**User-guide for Running CAMDT Python Script**

-Prepared by Eunjin Han[[1]](#footnote-1) at IRI, Columbia University –

(August, 2016)

**\*What is Python?**

|  |
| --- |
| * Python is a programming language which is ….. * Simple and easy to learn * Free and open source * Cross platform – Mac, Windows, Linux…. * Interpreted – no need to compile to binary code * **Object oriented**   **\* In Python, everything is an object**   * A unique ID, or location in the computer’s memory * A set of **properties (**or **attributes)** that describe the object * A set of **methods**, or things that the object can do |

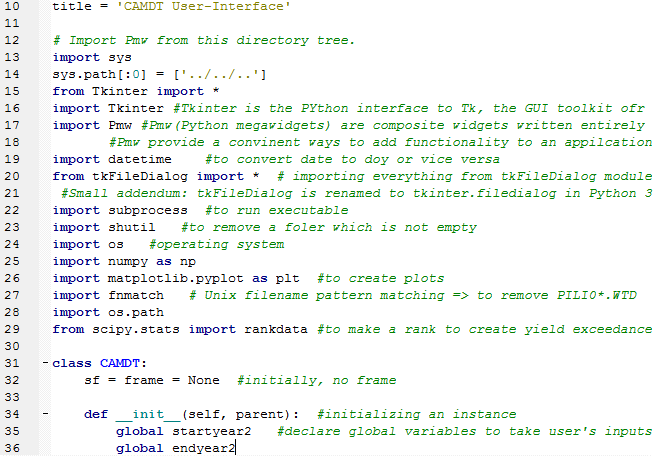
**\*What is Tkinter?**

|  |
| --- |
| * Tkinter is the Python interface to Tk, the GUI toolkit for Tcl/Tk   (source: Grayson, John E. "Python Tkinter Programming." (2011). |

**\*What are Classes in Python?**

|  |
| --- |
| ▪ A Python class is a user-defined data type  ▪ A class provides the following object descriptions:  ◦ The attributes (data-members) of the object  ◦ The behavior of the object (methods)  ◦ Where behavior is inherited from other classes (superclasses)  ▪ Calling the class as a function creates an ***instance*** of the class  ▪ Initializing an instance: Fields (instance variables) of an instance may be initialized by including an \_\_init\_\_ method in the class body. This method is executed automatically when a new instance of the class is created. Python passes the instance as the first argument. It is a convention to name it *self*.  (source: Grayson, John E. "Python Tkinter Programming." (2011). |

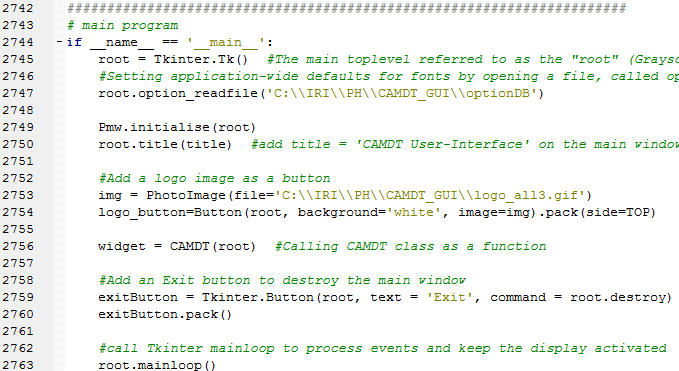
**1. Import all required modules**



Define a class called CAMDT

Import required modules

Initializing an instance

**2. Calling main program/module**

7) Add an “Exit” button to destroy the main window

1) \_\_name\_\_

2) “root” toplevel

3) Change default fonts

4) Add a title on the main window

The constructor initializes Pmw

5) Add a logo image

6) Call “CAMDT” class

8) Call mainloop to process events

1) A module’s \_\_name\_\_

|  |
| --- |
| Every module has a name and statements in a module can find out the name of its module. When a module is imported for the first time, the main block in that module is run. What if we want to run the block only if the program was used by itself and not when it was imported from another module? This can be achieved using the \_\_name\_\_ attribute of the module.  Every Python module has it's \_\_name\_\_ defined and if this is '\_\_main\_\_', it implies that the module is being run standalone by the user and we can do corresponding appropriate actions.  (source: http://ibiblio.org/g2swap/byteofpython/read/module-name.html) |

2) Toplevel

|  |
| --- |
| The *Toplevel* widget provides a separate container for other widgets, such as a Frame. For simple, single-window applications, the “root” Toplevel created when you initialize Tk may be the only shell that you need.  (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.32). |

3) Change default fonts

|  |
| --- |
| Setting application-wide defaults for fonts or colors.  The purpose of this line is to set the font for all widgets except labels to Verdana 9 as indicated in a file called *optionDB.* We can apply the values using an *option\_readfile* call. |

4) Add a title on the main window



5) Add a logo image



6) Call “CAMDT” class

→ The “CAMDT” class has almost all components for the CAMDT GUI including creating pages, tabs and combox etc. to get user’s input.

7) Add an “Exit” button to destroy the main window

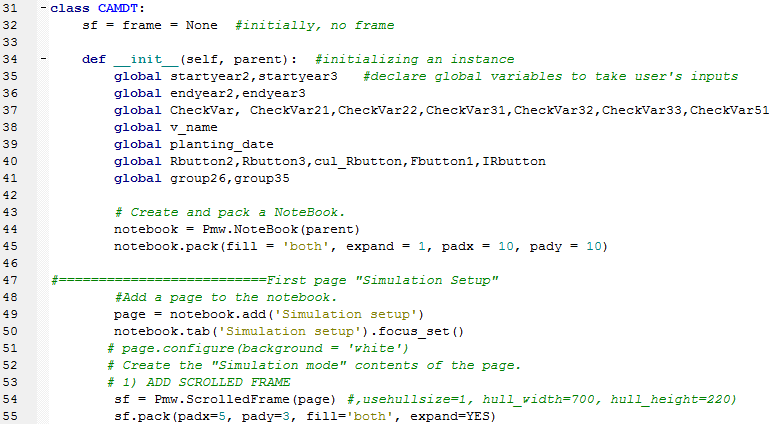


8) Call mainloop to process events

|  |
| --- |
| Call the Tkinter *mainloop* to process events\* and keep the display activated.  (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.10).  **\*What are events?**  -Events are notifications (messages in Windows parlance) sent by the windowing system to the client code. They indicate that something has occurred or that the state of some controlled objet has changed, either because of user input or because your code has made a request which causes the server to make a change. (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.96). |

