Jusqu'à présent, l'ensemble des exemples que nous avons traités dans les différents sujets traités se faisait uniquement en mode console. Nous profitons de cette étude pour élaborer des interfaces graphiques (fenêtres, boutons, zones de saisie, etc.) à l'aide de la bibliothèque **« tkinter »** qui est fournie par défaut avec Python.

Ce n'est certainement pas la plus sophistiquée puisque bon nombre de fonctionnalités n'existent pas, mais elle nous permettra d'élaborer facilement des applications fenêtrées rudimentaires de tout genre.

Nous aborderons systématiquement une approche orienté objet. Ce choix nous permet de créer nos propres composants spécifiques en héritant de ceux existants. Nous apporterons ainsi les modifications justes nécessaires sans tout réinventer. Par ailleurs, nous évitons, pour la composition de la fenêtre principale, de déclarer systématiquement plein de variables globales pour l'élaboration des méthodes qui gèrent la partie événementielle.

LES COMPOSANTS GRAPHIQUES DE TKINTER – LES WIDGETS

Comme toute interface graphique, il existe un certain nombre de composants graphiques préfabriqués, des **« widgets »** pour composer notre fenêtre principale avec tous les éléments nécessaires pour résoudre l'application que nous souhaitons mettre en œuvre. Voici, ci-dessous une liste non exhaustif des widgets les plus utilisés.

Widget	Caractéristiques associées
Tk	*Fenêtre principale d'une application python avec la bibliothèque **« tkinter »**.*
Button	*Un bouton classique, à utiliser pour provoquer l'exécution d'une commande.*
Canvas	*Un espace pour dessiner des formes, comme des lignes, des ovales, des polygones, des rectangles, etc.*
CheckButton	*Une case à cocher qui peut prendre uniquement deux états distincts (la case est cochée ou non). Un clic sur ce widget provoque le changement d'état. À l'aide de ces types de bouton, l'utilisateur peut sélectionner plusieurs options à la fois.*
Entry	*Zone de saisie sur une ligne pour accepter des valeurs (par défaut des textes) provenant de l'utilisateur.*
Frame	*Une surface rectangulaire dans la fenêtre prévue pour disposer d'autres widgets afin que l'apparence globale soit plus agréable. Cette surface peut être colorée et décorée avec une bordure (pour définir des zones particulières).*
Label	*Un texte (un libellé) utilisé pour fournir une légende pour d'autres widgets. Il peut également contenir des images.*
Listbox	*Une liste de choix proposés à l'utilisateur. Il est possible de configurer cette liste de telle manière que ces différents choix soient présentés comme une série de **« boutons radio »** ou de **« cases à cocher »**.*
MenuButton	*Utilisé pour implémenter des menus déroulants dans votre application.*
Menu	*Utilisé pour fournir différentes commandes à l'utilisateur. Ces commandes sont contenues à l'intérieur du **« MenuButton »**.*
Message	*Permet d'afficher un texte. Ce widget est une variante du widget **« Label »**, qui permet d'adapter automatiquement le texte affiché à une certaine taille ou à un certain rapport **« largeur/hauteur »**.*
RadioButton	*Représente (par un point noir dans un petit cercle) une des valeurs d'une variable qui possède plusieurs options possibles. Cliquer sur un bouton radio donne la valeur correspondante à la variable et désactive automatiquement les autres boutons radios associés à cette variable. l'utilisateur peut ainsi sélectionner une seule option à la fois.*
Scale	*Widget curseur qui permet de faire varier de manière très visuelle la valeur d'une variable, en déplaçant avec la souris le curseur le long d'une règle.*
Scrollbar	*Ascenseur ou barre de défilement que nous associons à d'autres widgets **(Canvas, Entry, Listbox, etc.)** afin d'augmenter la surface de travail dans une zone très limitée.*
Text	*Utilisé pour afficher ou éditer du texte sur plusieurs lignes.*
TopLevel	*Affiche une autre fenêtre au premier plan en plus de la fenêtre principale d'application.*
SpinBox	*Zone de saisie qui peut être utilisée pour sélectionner une valeur par rapport à une fourchette de valeurs pré-établies.*
PanedWindow	*Conteneur qui peut contenir un nombre quelconque de panneaux, disposés horizontalement ou verticalement.*
tkMessageBox	*Boîte de messages, d'avertissement, etc.*

PREMIÈRE APPLICATION FENÊTRÉE

Afin de bien comprendre le mécanisme inhérent aux applications avec interface graphique, je vous propose de réaliser un premier projet qui implémente notre première application fenêtrée.

*Pour cela, nous allons reprendre un projet que nous avons déjà mis en œuvre en mode console, qui permettait de connaître la surface d'un disque et le volume d'une sphère à partir du rayon d'un cercle. Bien sûr, nous gardons la classe **« Cercle »** qui effectue automatiquement tous les calculs nécessaires. Seuls, la partie graphique va être rajoutée.*

cercle.py

```
from math import pi, sqrt, pow

class Cercle:
    # Constructeur
    def __init__(self, rayon=50): self.rayon = rayon

    # Définition du diamètre du cercle
    @property
    def diamètre(self): return 2 * self.rayon
    @diamètre.setter
    def diamètre(self, nouveau): self.rayon = nouveau//2
    # Définition de la surface d'un disque
```

```
    @property
    def surface(self): return pi * self.rayon**2
    @surface.setter
    def surface(self, nouvelle): self.rayon = sqrt(nouvelle/pi)

    # Définition du volume d'une sphère
    @property
    def volume(self): return 4/3 * pi * self.rayon**3
    @volume.setter
    def volume(self, nouveau): self.rayon = pow(3*nouveau/(4*pi), 1/3)
```

Nous avons éliminé les membres de classe, attribut et méthode, qui permettait de connaître le nombre de cercles créés. Ici, un seul cercle sera généré pour réaliser les calculs nécessaires à l'application.

saisiefloat.py

```
from tkinter import *

# Nouvelle classe de saisie prévue pour les nombres réels
class SaisieFloat(Entry):
    def __init__(self, conteneur, libellé, valeur=0.0, état=DISABLED):
        Label(conteneur, text=libellé, anchor=W, width=30).pack()
        self.__valeur = DoubleVar(conteneur, valeur)
        super().__init__(conteneur, state=état, textvariable=self.__valeur, justify='right', width=30)
        self.pack()

    @property
    def valeur(self): return self.__valeur.get()
    @valeur.setter
    def valeur(self, nouvelle): self.__valeur.set(nouvelle)
```

*Un deuxième module s'occupe de la mise en œuvre d'une zone de saisie personnalisée **« SaisieFloat »** qui permet de récupérer directement une valeur de type **« float »**. Lorsque vous désirez créer un widget personnalisé, la technique est très simple, il suffit d'hériter du composant avec lequel vous souhaitez rajouter d'autres fonctionnalités. Ainsi, vous n'avez pas tout à refaire. C'est vraiment l'intérêt de la programmation objet.*

*Ce nouveau composant est décrit à l'aide de d'une propriété **« valeur »** en lecture et en écriture et d'un constructeur **« __init__ »** qui nous permet de le façonner pour qu'il soit capable de manipuler des valeurs réelles et de le configurer pour que son apparence soit conforme à notre désir.*

*Ce qui est particulier ici, c'est que ce composant est en réalité composé de deux éléments, d'une part d'une étiquette **« Label »** et d'autre part de la zone d'édition **« Entry »** proprement dite qui sont placés l'un au dessus de l'autre.*

À propos de placement, il est possible de préciser quel est l'élément conteneur (autre composant) qui va s'occuper de le placer, ici ce sera bien entendu la fenêtre (si le conteneur est la fenêtre principale, vous n'êtes pas obligé de donner cette précision). C'est systématiquement le premier argument que nous devons préciser à chacun des objets que nous créons, quelle que soit leur nature.

*Le placement du composant proprement dit se fait à l'aide d'une méthode spécifique **« pack() »** qui positionne par défaut les composants les uns au-dessus des autres, ce qui convient parfaitement ici. Nous verrons plus tard qu'il existe bien entendu d'autres solutions alternatives.*

Lorsque nous créons nos widgets, chacun possède plein de paramètres dans la phase de construction qui permettent de régler finement l'apparence et le comportement de chacun. Pour cela, vous devez qualifier le paramètre qui vous intéresse en donnant l'argument correspondant. Nous ne les montrerons pas tous, mais voici ceux que j'ai pris en compte pour notre application :

*Dans le cas du **« Label »** nous précisons l'intitulé du texte **« text=libellé »**, le positionnement du texte par rapport à la dimension du composant, ici à l'ouest **« anchor=W »** et la dimension du composant **« width=30 »**.*

*Dans le cas de **« Entry »**, nous spécifions l'état du composant (par défaut désactivé – grisé) **« state=état »**, la variable qui correspond à la valeur à traiter **« textvariable=self.__valeur »**, la justification (le placement) de la valeur par rapport à toute la zone d'édition, ici à droite **« justify='right' »** et la dimension du composant **« width=30 »**.*

*Par défaut, une zone de saisie traite du texte, mais il possible, comme nous venons de le voir, de fournir un argument au paramètre **« textvariable »** correspondant au type à traiter. Il existe quatre classes spécialisées pour cela, chacune traitant d'un type spécifique, **« StringVar »** pour le type **« str »**, **« BooleanVar »** pour le type **« bool »**, **« IntVar »** pour le type **« int »** et **« DoubleVar »** pour le type **« float »**.*

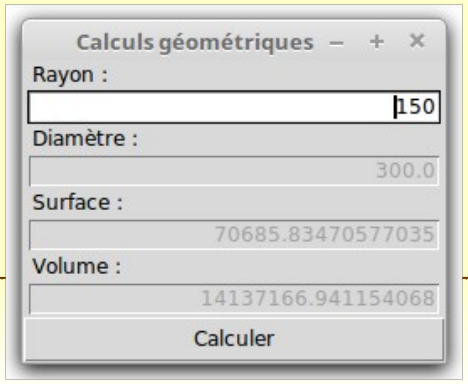
*L'utilisation de ces classes est très simple. Elles sont toutes capables de passer d'une chaîne de caractère, mode par défaut vers le type choisi, et vice versa. Elles possèdent chacune une méthode **« set »** pour soumettre une nouvelle valeur, et une méthode **« get »** pour récupérer la valeur en cours.*

