Documentation de l'environement python du lycée Diderot

Professeurs du lycée Diderot

sept. 19, 2022

General

CHAPITRE 1

Python à Diderot : les librairies pydiderotlibs

1.1 A propos

Ce site documente des librairies python utilisées par les enseignants regroupées sous le paquet pydiderotlibs du lycée Diderot à Marseille pour enseigner l'informatique. Le code est disponible sur ce dépot github et hebergé sur pypi.

L'objectif général est de cacher certaines difficultés techniques liées au langage de programmation afin de pouvoir cibler certains points pédagogiques. Ces librairies ont été écrites pour fonctionner avec l'environnement de travail pydiderotIDE

1.2 Installation

Sous windows vous pouvez installer facilement un environnement de dévellopement (IDE) python contenant déja les librairies pydiderotlibs en grace au projet PydiderotIDE.

Si vous avez déja un IDE python, deux méthodes d'installations sont disponibles :

— Avec pip : `pip3 install pydiderotlibs`.
— manuellement : Télécharger nos librairies zippées avec ce lien, décompresser le dossier et le placer à un emplacement de `PYTHONPATH`.

1.3 Librairies

Vous pouvez télécharger nos librairies zippées ici.

— repere : Permet l'affichage d'un repère du plan interactif (zoom, déplacer).
— entree : Fonctions d'entrées utilisateur avec des fenêtres tkinter.
— lycee : Regroupe les fonctions principales que sont amenés à utiliser les élèves de lycée en mathématiques (toutes filières confondues).
— graphique : Permet l'affichage d'une fenêtre graphique dynamique et une gestion simplifiée des entrées clavier et souris.

1.4 Participez !

Ce projet est un travail collaboratif initié par des enseignants du lycée Diderot à Marseille. Nous serions ravis de travailler avec vous et toute aide est la bienvenue. Si vous souhaitez participer, un guide est a votre disposition.

CHAPITRE 2

Entree

2.1 A propos

Cette librairie fournie des fonctions d'entrées utilisateur affichant une fenêtre avec un champ de saisie texte.

Nous constatons qu'un utilisateur peu expérimenté peut être surpris par l'invite d'entrée peu interactive de la console python et proposons cette librairie comme solution.

Concrètement cela peut remplacer avantageusement la fonction python `input()` dans un cadre pédagogique.

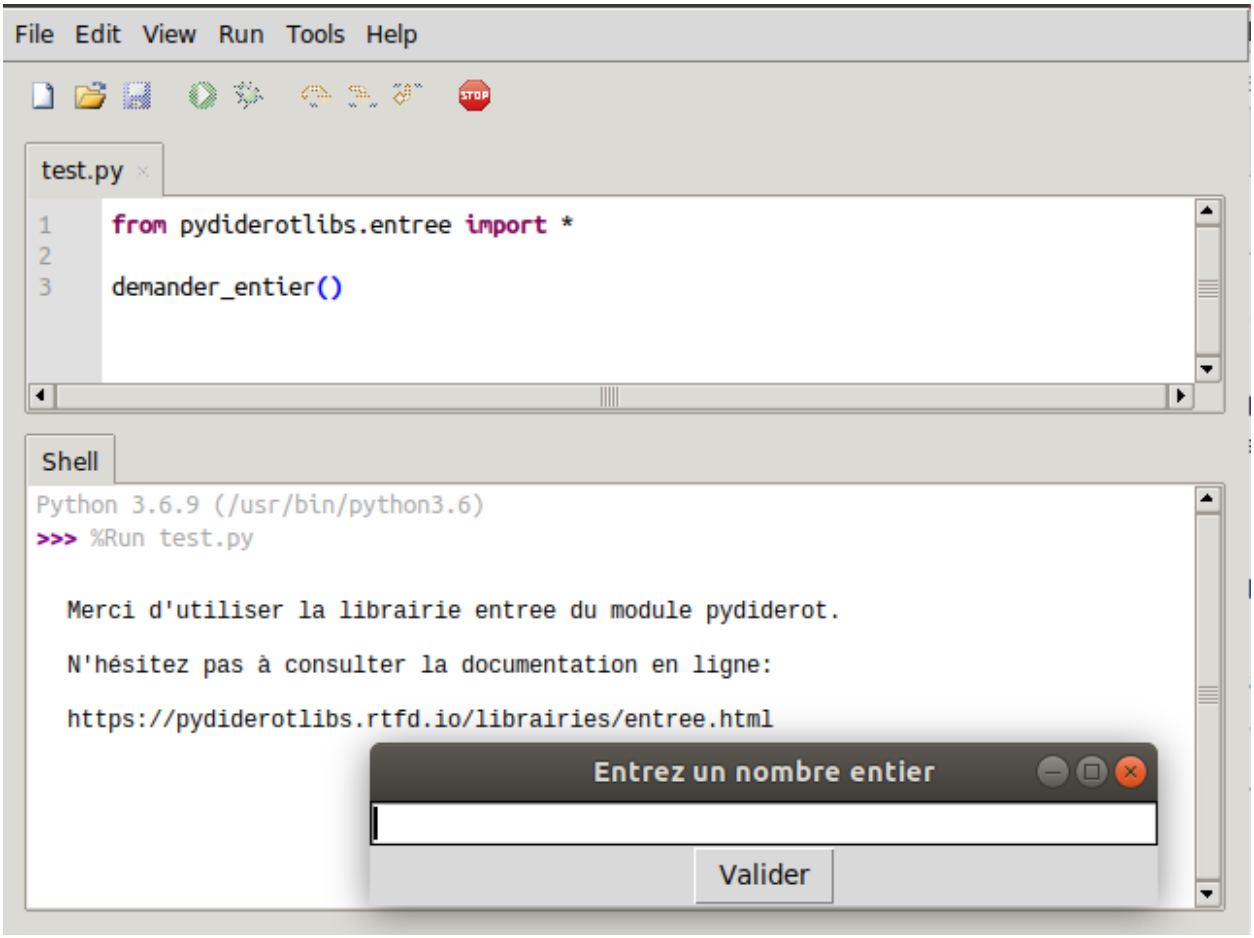
Cette librairie fournie également une fonction `demander_reel()` dont la sortie est un nombre réel de type `float` et une fonction `demander_entier()` dont la sortie est un nombre entier de type `int`.

2.2 Utilisation

```
# on importe la librairie
from pydiderotlibs.entree import *

# On demande une chaîne de caractères à l'utilisateur que l'on stocke dans la␣
↪variable x
x = demander_texte()
# x est une chaîne de caractère: str

# On demande un nombre réel à l'utilisateur que l'on stocke dans la variable y
y = demander_reel()
# y est un nombre réel: float
```

2.3 Exemple

```
# on importe la librairie
from pydiderotlibs.entree import *

# On demande un entier à l'utilisateur que l'on stocke dans la variable n
 n = demander_entier()
```



2.4 Documentation

demander_texte(*titre='Entrez un texte'*, *message=None*)

Ouvre une fenêtre avec le titre « titre » et attend une chaine de caractères.

Paramètres

— **titre** (*str, optionnel*) – Le titre de la fenetre (`"Entrez un texte"` par défaut).

— **message** (*str, optionnel*) – Si présent, on ajoute un champ de texte contenant `message`.

Renvoie La chaine de caractère (type `str`) entrée par l'utilisateur.

demander_reel(*titre='Entrez un nombre réel'*)

Ouvre une fenetre et attend un nombre réel.

Si ce n'est pas un nombre réel, on repose la question en ajoutant un message d'erreur.

