Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

AY: 2024-25

Class:	**SE**	**Semester:**	**IV**
Course Code:	**CSL405**	**Course Name:**	**Skills Based Python Programming Lab**

Name of Student:	**Shravani Sandeep Raut**
Roll No. :	**48**
Experiment No.:	**11**
Title of the Experiment:	**Program to demonstrate Data Series using Pandas.**
Date of Performance:	**25/03/2025**
Date of Submission:	**01/04/2025**

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty : Mr. Raunak Joshi

Signature :

Date:

Aim: Program to demonstrate Data Series using Pandas.

Theory:

Pandas is a powerful and flexible open-source data analysis and manipulation library for Python. It provides data structures and functions needed to work with structured data seamlessly. One of its core data structures is the **Series**, which is a one-dimensional labeled array capable of holding data of any type (integers, floats, strings, etc.).

Introduction to Pandas Series:

- **Pandas Series:** A one-dimensional labeled array similar to a column in an Excel spreadsheet or a dictionary with ordered keys. It consists of:
    - **Data**: The values stored in the Series.
    - **Index**: Labels corresponding to each data point, which makes data retrieval easier.
- **Why Use Series?**
    - Efficient data storage and manipulation.
    - Indexing allows for quick and intuitive data access.
    - Integration with NumPy for mathematical operations.
    - Compatibility with other Pandas data structures like DataFrames.

Creating a Pandas Series:

A Pandas Series can be created in several ways:

- From a Python list or tuple.
- From a NumPy array.
- Using a Python dictionary.
- Using a scalar value.

Accessing Data from a Series:

Pandas Series supports two main types of indexing:

- **Position-based indexing** using iloc[].
- **Label-based indexing** using loc[].

Applications of Pandas Series:

- Time series data analysis.
- Statistical data analysis.
- Data cleaning and preprocessing.
- Financial data manipulation.
- Machine learning feature extraction.

Implementation:

```
import pandas as pd
```

In [2]:

```
# Step 1: Creating a Data Series using a list
data_list = [10, 20, 30, 40, 50]
series_from_list = pd.Series(data_list)
```

In [3]:

```
# Step 2: Creating a Data Series using a dictionary
data_dict = {'A': 10, 'B': 20, 'C': 30, 'D': 40, 'E': 50}
series_from_dict = pd.Series(data_dict)
```

In [4]:

```
# Step 3: Display the Series
print("Series from List:")
print(series_from_list)

print("\nSeries from Dictionary:")
print(series_from_dict)
```

```
Series from List:
0    10
1    20
2    30
```

```
3    40
4    50
dtype: int64

Series from Dictionary:
A    10
B    20
C    30
D    40
E    50
dtype: int64
```

In [5]:

```
# Step 4: Accessing Elements (by index)
print("\nAccessing the element at index 2 in series_from_list:",
   series_from_list[2])
```

```
Accessing the element at index 2 in series_from_list: 30
```

In [6]:

```
# Step 5: Operations on the Series (e.g., adding a constant)
series_added = series_from_list + 10
print("\nAdding 10 to each element in series_from_list:")
print(series_added)
```

```
Adding 10 to each element in series_from_list:
0    20
1    30
2    40
3    50
4    60
dtype: int64
```

In [7]:

```
# Step 6: Applying a function (e.g., doubling the values)
doubled_series = series_from_list.apply(lambda x: x * 2)
print("\nDoubling the values of series_from_list:")
print(doubled_series)
```

```
Doubling the values of series_from_list:
0     20
1     40
2     60
3     80
4    100
dtype: int64
```

In [8]:

```
# Step 7: Checking for NaN values (example with missing data)
series_with_nan = pd.Series([1, 2, None, 4, 5])
print("\nSeries with NaN value:")
print(series_with_nan)
print("\nChecking for NaN values:")
print(series_with_nan.isna())
```

```
Series with NaN value:
0    1.0
1    2.0
2    NaN
3    4.0
4    5.0
dtype: float64

Checking for NaN values:
0    False
1    False
2     True
3    False
4    False
dtype: bool
```

Conclusion :

Pandas Series is a versatile data structure that provides labeled indexing, efficient storage, and powerful operations on one-dimensional data. Its integration with NumPy makes it suitable for numerical computations, while its flexibility with indexing makes it ideal for labeled data manipulation. Using Series, data access, manipulation, and analysis become straightforward and efficient.