[DSC4001-01] Python Programming for Data Science

Lecture 05: Pandas

Hyeryung Jang
(hyeryung.jang@dgu.ac.kr)
AI Department, Dongguk University

Syllabus: Today's Topic

Week	Topics
1	Introduction to Data Science, Environment Set-up
2	Python Basics 1
3	Python Basics 2
4	Python for Data Analysis: NumPy
5	**Python for Data Analysis: Pandas 1**
6	Python for Data Analysis: Pandas 2
7	Python for Data Analysis: Web Crawling
8	Midterm Exam
9	Python for Data Visualization: Basics
10	Python for Data Visualization: Advanced
11	Machine Learning with Python: Supervised Learning
12	Machine Learning with Python: Unsupervised Learning
13	Machine Learning with Python: Recommender System
14	Project Presentation
15	Final Exam

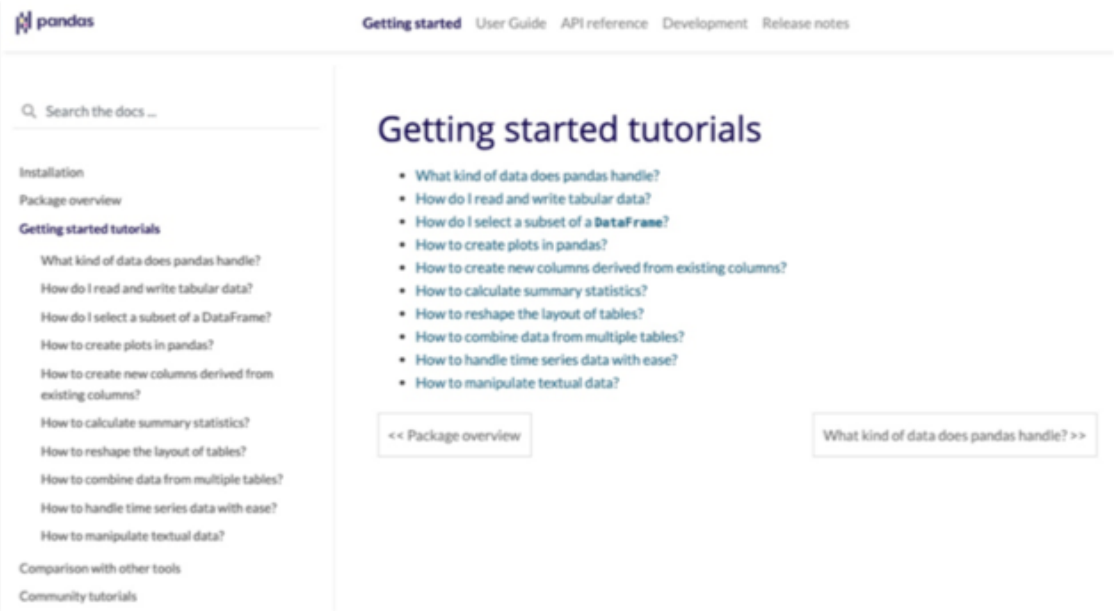
Introduction to Pandas

Pandas

- **Pandas**
  - An open source library built on top of NumPy
  - Provide fast, flexible, and expressive data structures
  - Working with **“labeled” data** both easy and intuitive
  - Fundamental high-level building block for doing practical, real-world data analysis

pandas

pandas.pydata.org



pandas

Getting started User Guide API reference Development Release notes

Search the docs ...

Installation

Package overview

Getting started tutorials

- What kind of data does pandas handle?
- How do I read and write tabular data?
- How do I select a subset of a DataFrame?
- How to create plots in pandas?
- How to create new columns derived from existing columns?
- How to calculate summary statistics?
- How to reshape the layout of tables?
- How to combine data from multiple tables?
- How to handle time series data with ease?
- How to manipulate textual data?

Comparison with other tools

Community tutorials

Getting started tutorials

- What kind of data does pandas handle?
- How do I read and write tabular data?
- How do I select a subset of a `DataFrame`?
- How to create plots in pandas?
- How to create new columns derived from existing columns?
- How to calculate summary statistics?
- How to reshape the layout of tables?
- How to combine data from multiple tables?
- How to handle time series data with ease?
- How to manipulate textual data?

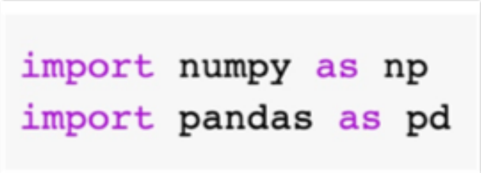
<< Package overview

What kind of data does pandas handle? >>

Install & Import

- Install Pandas by going to your terminal or command prompt and typing:
  - **pip install pandas**
  - **conda install pandas**
  - H
  - ttps://pandas.pydata.org/docs/getting_started/install.html

- Import



```
import numpy as np
import pandas as pd
```

Pandas

- Pandas is well suited for many different kinds of data:
  - **Tabular data** with heterogeneously-typed columns, e.g., Excel spreadsheet
  - Ordered and unordered **time series** data
  - Arbitrary matrix data with **row and column labels**
  - Any other form of observational / statistical data sets

Pandas: We will focus on

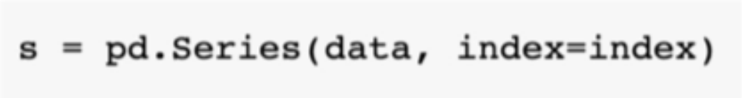
- Series
- DataFrames
- Missing Data
- GroupBy
- Merging, Joining, and Concatenating
- Operations
- Data Input and Output

Series

Pandas object

- Enhanced versions of NumPy arrays
  - Series: enhanced version of 1D ndarray
  - DataFrame: enhanced version of 2D ndarray

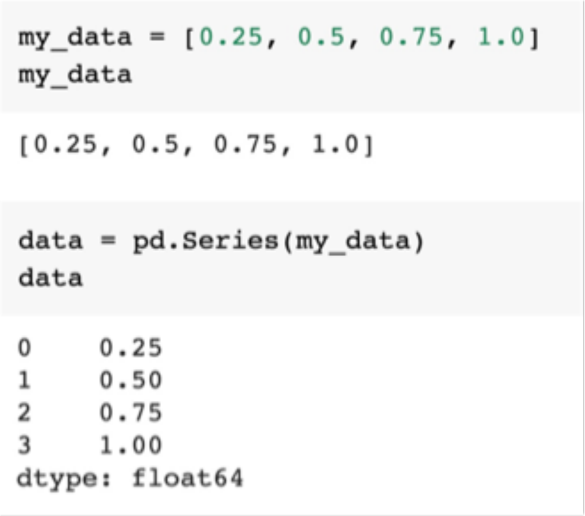
- **Series**
  - 1D array of indexed, labeled data
  - Sequence of **values** (any data type)
    - Integers, strings, floats, lists, dict, ndarray, …
  - Sequence of **indices**
    - List of axis labels



```
s = pd.Series(data, index=index)
```

Creating Series

- From Python list, ndarray, Python dict
  - If no index is passed: [0, 1, …, len(data)-1], or keys of the dictionary



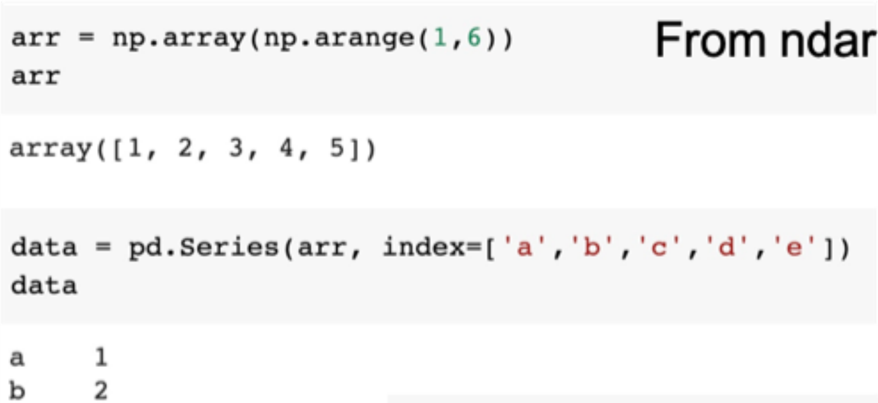
```
my_data = [0.25, 0.5, 0.75, 1.0]
my_data
```

```
[0.25, 0.5, 0.75, 1.0]
```

```
data = pd.Series(my_data)
data
```

```
0    0.25
1    0.50
2    0.75
3    1.00
dtype: float64
```

From Python list



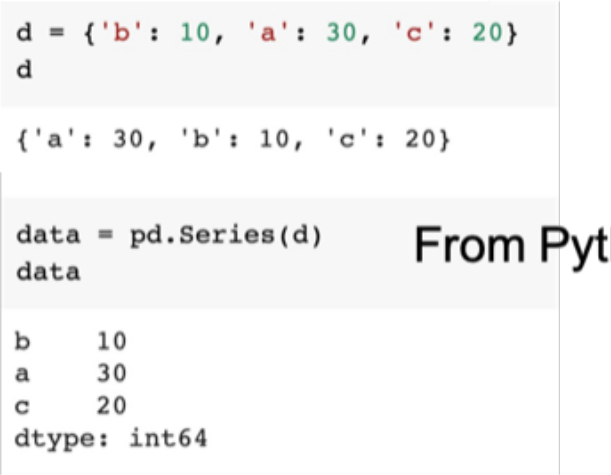
```
arr = np.array(np.arange(1,6))
arr
```

```
array([1, 2, 3, 4, 5])
```

```
data = pd.Series(arr, index=['a','b','c','d','e'])
data
```

```
a    1
b    2
c    3
d    4
e    5
dtype: int64
```

From ndarray



```
d = {'b': 10, 'a': 30, 'c': 20}
d
```

```
{'a': 30, 'b': 10, 'c': 20}
```

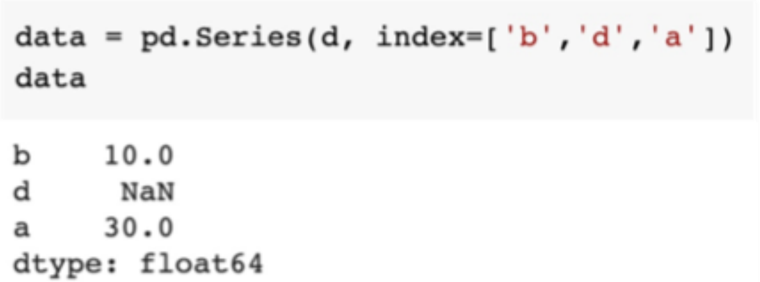
```
data = pd.Series(d)
data
```

```
b    10
a    30
c    20
dtype: int64
```