Pour que cela soit relativement transparent et très facile à manipuler, nous avons prévu une propriété « valeur » qui nous permettra de réaliser tous les calculs nécessaires à notre application, notamment pour la fenêtre principale que nous allons maintenant décrire.

principal.py

```
from tkinter import *
from saisiefloat import SaisieFloat
from cercle import Cercle
# Fenêtre principale de l'application
```

```
class Fenêtre(Tk):
   def __init__(self, titre):
      super().__init__()
      self.title(titre)
      self.cercle = Cercle()
      self.placerComposants()
      self.mainloop()

   def placerComposants(self):
      self.rayon = SaisieFloat(self,  'Rayon :', self.cercle.rayon, état="normal")
      self.diamètre = SaisieFloat(self, 'Diamètre :', self.cercle.diamètre)
      self.surface = SaisieFloat(self, 'Surface :', self.cercle.surface)
      self.volume = SaisieFloat(self, 'Volume :', self.cercle.volume)
      Button(self, text='Calculer', command=self.calcul, width=28).pack()
      self.rayon.focus()

   def calcul(self):
      self.cercle.rayon = self.rayon.valeur
      self.diamètre.valeur = self.cercle.diamètre
      self.surface.valeur = self.cercle.surface
      self.volume.valeur = self.cercle.volume

# Programme principal
if __name__ == '__main__' :
   programme = Fenêtre('Calculs géométriques')
```



Lorsque vous désirez créer une nouvelle application fenêtrée, n'hésitez pas à créer une nouvelle classe, ici ***« Fenêtre »*** *qui hérite de la classe* ***« Tk »*** *qui représente une classe principale d'application que nous personnalisons pour répondre simplement à notre attente. Trois méthodes sont implémentées :*

- ***placerComposants()*** *: comme son nom l'indique, cette méthode s'occupe de placer tous les composants graphiques nécessaires à la vue de notre application. C'est à ce moment-là que nous créons tous les objets relatifs à la classe que nous venons de mettre en œuvre* ***« SaisieFloat »****.*
- ***calcul()*** *: une autre méthode spécialisée qui s'occupe uniquement de tous les calculs relatifs aux traitements demandés. C'est dans cette méthode que nous utilisons la propriété* ***« valeur »*** *de la classe* ***« SaisieFloat »*** *que nous venons d'évoquer. C'est aussi là que nous utilisons les compétences de la classe* ***« Cercle »*** *puisque c'est elle qui dispose de toutes les méthodes adaptées aux différents traitements.*
- ***__init__()*** *: constructeur qui permet dans l'ordre : de préciser un titre à la fenêtre, de créer l'attribut* ***« cercle »*** *qui s'occupe des calculs, d'activer la méthode* ***« placerComposants »*** *et surtout d'activer la gestion événementielle grâce à l'appel de la méthode* ***« mainloop »****.*

Cette méthode « mainloop » est fondamentale, elle permet de faire en sorte de tenir compte de tous les événements issus de la souris ou d'une touche du clavier. Elle attend une demande spécifique de clôture de l'application grâce au menu système de la fenêtre principale de l'application. Entre temps, si d'autres événements sont proposés, ils sont tout simplement pris en compte, c'est-à-dire que les méthodes ou les fonctions de rappel sont sollicitées suivant les cas de figure.

Justement, ici nous proposons un événement associé au clic du bouton ***« Calculer »****. Sur ce composant, grâce à l'étiquette* ***« command »****, nous associons la méthode* ***« calcul() »*** *de la classe* ***« Fenêtre »****. Ainsi, à chaque fois que l'utilisateur cliquera sur ce bouton, la méthode associée sera automatiquement appelée (méthode dite de rappel).*

Pour conclure, vous voyez que la fin du module possède une écriture bien particulière ***« if __name__ == '__main__" »****. Cela veut simplement dire que le programme principal de l'application commence à cet endroit-là. Ce programme comprend une seule ligne d'instruction qui consiste à créer l'objet* ***« programme »*** *relatif à la classe principale de l'application* ***« Fenêtre »****.*

Le modèle de programmation que nous venons de mettre en œuvre s'appelle un modèle « MVC » (Modèle-Vue-Contrôleur) qui permet de séparer les différents éléments, chacun ayant sa propre fonction. La « vue » comme son nom l'indique ne s'intéresse qu'à l'apparence (implémentée ici par la méthode « placeComposants() », le « modèle » ne s'intéresse qu'au traitement de fond (implémentée ici par la classe « Cercle », et le « contrôleur » assure la cohésion entre les deux grâce à la gestion événementielle inhérent à la structure de la classe «Tk» avec la méthode « mainloop » d'une part et l'étiquette « command » d'autre part.

CHOISIR LE POSITIONNEMENT DES COMPOSANTS INTERNES – POSITIONNEMENT AUTOMATIQUE

Le chapitre précédent nous a permis de bien comprendre les principes généraux de la conception d'une interface graphique. Nous allons nous intéresser maintenant au placement automatique des widgets à l'intérieur de la fenêtre (ou d'autres widgets). Il existe trois méthodes spécifiques qui sont automatiquement intégrées à chacun des widgets :

- ***pack()*** *: très souvent utilisé, ce gestionnaire de placement organise les widgets dans les blocs avant de les placer dans le widget parent. La taille de chacun des widgets sera différente suivant leur propre dimension et suivant d'autres critères possibles.*
- ***grid()****: ce gestionnaire place les widgets dans une grille dans le widget parent à l'image d'un tableau à deux dimensions. Lorsque vous rajouter un widget dans la fenêtre, vous devez alors préciser à quelle ligne et dans quelle colonne il doit être placé et éventuellement s'il prend plusieurs colonnes ou plusieurs lignes ou les deux.*
- ***place()*** *: ce gestionnaire de disposition organise les widgets en les plaçant dans une position précisée par le développeur dans le widget parent.*

POSITIONNER UN WIDGET RELATIVEMENT AVEC LA MÉTHODE « PACK() »

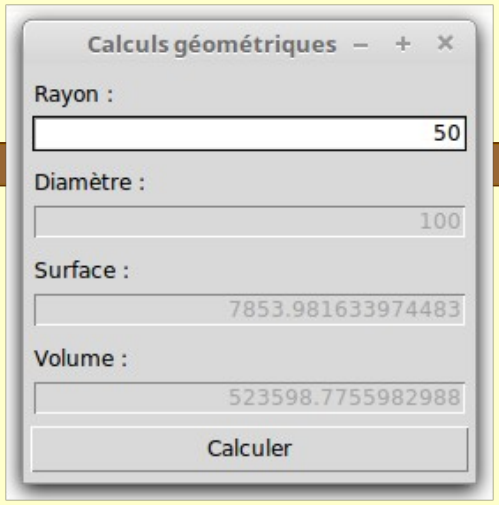
Reprenons le projet précédent en utilisant toujours la méthode **« pack() »** pour placer chacun des widgets dans la fenêtre principale, mais cette fois-ci nous proposons un certain nombre d'options intéressantes qui permettra d'éviter de donner systématiquement une même largeur **« width »** à tous les widgets que nous positionnons. Voici d'ailleurs quelques paramètres utiles que nous utiliserons au cours de notre étude :

- ***padx*** *et* ***pady*** *: propose une marge externe suivant l'axe des X et des Y autour du widget.*
- ***ipadx*** *et* ***ipady*** *: propose une marge interne suivant l'axe des X et des Y à l'intérieur du widget.*
- ***expand*** *: place des espaces supplémentaires au widget, la valeur doit être supérieure à* ***« 0 »*** *et vous précisez le nombre d'espace que vous souhaitez.*
- ***fill*** *: étend le composant au maximum jusqu'à atteindre le bord du widget parent, soit dans l'axe des* ***« X »****, soit dans l'axe des* ***« Y »****, soit les deux* ***« BOTH »*** *ou tout simplement aucune extension (par défaut* ***« default »****).*
- ***side*** *: propose un positionnement du widget dans le parent suivant les points cardinaux* ***« N, NE, E, SE, S, SW, W, NW »****.*

Voici ci-dessous une possibilité d'utilisation en prenant en compte quelques critères évoqués ci-dessus :

saisiefloat.py

```
from tkinter import *

# Nouvelle classe de saisie prévue pour les nombres réels
class SaisieFloat(Entry):
   def __init__(self, conteneur, libellé, valeur=0.0, état=DISABLED):
      Label(conteneur, text=libellé, anchor=SW).pack(fill=X, padx=4, pady=3, ipady=3)
      self.__valeur = DoubleVar(conteneur, valeur)
      super().__init__(conteneur, state=état, textvariable=self.__valeur, justify='right', width=30)
      self.pack(padx=5)

   @property
   def valeur(self): return self.__valeur.get()
   @valeur.setter
   def valeur(self, nouvelle): self.__valeur.set(nouvelle)
```



principal.py

```
from tkinter import *
from saisiefloat import SaisieFloat
from cercle import Cercle

# Fenêtre principale de l'application
class Fenêtre(Tk):
   def __init__(self, titre):
      super().__init__()
      self.title(titre)
      self.cercle = Cercle()
      self.placerComposants()
      self.mainloop()

   def placerComposants(self):
      self.rayon = SaisieFloat(self,  'Rayon :', self.cercle.rayon, état="normal")
      self.diamètre = SaisieFloat(self, 'Diamètre :', self.cercle.diamètre)
      self.surface = SaisieFloat(self, 'Surface :', self.cercle.surface)
      self.volume = SaisieFloat(self, 'Volume :', self.cercle.volume)
      Button(self, text='Calculer', command=self.calcul).pack(fill=X, padx=3, pady=5)
      self.rayon.focus()

   def calcul(self):
      self.cercle.rayon = self.rayon.valeur
      self.diamètre.valeur = self.cercle.diamètre
      self.surface.valeur = self.cercle.surface
      self.volume.valeur = self.cercle.volume

# Programme principal
if __name__ == '__main__' :
   programme = Fenêtre('Calculs géométriques')
```

