Partie IV

Manipulation des données avec Numpy

- Le paquet NumPy (Numeric Python) fournit des routines de base pour manipuler de grands tableaux et matrices de données numériques.

- Vous pouvez importer ce package :

```
import numpy
Or
import numpy as
np
```

- Les tableaux sont similaires aux listes en Python, sauf que chaque élément d'un tableau doit être du même type.

- La fonction array() prend deux arguments : la liste à convertir dans le tableau et le type de chaque membre de la liste.

```
a = np.array([1, 4, 5, 8],
float)
```

Manipulation des données avec Numpy

- Les éléments de tableau sont accessibles comme suit :

```
a = np.array([1, 4, 5, 8],
float)
print(a[:2])
print(a[3])
a[0] = 5
print(a)
```

- Les tableaux peuvent être multidimensionnels. Les différents axes sont accédés à l'aide de virgules à l'intérieur de la notation entre parenthèses.

```
a = np.array([[1, 2, 3], [4, 5, 6]],
float)
print(a[0,0]) #vaut 1.0
print(a[1,:]) #vaut [4. 5. 6.]
print(a[:,2]) #vaut [3. 6.]
```

Manipulation des données avec Numpy

- Chaque tableau a les attributs ndim (le nombre de dimensions), shape (la taille de chaque dimension) et size (la taille totale du tableau) :

```
x1 = np.random.randint(10, size=6)
x2 = np.random.randint(10, size=(3, 4))
x3 = np.random.randint(10, size=(3, 4, 5))
print(x1)
print("x3 ndim: ", x3.ndim)
print("x3 shape:", x3.shape)
print("x3 size: ", x3.size)
```

- La propriété dtype vous indique le type des éléments du tableau
- La fonction len renvoie la longueur du premier axe.
- L'instruction in peut être utilisée pour tester si des valeurs sont présentes dans un tableau.

```
x1.dtype
print(len(x1))
2 in x1
```

Manipulation des données avec Numpy

- Les tableaux peuvent être reformés en utilisant des tuples qui spécifient de nouvelles dimensions. La fonction reshape() crée un nouveau tableau et ne modifie pas le tableau d'origine.

```
a = np.array(range(10), float)
print(a)
b = a.reshape((5, 2))
print(b)
```

- La fonction copy peut être utilisée pour créer une nouvelle copie distincte d'un tableau en mémoire.

```
a = np.array([1, 2, 3], float)
b = a
c = a.copy()
a[0] = 0
print(a) #vaut array([0., 2., 3.])
print(b) #vaut array([0., 2., 3.])
print(c) #vaut array([1., 2., 3.])
```

Manipulation des données avec Numpy

- Vous pouvez construire une liste à partir d'un tableau :

```
a = np.array([1, 2, 3], float)
print(type(a))
a = a.tolist() #vaut [1.0, 2.0, 3.0]
print(type(a))
list(a)#vaut [1.0, 2.0, 3.0]
```

- Vous pouvez utiliser la fonction flatten()

```
a = np.array([[1, 2, 3], [4, 5, 6]], float)
a = a.flatten() # vaut array([ 1., 2., 3., 4., 5., 6.])
print(a)
```

- Vous pouvez utiliser la fonction arange()

```
np.arange(5, dtype=float)# array([ 0., 1., 2., 3., 4.])
np.arange(1, 6, 2, dtype=int) #array([1, 3, 5])
```

Manipulation des données avec Numpy

- Vous pouvez utiliser les fonctions zéros et ones pour créer de nouveaux tableaux de dimensions spécifiées

```
np.ones((2,3), dtype=float) #array([[1.,1.,1.], [1.,1.,1.]])
np.zeros(7, dtype=int) #array([0, 0, 0, 0, 0, 0, 0])
```

- Les fonctions zeroslike et oneslike créent un nouveau tableau avec les mêmes dimensions et le même type que celui existant

```
a = np.array([[1, 2, 3], [4, 5, 6]], float)
np.zeros_like(a) #array([[ 0., 0., 0.], [ 0., 0., 0.]])
np.ones_like(a) #array([[ 1., 1., 1.], [ 1., 1., 1.]])
```

- Vous pouvez créer une matrice d'identité d'une taille donnée

```
np.identity(4, dtype=float)
```

