**Q75: What are the different types of Joins? Demonstrate the types of Joins by creating two dictionaries containing Employee data.**

**Introduction to Joins:**

Joins are used in databases and data analysis to combine rows from two or more data structures based on a common key. In Python, particularly using **dictionaries**, we simulate joins manually (or use pandas for real join operations).

**Types of Joins:**

There are **four main types of joins**:

1. **Inner Join** – Returns only the matching keys from both dictionaries.
2. **Left Join** – Returns all keys from the left dictionary and matches from the right dictionary (if available).
3. **Right Join** – Returns all keys from the right dictionary and matches from the left dictionary (if available).
4. **Outer Join** – Returns all keys from both dictionaries. Missing values are filled with None or a placeholder.

Absolutely! Here’s a very simple and beginner-friendly example to help you understand different types of joins using Python dictionaries.

**Step 1: Create Two Simple Dictionaries**

Let’s say we have two small lists:

* **Employee Names** (with Employee ID)
* **Employee Departments** (with Employee ID)

python

*# Employee names*

employees = {

1: "Alice",

2: "Bob",

3: "Charlie"

}

*# Employee departments*

departments = {

2: "HR",

3: "Finance",

4: "Marketing"

}

**Step 2: Different Types of Joins**

**1. Inner Join**

**Keep only IDs that are in both dictionaries.**

python

inner\_join = {}

**for** emp\_id **in** employees:

**if** emp\_id **in** departments:

inner\_join[emp\_id] = (employees[emp\_id], departments[emp\_id])

**print**("Inner Join:", inner\_join)

**Output:**  
{2: ('Bob', 'HR'), 3: ('Charlie', 'Finance')}

**2. Left Join**

**Keep all employees, add department if it exists.**

python

left\_join = {}

**for** emp\_id **in** employees:

left\_join[emp\_id] = (employees[emp\_id], departments.get(emp\_id))

**print**("Left Join:", left\_join)

**Output:**  
{1: ('Alice', None), 2: ('Bob', 'HR'), 3: ('Charlie', 'Finance')}

**3. Right Join**

**Keep all departments, add employee name if it exists.**

python

right\_join = {}

**for** emp\_id **in** departments:

right\_join[emp\_id] = (employees.get(emp\_id), departments[emp\_id])

**print**("Right Join:", right\_join)

**Output:**  
{2: ('Bob', 'HR'), 3: ('Charlie', 'Finance'), 4: (None, 'Marketing')}

**4. Full (Outer) Join**

**Keep all IDs from both, add None where missing.**

python

full\_join = {}

all\_ids = set(employees) | set(departments)

**for** emp\_id **in** all\_ids:

full\_join[emp\_id] = (employees.get(emp\_id), departments.get(emp\_id))

**print**("Full Join:", full\_join)

**Output:**  
{1: ('Alice', None), 2: ('Bob', 'HR'), 3: ('Charlie', 'Finance'), 4: (None, 'Marketing')}

**Summary Table**

| **Type** | **What it does** | **Example Output** |
| --- | --- | --- |
| Inner Join | Only IDs in both | {2: ('Bob', 'HR'), 3: ('Charlie', 'Finance')} |
| Left Join | All employees, add dept if exists | {1: ('Alice', None), 2: ('Bob', 'HR'), 3: ('Charlie', 'Finance')} |
| Right Join | All departments, add name if exists | {2: ('Bob', 'HR'), 3: ('Charlie', 'Finance'), 4: (None, 'Marketing')} |
| Full Join | All IDs, add None where missing | {1: ('Alice', None), 2: ('Bob', 'HR'), 3: ('Charlie', 'Finance'), 4: (None, 'Marketing')} |

**Tip:**

* None means there was no matching data in that dictionary.
* This example is simple and easy to try out. Just copy and run each code block in Python!

Let me know if you want an even simpler example or more explanation!

**Example Using Two Dictionaries:**

Let’s consider the following data:

python

CopyEdit

# Dictionary 1: Employee ID and Name

emp\_names = {

101: 'Alice',

102: 'Bob',

103: 'Charlie',

104: 'David'

}

# Dictionary 2: Employee ID and Department

emp\_depts = {

102: 'HR',

103: 'Finance',

105: 'IT'

}

**1. Inner Join**

python

CopyEdit

print("Inner Join:")

for emp\_id in emp\_names.keys() & emp\_depts.keys():

print(emp\_id, emp\_names[emp\_id], emp\_depts[emp\_id])

**Output:**

sql

CopyEdit

Inner Join:

102 Bob HR

103 Charlie Finance

✅ Only IDs **102** and **103** exist in both dictionaries.

