

# Informatique

## **10**

## **Matplotlib**

*Cours*

<b>Tracés de courbes .....</b>	<b>3</b>
<b>1.I. Introduction .....</b>	<b>3</b>
<b>1.II. Import de la librairie.....</b>	<b>3</b>
<b>1.III. Un exemple basique .....</b>	<b>4</b>
<b>1.IV. Les options .....</b>	<b>5</b>
<b>1.V. Gestion de plusieurs figures .....</b>	<b>6</b>
<b>1.VI. Objet figure .....</b>	<b>9</b>
<b>1.VII. Le tout en une fonction .....</b>	<b>9</b>
<b>1.VIII. Tracer une fonction <math>f(x)</math>.....</b>	<b>10</b>
<b>1.IX. Réalisation d'un histogramme.....</b>	<b>10</b>
<b>1.X. Mise à jour d'une courbe dans le cadre d'une simulation .....</b>	<b>11</b>
1.X.1 Animation.....	11
1.X.2 Stockage d'images .....	13
1.X.3 Réalisation d'une vidéo.....	14
1.X.3.a Editeur de vidéo Windows .....	14
1.X.3.b ffmpeg sous Python .....	15

# Tracés de courbes

## 1.I. Introduction

Nous allons apprendre à tracer des courbes. Il ne me semble pas utile de détailler l'utilité de ce paragraphe...

Il est possible d'aller loin dans les possibilités qu'offrent les options de python. Nous ne développerons ici que les bases utiles le plus généralement. L'utilisateur qui souhaitera aller plus loin pourra trouver tout ce dont il a besoin sur internet en tapant simplement ce qu'il souhaite effectuer.

Il existe différentes méthodes pour tracer des courbes, je vous en propose une que j'ai adoptée et qui fonctionne bien.

Sachez que la création de courbes sous Python nécessite d'avoir deux listes, l'une pour les abscisses, l'autre pour les ordonnées. On n'écrit donc pas comme dans les calculatrices graphiques la fonction directement. C'est là une différence assez importante qu'il faut avoir comprise.

## 1.II. Import de la librairie

Pour tracer des courbes, il est nécessaire d'importer la librairie associée en écrivant :

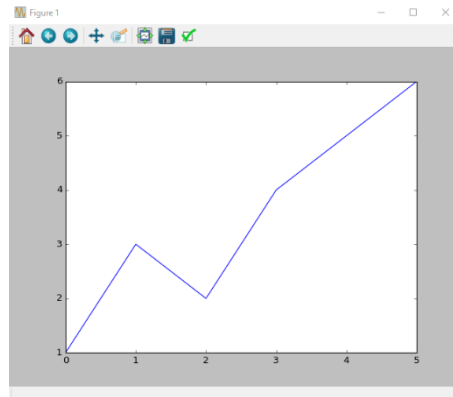
```
import matplotlib.pyplot as plt
```

Le fait de rajouter « `as plt` » permet par la suite de ne pas écrire `pyplot` mais juste `plt`. En fait, on change le nom de la fonction `pyplot` en `plt`.

### 1.III. Un exemple basique

```
X = [0,1,2,3,4,5]
Y = [1,3,2,4,5,6]
plt.plot(X,Y)
plt.show()
```

Le résultat est le suivant :



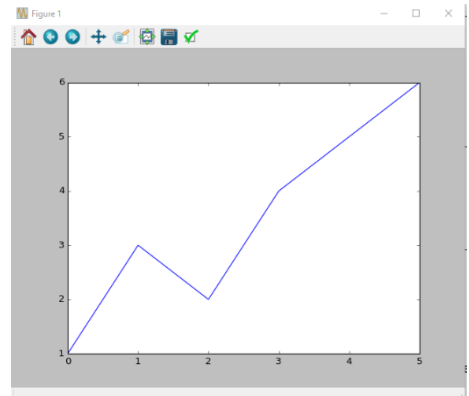
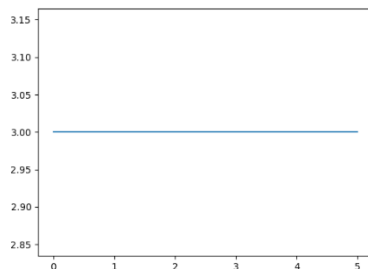
« `plt.plot(X,Y)` » met en mémoire un graphique affiché grace à la commande « `plt.show()` ».

