SAMSUNG

Samsung Innovation Campus

Coding, Programming & Data Science

Together for Tomorrow!
Enabling People
Education for Future Generations

Chapter 7.

Data Processing, Descriptive Statistics, and Data Visualization

Coding, Programming & Data Science

Chapter Description

Chapter objectives

- ✓ Learners will be able to collect various types of large amounts of data and organize them in a form that can be analyzed.
- ✓ Learners will be able to generate various descriptive statistics for organized data using Pandas.
- ✓ Learners can visualize data using the Python Visualization Library.

Chapter contents

- ✓ Unit 34. Using Python Modules
- ✓ Unit 35. Pandas Series for Data Processing
- ✓ Unit 36. Pandas DataFrame for Data Processing
- ✓ Unit 37. Data Tidying
- ✓ Unit 38. Time Series Data

Unit 35.

Pandas Series for Data Processing

Unit learning objective (1/3)

Learning objectives

- ✓ Be able to install the latest version of pandas library to the current virtual environment
- ✓ Be able to distinguish between series and dataframe when pandas-based data set is given
- ✓ Be able to process list and dictionary data sets into series
- ✓ Be able to select a specific data element from the series data set and perform arithmetic operation
- ✓ Be able to differentiate data feature and draw a suitable graph

Unit learning objective (2/3)

Learning overview

- ✓ Learn how to install the pandas library and other basic dependent libraries to the virtual environment
- ✓ Learn about two different types of pandas data structures
- ✓ Learn how to created series by using the list, dictionary, numpy, and scalar values
- ✓ Learn how to find a specific element in the series
- ✓ Learn how to perform arithmetic operation within the series structure
- ✓ Learn to visualize the series data set by using the matplotlib

Concepts you will need to know from previous units

- ✓ How to install and import python modules
- ✓ How to perform arithmetic operation
- ✓ Python data structures including list, dictionary, etc.

Unit learning objective (3/3)

Keywords

Pandas

Series

Data element

index

Line graph

bar graph

Mission

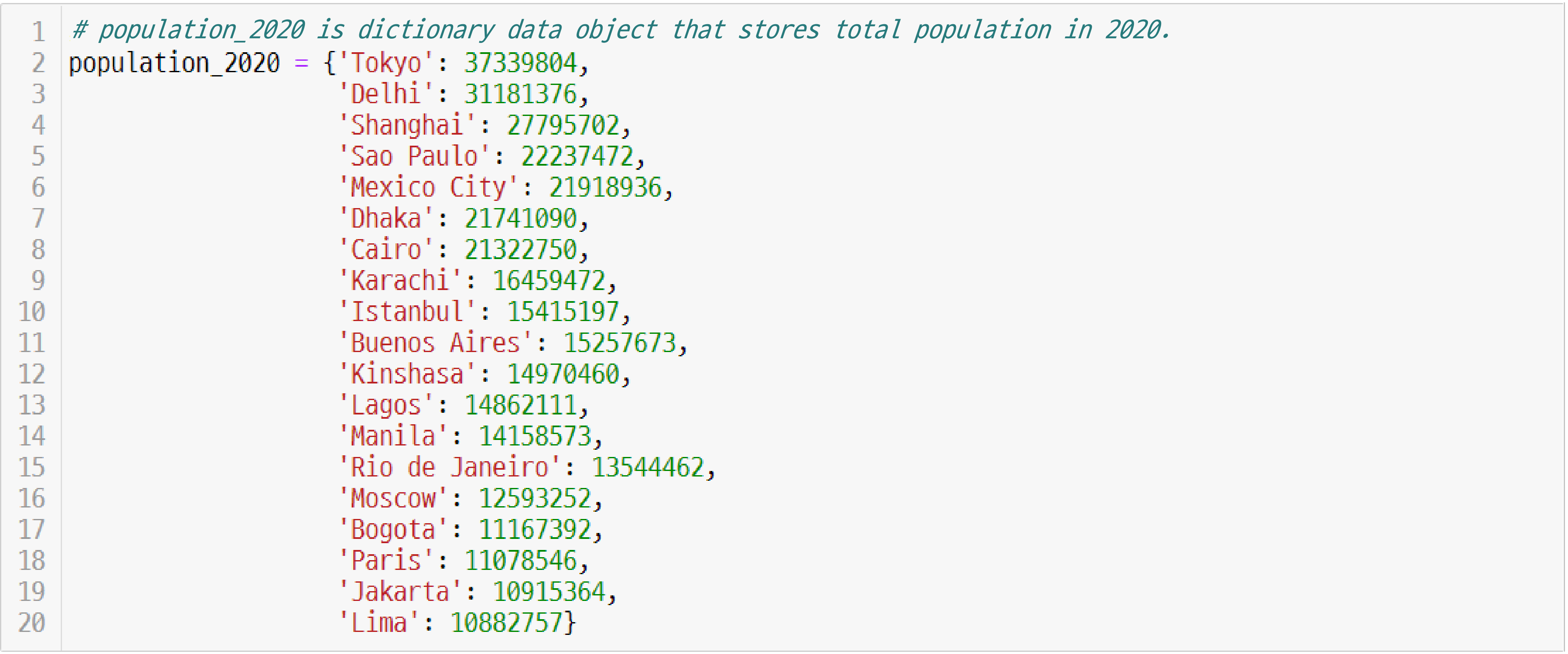
What is population density?

- Population density is the ratio of population in the unit area of a certain region. It is normally expressed as number of population in 1㎢.
- Population density can be used to explain the location, growth, and movement of many organisms.
  In the case of human, population density is often used for urbanization, immigration, and population statistics.
  Statistics related to global population density is traced by the UN Bureau of Statistics.
- Population density is easy to calculate.
  - Population density = Number of population / Area (㎢)

1. Convert the attached dictionary data into series objects.
2. Calculate the population fluctuation of each city in each year and express it as a bar graph.
3. Calculate the population density of each capital and find the cities with the maximum and minimum population density.
4. Visualize the population density calculation results.

- Use two attached dictionary data for solving the mission.
- Data source is https://unstats.un.org/ which organized 15 cities with largest populations.

What is population density?

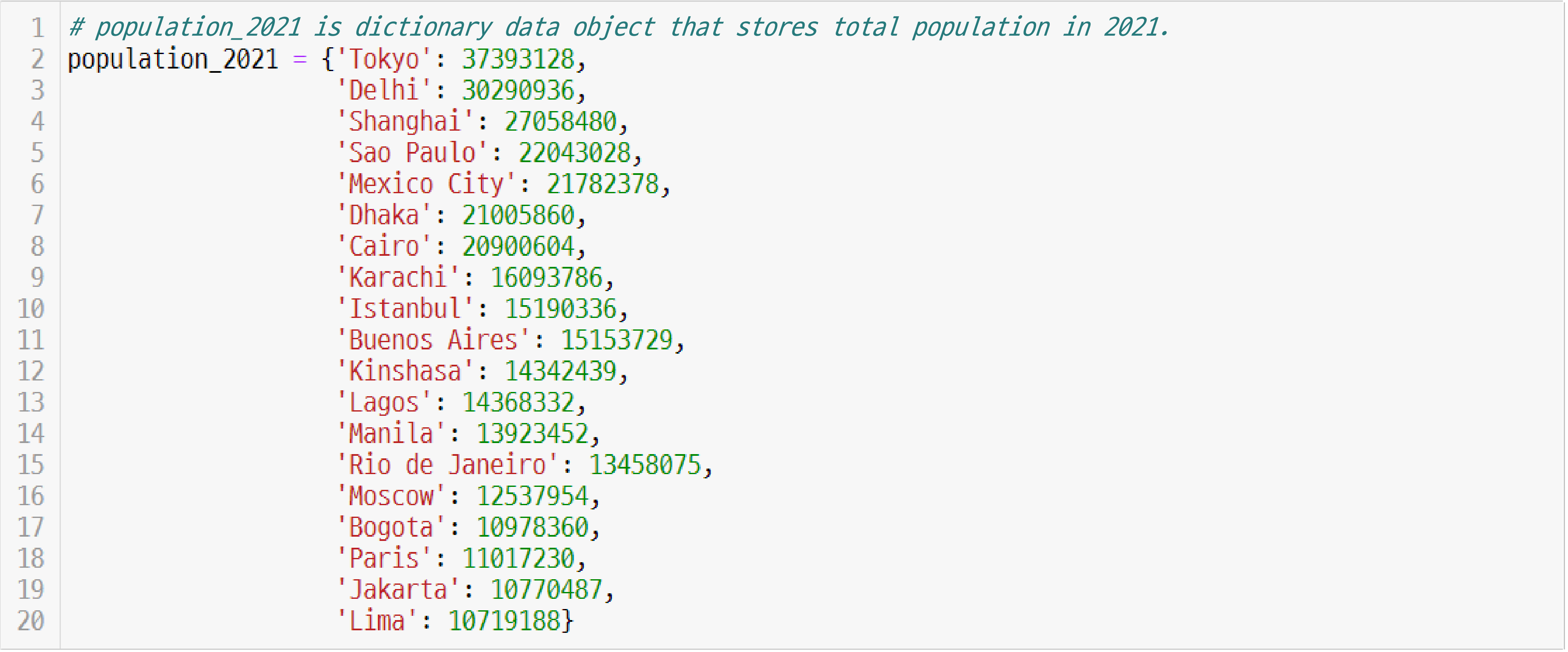


```
 1  # population_2020 is dictionary data object that stores total population in 2020.
 2  population_2020 = {'Tokyo': 37339804,
 3                     'Delhi': 31181376,
 4                     'Shanghai': 27795702,
 5                     'Sao Paulo': 22237472,
 6                     'Mexico City': 21918936,
 7                     'Dhaka': 21741090,
 8                     'Cairo': 21322750,
 9                     'Karachi': 16459472,
10                     'Istanbul': 15415197,
11                     'Buenos Aires': 15257673,
12                     'Kinshasa': 14970460,
13                     'Lagos': 14862111,
14                     'Manila': 14158573,
15                     'Rio de Janeiro': 13544462,
16                     'Moscow': 12593252,
17                     'Bogota': 11167392,
18                     'Paris': 11078546,
19                     'Jakarta': 10915364,
20                     'Lima': 10882757}
```

Line 1

- population_2020 is dictionary data object that stores total population of each city in 2020.

What is population density?

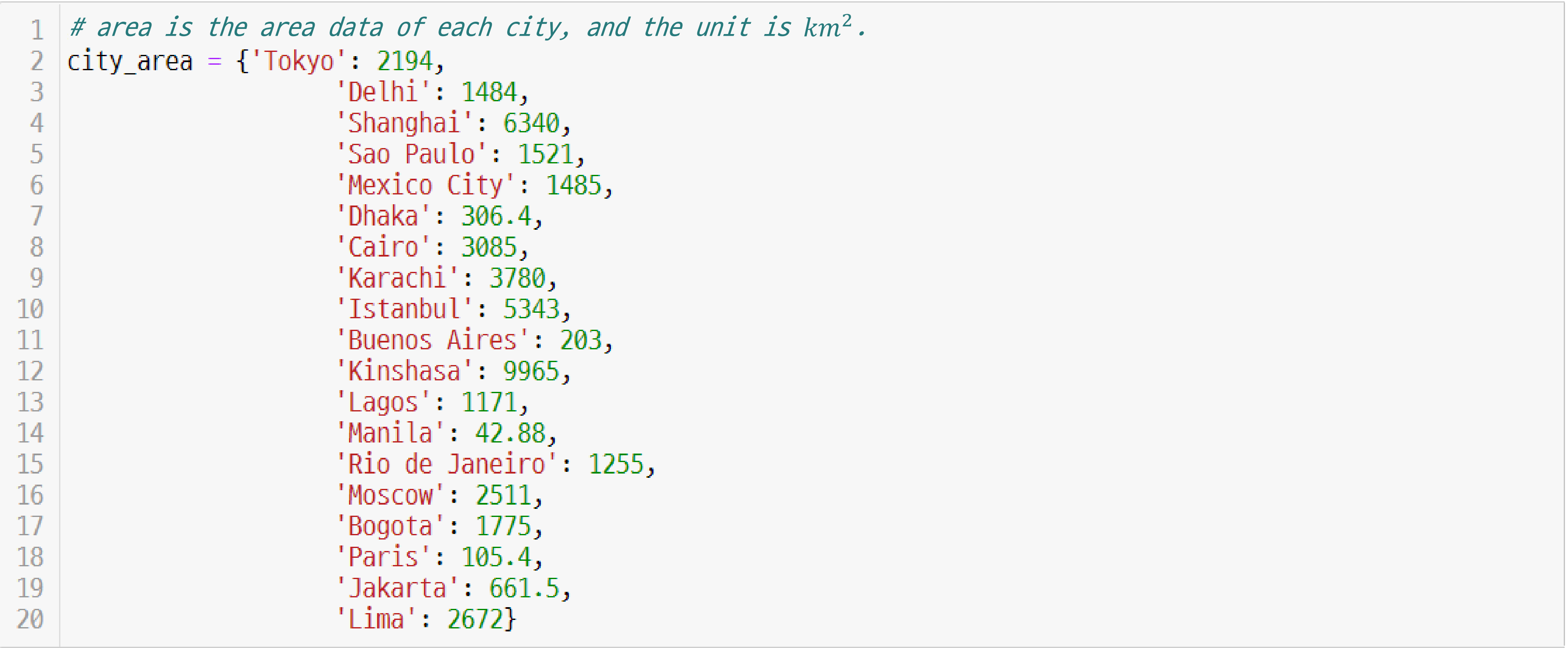


```
 1 # population_2021 is dictionary data object that stores total population in 2021.
 2 population_2021 = {'Tokyo': 37393128,
 3                    'Delhi': 30290936,
 4                    'Shanghai': 27058480,
 5                    'Sao Paulo': 22043028,
 6                    'Mexico City': 21782378,
 7                    'Dhaka': 21005860,
 8                    'Cairo': 20900604,
 9                    'Karachi': 16093786,
10                    'Istanbul': 15190336,
11                    'Buenos Aires': 15153729,
12                    'Kinshasa': 14342439,
13                    'Lagos': 14368332,
14                    'Manila': 13923452,
15                    'Rio de Janeiro': 13458075,
16                    'Moscow': 12537954,
17                    'Bogota': 10978360,
18                    'Paris': 11017230,
19                    'Jakarta': 10770487,
20                    'Lima': 10719188}
```

Line 1

- population_2021 is dictionary data object that stores total population of each city in 2021.

What is population density?



```python
1  # area is the area data of each city, and the unit is km².
2  city_area = {'Tokyo': 2194,
3                 'Delhi': 1484,
4                 'Shanghai': 6340,
5                 'Sao Paulo': 1521,
6                 'Mexico City': 1485,
7                 'Dhaka': 306.4,
8                 'Cairo': 3085,
9                 'Karachi': 3780,
10                'Istanbul': 5343,
11                'Buenos Aires': 203,
12                'Kinshasa': 9965,
13                'Lagos': 1171,
14                'Manila': 42.88,
15                'Rio de Janeiro': 1255,
16                'Moscow': 2511,
17                'Bogota': 1775,
18                'Paris': 105.4,
19                'Jakarta': 661.5,
20                'Lima': 2672}
```

Line 1

- area is the area data of each city, and the unit is km^2.

Key concept

1. Introduction of Pandas

- Developed in 2008 by Wes McKenny, pandas has been an open source since 2009 as a library for data processing.
- Pandas was originally designed for time series data operation and analysis in finance, especially stock price. To perform such operations, analytic tools including searching, indexing, refining, arranging, reshaping, and slicing were required, and pandas was developed as a solution.

1. Introduction of Pandas

Ever since switched to an open source, pandas supports effective data-related functions as follows due to the efforts made by many people. The list of functions provided below was complete by referring to https://Pandas.pydata.org/.

- Quick and effective series and dataframe objects for operating data with integrated indexing
- Intelligent data arrangement by using index and label
- Finding omitted data and supporting integrated processing
- Converting unorganized data into organized data
- Built-in tool to read and write data in the in-memory file, database, and web service
- Be able to process data stored in csv, excel, and json formats
- Flexible pivoting and remodeling of data set
- Slicing with index values, indexing, and subsetting of data set
- Addition and deletion of seta set columns
- Split-apply-combine functions to combine or change data as a powerful data grouping tool
- Integration with high performance data set merge
- Performance optimization through the critical path written with Cpython or C

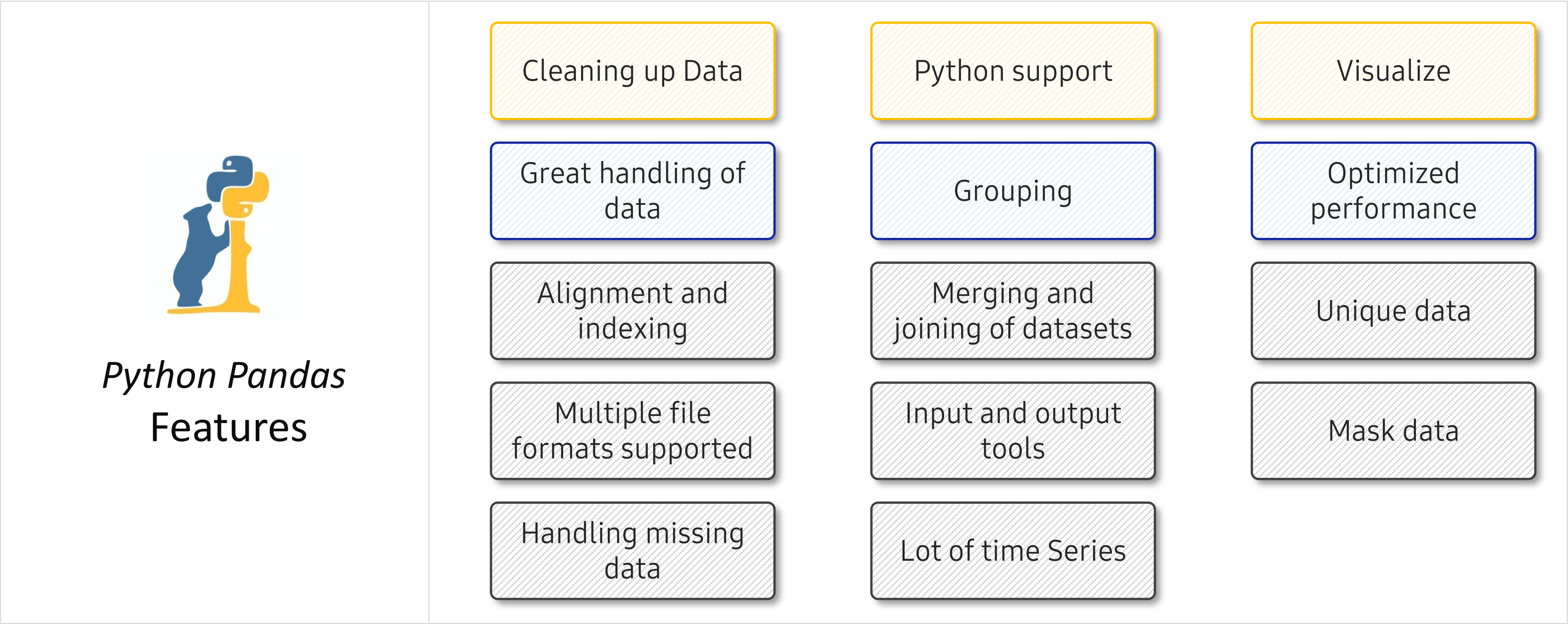
2. How to Use Pandas

- It is most appropriate to use pandas for data operation. However, not all issues related to comprehensive data can be solved only with pandas.
- Instead of being a comprehensive data analytic tool, pandas is a data operation tool that has some level of analytic function.
- For more in-depth analysis, data operation, and data visualization in different fields, use libraries including Scipy, Numpy, scikit-learn, matplotlib, seaborn, ggbis along with pandas. Likewise, a great advantage of pandas is that there are many python-based libraries to connect with.

