WMASDS-04: Introduction to Data Science with Python

Week 02: Python Libraries

Fundamental Python Libraries for Data Scientists

- NumPy,
  - ndarray
- Pandas,
  - Series
  - DataFrame
- Matplotlib
- Seaborn
- Scikit-Learn
- Seaborn
- SciPy,
- TensorFlow
- Keras

Numpy – Fundamental Scientific Computing

- NumPy is a fundamental package for scientific computing in Python.
- It offers tools for working with multi-dimensional arrays and matrices.
- It is helpful for mathematical functions and statistical computations for data science tasks.
- NumPy also has advanced indexing and selection capabilities, as well as broadcasting capabilities for arithmetic and logical operations on arrays with different shapes.

Key Features of NumPy

- Mathematical functions, including linear algebra and Fourier transforms
- Tools for working with polynomials, random numbers, and statistical distributions
- Advanced indexing and selection capabilities
- Broadcasting capabilities for arithmetic and logical operations on arrays with different shapes
- Capability to interface with C and Fortran code

Applications

- Extensively used in data analysis
- Creates powerful N-dimensional array
- Forms the base of other libraries, such as SciPy and scikit-learn
- Replacement of MATLAB when used with SciPy and matplotlib

Pros	Cons
Efficient for numerical operations on large arrays	Limited support for distributed computing
Provides support for linear algebra, Fourier analysis, and random number generation	Steep learning curve for beginners
Interoperable with other scientific computing libraries	Limited support for higher-level data analysis tasks
Large and active user community	Less convenient for working with structured data

Pandas – Data Manipulation and Analysis

- Pandas is a library for data manipulation and evaluation in Python.
- It offers data structures for storing and processing large information sets, in addition to tools for merging, joining, and reshaping data.
- The library has time-series capabilities and the capacity to handle empty records.
- Pandas is important for data training and analysis duties for data science projects.

Key Features of Pandas

- Provides data structures for efficient handling of structured data, including Series, DataFrame, and Panel
- Offers tools for data cleaning, merging, and reshaping, including pivot tables and slicing and indexing tools
- Enables integration with other data science libraries, including Matplotlib and Scikit-Learn
- Time-series functionality

Applications

- General data wrangling and data cleaning
- ETL (extract, transform, load) jobs for data transformation and data storage, as it has excellent support for loading CSV files into its data frame format
- Used in a variety of academic and commercial areas, including statistics, finance and neuroscience
- Time-series-specific functionality, such as date range generation, moving window, linear regression and date shifting.

Pandas

Pros	Cons
Provides powerful and flexible data manipulation capabilities	Can be slow on large datasets
Enables handling of structured and tabular data	Steep learning curve for beginners
Offers easy data cleaning, filtering, and transformation	Limited support for time series and machine learning tasks
Provides seamless integration with other data analysis libraries	Requires some understanding of data structures and manipulation

Matplotlib – Plotting and Visualization

- Matplotlib is a favored data visualization Python library that allows data scientists to create plots and charts, from simple line plots to complex 3D visualizations.
- It is an important library to add to a data science toolkit for creating informative visualizations for data science projects.
- Matplotlib is built atop NumPy and integrates seamlessly with other Python data analysis libraries like Pandas, providing data scientists with all of the flexibility and control they require to create high-quality visualizations.

Key Features of Matplotlib

- Provides a wide range of static, animated, and interactive visualization types, including scatter plots, line plots, bar charts, histograms, and more
- Enables customization of visualizations using a wide range of properties and settings
- Includes an object-oriented interface for creating and modifying visualizations

Applications

- Correlation analysis of variables
- Visualize 95 percent confidence intervals of the models
- Outlier detection using a scatter plot etc.
- Visualize the distribution of data to gain instant insights

Matplotlib

Pros	Cons
• Provides a wide range of visualization types and styles	Steep learning curve for beginners
Highly customizable and provides fine-grained control over visualizations	Can be slow on large datasets
Can handle large datasets and create complex visualizations	Limited support for interactive visualizations
Provides compatibility with other data analysis libraries	Can require more coding for complex visualizations

Scikit-Learn - Machine Learning and Data Mining

- Scikit-Learn is a staple for any data scientist who needs a library for machine learning. It comes equipped with built-in classifiers to help expedite your data science needs. Some of those classifiers include logistic regression, K-nearest neighbors, Decision trees, and more. It also has helpful tools like confusion matrices, classification reports, and feature extraction.

Key Features of Scikit-Learn

- Classification algorithms, including k-nearest neighbors, logistic regression, decision trees, and support vector machines
- Regression algorithms, including linear regression, ridge regression, and Lasso regression
- Clustering algorithms, including k-means clustering and hierarchical clustering
- Feature selection and dimensionality reduction algorithms
- Model selection and cross-validation tools

Applications

- clustering
- classification
- regression
- model selection
- dimensionality reduction

Scikit-Learn

Pros	Cons
Provides a wide range of machine learning algorithms	Limited support for deep learning tasks
Supports both supervised and unsupervised learning	Some algorithms may require hyperparameter tuning
Provides built-in tools for data preprocessing, model selection, and evaluation	Can be memory-intensive for large datasets
Offers easy integration with other data analysis libraries	May require some understanding of statistical concepts

Scipy – Fundamental Scientific Computing

- SciPy is a set of mathematical algorithms and convenient functions that are built on Python’s NumPy extension. It offers high-level commands and classes for manipulating and visualizing data, making it a powerful addition to the interactive Python session. Data scientists can benefit from using SciPy for tasks such as data optimization, integration, and statistical analysis.

Key Features of SciPy

- Provides a wide range of tools for scientific computing, including optimization, linear algebra, signal and image processing, and more
- Includes a range of routines for special functions, including gamma functions, Bessel functions, and more
- Offers integration with other data science libraries, including NumPy and Pandas
- Signal processing capabilities, including filtering and Fourier transforms
- Statistical testing and hypothesis testing tools

SciPy

Pros	**Cons**
Provides many scientific computing tools and algorithm options	Limited support for distributed computing
Offers a variety of modules for optimization, signal processing, interpolation, and more	Steep learning curve for beginners
Provides easy integration with other data analysis libraries	Some modules may require domain-specific knowledge
Large and active user community	May require some understanding of mathematical concepts

StatsModels – Statistical Modeling, Testing, and Analysis

- Statsmodels for statistical modeling. It is a Python module that allows users to explore data, estimate statistical models, and perform statistical tests. An extensive list of descriptive statistics, statistical tests, plotting functions, and result statistics are available for different types of data and each estimator.

Seaborn – For Statistical Data Visualization

- Seaborn for statistical data visualization. It is a library for making attractive and informative statistical graphics in Python. It is based on matplotlib. Seaborn aims to make visualization a central part of exploring and understanding data.

Keras

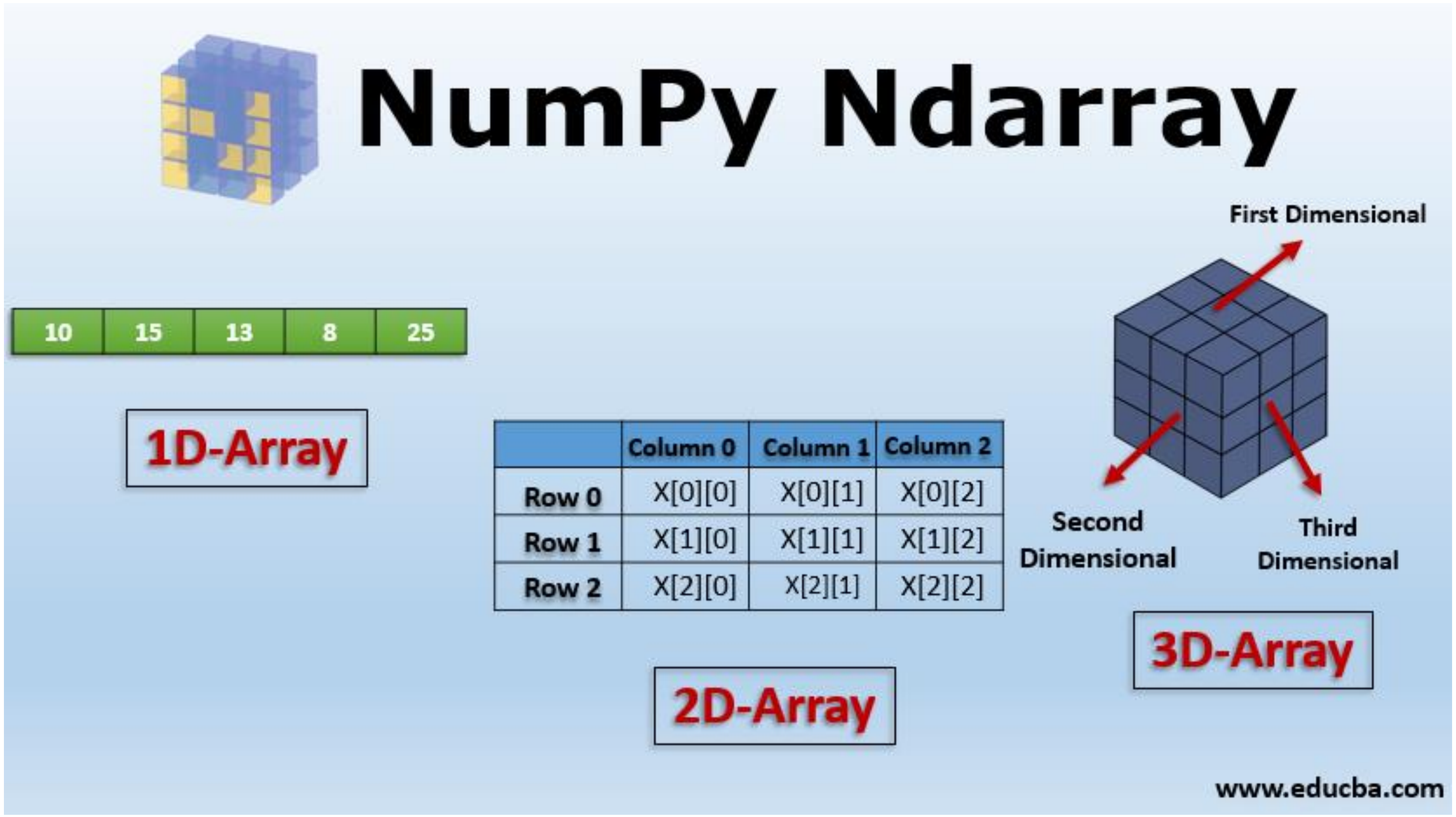
- Keras is a swell deep-learning library that’s open-source.
- It’s super user-friendly and makes it easy to create and train deep neural networks.
- Even for an inexperienced data scientist, Keras is flexible and extensible enough for anyone to use.
- Plus, it works seamlessly with other popular deep-learning frameworks like TensorFlow and Theano.
- With Keras, you can create all kinds of deep learning models, from CNNs to RNNs and beyond.
- It’s seriously powerful and perfect for creating complex models quickly.
- Applications:
- One of the most significant applications of Keras are the deep learning models that are available with their pretrained weights. You can use these models directly to make predictions or extract its features without creating or training your own new model.

