**The Series Data Structure**

The series is one of the core data structures in pandas. You think of it a cross between a list and a dictionary.
The items are all stored in an order and there are labels with which you can retrieve them.
An easy way to visualize this is two columns of data. The first is the special index, a lot like the dictionary keys. While the second is your actual data.

```
In [5]: import pandas as pd
        pd.Series?

Out[5]: pandas.core.series.Series
```

```
Init signature:
pd.Series(
    data=None,
    index=None,
    dtype=None,
    name=None,
    copy=False,
    fastpath=False,
)
```

It's important to note that the data column has a label of its own and can be retrieved using the .name attribute.
This is different than with dictionaries and is useful when it comes to merging multiple columns of data.

As you might expect, you can create a series by passing in a list of values.
When you do this, Pandas automatically assigns an index starting with zero and sets the name of the series to None.
Let's see an example of this.
First, I'll start off by importing the pandas library as pd, then let's take a look at the series object.
Here you could see the documentation indicates that you can pass in some data, an index and a name. The data can be anything, that's array-like, like a list.
So let's give that a try.
We'll just make a list of the three of animals, a tiger, a bear and a moose.
We'll see here that the pandas has automatically identified the type of the data being held in the list, in this case we passed in a list of strings and panda set the type to object.

```
: animals = ['Tiger', 'Bear', 'Moose']
  pd.Series(animals)

: 0    Tiger
  1     Bear
  2    Moose
  dtype: object
```

We don't have to use strings. If we passed in a list of whole numbers, for instance, we could see that panda sets the type to int 64.

```
numbers = [1, 2, 3]
pd.Series(numbers)
```

```
0    1
1    2
2    3
dtype: int64
```

Underneath panda stores series values in a typed array using the Numpy library.
This offers significant speed-up when processing data versus traditional python lists.

There are some other typing details that exist for performance that are important to know.
The most important is how Numpy and thus pandas handle missing data. In Python, we have the *none* type to indicate a lack of data.

```
animals = ['Tiger', 'Bear', None]
pd.Series(animals)
```

```
0    Tiger
1     Bear
2     None
dtype: object
```

But what do we do if we want to have a typed list like we do in the series object?

Underneath, pandas does some type conversion.
If we create a list of strings and we have one element, a None type, pandas inserts it as a None and uses the type object for the underlying array.
If we create a list of numbers, integers or floats, and put in the None type, pandas automatically converts this to a special floating point value designated as
NAN, which stands for not a number.

```
numbers = [1, 2, None]
pd.Series(numbers)
```

```
0    1.0
1    2.0
2    NaN
dtype: float64
```

For those who might not have done scientific computing in Python before, this is a pretty important point.

NAN is not none and when we try the equality test, it's false.

It turns out that you actually can't do an equality test of NAN to itself.
When you do, the answer is always false. You need to use special functions to test for the presence of not a number, such as the Numpy library is NAN.

```
import numpy as np
np.nan == None
```

```
False
```

```
np.nan == np.nan
```

```
False
```

```
np.isnan(np.nan)
```

```
True
```

Keep in mind when you see NAN, it's meaning is similar to none, but it's a numeric value and it's treated differently for efficiency reasons.

Let's talk more about how pandas' series can be created.

While my list of animals might be a common way to create some play data, often you have label data that you want to manipulate.
A series can be created from dictionary data.
If you do this, the index is automatically assigned to the keys of the dictionary that you provided and not just incrementing integers.
Here's an example using some data from Wikipedia on official national sports.
When we create the series, we see that, since it was string data, panda set the data type of the series to object.
We set the list of the countries as the value of the series and that the index values can be set to the keys from our dictionary.

```
sports = {'Archery': 'Bhutan',
          'Golf': 'Scotland',
          'Sumo': 'Japan',
          'Taekwondo': 'South Korea'}
s = pd.Series(sports)
s
```

```
Archery           Bhutan
Golf            Scotland
Sumo               Japan
Taekwondo    South Korea
dtype: object
```

```
s.index
```

```
Index(['Archery', 'Golf', 'Sumo', 'Taekwondo'], dtype='object')
```

Once the series has been created, we can get the index object using the index attribute.
You could also separate your index creation from the data by passing in the index as a list explicitly to the series.

```
s = pd.Series(['Tiger', 'Bear', 'Moose'], index=['India', 'America', 'Canada'])
s
```

```
India      Tiger
America     Bear
Canada     Moose
dtype: object
```

So what happens if your list of values in the index object are not aligned with the keys in your dictionary for creating the series?
Well, pandas overrides the automatic creation to favor only and all of the indices values that you provided.
So it will ignore it from your dictionary, all keys, which are not in your index, and pandas will add non type or NAN values for any index value you provide, which is not in your dictionary key list.
In this example, we pass in a dictionary of four items but only two are preserved in the series object because of the index list.
We see that hockey has been added but since it's also in the index list, it has no value associated with it.

```
sports = {'Archery': 'Bhutan',
          'Golf': 'Scotland',
          'Sumo': 'Japan',
          'Taekwondo': 'South Korea'}
s = pd.Series(sports, index=['Golf', 'Sumo', 'Hockey'])
s
```

```
Golf      Scotland
Sumo         Japan
Hockey         NaN
dtype: object
```

So that's how we create a series.