2. How to Use Pandas

Advantages of learning pandas

- It is necessary to learn data processing for machine learning or AI deep learning, and even if not working in the AI field, data processing and analysis that go further beyond excel in regular working environment will be practically great advantage.



Python Pandas
Features

Cleaning up Data

Python support

Visualize

Great handling of data

Grouping

Optimized performance

Alignment and indexing

Merging and joining of datasets

Unique data

Multiple file formats supported

Input and output tools

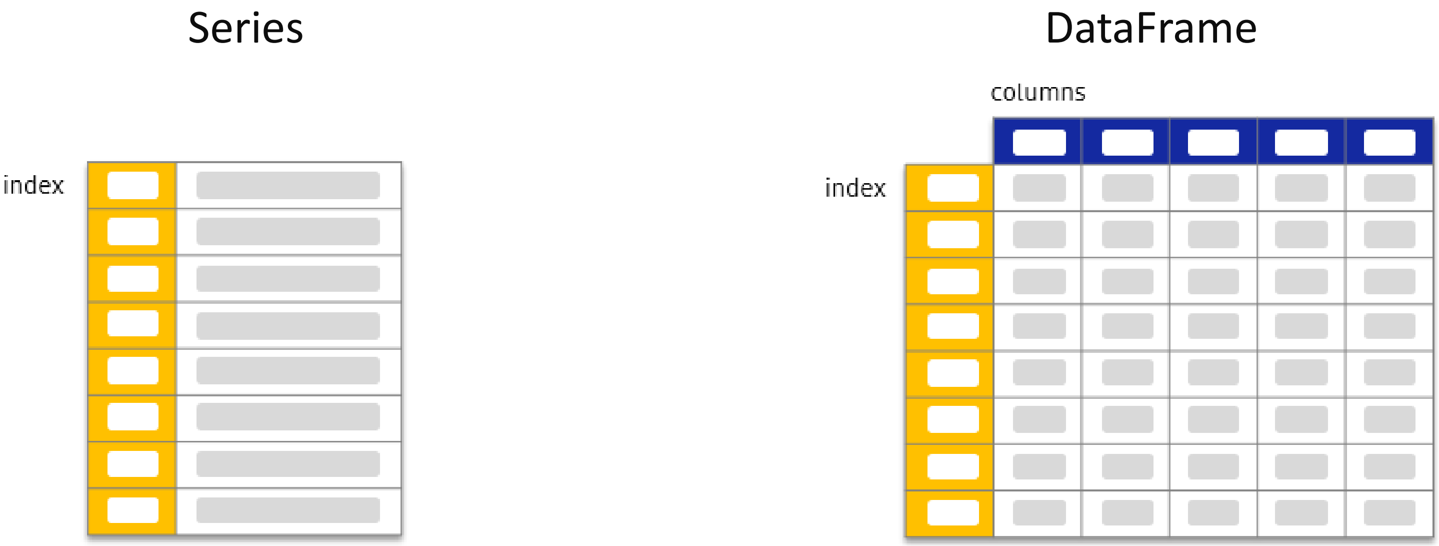
Mask data

Handling missing data

Lot of time Series

3. Pandas Has Two Different Data Structures

- Series and DataFrame are two different data structure types of Pandas. It is extremely important to fully understand those two data structures.
- The difference between the two data structures is that while Series has one-dimensional array data, DataFrame has two-dimensional array data.


Series

DataFrame

Sequentially arranged one-dimensional array
Index (yellow) and data (white) are 1:1.

Two-dimensional array consists of index and column (blue)
Adding multiple series can make one dataframe.

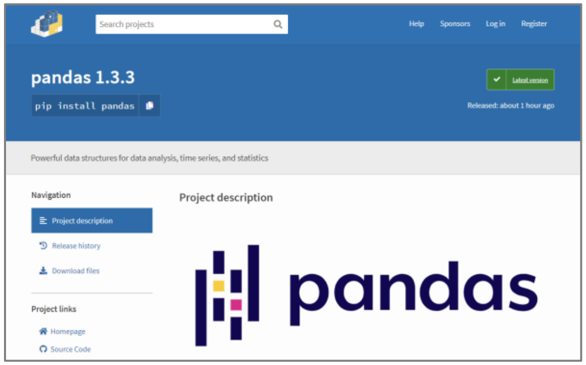
4. Install Pandas Library

1) The following is the official python version for pandas installation.

Officially Python 3.7.1 and above, 3.8, and 3.9.

- https://Pandas.pydata.org/docs/getting_started/install.html#python-version-support for reference

2) There are many different ways to install pandas, but this unit will only introduce pandas downloading from PyPl just like for external library installation introduced previously.



pandas 1.3.3

```
pip install pandas
```

Released: about 1 hour ago

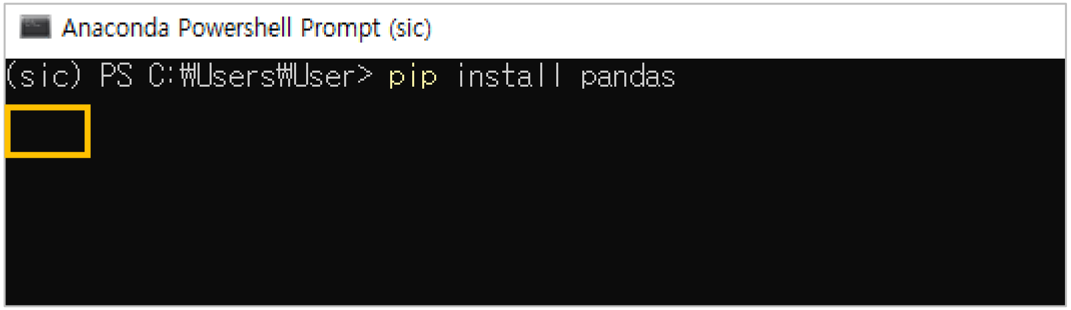
Powerful data structures for data analysis, time series, and statistics

Project description

- Just like any other external libraries, pandas can be installed from PyPl by using pip.
- https://pypi.org/project/Pandas/

4. Install Pandas Library

3) Make sure to run pip install Pandas after executing anaconda prompt and moving to the virtual environment. Installation must be done after moving to the currently used virtual environment. This is the most fundamental thing to remember for all of the other external library installation, but many beginners make mistakes.



Anaconda Powershell Prompt (sic)

```
(sic) PS C:₩Users₩User> pip install pandas
```

4) After installation, the Pandas.version command checks the installed pandas library version to determine if the installation is done appropriately.

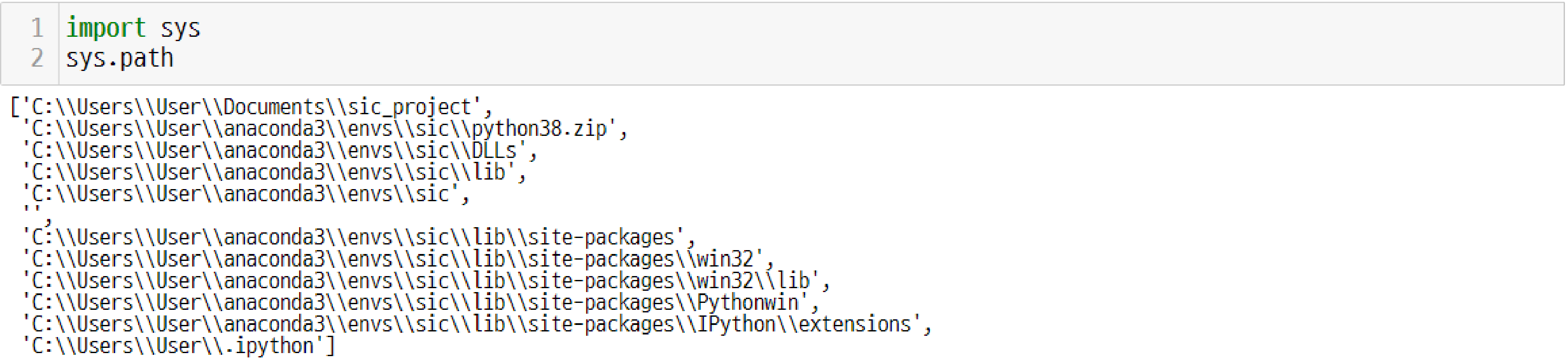
```python
import pandas as pd
pd.__version__
```

```
'1.3.2'
```

4. Install Pandas Library

5) Possible error after module installation (Solution for import ERROR).

- In general, ImportError means that it is impossible to find pandas from the python library list. Python has built-in directory list to find a package.
- Most of the time, error occurs because different versions of python are installed in the computer, and if the module is not installed to the currently used version of python installation environment but in python of a different version.
- Use the built-in python module sys to locate.



```
import sys
sys.path
```

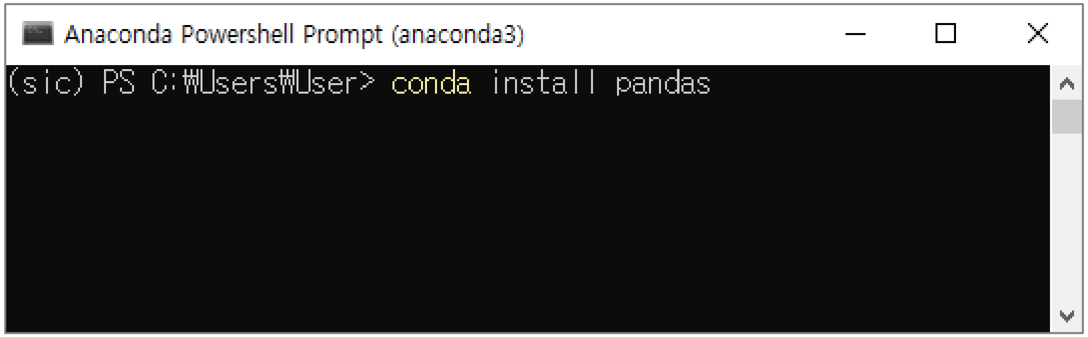
```
['C:\\Users\\User\\Documents\\sic_project',
 'C:\\Users\\User\\anaconda3\\envs\\sic\\python38.zip',
 'C:\\Users\\User\\anaconda3\\envs\\sic\\DLLs',
 'C:\\Users\\User\\anaconda3\\envs\\sic\\lib',
 'C:\\Users\\User\\anaconda3\\envs\\sic',
 '',
 'C:\\Users\\User\\anaconda3\\envs\\sic\\lib\\site-packages',
 'C:\\Users\\User\\anaconda3\\envs\\sic\\lib\\site-packages\\win32',
 'C:\\Users\\User\\anaconda3\\envs\\sic\\lib\\site-packages\\win32\\lib',
 'C:\\Users\\User\\anaconda3\\envs\\sic\\lib\\site-packages\\Pythonwin',
 'C:\\Users\\User\\anaconda3\\envs\\sic\\lib\\site-packages\\IPython\\extensions',
 'C:\\Users\\User\\.ipython']
```

- Linux/Mac can run a specific python with terminal and show which python installation is used. It is not recommended to use “/user/bin/python” because it uses the basic python of the system.

4. Install Pandas Library

6) The most precise and reliable installation method is to use conda.

- Conda is an open source package and environment management system that is executive in Windows, macOS, and Linux. Coda can easily and quickly install, execute, and update the package and relevant dependency. It is originally developed for python programs, but software packaging and distribution are possible for any other languages (Python, R, Ruby, Lua, Scala, Java, JavaScript, C/ C++, FORTRAN).
- Similar to installing other modules from pvpl, conda can be installed through the pip install conda, the it is highly likely that it's not in the latest version.
- Because conda is included in all versions of Anaconda®, Miniconda and Anaconda Repository, it is recommended to install Anaconda or Miniconda first.



```
(sic) PS C:₩Users₩User> conda install pandas
```

```
conda install Pandas(PKGNAME)==3.1.4(version)
```

5. Optional Dependencies

- When using pandas, there are many cases where you'd need dependencies packages to use specific methods. If you do not have them, it would result in difficulties. When you experience import error even if you used methods as learned from a book or lecture, you need to check dependencies packages.
- It is recommended to first install dependencies packages before using pandas.

Ex While the read_hdf() requires pytables package, DataFrame.to_markdown() requires tabulate package.

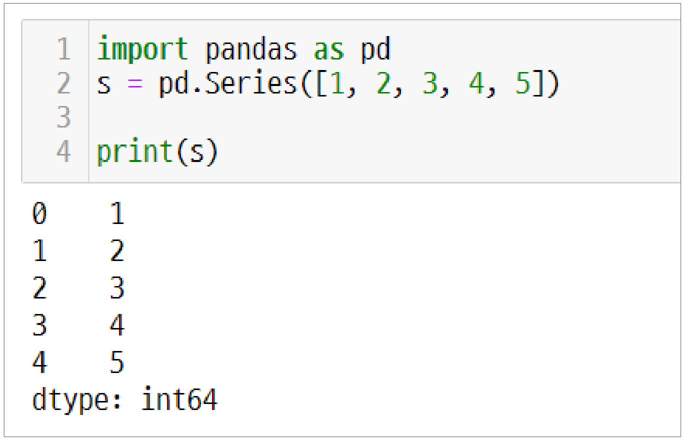
5. Optional Dependencies

The following table shows some of the optional dependencies required for this unit. It is mandatory to install them before continuing this chapter.

For computation	- Scipy : Miscellaneous statistical functions - numba : Alternative execution engine for rolling operations
Excel operation	- xlrd : Reading Excel - xlwt : Writing Excel - xlsxwriter : Writing Excel - openpyxl : Reading / writing for xlsx files - pyxlsb : Reading for xlsb files
HTML operation	- BeautifulSoup4 : HTML parser for read_html - html5lib : HTML parser for read_html - lxml : HTML parser for read_html
XML operation	- lxml : XML parser for read_xml and tree builder for to_xml
Data visualization	- matplotlib : Plotting library - seaborn : Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
SQL database	- SQLAlchemy : SQL support for databases other than sqlite - psycopg2 : PostgreSQL engine for sqlalchemy - pymysql : MySQL engine for sqlalchemy

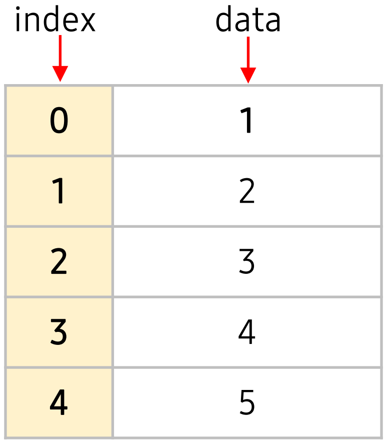
6. Series

- Series is the basic data structure of pandas.
- It is similar to Numpy array, but it is different that series has indices. As shown in the figure below, the index value and data have 1:1 ratio. Thus, the structure itself is similar to dictionary that has {key : value} structure.
- Each index value functions as the address of each data.



```python
import pandas as pd
s = pd.Series([1, 2, 3, 4, 5])

print(s)
```

```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```



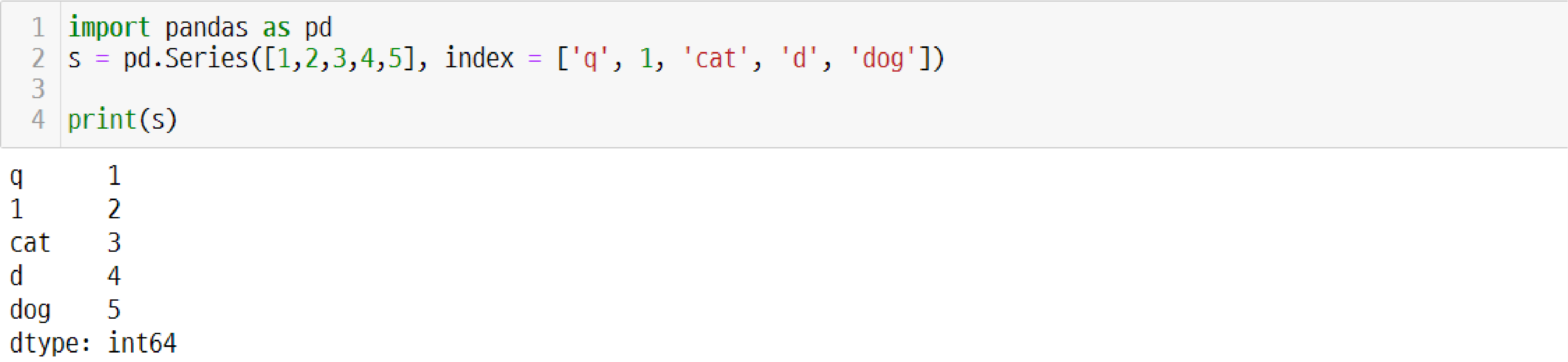
index	data
0	1
1	2
2	3
3	4
4	5

- From the sample code provided above, separate designation of an index is not required to create series. Pandas automatically generates integer type index from 0 that increases by 1 for each data item.

6. Series

TIP

- The series indices can be flexibly designated. It doesn’t necessarily start from 0. Indices other than integer type can be created as well.
- When creating series, use index parameter to designate desirable index values.



