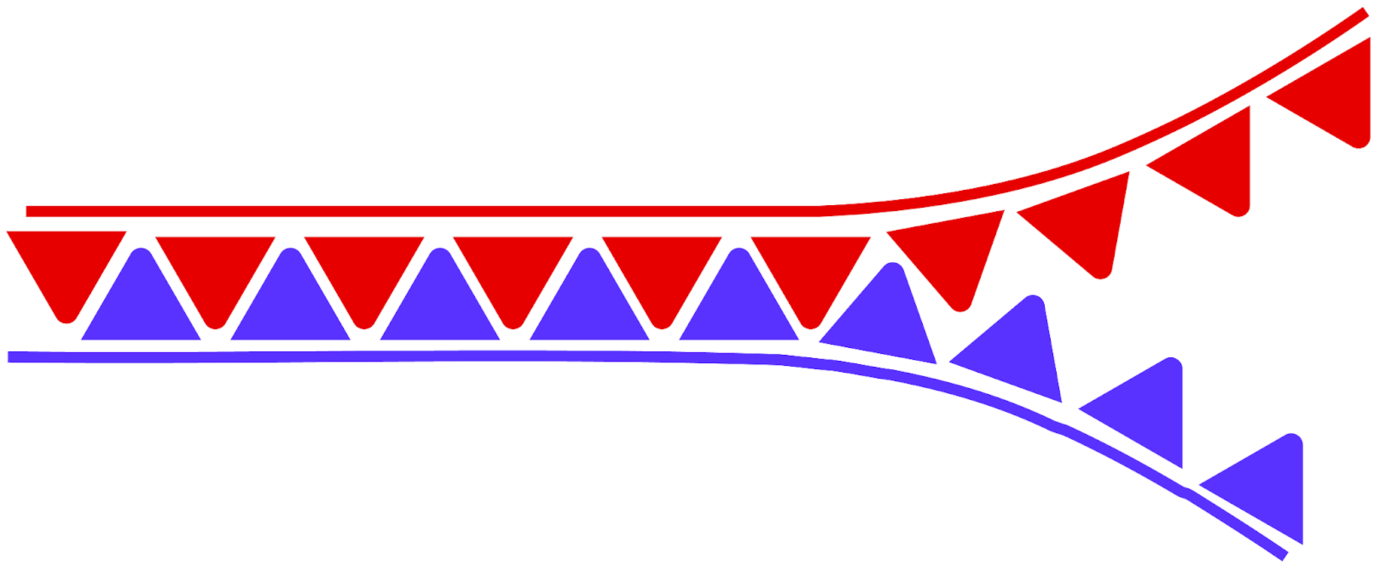
Reading: zip(), enumerate(), and list comprehension

You've learned much about iterable objects such as strings, lists, and tuples, and soon you'll learn more. These objects comprise many of Python's core data structures and, as a data professional, you'll work with them constantly. While working in Python, you'll often need to perform the same tasks and operations many times. This reading will introduce you to three time-saving tools: **zip()**, **enumerate()**, and list comprehension.

zip()

The zip() function is a built-in Python function that does what the name implies: It performs an element-wise combination of sequences.



The function returns an **iterator** that produces tuples containing elements from each of the input sequences. An iterator is an object that enables processing of a collection of items one at a time without needing to assemble the entire collection at once. Use an iterator with loops or other iterable functions such as **list()** or **tuple()**. Here's an example:

```
cities = ['Paris', 'Lagos', 'Mumbai']
countries = ['France', 'Nigeria', 'India']
places = zip(cities, countries)


print(places)
print(list(places))
```

Output:

```
<zip object at 0x7f6a18a40908>
[('Paris', 'France'), ('Lagos', 'Nigeria'), ('Mumbai', 'India')]
```

Notice that, in this case, the **list()** function is used to generate a list of tuples from the iterator object. Here are a few things to keep in mind when using the **zip()** function.

- It works with two or more iterable objects. The given example zips two sequences, but the **zip()** function will accept more sequences and apply the same logic.
- If the input objects are of unequal length, the resulting iterator will be the same length as the shortest input.
- If you give it only one iterable object as an argument, the function will return an iterator that produces tuples containing only one element from that iterable at a time.

Unzipping

You can also unzip an object with the ***** operator. Here's the syntax:

```
scientists = [('Nikola', 'Tesla'), ('Charles', 'Darwin'), ('Marie', 'Curie')]
given_names, surnames = zip(*scientists)
print(given_names)
print(surnames)
```

Output:

```
('Nikola', 'Charles', 'Marie')
('Tesla', 'Darwin', 'Curie')
```

Note that this operation unpacks the tuples in the original list element-wise into two tuples, thus separating the data into different variables that can be manipulated further.

enumerate()

The enumerate() function is another built-in Python function that allows you to iterate over a sequence while keeping track of each element's index. Similar to **zip()**, it returns an iterator that produces pairs of indices and elements. Here's an example:

```
letters = ['a', 'b', 'c']
for index, letter in enumerate(letters):
  print(index, letter)
```

Output:

```
0 a
1 b
2 c
```

Note that the default starting index is zero, but you can assign it to whatever you want when you call the **enumerate()** function. For example:

```
letters = ['a', 'b', 'c']
for index, letter in enumerate(letters, 2):
  print(index, letter)
```

Output:

```
2 a
3 b
4 c
```

In this case, the number two was passed as an argument to the function, and the first element of the resulting iterator had an index of two. The **enumerate()** function is useful when an element's place in a sequence must be used to determine how the element should be handled in an operation.

List comprehension

One of the most useful tools in Python is list comprehension. List comprehension is a concise and efficient way to create a new list based on the values in an existing iterable object. List comprehensions take the following form:

my_list = [expression for element in iterable if condition]

In this syntax:

- **expression** refers to an operation or what you want to do with each element in the iterable sequence.
- **element** is the variable name that you assign to represent each item in the iterable sequence.
- **iterable** is the iterable sequence.
- **condition** is any expression that evaluates to **True** or **False**. This element is optional and is used to filter elements of the iterable sequence.

Here are some examples of list comprehensions:

This list comprehension adds 10 to each number in the list:

```
numbers = [1, 2, 3, 4, 5]
new_list = [x + 10 for x in numbers]
print(new_list)
```

Output:

```
[11, 12, 13, 14, 15]
```

In the preceding example, **x + 10** is the expression, **x** is the element, and **numbers** is the iterable sequence. There is no condition.

This next list comprehension extracts the first and last letter of each word as a tuple, but only if the word is more than five letters long.

```
words = ['Emotan', 'Amina', 'Ibeno', 'Sankwala']
new_list = [(word[0], word[-1]) for word in words if len(word) > 5]
print(new_list)
```

Output:

```
[('E', 'n'), ('S', 'a')]
```

Note that multiple operations can be performed in the expression component of the list comprehension to result in a list of tuples. This example also makes use of a condition to filter out words that are not more than five letters long.

Key takeaways

zip(), **enumerate()**, and list comprehension make code more efficient by reducing the need to rely on loops to process data and simplifying working with iterables. Understanding these common tools will save you time and make your process much more dynamic when manipulating data.