By the way, because you are planning to distribute this code afterwards, I highly recommend you follow a standard Python style guide. It will increase readability and decrease profanity hurled your direction behind your back.

Here is a good article: [http://docs.python-guide.org/en/latest/writing/style/](https://www.mail.ubc.ca/owa/redir.aspx?C=sxRO9jpHVcpXYGp8i7qF3yCK9IM18f4z7yJlSpOIBYGpSuKdxs3VCA..&URL=http%3a%2f%2fdocs.python-guide.org%2fen%2flatest%2fwriting%2fstyle%2f)

Here is the official in-depth style guide for your reference: [https://www.python.org/dev/peps/pep-0008/#a-foolish-consistency-is-the-hobgoblin-of-little-minds](https://www.mail.ubc.ca/owa/redir.aspx?C=rtLxrRuGNz_z-PnKnue_HThUkmf1oNC2gZEECJu2rvupSuKdxs3VCA..&URL=https%3a%2f%2fwww.python.org%2fdev%2fpeps%2fpep-0008%2f%23a-foolish-consistency-is-the-hobgoblin-of-little-minds)

Also, I would recommend that complicated chunks of code (esp. with only a few inputs and outputs) be moved into a function. That way, your script will show high-level function calls which make it easy to see at a glance what the code is doing. For example, you high-level script could look like (totally made up code):

grain\_list = []

for \_ in range(n\_grains):

    x, y = calculate\_coordinates(map)

    local\_grain\_size = calculate\_local\_gsd(x, y, map)

    grain\_list.append(Grain(x, y, local\_grain\_size))

plot\_grains(grain\_list)

It is relatively easy to see that the code will calculate new coordinates, calculate the local grain size distributions, add new Grain objects to the grain\_list, and then plot all the Grain objects. A large majority of the time, the reader (including you) won't need to know the details of how each thing was calculated, but will want to know the overall flow of the code. The function definitions can be elsewhere in the file or even in another module. This will also reduce workspace variable clutter and chance for accidentally overwriting variables. As mentioned before, the variables inside the function are local-scope. You can use the same variable name in multiple function definitions.

def fu(data):

    # Extract data points with even indices

    return data[::2]

def bar(data):

    # Print out the data

    print(f"Data = {data}")

def baz(x=20):

    # Generate a new data set

    data, n = generate\_data(x)

    return data

The "data" variable in each of these functions have nothing to with the other ones as they are in different scopes . This allows shorter variable names and clearer code within the function because you don't have to make them unique names. Just be careful about your global variables. They can still be accessed (and overwritten!) inside a function.

Probably too much info... But I felt inspired!

Cheers,

Alex