Sans fermer la figure qui vient de s'ouvrir, modifiez votre liste Y pour celle-ci et exécutez le code :

```
Y = [3,3,3,3,3,3]
```

Vous remarquerez que la figure ne s'est pas mise à jour 😞

Fermez la fenêtre, et réexécutez votre code :



Oui, la mise à jour de la figure nécessite de fermer les fenêtres après un changement du code et une nouvelle exécution. Mais il y a une solution ! Il suffit d'ajouter, juste après l'import du module matplotlib, la commande « `plt.close('all')` ». Elle fermera toutes les fenêtres au début de l'exécution, et permettra ainsi de voir les courbes souhaitées. Je le mets dans tous mes codes.

```
import matplotlib.pyplot as plt
plt.close('all')
```

## 1.IV. Les options

Il existe une multitude d'options permettant de changer les couleurs, styles de traits, épaisseurs, titres des graphiques, titres des axes etc... Voyons ici les plus importants.

Lors de la création du plot, on peut préciser :

<code>plt.plot(X, Y, linewidth=2.0)</code>	On règle ainsi l'épaisseur du trait
<code>plt.plot(X, Y, 'o')</code>	Crée un nuage de points
<code>plt.plot(X, Y, '--')</code>	Le trait est en pointillés
<code>plt.plot(X,Y, 'r')</code>	On précise la couleur associée à la courbe : <i>b: blue - g: green - r: red - c: cyan - m: magenta - y: yellow - k: black - w: white - p: aléatoire dans une liste prédéfinie</i>
<code>plt.plot(X,Y, label="Texte")</code> <code>plt.legend()</code>	Ajout d'une légende à la courbe y(x) Ligne nécessaire pour afficher la légende
<code>plt.plot(X, Y, '--r')</code>	Options cumulées : Pointillés rouges

Après avoir créé un graphique via la commande « `plt.plot(X,Y)` » :

<code>plt.xlim(-2,2)</code>	Définit l'intervalle des abscisses de la figure
<code>plt.ylim(-2,2)</code>	Définit l'intervalle des ordonnées de la figure
<code>plt.axis([0, 6, 0, 20])</code>	Définit l'intervalle des abscisses (0 à 6) puis des ordonnées (0 à 20)
<code>plt.axis('equal')</code> <code>plt.axis('scaled')</code>	Repère orthonormé – A mettre avant xlim et ylim – Equal : redéfinit les valeurs xlim ylim – scaled : redéfinit la fenêtre
<code>plt.grid(True)</code>	Affiche la grille
<code>plt.title('Texte')</code>	Définit un titre au graphique
<code>plt.xlabel('Texte')</code>	Définit le nom des données en abscisses
<code>plt.ylabel('Texte')</code>	Définit le nom des données en ordonnée
<code>plt.show()</code>	Affiche le graphique
<code>plt.close()</code>	Ferme la dernière figure créée/appelée/ouverte
<code>plt.close('all')</code>	Ferme toutes les figures
<code>plt.clf()</code>	Vide la dernière figure créée/appelée/ouverte sans la fermer (lors d'une simulation, il est beaucoup plus rapide de vider que de fermer/ouvrir)
<code>plt.pause(2)</code>	Permet d'attendre par exemple 2 secondes avant de continuer
<code>plt.savefig('Nom')</code>	Enregistrement dans le répertoire courant des figures préaffichées au format standard (png) avec le nom précisé – Pensez à enregistrer votre code dans un dossier avant de l'exécuter