POSITIONNER LE WIDGET DANS UNE GRILLE AVEC LA MÉTHODE « GRID() »

Comme c'est le cas dans notre projet, lorsque vous avez à constituer une interface graphique sous l'aspect d'un formulaire, il est souvent plus judicieux de placer les composants alignés les uns par rapport aux autres en prenant plusieurs colonnes, ce qui correspond plus à un positionnement sous forme de grille. C'est ce que propose la méthode **« grid() »** dont voici quelques paramètres intéressants :

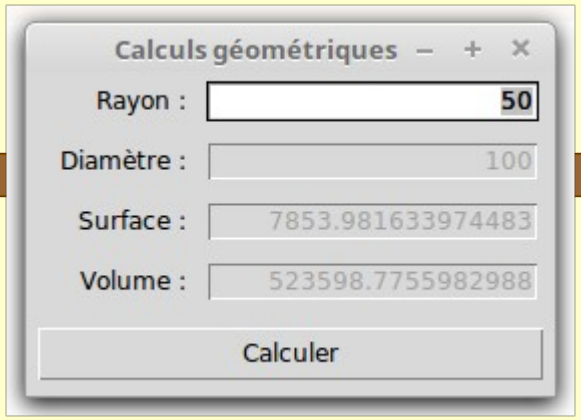
- ***padx*** *et* ***pady*** *: propose une marge externe suivant l'axe des X et des Y autour du widget.*
- ***ipadx*** *et* ***ipady*** *: propose une marge interne suivant l'axe des X et des Y à l'intérieur du widget.*

- ***column*** *: numéro de la colonne où vous souhaitez placer votre widget en partant de zéro. Sa valeur par défaut est* ***« 0 »****.*
- ***row*** *: Le numéro de ligne où vous souhaitez placer votre widget en partant de zéro. Sa valeur par défaut est le numéro de de la première ligne inoccupée.*
- ***columnspan*** *: Normalement un widget occupe seulement une cellule. Cependant, grâce à ce paramètre, vous pouvez regrouper plusieurs cellules d'une ligne en indiquant le nombre de cellules que vous désirez couvrir.*
- ***rowspan*** *: Normalement un widget occupe seulement une cellule. Cependant, grâce à ce paramètre, vous pouvez regrouper plusieurs cellules d'une colonne en indiquant le nombre de cellules que vous désirez couvrir.*
- ***sticky*** *: Cette option détermine la façon de distribuer l'espace inoccupé par un widget à l'intérieur d'une cellule. Si vous ne donnez aucune valeur à l'attribut* ***« sticky »****, le comportement par défaut est de centrer le widget dans sa cellule. Vous pouvez positionner le widget dans un des coins de la cellule en indiquant* ***« sticky='ne' »*** *(nord-est: en haut à droite),* ***« 'se' »*** *(en bas à droite),* ***« 'sw' »*** *(en bas à gauche), ou* ***« 'nw' »*** *(en haut à gauche). Vous pouvez centrer le widget contre l'un des bords de la cellule en utilisant* ***« sticky='n' »*** *(centré en haut),* ***« 'e' »*** *(centré à droite),* ***« 's' »*** *(centré en bas), ou* ***« 'w' »*** *(centré à gauche). Utilisez* ***« sticky='ns' »*** *pour l'étirer verticalement tout en le laissant centré horizontalement. Utilisez* ***« sticky='ew' »*** *pour l'étirer horizontalement tout en le laissant centré verticalement. Utilisez* ***« sticky='nesw' »*** *pour l'étirer dans les deux directions afin de remplir la cellule. Les autres combinaisons fonctionneront aussi. Par exemple,* ***« sticky='nsw' »*** *l'étirera verticalement en le plaçant contre le bord gauche de la cellule.*

Voici ci-dessous une possibilité d'utilisation en prenant en compte les critères évoqués ci-dessus :

saisiefloat.py

```
from tkinter import *

# Nouvelle classe de saisie prévue pour les nombres réels
class SaisieFloat(Entry):

   # Attribut de classe
   ligne = 0

   def __init__(self, conteneur, libellé, valeur=0.0, état=DISABLED):
      Label(conteneur, text=libellé, anchor=W).grid(row=SaisieFloat.ligne, column=0, sticky=E)
      self.__valeur = DoubleVar(conteneur, valeur)
      super().__init__(conteneur, state=état, textvariable=self.__valeur, justify='right')
      self.grid(row=SaisieFloat.ligne, column=1, pady=5)
      SaisieFloat.ligne += 1

   @property
   def valeur(self): return self.__valeur.get()
   @valeur.setter
   def valeur(self, nouvelle): self.__valeur.set(nouvelle)
```



principal.py

```
from tkinter import *
from saisiefloat import SaisieFloat
from cercle import Cercle

# Fenêtre principale de l'application
class Fenêtre(Tk):
   def __init__(self, titre):
      super().__init__()
      self.title(titre)
      self.cercle = Cercle()
      self.placerComposants()
      self.mainloop()

   def placerComposants(self):
      self.rayon = SaisieFloat(self,  'Rayon :', self.cercle.rayon, état="normal")
      self.diamètre = SaisieFloat(self, 'Diamètre :', self.cercle.diamètre)
      self.surface = SaisieFloat(self, 'Surface :', self.cercle.surface)
      self.volume = SaisieFloat(self, 'Volume :', self.cercle.volume)
      Button(self, text='Calculer', command=self.calcul, padx=100).grid(row=SaisieFloat.ligne, columnspan=2, padx=5, pady=7)
      self.rayon.focus()

   def calcul(self):
      self.cercle.rayon = self.rayon.valeur
      self.diamètre.valeur = self.cercle.diamètre
      self.surface.valeur = self.cercle.surface
      self.volume.valeur = self.cercle.volume

# Programme principal
if __name__ == '__main__' :
   programme = Fenêtre('Calculs géométriques')
```

PRENDRE EN COMPTE PLUSIEURS ÉVÉNEMENTS ASSOCIÉS AU CLAVIER ET À LA SOURIS

Toujours en relation avec notre projet, il serait intéressant de permettre le lancement du calcul dès que l'utilisateur appuie sur la touche **« Entrée »**, plutôt que d'être obligé de prendre systématiquement la souris et cliquer sur le bouton **« Calculer »**.

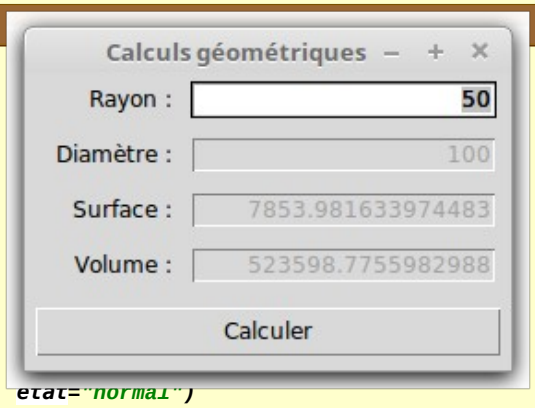
Grâce à cette approche, il est possible de lancer un événement particulier avec plusieurs sources différentes. Ensuite, dès que nous avons besoin du clavier, il est plus ergonomique de rester avec, nous supprimerons donc le bouton ultérieurement.

Pour activer un nouvel événement, il suffit d'utiliser la méthode « bind() » du composant qui gère l'événement, en spécifiant la touche du clavier souhaité, suivi du nom de la fonction ou de la méthode qui va traiter l'événement. Par contre, cette fonction ou cette méthode doit posséder le paramètre « event » avec lequel vous pourrez récupérer des informations supplémentaires pour un traitement particulier.

*Le nom des touches du clavier possède un formatage spécifique, vous devez systématiquement mettre son nom entre braquets **« <> »**. Voici quelques noms de touche que vous devez respecter :*

Widget	Caractéristiques associées
<Button-1>	Clic bouton gauche de la souris.
<Button-2>	Clic bouton milieu de la souris.
<Button-3>	Clic bouton droit de la souris.
<Double-Button-1>	Double-clic bouton gauche de la souris.
<Double-Button-2>	Double-clic bouton droit de la souris.
<KeyPress> <Key>	Action d'appuyer sur une touche quelconque du clavier.
<KeyPress-a>	Appui sur la touche **« a »** du clavier (minuscule).
<KeyPress-A>	Appui sur la touche **« A »** du clavier (majuscule).
<Return>	Appui sur la touche **« Entrée »** du clavier (validation).
<Escape>	Appui sur la touche **« Echap »** du clavier (annulation).
<Up>	Pression sur la flèche directionnelle haut.
<Down>	Pression sur la flèche directionnelle bas.
<ButtonRelease>	Action de relâcher un bouton de la souris.
<Motion>	Prise en compte du mouvement de la souris.
<B1-Motion>	Prise en compte du mouvement de la souris avec le maintien du bouton gauche (glisser-déposer).
<Enter>	Le curseur de la souris pénètre à l'intérieur du widget.
<Leave>	Le curseur de la souris sort de la surface délimitée par le widget.
<Configure>	Redimensionnement de la fenêtre.
<Map> <Unmap>	Ouverture et iconification de la fenêtre.
<MouseWheel>	Utilisation de la molette de la souris.
<Configure>	L'utilisateur a modifié la taille d'un widget, par exemple en déplaçant un coin ou un côté de la fenêtre.

principal.py




```
from tkinter import *
from saisiefloat import SaisieFloat
from cercle import Cercle

class Fenêtre(Tk):
   def __init__(self, titre):
      super().__init__()
      self.title(titre)
      self.cercle = Cercle()
      self.placerComposants()
      self.mainloop()

   def placerComposants(self):
      self.rayon = SaisieFloat(self,  'Rayon :', self.cercle.rayon, etat="normal")
      self.rayon.select_range(0, END) # sélection entière du nombre
      self.rayon.bind('<Return>', self.calcul)
      self.diamètre = SaisieFloat(self, 'Diamètre :', self.cercle.diamètre)
      self.surface = SaisieFloat(self, 'Surface :', self.cercle.surface)
      self.volume = SaisieFloat(self, 'Volume :', self.cercle.volume)
      Button(self, text='Calculer', command=self.calcul, padx=100).grid(row=SaisieFloat.ligne, columnspan=2, padx=5, pady=7)
      self.rayon.focus()

   def calcul(self, event):
      self.cercle.rayon = self.rayon.valeur
      self.diamètre.valeur = self.cercle.diamètre
      self.surface.valeur = self.cercle.surface
      self.volume.valeur = self.cercle.volume
```

```
# Programme principal
if __name__ == '__main__' :
   programme = Fenêtre('Calculs géométriques')
```