**2. Left Join**

python

CopyEdit

print("\nLeft Join:")

for emp\_id in emp\_names:

dept = emp\_depts.get(emp\_id, None)

print(emp\_id, emp\_names[emp\_id], dept)

**Output:**

sql

CopyEdit

Left Join:

101 Alice None

102 Bob HR

103 Charlie Finance

104 David None

✅ All employees from emp\_names are included. If department not found, None is shown.

**3. Right Join**

python

CopyEdit

print("\nRight Join:")

for emp\_id in emp\_depts:

name = emp\_names.get(emp\_id, None)

print(emp\_id, name, emp\_depts[emp\_id])

**Output:**

sql

CopyEdit

Right Join:

102 Bob HR

103 Charlie Finance

105 None IT

✅ All departments are listed. Name is None if employee not found in emp\_names.

**4. Outer Join**

python

CopyEdit

print("\nOuter Join:")

all\_ids = emp\_names.keys() | emp\_depts.keys()

for emp\_id in all\_ids:

name = emp\_names.get(emp\_id, None)

dept = emp\_depts.get(emp\_id, None)

print(emp\_id, name, dept)

**Output:**

sql

CopyEdit

Outer Join:

101 Alice None

102 Bob HR

103 Charlie Finance

104 David None

105 None IT

✅ All keys from both dictionaries are included.

**Conclusion:**

Joins are essential in combining related data. In Python, they can be implemented using set operations on dictionary keys. Understanding different join types helps in effective data merging and analysis.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Q76. How do you achieve the following in Pandas i) Renaming columns, ii) dropping rows & coulmns iii) Adding new column iv) Sorting data frame. Illustrate with examples**

**How to Perform Common Operations in Pandas**

The Python **pandas** library is widely used for data manipulation and analysis. Below, we explain how to perform the following operations in pandas with examples:

1. **Renaming Columns**
2. **Dropping Rows and Columns**
3. **Adding a New Column**
4. **Sorting a DataFrame**

We will use a simple DataFrame for illustration:

python

CopyEdit

import pandas as pd

# Sample DataFrame

data = {

'EmpID': [101, 102, 103],

'Name': ['Alice', 'Bob', 'Charlie'],

'Salary': [50000, 60000, 55000],

'Dept': ['HR', 'IT', 'Finance']

}

df = pd.DataFrame(data)

print("Original DataFrame:")

print(df)

**1. Renaming Columns**

**Explanation:**

* You can rename one or more columns using the rename() function with the columns parameter.
* Use inplace=True to apply changes directly to the DataFrame.

**Example:**

python

CopyEdit

# Rename 'EmpID' to 'Employee\_ID' and 'Dept' to 'Department'

df.rename(columns={'EmpID': 'Employee\_ID', 'Dept': 'Department'}, inplace=True)

print("\nAfter Renaming Columns:")

print(df)

**Output:**

Employee\_ID Name Salary Department

0 101 Alice 50000 HR

1 102 Bob 60000 IT

2 103 Charlie 55000 Finance

**2. Dropping Rows and Columns**

**a) Dropping Rows**

**Explanation:**

* Use drop() method with axis=0 to drop rows.
* You can drop by index values.

**Example:**

python

CopyEdit

# Drop the row with index 1 (i.e., Bob)

df\_dropped\_row = df.drop(index=1)

print("\nAfter Dropping Row (index 1):")

print(df\_dropped\_row)

**b) Dropping Columns**

**Explanation:**

* Use drop() method with axis=1 to drop columns.

**Example:**

python

CopyEdit

# Drop the 'Salary' column

df\_dropped\_column = df.drop(columns=['Salary'])

print("\nAfter Dropping 'Salary' Column:")

print(df\_dropped\_column)

**3. Adding a New Column**

**Explanation:**

* You can add a new column directly by assigning a list or computed values.

**Example:**

python

CopyEdit

# Add a new column 'Bonus' which is 10% of Salary

df['Bonus'] = df['Salary'] \* 0.10

print("\nAfter Adding New Column 'Bonus':")

print(df)

**Output:**

yaml

CopyEdit

Employee\_ID Name Salary Department Bonus

0 101 Alice 50000 HR 5000.0

1 102 Bob 60000 IT 6000.0

2 103 Charlie 55000 Finance 5500.0

**4. Sorting DataFrame**

**a) Sorting by One Column**

**Explanation:**

* Use sort\_values() method to sort by one or more columns.