From Python dict

Creating Series

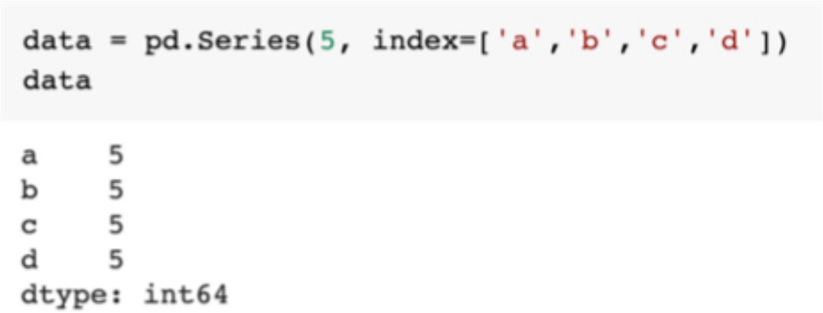
- From Python list, ndarray, Python dict
  - If an index is passed, the values corresponding to the labels will be pulled out



```
data = pd.Series(d, index=['b','d','a'])
data
```

```
b    10.0
d     NaN
a    30.0
dtype: float64
```

  - If data is a scalar value, the value will be repeated to all indices



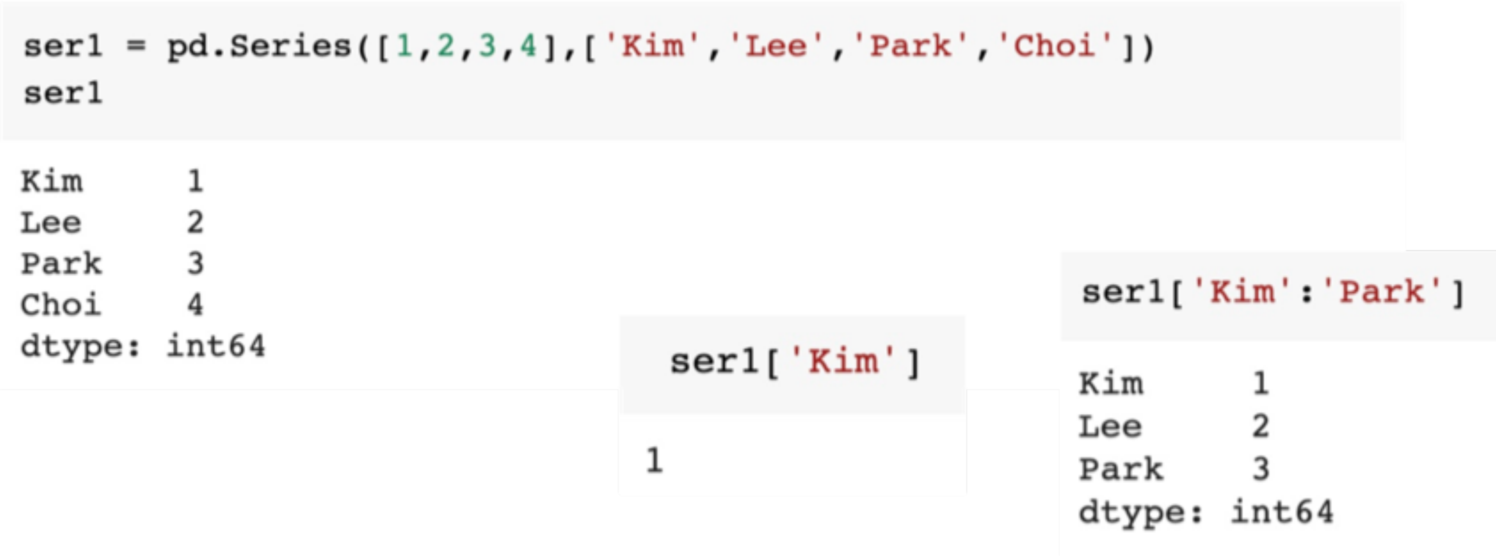
```
data = pd.Series(5, index=['a','b','c','d'])
data
```

```
a    5
b    5
c    5
d    5
dtype: int64
```

Series is 1D ndarray-like

- ndarray has an **implicitly** defined integer index
- Series has an **explicitly** defined (any type of) index associated with the values

- Use index to access the value



```
ser1 = pd.Series([1,2,3,4],['Kim','Lee','Park','Choi'])
ser1
```

```
Kim     1
Lee     2
Park    3
Choi    4
dtype: int64
```

```
ser1['Kim']
```

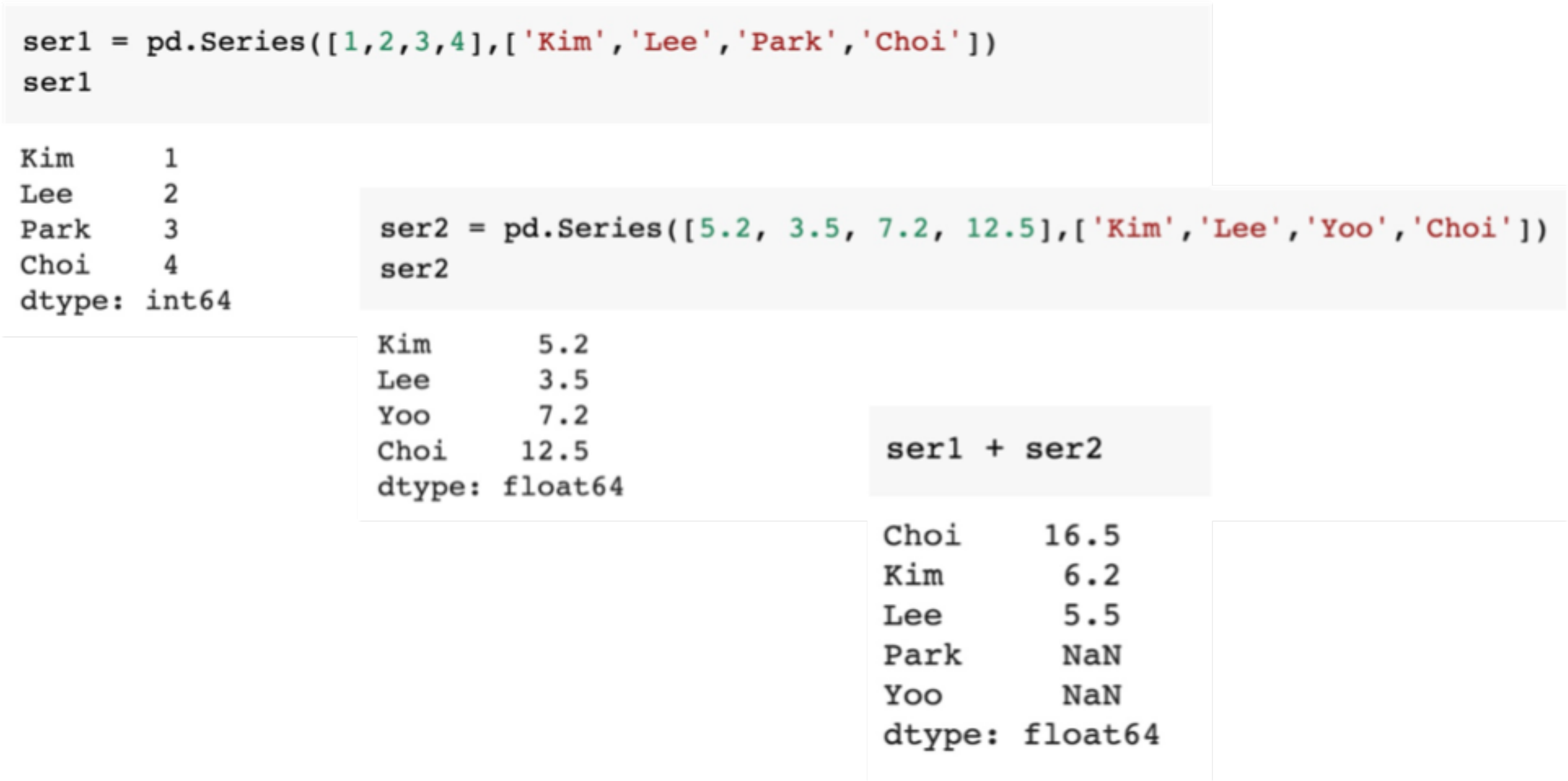
```
1
```

```
ser1['Kim':'Park']
```

```
Kim     1
Lee     2
Park    3
dtype: int64
```

Series is 1D ndarray-like

- Operations between Series **automatically align the data based on label**



```
ser1 = pd.Series([1,2,3,4],['Kim','Lee','Park','Choi'])
ser1
```

```
Kim     1
Lee     2
Park    3
Choi    4
dtype: int64
```

```
ser2 = pd.Series([5.2, 3.5, 7.2, 12.5],['Kim','Lee','Yoo','Choi'])
ser2
```

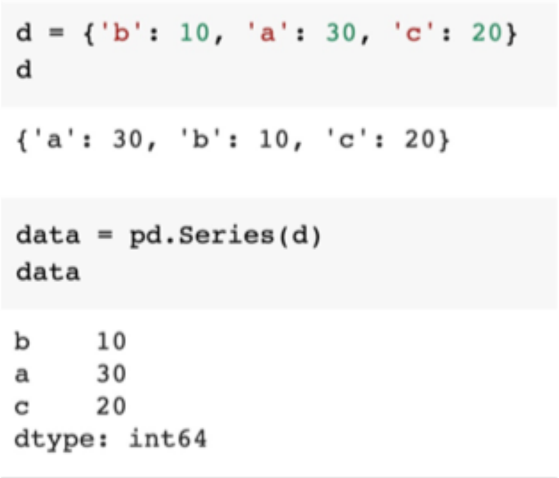
```
Kim      5.2
Lee      3.5
Yoo      7.2
Choi    12.5
dtype: float64
```

```
ser1 + ser2
```

```
Choi    16.5
Kim      6.2
Lee      5.5
Park     NaN
Yoo      NaN
dtype: float64
```

Series is dictionary-like

- Dictionary maps **arbitrary keys** to a set of **arbitrary values**
- Series maps **typed keys** to a set of **typed values**