Paramètres **titre** (*str, optionnel*) – Titre de la fenetre (`"Entrez un nombre réel"` par défaut).

Renvoie Le nombre réel entré par l'utilisateur (type `float`).

demander_entier(*titre='Entrez un nombre entier'*)

Ouvre une fenetre et attend un nombre entier.

Si ce n'est pas un nombre entier, on repose la question en ajoutant un message d'erreur.

Paramètres **titre** (*str, optionnel*) – Titre de la fenêtre (`"Entrez un nombre entier"` par défaut).

Renvoie Le nombre entier entré par l'utilisateur (type `int`).

CHAPITRE 3

Graphique

3.1 A propos

Cette librairie permet l'affichage d'une fenêtre graphique dynamique et fournie des fonctions permettant d'y afficher des objets géométriques simples (point, cercle, segment, vecteur, rectangle).

Basée sur pygame, vous pouvez également récupérer les événements clavier ou souris pour interagir avec l'utilisateur.

Vous pouvez, par exemple, l'utiliser pour construire un jeux de type pong.

3.2 Utilisation

Voici un exemple qui affiche une fenêtre graphique traversée en diagonale par un point.

```
# On importe la librairie
from pydiderotlibs.graphique import *
# Nous aurons également de la librairie time
from time import *

# On initialise les coordonnées du point au coin haut gauche de la fenêtre
x = 0
y = 0

# On créé la fenêtre graphique
creer_fenetre()

# Boucle principale
while 1:
    # Il est important d'appeler la fonction demande_evenements() qui gère la␣
↪fermeture de la fenêtre
    demande_evenements()

    # Trace un cercle au coordonnées (x,y)
```

(suite sur la page suivante)

(suite de la page précédente)

```
    cercle(x, y)
    # Attend un dixième de secondes
    sleep(0.1)
    # Efface le cercle
    cercle(x, y, couleur='blanc')
    # Ajoute le vecteur vitesse aux coordonnées du point
    x += 1
    y += 1
```

Note : Il est important d'appeler la fonction *demande_evenements()* qui gère la fermeture de la fenêtre

3.3 Événements

La fonction *demande_evenements()* permet une gestion simplifiée des entrées clavier et souris de l'utilisateur.

Elle retourne un dictionnaire contenant les touches pressées, les clics et déplacement souris.

3.3.1 Touches clavier

— Les touches spéciales sont présentes sous la forme : `'haut'`, `'bas'`, `'gauche'`, `'droite'` et `'espace'`
— Les touches alphanumériques sont présentes sous leur forme ascii : `'a'`, `'b'`,... `';'`,...

Vous pouvez par exemple tester si les touches `'haut'` et `'a'` sont pressées :

```
# On importe la librairie graphique
from pydiderotlibs.graphique import *
# On créé la fenêtre graphique
creer_fenetre()

# Boucle principale
while 1:

    evenements = demande_evenements()
    if 'haut' in evenements:
        print('la touche "haut" est enfoncée')
    if 'a' in evenements:
        print('la touche "a" est enfoncée')
```

3.3.2 Et la souris ?

— Un clic sur le bouton gauche de la souris sera présent sous la forme `'clic'`. Sa valeur contiendra les coordonnées de la souris au moment du clic.
— Un déplacement de souris sera présent sous la forme `'souris'`. Sa valeur contiendra les coordonnées de la souris après le déplacement.

Vous pouvez par exemple tester si un utilisateur bouge la souris ou clique et récupérer les coordonnées de la souris :

```
# On importe les librairie et time
from pydiderotlibs.graphique import *
# On créé la fenêtre graphique
```

(suite sur la page suivante)

(suite de la page précédente)

```
creer_fenetre()

# Boucle principale
while 1:
    evenements = demande_evenements()

    if 'souris' in evenements:
        # ici evenements['souris'] est une liste [x, y]
        print("Nouvelle abscisse : " + str(evenements['souris'][0]))

    if 'clic' in evenements:
        # ici evenements['clic'] est une liste [x, y]
        print('clic aux coordonées ' + str(evenements['clic']))
```

3.4 Rafraîchissement automatique

Par défaut, cette librairie rafraîchit automatiquement l'affichage a chaque modification de la fenêtre : création de point, de cercle, ...

Cela simplifie l'utilisation dans des cas simples mais, si vous souhaitez créer un programme complexe qui modifie fréquemment la fenêtre graphique, cela va créer des scintillements désagréables et ralentir le programme.

Vous pouvez désactiver ce rafraîchissement automatique en passant l'argument `autorefresh=fasle` à la fonction `fenetre()`. Il faudra alors appeler la fonction `rafraichir()` lorsque vous voulez rafraîchir l'affichage de la fenêtre graphique.

Notre exemple de départ devient alors :

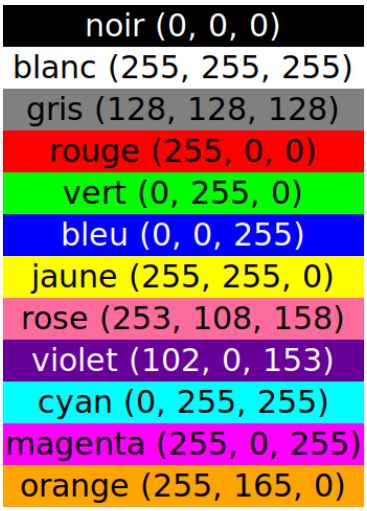
```
# On importe la librairie
from pydiderotlibs.graphique import *
# Nous aurons également de la librairie time
from time import *

# On initialise les coordonnées du point au coin haut gauche de la fenêtre
x = 0
y = 0

# On créé la fenêtre graphique en passant l'argument autorefresh=false
creer_fenetre(autorefresh=False)

# Boucle principale
while 1:
    # Il est important d'appeler la fonction demande_evenements() qui gère la␣
↪fermeture de la fenêtre
    demande_evenements()

    # Trace un cercle au coordonnées (x,y)
    trace_cercle(x, y)
    # On actualise la fenêtre graphique
    rafraichir()
    # Attend un dixième de secondes
    sleep(0.1)
    # Efface le cercle
    trace_cercle(x, y, couleur='blanc')
    # Ajoute le vecteur vitesse aux coordonnées du point
```

(suite sur la page suivante)

(suite de la page précédente)

```
    x += 1
    y += 1
```

3.5 Couleurs



Les formats de couleurs acceptés sont :

- Une chaîne de caractères (str) en francais dans la liste ci-contre (`'rouge'` par exemple).
- Une chaîne de caractères représentant la couleur sous forme hexadécimale. Par exemple la couleur rouge est `'#FF0000'`.
- Un tuple RVB `(r, v, b)` où `r`, `v`, `b` sont des entiers entre `0` et `255` . Par exemple la couleur rouge est `(255, 0, 0)`.
- Une chaîne de caractère en anglais dans la liste suivante : `'black'`, `'white'`,`'grey'`, `'red'`, `'green'`, `'blue'`, `'yellow'`, `'pink'`, `'purple'`.

3.6 Documentation

fenetre(*largeur=600*, *hauteur=500*, *orientation_axe_ordonnees=False*, *titre='Fenetre graphique'*, *autorefresh=True*)

Crée et affiche une fenêtre graphique.

