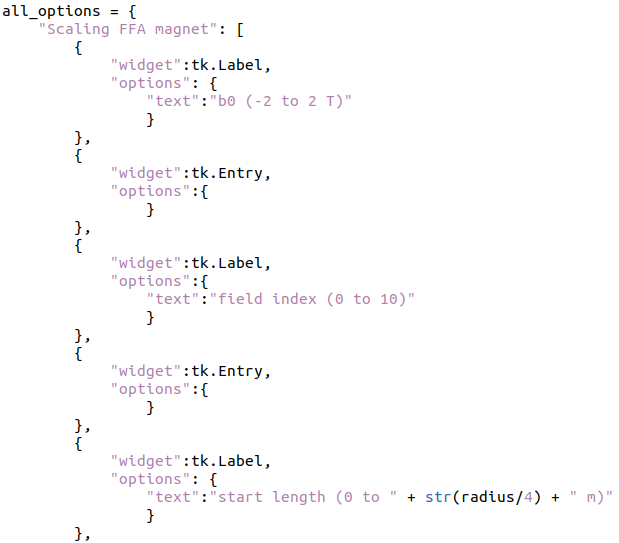
**Instructions on how to add ring elements to pyOpal GUI**

The way in which elements are added depends on how many options windows you need. For example, multipoles require the user to choose how many poles they want. A second window then opens asking them to choose the field value of each of these poles. The second set of attributes (field values) depends on the first and thus must be chosen in sequence.

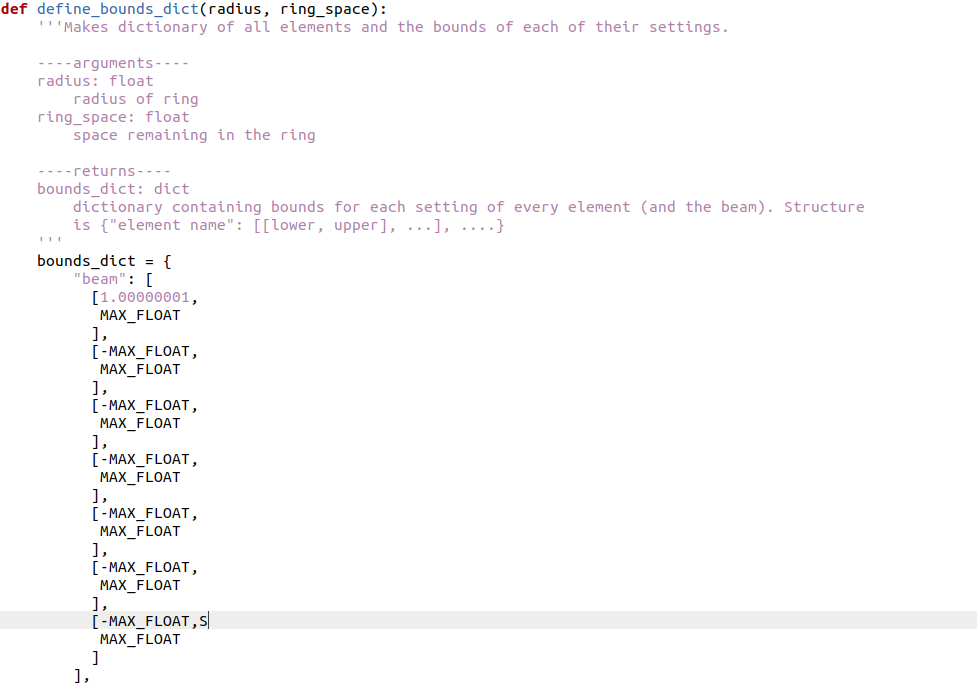
**If only 1 options window is needed:**

1. Make sure you know what your element is called in OPAL, what attributes you want the user to be able to choose, the default value of all other attributes, and which ones you want displaying in the element list.
2. Figure out all the widgets you will need to be able to get inputs for the attributes the user chooses. This should include a label widget and appropriate input widgets for each. Floats require entry widgets, and integers require sliders. You should also decide what bounds you want when validating each input.
3. Add your list of widgets to ALL\_OPTIONS in GUI\_dicts.py. This will have the following layout:

“element name”: [ {“widget” : tkinter widget class, “options” : dictionary args}, ... ]



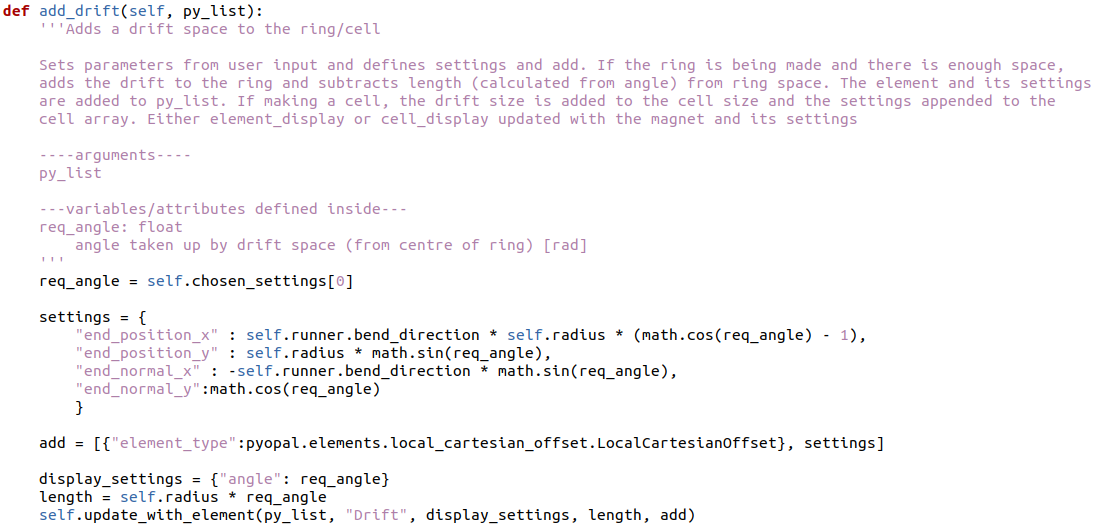
1. Add the bounds for each attribute to BOUNDS\_DICT. The structure is the element name as the key, with the value being a list of bounds lists. Each bounds list is of the form [lower bound, upper bound]. The bounds lists should be in the order of how the input widgets are displayed on screen.



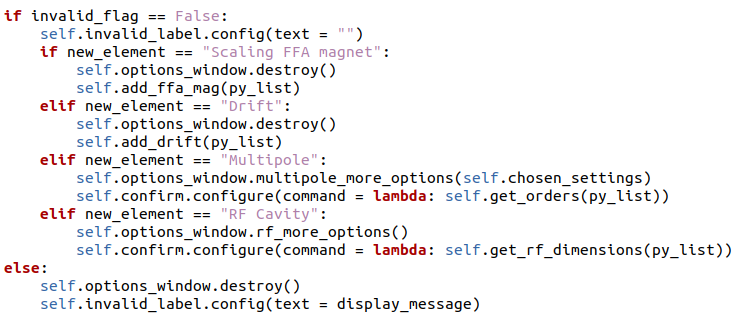
1. Make a Gui class method that first gets all the attributes from chosen\_settings (attributes are in this list in order how their input widgets are displayed). This should be named add\_ “element name”, and take py\_list as an argument.
2. Define the length variable as the length of your element
3. In your method, the attributes should then be put into a settings dictionary, structured as {“arg name” : arg, ...}. The dictionary called “add” should then be defined, with the structure [{“element\_type” : OPAL class name}, settings]
4. Choose which attributes you want displaying on screen in the element list. Make these into the display\_settings dictionary. This has the same structure as settings.
5. The last thing your method should do is call update\_display\_with\_element. This takes py\_list, the element name, display\_settings, length, and add as arguments.