```python
import pandas as pd
s = pd.Series([1,2,3,4,5], index = ['q', 1, 'cat', 'd', 'dog'])

print(s)
```

```
q      1
1      2
cat    3
d      4
dog    5
dtype: int64
```

Visit the link https://Pandas.pydata.org/docs/reference/Series.html# to check available parameters for series.

7. Creating Series

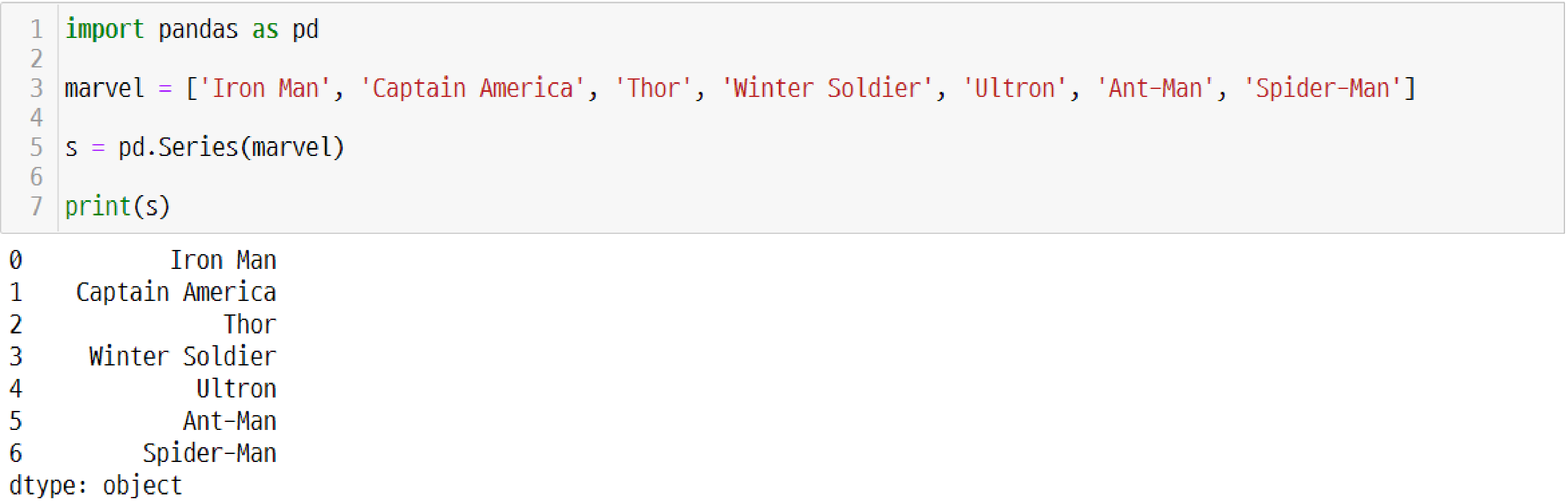
There are many different ways to create series, but only four of them are provided this chapter.

- Creating series by using python list
- Creating series by using python dictionary
- Creating series by using Numpy
- Creating series by using scalar value (a value that has designated range similar to integers)

7. Creating Series

7.1. Creating series from python list or dictionary

Pandas.Series (list or dictionary)



```python
import pandas as pd

marvel = ['Iron Man', 'Captain America', 'Thor', 'Winter Soldier', 'Ultron', 'Ant-Man', 'Spider-Man']

s = pd.Series(marvel)

print(s)
```

```
0           Iron Man
1    Captain America
2               Thor
3     Winter Soldier
4             Ultron
5            Ant-Man
6         Spider-Man
dtype: object
```

7. Creating Series

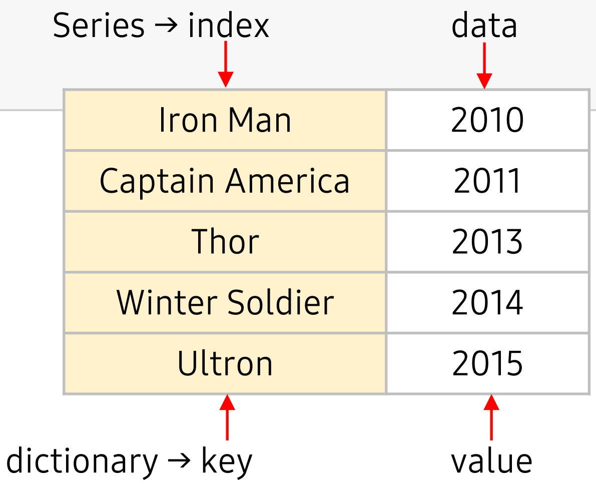
7.1. Creating series from python list or dictionary

Pandas.Series (list or dictionary)

- The dictionary key pairs with the index of series, while the dictionary value becomes each element (data value) of series.

```python
import pandas as pd

marvel_year = {'Iron Man': 2010, 'Captain America': 2011, 'Thor': 2013, 'Winter Soldier': 2014, 'Ultron': 2015}

s = pd.Series(marvel_year)

print(s)
s.index
```

```
Iron Man           2010
Captain America    2011
Thor               2013
Winter Soldier     2014
Ultron             2015
dtype: int64

Index(['Iron Man', 'Captain America', 'Thor', 'Winter Soldier', 'Ultron'], dtype='object')
```

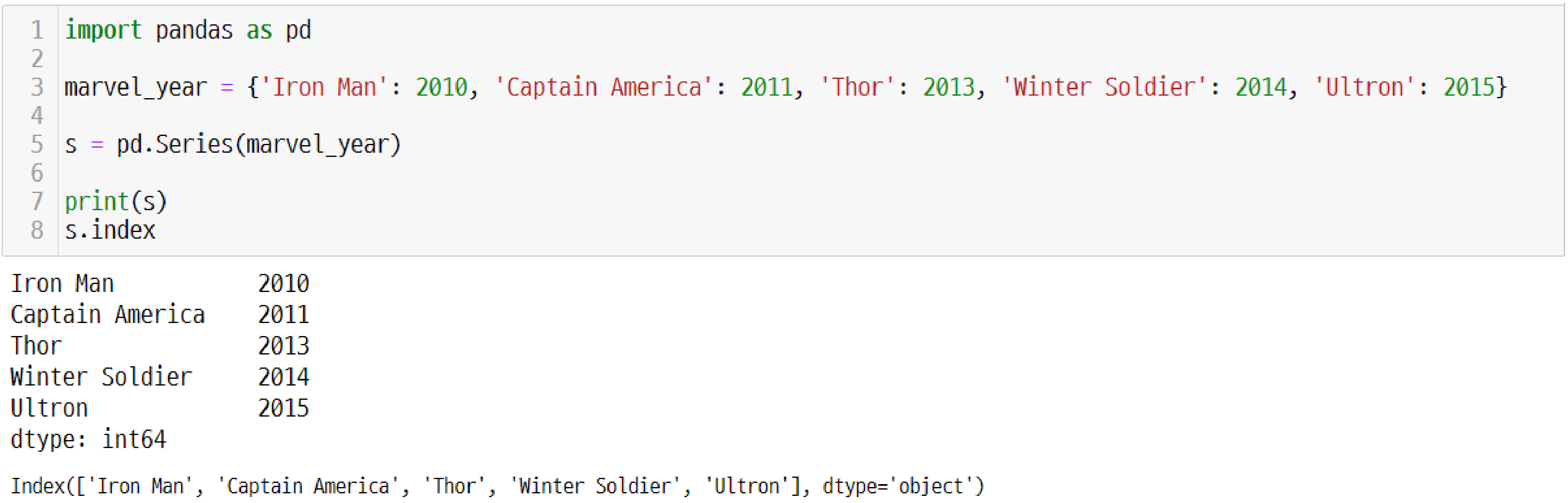


Iron Man	2010
Captain America	2011
Thor	2013
Winter Soldier	2014
Ultron	2015

7. Creating Series

7.1. Creating series from python list or dictionary

Pandas.Series (list or dictionary)



```
import pandas as pd

marvel_year = {'Iron Man': 2010, 'Captain America': 2011, 'Thor': 2013, 'Winter Soldier': 2014, 'Ultron': 2015}

s = pd.Series(marvel_year)

print(s)
s.index
```

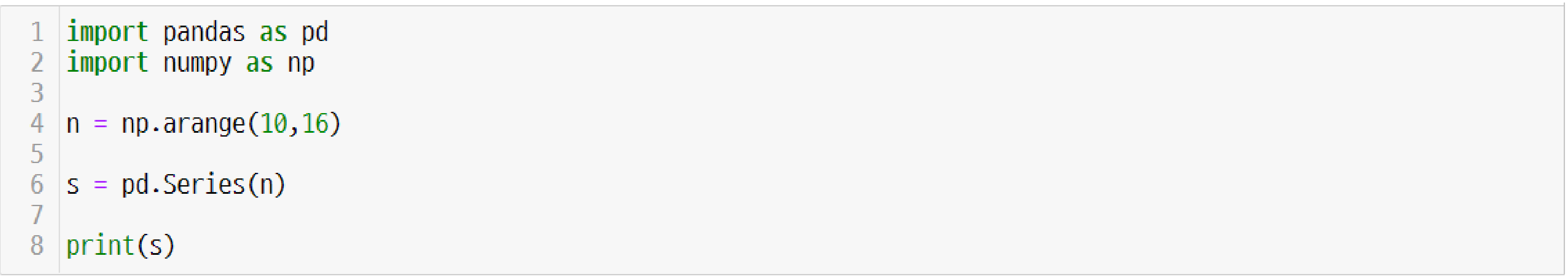
```
Iron Man           2010
Captain America    2011
Thor               2013
Winter Soldier     2014
Ultron             2015
dtype: int64
```

```
Index(['Iron Man', 'Captain America', 'Thor', 'Winter Soldier', 'Ultron'], dtype='object')
```

7. Creating Series

7.2. Creating series by using array data made with Numpy

- Series objects can be initialized by using different kinds of Numpy functions. Use array-creating functions and methods to create series.



```
1 import pandas as pd
2 import numpy as np
3 
4 n = np.arange(10,16)
5 
6 s = pd.Series(n)
7 
8 print(s)
```

```
0    10
1    11
2    12
3    13
4    14
5    15
dtype: int32
```

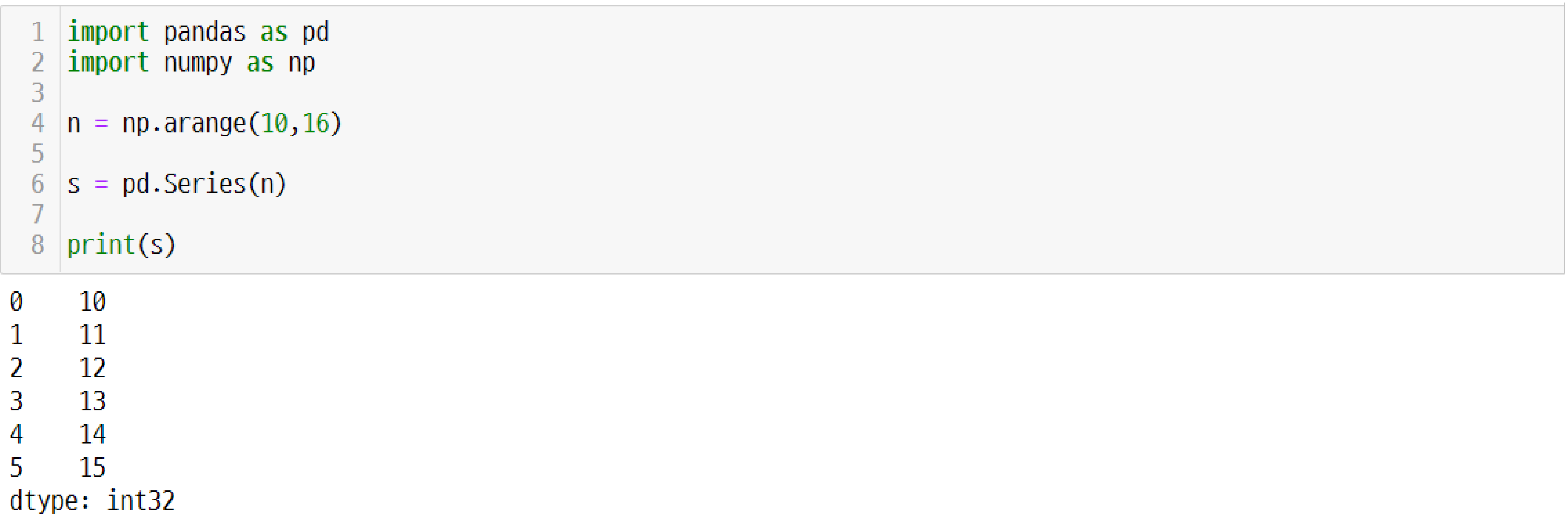
Line 4

- anrange() creates an array of integers from 10 to 15

7. Creating Series

7.2. Creating series by using array data made with Numpy

- Series objects can be initialized by using different kinds of Numpy functions. Use array-creating functions and methods to create series.



```
1 import pandas as pd
2 import numpy as np
3
4 n = np.arange(10,16)
5
6 s = pd.Series(n)
7
8 print(s)
```

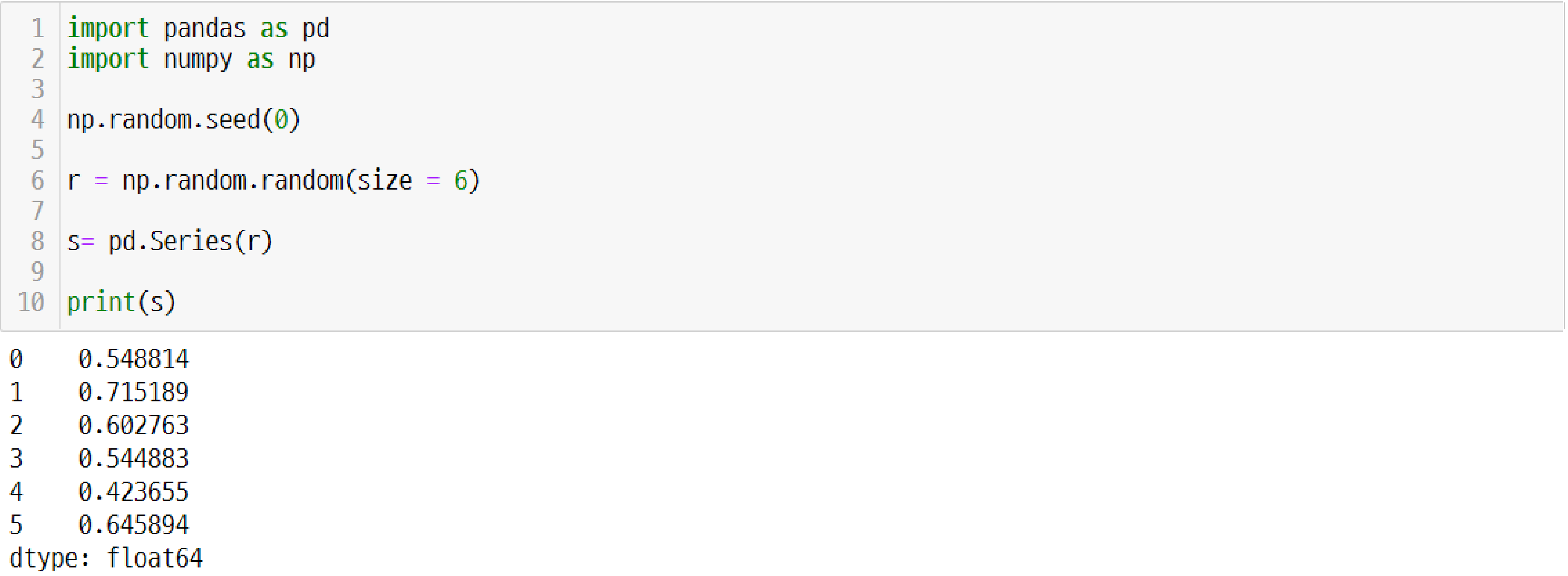
```
0    10
1    11
2    12
3    13
4    14
5    15
dtype: int32
```

Line 6

- The array created with numpy is converted into series object

7. Creating Series

7.2. Creating series by using array data made with Numpy



```
 1 import pandas as pd
 2 import numpy as np
 3 
 4 np.random.seed(0)
 5 
 6 r = np.random.random(size = 6)
 7 
 8 s= pd.Series(r)
 9 
10 print(s)
```

```
0    0.548814
1    0.715189
2    0.602763
3    0.544883
4    0.423655
5    0.645894
dtype: float64
```

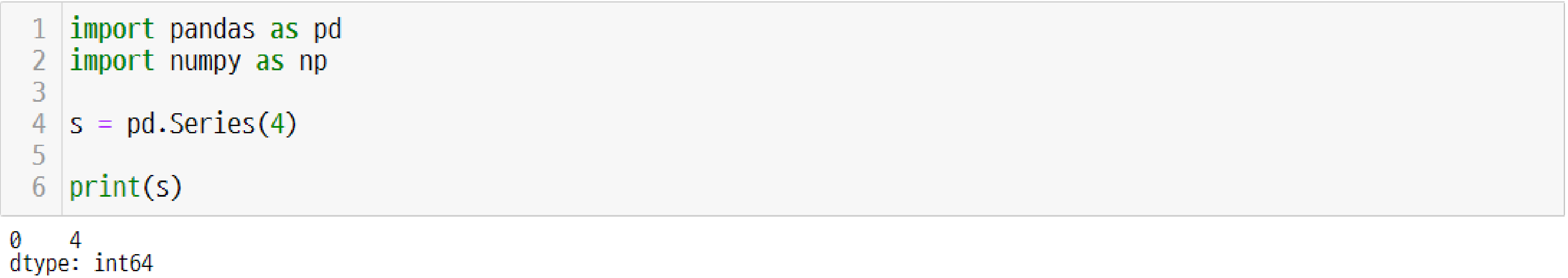
Line 6

- Creates 6 normally distributed random numbers and stores them to variable r.

7. Creating Series

7.3. Creating series by using scalar value

- Scalar value refers to a value that has designated range similar to integers.
- First, create a simple series object that has single data.