*Grâce à la méthode **« select-range() »**, il vous est possible de sélectionner une partie d'une zone d'édition en proposant le début et la fin de la sélection. La constante **« END »** spécifie la position juste après le dernier caractère (ou valeur numérique) introduite dans la zone de saisie.*

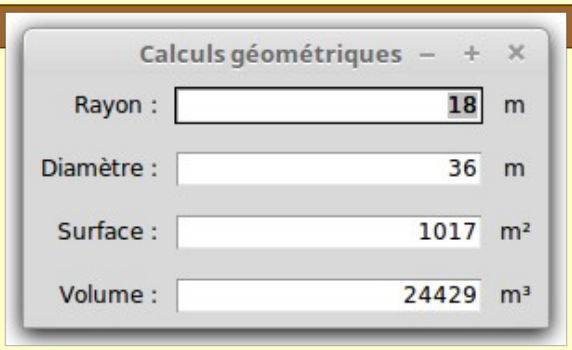
TOUTES LES ZONES DE SAISIE SONT ÉDITABLES

Pour finir sur ce projet, toutes les zones de saisie sont actives et propose un calcul circonstancié. Nous n'avons plus besoin du bouton et le résultat sera toujours présenté sont forme entière, même si les calculs sont réalisés avec des nombres réels. Ainsi, la classe **« SaisieFloat »** devient **« SaisieInt »**.

saisieint.py

```
from tkinter import *

# Nouvelle classe de saisie prévue pour les nombres entiers
class SaisieInt(Entry):

   # Attribut de classe
   ligne = 0

   def __init__(self, conteneur, libellé, valeur=0.0, unité='m'):
      Label(conteneur, text=libellé).grid(row=SaisieInt.ligne, column=0, sticky=E, padx=7)
      self.__valeur = IntVar(conteneur, int(valeur))
      super().__init__(conteneur, textvariable=self.__valeur, justify='right')
      self.select_range(0, END)
      self.grid(row=SaisieInt.ligne, column=1, pady=7)
      Label(conteneur, text=unité).grid(row=SaisieInt.ligne, column=2, padx=7)
      SaisieInt.ligne += 1

   @property
   def valeur(self): return self.__valeur.get()
   @valeur.setter
   def valeur(self, nouvelle): self.__valeur.set(int(nouvelle))
```

principal.py




```
from tkinter import *
from saisieint import SaisieInt
from cercle import Cercle

# Fenêtre principale de l'application
class Fenêtre(Tk):
   def __init__(self, titre):
      super().__init__()
      self.title(titre)
      self.cercle = Cercle()
      self.placerComposants()
      self.mainloop()

   def placerComposants(self):
      self.rayon = SaisieInt(self,  'Rayon :', self.cercle.rayon)
      self.rayon.bind('<Return>', self.duRayon)
      self.diamètre = SaisieInt(self, 'Diamètre :', self.cercle.diamètre)
      self.diamètre.bind('<Return>', self.duDiamètre)
      self.surface = SaisieInt(self, 'Surface :', self.cercle.surface, 'm²')
      self.surface.bind('<Return>', self.deLaSurface)
      self.volume = SaisieInt(self, 'Volume :', self.cercle.volume, 'm³')
      self.volume.bind('<Return>', self.duVolume)
      self.rayon.focus()
      self.rayon.select_range(0, END)

   def duRayon(self, event):
      self.cercle.rayon = self.rayon.valeur
      self.diamètre.valeur = self.cercle.diamètre
      self.surface.valeur = self.cercle.surface
      self.volume.valeur = self.cercle.volume

   def duDiamètre(self, event):
      self.cercle.diamètre = self.diamètre.valeur
      self.rayon.valeur = self.cercle.rayon
      self.surface.valeur = self.cercle.surface
      self.volume.valeur = self.cercle.volume
```

```
    def deLaSurface(self, event):
        self.cercle.surface = self.surface.valeur
        self.rayon.valeur = self.cercle.rayon
        self.diamètre.valeur = self.cercle.diamètre
        self.volume.valeur = self.cercle.volume

    def duVolume(self, event):
        self.cercle.volume = self.volume.valeur
        self.rayon.valeur = self.cercle.rayon
        self.surface.valeur = self.cercle.surface
        self.diamètre.valeur = self.cercle.diamètre

# Programme principal
if __name__ == '__main__' :
    programme = Fenêtre('Calculs géométriques')
```

LES IMAGES ET LES CANEVAS - « CANVAS »

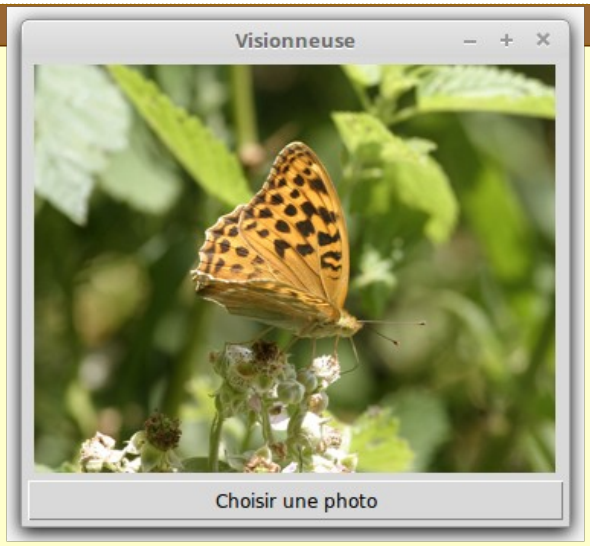
Un widget important est la classe **« Canvas »** qui représente un canevas, c'est-à-dire une feuille dessin (ou un calque) sur laquelle vous allez pouvoir tracer (dessiner) un certain nombre d'éléments, comme des formes complexes, des images, du texte et même d'autres widgets.

*Dans le projet qui suit, nous élaborons l'application **« Visionneuse »** qui permet d'afficher des photos, la totalité de la photo est toujours visible quelque soit la taille de la fenêtre. À ce sujet, si vous redimensionner votre fenêtre, la photo s'ajuste automatiquement en respectant le rapport entre la largeur et la hauteur de la photo originale.*

*Comme d'habitude, nous séparons notre application en plusieurs objets ce qui facilite grandement le travail. La classe importante est la classe **« Vignette »** qui hérite de la classe **« Canvas »** qui s'occupe de la gestion de la photo, c'est-à-dire la récupérer depuis le disque dur (ou autre) de votre ordinateur, de la tracer ensuite correctement dans toute la surface du canevas actuellement visible en s'auto-adaptant et en respectant le ratio de la photo.*

*La deuxième classe **« Fenêtre »** est la classe principale de l'application qui dispose de la vignette que nous venons d'évoquer qui prend la quasi totalité de la surface interne de la fenêtre et d'un bouton situé en bas qui permet de lancer la boîte de dialogue de sélection de fichiers pour choisir la (ou les) photos à visionner.*

vignette.py



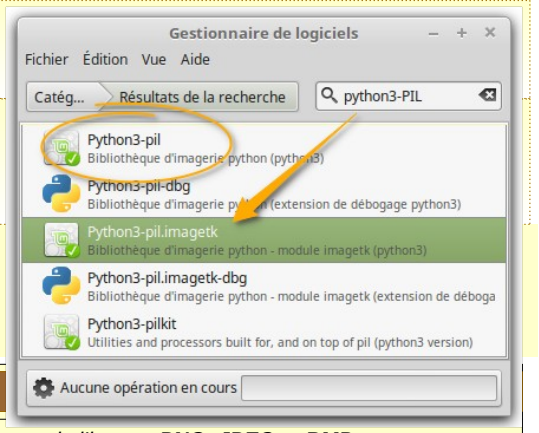

```
from tkinter import *
from tkinter.filedialog import *
from PIL import Image, ImageTk

# Création d'un vignette à partir d'un Canvas
class Vignette(Canvas):
    def __init__(self, fenêtre):
        super().__init__(fenêtre)
        self.pack(fill=BOTH, expand=1, padx=3)
        self.bind('<Configure>', self.retailler)
        self.fichier=''

    def retailler(self, event):
        self.afficher()

    def changer(self):
        self.fichier = askopenfilename()
        self.afficher()

    def afficher(self):
        if not len(self.fichier) == 0:
            image = Image.open(self.fichier)
            image.thumbnail((self.winfo_width(), self.winfo_height()))
            self.image = ImageTk.PhotoImage(image)
            self.create_image(5, 5, image=self.image, anchor='nw')
```

*La classe **« Vignette »** dispose, en plus du constructeur de trois méthodes qui s'occupent de la gestion événementielle :*

- *La méthode **« retailler() »** est lancée à chaque fois que nous redimensionnons la fenêtre.*
- *La méthode **« changer() »** est appelée lorsque l'utilisateur choisi une nouvelle photo. Cette méthode lance alors le sélecteur de fichiers qui permet de sélectionner la photo désirée.*
- *La méthode **« afficher() »** est appelée par les deux autres méthodes précédentes. Elle récupère le fichier photo sélectionné et le dessine (l'affiche) sur la quasi totalité du canevas sous forme de vignette en respectant le rapport largeur-hauteur de l'image originale.*

Par défaut, les widgets intégrés dans « tkinter » manipulent plutôt des images au format PNG ou GIF. Si vous souhaitez travailler avec des photos, le format est généralement du JPEG. Du coup, pour traiter des photos, vous devez prendre en compte le module « PIL » (bibliothèque PILLOW) qui possède des classes spécifiques pour ce format de fichier et qui prévoit des traitements très sophistiqués. Ce module possède deux classes importantes :