**2. CAMDT class**

**2-1) First Page**



2) Add a ScrolledFrame

Add a page to the notebook using “add” method

1) Packer

Use Pmw NoteBook widget which implements the popular property sheet motif. Methods allow a number of pages or panes to be created

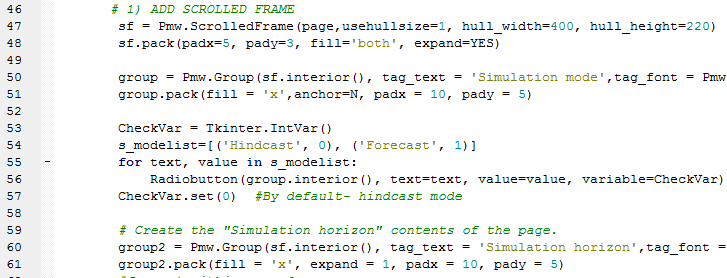
Declare global variables to take user’s input and write them into a param.txt file

1) Packer

|  |
| --- |
| The Packer positions slave widgets in the master by adding them one at a time from the outside edges to the center of the window. The Packer is used to manage rows, columns and combinations of the two.  Examples:  -Side=LEFT tells the Packer to start locating the widgets in the packing list from the left-hand side of the container  -Expand=YES tells the Packer to fill the available space within its parcel; whether it does expand is controlled by the “fill” option.  -Using fill=BOTH allows the widget to use all of its parcel.  -The padx and pady options allow the widget to be packed with additional space around it.  (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.79). |

2) ScrolledFrame

|  |
| --- |
| The ScrolledFrame widget is to provide a Frame widget with associated horizontal and vertical scrollbars.  The scrollbars can be dynamic, which means that a scrollbar will only be displayed if it is necessary – if the frame is smaller than the surrounding clipping frame, the scrollbar will be hidden.  (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.79). |



4) For loop to create multiple Radiobuttons

3) List of tuples

2) Tkinter variables

1) Create a group within the Scrolled Frame (named as “sf”) with a specific title (‘Simulation mode”) and font.

1) Group

|  |
| --- |
| The Group widget provides a convenient way to place a labeled frame around a group of widgets.  (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.57). |

2) Tkinter variables

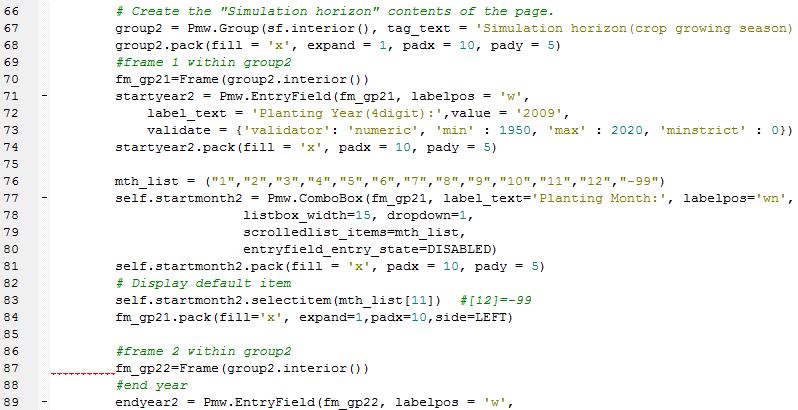
|  |
| --- |
| There are several ways for Pmw widgets to provide access to the widget’s content. Tk provides the ability to link the current value of many widgets (such as text) to an application variable. Tkinter does not support this mode, instead it provides a *Variable* class which may be subclassed to give access to the *variable, textvariable, value* and other options within the widget. Currently, Tkinter supports *StringVar, IntVar, DoubleVar* and *BoolenVar*. These objects define *get* and *set* methods to access the widget.  (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.152).  CheckVar.set(0) → CheckVar is set with 0 (hindcast mode) by default |

3) List of tuples to set integer variables for each Radiobutton

|  |
| --- |
| Two sequence classes – lists and tuples- are unique data types in Python. Lists and tuples have a lot in common. The major difference is that the elements of a list can be modified in place but a tuple is immutable: you have to deconstruct and then reconstruct a tuple to change individual elements.  Lst = [1,2,3,4] # Integer list  Lst=[[1,2,3],[ ‘a’, ‘b’, ‘c’]] #List of lists  Lst=[(1, ‘a’),(2, ‘b’),(3, ‘c’)] #List of tuples  (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.5).  s\_modelist=[('Hindcast', 0), ('Forecast', 1)] → s\_modelist was created to assign 0 and 1 to Hindcast and Forecast Radiobutton respectively. |

4) For loop to create multiple Radiobuttons

|  |
| --- |
| The *Radiobutton* widget is for exclusive selection: selecting one button deselects any button already selected. (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.37).  *Text* and *values* are from the s\_modelist=[('Hindcast', 0), ('Forecast', 1)]  For each Radiobutton, IntVar is assigned. |



1) Create a second group for “simulation horizon”

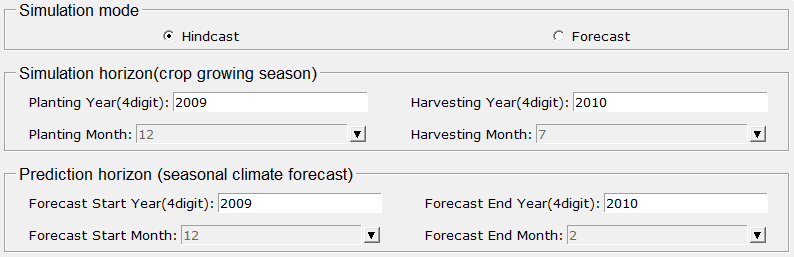
3) EntryField

2) Create a frame for start year and month

4) ComboBox

1) Create a second group for “simulation horizon”

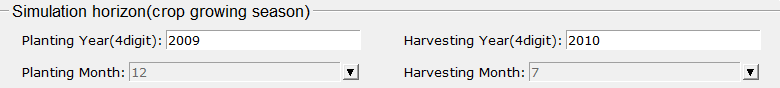
Group 1



Group 3

Group 2

2) Create a frame for start year and month



Frame 2

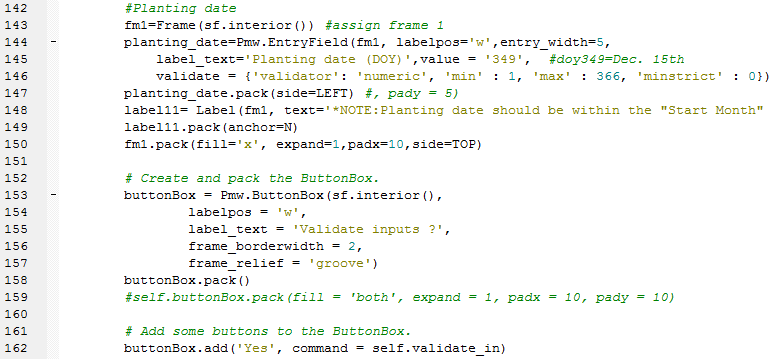
Frame 1

3) EntryField

|  |
| --- |
| The *EntryField* widget is an *Entry* widget with associated validation methods. The built-in validation provides validators for integer, hexadecimal, alphabetic, alphanumeric, real, time and date data formats. Some of the controls that may be placed on the validation include checking conformity with the selected data format and checking that entered data is between minimum and maximum limits. (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.56).  startyear2 = Pmw.EntryField(fm\_gp21, labelpos = 'w',  label\_text = 'Start Year(4digit):',  validate = {'validator': 'numeric', 'min' : 1950, 'max' : 2020, 'minstrict' : 0})  - fm\_gp21→ create the EntryField within the frame, fm\_gp21  - labelpos = 'w' → label position is “west’  - label\_text = 'Start Year(4digit):' → label has a text, 'Start Year(4digit):'  - validate = {'validator': 'numeric', 'min' : 1950, 'max' : 2020, 'minstrict' : 0}) → entered data type should be ‘numeric’ in a range of min = 1950 and max=2020.  \*Note: Due to this validator, the EntryField has pink color initially. If a user give correct input (i.e., numeric value greater than 1950 and less than 2020, the pink color will be gone. However, if the user give incorrect input, it will not allow the incorrect input to be written. |

4) ComboBox

|  |
| --- |
| The *ComboBox* widget allows the user to select from a list of options. The list may be displayed permanently or as a dropdown list. Using the dropdown form results in GUIs which require much less space to implement complex interfaces. (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.52).    mth\_list = ("1","2","3","4","5","6","7","8","9","10","11","12","-99")  self.startmonth2 = Pmw.ComboBox(fm\_gp21, label\_text='Planting Month:', labelpos='wn',  listbox\_width=15, dropdown=1,  scrolledlist\_items=mth\_list,  entryfield\_entry\_state=DISABLED)  self.startmonth2.pack(fill = 'x', padx = 10, pady = 5)  self.startmonth2.selectitem(mth\_list[11]) #[12]=-99 # Display default item  - mth\_list → list to contain values for dropdown list  - fm\_gp21→ create the ComboBox within the frame, fm\_gp21  - dropdown=1 → dropdown is activated (true)  - scrolledlist\_items=mth\_list → items for the ComboBox are from the list mth\_list  - entryfield\_entry\_state=DISABLED →  to disable the combobox' entry (i.e. user is not allowed to enter any input on the entryfield of the ComboBox  - self.startmonth2.selectitem → shows a default item before a user’s selection. |



2) buttonBox

1) Label

1) Label

|  |
| --- |
| Label widgets are used to display text or images.  (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.35). |

Label

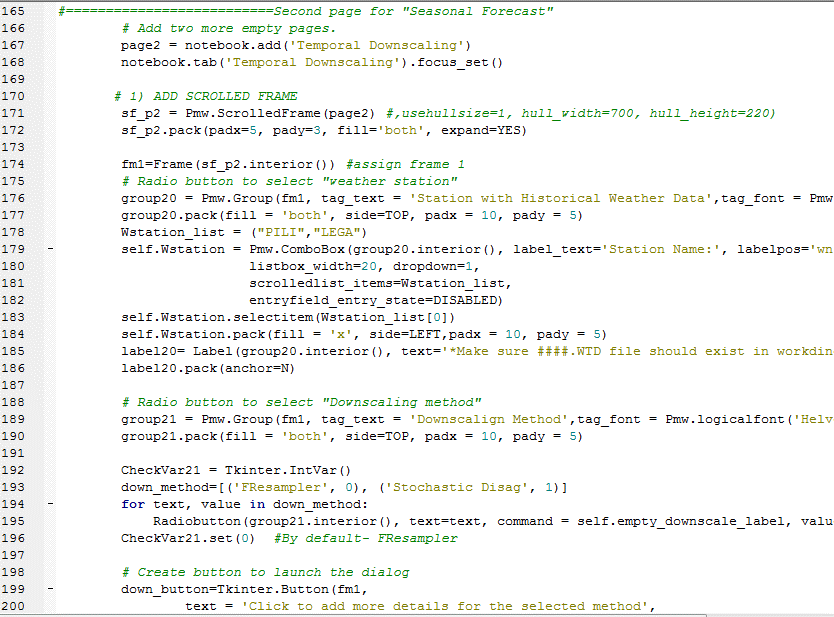


ButtonBox

2) buttonBox

|  |
| --- |
| The *ButtonBox* widget is to implement a number of buttons and it is usually used to provide a *command area* within an application. (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.51).  buttonBox.add('Yes', command = self.validate\_in)  - Add a button with a label “Yes”. If a user click this button, it will activate a command “self.validate\_in” and start to run the function “validate\_in”.    => The function, “validate\_in” check all user’s inputs including starting year, month, ending year and month etc. and then generates an error message if the input does not make sense (e.g., starting year is greater than ending year). |

**2-2) Second Page**



2) Button

Add a Radiobutton to select downscaling method (either FResampler or Stochastic Disag.)

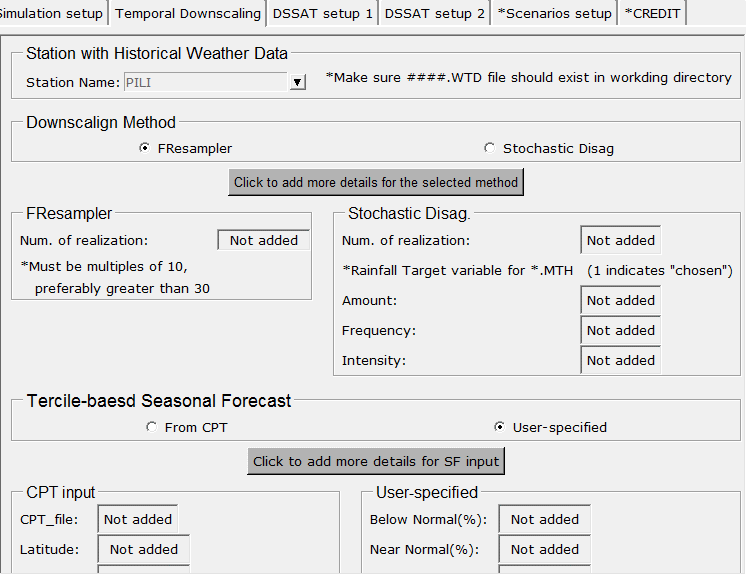
1) Frame

Add a ScrolledFrame

Add the second page to set up temporal downscaling method

1) Frame

|  |
| --- |
| Frame widgets are containers for other widgets.  (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.33). |