Manipulation des données avec Numpy

- Vous pouvez appliquer plusieurs opérations élément par élément sur les tableaux (Il faut avoir la même taille des tableaux) :

```
a = np.array([1,2,3], float)
b = np.array([5,2,6], float)
print(a + b)
print(a * b)
print(b / a)
print(a % b)
print(b**a)
```

- Les tableaux dont le nombre de dimensions ne correspond pas seront diffusés par Python pour effectuer des opérations mathématiques(le plus petit tableau sera répété autant de fois que nécessaire pour effectuer l'opération)

```
a = np.array([[1, 2], [3, 4], [5, 6]], float)
b = np.array([-1, 3], float)
print(a + b)
```

Manipulation des données avec Numpy

- NumPy propose une vaste bibliothèque de fonctions mathématiques communes pouvant être appliquées aux tableaux : abs, sign, sqrt, log, log10, exp, sin, cos, tan, arcsin, arccos, arctan, sinh, cosh, tanh, arcsinh, arccosh, ...

```
a = np.array([1, 4, 9], float)
np.sqrt(a) #vaut array([ 1., 2., 3.])
```

- Il est possible de trouver les valeurs minimales et maximales des éléments

```
a = np.array([2, 1, 9], float)
a.min() #1.0
a.max() #9.0
```

- Les fonctions argmin et argmax renvoient les indices dans le tableau correspondant à des valeurs minimale et maximale

```
a = np.array([2, 1, 9], float)
a.argmin() #1
a.argmax() #2
```

Manipulation des données avec Numpy

- Il est possible d'itérer sur les tableaux d'une manière similaire à celle des listes

```
a = np.array([1, 4, 5], int)
for x in a:
print(x)
a = np.array([[1, 2], [3, 4], [5, 6]], float)
for x in a:
print(x)
a = np.array([[1, 2], [3, 4], [5, 6]], float)
for (x, y) in a:
print (x * y)
```

- Les tableaux peuvent être triés

```
a = np.array([6, 2, 5, -1, 0], float)
sorted(a) #[-1.0, 0.0, 2.0, 5.0, 6.0]
a.sort()
a #array([-1., 0., 2., 5., 6.])
```

Manipulation des données avec Numpy

- Pour les tableaux multidimensionnels, chacune des fonction décrites jusqu'à présent peut prendre un axe d'argument optionnel qui effectuera une opération uniquement sur l'axe spécifié

```
a = np.array([2, 1, 9], float)
a.min() #1.0
a.max() #9.0
a = np.array([[0, 2], [3, -1], [3, 5]], float)
a.mean(axis=0) #array([ 2., 2.])
a.mean(axis=1) #array([ 1., 1., 4.])
a.min(axis=1) #array([ 0., -1., 3.])
a.max(axis=0) #array([3., 5.])
```

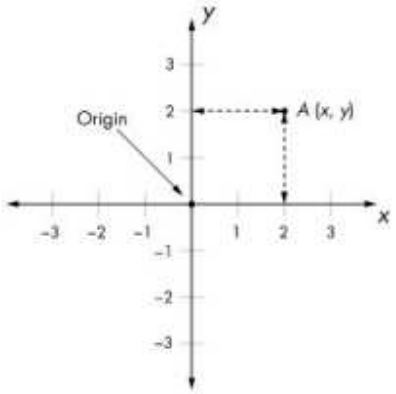
Manipulation des données avec Numpy

- Les tableaux entiers peuvent contenir des indices d'éléments à extraire d'un tableau.

```
a = np.array([2, 4, 6, 8], float)
b = np.array([0, 0, 1, 3, 2, 1], int)
a[b] # vaut array([ 2., 2., 4., 8., 6., 4.])
a = np.array([[1, 4], [9, 16]], float)
b = np.array([0, 0, 1, 1, 0], int)
c = np.array([0, 1, 1, 1, 1], int)
a[b,c]# vaut array([ 1., 4., 16., 16., 4.])
```

Plan de coordonnées cartésiennes

- Considérons une droite numérique, les entiers compris entre –3 et 3 sont marqués sur la ligne
- Tous les nombres du côté droit de 0 sont positifs et ceux du côté gauche sont négatifs.
- Considérons maintenant deux lignes numériques.
- Les lignes numériques se coupent à angle droit et se croisent au point 0 de chaque ligne.
- Cela forme un plan de coordonnées cartésiennes, ou un plan x-y, avec la ligne numérique horizontale appelée axe des x et la ligne verticale appelée axe des y.
- Vous décrivez le point A dans la figure avec deux nombres, x et y, généralement écrits sous la forme (x, y) et appelés coordonnées du point.