- ***Image*** *: capable de récupérer ou de sauvegarder un fichier photo au format* ***« JPEG »*** *et peut ensuite réaliser tout un tas de traitements classiques pour les photos : fonction miroir, histogramme, retailler, découper, solariser, etc.*
- ***ImageTk*** *: dont l'objectif principal et de permettre d'associer les objets issus de la classe* ***« Image »*** *afin qu'ils puissent être intégrés directement, comme pour les images de type* ***« PNG »****, dans n'importe quel widget du module* ***« tkinter »****.*



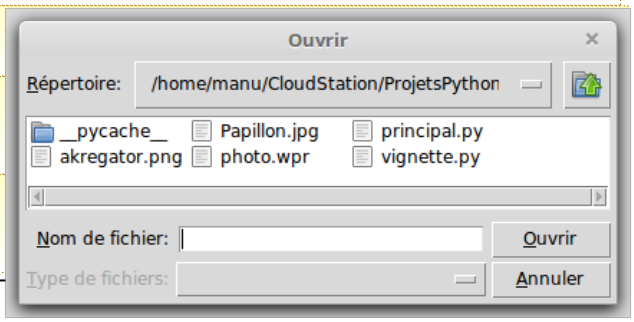
Attention, il est impératif que le module « PIL » soit installé dans votre système d'exploitation, notamment ce qui correspond à la classe « ImageTk » qui n'est généralement pas installé par défaut. Sur Linux Debian et dérivés :

sudo apt-get install python3-pil python3-pil-imagetk

Attributs et Méthodes associées à la classe « PIL.Image »	
format	*Attribut chaîne de caractères qui nous renseigne sur le format de l'image,* ***PNG****,* ***JPEG*** *et* ***BMP****.*
mode	*Attribut chaîne de caractères qui nous renseigne sur le codage des couleurs de l'image,* ***RGB****,* ***CMYK****, etc.*
size	*Attribut tuple qui retourne la dimension de l'image* ***(largeur, hauteur)****.*
width, height	*Deux attributs qui nous renseignement respectivement sur la largeur et la hauteur de l'image.*
open(fichier, mode='r')	*Permet d'ouvrir une image de tout type de format et cré un objet de type* ***« Image »****. La lecture du fichier ne se fait que si un traitement est spécifié par la suite en évitant ainsi d'utiliser trop de ressources matérielles.*
save(fichier, format)	*Sauvegarde l'image suivant le format désiré,* ***PNG****,* ***JPEG*** *ou* ***BMP****.*
show()	*Visualise directement l'image chargée en mémoire.*
split()	*Retourne un tuple composé des trois images séparées avec les trois couleurs fondamentales* ***RGB****.*
thumbnail(taille, algorithme=3)	*Crée une vignette en tenant compte du ratio de l'image originale. Par défaut, c'est l'algorithme qui donne le meilleur résultat qui est préconisé puisque le résultat est une image généralement beaucoup plus réduite. Les constantes possibles sont* ***NEREAST, BILINEAR, BICUBIC*** *(par défaut) et* ***LANCLOZ****. Attention, c'est l'objet image lui-même qui est transformé*
rotate(angle)	*Crée une nouvelle image qui a subit une rotation de l'image originale en précisant un angle en degré.*
transpose(méthode)	*Génère une nouvelle image qui subit une transformation géométrique suivant le choix spécifié dans l'argument. Miroir droite-gauche :* ***FLIP_LEFT_RIGHT****, miroir haut-bas* ***FLIP_TOP_BOTTOM****, rotations* ***ROTATE_90, ROTATE_180*** *et* ***ROTATE_270****.*
resize(taille, algorithme=0)	*Génère une nouvelle image avec les dimensions précisés en argument. Attention, c'est vous qui décidez de la largeur et de la hauteur, le ratio n'est pas spécialement respecté. Il est possible de préciser l'algorithme souhaité. Par défaut, c'est celui qui prend le moins de temps de traitement, donc le moins performant :* ***NEREAST*** *(par défaut),* ***BOX, BILINEAR, HAMMING, BICUBIC*** *et* ***LANCLOZ.***
filter(filtre)	*Génère une nouvelle image qui est le résultat de d'application d'un ou plusieurs filtres, à l'image de ce que nous pouvons faire sur les logiciels de traitement d'image. Nous utilisons dans ce cas là les constantes issues de la classe* ***« ImageFilter »****, soit : flou glaussien -* ***BLUR****, effet de dessin en favorisant contours uniquement –* ***CONTOUR****, accentuation fine –* ***DETAIL****, accentuation prononcée* ***EDGE_ENHANCE****, accentuation très forte –* ***EDGE_ENHANCE_MORE****, embossage* ***EMBOSS****, effet de dessin solarisé –* ***FIND_EDGE****, lissage (contours adoucis) plus ou moins fort –* ***SMOOTH – SMOOTH_MORE****, rendre plus net (accentuation)* ***SHARPEN****.*
copy()	*Génère une nouvelle image identique à l'originale.*
crop(boîte)	*Génère une partie de l'image en précisant un tuple spécifiant les distances respectives par rapport aux bords de l'image originale,* ***(gauche, haut, droite, bas)****.*
paste(logo, boîte)	*Génère une image qui correspond à la fusion de l'image originale avec une autre proposée en argument en spécifiant également le rectangle où sera positionnée la nouvelle image sous la forme du tuple* ***(gauche, haut, droite, bas)****.*
convert(couleur)	*Génère une image en sélectionnant un autre mode de couleur par rapport à l'image originale. Les trois modes de couleurs supportés sont* ***L*** *(nuance de gris),* ***RGB*** *et* ***CMYK****.*
histogram()	*Retourne l'histogramme de l'image sous la forme d'une liste de valeurs :* ***768*** *valeurs pour le mode* ***RGB****.*

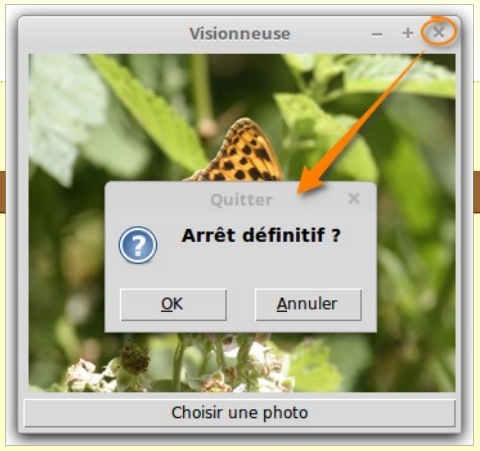
Le sous-module ***« tkinter.filedialog »*** *fournit des fonctions qui génère des boîtes de dialogue qui permettent à l'utilisateur de choisir un fichier. Une fois que l'utilisateur a fait son choix, la fonction associée retourne le chemin complet du fichier. Il existe deux fonctions spécifiques,* ***« askopenfilename() »*** *pour ouvrir le fichier sélectionné et* ***« asksaveasfilename() »*** *pour sauvegarder des valeurs dans le fichier désigné (nouveau ou existant). Ces deux fonctions disposent de paramètres identiques :*

- ***defaultextension*** *: définit l'extension qui sera automatiquement placée à la suite du nom du fichier, ex :* ***« .jpg »****.*
- ***filetypes*** *: précise quels sont les types de fichiers attendus. Vous devez précisez la liste des extensions possibles, ex :* ***[(« Photos », « *.jpg »), (« Images », « *.png »)]****.*
- ***initialdir*** *: Par défaut, le répertoire proposé est celui où se trouve le script. Si vous désirez que le sélecteur de fichier s'ouvre dans un répertoire spécifique, précisez le à l'aide de ce paramètre.*



- ***Initialfile*** *: il est aussi possible de proposer le nom du fichier qui sera automatiquement sélectionné.*
- ***title*** *: permet de préciser un titre personnalisé.*

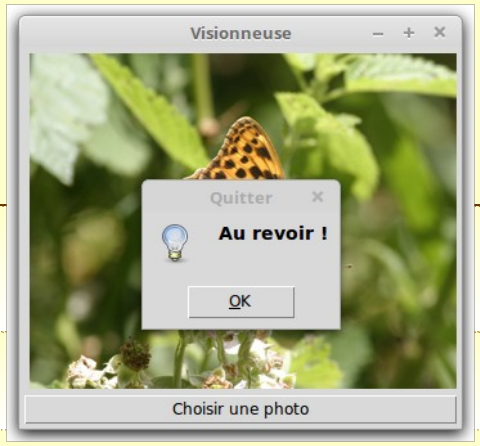
La classe ***« Canvas »*** *dispose de toutes les méthodes nécessaires pour réaliser tous les tracés que vous désirez. Toutes ces méthodes commencent systématiquement par le préfixe* ***« create_ »****, suivi du type de tracé : pour une portion d'ellipse -* ***arc()****, une image -* ***image()****, un ou plusieurs segments -* ***line()****, une ellipse (ou un cercle)* ***ellipse()****, un polygone -* ***polygon()****, un rectangle -* ***rectangle()****, pour insérer un texte* ***text()****, pour une fenêtre rectangulaire -* ***window()****.*



Attention, pour que la photo soit bien intégrée dans le canevas, elle doit être pris en compte grâce à la propriété « image ». Vous précisez ensuite comment la placer à l'aide de la méthode « create_image() ».

principal.py

```
from tkinter import *
from vignette import Vignette

# Fenêtre principale de l'application
class Fenêtre(Tk):
   def __init__(self, titre):
      super().__init__()
      self.title(titre)
      self.geometry('600x500')
      self.vignette = Vignette(self)
      Button(self, text='Choisir une photo', command=self.vignette.changer).pack(fill=X, padx=3, pady=3)
      self.mainloop()

   # Méthode redéfinie afin de proposer des boîtes de dialogue à la clôture de l'application
   def destroy(self):
      if askokcancel('Quitter', 'Arrêt définitif ?'):
         showinfo('Quitter', 'Au revoir !')
         super().destroy()

# Programme principal
if __name__ == '__main__' :
   programme = Fenêtre('Visionneuse')
```



Il est possible de définir une dimension précise lors de la création de la fenêtre indépendamment de sa constitution et des widgets qui la composent. Vous devez alors utiliser la méthode ***« geometry() »*** *en spécifiant les dimensions dans une chaîne de caractères au format spécifique :* ***« LARGEURxHAUTEUR+X+Y ».***

Dans notre fenêtre principale, nous redéfinissons la méthode ***« destroy() »*** *de telle sorte qu'il soit possible de demander à l'utilisateur s'il désire réellement quitter l'application. Une boîte de dialogue est alors proposée pour demander son avis.*

À ce sujet, il existe le sous-module « tkinter.messagebox » qui fournit un assortiment de boîtes de dialogue prédéfinies avec des fonctions dont le nom est assez évocateur. Voici la liste des huit boîtes mises à votre disposition : askquestion(), askretrycancel(), askyesno(), askyesnocancel(), showerror(), showinfo() et showwarning().

