

# Tkinter

---

## Tkinter

Pour tester son installation

Fonctionnement de *Tkinter*

La fenêtre *Tkinter*

Les widgets Tkinter - Le fonctionnement générique

Les widgets Tkinter - Quelques widgets de base

Les étiquettes `Label`

Les boutons `Button`

Les champs de saisie `Entry`

Les cases à cocher `Checkbutton`

Les widgets Tkinter - Les conteneurs

`Frame`

Les évènements

Les couleurs en Tkinter

Les formes, la classe `Canvas`

La bibliothèque `C3Geometry`

Fonctionnement de base (commune à toutes les classes)

La méthode `resize`

La méthode `translate`

La classe `Ligne`

La classe `Rectangle`

La classe `Carre`

La classe `Cercle`

La classe `Croix`

La classe `AfterEvent`

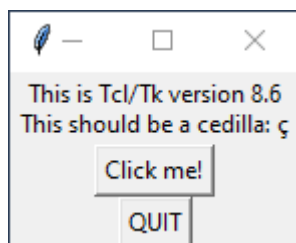
## Pour tester son installation

---

- Dans votre terminal, lancer la commande suivante :

```
python -m tkinter
```

- Si la fenêtre suivante apparaît, alors **tkinter** est bien installé sur votre système.



## Fonctionnement de *Tkinter*

---

- Toutes les classes de **Tkinter** sont situées dans le module **tkinter**.
- La première étape nécessite d'importer le module :

```
import tkinter as tk
```

- L'élément visuel de base est la fenêtre. Cet élément **racine** est une instance de la classe `Tk` du module **tkinter**.

```
root = tk.Tk()
```

- Pour qu'une fenêtre s'affiche, il faut lancer la **boucle principale**.
  - Cette boucle permet de rester à l'écoute des événements : clique, touche appuyer, etc.
  - Elle permet à la fenêtre de se mettre à jour.
  - La boucle se lance par une simple commande :

```
root.mainloop()
```

## La fenêtre *Tkinter*

- La méthode `title("str")` de la classe `Tk` permet de modifier le titre de la fenêtre.

```
root.title("Ma première fenêtre tkinter")
```

- La méthode `geometry("chaîne_formatée")` permet de modifier la taille de la fenêtre.
  - La chaîne en paramètre utilise le formatage (minimal) suivant :

```
{taille horizontale en px}x{taille verticale en px}
```

## Les widgets Tkinter - Le fonctionnement générique

- Les **widgets** sont les éléments graphiques. Ainsi, en tkinter, tout est un widget.
  - La fenêtre est souvent appelée le **widget principal**.
- Tous les widgets sont créés à partir d'une classe dédiée.
  - Le constructeur de ces classes a toujours la même forme :

```
un_widget = tk.Unwidget(widget_principal, autres_paramètres = "valeur")
```

- La méthode `config` permet de modifier un paramètre du widget après sa création.
  - Les paramètres varient selon le type de widget.

```
un_widget.config(un_parametre='valeur')
```

- La méthode `cget` permet de récupérer la valeur d'un paramètre

```
un_widget.cget('un_parametre')
```

- Pour s'afficher, un widget doit être positionné dans le widget parent.
- Il existe trois positionnements possibles :
  - **pack** : Le widget parent sera coupé en deux. La première partie sera la zone occupée par le widget. La seconde partie sera la partie libre.
  - **place** : Permet de positionner le widget au pixel près.
  - **grid** : Permet de positionner le widget sur une grille programmée par l'utilisateur.
- Pour placer un élément sur une grille, il suffit d'appeler la méthode `grid`.

```
un_widget.grid(column=1, row=1) # Place le widget à la deuxième colonne et à la deuxième ligne
un_widget.grid() # Place le widget automatiquement
```