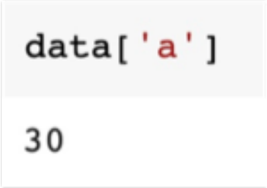
```
d = {'b': 10, 'a': 30, 'c': 20}
d
```

```
{'a': 30, 'b': 10, 'c': 20}
```

```
data = pd.Series(d)
data
```

```
b    10
a    30
c    20
dtype: int64
```

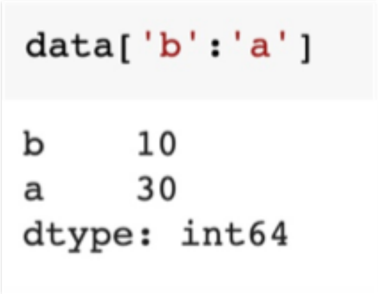
- Dictionary-style item access



```
data['a']
```

```
30
```

- Unlike a dictionary, Series supports array-style operations: e.g., slicing



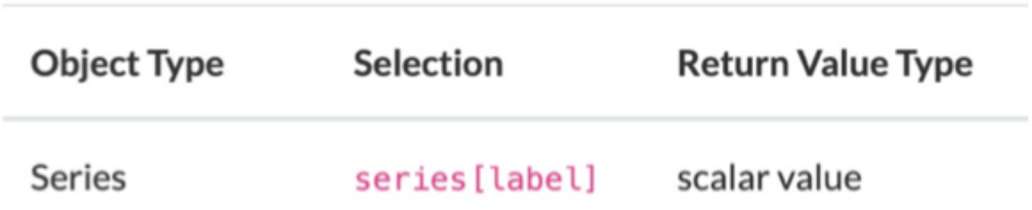
```
data['b':'a']
```

```
b    10
a    30
dtype: int64
```

Data Indexing and Selection in Series

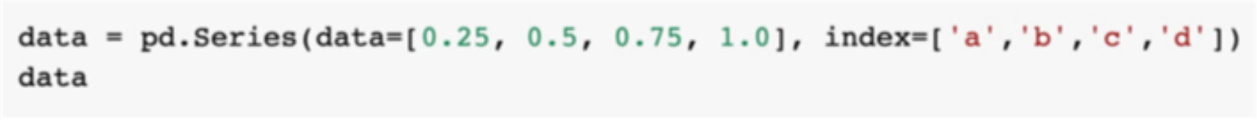
- In NumPy arrays, we use
  - Indexing arr[2,1]
  - Slicing arr[:, 1:5]
  - Masking arr[arr > 3]
  - …

- Series acts in many ways like a 1D ndarray, and in many ways like a dictionary

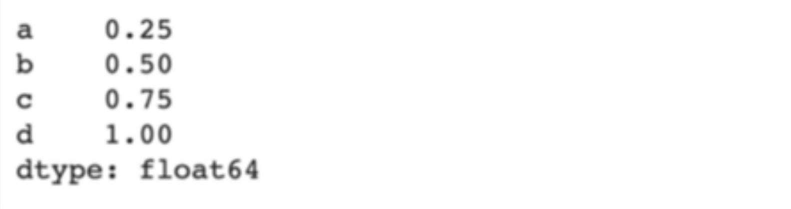


Object Type	Selection	Return Value Type
Series	`series[label]`	scalar value

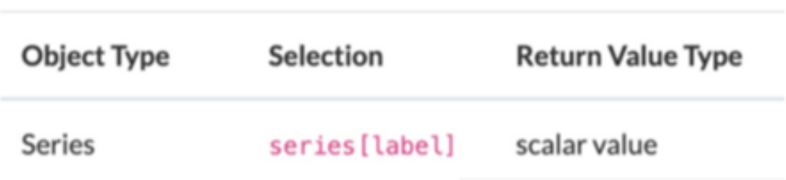
Data Indexing and Selection in Series



```
data = pd.Series(data=[0.25, 0.5, 0.75, 1.0], index=['a','b','c','d'])
data
```

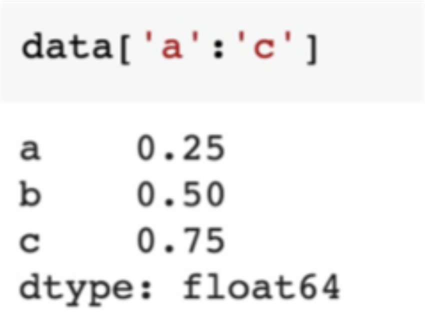


```
a    0.25
b    0.50
c    0.75
d    1.00
dtype: float64
```



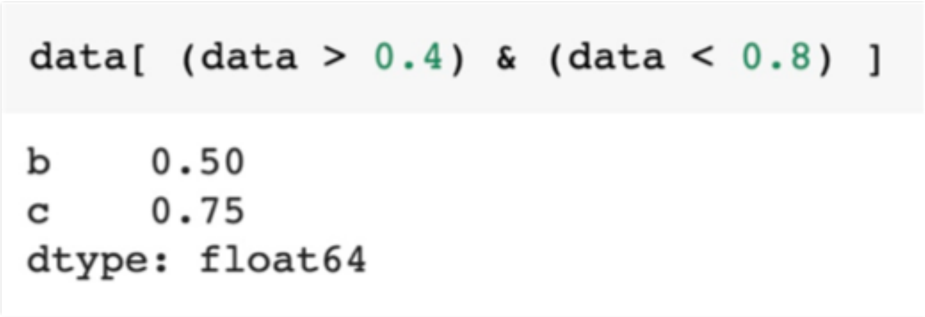
Object Type	Selection	Return Value Type
Series	series[label]	scalar value

- Same basic mechanisms as ndarrays



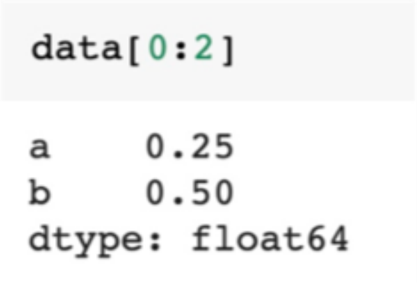
```
data['a':'c']
```

```
a    0.25
b    0.50
c    0.75
dtype: float64
```



```
data[ (data > 0.4) & (data < 0.8) ]
```

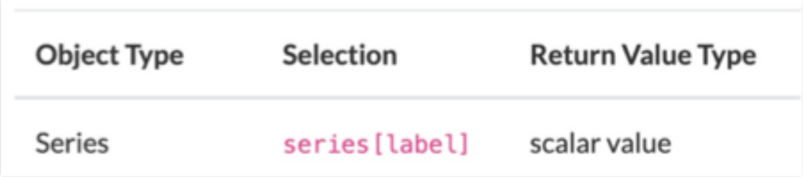
```
b    0.50
c    0.75
dtype: float64
```



```
data[0:2]
```

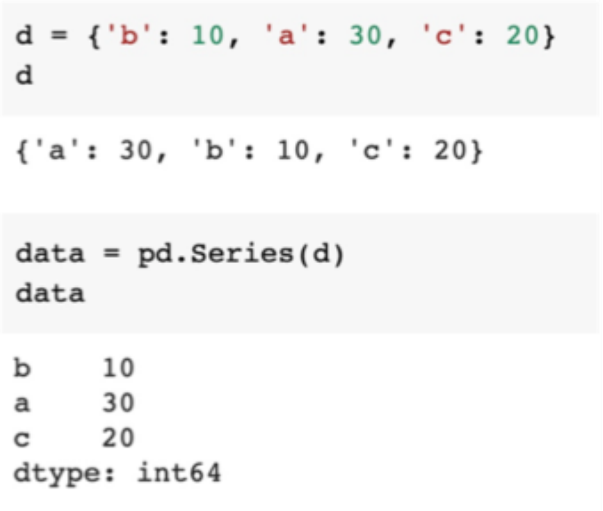
```
a    0.25
b    0.50
dtype: float64
```

Data Indexing and Selection in Series



Object Type	Selection	Return Value Type
Series	`series[label]`	scalar value

- Mapping from a collection of keys (=index) to a collection of values
- Extend a Series by assigning to a new index value

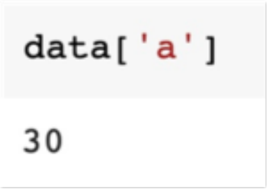


```
d = {'b': 10, 'a': 30, 'c': 20}
d
```

```
{'a': 30, 'b': 10, 'c': 20}
```

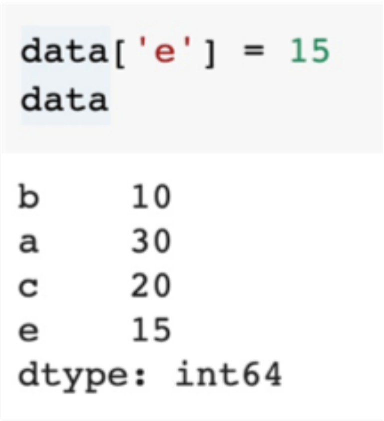
```
data = pd.Series(d)
data
```

```
b    10
a    30
c    20
dtype: int64
```



```
data['a']
```

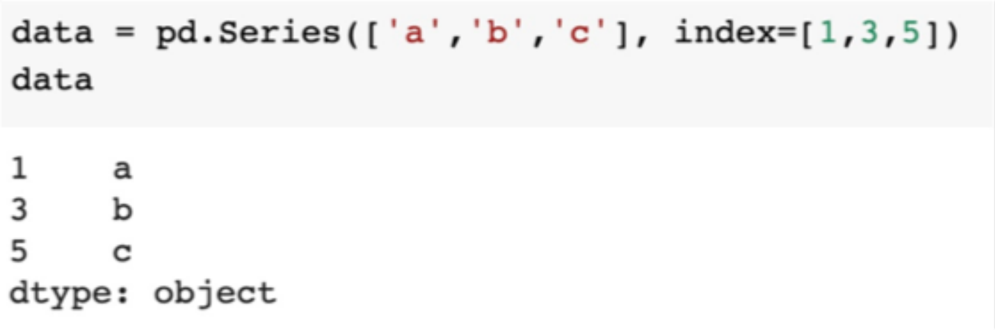
```
30
```



```
data['e'] = 15
data
```

