クレジット:
UTokyo Online Education Education コンピュータシステム概論 2018 小林克志

東京大学
THE UNIVERSITY OF TOKYO

CC BY NC ND

Pandas

この資料はThe Python Tutorial (https://docs.python.org/3.6/tutorial/index.html#the-python-tutorial) (日本語版 (https://docs.python.jp/3/tutorial/)) および Python for Data Analysis:Wrangling with Pandas, Numpy and IPython (http://shop.oreilly.com/product/0636920050896.do)を参考に作成した。

Pandas は Python でデータ分析をおこなうためのデータ構造やデータ操作ツールを提供している。
Pandas を利用するには、以下のようにモジュールを import する:

```
import pandas as pd
```

ここでは、Pandas のデータ構造のうち、`pd.Series` と `pd.DataFrame` を採り上げる。

pd.Series

`pd.Series` は一次元配列と配列に関連づけられたインデックス名で構成されている。
一次元リストから、`pd.Series`は、以下のように生成できる。
値とインデックスはそれぞれ `pd.Series.values`, `pd.Series.index` 属性から得る:

In [117]:
```
import pandas as pd
obj = pd.Series([9,8,7,6])
print("pd.Series:")
print(obj)
print()
print("pd.Series.values:")
print(obj.values)
print()
print("pd.Series.index:")
print(obj.index)
```

```
pd.Series:
0    9
1    8
2    7
3    6
dtype: int64

pd.Series.values:
[9 8 7 6]

pd.Series.index:
RangeIndex(start=0, stop=4, step=1)
```

インデックスとして文字列（ラベル）を使うこともできる:

In [118]:
```
import pandas as pd
obj2 = pd.Series([9,8,7,6], index=["a", "b", "c", "d"])
print("pd.Series:")
print(obj2)
print()
print("pd.Series.values:")
print(obj2.values)
print()
print("pd.Series.index:")
print(obj2.index)
```

```
pd.Series:
a    9
b    8
c    7
d    6
dtype: int64

pd.Series.values:
[9 8 7 6]

pd.Series.index:
Index(['a', 'b', 'c', 'd'], dtype='object')
```

numpy.ndarray との一番大きなちがいは、pd.Series はラベルを使って要素にアクセスできる:

In [119]:
```
import pandas as pd
obj2 = pd.Series([9,8,7,6], index=["a", "b", "c", "d"])
print('obj2["a"]:')
print(obj2["a"])
print()
obj2["a"] = 9999
obj2["e"] = 0
print("After substitutions:")
print(obj2)
```

```
obj2["a"]:
9

After substitutions:
a    9999
b       8
c       7
d       6
e       0
dtype: int64
```

pd.Series では numpy.ndarray と同様のフィルタリング、演算がおこなえる:

In [120]:
```
import pandas as pd
obj2 = pd.Series([9,8,7,6], index=["a", "b", "c", "d"])
print('obj2[obj2 < 8]:')
print(obj2[obj2 < 8])
print()
print('obj2 * 2')
print(obj2 * 2)
```

```
obj2[obj2 < 8]:
c    7
d    6
dtype: int64

obj2 * 2
a    18
b    16
c    14
d    12
dtype: int64
```

辞書形式のデータがあれば、それから pd.Seriesを生成することもできる。key がラベルに、value が要素となる:

In [121]:
```
import pandas as pd
data_dict = {"Tokyo":1000, "Kanagawa":900, "Chiba":800}

obj3 = pd.Series(data_dict)

print("pd.Series:")
print(obj3)
print()
print("pd.Series.values:")
print(obj3.values)
print()
print("pd.Series.index:")
print(obj3.index)
```

```
pd.Series:
Chiba        800
Kanagawa     900
Tokyo       1000
dtype: int64

pd.Series.values:
[ 800  900 1000]

pd.Series.index:
Index(['Chiba', 'Kanagawa', 'Tokyo'], dtype='object')
```

辞書から pd.Series 生成時にインデックスを指定することもできる。
以下の例では辞書の key にないラベルとして Saitama が与えられている。 生成された pd.Series では、対応する value は NaN (Not a Number) となる。
一方で、Kanagawa というラベルはインデックスとして与えられていないため pd.Series からは除かれる:

In [122]:
```
import pandas as pd
data_dict = {"Tokyo":1000, "Kanagawa":900, "Chiba":800}

obj4 = pd.Series(data_dict, index=["Tokyo", "Chiba", "Saitama"])

print("pd.Series:")
print(obj4)
print()
```

```
pd.Series:
Tokyo      1000.0
Chiba       800.0
Saitama       NaN
dtype: float64
```

データ欠落の検出には `pd.isnull()`を、有効データの検査には `pd.notnull()` を使う:

In [123]:
```
import pandas as pd
data_dict = {"Tokyo":1000, "Kanagawa":900, "Chiba":800}

obj4 = pd.Series(data_dict, index=["Tokyo", "Chiba", "Saitama"])

print("pd.isnull():")
print(pd.isnull(obj4))
print()
print("pd.notnull():")
print(pd.notnull(obj4))
```

```
pd.isnull():
Tokyo      False
Chiba      False
Saitama     True
dtype: bool

pd.notnull():
Tokyo       True
Chiba       True
Saitama    False
dtype: bool
```

`pd.Series` オブジェクトおよびインデックスに名付けすることもできる:

In [124]:
```
import pandas as pd
data_dict = {"Tokyo":1000, "Kanagawa":900, "Chiba":800}
obj4 = pd.Series(data_dict, index=["Tokyo", "Chiba", "Saitama"])

obj4.name = "年収"
obj4.index.name = "都道府県"

print(obj4)
```

```
都道府県
Tokyo      1000.0
Chiba       800.0
Saitama       NaN
Name: 年収, dtype: float64
```

インデックス、選択、フィルタリング

pd.Series には Numpy と同じように要素番号でアクセスできる。 添字には、数字だけではなくインデックスも使える:

In [125]:
```
import pandas as pd
import numpy as np

obj = pd.Series(range(5), index = ["a", "b", "c", "d", "e"])

print("Sample:")
print(obj)
print()
print('obj["b"], obj[1]')
print(obj["b"], obj[1])
print()
print('obj[["b", "d", "e"]] : fancy index')
print(obj[["b", "d", "e"]])
print()
print('obj[obj > 2] : filtering')
print(obj[obj > 2])
```

```
Sample:
a    0
b    1
c    2
d    3
e    4
dtype: int64

obj["b"], obj[1]
1 1

obj[["b", "d", "e"]] : fancy index
b    1
d    3
e    4
dtype: int64

obj[obj > 2] : filtering
d    3
e    4
dtype: int64
```

インデックスを使ったスライシングも使える。
通常のスライシングと違うのは、stop の要素も含まれる。 スライスされた要素への代入もできる:

In [126]:
```
import pandas as pd
import numpy as np

obj = pd.Series(range(5), index = ["a", "b", "c", "d", "e"])

print("Sample:")
print(obj)
print()

print('obj[1:4]')
print(obj[1:4])
print()

print('obj["a":"c"]')
print(obj["a":"c"])
print()

print('after obj["a":"c"] = 99')
obj["a":"c"] = 99
print(obj)
```

```
Sample:
a    0
b    1
c    2
d    3
e    4
dtype: int64

obj[1:4]
b    1
c    2
d    3
dtype: int64

obj["a":"c"]
a    0
b    1
c    2
dtype: int64

after obj["a":"c"] = 99
a    99
b    99
c    99
d     3
e     4
dtype: int64
```

pd.DataFrame

pd.DataFrame は複数の列、column(s)、から構成された２次元の表である。 それぞれの列で異なる要素型とすることができる。 pd.DataFrame は行、列ともににインデックスを与えることができる（Jupyter notebook では pd.DataFrame は整形されるので以降これを利用する）：

In [127]:
```
import pandas as pd
data_dict = {"prefecture":["Tokyo", "Chiba", "Kanagawa", "Saitam
a"],
             "year":[2017, 2012, 2012, 2012],
             "pop":[13.7, 6.14, 9.1, 7.2]}
df = pd.DataFrame(data_dict)

df
```

Out[127]:

	pop	prefecture	year
0	13.70	Tokyo	2017
1	6.14	Chiba	2012
2	9.10	Kanagawa	2012
3	7.20	Saitama	2012

`pd.DataFrame()` では、`columns` 引数によって列の順序を調整できる。
相当するインデックス（この場合キー）がなければその列は `NaN` となる:

In [128]:
```
import pandas as pd
data_dict = {"prefecture":["Tokyo", "Chiba", "Kanagawa", "Saitam
a"],
             "year":[2017, 2012, 2012, 2012],
             "pop":[13.7, 6.14, 9.1, 7.2]}

df2 = pd.DataFrame(data_dict, columns=["year", "prefecture", "po
p", "debt"],
                   index=["one", "two", "three", "four"])

print("pd.DataFrame:")
df2
```

```
pd.DataFrame:
```

Out[128]:

	year	prefecture	pop	debt
one	2017	Tokyo	13.70	NaN
two	2012	Chiba	6.14	NaN
three	2012	Kanagawa	9.10	NaN
four	2012	Saitama	7.20	NaN

