1. Introduction to Pandas

- **Pandas** is a Python library for data manipulation and analysis.
- It provides two main data structures:
    - **Series** → 1D labeled array (vertical)
    - **DataFrame** → 2D table made up of multiple Series

---

2. Importing Pandas

```
import pandas as pd
import numpy as np
```

---

3. Pandas Series

Creating a Series

```
l1 = [10, 20, 30]
l2 = ['a', 'b', 'c']
print(pd.Series(l1, index=l2))
```

📘 **Key Points:**

- A **Series** can hold any data type (int, float, string, etc.)
- It is **1-dimensional** and **labeled vertically**
- Custom **index** can be assigned

---

4. Pandas DataFrame

Creating a DataFrame

```
data = {
    'Name': ['Ali', 'Qasim'],
    'Age': [11, 12]
}
df = pd.DataFrame(data)
print(df)
```

✅ A **DataFrame** is made up of multiple Series combined together.

---

5. Selecting and Indexing Columns

Selecting Columns

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

Adding a New Column

```
df['Hi'] = [1, 2]
```

Removing a Column

```
df.drop('Name', axis=1)                    # Removes temporarily
df.drop('Name', axis=1, inplace=True)      # Removes permanently
```

📘 **Axis Meaning:**

- `axis = 0` → rows (horizontal)
- `axis = 1` → columns (vertical)

---

6. Selecting Rows

Using loc (label-based)

```
df.loc[0]                # Select first row
df.loc[[0, 1]]           # Select multiple rows
df.loc[[0, 1], ['Name', 'Age']]   # Select specific rows & columns
```

Using iloc (index-based)

```
df.iloc[0:2, 0:2]        # Rows 0–1, Columns 0–1
```

---

7. Conditional Selections

```
df[df['Age'] > 19]
df[(df['Age'] > 19) & (df['City'] == 'Paris')]
```

📘 **Tip:**
Always use parentheses `()` when combining multiple conditions with `&` or `|`.

---

8. Handling Missing Data

Identifying Missing Values

```
df.isna()          # Shows True for missing values
df.isna().sum()    # Count of missing values per column
df.isna().any()    # True if column has any NaN
```

Dropping Missing Values

```
df.dropna(thresh=2)  # Keeps rows with at least 2 non-null values
```

Filling Missing Values

```
df.fillna(0)            # Fill NaN with 0
df.fillna(df.mean())    # Fill NaN with column mean
```

---

9. Merging and Joining DataFrames

Merging (based on common column key)

```
data1 = pd.DataFrame({
    'A': [1, 2, np.nan, 4, 5],
    'B': [np.nan, 2, 3, 4, 5]
})

data2 = pd.DataFrame({
    'A': [1, 2, np.nan, 4, 5],
    'D': [np.nan, 22, 43, 4, 5]
})

pd.merge(data1, data2, on='A')           # Merge on common column A
pd.merge(data1, data2, how='inner')      # Only common values
pd.merge(data1, data2, how='outer')      # All values from both
pd.merge(data1, data2, how='left')       # All from left
pd.merge(data1, data2, how='right')      # All from right
```

Joining (based on index)

```
df1.join(df2, how='outer')
```

---

10. Grouping and Aggregation

Example

```
data = {
    'Category': ['A', 'B', 'C'],
    'Store': ['51', '55', '51'],
    'Sales': [100, 200, 300],
    'Quantity': [10, 15, 5],
    'Date': pd.date_range('2023-01-01', periods=3)
}

df = pd.DataFrame(data)
```

```
# Group by single column
df.groupby('Category')['Sales'].sum()

# Group by multiple columns
df.groupby(['Store', 'Category'])['Sales'].sum()
```

Aggregation

```
df['Sales'].max()
df['Sales'].mean()
df['Sales'].median()
df['Sales'].agg(['mean', 'max', 'min'])
```

---

11. Pivot Tables and Cross Tabs

Pivot Table

```
pd.pivot_table(df, values='Sales', index='Region', columns='Product')
pd.pivot_table(df, values='Sales', index='Region', columns='Product',
aggfunc='median')
```

Cross Tabulation

```
pd.crosstab(df['Region'], df['Product'])  # Counts occurrences
```

---

12. Common DataFrame Operations

```
print(df.shape)       # (rows, columns)
print(df.columns)     # Column names
print(df.info())      # Data info and types
print(df.describe())  # Summary statistics
```

---

13. Applying Functions

```
def sqr(x):
    return x ** 2

df['B'] = df['B'].apply(sqr)
```

---

14. Extracting Features from Text Columns

Example 1: Extract Numbers from Text

```
def extract_episode(data):
    check = False
    num = ""
    for i in data:
        if i == '(':
            check = True
            continue
        if i == ')':
            break
        if check:
            num += i
    return num

def just_numbers(data):
    value = ""
    for i in data:
        if i >= '0' and i <= '9':
            value += i
    return value

df = pd.read_csv(r'C:\Users\Rapid\Downloads\anime.csv')
df['Episodes'] = df['Title'].apply(extract_episode)
df['Episodes'] = df['Episodes'].apply(just_numbers)
df['Episodes'] = df['Episodes'].astype(int)
```

Example 2: Extract Text After a Symbol

```
def extract_time(data):
```

```
    txt = ""
    for i in range(len(data)):
        if data[i] == ')':
            for j in range(i+1, i+20):
                txt += data[j]
            return txt

df['Total Time'] = df['Title'].apply(extract_time)
```

---

15. Summary of Common Functions

Function	Description
`pd.read_csv()`	Read CSV file
`df.to_csv()`	Save DataFrame to CSV
`df.head()`	Display first 5 rows
`df.tail()`	Display last 5 rows
`df.sort_values(by='col')`	Sort by a column
`df.rename(columns={'A':'B'})`	Rename columns
`df.drop_duplicates()`	Remove duplicate rows
`df.reset_index(drop=True)`	Reset index