Matplotlib

- Lorsque vous réalisez des graphiques avec Python, vous allez travailler avec des listes et des tuples ou bien les tableaux de numpy.

```
l = [1, 2, 3]
for item in l:
    print(item)
```

- Vous pouvez utiliser la fonction enumerate () pour parcourir tous les éléments d'une liste et renvoyer l'index d'un élément ainsi que l'élément lui-même.

```
l = [1, 2, 3]
for index, item in
enumerate(l):
    print(index, item)
```

- Vous pouvez commencer avec un graphique simple comportant seulement trois points : (1,
- 2), (3, 3) ,(4, 5) et (6,1)
- Importation :

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

Matplotlib

- L'instruction plot() permet de tracer des courbes qui relient des points dont les abscisses et
- ordonnées sont fournies dans des tableaux.

```
import numpy as np
import matplotlib.pyplot
as plt
x = np.array([1, 3, 4, 6])
y = np.array([2, 3, 5, 1])
plt.plot(x, y)
```

- La fonction plt.show () recherche tous les objets de figure actuellement actifs et ouvre une ou plusieurs fenêtres interactives qui affichent ces figures :

```
import matplotlib.pyplot as
plt
import numpy as np
x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
plt.show()
```

Matplotlib

- La fonction subplot permet de regrouper plusieurs graphes, modélisé par un objet Figure

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 100)
# create the first of two panels and set current axis
plt.subplot(2, 1, 1) # (rows, columns, panel number)
plt.plot(x, np.sin(x))
# create the second panel and set current axis
plt.subplot(2, 1, 2)
plt.plot(x, np.cos(x))
plt.show()
```

Matplotlib

- Vous pouvez peut utiliser un argument de mot clé supplémentaire lors de l'appel de la
- fonction plot () pour marquer les points :

```
plt.plot(x_numbers, y_numbers, marker='o')
```

- Vous pouvez choisir parmi plusieurs options de marqueur, y compris "o", "*", "x" et "+".
- Vous pouvez créer un graphique qui ne marque que les points que vous avez spécifié, sans
- aucune ligne les reliant, en supprimant marker
- Exemple : Les valeurs de température moyennes mesurées à une ville au cours des années 2005 à 2017 sont les suivantes : 25.9, 30.3, 30.4, 27.4, 35.5, 36.8, 28.8, 31.0, 32.0, 35.3, 34.0,
- 36.7 et 36.4 degrés

```
nyc_temp = [25.9, 30.3, 30.4, 27.4, 35.5, 36.8, 28.8, 31.0,32.0, 35.3, 34.0, 36.7,
36.4 ]
years = range(2005, 2018)
plt.plot(years, nyc_temp, marker='o')
```

- Créer trois listes pour stocker la température. Chaque liste sera composée de 12 chiffres
- correspondant à la température moyenne de janvier à décembre de chaque année

Matplotlib

```
nyc_temp_2005 = [30.3, 21.3, 22.2, 27.0, 25.5, 20.3, 19.3, 32.7, 29.0, 26.0, 25.3, 25.1]
nyc_temp_2010 = [40.3, 41.3, 38.2, 39.0, 20.9, 21.3, 20.3,35.6, 38.0, 37.0, 32.1, 31.0]
nyc_temp_2017 = [36.3, 32.3, 31.2, 40.0, 22.4, 20.2, 19.3, 32.9, 39.0, 36.2, 31.2, 30.0]
```

- Vous pouvez tracer les trois ensembles de données sur trois graphiques différents
- Mais pour pouvoir les comparer, vous pouvez les tracer sur un même graphique
- Vous pouvez aussi appeler la fonction plot trois fois

```
months = range(1, 13)
plt.plot(months, nyc_temp_2005, months, nyc_temp_2010, months,
nyc_temp_2017)

plt.plot(months, nyc_temp_2005)
plt.plot(months, nyc_temp_2010)
plt.plot(months, nyc_temp_2017)
```

Matplotlib

- La fonction legend () permet d'ajouter une légende au graphique

