



ĐẠI HỌC ĐÀ NẴNG

**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

## CHƯƠNG 3: LẬP TRÌNH PYTHON

### CHO KHDL

#### 3.3. Thao tác dữ liệu với Pandas



Khoa Công nghệ thông tin

TS. Phạm Công Thắng

D  
BACH KHOA

N  
A  
N  
G

# Tài liệu tham khảo

- Jake VanderPlas, *Python Data Science Handbook: Essential Tools for Working with Data*, O'Reilly Media, 2016
- Pandas documentation: <https://pandas.pydata.org/docs/>
- API reference: <https://pandas.pydata.org/pandas-docs/stable/reference/api/>

# Nội dung

---

- Giới thiệu chung
- Các đối tượng Pandas
- Định vị và lựa chọn dữ liệu
- Thao tác dữ liệu trong Pandas
- Định chỉ số phân cấp
- Đọc/ghi dữ liệu với file CSV

# Giới thiệu về Pandas

- Pandas là một package được xây dựng dựa trên NumPy và cung cấp khả năng thực thi hiệu quả đối tượng *DataFrame*
- DataFrame là các mảng đa chiều có các hàng và cột được gán nhãn, thường chứa dữ liệu có kiểu không đồng nhất và/hoặc dữ liệu bị thiếu
- Pandas cài đặt nhiều thao tác xử lý dữ liệu mạnh mẽ quen thuộc với người dùng của các hệ cơ sở dữ liệu (vd: SQL) và chương trình bảng tính (vd: MS-Excel)

# Cài đặt và sử dụng Pandas

- Cài đặt
  - Cần cài đặt NumPy trước
  - Cách cài đặt: tham khảo <https://pandas.pydata.org/>
- Sử dụng
  - Thường import Pandas với "bí danh" là pd

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

- Kiểm tra phiên bản đã được cài đặt

```
pd.__version__  
'0.25.1'
```

# Nội dung

---

- Giới thiệu chung
- **Các đối tượng Pandas**
- Định vị và lựa chọn dữ liệu
- Thao tác dữ liệu trong Pandas
- Định chỉ số phân cấp
- Đọc/ghi dữ liệu với file CSV

# Các đối tượng Pandas

- Có thể xem như phiên bản được tăng cường của NumPy array với các hàng và cột được đánh chỉ số bằng nhãn (label) thay cho chỉ số nguyên
- 03 cấu trúc dữ liệu cơ bản là
  - **Series**
  - DataFrame
  - Index

# Đối tượng Series

- Series là mảng 1 chiều gồm dữ liệu được đánh chỉ số (indexed)
- Nó có thể được tạo từ 1 danh sách:

```
data = pd.Series([0.25, 0.5, 0.75, 1.0])  
data  
0 0.25  
1 0.50  
2 0.75  
3 1.00  
dtype: float64
```

- Series bao gồm cả chuỗi giá trị và chuỗi chỉ số mà chúng ta có thể truy cập bằng các thuộc tính **values** và **index**

```
data.values  
array([ 0.25, 0.5 , 0.75, 1. ]) # a NumPy array
```

```
data[1]  
0.5
```

```
data[1:3]  
1 0.50  
2 0.75  
dtype: float64
```

```
data.index  
RangeIndex(start=0, stop=4, step=1) # an array-like object of type pd.Index
```



# Series như là mảng NumPy tổng quát

- Series về cơ bản giống với mảng NumPy một chiều
- Sự khác biệt chủ yếu nằm ở chỉ số
  - Mảng NumPy có chỉ số nguyên ngầm định
  - Series có chỉ số tường minh

# Series như là mảng NumPy tổng quát

- Chỉ số của Series không cần phải là số nguyên mà có thể chứa giá trị có kiểu bất kỳ

```
# use strings as indices
```

```
data = pd.Series([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
```

```
# use noncontiguous or nonsequential indices
```

```
data = pd.Series([0.25, 0.5, 0.75, 1.0], index=[2, 5, 3, 7])
```

```
data
```

```
2    0.25
```

```
5    0.50
```

```
3    0.75
```

```
7    1.00
```

```
dtype: float64
```

# Series như là từ điển chuyên dụng

- Từ điển là cấu trúc ánh xạ các khoá bất kỳ sang các giá trị bất kỳ
- Series là cấu trúc ánh xạ các khoá định kiểu sang các giá trị định kiểu
- Thông tin kiểu của Series làm nó thực thi hiệu quả hơn nhiều so với Từ điển trong một số thao tác

# Series như là từ điển chuyên dụng

- Có thể tạo Series từ Từ điển:

```
population_dict = {'California': 38332521, 'Texas': 26448193,  
'New York': 19651127, 'Florida': 19552860}  
population = pd.Series(population_dict)  
population  
California    38332521  
Texas         26448193  
New York      19651127  
Florida       19552860  
dtype: int64
```

- Truy nhập giá trị thông qua khoá tương tự như Từ điển:

```
population['California']  
38332521
```

- Khác với Từ điển, Series hỗ trợ các thao tác mảng như “cắt lát” (slicing):

```
population['California':'New York']  
California    38332521  
Texas         26448193  
New York      19651127  
dtype: int64
```

# Tạo các đối tượng Series

Cú pháp rút gọn:

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

với **data** là đối tượng chứa dữ liệu của Series (có kiểu *array-like*, *Iterable*, *dict*, hay *scalar value*)  
**index** là chuỗi giá trị có cùng độ dài với **data** (có kiểu *array-like* hay *Index (1d)*)

*# data can be a list or NumPy array, index defaults to an integer sequence*

```
pd.Series([2, 4, 6])
```

```
0    2
```

```
1    4
```

```
2    6
```

```
dtype: int64
```

*# data can be a scalar, which is repeated to fill the specified index*

```
pd.Series(5, index=[100, 200, 300])
```

```
100    5
```

```
200    5
```

```
300    5
```

```
dtype: int64
```

# Tạo các đối tượng Series

*# data can be a dictionary, in which index defaults to the dictionary keys*

```
pd.Series({'2':'a', '1':'b', '3':'c'})
```

```
2    a
```

```
1    b
```

```
3    c
```

```
dtype: object
```

*# index can be explicitly set if a different result is preferred*

```
pd.Series({'2':'a', '1':'b', '3':'c'}, index=[3, 2])
```

```
3    c
```

```
2    a
```

```
dtype: object
```

# Các đối tượng Pandas

- Có thể xem như phiên bản được tăng cường của NumPy array với các hàng và cột được đánh chỉ số bằng nhãn thay cho chỉ số nguyên
- 03 cấu trúc dữ liệu cơ bản là
  - Series
  - **DataFrame**
  - Index

# DataFrame như là mảng NumPy tổng quát

- DataFrame giống với mảng NumPy 2 chiều có tên cột và chỉ số hàng
- Có thể xem DataFrame như chuỗi gồm các Series chia sẻ cùng chỉ số

*# First, construct a Series listing the area of 3 states*

```
area_dict = {'California': 423967, 'Texas': 695662, 'New York': 141297}
```

```
area = pd.Series(area_dict)
```

```
area
```

```
California    423967
```

```
Texas        695662
```

```
New York     141297
```

```
dtype: int64
```

*# Second, construct a Series listing the population of 3 states*

```
population_dict = {'California': 38332521, 'Texas': 26448193, 'New York': 19651127}
```

```
population = pd.Series(population_dict)
```

```
population
```

```
California    38332521
```

```
Texas        26448193
```

```
New York     19651127
```

```
dtype: int64
```



# DataFrame như là mảng NumPy tổng quát

- Có thể xem DataFrame như chuỗi gồm các Series chia sẻ cùng chỉ số (tiếp vd trước)

```
# Final, use a dictionary to create a DataFrame object containing this information
states = pd.DataFrame({'population': population, 'area': area})
states
```

	population	area
California	38332521	423967
Texas	26448193	695662
New York	19651127	141297

- DataFrame cũng có thuộc tính **index** để truy nhập các chỉ số (hàng)

```
states.index
Index(['California', 'Texas', 'New York'], dtype='object')
```

- DataFrame có thêm thuộc tính **columns**, là đối tượng **Index** chứa các nhãn cột

```
states.columns
Index(['population', 'area'], dtype='object')
```

# DataFrame như là từ điển chuyên dụng

- DataFrame ánh xạ tên cột sang đối tượng Series chứa dữ liệu cột

```
states['area']  
California    423967  
Texas         695662  
New York     141297  
Name: area, dtype: int64
```

- Chú ý điểm dễ gây nhầm lẫn:
    - Trong mảng 2 chiều NumPy, `data[0]` trả lại hàng đầu tiên
    - Trong DataFrame, `data['column0']` trả lại cột đầu tiên
- Tốt hơn là nên xem DataFrame như từ điển (thay vì mảng NumPy)

# Tạo các đối tượng DataFrame

Cú pháp rút gọn:

```
pd.DataFrame(data=None, index=None, columns=None)
```

với **data** là đối tượng có kiểu *ndarray*, *Iterable*, *dict*, or *DataFrame*

**index** là chỉ số dùng cho DataFrame (kiểu *Index* or *array-like*)

**columns** là nhãn cột dùng cho DataFrame (kiểu *Index* or *array-like*)

*# create a DataFrame with any specified column and index names from a 2D NumPy array*

```
pd.DataFrame(np.random.rand(3, 2), columns=['foo', 'bar'], index=['a', 'b', 'c'])
```

	foo	bar
a	0.872485	0.749756
b	0.522873	0.412174
c	0.827166	0.327935

# Tạo các đối tượng DataFrame

*# Any list of dictionaries can be made into a DataFrame.*

```
data = [{'a': i, 'b': 2 * i} for i in range(3)]  
pd.DataFrame(data)
```

	a	b
0	0	0
1	1	2
2	2	4

*# if some keys in the dictionary are missing, Pandas will fill them in with NaN values*

```
pd.DataFrame([{'a': 1, 'b': 2}, {'b': 3, 'c': 4}])
```

	a	b	c
0	1.0	2	NaN
1	NaN	3	4.0

# Tạo các đối tượng DataFrame

*# a DataFrame can be constructed from a dictionary of Series objects*

```
pd.DataFrame({'population': population, 'area': area})
```

	population	area
California	38332521	423967
Texas	26448193	695662
New York	19651127	141297

*# a single column DataFrame can be constructed from a single Series*

```
pd.DataFrame(population, columns=['population'])
```

	population
California	38332521
Texas	26448193
New York	19651127

# Các đối tượng Pandas

- Có thể xem như phiên bản được tăng cường của NumPy array với các hàng và cột được đánh chỉ số bằng nhãn thay cho chỉ số nguyên
- 03 cấu trúc dữ liệu cơ bản là
  - Series
  - DataFrame
  - **Index**