Il est possible de faire apparaître plusieurs graphiques dans la même figure, par exemple :

```

from math import cos,sin
n = 100
X = [i/10 for i in range(n)]
Y1 = [sin(X[i]) for i in range(n)]
Y2 = [cos(X[i]) for i in range(n)]
plt.subplot(211)
plt.plot(X,Y1)
plt.subplot(224)
plt.plot(X,Y2)
plt.show()
211 veut dire : séparation en 2 lignes et 1 colonnes, choix de la première cellule parmi 2
224 veut dire : séparation en 2 lignes et 2 colonnes, choix de la dernière cellule parmi 4

```

Google vous donnera le reste...

## 1.V. Gestion de plusieurs figures

Il n'existe pas une seule façon de faire. Personnellement, j'aime bien avoir la main sur les figures que je crée afin de pouvoir :

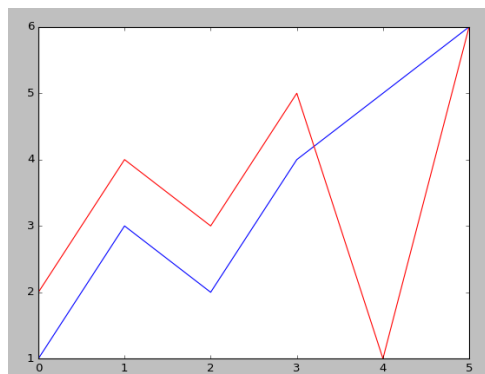
- En ouvrir autant que je le souhaite en parallèle
- Mettre à jour l'une d'elles
- Fermer l'une d'elles

Si on écrit :

```
X = [0,1,2,3,4,5]
Y = [1,3,2,4,5,6]
plt.plot(X,Y,'b')
plt.show()
```

```
X = [0,1,2,3,4,5]
Y = [2,4,3,5,1,6]
plt.plot(X,Y,'r')
plt.show()
```

On obtient :



Les deux courbes sont tracées sur le même graphique...

Si vous devez fermer la fenêtre pour qu'une seconde s'ouvre, faites ce qui suit, sinon allez à la page suivante :

```
import matplotlib
print(matplotlib.get_backend()) # tkagg
Selon que vous utilisez Miniconda/Anaconda (conda), ou Python (pip)
pip install PyQt5
conda install PyQt5
matplotlib.use("Qt5Agg")
print(matplotlib.get_backend()) # Qt5Agg
Puis redémarrer Pyzo
Refaites tourner votre code, le problème est résolu !
```

C'est pourquoi, à chaque fois que je fais une figure, j'utilise en premier lieu la commande :

```
plt.figure(i)
```

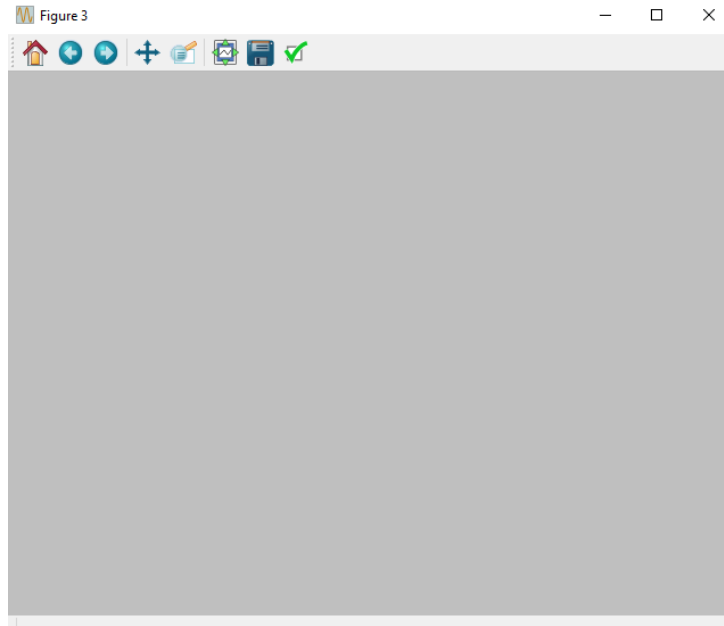
Où *i* est un nombre entier qui définit un numéro de figure.

