Le module `turtle` contient les commandes du célèbre langage LOGO sous forme de fonctions et classes. Ces commandes permettent, entre autre, de déplacer, lever et baisser un stylo (incarné par une tortue) sur une fenêtre graphique. On peut faire pivoter la tortue pour changer de direction. On réalise ainsi des dessins de manière simple, rapide et amusante (l'univers LOGO est riche malgré son apparente simplicité).

Le module `turtle` est programmé en langage Python avec l'aide du module Tkinter pour toute la partie graphique.

Le module contient 8 liens vers des modules standard (utilisés en interne), 15 classes et 122 fonctions. Le lecteur écrira un programme pour vérifier ces chiffres.

On importe le contenu du module pour la suite :

```
>>> from turtle import *       # déconseillé !
```

La plupart des fonctions Turtle possède un ou plusieurs alias. Il s'agit d'alias au sens interne (Turtle) mais pas au sens Python du terme. Par exemple, `up` et `pu` sont des alias de `penup`, et pourtant :

```
>>> penup is up
False
>>> id(penup), id(up), id(pu)
(4319016288, 4319108352, 4319017248)
```

Dès qu'on appelle une fonction (ou une classe) du module, Python ouvre une fenêtre Turtle. Les fonctions qui commandent la tortue créent la *tortue courante* si celle-ci n'est pas déjà présente. On montre à la sous-section 39.15.3 comment créer plusieurs tortues en même temps (voir aussi 39.16.2).

1 Coordonnées (modes)

Le canevas dans lequel se déplace la tortue est muni d'un système de coordonnées cartésiennes (x, y) attaché à un repère orthonormé $(O, \vec{\imath}, \vec{\jmath})$ et d'un système de coordonnées polaires (r, θ) de même origine et de vecteur $\vec{u}$.

L'origine du repère est le centre du canevas. L'axe des abscisses est horizontal orienté de la gauche vers la droite. L'axe des ordonnées est vertical orienté du bas vers le haut. L'unité est le pixel.

La tortue apparaît à l'origine, orientée selon $\vec{u}$.

Le vecteur $\vec{u}$ (référence des angles) et l'orientation du plan dépendent du mode de travail :

- mode `"standard"` et `"world"` : le plan est orienté positivement (le repère est direct) et $\vec{u} = \vec{\imath}$;
- mode `"logo"` : le plan est orienté négativement (le repère est indirect) et $\vec{u} = \vec{\jmath}$.

On fixe le mode avec `mode` :

```
>>> mode("world")
```

Trois valeurs possibles : `"standard"`, `"world"` et `"logo"`. Le mode par défaut est `"standard"`.

Un appel de `mode` sans argument retourne le mode :

```
>>> mode()
'world'
```

Le mode `"world"` permet de changer le système de coordonnées :

```
>>> setworldcoordinates(xg, yb, xd, yh)
```

Résultat :

- les points du bord gauche ont pour abscisse `xg` ;
- les points du bord droit ont pour abscisse `xd` ;
- les points du bord haut ont pour ordonnée `yh` ;
- les points du bord bas ont pour ordonnée `yb` ;

Il suffit de de prendre $x_g > x_d$ (resp. $y_b > y_h$) pour inverser l'orientation de l'axe des x (resp. y).

L'appel de `setworldcoordinates` fixe automatiquement le mode `"world"`.

Attention. Définir un repère non orthonormé rend les angles difficiles à apréhender.

2 Angles

2.1 Unités d'angle

L'unité d'angle par défaut est le degré mais on peut prendre la n-ème partie du tour à la place :

```
>>> degrees(n)
```

La signature de `degrees` est `(fullcircle=360)`.

Si on veut le grade comme unité :

```
>>> degrees(400)
```

Si on veut l'heure comme unité :

```
>>> degrees(12)      # 1/12 de tour
```

Si on veut le radian comme unité :

```
>>> from math import pi
>>> degrees(2 * pi)
```

ou plus simplement :

```
>>> radians()
```

2.2 Angle de la tortue

Pour connaître le *cap*, c'est-à-dire la mesure de l'angle $(\vec{\tau}, \vec{u})$ entre la tortue $\vec{\tau}$ et le vecteur du système des coordonnées polaires :

```
>>> angle = heading()       # cap
```

La réponse est donnée avec l'unité d'angle courante (voir sous-section précédente).