Chaque fonction ***« ask...(titre, message, ...) »*** *prévoit comme vous le voyez deux arguments pour le titre et le message correspondant à la question ou à l'avertissement. Ensuite, suivant la boîte de dialogue et le choix fait par l'opérateur, le retour sera* ***True, False, 'yes', 'no'*** *ou* ***None****.*

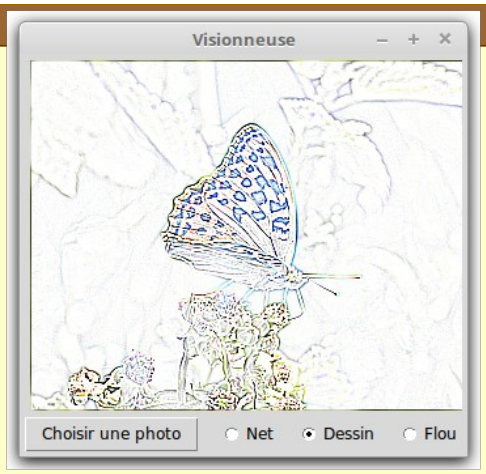
LES BOUTONS RADIOS – RADIOBUTTON

Dans ce projet, nous rajoutons des boutons radio qui vont nous permettre de choisir le type de traitement photo que vous souhaitez réaliser, une photo nette, un flou glaussien ou une photo ressemblant à un dessin.

C'est la classe ***« Radiobutton »*** *qui implémente un choix possible parmi plusieurs choix proposés. Le principe du bouton-radio c'est que lorsqu'il est sélectionné, tous les autres associés au même groupe sont automatiquement désactivés, ce qui permet bien de choisir une seule valeur parmi celles proposées. Voici ci-dessous les paramètres importants associés au constructeur de la classe* ***« Radiobutton »*** *:*

Paramètres du constructeur et Méthodes associés à la classe « Radiobutton »	
text	*Intitulé qui apparaît à droite du bouton-radio*
value	*Valeur correspondant au choix de ce bouton*
command	*Fonction ou méthode lancée automatiquement lorsque l'utilisateur clique sur ce bouton, à l'image d'un bouton classique.*
variable	*Variable qui prend automatiquement la valeur choisie par l'opérateur dans le groupe de boutons-radio. Tous les boutons du même groupe doivent impérativement posséder cette même variable. Elle doit être obligatoirement soit du type* ***« StringVar »****, soit du type* ***« IntVar »****.*
select()	*Méthode qui active le bouton-radio.*
deselect()	*Méthode qui désactive le bouton-radio.*
invoke()	*Produit le même effet que lorsque l'utilisateur clique sur le bouton pour changer son état.*

vignette.py




```
from tkinter import *
from tkinter.filedialog import *
from PIL import Image, ImageFilter, ImageTk

# Création d'un vignette à partir d'un Canvas
class Vignette(Canvas):
   def __init__(self, fenêtre):
      super().__init__(fenêtre)
      self.pack(fill=BOTH, expand=1, padx=3)
      self.bind('<Configure>', self.retailler)
      self.fichier=''
      self.choix=ImageFilter.SHARPEN

   def retailler(self, event):
      self.afficher()

   def changer(self):
      self.fichier = askopenfilename(title='Choisir une photo', filetypes=[('Photos', '*.jpg')])
      self.afficher()

   def filtre(self, choix):
      self.choix = [ImageFilter.SHARPEN, ImageFilter.BLUR, ImageFilter.CONTOUR][choix]
      self.afficher()

   def afficher(self):
      if not len(self.fichier) == 0:
         image = Image.open(self.fichier)
         image.thumbnail((self.winfo_width(), self.winfo_height()))
         self.image = ImageTk.PhotoImage(image.filter(self.choix))
         self.create_image(5, 5, image=self.image, anchor='nw')
```

principal.py

```
from tkinter import *
from tkinter.messagebox import *
from vignette import Vignette

# Fenêtre principale de l'application
class Fenêtre(Tk):
   def __init__(self, titre):
      super().__init__()
      self.title(titre)
      self.geometry('600x500')
      self.vignette = Vignette(self)
      Button(self, text='Choisir une photo', command=self.vignette.changer).pack(side=LEFT, padx=3, pady=3)
      self.boutons()
      self.mainloop()

   # Méthode redéfinie afin de proposer des boîtes de dialogue à la clôture de l'application
   def destroy(self):
      if askokcancel('Quitter', 'Arrêt définitif ?'):
         showinfo('Quitter', 'Au revoir !')
         super().destroy()

   # Mise en place des boutons radios
   def boutons(self):
      self.val = IntVar()
      net = Radiobutton(self, text='Net', variable=self.val, value=0, command=self.filtrer)
      Radiobutton(self, text='Flou', variable=self.val, value=1, command=self.filtrer).pack(side=RIGHT, padx=5)
      Radiobutton(self, text='Dessin', variable=self.val, value=2, command=self.filtrer).pack(side=RIGHT, padx=5)
      net.pack(side=RIGHT, padx=5)
      net.select()

   # Événement associé au choix du filtre pour le traitement de la photo
   def filtrer(self): self.vignette.filtre(self.val.get())

# Programme principal
if __name__ == '__main__' :
   programme = Fenêtre('Visionneuse')
```

Widget Échelle – Scale

Nous allons maintenant rajouter à notre application le choix du zoom sur la photo qui est en cours de visualisation. Nous le mettons en œuvre grâce au widget **« Scale »** qui est très facile à manipuler puisqu'il suffit de déplacer un curseur dans une échelle de valeurs, ici de 10 % à 100 %.

*Le widget **« Scale »** - intervalle de sélection – permet à l'utilisateur de choisir une valeur entière ou réelle à l'intérieur d'un intervalle précis. Il peut être placé horizontalement ou verticalement (mode par défaut). Voici ci-dessous, les paramètres importants pour le constructeur et les méthodes qui peuvent être utilises pour la gestion de cette échelle de valeurs :*

Paramètres du constructeur et Méthodes associés à la classe «Scale»	
orient	*Permet de choisir l'orientation de l'échelle à l'aide des constantes **« HORIZONTAL »** ou **« VERTICAL »** (par défaut).*
from_ et to	*Deux paramètres importants qui permettent de définir respectivement la valeur minimale et maximale.*
command	*Fonction ou méthode lancée automatiquement à chaque fois que l'utilisateur déplace le curseur. La méthode ou la fonction reçoit un argument (sous forme de chaîne de caractères) qui est la nouvelle valeur sélectionnée dans l'intervalle.*
resolution	*Sert à modifier l'incrément entre deux valeurs consécutives (par défaut **« 1,0 »**).*
label	*Vous pouvez afficher une étiquette avec le texte souhaité qui apparaîtra dans le coin supérieur gauche si le widget est orienté horizontalement et dans le coin supérieur droit s'il est orienté verticalement.*
length	*La longueur du widget dans la direction où celui-ci est orienté (par défaut 100 pixels).*
showvalue	*Par défaut, la valeur courante du curseur est affichée (au dessus du curseur s'il est horizontal ou à gauche s'il est vertical). Mettre cette option à 0 pour supprimer cet affichage.*
tickinterval	*Par défaut, sa valeur est **« 0 »** ce qui a pour effet de ne pas afficher de graduation le long de l'intervalle. Pour afficher une telle graduation, réglez ce paramètre avec une valeur entière ou réelle correspondant au pas entre deux valeurs successives.*
get()	*Méthode qui retourne la valeur courante du curseur.*
set(valeur)	*Méthode qui sert à positionner la valeur et le curseur de l'échelle.*

vignette.py

```
from tkinter import *
from tkinter.filedialog import *
from PIL import Image, ImageFilter, ImageTk

# Création d'un vignette à partir d'un Canvas
class Vignette(Canvas):
   def __init__(self, fenêtre):
      super().__init__(fenêtre)
      self.pack(fill=BOTH, expand=1, padx=3)
      self.fichier=''
      self.choix = ImageFilter.SHARPEN
      self.zoom = 10

   def changer(self):
      self.fichier = askopenfilename(title='Choisir une photo', filetypes=[('Photos', '*.jpg')])
      self.afficher()

   def filtre(self, choix):
      self.choix = [ImageFilter.SHARPEN, ImageFilter.BLUR, ImageFilter.CONTOUR][choix]
      self.afficher()

   def zoomer(self, zoom):
      self.zoom = int(zoom)
      self.afficher()

   def afficher(self):
      if not len(self.fichier) == 0:
         photo = Image.open(self.fichier)
         photo.thumbnail((photo.width*self.zoom/100, photo.height*self.zoom/100))
         self.image = ImageTk.PhotoImage(photo.filter(self.choix))
         self.create_image(5, 5, image=self.image, anchor='nw')
```

