disque
- ▶ Il y a trois types d'**évènements** possibles
  - ▶ clic sur le bouton « rouge »
  - ▶ clic sur le bouton « bleu »
  - ▶ déplacement du curseur
- ▶ Au début on initialise l'état et on l'affiche
  - ▶ L'utilisateur change le rayon sur le curseur : l'état change et on l'affiche
  - ▶ L'utilisateur clique sur le bouton « rouge » : l'état change et on l'affiche
  - ▶ etc.



- ▶ On stocke l'état dans un objet
  - ▶ Avec deux attributs (couleur et rayon) et une méthode (**affichage**)

SCRIPT

```
class État():  
  
    def __init__(self):  
        self.couleur='green'  
        self.rayon=Largeur/4  
        self.affichage() # On fait le premier affichage  
  
    def affichage(self):  
        Dessin.delete('all') # On efface tout  
        (x0,y0)=(Largeur//2,Hauteur//2)  
        disque(x0,y0,self.rayon,self.couleur)  
  
état=État() # On définit l'état de manière globale
```

- ▶ Remarquez que l'on fait appel à la méthode `affichage` dès l'initialisation
- ▶ À chaque fois que l'on fait l'affichage :
  - ▶ on efface les objets précédents pour ne pas saturer la mémoire

- ▶ On crée maintenant des boutons et un curseur
- ▶ Ainsi que les fonctions correspondantes aux évènements associés

SCRIPT

```
def rouge(): # Appelée lorsqu'on clique sur Bouton 1
    état.couleur='red'
    état.affichage()

def bleu(): # Appelée lorsqu'on clique sur Bouton 2
    état.couleur='blue'
    état.affichage()

def change_taille(x): # Appelée lorsqu'on bouge le curseur
    état.rayon=int(x)
    état.affichage()

bouton1 = tk.Button(root,text="Rouge",command=rouge,width=9)
bouton2 = tk.Button(root,text="Bleu",command=bleu,width=9)
curseur = tk.Scale(root, orient="horizontal", length=Largeur,
                  label='Rayon',command=change_taille,
                  from_=1, to=Hauteur//2)

curseur.set(état.rayon) # Placement initial du curseur
```

SCRIPT

```
import tkinter as tk
Hauteur,Largeur = 800,800
root = tk.Tk()
root.title("Cercle")
Dessin = tk.Canvas(root,height=Hauteur,width=Largeur,bg='white')
Dessin.pack()

def disque(x,y,r,c): # (cf cours 5)

class État(): # Définit l'état et la méthode affichage
    état=État() # On initialise l'état

def rouge():          # Les actions associées
def bleu():           # aux évènements
def change_taille(x):#

bouton1 = # Les éléments de l'interface
bouton2 = # graphiques qui déclencheront
curseur = # les évènements

bouton1.pack() # On les positionne
bouton2.pack() # sur la fenêtre
curseur.pack()

root.mainloop() # À mettre à la fin de chaque programme Tk
```

## ▶ Souris

<Button-1>	Clic gauche	<Motion>	La souris bouge
<Button-2>	Clic central	<Button>	Clic sur un bouton
<Button-3>	Clic droit	<ButtonRelease>	Fin d'un clic

## ▶ Clavier

<Up>	touche 	<Key-a>	touche  (minuscule)
<Down>	touche 	<Key-B>	touche  (majuscule)
<Left>	touche 	<KeyPress>	on appuie sur une touche
<Right>	touche 	<KeyRelease>	on relâche une touche

## ▶ Pour associer l'évènement à une commande :

SCRIPT

```
def ça_bouge(event): # l'argument est obligatoire
    # Coordonnée et bouton de la souris (Bouton 1, 2 ou 3)
    print(event.x, event.y, event.num)
    # Symbole clavier (a, B, Enter, space)
    print(event.keysym)

root.bind('<Motion>', ça_bouge)
```

- ▶ On ajoute trois nouveaux évènements
  - ▶  ou  : le rayon augmente ou diminue
  - ▶ Mouvement souris : si on est sur le disque il devient rouge (bleu sinon)