# Đối tượng Index

- Cả Series và DataFrame đều chứa chỉ số (index) cho phép tham chiếu và chỉnh sửa dữ liệu
- Đối tượng Index có thể xem như mảng bất biến (immutable array) hoặc tập hợp có thứ tự (ordered set)
- Hệ quả là có các phép xử lý thú vị trên đối tượng Index

# Index xem như mảng bất biến

- Index hoạt động giống với mảng NumPy trong nhiều trường hợp

```
# construct an Index from a list of integers  
ind = pd.Index([2, 3, 5, 7, 11])  
ind  
Int64Index([2, 3, 5, 7, 11], dtype='int64')
```

```
# use standard Python indexing notation to retrieve values or slices  
ind[1]  
3  
ind[::2]  
Int64Index([2, 5, 11], dtype='int64')
```

- Index cũng có nhiều thuộc tính giống với mảng NumPy

```
print(ind.size, ind.shape, ind.ndim, ind.dtype)  
5 (5,) 1 int64
```



# Index xem như mảng bất biến

- Một khác biệt giữa Index và mảng NumPy là Index là bất biến (immutable)

```
ind[1] = 0
```

```
...
```

```
TypeError: Index does not support mutable operations
```

- Tính chất này đảm bảo cho việc dùng chung các chỉ số giữa nhiều DataFrame hoặc nhiều mảng NumPy mà không lo ngại việc chỉ số bị thay đổi

# Index xem như tập hợp có thứ tự

- Các đối tượng Pandas được thiết kế để thực hiện các thao tác phụ thuộc vào phép toán trên tập hợp (vd như kết hợp các tập dữ liệu)
  - Index tuân theo nhiều quy tắc của cấu trúc dữ liệu tập hợp (set) của Python
- Hợp, giao, hiệu và các tổ hợp khác có thể tính một cách dễ dàng

```
indA = pd.Index([1, 3, 5, 7, 9])  
indB = pd.Index([2, 3, 5, 7, 11])  
indA & indB # intersection  
Int64Index([3, 5, 7], dtype='int64')
```

```
indA | indB # union  
Int64Index([1, 2, 3, 5, 7, 9, 11], dtype='int64')
```

```
indA ^ indB # symmetric difference  
Int64Index([1, 2, 9, 11], dtype='int64')
```

# Nội dung

---

- Giới thiệu chung
- Các đối tượng Pandas
- Định vị và lựa chọn dữ liệu
- Thao tác dữ liệu trong Pandas
- Định chỉ số phân cấp
- Đọc/ghi dữ liệu với file CSV

# Lựa chọn dữ liệu với Series

- Series như từ điển

```
data = pd.Series([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
```

*# Series provides a mapping from keys to values*

```
data['b']
```

```
0.5
```

*# use dictionary-like Python expressions and methods to examine  
# the keys/indices and values*

```
'a' in data
```

```
True
```

```
data.keys()
```

```
Index(['a', 'b', 'c', 'd'], dtype='object')
```

```
list(data.items())
```

```
[('a', 0.25), ('b', 0.5), ('c', 0.75), ('d', 1.0)]
```

# Lựa chọn dữ liệu với Series

- Series như từ điển

*# Just as you can extend a dictionary by assigning to a new key,*

*# you can extend a Series by assigning to a new index value*

```
data['e'] = 1.25
```

```
data
```

```
a    0.25
```

```
b    0.50
```

```
c    0.75
```

```
d    1.00
```

```
e    1.25
```

```
dtype: float64
```

# Lựa chọn dữ liệu với Series

- Series như mảng 1 chiều

*# slicing by explicit index*

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

```
a    0.25
```

```
b    0.50
```

```
c    0.75
```

```
dtype: float64
```

*# slicing by implicit integer index*

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

```
a    0.25
```

```
b    0.50
```

```
dtype: float64
```

*# masking*

```
data[(data > 0.3) & (data < 0.8)]
```

```
b    0.50
```

```
c    0.75
```

```
dtype: float64
```

*# fancy indexing*

```
data[['a', 'e']]
```

```
a    0.25
```

```
e    1.25
```

```
dtype: float64
```

# Lựa chọn dữ liệu với Series

- Các thuộc tính định vị trí (indexer): hiển thị một giao diện cắt lát (slicing interface) cụ thể cho dữ liệu trong Series
- Thuộc tính **loc** cho phép định vị và cắt lát luôn luôn tham chiếu đến chỉ số tường minh (explicit index)

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

```
data
```

```
1    a  
3    b  
5    c
```

```
dtype: object
```

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

```
'a'
```

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

```
1    a  
3    b
```

```
dtype: object
```

# Lựa chọn dữ liệu với Series

- Thuộc tính **iloc** cho phép định vị và cắt lát luôn luôn tham chiếu đến chỉ số ngầm định (implicit index) kiểu Python

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

```
data
```

```
1    a
```

```
3    b
```

```
5    c
```

```
dtype: object
```

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

```
'b'
```

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

```
3    b
```

```
5    c
```

```
dtype: object
```

Nên dùng **loc** và **iloc** để giúp code dễ đọc và tránh các bug do trộn lẫn các kiểu indexing/slicing khác nhau



