BE CAREFUL WITH THE CODE, AS IT HAS THE POWER TO MODIFY THE PROGRAM OR DAMAGE YOUR COMPUTER IF USED INCORRECTLY.

Any valid Python code can be entered into the formula box. Clicking the `Run fitting` button will check the syntax of the code and then execute it.

The fitting algorithm will vary the parameters listed in the `Fitting Parameters` window. The names given to these parameters can be used and modified in the code. Parameters must be alphanumeric, cannot be duplicated, and cannot be a Python reserved word:

- and
- as
- assert
- break
- class
- continue
- def
- del
- elif
- else
- except
- finally
- False
- for
- from
- global
- if
- import
- in
- is
- lambda
- nonlocal
- None
- not
- or
- pass
- raise
- return
- True
- try
- with
- while
- yield

Code can be commented by using the # sign, and long lines of code can be broken by placing \ at the end of the line and then continuing on the next line.

Several variables and functions are predefined in the code:

- The frequencies from the input file can be accessed as an array named `freq`. For instance, to access the first frequency, use `freq[0]` in the code
- The imaginary part of the impedance is an array named `Zj`. For instance, to access the first imaginary impedance point, use `Zj[0]` in the code
- The real part of the impedance is an array named `Zr`. For instance, to access the first real impedance point, use `Zr[0]` in the code
- The imaginary number can be accessed as `1j`
- The code should set variables named `Zreal` and `Zimag` to arrays of the values for the real and imaginary parts of the impedance, respectively. Each point in the array should correspond to the frequency at that index. For instance, `Zreal[0]` should be the real part of the model impedance evaluated at `freq[0]`

- If custom weighting is used, an array name `weighting` should be created with the weights by frequency; for instance, `weighting[0]` should be the weighting at `freq[0]`. Therefore, custom weighting is the same between the real and imaginary parts.
- Python built-ins can be used and packages can be imported; `numpy` is pre-imported as `np`
- Built-in functions include:
  - PI
  - SQRT
  - ABS
  - EXP [e^x]
  - SIN, COS, and TAN
  - ARCSIN, ARCCOS, and ARCTAN
  - SINH, COSH, and TANH
  - ARCSINH, ARCCOSH, and ARCTANH
  - LN and LOG [LN is base-e, LOG is base-10]
  - RAD2DEG and DEG2RAD

    Note: Trig functions take their arguments/return results in radians

Code is automatically syntax highlighted. Supplied variables (`Zr`, `Zj`, and `freq`), fitting variables, and result variables (`Zreal`, `Zimag`, and `weighting`) are highlighted red. Included functions, the imaginary number, and Python built-in functions are highlighted purple. Strings are highlighted green, comments are highlighted gray, and Python reserved words are highlighted blue.

Example:

We will fit one Voigt element: $Z = R_e + \frac{R_1}{1+j(2\pi f)\tau_1}$

1. First load a *.mmfile* under `Browse…`
2. Next, click `Fitting Parameters`, then click `Add Parameter` three times (one for each variable we will be fitting)
3. Rename `var0` to `Re`, `var1` to `R1`, and `var2` to `T1`, and type their initial guesses into the boxes to the right
4. In the dropdown to the right of `T1`, select `+` to constrain the time constant to be positive

5. Set the desired fit type, number of Monte Carlo simulations, and weighting strategy (see the Measurement Model Guide link in the "Help and About" tab)
6. Enter the following into the `Code` window (the code should syntax-highlight as seen below):

```
Z = Re + R1/(1+1j*2*PI*freq*T1)
Zreal = Z.real
Zimag = Z.imag
```

7. Click `Run fitting` and wait for results to appear