- Tkinter possède plusieurs paramètres de base aux widgets :
  - **foreground** : Couleur du texte du widget
  - **activeforeground** : Couleur du texte lorsque le curseur survole le widget
  - **background** : Couleur de fond du widget
  - **activebackground** : Couleur de fond du widget lorsque le curseur survole le widget
  - **padx** : Marge horizontale entre les bords du widget et de son widget parent
  - **pady** : Marge verticale entre les bords du widget et de son widget parent
  - **width** : Largeur du widget en taille de police
  - **height** : Hauteur du widget en taille de police

## Les widgets Tkinter - Quelques widgets de base

### Les étiquettes `Label`

- Les étiquettes sont créées à partir de la classe `Label`.
- Le paramètre `text` définit la chaîne que le widget affichera.
- Le paramètre `justify` permet de modifier l'alignement du texte (`tk.LEFT` ou `tk.RIGHT`)

```
label = tk.Label(root, text='Hello world', foreground='#892222',
                  background='#FFAAAA', activebackground="#FFBBBB", padx="10",
                  pady="10")
```

### Les boutons `Button`

- Les boutons sont créés à partir de la classe `Button`.
- Le paramètre `text` définit la chaîne qui s'affichera dans le bouton.
- Le paramètre `command` définit la fonction à exécuter lorsque l'utilisateur clique sur le bouton.

```
def foo() :
    print("click")

button = tk.Button(root, text='Cliquez sur moi', command=foo)
```

- Pour appeler une fonction avec des paramètres, il faut encapsuler la fonction et les paramètres grâce à la fonction `partial` du module `functools`.

```
from functools import partial

def bar(label, msg) :
    label.config(text=msg)

etiquette = tk.Label(root, text='Ce msg va bientôt changer')
btn = tk.Button(root, text='On change le Label', command=partial(bar, etiquette,
"Le nouveau message"))
```

## Les champs de saisie Entry

- Les champs de saisie (texte) sont créés à partir de la classe `Entry`.
- Le paramètre `textvariable` permet de récupérer la valeur du champ.
  - Il est de type `StringVar`. Le constructeur de `StringVar` prend le widget principal comme paramètre.
  - Pour récupérer le texte d'un `StringVar`, il suffit d'appeler sa méthode `get`.
  - Si la variable est modifiée, la valeur du champ est alors modifiée.

```
texte_champ = tk.StringVar(root)
champ = tk.Entry(root, textvariable=texte_champ)
```

## Les cases à cocher Checkbutton

- Les cases à cocher sont créées à partir de la classe `Checkbutton`.
- Le paramètre `variable` permet de récupérer la valeur du champ (cocher ou non).
  - Il est de type `BooleanVar`. Le constructeur de `BooleanVar` prend le widget principal et la valeur du champ (`True` ou `False`) comme paramètres.
  - Pour récupérer le texte d'un `BooleanVar`, il suffit d'appeler sa méthode `get`.
  - Si la variable est modifiée, la valeur du champ est alors modifiée.

```
is_checked = tk.BooleanVar(root, False)
checkbox = tk.Checkbutton(root, text="Changer le champs ?", variable=is_checked)
```

## Les widgets Tkinter - Les conteneurs

- Ces widgets permettent de contenir d'autre widget comme pour la fenêtre. Le placement est alors simplifié.
- Leurs utilisations pour les widgets communs reste identiques que pour une fenêtre.

### Frame

- Possède deux attributs intéressants :
  - **relief** : type de bordure ([voir les options possibles](#))
  - **bd** : Taille, en pixel, de la bordure (par défaut, 2 pixels)
  - **bg** : Couleur normale de fond
  - **height** : hauteur du frame
  - **width** : Largeur du frame

```
import tkinter as tk

root = tk.Tk()
root.geometry("280x280")
# Pas de bordure
f1 = tk.Frame(root, bd=0)
# Petit message
label = tk.Label(f1, text="Hello world")
# Étiquette centré
label.pack(side=tk.LEFT)

f1.pack()
root.mainloop()
```