# Lựa chọn dữ liệu với DataFrame

- DataFrame như là từ điển gồm các Series có chung chỉ số

*# create a DataFrame from a dictionary of Series*

```
area = pd.Series({'California': 423967, 'Texas': 695662, 'New York': 141297})
pop = pd.Series({'California': 38332521, 'Texas': 26448193, 'New York': 19651127})
data = pd.DataFrame({'area':area, 'pop':pop})
data
```

	area	pop
California	423967	38332521
Texas	695662	26448193
New York	141297	19651127

*# individual Series of the DataFrame can be accessed via*

*# dictionary-style indexing of the column name*

```
data['area']
```

```
California    423967
Texas         695662
New York      141297
Name: area, dtype: int64
```

# Lựa chọn dữ liệu với DataFrame

- DataFrame như là từ điển gồm các Series có chung chỉ số

*# create a DataFrame from a dictionary of Series*

```
area = pd.Series({'California': 423967, 'Texas': 695662, 'New York': 141297})
pop = pd.Series({'California': 38332521, 'Texas': 26448193, 'New York': 19651127})
data = pd.DataFrame({'area':area, 'pop':pop})
data
```

	area	pop
California	423967	38332521
Texas	695662	26448193
New York	141297	19651127

*# we can use attribute-style access with column names that are strings*

```
data.area
California    423967
Texas         695662
New York      141297
Name: area, dtype: int64
```

Chỉ dùng truy nhập kiểu thuộc tính khi tên cột là chuỗi ký tự và không xung đột với tên phương thức của DataFrame → `data.pop` <> `data['pop']`

# Lựa chọn dữ liệu với DataFrame

- DataFrame như là từ điển gồm các Series có chung chỉ số

```
# this dictionary-style syntax can also be used to modify the object,  
# in this case to add a new column  
data['density'] = data['pop'] / data['area']  
data
```

	area	pop	density
California	423967	38332521	90.413926
Texas	695662	26448193	38.018740
New York	141297	19651127	139.076746

# Lựa chọn dữ liệu với DataFrame

- DataFrame như là mảng 2 chiều

*# We can examine the underlying data array using the **values** attribute*

`data.values`

```
array([[4.23967000e+05, 3.83325210e+07, 9.04139261e+01],
       [6.95662000e+05, 2.64481930e+07, 3.80187404e+01],
       [1.41297000e+05, 1.96511270e+07, 1.39076746e+02]])
```

- Có thể thực hiện nhiều thao tác tương tự kiểu mảng trên DataFrame

*# transpose the full DataFrame to swap rows and columns*

`data.T`

	California	Texas	New York
area	4.239670e+05	6.956620e+05	1.412970e+05
pop	3.833252e+07	2.644819e+07	1.965113e+07
density	9.041393e+01	3.801874e+01	1.390767e+02

# Lựa chọn dữ liệu với DataFrame

- DataFrame như là mảng 2 chiều

*# passing a single index to an array accesses a row*

```
data.values[0]
```

```
array([4.23967000e+05, 3.83325210e+07, 9.04139261e+01])
```

*# passing a single "index" to a DataFrame accesses a column*

```
data['area']
```

```
California    423967
```

```
Texas         695662
```

```
New York      141297
```

```
Name: area, dtype: int64
```

# Lựa chọn dữ liệu với DataFrame

- Để định chỉ số kiểu mảng (array-style), Pandas dùng các thuộc tính định vị **loc** và **iloc**

```
# Using the iloc indexer, we can index the underlying array  
# as if it is a simple NumPy array  
data.iloc[:2, :2]
```

	area	pop
California	423967	38332521
Texas	695662	26448193

```
# Using the loc indexer to produce the same result  
data.loc[:'Texas', :'pop']
```

	area	pop
California	423967	38332521
Texas	695662	26448193

# Lựa chọn dữ liệu với DataFrame

- Bất kỳ cú pháp truy nhập dữ liệu NumPy-style nào cũng đều áp dụng được cho các thuộc tính định vị **loc** và **iloc**

```
# in the loc indexer we can combine masking and fancy indexing  
data.loc[data.density > 100, ['pop', 'density']]
```

	pop	density
New York	19651127	139.076746

```
# modifying values is done in the similar way with NumPy  
data.iloc[0, 2] = 90  
data
```

	area	pop	density
California	423967	38332521	90.000000
Texas	695662	26448193	38.018740
New York	141297	19651127	139.076746

# Lựa chọn dữ liệu với DataFrame

- Các quy tắc định vị hữu dụng khác

*# slicing refers to rows*  
`data['Texas':'New York']`

	area	pop	density
Texas	695662	26448193	38.018740
New York	141297	19651127	139.076746

*# Such slices can also refer to rows by number rather than by index*  
`data[1:3]`

	area	pop	density
Texas	695662	26448193	38.018740
New York	141297	19651127	139.076746

*# direct masking are handled row-wise rather than column-wise*  
`data[data.density > 100]`

	area	pop	density
New York	141297	19651127	139.076746



# Nội dung

---

- Giới thiệu chung
- Các đối tượng Pandas
- Định vị và lựa chọn dữ liệu
- **Thao tác dữ liệu trong Pandas**
- Định chỉ số phân cấp
- Đọc/ghi dữ liệu với file CSV

