
🧠 10 Theoretical Pandas Questions and Answers

1. What is Pandas in Python?

Answer:
Pandas is an open-source Python library used for **data manipulation and analysis**. It provides powerful data structures like **Series** (1D) and **DataFrame** (2D) for handling structured data efficiently.

---

2. What are the main data structures in Pandas?

Answer:

- **Series:** One-dimensional labeled array.
- **DataFrame:** Two-dimensional labeled data structure (like a table).
- **Panel:** (Deprecated) 3D data structure.

---

3. What is the difference between a Series and a DataFrame?

Answer:

- **Series:** 1D array with labels (like a single column).
- **DataFrame:** 2D structure made up of multiple Series objects (like an Excel sheet).

---

4. How does Pandas handle missing data?

Answer:
Using:

- isnull() or notnull() → to detect missing values.
- fillna() → to replace missing values.
- dropna() → to remove missing rows or columns.

---

5. What are DataFrame indexes in Pandas?

Answer:
Indexes are **labels** used to identify rows and columns uniquely. They make data selection, alignment, and merging more efficient.

---

6. What is the difference between loc[] and iloc[]?

Answer:

- loc[]: Label-based indexing (uses row/column names).
- iloc[]: Integer-based indexing (uses numerical positions).

---

7. How do you merge and join datasets in Pandas?

Answer:

- merge() → SQL-style joins (inner, outer, left, right).
- join() → Combines columns with the same index.
- concat() → Combines along rows or columns.

---

8. What are vectorized operations in Pandas?

Answer:
Vectorized operations allow applying mathematical or logical operations directly on entire Series/DataFrames without using explicit loops.

---

9. What is the purpose of the groupby() function?

Answer:
It is used to **split** data into groups based on some criteria, **apply** functions, and **combine** the results — useful for data aggregation and summarization.

---

10. What is the difference between apply(), map(), and applymap()?

Answer:

- map() → Works on **Series** (element-wise).
- apply() → Works on **DataFrame rows or columns**.
- applymap() → Works **element-wise on entire DataFrame**.

💻 **10 Practical Pandas Questions and Answers**

1. Create a DataFrame from a dictionary.

```
import pandas as pd


data = {'Name': ['Gaurav', 'Ravi', 'Asha'],
    'Age': [21, 22, 20],
    'City': ['Ambikapur', 'Raipur', 'Bilaspur']}


df = pd.DataFrame(data)
print(df)
```

---

2. Read a CSV file into a DataFrame.

```
df = pd.read_csv('data.csv')
print(df.head())  # Shows first 5 rows
```

---

3. Select a single column and multiple columns.

```
print(df['Name'])          # Single column
print(df[['Name', 'Age']])  # Multiple columns
```

---

4. Select specific rows using loc and iloc.

```
print(df.loc[0])      # Row by label
print(df.iloc[1:3])   # Rows by position
```

---

5. Filter rows based on a condition.

```
print(df[df['Age'] > 21])
```

---

6. Add a new column to the DataFrame.

```
df['Score'] = [85, 90, 88]
print(df)
```

---

7. Drop a column from the DataFrame.

```
df = df.drop('City', axis=1)
print(df)
```

---

8. Find the mean, min, and max of numeric columns.

```
print(df['Age'].mean())
print(df['Age'].min())
print(df['Age'].max())
```

---

9. Group data and find average per group.

```
data = {'Department': ['IT', 'IT', 'HR', 'HR'],
        'Salary': [40000, 50000, 45000, 55000]}
df = pd.DataFrame(data)

print(df.groupby('Department')['Salary'].mean())
```

Output:

Department

HR 50000.0

IT 45000.0

Name: Salary, dtype: float64

---

10. Merge two DataFrames on a common column.

df1 = pd.DataFrame({'ID': [1, 2, 3], 'Name': ['A', 'B', 'C']})

df2 = pd.DataFrame({'ID': [1, 2, 4], 'Marks': [85, 90, 75]})

merged = pd.merge(df1, df2, on='ID', how='inner')

print(merged)

Output:

	ID	Name	Marks
0	1	A	85
1	2	B	90
