To do these exercises you may need to “look things up”. Remember that google is your best friend.

Exercise 1
Consider the following set of seven x values $x = \text{np.array([1,2,3,4,5,6,7])}$ and the following function $f(x) = a_0x^2 + b_0x + c_0$ with $a_0 = 0.5$, $b_0 = 0.3$, $c_0 = 3$.
Construct another numpy array $y = f(x) + \delta$ where $\delta$ is another seven-dimensional numpy array where each element of the array is picked from a gaussian distribution of mean zero and standard deviation 2 (to be clear: the seven elements are not all the same). Refer to Exercise 5 in the discussion session of Week 4 if you do not know how to pick numbers from a Gaussian.
Next, fit $y$ vs. $x$ to a function $f(x) = Ax^2 + Bx + C$. In the fit assign an uncertainty of 2 to each point $y_i$. (Consistent with the “smear” $\delta$ that was used to construct the $y$ array). Print out the following quantities:

- The fitted vales of $A$, $B$, $C$.
- The uncertainties on these fitted value.
- The number of degrees of freedom.
- The value of $\chi^2$ at the minimum.
- The probability of $\chi^2$.

Do this using both polyfit and curve_fit, and verify that you get the same answer (at least within some reasonable numerical precision). Plot the points (with errors) and superimpose the fit.
Refer to the slides shown in the Week 6 lecture (especially pages 27 and 28. Also, the links to the jupyter-notebook examples given on page 29).

Important advice: explicitly specify the seed of the random number generator before picking the Gaussian random numbers. This is so that when you run and re-run the code you will get reproducible results. This is very useful when debugging code.
Exercise 2
The great thing about numpy arrays is that you can do fast vectorized (i.e., element-by-element) operations using the same syntax as for numerical variables. But for mathematical operations we have been doing things a little differently, e.g. np.sin(..) vs. math.sin(..). Using numpy functions also for numerical variables works, as is shown below. However, the output is of different "type", float vs. numpy.float64.

```python
zarray = np.array([4,9])
z = 25
print(np.sqrt(zarray))
print(math.sqrt(z))
print(np.sqrt(z))
print(type(math.sqrt(z)))
print(type(np.sqrt(z)))
```

```
[2. 3.]
5.0
5.0
<class 'float'>
<class 'numpy.float64'>
```

In practice these are the same, but "under tho hood" it is not entirely clear to me that they really are always interchangeable\(^1\).

There are times when we write a functions that we intend to use with inputs that are either numpy arrays or numerical values. While it is probably overkill, I tend to deal with these by checking whether the input is numpy or numeric before performing mathematical operations.

Write a function fun(x) that returns np.sqrt(x) or math.sqrt(x) depending on whether the input is numpy or just a number. There are probably several ways of doing this...use google to figure out how to check the type of x.

---

\(^1\)https://www.edureka.co/community/39993/difference-between-python-float-and-numpy-float32