Frame 1

Frame 2

Group26

Group25

Group24

Button

Group23

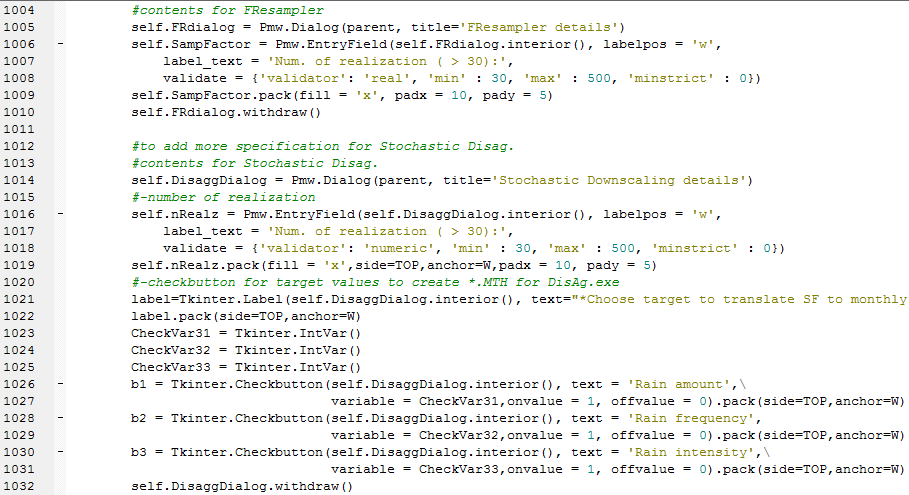
Group22

Group21

Group20

2) Button

|  |  |  |
| --- | --- | --- |
| Strictly, *buttons* are *labels* that react to mouse and keyboard events. You bind a method call or callback that is invoked when the button is activated. (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.36).  # Create button to launch the dialog  down\_button=Tkinter.Button(fm1,  text = 'Click to add more details for the selected method',  command = self.getmoreinput,bg='gray70',font = ("weight bold",10)).pack(side=TOP,anchor=N) →Add a button to activate a new window which is to get more detailed information about the selected downscaling method. If a user click this button, it will activate a command “self.getmoreinput” and start to run the function “getmoreinput”.    Get user’s inputs from the new dialog (a) and save them on labels in the group 22 above. If the new inputs are added, the color of the labels turns to ‘honeydew’ color from originally grey (see (c)).  If Stochastic Disag. is chosen, activate a new dialog (b)  If FResampler is chosen, activate a new dialog (a)   |  |  | | --- | --- | | (a) | (b) |   **(c)** |



2) Checkbutton

Add “EntryField” and Checkbutton on the new dialog (b) to get more details on the Stochastic Disag.

1) Dialog

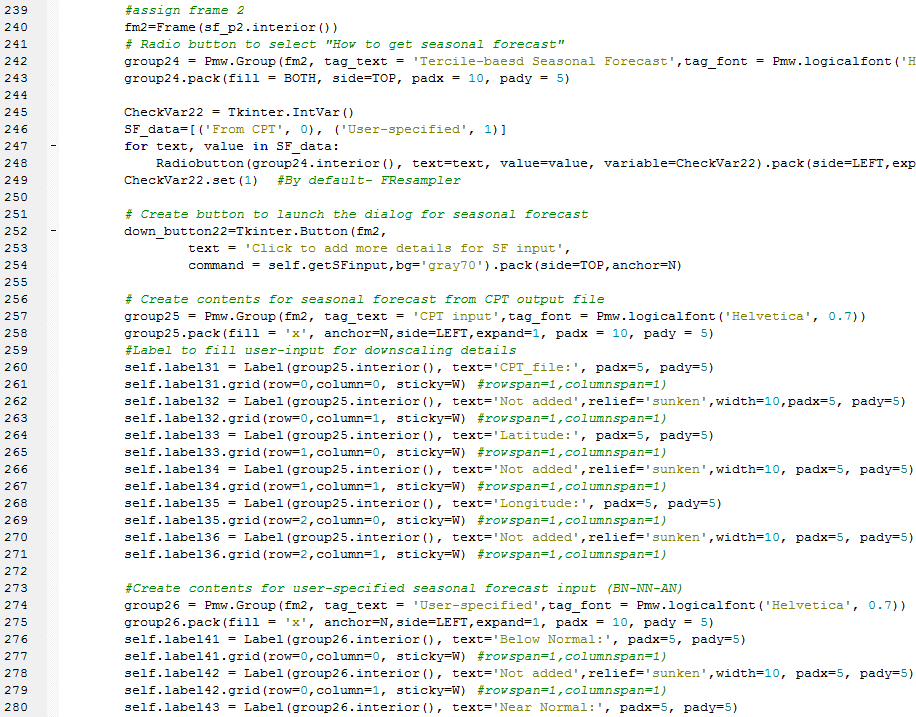
Add “EntryField”s on the new dialog (a) to get more details on the FResampler

1) Dialog

|  |
| --- |
| The *Dialog* widget provides a simple way to create a *toplevel* containing a *ButtonBox* and a child site area. |

2) Checkbutton

|  |
| --- |
| Checkbutton widgets are used to provide on/off selections for one or more items. Unlike radiobuttons there is no interaction between checkbuttons. (Source: Grayson, John E. "Python Tkinter Programming." 2011 p.38).    Checkbutton -> User can select one or multiple items |

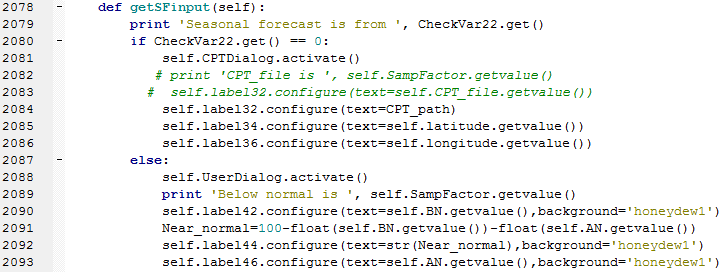


1) Clicking this button will generate new pop-up windows.

Add Button to get more details about the selected seasonal forecast input.

