(version 4.75 du 3/3/2025)

Notice d'utilisation de la bibliothèque graphique tkiteasy

`tkiteasy` est une librairie "*maison*" qui va vous permettre de gérer des graphismes sans avoir besoin de vous plonger dans les méandres techniques de `tkinter`. Elle est en fait une interface simple entre vous et ce module `tkinter` qui nécessite un certain apprentissage.

1 Installation de la librairie `tkiteasy`

La librairie `tkiteasy` est écrite en `python3`. Elle s'appuie sur la librairie graphique `tkinter`. Elle nécessite également la librairie `PIL`, qui permet de gérer des images. Pour utiliser `tkiteasy`, vous devez donc installer les paquets `python3-tk`, `python3-pil` et `python3-pil.imagetk` .

2 Ouverture de session graphique et objet Canevas

La première méthode (c'est à dire fonction) à utiliser pour pouvoir lancer `tkiteasy` est `ouvrirFenetre(x,y)`, où `x` indique la largeur en pixels de la fenêtre graphique, et où `y` indique sa hauteur. Exemple de lancement d'une session graphique:

```
g = ouvrirFenetre(800,600)
```

On a ici créé une fenêtre de 800 pixels de large sur 600 pixels de haut.
IMPORTANT: le point de coordonnées `(0,0)` se trouve toujours en haut à gauche de la fenêtre.

L'appel à `ouvrirFenetre` renvoie un objet `Canevas` (ici `g`). C'est un objet que vous devez conserver et utiliser tout au long de votre programme. Il vous permettra de lancer les méthodes graphiques qui sont présentées ci-dessous.

3 Méthodes du `Canevas`

À partir du moment où vous disposez d'un `Canevas`, vous allez pouvoir créez et manipuler toutes sortes de figures géométriques, textes, images dans votre fenêtre graphique. Voici les méthodes permettant de créer ces figures.
Le fichier `main.py` fourni avec `tkiteasy` vous montre comment manipuler ces objets graphiques et lancer les différentes méthodes de la librairie.

Création de figures géométriques, images, textes

`dessinerRectangle(x, y, l, h, col)`

Crée un rectangle plein, dont le coin supérieur gauche se trouve en `(x,y)`, dont la largeur est `l`, la hauteur `h` et la couleur `col` (voir la section dédiée aux couleurs).

IMPORTANT : cette méthode, ainsi que toutes les méthodes qui créent des figures géométriques, renvoie un objet. Vous pouvez récupérer cet objet dans une variable, ce qui vous permettra ensuite de le modifier, le déplacer, le supprimer, ou bien ignorer cet objet si vous pensez ne plus en avoir besoin ultérieurement.

`dessinerLigne(x, y, x2, y2, col [,ep])`

Cette méthode dessine une ligne entre le point `(x,y)` et le point `(x2, y2)`, de couleur `col`. Le dernier paramètre `ep` est optionnel: il permet de préciser l'épaisseur de la ligne tracée, et vaut 1 par défaut.

`dessinerCercle(x, y, r, col)`

Dessine un cercle de centre `(x,y)` et de rayon `r`, de couleur `col`.

`dessinerDisque(x, y, r, col)`

Dessine un disque de centre `(x,y)` et de rayon `r`, de couleur `col`.

`dessinerFleche(x, y, x2, y2, col [,ep])`

Dessine une flèche du point `(x,y)` vers le point `(x2, y2)`, de couleur `col`. Le dernier paramètre `ep` est optionnel: il permet de préciser l'épaisseur de la ligne tracée, et vaut 1 par défaut.

`changerPixel(x, y, col)`

Dessine un pixel de coordonnées `(x,y)` et de couleur `col`.

`afficherTexte(txt, x, y [, col , sizefont, font])`

Écrit un texte `txt` **centré** en position `(x,y)`, de couleur `col` (optionnel, blanc par défaut), de taille `sizefont` (optionnel, 18 par défaut) et de fonte `font` (optionnel, `Helvetica` par défaut). De nombreuses autres polices sont disponibles. Vous pouvez en voir la liste en exécutant `print(tkFont.families())` ou le programme `test_polices.py`.