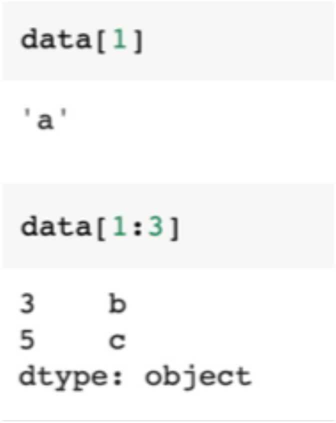
```
b    10
a    30
c    20
e    15
dtype: int64
```

Data Indexing and Selection in Series



```
data = pd.Series(['a','b','c'], index=[1,3,5])
data
```

```
1     a
3     b
5     c
dtype: object
```



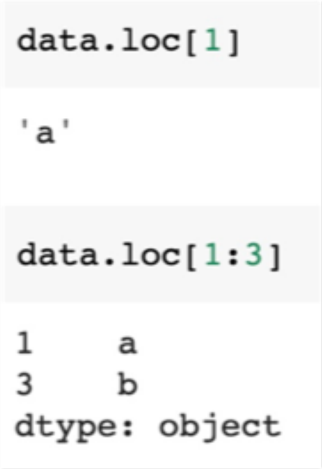
```
data[1]
```

```
'a'
```

```
data[1:3]
```

```
3     b
5     c
dtype: object
```

- **loc**
  - Indexing and slicing with **explicit index**



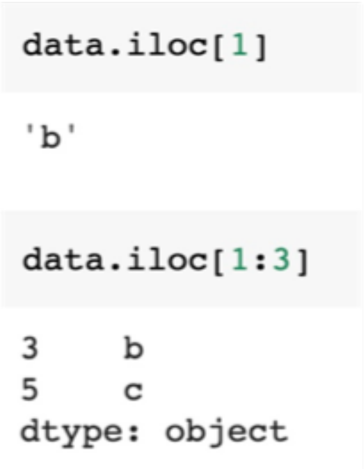
```
data.loc[1]
```

```
'a'
```

```
data.loc[1:3]
```

```
1     a
3     b
dtype: object
```

- **iloc**
  - Indexing and slicing with **implicit Python-style (integer) index**



```
data.iloc[1]
```

```
'b'
```

```
data.iloc[1:3]
```

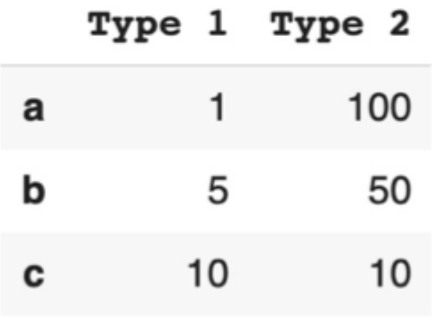
```
3     b
5     c
dtype: object
```

DataFrames

DataFrames

- 2D labeled data structure with columns of potentially different types
  - Dictionary of 1D ndarrays, lists, dicts, or Series
  - 2D ndarray
  - …

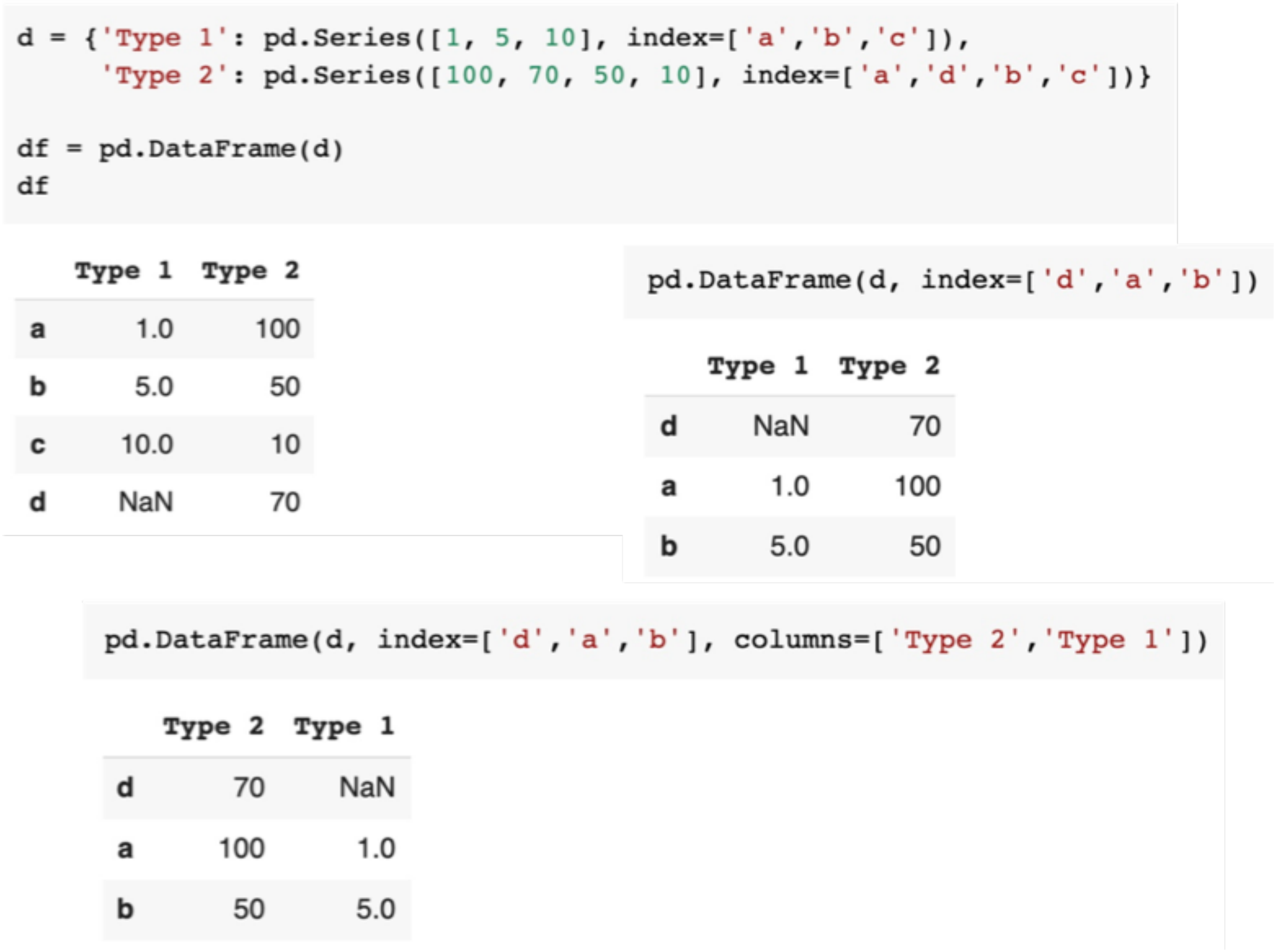
- **Index** = row labels
- **Columns** = column labels
- **Data** = values



	Type 1	Type 2
a	1	100
b	5	50
c	10	10

Creating DataFrames

- Dictionary of Series



```
d = {'Type 1': pd.Series([1, 5, 10], index=['a','b','c']),
     'Type 2': pd.Series([100, 70, 50, 10], index=['a','d','b','c'])}

df = pd.DataFrame(d)
df
```

	Type 1	Type 2
a	1.0	100
b	5.0	50
c	10.0	10
d	NaN	70

```
pd.DataFrame(d, index=['d','a','b'])
```

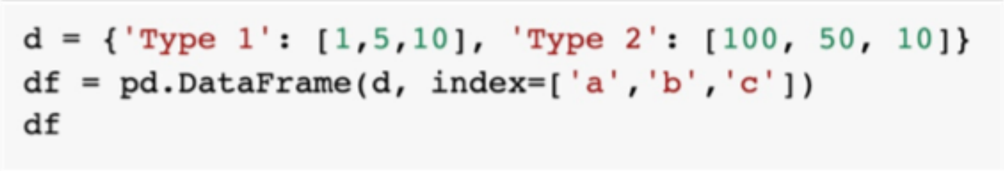
	Type 1	Type 2
d	NaN	70
a	1.0	100
b	5.0	50

```
pd.DataFrame(d, index=['d','a','b'], columns=['Type 2','Type 1'])
```

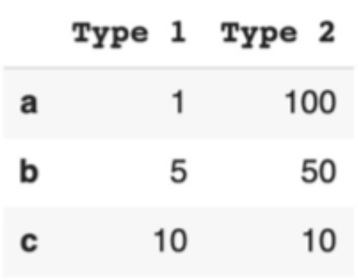
	Type 2	Type 1
d	70	NaN
a	100	1.0
b	50	5.0

Creating DataFrames

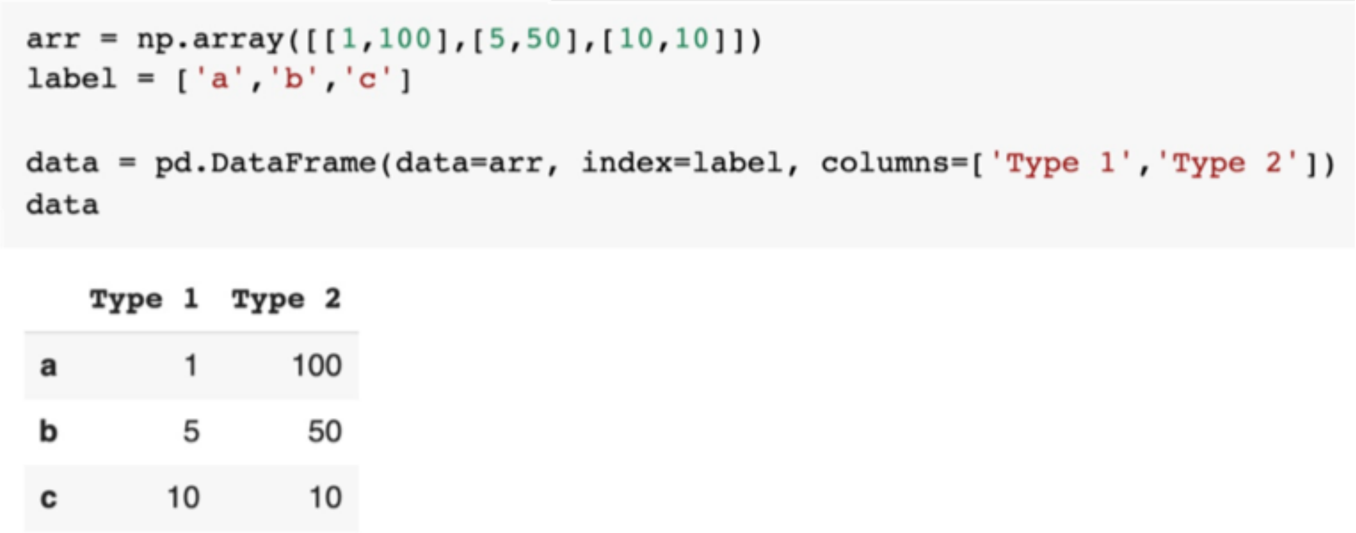
- Dictionary of Lists, Arrays



```python
d = {'Type 1': [1,5,10], 'Type 2': [100, 50, 10]}
df = pd.DataFrame(d, index=['a','b','c'])
df
```



