Module – 5

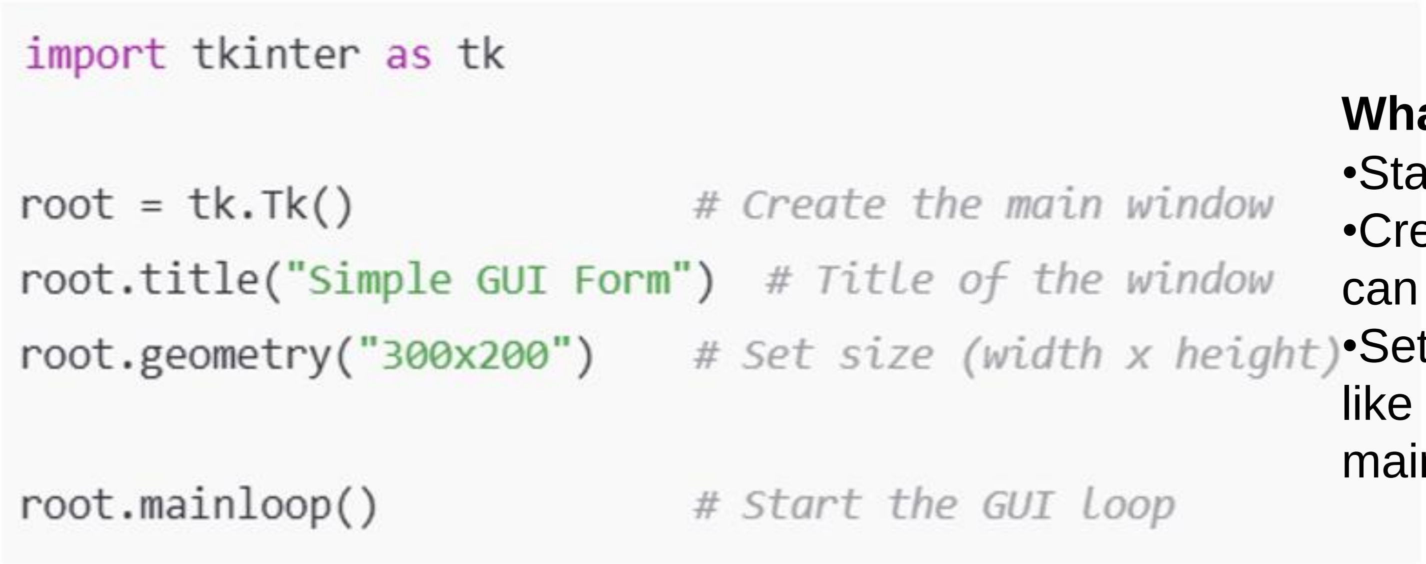
Chapter -1

Creating GUI Form and Adding Widgets

5.1 GUI (Graphical user Interface):

A GUI (Graphical User Interface) is a visual interface that allows users to interact with a program using graphical elements like **buttons, labels, text boxes, menus, and windows**, rather than typing commands in a text-based interface.

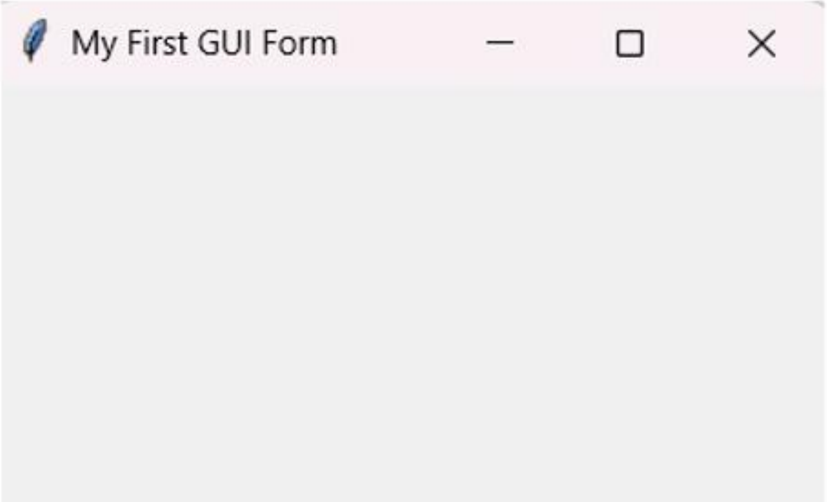
tkinter is Python’s **standard GUI library**, which provides tools to create windows, buttons, labels, text boxes, and other visual elements. In tkinter, we create this window using Tk(). This window remains open with the mainloop() function, which keeps the application running and responsive.



```python
import tkinter as tk

root = tk.Tk()                    # Create the main window
root.title("Simple GUI Form")  # Title of the window
root.geometry("300x200")      # Set size (width x height)

root.mainloop()               # Start the GUI loop
```

What Tk() Does

- Starts the **Tkinter interpreter**.
- Creates the **main application window** where you can place GUI widgets like buttons, labels, etc.
- Sets up the **event loop** that waits for user actions like clicks or key presses (you activate it using mainloop()).



Calling mainloop () will tell Tkinter to:

1. **Display the window** created with tk.Tk().
2. **Keep the application running** and responsive.
3. **Listen for events** (like button clicks or key presses).
4. Call the appropriate **callback functions** if any are defined.

5. 2. Adding Widgets (Labels, Entries, Buttons)

Widgets are the building blocks of GUI. Common widgets include:

Label: Displays static text.

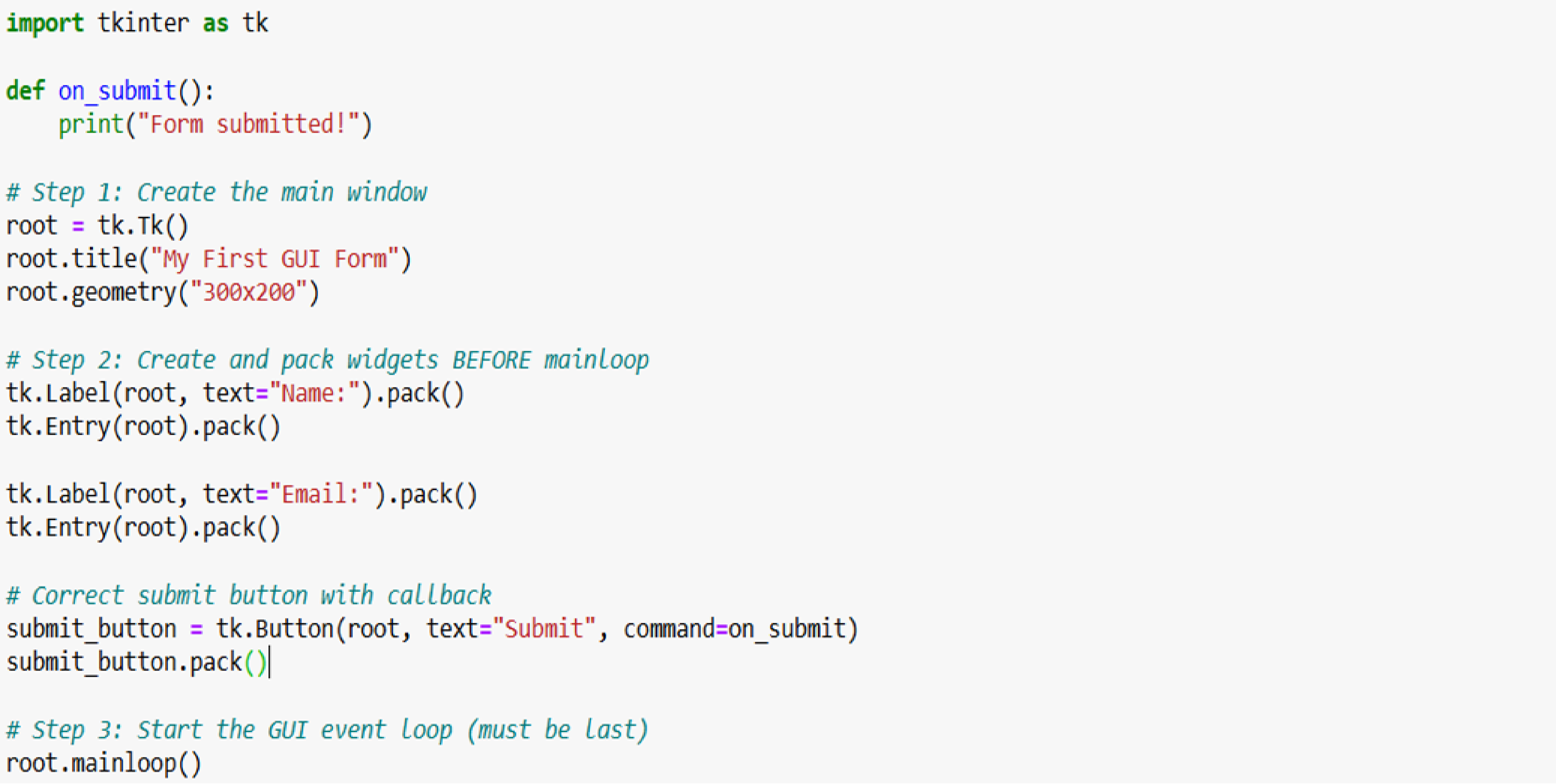
Entry: Input field for text.

Button: Triggers actions.

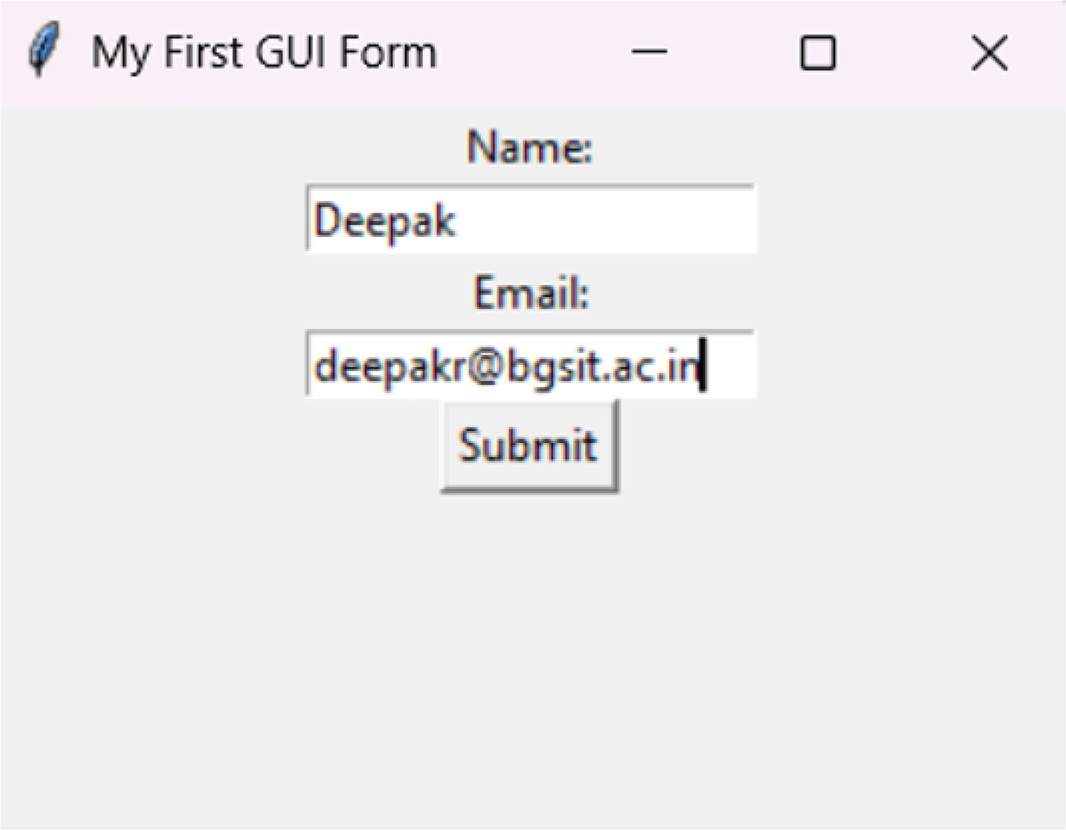
Widgets are placed using methods like .pack(), .grid(), or .place().

Layout Methods:

- .pack(): Stack widgets vertically or horizontally.
- .grid(): Place widgets in a table format (rows & columns).
- .place(): Absolute positioning (not recommended for beginners).

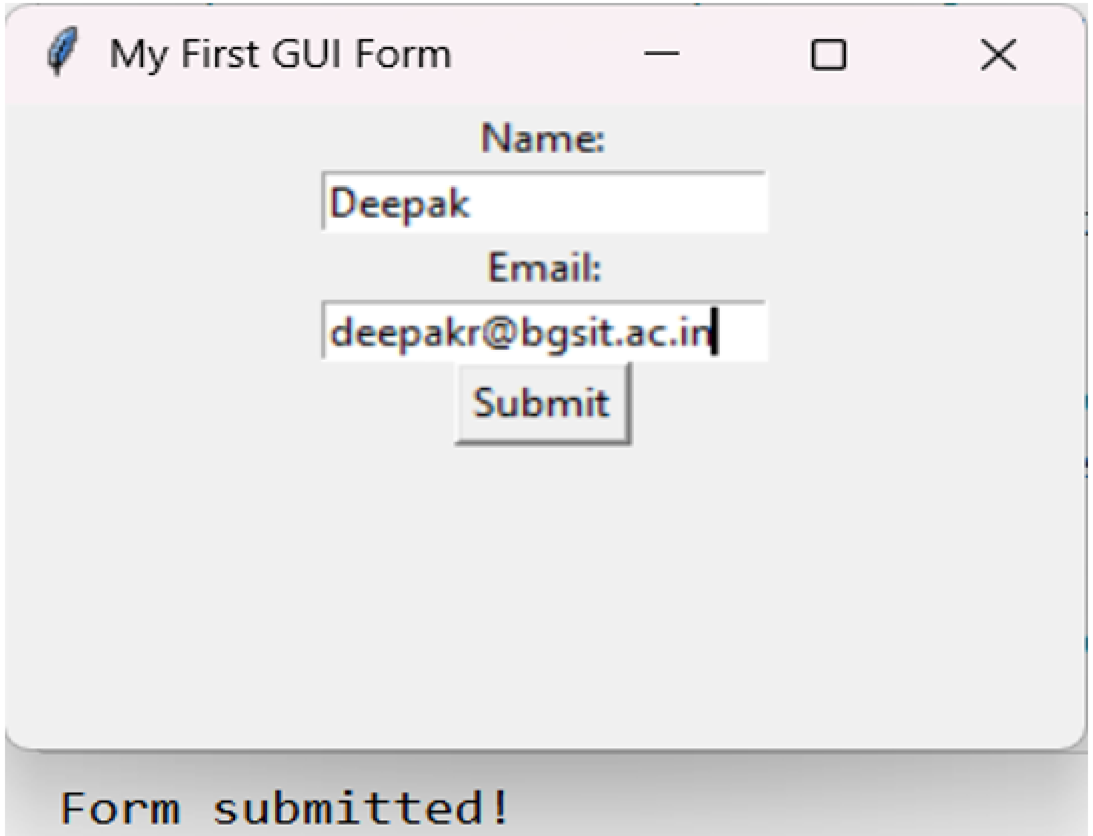


```python
import tkinter as tk

def on_submit():
    print("Form submitted!")

# Step 1: Create the main window
root = tk.Tk()
root.title("My First GUI Form")
root.geometry("300x200")

# Step 2: Create and pack widgets BEFORE mainloop
tk.Label(root, text="Name:").pack()
tk.Entry(root).pack()

tk.Label(root, text="Email:").pack()
tk.Entry(root).pack()

# Correct submit button with callback
submit_button = tk.Button(root, text="Submit", command=on_submit)
submit_button.pack()

# Step 3: Start the GUI event loop (must be last)
root.mainloop()
```