SCRIPT

```
def sur_cercle(x1,y1):
    (x0,y0)=(Largeur/2,Hauteur/2)
    return (x0-x1)**2 + (y0-y1)**2 < état.rayon**2

def ça_bouge(event):
    if sur_cercle(event.x,event.y): état.couleur='red'
    else: état.couleur='blue'
    état.affichage()

def plus_grand(event):
    état.rayon = état.rayon+1
    état.affichage()

def plus_petit(event):
    état.rayon = état.rayon-1
    état.affichage()

root.bind('<Motion>', ça_bouge)
root.bind('<Down>', plus_petit)
root.bind('<Up>', plus_grand)
```

 Partie I. Programmation événementielle

 Partie II. Animations

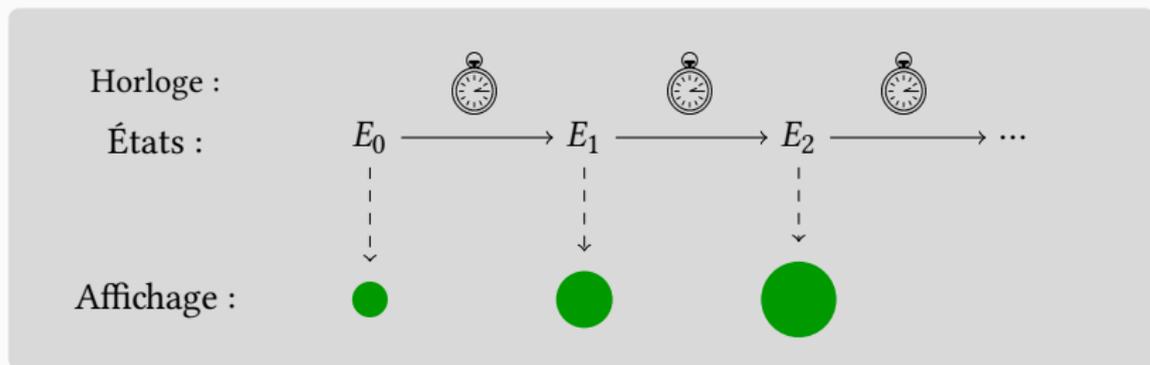
 Partie III. Projet

 Partie IV. Écrire du code propre

 Partie V. Table des matières

- ▶ Pour l'instant, l'état évoluait suite à une action de l'utilisateur
  - ▶ Les évènements appelaient des fonctions qui modifiaient l'état.
- ▶ On veut maintenant que l'état puisse évoluer au cours du temps.
  - ▶ On va créer une nouvelle fonction
  - ▶ On va appeler cette fonction à intervalle régulier

- ▶  Tic Tac Tic



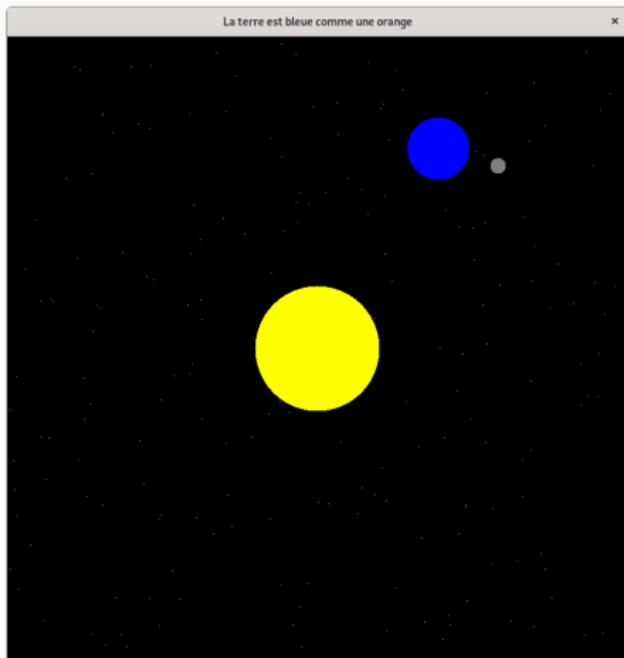
- ▶ Le disque donne l'impression de grossir au cours du temps
  - ▶ 24 images par seconde donnent l'illusion d'un mouvement continu

- ▶ On va rajouter un attribut `self.temps` à notre état.
- ▶ Il suffit que notre affichage dépende de `état.temps`.

```
def tictac():  
    état.temps = état.temps+1  
    état.affichage()  
    Dessin.after(10,tictac)
```

SCRIPT

- ▶ Pour utiliser `tictac`
  - ▶ Il faut le lancer une première fois
  - ▶ À chaque étape la fonction incrémente l'attribut `temps` de `état` ...
  - ▶ ... et relance l'affichage()
  - ▶ Enfin, la fonction demande a être rappelée dans 10 ms.