On peut orienter la tortue avec `setheading` (alias `seth`). Il suffit de passer la valeur désirée pour l'angle $(\vec{\tau}, \vec{u})$ dans l'unité courante :

```
>>> setheading(20.5)     # fixe le cap
```

Soit T la position de la tortue et M un point de coordonnées (x, y). Pour connaître la mesure de l'angle $(\vec{u}, \overrightarrow{TM})$:

```
>>> angle = towards(x,y)
>>> angle = towards((x,y))
```

3 Fonctions de base pour dessiner

3.1 Lever et baisser le stylo

Pour lever : `up` (alias `penup`, `pu`).

```
>>> up()
```

Pour baisser : `down` (alias `pendown`, `pd`).

```
>>> down()
```

Pour connaître l'état *levé* ou *baissé* du stylo : `isdown`.

```
>>> isdown()
True
```

3.2 Avancer et reculer

Pour avancer de `x` pixels : `forward` (alias `fd`).

```
>>> forward(x)           # x : int, float
```

On peut prendre `x` négatif (ce qui fait reculer la tortue).

Pour reculer de `x` pixels : `backward` (alias `back`, `bk`).

```
>>> backward(x)          # exemples : 12.7, -78
```

Ici aussi `x` peut être négatif.

Pour connaître la distance entre la tortue et le point de coordonnées (x, y) :

```
>>> d = distance(x,y)
>>> d = distance((x,y))
```

3.3 Pivoter

Pour pivoter positivement (ie. trigonométriquement) de `x` unités d'angle : `left` (alias `lt`).

```
>>> left(x)              # on peut prendre x < 0
```

Pour pivoter négativement de `x` unités d'angle : `right` (alias `rt`).

```
>>> right(x)             # on peut prendre x < 0
```

L'unité d'angle par défaut est le degré.

3.4 Remplir une surface

Pour colorier l'intérieur d'une surface, il suffit d'appeler `begin_fill` avant de dessiner le bord, puis d'appeler `end_fill` :

```
begin_fill()
forward(100)
left(120)
forward(100)
left(120)
forward(100)
end_fill()
```

Le lecteur remplira le disque bordé par `circle(100, steps=50)`.

Si la courbe dessinée n'est pas fermée, Python considère que la bordure s'achève avec le segment qui revient à l'origine du canevas.

Pendant une séquence de remplissage, `filling()` retourne `True`.

4 Cercles

Pour dessiner un cercle : `circle(radius, extent=None, steps=None)`.

```
>>> circle(133)    # cercle de rayon 133
```

Le centre se trouve à 133 pixels à gauche de la tortue car celle-ci a tourné positivement pour décrire le cercle.

Si on veut que la tortue tourne négativement, on prend un rayon négatif.

Le paramètre `extent` reçoit un angle. Avec un angle inférieur à 1 tour, on obtient un arc de cercle. Si on prend un angle négatif, la tortue exécute le mouvement en reculant. Voici le dessin d'un S (du bas vers le haut) :

```
circle(100, 180)
circle(-100, 180)
```

Finalement, la tortue tourne positivement dans les cas `++` et `--`.

En vérité, `circle` dessine un polygone régulier possédant un grand nombre de côtés, ce qui donne l'illusion d'un cercle. On contrôle le nombre de côtés avec le paramètre `steps`. Il s'ensuit que l'instruction

```
>>> circle(100, steps=7)
```

dessine un heptagone.

5 Points

Pour dessiner un point : `dot(size=None, *color)`.

```
>>> dot()
```

La taille par défaut est max (pensize + 4 ; 2 × pensize).

```
>>> dot(30, "cyan")           # color : voir 39.10
```

6 Empreintes

Pour que la tortue marque une empreinte (coup de tampon) : `stamp`.

```
>>> num = stamp()
```

La fonction retourne un entier qui sert d'identifiant.

Si on définit la forme (*shape*) de la tortue avec une image (GIF), on peut profiter de la fonction `stamp` pour afficher des copies de cette image n'importe où sur le canevas.

Pour effacer une empreinte de tortue : `clearstamp`.

```
>>> clearstamp(num)
```

Il faut bien sûr faire passer l'identifiant de l'empreinte visée.

Pour effacer plusieurs empreintes : `clearstamps`.

Un appel sans argument efface toutes les empreintes :

```
>>> clearstamps()
```

Si on fait passer un entier `n`, on distingue deux cas :

- $n > 0$: efface les n premières empreintes ;
- $n < 0$: efface les n dernières.