```
months = range(1, 13)
plt.plot (months, nyc_temp_2005, months, nyc_temp_2010, months,
nyc_temp_2017)
plt.legend([2005, 2010, 2017])
```

- Vous pouvez spécifier un second argument à la fonction qui spécifiera la position de la légende. Par défaut, il est toujours placé en haut à droite du graphique.
- Vous pouvez spécifier une position particulière, telles que 'lower center', 'center left', and
- 'upper left' avec upper, lower, left et right

```
months = range(1, 13)
plt.plot (months, nyc_temp_2005, months, nyc_temp_2010, months,
nyc_temp_2017)
plt.legend([2005, 2010, 2017],loc='lower center')
```

Matplotlib

- Pour ajouter un titre au graphique, vous allez utiliser la fonction title ()
- Pour ajouter des étiquettes aux axes x et y du graphique, vous allez utiliser les fonctions
- xlabel () et ylabel ()

```
months = range(1, 13)
plt.plot (months, nyc_temp_2005, months, nyc_temp_2010, months,
nyc_temp_2017)
plt.legend([2005, 2010, 2017],loc='lower right')
plt.title(' Température moyennes')
plt.xlabel('les différents Mois')
plt.ylabel('Les différentes Valeurs')
```

Pylab

- PyLab permet d'utiliser de manière aisée les bibliothèques NumPy et matplotlib pour de la
- programmation scientifique avec Python.
- Utilise la fonction arange() de numpy : la difference entre range() et arange() est qu'elle
- retourne un tableau à la différence de range(). Tester

```
from pylab import *
m = arange(3,15,2)
print(m)
```

- Utilise la fonction linspace() de numpy : linspace() permet d'obtenir un tableau 1D allant d'une valeur de départ à une valeur de fin avec un nombre donné d'éléments. Tester :

```
from pylab import
*
linspace(3,9,10)
```

Pylab

- PyLab permet d'utiliser de manière aisée les bibliothèques NumPy et matplotlib pour de la
- programmation scientifique avec Python.
- Utilise la fonction arange() de numpy : la difference entre range() et arange() est qu'elle
- retourne un tableau à la différence de range(). Tester

```
from pylab import arange
m = arange(3,15,2)
print(m)
```

- Utilise la fonction linspace() de numpy : linspace() permet d'obtenir un tableau 1D allant d'une valeur de départ à une valeur de fin avec un nombre donné d'éléments. Tester :

```
from pylab import linespace
linspace(3,9,10)
```

Pylab

```
from pylab import plot, title, xlabel, ylabel, legend, show

nyc_temp_2005 = [30.3, 21.3, 22.2, 27.0, 25.5, 20.3, 19.3, 32.7, 29.0, 26.0, 25.3, 25.1]
nyc_temp_2010 = [40.3, 41.3, 38.2, 39.0, 20.9, 21.3, 20.3,35.6, 38.0, 37.0, 32.1, 31.0]
nyc_temp_2017 = [36.3, 32.3, 31.2, 40.0, 22.4, 20.2, 19.3, 32.9, 39.0, 36.2, 31.2, 30.0]
months = range(1, 13)
plot(months, nyc_temp_2005, months, nyc_temp_2010,
months, nyc_temp_2017)
legend([2005, 2010, 2017],loc='lower center')
title('Average monthly temperature')
xlabel('Month')
ylabel('Temperature')
show()
```

Pylab

- Vous pouvez ajuster la plage des axes en utilisant la fonction axis ()

```
from pylab import axis, plot, show,savefig
nyc_temp = [25.9, 30.3, 30.4, 27.4, 35.5, 36.8, 28.8,
31.0, 32.0, 35.3, 34.0, 36.7, 36.4 ]
plot(nyc_temp, marker='o')
axis()
axis(ymin=0)
axis ([0, 10, 0, 50])
show()
```

En utilisant la bibliothèque Matplotlib (ou PyLab), les fonctions grid et scatter sont utilisées pour ajouter une grille à un graphique et pour créer un nuage de points (scatter plot), respectivement

Vous pouvez enregistrer votre graphe avec savefig()