Alias : `window()`, `creer_fenetre()`

Paramètres

- **largeur** (*int, optionnel*) – Largeur de la fenetre en pixels (`600` par défaut)
- **hauteur** (*int, optionnel*) – Hauteur de la fenetre en pixels (`500` par défaut)
- **orientation_axe_ordonnees** – Si on met cet argument à True, l'axe des ordonnées sera orienté de bas en haut comme en maths. Sinon il est orienté dans l'autre sens comme habituellement en informatique (`False` par défaut)
- **titre** (*str, optionnel*) – Titre de la fenetre (`Fenetre graphique` par défaut)
- **autorefresh** (*bool, optionnel*) – Active le rafraichissement automatique de la fenetre graphique (*False* par défaut)

rafraichir()

Rafraîchie la fenêtre graphique. C'est uniquement utile si vous désactivez l'option *autorefresh*.

demande_evenements()

Récupère les évenements pygame gère la fermeture de la fenetre et retourne les évenements formatés.

Renvoie un dictionnaire d'évenements formaté comme suit : `{'touche1': None, 'touche2':None, 'souris': [x,y], 'click': [x,y]}`

Les valeurs `None` pour les touches peuvent surprendre mais il est nécéssaire d'utiliser un dictionnaire pour avoir les coordonnées éventuelles de la souris lors d'un click par exemple. Pour les touches clavier, l'importance est la présence de la cléf et la valeur associée est donc `None`.

— Les caractères alphanumériques sont encodés en ascii (`'a'`, `'n'`, `';'`) et, si présent, leur valeur est `None`.
— les touches spéciales ont les clefs `'espace'`, `'haut'`, `'bas'`, `'droite'`, `'gauche'` et, si présent, leur valeur est `None`.
— Un clic avec le bouton gauche de la souris ajoute une clef `'clic'`. Sa valeur est une liste `[x, y]` des coordonnées de la souris.
— Un déplacement de la souris ajoute une clef `'souris'`. Sa valeur est une liste `[x, y]` des coordonnées de la souris.

Alias : `events()`, `ecoute_evenements()`

efface(*couleur='blanc'*)

Efface l'écran.

Paramètres **couleur** (*couleur*, optionnel) – Couleur de remplissage de l'écran (`'blanc'` par défaut).

cercle(*x*, *y*, *couleur='bleu'*, *rayon=25*, *epaisseur=0*)

Trace un cercle dans la fenetre graphique.

Alias : `circle()`, `trace_cercle()`

Paramètres
— **x** (*int*) – Abscisse du centre du cercle
— **y** (*int*) – Ordonnée du centre du cercle
— **rayon** (*int, optionnel*) – Rayon du cercle (25 par défaut)
— **epaisseur** (*int, optionnel*) – Epaisseur du cercle (`0` par défaut). Si `0`, le cercle sera rempli et apparaitra comme un disque.
— **couleur** (*couleur*, optionnel) – Couleur du cercle (bleu par défaut)

cercle_aleatoire(*couleur='bleu'*, *rayon=5*, *epaisseur=0*)

Trace un (petit) cercle dans la fenetre graphique, à un endroit choisit au hasard. (Utile pour fairre de la neige par exemple.)

Alias : `random_circle()`, `randcircle()`, `trace_cercle_aleatoire()`

Paramètres
— **rayon** (*int, optionnel*) – Rayon du cercle (5 par défaut)
— **epaisseur** (*int, optionnel*) – Epaisseur du cercle (`0` par défaut). Si `0`, le cercle sera rempli et apparaitra comme un disque.
— **couleur** (*couleur*, optionnel) – Couleur du cercle (bleu par défaut)

point(*x*, *y*, *couleur='bleu'*)

Trace un point dans la fenetre graphique.

Alias : `trace_point()`

Paramètres
— **x** (*int*) – Abscisse du point
— **y** (*int*) – Ordonnée du point
— **couleur** (*couleur*, optionnel) – Couleur du point (bleu par défaut)

rectangle(*x*, *y*, *largeur=100*, *hauteur=50*, *couleur='bleu'*, *epaisseur=0*)

Trace un rectangle horizontal dans la fenetre graphique .

Le sommet haut-gauche à pour coordonnées `(x,y)`, la `largeur` est la taille en abscisse et la `hauteur` la taille en ordonnée.

Alias : `trace_rectangle()`

Paramètres
- **x** (*int*) – abscisse du sommet haut gauche du rectangle
- **y** (*int*) – ordonnée du sommet haut gauche du rectangle
- **largeur** (*int*) – taille du rectangle sur l'axe des abscisses
- **hauteur** (*int*) – taille du rectangle sur l'axe des ordonnées
- **couleur** (*couleur*, optionnel) – Couleur du rectangle (`bleu` par défaut)
- **epaisseur** (*int, optionnel*) – Epaisseur des cotés du rectangle (`0` par défaut). Si `0`, le rectangle est rempli.

triangle(*x1*, *y1*, *x2*, *y2*, *x3*, *y3*, *couleur='bleu'*, *epaisseur=0*)

Trace un triangle dans la fenetre graphique .

Alias : `trace_triangle()`

Paramètres
- **x1** (*int*) – abscisse du premier sommet du triangle
- **y1** (*int*) – ordonnée du premier sommet du triangle
- **x2** (*int*) – abscisse du deuxième sommet du triangle
- **y2** (*int*) – ordonnée du deuxième sommet du triangle
- **x3** (*int*) – abscisse du troisième sommet du triangle
- **y3** (*int*) – ordonnée du troisième sommet du triangle
- **couleur** (*couleur*, optionnel) – Couleur du triangle (`bleu` par défaut)
- **epaisseur** (*int, optionnel*) – Epaisseur des cotés du triangle (`0` par défaut). Si `0`, le triangle est rempli.

segment(*x1*, *y1*, *x2*, *y2*, *couleur='bleu'*, *epaisseur=2*)

Trace un segment entre les points de coordonées `(x1, y1)` et `(x2, y2)`.

Alias : `trace_segment()`

Paramètres
- **x1** (*int*) – abscisse de la première extremité du segment
- **y1** (*int*) – ordonnée de la première extremité du segment
- **x2** (*int*) – abscisse de la deuxieme extrémité du segment
- **y2** (*int*) – ordonnée de la deuxieme extrémité du segment
- **couleur** (*couleur*, optionnel) – Couleur du segment (`bleu` par défaut)
- **epaisseur** (*int, optionnel*) – Epaisseur du segment (`2` par défaut)

vecteur(*x*, *y*, *v*, *couleur='rouge'*, *epaisseur=2*)

Trace la représentation du vecteur `v` à partir du point d'origine `(x, y)`.

Alias : `vector()`, `trace_vecteur()`

Paramètres
- **x** (*int*) – abscisse du point d'origine de la représentation du vecteur
- **y** (*int*) – ordonnée du point d'origine de la représentation du vecteur
- **v** (*list*) – Coordonnées de la deuxieme extrémité du segment
- **couleur** (*couleur*, optionnel) – Couleur du segment (`rouge` par défaut)
- **epaisseur** (*int, optionnel*) – Epaisseur du segment (`2` par défaut)

vecteur2(*xv*, *yv*, *couleur='rouge'*, *epaisseur=2*)

Trace la représentation du vecteur de coordonnées `(xv, yv)` à partir d'une origine choisie au hasard.

Alias : `vector2()`, `trace_vecteur2()`

Paramètres
- **xv** (*int*) – abscisse du vecteur
- **yv** (*int*) – ordonnée du vecteur
- **couleur** (*couleur*, optionnel) – Couleur du segment (`rouge` par défaut)
- **epaisseur** (*int, optionnel*) – Epaisseur du segment (`2` par défaut)

image(*x*, *y*, *nom*, *largeur=100*, *hauteur=100*)

Trace une image dans la fenetre graphique.