The resulting method should be structured:

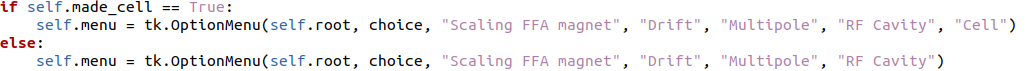
1. Get attributes from chosen\_settings
2. Define settings
3. Define add
4. Define display\_settings
5. Define length
6. Call update\_display\_with\_element



1. Add your method to the if statement in the get\_choices Gui method.



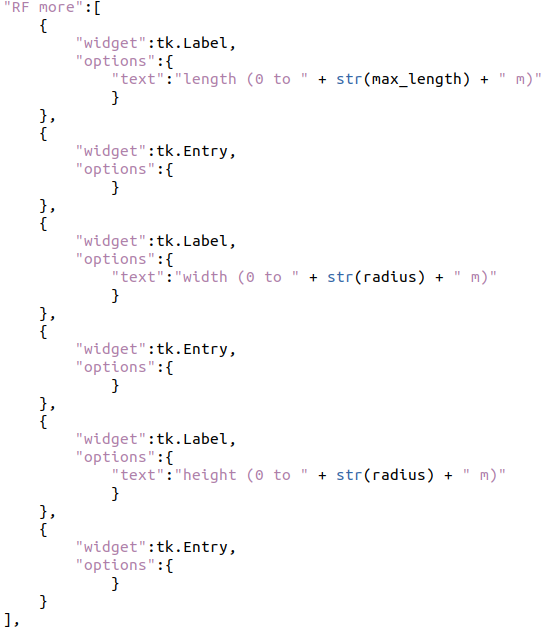
1. Add the element name string to the drop-down lists used for building the cell and the ring. These are defined in make\_cell and design\_ring



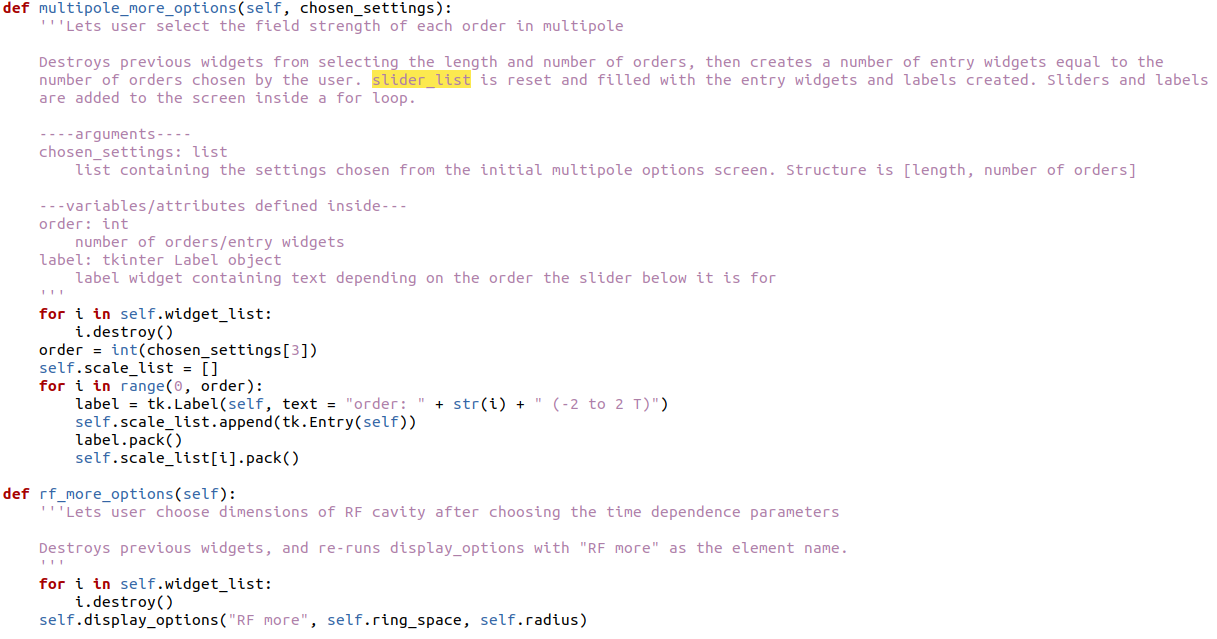
**If more than 1 options window is needed:**

This is similar to the above instructions, except that two methods will need to be defined in the Gui class, and there will be 2 additions to ALL\_OPTIONS and BOUNDS\_DICT.