# Thao tác dữ liệu trong Pandas

- Một trong những điểm quan trọng của NumPy là khả năng thực hiện nhanh chóng các phép toán trên từng phần tử (element-wise), cả với các phép toán số học cơ bản (cộng, trừ, nhân, v.v.) và với các phép toán phức tạp hơn (hàm lượng giác, hàm mũ, hàm logarit, v.v.)
- Pandas thừa hưởng phần lớn chức năng này từ NumPy và các **ufuncs** (Universal Functions hay hàm vạn năng) là chìa khoá của việc này
- Đ/v các phép toán 1 ngôi như các hàm đổi dấu hoặc hàm lượng giác, ufuncs bảo toàn chỉ số và nhãn cột trong kết quả
- Đ/v các phép toán 2 ngôi như các hàm cộng hoặc nhân, Pandas sẽ tự động căn chỉnh chỉ số khi truyền các đối tượng vào ufuncs
- Điều này làm cho việc bảo toàn ngữ cảnh của dữ liệu và việc kết hợp dữ liệu từ các nguồn khác nhau trở nên dễ dàng với Pandas (vốn dễ xảy ra lỗi nếu dùng các mảng NumPy thuần túy)

# Ufuncs: Bảo toàn chỉ số

- Khi áp dụng một NumPy ufunc bất kỳ lên các đối tượng Series và DataFrame thì nhận được đối tượng cùng loại với chỉ số được bảo toàn

*# defining a simple Series*

```
rng = np.random.RandomState(42)
ser = pd.Series(rng.randint(0, 10, 4))
ser
```

```
0    6
1    3
2    7
3    4
dtype: int64
```

```
np.exp(ser)
```

```
0    403.428793
1    20.085537
2   1096.633158
3    54.598150
dtype: float64
```

# Ufuncs: Bảo toàn chỉ số

- Khi áp dụng một NumPy ufunc bất kỳ lên các đối tượng Series và DataFrame thì nhận được đối tượng cùng loại với chỉ số được bảo toàn

```
# defining a simple DataFrame
```

```
df = pd.DataFrame(rng.randint(0,10,(3,4)),columns=['A','B','C','D'])  
df
```

	A	B	C	D
0	6	9	2	6
1	7	4	3	7
2	7	2	5	4

```
np.sin(df * np.pi / 4)
```

	A	B	C	D
0	-1.000000	7.071068e-01	1.000000	-1.000000e+00
1	-0.707107	1.224647e-16	0.707107	-7.071068e-01
2	-0.707107	1.000000e+00	-0.707107	1.224647e-16

# Ufuncs: Căn chỉnh chỉ số

- Pandas sẽ căn chỉnh chỉ số khi thực hiện các phép toán 2 ngôi trên các đối tượng Series và DataFrame → thuận tiện khi làm việc với dữ liệu ko đầy đủ
- **Căn chỉnh chỉ số với Series**

*# suppose we are combining two diff data sources: area & population*

```
area = pd.Series({'Alaska': 1723337, 'Texas': 695662, 'California': 423967}, name='area')
```

```
population = pd.Series({'California': 38332521, 'Texas': 26448193, 'New York': 19651127}, name='population')
```

*# then compute the population density*

```
population / area
```

```
Alaska      NaN
California   90.413926
New York     NaN
Texas       38.018740
dtype: float64
```

- Mảng kết quả chứa hợp (union) của các chỉ số của 2 mảng đầu vào.
- Dữ liệu thiếu được đánh dấu là NaN (Not a Number)

# Ufuncs: Căn chỉnh chỉ số

- Căn chỉnh chỉ số với Series

*# any missing values are filled in with NaN by default for any of Python's built-in arithmetic expressions*

```
A = pd.Series([2, 4, 6], index=[0, 1, 2])  
B = pd.Series([1, 3, 5], index=[1, 2, 3])  
A + B
```

```
0    NaN  
1    5.0  
2    9.0  
3    NaN  
dtype: float64
```

*# we can modify the fill value using appropriate object methods in place of the operators*

```
A.add(B, fill_value=0)
```

```
0    2.0  
1    5.0  
2    9.0  
3    5.0  
dtype: float64
```

# Ufuncs: Căn chỉnh chỉ số

- Căn chỉnh chỉ số với DataFrame:** xảy ra với cả nhãn cột và chỉ số (hàng)

```
A = pd.DataFrame(rng.randint(0, 20, (2, 2)), columns=list('AB'))
```

A

	A	B
0	1	11
1	5	1

```
B = pd.DataFrame(rng.randint(0, 10, (3, 3)), columns=list('BAC'))
```

B

	B	A	C
0	4	0	9
1	5	8	0
2	9	2	6

A + B

	A	B	C
0	1.0	15.0	NaN
1	13.0	6.0	NaN
2	NaN	NaN	NaN

- Các chỉ số được căn chỉnh chính xác bất kể thứ tự của chúng trong hai đối tượng
- Các chỉ số trong kết quả được sắp xếp

# Ufuncs: Căn chỉnh chỉ số

- **Căn chỉnh chỉ số với DataFrame:** có thể dùng hàm số học của đối tượng và truyền giá trị **fill\_value** mong muốn thay cho các dữ liệu bị thiếu

```
# we'll fill with the mean of all values in A  
# (which we compute by first stacking the rows of A)  
fill = A.stack().mean()  
A.add(B, fill_value=fill)
```

