**Welcome to the case study!**

It's now time to flex those muscles and see what metal you're made of! In this chapter, you'll use your recently-gained skills of writing user-defined functions, iterators, list comprehensions and generators to wrangle and extract meaningful information from a real-world dataset,

**World bank data**

the World Bank World Development Indicators dataset. This dataset contains data on 217 world economies for over half a century, from 1960 up until 2015. The data contains hundreds of indicators from population, electricity consumption and CO2 emissions to literacy rates, unemployment and mortality rates. In the following few exercises, you'll use dictionaries, zipping, user-defined functions, and list comprehensions to wrangle some of this wonderfully rich dataset. Let's quickly recap these tools:

**Using zip()**

The zip function accepts an arbitrary number of iterables and returns an iterator of tuples. We bring back our two lists from before, one of the avengers, the other of their names. Passing these to the zip function creates a zip object which is an iterator of tuples. We can turn them into a list and then print this list.

**Defining a function**

For writing functions, function headers begin with the keyword def, followed by the function name, arguments inside parentheses and a colon. We then have the function body, with the docstrings enclosed in triple quotation marks; the rest of the function body performs the computation that the function does and closes with the keyword return, followed by the value or values to return.

**Re-cap: list comprehensions**

Comprehensions, in their most basic forms, are enclosed in square brackets and are structured as output expression for iterator variable in iterable. More advanced comprehensions can include conditionals on the output expression and/or conditionals on the iterable.

**Using Python generators for streaming data**

Congratulations on your first foray into the World Bank's World Indicators dataset! Having played around with some of the data, we're now going to step back a bit and look at importing the dataset, which is pretty large. We already saw that, for a large dataset,

**Generators for the large data limit**

we can use an iterator to load it in chunks. We can also write a generator to load it in line by line and one of the really cool things about this method is that if the data is streaming, that is, if new lines are being written to the file you're reading, this method will keep on reading and processing the file until there are no lines left for it to read. In the following exercises,

**Build a generator function**

you'll be writing a generator function that can process large amounts of data, so let me remind you of our typical example of a generator function:Here is a generator function that, when called with a number n, produces a generator object that generates integers 0 though n. In the function definition, i is initialized to 0. The first time the generator object is called, it yields i equal to 0. It then adds one to i and will then yield one on the next iteration and so on. The while loop is true until i equals equals n and then the generator ceases to yield values. In the exercises, you'll be writing a generator function that, on each iteration, reads a new line of the file!

**Using pandas' read\_csv iterator for streaming data**

You're on the home stretch now! That was some great work with generators to read in large and/or streaming files.

**Reading files in chunks**

In the remaining exercises, you'll revisit the pandas function read\_csv, along with its chunksize argument in order to read in the World Indicator data in chunks, process each chunk and aggregate. Herein, you'll do so in order to look at specific indicators in specific countries and to plot the results over all the years for which the relevant data exists. You'll then try your hand at writing a function to perform the same task which will save so much work when you want to look at the data for a different country.