## Data Frame Column Indexing

Pandas provides three common ways of indexing a data frame. `df` is a data frame with 8 rows and 4 columns. The columns are labled A, B, C, and D. The rows are labled from 0 to 7.

```
>>> import numpy as np
>>> import pandas as pd
>>>
>>> df = pd.DataFrame(
...       np.random.randint(0,10,(8,4)),
...       columns=['A', 'B', 'C', 'D'])
>>> df.index
RangeIndex(start=0, stop=8, step=1)
>>> df.columns
Index(['A', 'B', 'C', 'D'], dtype='object')
>>> df
   A  B  C  D
0  3  9  2  2
1  1  0  2  6
2  4  4  5  7
3  7  3  2  7
4  1  0  4  7
5  6  1  4  5
6  1  8  8  9
7  5  4  7  1
```

The `A` column can be selected by one of the following:

```
>>> df['A']  # __getitem__
0    3
1    1
2    4
3    7
4    1
5    6
6    1
7    5
Name: A, dtype: int64
>>> df.A # can be used if the column name is a proper identifier
0    3
1    1
2    4
3    7
4    1
5    6
6    1
7    5
Name: A, dtype: int64
>>> df.loc[ :, 'A'] # pandas location indexing
0    3
1    1
2    4
3    7
4    1
5    6
6    1
7    5
```

```
Name: A, dtype: int64
>>> # df.loc['A'] is an error
>>> df.iloc[ :, 0] # first column by position
0    3
1    1
2    4
3    7
4    1
5    6
6    1
7    5
Name: A, dtype: int64
```

Rows and columns are selected with `df.loc` by `df.loc[ row-select, column-select]`. The row-select and column-select use the data frame's labels. `df.iloc` uses the positional indexes.

A collection of columns can be selected with:

```
>>> df[ ['A','B'] ]
   A  B
0  3  9
1  1  0
2  4  4
3  7  3
4  1  0
5  6  1
6  1  8
7  5  4
>>> df[ ['B','A'] ] # order is important
   B  A
0  9  3
1  0  1
2  4  4
3  3  7
4  0  1
5  1  6
6  8  1
7  4  5
>>> df.loc[ :, 'A':'B' ] # a pandas slice
   A  B
0  3  9
1  1  0
2  4  4
3  7  3
4  1  0
5  6  1
6  1  8
7  5  4
>>> df.iloc[ :, 0:2 ] # a python slice
   A  B
0  3  9
1  1  0
2  4  4
3  7  3
4  1  0
5  6  1
6  1  8
7  5  4
```

Is the following good practice?

```
>>> df[ list('AC') ]
   A  C
0  3  2
1  1  2
2  4  5
3  7  2
4  1  4
5  6  4
6  1  8
7  5  7
```

## Data Frame Column is a Series

The values returned by selecting a single column is a `pd.Series`.

```
>>> c = df[ 'C' ]
>>> c
0    2
1    2
2    5
3    2
4    4
5    4
6    8
7    7
Name: C, dtype: int64
>>> type(c)
<class 'pandas.core.series.Series'>
```

The `pd.Series` can be indexed to return an item from it. The index of the column is the same as the row index from the data frame.

```
>>> c.index
RangeIndex(start=0, stop=8, step=1)
```

The third item is accessed with:

```
>>> c[2]  # by label
5
>>> c.loc[ 2] # by label
5
>>> c.iloc[ 2] # by position
5
```

Using a copy and modifying the index of the copy to see the difference between labels and positions.

```
>>> cc = c.copy()
>>> cc.index += 10
>>> cc[ 12 ] # by label
5
```

```
>>> cc.loc[ 12] # by label
5
>>> cc.iloc[ 2] # by position
5
```

A slice returns a slice of the `pd.Series`.

```
>>> cc[ 2:3 ]
12    5
Name: C, dtype: int64
>>> type( cc[ 2:3 ] )
<class 'pandas.core.series.Series'>
>>> type( cc[ 12 ] )
<class 'numpy.int64'>
```

An item can be accessed from a data frame with two index operations, first by column, returning a series, and then with the series.

```
>>> df['C'][2] # by label, label
5
>>> df.loc[:,'C'][2]  # by label, label
5
>>> df.iloc[:,2][2]  # by position, label
5
>>> df['C'].loc[2] # by label, label
5
>>> df.loc[:,'C'].loc[2] # by label, label
5
>>> df.iloc[:,2].loc[2]  # by position, label
5
>>> df['C'].iloc[2] # by label, position
5
>>> df.loc[:,'C'].iloc[2] # by label, position
5
>>> df.iloc[:,2].iloc[2] # by postion, position
5
```

However for `loc` and `iloc` it makes more sense to be direct.

```
>>> df.loc[2,'C']
5
>>> df.iloc[2, 2]
5
```

## Data Frame Row Indexing

The third row of `df` can be accessed with:

```
>>> df[ 2:3 ] # position using a slice
   A  B  C  D
2  4  4  5  7
>>> type( _ ) # returns a pd.DataFrame slice
<class 'pandas.core.frame.DataFrame'>
```

```
>>> df.loc[ 2 ] # by label
A    4
B    4
C    5
D    7
Name: 2, dtype: int64
>>> type( _ )
<class 'pandas.core.series.Series'>
>>> df.iloc[ 2 ] # by position
A    4
B    4
C    5
D    7
Name: 2, dtype: int64
>>> type( _ )
<class 'pandas.core.series.Series'>
```

`df[ selector ]` is the same as `df.__getitem__( selector )`. The type of selector is used to determine what indexing operation to perform.