7 Texte

Pour écrire du texte : `write`. Un appel `write(x)` affiche `str(x)` :

```
>>> write(x)
```

La signature est :

```
(arg, move=False, align="left", font=("Arial",8,"normal"))
```

En prenant `move=True`, la tortue se déplace avec le texte écrit. Le paramètre `align` fixe l'alignement par rapport à la position de la tortue : `"left"`, `"right"` ou `"center"`. Le paramètre `font` fixe la fonte selon la syntaxe Tkinter, voir [?].

```
>>> texte = "https://www.youtube.com/@davidreeds-s2x"
>>> fonte = ("courier", 23, "bold", "italic", "underline")
>>> write(texte, 0, "center", fonte)
```

8 Déplacements absolus

Pour connaître la position de la tortue : `position` (alias `pos`).

```
>>> vec = position()
>>> x, y = position()
```

La fonction retourne un objet de type `Vec2D` (voir 39.16.4).

Pour connaître l'abscisse et l'ordonnée : `xcor` et `ycor`.

```
>>> x, y = xcor(), ycor()
```

Pour envoyer la tortue à un endroit donné : `setposition` (alias `setpos`, `goto`).

```
>>> setposition(x,y)
>>> setposition((x,y))
```

Pour modifier l'abscisse et l'ordonnée : `setx` et `sety`.

```
>>> setx(x)
```

```
>>> sety(y)
```

Pour envoyer la tortue à un point `(x,y)` stylo levé :

```
>>> teleport(x,y)
```

Si on est en mode remplissage (`begin_fill`), prendre « `fill_gap = False` » finalise le coloriage et le mode remplissage reprend dès qu'on arrive à la position demandée.

```
(x=None, y=None, *, fill_gap=False)
```

On peut modifier une seule coordonnée :

```
>>> teleport(x=120)
>>> teleport(y=50)
```

9 Enregistrer un polygone

Un appel de `begin_poly` lance l'enregistrement de tous les sommets parcourus par la tortue.

Un appel de `get_poly` retourne la liste des sommets enregistrés (sous forme de `tuple`).

Un appel de `end_poly` arrête l'enregistrement. Si on rappelle `begin_poly`, l'enregistrement redémarre à zéro.

Ci-dessous nous enregistrons les sommets d'une étoile à cinq branches et nous récupérons le résultat juste après l'arrêt de l'enregistrement (nous avons arrêté l'enregistrement pour faire les choses proprement) :

```
begin_poly()
for k in range(5):
    forward(200)
    right(144)
end_poly()
enregistrement = get_poly()
```

Regardons le résultat :

```
>>> enregistrement
((0.00,0.00), (200.00,0.00), (38.20,-117.56), (100.00,72.65),
(161.80,-117.56), (-0.00,-0.00))
```

10 Couleurs

Le système RGB encode chaque couleur avec un triplet (r, g, b) de nombres de l'intervalle $[0\,;1]$. Ces nombres sont respectivement les intensités de rouge (*red*), vert (*green*) et bleu (*blue*) permettant d'obtenir par synthèse additive la couleur désignée.

Dans le monde Turtle, il existe trois manières d'écrire une couleur (trois syntaxes) :

- un triplet RGB `(r,g,b)` de nombres de $[0\,;1]$;
- un triplet RGB `(r,g,b)` d'entiers compris entre 0 et 255 ;
- une chaîne de caractères suivant la syntaxe Tkinter.

10.1 Mode couleur 1

On fixe le mode couleur `1` comme ceci :

```
>>> colormode(1)     # idem avec 1.0
```

Dans ce mode, tout triplet `(r,g,b)` désignant une couleur doit être composé de nombres de l'intervalle $[0\,;1]$.

10.2 Mode couleur 255

On fixe le mode couleur `255` comme ceci :

```
>>> colormode(255)
```

Dans ce mode, tout triplet `(r,g,b)` désignant une couleur doit être composé d'entiers compris entre 0 et 255.

Note : tout appel `colormode(x)` avec `x` entier ou décimal différent de 1 déclenche le mode `255`. Un appel `colormode()` sans argument retourne le mode courant.

10.3 Syntaxe Tkinter

Quel que soit le mode, on peut écrire une couleur avec une chaîne de caractères selon la syntaxe Tkinter. La syntaxe Tkinter comprend 2 formes :

- un nom de couleur en anglais ;
- une chaîne `"#RRGGBB"`.

Les noms de couleurs sont `"white"`, `"black"`, `"red"`, `"green"`, `"blue"`, `"cyan"`, `"yellow"`, `"magenta"`, etc. Tkinter et Turtle en connaissent à peu près 700.

Les chaînes `"#RRGGBB"` sont composées de 3 entiers `RR`, `GG`, `BB` écrits en base 16 représentant les intensités R, G, B : deux chiffres par canal. Il existe des variantes à 1, 3 et 4 chiffres hexadécimaux par canal. Le lecteur trouvera une description complète de ces syntaxes dans [?].