principal.py

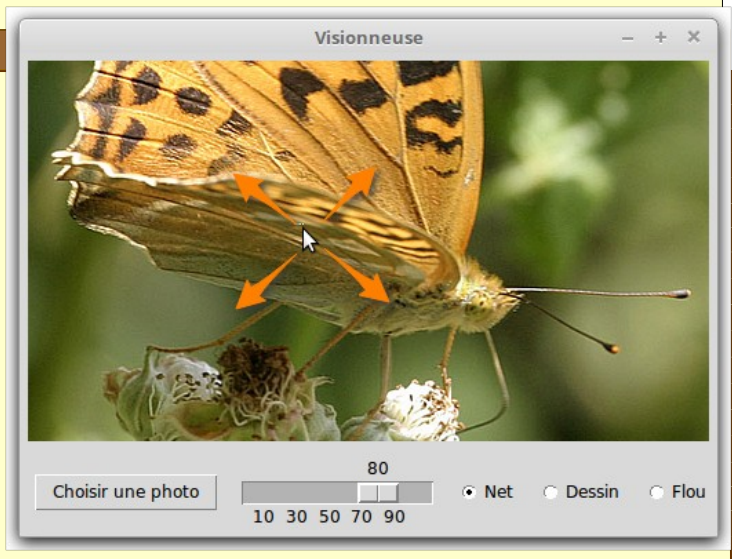
```
from tkinter import *
from tkinter.messagebox import *
from vignette import Vignette

# Fenêtre principale de l'application
class Fenêtre(Tk):
   def __init__(self, titre):
      super().__init__()
      self.title(titre)
      self.geometry('600x500')
      self.vignette = Vignette(self)
      Button(self, text='Choisir une photo', command=self.vignette.changer).pack(side=LEFT, padx=10, pady=3)
      self.réglages()
      self.mainloop()
```

```
   # Méthode redéfinie afin de proposer des boîtes de dialogue à la clôture de l'application
   def destroy(self):
      if askokcancel('Quitter', 'Arrêt définitif ?'):
         showinfo('Quitter', 'Au revoir !')
         super().destroy()

   # Mise en place des boutons radios et de l'échelle pour choisir le zoom de la photo
   def réglages(self):
      self.val = IntVar()
      net = Radiobutton(self, text='Net', variable=self.val, value=0, command=self.filtrer)
      Radiobutton(self, text='Flou', variable=self.val, value=1, command=self.filtrer).pack(side=RIGHT, padx=5)
      Radiobutton(self, text='Dessin', variable=self.val, value=2, command=self.filtrer).pack(side=RIGHT, padx=5)
      net.pack(side=RIGHT, padx=5)
      net.select()
      zoom = Scale(self, orient=HORIZONTAL, from_=10, to=100, resolution=5, tickinterval=10, command=self.vignette.zoomer)
      zoom.pack(side=BOTTOM, fill=X, padx=5, pady=3)
      zoom.set(50)

   # Événement associé au choix du filtre pour le traitement de la photo
   def filtrer(self): self.vignette.filtre(self.val.get())

# Programme principal
if __name__ == '__main__' :
   programme = Fenêtre('Visionneuse')
```

DÉPLACER LA PHOTO EN MODE ZOOM AVEC LA SOURIS

Pour finir avec cette application et afin qu'elle soit parfaitement ergonomique, je vous propose de rajouter le glisser-déposer associé à la souris afin de pouvoir visualiser la région de la photo que vous souhaitez montrer lorsque le zoom est relativement grand.

Événements associés à la souris	
<Button-1>	***L'instant où nous commençons à appuyer sur le bouton gauche de la souris.*** *Il faut alors enregistrer le point de contact sur la photo par rapport au bord haut-gauche de l'image.*
<ButtonRelease>	***Relâchement du bouton gauche de la souris.*** *Nous déplaçons souvent la photo en plusieurs glisser-déposer, il faut donc enregistrer tous les décalages intermédiaires.*
<B1-Motion>	***Déplacement de la souris avec le maintien sur le bouton gauche.*** *C'est durant cette phase que vous devez calculer où doit se trouver le bord haut-gauche de l'image afin que seule la partie de la photo que vous souhaitez visualiser soit affichée dans la partie visible du canevas. Ensuite, il faut ré-afficher la photo en temps réel afin qu'elle puisse suivre le mouvement de la souris.*



vignette.py

```
from tkinter import *
from tkinter.filedialog import *
from PIL import Image, ImageFilter, ImageTk

class Vignette(Canvas):
   '''Création d'une vignette à partir d'un Canvas'''
   def __init__(self):
      super().__init__()
      self.pack(fill=BOTH, expand=1, padx=5, pady=5)
      self.choix = ImageFilter.SHARPEN
      self.fichier=''
      self.bind('<Button-1>', self.actionBoutonGauche)
      self.bind('<B1-Motion>', self.déplacementPhoto)
      self.zoom = 100
      self.x = self.y = 0


   def changer(self):
      '''Sélectionne une nouvelle photo'''
      self.fichier = askopenfilename(title='Choisir une photo',
                                     filetypes=[('Photos', '*.jpg')], initialdir='/home/manu/Images')
      self.afficher()

   def filtre(self, choix):
      '''Change l'aspect de la photo : mode Normal, mode Dessin ou Flou Glaussien'''
      self.choix = [ImageFilter.SHARPEN, ImageFilter.BLUR, ImageFilter.CONTOUR][choix]
      self.afficher()

   def zoomer(self, nouveau):
      '''Effectue un zoom en gardant la même portion d'image'''
      ancienzoom = self.zoom
      self.zoom = int(nouveau)
      largeur = self.winfo_width()/2
```

```
        hauteur = self.winfo_height()/2
        centrex = largeur - self.x
        centrey = hauteur - self.y
        self.x = largeur - centrex*self.zoom/ancienzoom
        self.y = hauteur - centrey*self.zoom/ancienzoom
        self.afficher()

    def afficher(self):
        '''Fait apparaître la photo en tenant compte des différents réglages,
           la position actuelle, le choix du zoom et le filtre en cours'''
        if not len(self.fichier) == 0:
            photo = Image.open(self.fichier)
            photo.thumbnail((photo.width*self.zoom/100, photo.height*self.zoom/100))
            self.imageFiltrée = ImageTk.PhotoImage(photo.filter(self.choix))
            self.image = self.create_image(self.x, self.y, image=self.imageFiltrée, anchor='nw')

    def actionBoutonGauche(self, event):
        '''Enregistrer l'endroit de la souris au moment du clic'''
        self.px, self.py = event.x, event.y

    def déplacementPhoto(self, event):
        '''Glissement de la photo à l'aide du bouton gauche de la souris'''
        self.move(self.image, event.x-self.px, event.y-self.py)
        self.px, self.py = event.x, event.y
        self.x, self.y = self.coords(CURRENT)
```

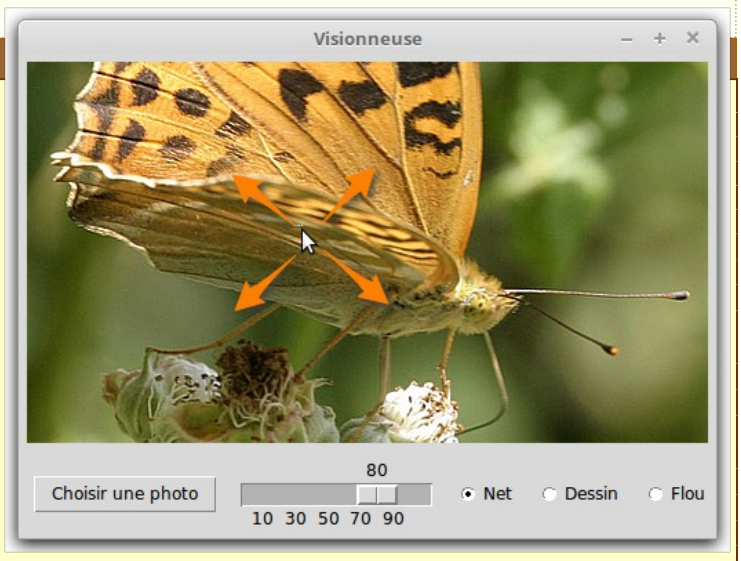
MÊME PROJET MAIS EN UTILISANT UNIQUEMENT DES FONCTIONS SANS DÉVELOPPEMENT OBJET

Le développement objet sous Python, contrairement à d'autres langages usuels comme le C++ ou le Java, pose un petit problème d'écriture puisque nous sommes systématiquement obligés de spécifier le paramètre « **self** » (ou un autre nom) dans nos méthodes pour faire référence à l'objet dans lequel nous sommes actuellement.

*Dans les codes précédents, vous voyez apparaître « **self** » partout, ce qui pour moi alourdit considérablement le code. Si vous avez plusieurs objets pour une même classe, ce n'est pas trop gênant. Par contre, pour les applications fenêtrées, comme c'est souvent le cas, nous avons besoin que d'une seule fenêtre, nous pouvons alors la décrire sous forme de fonctions séparées.*

Le seul inconvénient dans cette approche, c'est que nous devons utiliser des variables globales pour quelles puissent être utilisées dans les différents fonctions qui décrivent le projet, notamment lors de la prise en compte de la gestion événementielle.

Afin d'illustrer mes propos, je vous propose de reprendre le projet précédent en utilisant cette fois-ci que des fonctions sans structure objet, le tout dans un seul et unique fichier. Vous pourrez ainsi vous rendre compte de la différence. Chaque type d'approche propose des solutions intéressantes, mais aussi leurs inconvénients. À vous de choisir celle qui vous convient.