Remarque : on peut mettre un nom (chaîne de caractères) : `plt.figure("Nom")`

Ainsi, le code suivant ouvre la figure 3 :

```
plt.figure(3)
plt.show()
```

On obtient alors :



Il est alors très simple de fermer une figure parmi toutes les figures ouvertes à l'aide de la commande :

```
plt.figure(3)
plt.close()
```

De même, il est possible d'afficher une seconde courbe à la figure 3 en rappelant la figure en question :

```
plt.figure(3)
plt.plot(...)
plt.show()
```

Ou encore de vider la figure 2 en écrivant :

```
plt.figure(2)
plt.clf()
```



## 1.VI. Objet figure

Il est possible de créer un objet associé à une figure, pour cela il suffit d'écrire par exemple :

```
Fig_1 = plt.figure(1)
```

Il n'est plus alors obligatoire de rappeler la figure avec « `plt.figure(1)` » pour la fermer par exemple, il suffit d'écrire « `plt.close(Fig_1)` ».

Cela peut avoir d'autres intérêts que nous ne détaillerons pas ici.

## 1.VII. Le tout en une fonction

Idéalement, si vous souhaitez afficher une courbe, il est intéressant de créer une fonction qui prend en argument les deux listes de la courbe en question, et le numéro de la figure associée.

Ainsi, vous écrirez :

```
# Import librairie
from matplotlib import pyplot as plt

# Fermeture d'éventuelles fenêtres ouvertes
plt.close('all')

# Définition de la fonction
def f_courbe(X,Y,N_Fig,Legende):
    plt.figure(N_Fig)
    # Options à définir
    plt.plot(X,Y,label=Legende)
    plt.xlabel('Abscisses')
    plt.ylabel('Ordonnées')
    plt.legend()
    plt.show()

# Tracé
X = [1,2,3]
Y = [0,1,2]
f_courbe(X,Y,1, 'Legende')
```

Remarque : plusieurs appels de `f_courbe` permettent de tracer plusieurs courbes sur la même figure. Cela fonctionne très bien depuis la zone de code avec F5, beaucoup moins bien depuis la console...

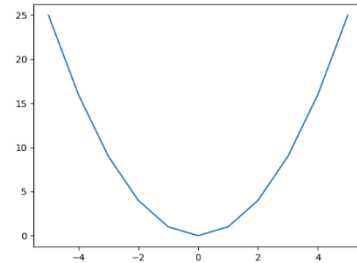
## 1.VIII. Tracer une fonction $f(x)$

Pour tracer une fonction, il est nécessaire de passer par 2 listes, l'une pour les abscisses, l'autre pour les ordonnées. Exemple :

```
import matplotlib.pyplot as plt
plt.close('all')
Lx = [-5,-4,-3,-2,-1,0,1,2,3,4,5]
Lfx = [f(x) for x in Lx]
```

```
def Affiche(X,Y):
    plt.plot(X,Y)
    plt.show()
```

```
Affiche(Lx,Lfx)
```



## 1.IX. Réalisation d'un histogramme

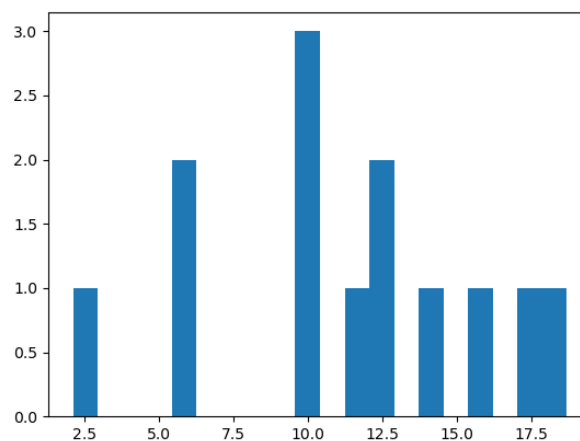
Soit le code suivant :

```
from matplotlib import pyplot as plt
plt.close('all')
```

```
def Histo(L,fig):
    plt.figure(fig)
    plt.hist(L,range=(0,20),bins=20)
    plt.show()
```

```
L = [12.2,10.1,15.4,14.3,2.1,18.7,17.8,5.5,12.7,11.5,10.1,6.1,10.3]
Histo(L,1)
```