```
1  import pandas as pd
2  import numpy as np
3  
4  s = pd.Series(4)
5  
6  print(s)
```

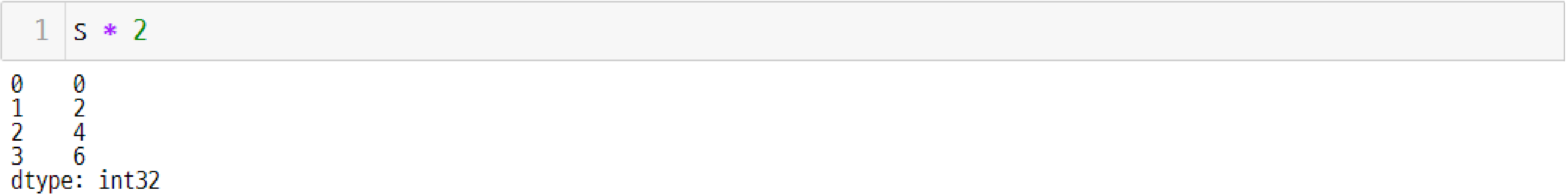
```
0    4
dtype: int64
```

Line 6

- Create series object consists of one index that has data of 4.

7. Creating Series

7.3. Creating series by using scalar value

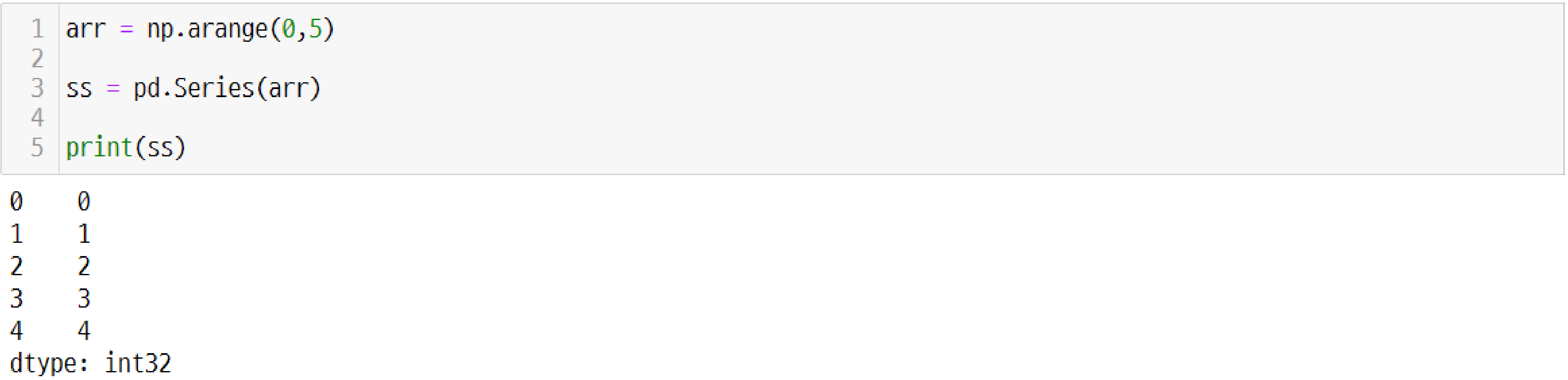


```
s * 2
```

```
0    0
1    2
2    4
3    6
dtype: int32
```

7. Creating Series

7.3. Creating series by using scalar value



```
1 arr = np.arange(0,5)
2
3 ss = pd.Series(arr)
4
5 print(ss)
```

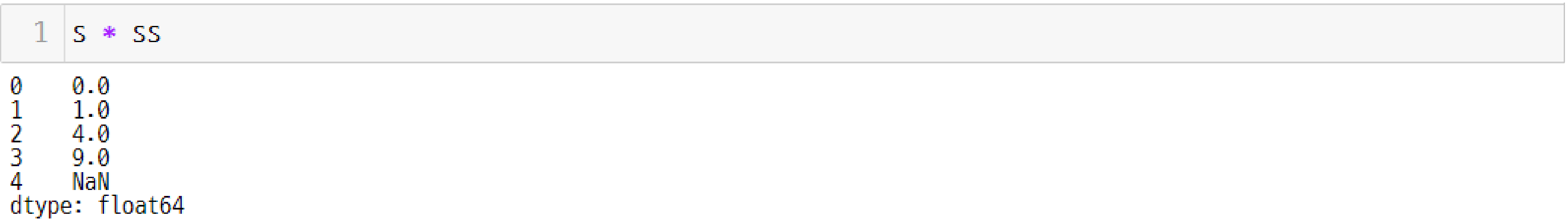
```
0    0
1    1
2    2
3    3
4    4
dtype: int32
```

Line 1, 3

- 1: Creates an array of integers from 0 to 5 in order
- 3: Uses the array data to create series object

7. Creating Series

7.3. Creating series by using scalar value

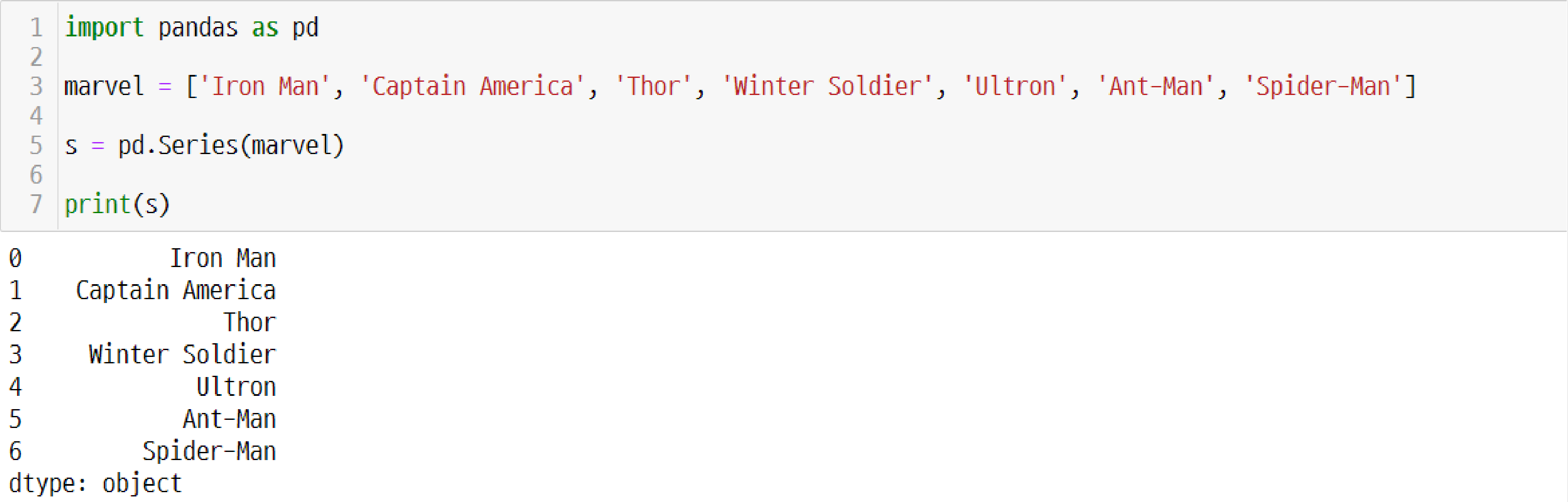


```
s * ss
```

```
0    0.0
1    1.0
2    4.0
3    9.0
4    NaN
dtype: float64
```

TIP

- In general, series is used to express time series data with indices of data and time.
  Use the Pandas.date_range() function to create range of data for an index value.



```
1  import pandas as pd
2
3  marvel = ['Iron Man', 'Captain America', 'Thor', 'Winter Soldier', 'Ultron', 'Ant-Man', 'Spider-Man']
4
5  s = pd.Series(marvel)
6
7  print(s)
```

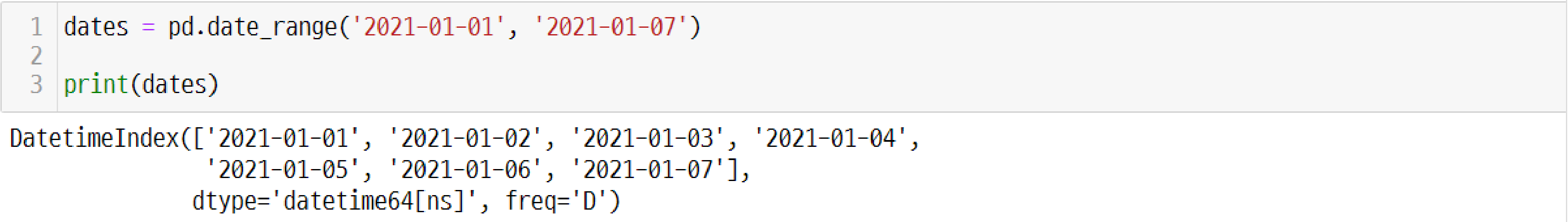
```
0          Iron Man
1    Captain America
2              Thor
3     Winter Soldier
4            Ultron
5           Ant-Man
6        Spider-Man
dtype: object
```

Line 5, 7

- 5: Converts a list into series object to prepare data for practice
- 7: When printed, the indices will consist of integers starting from 0.

TIP

- In general, series is used to express time series data with indices of data and time. Use the Pandas.date_range() function to create range of data for an index value.



```
1  dates = pd.date_range('2021-01-01', '2021-01-07')
2
3  print(dates)
```

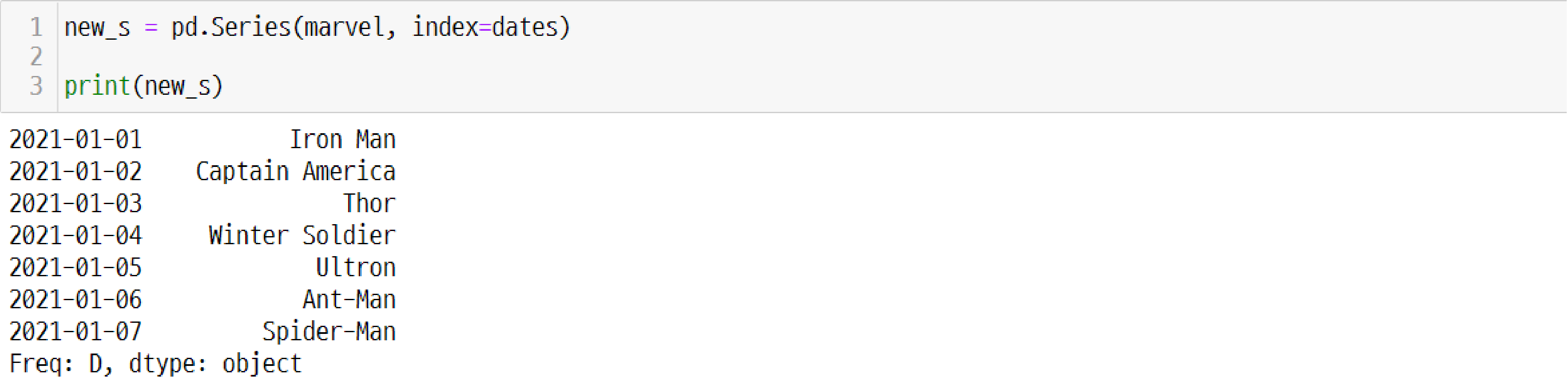
```
DatetimeIndex(['2021-01-01', '2021-01-02', '2021-01-03', '2021-01-04',
               '2021-01-05', '2021-01-06', '2021-01-07'],
              dtype='datetime64[ns]', freq='D')
```

Line 1, 3

- 1: Creates a range of date in the form of Pandas.date_range(‘starting date', ‘end data')
- 3: When printed, the special index data DatetimeIndex will be created.

TIP

- In general, series is used to express time series data with indices of data and time.
  Use the Pandas.date_range() function to create range of data for an index value.



```
1 new_s = pd.Series(marvel, index=dates)
2
3 print(new_s)
```

```
2021-01-01          Iron Man
2021-01-02    Captain America
2021-01-03              Thor
2021-01-04     Winter Soldier
2021-01-05            Ultron
2021-01-06           Ant-Man
2021-01-07        Spider-Man
Freq: D, dtype: object
```

Line 1~3

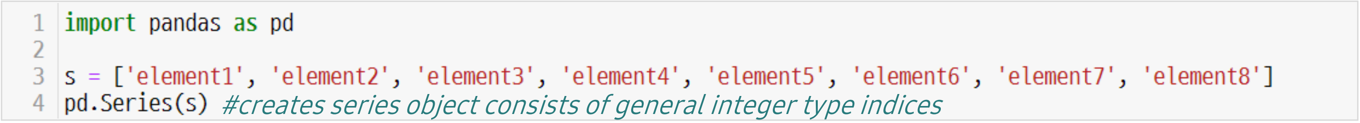
- 1, 2: Replaces the index of marvel, which is series object, to the data range that was previously created.

  Of course, the data amount of newly created index and object numbers must meet to prevent an error.
- 3: Checks the series object with changed index.

8. Selecting a Specific Element (Data) from Series

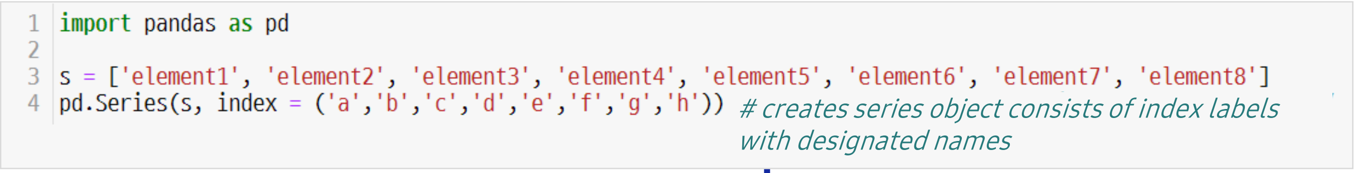
- In series object, each index has data that makes a 1:1 pair. Consider the index as intrinsic address of each data and use it to find, arrange, and select each data.
- As explained earlier, there are two types of indices. There are integer type index that is automatically created form 0 when nothing is designated, and index label that you can specifically designate index names.

Series consists of integer type index starting from 0

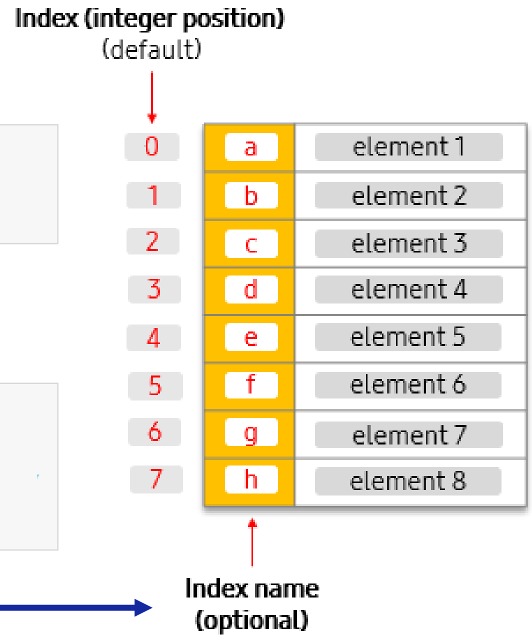


```
import pandas as pd

s = ['element1', 'element2', 'element3', 'element4', 'element5', 'element6', 'element7', 'element8']
pd.Series(s) #creates series object consists of general integer type indices
```

Series consists of index label other than integers

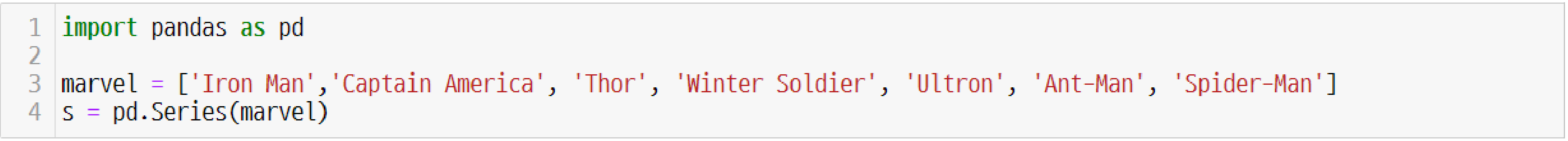


```
import pandas as pd

s = ['element1', 'element2', 'element3', 'element4', 'element5', 'element6', 'element7', 'element8']
pd.Series(s, index = ('a','b','c','d','e','f','g','h')) # creates series object consists of index labels
                                                          with designated names
```



8. Selecting a Specific Element (Data) from Series

- Series name.index: Inquires entire index values of the series data
- Series name.values: Inquires entire element values of the series data



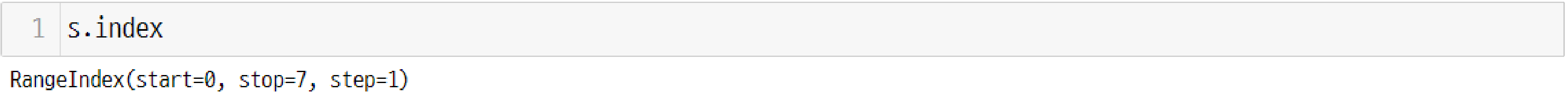
```
1  import pandas as pd
2  
3  marvel = ['Iron Man','Captain America', 'Thor', 'Winter Soldier', 'Ultron', 'Ant-Man', 'Spider-Man']
4  s = pd.Series(marvel)
```

Line 4

- Converts the list into series object to prepare data for practice.

8. Selecting a Specific Element (Data) from Series

- Series name.index: Inquires entire index values of the series data
- Series name.values: Inquires entire element values of the series data

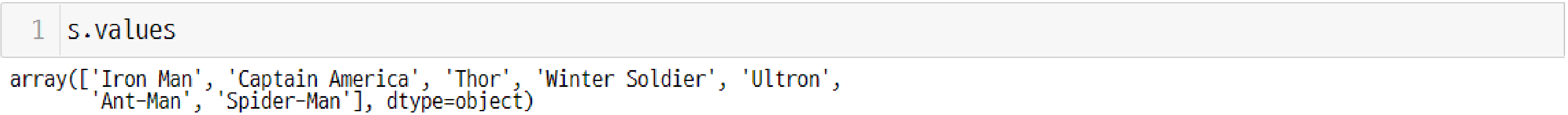


```
s.index
```

```
RangeIndex(start=0, stop=7, step=1)
```

Line 1

- Use index property to inquire the index value of the series



```
s.values
```

```
array(['Iron Man', 'Captain America', 'Thor', 'Winter Soldier', 'Ultron',
       'Ant-Man', 'Spider-Man'], dtype=object)
```

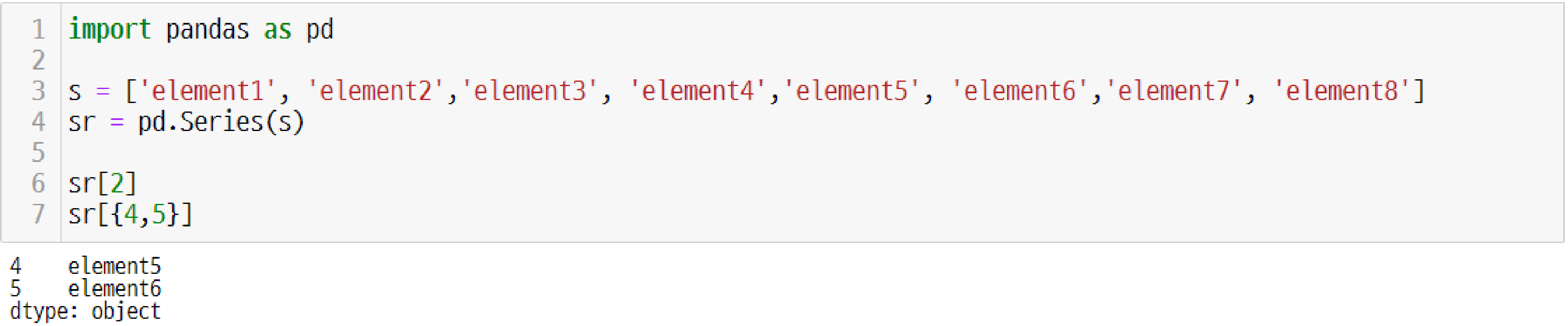
Line 1

- Use values property to inquire all of the data elements of the series

8. Selecting a Specific Element (Data) from Series

8.1. Selecting series element

- A method to select an element is different for integer type index or index label.
- How to select an element from integer type index



```
1 import pandas as pd
2 
3 s = ['element1', 'element2','element3', 'element4','element5', 'element6','element7', 'element8']
4 sr = pd.Series(s)
5 
6 sr[2]
7 sr[{4,5}]
```

```
4    element5
5    element6
dtype: object
```

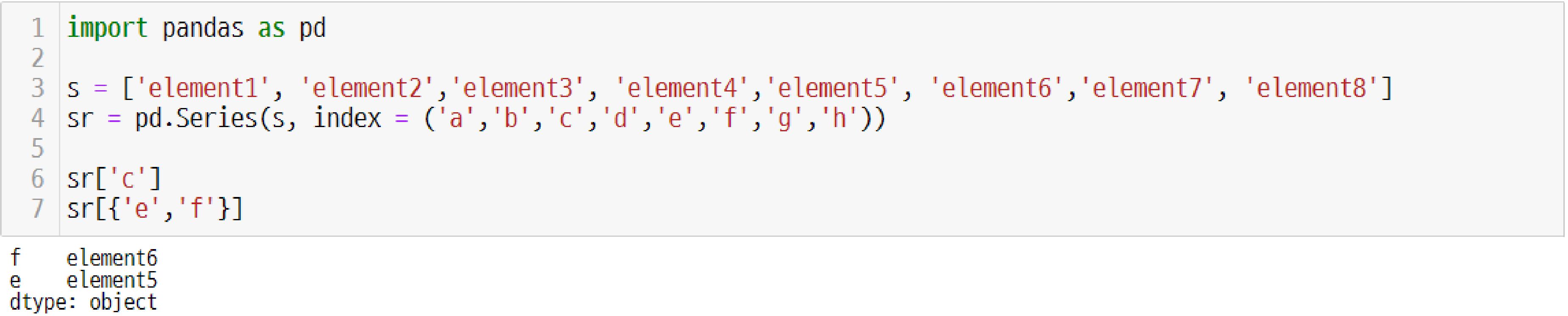
Line 6, 7

- 6: For selecting data with a single index address
- 7: For selecting data in index number range

8. Selecting a Specific Element (Data) from Series

8.1. Selecting series element

- How to select an element from index label



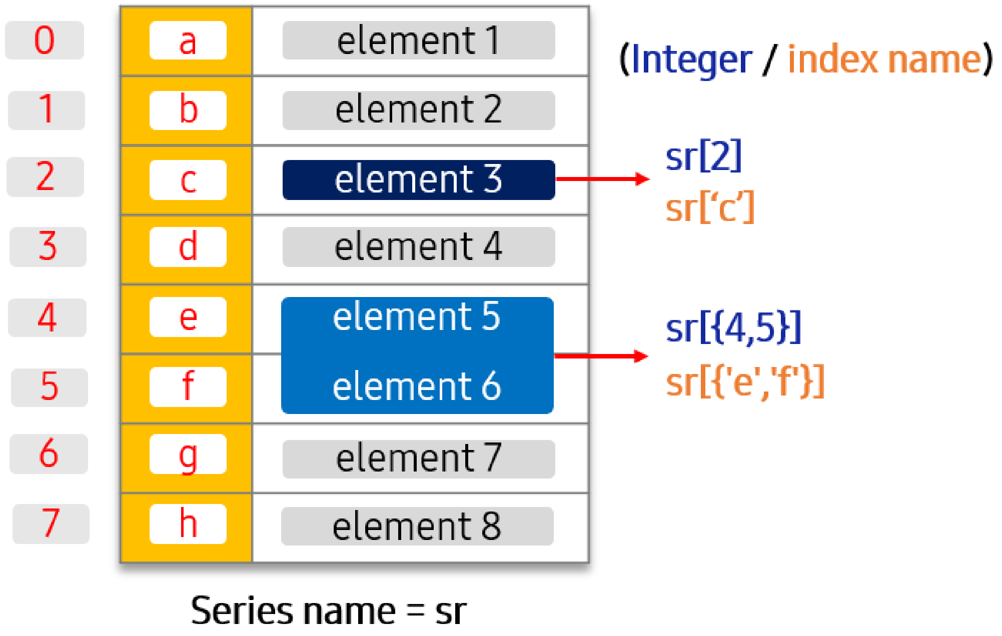
```
import pandas as pd

s = ['element1', 'element2','element3', 'element4','element5', 'element6','element7', 'element8']
sr = pd.Series(s, index = ('a','b','c','d','e','f','g','h'))

sr['c']
sr[{'e','f'}]
```

```
f    element6
e    element5
dtype: object
```

8. Selecting a Specific Element (Data) from Series

8.1. Selecting series element

- How to select an element from index label

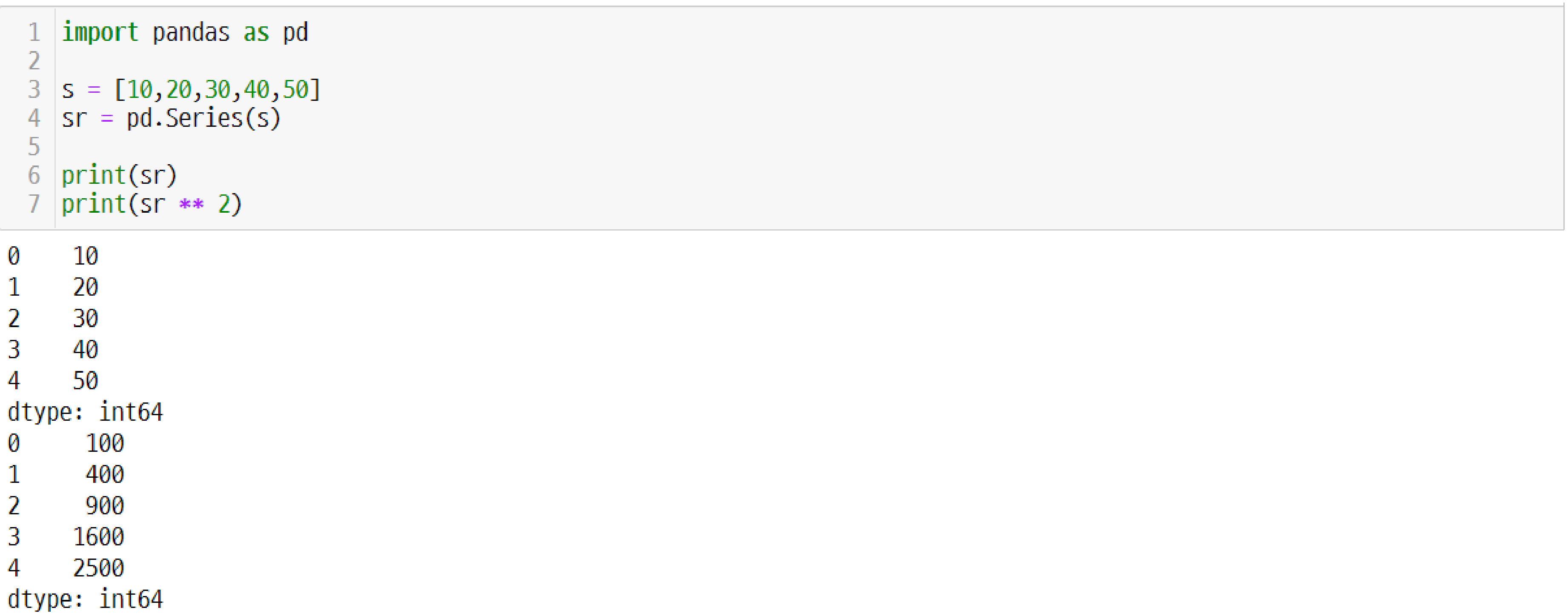


9. Basic Arithmetic Operation for Series Object (+, -, /, *, ...)

- Pandas perform three built-in stages of processes for arithmetic operation between data objects. Data are arranged in each row and column, and the elements on the same index (location) are paired up. If so, elements that are not paired up are classified as Nan.

9. Basic Arithmetic Operation for Series Object (+, -, /, *, ...)

9.1. Arithmetic operation for single series object



```
1 import pandas as pd
2
3 s = [10,20,30,40,50]
4 sr = pd.Series(s)
5
6 print(sr)
7 print(sr ** 2)
```

```
0    10
1    20
2    30
3    40
4    50
dtype: int64
0     100
1     400
2     900
3    1600
4    2500
dtype: int64
```

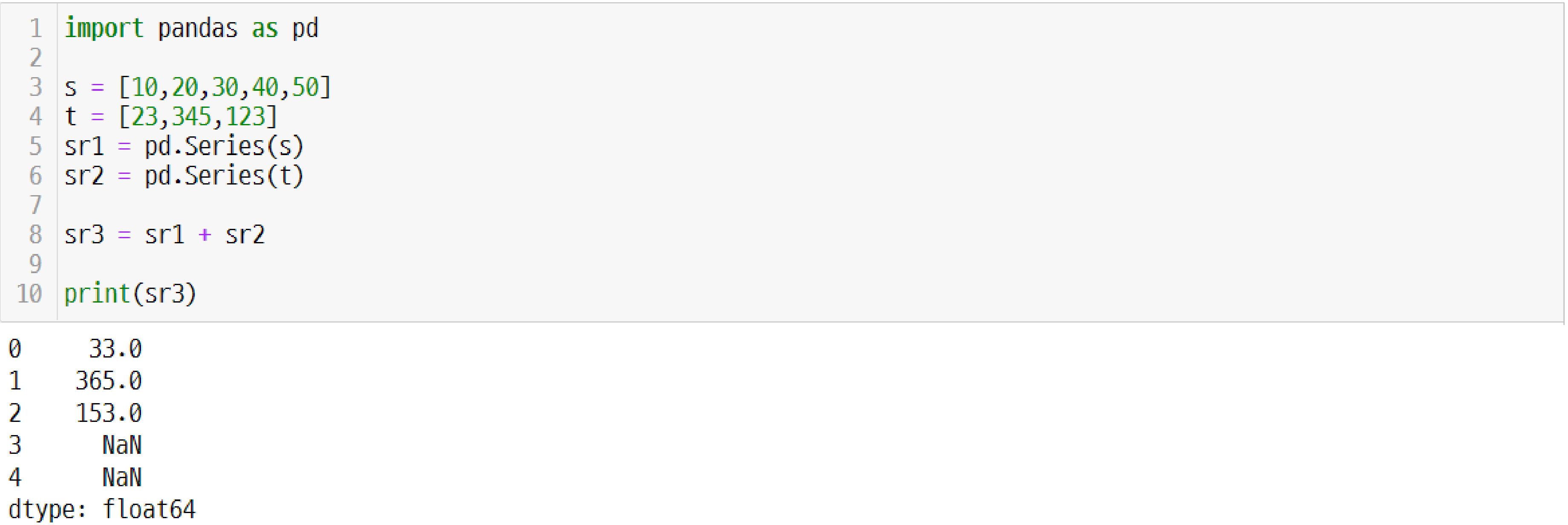
Line 7

- All of the elements in the sr Series object are multiplied by 2

9. Basic Arithmetic Operation for Series Object (+, -, /, *, ...)

9.2. Operation with index for 2 series objects

- If the elements of each index are not matched, the operation result returns NaN.



```
 1  import pandas as pd
 2  
 3  s = [10,20,30,40,50]
 4  t = [23,345,123]
 5  sr1 = pd.Series(s)
 6  sr2 = pd.Series(t)
 7  
 8  sr3 = sr1 + sr2
 9  
10  print(sr3)
```

```
0     33.0
1    365.0
2    153.0
3      NaN
4      NaN
dtype: float64
```

Line 8

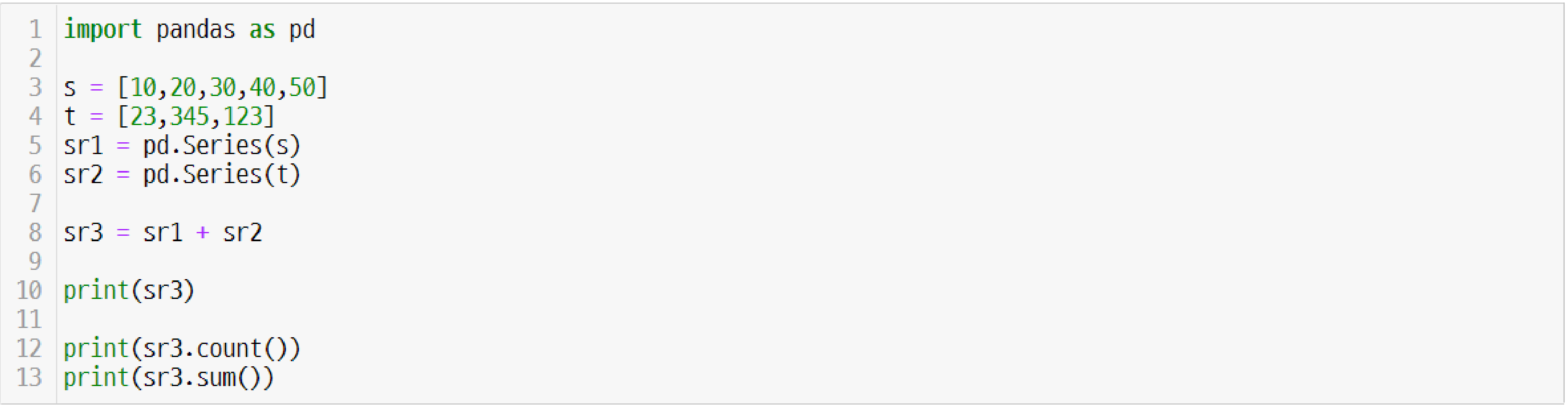
- Check how each element value changes from the addition of s1 and s2.

10. Pandas Provides Many Accessible Descriptive Statistics

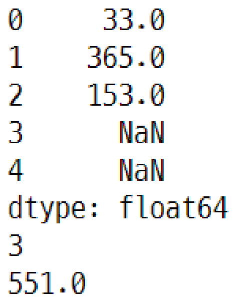
The descriptive statistics methods provided below are both used for Series objects and DataFrame objects.

count()	Number of non-null observations
sum()	Sum of values
mean()	Mean of Values
median()	Median of Values
mode()	Mode of values
std()	Standard Deviation of the Values
min()	Minimum Value
Max()	Maximum Value
Abs()	Absolute Value
prod()	Product of Values
cumsum()	Cumulative Sum
cumprod()	Cumulative Product

10. Pandas Provides Many Accessible Descriptive Statistics



```
 1  import pandas as pd
 2
 3  s = [10,20,30,40,50]
 4  t = [23,345,123]
 5  sr1 = pd.Series(s)
 6  sr2 = pd.Series(t)
 7
 8  sr3 = sr1 + sr2
 9
10  print(sr3)
11
12  print(sr3.count())
13  print(sr3.sum())
```

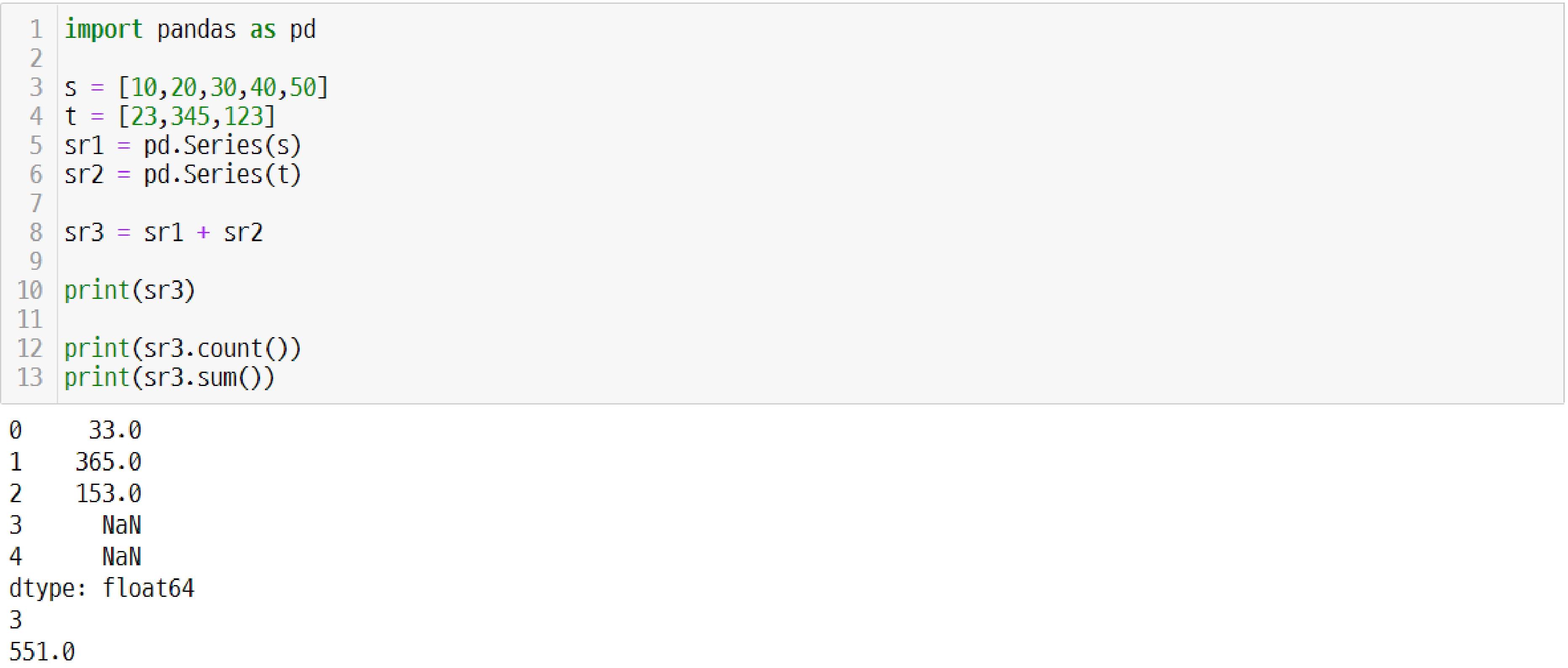