```
savefig("mon_graphique.png")
```

Matplotlib et csv

```
import matplotlib.pyplot as plt
import csv
import numpy as np

def read_csv(filename):
    with open(filename) as f:
        reader = csv.reader(f)
        next(reader)
        summer = []
        highest_correlated = []
        for row in reader:
            summer.append(float(row[1]))
            highest_correlated.append(float(row[2]))
    return summer, highest_correlated

def correlation(x, y):
    # Calcul de la corrélation entre deux ensembles de données
    corr_matrix = np.corrcoef(x, y)
    corr = corr_matrix[0, 1]
    return corr

def scatter_plot(x, y):
    plt.scatter(x, y)
    plt.xlabel('Summer Data')
    plt.ylabel('Highest Correlated Data')
    plt.title('Scatter Plot')
    plt.grid(True)

summer, highest_correlated = read_csv('correlate-summer.csv')
corr = correlation(summer, highest_correlated)
print('Highest correlation: {0}'.format(corr))
scatter_plot(summer, highest_correlated)
```

Tkinter

- Tkinter est une bibliothèque graphique standard de l'interface utilisateur (UI) pour le langage de programmation Python.
- Elle permet de créer des interfaces graphiques conviviales et interactives pour les applications Python.
- Tkinter est basé sur le toolkit graphique Tk, qui est un ensemble d'outils pour créer des interfaces graphiques dans divers langages de programmation.
- Avec Tkinter, on peut créer des fenêtres, des boutons, des boîtes de dialogue, des champs de texte, des cases à cocher, des boutons radio et d'autres éléments d'interface utilisateur pour leurs applications.
- Elle offre également la possibilité de créer des menus, des barres d'outils et des graphiques simples.

Tkinter

- Premier exemple de création d'une fenêtre:

```
import tkinter
window = tkinter.Tk()
window.title ('Hello Python')
window.geometry('250x250'
)
window.mainloop()
```

Pour créer un logiciel graphique on doit ajouter dans une fenêtre des éléments graphiques que l'on nomme widget.
Ce widget peut être tout aussi bien un bouton,une liste déroulante que du texte.

- **Les boutons :**

```
def button_click():
    print("Bouton cliqué !")
# Création d'un bouton
button = tkinter.Button(window, text="Cliquez-moi",
command=button_click)
button.pack()
```

Tkinter

- Utilisation de side

```
import tkinter as tk

fenetre = tk.Tk()
fenetre.title('Exemple de Side')

# Création d'un cadre
frame = tk.Frame(fenetre)
frame.pack()

# Création de boutons et utilisation du paramètre 'side'
button1 = tk.Button(frame, text='Bouton 1')
button1.pack(side=tk.LEFT)

button2 = tk.Button(frame, text='Bouton 2')
button2.pack(side=tk.LEFT)

fenetre.mainloop()
```

Tkinter

Les labels :

on ajoute un label à votre fenêtre en utilisant tkinter.Label. Le texte du label est défini comme "Bienvenue dans Tkinter". Lorsque le bouton est cliqué,

```
import tkinter

def button_click():
    label.config(text="Bouton cliqué !")
window = tkinter.Tk()
window.title('Hello Python')
window.geometry('250x250')
# Création d'un label
label = tkinter.Label(window, text="Bienvenue dans Tkinter")
label.pack()
# Création d'un bouton
button = tkinter.Button(window, text="Cliquez-moi", command=button_click)
button.pack()
window.mainloop()
```

Tkinter

- Création d'une zone de saisie : **Entry**

```
import tkinter

def button_click():
    label.config(text="Bonjour, " + entry.get())

window = tkinter.Tk()
window.title('Hello Python')
window.geometry('250x250')

# Création d'un label
label = tkinter.Label(window, text="Bienvenue dans Tkinter")
label.pack()

# Création d'une zone de saisie (entry)
entry = tkinter.Entry(window)
entry.pack()

# Création d'un bouton
button = tkinter.Button(window, text="Cliquez-moi", command=button_click)
button.pack()

window.mainloop()
```

Tkinter

- Création d'une case à cocher : **Checkbutton**

```
import tkinter
def toggle_checkbox():
    if var.get():
        label.config(text="Case cochée")
    else:
        label.config(text="Case décochée")
window = tkinter.Tk()
window.title('Hello Python')
window.geometry('250x250')
# Création d'un label
label = tkinter.Label(window, text="Cochez la case")
label.pack()
# Création d'une variable de contrôle pour la case à cocher
var = tkinter.BooleanVar()
checkbox = tkinter.Checkbutton(window, text="Cocher", variable=var,
command=toggle_checkbox)
checkbox.pack()