On cherche à représenter le Soleil,  
la Terre et la Lune en rotation

- ▶ Le tout sur un ciel étoilé
- ▶ on rappelle la formule d'un point sur un cercle (cours 5)

$$\begin{cases} x(t) = x_0 + R \cos(t), \\ y(t) = y_0 + R \sin(t). \end{cases}$$

```
def rotation(x,y,r,w,t)
# position d'un point à l'instant t, tournant sur
# un cercle de centre (x,y) et de rayon r avec une
# vitesse angulaire w
return (x + r*cos(-t*w) , y + r*sin(-t*w))
```

SCRIPT

SCRIPT

```
class État():
    def __init__(self): # Deux attributs : le temps et une liste d'étoiles
        (L,H) = (Largeur,Hauteur)
        self.étoiles = [(randint(0,L),randint(0,H)) for e in range(200)]
        self.temps=0
        self.affichage()

    def affichage(self):
        Dessin.delete('all') # On efface tout
        for (x,y) in self.étoiles: # Les étoiles
            Dessin.create_rectangle((x-1,y-1),(x+1,y+1),fill='white')

            (x0,y0) = (Largeur//2,Hauteur//2) # Le Soleil
            (x1,y1) = rotation(x0,y0,300,1/100,self.temps) # La Terre
            (x2,y2) = rotation(x1,y1, 80,12/100,self.temps) # La Lune
            disque(x0,y0,80,'yellow')
            disque(x1,y1,40,'blue')
            disque(x2,y2,10,'gray')

def tictac(): # Toutes les 20 ms, on change l'état et on l'affiche
    état.temps = état.temps+1
    état.affichage()
    Dessin.after(20,tictac)

état=État()
tictac() # On lance l'horloge
root.mainloop() # À mettre à la fin de chaque programme Tk
```

- ▶ Ce cours n'a pas pour objectif de faire de vous des experts en Tk.
    - ▶ le but était de vous montrer les possibilités,
    - ▶ de vous expliquer les grands principes (programmation événementielle)
    - ▶ et de vous permettre de continuer seul.
  - ▶ Écrire des programmes est un excellent moyen de progresser
    - ▶ C'est en codant qu'on apprend à programmer
    - ▶ Les jeux et animations sont un bon prétexte pour se motiver
  - ▶ Pour les curieux, il y a le site très complet en français :
- <http://pascal.ortiz.free.fr/contents/tkinter/tkinter/>
- ▶ Tk est très pratique pour faire des interfaces graphiques complètes
    - ▶ Thonny est fait en Tk.
    - ▶ Si vous êtes intéressés par les jeux, vous pouvez aussi regarder pygame.

- 🍃 Partie I. Programmation événementielle
- 🍃 Partie II. Animations
- 🍃 **Partie III. Projet**
- 🍃 Partie IV. Écrire du code propre
- 🍃 Partie V. Table des matières

- ▶ il y a un projet Tk pour ceux et celles qui le souhaitent.
- ▶ Le projet sera noté sur 2 points.
  - ▶ 1 point bonus sur le partiel
  - ▶ 1 point bonus sur l'examen terminal
- ▶ Il faudra créer un petit programme avec un canevas correspondant :
  - soit à un petit jeu,
  - soit à une animation
  - soit à une image statique paramétrable :
    - ▶ avec des boutons,
    - ▶ des curseurs
    - ▶ et les touches du clavier

- ▶ Le projet est facultatif et individuel.
- ▶ Consignes pour le rendu :
  - Devra être rendu sur Moodle
  - au plus tard le jour de l'examen.
  - Si plusieurs fichiers, il faudra rendre une archive zip.
- ▶ Consignes spécifiques
  - ne doit utiliser que les outils vu en cours.
  - Si vous souhaitez utiliser une bibliothèque tierce, il faudra en faire la demande préalable.

- ▶ Votre code devra être conçu suivant le format du cours
  - un objet état possédant une méthode `affichage` et responsable de la production de l'image dans le canevas.
  - Pour une animation, on utilisera une fonction `tictac` pour faire évoluer l'état au cours du temps.
- ▶ Le projet devra être codé proprement

En particulier je dois comprendre rapidement ce que fait votre code en le lisant. Les noms de fonctions et de variables doivent être explicites. Les choix non triviaux doivent être expliqués en commentaire.

- ▶ Zéro si :
  - ▶ non respect des consignes
  - ▶ rendu en retard
  - ▶ ne fonctionne pas sur ma machine

- 🍃 Partie I. Programmation événementielle
- 🍃 Partie II. Animations
- 🍃 Partie III. Projet
- 🍃 Partie IV. Écrire du code propre
- 🍃 Partie V. Table des matières

- ▶ Un bon code se passe de commentaires.
  - ▶ noms de variables claires et explicites
  - ▶ noms de fonctions claires et explicites
- ▶ Les commentaires servent à expliquer :
  - ▶ les points subtiles (commentaires ajoutés lors de la correction d'un bug)
  - ▶ les algos non triviaux
  - ▶ les interfaces des types abstraits (= documentation).
  - ▶ pour le code de base, ils sont souvent inutiles
- ▶ Pas de variables globales
  - ▶ ou alors en début de fichier
  - ▶ et commentée
  - ▶ et seulement si ça simplifie significativement votre code
- ▶ Privilégiez les fonctions pures
- ▶ Un code non testé est un code faux !

- ▶ Pouvez-vous comprendre votre code :
  - ▶ sans y avoir touché depuis un an ?
  - ▶ en étant à moitié bourré ?
  - ▶ à cinq heures du matin ?
- ▶ si la réponse est non, c'est que votre code est trop compliqué !
- ▶ Simplifier votre code !
  - ▶ dès que votre code marche : simplifiez-le !
  - ▶ si vous ne comprenez plus votre code : simplifiez-le !
  - ▶ si vous cherchez à déboguer votre code : simplifiez-le !
- ▶ Un vrai programme est une interaction entre de nombreuses fonctions.
  - ▶ même avec des fonctions triviales et bien écrites
  - ▶ le comportement dû aux interactions peut-être complexe
  - ▶ pour cette raison : privilégiez les fonctions pures

**Brian KERNIGHAN**

« Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as you can, you are, by definition, not smart enough to debug it. »

Déboguer est deux fois plus compliqué qu'écrire le code dans un premier temps. Ainsi, si vous avez écrit le code en y mettant toute votre ingéniosité, vous n'êtes par définition pas assez intelligent pour le déboguer.



Merci pour votre attention

Questions



# Cours 9 — Animations avec Tk

## Partie I. Programmation événementielle

Objectif

Programmation événementielle

Évolution de l'état

Définir l'état

Ajouter des boutons

Le code (presque) complet

Autres événements

Exemple

## Partie II. Animations

Faire une animation

Gérer le temps

Révolutions!

Révolutions : le code

Conclusion

## Partie III. Projet

Le projet

Consignes générales

Consignes sur le code

## Partie IV. Écrire du code propre

Coder proprement

Coder simplement

Citations

Dernier TP : créez votre propre python

## Partie V. Table des matières

- ▶ © 2024 — Olivier Baldellon
- ▶ Ce document est publié sous licence **CC-BY Attribution 4.0** 
  - Vous êtes autorisé à :
    - ▶ **Partager** — copier, distribuer et communiquer le matériel par tous moyens et sous tous formats pour toute utilisation, y compris commerciale.
    - ▶ **Adapter** — remixer, transformer et créer à partir du matériel pour toute utilisation, y compris commerciale.
  - Selon les conditions suivantes :
    - ▶ **Attribution** — Vous devez créditer l'œuvre, intégrer un lien vers la licence et indiquer si des modifications ont été effectuées à l'œuvre. Vous devez indiquer ces informations par tous les moyens raisonnables, sans toutefois suggérer que l'Offrant vous soutient ou soutient la façon dont vous avez utilisé son œuvre.
- ▶ <https://creativecommons.org/licenses/by/4.0/deed.fr>