```
0     33.0
1    365.0
2    153.0
3      NaN
4      NaN
dtype: float64
3
551.0
```

Line 10

- When printing, the elements of indices 3 and 4 are NAN data.

10. Pandas Provides Many Accessible Descriptive Statistics



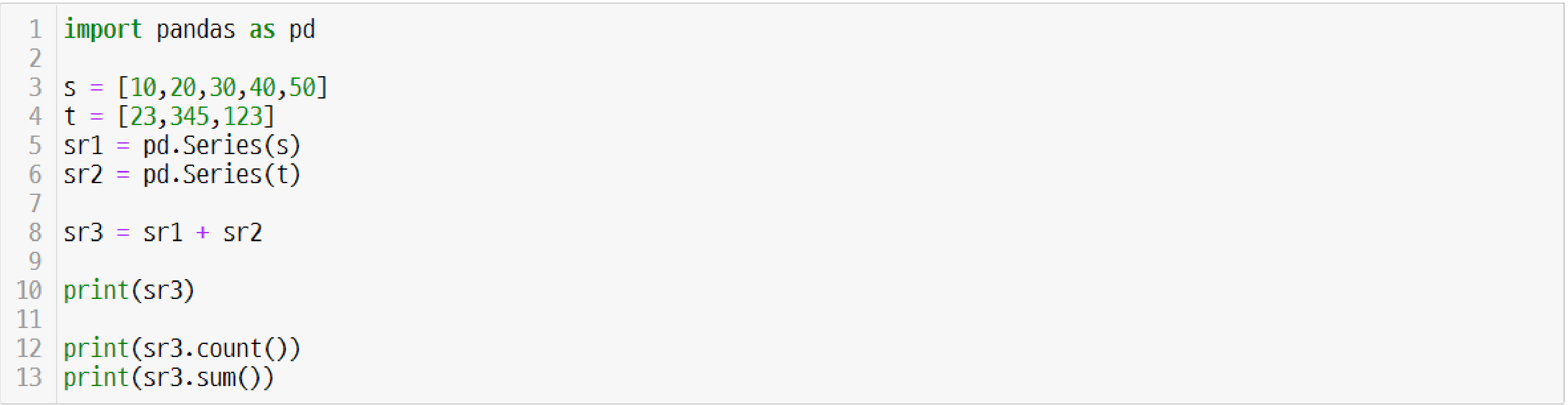
```
 1  import pandas as pd
 2
 3  s = [10,20,30,40,50]
 4  t = [23,345,123]
 5  sr1 = pd.Series(s)
 6  sr2 = pd.Series(t)
 7
 8  sr3 = sr1 + sr2
 9
10  print(sr3)
11
12  print(sr3.count())
13  print(sr3.sum())
```

```
0     33.0
1    365.0
2    153.0
3      NaN
4      NaN
dtype: float64
3
551.0
```

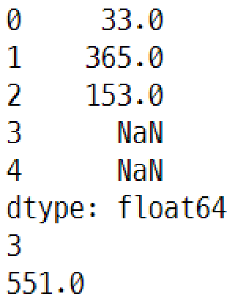
Line 12

- Returns the number of elements except for missing data.

10. Pandas Provides Many Accessible Descriptive Statistics



```
 1  import pandas as pd
 2
 3  s = [10,20,30,40,50]
 4  t = [23,345,123]
 5  sr1 = pd.Series(s)
 6  sr2 = pd.Series(t)
 7
 8  sr3 = sr1 + sr2
 9
10  print(sr3)
11
12  print(sr3.count())
13  print(sr3.sum())
```



```
0     33.0
1    365.0
2    153.0
3      NaN
4      NaN
dtype: float64
3
551.0
```

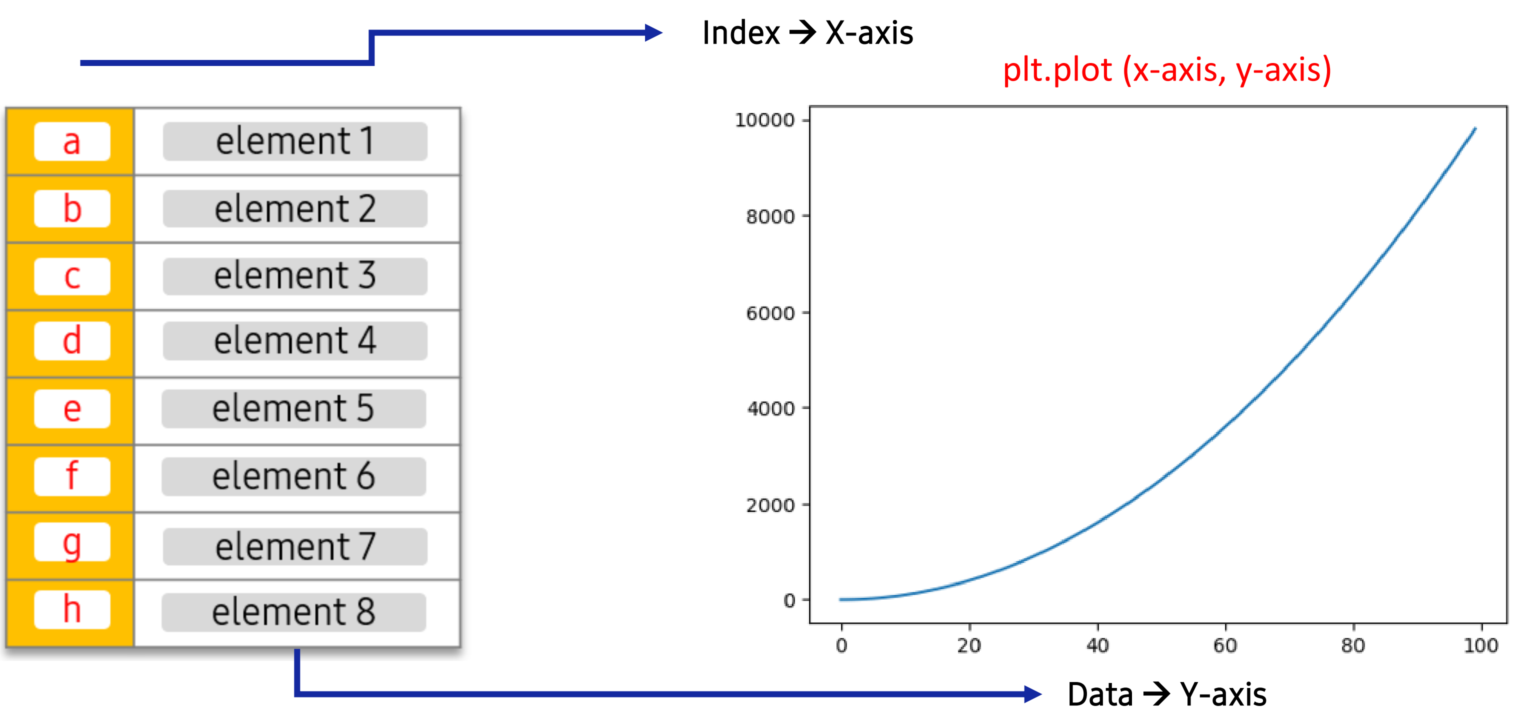
Line 13

- Returns sum of the elements except for missing data.

11. Basic of Data Visualization Tool Matploylib

11.1. Drawing a line plot

- Assigning the plt.plot (x-axis and y-axis data) passes series index to x-axis and data element of each index to y-axis.
- Or, drawing a graph is also possible with the plt.plot (Series object or DataFrame object).



Index → X-axis

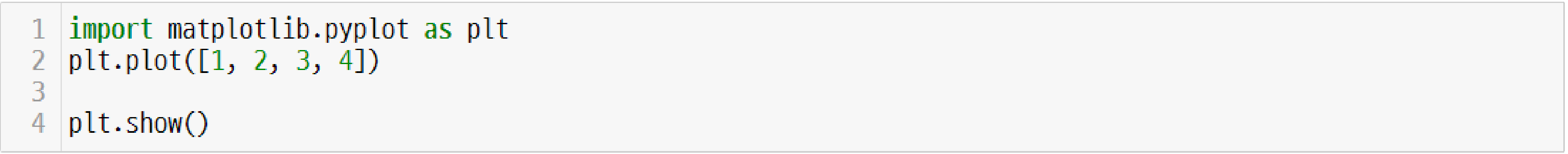
plt.plot (x-axis, y-axis)

a	element 1
b	element 2
c	element 3
d	element 4
e	element 5
f	element 6
g	element 7
h	element 8

Data → Y-axis

11. Basic of Data Visualization Tool Matploylib

11.1. Drawing a line plot



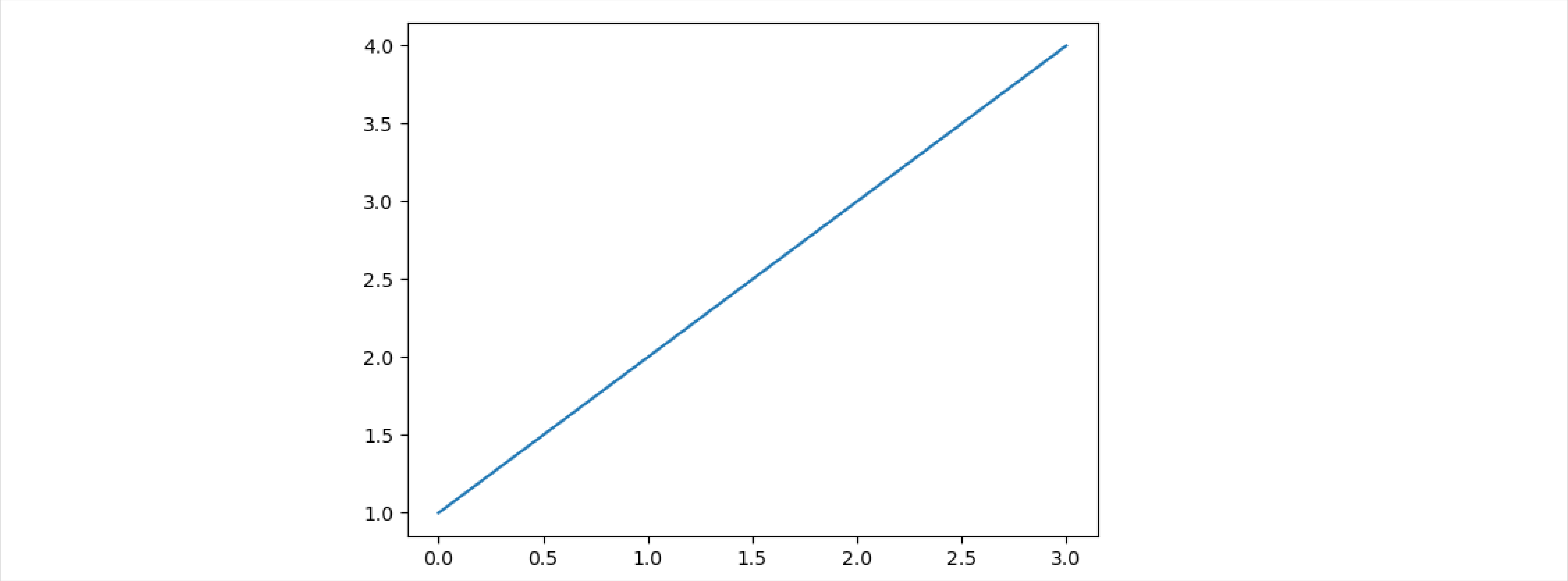
```
1 import matplotlib.pyplot as plt
2 plt.plot([1, 2, 3, 4])
3 
4 plt.show()
```

Line 2

- If not assigning separate x values, this value is assigned to the sequence of y-axis.

11. Basic of Data Visualization Tool Matploylib

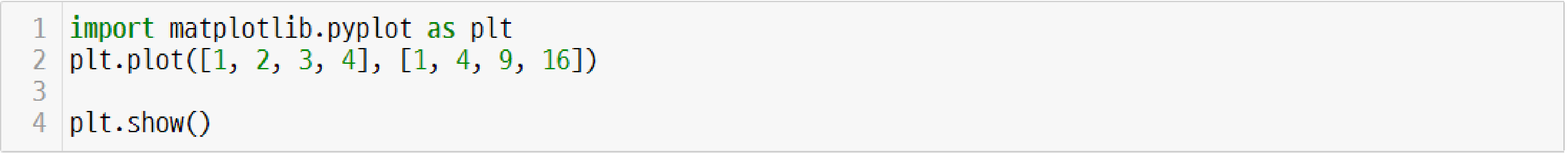
11.1. Drawing a line plot



- From the chart above, x-axis is 0-3 and y-axis is 1-4. This is because when providing a single list or array for floating, matplotlib assumes that it is the sequence of y value and automatically generates x value.
- Python range starts from 0, so the basic x vector has the same length with y, but it starts from 0. Thus, the x data become [0, 1, 2, 3].

11. Basic of Data Visualization Tool Matploylib

11.1. Drawing a line plot



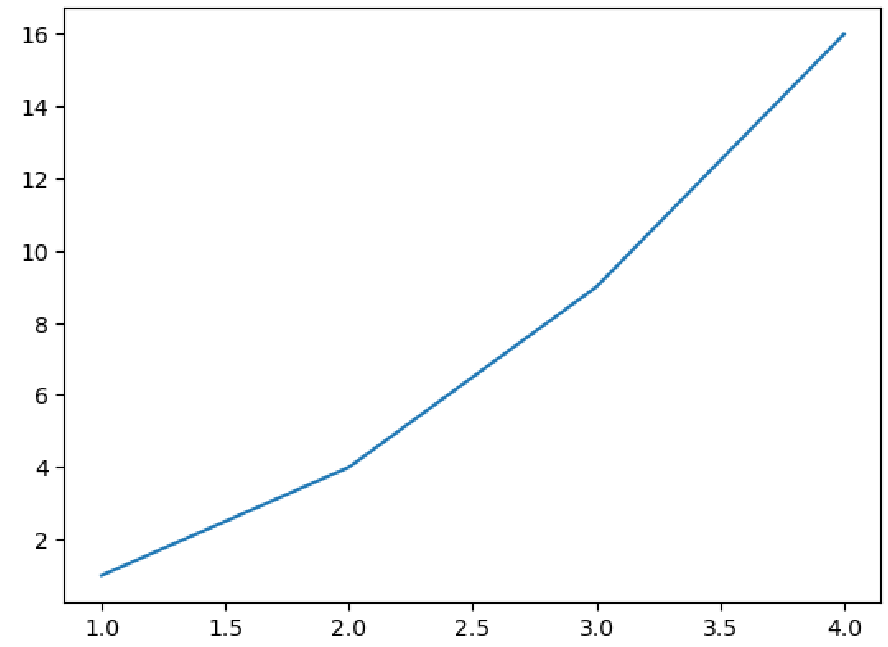
```
1 import matplotlib.pyplot as plt
2 plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
3
4 plt.show()
```

Line 2

- Assigns both x-axis and y-axis

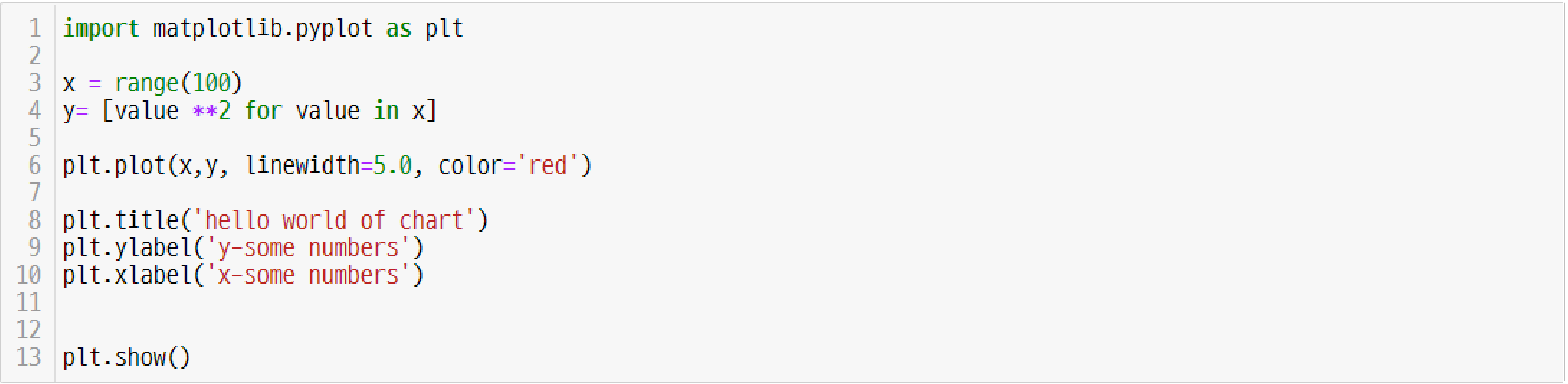
11. Basic of Data Visualization Tool Matploylib

11.1. Drawing a line plot



11. Basic of Data Visualization Tool Matploylib

11.1. Drawing a line plot



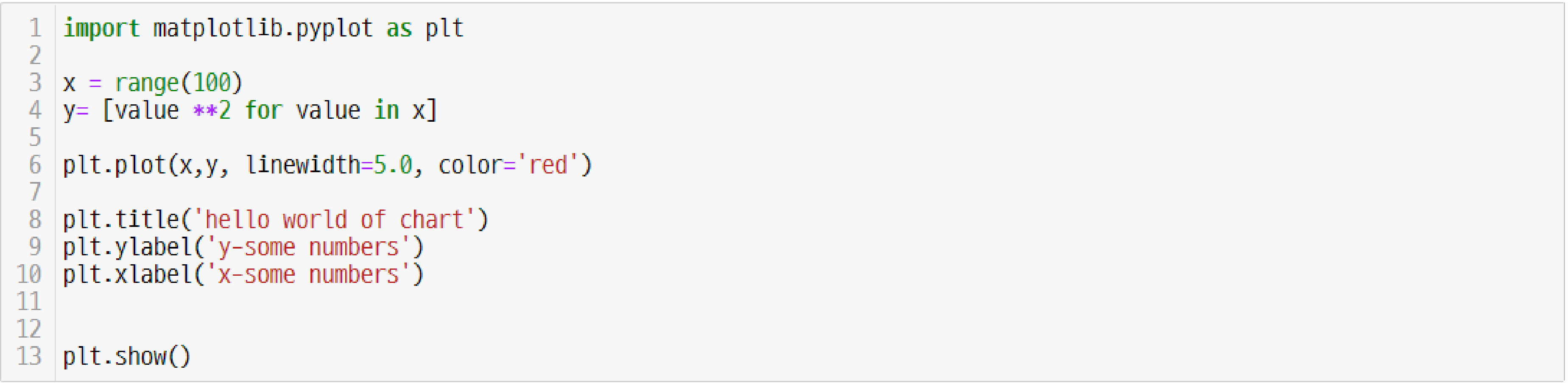
```
 1 import matplotlib.pyplot as plt
 2 
 3 x = range(100)
 4 y= [value **2 for value in x]
 5 
 6 plt.plot(x,y, linewidth=5.0, color='red')
 7 
 8 plt.title('hello world of chart')
 9 plt.ylabel('y-some numbers')
10 plt.xlabel('x-some numbers')
11 
12 
13 plt.show()
```

Line 1, 4, 6, 8

- 1: matplotlib is an external module that needs to be installed.
- 4: uses list comprehension.
- 6: linewidth is a property that adjusts the thickness of a line graph, while color designates specific colors.
- 8: adds the title of the chart.

11. Basic of Data Visualization Tool Matploylib

11.1. Drawing a line plot



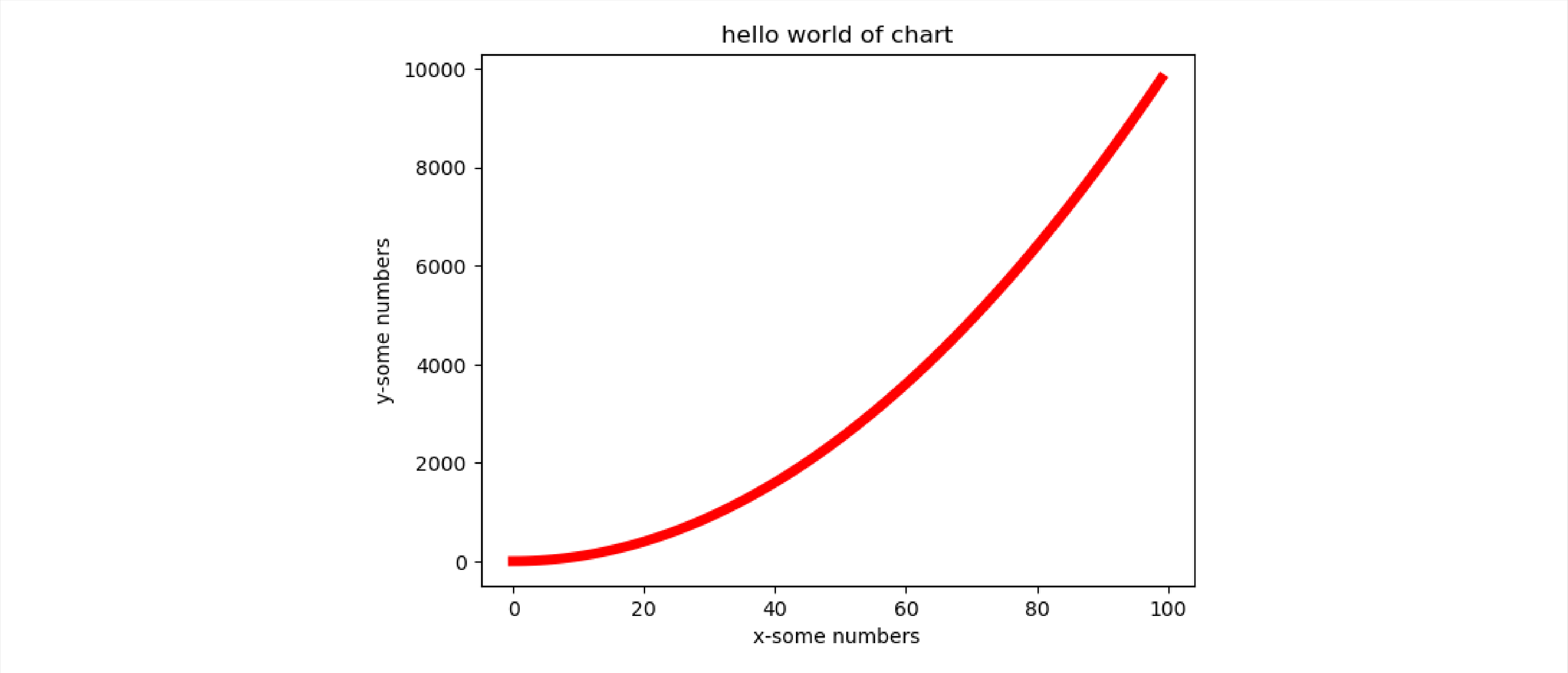
```
1  import matplotlib.pyplot as plt
2
3  x = range(100)
4  y= [value **2 for value in x]
5
6  plt.plot(x,y, linewidth=5.0, color='red')
7
8  plt.title('hello world of chart')
9  plt.ylabel('y-some numbers')
10 plt.xlabel('x-some numbers')
11
12
13 plt.show()
```

Line 9, 10, 13

- 9: adds the name of x-axis of the chart.
- 10: adds the name of y-axis of the chart.
- 13: command function to print a graph.

11. Basic of Data Visualization Tool Matploylib

11.1. Drawing a line plot



- It is result of drawing y = x **2 curve where x is between [0,99].

11. Basic of Data Visualization Tool Matploylib

11.2. Drawing a bar plot

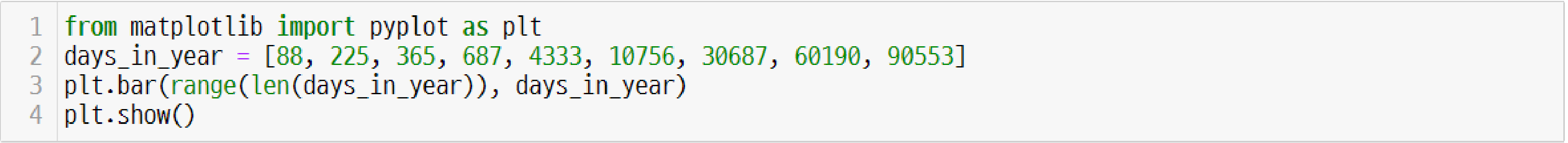
A bar plot represents the height proportional to data value size as a rectangular bar.
It is possible to compare the data size through the relative difference between bar height, and there are two different bar types including vertical and horizontal bar.

Vertical bar plt.bar()	Horizontal bar plt.barh()
- Suitable to show differences between data values of two points that have time difference. - In other words, it is suitable for time series data expression.	- Suitable so show differences between variable value sizes.

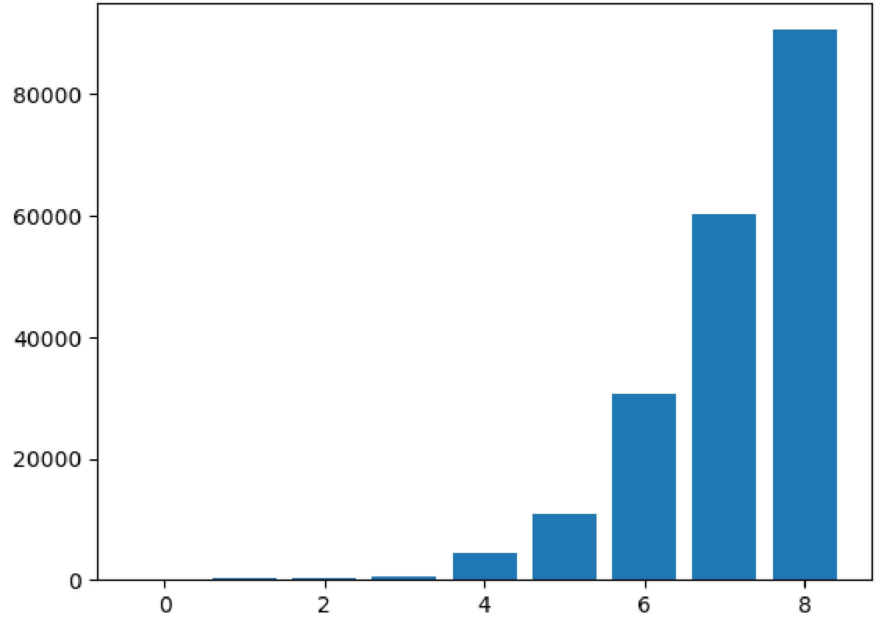
11. Basic of Data Visualization Tool Matploylib

11.2. Drawing a bar plot

- If the elements of each index are not matched, the operation result returns NaN.