# Création d'un bouton
button = tkinter.Button(window, text="Cliquez-moi", command=toggle_checkbox)
button.pack()
window.mainloop()
```

Tkinter

- Création d'un bouton radio : **Radiobutton**

```
import tkinter

def update_label():
    selected_option = var.get()
    if selected_option == 1:
        label.config(text="Option 1 sélectionnée")
    elif selected_option == 2:
        label.config(text="Option 2 sélectionnée")
    elif selected_option == 3:
        label.config(text="Option 3 sélectionnée")

window = tkinter.Tk()
window.title('Hello Python')
window.geometry('250x250')

# Création d'un label
label = tkinter.Label(window, text="Sélectionnez une option")
label.pack()

# Variable de contrôle pour les boutons radio
var = tkinter.IntVar()

# Création des boutons radio
radio_button1 = tkinter.Radiobutton(window, text="Option 1", variable=var, value=1, command=update_label)
radio_button2 = tkinter.Radiobutton(window, text="Option 2", variable=var, value=2, command=update_label)
radio_button3 = tkinter.Radiobutton(window, text="Option 3", variable=var, value=3, command=update_label)

radio_button1.pack()
radio_button2.pack()
radio_button3.pack()
```

Tkinter

- Création d'une liste : **listbox**

```
import tkinter

def update_label():
    selected_item = listbox.get(tkinter.ACTIVE)
    label.config(text="Élément sélectionné : " + selected_item)

window = tkinter.Tk()
window.title('Hello Python')
window.geometry('250x250')

# Création d'un label
label = tkinter.Label(window, text="Sélectionnez un élément")
label.pack()

# Création d'une liste déroulante (Listbox)
listbox = tkinter.Listbox(window)
listbox.insert(1, "Élément 1")
listbox.insert(2, "Élément 2")
listbox.insert(3, "Élément 3")

listbox.pack()

# Bouton pour mettre à jour le label avec l'élément sélectionné
button = tkinter.Button(window, text="Afficher", command=update_label)
button.pack()
```

Tkinter

- Création de **spinbox et labelframe**

```
import tkinter

def update_label():
    selected_value = spinbox.get()
    label_result.config(text="Valeur sélectionnée : " + selected_value)

fenetre = tkinter.Tk()
fenetre.title('Exemple avec Spinbox et LabelFrame')
fenetre.geometry('400x300')
fenetre['bg'] = 'white'

# Création d'un LabelFrame
label_frame = tkinter.LabelFrame(fenetre, text="LabelFrame")
label_frame.pack(padx=10, pady=10)

# Création d'une Spinbox dans le LabelFrame
spinbox = tkinter.Spinbox(label_frame, from_=0, to=100, width=10)
spinbox.pack()

# Création d'un bouton pour mettre à jour le label
update_button = tkinter.Button(label_frame, text="Mettre à jour", command=update_label)
update_button.pack()

# Création d'un label pour afficher le résultat
label_result = tkinter.Label(label_frame, text="Valeur sélectionnée :")
label_result.pack()

fenetre.mainloop()
```

Tkinter

- Création d'un canevas : **canvas** : espace qui permet d'écrire ou déssiner
- Exemple1

```
import tkinter

def draw_rectangle():
    canvas.create_rectangle(50, 50, 150, 150, fill="blue")

window = tkinter.Tk()
window.title('Hello Python')
window.geometry('300x300')

# Création d'un canevas (Canvas)
canvas = tkinter.Canvas(window, width=200, height=200)
canvas.pack()

# Création d'un bouton pour dessiner un rectangle
button = tkinter.Button(window, text="Dessiner Rectangle", command=draw_rectangle)
button.pack()

window.mainloop()
```

Tkinter

- Création d'un canevas : **canvas** : espace qui permet d'écrire ou déssiner
- Exemple2