On obtient l'histogramme suivant :



On voit ainsi le nombre de notes dans  $[0,1[$ ,  $[1,2[$  etc. Le paramètre « bins » précise le nombre d'intervalles en abscisses et le range donne la plage à découper en « bins » intervalles.

## 1.X. Mise à jour d'une courbe dans le cadre d'une simulation

On peut vouloir mettre à jour une ou plusieurs figures dans le cadre d'une simulation avec affichage des positions successives d'un objet, des températures à tous les pas de temps...

### 1.X.1 Animation

Soit le code suivant permettant d'afficher 9 cercles de rayons allant de 1 à 10 dans une fenêtre orthonormée qui s'adapte aux dimensions souhaitées -10 10 en abscisses et ordonnées ET en utilisant « scaled » (important dans ce type de simulations, sinon la fenêtre s'adapte et on voit toujours le cercle à la même taille...):

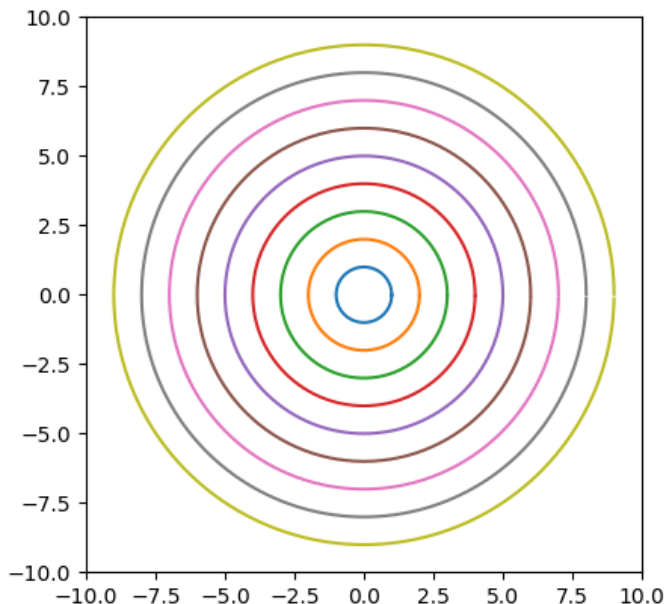
```
import matplotlib.pyplot as plt
plt.close('all')

from math import pi,cos,sin

def Cercle(r):
    X = []
    Y = []
    for i in range(360):
        teta = i*pi/180
        x = r*cos(teta)
        y = r*sin(teta)
        X.append(x)
        Y.append(y)
    return X,Y

def Affiche(fig,X,Y,Chemin):
    plt.figure(fig)
    plt.plot(X,Y)
    plt.axis('scaled')
    plt.xlim([-10,10])
    plt.ylim([-10,10])
    plt.show()

for r in range(1,10):
    X,Y = Cercle(r)
    Chemin = str(r)
    Affiche(1,X,Y,Chemin)
```



A son exécution, on obtient une figure contenant tous les cercles.

On souhaiterait voir « en temps réel » s'afficher un cercle à la fois dans la figure, grandissant avec le temps. Il faut donc effacer celle-ci à chaque nouvel appel.