```python
from matplotlib import pyplot as plt
days_in_year = [88, 225, 365, 687, 4333, 10756, 30687, 60190, 90553]
plt.bar(range(len(days_in_year)), days_in_year)
plt.show()
```

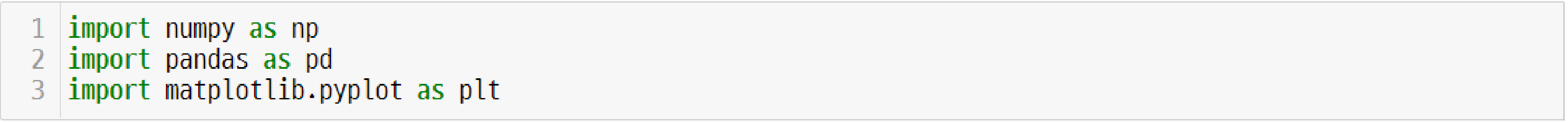


Let’s code

Step 1

1) Consider which libraries would be required to solve the mission. Find necessary libraries and install ones that you don't have. Import the module to the code.

- Numpy for making an array while data processing
- Pandas for creating series objects
- Matplotlib for data visualization

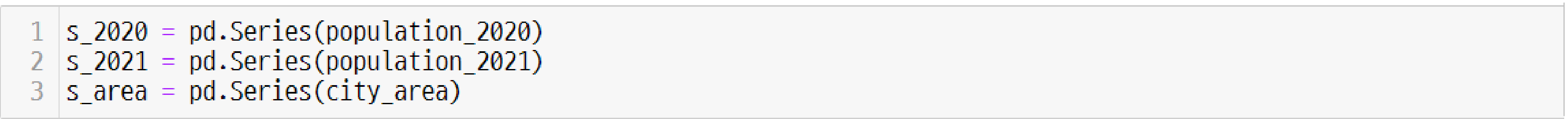


```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Step 1

2) Convert each of three dictionaries into a series object.

- Make sure to check the index label of each series object for operation because if index label or index number (number of data) is different, it may return NaN.



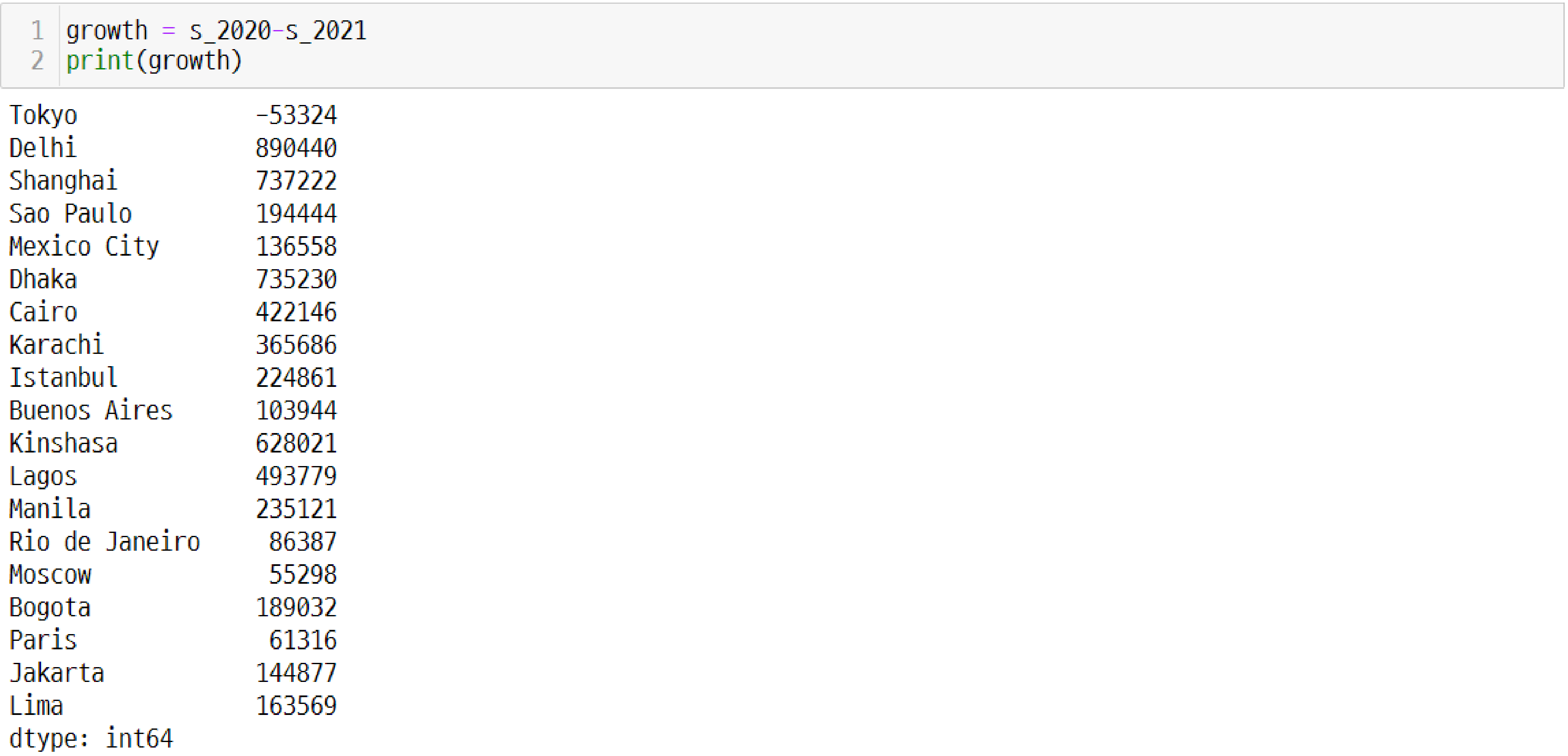
```
1 s_2020 = pd.Series(population_2020)
2 s_2021 = pd.Series(population_2021)
3 s_area = pd.Series(city_area)
```

Line 1~3

- 1: population_2020 is dictionary data object that stores total population of each city in 2020.
- 2: population_2021 is dictionary data object that stores total population of each city in 2021.
- 3: city_area is dictionary data object that stores area of each city.

Step 2

Use two series arithmetic operations to calculate population fluctuation of each year.

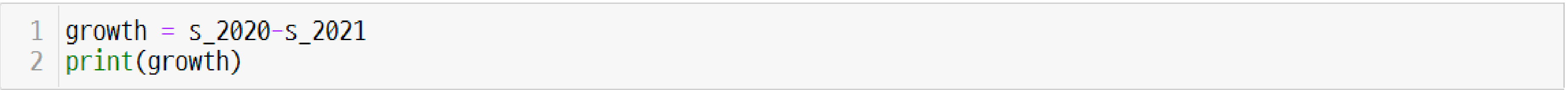


```python
growth = s_2020-s_2021
print(growth)
```

```
Tokyo             -53324
Delhi             890440
Shanghai          737222
Sao Paulo         194444
Mexico City       136558
Dhaka             735230
Cairo             422146
Karachi           365686
Istanbul          224861
Buenos Aires      103944
Kinshasa          628021
Lagos             493779
Manila            235121
Rio de Janeiro     86387
Moscow             55298
Bogota            189032
Paris              61316
Jakarta           144877
Lima              163569
dtype: int64
```

Step 2

Use two series arithmetic operations to calculate population fluctuation of each year.



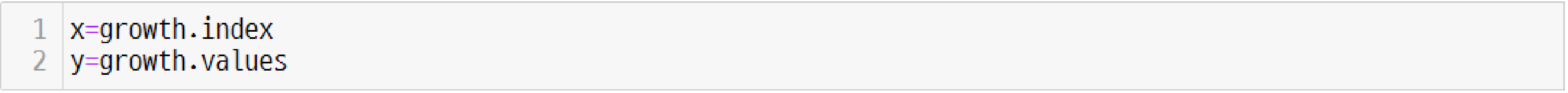
```
1 growth = s_2020-s_2021
2 print(growth)
```

Line 1~2

- 1: Creates a new series object by calculating population fluctuation of each city (index label) through two series arithmetic operations (-).
- 2: Checks data of new series object that stores arithmetic operation results.

Step 3

- As previously explained, when visualizing data through a bar graph, vertical graph is suitable for time series data that have consecutive values, but horizontal bar graph is suitable here because it would show data difference between each variable (city name).
- The x-axis of the graph is index label of time series object, and the y-axis consists of data elements.
- To draw a chart of time series object of 'growth,' save index information to x-axis and data values to y-axis.



```
1 x=growth.index
2 y=growth.values
```

Line 1~2

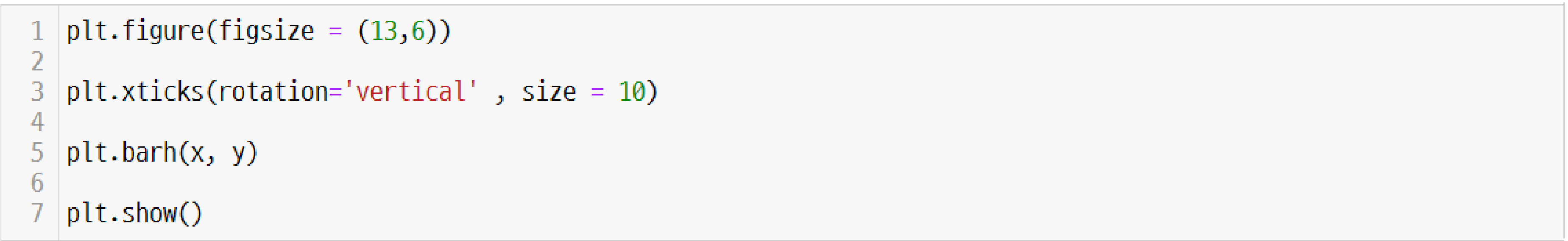
- 1: Extracts index value only.
- 2: Extracts data elements only.

Step 3

- When creating a bar graph, overlapping sometime occurs if there are many label names on x-axis. To solve this problem, adjust the graph size to clearly see label names on x-axis.
- Use the plt.figure() function, figsize =(horizontal, vertical size), and parameters in inch to adjust the chart size. If not designating the size, the default chart size becomes 6.4 and 4.8 inches.
- Use the plt.xticks( ) function to adjust the angle of label name printing direction.