	Type 1	Type 2
a	1	100
b	5	50
c	10	10



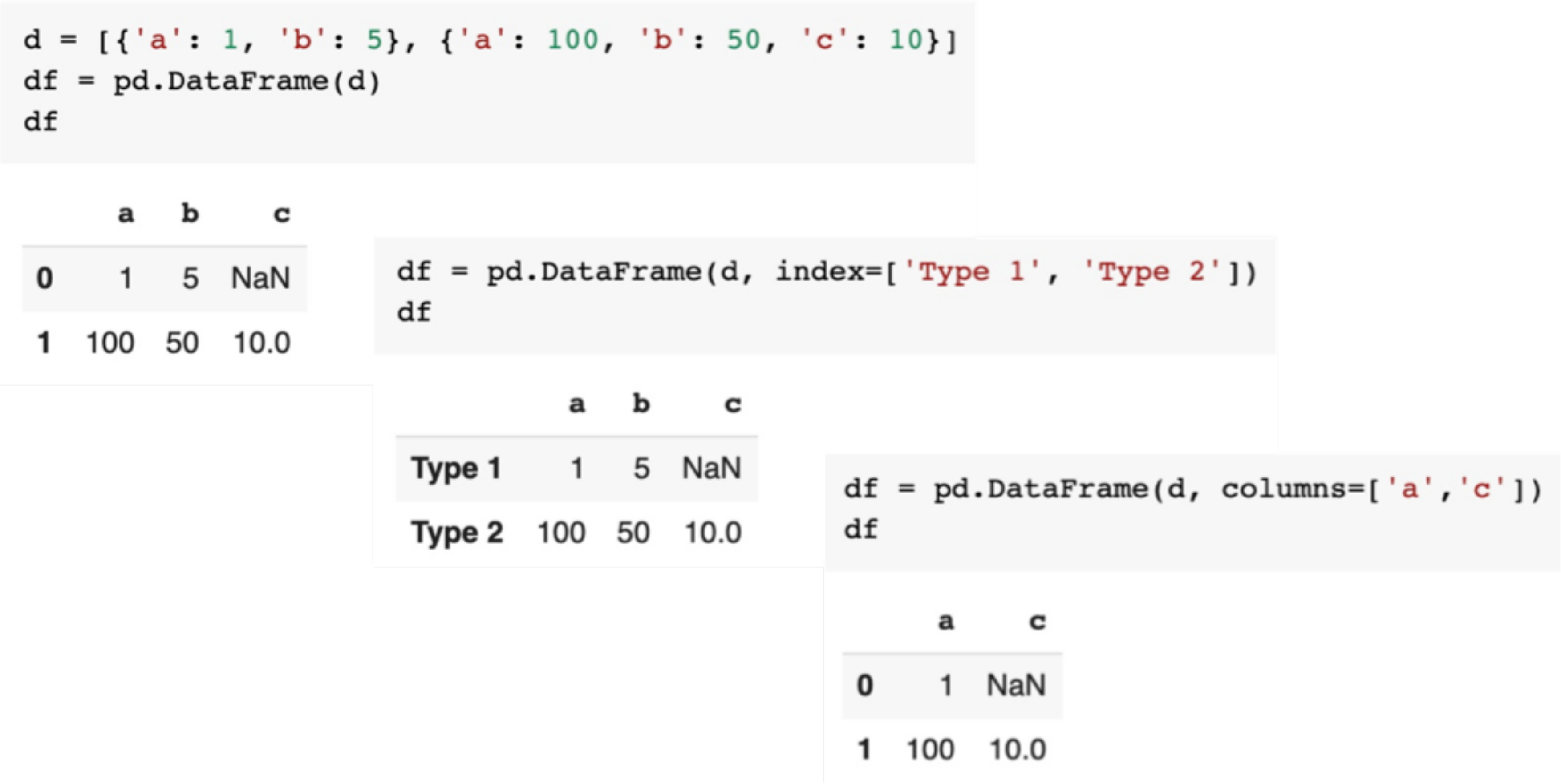
```python
arr = np.array([[1,100],[5,50],[10,10]])
label = ['a','b','c']

data = pd.DataFrame(data=arr, index=label, columns=['Type 1','Type 2'])
data
```

	Type 1	Type 2
a	1	100
b	5	50
c	10	10

Creating DataFrames

- List of Dictionaries



```
d = [{'a': 1, 'b': 5}, {'a': 100, 'b': 50, 'c': 10}]
df = pd.DataFrame(d)
df
```

```
     a   b     c
0    1   5   NaN
1  100  50  10.0
```

```
df = pd.DataFrame(d, index=['Type 1', 'Type 2'])
df
```

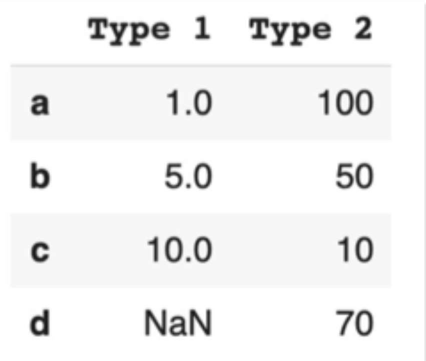
```
          a   b     c
Type 1    1   5   NaN
Type 2  100  50  10.0
```

```
df = pd.DataFrame(d, columns=['a','c'])
df
```

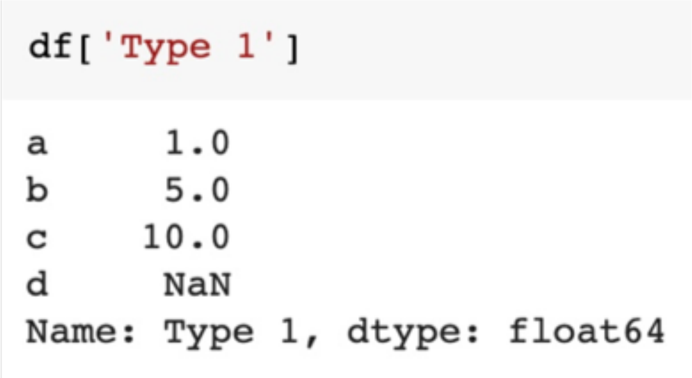
```
     a     c
0    1   NaN
1  100  10.0
```

DataFrame is dictionary-like

- Dictionary maps a **key** to a **value**
- DataFrame maps a **column name** to a **Series of column data**



	Type 1	Type 2
a	1.0	100
b	5.0	50
c	10.0	10
d	NaN	70



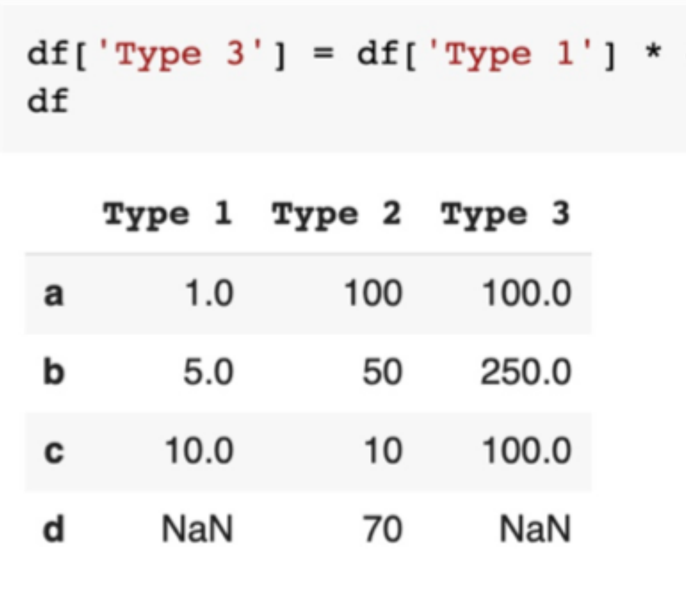
```
df['Type 1']
```

```
a      1.0
b      5.0
c     10.0
d      NaN
Name: Type 1, dtype: float64
```

- Getting, setting, and deleting columns = same syntax as the dict operations

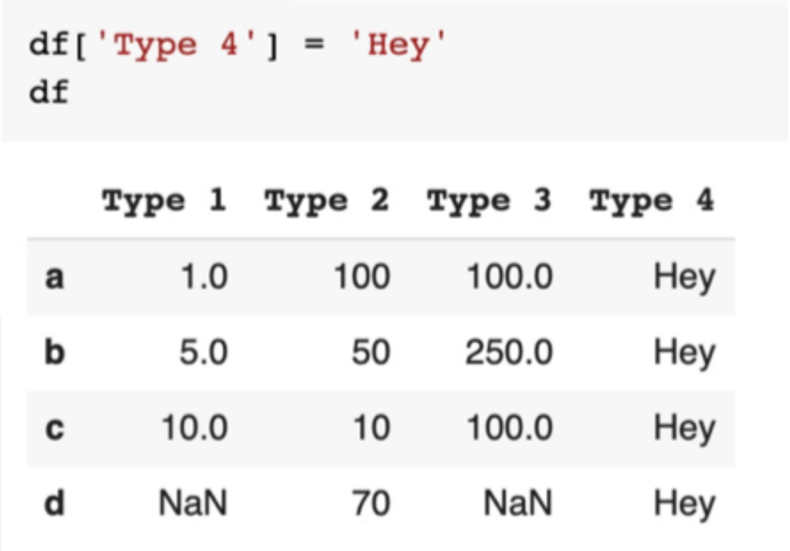
DataFrame is dictionary-like

- Getting, setting, and deleting columns = same syntax as the dict operations



```
df['Type 3'] = df['Type 1'] * df['Type 2']
df
```