```
import tkinter

def write_text():
    text = entry.get()
    canvas.create_text(100, 100, text=text, font=("Helvetica", 12), fill="black")

window = tkinter.Tk()
window.title('Hello Python')
window.geometry('300x300')

# Création d'un canevas (Canvas)
canvas = tkinter.Canvas(window, width=200, height=200)
canvas.pack()

# Création d'une zone de saisie (entry)
entry = tkinter.Entry(window)
entry.pack()

# Création d'un bouton pour écrire le texte
button = tkinter.Button(window, text="Écrire Texte", command=write_text)
button.pack()
```

Tkinter

- Les **Frames**

```
import tkinter

fenetre = tkinter.Tk()
fenetre.title('Exemple avec Frames')
fenetre.geometry('400x300')
fenetre['bg'] = 'white'

# Frame 1
frame1 = tkinter.Frame(fenetre, borderwidth=2, relief=tkinter.GROOVE)
frame1.pack()

# Frame 2
frame2 = tkinter.Frame(fenetre, borderwidth=2, relief=tkinter.GROOVE)
frame2.pack(side=tkinter.LEFT, padx=10, pady=10)

# Frame 3 dans Frame 2
frame3 = tkinter.Frame(frame2, borderwidth=2, relief=tkinter.GROOVE)
frame3.pack(side=tkinter.RIGHT, padx=5, pady=5)

# Ajout de labels dans les cadres
label1 = tkinter.Label(frame1, text="Label dans Frame 1")
label1.pack()

label2 = tkinter.Label(frame2, text="Label dans Frame 2")
label2.pack()

label3 = tkinter.Label(frame3, text="Label dans Frame 3")
label3.pack()

fenetre.mainloop()
```

On peut utiliser: SUNKEN, RAISED, FLAT, GROOVE et RIDGE

Tkinter

- tkinter.PanedWindow pour créer une fenêtre divisée en panneaux redimensionnables :

```
import tkinter

fenetre = tkinter.Tk()
fenetre.title('Exemple avec PanedWindow')
fenetre.geometry('400x300')
fenetre['bg'] = 'white'

# Création d'une PanedWindow
paned_window = tkinter.PanedWindow(fenetre, orient=tkinter.HORIZONTAL)
paned_window.pack(expand=True, fill=tkinter.BOTH)

# Cadre 1 (panneau de gauche)
frame1 = tkinter.Frame(paned_window, borderwidth=2, relief=tkinter.GROOVE)
paned_window.add(frame1)

# Cadre 2 (panneau de droite)
frame2 = tkinter.Frame(paned_window, borderwidth=2, relief=tkinter.GROOVE)
paned_window.add(frame2)

# Ajout de labels dans les cadres
label1 = tkinter.Label(frame1, text="Panneau Gauche")
label1.pack()

label2 = tkinter.Label(frame2, text="Panneau Droit")
label2.pack()

fenetre.mainloop()
```

Tkinter

- tkinter.PanedWindow pour créer une fenêtre divisée en panneaux redimensionnables :

```
import tkinter

fenetre = tkinter.Tk()
fenetre.title('Exemple avec PanedWindow')
fenetre.geometry('400x300')
fenetre['bg'] = 'white'

# Création d'une PanedWindow
paned_window = tkinter.PanedWindow(fenetre, orient=tkinter.HORIZONTAL)
paned_window.pack(expand=True, fill=tkinter.BOTH)

# Cadre 1 (panneau de gauche)
frame1 = tkinter.Frame(paned_window, borderwidth=2, relief=tkinter.GROOVE)
paned_window.add(frame1)

# Cadre 2 (panneau de droite)
frame2 = tkinter.Frame(paned_window, borderwidth=2, relief=tkinter.GROOVE)
paned_window.add(frame2)

# Ajout de labels dans les cadres
label1 = tkinter.Label(frame1, text="Panneau Gauche")
label1.pack()

label2 = tkinter.Label(frame2, text="Panneau Droit")
label2.pack()

fenetre.mainloop()
```

Tkinter

- Les alertes

```
import tkinter
import tkinter.messagebox

def show_message():
    tkinter.messagebox.showinfo("Message", "Ceci est un message d'exemple.")

fenetre = tkinter.Tk()
fenetre.title('Exemple de MessageBox')
fenetre.geometry('400x300')
fenetre['bg'] = 'white'