Name: Deepak

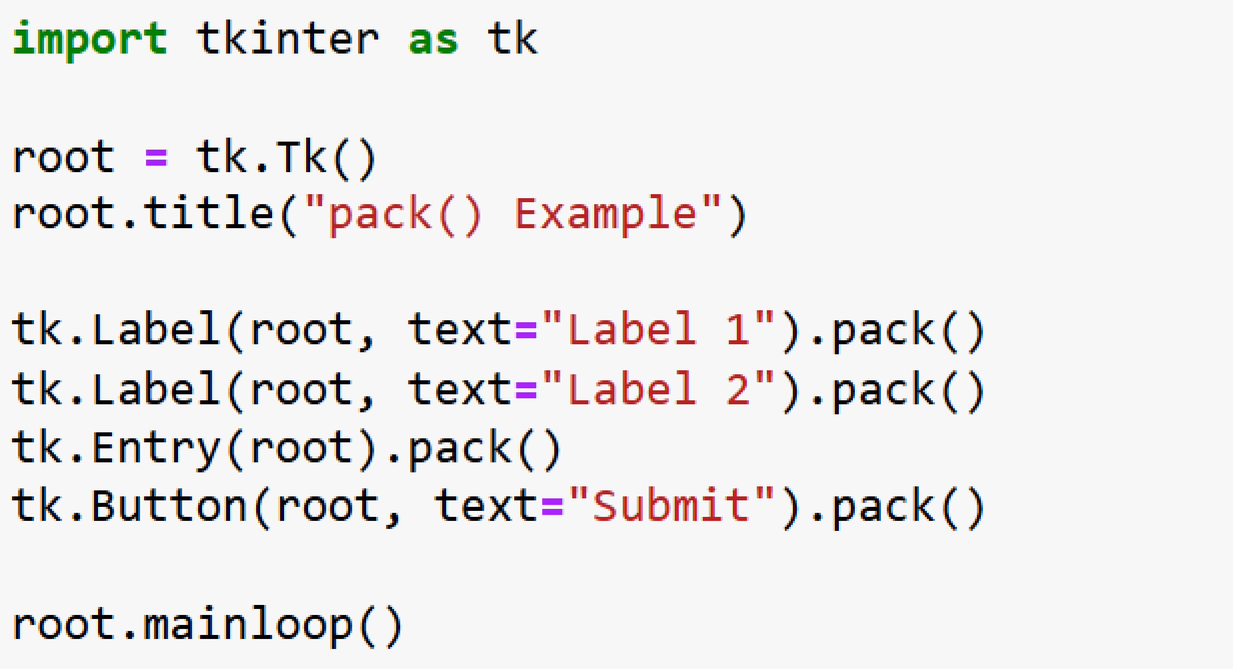
Email: deepakr@bgsit.ac.in



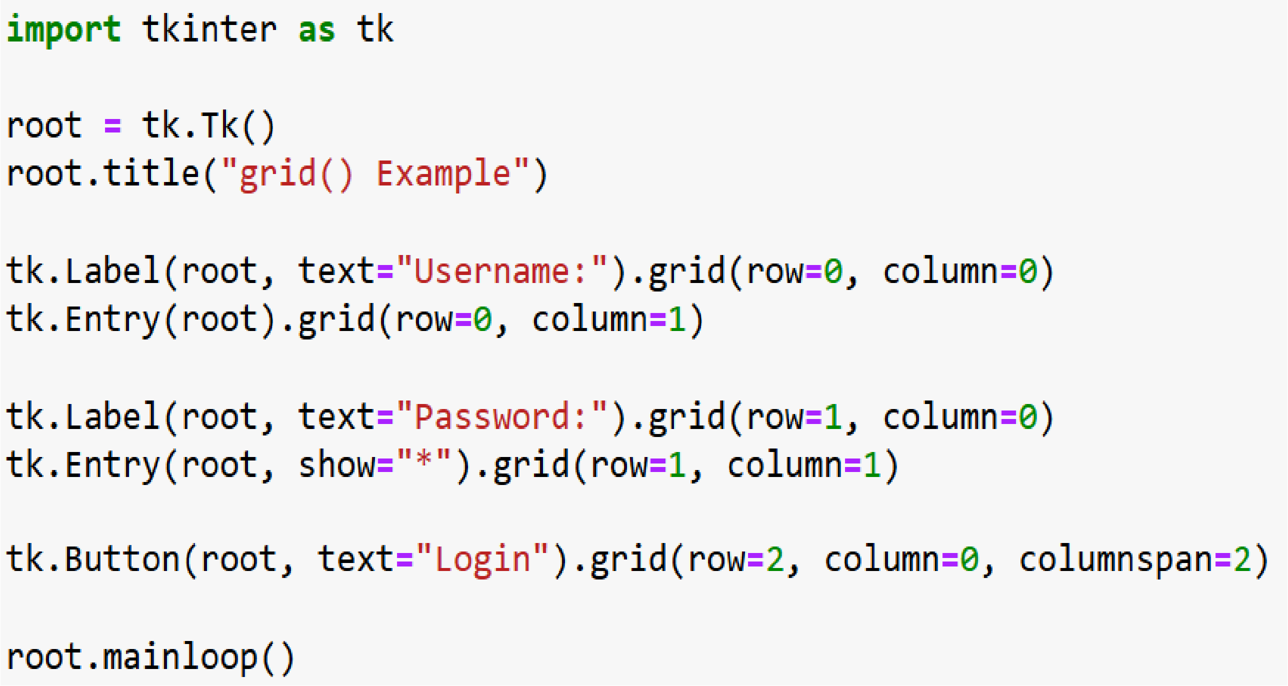
Name: Deepak

Email: deepakr@bgsit.ac.in

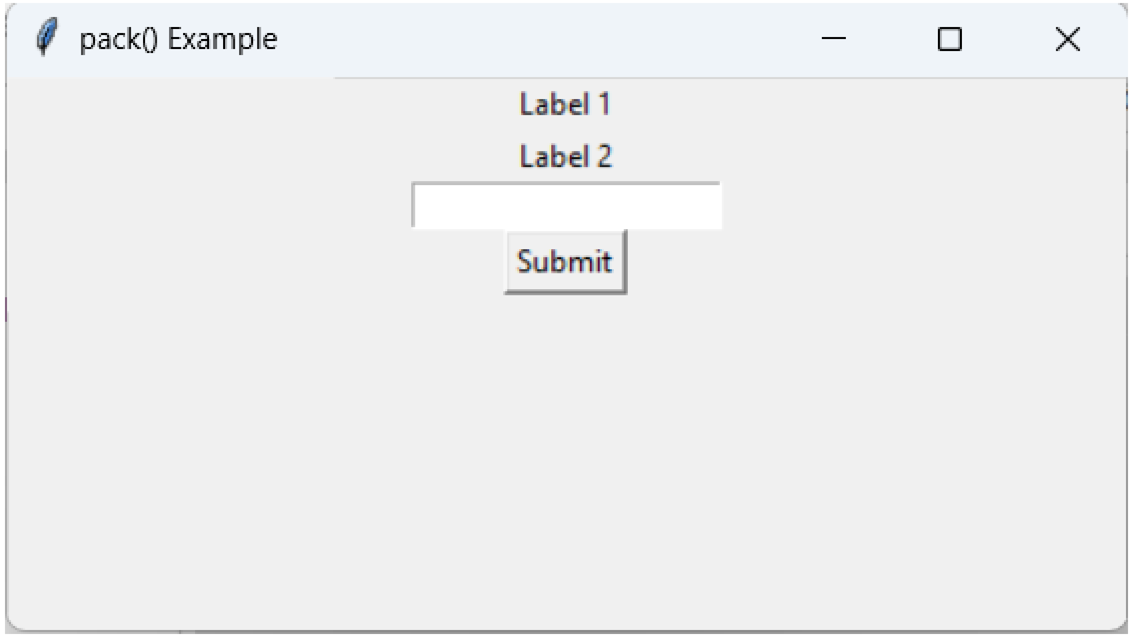
```
Form submitted!
```

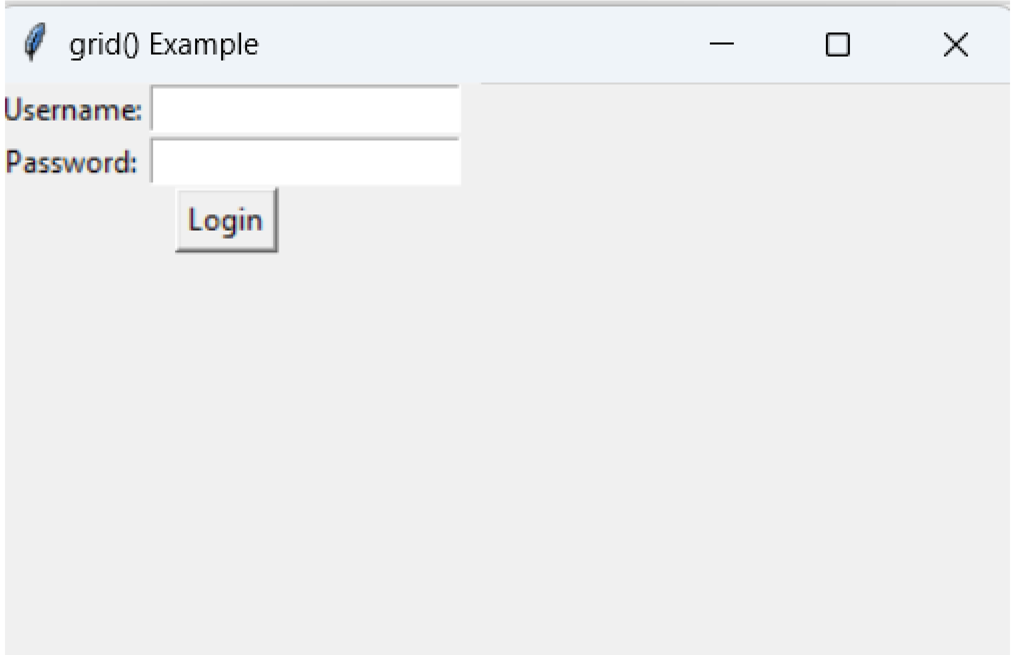


```python
import tkinter as tk

root = tk.Tk()
root.title("pack() Example")

tk.Label(root, text="Label 1").pack()
tk.Label(root, text="Label 2").pack()
tk.Entry(root).pack()
tk.Button(root, text="Submit").pack()

root.mainloop()
```

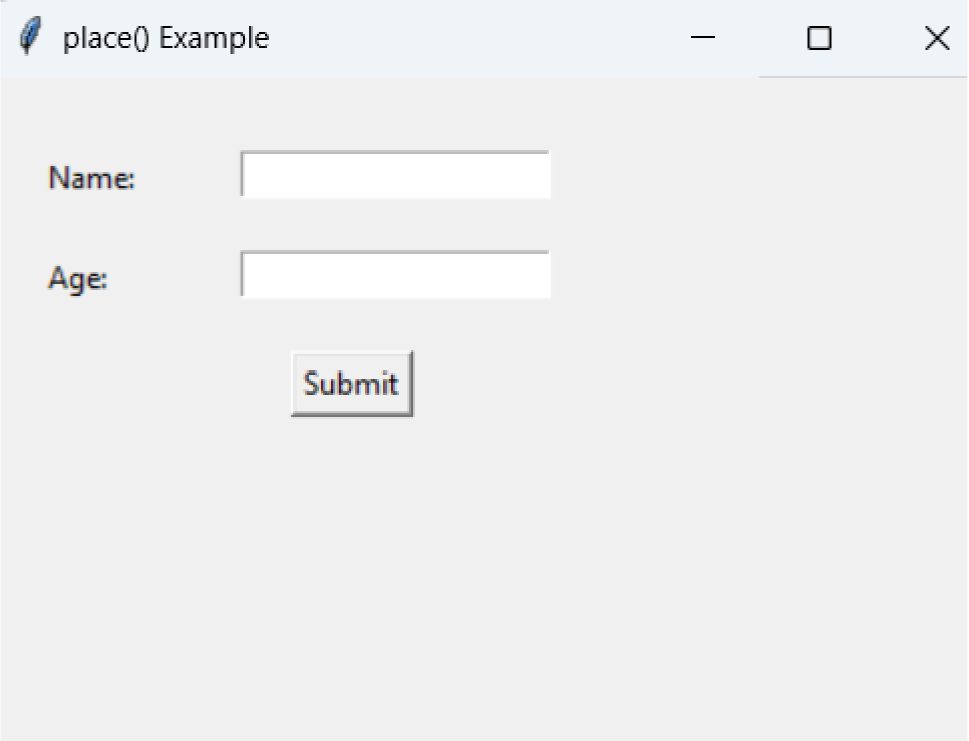


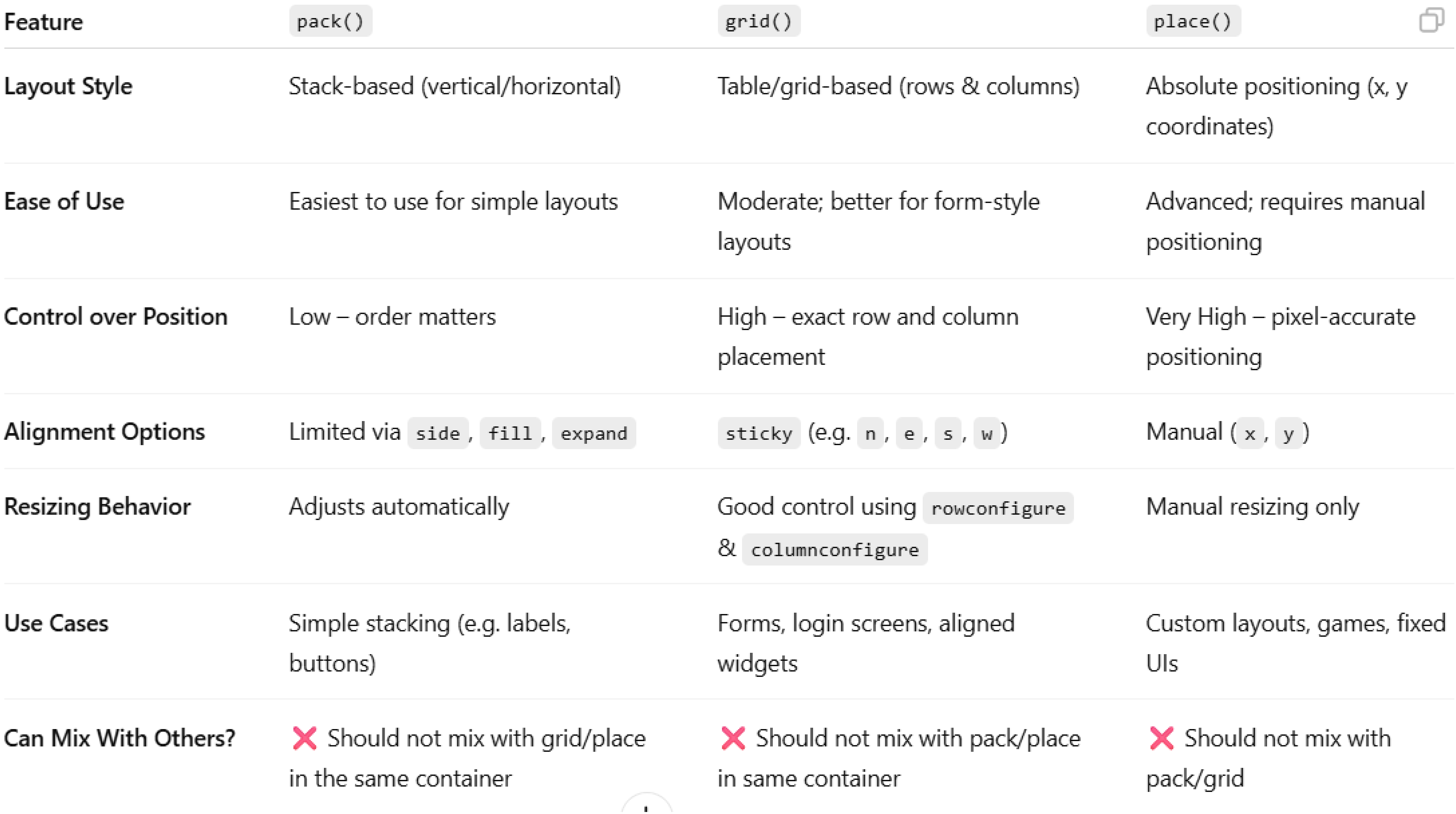
```python
import tkinter as tk

root = tk.Tk()
root.title("grid() Example")

tk.Label(root, text="Username:").grid(row=0, column=0)
tk.Entry(root).grid(row=0, column=1)

tk.Label(root, text="Password:").grid(row=1, column=0)
tk.Entry(root, show="*").grid(row=1, column=1)

tk.Button(root, text="Login").grid(row=2, column=0, columnspan=2)

root.mainloop()
```





```python
import tkinter as tk

root = tk.Tk()
root.title("place() Example")
root.geometry("300x200")

tk.Label(root, text="Name:").place(x=20, y=30)
tk.Entry(root).place(x=100, y=30)

tk.Label(root, text="Age:").place(x=20, y=70)
tk.Entry(root).place(x=100, y=70)

tk.Button(root, text="Submit").place(x=120, y=110)

root.mainloop()
```





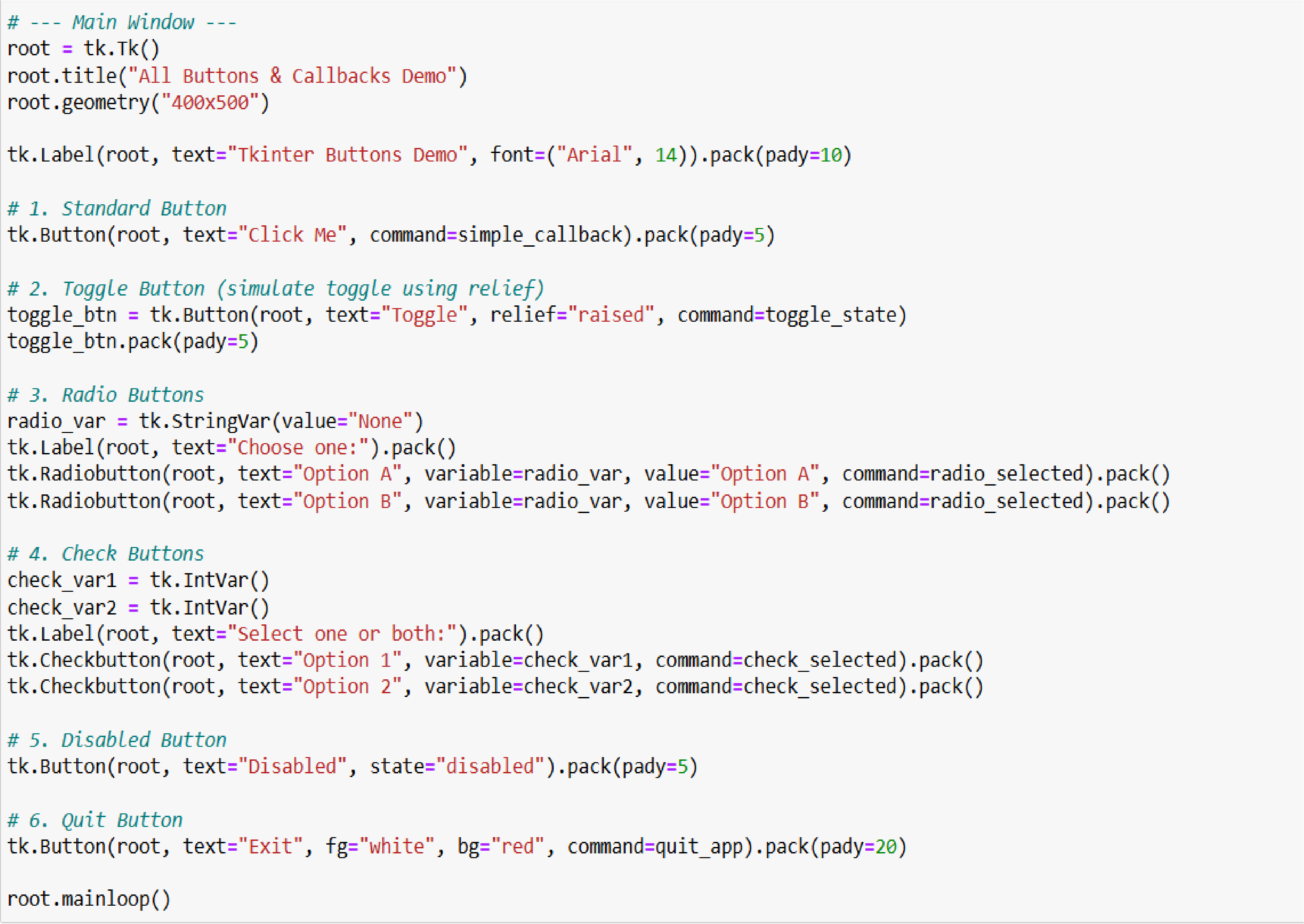
Feature	`pack()`	`grid()`	`place()`
Layout Style	Stack-based (vertical/horizontal)	Table/grid-based (rows & columns)	Absolute positioning (x, y coordinates)
Ease of Use	Easiest to use for simple layouts	Moderate; better for form-style layouts	Advanced; requires manual positioning
Control over Position	Low – order matters	High – exact row and column placement	Very High – pixel-accurate positioning
Alignment Options	Limited via `side`, `fill`, `expand`	`sticky` (e.g. `n`, `e`, `s`, `w`)	Manual (`x`, `y`)
Resizing Behavior	Adjusts automatically	Good control using `rowconfigure` & `columnconfigure`	Manual resizing only
Use Cases	Simple stacking (e.g. labels, buttons)	Forms, login screens, aligned widgets	Custom layouts, games, fixed UIs
Can Mix With Others?	❌ Should not mix with grid/place in the same container	❌ Should not mix with pack/place in same container	❌ Should not mix with pack/grid

5. 3. Buttons and Callbacks

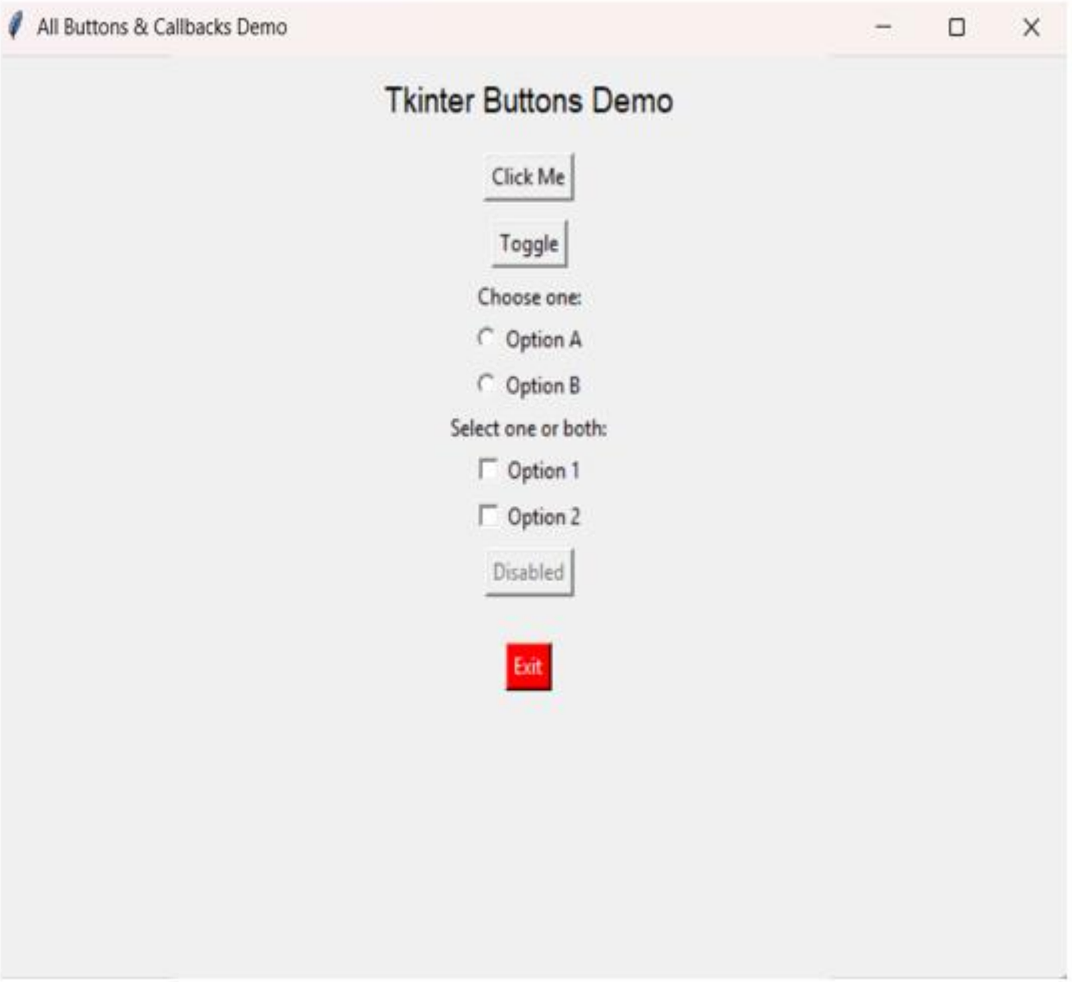
A callback is a function that runs in response to a user action, like clicking a button. In tkinter, you attach callbacks using the command parameter of a Button.

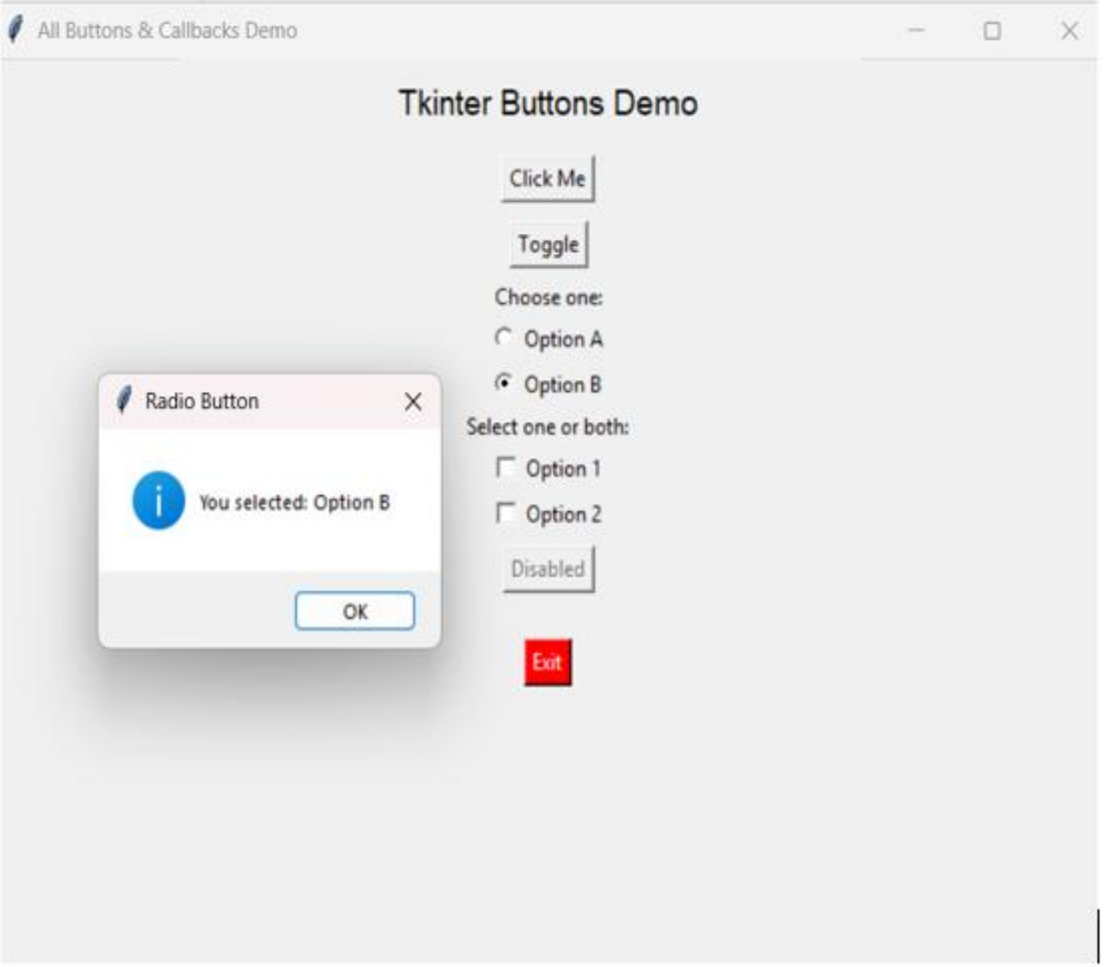
They allow your application to **react** to user actions like:

- Clicking a button
- Choosing a menu option
- Typing in a field



```python
# --- Main Window ---
root = tk.Tk()
root.title("All Buttons & Callbacks Demo")
root.geometry("400x500")

tk.Label(root, text="Tkinter Buttons Demo", font=("Arial", 14)).pack(pady=10)

# 1. Standard Button
tk.Button(root, text="Click Me", command=simple_callback).pack(pady=5)

# 2. Toggle Button (simulate toggle using relief)
toggle_btn = tk.Button(root, text="Toggle", relief="raised", command=toggle_state)
toggle_btn.pack(pady=5)

# 3. Radio Buttons
radio_var = tk.StringVar(value="None")
tk.Label(root, text="Choose one:").pack()
tk.Radiobutton(root, text="Option A", variable=radio_var, value="Option A", command=radio_selected).pack()
tk.Radiobutton(root, text="Option B", variable=radio_var, value="Option B", command=radio_selected).pack()

# 4. Check Buttons
check_var1 = tk.IntVar()
check_var2 = tk.IntVar()
tk.Label(root, text="Select one or both:").pack()
tk.Checkbutton(root, text="Option 1", variable=check_var1, command=check_selected).pack()
tk.Checkbutton(root, text="Option 2", variable=check_var2, command=check_selected).pack()

# 5. Disabled Button
tk.Button(root, text="Disabled", state="disabled").pack(pady=5)

# 6. Quit Button
tk.Button(root, text="Exit", fg="white", bg="red", command=quit_app).pack(pady=20)

root.mainloop()
```





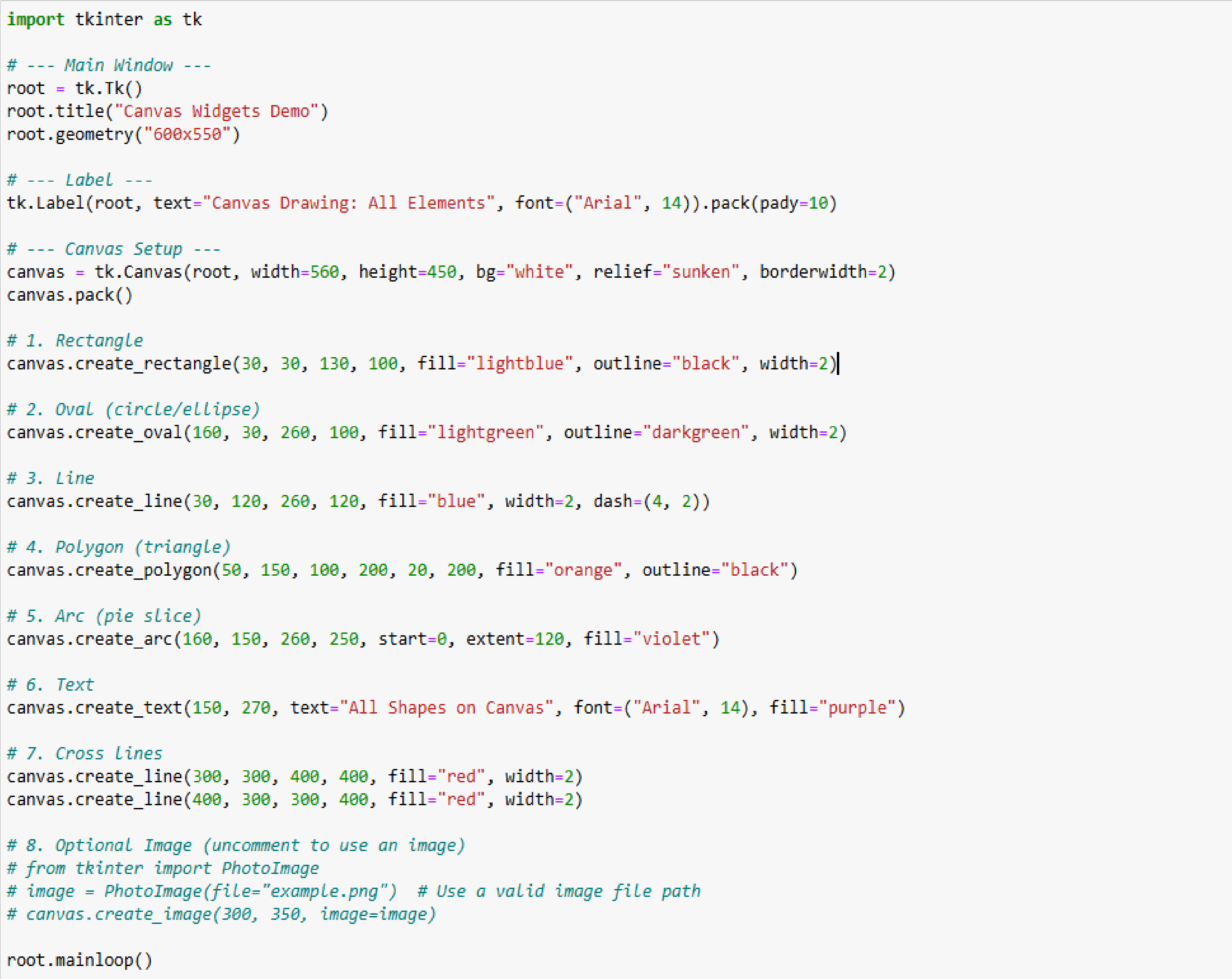
Radio Button

You selected: Option B

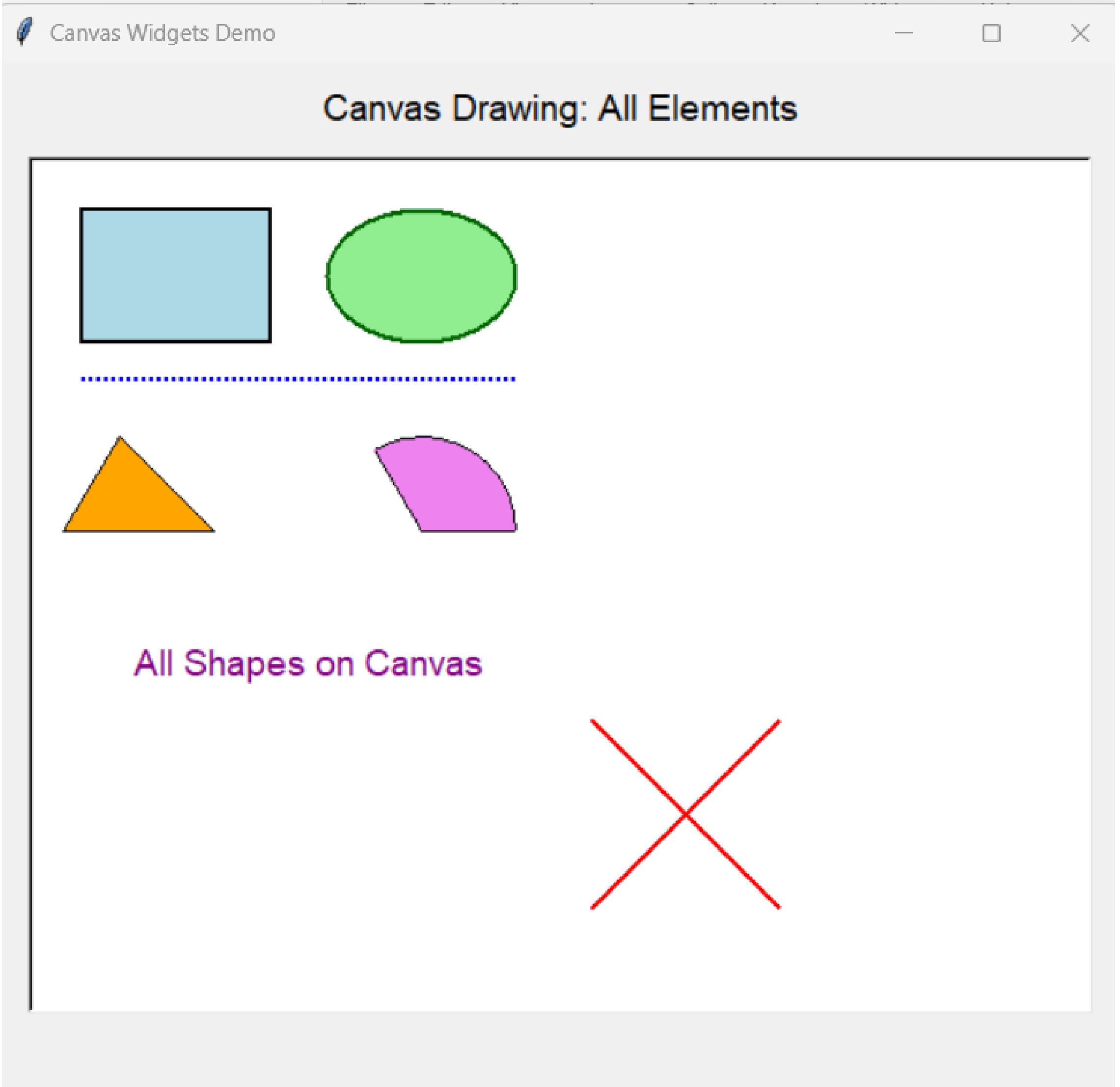
4. Canvas Widgets

The Canvas widget allows you to **draw shapes**, such as rectangles, circles, and text. It is useful for:

- Custom graphics
- Drawing lines, arcs, or images
- Games or visual dashboards



```
import tkinter as tk

# --- Main Window ---
root = tk.Tk()
root.title("Canvas Widgets Demo")
root.geometry("600x550")

# --- Label ---
tk.Label(root, text="Canvas Drawing: All Elements", font=("Arial", 14)).pack(pady=10)

# --- Canvas Setup ---
canvas = tk.Canvas(root, width=560, height=450, bg="white", relief="sunken", borderwidth=2)
canvas.pack()

# 1. Rectangle
canvas.create_rectangle(30, 30, 130, 100, fill="lightblue", outline="black", width=2)

# 2. Oval (circle/ellipse)
canvas.create_oval(160, 30, 260, 100, fill="lightgreen", outline="darkgreen", width=2)

# 3. Line
canvas.create_line(30, 120, 260, 120, fill="blue", width=2, dash=(4, 2))

# 4. Polygon (triangle)
canvas.create_polygon(50, 150, 100, 200, 20, 200, fill="orange", outline="black")

# 5. Arc (pie slice)
canvas.create_arc(160, 150, 260, 250, start=0, extent=120, fill="violet")

# 6. Text
canvas.create_text(150, 270, text="All Shapes on Canvas", font=("Arial", 14), fill="purple")

# 7. Cross Lines
canvas.create_line(300, 300, 400, 400, fill="red", width=2)
canvas.create_line(400, 300, 300, 400, fill="red", width=2)

# 8. Optional Image (uncomment to use an image)
# from tkinter import PhotoImage
# image = PhotoImage(file="example.png")  # Use a valid image file path
# canvas.create_image(300, 350, image=image)

root.mainloop()
```