**Example:**

python

CopyEdit

# Sort by Salary in ascending order

df\_sorted = df.sort\_values(by='Salary')

print("\nSorted by Salary (Ascending):")

print(df\_sorted)

**b) Sorting in Descending Order**

python

CopyEdit

# Sort by Bonus in descending order

df\_sorted\_desc = df.sort\_values(by='Bonus', ascending=False)

print("\nSorted by Bonus (Descending):")

print(df\_sorted\_desc)

**Summary Table**

| **Operation** | **Function Used** | **Key Argument(s)** |
| --- | --- | --- |
| Renaming Columns | rename() | columns={} |
| Dropping Rows | drop() | index=, axis=0 |
| Dropping Columns | drop() | columns=, axis=1 |
| Adding New Column | Assignment | df['NewCol'] = value |
| Sorting DataFrame | sort\_values() | by=, ascending= |

**Conclusion**

These operations are essential for cleaning, transforming, and analyzing data in pandas. Understanding how to rename, drop, add, and sort columns or rows allows you to manipulate DataFrames effectively, making it easier to prepare data for further analysis or machine learning tasks.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Bottom of Form

**Q77: Describe the steps to visualize a DataFrame using pandas built-in plotting capabilities. How does it internally use matplotlib? Provide a use case showing line and bar plots.**

**Introduction:**

Pandas has built-in **plotting capabilities** that make it easy to generate common charts directly from a DataFrame or Series. These are built on top of **matplotlib**, a powerful 2D plotting library in Python.

When you use .plot() in pandas, it internally calls **matplotlib’s plotting functions** to create the graph, which means you get both simplicity (from pandas) and power (from matplotlib).

**Steps to Visualize a DataFrame using Pandas:**

1. **Import Required Libraries**
2. **Create or Load Data into a DataFrame**
3. **Use the .plot() method to generate a graph**
4. **Customize the plot (title, labels, legend)**
5. **Display the plot using plt.show() (from matplotlib)**

**How pandas uses matplotlib internally:**

* pandas.DataFrame.plot() and pandas.Series.plot() use matplotlib.pyplot under the hood.
* By default, when you call df.plot(), pandas automatically imports and uses matplotlib.pyplot to render the plots.
* This allows pandas to support various plot types like: 'line', 'bar', 'hist', 'box', 'area', 'scatter', etc.

**Example: Visualizing Data Using Line and Bar Plots**

**Step 1: Import Libraries**

python

CopyEdit

import pandas as pd

import matplotlib.pyplot as plt

**Step 2: Create Sample Data**

python

CopyEdit

# Create a DataFrame

data = {

'Month': ['Jan', 'Feb', 'Mar', 'Apr', 'May'],

'Sales': [25000, 27000, 30000, 22000, 32000],

'Expenses': [18000, 20000, 21000, 19000, 23000]

}

df = pd.DataFrame(data)

**Step 3: Set 'Month' as Index (for better plotting)**

python

CopyEdit

df.set\_index('Month', inplace=True)

**A) Line Plot**

**Use Case: Visualize Sales and Expenses Trend over Months**

python

CopyEdit

# Line plot

df.plot(kind='line', marker='o', title='Monthly Sales and Expenses')

plt.xlabel('Month')

plt.ylabel('Amount (in Rs.)')

plt.grid(True)

plt.tight\_layout()

plt.show()

**Explanation:**

* kind='line': Specifies a line chart
* marker='o': Draws circles at each data point
* plt.show(): Displays the plot

**B) Bar Plot**

**Use Case: Compare Sales and Expenses for each Month**

python

CopyEdit

# Bar plot

df.plot(kind='bar', title='Sales vs Expenses')

plt.xlabel('Month')

plt.ylabel('Amount (in Rs.)')

plt.tight\_layout()

plt.show()

**Explanation:**

* kind='bar': Generates vertical bars for each column
* Each bar group corresponds to a month, comparing Sales and Expenses side by side

**Output (Expected):**

Two charts:

1. A **line chart** with two lines (one for Sales, one for Expenses) showing trends across months.
2. A **bar chart** with grouped bars per month.

**Summary Table of Plot Types in Pandas:**

| **Plot Type** | **kind Argument** | **Description** |
| --- | --- | --- |
| Line | 'line' | Default; good for time series |
| Bar | 'bar' | Vertical bar chart |
| Horizontal Bar | 'barh' | Horizontal bar chart |
| Histogram | 'hist' | Frequency distribution |
| Box | 'box' | Box-and-whisker plot |
| Area | 'area' | Filled area plot |
| Scatter | 'scatter' | Scatter plot (needs x and y values) |
| Pie | 'pie' | Pie chart (only for Series) |

**Conclusion:**

Pandas makes it easy to create visualizations with minimal code. Internally, it uses matplotlib to handle plotting. Using line and bar plots, users can visualize trends, comparisons, and patterns in data effectively. These visualizations are crucial for understanding business metrics, scientific data, or any structured dataset.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Q78: Define Data Visualization. Discuss the purpose of data visualization in data analysis and explain any three commonly used types of graphs (such as bar chart, line graph, scatter plot) with suitable examples.**

**Definition: What is Data Visualization?**

**Data Visualization** is the graphical representation of information and data using visual elements like charts, graphs, and maps. It helps transform complex data into an easily understandable format, allowing users to identify patterns, trends, and insights quickly.