Add Radiobuttons to select options for how to get seasonal forecast (either CPT output file or user-specified)

1) Button for seasonal forecast input sources



1-b) UserDialog

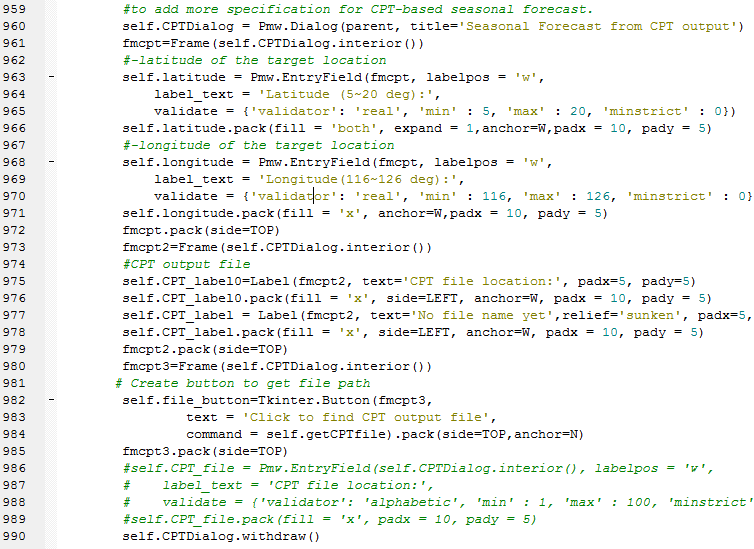
1-a) CPTDialog

If ‘User-specified” is chosen, activate a new dialog (b)

If ‘from CPT” is chosen, activate a new dialog (a)

|  |  |
| --- | --- |
| (a) | (b) |

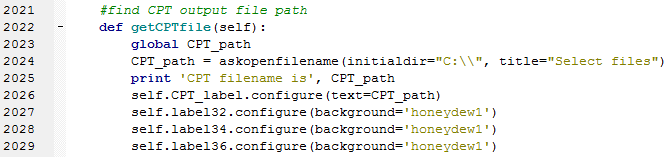
1-a) CPTDialog



If user click the button “Click to find CPT output file”, it will activate “getCPTfile” function to open a pop-up window to search for a target file.

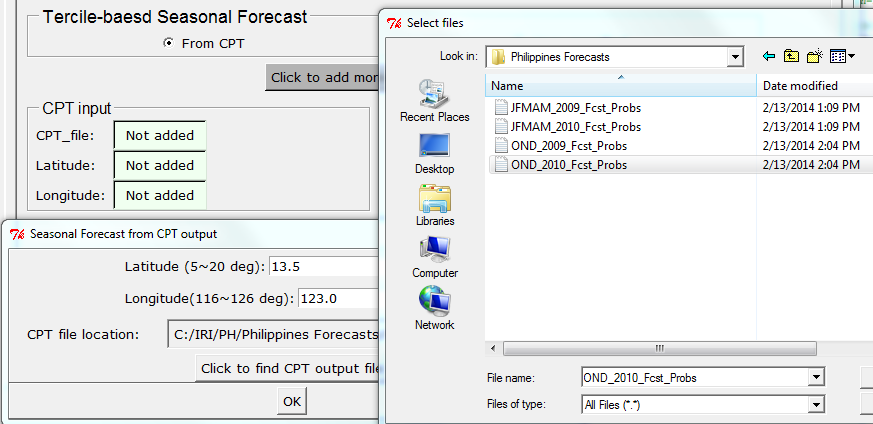
\* Use “askopenfilename” to get a file name to open. Initial searching directory is set as “C:\\”

In order to use this, tkFileDialog module should be imported.



\*askopenfilename

|  |
| --- |
| tkFileDialog  If you want to open or save a file or to choose a directory using a filedialog you dont need to implement it on your own. The module ***tkFileDialog*** is just for you.  (source: http://tkinter.unpythonic.net/wiki/tkFileDialog) |

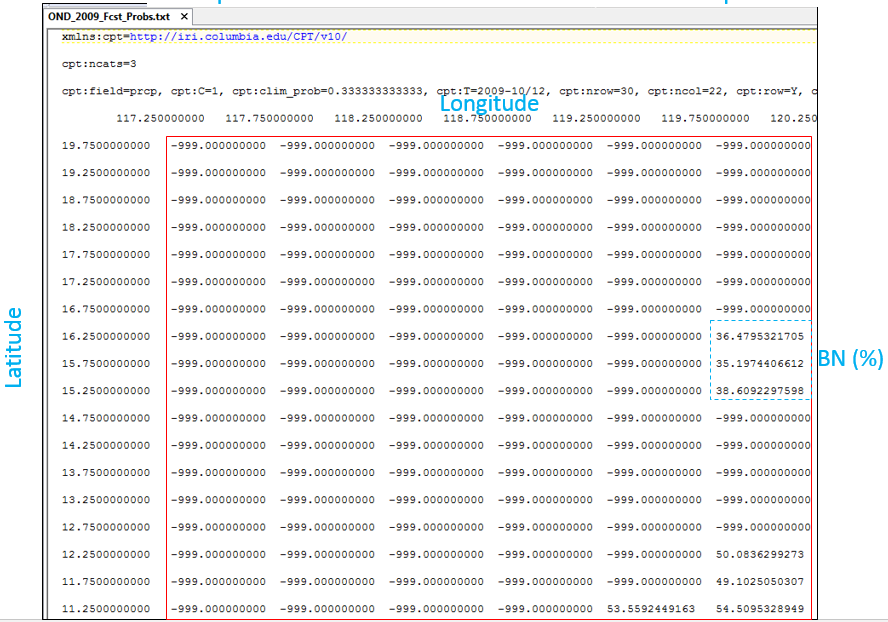


GetCPTfile function

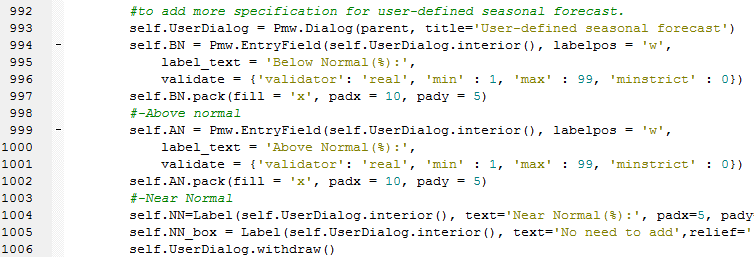
-> askopenfilename

CPTDialog

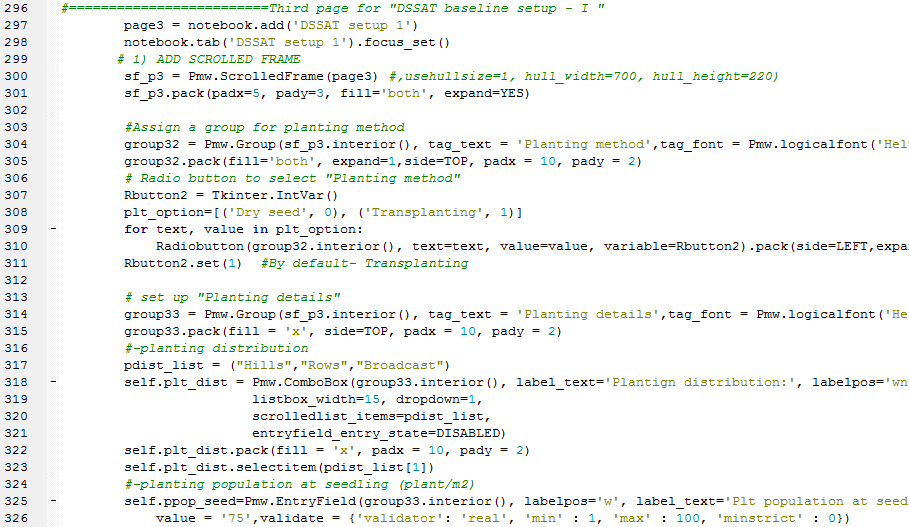
\*Example of seasonal Climate Forecast from CPT output → Based on the given latitude and longitude, the seasonal forecast (BN-NN-AN) will be determined from the CPT output file as shown below.



1-b) UserDialog



**2-3) Third Page to set up DSSAT inputs**

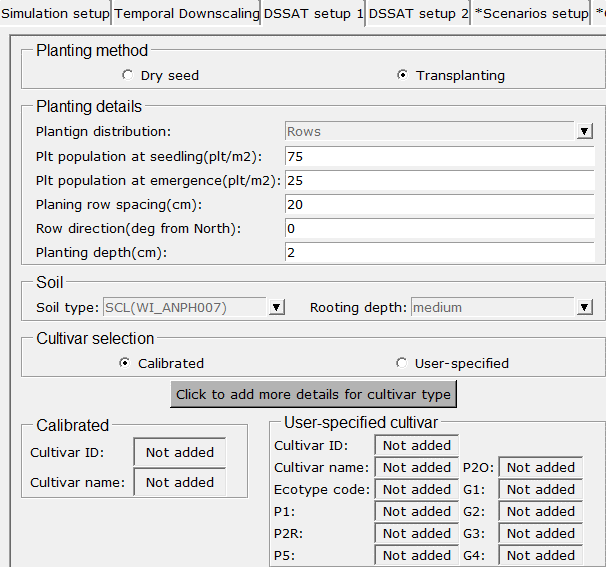


Add Radiobuttons to select planting options

Add a ScrolledFrame

Add a new page to set up DSSAT inputs

**\*\*Widgets used to create the rest of pages are similar to the previous ones. Therefore, all details for the pages 3 and 4 are skipped.**



Group35

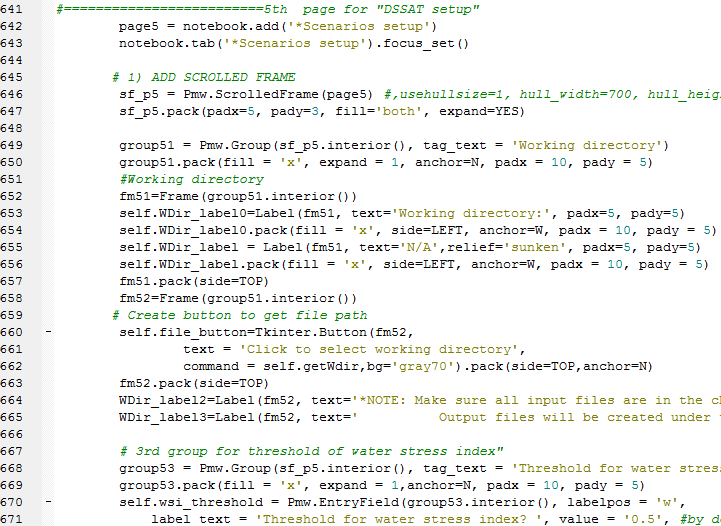
Frame 33

Frame 34

Group33

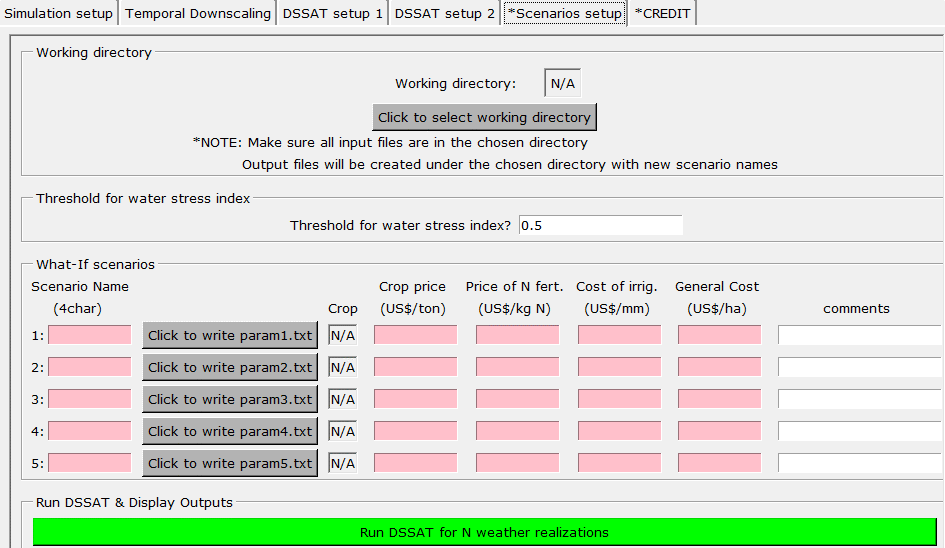
Group32

**2-5) Fifth Page to set up What-If scenarios**



Add a new page to set up what-if scenarios

Add a ScrolledFrame



Frame 58

Frame 56

Frame 55

Frame 60

Frame 59

Frame 57

Frame 54

Frame 53

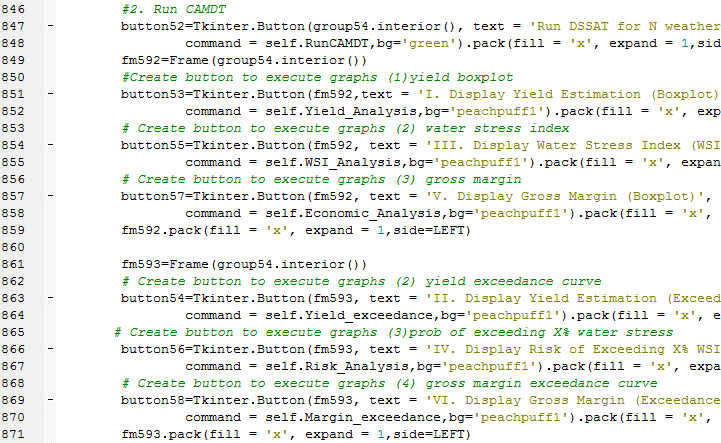
Group52

Group53

Frame 52

Frame 51

Group51



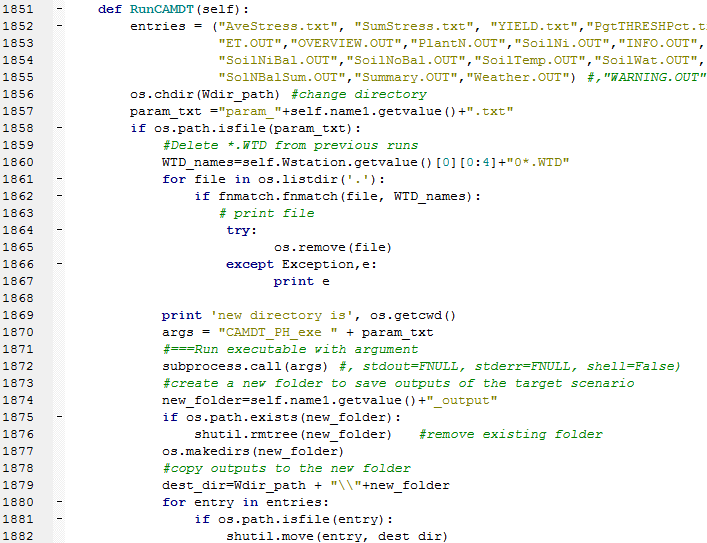
Run “Economic analysis”

Run “WSI\_Analysis”

If a user clicks the button “Run DSSAT for N realizations”, it will run “RunCAMDT” function (see (a) below).

Run “Yield\_Analysis” function (see (b) below).

**(a) RunCAMDT**



Output files (specified in the “entries” list above) after running DSSAT will be moved to a newly created folder for the specific scenario.

1) Run executable (CAMDT\_PH\_exe.exe) with an argument

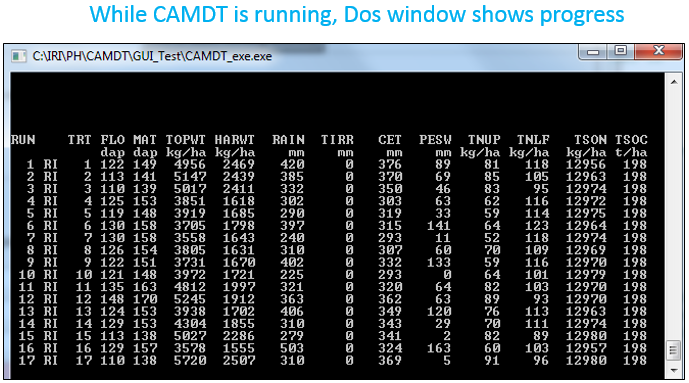
If param\*.txt is created successfully, then delete temporary \*.WTD files created in the previous runs

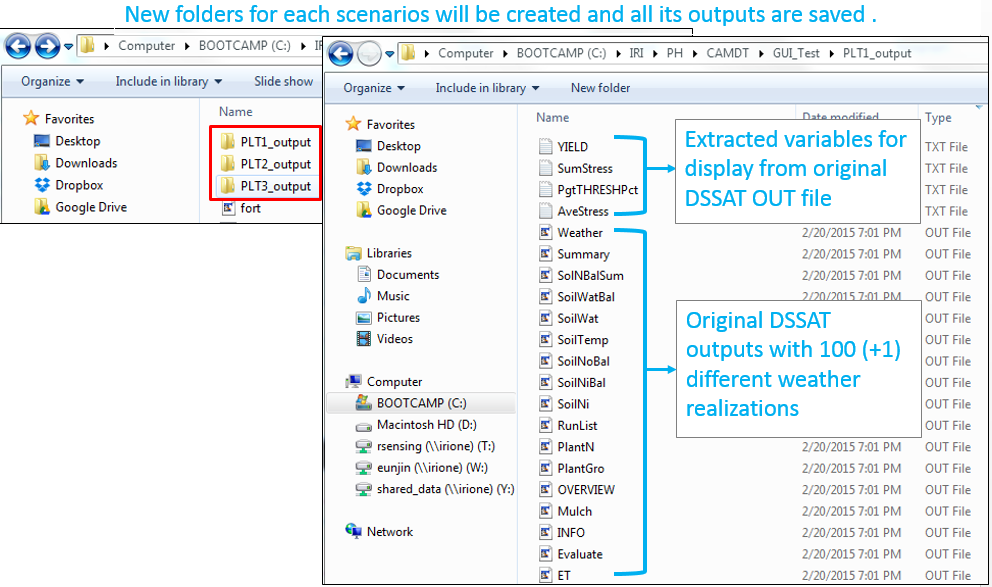
List of output file names which will be moved to a new directory

\*Note: “print” statement is only for debugging purpose (i.e. not related to any modeling processes at all)

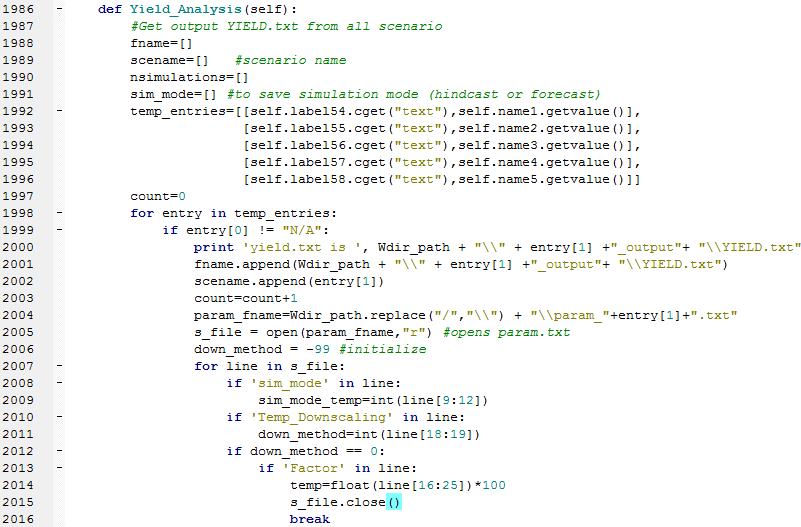
1. Run executable using subporcess

|  |
| --- |
| The subprocess module allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes.  The recommended way to launch subprocesses is to use the following convenience functions.  *subprocess.call(args, \*, stdin=None, stdout=None, stderr=None, shell=False)*  Run the command described by args. Wait for command to complete, then return the returncode attribute. (source: <https://docs.python.org/2/library/subprocess.html>)  → This subprocess will run “CAMDT\_PH\_exe.exe” with an argument “param\*.txt”. This is same as running the executable on MS-Dos window by typing “CAMDT\_PH\_exe param\*.txt”  The executable, “CAMDT\_PH\_exe.exe” was created from Fortran compiler. The Fortran code developed by Eunjin will run temporal downscaling algorithm and then run DSSAT using the user-provided input written in param\*.txt (simulation period, management practices, soil, irrigation etc. ) |





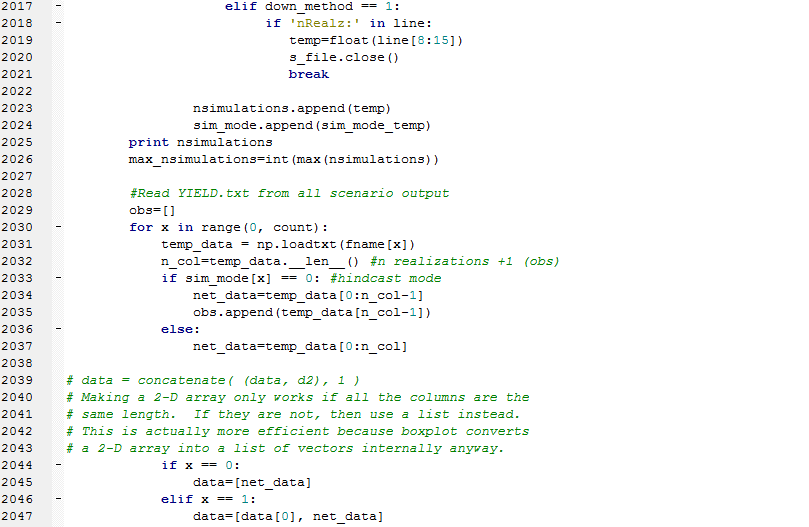
**(b) Run “Yield\_Analysis”**



Find corresponding param\*.txt file and read simulation mode (hindcast or forecast) to figure out total number of simulations (note: each scenario can have different no of realizations)

Count number of scenarios (by counting non-“N/A” labels).

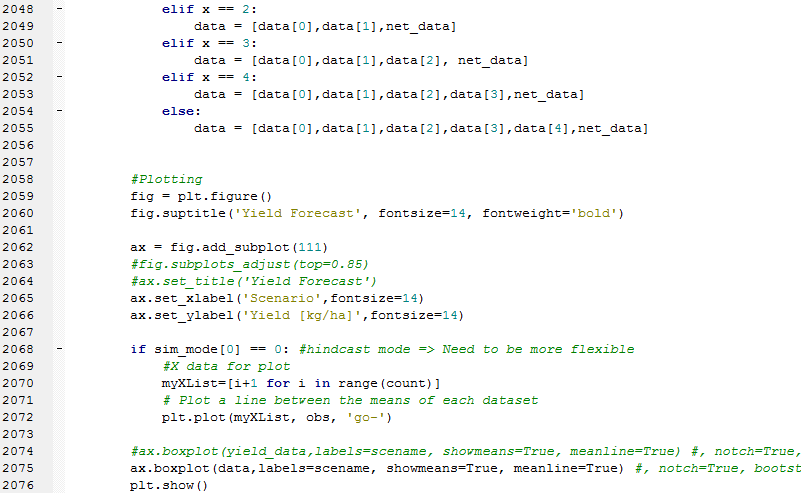
Get contents of labels in Frame 53 and 55



Concatenate each array from yield.txt to make one complete matrix for boxplot.

1) Use “loadtxt” from NumPy to read “YIELD.txt”

Last value of the output is from the observed weather data (in this case, hindcast mode)



2) Create a line plot using “obs” data (‘go-‘ is properties of the plot-> ‘g’ means green color and ‘o’ means marker etc.

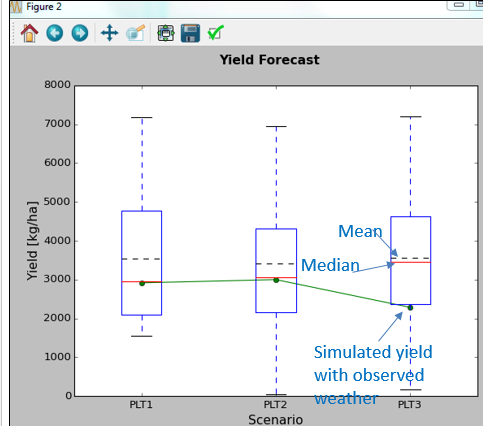
Create a boxplot using data saved in “yield\_data”. Show mean and meanline

1. Numpy

|  |
| --- |
| NumPy is the fundamental package for scientific computing with Python. It contains among other things:  - a powerful N-dimensional array object  - sophisticated (broadcasting) functions  - tools for integrating C/C++ and Fortran code  - useful linear algebra, Fourier transform, and random number capabilities  Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.  (source: http://www.numpy.org/) |

|  |  |
| --- | --- |
| Original “YIELD.txt” output | After loading using “loadtxt” |
|  | → Original data from YIELD.txt is loaded as one long row as shown above. Therefore, the loaded data are vertically stacked (np.vstack) to create a matrix which will be an input to create multiple boxplots. |

\*Example output (three scenarios with different planting dates)



2) Pyplot/plot

|  |
| --- |
| matplotlib.pyplot is a collection of command style functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., create a figure, create a plotting area in a figure, plot some lines in a plotting area, decorate the plot with labels, etc.  **plot()** is a versatile command, and will take an arbitrary number of arguments. For example, to plot x versus y, you can issue the commanplt.  *plot([1,2,3,4], [1,4,9,16])d:*  (source: <http://matplotlib.org/users/pyplot_tutorial.html>) |

**\*\* Script for “Yield\_exceedance “, “WSI\_Analysis” and “Risk\_Analsys”, “Economic\_Analysis” and “Margin\_exceedance” functions are similar to “Yield\_Analysis”. Therefore detailed explanations are skipped.**

1. Contact info: [eunjin@iri.columbia.edu](mailto:eunjin@iri.columbia.edu) [↑](#footnote-ref-1)