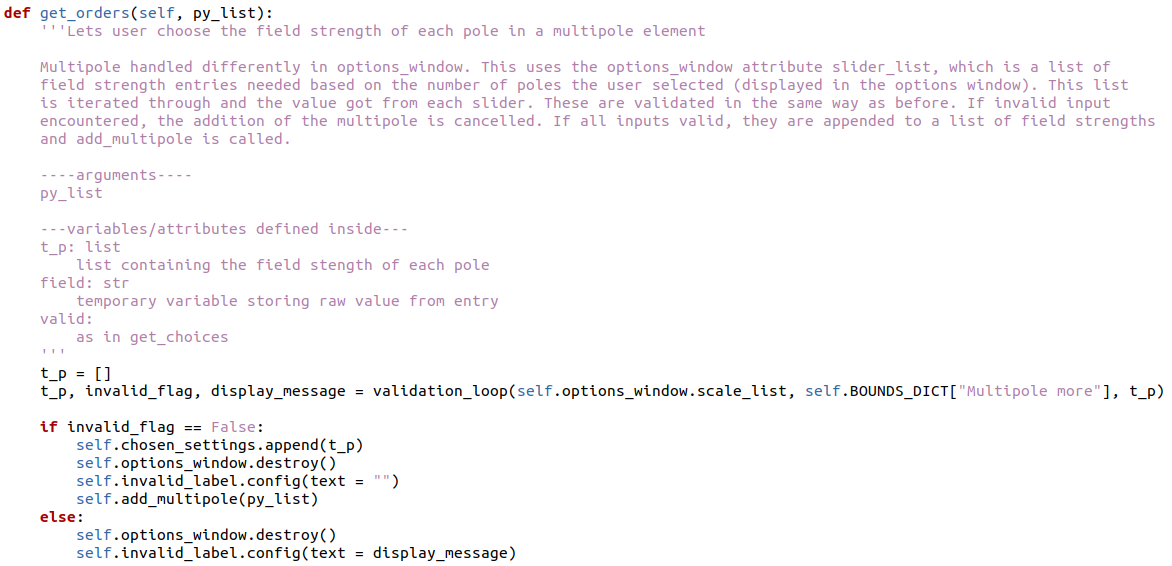
1. Decide which options need to be chosen first based on which attributes depend on other attributes / choices. These should be chosen second.
2. Add the widgets for the first window to ALL\_OPTIONS in the same way as above.
3. Add the bounds for the first set of choices to BOUNDS\_DICT in the same way as above.
4. Add the widgets for the second window to ALL\_OPTIONS, with the only difference being that the key is “[element name] more”.



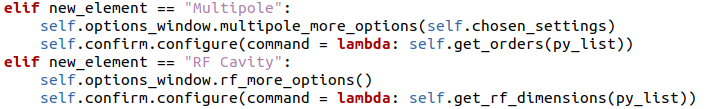
1. Add the bounds for the second window to BOUNDS\_LIST with the same naming convention as in the previous step.
2. Define an Options\_Window method that displays the second set of widgets or call the display\_options method with “[element name] more” as the choice argument.



1. Define a Gui or Options\_Window class method that gets the choices from the inputs in the second window (first are simply “got” from the get\_choices Gui method as before and require no attention). This should handle any dependencies these choices have on the last ones.



1. Define a Gui class method that gets the second set of choices and defines settings, args etc in the same way as before.
2. Add your element to the if statement in get\_choices as before, but instead of calling the “add\_[element name]” method, call the one that displays the next set of choices. Configure the confirm button so it calls “add\_[element name]”.



1. Add your element to the drop-down menus as before.