**Purpose of Data Visualization in Data Analysis:**

1. ✅ **Simplifies Complex Data**: Helps convert raw data into a visual format that is easier to interpret.
2. ✅ **Reveals Patterns and Trends**: Makes it easier to observe trends, outliers, or anomalies.
3. ✅ **Improves Decision-Making**: Stakeholders can make informed decisions based on visual insights.
4. ✅ **Enhances Communication**: Visualization helps communicate findings more effectively to technical and non-technical audiences.
5. ✅ **Aids in Exploratory Data Analysis (EDA)**: Helps analysts explore datasets before building models.

**Commonly Used Graph Types in Data Visualization:**

Below are three widely used graphs in Python using the matplotlib and pandas libraries.

**1. Bar Chart**

**Definition:**

A **bar chart** displays data with rectangular bars. The length of each bar is proportional to the value it represents.

**Use Case:**

Compare **sales** across different months.

**Example:**

python

CopyEdit

import pandas as pd

import matplotlib.pyplot as plt

# Data

months = ['Jan', 'Feb', 'Mar', 'Apr']

sales = [25000, 27000, 30000, 22000]

# Plot

plt.bar(months, sales, color='skyblue')

plt.title('Monthly Sales')

plt.xlabel('Month')

plt.ylabel('Sales (in Rs.)')

plt.show()

**Output:**

A vertical bar chart comparing sales in January to April.

**2. Line Graph**

**Definition:**

A **line graph** connects data points with lines and is used to show changes or trends over time.

**Use Case:**

Show the **trend of website visitors** over a week.

**Example:**

python

CopyEdit

days = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri']

visitors = [200, 240, 250, 300, 280]

plt.plot(days, visitors, marker='o', linestyle='-', color='green')

plt.title('Daily Website Visitors')

plt.xlabel('Day')

plt.ylabel('Number of Visitors')

plt.grid(True)

plt.show()

**Output:**

A line chart showing how visitors changed each day from Monday to Friday.

**3. Scatter Plot**

**Definition:**

A **scatter plot** shows individual data points based on two variables. It's useful to observe relationships, trends, or clusters.

**Use Case:**

Visualize the **relationship between study hours and exam scores**.

**Example:**

python

CopyEdit

study\_hours = [1, 2, 3, 4, 5]

exam\_scores = [55, 60, 65, 75, 85]

plt.scatter(study\_hours, exam\_scores, color='red')

plt.title('Study Hours vs Exam Score')

plt.xlabel('Study Hours')

plt.ylabel('Exam Score')

plt.show()

**Output:**

A scatter plot indicating that more study hours tend to lead to higher scores.

**Comparison Table:**

| **Chart Type** | **Purpose** | **Best For** |
| --- | --- | --- |
| Bar Chart | Compare categories | Sales, Counts, Category Data |
| Line Graph | Show trends over time | Time series, Growth patterns |
| Scatter Plot | Show correlation between variables | Relationship between values |

**Conclusion:**

Data visualization is a critical part of data analysis that allows users to understand and explore data effectively. Bar charts, line graphs, and scatter plots are essential tools for visualizing various types of data. These graphs not only enhance understanding but also enable better decision-making.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Q79. Write a pandas program to read a CSV file and filter data based on multiple conditions (e.g., Salesman = "Ravish" and Qty > 5, city Name like ). Display only selected columns.**

Here is a **simple and clear Python program** using **pandas** that reads a CSV file, filters data based on multiple conditions, and displays only selected columns.

**✅ Question**

Write a pandas program to read a CSV file and filter data where:

* Salesman = "Ravish"
* Qty > 5
* City name **starts with** a specific letter (e.g., 'B')  
  Display only selected columns: Salesman, Qty, and City.