Alias :`trace_image()`

Paramètres
- **x** (*int*) – Abscisse du centre de l'image
- **y** (*int*) – Ordonnée du centre de l'image
- **nom** (*str*) – nom du fichier image (qui doit être dans le répertoire du script)
- **largeur** (*int, optionnel*) – Largeur de l'image (100 par défaut)
- **hauteur** (*int, optionnel*) – Hauteur de l'image (100 par défaut)

explosion(*x*, *y*, *couleur='orange'*, *r=25*, *c=0.5*, *n=10*)

Trace un polygône régulier étoilé à `2n` côté, de rayon extérieur `r`, et tel que le rayon intérieur est égal à `c*r` (pour `c=0`, le polygône est réduit à `n` rayons du cencle de rayon `r` pour `c=1`, c'est un polygône régulier à `2n` côtés)

Alias : `trace_explosion()`

Paramètres
- **x** (*int*) – Abscisse du centre de l'explosion
- **y** (*int*) – Ordonnée du centre de l'explosion
- **couleur** (*couleur*, optionnel) – Couleur (`orange` par défaut)
- **r** (*int*) – Rayon extérieur
- **c** (*float*) – Coefficient pour obtenir le rayon intérieur égal à `c*r`
- **n** (*int*) – Nombre de sommets

axes(*color='noir'*)

Dessine les axes de coordonnées pour une meilleure compréhension par les élèves.

Alias : `trace_axes()`

texte(*message*, *x*, *y*, *police=''*, *taille=12*, *couleur='noir'*)

Affiche un texte dans la fenetre graphique.

Paramètres
- **message** (*str*) – le texte à afficher
- **x** (*int*) – abscisse du début du texte
- **y** (*int*) – ordonnée du haut du début du texte
- **police** (*str, optionnel*) – la police de caractère à utiliser. Si non renseigné ou si la police n'est pas installée, on utilise la police de caractère défaut du system
- **taille** (*int, optionnel*) – taille du texte
- **couleur** (*couleur*, optionnel) – couleur du texte

polices_disponibles()

Retourne les polices caractères disponibles

test_police(*police*)

Test si une police de caractère est installée.

Paramètres **police** (*str*) – police de caractère à tester

Renvoie True si la police est installée, False sinon

CHAPITRE 4

Images

4.1 A propos

Cette librairie a été conçue pour l'enseignement de la SNT.

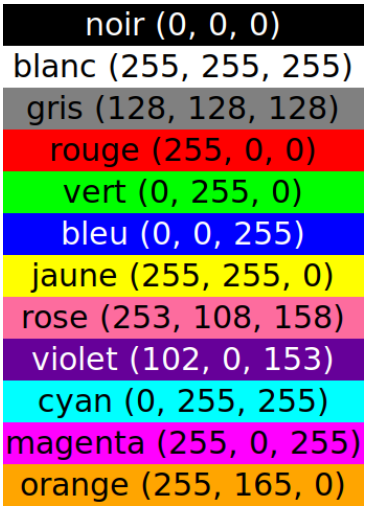
Elle permet de manipuler des images assez facilement.

L'objectif est de permettre aux élèves d'accéder au contenu des pixels d'une image numérique (enregistrée dans le même répertoire que le fichier python sur lequel on travaille), de modifier les valeurs des composantes des pixels et finalement de rendre possible l'exécution de petits scripts de manipulation d'images.

La librairie est basée sur PIL et utilise les noms de couleurs de pydiderotlibs, comme détaillé dans la section suivante.

Vous pouvez, par exemple, l'utiliser pour créer un script convertissant une image couleurs en niveaux de gris ou encore un script générant le négatif d'une image donnée.

4.2 Couleurs



Les formats de couleurs acceptés sont :

- Une chaîne de caractères (str) en francais dans la liste ci-contre (`'rouge'` par exemple).
- Une chaîne de caractères représentant la couleur sous forme hexadécimale. Par exemple la couleur rouge est `'#FF0000'`.
- Un tuple RVB `(r, v, b)` où `r`, `v`, `b` sont des entiers entre `0` et `255` . Par exemple la couleur rouge est `(255, 0, 0)`.
- Une chaîne de caractère en anglais dans la liste suivante : `'black'`, `'white'`,`'grey'`, `'red'`, `'green'`, `'blue'`, `'yellow'`, `'pink'`, `'purple'`.

4.3 Utilisation

Voici quelques exemples :

```
# On importe la librairie
from pydiderotlibs.images import *

# On affiche dans la console la composante bleue du pixel (100,200,300) (le résultat␣
↪est '300')
print(bleu((100,200,300)))

# On crée une version de l'image ``essai.png`` qui sera manipulable par Python et qui␣
↪sera stockée sous le nom de variable ``img``
img = creer_image('essai.png')

# On affiche la définition de l'image, puis sa largeur.
print(definition_image(img))
print(largeur_image(img))

# On affiche l'image dans la visionneuse du système d'exploitation
afficher_image(img)

# On affiche le pixel de composantes RVB (100,100,100) (c'est un pixel de couleur␣
↪grise)
afficher_pixel((100, 100, 100))
```

(suite sur la page suivante)

(suite de la page précédente)

```
# On affiche les trois composantes RVB du pixel situé juste à droite du pixel de␣
↪coordonnées (100,100).
# Ce pixel voisin à pour coordonnées (101,100)
print(pixel_voisin(img, (100, 100)))

# On affiche les 3 composantes RVB du pixel de coordonnées (10,10)
print(copier_pixel(img, (10, 10)))

# On remplace le pixel de coordonnées (10,10) par un pixel noir (0,0,0)
coller_pixel(img, (10, 10), (0, 0, 0))

# On définit une fonction qui convertit un pixel donné en pixel "niveau de gris"
def gris(pixel):
    g = int((rouge(pixel) + vert(pixel) + bleu(pixel)) / 3)
    return (g, g, g)

# On applique la fonction ``gris()`` précédente aux pixels de l'image ``img`` dont␣
↪les coordonnées sont situées dans le rectangle de diagonale '(100,300)--(200,500)'
changer_les_pixels(img,gris,100,200,300,500)
```

4.4 Documentation

importer_image (*fichier*)

Crée, à partir d'une image enregistrée sur l'ordinateur (dans le même dossier que le fichier python sur lequel on travaille), une image manipulable dans le langage python (avec la librairie PIL)

Arguments : fichier : nom du fichier enregistré sur l'ordinateur (avec le suffixe : par exemple « monimage.png ») Le plus simple est d'avoir le fichier de l'image dans le même dossier que le fichier python sur lequel on travaille, sinon il faut mettre le chemin comme par exemple « U :Documentsmonimage.jpg »

Alias : creer_image()

importer_image_auto (*fichier*)

Crée, à partir d'une image enregistrée sur l'ordinateur (dans le même dossier que le fichier python sur lequel on travaille), une image manipulable dans le langage python (avec la librairie PIL)

Arguments : fichier : nom du fichier enregistré sur l'ordinateur (avec le suffixe : par exemple « monimage.png ») Le plus simple est d'avoir le fichier de l'image dans le même dossier que le fichier python sur lequel on travaille Si l'image n'est pas trouvée dans ce dossier, les dossiers "téléchargement" et "images" de Windows sont explorés. Si aucune extension de fichier n'est indiquée, une série d'extensions correspondants aux principaux formats d'images est essayée.

Alias : importer_image_panic() creer_image_auto() creer_image_panic()

definition_image (*image*)

Retourne la définition de l'image `image`. C'est une liste de deux nombres. Le premier est la largeur (en pixels), le second la hauteur (en pixels).

Par exemple : (200,300)

Arguments : img : nom de la variable Python contenant l'image

largeur_image (*image*)

Retourne la largeur de l'image `image` (en pixels).