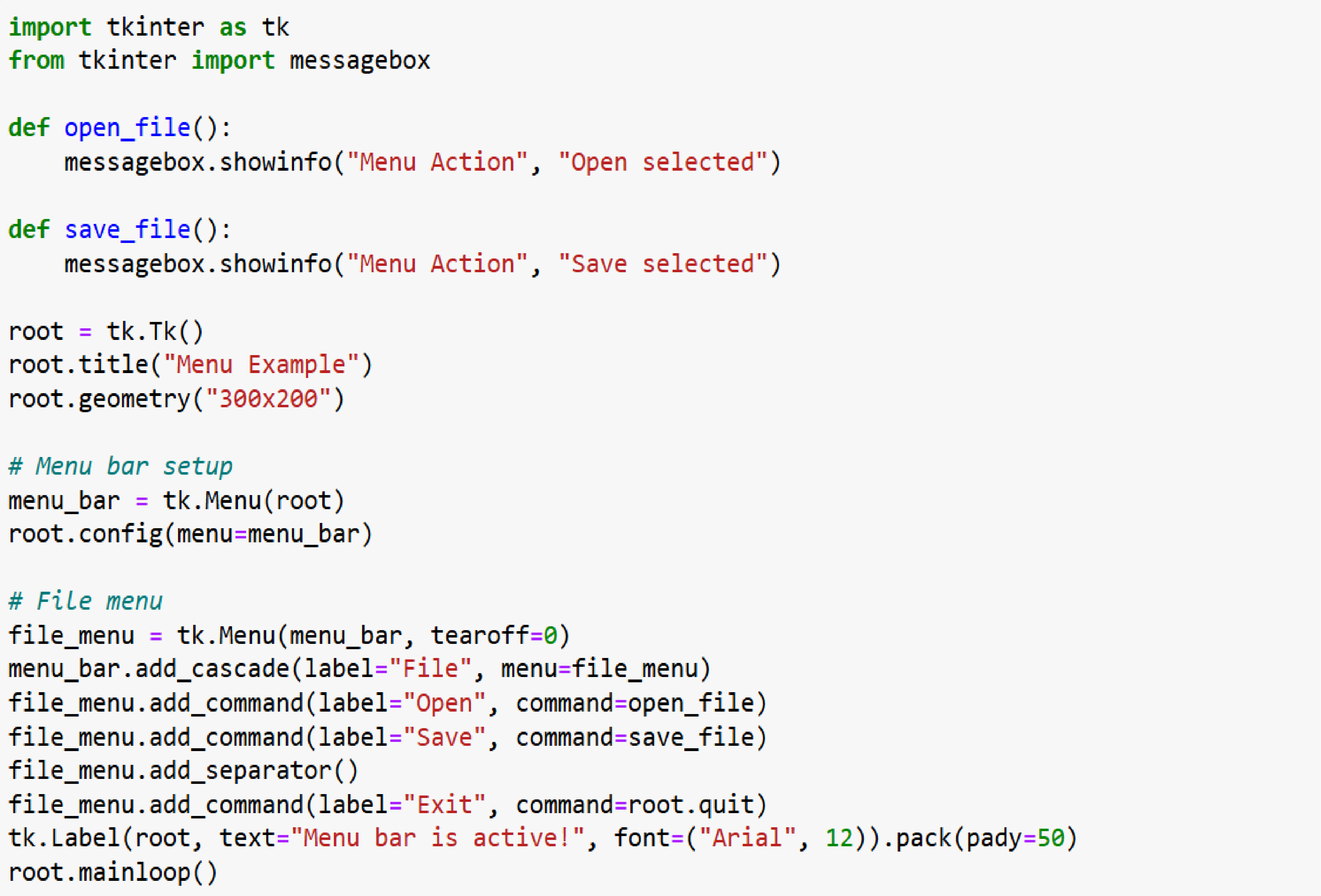

Canvas Widgets Demo
Canvas Drawing: All Elements
All Shapes on Canvas

5. Menus

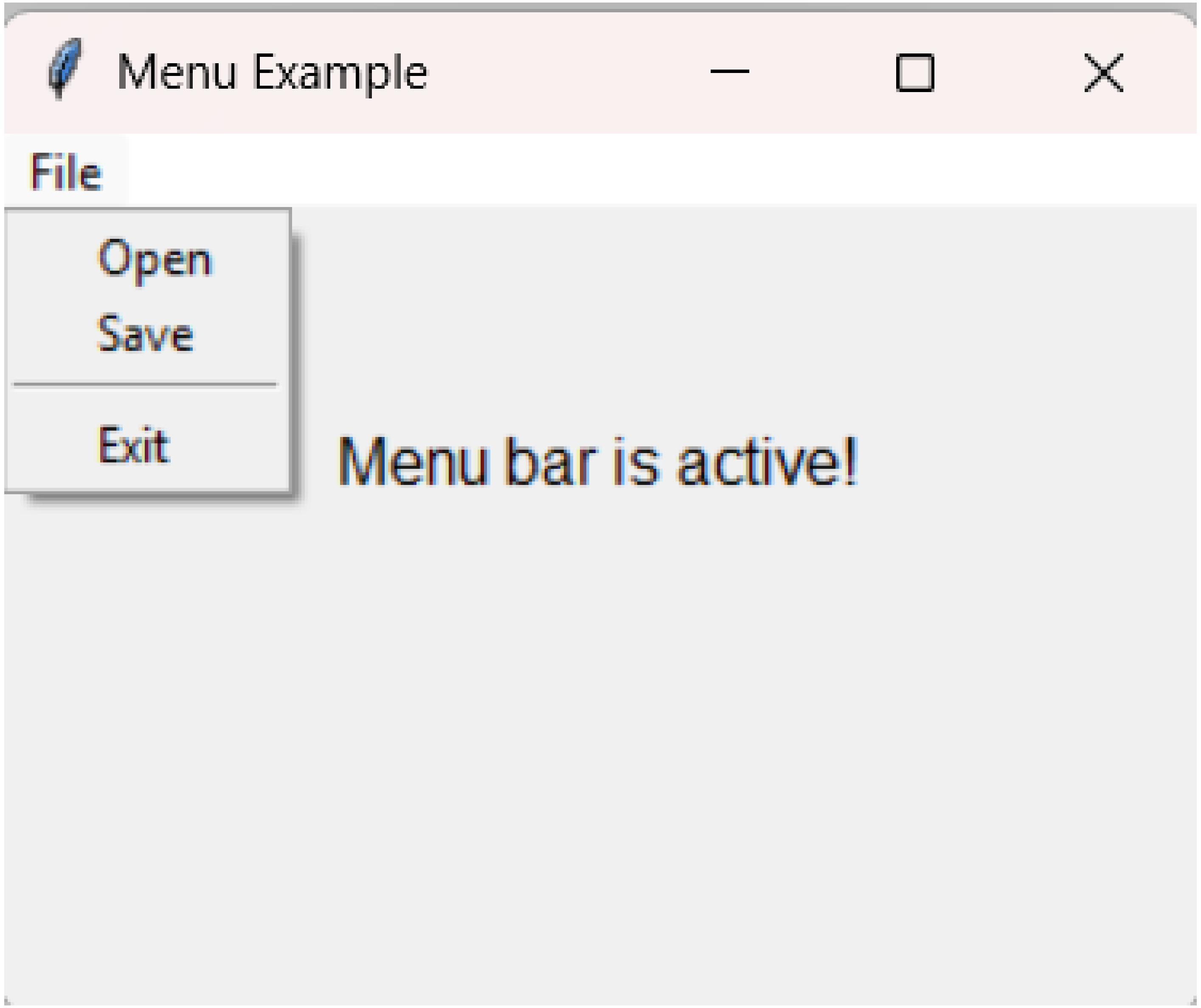
Menus provide a way to **organize commands** like Open, Save, Exit. You use Menu() objects and add them to the main window using config(menu=menu).

Menus can contain:

- **Menu items** (commands)
- **Separators**
- **Submenus**



```python
import tkinter as tk
from tkinter import messagebox

def open_file():
    messagebox.showinfo("Menu Action", "Open selected")

def save_file():
    messagebox.showinfo("Menu Action", "Save selected")

root = tk.Tk()
root.title("Menu Example")
root.geometry("300x200")

# Menu bar setup
menu_bar = tk.Menu(root)
root.config(menu=menu_bar)

# File menu
file_menu = tk.Menu(menu_bar, tearoff=0)
menu_bar.add_cascade(label="File", menu=file_menu)
file_menu.add_command(label="Open", command=open_file)
file_menu.add_command(label="Save", command=save_file)
file_menu.add_separator()
file_menu.add_command(label="Exit", command=root.quit)
tk.Label(root, text="Menu bar is active!", font=("Arial", 12)).pack(pady=50)
root.mainloop()
```



Menu bar is active!

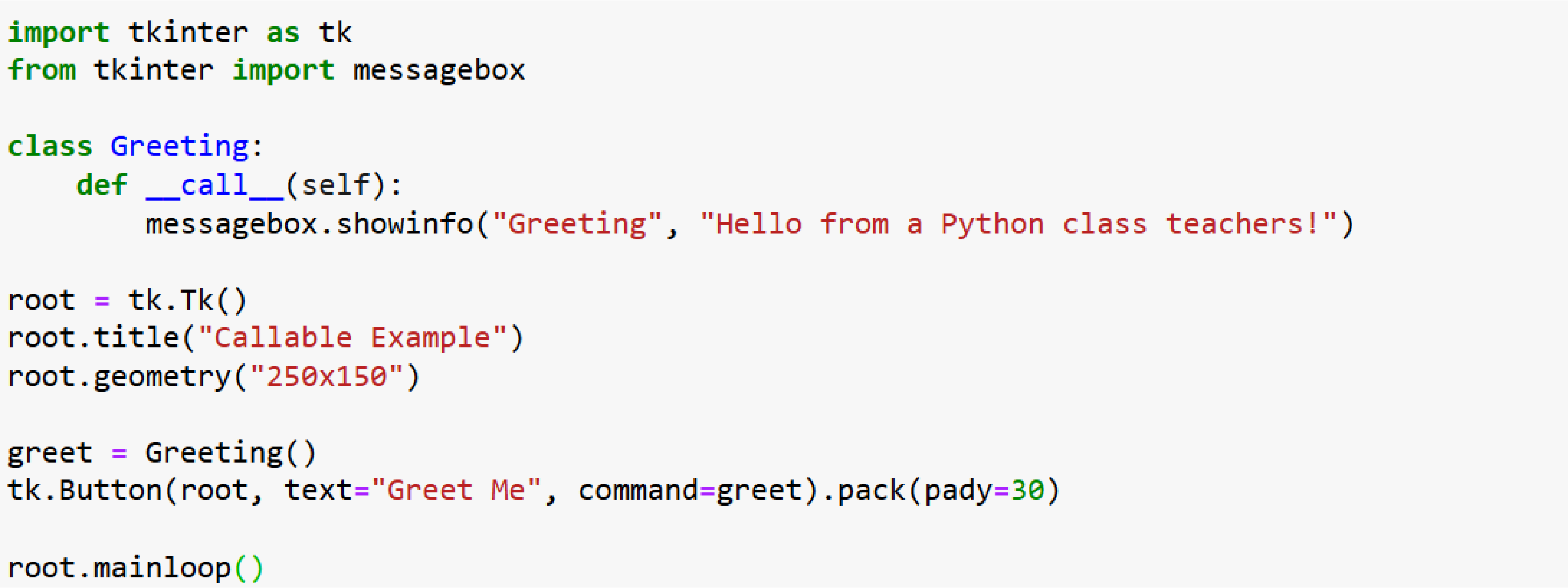
6. Callable Functions

In Python, a **callable** is anything that can be executed using parentheses, like a function or an object with a __call__ method. In tkinter, the command= parameter accepts any callable. You can define classes that act like functions by implementing the __call__ method.

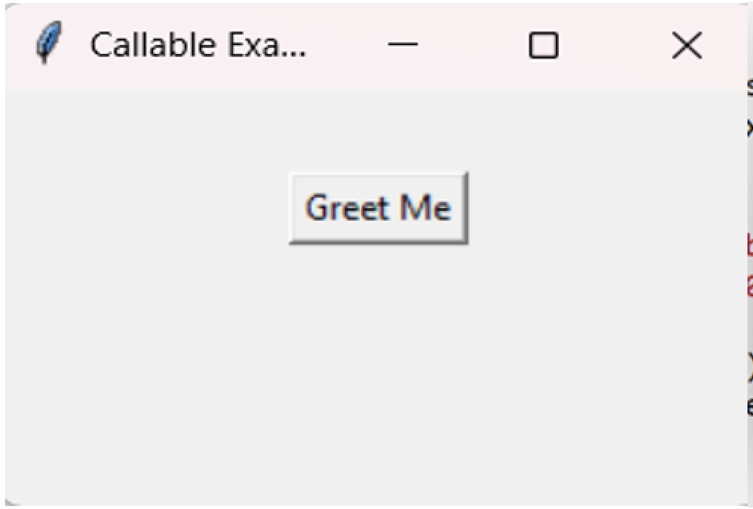
Instead of passing just a function, you can pass a **callable object** that has more data or logic inside it — useful in object-oriented GUI applications.

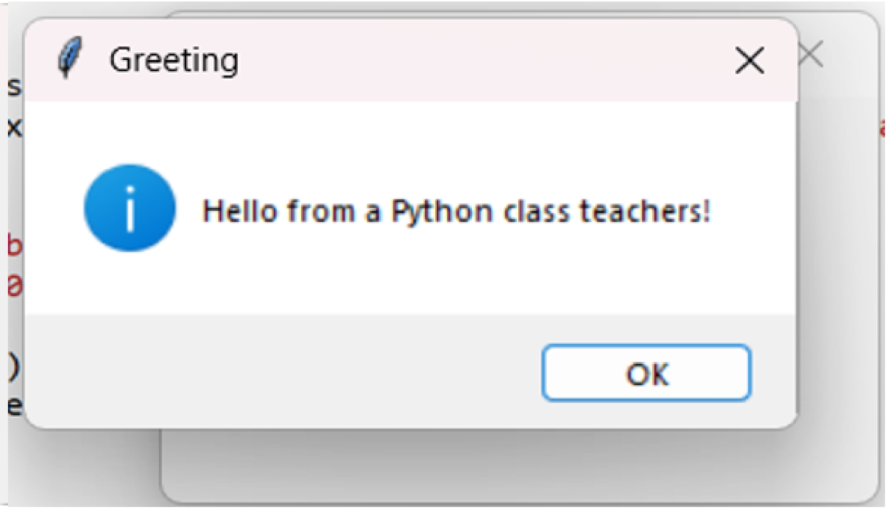
In Python, anything we can "call" with parentheses is a **callable**. This includes:

- Functions
- Lambdas
- Classes with a __call__() method



```python
import tkinter as tk
from tkinter import messagebox

class Greeting:
    def __call__(self):
        messagebox.showinfo("Greeting", "Hello from a Python class teachers!")

root = tk.Tk()
root.title("Callable Example")
root.geometry("250x150")

greet = Greeting()
tk.Button(root, text="Greet Me", command=greet).pack(pady=30)

root.mainloop()
```

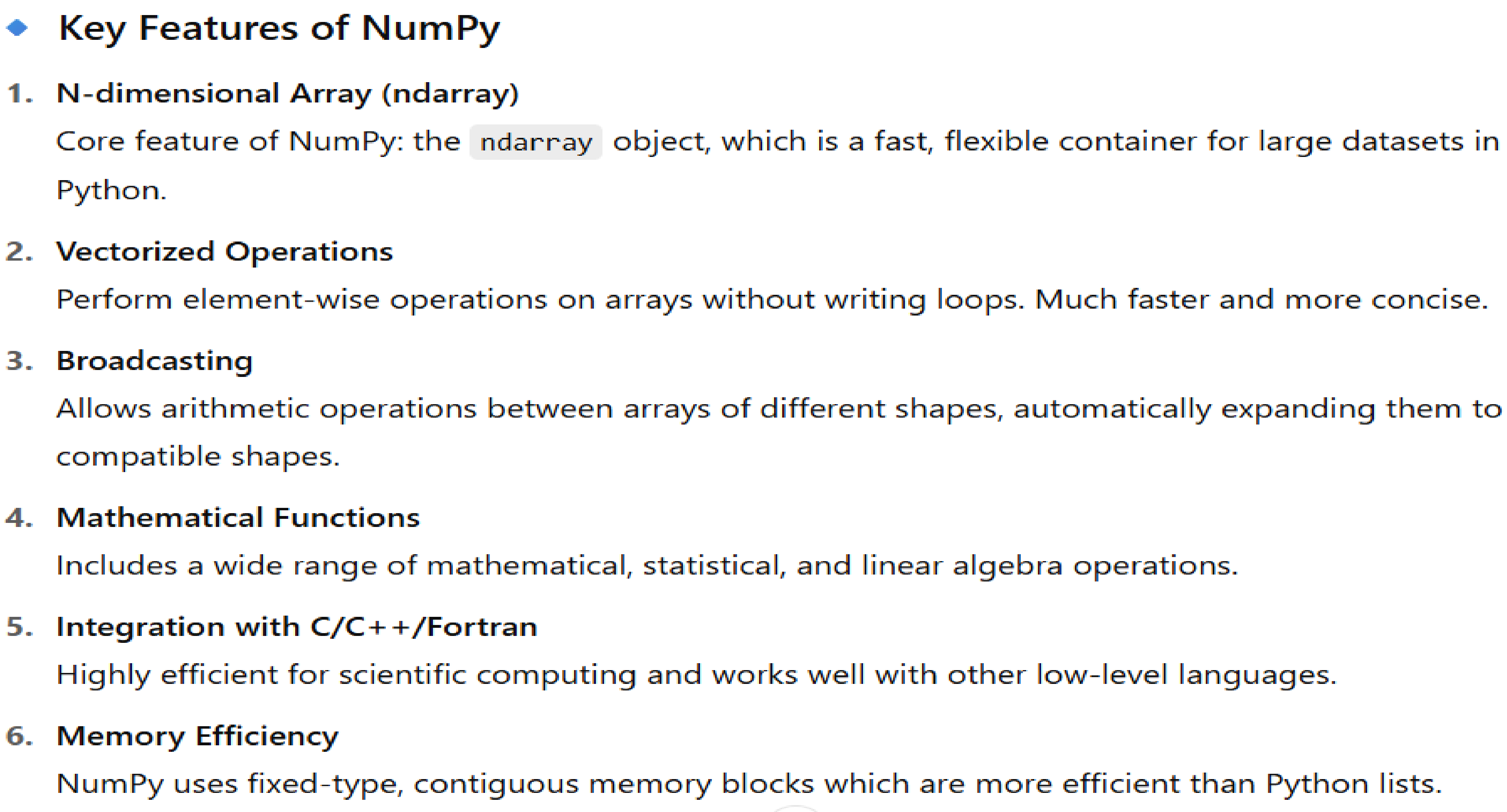




Hello from a Python class teachers!