`afficherImage(x, y, filename [, sx, sy])`

Affiche une image en position `(x,y)`, provenant du fichier `filename`. Le fichier doit être précisé **avec son chemin relatif au script python**. Cette méthode accepte les principaux formats bruts (`PNG, BMP`) ou compressés (`JPG, GIF`).
Les paramètres `sx` et `sy` sont optionnels et permettent de redimensionner l'image à la dimension $sx \times sy$.

Méthodes de modification d'objets existants

Les méthodes qui suivent permettent de modifier un objet existant: changer ses caractéristiques (couleur, texte), le déplacer ou le supprimer. Pour ce faire, vous devez avoir conservé une référence à l'objet créé, au moment de sa création. Voir exemple ci-dessous.

`deplacer(obj, dx, dy)`

Permet de déplacer un objet `obj` de `dx` pixels horizontalement et `dy` pixels verticalement.
Exemple:

```
c = g.dessinerCercle(800,600,10,"pink") # c contient une reference au cercle cree
# plus tard dans le programme...
g.deplacer(c, 10, 0) # ...on deplace le cercle de 10 pixels
                     # vers la droite en utilisant cette reference
```

`supprimer(obj)`

Supprime l'objet `obj`.

`supprimerTout()`

Supprime tous les objets de la fenêtre courante.

`changerCouleur(obj, col)`

Change la couleur de l'objet `obj` en `col`.

`changerTexte(obj, txt)`

Change le texte de l'objet `obj` (nécessairement un objet texte) en `txt`.

`placerAuDessus(obj)`

Place l'objet `obj` au premier plan.

`placerAuDessous(obj)`

Place l'objet `obj` au dernier plan.

Gestion des événements

On appelle *événement* une interaction entre l'utilisateur et le programme: clic souris, appui touche clavier, déplacement souris.

`attendreTouche()`

Attend qu'une touche soit pressée au clavier. La variable renvoyée est une *string* qui contient la description de la touche: `"a", "b", "c",`... Elle peut également contenir le nom de certaines touches spéciales (curseurs, touche de fonction...) : `"Right", "Left", "Up", "Down",`...voir la section dédiée aux touches
La variable contient `None` si aucune touche n'a été pressée depuis la dernière récupération de touche.

`recupererTouche()`

Même fonctionnement que la méthode précédente (renvoie la touche cliquée), mais la fonction est non bloquante: elle renvoie la dernière touche pressée sans bloquer le déroulement du programme. Elle renvoie `None` si aucune touche n'a été pressée depuis la dernière récupération de touche.

`attendreClic()`

Attend qu'un bouton souris soit pressé. La variable renvoyée est un `Event` qui contient des champs `x` et `y`. Ainsi, vous pouvez obtenir les coordonnées du point cliqué de la façon suivante:

```
p = g.attendreClic()
print(p.x, p.y)
```

Un champ `num` permet également de tester quel bouton a été cliqué: `num=1` pour le bouton gauche, `num=2` pour le bouton milieu. `num=3` pour le bouton droit.

`recupererClic()`

Même fonctionnement que la méthode précédente (renvoie la position cliquée), mais la fonction est non bloquante: elle renvoie la dernière position cliquée sans bloquer le déroulement du programme. Si aucun clic n'a eu lieu depuis la dernière récupération de position, la méthode renvoie `None`.

`recupererPosition()`

Permet de récupérer la dernière position de souris suite à un déplacement de souris. La variable renvoyée est un `Event` qui contient des champs `x` et `y`: les coordonnées `(x,y)` de la souris, ou `(0,0)` si aucun déplacement n'a eu lieu depuis le lancement du programme.

`recupererObjet(x,y)`

Permet de récupérer l'objet se trouvant au premier plan en position `(x,y)`. Renvoie `None` si aucun objet ne se trouve à cet emplacement.

Autres fonctions

`actualiser()`

Cette méthode, placée après une méthode graphique, force le rafraîchissment. À utiliser avec modération au risque de ralentir votre programme.