For `df.loc` and `df.iloc` the rest of the selector is provided by default. An explicit version is:

```
>>> df.loc[ 2, : ] # by label
A    4
B    4
C    5
D    7
Name: 2, dtype: int64
>>> df.iloc[ 2, : ] # by position
A    4
B    4
C    5
D    7
Name: 2, dtype: int64
```

## Shared numpy array

Pandas used numpy to store its data. A series uses a 1D array and a data frame uses a 2D array. Slices in numpy reference the same storage.

```
>>> arr = np.random.randint(0,10,(8,4))
>>> arr
array([[6, 4, 4, 3],
       [4, 2, 2, 2],
       [9, 0, 0, 5],
       [4, 4, 7, 4],
       [8, 5, 0, 0],
       [5, 7, 3, 8],
       [0, 7, 1, 1],
       [7, 9, 1, 4]])
```

Changes to a numpy slice also changes the associated numpy array. A column slice example:

```
>>> arr[ :, 1]
array([4, 2, 0, 4, 5, 7, 7, 9])
```

```
>>> arr[ :, 1] += 10
>>> arr[ :, 1]
array([14, 12, 10, 14, 15, 17, 17, 19])
>>> arr
array([[ 6, 14,  4,  3],
       [ 4, 12,  2,  2],
       [ 9, 10,  0,  5],
       [ 4, 14,  7,  4],
       [ 8, 15,  0,  0],
       [ 5, 17,  3,  8],
       [ 0, 17,  1,  1],
       [ 7, 19,  1,  4]])
```

A row slice example:

```
>>> arr[ 3, :]
array([ 4, 14,  7,  4])
>>> arr[ 3, :] += 10
>>> arr[ 3, :]
array([14, 24, 17, 14])
>>> arr
array([[ 6, 14,  4,  3],
       [ 4, 12,  2,  2],
       [ 9, 10,  0,  5],
       [14, 24, 17, 14],
       [ 8, 15,  0,  0],
       [ 5, 17,  3,  8],
       [ 0, 17,  1,  1],
       [ 7, 19,  1,  4]])
```

A slice with rows and columns:

```
>>> arr[ 6:, 2:]
array([[1, 1],
       [1, 4]])
>>> arr[ 6:, 2:] += 20
>>> arr[ 6:, 2:]
array([[21, 21],
       [21, 24]])
>>> arr
array([[ 6, 14,  4,  3],
       [ 4, 12,  2,  2],
       [ 9, 10,  0,  5],
       [14, 24, 17, 14],
       [ 8, 15,  0,  0],
       [ 5, 17,  3,  8],
       [ 0, 17, 21, 21],
       [ 7, 19, 21, 24]])
```

## Shared numpy array in Pandas

The identical behaviour with pandas for column slices is:

```
>>> cdf = df.copy()
>>> cdf
   A  B  C  D
```

```
0  3  9  2  2
1  1  0  2  6
2  4  4  5  7
3  7  3  2  7
4  1  0  4  7
5  6  1  4  5
6  1  8  8  9
7  5  4  7  1
>>> cdf.iloc[ :, 1]
0    9
1    0
2    4
3    3
4    0
5    1
6    8
7    4
Name: B, dtype: int64
>>> cdf.iloc[ :, 1] += 10
>>> cdf.iloc[ :, 1]
0    19
1    10
2    14
3    13
4    10
5    11
6    18
7    14
Name: B, dtype: int64
>>> cdf
   A   B  C  D
0  3  19  2  2
1  1  10  2  6
2  4  14  5  7
3  7  13  2  7
4  1  10  4  7
5  6  11  4  5
6  1  18  8  9
7  5  14  7  1
```

A row slice example:

```
>>> cdf.iloc[ 3, :]
A     7
B    13
C     2
D     7
Name: 3, dtype: int64
>>> cdf.iloc[ 3, :] += 10
>>> cdf.iloc[ 3, :]
A    17
B    23
C    12
D    17
Name: 3, dtype: int64
>>> cdf
    A   B   C   D
0   3  19   2   2
1   1  10   2   6
2   4  14   5   7
```

```
3  17  23  12  17
4   1  10   4   7
5   6  11   4   5
6   1  18   8   9
7   5  14   7   1
```

A slice with rows and columns:

```
>>> cdf.iloc[ 6:, 2:]
   C  D
6  8  9
7  7  1
>>> cdf.iloc[ 6:, 2:] += 20
>>> cdf.iloc[ 6:, 2:]
    C   D
6  28  29
7  27  21
>>> cdf
    A   B   C   D
0   3  19   2   2
1   1  10   2   6
2   4  14   5   7
3  17  23  12  17
4   1  10   4   7
5   6  11   4   5
6   1  18  28  29
7   5  14  27  21
```

The underlying numpy array is accessed with:

```
>>> cdf.values
array([[ 3, 19,  2,  2],
       [ 1, 10,  2,  6],
       [ 4, 14,  5,  7],
       [17, 23, 12, 17],
       [ 1, 10,  4,  7],
       [ 6, 11,  4,  5],
       [ 1, 18, 28, 29],
       [ 5, 14, 27, 21]])
>>> type(cdf.values)
<class 'numpy.ndarray'>
>>> # set all odd numbers to 0
>>> cdf.values[ (cdf.values % 2) == 1 ] = 0
>>> cdf.values
array([[ 0,  0,  2,  2],
       [ 0, 10,  2,  6],
       [ 4, 14,  0,  0],
       [ 0,  0, 12,  0],
       [ 0, 10,  4,  0],
       [ 6,  0,  4,  0],
       [ 0, 18, 28,  0],
       [ 0, 14,  0,  0]])
>>> cdf
   A   B   C  D
0  0   0   2  2
1  0  10   2  6
2  4  14   0  0
3  0   0  12  0
```

```
4   0   10    4   0
5   6    0    4   0
6   0   18   28   0
7   0   14    0   0
```