

# KEY\_Lesson10\_Pandas-Intro

July 15, 2019

## 1 Introduction to Pandas

You may have used a program like Microsoft Excel or Google Sheets to record data or perform calculations for school. Datasets are often organized in rows and columns. Here's an example of a table:

Product	Price	Quantity Sold
Apples	\$ 1.50	26
Bananas	\$ 0.50	32
Lemons	\$ 1.99	17

In this case, we have prices of fruit from a grocery store. Each row in this dataset corresponds to a type of fruit, and each column is an observation or measurement about that fruit. What measurements do we have here?

**Measurements:** Price, Quantity Sold

To handle table data in Python, we use a package called pandas. (Despite the name, it actually doesn't have anything to do with panda bears - disappointing, we know.)

Before we can use the package, we need to import it. Try writing a line of code to import the pandas package:

```
[0]: # write the code to import the package
import pandas
```

We're going to be using pandas a lot. To save us some time typing, let's tell Python to rename the package for us to something a little shorter:

```
[0]: import pandas as pd
```

Adding "as pd" to our import statement tells Python that every time we use pd, we actually mean pandas. It's like giving the package a nickname! Most people use this same nickname for pandas, so if you see other people's code, you'll know that pd means pandas.

Now that we've imported pandas, we're ready to use it. Datasets are stored in pandas in a special container called a DataFrame. DataFrames help us handle data that are organized in rows and columns, just like the grocery store dataset above. We can create that DataFrame by calling the DataFrame function:

```
[3]: pd.DataFrame({'Product': ['Apples', 'Bananas', 'Lemons'],
                    'Price': [1.50, 0.50, 1.99],
                    'QuantitySold': [26, 32, 17]})
```

```
[3]:   Product  Price  QuantitySold
0   Apples   1.50             26
1  Bananas   0.50             32
2   Lemons   1.99             17
```

There are a few things to notice:

1. Python helpfully showed us what our DataFrame looks like with rows and columns.
2. Python gave each row in the DataFrame a number, starting from zero. Does that remind you of something about lists?

When we ran that cell, Python showed us what the DataFrame looks like, but it didn't save the DataFrame anywhere. If we want to do more things with it, we'll need to save it to a variable - the same way we learned to save integers or strings to variables. In the below cell, save the same DataFrame to a variable called `fruit_data`:

```
[0]: # save the DataFrame to a variable called fruit_data
fruit_data = pd.DataFrame({'Product': ['Apples', 'Bananas', 'Lemons'],
                           'Price': [1.50, 0.50, 1.99],
                           'QuantitySold': [26, 32, 17]})
```

When we run the above cell, does anything happen?

Since we saved the DataFrame to a variable, Python assumed that we don't want to take a peek at it. If we do want to see what our DataFrame looks like, we can write the variable on a line by itself like this:

```
[5]: # print out fruit_data
fruit_data
```

```
[5]:   Product  Price  QuantitySold
0   Apples   1.50             26
1  Bananas   0.50             32
2   Lemons   1.99             17
```

Sometimes we don't want to see all of the data at once. To view just the first few rows of a DataFrame, we use a method called `head`.

Recall: methods are special functions that belong to certain types of variables. They only work with the type of variable that they belong to - so the `head` method only works on pandas dataframe objects.

Let's use the `head` method to view only the first two rows of our fruit data:

```
[6]: fruit_data.head(2)
```

```
[6]:   Product  Price  QuantitySold
0   Apples   1.5             26
1  Bananas   0.5             32
```

After typing `head`, the name of the method, we wrote the number 2 between the parentheses. This told `head` that we wanted to see only the first two lines of our DataFrame. What would you do if you only wanted to see the first line of `fruit_data`?

```
[7]: # write the code to see only the first line of the DataFrame
fruit_data.head(1)
```

```
[7]: Product Price QuantitySold
0 Apples 1.5 26
```

Great! What if instead of viewing the first few lines, we wanted to see the last few lines? For that we will use another method called `tail`. It is very similar to `head`, except it will show us the last few lines instead of the first few. Try writing a line of code to view the very last line of `fruit_data`:

```
[8]: # write the code to see only the last line of the DataFrame
fruit_data.tail(1)
```

```
[8]: Product Price QuantitySold
2 Lemons 1.99 17
```

`head` and `tail` are especially useful for big datasets with many rows so you can get a sense for what the `DataFrame` looks like without flooding your notebook with too much information. They will come in handy later!

Now let's talk about the types of variables we have in the `fruit_data` `DataFrame`. What are some of the types of variables we've learned about in Python?

**Types:** string, int, float, list

Based on what you know about these types of variables, what type do you think the values in each of the columns belong to in `fruit_data`?

**Type of the Product column:** string (explanation: the names of the fruit are words)

**Type of the Price column:** float (explanation: the prices have decimal points)

**Type of the QuantitySold column:** int (explanation: quantity sold is a count. stores won't sell only half a banana.)

Notice how each of the columns in `fruit_data` have different types. That's one of the many cool things about pandas -- it let's us store many different types of data in `DataFrames`.