## Les évènements

- Pour « écouter » un évènement, il faut ajouter cette écoute grâce à la fonction `bind` d'un widget.
- Il existe plusieurs évènements différents :
  - `<ButtonPress>` : Pression d'un bouton de la souris ;
    - `<ButtonPress-1>` : Représente le bouton gauche;
    - `<ButtonPress-2>` : Représente le bouton du centre;
    - `<ButtonPress-3>` : Représente le bouton droit;
  - `<ButtonRelease>` : Relâchement d'un bouton de la souris ;
  - `<KeyPress>` : Pression d'un bouton du clavier ;
    - `<KeyPress-a>` : Pression du bouton **a** du clavier ;
  - `<KeyRelease>` : Relâchement d'un bouton du clavier ;
- Il existe aussi des modificateurs :
  - `Control` : la touche Control sera pressée ;
    - `<Control-KeyPress-s>` : Les touches **contrôles** et **s** ont été appuyées.
  - `Shift` : la touche Shift sera pressée ;
  - `Alt` : la touche Alt sera pressée ;
  - `Double` : double clique
  - `Triple` : triple clique
  - `Quadruple` : quadruple clique
- La fonction à appeler doit toujours prendre un évènement `event` en paramètre

```
def foo(texte, event) :
    print(texte)

root.bind("<ButtonPress-1>", partial(foo, "Click !"))
```

- Pour avoir la position du d'un évènement, il suffit d'appeler les attributs `x` et `y` de `event`.

## Les couleurs en Tkinter

- Concernant les couleurs, Tkinter accepte les codes hexadécimaux.

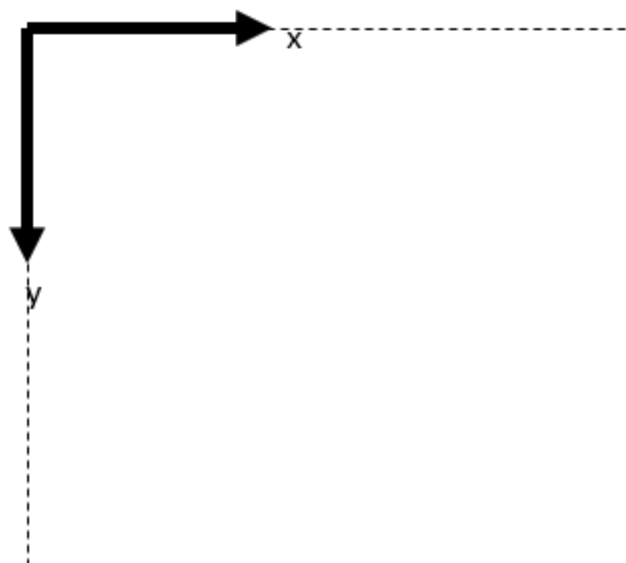
- Les codes abrégés sont aussi supportés : `#665599` == `# 659`.
- La bibliothèque accepte aussi les couleurs nommées comme sur l'image suivante :