TensorFlow

- Tensorflow is a neat open-source framework for machine learning. it allows data scientists to create graphs that show how data flows through various processing nodes. Each node represents a specific mathematical operation, and they're all connected by multidimensional data arrays known as tensors. it delivers a powerful platform for building, training, and deploying machine learning models at scale.
- Key Features
  - High-level API for creating and training deep neural networks
  - Pre-built neural network architectures for image and speech recognition
  - Support for reinforcement learning and generative models
- TensorFlow is particularly useful for the following applications:
  - Speech and image recognition
  - Text-based applications
  - Time-series analysis
  - Video detection

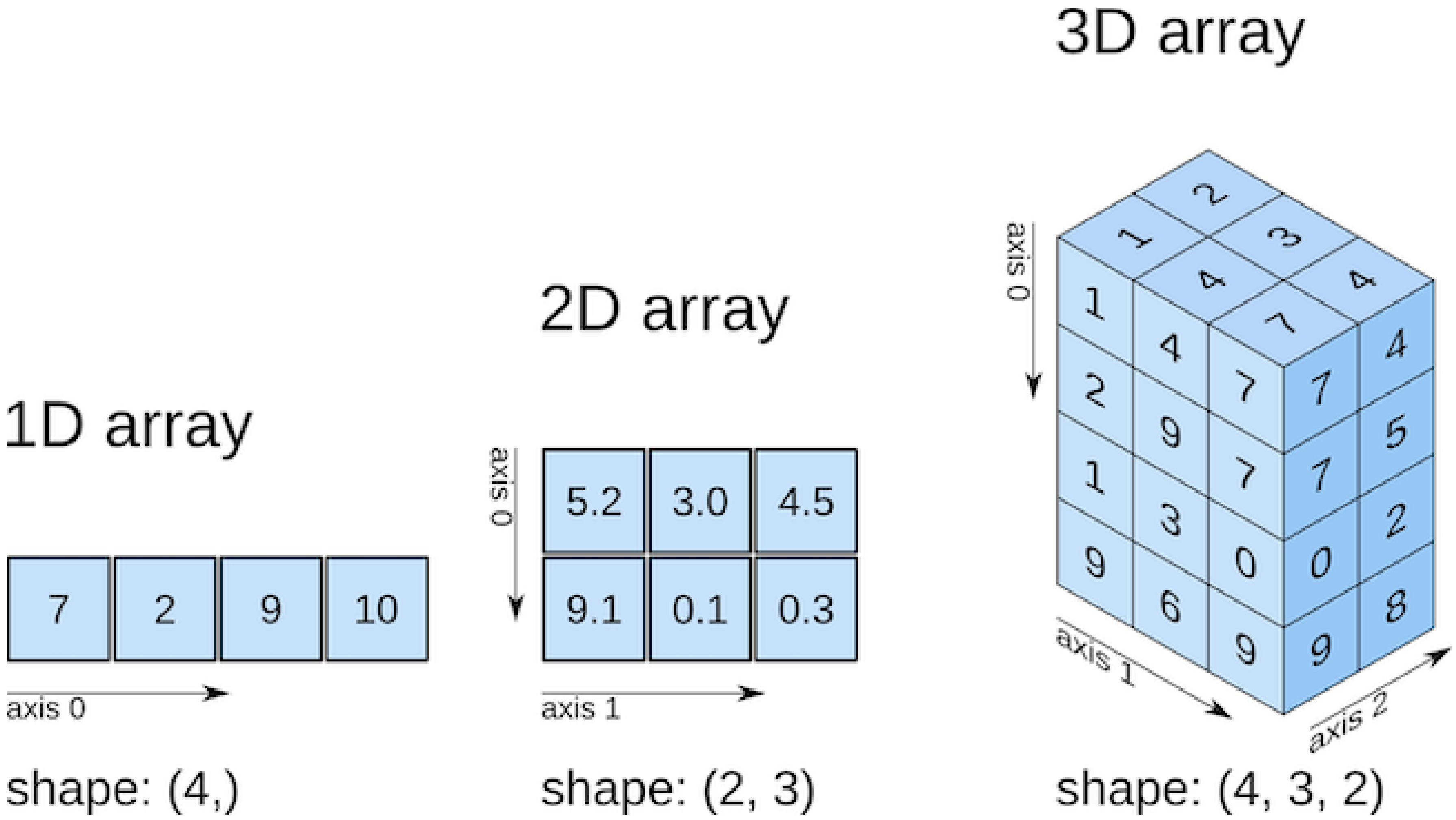
Data Structures in Python Libraries

- Numpy array
- Pandas Series
- Pandas DataFrame



	Column 0	Column 1	Column 2
Row 0	X[0][0]	X[0][1]	X[0][2]
Row 1	X[1][0]	X[1][1]	X[1][2]
Row 2	X[2][0]	X[2][1]	X[2][2]

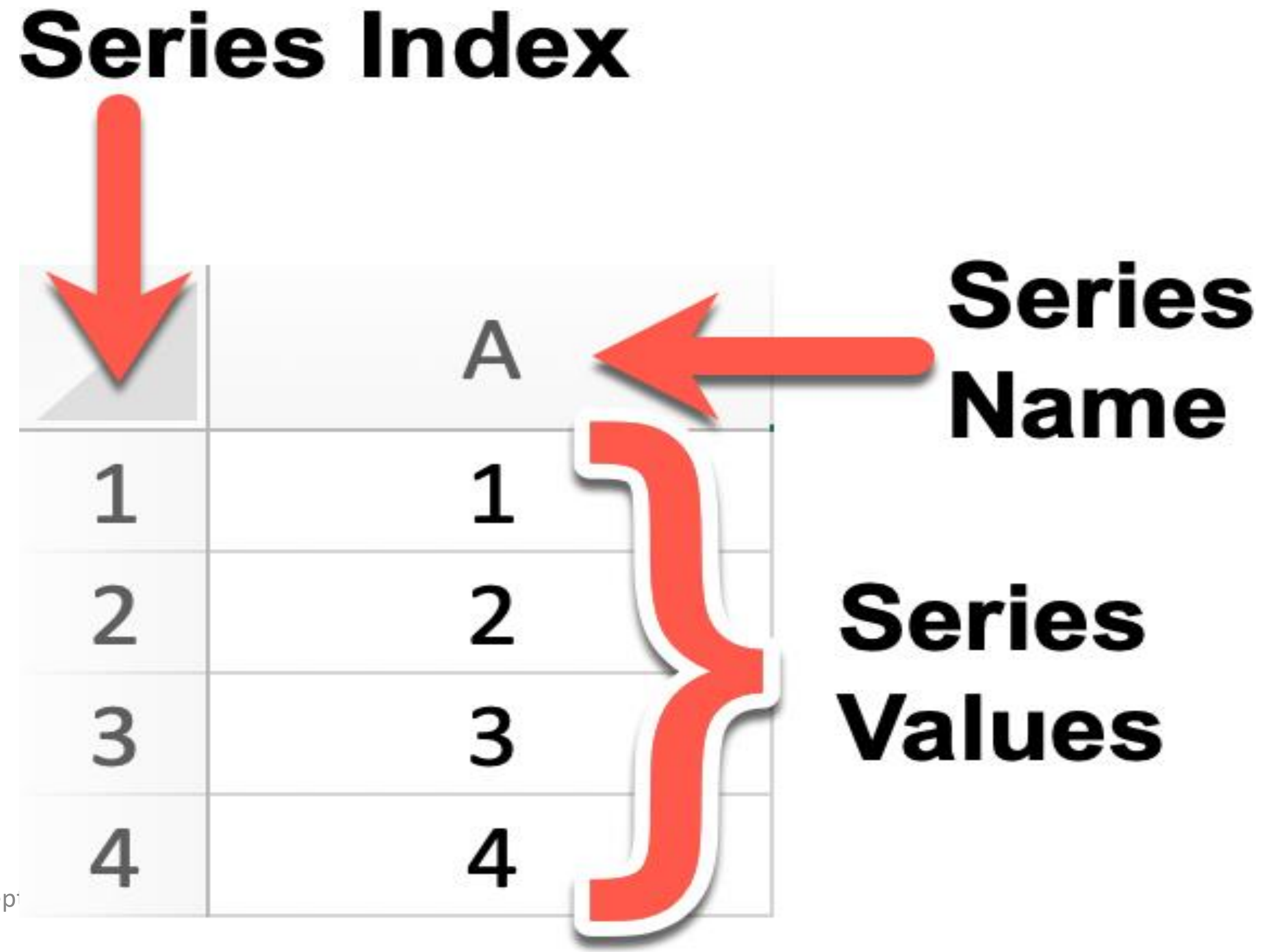
ndarray



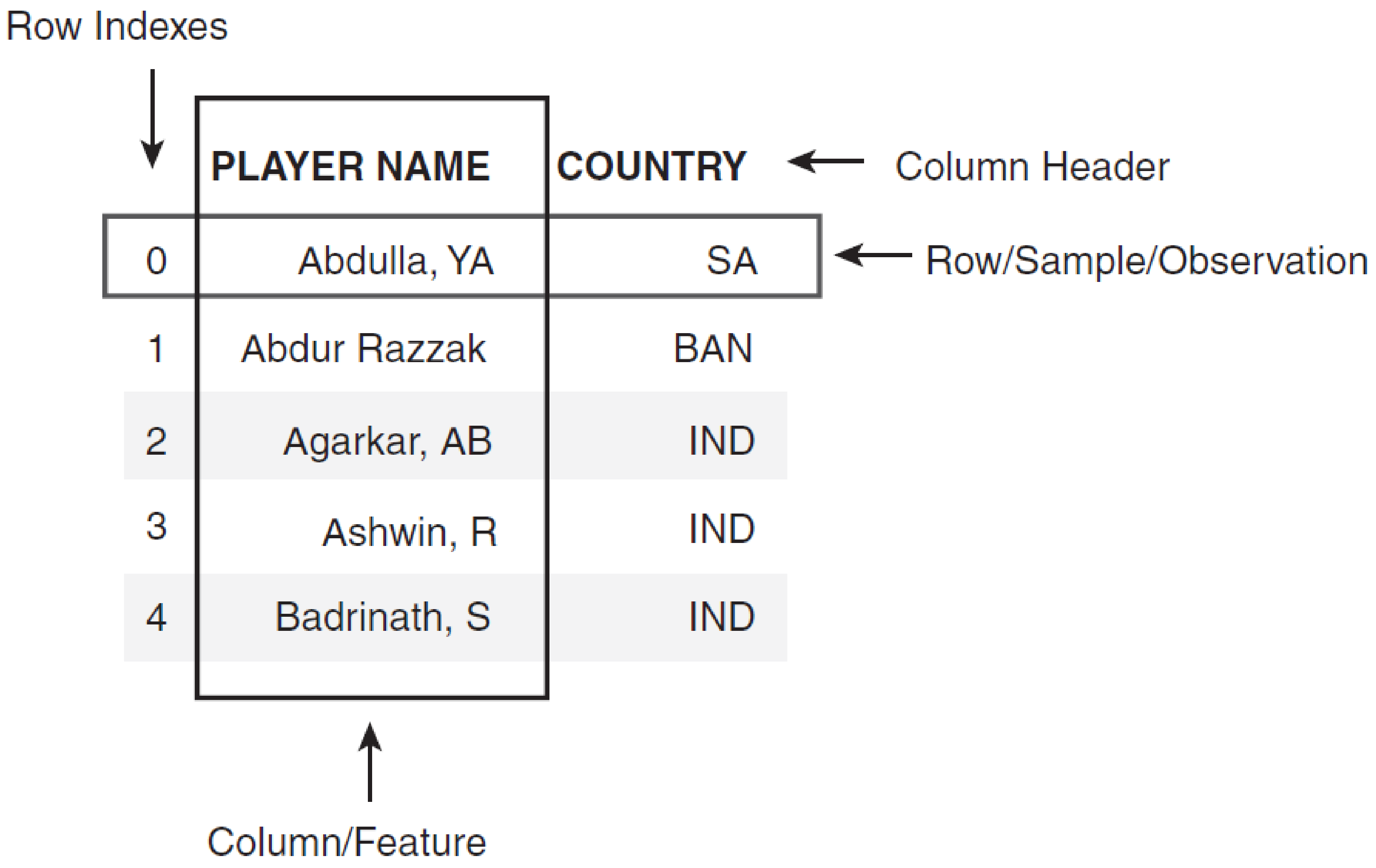
3D array
2D array
1D array
7
2
9
10
axis 0
shape: (4,)
axis 0
5.2
3.0
4.5
9.1
0.1
0.3
axis 1
shape: (2, 3)
axis 0
axis 1
axis 2
shape: (4, 3, 2)