NumPy in Python

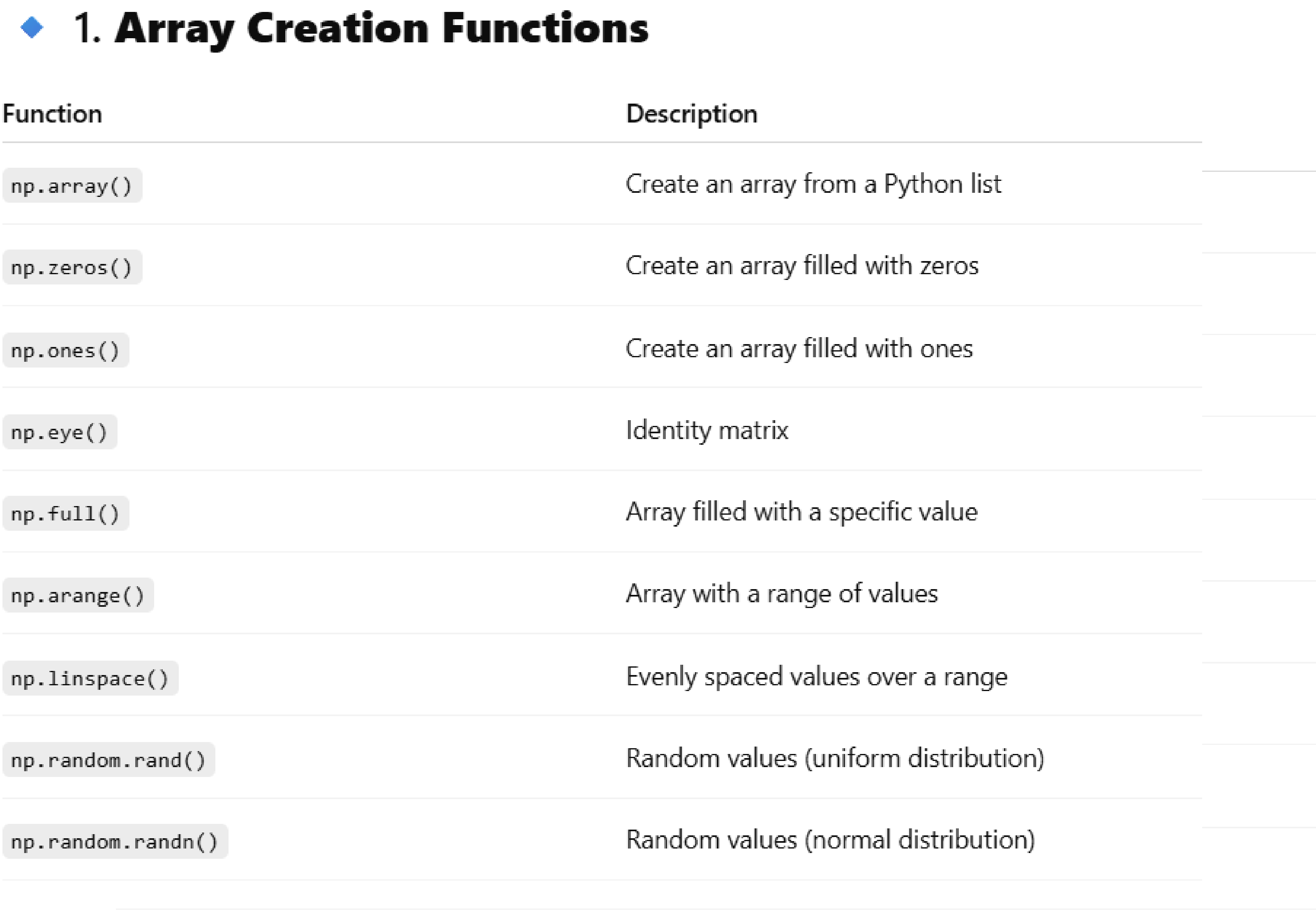
NumPy (short for **Numerical Python**) is a powerful open-source library in Python used for **numerical computing**. It provides efficient tools for working with **arrays**, performing **mathematical operations**, and manipulating large datasets.



◆ Key Features of NumPy

1. **N-dimensional Array (ndarray)**
   Core feature of NumPy: the `ndarray` object, which is a fast, flexible container for large datasets in Python.
2. **Vectorized Operations**
   Perform element-wise operations on arrays without writing loops. Much faster and more concise.
3. **Broadcasting**
   Allows arithmetic operations between arrays of different shapes, automatically expanding them to compatible shapes.
4. **Mathematical Functions**
   Includes a wide range of mathematical, statistical, and linear algebra operations.
5. **Integration with C/C++/Fortran**
   Highly efficient for scientific computing and works well with other low-level languages.
6. **Memory Efficiency**
   NumPy uses fixed-type, contiguous memory blocks which are more efficient than Python lists.

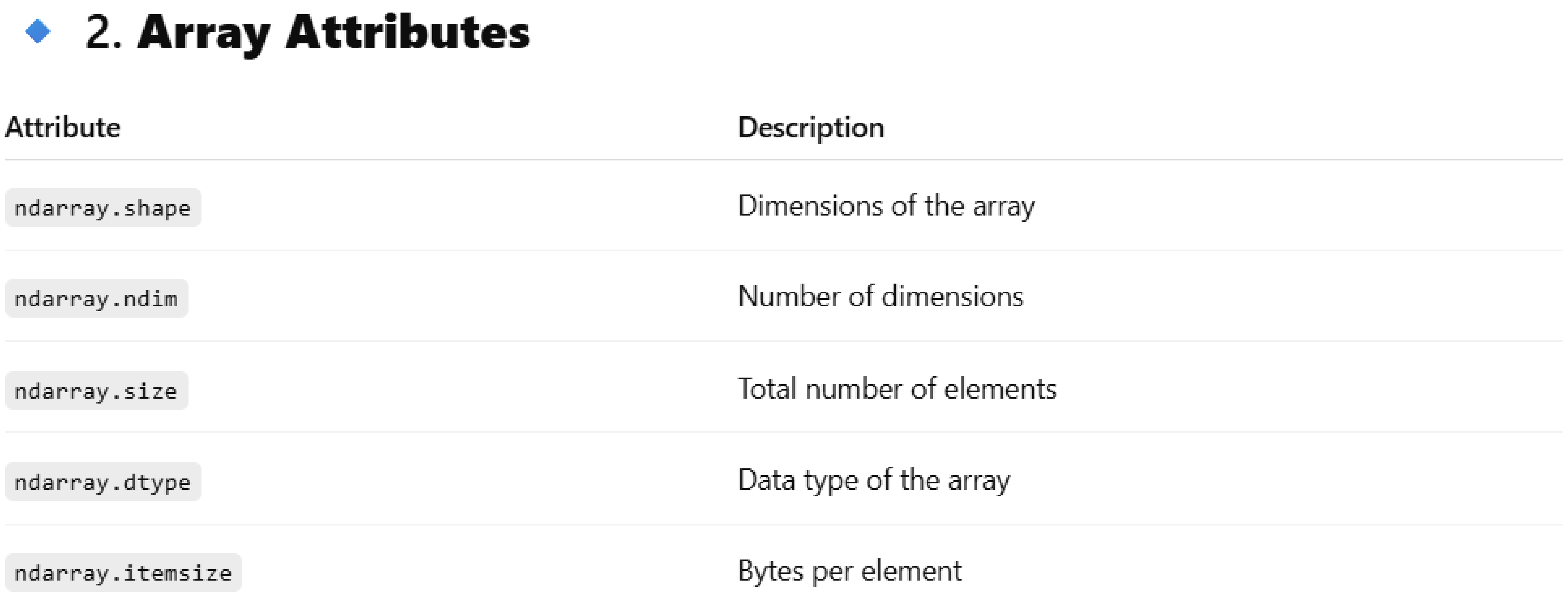
Useful NumPy in Python



◆ 1. **Array Creation Functions**

Function	Description
`np.array()`	Create an array from a Python list
`np.zeros()`	Create an array filled with zeros
`np.ones()`	Create an array filled with ones
`np.eye()`	Identity matrix
`np.full()`	Array filled with a specific value
`np.arange()`	Array with a range of values
`np.linspace()`	Evenly spaced values over a range
`np.random.rand()`	Random values (uniform distribution)
`np.random.randn()`	Random values (normal distribution)

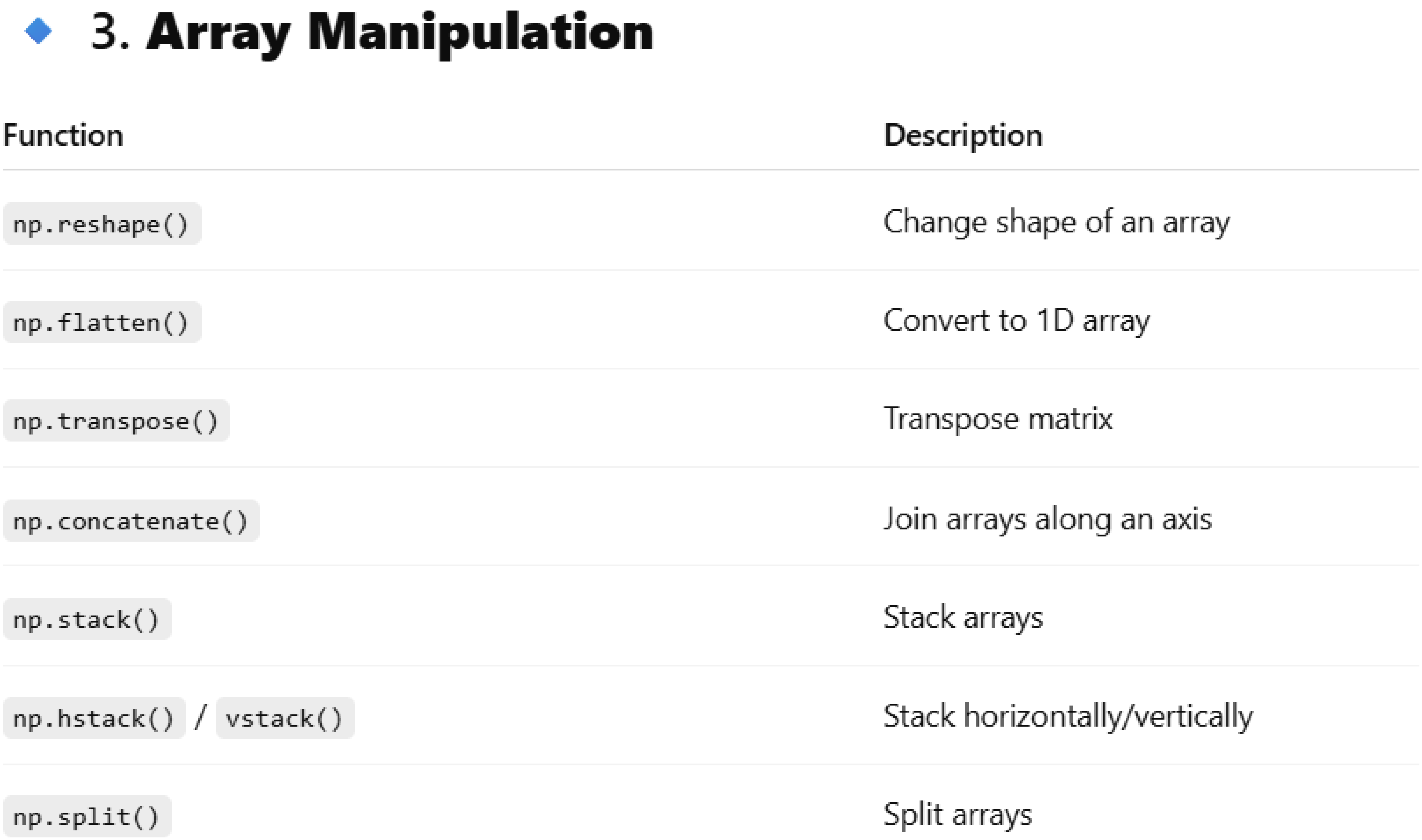
Useful NumPy in Python



🔹 2. **Array Attributes**

Attribute	Description
`ndarray.shape`	Dimensions of the array
`ndarray.ndim`	Number of dimensions
`ndarray.size`	Total number of elements
`ndarray.dtype`	Data type of the array
`ndarray.itemsize`	Bytes per element

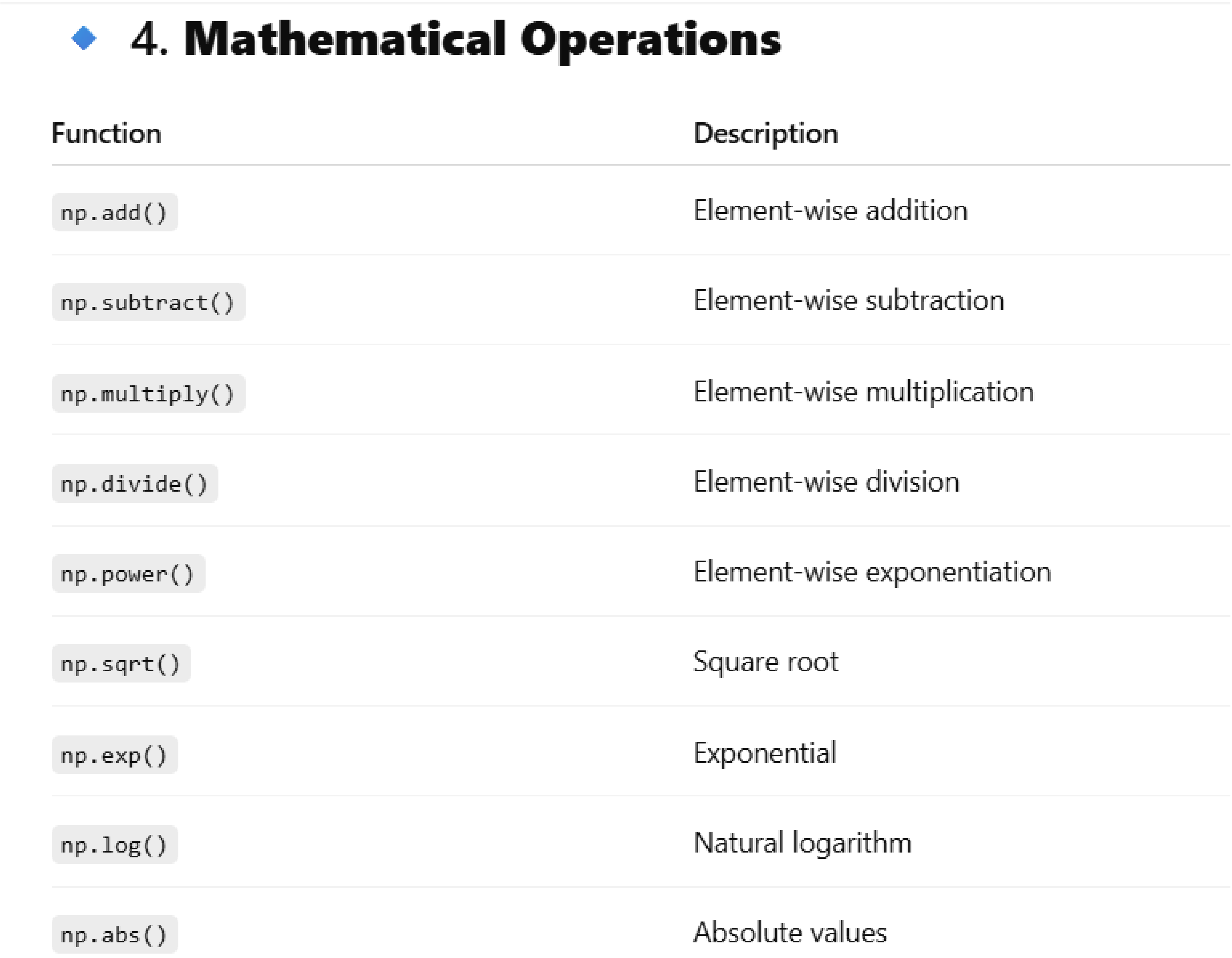
Useful NumPy in Python



🔷 3. **Array Manipulation**

Function	Description
`np.reshape()`	Change shape of an array
`np.flatten()`	Convert to 1D array
`np.transpose()`	Transpose matrix
`np.concatenate()`	Join arrays along an axis
`np.stack()`	Stack arrays
`np.hstack()` / `vstack()`	Stack horizontally/vertically
`np.split()`	Split arrays

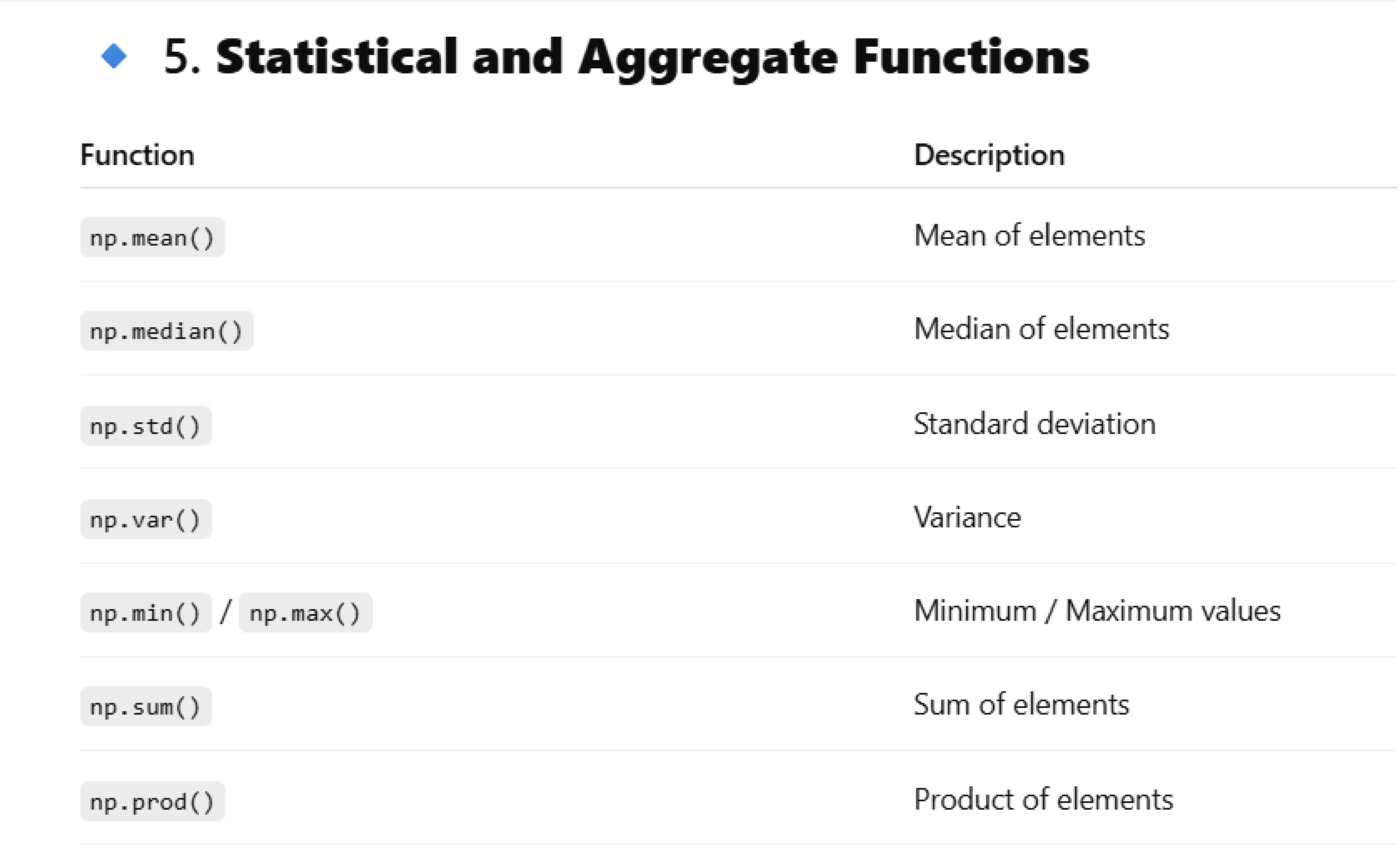
Useful NumPy in Python



🔹 4. Mathematical Operations

Function	Description
`np.add()`	Element-wise addition
`np.subtract()`	Element-wise subtraction
`np.multiply()`	Element-wise multiplication
`np.divide()`	Element-wise division
`np.power()`	Element-wise exponentiation
`np.sqrt()`	Square root
`np.exp()`	Exponential
`np.log()`	Natural logarithm
`np.abs()`	Absolute values