Named colour chart																								
snow	deep sky blue	gold	seashell3	SlateBlue3	LightBlue3	SpringGreen2	DarkGoldenrod1	brown4	pink3	purple1	gray26	gray64												
ghost white	sky blue	light goldenrod	seashell4	SlateBlue3	LightBlue4	SpringGreen3	DarkGoldenrod2	salmon1	pink4	purple2	gray27	gray65												
white smoke	light sky blue	goldenrod	AntiqueWhite1	SlateBlue4	LightCyan2	SpringGreen4	DarkGoldenrod3	salmon2	LightPink1	purple3	gray28	gray66												
gainsboro	steel blue	dark goldenrod	AntiqueWhite2	RoyalBlue1	LightCyan3	green2	DarkGoldenrod4	salmon3	LightPink2	purple4	gray29	gray67												
floral white	light steel blue	rosy brown	AntiqueWhite3	RoyalBlue2	LightCyan4	green3	RosyBrown1	salmon4	LightPink3	MediumPurple1	gray30	gray68												
old lace	light blue	indian red	AntiqueWhite4	RoyalBlue3	PaleTurquoise1	green4	RosyBrown2	LightSalmon2	LightPink4	MediumPurple2	gray31	gray69												
linen	powder blue	saddle brown	bisque2	RoyalBlue4	PaleTurquoise2	chartreuse2	RosyBrown3	LightSalmon3	PaleVioletRed1	MediumPurple3	gray32	gray70												
antique white	pale turquoise	sandy brown	bisque3	blue2	PaleTurquoise3	chartreuse3	RosyBrown4	LightSalmon4	PaleVioletRed2	MediumPurple4	gray33	gray71												
papaya whip	dark turquoise	dark salmon	bisque4	blue4	PaleTurquoise4	chartreuse4	OliveDrab1	IndianRed1	orange2	thistle1	gray34	gray72												
blanched almond	medium turquoise	salmon	PeachPuff2	DodgerBlue2	CadetBlue1	OliveDrab1	IndianRed2	orange3	PaleVioletRed4	thistle2	gray35	gray73												
bisque	turquoise	light salmon	PeachPuff3	DodgerBlue3	CadetBlue2	OliveDrab2	IndianRed3	orange4	maroon1	thistle3	gray36	gray74												
peach puff	cyan	orange	PeachPuff4	DodgerBlue4	CadetBlue3	OliveDrab3	IndianRed4	DarkOrange1	maroon2	thistle4	gray37	gray75												
navajo white	light cyan	dark orange	NavajoWhite2	SteelBlue1	CadetBlue4	DarkOliveGreen1	sienna1	DarkOrange2	maroon3		gray38	gray76												
lemon chiffon	cadet blue	coral	NavajoWhite3	SteelBlue2	turquoise1	DarkOliveGreen2	sienna2	DarkOrange3	maroon4		gray39	gray77												
mint cream	medium aquamarine	light coral	NavajoWhite4	SteelBlue3	turquoise2	DarkOliveGreen3	sienna3	DarkOrange4	VioletRed1		gray40	gray78												
azure	aquamarine	tomato	LemonChiffon2	SteelBlue4	turquoise3	DarkOliveGreen4	sienna4	coral1	VioletRed2		gray41	gray42	gray79											
alice blue	dark green	orange red	LemonChiffon3	DeepSkyBlue2	turquoise4	khaki1	burlywood1	coral2	VioletRed3		gray43	gray80												
lavender	dark olive green	red	LemonChiffon4	DeepSkyBlue3	cyan2	khaki2	burlywood2	coral3	VioletRed4		gray44	gray81												
lavender blush	dark sea green	hot pink	cornsilk2	DeepSkyBlue4	cyan3	khaki3	burlywood3	coral4	magenta2		gray45	gray82												
misty rose	sea green	deep pink	cornsilk3	SkyBlue1	cyan4	khaki4	burlywood4	tomato2	magenta3		gray46	gray83												
dark slate gray	medium sea green	pink	cornsilk4	SkyBlue2	DarkSlateGray1	LightGoldenrod1	wheat1	tomato3	magenta4		gray47	gray84												
dim gray	light sea green	light pink	ivory2	SkyBlue3	DarkSlateGray2	LightGoldenrod2	wheat2	tomato4	orchid1		gray48	gray85												
slate gray	pale green	pale violet red	ivory3	SkyBlue4	DarkSlateGray3	LightGoldenrod3	wheat3	OrangeRed2	orchid2		gray49	gray86												
light slate gray	spring green	maroon	ivory4	LightSkyBlue1	DarkSlateGray4	LightGoldenrod4	wheat4	OrangeRed3	orchid3		gray50	gray87												
gray	lawn green	medium violet red	honeydew2	LightSkyBlue2	aquamarine2	LightYellow2	tan1	OrangeRed4	orchid4		gray51	gray88												
light gray	medium spring green	violet red	honeydew3	LightSkyBlue3	aquamarine4	LightYellow3	tan2	red2	plum1		gray52	gray89												
midnight blue	green yellow	medium orchid	honeydew4	LightSkyBlue4	DarkSeaGreen1	LightYellow4	tan4	red3	plum2		gray53	gray90												
navy	lime green	dark orchid	LavenderBlush2	SlateGray1	DarkSeaGreen2	yellow2	chocolate1	red4	plum3		gray54	gray91												
cornflower blue	yellow green	dark violet	LavenderBlush3	SlateGray2	DarkSeaGreen3	yellow3	chocolate2	DeepPink2	plum4		gray55	gray92												
dark slate blue	forest green	blue violet	LavenderBlush4	SlateGray3	DarkSeaGreen4	yellow4	chocolate3	DeepPink3	MediumOrchid1		gray56	gray93												
slate blue	olive drab	purple	MistyRose2	SlateGray4	SeaGreen1	gold2	firebrick1	DeepPink4	MediumOrchid2		gray57	gray94												
medium slate blue	dark khaki	medium purple	MistyRose3	LightSteelBlue1	SeaGreen2	gold3	firebrick2	HotPink1	MediumOrchid3		gray58	gray95												
light slate blue	khaki	thistle	MistyRose4	LightSteelBlue2	SeaGreen3	gold4	firebrick3	HotPink2	MediumOrchid4		gray59	gray97												
medium blue	pale goldenrod	snow2	azure2	LightSteelBlue3	PaleGreen1	goldenrod1	firebrick4	HotPink3	DarkOrchid1		gray60	gray98												
royal blue	light goldenrod yellow	snow3	azure3	LightSteelBlue4	PaleGreen2	goldenrod2	brown1	HotPink4	DarkOrchid2		gray61	gray99												
blue	light yellow	snow4	azure4	LightBlue1	PaleGreen3	goldenrod3	brown2	pink1	DarkOrchid3		gray62	gray62												
dodger blue	yellow	seashell2	SlateBlue1	LightBlue2	PaleGreen4	goldenrod4	brown3	pink2	DarkOrchid4		gray63	gray63												