Pour cela, intégrons un `plt.clf()`. On le place volontairement au début de la fonction de manière à avoir le dernier cercle tracé quand le code s'arrête.

```
def Affiche(fig,X,Y,Chemin):  
    plt.figure(fig)  
    plt.clf()  
    plt.plot(X,Y)  
    plt.axis('scaled')  
    plt.xlim([-10,10])  
    plt.ylim([-10,10])  
    plt.show()
```

A l'exécution du code ci-dessus, on ne voit rien d'autre que la dernière courbe. En effet, l'ordinateur a très rapidement affiché les 9 courbes et n'a même pas eu le temps de nous les montrer. Nous devons donc finalement ajouter un dernier élément afin d'avoir le temps de voir les courbes, c'est la commande `plt.pause(1)` qui permet d'attendre 1 secondes avant d'afficher la prochaine courbe. Voilà le code final :

```
def Affiche(fig,X,Y,Chemin):  
    plt.figure(fig)  
    plt.clf()  
    plt.plot(X,Y)  
    plt.axis('scaled')  
    plt.xlim([-10,10])  
    plt.ylim([-10,10])  
    plt.show()  
    plt.pause(1)
```

Vous voyez alors s'animer les cercles 😊

Remarque importante : A la place de `clf`, on aurait pu utiliser « `plt.close()` » qui ferme et ouvre la fenêtre à chaque tracé. Mais attention, cela nuit fortement à la visualisation d'une part, mais en plus si cela va trop vite, la fenêtre n'a même pas le temps d'apparaître qu'elle se referme.

## 1.X.2 Stockage d'images

Enfin, on peut même stocker dans le répertoire courant (votre code doit être enregistré dans un répertoire connu) les images successives, en spécifiant le chemin, par exemple avec le rayon :

```
def Affiche(fig,X,Y,Chemin):  
    plt.figure(fig)  
    plt.clf()  
    plt.plot(X,Y)  
    plt.axis('scaled')  
    plt.xlim([-10,10])  
    plt.ylim([-10,10])  
    plt.show()  
    plt.pause(0.1)  
    plt.savefig(Chemin)
```

Oui, vous aviez peut-être remarqué dans le premier code, la présence de « Chemin » qui restait inutilisé...

Avec cette sauvegarde, il n'est plus nécessaire de garder le « pause », ce qui peut gagner du temps dans une simulation où l'on ne souhaite pas afficher les résultats en même temps que le calcul.

ATTENTION : ne pas garder de show en plus de savefig, sinon vous verrez une image qui devient de plus en plus petite...

Si l'on souhaite enregistrer les images dans un sous dossier du type « images », après avoir créé ce dossier au bon endroit, on écrira :

```
Chemin = "images/" + str(r)
```

Remarques :

- On pourra aussi utiliser un chemin absolu
- On pourra changer l'extension de l'image qui de base, est en .png :  
`Chemin = "images/" + str(r) + ".jpg"`

Si le format n'est pas supporté, on aura une liste des extensions possibles :

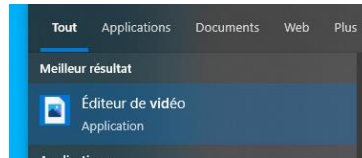
**ValueError: Format 'bmp' is not supported (supported formats: eps, jpeg, jpg, pdf, pgf, png, ps, raw, rgba, svg, svgz, tif, tiff)**

