

# Indexing

Understand how DataFrame values can be accessed via indexing.

## Chapter Goals:

- Learn how to index a DataFrame to retrieve rows and columns
- Write code for indexing a DataFrame

### A. Direct indexing

When indexing into a DataFrame, we can treat the DataFrame as a dictionary of Series objects, where each column represents a Series. Each column label then becomes a key, allowing us to directly retrieve columns using dictionary-like bracket notation.

The code below shows how to directly index into a DataFrame's columns.

```
df = pd.DataFrame({'c1': [1, 2], 'c2': [3, 4],  
                  'c3': [5, 6]}, index=['r1', 'r2'])  
  
col1 = df['c1']  
# Newline for separating print statements  
print('{}\n'.format(col1))  
  
col1_df = df[['c1']]  
print('{}\n'.format(col1_df))  
  
col23 = df[['c2', 'c3']]  
print('{}\n'.format(col23))
```



Note that when we use a single column label inside the bracket (as was the case for `col1` in the code example), the output is a Series representing the corresponding column. When we use a list of column labels (as was the case for `col1_df` and `col23`), the output is a DataFrame that contains the corresponding columns.

We can also use direct indexing to retrieve a subset of the rows (as a

DataFrame). However, we can only retrieve rows based on slices, rather than specifying particular rows.

The code below shows how to directly index into a DataFrame's rows.

```
df = pd.DataFrame({'c1': [1, 2, 3], 'c2': [4, 5, 6],
                  'c3': [7, 8, 9]}, index=['r1', 'r2', 'r3'])

print('{}\n'.format(df))

first_two_rows = df[0:2]
print('{}\n'.format(first_two_rows))

last_two_rows = df['r2':'r3']
print('{}\n'.format(last_two_rows))

# Results in KeyError
df['r1']
```

You'll notice that when we used integer indexing for the rows, the end index was exclusive (e.g. `first_two_rows` excluded the row at index 2). However, when we use row labels, the end index is inclusive (e.g. `last_two_rows` included the row labeled `'r3'`).

Furthermore, when we tried to retrieve a single row based on its label, we received a `KeyError`. This is because the DataFrame treated `'r1'` as a column label.

## B. Other indexing

Apart from direct indexing, a DataFrame object also contains the `loc` and `iloc` properties for indexing.

We use `iloc` to access rows based on their integer index. Using `iloc` we can access a single row as a Series, and specify particular rows to access through a list of integers or a boolean array.

The code below shows how to use `iloc` to access a DataFrame's rows.

```
df = pd.DataFrame({'c1': [1, 2, 3], 'c2': [4, 5, 6],
                  'c3': [7, 8, 9]}, index=['r1', 'r2', 'r3'])

print('{}\n'.format(df))

print('{}\n'.format(df.iloc[1]))
```

```
print('{}\n'.format(df.iloc[[0, 2]]))

bool_list = [False, True, True]
print('{}\n'.format(df.iloc[bool_list]))
```



The `loc` property provides the same row indexing functionality as `iloc`, but uses row labels rather than integer indexes. Furthermore, with `loc` we can perform column indexing along with row indexing, and set new values in a DataFrame for specific rows and columns.

The code below shows example usages of `loc`.

```
df = pd.DataFrame({'c1': [1, 2, 3], 'c2': [4, 5, 6],
                  'c3': [7, 8, 9]}, index=['r1', 'r2', 'r3'])

print('{}\n'.format(df))

print('{}\n'.format(df.loc['r2']))

bool_list = [False, True, True]
print('{}\n'.format(df.loc[bool_list]))

single_val = df.loc['r1', 'c2']
print('Single val: {}\n'.format(single_val))

print('{}\n'.format(df.loc[['r1', 'r3'], 'c2']))

df.loc[['r1', 'r3'], 'c2'] = 0
print('{}\n'.format(df))
```



You'll notice that the way we access rows and columns together with `loc` is similar to how we access 2-D NumPy arrays.

Since we can't access columns on their own with `loc` or `iloc`, we still use bracket indexing when retrieving columns of a DataFrame.

## Time to Code!

The coding exercises for this chapter involve directly indexing into a predefined DataFrame, `df`.

We'll initially use direct indexing to get the first column of `df` as well as the first two rows.

Set `col_1` equal to `df` directly indexed with `'c1'` as the key.

Set `row_12` equal to `df` directly indexed with `0:2` as the key.

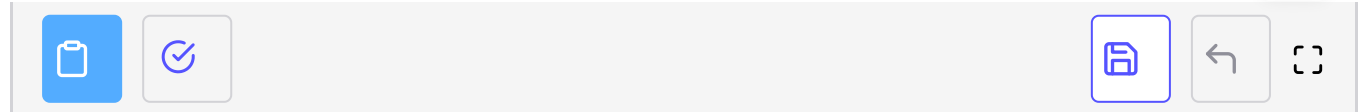
```
# CODE HERE
```



Next, we'll use `iloc` to retrieve the first and third rows of `df`.

Set `row_13` equal to `df.iloc` indexed with `[0, 2]` as the key.

```
# CODE HERE
```



Finally, we use `loc` to set each value of the second column, in the third and fourth rows, equal to 12. The row key we use for indexing will be `['r3', 'r4']`, while the column key will be `'c2'`.

Set `df.loc`, indexed with the specified row and column keys, equal to 12.

```
# CODE HERE
```