# Création d'un bouton pour afficher la boîte de dialogue
button = tkinter.Button(fenetre, text="Afficher le message", command=show_message)
button.pack(pady=20)

fenetre.mainloop()
```

showinfo(title, message)
showwarning(title, message)
showerror(title, message)
askquestion(title, message)
askokcancel(title, message)
askyesno(title, message)
askretrycancel(title, message)
askquestion(title, message, **options)
askyesnocancel(title, message)

La représentation des données (Tkinter)

Tkinter

- Les barres de menu

```
import tkinter as tk
from tkinter import Menu

def open_file():
    print("Ouvrir un fichier")

def save_file():
    print("Enregistrer le fichier")

def exit_app():
    fenetre.quit()

fenetre = tk.Tk()
fenetre.title("Exemple de Barre de Menu")

# Barre de menu
menu_bar = Menu(fenetre)
fenetre.config(menu=menu_bar)

# Menu Fichier
file_menu = Menu(menu_bar, tearoff=0)
menu_bar.add_cascade(label="Fichier", menu=file_menu)
file_menu.add_command(label="Ouvrir", command=open_file)
file_menu.add_command(label="Enregistrer", command=save_file)
file_menu.add_separator()
file_menu.add_command(label="Quitter", command=exit_app)

# Menu Édition
edit_menu = Menu(menu_bar, tearoff=0)
menu_bar.add_cascade(label="Édition", menu=edit_menu)
edit_menu.add_command(label="Couper")
edit_menu.add_command(label="Copier")
edit_menu.add_command(label="Coller")
```

Tkinter

- Insertion des images

```
import tkinter as tk
from PIL import Image, ImageTk

fenetre = tk.Tk()
fenetre.title("Exemple d'intégration d'image (avec PIL)")

# Charger l'image avec PIL
image_pil = Image.open("rose.jpeg")  # Remplacez par le chemin de votre image au format JPEG
image_tk = ImageTk.PhotoImage(image_pil)

# Afficher l'image dans un Label
label_image = tk.Label(fenetre, image=image_tk)
label_image.pack()

fenetre.mainloop()
```

Tkinter

- Insertion des images

```
import tkinter as tk
from PIL import Image, ImageTk

def open_image():
    image_pil = Image.open("rose.jpeg")
    image_tk = ImageTk.PhotoImage(image_pil)
    label_image.config(image=image_tk)
    label_image.image = image_tk  # Conserver une référence pour éviter la suppression par le garbage collector

fenetre = tk.Tk()
fenetre.title("Exemple d'intégration d'image et d'éléments")

# Charger l'image avec PIL (remplacez par le chemin de votre image JPEG)
image_pil = Image.open("rose.jpeg")
image_tk = ImageTk.PhotoImage(image_pil)

# Afficher l'image dans un Label
label_image = tk.Label(fenetre, image=image_tk)
label_image.pack()

# Bouton pour ouvrir l'image
button_open = tk.Button(fenetre, text="Ouvrir Image", command=open_image)
button_open.pack()

# Label de texte
label_text = tk.Label(fenetre, text="Ceci est un exemple avec des boutons et des labels.")
label_text.pack()

fenetre.mainloop()
```

Tkinter

- tkinter.filedialog pour créer une boîte de dialogue de sélection de fichier dans Tkinter

```
import tkinter as tk
from tkinter import filedialog

def open_file_dialog():
    file_path = filedialog.askopenfilename(title="Sélectionner un fichier", filetypes=[("Fichiers texte", "*.txt"), ("Tous les fichiers", "*.*")])
    if file_path:
        selected_file_label.config(text="Fichier sélectionné : " + file_path)
    else:
        selected_file_label.config(text="Aucun fichier sélectionné")

# Création de la fenêtre principale
root = tk.Tk()
root.title("Sélection de fichier")

# Création d'un bouton pour ouvrir la boîte de dialogue de sélection de fichier
open_button = tk.Button(root, text="Ouvrir un fichier", command=open_file_dialog)
open_button.pack(pady=20)

# Étiquette pour afficher le chemin du fichier sélectionné
selected_file_label = tk.Label(root, text="Aucun fichier sélectionné")
selected_file_label.pack()

# Lancement de la boucle principale de l'application
root.mainloop()
```

Partie V