1. Introduction to Pandas

Pandas is a powerful Python library used for data manipulation and analysis. It provides data structures and functions that make it easy to manipulate structured data.

2. Reading Data

Pandas can read data from various file formats like CSV, JSON, Excel, and TXT.

CSV:

```
import pandas as pd
df = pd.read_csv('filename.csv')
```

JSON:

```
df = pd.read_json('filename.json')
```

Excel:

```
df = pd.read_excel('filename.xlsx')
```

TXT with a specified separator:

```
df = pd.read_csv('filename.txt', sep='\t')  # Tab-separated values
```

3. DataFrame

A DataFrame is a 2D labeled data structure with columns that can be of different types. Think of it like a spreadsheet or a SQL table.

```
data = {'Name': ['John', 'Anna'], 'Age': [28, 22]}
df = pd.DataFrame(data)
```

4. Series

A Series is a one-dimensional array-like object containing a sequence of values and an associated array of data labels, called its index.

```
series = pd.Series([1, 3, 5, 7, 9])
```

5. DataFrame Information

- `info()`: Provides a concise summary of a DataFrame.

```
df.info()
```

- `shape`: Returns a tuple representing the dimensionality of the DataFrame.

```
df.shape
```

- `set_option`: This method allows you to customize the behavior of your pandas environment.

```
pd.set_option('display.max_rows', None)  # Display all rows
pd.set_option('display.max_columns', None)  # Display all columns
```

6. Viewing Data

- `head()`: Views the first few rows of the DataFrame.

```
df.head()
```

- `tail()`: Views the last few rows of the DataFrame.

```
df.tail()
```

7. Accessing Data

- Single column:

```
df['columnName']
```

- Multiple columns:

```
df[['column1', 'column2']]
```

8. loc and iloc

- `loc`: Accesses a group of rows and columns by labels.

```
df.loc[0]  # First row
df.loc[:, 'columnName']  # Specific column
```

- `iloc`: Accesses a group of rows and columns by integer positions.

```
df.iloc[0]  # First row
df.iloc[:, 0]  # First column
```

9. Index

Indexes are immutable arrays that hold the axis labels and metadata like names and axis names. They are used for fast lookup and alignment.

10. Setting Index

`set_index()`: Sets the DataFrame index using existing columns.

```
df.set_index('columnName', inplace=True)
```

11. value_counts()

Returns a Series containing counts of unique values.

```
df['columnName'].value_counts()
```

12. Slicing

Slicing is used to select a set of rows or columns from a DataFrame.

```
df[1:3]  # Rows 1 to 2
```

13. Filtering

You can use conditions to filter rows.

```
df[df['Age'] > 25]
```

14. filter method

Filters labels based on whether they contain a certain string.

```
df.filter(like='substring', axis=1)
```

15. str.contains

Used to filter rows based on whether a column contains a specific string.

```
df[df['columnName'].str.contains('substring')]
```

16. Sorting

`sort_values()`: Sorts a DataFrame by one or more columns.

```
df.sort_values(by='columnName')
```

More Examples on Filtering

Filtering Based on a Single Condition: Find all entries where the age is above 30.

```
df[df['Age'] > 30]
```

1.

Filtering Based on Multiple Conditions: Select entries where age is greater than 25 and gender is 'Female'.

```
df[(df['Age'] > 25) & (df['Gender'] == 'Female')]
```

2.

Filtering with `isin` Method: Filter rows where the column value is in a list of values.

```
df[df['Country'].isin(['USA', 'Canada', 'UK'])]
```

3.

Using `query` Method for Filtering: This method is useful for complex filtering.

```
df.query('Age > 30 and Gender == "Male"')
```

4.

Filtering Based on String Methods: Select entries where a name column starts with 'J'.

```
df[df['Name'].str.startswith('J')]
```

5.

More Examples on Sorting

Sorting by a Single Column: Sort the DataFrame by the 'Age' column in ascending order.

```
df.sort_values(by='Age')
```

1.

Sorting in Descending Order: Sort by 'Salary' in descending order.

```
df.sort_values(by='Salary', ascending=False)
```

2.

Sorting by Multiple Columns: Sort by 'Department' first and then by 'Salary' within each department.

```
df.sort_values(by=['Department', 'Salary'])
```

3.

4. **Sorting with `nlargest` and `nsmallest`:**

Get the top 5 entries with the highest ages.

```
df.nlargest(5, 'Age')
```

o

Get the 3 entries with the smallest salaries.

```
df.nsmallest(3, 'Salary')
```

o

Sorting by Index: Reset index after sorting.

```
df.sort_values(by='Age', inplace=True)
df.reset_index(drop=True, inplace=True)
```

5.

Remember, when using `sort_values()`, the `inplace=True` parameter can be added to modify the DataFrame in place. Without it, `sort_values()` returns a new DataFrame, leaving the original unchanged.