Arguments : img : nom de la variable Python contenant l'image

hauteur_image (*image*)

Retourne la hauteur de l'image `image` (en pixels).

Arguments : img : nom de la variable Python contenant l'image

afficher_image(*image*)
Affiche `image` (attention : sans l'enregistrer !).

Arguments : img : nom de la variable Python contenant l'image

afficher_pixel(*pixel*)
Affiche une petite image unie, de 10x10 pixels, de la couleur du pixel `pixel`.

Arguments : pixel : un pixel, c'est à dire un triplet de trois nombres entiers entre 0 et 255. Par exemple : (0,0,0) (pixel noir) ou : (255,255,255) (pixel blanc)

rouge(*pixel*)
Donne la valeur de rouge du pixel `pixel`.

Arguments : pixel : un pixel, c'est à dire un triplet de trois nombres entiers entre 0 et 255. Par exemple : (0,0,0) (pixel noir) ou : (255,255,255) (pixel blanc)

vert(*pixel*)
Donne la valeur de vert du pixel `pixel`.

Arguments : pixel : un pixel, c'est à dire un triplet de trois nombres entiers entre 0 et 255. Par exemple : (0,0,0) (pixel noir) ou : (255,255,255) (pixel blanc)

bleu(*pixel*)
Donne la valeur de bleu du pixel `pixel`.

Arguments : pixel : un pixel, c'est à dire un triplet de trois nombres entiers entre 0 et 255. Par exemple : (0,0,0) (pixel noir) ou : (255,255,255) (pixel blanc)

pixel_voisin(*image*, *coord*)
Donne les coordonnées du pixel situé à droite du pixel `pixel`. (S'il n'y a pas de pixel plus à droite car on est au bord de l'image, retourne les mêmes coordonnées.)

Arguments : coord : coordonnées du pixel image : nom de la variable Python contenant l'image

copier_pixel(*image*, *coord*)
Retourne le pixel de l'image `image`, situé aux coordonnées `coord`.

Arguments : coord : coordonnées du pixel image : nom de la variable Python contenant l'image

coller_pixel(*image*, *coord*, *pixel*)
Remplace le pixel de l'image `image`, situé aux coordonnées `coord`, par le pixel `pixel`.

Arguments : coord : coordonnées du pixel image : nom de la variable Python contenant l'image pixel : un pixel, c'est à dire un triplet de trois nombres entiers entre 0 et 255. Par exemple : (0,0,0) (pixel noir) ou : (255,255,255) (pixel blanc)

enregistrer_image(*image*, *nom*)
Enregistre l'image `image`, avec le nom de fichier indiqué.

Arguments : image : nom de la variable Python contenant l'image nom : nom du fichier avec extension, par exemple « monimage.png »

changer_les_pixels(*image*, *fonction*, *x0=0*, *x1=0*, *y0=0*, *y1=0*)
Modifie les pixels de l'image `image`, en leur appliquant la fonction `fonction`.

Arguments : image : nom de la variable Python contenant l'image fonction : fonction à appliquer aux pixels. Ce doit être une fonction qui prend un pixel en argument et qui retourne un pixel x0 (optionnel) : valeur minimale de la première coordonnée des pixels à modifier (0 par défaut) x1 (optionnel) : valeur maximale de la première coordonnée des pixels à modifier (0 par défaut, ce qui signifie que les pixels seront modifiés jusqu'à xmax) y0 (optionnel) : valeur minimale de la deuxième coordonnée des pixels à modifier (0 par défaut) y1 (optionnel) : valeur maximale de la deuxième coordonnée des pixels à modifier (0 par défaut, ce qui signifie que les pixels seront modifiés jusqu'à ymax)

CHAPITRE 5

Lycee

5.1 A propos

Le module lycee a pour objectif de simplifier un certain nombre de manipulations avec python au lycée (cosinus en degré, calcul d'une moyenne d'une liste, représentation statistiques variées, . . .)

Grandement inspiré par le travail du groupe AMIENS PYTHON sous licence CECILL.

Pour l'utiliser, il suffit d'ajouter en début de programme

```
from pydiderotlibs.lycee import *
```

5.2 Documentation

Module avec les fonctions de la classe de Seconde 2018 pour le lycée diderot (marseille). On prend comme fichier de départ le module de l'irem d'Amiens http://download.tuxfamily.org/amienspython/lycee.py Licence http://www.cecill.info/

repeter(*f*, *n*)

Appelle *n* fois la fonction `f`.

Alias disponible : `repeat()`

alea_entre_bornes(*a*, *b*, *p=15*)

Choisit un nombre (pseudo) aléatoire entre `a` et `b` avec `p` décimales.

Paramètres

- **a** (*float*) – valeur minimale
- **b** (*float*) – valeur maximale
- **p** (*integer, optionnel*) – nombre de décimales s'il est compris entre 0 et 15. (`15` par défaut)

Alias disponible : `random_between()`

random() → x in the interval [0, 1).

5.2.1 Trigonométrie

Partie trigonométrie du module lycee. Deux familles de fonctions : `_degré` et `_radian`.

cos_radian(*angle*)
Renvoie le cosinus de l'angle.

Paramètres **angle** (*float*) – La mesure d'un angle en radians

sin_radian(*angle*)
Renvoie le sinus de l'angle.

Paramètres **angle** (*float*) – La mesure d'un angle en radians

tan_radian(*angle*)
Renvoie la tangente de l'angle.

Paramètres **angle** (*float*) – La mesure d'un angle en radians

acos_radian(*x*)
Renvoie un angle en radians dont le cosinus vaut `x`.

Paramètres **x** (*float*) – Un nombre entre -1 et 1

asin_radian(*x*)
Retourne un angle en radians dont le sinus vaut `x`.

Paramètres **x** (*float*) – Un nombre entre -1 et 1

atan_radian(*x*)
Renvoie un angle en radians dont la tangente vaut `x`.

Paramètres **x** (*float*) – Un nombre entre -1 et 1

cos_degre(*angle*)
Renvoie le cosinus de l'angle (type''float'').

Paramètres **angle** (*float*) – La mesure d'un angle en degré

sin_degre(*angle*)
Renvoie le sinus de l'angle (type''float'').

Paramètres **angle** (*float*) – La mesure d'un angle en degré

tan_degre(*angle*)
Renvoie la tangente de l'angle (type''float'').

Paramètres **angle** (*float*) – La mesure d'un angle en degré.

acos_degre(*x*)
Renvoie un angle en degré (float) dont le cosinus vaut `x`.

Paramètres **x** (*float*) – Un nombre réel

asin_degre(*x*)
Renvoie un angle en degrés (foat) dont le sinus vaut `x`.

Paramètres **x** (*float*) – Un nombre réel

atan_degre(*x*)
Renvoie un angle en degrés (foat) dont le sinus vaut `x`.

Paramètres **x** (*float*) – un nombre réel

5.2.2 Arithmétique

Partie arithmetique du module lycee.

pgcd(*a*, *b*)
Renvoie le Plus Grand Diviseur Communs des entiers `a` et `b`.

Paramètres
— **a** (*int*) – un nombre entier
— **b** (*int*) – un nombre entier

reste(*a*, *b*)
Renvoie le reste de la division de `a` par `b`.

Paramètres
— **a** (*int*) – Un nombre entier.
— **b** (*int*) – Un nombre entier non nul.

quotient(*a*, *b*)
Le quotient de la division de `a` par `b`.