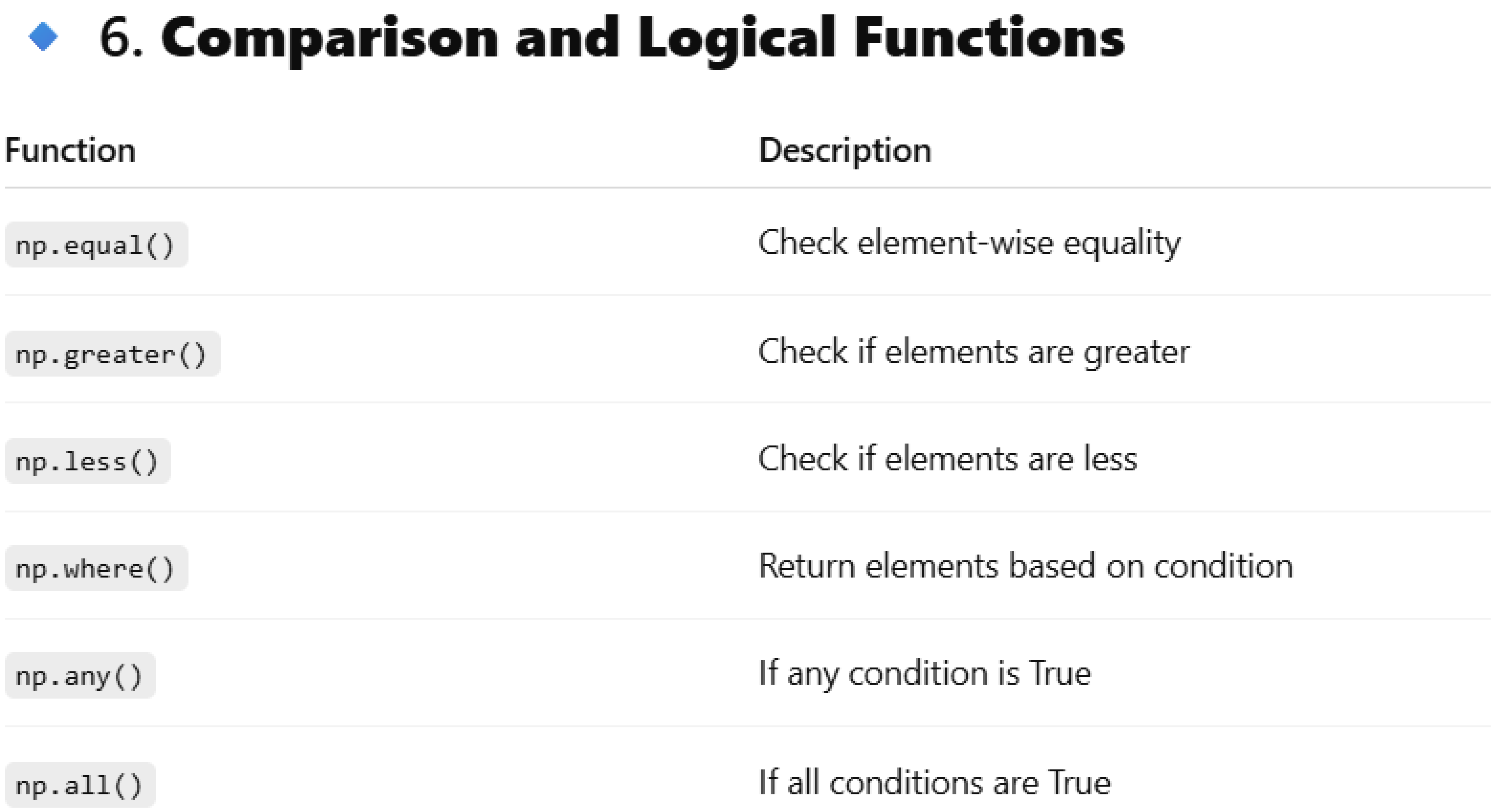
Useful NumPy in Python



🔹 5. Statistical and Aggregate Functions

Function	Description
`np.mean()`	Mean of elements
`np.median()`	Median of elements
`np.std()`	Standard deviation
`np.var()`	Variance
`np.min()` / `np.max()`	Minimum / Maximum values
`np.sum()`	Sum of elements
`np.prod()`	Product of elements

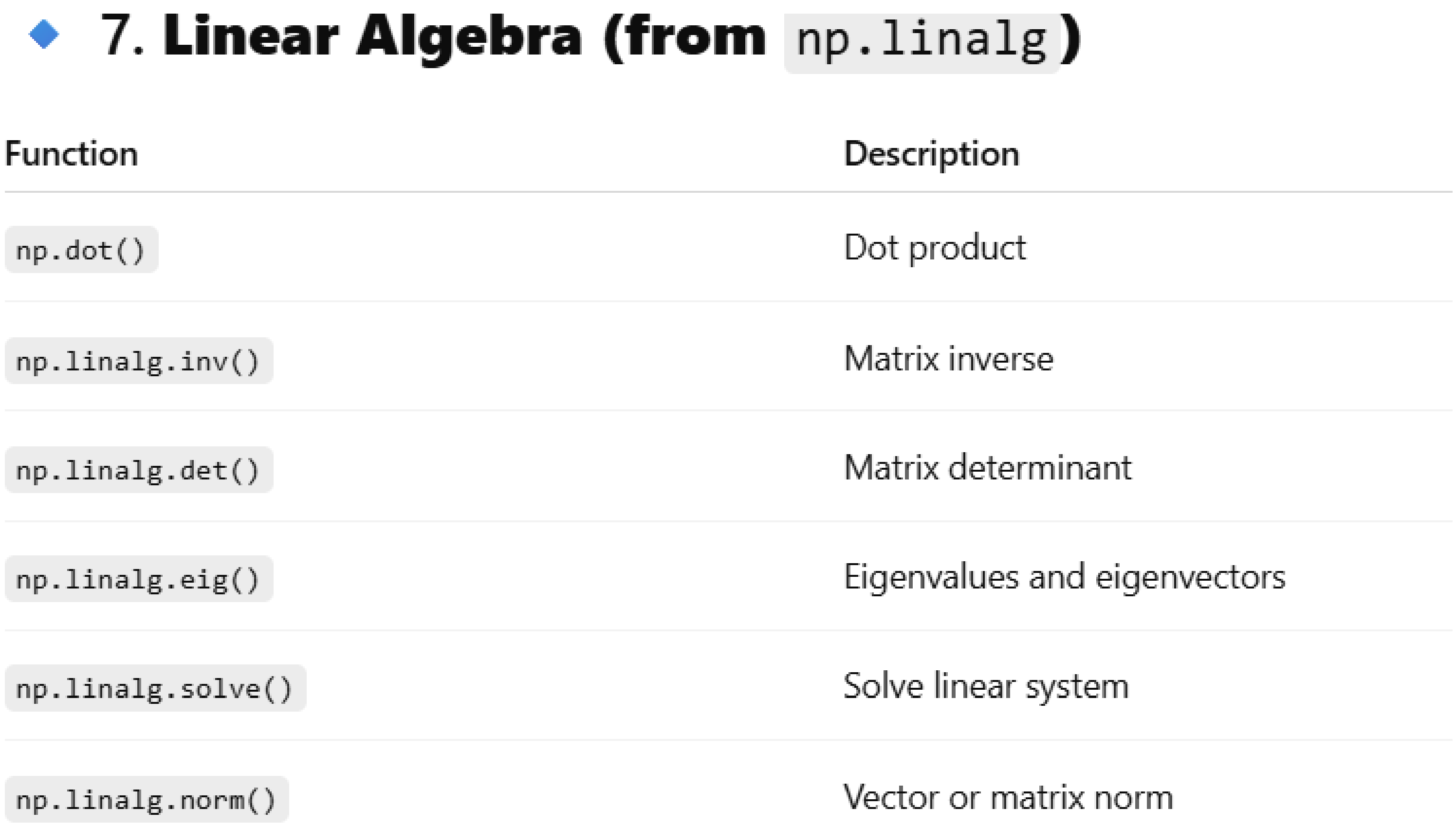
Useful NumPy in Python



🔹 6. **Comparison and Logical Functions**

Function	Description
`np.equal()`	Check element-wise equality
`np.greater()`	Check if elements are greater
`np.less()`	Check if elements are less
`np.where()`	Return elements based on condition
`np.any()`	If any condition is True
`np.all()`	If all conditions are True

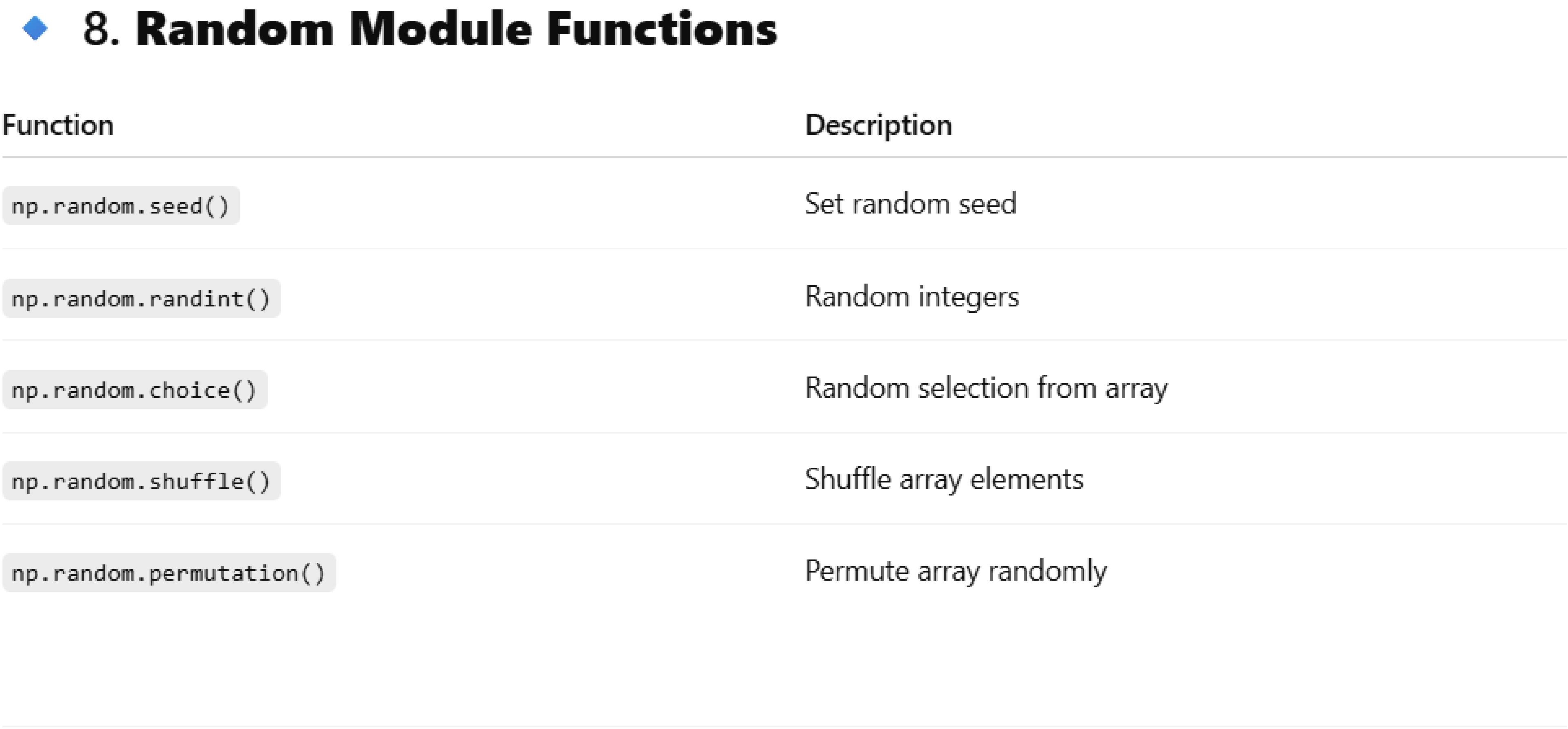
Useful NumPy in Python



◆ 7. **Linear Algebra (from `np.linalg`)**

Function	Description
`np.dot()`	Dot product
`np.linalg.inv()`	Matrix inverse
`np.linalg.det()`	Matrix determinant
`np.linalg.eig()`	Eigenvalues and eigenvectors
`np.linalg.solve()`	Solve linear system
`np.linalg.norm()`	Vector or matrix norm

Useful NumPy in Python

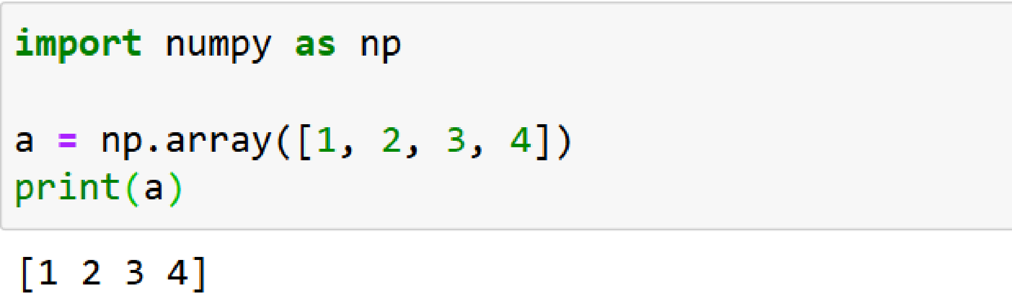


◆ 8. **Random Module Functions**

Function	Description
`np.random.seed()`	Set random seed
`np.random.randint()`	Random integers
`np.random.choice()`	Random selection from array
`np.random.shuffle()`	Shuffle array elements
`np.random.permutation()`	Permute array randomly

Creating Arrays

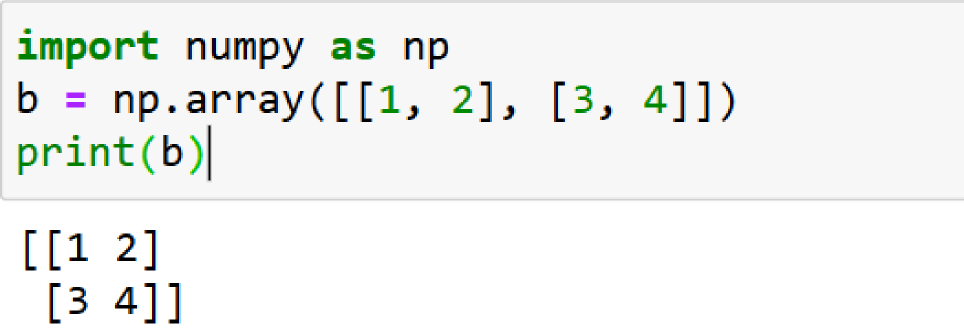
From a list



```python
import numpy as np

a = np.array([1, 2, 3, 4])
print(a)
```

```
[1 2 3 4]
```

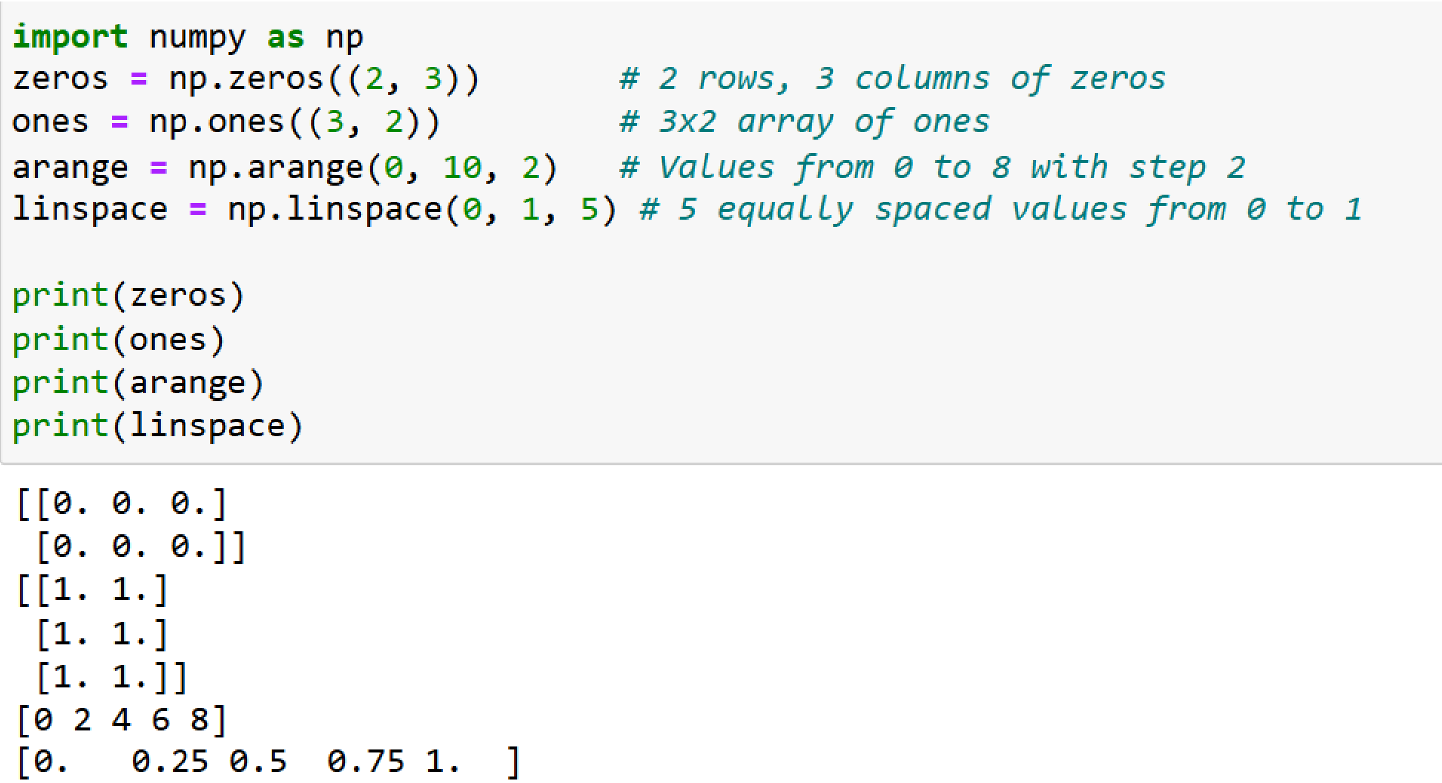
Multi-dimensional array:



```python
import numpy as np
b = np.array([[1, 2], [3, 4]])
print(b)
```

```
[[1 2]
 [3 4]]
```

Using built-in functions:

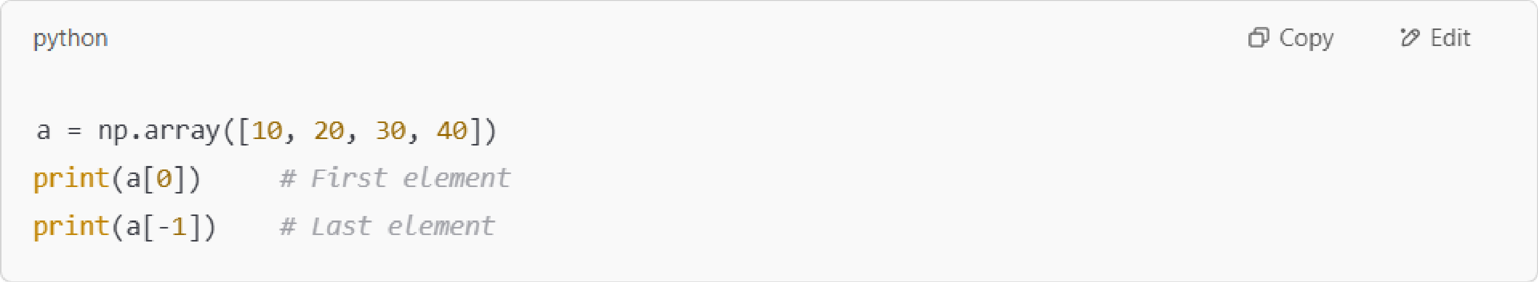


```python
import numpy as np
zeros = np.zeros((2, 3))        # 2 rows, 3 columns of zeros
ones = np.ones((3, 2))          # 3x2 array of ones
arange = np.arange(0, 10, 2)    # Values from 0 to 8 with step 2
linspace = np.linspace(0, 1, 5) # 5 equally spaced values from 0 to 1

print(zeros)
print(ones)
print(arange)
print(linspace)
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
[[1. 1.]
 [1. 1.]
 [1. 1.]]
[0 2 4 6 8]
[0.   0.25 0.5  0.75 1.  ]
```

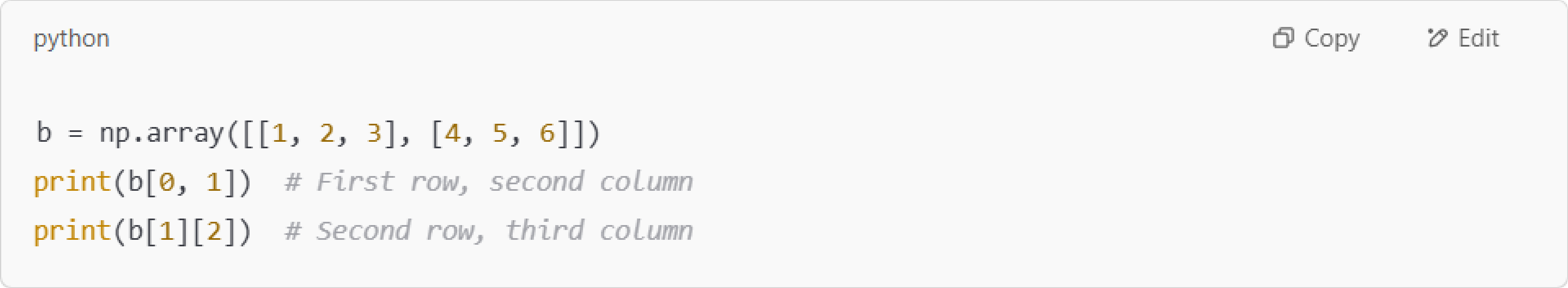
Indexing Arrays

➤ Accessing elements (1D):



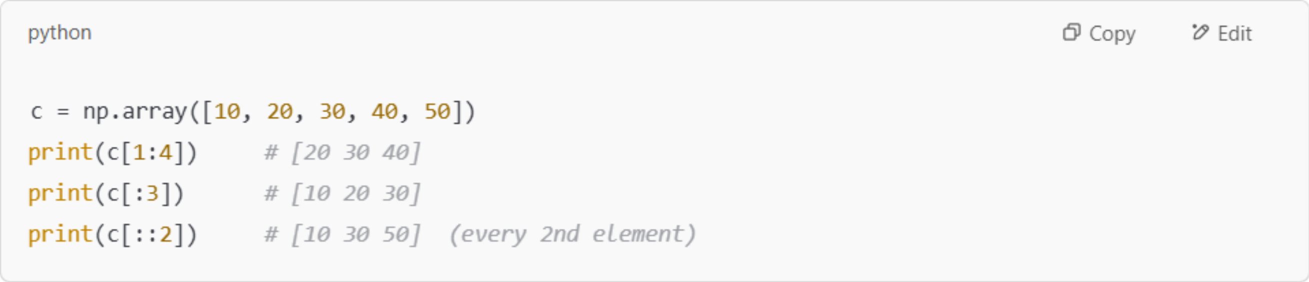
```python
a = np.array([10, 20, 30, 40])
print(a[0])     # First element
print(a[-1])    # Last element
```

➤ Accessing elements (2D):



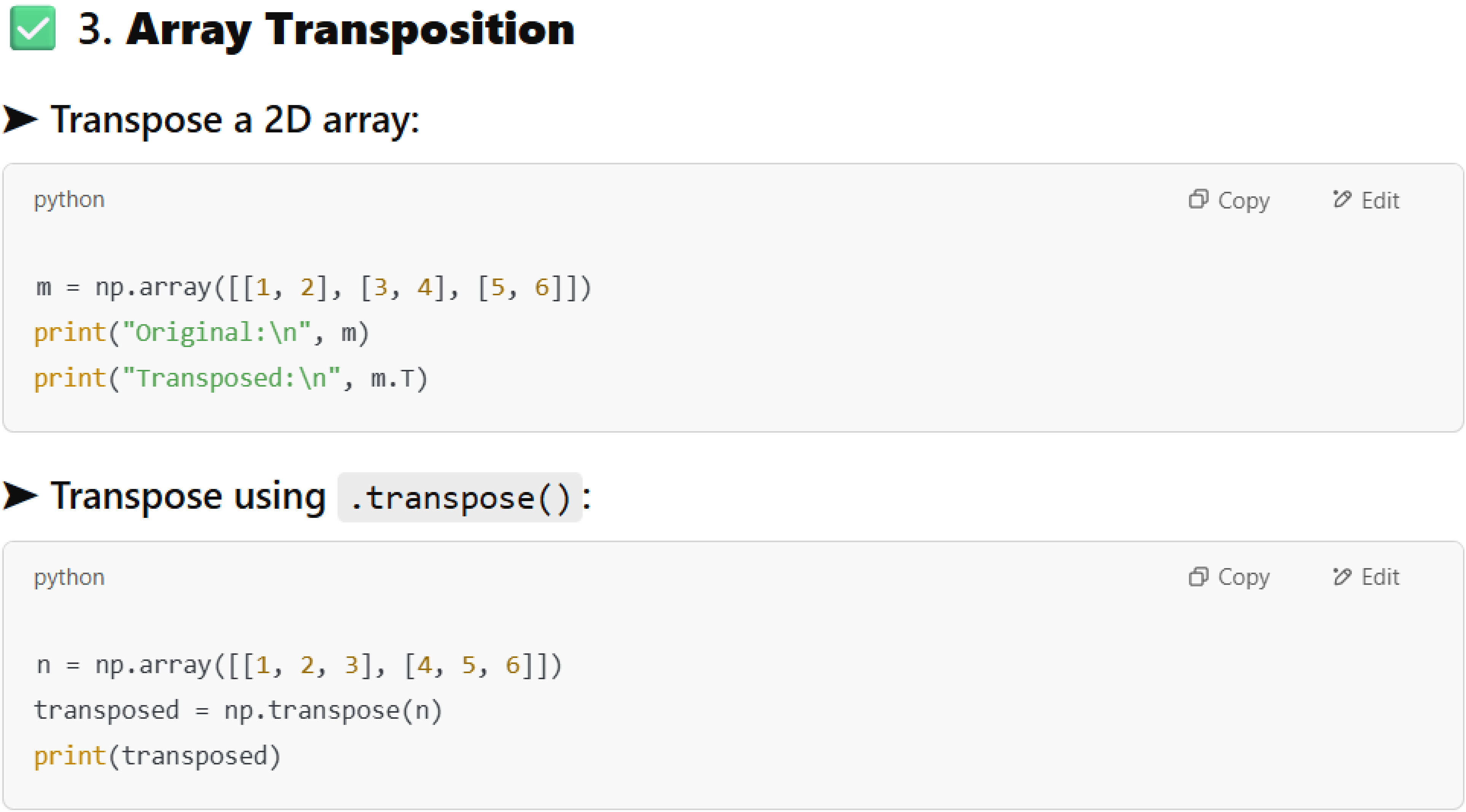
```python
b = np.array([[1, 2, 3], [4, 5, 6]])
print(b[0, 1])  # First row, second column
print(b[1][2])  # Second row, third column
```

➤ Slicing:



```python
c = np.array([10, 20, 30, 40, 50])
print(c[1:4])     # [20 30 40]
print(c[:3])      # [10 20 30]
print(c[::2])     # [10 30 50]  (every 2nd element)
```

Array Transposition



✅ 3. **Array Transposition**

➤ Transpose a 2D array:

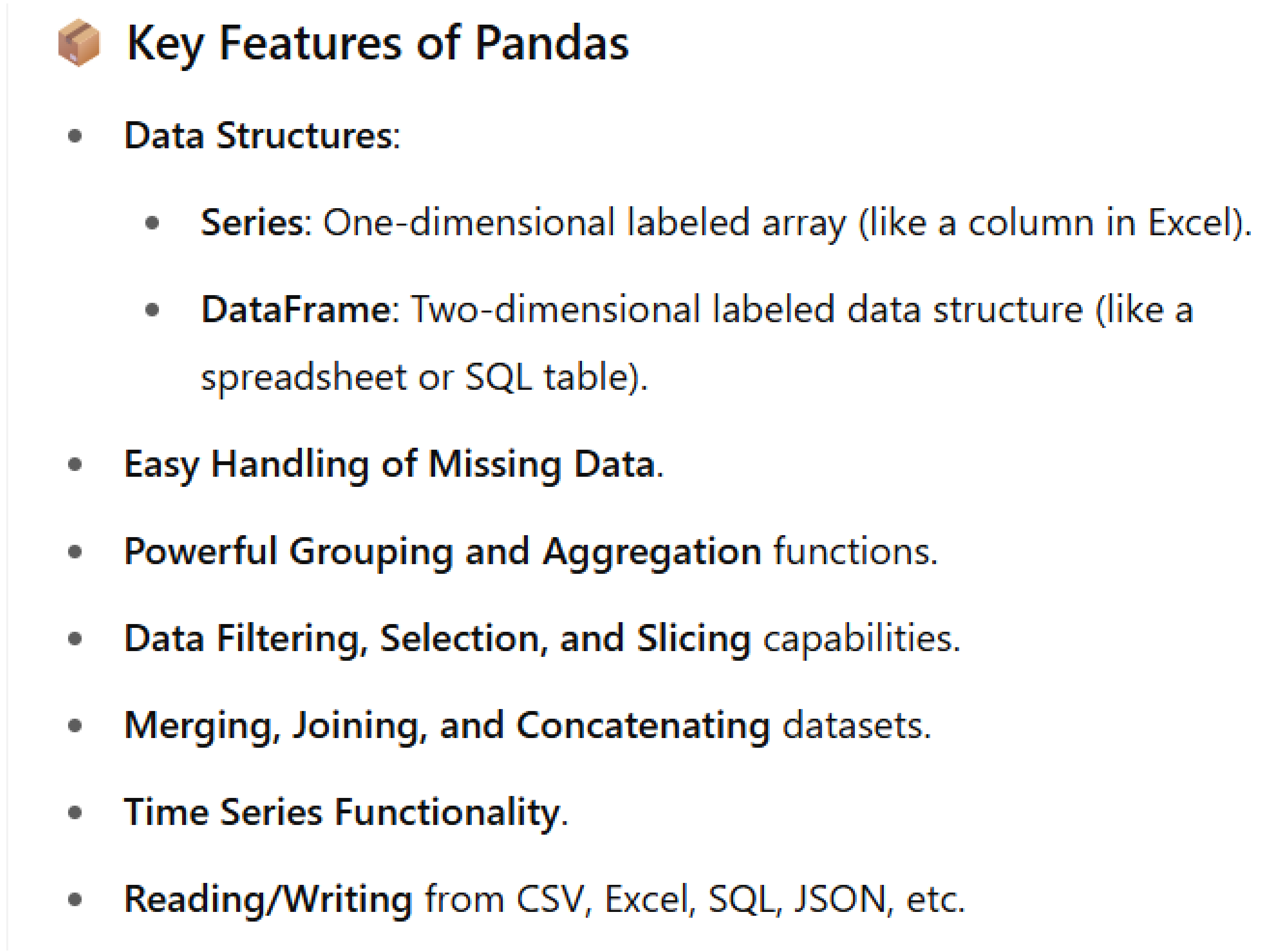
```python
m = np.array([[1, 2], [3, 4], [5, 6]])
print("Original:\n", m)
print("Transposed:\n", m.T)
```

➤ Transpose using `.transpose()`:

```python
n = np.array([[1, 2, 3], [4, 5, 6]])
transposed = np.transpose(n)
print(transposed)
```

Pandas in Python

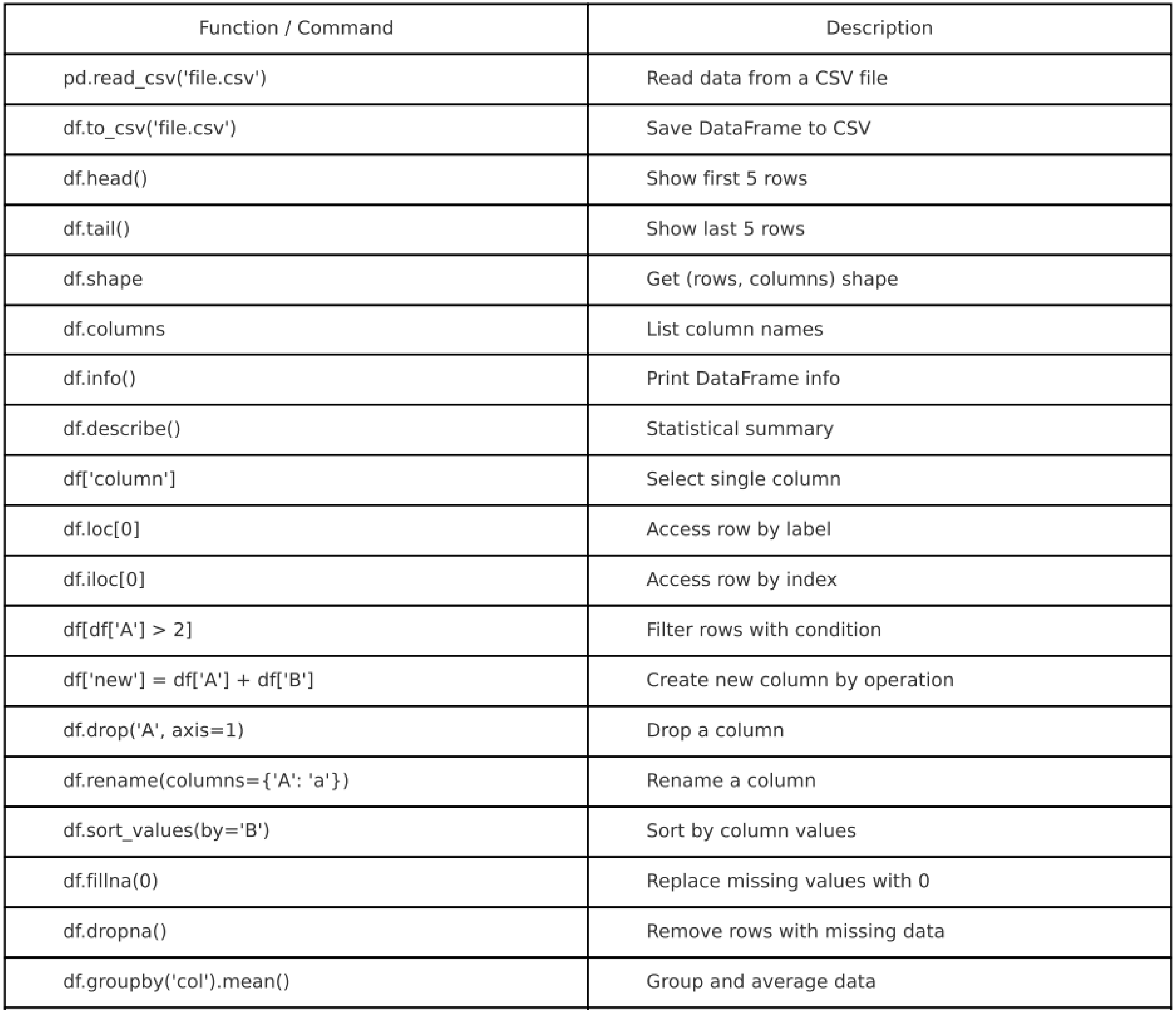
pandas is a powerful **data analysis and manipulation** library for Python. It provides **data structures** like Series and DataFrame that make **handling structured data** fast and easy.



📦 Key Features of Pandas

- **Data Structures**:
  - **Series**: One-dimensional labeled array (like a column in Excel).
  - **DataFrame**: Two-dimensional labeled data structure (like a spreadsheet or SQL table).
- **Easy Handling of Missing Data**.
- **Powerful Grouping and Aggregation** functions.
- **Data Filtering, Selection, and Slicing** capabilities.
- **Merging, Joining, and Concatenating** datasets.
- **Time Series Functionality**.
- **Reading/Writing** from CSV, Excel, SQL, JSON, etc.