- Voir la [page suivante](#) pour le code source qui génère cette application.

## Les formes, la classe Canvas

- La classe `Canvas` est un widget qui permet de dessiner des formes.
- Les positions sont définies par un plan cartésien.
  - L'origine est en haut à gauche.
  - La variation de l'axe `x` (largeur) est positive vers la droite.
  - La variation de l'axe `y` (hauteur) est positive vers le bas.



- La fonction `create_line((x1, y1), (x2, y2), ..., (xn, yn))` permet de dessiner une ou plusieurs lignes.

- Les tuples `(x, y)` représentent les positions d'un sommet.
- Le code suivant dessine une ligne qui part du point `(10, 10)` et qui se rend à la position `(10, 200)`.

```
import tkinter as tk
root = tk.Tk()
root.geometry("280x280")

ligne = tk.Canvas(root, background='white')
ligne.create_line((10, 10), (200, 10))

ligne.grid()
root.mainloop()
```

- Le code suivant créer un triangle avec une base de 50 pixels et une hauteur de 50 pixels.

```
import tkinter as tk
root = tk.Tk()
root.geometry("280x280")

ligne = tk.Canvas(root, background='white')
ligne.create_line((10, 60), (60, 60), (35, 10), (10, 60), fill="blue")

ligne.grid()
root.mainloop()
```

- Il est aussi possible de créer un rectangle :

```
carre = tk.Canvas(root, background='white')
carre.create_rectangle((10,10), (50,50))
```

- Il est aussi possible de lui ajouter une couleur de fond :

```
carre = tk.Canvas(root, background='white')
carre.create_rectangle((10,10), (50,50), fill='red')
```

- Ainsi que des bordures :

```
carre = tk.Canvas(root, background='white')
carre.create_rectangle((10,10), (50,50), width=3, outline='green')
```

- Pour ce simplifier la vie, nous pouvons utiliser la bibliothèque **C31Geometry**.

