

How to Give Your Python Code a Magic Touch

Building a powerful polynomial engine for easy study and research



Kasper Müller · 2 days ago · 6 min read ★

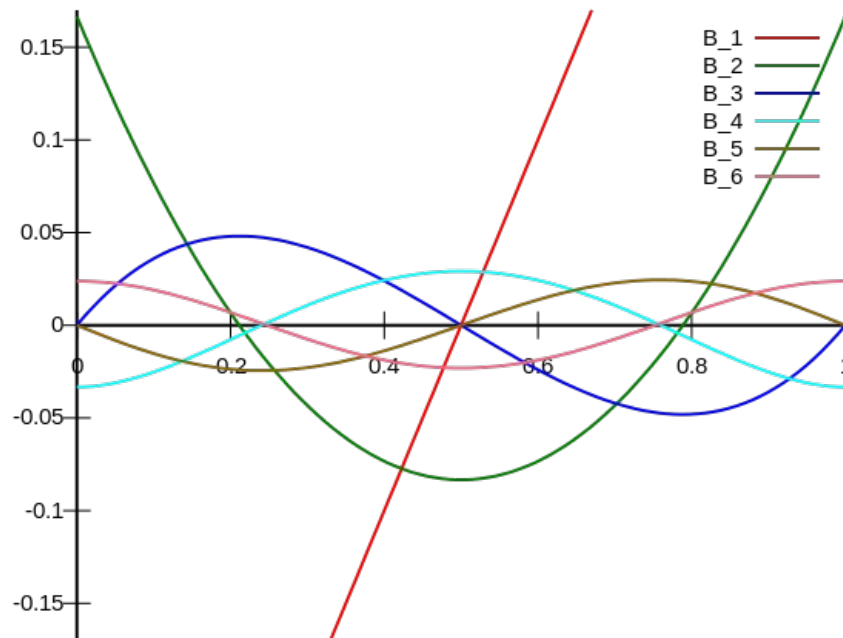


Image by [Wikimedia Commons](#)

In this article, I will show you how to build a tool for studying polynomials in Python.

Moreover, this tool should be intuitive to use and feel like the built-in types that Python comes with out of the box.

We will do this together using the so-called *dunder methods*.

Table of contents

- [Intro](#)
- [Magic methods](#)
 - [__str__](#)
 - [__len__](#)
 - [__getitem__](#)
 - [__eq__](#), [__add__](#), [__sub__](#), [__mul__](#) and [__floordiv__](#)
- [Polynomials as functions](#)
- [Zeros and Graphs](#)

Intro

A couple of years ago I discovered something interesting about a certain type of polynomial.

I wanted to study their properties but didn't have the tools to explore them properly. Armed with technology, I will take up the fight by building such a tool.

Before we build it though, let's have a quick refresher regarding Python's double-under or “*dunder*” methods.

Magic methods

Dunder methods, sometimes called *magic methods*, are methods of a class that begins and ends with two underscores.

These methods emulate the built-in functionality of Python like the use of keywords like **len**, **+**, **with** etc.

To better understand this, let's build a class that represents a polynomial and has support for Python syntax.

What is the minimum amount of information we need to describe a polynomial? It's the coefficients, isn't it?

Let try the following:

We definitely need the coefficients and the rest is just for show. The coefficients are just given as arguments to the class constructor starting with the constant term. We will give

the polynomial the default name f , which we can of course change when instantiating the object.

`__str__`

Speaking of “show”, let us make sure that the user sees something useful when he or she prints out the polynomial. We will add a dunder method called `__str__` to the above.

The output from the `__str__` method is what you see when you print out your object. If you don’t have that method in your class, then you will just get your class name printed out which might not be so useful.

Don’t worry too much about the content of the method, you will see what it does in a minute.

Now, we are able to print the polynomial out as we would expect to see it. You can of course change this to fit your needs, notice that we also have a `__repr__` dunder which is in the same family but is more for the developer than the user.

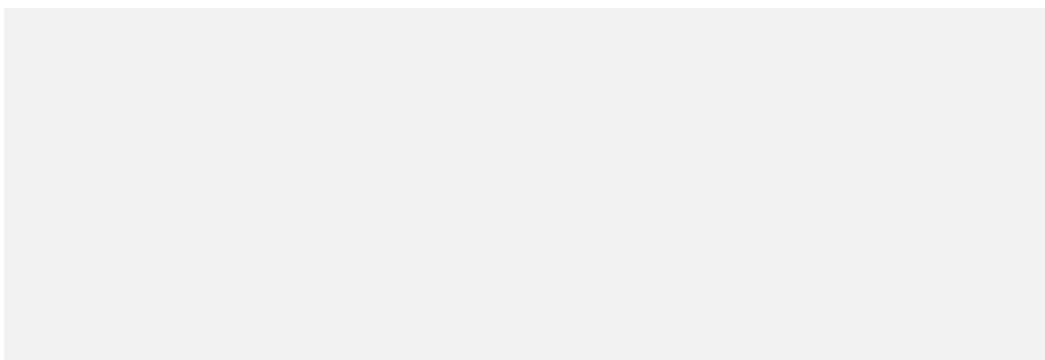




Image by author

This is all fine, but we have not really done anything with the polynomial yet. To add some functionality to the class let's create code for adding, subtracting, multiplying and dividing polynomials as well as other fun operations and functionalities.