10.4 Couleur du stylo

Pour fixer la couleur du stylo : `pencolor`.

```
>>> pencolor((0.902, 0.980, 0.980))  # mode 1
>>> pencolor(230, 250, 250)          # mode 255
>>> pencolor((230, 250, 250))        # mode 255
>>> pencolor("lavender")             # mode quelconque
>>> pencolor("#e6e6fa")              # mode quelconque
```

Un appel `pencolor()` sans argument retourne la couleur du stylo :

```
>>> pencolor()      # mode 255
(230.0, 230.0, 250.0)
```

C'est étrange : en mode `255`, `pencolor` retourne des décimaux !

En mode `1`, on aurait obtenu à peu près `(0.90196, 0.90196, 0.98039)` (beaucoup plus de décimales en fait), le lecteur testera.

10.5 Couleur de remplissage

Pour choisir la couleur de remplissage : `fillcolor`.

```
>>> fillcolor("sienna")
```

Le lecteur testera d'autres syntaxes pour la couleur.

Pour connaître la couleur de remplissage :

```
>>> fillcolor()
'sienna'
```

10.6 Stylo et remplissage

Pour fixer les couleurs du stylo et du remplissage d'un coup :

```
>>> color("red", "brown")   # pen, fill
```

La même couleur pour le stylo et le remplissage :

```
>>> color("red")
```

Pour connaître les deux couleurs :

```
>>> color()
('red', 'red')
```

10.7 Fond du canevas

Pour changer la couleur de fond du canevas : `bgcolor`.

```
>>> bgcolor("grey70")
```

Pour connaître la couleur de fond :

```
>>> bgcolor()
'grey70'
```

Pour mettre une image au format GIF en fond d'écran :

```
>>> bgpic("AlexandreTrijoulet.gif")
```

Pour connaître l'image de fond :

```
>>> bgpic()
'AlexandreTrijoulet.gif'
```

Pour retirer l'image :

```
>>> bgpic("nopic")
```

Note. On peut aussi tamponner (`stamp`) des tortues portant un GIF pour coller des images sur le canevas.

11 Effacer, remettre à zéro

Pour effacer les dessins de la tortue (sans déplacer la tortue) : `clear`.

```
>>> clear()
```

Pour effacer les dessins de la tortue et la remettre dans son état initial (réinitialiser ses attributs) : `reset`.

```
>>> reset()
```

Pour effacer la toile, détruire les tortues, et remettre le fond blanc : `clearscreen`.

```
>>> clearscreen()
```

Pour effacer la toile et remettre les tortues dans leur état initial : `resetscreen`.

```
>>> resetscreen()
```

Pour que la tortue revienne à sa position initiale : `home`.

```
>>> home()
```

Pour défaire la dernière action : `undo`.

```
>>> undo()
```

On peut répéter `undo` un nombre limité de fois. On contrôle ce nombre avec `setundobuffer` :

```
>>> setundobuffer(100)
```

La taille par défaut du tampon *undo* est `1000`. Un appel `setundobuffer(0)` réduit la taille à zéro et `setundobuffer(None)` supprime le tampon.

Pour connaître le nombre d'actions enregistrées dans le tampon : `undobufferentries`.

```
>>> undobufferentries()
38
```

Pour fermer la fenêtre *Python Turtle Graphics* : `bye`.

```
>>> bye()
```

12 Aspect

12.1 Épaisseur du stylo

Pour fixer l'épaisseur du stylo : `pensize` (alias `width`) :

```
>>> pensize(30)
```

Un appel sans argument retourne la taille courante :

```
>>> pensize()
30
```

Pour fixer les propriétés de la tortue et de son stylo : `pen(pen=None, **pendict)`. Les propriétés sont :

- `shown` : booléen « tortue visible » (39.12.3) ;
- `pendown` : booléen « tortue baissée » ;
- `pencolor` : couleur du stylo ;

- `fillcolor` : couleur du remplissage ;
- `pensize` : taille du stylo ;
- `speed` : vitesse d'animation (39.13) ;
- `resizemode` : mode de redimensionnement de la tortue (39.12.3) ;
- `stretchfactor` : facteurs d'étirement de la tortue (39.12.3) ;
- `shearfactor` : angle de cisaillement de la tortue (39.12.3) ;
- `outline` : épaisseur du contour de la tortue (39.12.3) ;
- `tilt` : rotation apparente de la tortue (39.12.3).

Le paramètre `pen` reçoit éventuellement une tortue (voir sous-section 39.15.3).