Series

•



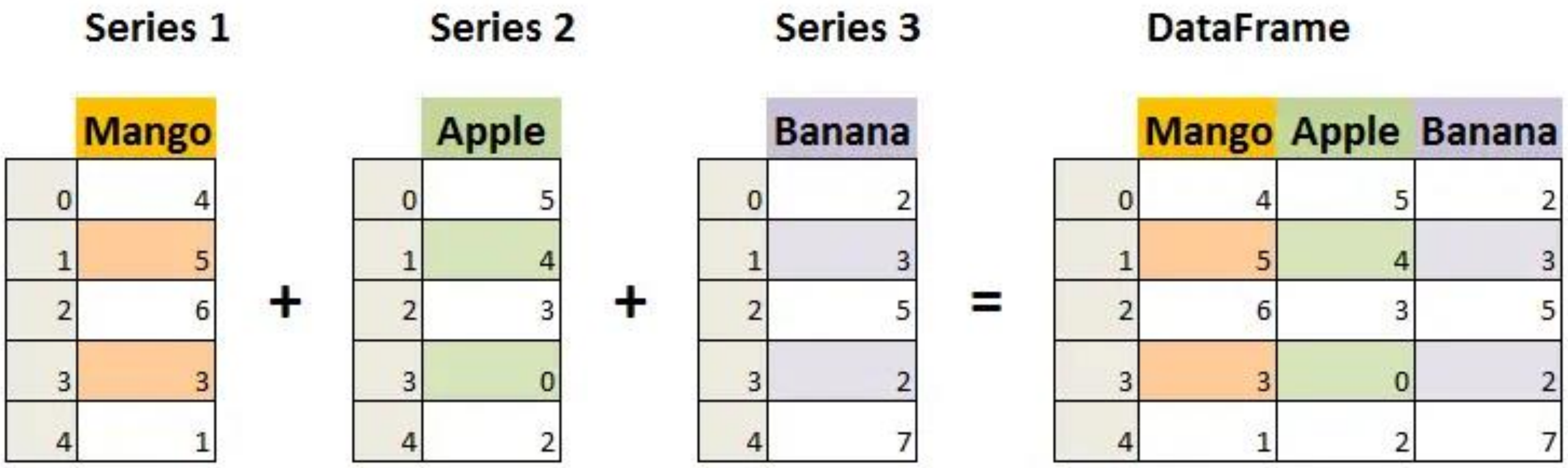
DataFrame



	PLAYER NAME	COUNTRY
0	Abdulla, YA	SA
1	Abdur Razzak	BAN
2	Agarkar, AB	IND
3	Ashwin, R	IND
4	Badrinath, S	IND

FIGURE 2.1 Structure of a DataFrame.

- .



Series 1

	Mango
0	4
1	5
2	6
3	3
4	1

+

Series 2

	Apple
0	5
1	4
2	3
3	0
4	2

+

Series 3

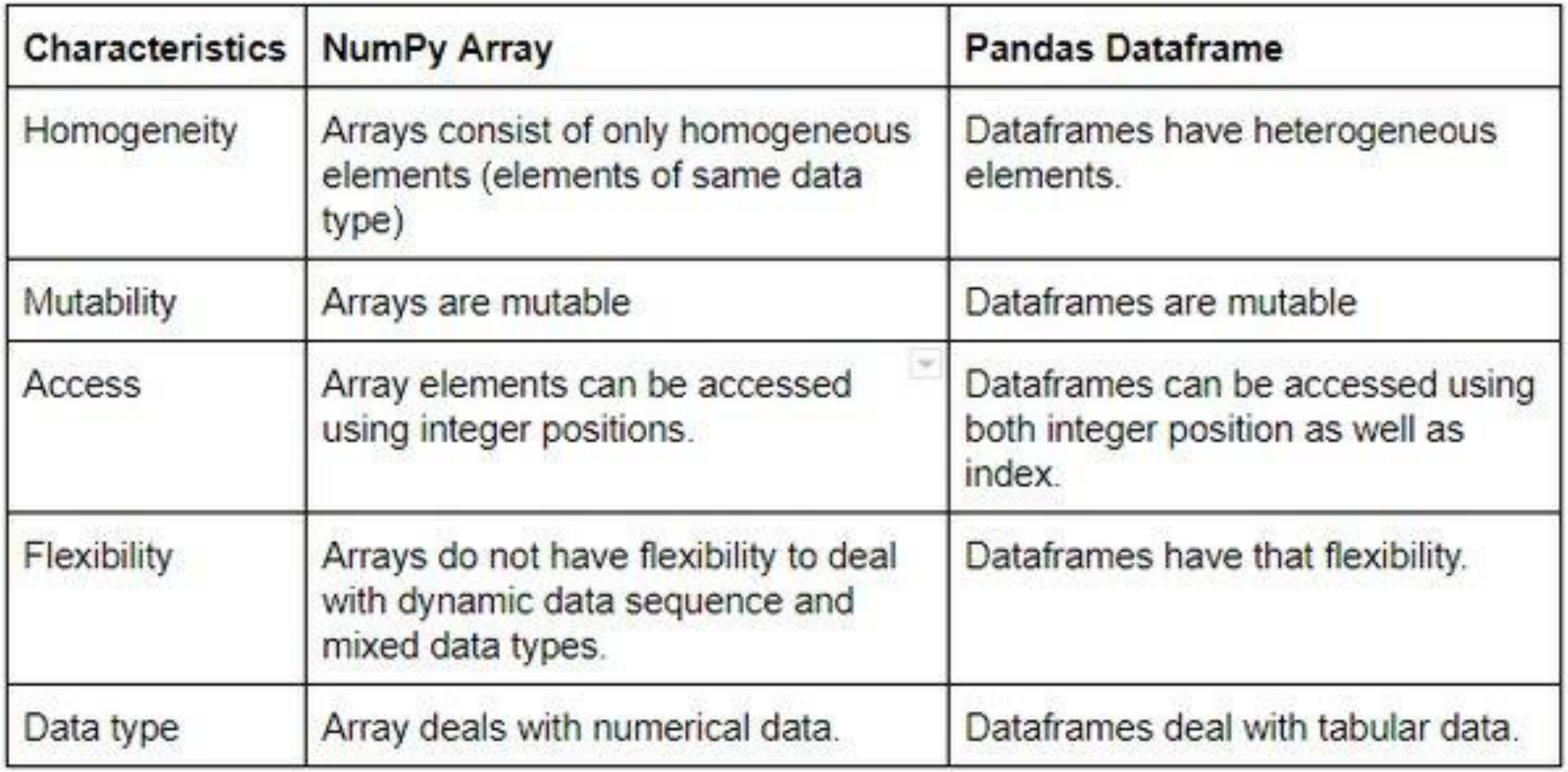
	Banana
0	2
1	3
2	5
3	2
4	7

=

DataFrame

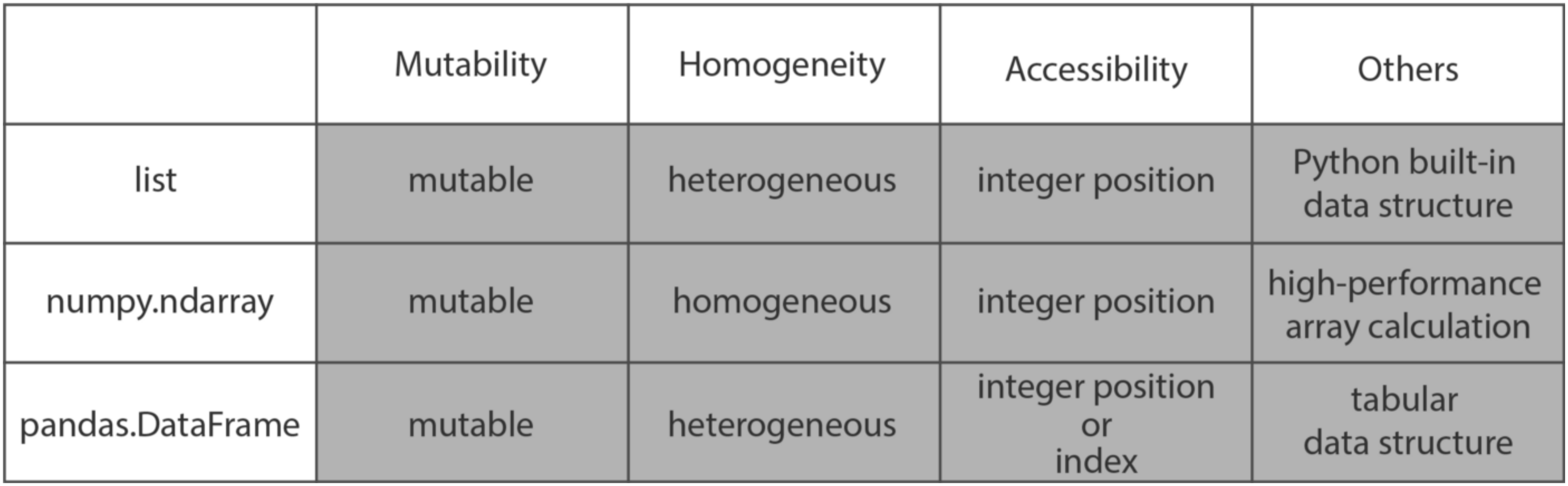
	Mango	Apple	Banana
0	4	5	2
1	5	4	3
2	6	3	5
3	3	0	2
4	1	2	7

Pandas Series	Pandas DataFrame
One-dimensional	Two-dimensional
Homogenous – Series elements must be of the same data type.	Heterogenous – DataFrame elements can have different data types.
Size-immutable – Once created, the size of a Series object cannot be changed.	Size-mutable – Elements can be dropped or added in an existing DataFrame.



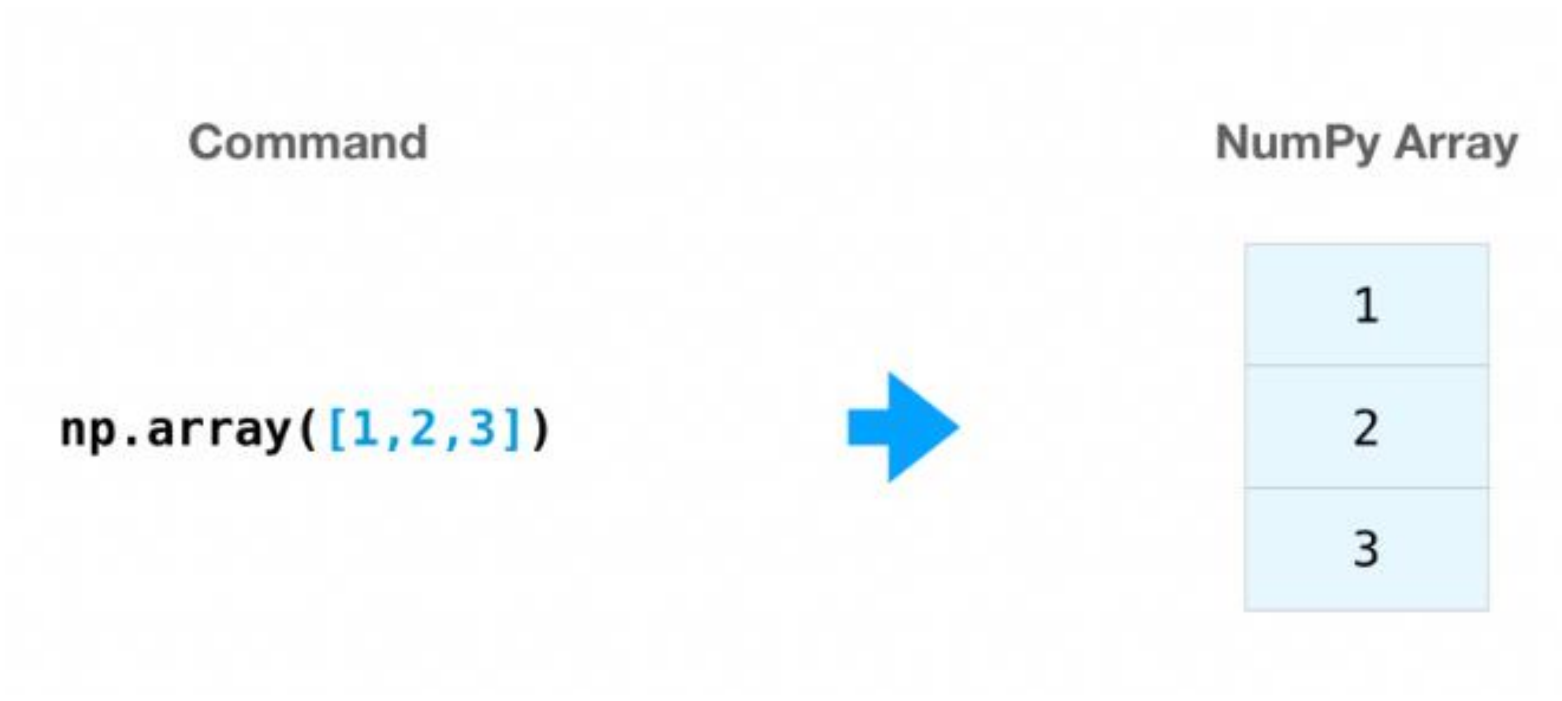
Characteristics	NumPy Array	Pandas Dataframe
Homogeneity	Arrays consist of only homogeneous elements (elements of same data type)	Dataframes have heterogeneous elements.
Mutability	Arrays are mutable	Dataframes are mutable
Access	Array elements can be accessed using integer positions.	Dataframes can be accessed using both integer position as well as index.
Flexibility	Arrays do not have flexibility to deal with dynamic data sequence and mixed data types.	Dataframes have that flexibility.
Data type	Array deals with numerical data.	Dataframes deal with tabular data.