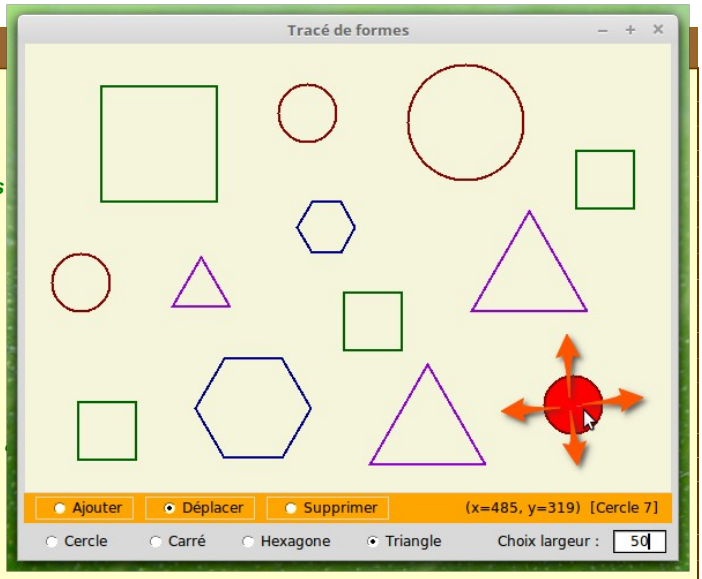
principal.py

```
from tkinter import *
from tkinter.messagebox import *
from tkinter.filedialog import *
from PIL import Image, ImageFilter, ImageTk


def créationVignette():
    '''Création d'une vignette à partir d'un Canvas'''
    global vignette, zoom, x, y
    vignette = Canvas()
    zoom = 50
    x = y = 0
    vignette.pack(fill=BOTH, expand=1, padx=5, pady=5)
    vignette.bind('<Button-1>', actionBoutonGauche)
    vignette.bind('<B1-Motion>', déplacementPhoto)
    vignette.bind('<Button-4>', changerZoom)
    vignette.bind('<Button-5>', changerZoom)

def ajoutComposants():
    '''Mise en place des boutons radios et de l'échelle pour choisir le zoom de la photo'''
    global choixFiltre, échelle
    Button(text='Choisir une photo', command=changerPhoto).pack(side=LEFT, padx=10, pady=3)
    choixFiltre = IntVar()
    net = Radiobutton(text='Net', variable=choixFiltre, value=0, command=changerFiltre)
    Radiobutton(text='Flou', variable=choixFiltre, value=1, command=changerFiltre).pack(side=RIGHT, padx=5)
    Radiobutton(text='Dessin', variable=choixFiltre, value=2, command=changerFiltre).pack(side=RIGHT, padx=5)
    net.pack(side=RIGHT, padx=5)
    net.invoke()
    échelle = Scale(orient=HORIZONTAL, from_=10, to=100, resolution=5, tickinterval=10, command=zoomer)
    échelle.pack(side=BOTTOM, fill=X, padx=5, pady=3)
    échelle.set(50)
```

```
def changerPhoto():
    '''Sélectionne une nouvelle photo'''
    global fichier
    fichier = askopenfilename(title='Choisir une photo', filetypes=[('Photos', '*.jpg')], initialdir='/home/manu/Images')
    afficher()


def changerFiltre():
    '''Change l'aspect de la photo : mode Normal, mode Dessin ou Flou Glaussien'''
    global filtre
    filtre = [ImageFilter.SHARPEN, ImageFilter.BLUR, ImageFilter.CONTOUR][choixFiltre.get()]
    afficher()


def zoomer(nouveau):
    '''Effectue un zoom en gardant la même portion d'image dans la fenêtre'''
    global zoom, x, y
    ancienzoom = zoom
    zoom = int(nouveau)
    largeur = vignette.winfo_width()/2
    hauteur = vignette.winfo_height()/2
    centrex = largeur - x
    centrey = hauteur - y
    x = largeur - centrex*zoom/ancienzoom
    y = hauteur - centrey*zoom/ancienzoom
    afficher()


def afficher():
    '''Fait apparaître la photo en tenant compte des différents réglages,
       la position actuelle, le choix du zoom et le filtre en cours'''
    global imageFiltrée, image
    if not len(fichier) == 0:
        photo = Image.open(fichier)
        photo.thumbnail((photo.width*zoom/100, photo.height*zoom/100))
        imageFiltrée = ImageTk.PhotoImage(photo.filter(filtre))
        image = vignette.create_image(x, y, image=imageFiltrée, anchor='nw')


def actionBoutonGauche(event):
    '''Enregistre l'endroit de la souris au moment du clic'''
    global px, py
    px, py = event.x, event.y


def déplacementPhoto(event):
    '''Glissement de la photo à l'aide du bouton gauche de la souris'''
    global x, y, px, py
    vignette.move(image, event.x-px, event.y-py)
    px, py = event.x, event.y
    x, y = vignette.coords(CURRENT)


def changerZoom(event):
    '''Prise en compte du zoom en relation avec la molette de la souris'''
    global échelle
    if event.num==4 and zoom<100 : échelle.set(zoom+5)
    elif event.num==5 and zoom>10 : échelle.set(zoom-5)


def demandeQuitter():
    '''Proposition de boîtes de dialogue à la clôture de l'application'''
    if askokcancel('Quitter', 'Arrêt définitif ?'):
        showinfo('Quitter', 'Au revoir !')
        fenêtre.quit()


if __name__ == '__main__' : '''Programme principal'''
    fichier=''
    fenêtre = Tk()
    fenêtre.title('Visionneuse')
    fenêtre.geometry('600x500')
    fenêtre.protocol("WM_DELETE_WINDOW", demandeQuitter)
    créationVignette()
    ajoutComposants()
    fenêtre.mainloop()
```

DESSINER DANS UN CANEVAS AVEC UN DÉVELOPPEMENT SOUS FORME DE FONCTIONS

Pour continuer dans ce style de développement, je vous propose un dernier projet qui permet de tracer des formes (cercle, carré, hexagone et triangle) dans une zone spécifique. Ces formes de dimension variables peuvent être déplacées par la suite ou même supprimées.



formes.py

```
from tkinter import *
from math import sqrt

def ajoutFeuilleDessin():
    '''Ajoute le canevas pour dessiner les formes choisies
    global dessin
    dessin = Canvas(bg='beige')
    dessin.pack(fill=BOTH, expand=TRUE, padx=5)
    dessin.bind('<Button-1>', gérerFormes)
    dessin.bind('<B1-Motion>', déplacerForme)
    dessin.bind('<Motion>', déplacementSouris)


def ajoutGestionFormes():
    '''Ajout des boutons radio pour la gestion des formes
       des coordonnées de la souris'''
    global formes, coordonnées, gestion
    formes = []
    cadre = Frame(bg='orange')
    cadre.pack(fill=X, padx=5)
    coordonnées = StringVar()
    Label(cadre, textvariable=coordonnées, bg='orange').pack(side=RIGHT, pady=3, padx=10)
    gestion = StringVar()
    boutonAjouter = Radiobutton(cadre, text='Ajouter', variable=gestion,
                    value='Ajouter', bg='orange', activebackground='beige')
    boutonAjouter.pack(padx=10, ipadx=7, pady=3, side=LEFT)
    Radiobutton(cadre, text='Déplacer', variable=gestion, value='Déplacer', bg='orange',
                activebackground='beige').pack(pady=3, ipadx=7, side=LEFT)
    Radiobutton(cadre, text='Supprimer', variable=gestion, value='Supprimer', bg='orange',
                activebackground='beige').pack(padx=10, ipadx=7, pady=3, side=LEFT)
    boutonAjouter.select()


def ajoutChoixFormes():
    '''Place tous les boutons radio associés au choix des formes (Cercle par défaut)'''
    global forme
    forme = StringVar()
    boutonCercle = Radiobutton(text='Cercle', variable=forme, value='Cercle')
    boutonCercle.pack(padx=5, ipadx=10, pady=3, side=LEFT)
    Radiobutton(text='Carré', variable=forme, value='Carré').pack(pady=3, ipadx=10,  side=LEFT)
    Radiobutton(text='Hexagone', variable=forme, value='Hexagone').pack(ipadx=10, pady=3, side=LEFT)
    Radiobutton(text='Triangle', variable=forme, value='Triangle').pack(ipadx=10, pady=3, side=LEFT)
    boutonCercle.select()


def ajoutChoixLargeur():
    '''Composants qui permet de spécifier la largeur de la prochaine forme'''
    global largeur
    largeur = IntVar()
    largeur.set(100)
    Entry(width=5, textvariable=largeur, justify=CENTER).pack(padx=10, pady=3, side=RIGHT)
    Label(text='Choix largeur :').pack(pady=7, side=RIGHT)


def gérerFormes(event):
    '''Ajoute ou supprime une forme. Garde en mémoire les coordonnées de la souris pour le déplacement'''
    global px, py
    px, py = event.x, event.y
    if gestion.get()=='Ajouter': tracerForme(event)
    elif gestion.get()=='Supprimer': dessin.delete(CURRENT)


def tracerForme(event):
    '''Trace la forme sélectionnée et l'enregistre dans la liste des formes'''
    l = largeur.get()/2
    x = event.x
    y = event.y
    if forme.get()=='Cercle':
        cercle = dessin.create_oval(x-l, y-l, x+l, y+l, outline='darkred', width=2, activefill='red', tags='Cercle')
```

```
      formes.append(cercle)
   elif forme.get()=='Carré':
      carré = dessin.create_rectangle(x-l, y-l, x+l, y+l, outline='darkgreen', width=2, activefill='green', tags='Carré')
      formes.append(carré)
   elif forme.get()=='Hexagone':
      r = sqrt(3)/4
      points = [(x+l, y), (x+l/2, y-2*r*l), (x-l/2, y-2*r*l), (x-l, y), (x-l/2, y+2*r*l), (x+l/2, y+2*r*l)]
      hexagone = dessin.create_polygon(points, fill='', outline='darkblue', width=2, activefill='blue', tags='Hexagone')
      formes.append(hexagone)
   else:
      b = l*sqrt(3)/3
      h = 2*b
      points = [(x-l, y+b), (x, y-h), (x+l, y+b)]
      triangle = dessin.create_polygon(points, fill='', outline='darkviolet', width=2, activefill='violet', tags='Triangle')
      formes.append(triangle)


def déplacerForme(event):
   '''Déplace la forme sélectionnée par glisser-déposer'''
   global px, py
   if gestion.get()=='Déplacer':
      dessin.move(CURRENT, event.x-px, event.y-py)
      px, py = event.x, event.y
      déplacementSouris(event)


def déplacementSouris(event):
   l = largeur.get()
   x = event.x
   y = event.y
   formesTrouvées = dessin.find_withtag(CURRENT)
   formeTrouvée=0
   if len(formesTrouvées)!=0:
      formeTrouvée = formesTrouvées[0]
      coordonnées.set("(x={}, y={})  [{} {}]".format(x, y, dessin.gettags(CURRENT)[0], formeTrouvée))
   else: coordonnées.set("(x={}, y={})  [Aucune forme]".format(x, y))


if __name__ == '__main__':
   '''Création et lancement de la fenêtre principale de l'application'''
   fenêtre = Tk()
   fenêtre.title('Tracé de formes')
   fenêtre.geometry('640x480')
   ajoutFeuilleDessin()
   ajoutGestionFormes()
   ajoutChoixFormes()
   ajoutChoixLargeur()
   fenêtre.mainloop()
```