	Type 1	Type 2	Type 3
a	1.0	100	100.0
b	5.0	50	250.0
c	10.0	10	100.0
d	NaN	70	NaN

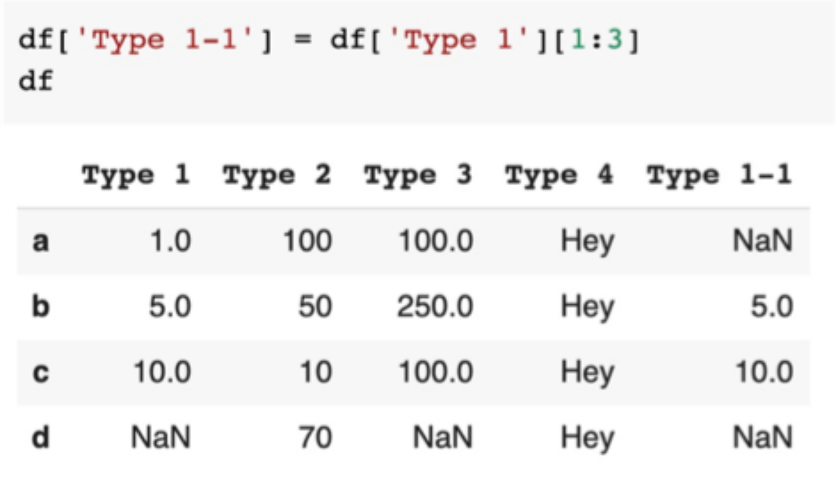


```
df['Type 4'] = 'Hey'
df
```

	Type 1	Type 2	Type 3	Type 4
a	1.0	100	100.0	Hey
b	5.0	50	250.0	Hey
c	10.0	10	100.0	Hey
d	NaN	70	NaN	Hey

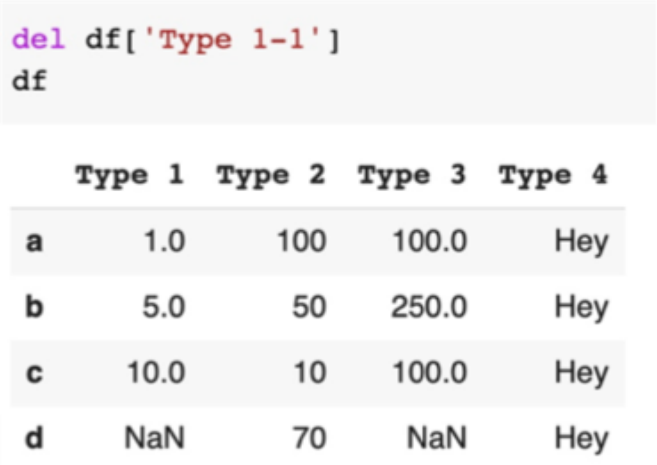
DataFrame is dictionary-like

- Getting, setting, and deleting columns = same syntax as the dict operations



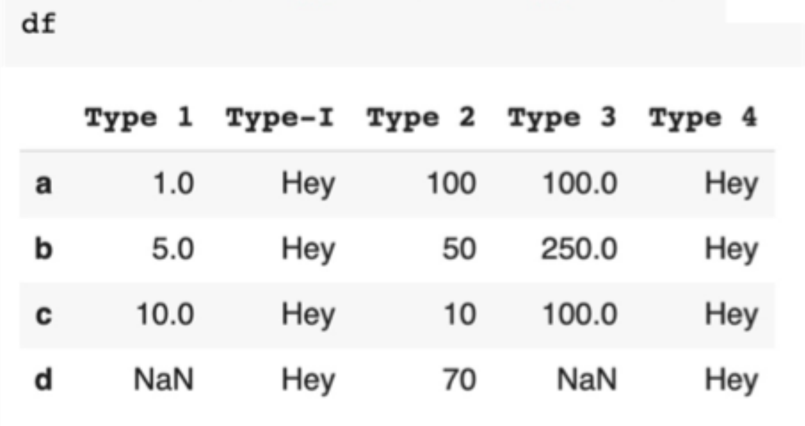
```
df['Type 1-1'] = df['Type 1'][1:3]
df
```

	Type 1	Type 2	Type 3	Type 4	Type 1-1
a	1.0	100	100.0	Hey	NaN
b	5.0	50	250.0	Hey	5.0
c	10.0	10	100.0	Hey	10.0
d	NaN	70	NaN	Hey	NaN



```
del df['Type 1-1']
df
```

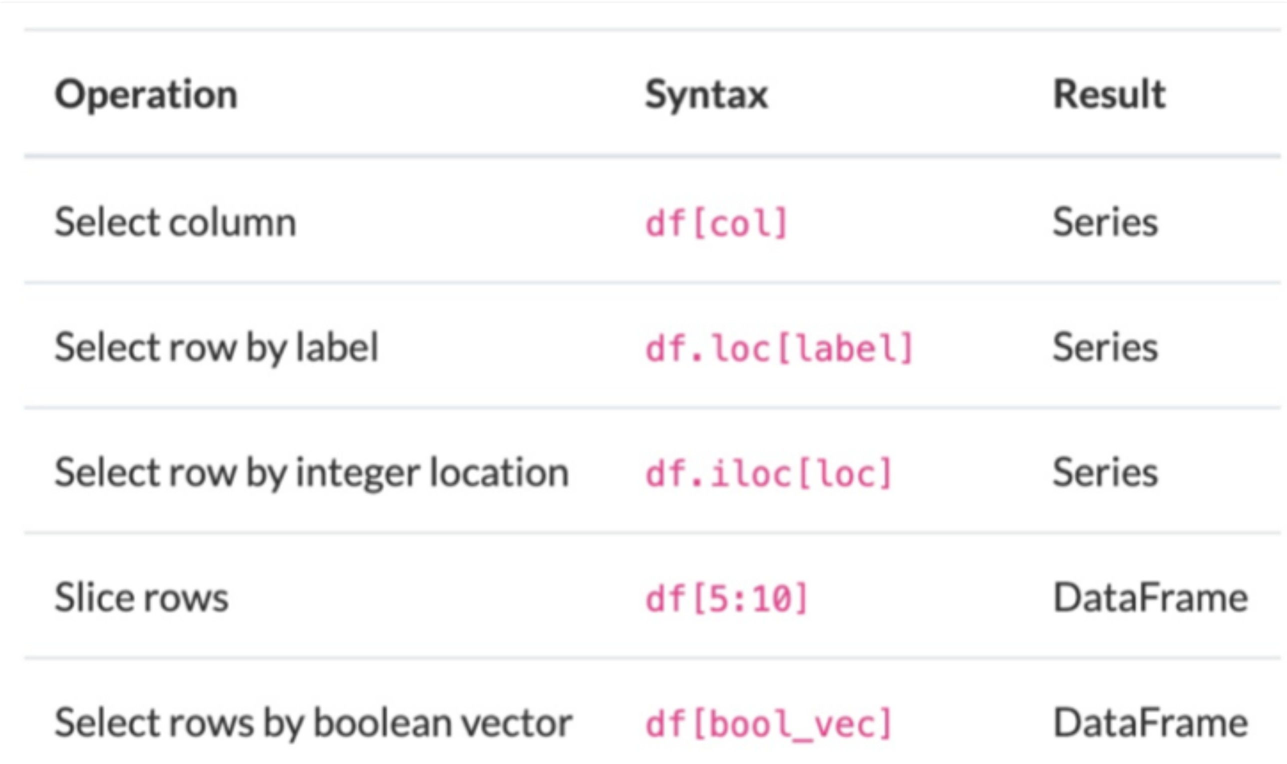
	Type 1	Type 2	Type 3	Type 4
a	1.0	100	100.0	Hey
b	5.0	50	250.0	Hey
c	10.0	10	100.0	Hey
d	NaN	70	NaN	Hey



```
df.insert(1, 'Type-I', df['Type 4'])
df
```

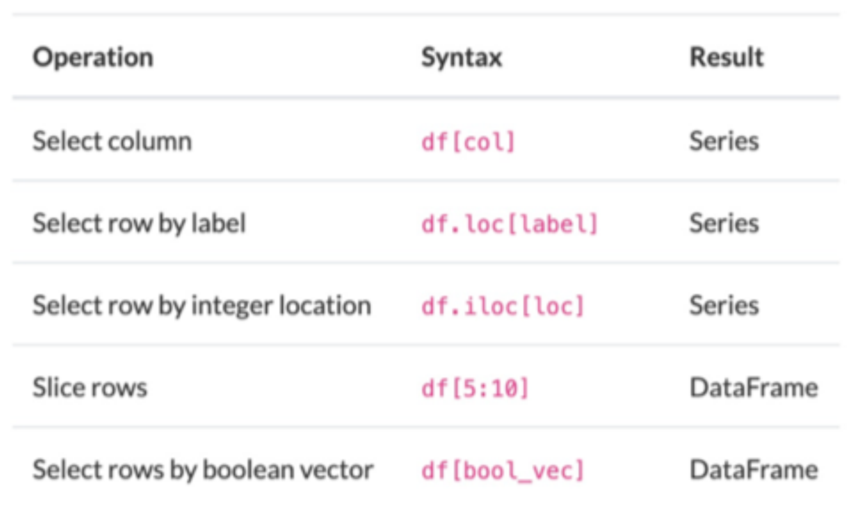
	Type 1	Type-I	Type 2	Type 3	Type 4
a	1.0	Hey	100	100.0	Hey
b	5.0	Hey	50	250.0	Hey
c	10.0	Hey	10	100.0	Hey
d	NaN	Hey	70	NaN	Hey

Data Indexing and Selection in DataFrames

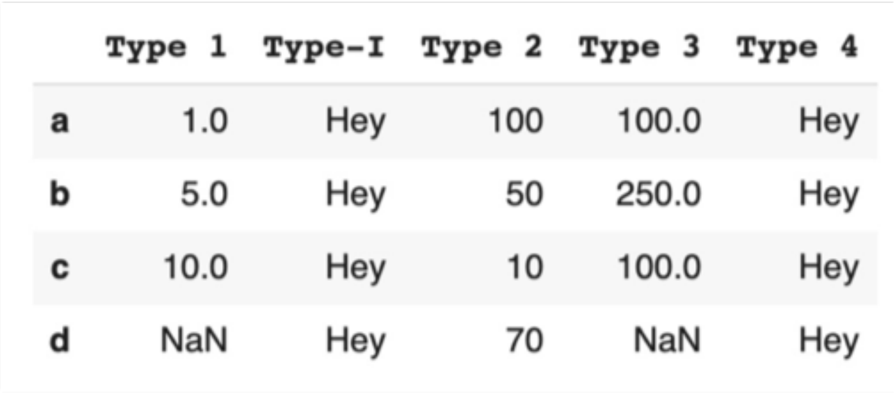


Operation	Syntax	Result
Select column	`df[col]`	Series
Select row by label	`df.loc[label]`	Series
Select row by integer location	`df.iloc[loc]`	Series
Slice rows	`df[5:10]`	DataFrame
Select rows by boolean vector	`df[bool_vec]`	DataFrame