**✅ Sample CSV Content (sales.csv)**

Suppose sales.csv has the following data:

csv

CopyEdit

Salesman,Qty,City,Amount

Ravish,10,Bangalore,5000

Amit,3,Mumbai,2000

Ravish,4,Bhopal,2500

Ravish,8,Bangalore,6000

Sunil,6,Delhi,3000

**✅ Python Program**

python

CopyEdit

import pandas as pd

# Step 1: Read the CSV file

df = pd.read\_csv('sales.csv')

# Step 2: Apply multiple filters

filtered\_df = df[

(df['Salesman'] == 'Ravish') &

(df['Qty'] > 5) &

(df['City'].str.startswith('B'))

]

# Step 3: Display selected columns

result = filtered\_df[['Salesman', 'Qty', 'City']]

print(result)

**✅ Expected Output**

nginx

CopyEdit

Salesman Qty City

0 Ravish 10 Bangalore

3 Ravish 8 Bangalore

**✅ Explanation:**

* pd.read\_csv('sales.csv'): Loads the data from the CSV file.
* (df['Salesman'] == 'Ravish'): Filters rows where Salesman is Ravish.
* (df['Qty'] > 5): Filters rows where Qty is more than 5.
* df['City'].str.startswith('B'): Checks if city name starts with letter 'B'.
* [['Salesman', 'Qty', 'City']]: Displays only selected columns.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Bottom of Form

**Q80**. **What is Matplotlib? Create a Python dictionary containing sales details of salesmen, convert it into a Pandas DataFrame, group the data by total sales per salesman, and then draw a bar graph and a pie chart to visualize the total sales of each salesman.**

Q80: What is Matplotlib?

✅ Definition:

Matplotlib is a popular data visualization library in Python. It provides a wide range of tools for creating static, interactive, and animated visualizations such as:

* Line charts
* Bar graphs
* Pie charts
* Histograms
* Scatter plots, etc.

The core module in Matplotlib is pyplot, which offers a simple interface similar to MATLAB for plotting.

✅ Key Features of Matplotlib:

* Works well with NumPy and Pandas.
* Allows for complete control over plot elements (title, labels, colors, etc.).
* Supports exporting plots in high-quality formats like PNG, PDF, and SVG.

✅ Use Case: Sales Data Visualization

Let’s walk through a complete example as per the question:

🧾 Task Summary:

1. Create a Python dictionary with sales details.
2. Convert it to a Pandas DataFrame.
3. Group the data by salesman to get total sales.
4. Use Matplotlib to:
   * Draw a bar graph
   * Draw a pie chart of total sales per salesman.

✅ Step-by-Step Python Code

python

CopyEdit

import pandas as pd

import matplotlib.pyplot as plt

# Step 1: Create a dictionary with sales data

sales\_data = {

'Salesman': ['Ravi', 'Anil', 'Ravi', 'Anil', 'Sunil', 'Ravi', 'Sunil'],

'Amount': [5000, 4000, 3000, 3500, 2500, 4500, 3000]

}

# Step 2: Convert dictionary into a DataFrame

df = pd.DataFrame(sales\_data)

print("Original Data:")

print(df)

# Step 3: Group by Salesman and sum the Amount

grouped\_sales = df.groupby('Salesman')['Amount'].sum().reset\_index()

print("\nTotal Sales Per Salesman:")

print(grouped\_sales)

# Step 4: Plot Bar Graph

plt.figure(figsize=(8, 5))

plt.bar(grouped\_sales['Salesman'], grouped\_sales['Amount'], color='skyblue')

plt.title('Total Sales per Salesman - Bar Chart')

plt.xlabel('Salesman')

plt.ylabel('Total Sales (Rs.)')

plt.grid(True)

plt.tight\_layout()

plt.show()

# Step 5: Plot Pie Chart

plt.figure(figsize=(6, 6))

plt.pie(grouped\_sales['Amount'], labels=grouped\_sales['Salesman'], autopct='%1.1f%%', startangle=140)

plt.title('Total Sales per Salesman - Pie Chart')

plt.axis('equal') # Equal aspect ratio ensures the pie chart is circular.

plt.show()

✅ Output Explanation:

📄 Grouped Data Output:

yaml

CopyEdit

Salesman Amount

0 Anil 7500

1 Ravi 12500

2 Sunil 5500

📊 Bar Graph:

* X-axis: Salesmen (Ravi, Anil, Sunil)
* Y-axis: Total sales amount
* Each bar shows how much each salesman sold in total.

🥧 Pie Chart:

* Each slice represents the percentage share of sales for each salesman.

✅ Why This is Useful:

* Bar Graph: Makes it easy to compare total sales between salesmen.
* Pie Chart: Helps in understanding the contribution percentage of each salesman to the total.

✅ Conclusion:

Matplotlib is a powerful visualization tool in Python that works seamlessly with Pandas. By grouping data and using bar() and pie() plots, you can create meaningful visual summaries of business metrics like sales performance.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***