Paramètres
— **a** (*int*) – Un nombre entier.
— **b** (*int*) – Un nombre entier non nul.

bezout(*a*, *b*)
Renvoie un triplet d'entiers (d,u,v) tel que u*a + b*v = d = pgcd(a,b).

Paramètres
— **a** (*int*) – un nombre entier
— **b** (*int*) – un nombre entier

puissance_mod(*n*, *p*, *m*)
Renvoie n^p modulo m.

Paramètres
— **n** (*int*) – un nombre entier
— **p** (*int*) – un nombre entier
— **m** (*int*) – un nombre entier

5.2.3 Fonctions usuelles

Partie fonctions fonctions_usuelles du module lycee.

puissance(*a*, *n*)
Renvoie le nombre réel (float) (a^n).

Paramètres
— **a** (*float*) – un nombre décimal.
— **b** (*float*) – un nombre décimal représentant l'exposant.

carre(*a*)
Renvoie le carre d'un nombre reel (float)

Paramètres **a** (*float*) – un nombre décimal.

racine(*x*)
Renvoie le nombre réel (float) racine carré de `x` (sqrt{x}).

Paramètres
— **a** (*float*) – un nombre décimal.
— **b** (*float*) – un nombre décimal représentant l'exposant.

factoriel(*n*)
Renvoie (n ! = n x (n-1) x ... x 3 x 2 x 1)

Paramètres **n** (*int*) – Un nombre entier positif.

partie_entiere(*x*)
Renvoie la partie entiere du nombre x, c'est a dire le plus grand entier inferieur au reel x.

Paramètres **x** (*float*) – Un nombre décimal.

sans_virgule(*x*)
Retourne la partie du nombre x sans sa partie décimale. Ex : -2.5 devient -2

Paramètres **x** (*float*) – Un nombre decimal.

exp(*x*)
Renvoie l'image du nombre x par la fonction exponentielle : (e^x).

Paramètres **x** (*float*) – Un nombre decimal.

ln(*x*)
Renvoie l'image du nombre x par la fonction logarithme népérien : (ln x).

Paramètres **x** (*float*) – Un nombre décimal strictement positif.

log(*x*)
Renvoie l'image du nombre x par la fonction logarithme décimal : (log x).

Paramètres **x** (*float*) – Un nombre décimal strictement positif.

5.2.4 Vecteurs

Partie vecteurs du module lycee.

vecteur(*x*, *y*, *z=None*)
Renvoie un vecteur de coordonées `(x,y)` ou `(x,y,z)`.

Paramètres
- **x** (*float*) – Abscisse du vecteur
- **y** (*float*) – Ordonnée du vecteur
- **z** (*float, optionnel*) – Cote du vecteur.

norme(*v*)
Renvoie la norme du vecteur v.

Paramètres **v** (*array*) – Un vecteur du plan ou de l'espace

abscisse(*v*)

Renvoie l'abscisse du vecteur v.

Arguments : v (array) : Un vecteur du plan ou de l'espace

ordonnee(*v*)
Renvoie l'ordonnée du vecteur v.

Paramètres **v** (*array*) – Un vecteur du plan ou de l'espace

cote(*v*)
Renvoie la cote du vecteur v.

Paramètres **v** (*array*) – Un vecteur de l'espace

5.2.5 Listes

Partie listes du module lycee.

CSV_ligne(*num*, *fichier='optionnel'*)

Retourne une liste de nombres (float) correspondant à la ligne du fichier `fichier`.

Si fichier n'est pas précisé, ouvre une boite de dialogue pour le choisir Le fichier ne doit contenir que des nombres et le séparateur doit être **;**

Paramètres

- **num** (*int*) – Un numéro de ligne.
- **fichier** (*file, optionnel*) – Le nom complet (avec le chemin) d'un fichier contenant des nombres.

CSV_colonne(*num*, *fichier='optionnel'*)

Retourne une liste de nombres (float) correspondant à la colonne du fichier `fichier`.

Si fichier n'est pas précisé, ouvre une boite de dialogue pour le choisir Le fichier ne doit contenir que des nombres et le séparateur doit être **;**

Paramètres

- **num** (*int*) – Un numéro de colonne.
- **fichier** (*file, optionnel*) – Le nom complet (avec le chemin) d'un fichier contenant des nombres.

liste2CSV(*L*, *fichier='optionnel'*)

Enregistre sous le nom `fichier` la liste `L`.

Si fichier n'est pas précisé, ouvre une boite de dialogue pour le choisir

Paramètres

- **L** (*list*) – Une liste
- **fichier** (*file, optionnel*) – Le nom complet (avec le chemin) d'un fichier contenant du texte brut.

trier(*liste1*, *liste2=[]*)

Retourne *liste1* triée. Si *liste2* est renseignée, elle est réorganisée de la même manière que *liste1*. Ex 1 : trier([5,3,4])=[3,4,5]

Ex 2 : *liste1=[3,2,1]* et *liste2=[10,20,30]*. trier(liste1,liste2)=([1,2,3],[30,20,10])

Paramètres

- **liste1** (*list*) – Une liste avec UNIQUEMENT des nombres OU UNIQUMENT des chaines de caractères
- **liste2** (*list, optionnel*) – Une liste quelconque mais de même taille que *liste1*

transposer(*L*)

L est une liste de liste, comme une matrice NxM. On prend la transposée. Ex : L=[[a,b,c],[1,2,3]]. Retourne la liste [[a,1],[b,2],[c,3]]

Paramètres **L** (*list*) – une liste de listes

serie(*deb*, *fin*, *pas=1*)

Renvoie une liste de nombre (float) de *deb* à *fin* (inclu) avec un *pas*

Arguments deb (float ou int) : début de la série fin (float ou int) : fin de la série pas (float ou int, optionnel) : pas de la série

affiche_poly(*L*)

Affiche la liste L sous forme d'un polynôme (L[n] étant le coefficient de degré n).

Paramètres **L** (*list*) – Une liste

5.2.6 Chaines de caractères

Partie chaines de caractères du module lycee. Créé à partir d" Edupython : http://edupython.tuxfamily.org/

Licence CECILL http://www.cecill.info/

taille(*objet*)

Retourne la longueur de cette chaine ou de cettte liste.

Paramètres **objet** (*str ou list*) – Une chaine de caractères ou une liste.

fich2chaine(*fichier='optionnel'*, *message=''*)

Retourne chaine formée du contenu du fichier `fichier`.

Si `fichier` n'est pas précisé, ouvre une boite de dialogue pour le sélectionner.

Paramètres **fichier** (*file, optionnel*) – Nom complet (avec le chemin) d'un fichier contenant du texte brut.

chaine2fich(*ch*, *fichier='optionnel'*)

Enregistre sous le nom `fichier` la chaine `ch`.

Si fichier n'est pas précisé, ouvre une boite de dialogue pour le sélectionner.

Paramètres

- **ch** (*str*) – Une chaine de caractères
- **fichier** (*file, optionnel*) – Le nom complet (avec le chemin) d'un fichier contenant du texte brut.

5.2.7 Stats et Proba

Partie statistiques et probobilité du module lycee.

binomial(*n*, *p*)

Renvoie un entier (int) représentant le coefficient binomial `p` parmi `n`.

C'est à dire le nombre de chemins de l'arbre réalisant `p` succès pour `n` répétitions.

Paramètres

- **n** (*int*) – Un nombre entier
- **p** (*int*) – Un nombre entier

tirage_binomial(*n*, *p*)

Renvoie un nombre entier (int) choisi de manière aléatoire selon une loi binomiale B(n,p) : `p` parmi .