## La bibliothèque C31Geometry

- Cette bibliothèque fournit une série de classe qui permet de dessiner plus facilement des formes de base.
  - La bibliothèque possède la classe `Forme`. Cette dernière est inutilisable. Il s'agit du comportement commun à toutes les classes de `C31Geometry`.
- Voici un exemple complet d'utilisation. Son fonctionnement sera expliqué après.

```
import tkinter as tk
```

```

from functools import partial
from c3lGeometry import *

def foo(line, multiplicateur, root, event):
    line.resize(multiplicateur)
    line.draw()
    #root.update()
def bar(line, add, root, event):
    line.localRotate(add)
    line.draw()
    #root.update()

root = tk.Tk()
root.title("Les canvas en Tkinter")
root.geometry("280x280")

btn_plus = tk.Button(root, text="Augmenter")
btn_minus = tk.Button(root, text="Diminuer")

btn_plus.grid()
btn_minus.grid()

#####
# Zone Canevas
#####

canvas = tk.Canvas(root, background='white')

ligne = Ligne(canvas, Point(50, 50), 200, 0)
#ligne = Rectangle(canvas, Point(100, 100), 50, 25)
#ligne = Carre(canvas, Point(100, 155), 50)
#ligne = Cercle(canvas, Point(100, 100), 50)
#ligne = Croix(canvas, Point(100, 100), 50)

ligne.draw()
canvas.grid()
canvas.bind("<ButtonPress-1>", partial(foo, ligne, 1.1, root))
canvas.bind("<ButtonPress-3>", partial(foo, ligne, 0.1, root))
root.bind("<KeyPress-e>", partial(bar, ligne, 1, root))
root.bind("<KeyPress-q>", partial(bar, ligne, -1, root))
root.mainloop()

```

## Fonctionnement de base (commune à toutes les classes)

- Pour toutes les classes de la bibliothèque, vous devez créer, au préalable, un widget `Canvas`. Pour chaque constructeur, il s'agit du premier paramètre à fournir.

```

canvas = tk.Canvas(root, background='white')
ligne = Ligne(canvas, ...)

```

- Les objets `Point` représentent les coordonnées dans la bibliothèque. À chaque fois qu'une position ou un déplacement est demandé, vous devez fournir un objet de type `Point`.
  - L'objet de type `Point` gère les opérations vectorielles.
  - Le constructeur de `Point` prend deux paramètres : sa position en `x` et sa position en `y`.



- Les longueurs sont exprimées par des **entiers** ou des **nombre flottants**.
- Les angles sont exprimés en **degrés**.
  - **Attention !** N'exprimez jamais les angles en radians. La bibliothèque effectue automatiquement les changements.
- La bibliothèque réalise toujours la vérification de type.
- Pour afficher une forme, vous devez toujours appeler sa fonction `draw()`.
  - Pour le premier affichage.
  - Pour les mises à jour (déplacement, rotation, etc.)
- Dans la version actuelle, les mises à jour ne sont pas supportées.

## La méthode `resize`

- Cette méthode, commune à toutes les classes, permet de modifier la grosseur d'un objet sans affecter la position de son centre.
- Elle prend un **facteur multiplicateur**. C'est-à-dire une valeur par laquelle la taille sera augmentée ou diminuée.
  - Ce facteur est toujours un entier ou un nombre flottant **positif**.
  - Un facteur plus grand que **1** augmente la grosseur.
  - Un facteur plus petit que **1** diminue la grosseur.
  - Exemple :
    - Avec un facteur de **2**, la forme sera deux fois **plus grosse**.
    - Avec un facteur de **0.5**, la forme sera deux fois **plus petite**.

```
forme.resize(multiplicateur)
```

## La méthode `translate`

- Cette méthode permet de déplacer l'origine de la forme.
- Cette méthode prend un type `Point` en paramètre.
- La valeur du paramètre détermine le déplacement à faire.
  - C'est-à-dire que la valeur du paramètre sera ajoutée à l'origine.

```
deplacement = Point(x, y)
forme.translate(deplacement)
```