Un appel `pen()` retourne le dictionnaire des propriétés de la tortue courante.

12.2 Titre et taille de la fenêtre Turtle

Pour changer le titre : `title`. Un appel

```
>>> title(x)
```

fixe un titre égal à `str(x)`.

Pour changer la taille du canevas (et la couleur de fond) : `screensize`.

```
>>> screensize(400, 300, "red")
>>> screensize()
(400, 300)
```

Pour changer la taille et la position de la fenêtre *Turtle* : `setup`. Pour une fenêtre de 500×200 pixels avec coin haut-gauche en $(10, 20)$:

```
>>> setup(500, 200, 10, 20)
```

Pour une fenêtre occupant la moitié de la longueur de l'écran et le tiers de la hauteur :

```
>>> setup(0.5, 0.333)      # proportions
```

12.3 Apparence de la tortue

Pour rendre la tortue invisible : `hideturtle` (alias `ht`).

```
>>> hideturtle()
```

Cela a pour effet d'accélérer le dessin.

Pour rendre la tortue visible : `showturtle` (alias `st`).

```
>>> showturtle()
```

Pour connaître l'état *visible-invisible* de la tortue : `isvisble`.

```
>>> isvisible()
True
```

Pour changer la forme de la tortue : `shape`.

```
>>> shape("turtle")
>>> shape()
'turtle'
```

La forme par défaut est `"classic"`. Les autres valeurs possibles sont `"turtle"`, `"circle"`, `"square"`, `"triangle"` et toute chaîne désignant une forme enregistrée avec `register_shape`.

Pour récupérer le polygone qui donne sa forme (*shape*) à la tortue : `get_shapepoly`.

```
>>> get_shapepoly()
((0, 0), (-5, -9), (0, -7), (5, -9))
```

Pour enregister une forme de tortue (*shape*) : `register_shape` (alias `addshape`).

La signature est

`(name, shape=None)`

On peut enregistrer

- un fichier GIF en faisant passer son nom ;
- un *polygone* en lui donnant un nom ;
- un objet de type *shape* en lui donnant un nom.

Un polygone est une liste de coordonnées `(x,y)` de sommets, comme celle que l'on obtient avec `get_shapepoly` ou `get_poly` (39.9) :

```
begin_poly()
for k in range(5):
    forward(200)
    right(144)
end_poly()
enregistrement = get_poly()
register_shape("étoile", enregistrement)
```

Attention. Les coordonnées des sommets de la forme de la tortue sont exprimées dans le repère local associé à cette dernière. Il s'agit d'un repère orthonormé direct d'unité le pixel et d'axe des ordonnées l'axe de la tortue.

Pour obtenir la liste des noms des formes (*shapes*) enregistrées : `getshapes`.

```
>>> getshapes()
['arrow', 'blank', 'circle', 'classic', 'square', 'triangle',
'turtle', 'étoile']           # on retrouve notre "étoile"
```

Pour que l'aspect de la tortue dépende de l'état : `resizemode(rmode=None)`.

- `rmode = "auto"` : taille proportionnelle à `pensize` ;
- `rmode = "user"` : aspect dépendant de `stretchfactor` et `outlinewidth` ;
- `rmode = "noresize"` : aspect invariant.

```
>>> resizemode("user")
>>> resizemode()
'user'
```

Pour modifier les proportions de la tortue : `turtlesize` (alias `shapesize`). Un appel `turtlesize(a,b,c)` fixe `resizemode` à `"user"` et multiplie les dimensions :

- `a` : facteur d'étirement perpendiculaire au cap ;
- `b` : facteur d'étirement dans la direction du cap ;
- `c` : épaisseur du contour.

```
>>> shape("turtle")
>>> pencolor("red")          # affecte le contour
>>> turtlesize(8, 40, 20)
>>> turtlesize()
(8, 40, 20)
```

`(stretch_wid=None, stretch_len=None, outline=None)`

Pour faire pencher (incliner) la forme de la tortue : `tilt`.

```
>>> tilt(x)
```

La forme tourne alors de `x` unités d'angle, mais attention, son cap (*heading*) ne change pas.

Pour fixer l'inclinaison de manière absolue : `tiltangle` (alias `settiltangle`).

```
>>> tiltangle(x)
```

Pour modifier les proportions et l'inclinaison selon une tranformation linéaire : `shapetransform`. Un appel :

```
>>> shapetransform(a,b,c,d)
```

Transforme la silhouette de la tortue selon l'application de matrice :

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

```
>>> shapetransform()
(3, 1, 0, 2)
```

Signature :

```
(t11=None, t12=None, t21=None, t22=None)
```

On peut modifier la forme par cisaillement avec `shearfactor`. Un appel `shearfactor(t)`

```
>>> shearfactor(t)
```

applique le cisaillement d'angle $\arctan(t)$, c'est-à-dire un `shapetransform(1,t,0,1)`.