	A	B	C
0	1.0	15.0	13.5
1	13.0	6.0	4.5
2	6.5	13.5	10.5



# Ufuncs: Căn chỉnh chỉ số

- Ánh xạ giữa toán tử của Python và hàm số học của Pandas

Python operator	Pandas method(s)
+	add()
-	sub(), subtract()
*	mul(), multiply()
/	truediv(), div(), divide()
//	floordiv()
%	mod()
**	pow()

# Ufuncs: Các phép toán giữa DataFrame và Series

- Phép toán giữa DataFrame và Series tương tự với phép toán giữa mảng 2 chiều và mảng 1 chiều NumPy

*# find the difference of a 2D array and one of its rows*

```
A = rng.randint(10, size=(3, 4))
```

A

```
array([[3, 8, 2, 4],  
       [2, 6, 4, 8],  
       [6, 1, 3, 8]])
```

*A - A[0] # subtraction is applied row-wise*

```
array([[ 0,  0,  0,  0],  
       [-1, -2,  2,  4],  
       [ 3, -7,  1,  4]])
```

# Ufuncs: Các phép toán giữa DataFrame và Series

- Pandas cũng mặc định thực hiện phép toán theo từng hàng (row-wise)

```
df = pd.DataFrame(A, columns=list('QRST'))  
df - df.iloc[0]
```

	Q	R	S	T
0	0	0	0	0
1	-1	-2	2	4
2	3	-7	1	4

- Nếu muốn thực hiện phép toán theo từng cột (column-wise) thì dùng hàm của đối tượng và đặc tả từ khoá **axis**

```
df.subtract(df['R'], axis=0)
```

	Q	R	S	T
0	-5	0	-6	-4
1	-4	0	-2	2
2	5	0	2	7

# Ufuncs: Các phép toán giữa DataFrame và Series

- Khi thực hiện các phép toán giữa DataFrame và Series, chỉ số và cột cũng được căn chỉnh tự động

df

	Q	R	S	T
0	3	8	2	4
1	2	6	4	8
2	6	1	3	8

```
halfrow = df.iloc[0, ::2]
```

halfrow

```
Q      3
S      2
Name: 0, dtype: int64
```

df - halfrow

	Q	R	S	T
0	0.0	NaN	0.0	NaN
1	-1.0	NaN	2.0	NaN
2	3.0	NaN	1.0	NaN

Việc bảo toàn và căn chỉnh các chỉ số và cột này đảm bảo rằng các thao tác trên dữ liệu trong Pandas sẽ luôn duy trì ngữ cảnh dữ liệu

# Nội dung

- Giới thiệu chung
- Các đối tượng Pandas
- Định vị và lựa chọn dữ liệu
- Thao tác dữ liệu trong Pandas
- Định chỉ số phân cấp
- Đọc/ghi dữ liệu với file CSV

# Định chỉ số phân cấp

- Để lưu trữ và xử lý dữ liệu có nhiều hơn 2 chiều, Pandas hỗ trợ định chỉ số phân cấp (hierarchical indexing) để tích hợp nhiều mức chỉ số trong 1 chỉ số duy nhất
- Giả sử ta muốn theo dõi dữ liệu dân số về các bang trong 2 năm khác nhau, ta có thể dùng kiểu dữ liệu tuple (bộ) làm khoá để sinh ra Series

```
index = [('California', 2000), ('California', 2010), ('New York', 2000), ('New York', 2010),  
('Texas', 2000), ('Texas', 2010)]  
populations = [33871648, 37253956, 18976457, 19378102, 20851820, 25145561]  
pop = pd.Series(populations, index=index)  
pop
```

```
(California, 2000)    33871648  
(California, 2010)    37253956  
(New York, 2000)     18976457  
(New York, 2010)     19378102  
(Texas, 2000)        20851820  
(Texas, 2010)        25145561  
dtype: int64
```

# Định chỉ số phân cấp

- Nếu cần lọc ra dữ liệu của năm 2010 thì ta phải dùng câu lệnh khá luộm thuộm và có thể không hiệu quả (chậm) sau:

```
pop[[i for i in pop.index if i[1] == 2010]]
```

```
(California, 2010)    37253956  
(New York, 2010)     19378102  
(Texas, 2010)        25145561  
dtype: int64
```

- Cách trên làm cho code khó đọc hơn và chạy kém hiệu quả hơn (đ/v tập dữ liệu lớn) so với cú pháp cắt lát (slicing)
- Pandas cung cấp kiểu **Multindex** để chứa nhiều mức chỉ số

# Định chỉ số phân cấp

- Có thể tạo đối tượng MultiIndex từ tuple:

```
index = pd.MultiIndex.from_tuples(index)  
index
```

```
MultiIndex([( 'California', 2000),  
            ( 'California', 2010),  
            (   'New York', 2000),  
            (   'New York', 2010),  
            (    'Texas', 2000),  
            (    'Texas', 2010)],  
           )
```

- Xem biểu diễn phân cấp của dữ liệu bằng cách “reindex” Series vừa tạo:

```
pop = pop.reindex(index)  
pop
```

```
California  2000    33871648  
            2010    37253956  
New York    2000    18976457  
            2010    19378102  
Texas       2000    20851820  
            2010    25145561  
dtype: int64
```

→ 2 cột đầu tiên là các giá trị chỉ số trong biểu diễn đa mức, cột thứ ba là dữ liệu