`__len__`

The way we want to do this is to use dunder methods to gain intuitiveness to the tool. Specifically, the `__len__` method will let us use the built-in **len** function to display some kind of length to our polynomial. I have chosen to use the number of terms as the metric in this case.

`__getitem__`

When you slice a list or fetch the nth element, you use the `list[n]` syntax. We can do the same for the polynomials. In this case, I decided that we should get the nth coefficient in this case.

`__eq__`, `__add__`, `__sub__`, `__mul__` and `__floordiv__`

The above methods correspond to `==`, `+`, `-`, `*`, `//` respectively and the only method that requires explanation is probably the `__floordiv__` method which I decided to implement because might want to do polynomial division.

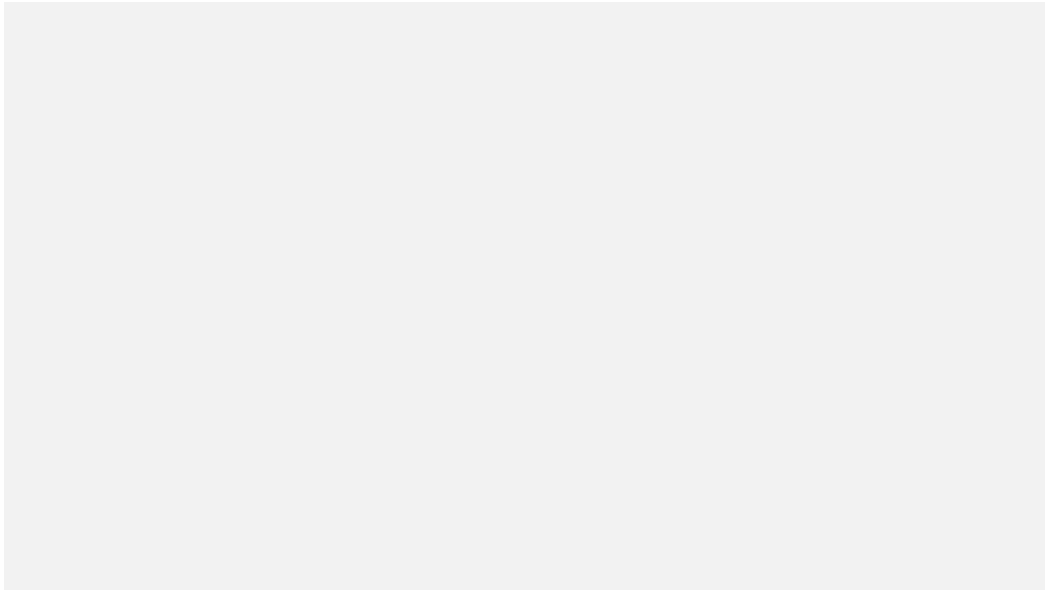
I am only returning the quotient and not the remainder if it should be non-zero. We could also have chosen to use the `__truediv__` dunder method, but I thought that the floor division made more sense in this case.

To better understand this, you should play around with the code yourself. For now, let us add the above methods to our class:

Polynomials as functions

We now have a bunch of options.

Notice that now Python understands how to add polynomials by using the symbol `+` for example which gives it a more natural feel when writing code. After all, we are now able to write it as we would write mathematics.



Let's add functionality for differentiating.

We will implement a *prime* property method and a more general method for differentiating n times called *diff*. We change the names accordingly to help the user keep track of the different objects.

Since *prime* is returning a Polynomial object as well, our other methods work on the derivative as well making us capable of printing it out nicely and to keep working with it.

We will also add a method called *negating_shift*.

The reason that we need this method has to do with the research that I mentioned at the top of this article. The method will take a polynomial $f(x)$ as well as a real number α and return the corresponding polynomial $f(\alpha - x)$.

Now, what would a polynomial be without being able to call it?