13 Vitesse

On fixe la vitesse de la tortue avec `speed` :

```
>>> speed(9)
```

La vitesse est un entier compris entre 0 et 10. Curieusement, la vitesse 0 est la plus grande vitesse.

Si `x` est un `float` compris entre 0 et 10, `speed(x)` équivaut à `speed(round(x))`. Si `x` est négatif, Turtle exécute `speed(0)`. S'il dépasse 10, `speed(10)`. On peut aussi faire passer l'une des chaînes ci-dessous :

`"slowest"`	1
`"slow"`	3
`"normal"`	6
`"fast"`	10
`"fastest"`	0

Pour connaître la vitesse :

```
>>> speed()
9
```

la vitesse par défaut est `3` (`"slow"`).

Un appel à la fonction `delay` diminue la vitesse :

```
>>> delay(100)
```

proportionnellement au nombre passé en argument. La docstring n'est pas claire : elle affirme que `x` est une durée exprimée en millisecondes, mais il est difficile de comprendre l'effet réel de cette fonction.

Pour connaître la durée du *delay* :

```
>>> delay()
100
```

La fonction `tracer(n=None, delay=None)` oblige Turtle à ne montrer les effets graphiques des fonctions exécutées que tous les n mouvements. Ainsi, avec `n` différent de `1`, on désactive le mode d'animation habituel de la tortue. En prenant `n` égal à `0`, Turtle n'affiche plus le dessin automatiquement : on doit appeler `update` pour forcer l'affichage. Le paramètre `delay` correspond à la fonction `delay`.

```
from time import sleep
tracer(5, delay=5000)
for k in range(36):
    forward(100)
    left(50)
sleep(5)       # pause de 5 secondes
update()
sleep(5)
teleport(200, 200)
tracer(0)
for k in range(36):
    forward(100)
    left(50)
sleep(5)
update()
sleep(5)
teleport(0,0)
write("FIN", 0, "center", ("", 60, ""))
```

14 Programmation événementielle

On lance la boucle des évènements avec `mainloop` (alias `done`). C'est obligatoirement la dernière commande Turtle du programme. Turtle propose des fonctions permettant de lier un élément de la fenêtre (canevas, tortue, etc.), un type d'évènement (clic, touche pressée, etc.) et une fonction (gestionnaire d'évènement). Le principe est simple : une fois le lien (*bind*) établi, tout évènement du type spécifié sur l'élément choisi déclenche la fonction.

14.1 Saisie au clavier

Fonction `textinput(title, prompt)`

Fait apparaître une fenêtre pop-up affichant le titre et le texte demandés ainsi qu'un champ permettant de saisir une chaîne de caractères. La saisie est validée avec le bouton OK. Le bouton CANCEL retournera `None` :

```
ch = textinput("titre", "blabla")
print(ch)      # ch est un str
```

Fonction `numinput(title, prompt, default=None, minval=None, maxval=None)`

Comme `textinput` sauf que la saisie retourne un `float`. Le champ est prérempli avec la valeur par défaut. Si le nombre saisi n'est pas dans l'intervalle fermé défini par `minval` et `maxval`, Turtle nous alerte avec un pop-up et remet une fenêtre `nunminput`.

```
x = numinput("titre", "blabla", 2.5, -10, 10)
print(x)       # x est un float
```

14.2 Évènements sur le canevas

Fonction `exitonclick()` : un clic sur le canevas déclenche `bye()`, ce qui ferme la fenêtre Turtle.

Fonction `onscreenclick(fun, btn=1, add=None)` : crée un lien entre le canevas, les évènements de type clic et la fonction passée à `fun`. La fonction est appelée en faisant passer les coordonnées `(x,y)` du lieu du clic.

```
def f(x,y): print(x,y)
onscreenclick(f)
```

On précise le type de clic avec `btn` (*button*) : clic droit (`1`), clic gauche (`3`) ou clic milieu (`2`).

En prenant `add = True`, on autorise le lien d'autres fonctions pour ce même type d'évènement.

Fonction `onkeypress(fun, key=None)` : crée un lien entre le canevas, les évènements « on appuie sur une touche » et la fonction passée à `fun`. Le gestionnaire est appelé sans arguments.

Pour que cela fonctionne, il faut mettre le focus sur la fenêtre en appelant `listen`.