## 1.X.3 Réalisation d'une vidéo

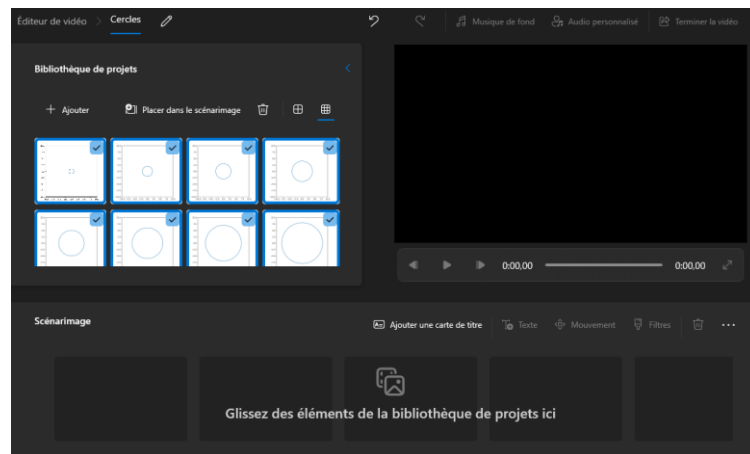
### 1.X.3.a Editeur de vidéo Windows

Jusqu'en 2023, il existait sous Windows 10 l'éditeur de vidéos. Si en le lançant, vous avez un message vous invitant à utiliser « Clipchamp », oubliez... On doit ajouter les images une à une dans la timeline.

Taper dans le menu démarrer : « vidéo » avec l'accent et ouvrir « Editeur de vidéo » :



Créer un nouveau projet, lui donner un nom, puis glisser les images ici :



Alors qu'elles sont toutes sélectionnées, cliquer sur « Placer dans le scénarimage ». Elles apparaissent en bas, vérifiez leur bon ordre et si besoin, réordonner.

Sélectionner une des images en bas, faire « Ctrl + A » pour les sélectionner toutes, puis cliquer sur « Durée » :



Choisir le temps de chaque image, par exemple 0,5s. Cliquer ensuite sur « Terminer » en haut à droite, et enregistrer la vidéo.

Voici ce que j'obtiens : [LIEN DROPBOX](#)

Remarque : pour modifier après coup la vitesse d'une vidéo, il suffit de la réimporter dans un nouveau projet, et de changer sa vitesse avec le même logiciel.

### 1.X.3.b ffmpeg sous Python

Installer le module ffmpeg avec pip :

<code>pip install ffmpeg</code>	<code>conda install ffmpeg</code>
---------------------------------	-----------------------------------

Remarques:

- Des lignes rouges à l'installation ne sont pas forcément problématiques pour la suite, attendez de voir :  
`Preparing transaction: done`  
`Verifying transaction: done`  
`Executing transaction: done`
- En cas d'erreur « 'ffmpeg' n'est pas reconnu en tant que commande interne ou externe, un programme exécutable ou un fichier de commandes. » à l'exécution des lignes plus bas, tenter conda). Si cette erreur persiste, consulter la suite !

Organisez vos images pour qu'elles possèdent un nom de type entier croissant : « i.png ».

Coller le code ci-dessous dans un fichier python enregistré dans le même répertoire que les images, en veillant à ce que l'option du menu « Run » ou « Exécuter » de Pyzo « Change directory when executing file » est cochée :

```
import os
os.system("ffmpeg -framerate 20 -i %d.png video.avi")
```

Exécuter alors ce code avec la touche F5.

La vidéo créée possèdera le nom « video.avi » au format « .avi » donc. Vous pourrez changer le nombre d'images par seconde en changeant le **framerate** qui vaut actuellement 20.

Attention : à priori, si la vidéo existe déjà, il faut d'abord la supprimer, sinon python semble attendre indéfiniment...

En cas d'erreur persistante « 'ffmpeg' n'est pas reconnu en tant que commande interne ou externe, un programme exécutable ou un fichier de commandes. », voici une solution qui a fonctionné chez moi (cf. ChatGPT) :

- Télécharger et dézipper <https://www.gyan.dev/ffmpeg/builds/ffmpeg-release-essentials.zip> dans un dossier de l'ordinateur – Identifier le chemin de bin : « C:\...\ffmpeg\ffmpeg-7.1.1-essentials\_build\bin »
- Dans le menu Windows – Taper « Variables » et cliquer sur « Modifier les variables d'environnement système » – Variables d'environnement – Dans Variables utilisateur, trouver Path – Modifier – Ajouter le chemin précédent, puis tout valider
- Dans l'invite de commande « cmd » : « ffmpeg -version » – Entrer – Vérifier qu'il trouve bien ffmpeg
- Relancer Pyzo (important)