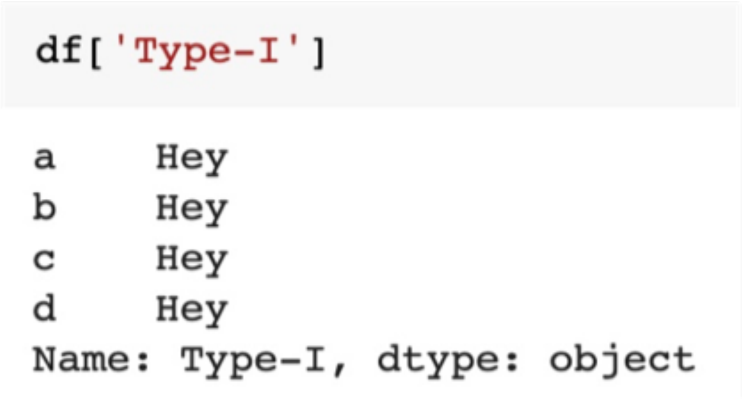
Data Indexing and Selection in DataFrames



Operation	Syntax	Result
Select column	`df[col]`	Series
Select row by label	`df.loc[label]`	Series
Select row by integer location	`df.iloc[loc]`	Series
Slice rows	`df[5:10]`	DataFrame
Select rows by boolean vector	`df[bool_vec]`	DataFrame



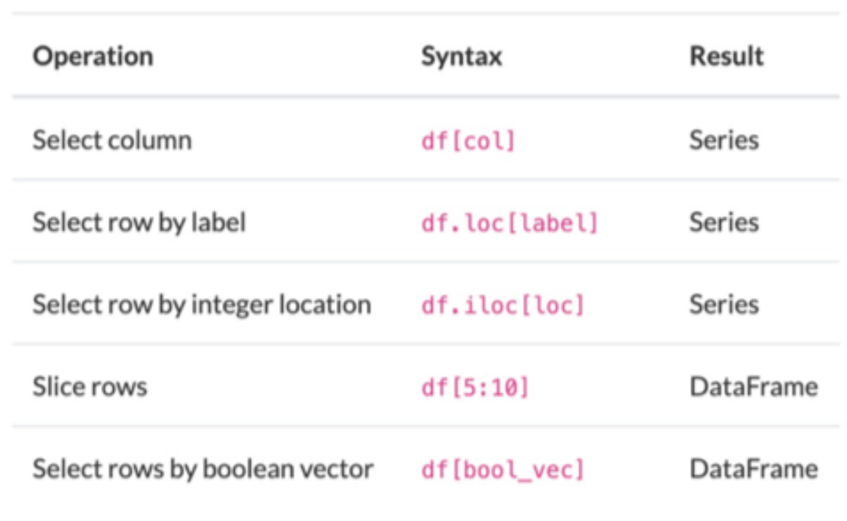
	Type 1	Type-I	Type 2	Type 3	Type 4
a	1.0	Hey	100	100.0	Hey
b	5.0	Hey	50	250.0	Hey
c	10.0	Hey	10	100.0	Hey
d	NaN	Hey	70	NaN	Hey



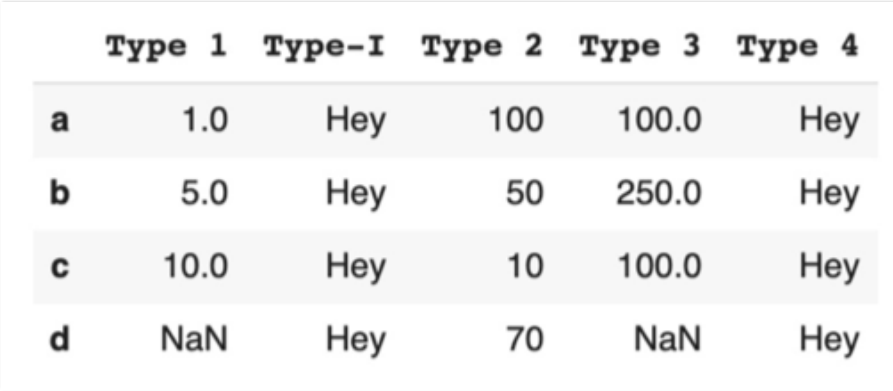
```
df['Type-I']
```

```
a      Hey
b      Hey
c      Hey
d      Hey
Name: Type-I, dtype: object
```

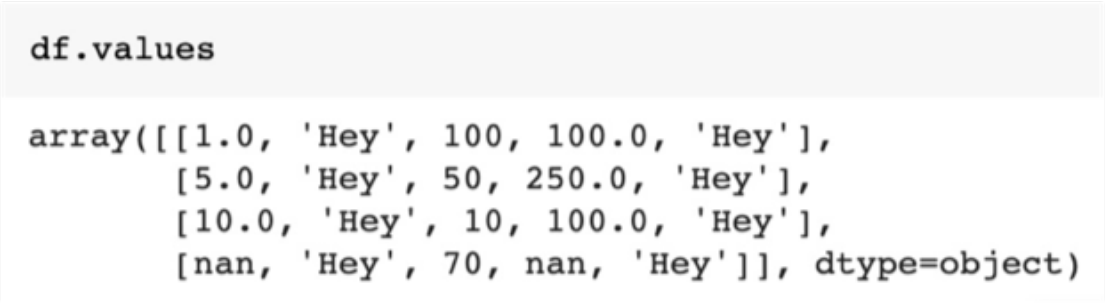
Data Indexing and Selection in DataFrames



Operation	Syntax	Result
Select column	df[col]	Series
Select row by label	df.loc[label]	Series
Select row by integer location	df.iloc[loc]	Series
Slice rows	df[5:10]	DataFrame
Select rows by boolean vector	df[bool_vec]	DataFrame

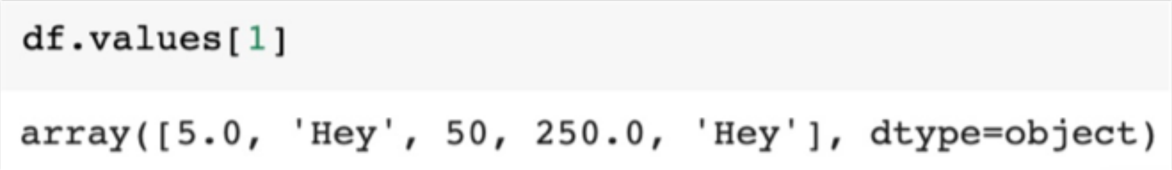


	Type 1	Type-I	Type 2	Type 3	Type 4
a	1.0	Hey	100	100.0	Hey
b	5.0	Hey	50	250.0	Hey
c	10.0	Hey	10	100.0	Hey
d	NaN	Hey	70	NaN	Hey



```
df.values
```

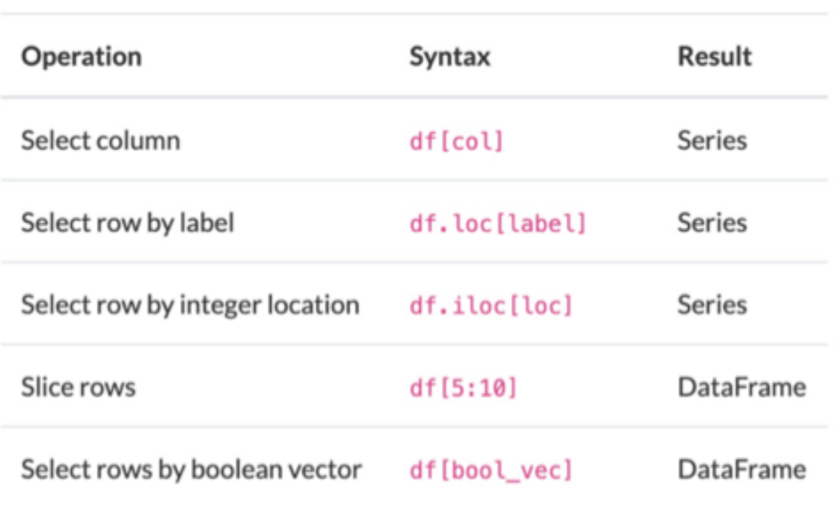
```
array([[1.0, 'Hey', 100, 100.0, 'Hey'],
       [5.0, 'Hey', 50, 250.0, 'Hey'],
       [10.0, 'Hey', 10, 100.0, 'Hey'],
       [nan, 'Hey', 70, nan, 'Hey']], dtype=object)
```



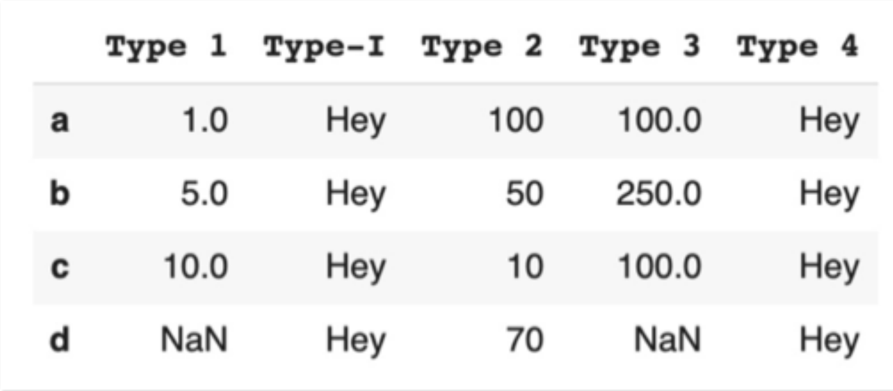
```
df.values[1]
```

```
array([5.0, 'Hey', 50, 250.0, 'Hey'], dtype=object)
```

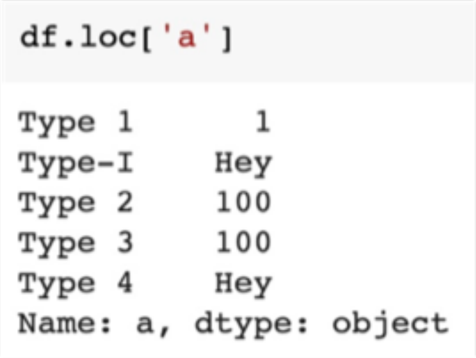
Data Indexing and Selection in DataFrames



Operation	Syntax	Result
Select column	`df[col]`	Series
Select row by label	`df.loc[label]`	Series
Select row by integer location	`df.iloc[loc]`	Series
Slice rows	`df[5:10]`	DataFrame
Select rows by boolean vector	`df[bool_vec]`	DataFrame

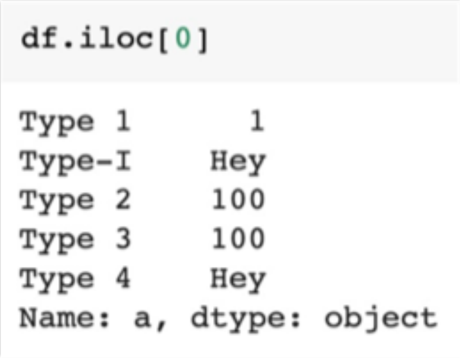


	Type 1	Type-I	Type 2	Type 3	Type 4
a	1.0	Hey	100	100.0	Hey
b	5.0	Hey	50	250.0	Hey
c	10.0	Hey	10	100.0	Hey
d	NaN	Hey	70	NaN	Hey



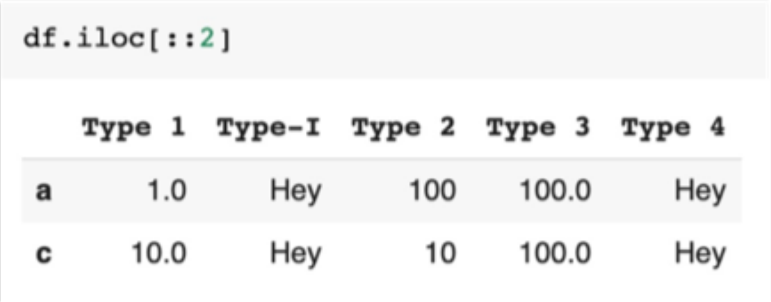
```
df.loc['a']
```

```
Type 1       1
Type-I     Hey
Type 2     100
Type 3     100
Type 4     Hey
Name: a, dtype: object
```



```
df.iloc[0]
```

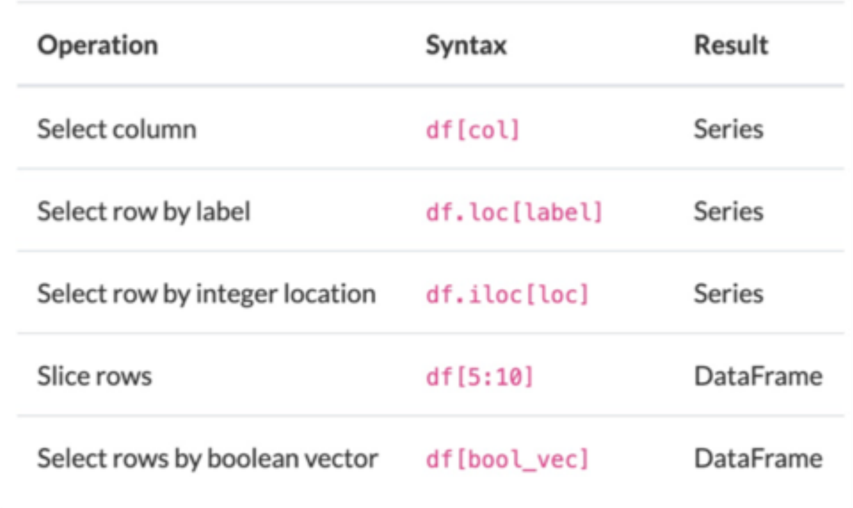
```
Type 1       1
Type-I     Hey
Type 2     100
Type 3     100
Type 4     Hey
Name: a, dtype: object
```



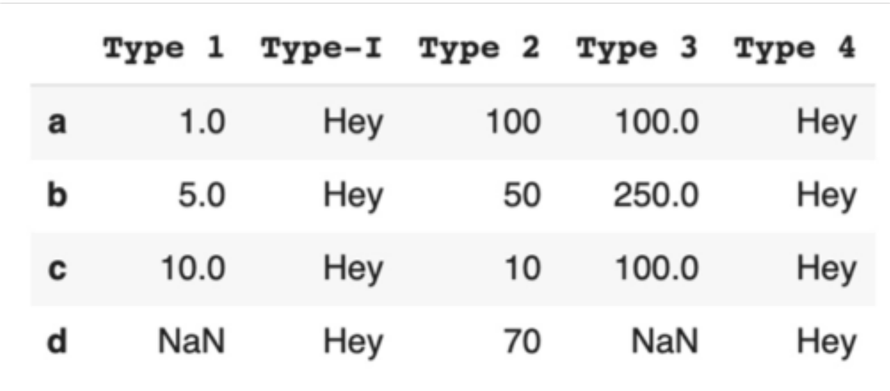
```
df.iloc[::2]
```

	Type 1	Type-I	Type 2	Type 3	Type 4
a	1.0	Hey	100	100.0	Hey
c	10.0	Hey	10	100.0	Hey

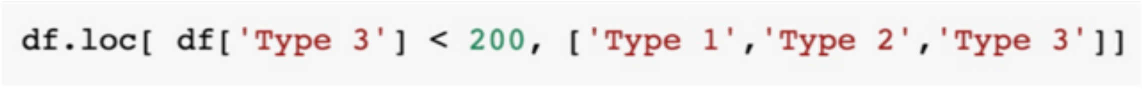
Data Indexing and Selection in DataFrames



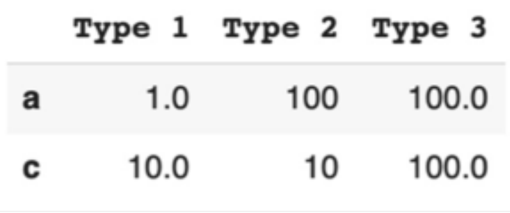
Operation	Syntax	Result
Select column	`df[col]`	Series
Select row by label	`df.loc[label]`	Series
Select row by integer location	`df.iloc[loc]`	Series
Slice rows	`df[5:10]`	DataFrame
Select rows by boolean vector	`df[bool_vec]`	DataFrame



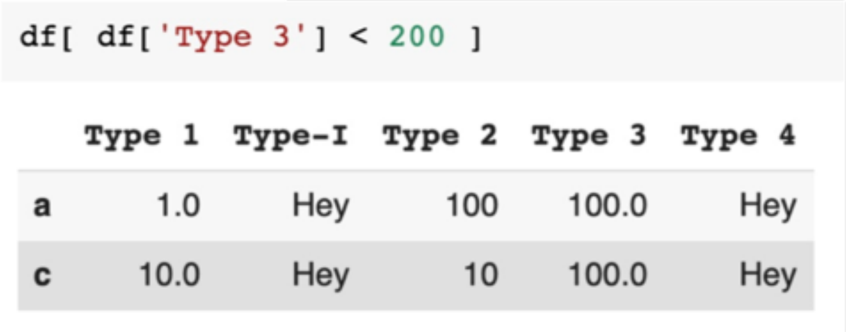
	Type 1	Type-I	Type 2	Type 3	Type 4
a	1.0	Hey	100	100.0	Hey
b	5.0	Hey	50	250.0	Hey
c	10.0	Hey	10	100.0	Hey
d	NaN	Hey	70	NaN	Hey



```
df.loc[ df['Type 3'] < 200, ['Type 1','Type 2','Type 3']]
```



	Type 1	Type 2	Type 3
a	1.0	100	100.0
c	10.0	10	100.0



```
df[ df['Type 3'] < 200 ]
```

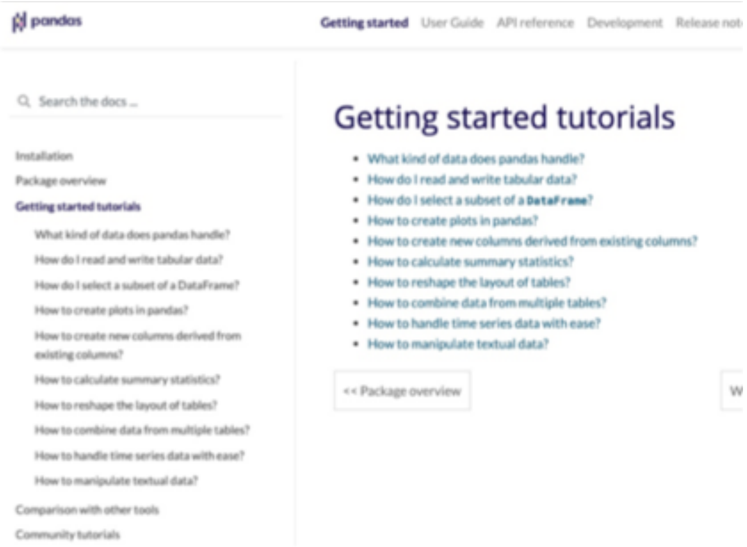
	Type 1	Type-I	Type 2	Type 3	Type 4
a	1.0	Hey	100	100.0	Hey
c	10.0	Hey	10	100.0	Hey

In this lesson, you have learned:

- Pandas
- Pandas Object
  - Series
  - DataFrames
- Indexing and Selection in Pandas

pandas

pandas.pydata.org



Installation

Package overview

Getting started tutorials

- What kind of data does pandas handle?
- How do I read and write tabular data?
- How do I select a subset of a DataFrame?
- How to create plots in pandas?
- How to create new columns derived from existing columns?
- How to calculate summary statistics?
- How to reshape the layout of tables?
- How to combine data from multiple tables?
- How to handle time series data with ease?
- How to manipulate textual data?

Comparison with other tools

Community tutorials

Getting started tutorials

- What kind of data does pandas handle?
- How do I read and write tabular data?
- How do I select a subset of a `DataFrame`?
- How to create plots in pandas?
- How to create new columns derived from existing columns?
- How to calculate summary statistics?
- How to reshape the layout of tables?
- How to combine data from multiple tables?
- How to handle time series data with ease?
- How to manipulate textual data?

Thank you!

Any Questions?

hyeryung.jang@dgu.ac.kr