IMPORTANT: Lorsqu'on crée ou qu'on modifie un objet graphique, il n'est pas modifié immédiatement à l'écran. Le système de gestion graphique attend un certain temps pour effectuer ce rafraîchissement, afin de réunir plusieurs modifications et réduire ainsi le nombre de rafraîchissements, opérations assez coûteuses.
Si vous dessinez un simple objet graphique, vous n'aurez donc pas besoin de forcer un rafraîchissement de l'écran. Mais lorsqu'on demande de nombreuses modifications, comme par exemple lorsqu'on déplace de nombreux objets simultanément, cela peut devenir nécessaire.
Si vous ne voyez pas à l'écran les résultats des fonctions graphiques que vous avez appelées, essayez un `actualiser()`!

`pause(sec)`

Parfois, le programme crée se déroule trop rapidement et nécessite d'être ralenti. Cette méthode permet de forcer une pause, d'une durée de `sec` secondes. `sec` est un flottant, on peut donc demander une pause en dixièmes, centièmes, millièmes de secondes. Par défaut, cette méthode crée une pause d'une demi milliseconde.

`fermerFenetre()`

Ferme la fenêtre graphique.

4 `ObjetGraphique`

Toutes les méthodes qui créent des objets graphiques (rectangle, disque, image...etc) renvoient un `ObjetGraphique`. Cet `ObjetGraphique` contient trois champs utiles si vous souhaitez récupérer certaines informations concernant cet objet:

- `x` et `y`, qui contiennent les coordonnées actuelles de l'objet
- `col`, qui contient sa couleur

5 Les couleurs

Il existe deux façons d'indiquer une couleur:

- par son code RVB sous la forme hexadécimale: `#rrvvbb` où `rr,vv,bb` sont les composantes rouges, vertes et bleues de la couleur souhaitée. Exemple: `#ff0000` indique le rouge.
  ATTENTION: cet hexadécimal doit être transmis sous forme de `string`.
  Exemple: `dessinerCercle(x,y,r,"#ff00ab")`
- par une `string` prédéfinie. Les couleurs `'white', 'black', 'red', 'green', 'blue', 'cyan', 'yellow'`, et `'magenta'` sont toujours disponibles. De nombreuses autres couleurs le sont également, en fonction de la configuration locale de votre ordinateur. Faites des tentatives: `"pink", "gold"`,...

6 Les touches clavier

`Return`	La touche Entrée
`space`	La barre espace
`Tab`	La touche de Tabulation, Tab
`Up`	↑
`Down`	↓
`Left`	←
`Right`	→
`Alt_L`	La touche Alt située à gauche.
`Alt_R`	La touche Alt située à droite.
`Control_L`	La touche Ctrl de gauche
`Control_R`	La touche Ctrl de droite
`Shift_L`	La touche Maj de gauche
`Shift_R`	La touche Maj de droite
`Caps_Lock`	Verr Maj
`Delete`	Suppr
`BackSpace`	La touche Retour Arrière
`Home`	Début
`End`	Fin
`Insert`	Inser
`Escape`	Echap
`F1`	La touche fonction F1
`F2`	La touche fonction F2
`Next`	PageDown
`Prior`	PageUp
`Pause`	Pause
`Num_Lock`	Verr Num
`Print`	ImprÉcran
`KP_0`	0 sur le clavier numérique
`KP_1`	1 sur le clavier numérique
`KP_Up`	↑ sur le clavier numérique
`KP_Down`	↓ sur le clavier numérique
`KP_Left`	← sur le clavier numérique
`KP_Right`	→ sur le clavier numérique
`KP_Add`	+ sur le clavier numérique
`KP_Multiply`	× sur le clavier numérique
`KP_Subtract`	- sur le clavier numérique
`KP_Divide`	/ sur le clavier numérique
`KP_Next`	PageDown sur le clavier numérique
`KP_Prior`	PageUp sur le clavier numérique
`KP_Decimal`	Symbole de la ponctuation décimale (,) sur le clavier numérique
`KP_Delete`	Suppr sur le clavier numérique
`KP_End`	Fin sur le clavier numérique
`KP_Enter`	Entrée sur le clavier numérique
`KP_Home`	Début sur le clavier numérique
`KP_Insert`	Insert sur le clavier numérique