Avec `key` on précise la touche :

```
def g(x,y): print("Touche [a] déclenche g()")
listen()
onkeypress(g, "a")
```

Un appel `onkeypress(g)` lie toutes les touches à `g`. Hélas, on ne peut pas récupérer la touche pressée.

Fonction `onkeyrelease(fun, key=None)`, alias `onkey`, crée un lien entre le canevas, les évènements « on relâche une touche » et la fonction passée à `fun`. Fonctionne comme `onkeypress`.

14.3 Évènements sur la tortue

Fonction `onclick(fun, btn=1, add=None)` : crée un lien entre la tortue, les évènements de type « clic sur la tortue » et la fonction passée à `fun`. Le fonctionnement est le même qu'avec `onscreenclick`.

```
def f(x,y): print(x,y)
onclick(f)            # clic sur la tortue
```

Fonction `onrelease(fun, btn=1, add=None)` : crée un lien entre la tortue, les évènements de type « on relâche le clic sur la tortue » et la fonction passée à `fun`. Le fonctionnement est le même qu'avec `onclick`.

Fonction `ondrag(fun, btn=1, add=None)` : crée un lien entre la tortue, les évènements de type « la souris bouge » et la fonction passée à `fun`. Le fonctionnement est le même qu'avec `onclick`.

14.4 Temporiser

Fonction `ontimer(fun, t=0)`

Déclenche la fonction donnée au bout de t millisecondes ($t \geqslant 0$).

```
from random import randint
def fonc():
    right(randint(0,3) * 90)
    forward(5 * randint(1,7))
    fonc()
ontimer(fonc, 0)   # tester avec speed(0) et hideturtle()
```

15 Éléments d'une fenêtre Turtle

15.1 Fenêtre Turtle elle-même

Le titre par défaut de la fenêtre Turtle est « `Python Turtle graphics` ».

On peut créer une fenêtre Turtle sans tortue, avec la fonction `Screen` :

```
>>> Screen()
```

```
<turtle._Screen object at 0x103263a40>
```

La fonction fait apparaître une fenêtre Turtle et retourne un objet *écran* qui la représente.

Maintenant que la fenêtre est créée, `Screen()` retourne l'objet *écran* `<0x103263a40>`.

On peut aussi obtenir l'objet *écran* avec la fonction `getscreen` :

```
>>> écran = getscreen()
>>> écran
<turtle._Screen object at 0x103263a40>
```

La fenêtre Turtle est de type `_Screen`, classe fille de `TurtleScreen`.

```
>>> issubclass(type(écran), TurtleScreen)
True
```

Fonctions pour connaître les dimensions de la fenêtre Turtle :

```
>>> window_width()
855
>>> window_height()
830
```

15.2 Canevas

La fenêtre contient un canevas (objet Tkinter de type `Canvas`).

Pour obtenir le canevas :

```
>>> getcanvas()
<turtle.ScrolledCanvas object .!scrolledcanvas>
```

Le lecteur affichera les attributs `canvwidth` et `canvheight` de l'objet *écran*.

Le canevas reçoit la *tortue courante* et les tortues supplémentaires.

15.3 Tortues

La tortue courante est créée dès que l'on appelle une fonction commandant un mouvement ou un tracé.

Supposons la tortue courante instanciée.

Pour obtenir la tortue courante, on appelle `getturtle` (alias `getpen`) :

```
>>> getturtle()
```

```
<turtle.Turtle object at 0x1058827b0>
```

Pour créer d'autres tortues, soit on appelle la classe `Turtle` :

```
>>> Ysé = Turtle()
```

soit on clone la tortue courante :

```
>>> Yanis = clone()
```

La copie ainsi obtenue possède (au départ) les mêmes propriétés que le modèle original.

On commande les tortues supplémentaires en appelant leurs méthodes (39.16.2).

Pour obtenir toutes les tortues présentes dans la fenêtre : `turtles`.

```
>>> turtles()
[<turtle.Turtle object at 0x1058827b0>,
 <turtle.Turtle object at 0x10585a960>,
 <turtle.Turtle object at 0x105c94cb0>]
```

16 Classes

Nous ne présentons ici que quelques classes du module.

16.1 Classe `TurtleScreen`

C'est (quasiment) le type de la fenêtre Turtle. En vérité, on ne l'instancie pas : pour créer une fenêtre, on appelle la fonction `Screen`.