## La classe `Ligne`

- Cette classe permet de dessiner une ligne.
- Le constructeur prend quatre paramètres (en plus de canvas) :
  - `origine` : Coordonnée de départ de la ligne.
  - `distance` : Longueur, sur l'axe de la ligne, de cette dernière.
  - `angle` : rotation de la ligne.
  - `fill` : couleur de la ligne (par défaut **noir**).

```
ligne = Ligne(canvas, origine, distance, angle = 0, fill = 'black')
```

## La classe `Rectangle`

- Cette classe permet de dessiner un rectangle.
- Le constructeur prend six paramètres (en plus de canvas) :
  - `origine` : Coordonnée du centre du rectangle.
  - `largeur` : Longueur sur l'axe `x` de votre rectangle.
  - `hauteur` : Longueur sur l'axe `y` de votre rectangle.
  - `fill` : couleur du fond de la forme (par défaut **transparent**).
  - `outline` : couleur de la bordure (par défaut **noir**).
  - `width` : Épaisseur<sup>1</sup> de la bordure.

```
rectangle = Rectangle(canvas, origine, largeur, hauteur, fill = '',  
outline='black', width=1)
```

## La classe `Carre`

- Cette classe permet de dessiner un carré.
- Le constructeur prend cinq paramètres (en plus de canvas) :
  - `origine` : Coordonnée du centre du carré.
  - `largeur` : Longueur d'un côté de votre carré.
  - `fill` : couleur du fond de la forme (par défaut **transparent**).
  - `outline` : couleur de la bordure (par défaut **noir**).
  - `width` : Épaisseur<sup>1</sup> de la bordure.

```
carre = Carre(canvas, origine, largeur, fill = '', outline='black', width=1)
```

## La classe `Cercle`

- Cette classe permet de dessiner un cercle.
- Le constructeur prend cinq paramètres (en plus de canvas) :
  - `origine` : Coordonnée du centre du cercle.
  - `rayon` : Longueur du rayon de votre cercle.
  - `fill` : couleur du fond de la forme (par défaut **transparent**).
  - `outline` : couleur de la bordure (par défaut **noir**).
  - `width` : Épaisseur<sup>1</sup> de la bordure (par défaut **1px**).

```
cercle = Cercle(canvas, origine, rayon, fill = '', outline='black', width=1)
```

## La classe `Croix`

- Cette classe permet de dessiner une croix (X).
- Le constructeur prend cinq paramètres (en plus de canvas) :
  - `origine` : Coordonnée du centre de la croix.
  - `largeur` : Longueur de la croix autant à l'horizontale qu'à la verticale.
  - `fill` : couleur du fond de la forme (par défaut **transparent**).
  - `width` : Épaisseur<sup>1</sup> de la bordure (par défaut **1px**).

```
croix = Croix(canvas, origine, largeur, fill = 'black', width = 1)
```

## La classe AfterEvent

- La classe permet d'exécuter une fonction de votre choix à toutes les **n** millisecondes.
- Pour utiliser la classe, il suffit de lui donner une fonction, sans paramètre, à son constructeur et le délai entre deux exécutions en milliseconde.
  - Pour rappel, **1000 millisecondes correspondent à 1 seconde**\*\*.
- Pour lancer la boucle d'exécution, il suffit d'appeler la fonction `start()`.
  - Votre méthode sera exécutée après le délai mentionné dans le constructeur.
  - Pour exécuter immédiatement votre fonction, utilisez la fonction `startImmediately()`.
    - Par la suite, votre boucle sera exécutée après le délai mentionné dans le constructeur.

```
def evenement() :  
    print('Ceci est un exemple.')
```

```
e = AfterEvent(root, evenement, 500)  
# La fonction evenement sera exécutée toutes les demi-secondes.  
e.start()  
# La fonction evenement sera exécuté dans exactement un demi-seconde.
```

---

1. L'épaisseur est toujours exprimée par un nombre entier. [↩](#) [↩](#) [↩](#) [↩](#)