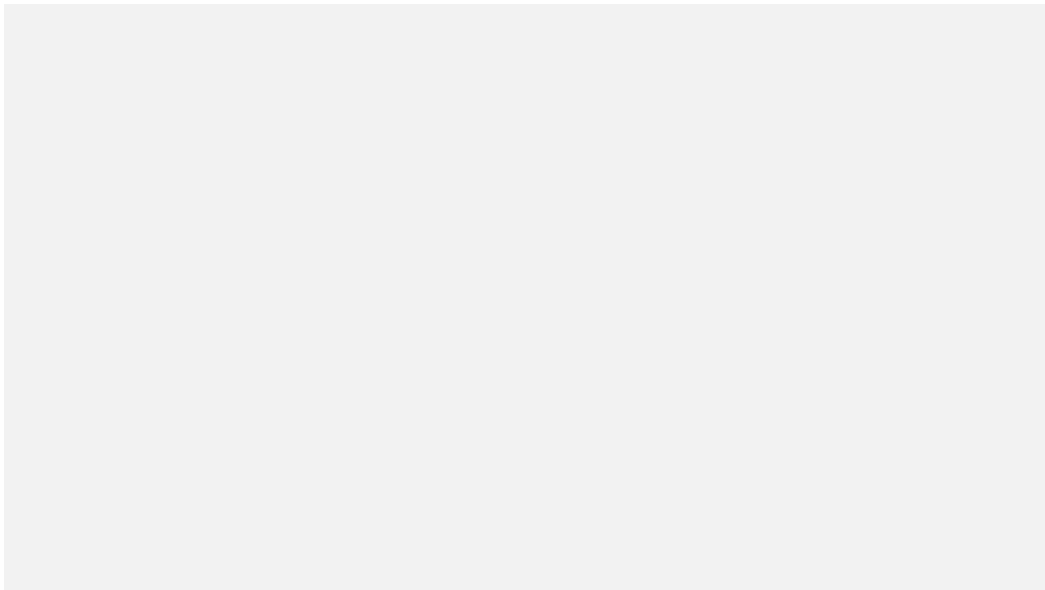
We need to make it a *function object* which should be able to evaluate real and complex numbers as we would use any other function.

Let us add all this to the module as well.

Full code up to this point:

Now we can differentiate n times by a simple method. Notice also that now we can call the polynomials like the functions they represent!

This is thanks to the `__call__` dunder method.



Zeros and graphs

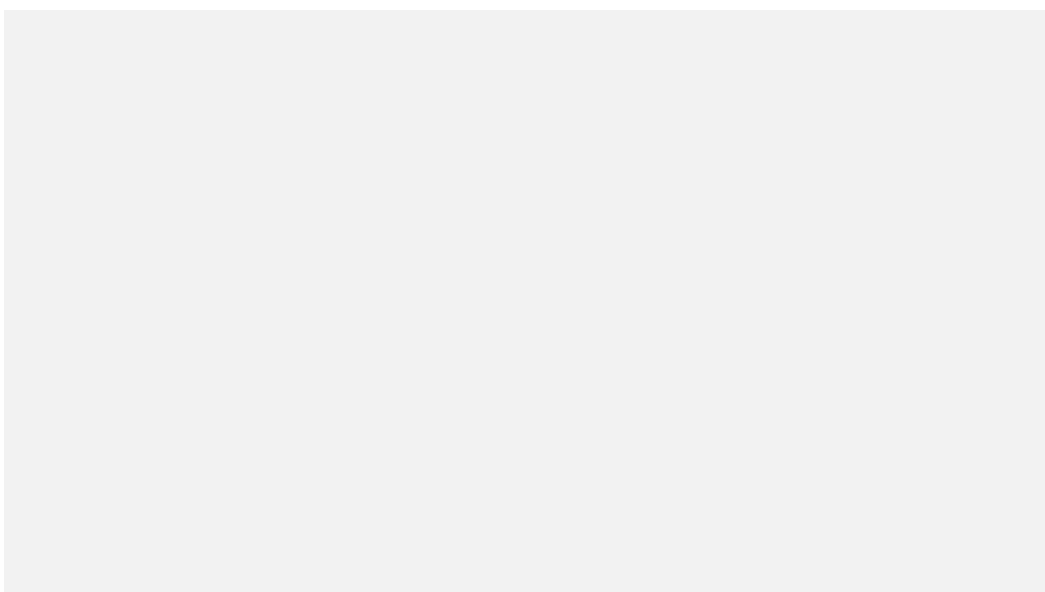
We definitely want to be able to find the roots and we also want to be able to display the graph of the polynomial and now that we have the roots, we might as well find the local minima and maxima of the function.

It will also be useful to be able to display the position of the roots in the complex plane.

Thus we can add methods for that too:

Notice that we are able to change the size of the axes as arguments. This will probably be convenient at some point.

Now we can get the roots or zeroes as well as make beautiful displays and graphs. We see below that the 5 roots of unity make a pentagram in the complex plane.



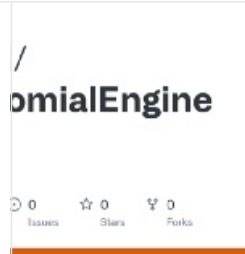
The points in the display are the complex zeros of the polynomial.

By now, we have a lot of nice Pythonic functionality in place. I will add more in the future like discriminants, factorization etc. In the meantime, it is time to study the class of polynomials that I wanted to study in the first place.

This will be a separate article since this one would become too long.


You can get the full code here:

GitHub - KRBM/PolynomialEngine
Specify a polynomial by its coefficients: We can add and subtract polynomials by + and - respectively and p.zeros gives...
github.com



If you have any questions, comments or concerns, please reach out on LinkedIn:

Kasper Müller - Senior Consultant, Data and Analytics, FS, Technology Consulting - EY | LinkedIn
Programming, mathematics and teaching are some of my greatest interests. Data science, machine learning, programming...
www.linkedin.com



Mathematics

Science

Programming

Technology

Python