Les méthodes d'une instance de `TurtleScreen` (c'est-à-dire de notre objet *écran*) sont `mode`, `setworldcoordinates`, `colormode`, `bgcolor`, `bgpic`, `clear` (alias `clearscreen`), `reset` (alias `resetscreen`), `bye`, `title`, `screensize`, `setup`, `register_shape` (alias `addshape`), `getshapes`, `delay`, `tracer`, `update`, `mainloop`, `textinput`, `numinput`, `exitonclick`, `onscreenclick` (alias `onclick`), `listen`, `onkeypress`, `onkeyrelease` (alias `onkey`), `ontimer`, `window_height`, `window_width`, `getcanvas` et `turtles`. Elles fonctionnent comme les fonctions homonymes.

Les attributs-données sont `canvwidth`, `canvheight`, `cv`, `xscale` et `yscale`.

Notons que `cv` pointe vers le canevas et `xscale` (resp. `yscale`) contient l'unité de l'axe des abscisses (resp. ordonnées) en pixels (égal à 1 sauf si on a fait appel à `setworldcoordinates`).

16.2 Classe `Turtle`

Chaque instanciation de `Turtle` (alias `Pen`) crée une tortue.

Les méthodes d'une tortue sont : `degrees`, `radians`, `heading`, `setheading` (alias `seth`), `towards`, `up` (alias `penup`, `pu`), `down` (alias `pendown`, `pd`), `isdown`, `forward` (alias `fd`), `backward` (alias `back`, `bk`), `distance`, `left` (alias `lt`), `right` (alias `rt`), `begin_fill`, `end_fill`, `filling`, `circle`, `dot`, `stamp`, `clearstamp`, `clearstamps`, `write`, `position` (alias `pos`), `xcor`, `ycor`, `setposition` (alias `setpos`, `goto`), `setx`, `sety`, `teleport`, `begin_poly`, `end_poly`, `get_poly`, `pencolor`, `fillcolor`, `color`, `clear`, `reset`, `home`, `undo`, `setundobuffer`, `undobufferentries`, `pensize` (alias `width`), `pen`, `hideturtle` (alias `ht`), `showturtle` (alias `st`), `isvisible`, `shape`, `get_shapepoly`, `resizemode`, `turtlesize` (alias `shapesize`), `tilt`, `tiltangle` (alias `settiltangle`), `shapetransform`, `shearfactor`, `speed`, `clone`, `onclick`, `onrelease`, `ondrag`, `getpen`, `getscreen` et `getturtle` (`getpen`). Elles fonctionnent comme les fonctions homonymes.

Si `t` est une tortue, `t.getscreen()` retourne l'objet *écran*, c'est-à-dire l'attribut `t.screen`. De plus, `t.getturtle()` retourne `t`.

L'attribut `t.undobuffer` contient un objet représentant la pile *undo*.

16.3 Classe **Shape**

Un objet de type `Shape` représente une forme que l'on peut donner à une tortue.

La signature de `Shape` est :

```
(type_, data=None)
```

où `type_` reçoit l'une des valeurs suivantes :

- `"polygon"` : un polygone ;
- `"compound"` : plusieurs polygones ;
- `"image"` : image GIF.

Les formes de la catégorie `"compound"` supportent la méthode `addcomponent` pour ajouter les composantes polygonales. Sa signature est :

```
(poly, fill, outline=None)
```

Créons une forme `"polygon"` :

```
>>> polygone = ((0,0), (40,0), (40,10))
>>> forme1 = Shape("polygon", polygone)
>>> register_shape("f1", forme1)
>>> shape("f1")
```

Attention : les coordonnées des sommets de la forme sont exprimées dans le repère local associé à la tortue, voir 39.12.3.

Créons une forme `"compound"` :

```
>>> forme2 = Shape("compound")   # pas de data avec compound
>>> poly1 = ((0,0), (10,-5), (0,10), (-10,-5))
>>> forme2.addcomponent(poly1, "red", "blue")
```

```
>>> register_shape("f2", forme2)
>>> shape("f2")
```

Ajoutons une composante à `forme2` :

```
>>> poly2 = ((0,0), (10,10), (-10,10))
>>> forme2.addcomponent(poly2, "yellow", "brown")
>>> shape("f2")
```

16.4 Classe `Vec2D`

Un objet de type `Vec2D` représente un vecteur du plan. La signature de `Vec2D` est

`(x,y)`

Ces vecteurs supportent l'addition (`+`), la soustraction (`-`), l'opposition (`-`), le produit scalaire (`*`) et la norme euclidienne (`abs`).

La méthode `rotate` d'un vecteur retourne la rotation du vecteur selon l'angle demandé.

La classe `Vec2D` dérive de la classe de `tuple`.