Paramètres

- **n** (*int*) – Premier paramètre de la loi binomiale à simuler.
- **p** (*int*) – Second paramètre de la loi binomiale à simuler.

alea_entier(*min*, *max*)

Renvoie un entier (int) choisi de manière (pseudo)aléatoire et équiprobable dans l'intervalle [`min` ; `max`].

Paramètres

- **min** (*int*) – Un nombre entier
- **max** (*int*) – Un nombre entier

tirage_uniforme(*min*, *max*)

Renvoie un nombre décimal (float) choisi de manière (pseudo)aléatoire et uniforme de l'intervalle [`min` ; `max`[.

Paramètres

- **min** (*float*) – Un nombre réel.
- **max** (*float*) – Un nombre réel.

choix(*liste*)

Renvoie un élément de la liste `liste` choisi (pseudo)aléatoirement et de manière équipropable

Paramètres **liste** (*int*) – La liste dans laquelle on choisit un élément.

alea()
Renvoie au hasard un décimal de l'intervalle [0 ; 1[

tirage_expo(*x*)
Renvoie un nombre décimal (float) choisi de manière aléatoire selon une loi exponentielle de paramètre `x`.

Paramètres **x** (*float*) – est un réel strictement positif.

repartition_normale(*x*, *mu=0*, *sigma=1*)
Renvoie la probabilité P(y < x) pour P loi normale de moyenne mu et d'écart type sigma.

Paramètres
- **x** (*float*) –
- **mu** (*float*) – Un nombre décimal. L'éspérance de la loi normale
- **sigma** (*float*) – Un nombre décimal. L'écart type de la loi normale

tirage_normale(*mu*, *sigma*, *precision=15*)
Renvoie un nombre décimal (float) choisi de manière aléatoire selon une loi nomale d'espérance `mu` et d'écart type `sigma`.

Paramètres
- **mu** (*float*) – Un nombre décimal. L'éspérance de la loi normale
- **sigma** (*float*) – Un nombre décimal. L'écart type de la loi normale

tirage_gauss(*mu*, *sigma*)
Renvoie un nombre décimal (float) choisi de manière aléatoire selon une loi nomale d'espérance `mu` et d'écart type `sigma`. :param mu : Un nombre décimal. L'éspérance de la loi normale :type mu : float :param sigma : Un nombre décimal. L'écart type de la loi normale :type sigma : float

compte(*liste_critere*, *liste_effectif=[]*)
Retourne la liste non triée, sans les doublons, les effectifs, les fréquences correspondantes et l'effectifs totales. La fonctionne NE TRIE PAS par ordre des critères : les critères n'ont pas nécessairement un ordre. Pour trier le résultat, utililser la fonctions trier()

Paramètres
- **liste_critere** (*list*) – une liste de critère d'une population
- **liste_effectif** (*list, optionnel*) – liste des frequences ou des effectifs des critères

centres(*L*)
Renvoie une liste de longueur n-1 contenant les valeurs (L[i]+L[i+1])/2.

Paramètres **L** (*list*) – Une liste de taille n

moyenne(*xi*, *ni=[]*)
Renvoie la moyenne de la liste `xi`.

Paramètres
- **xi** (*list*) – liste de valeurs
- **ni** (*liste, optionnel*) – série des effectifs ou des fréquences associés

variance(*xi*, *ni=[]*)
Retourne la variance de la liste.

Paramètres
- **xi** (*list*) – Liste de valeurs
- **ni** (*list, optionnel*) – Liste des effectifs associés

ecartype(*xi*, *ni=[]*)
Retourne l'écart-type de la liste.

Paramètres
- **xi** (*list*) – Liste de valeurs
- **ni** (*list, optionnel*) – Liste des effectifs associés

CHAPITRE 6

Repere

6.1 A propos

Cette bibliothèque facilite l'affichage d'une fenêtre munie d'un repère interactif (zoom, déplacement). Des fonctions sont disponibles pour tracer des objets géométriques simples.

C'est une Version légèrement modifiée de cette librairie écrite par Olivier Brebant en 2011 et publié par l'académie d'Aix-Marseille en 2012, sous licence MIT depuis novembre 2018.

6.2 A quoi ca sert ?

A tracer un graphique **non dynamique** muni (ou non) d'un repère. On peut par exemple construire facilement un traceur de courbes.

Par non dynamique, j'entends qu'on ne peut pas facilement utiliser cette librairie pour interagir directement avec les objets. Par exemple cette librairie n'est pas adaptée au codage d'un jeu de type pong ou même d'un objet en mouvement.

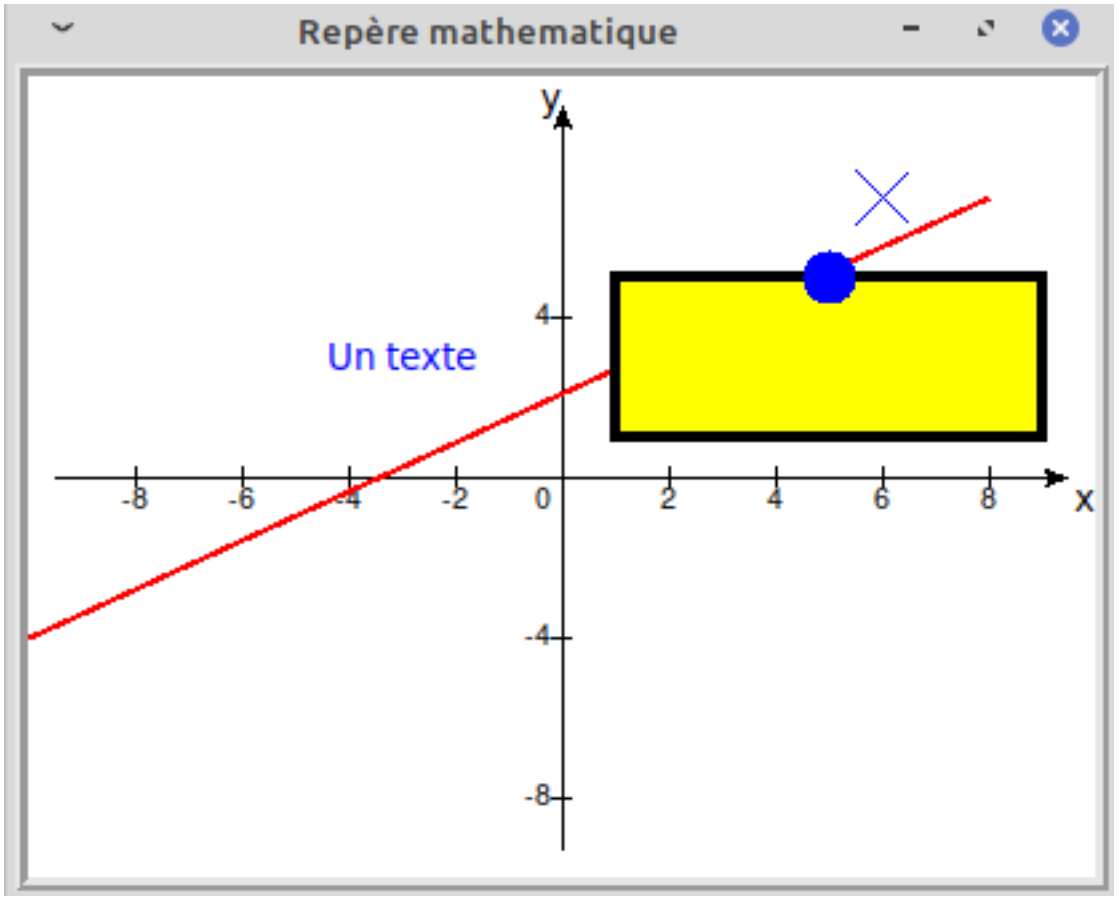
6.3 Comment l'utiliser

```
# On importe la librairie
from pydiderotlibs.repere import *

# On initialise la fenetre
creer_fenetre()

# On créé des objects geométriques
point(5, 3)
segment(-10, -4, 8, 7, couleur='rouge', taille=2)
rectangle(1, 1, 8, 4, couleur='noir', taille=4, remplissage='jaune')
```

