# Changing index of a DataFrame

Indexes are immutable objects. This means that if you want to change or modify the index in a DataFrame, then you need to change the whole index. You will do this now, using a list comprehension to create the new index.

A list comprehension is a succinct way to generate a list in one line. For example, the following list comprehension generates a list that contains the cubes of all numbers from 0 to 9: cubes = [i\*\*3 for i in range(10)]. This is equivalent to the following code:

cubes = []

for i in range(10):

cubes.append(i\*\*3)

Create a sales DataFrame from sales.csv file and verify that the index is given by month abbreviations containing lowercase characters, if not then set the Month as index

##### INSTRUCTIONS

* Create a list new\_idx with the same elements as in sales.index, but with all characters capitalized.
* Assign new\_idx to sales.index.
* Print the sales dataframe. # Create the list of new indexes: new\_idx

# Changing index name labels

Notice that in the previous exercise, the index was not labeled with a name. In this exercise, you will set its name to 'MONTHS'.

Similarly, if all the columns are related in some way, you can provide a label for the set of columns.

To get started, print the sales DataFrame in the IPython Shell and verify that the index has no name, only its data (the month names).

##### INSTRUCTIONS

* Assign the string 'MONTHS' to sales.index.name to create a name for the index.
* Print the sales dataframe to see the index name you just created.
* Now assign the string 'PRODUCTS' to sales.columns.name to give a name to the set of columns.
* Print the sales dataframe again to see the columns name you just created.

# Building an index, then a DataFrame

You can also build the DataFrame and index independently, and then put them together. If you take this route, be careful, as any mistakes in generating the DataFrame or the index can cause the data and the index to be aligned incorrectly.

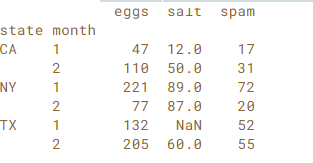
In this exercise, first delete the column Mont. Your job is to build this index separately and then assign it to the sales DataFrame. Before getting started, print the sales DataFrame in the IPython Shell and note that it's missing the month information.

##### INSTRUCTIONS

* Generate a list months with the data ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']. This has been done for you.
* Assign months to sales.index.
* Print the modified sales dataframe and verify that you now have month information in the index.

# Extracting data with a MultiIndex

Create a new column State referring the image below to the existing sales dataframe.



Extracting elements from the outermost level of a MultiIndexis just like in the case of a single-level Index. You can use the .loc[] accessor

##### INSTRUCTIONS

* Print sales.loc[['CA', 'TX']]. Note how New York is excluded.
* Print sales['CA':'TX']. Note how New York is included.

# Setting & sorting a MultiIndex

In the previous exercise, the MultiIndex was created and sorted for you. Now, you're going to do this yourself! With a MultiIndex, you should always ensure the index is sorted. You can skip this only if you know the data is already sorted on the index fields.

To get started, print the sales DataFrame in the IPython Shell to verify that there is no MultiIndex.

##### INSTRUCTIONS

* Create a MultiIndex by setting the index to be the columns ['state', 'month'].
* Sort the MultiIndex using the .sort\_index() method.
* Print the sales DataFrame to verify that indeed you have an index with the fields state and month!

# Using .loc[] with nonunique indexes

It is always preferable to have a meaningful index that uniquely identifies each row. Even though pandas does not require unique index values in DataFrames, it works better if the index values are indeed unique. To see an example of this, you will index your sales data by 'state' in this exercise.

As always, begin by printing the sales DataFrame in the IPython Shell and inspecting it.

##### INSTRUCTIONS

* Set the index of sales to be the column 'state'.
* Print the sales DataFrame to verify that indeed you have an index with state values.
* Access the data from 'NY' and print it to verify that you obtain two rows.

# Indexing multiple levels of a MultiIndex

Looking up indexed data is fast and efficient. And you have already seen that lookups based on the outermost level of a MultiIndex work just like lookups on DataFrames that have a single-level Index.

Looking up data based on inner levels of a MultiIndex can be a bit trickier. In this exercise, you will use your sales DataFrame to do some increasingly complex lookups.

The trickiest of all these lookups are when you want to access some inner levels of the index. In this case, you need to use slice(None) in the slicing parameter for the outermost dimension(s) instead of the usual :, or use pd.IndexSlice. The following code to extract rows from all Symbols for the dates Oct. 3rd through 4th inclusive:

stocks.loc[(slice(None), slice('2016-10-03', '2016-10-04')), :]

Pay particular attention to the tuple (slice(None), slice('2016-10-03', '2016-10-04')).

##### INSTRUCTIONS

* Look up data for the New York column ('NY') in month 1.
* Look up data for the California and Texas columns ('CA', 'TX') in month 2.
* Look up data for all states in month 2. Use (slice(None), 2) to extract all rows in month 2.