Comparison among List-array-dataframe



	Mutability	Homogeneity	Accessibility	Others
list	mutable	heterogeneous	integer position	Python built-in data structure
numpy.ndarray	mutable	homogeneous	integer position	high-performance array calculation
pandas.DataFrame	mutable	heterogeneous	integer position or index	tabular data structure

Numpy Array: Container of Data



```
np.array([1,2,3])
```

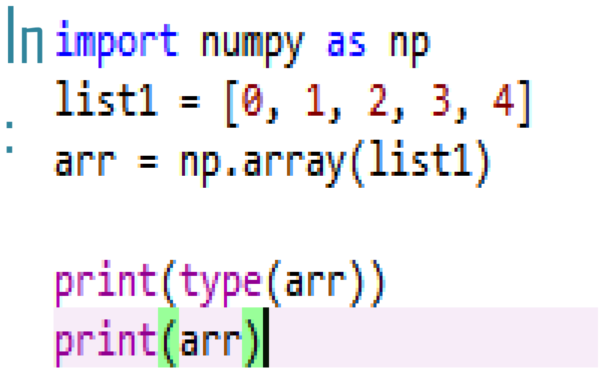
September 15, 2023 fad_ju_spring_23 35

Basic Array Manipulations

- Attributes of arrays
  - Determining the size, shape, memory consumption, and data types of arrays
- Indexing of arrays
  - Getting and setting the values of individual array elements
- Slicing of arrays
  - Getting and setting smaller subarrays within a larger array
- Reshaping of arrays
  - Changing the shape of a given array
- Joining and splitting of arrays
  - Combining multiple arrays into one, and splitting one array into many

Creating a NumPy array

- NumPy array cane be created from a list by passing it to the np.array() function.



```
In import numpy as np
:  list1 = [0, 1, 2, 3, 4]
   arr = np.array(list1)

   print(type(arr))
   print(arr)
```

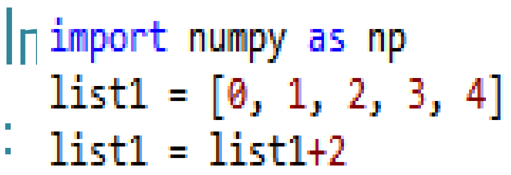
- Other functions used to create array:
  - np.array(), np.zeros(), np.ones(), np.empty(), np.arange(), np.linspace(),

Differences between lists and ndarrays

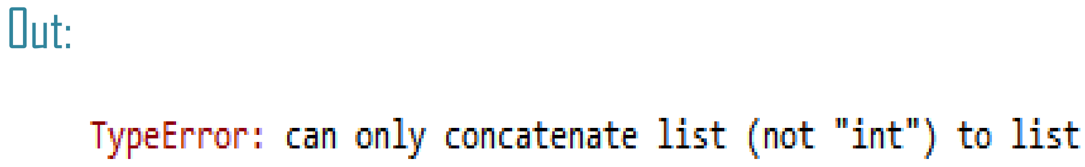
- NumPy provides us an enormous range of fast and efficient ways of creating arrays and manipulating numerical data inside them.
- While a Python list can contain different data types within a single list, all of the elements in a NumPy array should be homogeneous.
- The key difference between an array and a list is that arrays are designed to handle vectorised operations while a python lists are not.
- That means, if you apply a function, it is performed on every item in the array, rather than on the whole array object.

Python List vs Numpy Array

- Let's suppose you want to add the number 2 to every item in the list. The intuitive way to do this is something like this:

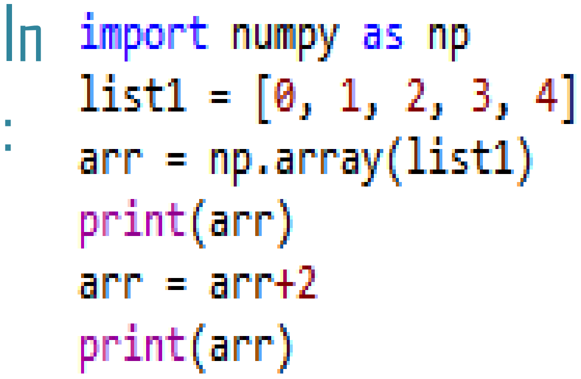


```
In :
import numpy as np
list1 = [0, 1, 2, 3, 4]
list1 = list1+2
```

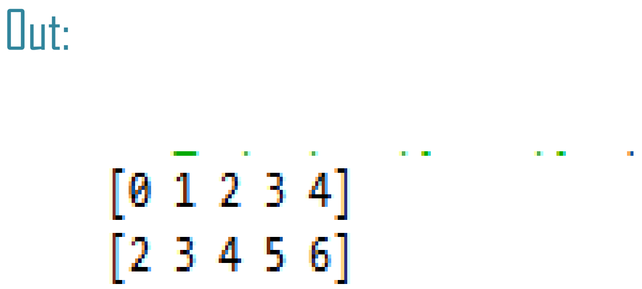


```
Out:
TypeError: can only concatenate list (not "int") to list
```

- That was not possible with a list, but you can do that on an array:



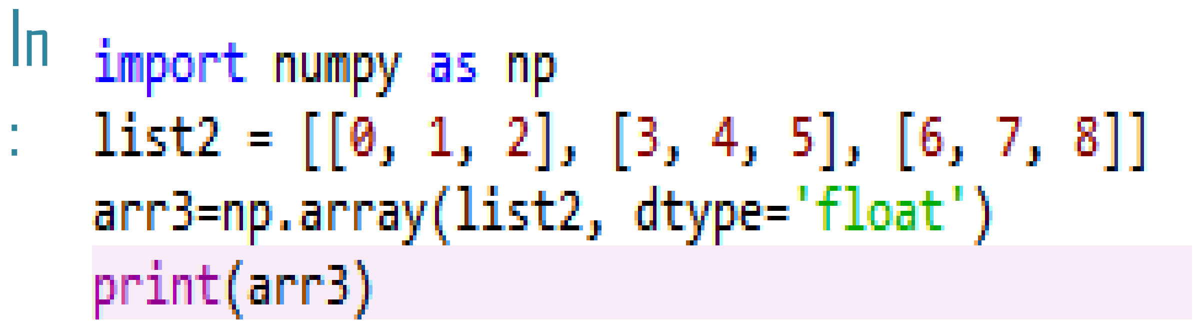
```
In :
import numpy as np
list1 = [0, 1, 2, 3, 4]
arr = np.array(list1)
print(arr)
arr = arr+2
print(arr)
```



```
Out:
[0 1 2 3 4]
[2 3 4 5 6]
```

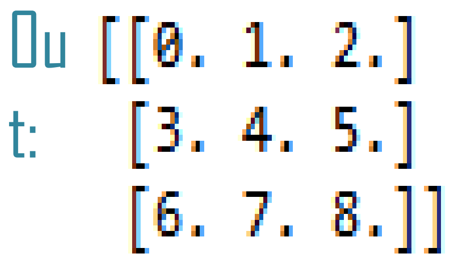
The dtype argument

- You can specify the data-type by setting the dtype() argument.
- Some of the most commonly used NumPy dtypes are: float, int, bool, str, and object.



In :

```
import numpy as np
list2 = [[0, 1, 2], [3, 4, 5], [6, 7, 8]]
arr3=np.array(list2, dtype='float')
print(arr3)
```

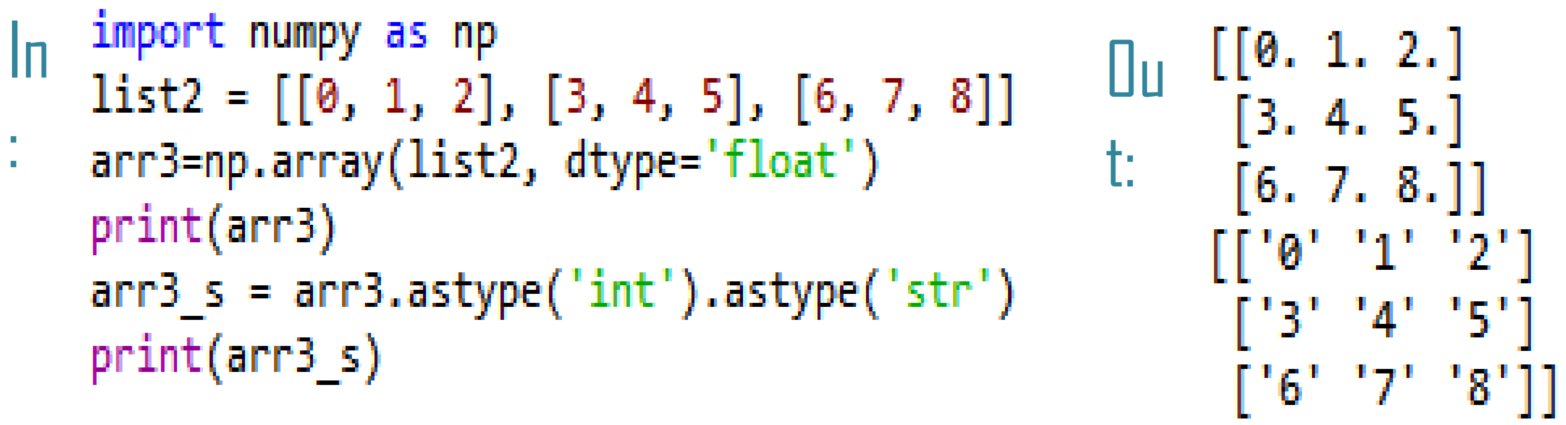


Out:

```
[[0. 1. 2.]
 [3. 4. 5.]
 [6. 7. 8.]]
```

The astype argument

- You can also convert it to a different data-type using the astype method.



In :

```
import numpy as np
list2 = [[0, 1, 2], [3, 4, 5], [6, 7, 8]]
arr3=np.array(list2, dtype='float')
print(arr3)
arr3_s = arr3.astype('int').astype('str')
print(arr3_s)
```

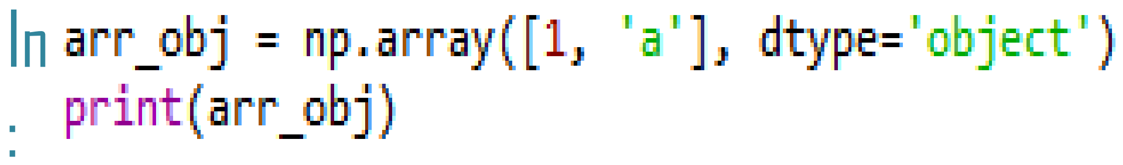
Out:

```
[[0. 1. 2.]
 [3. 4. 5.]
 [6. 7. 8.]]
[['0' '1' '2']
 ['3' '4' '5']
 ['6' '7' '8']]
```

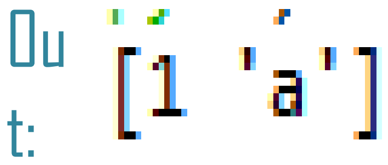
- Remember that, unlike lists, all items in an array have to be of the same type.

dtype='object'

- However, if you are uncertain about what data type your array will hold, or if you want to hold characters and numbers in the same array, you can set the dtype as 'object'.



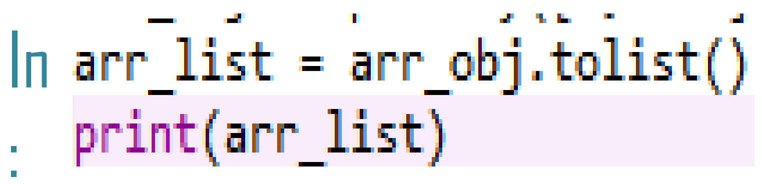
```
In arr_obj = np.array([1, 'a'], dtype='object')
:  print(arr_obj)
```



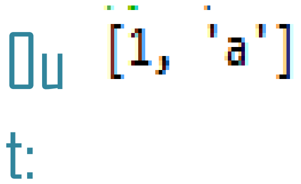
```
Out: [1 'a']
```

The tolist() function

- You can always convert an array into a list using the tolist() command.



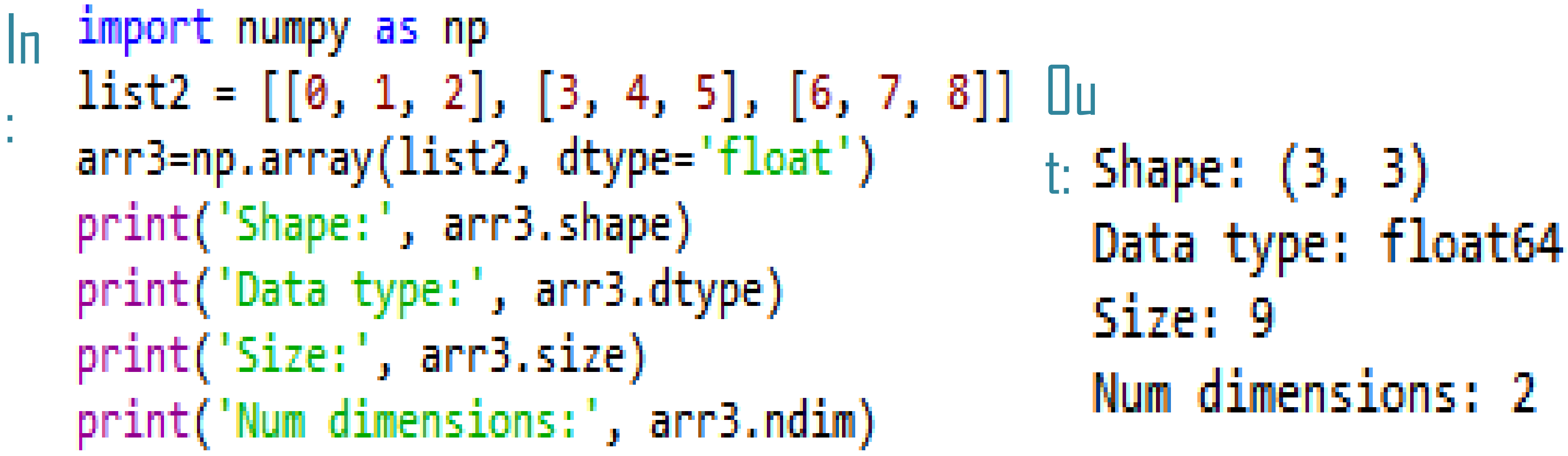
```
In
:  arr_list = arr_obj.tolist()
   print(arr_list)
```



```
Out: [1, 'a']
```

Inspecting a NumPy array

- There are a range of functions built into NumPy that allow you to inspect different aspects of an array:

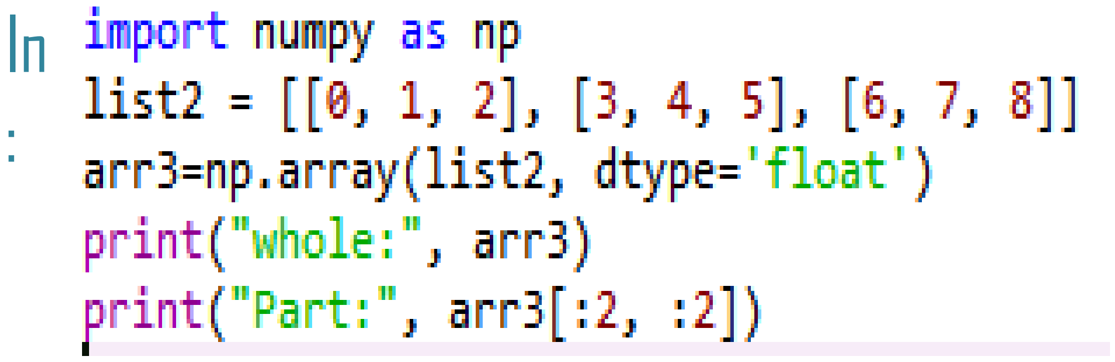


```
In
:
import numpy as np
list2 = [[0, 1, 2], [3, 4, 5], [6, 7, 8]]
arr3=np.array(list2, dtype='float')
print('Shape:', arr3.shape)
print('Data type:', arr3.dtype)
print('Size:', arr3.size)
print('Num dimensions:', arr3.ndim)
```

```
Ou
t: Shape: (3, 3)
   Data type: float64
   Size: 9
   Num dimensions: 2
```

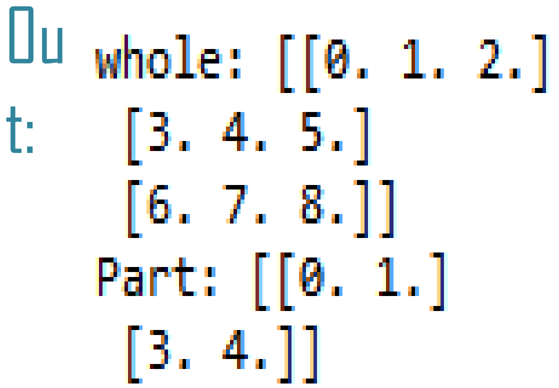
Extracting specific items from an array

- You can extract portions of the array using indices, much like when you’re working with lists.
- Unlike lists, however, arrays can optionally accept as many parameters in the square brackets as there are number of dimensions



In :

```
import numpy as np
list2 = [[0, 1, 2], [3, 4, 5], [6, 7, 8]]
arr3=np.array(list2, dtype='float')
print("whole:", arr3)
print("Part:", arr3[:2, :2])
```

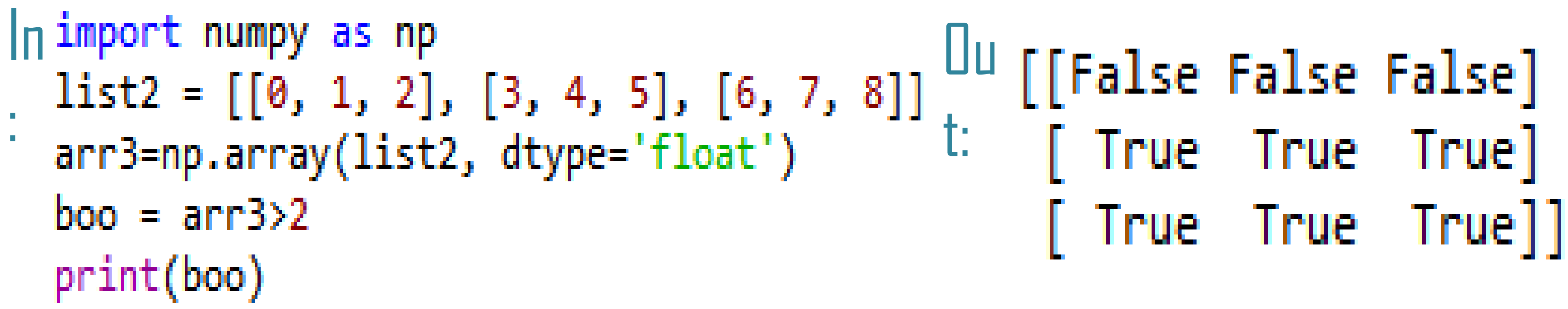


Out:

```
whole: [[0. 1. 2.]
 [3. 4. 5.]
 [6. 7. 8.]]
Part: [[0. 1.]
 [3. 4.]]
```

Boolean indexing

- A boolean index array is of the same shape as the array-to-be-filtered, but it only contains TRUE and FALSE values.



In :
```
import numpy as np
list2 = [[0, 1, 2], [3, 4, 5], [6, 7, 8]]
arr3=np.array(list2, dtype='float')
boo = arr3>2
print(boo)
```

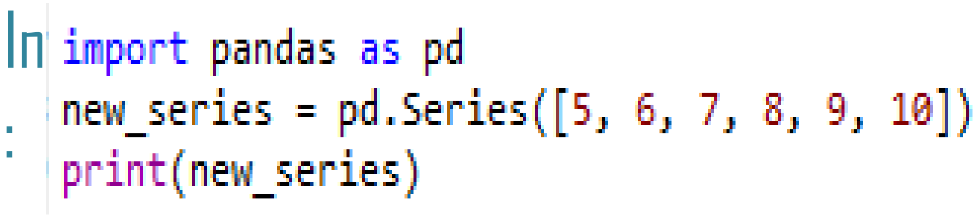
Out:
```
[[False False False]
 [ True  True  True]
 [ True  True  True]]
```

Data Structures in Pandas

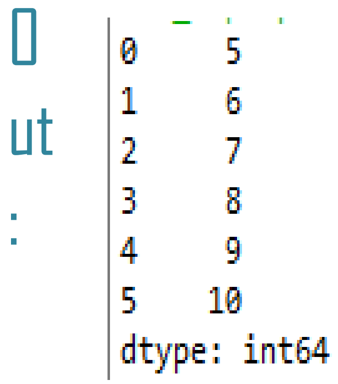
- There are two main structures used by pandas;
  - *data frames* and
  - *series*.

Indices in a pandas series

- A pandas series is similar to a list, but differs in the fact that a series associates a label with each element. This makes it look like a dictionary.
- If an index is not explicitly provided by the user, pandas creates a `RangeIndex` ranging from 0 to *N*-1.
- Each series object also has a data type.



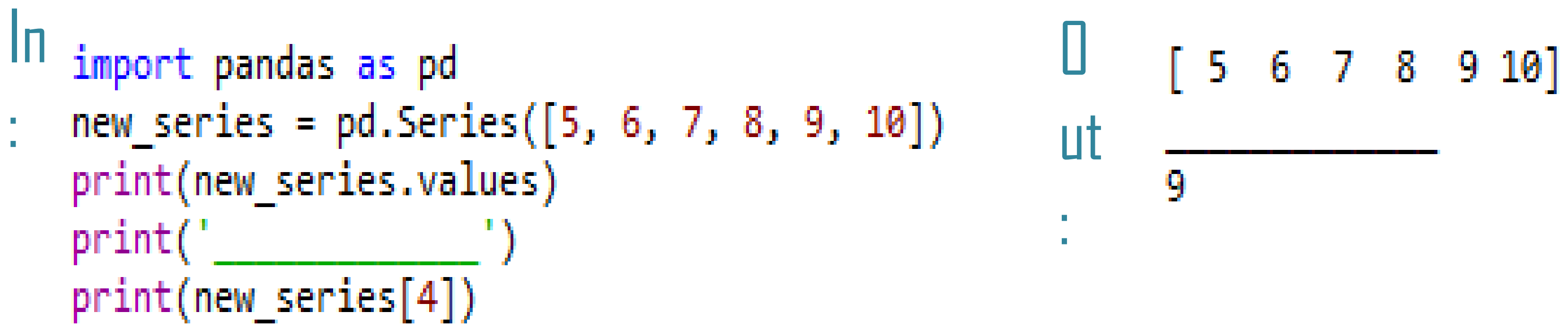
```
In :
import pandas as pd
new_series = pd.Series([5, 6, 7, 8, 9, 10])
print(new_series)
```



```
Out:
0     5
1     6
2     7
3     8
4     9
5    10
dtype: int64
```

Indices in a pandas series

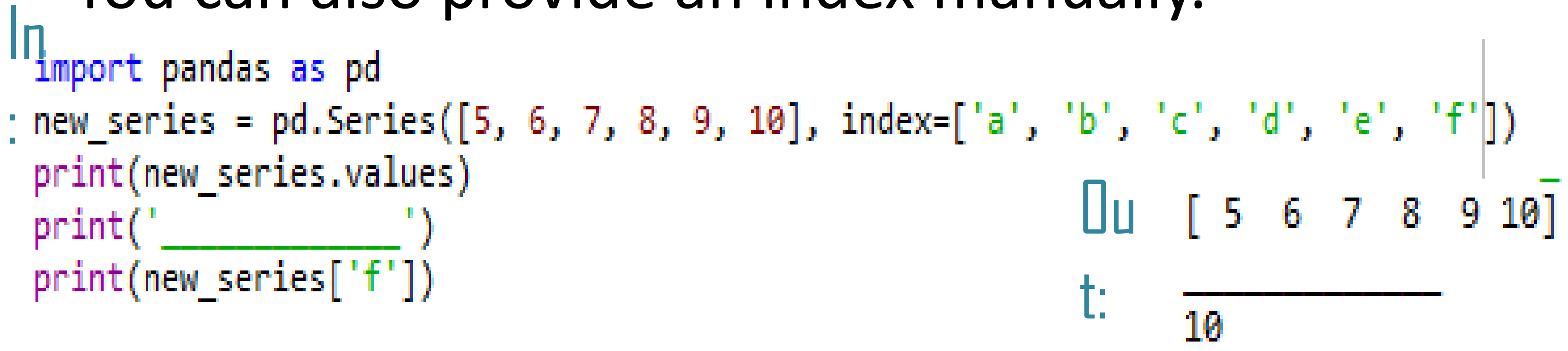
- series allows extract all of the values in the series, as well as individual elements by index.



```
In :  import pandas as pd
      new_series = pd.Series([5, 6, 7, 8, 9, 10])
      print(new_series.values)
      print('____________')
      print(new_series[4])
```

```
Out:  [ 5  6  7  8  9 10]
      ____________
      9
```

- You can also provide an index manually.

