```
                           'skills': ['math', 'spreadsheets', 'coding', 'linux',
                                      'spreadsheets', 'organization']})
print(df1); print(df5); print(pd.merge(df1, df5))
```

```
df1                          df5
  employee        group                group        skills
0      Bob   Accounting      0   Accounting          math
1     Jake  Engineering      1   Accounting  spreadsheets
2     Lisa  Engineering      2  Engineering        coding
3      Sue           HR      3  Engineering         linux
                             4           HR  spreadsheets
                             5           HR  organization

pd.merge(df1, df5)
  employee        group        skills
0      Bob   Accounting          math
1      Bob   Accounting  spreadsheets
2     Jake  Engineering        coding
3     Jake  Engineering         linux
4     Lisa  Engineering        coding
5     Lisa  Engineering         linux
6      Sue           HR  spreadsheets
7      Sue           HR  organization
```

These three types of joins can be used with other Pandas tools to implement a wide array of functionality. But in practice, datasets are rarely as clean as the one we're working with here. In the following section, we'll consider some of the options provided by `pd.merge()` that enable you to tune how the join operations work.

Specification of the Merge Key

We've already seen the default behavior of `pd.merge()`: it looks for one or more matching column names between the two inputs, and uses this as the key. However, often the column names will not match so nicely, and `pd.merge()` provides a variety of options for handling this.

The on keyword

Most simply, you can explicitly specify the name of the key column using the `on` keyword, which takes a column name or a list of column names:

```
In[6]: print(df1); print(df2); print(pd.merge(df1, df2, on='employee'))
```

```
df1                          df2
  employee        group           employee  hire_date
0      Bob   Accounting        0     Lisa       2004
1     Jake  Engineering        1      Bob       2008
2     Lisa  Engineering        2     Jake       2012
3      Sue           HR        3      Sue       2014
```

```
pd.merge(df1, df2, on='employee')
  employee        group  hire_date
0      Bob   Accounting       2008
1     Jake  Engineering       2012
2     Lisa  Engineering       2004
3      Sue           HR       2014
```

This option works only if both the left and right `DataFrames` have the specified column name.

The left_on and right_on keywords

At times you may wish to merge two datasets with different column names; for example, we may have a dataset in which the employee name is labeled as "name" rather than "employee". In this case, we can use the `left_on` and `right_on` keywords to specify the two column names:

```
In[7]:
df3 = pd.DataFrame({'name': ['Bob', 'Jake', 'Lisa', 'Sue'],
                    'salary': [70000, 80000, 120000, 90000]})
print(df1); print(df3);
print(pd.merge(df1, df3, left_on="employee", right_on="name"))
```

```
df1                          df3
  employee        group          name  salary
0      Bob   Accounting        0   Bob   70000
1     Jake  Engineering        1  Jake   80000
2     Lisa  Engineering        2  Lisa  120000
3      Sue           HR        3   Sue   90000


pd.merge(df1, df3, left_on="employee", right_on="name")
  employee        group  name  salary
0      Bob   Accounting   Bob   70000
1     Jake  Engineering  Jake   80000
2     Lisa  Engineering  Lisa  120000
3      Sue           HR   Sue   90000
```

The result has a redundant column that we can drop if desired—for example, by using the `drop()` method of `DataFrames`:

```
In[8]:
pd.merge(df1, df3, left_on="employee", right_on="name").drop('name', axis=1)
```

```
Out[8]:   employee        group  salary
        0      Bob   Accounting   70000
        1     Jake  Engineering   80000
        2     Lisa  Engineering  120000
        3      Sue           HR   90000
```

The left_index and right_index keywords

Sometimes, rather than merging on a column, you would instead like to merge on an index. For example, your data might look like this:

```
In[9]: df1a = df1.set_index('employee')
       df2a = df2.set_index('employee')
       print(df1a); print(df2a)
```

```
df1a                         df2a
                group                      hire_date
employee                     employee
Bob        Accounting        Lisa           2004
Jake      Engineering        Bob            2008
Lisa      Engineering        Jake           2012
Sue                HR        Sue            2014
```

You can use the index as the key for merging by specifying the `left_index` and/or `right_index` flags in `pd.merge()`:

```
In[10]:
print(df1a); print(df2a);
print(pd.merge(df1a, df2a, left_index=True, right_index=True))
```

```
df1a                         df2a
                group                      hire_date
employee                     employee
Bob        Accounting        Lisa           2004
Jake      Engineering        Bob            2008
Lisa      Engineering        Jake           2012
Sue                HR        Sue            2014

pd.merge(df1a, df2a, left_index=True, right_index=True)
                 group  hire_date
employee
Lisa      Engineering       2004
Bob        Accounting       2008
Jake      Engineering       2012
Sue                HR       2014
```

For convenience, `DataFrames` implement the `join()` method, which performs a merge that defaults to joining on indices:

```
In[11]: print(df1a); print(df2a); print(df1a.join(df2a))
```

```
df1a                        df2a
                group                     hire_date
employee                    employee
Bob        Accounting       Lisa           2004
Jake      Engineering       Bob            2008
Lisa      Engineering       Jake           2012
Sue                HR       Sue            2014
```

```
df1a.join(df2a)
                group  hire_date
employee
Bob        Accounting       2008
Jake      Engineering       2012
Lisa      Engineering       2004
Sue                HR       2014
```

If you'd like to mix indices and columns, you can combine `left_index` with `right_on` or `left_on` with `right_index` to get the desired behavior:

```
In[12]:
print(df1a); print(df3);
print(pd.merge(df1a, df3, left_index=True, right_on='name'))
```

```
df1a                          df3
                group
employee                      name  salary
Bob        Accounting    0   Bob   70000
Jake      Engineering    1  Jake   80000
Lisa      Engineering    2  Lisa  120000
Sue                HR    3   Sue   90000

pd.merge(df1a, df3, left_index=True, right_on='name')
         group  name  salary
0   Accounting   Bob   70000
1  Engineering  Jake   80000
2  Engineering  Lisa  120000
3           HR   Sue   90000
```

All of these options also work with multiple indices and/or multiple columns; the interface for this behavior is very intuitive. For more information on this, see the "Merge, Join, and Concatenate" section of the Pandas documentation.

Specifying Set Arithmetic for Joins

In all the preceding examples we have glossed over one important consideration in performing a join: the type of set arithmetic used in the join. This comes up when a value appears in one key column but not the other. Consider this example:

```
In[13]: df6 = pd.DataFrame({'name': ['Peter', 'Paul', 'Mary'],
                            'food': ['fish', 'beans', 'bread']},
                           columns=['name', 'food'])
        df7 = pd.DataFrame({'name': ['Mary', 'Joseph'],
                            'drink': ['wine', 'beer']},
                           columns=['name', 'drink'])
        print(df6); print(df7); print(pd.merge(df6, df7))
```