🧾 Pandas Series Cheat Sheet (Fully Functional + One-line Comments)

```
import pandas as pd

import numpy as np
```

🟢 Series Creation

```
pd.Series([1, 2, 3])                    # From list

pd.Series([1, 2, 3], index=['a', 'b', 'c'])  # List with custom index

pd.Series({'a': 1, 'b': 2})             # From dictionary

pd.Series(np.array([10, 20]))           # From NumPy array

pd.read_csv('file.csv', squeeze=True)   # From CSV column (1D)
```

🔷 Attributes

```
s.size        # Number of elements

s.dtype       # Data type of values

s.index       # Index object

s.values      # Numpy array of values

s.name        # Name of the Series

s.is_unique   # Check if all values are unique

s.ndim        # Number of dimensions (always 1)

s.shape       # Tuple of (size,)

s.empty       # Check if Series is empty
```

🟡 Indexing / Selection

```
s[0]          # By position

s['a']        # By label

s[1:3]        # Slicing by position

s.loc['a']    # Label-based access
```

```
s.iloc[1]     # Position-based access
s.at['a']     # Fast label access (no slicing)
s.iat[0]      # Fast position access
```

Math & Stats Methods

```
s.count()     # Non-NA values count
s.sum()       # Sum of all values
s.prod()      # Product of all values
s.mean()      # Mean
s.median()    # Median
s.mode()      # Most frequent value(s)
s.min()       # Minimum value
s.max()       # Maximum value
s.std()       # Standard deviation
s.var()       # Variance
s.describe()  # Summary statistics
```

Functional Methods

```
s.apply(func)             # Apply function to each element
s.map(func_or_dict)       # Element-wise mapping
s.agg(['sum', 'mean'])    # Multiple aggregations
```

Type & Conversion

```
s.astype('float')     # Change data type
s.to_list()           # Convert to Python list
s.to_dict()           # Convert to dict
s.to_numpy()          # Convert to NumPy array
s.copy()              # Deep copy of the Series
```

Sorting / Ranking

```
s.sort_values()          # Sort by values (ascending)

s.sort_index()           # Sort by index

s.rank()                 # Ranks of elements
```

Handling Duplicates

```
s.duplicated()           # Detect duplicate values

s.drop_duplicates()      # Drop duplicate values
```

Handling Missing Data

```
s.isnull()               # Check for NaN

s.notnull()              # Opposite of isnull()

s.dropna()               # Drop NaN values

s.fillna(0)              # Replace NaNs with value

s.fillna(method='ffill')    # Forward fill

s.fillna(method='bfill')    # Backward fill
```

Filtering / Boolean Indexing

```
s[s > 10]                # Filter values > 10

s[s.between(5, 15)]      # Values between 5 and 15

s[s.isin([1, 2, 3])]     # Values in list
```

Value Checks

```
s.unique()               # Unique values

s.nunique()              # Count of unique values

s.value_counts()         # Frequency of each value

s.equals(s2)             # Check equality with another Series
```

🟧 Modifying Data

```
s['a'] = 100            # Modify value by label

s[0] = 50               # Modify by position

s['new'] = 999          # Add new value

s.drop('a')             # Drop element (new Series)

s.drop('a', inplace=True)   # Drop in-place
```

🟪 Clip / Limit / Replace

```
s.clip(lower=10, upper=50) # Restrict values within range

s.replace(0, np.nan)      # Replace values
```

🟫 String Operations (if dtype is str)

```
s.str.upper()           # Convert to uppercase

s.str.contains('a')       # Check if contains 'a'

s.str.replace('x', 'y')   # Replace substrings
```

🔶 Plotting (Basic)

```
s.plot()                # Line plot

s.value_counts().plot(kind='bar')  # Bar plot
```

🧠 Bonus: DateTime Series

```
s = pd.Series(pd.date_range("2023-01-01", periods=3))

s.dt.year               # Extract year

s.dt.month              # Extract month

s.dt.day_name()         # Day name
```