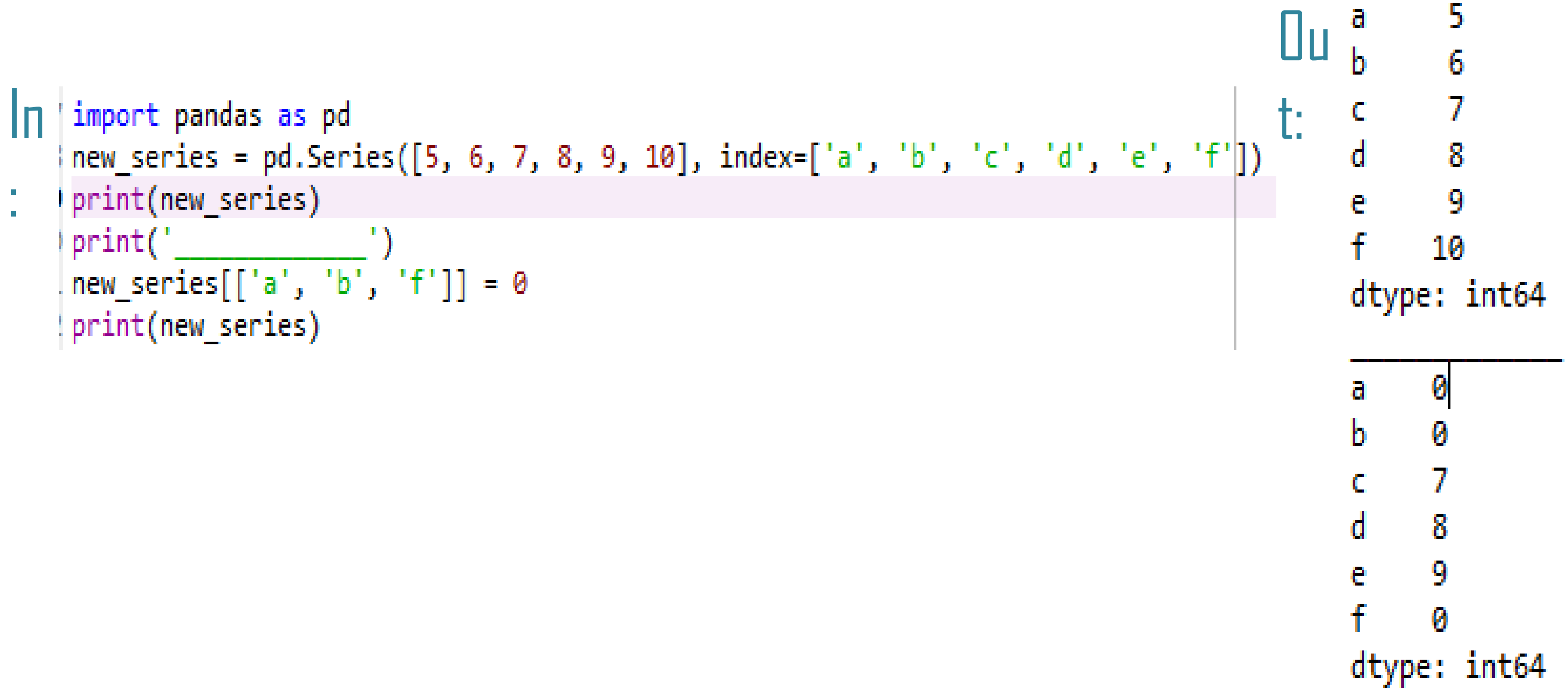


```
In :  import pandas as pd
      new_series = pd.Series([5, 6, 7, 8, 9, 10], index=['a', 'b', 'c', 'd', 'e', 'f'])
      print(new_series.values)
      print('____________')
      print(new_series['f'])
```

```
Out:  [ 5  6  7  8  9 10]
      ____________
      10
```

Indices in a pandas series

- It is easy to retrieve several elements of a series by their indices or make group assignments.



In:

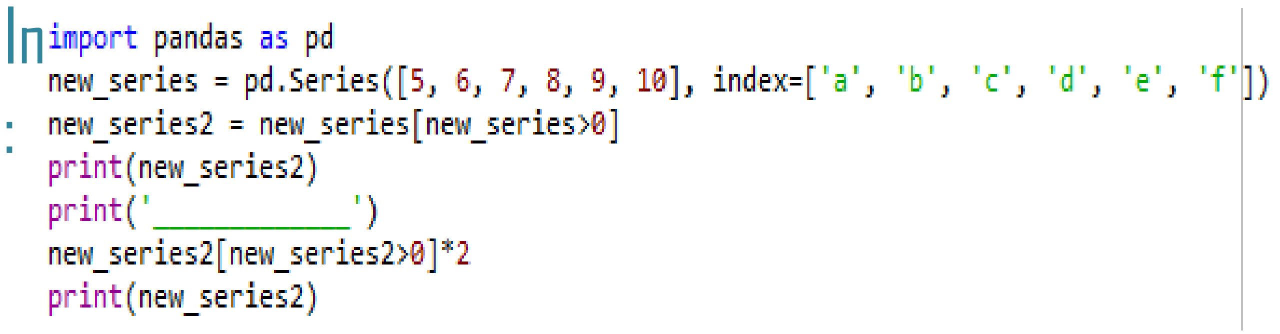
```
import pandas as pd
new_series = pd.Series([5, 6, 7, 8, 9, 10], index=['a', 'b', 'c', 'd', 'e', 'f'])
print(new_series)
print('_____________')
new_series[['a', 'b', 'f']] = 0
print(new_series)
```

Out:

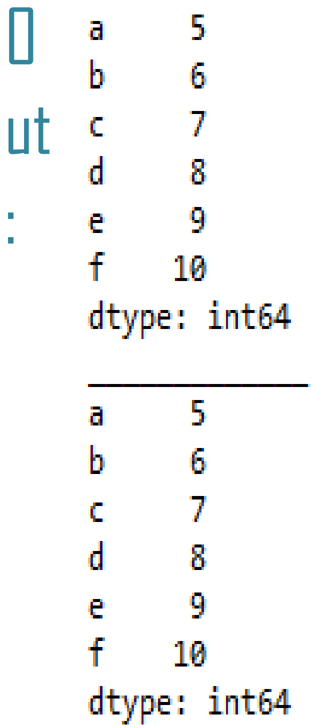
```
a      5
b      6
c      7
d      8
e      9
f     10
dtype: int64
_____________
a     0
b     0
c     7
d     8
e     9
f     0
dtype: int64
```

Filtering and maths operations

- Filtering and maths operations are easy with Pandas as well.



```
In
:  import pandas as pd
   new_series = pd.Series([5, 6, 7, 8, 9, 10], index=['a', 'b', 'c', 'd', 'e', 'f'])
   new_series2 = new_series[new_series>0]
   print(new_series2)
   print('_____________')
   new_series2[new_series2>0]*2
   print(new_series2)
```



```
Out
:  a     5
   b     6
   c     7
   d     8
   e     9
   f    10
   dtype: int64
   _____________
   a     5
   b     6
   c     7
   d     8
   e     9
   f    10
   dtype: int64
```

Data Frame in Pandas

- A dataframe is a two dimensional, heterogenous tabular data structure in Pandas.
- Each column has varied data types
- The dataframe object has two axes: axis 0 [rows] and axis 1 [columns]
- Both axes are labeled

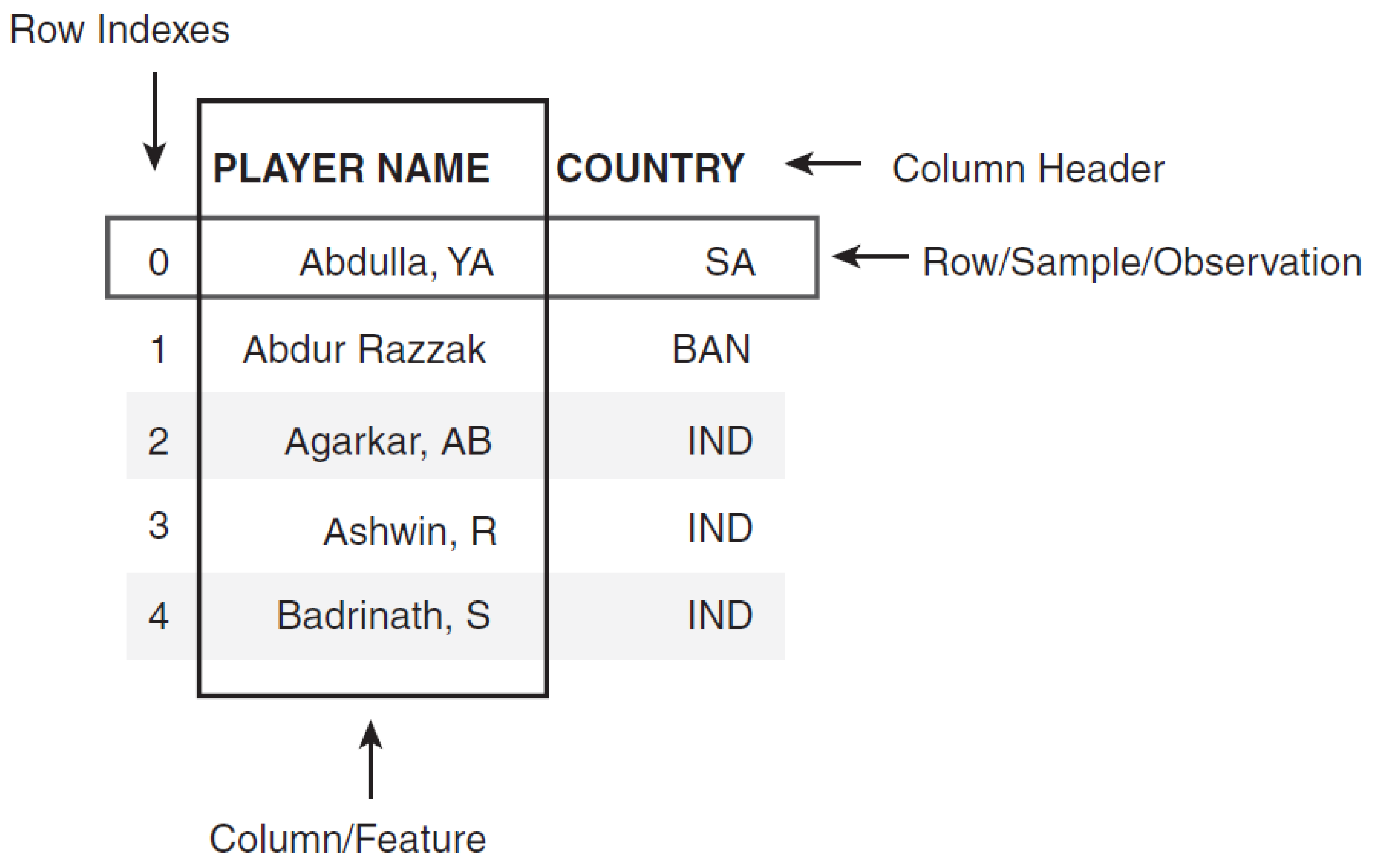


FIGURE 2.1 Structure of a DataFrame.

