

A Series is like an array: a 1-D list of homogenously-typed items.

```
In [1]: import pandas as pd
a = pd.Series([1, 2, 3])
a
```

```
Out[1]: 0    1
        1    2
        2    3
```

```
In [2]: a.dtype
```

```
Out[2]: dtype('int64')
```

Here's a Series() of floating point numbers:

```
In [3]: b = pd.Series([1, 2.3, 3])
b
```

```
Out[3]: 0    1.0
        1    2.3
        2    3.0
```

```
In [4]: b.dtype
```

```
Out[4]: dtype('float64')
```

Of course if you mix things up, everything in Python is an object in the end.

```
In [5]: c = pd.Series(['a', None, 5])
c
```

```
Out[5]: 0    a
        1  None
        2    5
```

```
In [6]: c.dtype
```

```
Out[6]: dtype('object')
```

## Broadcasting operations across a Series

You can apply conditional expressions to a Series, and it will return another Series with the result of that expression applied to each value. NumPy calls this "broadcasting".

```
In [7]: a
```

```
Out[7]: 0    1
        1    2
        2    3
```

```
In [8]: a > 1
```

```
Out[8]: 0    False
        1     True
        2     True
```

```
In [9]: a == 1
```

```
Out[9]: 0     True
        1    False
        2    False
```

It's also easy to broadcast your own callable by using `Series.map()`:

```
In [10]: a.map(lambda x: x % 2 == 0)
```

```
Out[10]: 0    False
         1     True
         2    False
```

## DataFrames

A DataFrame is essentially a set of Series objects (as columns) with a shared index (the row labels).

```
In [11]: d = pd.DataFrame(
        [
            [1, 2.3, 'three'],
            [4, 5, 6],
            [7, 8, 9],
            [10, 11, 12]],
        columns=['Integers', 'Floats', 'Objects'],
        index=[1, 2, 3, 4])
d
```

Out[11]:

	Integers	Floats	Objects
1	1	2.3	three
2	4	5.0	6
3	7	8.0	9

4	10	11.0	12
---	----	------	----

In [12]: `d.dtypes`

```
Out[12]: Integers      int64
          Floats       float64
          Objects      object
```

## Selecting data

Selecting by column by using a key lookup:

In [13]: `d['Floats']`

```
Out[13]: 1      2.3
          2      5.0
          3      8.0
          4     11.0
          Name: Floats
```

You can look up two columns by indexing using a list of columns:

In [14]: `d[['Integers', 'Objects']]`

Out[14]:

	Integers	Objects
1	1	three
2	4	6
3	7	9
4	10	12

You can select a range of rows using list slices. Note that this refers to the rows as if they were in a Python `list()`!

In [15]: `d[2:]`

Out[15]:

	Integers	Floats	Objects
3	7	8	9
4	10	11	12

You can also avoid the magic and just use `DataFrame.xs()` to access the rows by their indexed name:

```
In [16]: d.xs(3, axis=0)
```

```
Out[16]: Integers    7
         Floats      8
         Objects     9
         Name: 3
```

Or specifying column names:

```
In [17]: d.xs('Floats', axis=1)
```

```
Out[17]: 1    2.3
         2    5.0
         3    8.0
         4   11.0
         Name: Floats
```

Row indexing can also be done using a mask:

```
In [18]: mask = [True, False, True, False]
         d[mask]
```

Out[18]:

	Integers	Floats	Objects
1	1	2.3	three
3	7	8.0	9

Combined with conditional expression broadcasting, and you can get some really interesting results:

```
In [19]: where_Integers_is_gt_4 = d['Integers'] > 4
         d[where_Integers_is_gt_4]
```

Out[19]:

	Integers	Floats	Objects
3	7	8	9
4	10	11	12

```
In [20]: d[np.invert(where_Integers_is_gt_4)]
```

Out[20]:

	Integers	Floats	Objects
1	1	2.3	three

1	1	2.5	three
2	4	5.0	6

To get a subset of the rows based on the index value:

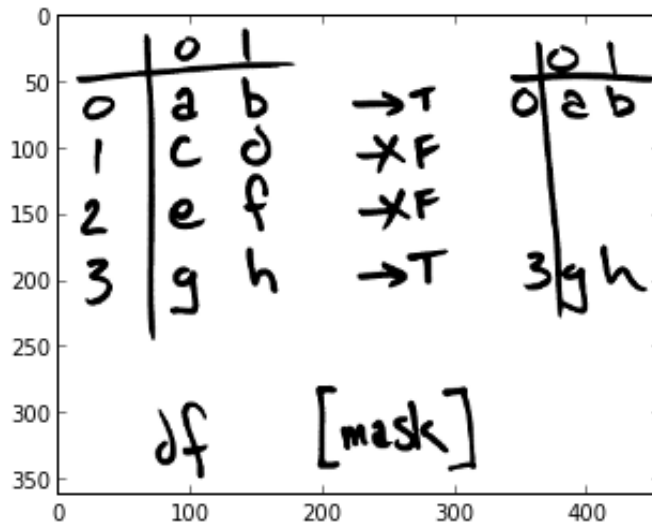
In [21]: `d[d.index > 2]`

Out[21]:

	Integers	Floats	Objects
3	7	8	9
4	10	11	12

In [22]: `imshow(imread('masking.png'), cmap='gray')`

Out[22]: `<matplotlib.image.AxesImage at 0x10521a3d0>`



In [22]: