Multiplying Polynomials in Python

At this point, you have created a nice toolbox of functions for dealing with lists of coefficients as polynomials. Create a file called `poly.py` and copy the folowing functions into it:

- `evaluate_polynomial`

- `polynomial_to_string`

- `add_polynomials`

- `scalar_polynomial_multiply`

- `subtract_polynomial`

Now, create another file in the same directory called `test.py`. Type this into that file:

```python
import poly

polynomial_a = [9.0, -4.0, 3.0, -5.0]
print('Polynomial A =', poly.polynomial_to_string(polynomial_a))

polynomial_b = [-9.0, 0.0, 4.0, 2.0, 1.0]
print('Polynomial B =', poly.polynomial_to_string(polynomial_b))

# Evaluation
value_of_b = poly.evaluate_polynomial(polynomial_b, 3)
print('Polynomial B at 3 =', value_of_b)

# Adding
a_plus_b = poly.add_polynomials(polynomial_a, polynomial_b)
print('A + B =', poly.polynomial_to_string(a_plus_b))

# Scalar multiplication
b_scalar = poly.scalar_polynomial_multiply(-3.2, polynomial_b)
print('-3.2 * Polynomial B =', poly.polynomial_to_string(b_scalar))

# Subtraction
```

```
a_minus_b = poly.subtract_polynomial(polynomial_a, polynomial_b)
print('A - B =', poly.polynomial_to_string(a_minus_b))
```

When you run it, you should get the following:

```
Polynomial A = -5.0x^3 + 3.0x^2 + -4.0x + 9.0
Polynomial B = 1.0x^4 + 2.0x^3 + 4.0x^2 + -9.0
Polynomial B at 3 = 162.0
A + B = 1.0x^4 + -3.0x^3 + 7.0x^2 + -4.0x
-3.2 * Polynomial B = -3.2x^4 + -6.4x^3 + -12.8x^2 + 28.8
A - B = -1.0x^4 + -7.0x^3 + -1.0x^2 + -4.0x + 18.0
```

You are now ready to implement the multiplication of polynomials. The function will look like this:

```
def multiply_polynomials(a, b):
  ...Your code here...
```

It will return a list of coefficients.

In an exercise in the last chapter, you were asked " Let's say I have two polynomials, p_1 and p_2. p_1 has degree 23. p_2 has degree 12. What is the degree of their product?" The answer was $23 + 12 = 35$.

In our implementation, a polynomial of degree 23 is held in a list of length 24.

In Python, we wil be trying to multiply a polynomial a and a polynomial b represented as lists. What is the degree of that product?

```
        result_degree = (len(a) - 1) + (len(b) - 1)
```

Now, we need to create an array of zeros that is one longer than that. Here is a cute Python trick: If you have a list, you can replicate it using the * operator.

```
a = [5,7]
b = a * 4
print(b)
# [5, 7, 5, 7, 5, 7, 5, 7]
```

Here iss how you will get a list of zeros:

```
        result = [0.0] * (result_degree + 1)
```

We will step through a, getting the index and value of each entry. You can do this in one line using enumerate:

```
for a_degree, a_coefficient in enumerate(a):
```

For each of those, we will step through the entire b polynomial. As you multiply together each term, you will add it to the appropriate coefficient of the result.

Here is the whole function:

```
def multiply_polynomials(a, b): # What is the degree of the resulting
polynomial?  result_degree = (len(a) - 1) + (len(b) - 1)

    # Make a list of zeros to hold the coefficents result = [0.0] *
    (result_degree + 1)

    # Iterate over the indices and values of a for a_degree,
    a_coefficient in enumerate(a):

        # Iterate over the indices and values of b for b_degree,
        b_coefficient in enumerate(b):

            # Calculate the resulting monomial coefficient =
            a_coefficient * b_coefficient degree = a_degree + b_degree

            # Add it to the right bucket
            result[degree] = result[degree] + coefficient

    return result
```

Take a long look at that function. When you understand it, type it into poly.py.

In test.py, try out the new function:

```
# Multiplication
a_times_b = poly.multiply_polynomials(polynomial_a, polynomial_b)
print('A x B =', poly.polynomial_to_string(a_times_b))
```

This is an example of a *nested loop*. The outer loop steps through the polynomial a. For each step it takes, the inner loop steps through the entire polynomial b.

1.1   Something surprising about lists

You can imagine that you might want to create two very similar polynomials. Let's say polynomial c is $x^2 + 2x + 1$ and polynomial d is $x^2 - 2x + 1$. You might think you are very clever to just alter that degree 1 coefficient like this:

```
c = [1.0, 2.0, 1.0]
d = c
d[1] = -2.0
```

If you printed out c, you would get $[1.0, -2.0, 1.0]$. Why? You assigned two variables (c and d) to the *the same list*. So, when you use one reference (d) to change the list, you see the change if you look at the list from either reference. *FIXME: Diagram of two references to the same list here.*

To create two separate lists, you would need to explicitly make a copy:

```
c = [1.0, 2.0, 1.0]
d = c.copy()
d[1] = -2.0
```

This is a draft chapter from the Kontinua Project. Please see our website (https://kontinua. org/) for more details.

Answers to Exercises