zip

zip() makes an iterator that aggregates elements from each of the iterables.

Returns an iterator of tuples, where the i-th tuple contains the i-th element from each of the argument sequences or iterables. The iterator stops when the shortest input iterable is exhausted. With a single iterable argument, it returns an iterator of 1-tuples. With no arguments, it returns an empty iterator.

zip() is equivalent to:

```
def zip(*iterables):
    # zip('ABCD', 'xy') --> Ax By
    sentinel = object()
    iterators = [iter(it) for it in iterables]
    while iterators:
        result = []
        for it in iterators:
            elem = next(it, sentinel)
            if elem is sentinel:
                return
            result.append(elem)
        yield tuple(result)
```

zip() should only be used with unequal length inputs when you don’t care about trailing, unmatched values from the longer iterables.

Lets see it in action in some examples:

Examples

In [1]:
```
x = [1,2,3]
y = [4,5,6]

# Zip the lists together
zip(x,y)
```

Out[1]:
```
[(1, 4), (2, 5), (3, 6)]
```

Note how tuples are returned. What if one iterable is longer than the other?

In [2]:
```
x = [1,2,3]
y = [4,5,6,7,8]

# Zip the lists together
zip(x,y)
```

Out[2]:
```
[(1, 4), (2, 5), (3, 6)]
```

Note how the zip is defined by the shortest iterable length. Its generally advised not to zip unequal length iterables unless your very sure you only need partial tuple pairings.

What happens if we try to zip together dictionaries?

In [4]:
```
d1 = {'a':1,'b':2}
d2 = {'c':4,'d':5}

zip(d1,d2)
```

Out[4]:
```
[('a', 'c'), ('b', 'd')]
```

This makes sense because simply iterating through the dictionaries will result in just the keys. We would have to call methods to mix keys and values:

In [6]:
```
zip(d2,d1.itervalues())
```

Out[6]:
```
[('c', 1), ('d', 2)]
```

Great! Finally lets use zip a to switch the keys and values of the two dictionaries:

In [7]:
```
def switcharoo(d1,d2):
    dout = {}

    for d1key,d2val in zip(d1,d2.itervalues()):
        dout[d1key] = d2val

    return dout
```

In [8]:
```
switcharoo(d1,d2)
```

Out[8]:
```
{'a': 4, 'b': 5}
```

Great! You can use zip to save a lot of typing in many situations! You should now have a good understanding of zip() and some possible use cases.