- rotation = enters an angle that rotates the number counterclockwise.
- size = font size

Step 3

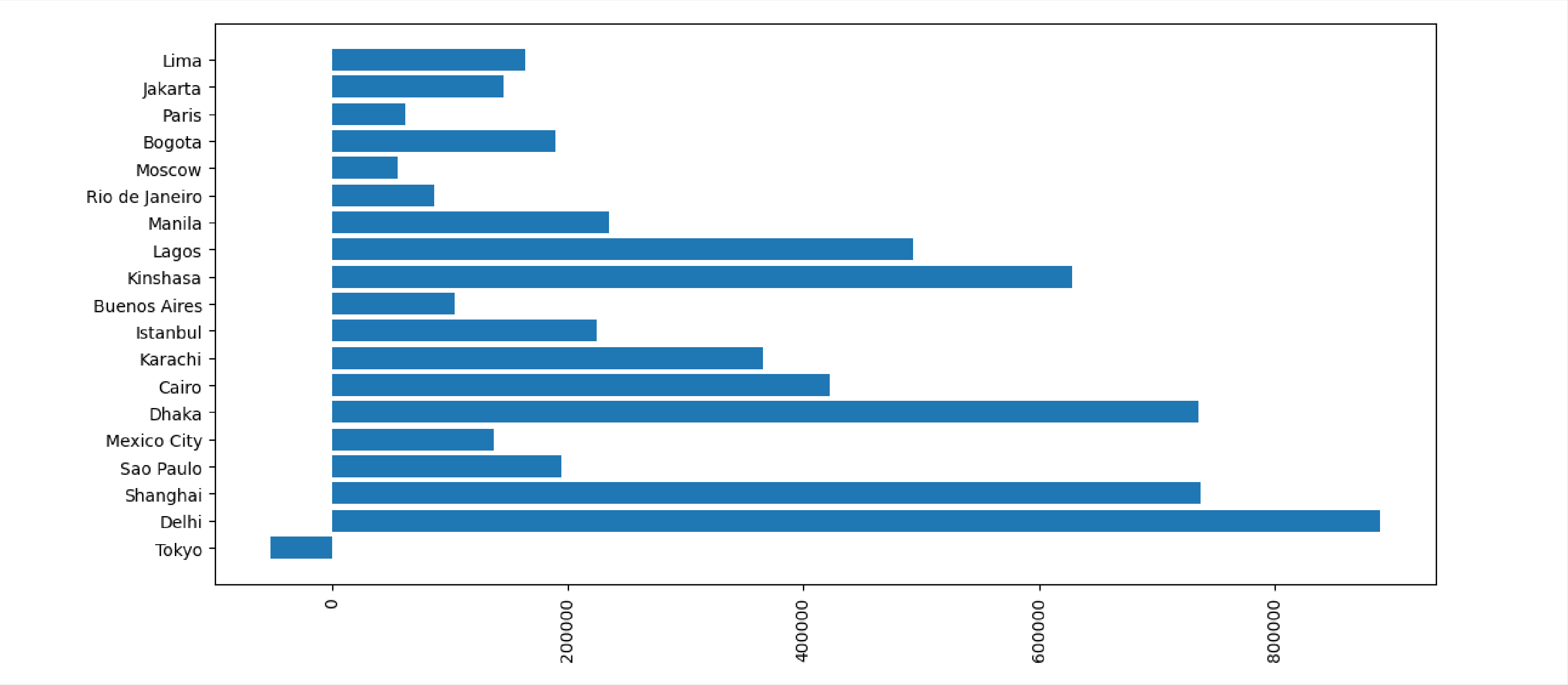


```
1 plt.figure(figsize = (13,6))
2 
3 plt.xticks(rotation='vertical' , size = 10)
4 
5 plt.barh(x, y)
6 
7 plt.show()
```

Line 1, 3, 4, 5

- 1: Figure dimension (width, height) in inches.
- 3: It is possible to enter a number that signifies an angle instead of vertical. If rotation=90, it means that it rotates 90 degrees counterclockwise.
- 4: size=10 refers to font size.
- 5: Use plt.barh() to plot a horizontal bar plot.

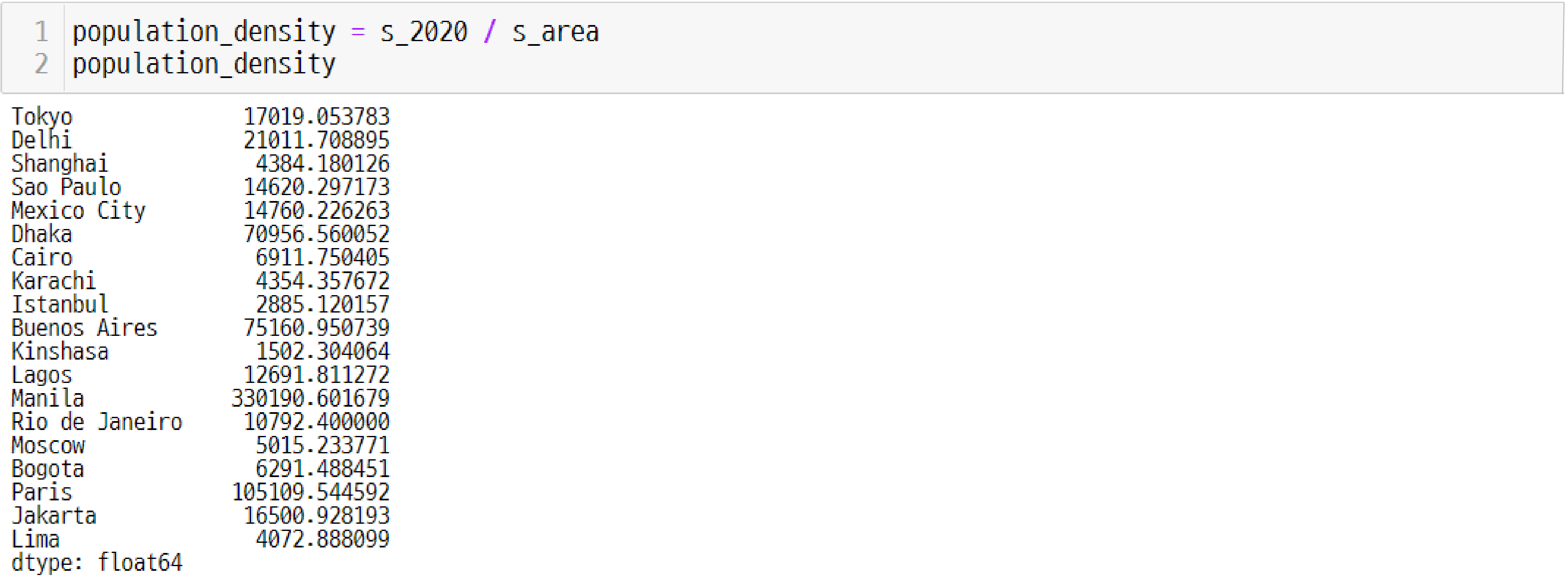
Step 3



- When calculating data with cities with the largest population, it shows that population growth rate is decreased in Tokyo.

Step 4

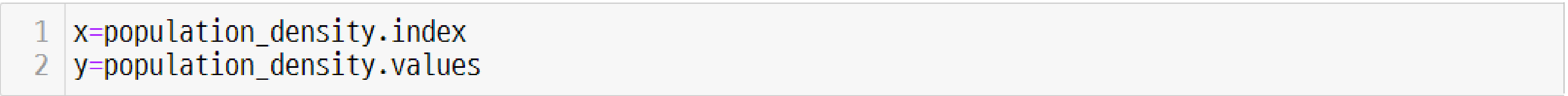
- Let's calculate population density.
- The series objects for calculation include s_2020 that has population data in 2020 and s_area that has area data of each city.



```python
population_density = s_2020 / s_area
population_density
```

```
Tokyo              17019.053783
Delhi              21011.708895
Shanghai            4384.180126
Sao Paulo          14620.297173
Mexico City        14760.226263
Dhaka              70956.560052
Cairo               6911.750405
Karachi             4354.357672
Istanbul            2885.120157
Buenos Aires       75160.950739
Kinshasa            1502.304064
Lagos              12691.811272
Manila            330190.601679
Rio de Janeiro     10792.400000
Moscow              5015.233771
Bogota              6291.488451
Paris             105109.544592
Jakarta            16500.928193
Lima                4072.888099
dtype: float64
```

Step 4

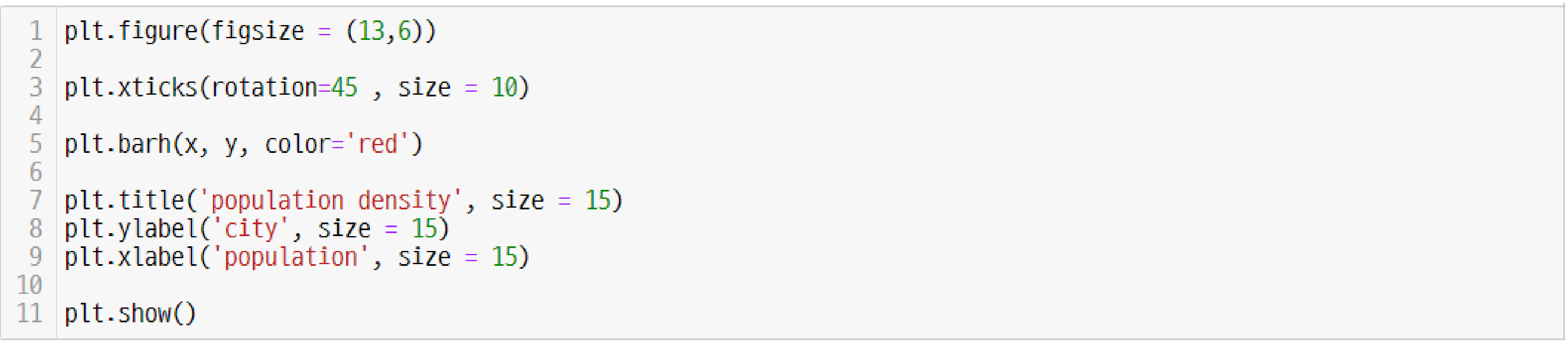


```
1 x=population_density.index
2 y=population_density.values
```

Line 1~2

- 1: Extracts index values only.
- 2: Extracts data elements only.

Step 4

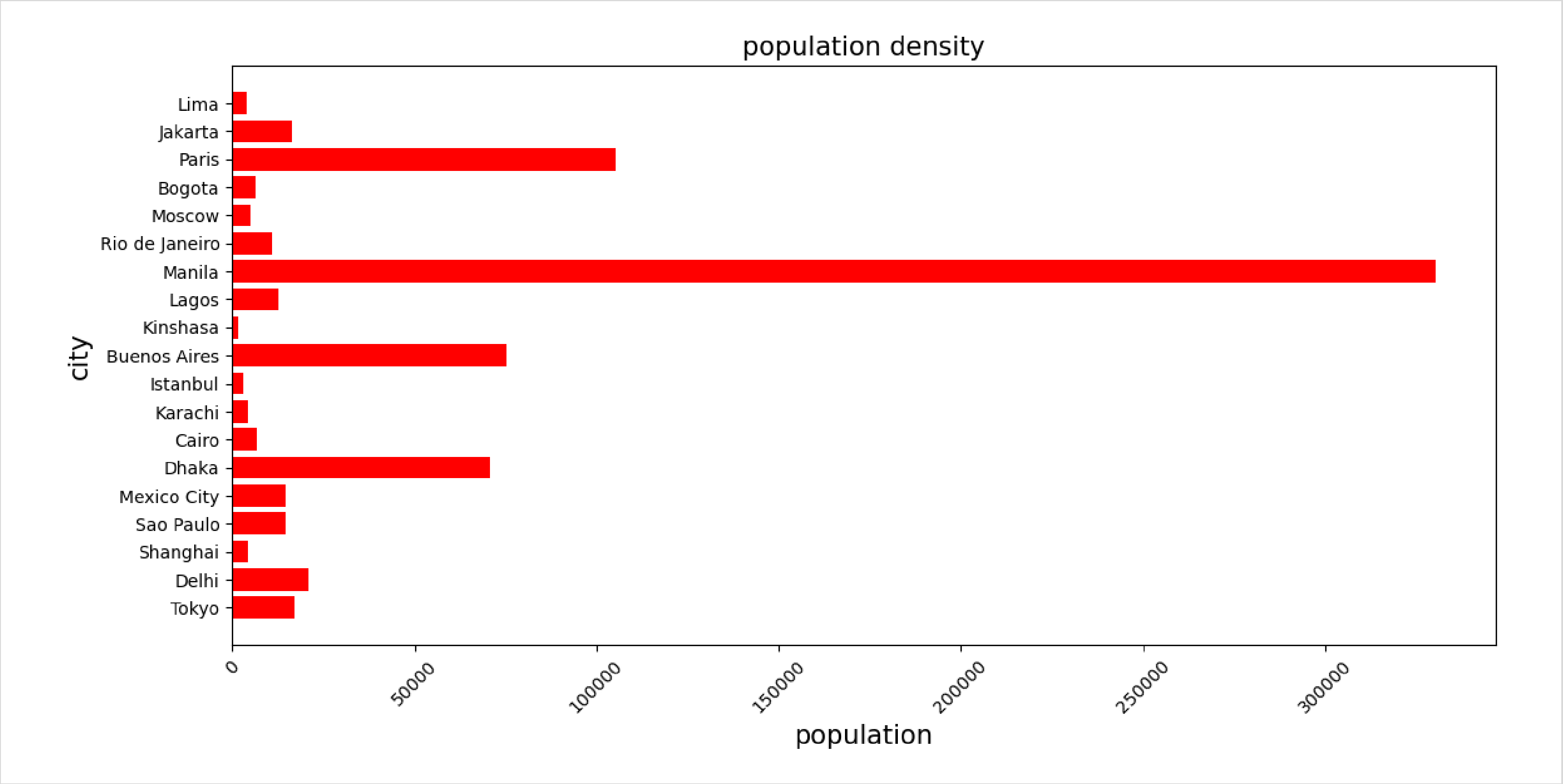


```
 1 plt.figure(figsize = (13,6))
 2
 3 plt.xticks(rotation=45 , size = 10)
 4
 5 plt.barh(x, y, color='red')
 6
 7 plt.title('population density', size = 15)
 8 plt.ylabel('city', size = 15)
 9 plt.xlabel('population', size = 15)
10
11 plt.show()
```

Line 1, 5, 7, 8, 9

- 1: Figure dimension (width, height) in inches.
- 5: Use color parameter to designate color of the graph.
- 7: Adds the title of the chart.
- 8: Adds the name of x-axis of the chart.
- 9: Adds the name of y-axis of the chart.

Step 4


population density
city
Lima
Jakarta
Paris
Bogota
Moscow
Rio de Janeiro
Manila
Lagos
Kinshasa
Buenos Aires
Istanbul
Karachi
Cairo
Dhaka
Mexico City
Sao Paulo
Shanghai
Delhi
Tokyo
0
50000
100000
150000
200000
250000
300000
population

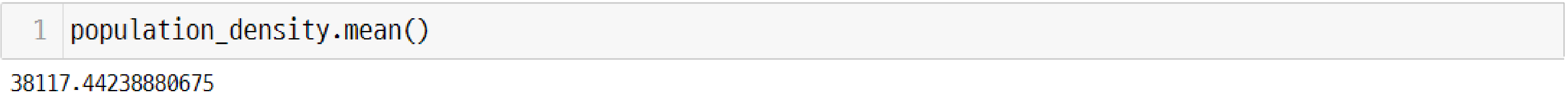
Step 5

- Calculate the cities with the highest and lowest population density by using the descriptive statistics functions of the pandas series object.

count()	Number of non-null observations
sum()	Sum of values
mean()	Mean of Values
median()	Median of Values
mode()	Mode of values
std()	Standard Deviation of the Values
min()	Minimum Value
Max()	Maximum Value
Abs()	Absolute Value
prod()	Product of Values
cumsum()	Cumulative Sum
cumprod()	Cumulative Product

Step 5

Calculate the cities with the highest and lowest population density by using the descriptive statistics functions of the pandas series object.

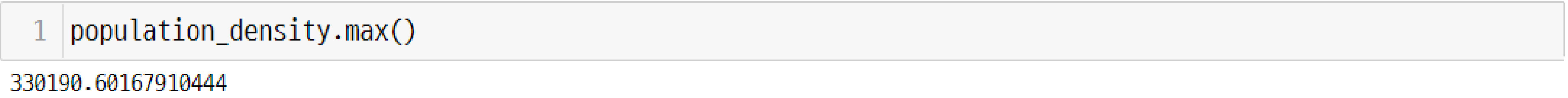


```
1  population_density.mean()
```

```
38117.44238880675
```

Line 1

- Finds the minimum value among the entire data element values of the series object



```
1  population_density.max()
```

```
330190.60167910444
```

Line 1

- Finds the maximum value among the entire data element values of the series object

Pair programming

Pair Programming Practice

Guideline, mechanisms & contingency plan

Preparing pair programming involves establishing guidelines and mechanisms to help students pair properly and to keep them paired. For example, students should take turns “driving the mouse.” Effective preparation requires contingency plans in case one partner is absent or decides not to participate for one reason or another. In these cases, it is important to make it clear that the active student will not be punished because the pairing did not work well.

Pairing similar, not necessarily equal, abilities as partners

Pair programming can be effective when students of similar, though not necessarily equal, abilities are paired as partners. Pairing mismatched students often can lead to unbalanced participation. Teachers must emphasize that pair programming is not a “divide-and-conquer” strategy, but rather a true collaborative effort in every endeavor for the entire project. Teachers should avoid pairing very weak students with very strong students.

Motivate students by offering extra incentives

Offering extra incentives can help motivate students to pair, especially with advanced students. Some teachers have found it helpful to require students to pair for only one or two assignments.

Pair Programming Practice

Prevent collaboration cheating

The challenge for the teacher is to find ways to assess individual outcomes, while leveraging the benefits of collaboration. How do you know whether a student learned or cheated? Experts recommend revisiting course design and assessment, as well as explicitly and concretely discussing with the students on behaviors that will be interpreted as cheating. Experts encourage teachers to make assignments meaningful to students and to explain the value of what students will learn by completing them.

Collaborative learning environment

A collaborative learning environment occurs anytime an instructor requires students to work together on learning activities. Collaborative learning environments can involve both formal and informal activities and may or may not include direct assessment. For example, pairs of students work on programming assignments; small groups of students discuss possible answers to a professor’s question during lecture; and students work together outside of class to learn new concepts. Collaborative learning is distinct from projects where students “divide and conquer.” When students divide the work, each is responsible for only part of the problem solving and there are very limited opportunities for working through problems with others. In collaborative environments, students are engaged in intellectual talk with each other.

Q1. The Spotify dataset has various column values other than the artist name, ranking, and popularity that we used in the mission. Discuss with your classmate to create a special playlist that is made with other column values.