# Định chỉ số phân cấp

- Bây giờ để truy nhập dữ liệu của năm 2010 (chỉ số thứ hai là 2010), ta chỉ cần dùng cú pháp slicing → thuận tiện và hiệu quả hơn nhiều:

```
pop
```

```
California  2000    33871648  
            2010    37253956  
New York    2000    18976457  
            2010    19378102  
Texas       2000    20851820  
            2010    25145561  
dtype: int64
```

```
pop[:, 2010]
```

```
California    37253956  
New York      19378102  
Texas         25145561  
dtype: int64
```

# Multindex as extra dimension

- Trong vd trước, ta có thể lưu trữ dữ liệu trong DataFrame 1 cách đơn giản
- Hàm `unstack()` chuyển Series có chỉ số phân cấp thành DataFrame

```
pop_df = pop.unstack()  
pop_df
```

	2000	2010
California	33871648	37253956
New York	18976457	19378102
Texas	20851820	25145561

- Hàm `stack()` làm điều ngược lại

```
pop_df.stack()
```

California	2000	33871648
	2010	37253956
New York	2000	18976457
	2010	19378102
Texas	2000	20851820
	2010	25145561

dtype: int64

# MultIndex as extra dimension

- Ta có thể dùng chỉ số phân cấp để biểu diễn dữ liệu nhiều chiều cho Series và DataFrame vì mỗi mức trong chỉ số phân cấp biểu diễn 1 chiều của dữ liệu
- Việc thêm 1 cột dữ liệu với đối tượng MultiIndex khá dễ dàng

```
pop_df = pd.DataFrame({'total': pop,  
                        'under18': [9267089, 9284094, 4687374, 4318033, 5906301, 6879014]})  
pop_df
```

		total	under18
California	2000	33871648	9267089
	2010	37253956	9284094
New York	2000	18976457	4687374
	2010	19378102	4318033
Texas	2000	20851820	5906301
	2010	25145561	6879014

# Multindex as extra dimension

- Tất cả tính năng về thao tác với dữ liệu (phần trước) đều áp dụng với chỉ số phân cấp

```
# compute the fraction of people under 18 by year
```

```
f_u18 = pop_df['under18'] / pop_df['total']
```

```
f_u18
```

```
California  2000    0.273594  
            2010    0.249211  
New York    2000    0.247010  
            2010    0.222831  
Texas       2000    0.283251  
            2010    0.273568  
dtype: float64
```

```
f_u18.unstack()
```

```
          2000    2010  
California 0.273594 0.249211  
New York   0.247010 0.222831  
Texas      0.283251 0.273568
```

# Các cách tạo MultiIndex

- Cách đơn giản nhất để tạo một Series hoặc DataFrame được lập chỉ số phân cấp (hay chỉ số bội) là truyền một danh sách gồm hai hoặc nhiều mảng chỉ số vào hàm tạo

```
df = pd.DataFrame(np.random.rand(4, 2),  
index=[['a', 'a', 'b', 'b'], [1, 2, 1, 2]], columns=['data1', 'data2'])  
df
```

		data1	data2
a	1	0.199076	0.736103
	2	0.346474	0.307717
b	1	0.462784	0.163726
	2	0.305403	0.561836

# Các cách tạo MultiIndex

- Cách khác là truyền một từ điển với các tuple thích hợp làm khóa vào hàm tạo

```
data = {('California', 2000): 33871648,  
        ('California', 2010): 37253956,  
        ('Texas', 2000): 20851820,  
        ('Texas', 2010): 25145561,  
        ('New York', 2000): 18976457,  
        ('New York', 2010): 19378102}
```

```
pd.Series(data)
```

California	2000	33871648
	2010	37253956
Texas	2000	20851820
	2010	25145561
New York	2000	18976457
	2010	19378102

dtype: int64

# Các cách tạo MultiIndex

- Dùng các hàm tạo của lớp **pd.MultiIndex** để tăng tính uyển chuyển tạo index

```
pd.MultiIndex.from_arrays([[ 'a', 'a', 'b', 'b'], [1, 2, 1, 2]])
pd.MultiIndex.from_tuples([( 'a', 1), ( 'a', 2), ( 'b', 1), ( 'b', 2)])
pd.MultiIndex.from_product([[ 'a', 'b'], [1, 2]])
pd.MultiIndex(levels=[[ 'a', 'b'], [1, 2]],
               labels=[[0, 0, 1, 1], [0, 1, 0, 1]])
```

Mỗi câu lệnh trên đều cho cùng kết quả

```
MultiIndex([( 'a', 1),
            ( 'a', 2),
            ( 'b', 1),
            ( 'b', 2)],
           )
```

- Sau đó truyền đối tượng **MultiIndex** vừa tạo vào đối số **index** của hàm tạo Series/DataFrame, hoặc vào hàm **reindex()** của đối tượng Series/DataFrame

# Multindex cho cột

- Trong DataFrame, cột cũng có thể có nhiều mức chỉ số như hàng

```
# hierarchical indices and columns
```

```
index = pd.MultiIndex.from_product([[2013, 2014], [1, 2]],  
                                   names=['year', 'visit'])  
columns = pd.MultiIndex.from_product(['Bob', 'Guido', 'Sue'], ['HR',  
                                                                'Temp'],  
                                   names=['subject', 'type'])
```

```
# mock some data
```

```
data = np.round(np.random.randn(4, 6), 1)
```

```
data[:, ::2] *= 10
```

```
data += 37
```

```
# create the DataFrame
```

```
health_data = pd.DataFrame(data, index=index, columns=columns)
```

```
health_data
```

year	visit	subject Bob		Guido		Sue	
		HR	Temp	HR	Temp	HR	Temp
2013	1	39.0	35.6	45.0	38.0	42.0	36.8
	2	37.0	35.5	44.0	36.1	48.0	35.3
2014	1	35.0	36.8	21.0	36.5	54.0	36.7
	2	36.0	38.5	24.0	37.6	34.0	37.1

