

Chapter 1 · Pandas — Series

CBSE Class 12 Informatics Practices (065) · Unit 1 = 25 marks (largest unit) · Ultimate Revision Sheet

60-SECOND RECALL

Import	<code>import pandas as pd</code> · <code>import numpy as np</code>
Series	One-dimensional labelled array, holds any data type, size immutable
Default index	0, 1, 2 ... unless you pass <code>index=</code>
head() / tail()	First / last 5 rows by default
Label slicing	<code>s['a':'c']</code> → INCLUDES the end label
Position slicing	<code>s[0:3]</code> → EXCLUDES the end position

1. Pandas and NumPy — the Basics

DEFINITIONS Pandas (Panel Data) is an open-source Python library for data manipulation and analysis. Its two main data structures are **Series** (1-D) and **DataFrame** (2-D).

NumPy (Numerical Python) provides the **ndarray**, a fast n-dimensional array that Pandas is built on.

Matplotlib is the library used for plotting.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Basis	SERIES	DATAFRAME
Dimension	1-D	2-D
Structure	A single column with an index	Rows and columns, like a table
Data types	Homogeneous (one dtype)	Heterogeneous (different dtype per column)
Size	Size immutable	Size mutable
Value mutability	Both are value mutable — you can change the data	

SERIES — 1 dimension

index	values
a	10
b	20
c	30
d	40

one column of data + an index
.shape → (4,)

DATAFRAME — 2 dimensions

index	Name	Marks	City
0	Asha	91	Pune
1	Bala	78	Delhi
2	Chen	85	Kochi

rows + named columns
.shape → (3, 3)

Pull **ONE** column out of a DataFrame and what you get back is a **Series**.

`df['Marks']` → Series · `df[['Marks']]` → still a DataFrame. One bracket or two decides the type.

A DataFrame is a dictionary of Series that all share one index.

2. Creating a Series

SYNTAX

```
pd.Series(data, index=None, dtype=None)
```

THE FOUR WAYS — every exam uses one of these

1. From a LIST

```
s1 = pd.Series([10, 20, 30, 40])
```

OUTPUT

```
0    10
1    20
2    30
3    40
dtype: int64
```

2. From an ndarray, with a custom index

```
arr = np.array([1, 2, 3])
s2 = pd.Series(arr, index=['a', 'b', 'c'])
```

OUTPUT

```
a    1
b    2
c    3
dtype: int32
```

3. From a DICTIONARY — keys become the index

```
d = {'Maths': 95, 'Science': 88, 'English': 72}
s3 = pd.Series(d)
```

OUTPUT

```
Maths    95
Science   88
English   72
dtype: int64
```

4. From a SCALAR value — index is compulsory, value is repeated

```
s4 = pd.Series(5, index=[1, 2, 3, 4])
```

OUTPUT

```
1    5
2    5
3    5
4    5
dtype: int64
```

TRAP When creating from a *scalar*, the *index* argument is mandatory — without it Python raises an error, because Pandas cannot know how many times to repeat the value.

3. Series Attributes

Attribute	Returns
s.index	The index labels
s.values	The data as an ndarray
s.dtype	Data type of the elements
s.size	Number of elements

<code>s.shape</code>	Tuple, e.g. (4,)
<code>s.ndim</code>	1 — always, for a Series
<code>s.empty</code>	True if the Series has no element
<code>s.hasnans</code>	True if any value is NaN
<code>s.name</code>	Name given to the Series

TRAP *Attributes take no brackets (`s.size`), but methods do (`s.head()`). Writing `s.size()` is an error.*

4. `head()`, `tail()` and Mathematical Operations

```
s = pd.Series([10, 20, 30, 40, 50, 60, 70])
```

```
s.head()      # first 5 by default
s.head(3)     # first 3
s.tail(2)     # last 2
```

OUTPUT of `s.head(3)`

```
0    10
1    20
2    30
dtype: int64
```

VECTORISED OPERATIONS

```
s = pd.Series([10, 20, 30])
s + 5      # adds 5 to every element
s * 2      # multiplies every element
s > 15     # returns a Boolean Series
```