Pandas data frame

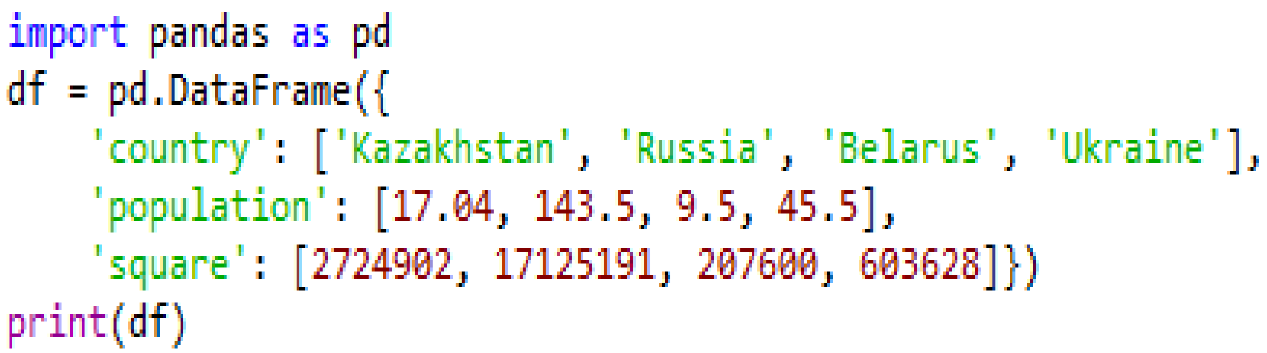
- Simplistically, a data frame is a table, with rows and columns.
- Each column in a data frame is a series object.
- Rows consist of elements inside series.

Case ID	Variable one	Variable two	Variable 3
1	123	ABC	10
2	456	DEF	20
3	789	XYZ	30

Creating a Pandas data frame

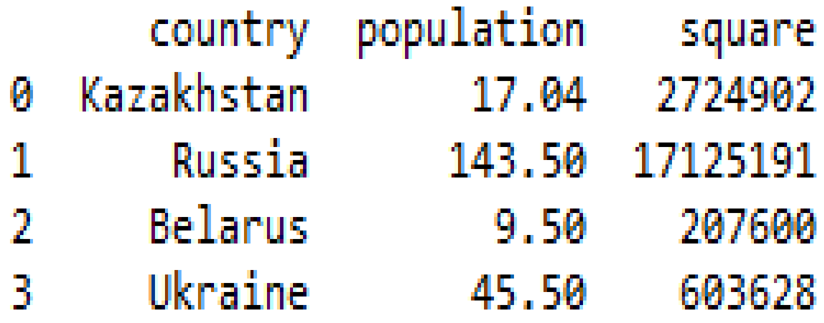
- Pandas data frames can be constructed using Python dictionaries.

In :



```
import pandas as pd
df = pd.DataFrame({
    'country': ['Kazakhstan', 'Russia', 'Belarus', 'Ukraine'],
    'population': [17.04, 143.5, 9.5, 45.5],
    'square': [2724902, 17125191, 207600, 603628]})
print(df)
```

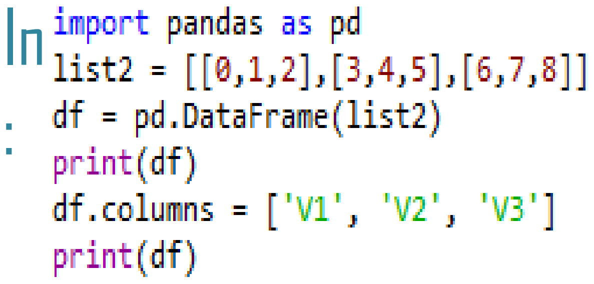
Out:



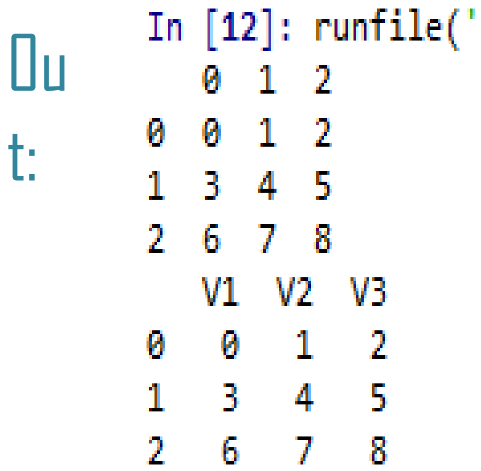
```
      country  population    square
0  Kazakhstan       17.04   2724902
1      Russia      143.50  17125191
2     Belarus        9.50    207600
3     Ukraine       45.50    603628
```

to DataFrame from list

- You can also create a data frame from a list.

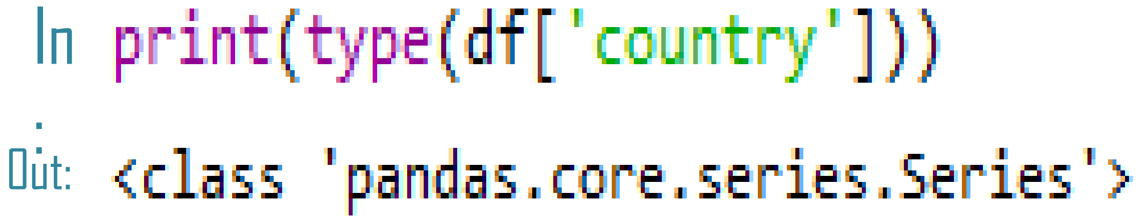


```
In
:
import pandas as pd
list2 = [[0,1,2],[3,4,5],[6,7,8]]
df = pd.DataFrame(list2)
print(df)
df.columns = ['V1', 'V2', 'V3']
print(df)
```



```
Ou
t:
In [12]: runfile('
   0  1  2
0  0  1  2
1  3  4  5
2  6  7  8
   V1  V2  V3
0   0   1   2
1   3   4   5
2   6   7   8
```

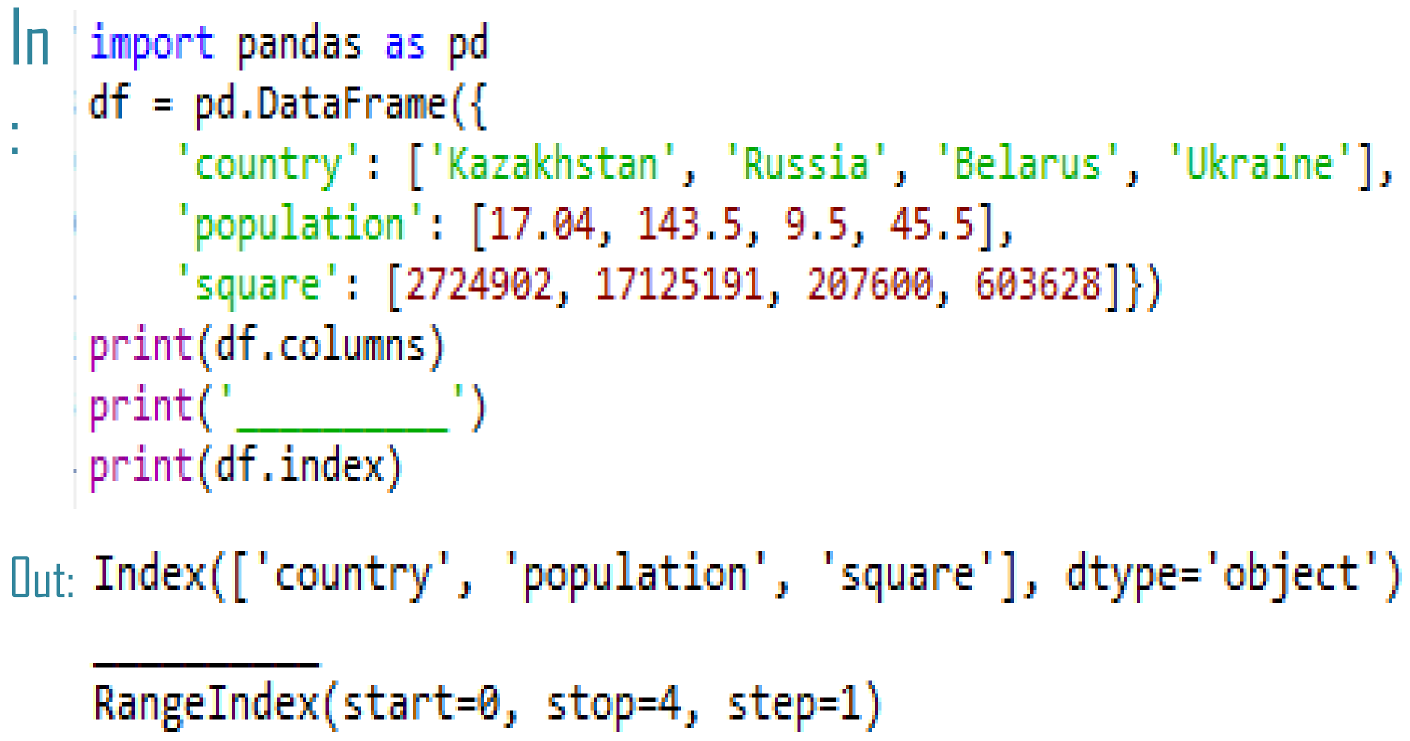
- You can ascertain the type of a column with the type() function.



```
In
:   print(type(df['country']))
Out: <class 'pandas.core.series.Series'>
```

Indices in Pandas data frame

- A Pandas data frame object has two indices; a column index and row index.
- Again, if you do not provide one, Pandas will create a RangeIndex from 0 to *N*-1.



In :

```
import pandas as pd
df = pd.DataFrame({
    'country': ['Kazakhstan', 'Russia', 'Belarus', 'Ukraine'],
    'population': [17.04, 143.5, 9.5, 45.5],
    'square': [2724902, 17125191, 207600, 603628]})
print(df.columns)
print('__________')
print(df.index)
```

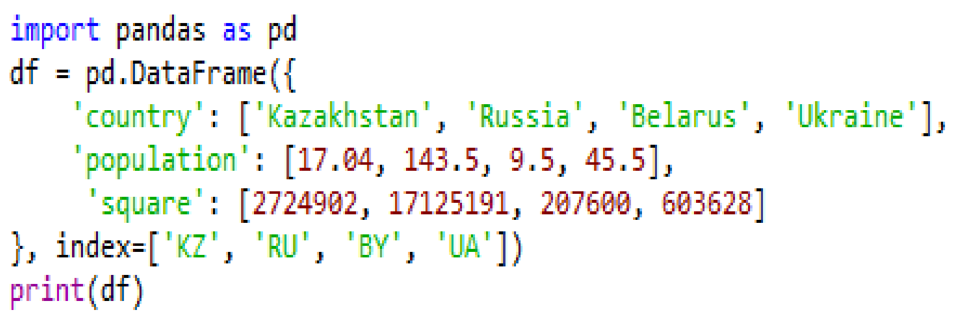
Out:

```
Index(['country', 'population', 'square'], dtype='object')
__________
RangeIndex(start=0, stop=4, step=1)
```

Indices in Pandas data frame

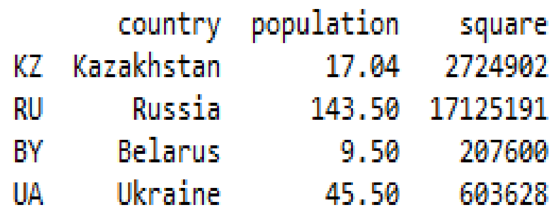
- There are numerous ways to provide row indices explicitly.
- For example, you could provide an index when creating a data frame:

In :



```
import pandas as pd
df = pd.DataFrame({
     'country': ['Kazakhstan', 'Russia', 'Belarus', 'Ukraine'],
     'population': [17.04, 143.5, 9.5, 45.5],
      'square': [2724902, 17125191, 207600, 603628]
}, index=['KZ', 'RU', 'BY', 'UA'])
print(df)
```

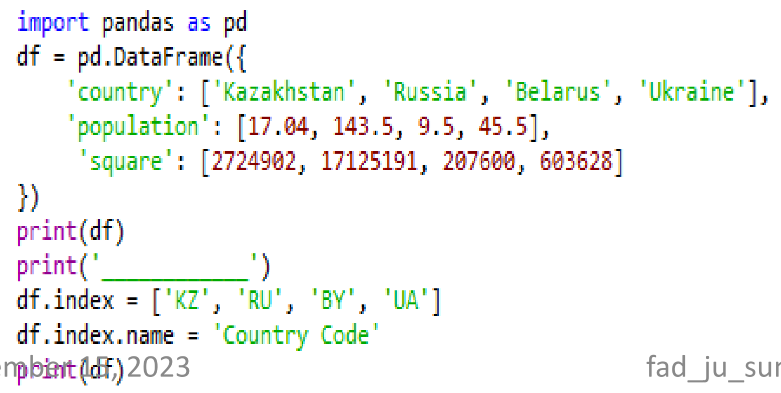
Out:



```
       country  population    square
KZ  Kazakhstan       17.04   2724902
RU      Russia      143.50  17125191
BY     Belarus        9.50    207600
UA     Ukraine       45.50    603628
```

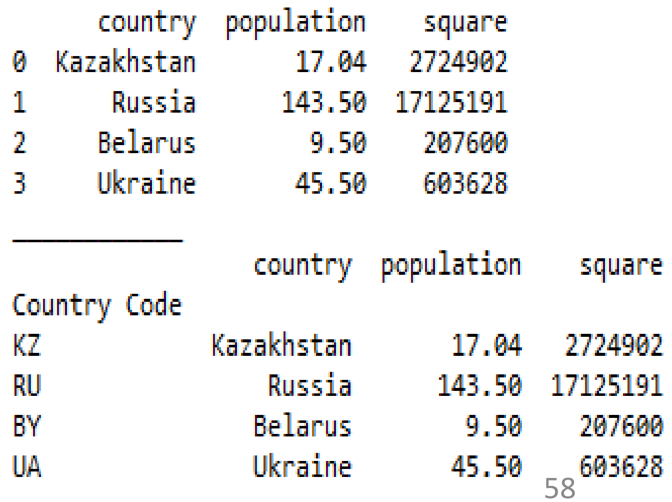
- Or rename index after manually

In :



```
import pandas as pd
df = pd.DataFrame({
    'country': ['Kazakhstan', 'Russia', 'Belarus', 'Ukraine'],
    'population': [17.04, 143.5, 9.5, 45.5],
     'square': [2724902, 17125191, 207600, 603628]
})
print(df)
print('__________')
df.index = ['KZ', 'RU', 'BY', 'UA']
df.index.name = 'Country Code'
print(df)
```

Out:

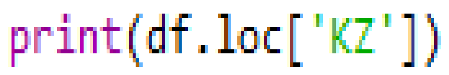


```
      country  population    square
0  Kazakhstan       17.04   2724902
1      Russia      143.50  17125191
2     Belarus        9.50    207600
3     Ukraine       45.50    603628
__________
                 country  population    square
Country Code
KZ            Kazakhstan       17.04   2724902
RU                Russia      143.50  17125191
BY               Belarus        9.50    207600
UA               Ukraine       45.50    603628
```

Accessing data frame using Index

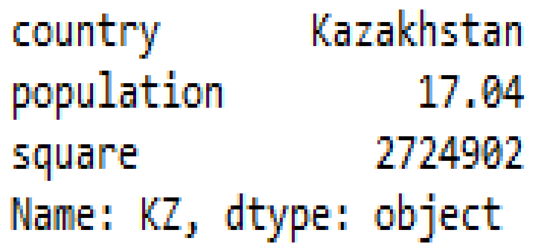
- Row access using index can be performed in several ways.
- First, you could use .loc() and provide an index label.

In :



```
print(df.loc['KZ'])
```

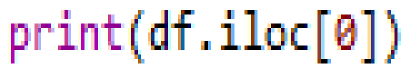
Out:



```
country       Kazakhstan
population         17.04
square           2724902
Name: KZ, dtype: object
```

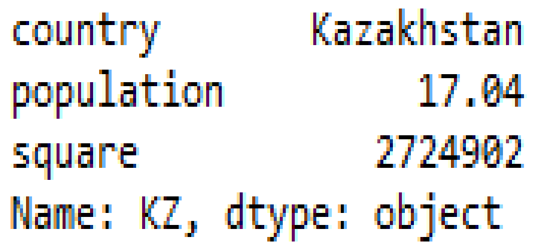
- Second, you could use .iloc() and provide an index number

In :



```
print(df.iloc[0])
```

Out:



```
country       Kazakhstan
population         17.04
square           2724902
Name: KZ, dtype: object
```

Slicing in Pandas data frame

- A selection of particular rows and columns can be selected this way.

```
In print(df.loc[['KZ', 'RU'], 'population'])
:
```

```
Ou Country Code
t: KZ      17.04
   RU     143.50
   Name: population, dtype: float64
```

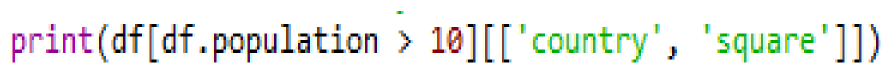
- You can feed .loc() two arguments, index list and column list, slicing operation is supported as well:

```
In print(df.loc['KZ':'BY', :])
:
```

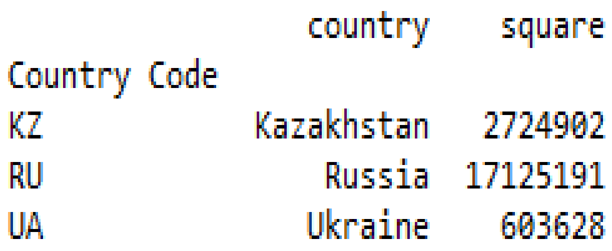
```
Ou                   country  population    square
t: Country Code
   KZ              Kazakhstan       17.04   2724902
   RU                  Russia      143.50  17125191
   BY                 Belarus        9.50    207600
```

Filtering

- Filtering is performed using so-called Boolean arrays.



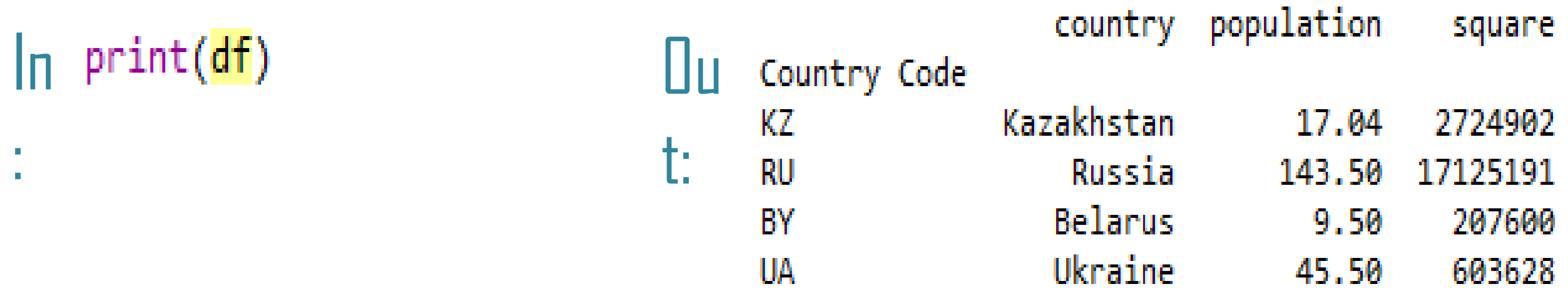
```
print(df[df.population > 10][['country', 'square']])
```



```
                   country    square
Country Code
KZ              Kazakhstan   2724902
RU                  Russia  17125191
UA                 Ukraine    603628
```

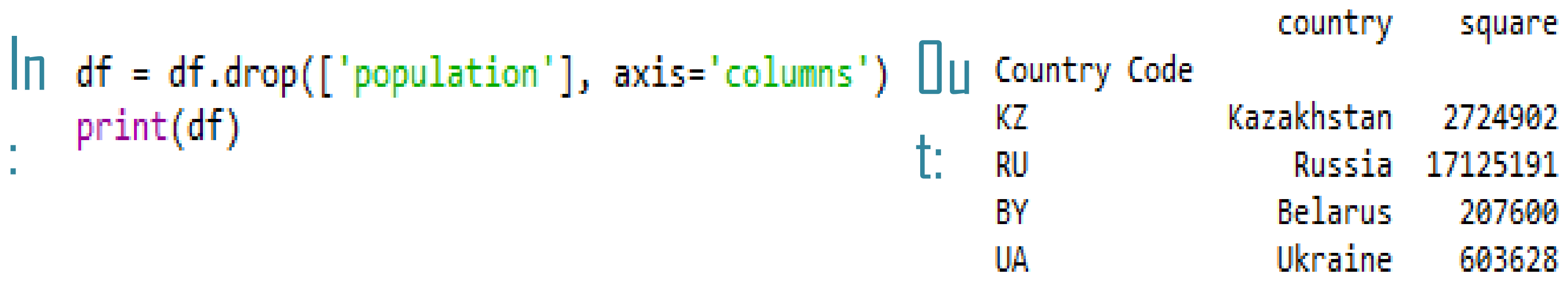
Deleting columns

- You can delete a column using the drop() function.



```
In :  print(df)
```

```
Out:               country  population    square
     Country Code
     KZ            Kazakhstan       17.04   2724902
     RU                Russia      143.50  17125191
     BY               Belarus        9.50    207600
     UA               Ukraine       45.50    603628
```

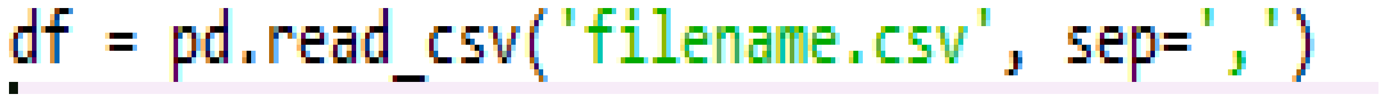


```
In :  df = df.drop(['population'], axis='columns')
      print(df)
```

```
Out:               country    square
     Country Code
     KZ            Kazakhstan   2724902
     RU                Russia  17125191
     BY               Belarus    207600
     UA               Ukraine    603628
```

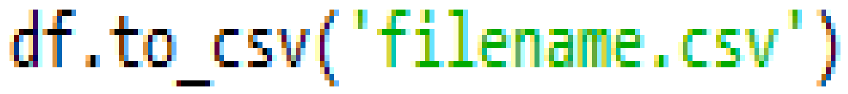
Reading from and writing to a file

- Pandas supports many popular file formats including CSV, XML, HTML, Excel, SQL, JSON, etc.
- Out of all of these, CSV is the file format that you will work with the most.
- You can read in the data from a CSV file using the read_csv() function.



```
df = pd.read_csv('filename.csv', sep=',')
```

- Similarly, you can write a data frame to a csv file with the to_csv() function.



```
df.to_csv('filename.csv')
```

Function vs Methods

Functions in Python	**Methods in Python**
Functions are outside a class	Methods are created inside a class
Functions are not linked to anything	Methods are linked with the classes they are created in
Functions can be executed just by calling with its name	To execute methods, we need to use either an object name or class name and a dot operator.
Functions can have zero parameters.	Methods should have a default parameter either self or cls to get the object's or class's address.
Functions can not access or modify class attributes	Methods can access and modify class attributes
Functions are independent of classes	Methods are dependent on classes

THANK YOU