(suite sur la page suivante)

(suite de la page précédente)

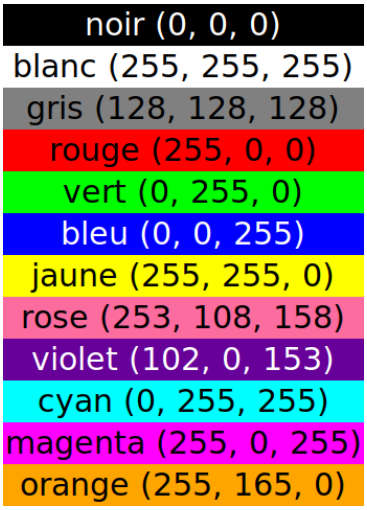
```
point(5, 5, couleur='bleu', taille=5)
point(6, 7, couleur='bleu', taille=5, forme='croix')
texte(-3, 3, "Un texte", couleur='bleu')
```



On peut agir sur le repère grâce à la souris :

- La *roulette* de la souris :
  - Si le curseur est proche d'un axe, modifie l'échelle uniquement pour cet axe.
  - Sinon zoome/dézoome par rapport à la position courante du curseur
- Un *Cliquer-glisser* :
  - Si le curseur est proche d'un axe, translate le repère suivant cet axe.
  - Sinon déplace le repère suivant le mouvement libre de la souris.
- Un *double clic* rend le repère orthonormé en se basant sur l'axe des abscisses.

6.4 Couleurs



Les formats de couleurs acceptés sont :

- Une chaîne de caractères (str) en francais dans la liste ci-contre (`'rouge'` par exemple).
- Une chaîne de caractères représentant la couleur sous forme hexadécimale. Par exemple la couleur rouge est `'#FF0000'`.
- Un tuple RVB `(r, v, b)` où `r`, `v`, `b` sont des entiers entre `0` et `255` . Par exemple la couleur rouge est `(255, 0, 0)`.
- Une chaîne de caractère en anglais dans la liste suivante : `'black'`, `'white'`,`'grey'`, `'red'`, `'green'`, `'blue'`, `'yellow'`, `'pink'`, `'purple'`.

6.5 Techniquement

Le coté non dynamique mentionné plus haut vient de l'utilisation de Tkinter. C'est techniquement possible avec la méthode after mais pas réaliste dans un cadre pédagogique.

La principale modification par rapport à la version de Olivier Brebant est de créer une variable globale `fenetre`. Cela simplifie l'utilisation pour les élèves auquel on cache le coté méthodes et attributs de la programmation orientée objet. Concrètement, on passe d'une utilisation :

```
fen = creer_fenetre()
fen.trace_point(5, 5, couleur='blue', taille=5)
fen.loop()
```

à

```
creer_fenetre()
trace_point(5, 5, couleur='blue', taille=5)
```

6.6 Documentation

Ce module a ete ecrit par Olivier Brebant en aout 2011.

On peut l'utiliser librement sous licence MIT

point(*x*, *y*, *couleur='noir'*, *taille=1*, *forme='rond'*)

Ajoute un point dans la fenetre graphique aux coordonees `(x, y)`.

Alias disponible : `trace_point()`

Paramètres

- **x** (*float*) – abscisse du point
- **y** (*float*) – ordonnée du point
- **couleur** (*couleur*, optionnel) – couleur du point (`noir` par défaut)
- **taille** (*int, optionnel*) – taille du point (`1` par défaut)
- **forme** (*str, optionnel*) – forme du point : rond/croix (`'rond'` par défaut)

cercle(*x*, *y*, *couleur='noir'*, *taille=5*)

Trace un disque dont le centre a pour coordonnées `(x, y)` et dont le rayon est 2 fois `taille` (en pixels).

Alias : `trace_cercle()`, `circle()`

Paramètres

- **x,y** (*float*) – Coordonnées du centre.
- **couleur** – Couleur du disque (`noir` par défaut).
- **taille** (*int, optionl*) – demi rayon (`2` par défaut).

texte(*x*, *y*, *message*, *couleur='noir'*)

Trace un texte dans la fenêtre graphique au coordonnées `x, y`.

Alias : "trace_texte()" et "text()"

Paramètres

- **x** (*float*) – abscisse du point
- **y** (*float*) – ordonnée du point
- **message** (*str*) – Texte à placer dans la fenêtre graphique
- **couleur** – Couleur du texte (`noir` par défaut)

segment(*x1*, *y1*, *x2*, *y2*, *couleur='noir'*, *taille=2*)

Trace un segment entre les points de coordonnées `(x1, y1)` et `(x2, y2)`.

Alias : `trace_segment()`

Paramètres

- **x1,y1,x2,y2** (*float*) – Coordonnées des extrémités du segment.
- **couleur** – Couleur du segment (`noir` par défaut).
- **taille** (*int, optionl*) – Epaisseur du segment (`2` par défaut).

rectangle(*x1*, *y1*, *largeur*, *hauteur*, *couleur='noir'*, *taille=2*, *remplissage='jaune'*)

Trace un rectangle dont le sommet en bas à gauche a pour coordonnées `(x1, y1)`.

Alias : `trace_rectangle()`

Paramètres

- **x1,y1** (*float*) – Coordonnées du sommet en bas à gauche du rectangle.
- **largeur,hauteur** (*float*) – Largeur et hauteur du rectangle
- **couleur** – Couleur des cotés du rectangle (`noir` par défaut).
- **taille** (*int, optionnel*) – épaisseur des cotés du rectangle. (`2` par défaut).
- **remplissage** (*str, optionnel*) – Couleur de l'intérieur du rectangle (`yellow` par default)

fenetre(*xmin=-10*, *xmax=10*, *ymin=-10*, *ymax=10*, *fond='blanc'*, *titre='Repère mathematique'*, *axes=True*)

Initialise l'object fenetre graphique sans l'afficher.

Alias : `creer_fenetre()` et `window()`

Paramètres

- **titre** (*str*) – Titre de la fenêtre. La valeur par défaut est `Repère mathematique`.
- **xmin,xmax,ymin,ymax** (*float*) – Dimensions du repère. Les valeurs par défaut sont -10, 10, -10, 10

— **fond** – Couleur de fond de la fenêtre (`blanc` par défaut).
— **axes** (*bool, optionnel*) – Affiche les axes du repère si `True` (`True` par défaut).

courbe(*valeurs=[(1, 1), (2, 2)], couleur='noir', type='courbe', taille=2, forme='rond'*)

Crée automatiquement une courbe en fonction des points donnés.

Paramètres

— **valeurs** (*list*) – Les points par lesquelles passe la courbe. La valeur par défaut est `[(1, 1), (2,2)]`.
— **couleur** – Couleur des cotés du rectangle (`noir` par défaut).
— **type** (*str, optionnel*) – Le type de courbe à creer (`double`, `courbe` ou `points`)
— **taile** (*int, optionel*) – La taille de la courbe (“2‘‘par default)
— **forme** (str, requis pour *points*) – La forme des points (`rond` par défault)

Index des modules Python

p

Index

T

V