*Series + Series aligns on the **INDEX**, not position:*

```
a = pd.Series([1, 2, 3], index=['x', 'y', 'z'])
b = pd.Series([10, 20, 30], index=['y', 'z', 'w'])
print(a + b)
```

OUTPUT

```
      w    NaN
x    NaN
y   12.0
z   23.0
dtype: float64
```

THE MOST-ASKED OUTPUT QUESTION *When two Series are added, Pandas **matches labels**. Any label present in one but not the other gives **NaN**, and the result becomes **float64** because NaN is a float. Expect this exact question.*

5. Selection, Indexing and Slicing

THE CRITICAL DIFFERENCE

```
s = pd.Series([10, 20, 30, 40], index=['a', 'b', 'c', 'd'])
```

```
s['b']          # 20 - single label
```

```
s[1]           # 20 - positional
```

```
s[['a', 'c']]  # multiple labels
```

```
s['a':'c']     # LABEL slice - INCLUDES 'c'
```

```
s[0:2]        # POSITION slice - EXCLUDES index 2
```

s['a':'c'] gives THREE items

```
a    10
```

```
b    20
```

```
c    30
```

s[0:2] gives TWO items

```
a    10
```

```
b    20
```

REMEMBER THIS ONE LINE Label-based slicing *INCLUDES* the end. Position-based slicing *EXCLUDES* the end.

This single rule is worth a mark almost every year.

```
s['b'] = 25          # modify a value
s[s > 20]           # Boolean filtering
```

6. Common Series Methods

Method	What it does
<code>s.head(n)</code> / <code>s.tail(n)</code>	First / last n elements
<code>s.count()</code>	Number of non-null values
<code>s.sum()</code> <code>s.mean()</code> <code>s.max()</code> <code>s.min()</code>	Aggregations, skipping NaN
<code>s.sort_values()</code>	Sort by value
<code>s.sort_index()</code>	Sort by index
<code>s.drop('b')</code>	Remove the element with that label
<code>s.rename()</code>	Change the index labels or the Series name

TRAP `count()` excludes NaN, but `size` includes it. For a Series with one NaN among four values, `size` is 4 and `count()` is 3.

7. Predict the Output the highest-frequency question type

Q1 `s = pd.Series([1,2,3,4], index=['a','b','c','d'])`
`print(s['a':'c'])`

Q2 `s = pd.Series([1,2,3,4])`
`print(s[1:3])`

Q3 `a = pd.Series([1,2], index=['x','y'])`
`b = pd.Series([10,20], index=['y','z'])`
`print(a+b)`

```
Q4 s = pd.Series(5, index=[1,2,3])
print(s.size, s.ndim)
```

```
Q5 s = pd.Series([10, None, 30])
print(s.size, s.count())
```

8. Task Triggers wording → which method?

If the question says...	Use
"display the first 3 records"	<code>s.head(3)</code>
"display the last 2 records"	<code>s.tail(2)</code>
"create a Series from a dictionary"	Keys become the index
"create a Series with the same value repeated"	Scalar + index is compulsory
"how many non-null values?"	<code>s.count()</code>
"how many elements in total?"	<code>s.size</code>
"arrange in ascending order of marks"	<code>s.sort_values()</code>
"arrange alphabetically by name"	<code>s.sort_index()</code>
"show only values greater than 50"	<code>s[s > 50]</code>
"remove the entry for 'b'"	<code>s.drop('b')</code>
"check if the Series has any missing value"	<code>s.hasnans</code>

MOST-ASKED BOARD QUESTIONS

1. [2-3M] Write code to create a Series from a list / dictionary / ndarray with a given index.
2. [2M] Predict the output of the given Series code.
3. [2M] Differentiate between a Series and a DataFrame.
4. [1M] Difference between `size` and `count()`.
5. [1M] What does `s['a':'c']` return? → Includes 'c'

THE HABIT THAT SCORES — In code answers, always write `import pandas as pd` on the first line. Examiners deduct for a missing import even when the rest is perfect. And when asked to predict output, **write the index column too** — the output of a Series is two columns, not one.