列にはインデックス名を辞書 key あるいは属性としてアクセスすることができる （行へのアクセスは、`pd.DataFrame.loc` あるいは `pd.DataFrame.iloc` を利用する）:

In [129]:
```
import pandas as pd
data_dict = {"prefecture":["Tokyo", "China", "Kanagawa", "Saitam
a"],
             "year":[2017, 2012, 2012, 2012],
             "pop":[13.7, 6.14, 9.1, 7.2]}

df2 = pd.DataFrame(data_dict, columns=["year", "prefecture", "po
p", "debt"],
                   index=["one", "two", "three", "four"])

print("Pick column as dictionary:")
print(df2["prefecture"])
print()
print("Pick column as attribute:")
print(df2.year)
```

```
Pick column as dictionary:
one         Tokyo
two         China
three    Kanagawa
four      Saitama
Name: prefecture, dtype: object

Pick column as attribute:
one      2017
two      2012
three    2012
four     2012
Name: year, dtype: int64
```

列の内容を更新することは可能、例えば debt 列を更新する場合以下のようにできる。
list など配列で更新する場合はその長さを pd.DataFrame 長さに合わせる必要がある。
インデックス付きの pd.Series を使えば長さをそろえる必要はない:

In [130]:
```
import pandas as pd
data_dict = {"prefecture":["Tokyo", "Chiba", "Kanagawa", "Saitam
a"],
             "year":[2017, 2012, 2012, 2012],
             "pop":[13.7, 6.14, 9.1, 7.2]}

df2 = pd.DataFrame(data_dict, columns=["year", "prefecture", "po
p", "debt"],
                   index=["one", "two", "three", "four"])

df2["debt"] = 16.5 # Fill a scalar value
df2
```

Out[130]:

	year	prefecture	pop	debt
one	2017	Tokyo	13.70	16.5
two	2012	Chiba	6.14	16.5
three	2012	Kanagawa	9.10	16.5
four	2012	Saitama	7.20	16.5

In [131]:
```
df2["debt"] = range(4) #Fill sequence values
df2
```

Out[131]:

	year	prefecture	pop	debt
one	2017	Tokyo	13.70	0
two	2012	Chiba	6.14	1
three	2012	Kanagawa	9.10	2
four	2012	Saitama	7.20	3

In [132]:
```
val = pd.Series([10, 12], index=["two", "four"])
df2["debt"] = val # Replace partial elements
df2
```

Out[132]:

	year	prefecture	pop	debt
one	2017	Tokyo	13.70	NaN
two	2012	Chiba	6.14	10.0
three	2012	Kanagawa	9.10	NaN
four	2012	Saitama	7.20	12.0

インデックスのない列への代入では列が作成される。`del` によって列を削除することも可能:

In [133]:
```
import pandas as pd
data_dict = {"prefecture":["Tokyo", "Chiba", "Kanagawa", "Saitam
a"],
             "year":[2017, 2012, 2012, 2012],
             "pop":[13.7, 6.14, 9.1, 7.2]}

df2 = pd.DataFrame(data_dict, columns=["year", "prefecture", "po
p", "debt"],
                   index=["one", "two", "three", "four"])


df2["capital"] = df2.prefecture == "Tokyo" #Add capital culumn
df2
```

Out[133]:

	year	prefecture	pop	debt	capital
one	2017	Tokyo	13.70	NaN	True
two	2012	Chiba	6.14	NaN	False
three	2012	Kanagawa	9.10	NaN	False
four	2012	Saitama	7.20	NaN	False

In [134]:
```
del df2["capital"] # Delete the culumn
df2
```

Out[134]:

	year	prefecture	pop	debt
one	2017	Tokyo	13.70	NaN
two	2012	Chiba	6.14	NaN
three	2012	Kanagawa	9.10	NaN
four	2012	Saitama	7.20	NaN

pd.DataFrame 生成時に辞書が入れ子になっている場合は、以下のように展開される:

In [135]:
```
import pandas as pd
pop2 = {"Tokyo":{2012:13.1, 2017:13.7},
        "Kanagawa":{2012:9.1, 2017:9.1},
        "Chiba":{2012:6.1}}

df3 = pd.DataFrame(pop2)
df3
```

Out[135]:

	Chiba	Kanagawa	Tokyo
2012	6.1	9.1	13.1
2017	NaN	9.1	13.7

行と列を入れ替える（転置）は、Numpy と同様に pd.DataFrame.T が使える:

In [136]:
```
import pandas as pd
pop2 = {"Tokyo":{2012:13.1, 2017:13.7},
        "Kanagawa":{2012:9.1, 2017:9.1},
        "Chiba":{2012:6.1}}

df3 = pd.DataFrame(pop2)
df3
```

Out[136]:

	Chiba	Kanagawa	Tokyo
2012	6.1	9.1	13.1
2017	NaN	9.1	13.7

In [137]:
```
df3.T
```

Out[137]:

	2012	2017
Chiba	6.1	NaN
Kanagawa	9.1	9.1
Tokyo	13.1	13.7

pd.Dataframe の行、列に名前をつけることもできる（pd.Series と同様）：

In [138]:
```
import pandas as pd
pop2 = {"Tokyo":{2012:13.1, 2017:13.7},
       "Kanagawa":{2012:9.1, 2017:9.1},
       "Chiba":{2012:6.1}}
df3 = pd.DataFrame(pop2)

df3.index.name = "year"
df3.columns.name = "prefecture"
df3
```

Out[138]:

prefecture	Chiba	Kanagawa	Tokyo
year			
2012	6.1	9.1	13.1
2017	NaN	9.1	13.7

pd.DataFrame.values 属性によって、np.ndarray を取り出すこともできる。
異なる型がまざっていた場合は（np.ndarray は型が一つしかとれないので）もっとも適切な型が選択される:

In [139]:
```
import pandas as pd
import numpy as np
pop2 = {"Tokyo":{2012:13.1, 2017:13.7},
       "Kanagawa":{2012:9.1, 2017:9.1},
       "Chiba":{2012:6.1}}
df3 = pd.DataFrame(pop2)

print("NumPy dtype of df3:")
print(df3.values.dtype)

data_dict = {"prefecture":["Tokyo", "Chiba", "Kanagawa", "Saitam
a"],
             "year":[2017, 2012, 2012, 2012],
             "pop":[13.7, 6.14, 9.1, 7.2]}

df4= pd.DataFrame(data_dict, columns=["year", "prefecture", "pop
", "debt"],
                      index=["one", "two", "three", "four"])

print("NumPy dtype of df4:")
print(df4.values.dtype)
```

```
NumPy dtype of df3:
float64
NumPy dtype of df4:
object
```


再インデックス（pd.Series, pd.DataFrame）

pd.Series.reindex() メソッドは、与えられるインデックスに対応した新しいオブジェクトを生成する。

In [140]:
```
import pandas as pd
obj = pd.Series([-3,-2,0,2], index=["d", "b", "c", "a"])
print(obj)
print()

obj2 = obj.reindex(index = ["a", "b", "c", "d", "e"])
print(obj2)
```

```
d   -3
b   -2
c    0
a    2
dtype: int64

a    2.0
b   -2.0
c    0.0
d   -3.0
e    NaN
dtype: float64
```

時系列データなど補完が必要なときは、method = ffill()、によって補完することも可能:

In [141]:
```
import pandas as pd
obj3 = pd.Series(["blue", "purple", "yellow"], index=[0,2,4])
print(obj3.reindex(index=range(6), method="ffill"))
```

```
0      blue
1      blue
2    purple
3    purple
4    yellow
5    yellow
dtype: object
```

pd.DataFrame.reindex()は行、列あるいはその双方に適用できる。
列に適用する場合は columns オプションで与える:

In [142]:
```
import pandas as pd
import numpy as np
df4 = pd.DataFrame(np.arange(9).reshape([3,3]),
                   index=["a", "b", "d"],
                   columns=["Tokyo", "Kanagawa", "Chiba"])
df4
```

Out[142]:

	Tokyo	Kanagawa	Chiba
a	0	1	2
b	3	4	5
d	6	7	8

In [143]:
```
print("Re-indexed row:")
df4.reindex(["a", "b", "c"])
```

```
Re-indexed row:
```

Out[143]:

	Tokyo	Kanagawa	Chiba
a	0.0	1.0	2.0
b	3.0	4.0	5.0
c	NaN	NaN	NaN

In [144]:
```
print("Re-indexed cilumn:")
prefectures = ["Tokyo", "Chiba", "Saitama"]
df4.reindex(columns=prefectures)
```

```
Re-indexed cilumn:
```

Out[144]:

	Tokyo	Chiba	Saitama
a	0	2	NaN
b	3	5	NaN
d	6	8	NaN

行・列の削除（pd.Series, pd.DataFrame）

pd.Series および pd.DataFrameの drop メソッドは引数に指定したデータを削除したオブジェクトをあらたに作成する:

In [145]:
```
import pandas as pd
import numpy as np

obj = pd.Series(range(5), index = ["a", "b", "c", "d", "e"])
print("Sample:")
print(obj)
print()
print('After drop("c"):')
obj2 = obj.drop("c")
print(obj2)
```

```
Sample:
a    0
b    1
c    2
d    3
e    4
dtype: int64

After drop("c"):
a    0
b    1
d    3
e    4
dtype: int64
```

pd.DataFrame で列を削除したい場合は、axis=1 あるいは axis="columns" を指定する。
また、既存オブジェクトを更新する場合は、inplace = True とする:

In [146]:
```
df2 = pd.DataFrame(np.arange(16).reshape([4,4]),
                   index = ["a", "b", "c", "d"],
                   columns = ["Tokyo", "Chiba", "Saitama", "Kanagaw
a"])
print("Sample:")
df2
```

```
Sample:
```

Out[146]:

	Tokyo	Chiba	Saitama	Kanagawa
a	0	1	2	3
b	4	5	6	7
c	8	9	10	11
d	12	13	14	15

In [147]:
```
print('df2.drop("c"):')
df2.drop("c")
```

```
df2.drop("c"):
```

Out[147]:

	Tokyo	Chiba	Saitama	Kanagawa
a	0	1	2	3
b	4	5	6	7
d	12	13	14	15

In [148]:
```
print('df2.drop("Saitama", axis=1):')
df2.drop("Saitama", axis=1)
```

```
df2.drop("Saitama", axis=1):
```

Out[148]:

	Tokyo	Chiba	Kanagawa
a	0	1	3
b	4	5	7
c	8	9	11
d	12	13	15

インデックス、スライシング、フィルタリング

`pd.DataFrame` の列単位のアクセスは辞書のようにおこなえる。フィルタリングも同様:

In [149]:
```
df2 = pd.DataFrame(np.arange(16).reshape([4,4]),
                   index = ["a", "b", "c", "d"],
                   columns = ["Tokyo", "Chiba", "Saitama", "Kanagaw
a"])
print("Sample:")
df2
```

```
Sample:
```

Out[149]:

	Tokyo	Chiba	Saitama	Kanagawa
a	0	1	2	3
b	4	5	6	7
c	8	9	10	11
d	12	13	14	15

In [150]:
```
print('df2["Chiba"]:')
df2["Chiba"]
```

```
df2["Chiba"]:
```

Out[150]:
```
a     1
b     5
c     9
d    13
Name: Chiba, dtype: int64
```

In [151]:
```
print('df2[["Kanagawa", "Tokyo"] as fancy index:')
df2[["Kanagawa", "Tokyo"]]
```

```
df2[["Kanagawa", "Tokyo"] as fancy index:
```

Out[151]:

	Kanagawa	Tokyo
a	3	0
b	7	4
c	11	8
d	15	12

In [152]:
```
print('df2[df2["Kanagawa"] > 7] as filtering:')
df2[df2["Kanagawa"] > 7]
```

```
df2[df2["Kanagawa"] > 7] as filtering:
```

Out[152]:

	Tokyo	Chiba	Saitama	Kanagawa
c	8	9	10	11
d	12	13	14	15

In [ ]:
```

```

ただしスライシングは行に対して適用される。

In [153]:
```
df2 = pd.DataFrame(np.arange(16).reshape([4,4]),
                   index = ["a", "b", "c", "d"],
                   columns = ["Tokyo", "Chiba", "Saitama", "Kanagaw
a"])

print("df2[:2] as slicing:")
df2[:2]
```

```
df2[:2] as slicing:
```

Out[153]:

	Tokyo	Chiba	Saitama	Kanagawa
a	0	1	2	3
b	4	5	6	7

pd.DataFrame からブール要素だけの別の DataFrame を作成することもできる。 特定の条件を満たす要素に代入することもできる:

In [154]:
```
df2 = pd.DataFrame(np.arange(16).reshape([4,4]),
                   index = ["a", "b", "c", "d"],
                   columns = ["Tokyo", "Chiba", "Saitama", "Kanagaw
a"])

print('df2 < 7:')
df2 < 7
```

```
df2 < 7:
```

Out[154]:

	Tokyo	Chiba	Saitama	Kanagawa
a	True	True	True	True
b	True	True	True	False
c	False	False	False	False
d	False	False	False	False

In [155]:
```
print('df[df2 < 7] = 0:')
df2[df2 < 7] = 0
df2
```

```
df[df2 < 7] = 0:
```

Out[155]:

	Tokyo	Chiba	Saitama	Kanagawa
a	0	0	0	0
b	0	0	0	7
c	8	9	10	11
d	12	13	14	15

行・列を指定した pd.DataFrame アクセス

pd.DataFrame["インデックス"] のようなアクセスは列に対してのみおこなわれる。
行単位でアクセスしたい場合は、インデックスで指定する pd.DataFrame.loc, あるいは行番号で指定するpd.DataFram.iloc 属性を利用する。 これらは、行・列インデックスあるいは行・列数で指定する:

In [156]:
```
import pandas as pd
df2 = pd.DataFrame(np.arange(16).reshape([4,4]),
                   index = ["a", "b", "c", "d"],
                   columns = ["Tokyo", "Chiba", "Saitama", "Kanagaw
a"])
print("Sample:")
df2
```

```
Sample:
```

Out[156]:

	Tokyo	Chiba	Saitama	Kanagawa
a	0	1	2	3
b	4	5	6	7
c	8	9	10	11
d	12	13	14	15

In [157]:
```
print('df2.loc["d", ["Tokyo", "Kanagawa"]]')
df2.loc["d", ["Tokyo", "Kanagawa"]]        # index oriented
```

```
df2.loc["d", ["Tokyo", "Kanagawa"]]
```

Out[157]:
```
Tokyo       12
Kanagawa    15
Name: d, dtype: int64
```

In [158]:
```
print('df2.iloc[3, [0,3]]')
df2.iloc[3, [0,3]]        # number oriented
```

```
df2.iloc[3, [0,3]]
```

Out[158]:
```
Tokyo       12
Kanagawa    15
Name: d, dtype: int64
```

もちろんスライスも使える。ただし、インデックスで指定する `pd.DataFrame.loc()` では stop も含まれることに注意:

In [159]:
```
import pandas as pd
df2 = pd.DataFrame(np.arange(16).reshape([4,4]),
                   index = ["a", "b", "c", "d"],
                   columns = ["Tokyo", "Chiba", "Saitama", "Kanagaw
a"])
print("Sample:")
df2
```

```
Sample:
```

Out[159]:

	Tokyo	Chiba	Saitama	Kanagawa
a	0	1	2	3
b	4	5	6	7
c	8	9	10	11
d	12	13	14	15

In [160]:
```
print('df2.loc["b":"c",:"Saitama"]')
df2.loc["b":"c", :"Saitama"]      # index oriented , to include
stop index number
```

```
df2.loc["b":"c",:"Saitama"]
```

Out[160]:

	Tokyo	Chiba	Saitama
b	4	5	6
c	8	9	10

In [161]:
```
'df2.iloc[2:3,:2]'
df2.iloc[2:3,:2]                    # number oriented, not to inc
lude stop index number
```

Out[161]:

	Tokyo	Chiba
c	8	9

行・列を指定して 1 要素のみにアクセスするには、 pd.DataFrame.at あるいは pd.DataFrame.iat 属性を利用する。属性なので代入もできる:

In [162]:
```
import pandas as pd
df2 = pd.DataFrame(np.arange(16).reshape([4,4]),
                   index = ["a", "b", "c", "d"],
                   columns = ["Tokyo", "Chiba", "Saitama", "Kanagaw
a"])
print("Sample:")
print(df2)
print()
print('df2.at["c","Saitama"]')
print(df2.at["c","Saitama"])        # index oriented , to includ
e stop index number
print()

print('df2.at[2,2]')
print(df2.iat[2,2])                 # number oriented
print()

print('after df2.at["b","Saitama"] = 9999')
df2.at["b","Saitama"] = 9999
print(df2)
print()
```

```
Sample:
   Tokyo  Chiba  Saitama  Kanagawa
a      0      1        2         3
b      4      5        6         7
c      8      9       10        11
d     12     13       14        15

df2.at["c","Saitama"]
10

df2.at[2,2]
10

after df2.at["b","Saitama"] = 9999
   Tokyo  Chiba  Saitama  Kanagawa
a      0      1        2         3
b      4      5     9999         7
c      8      9       10        11
d     12     13       14        15
```

pd.Series, pd.DataFrame の算法

演算子の性質は行・列番号ではなく、インデックスによっておこなわれることを除けばNumPy.ndarray とほぼ同じ。
多くの演算が要素単位におこなわれる。

pd.Seriesへの演算子の適用は同じインデックスをもつ要素同士でおこなわれ、インデックスの調整（拡張）もおこなわれる:

In [163]:
```
import pandas as pd
data_dict = {"Tokyo":1000, "Kanagawa":900, "Chiba":800}
obj3 = pd.Series(data_dict)
obj4 = pd.Series(data_dict, index=["Tokyo", "Chiba", "Saitama"])

print("obj3:")
print(obj3)
print()
print("obj4:")
print(obj4)
print()
print("obj3 + obj4:")
print(obj3 + obj4)
```

```
obj3:
Chiba        800
Kanagawa     900
Tokyo       1000
dtype: int64

obj4:
Tokyo      1000.0
Chiba       800.0
Saitama       NaN
dtype: float64

obj3 + obj4:
Chiba       1600.0
Kanagawa       NaN
Saitama        NaN
Tokyo       2000.0
dtype: float64
```

pd.DataFrame も同じ:

In [164]:
```
import pandas as pd

df2 = pd.DataFrame(np.arange(16).reshape([4,4]),
                   index = ["a", "b", "c", "d"],
                   columns = ["Tokyo", "Chiba", "Saitama", "Kanagaw
a"])

df3 = pd.DataFrame(np.arange(16, 32).reshape([4,4]),
                   index = ["a", "b", "e", "f"],
                   columns = ["Tokyo", "Chiba", "Ibaragi", "Yamanas
hi"])

df2 + df3
```

Out[164]:

	Chiba	Ibaragi	Kanagawa	Saitama	Tokyo	Yamanashi
a	18.0	NaN	NaN	NaN	16.0	NaN
b	26.0	NaN	NaN	NaN	24.0	NaN
c	NaN	NaN	NaN	NaN	NaN	NaN
d	NaN	NaN	NaN	NaN	NaN	NaN
e	NaN	NaN	NaN	NaN	NaN	NaN
f	NaN	NaN	NaN	NaN	NaN	NaN

`pd.DataFrame.add()` メソッドでは演算の際に NaN を埋める`fill_value`オプションもある（両方の値がなければ NaN のまま）：

In [165]:
```
import pandas as pd

df2 = pd.DataFrame(np.arange(16).reshape([4,4]),
                   index = ["a", "b", "c", "d"],
                   columns = ["Tokyo", "Chiba", "Saitama", "Kanagaw
a"])

df3 = pd.DataFrame(np.arange(16, 32).reshape([4,4]),
                   index = ["a", "b", "e", "f"],
                   columns = ["Tokyo", "Chiba", "Ibaragi", "Yamanas
hi"])

df2.add(df3, fill_value=0)
```

Out[165]:

	Chiba	Ibaragi	Kanagawa	Saitama	Tokyo	Yamanashi
a	18.0	18.0	3.0	2.0	16.0	19.0
b	26.0	22.0	7.0	6.0	24.0	23.0
c	9.0	NaN	11.0	10.0	8.0	NaN
d	13.0	NaN	15.0	14.0	12.0	NaN
e	25.0	26.0	NaN	NaN	24.0	27.0
f	29.0	30.0	NaN	NaN	28.0	31.0

In [166]:
```
1/df2
```

Out[166]:

	Tokyo	Chiba	Saitama	Kanagawa
a	inf	1.000000	0.500000	0.333333
b	0.250000	0.200000	0.166667	0.142857
c	0.125000	0.111111	0.100000	0.090909
d	0.083333	0.076923	0.071429	0.066667

汎用関数と関数 `map`

Numpy の汎用関数、`np.exp`, `np.sqrt` などは、`pd.Series`, `pd.DataFrame`各要素に対して適用される:

In [167]:
```
import pandas as pd

df2 = pd.DataFrame(np.arange(16).reshape([4,4]),
                   index = ["a", "b", "c", "d"],
                   columns = ["Tokyo", "Chiba", "Saitama", "Kanagaw
a"])
np.sqrt(df2)
```

Out[167]:

	Tokyo	Chiba	Saitama	Kanagawa
a	0.000000	1.000000	1.414214	1.732051
b	2.000000	2.236068	2.449490	2.645751
c	2.828427	3.000000	3.162278	3.316625
d	3.464102	3.605551	3.741657	3.872983

よく使われる別の方法として、関数mapも利用される:

In [168]:
```
import pandas as pd

df2 = pd.DataFrame(np.arange(16).reshape([4,4]),
                   index = ["a", "b", "c", "d"],
                   columns = ["Tokyo", "Chiba", "Saitama", "Kanagaw
a"])
df2
```

Out[168]:

	Tokyo	Chiba	Saitama	Kanagawa
a	0	1	2	3
b	4	5	6	7
c	8	9	10	11
d	12	13	14	15

In [169]:
```
f = lambda x: x.sum()
df2.apply(f)
```

Out[169]:
```
Tokyo       24
Chiba       28
Saitama     32
Kanagawa    36
dtype: int64
```


In [170]:
```
import numpy as np
import pandas as pd
ser = pd.Series(np.random.randn(8),
                  index=[["a", "a", "b", "c", "d", "d", "e", "e"]
,
                         ["2000","2010","2000","2000","2000","201
0","2000","2010"]])
ser
```

Out[170]:
```
a  2000   -0.606417
   2010    0.504888
b  2000   -0.947369
c  2000    0.228710
d  2000   -0.063714
   2010   -0.318073
e  2000   -1.951952
   2010    0.716546
dtype: float64
```

これをインデックスで指定すると:

In [171]:
```
ser["a"]
```

Out[171]:
```
2000   -0.606417
2010    0.504888
dtype: float64
```

In [172]:
```
ser["b":"c"]
```

Out[172]:
```
b  2000   -0.947369
c  2000    0.228710
dtype: float64
```

In [173]:
```
ser.loc[["a", "d"]]
```

Out[173]:
```
a  2000   -0.606417
   2010    0.504888
d  2000   -0.063714
   2010   -0.318073
dtype: float64
```

`iloc()` の働きは（もちろん）同じ:

In [174]:
```
ser.iloc[2:4]
```

Out[174]:
```
b  2000   -0.947369
c  2000    0.228710
dtype: float64
```

低い階層のインデックスも指定するには、コンマで区切る:

In [175]:
```
ser.loc[:,"2000"]
```

Out[175]:
```
a   -0.606417
b   -0.947369
c    0.228710
d   -0.063714
e   -1.951952
dtype: float64
```

インデックスからデータセットを作り直すには、`pd.unstack()` が使える:

In [176]:
```
ser.unstack()
```

Out[176]:

	2000	2010
a	-0.606417	0.504888
b	-0.947369	NaN
c	0.228710	NaN
d	-0.063714	-0.318073
e	-1.951952	0.716546

`pd.stack()` は逆をおこなう:

In [177]:
```
ser.unstack().stack()
```

Out[177]:
```
a  2000   -0.606417
   2010    0.504888
b  2000   -0.947369
c  2000    0.228710
d  2000   -0.063714
   2010   -0.318073
e  2000   -1.951952
   2010    0.716546
dtype: float64
```

`pd.DataFrame` の Multiindex

`pd.DataFrame` では行・列いずれにも Multiindex は適用できる:

In [178]:
```
import numpy as np
import pandas as pd
df = pd.DataFrame(np.random.randn(24).reshape(6,4),
                  index =[["a", "a", "b", "b", "c", "c"],["2000",
"2010", "2000", "2010", "2000", "2010"]],
                  columns = [["Tokyo", "Tokyo", "Saitama", "Nagan
o"],["Mainland", "Isrands", "Mainland", "Mainland"]])
df
```

Out[178]:

		Tokyo		Saitama	Nagano
		Mainland	Isrands	Mainland	Mainland
a	2000	1.610434	1.579511	-2.132923	-1.529732
	2010	0.193866	0.564393	1.504812	0.803416
b	2000	-1.172775	-0.268446	-0.767634	1.585794
	2010	0.752033	-0.443915	1.378504	-0.158456
c	2000	-0.549010	0.576572	-1.214103	-0.717802
	2010	2.740482	0.071307	1.351915	1.529581

インデックスの指定は、pd.Seriesと同じ:

In [179]:
```
df["Tokyo"]
```

Out[179]:

		Mainland	Isrands
a	2000	1.610434	1.579511
	2010	0.193866	0.564393
b	2000	-1.172775	-0.268446
	2010	0.752033	-0.443915
c	2000	-0.549010	0.576572
	2010	2.740482	0.071307

In [180]:
```
df["Tokyo","Mainland"]
```

Out[180]:
```
a  2000    1.610434
   2010    0.193866
b  2000   -1.172775
   2010    0.752033
c  2000   -0.549010
   2010    2.740482
Name: (Tokyo, Mainland), dtype: float64
```


In [181]:
```
df.loc["a"]
```

Out[181]:

	Tokyo		Saitama	Nagano
	Mainland	Isrands	Mainland	Mainland
2000	1.610434	1.579511	-2.132923	-1.529732
2010	0.193866	0.564393	1.504812	0.803416

ただし、pd.DataFrame なのでスライスは行に対して適用される:

In [182]:
```
df["a":"b"]
```

Out[182]:

		Tokyo		Saitama	Nagano
		Mainland	Isrands	Mainland	Mainland
a	**2000**	1.610434	1.579511	-2.132923	-1.529732
	2010	0.193866	0.564393	1.504812	0.803416
b	**2000**	-1.172775	-0.268446	-0.767634	1.585794
	2010	0.752033	-0.443915	1.378504	-0.158456

整数インデックス（参考）

Pandas の添字の与え方は、通常の Python とちがうことに注意する。
たとえば、以下はエラーとなる。インデックスが整数型で、インデックスにない番号でアクセスされたため:

In [183]:
```
import pandas as pd
import numpy as np
ser = pd.Series(np.arange(3.0))
print(ser)
print(ser[-1])
```

一方で、以下のようにインデックスが非整数の場合はエラーとならない:

In [184]:
```
import pandas as pd
import numpy as np
ser2 = pd.Series(np.arange(3.0), index=["a", "b", "c"])
print(ser2)
print(ser2[-1])
```

```
a    0.0
b    1.0
c    2.0
dtype: float64
2.0
```