Useful Pandas function in Python



Function / Command	Description
pd.read_csv('file.csv')	Read data from a CSV file
df.to_csv('file.csv')	Save DataFrame to CSV
df.head()	Show first 5 rows
df.tail()	Show last 5 rows
df.shape	Get (rows, columns) shape
df.columns	List column names
df.info()	Print DataFrame info
df.describe()	Statistical summary
df['column']	Select single column
df.loc[0]	Access row by label
df.iloc[0]	Access row by index
df[df['A'] > 2]	Filter rows with condition
df['new'] = df['A'] + df['B']	Create new column by operation
df.drop('A', axis=1)	Drop a column
df.rename(columns={'A': 'a'})	Rename a column
df.sort_values(by='B')	Sort by column values
df.fillna(0)	Replace missing values with 0
df.dropna()	Remove rows with missing data
df.groupby('col').mean()	Group and average data

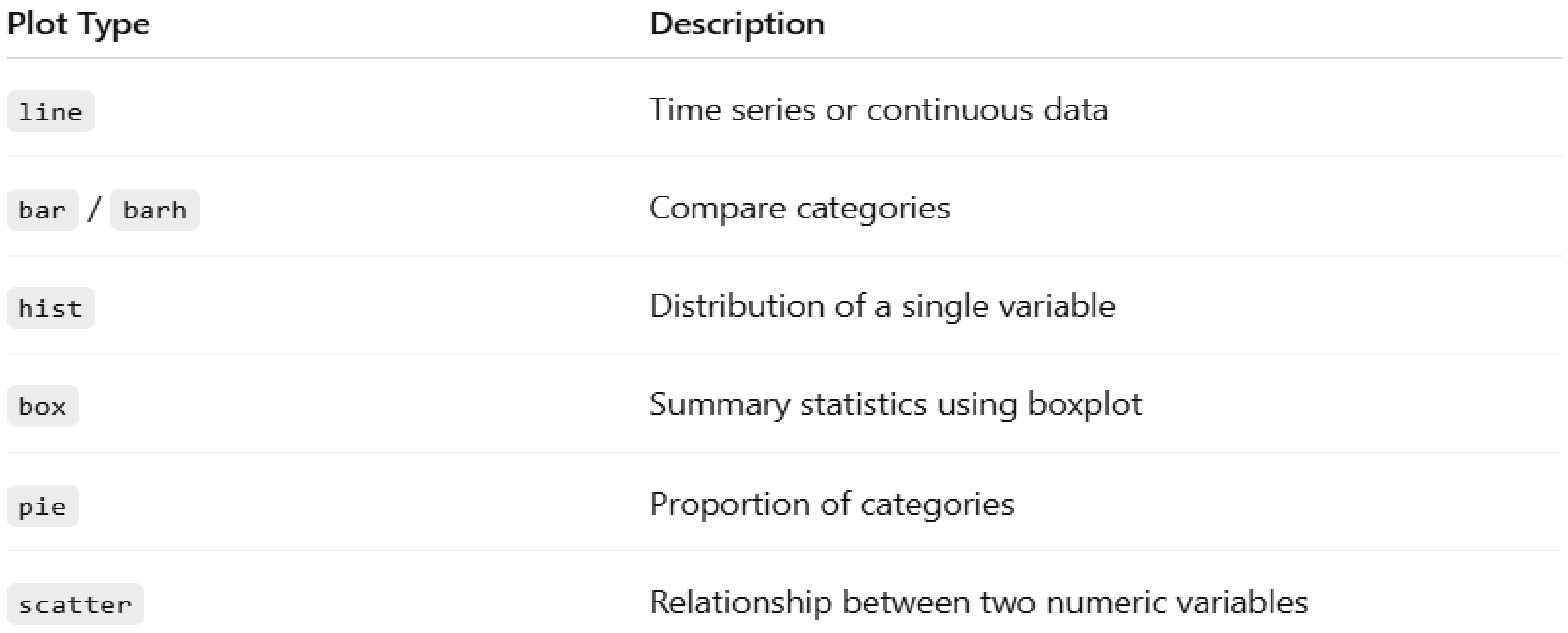
Data Visualization

Data Visualization is the graphical representation of data to help understand trends, patterns, and outliers. It is essential for **data analysis, storytelling, and decision-making**.

In Python, popular libraries for data visualization include:

- **Matplotlib** – Basic plotting library
- **Seaborn** – Statistical data visualization (built on Matplotlib)
- **Pandas** – Offers basic plotting through .plot() function

Types of Visualizations in Pandas



Plot Type	Description
`line`	Time series or continuous data
`bar` / `barh`	Compare categories
`hist`	Distribution of a single variable
`box`	Summary statistics using boxplot
`pie`	Proportion of categories
`scatter`	Relationship between two numeric variables

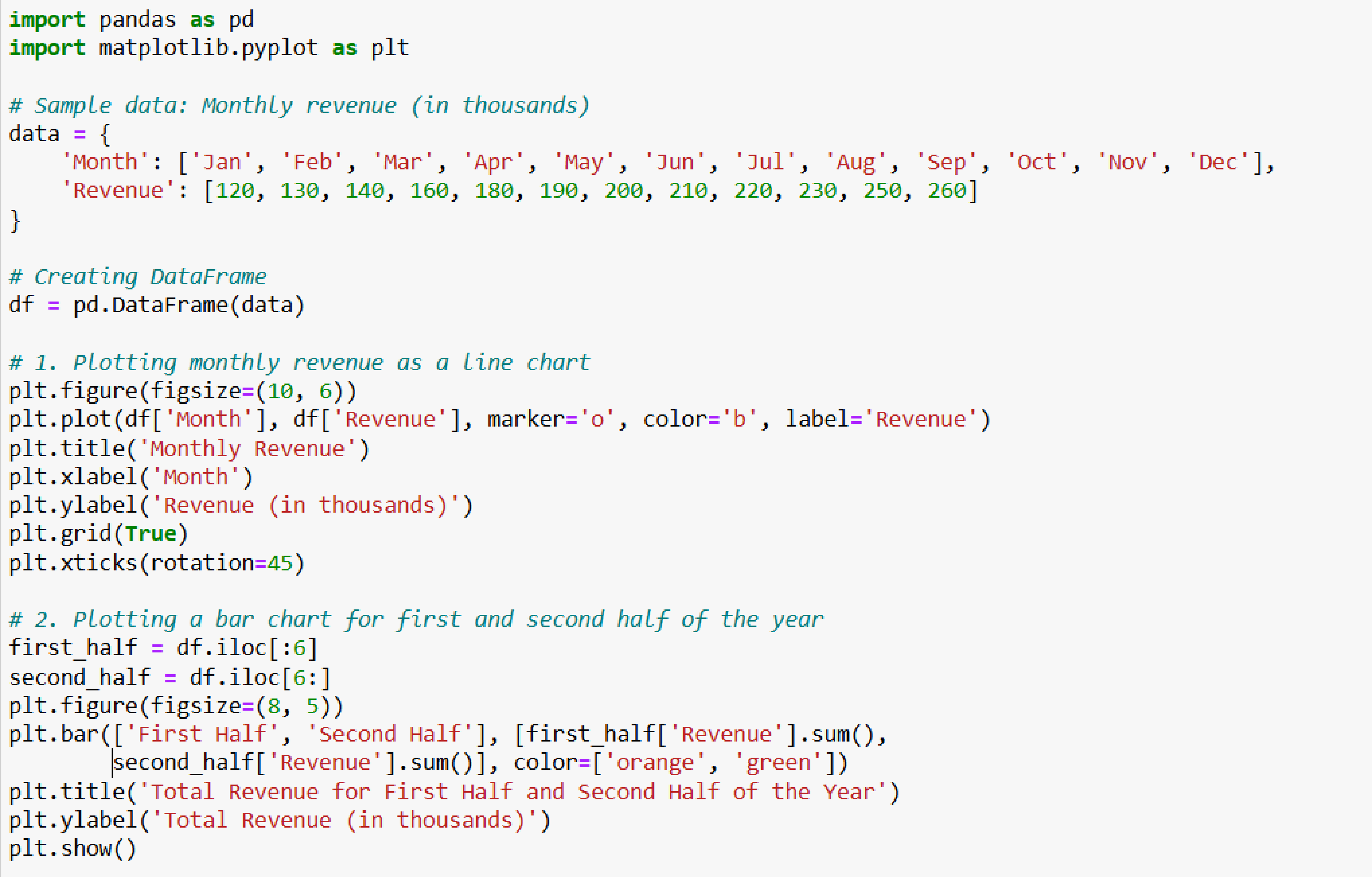
12. You have a dataset that records monthly revenue of a company over the past year. The dataset includes the columns `Month` and `Revenue`. Using **Pandas** and **Matplotlib**, write a program to: 10M

Month	Revenue
Jan	120
Feb	130
Mar	140
Apr	160
May	180
Jun	190
Jul	200
Aug	210
Sep	220
Oct	230
Nov	250
Dec	260

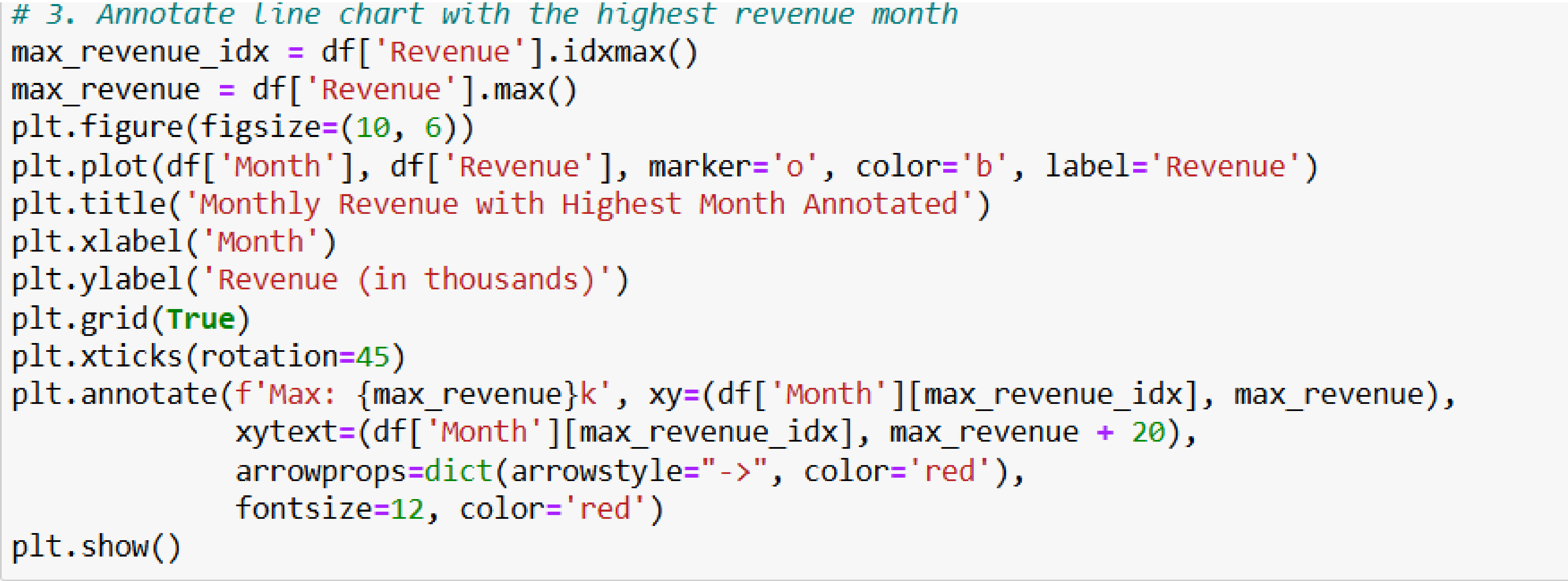
a) Plot the **monthly revenue** as a line chart.

b) Plot a **bar chart** to show the total revenue for the first half and the second half of the year.

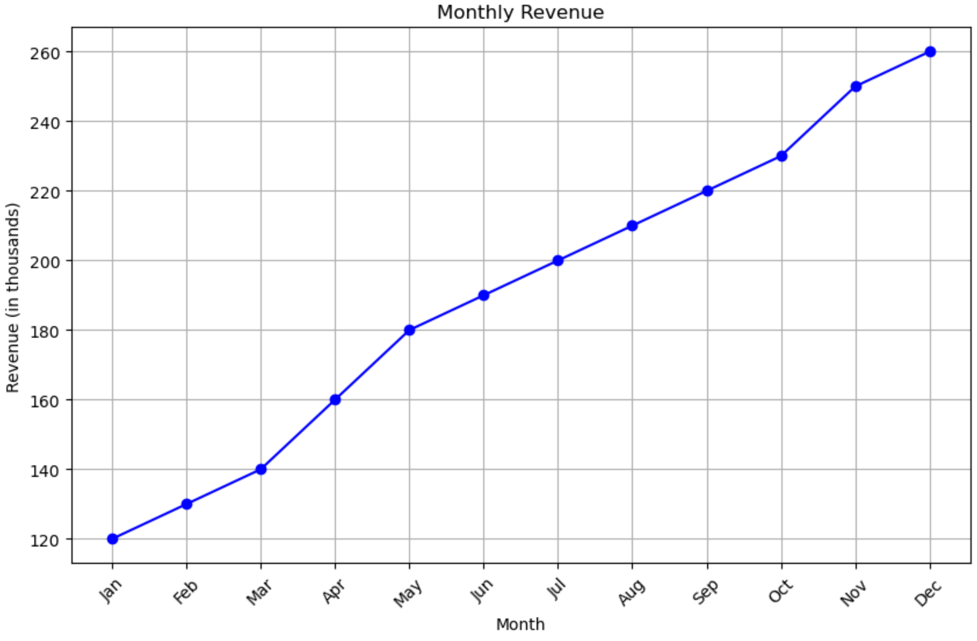
c) Annotate the line chart with the highest revenue month and value.



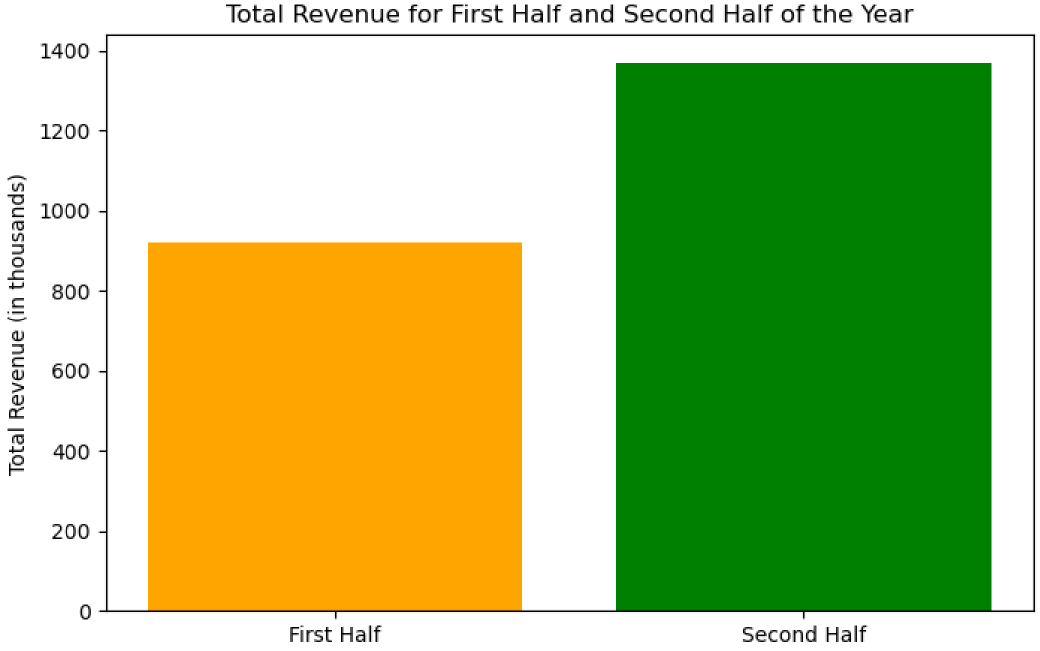
```python
import pandas as pd
import matplotlib.pyplot as plt

# Sample data: Monthly revenue (in thousands)
data = {
    'Month': ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'],
    'Revenue': [120, 130, 140, 160, 180, 190, 200, 210, 220, 230, 250, 260]
}

# Creating DataFrame
df = pd.DataFrame(data)

# 1. Plotting monthly revenue as a line chart
plt.figure(figsize=(10, 6))
plt.plot(df['Month'], df['Revenue'], marker='o', color='b', label='Revenue')
plt.title('Monthly Revenue')
plt.xlabel('Month')
plt.ylabel('Revenue (in thousands)')
plt.grid(True)
plt.xticks(rotation=45)

# 2. Plotting a bar chart for first and second half of the year
first_half = df.iloc[:6]
second_half = df.iloc[6:]
plt.figure(figsize=(8, 5))
plt.bar(['First Half', 'Second Half'], [first_half['Revenue'].sum(),
        second_half['Revenue'].sum()], color=['orange', 'green'])
plt.title('Total Revenue for First Half and Second Half of the Year')
plt.ylabel('Total Revenue (in thousands)')
plt.show()
```



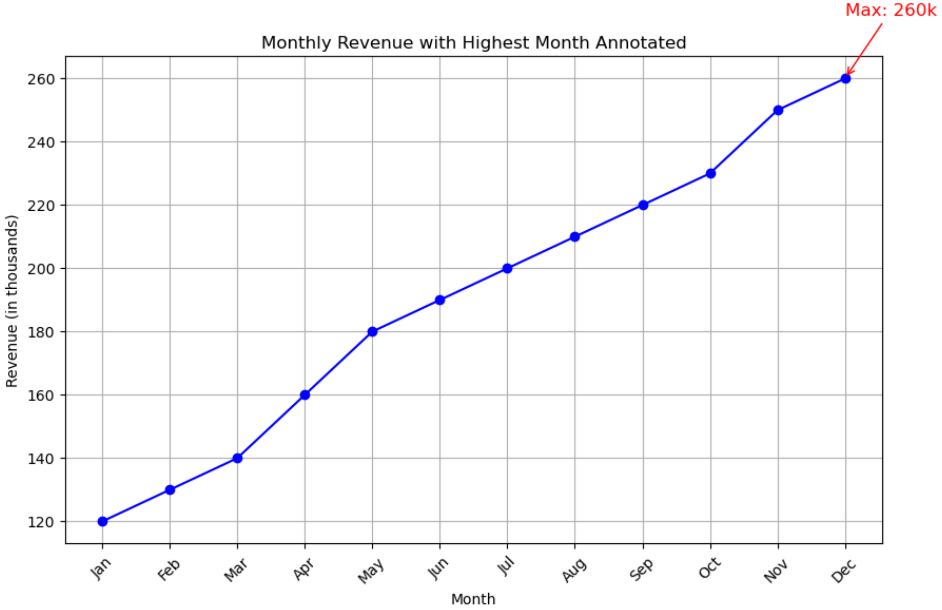
```
# 3. Annotate line chart with the highest revenue month
max_revenue_idx = df['Revenue'].idxmax()
max_revenue = df['Revenue'].max()
plt.figure(figsize=(10, 6))
plt.plot(df['Month'], df['Revenue'], marker='o', color='b', label='Revenue')
plt.title('Monthly Revenue with Highest Month Annotated')
plt.xlabel('Month')
plt.ylabel('Revenue (in thousands)')
plt.grid(True)
plt.xticks(rotation=45)
plt.annotate(f'Max: {max_revenue}k', xy=(df['Month'][max_revenue_idx], max_revenue),
             xytext=(df['Month'][max_revenue_idx], max_revenue + 20),
             arrowprops=dict(arrowstyle="->", color='red'),
             fontsize=12, color='red')
plt.show()
```



a) the **monthly revenue** as a line chart



b) a **bar chart** to show the total revenue for the first half and the second half of the year



c) the line chart with the highest revenue month and value.