→ Dữ liệu 4 chiều: subject, measurement type, year, visit number



# MultilIndex cho cột

- Trong DataFrame, cột cũng có thể có nhiều mức chỉ số như hàng

*# index toplevel column by the person's name and get a full Data Frame# containing just that person's information*

```
health_data['Guido']
```

	type	HR	Temp
year	visit		
2013	1	45.0	38.0
	2	44.0	36.1
2014	1	21.0	36.5
	2	24.0	37.6

# Indexing and Slicing a MultiIndex

- Series

pop

```
California  2000    33871648
            2010    37253956
New York    2000    18976457
            2010    19378102
Texas       2000    20851820
            2010    25145561
dtype: int64
```

*# We can access single elements by indexing with multiple terms*

```
pop['California', 2000]
33871648
```

*# MultiIndex supports indexing just one of the levels in the index*

```
pop['California']
```

```
2000    33871648
2010    37253956
dtype: int64
```

# Indexing and Slicing a MultiIndex

- Series

*# Partial slicing is available as long as the MultiIndex is sorted*

```
pop.loc['California':'New York']
```

```
California  2000    33871648
            2010    37253956
New York    2000    18976457
            2010    19378102
dtype: int64
```

*# With sorted indices, we can perform partial indexing on lower levels#  
by passing an empty slice in the first index*

```
pop[:, 2000]
```

```
California    33871648
New York      18976457
Texas         20851820
dtype: int64
```

# Indexing and Slicing a MultiIndex

- DataFrame

health\_data

year	visit	subject Bob		Guido		Sue	
		HR	Temp	HR	Temp	HR	Temp
2013	1	39.0	35.6	45.0	38.0	42.0	36.8
	2	37.0	35.5	44.0	36.1	48.0	35.3
2014	1	35.0	36.8	21.0	36.5	54.0	36.7
	2	36.0	38.5	24.0	37.6	34.0	37.1

*# we can recover Guido's heart rate data*

```
health_data['Guido', 'HR']
```

```
year  visit
2013   1      45.0
       2      44.0
2014   1      21.0
       2      24.0
Name: (Guido, HR), dtype: float64
```

# Indexing and Slicing a MultiIndex

- DataFrame

```
health_data
```

year	visit	subject Bob		Guido		Sue		
		type	HR	Temp	HR	Temp	HR	Temp
2013	1	39.0	35.6	45.0	38.0	42.0	36.8	
	2	37.0	35.5	44.0	36.1	48.0	35.3	
2014	1	35.0	36.8	21.0	36.5	54.0	36.7	
	2	36.0	38.5	24.0	37.6	34.0	37.1	

```
health_data.iloc[:2, :2]
```

year	subject	Bob	
	type	HR	Temp
	visit		
2013	1	39.0	35.6
	2	37.0	35.5

```
health_data.loc[:, ('Bob', 'HR')]
```

```
year  visit
2013    1    39.0
        2    37.0
2014    1    35.0
        2    36.0
Name: (Bob, HR), dtype: float64
```

# Tổng hợp dữ liệu trên MultiIndex

- Các hàm `mean()`, `sum()`, `max()`,... cần được truyền tham số **level** để chỉ ra tập con nào của dữ liệu được thực hiện tính toán

```
# average out the measurements in the two visits each year  
data_mean = health_data.mean(level='year')  
data_mean
```

subject	Bob		Guido		Sue	
type	HR	Temp	HR	Temp	HR	Temp
year						
2013	38.0	35.55	44.5	37.05	45.0	36.05
2014	35.5	37.65	22.5	37.05	44.0	36.90

```
# we can take the mean among levels on the columns as well  
data_mean.mean(axis=1, level='type')
```

type	HR	Temp
year		
2013	42.5	36.216667
2014	34.0	37.200000

# Các hàm tổng hợp dữ liệu của Pandas

Aggregation	Description
<code>count()</code>	Total number of items
<code>first()</code> , <code>last()</code>	First and last item
<code>mean()</code> , <code>median()</code>	Mean and median
<code>min()</code> , <code>max()</code>	Minimum and maximum
<code>std()</code> , <code>var()</code>	Standard deviation and variance
<code>mad()</code>	Mean absolute deviation
<code>prod()</code>	Product of all items
<code>sum()</code>	Sum of all items

# Nội dung

---

- Giới thiệu chung
- Các đối tượng Pandas
- Định vị và lựa chọn dữ liệu
- Thao tác dữ liệu trong Pandas
- Định chỉ số phân cấp
- Đọc/ghi dữ liệu với file CSV



# Đọc/ghi dữ liệu với file CSV

- File CSV (comma-separated value) là định dạng phổ biến để lưu dữ liệu dạng bảng
- Đọc file CSV vào DataFrame: `df = pd.read_csv('path_to_file.csv')`
- Ghi DataFrame vào file CSV: `df.to_csv('path_to_file.csv')`