Python in the VE

VE Scripts – using a GUI with VE scripts

What? We build a dialog with user entry boxes, a box, buttons, code that creates a room group schema, code that finds APS files and code that tests for overheating & that exports the results to a spreadsheet. We use an event loop that drives the operation of the dialog

Why? In this example we use a dialog for data / choice entry for users of a script. We use the Tkinter module to create the dialog and show how the structure of the script is different as have to use an event loop (this waits for the user to do something on the dialog). The example uses a Class structure and is an example of Object Oriented Programming (OOP)

```
"""
================================
Overheating analysis using a GUI
================================

Module description
------------------
Demonstrates a simple TKinter GUI and event loop.
The GUI is used to create a room group scheme that allows the user to identify the rooms
for analysis, the target aps file, a xlsx filename and trigger an overheating analysis
for testing Ta 25 & 28 deg. The results are then exported to an Excel spreadsheet.

Notes
-----
Further sources of information on TKinter GUI programming:
https://www.cs.uct.ac.za/mit_notes/python/Introduction_to_GUI_Programming.html
https://stackoverflow.com/questions/17466561/best-way-to-structure-a-tkinter-application

"""
```

A GUI requires a specific code structure – the event loop; if you want to read up further try these links

```
import iesve
import tkinter as tk
import tkinter.messagebox as messagebox
import xlsxwriter
import os
```

Download the modules we need Messagebox is in a sub module to Tkinter so we need to load it specifically as it is not in the Tkinter module

```
class Dialog(tk.Frame):
    """
    Class defines a type of object by template, in this case a GUI window. The class
    inherits from the Tk Frame class; it is a container to organize widgets. The class
    definition contains functions to initialize the class Window, add widgets and
    functions that are called when events are triggered by buttons

    Args
    ----
    tk.Frame : Tk frame class

    """
```

We create a class (a ‘template’); this will contain all that we need (a ‘blueprint’) to create the dialog and execute the overheating analysis. Our class inherits from the Tkinter *Frame* class; this means it inherits (we can use) the functions in the *Frame* class

```
    def __init__(self, parent):
        """
        This function is called as soon as a class instance is created. It initializes
        the class instance and the variables we want to use. It's parent is set by the
        parent argument (in this example root). We use the self prefix to reference the
        variables to the instance we have created; these variables will be available
        to all functions inside the class. It then calls the function to create the
        dialog window

        Args
        ----
        self :  pointer to the class instance
        parent :  the parent object (root)
        """
```

__init__ is a special function that is executed as soon as an instance of the class is created

self is a reference to the Dialog class instance and it means ‘I belong to ..’

```
        # Create a frame widget; we will add widgets within the frame object
        self.frame = tk.Frame(parent)
        self.parent = parent

        # create the VE variables we want to use within the class
        self.project = iesve.VEProject.get_current_project()
        self.project_folder = self.project.path
        self.ve_folder = iesve.get_application_folder()
        self.save_file_name = 'Overheating_Results'
        self.results_reader = iesve.ResultsReader
        self.room_groups = iesve.RoomGroups()

        # Call the initialize window function
        self.init_window()
```

We create a Frame widget; all widgets inside the dialog will be children of this Frame instance

Within the class all functions can access variables prefixed with self, otherwise they are local to the function in which they are declared

We create the variables we want to use throughout the class

We call the function to create the dialog

```
    def init_window(self):
        """
        This function creates the window. We set attributes and define widgets ... labels,
        buttons etc, position them, set their default value and define any functions that
        they call.

        Args
        ----
        self :  pointer to class instance
        """

        # Set parent dialog title
        self.parent.title("Overheating Results")

        # Configure the frame widget
        self.frame.grid(row=0, column=0, sticky="nsew")
        self.frame.rowconfigure(6, weight=1)
```

We use self.parent to access an attribute of the parent object – the title label of the dialog

We configure the grid manager for self.frame; this will allow us to easily place widgets on the dialog. The grid is zero indexed from the top left corner

```
        # Widgets for grouping scheme
        # add label
        instructions_label = tk.Label(self.frame, text=('Click the button below to create'
        ' a new grouping scheme in your model. \nManually add the rooms you wish to be'
        ' analysed to the group "Analyze Overheating Results"'))
        instructions_label.grid(row=0, column=2, columnspan=2, sticky=tk.W)
        # add blank line
        tk.Label(self.frame, text=' ').grid(row=1, column=0, sticky=tk.W)

        # add button; this calls the create_grouping function when pressed
        tk.Button(self.frame, text="Create Grouping Scheme", command=self.create_grouping)\
        .grid(row=2, column=2, sticky=tk.W)

        # add blank lines
        tk.Label(self.frame, text=' ').grid(row=3, column=1, sticky=tk.W)
```

We create a label widget from the Tk module; it is a child of self.frame

We position the label widget on the grid. Sticky sets which side of the grid cell the widget should sit if smaller than the grid cell (compass points are used)

We create a button widget from the Tk module; it is a child of self.frame. Command sets the function to be called when the button is activated

We force some white space on the grid

```
        # Widgets for results file
        # add label
        results_label = tk.Label(self.frame, text='Select a Vista Results File (.aps)')
        results_label.grid(row=5, column=2, columnspan=2, sticky=tk.W)
        # get a list of aps files
        path = self.project_folder + 'Vista'
        files = os.listdir(path)
        aps_files = []
        for file in files:
            file = file.split('.')
            if file[-1] == 'aps':
                file = ('.').join(file)
                aps_files.append(file)

        # cerate a listbox and add aps file names
        self.listbox = tk.Listbox(self.frame)
        for file in aps_files:
            self.listbox.insert(tk.END, file)

        # configure listbox
        self.listbox.select_set(0)
        self.listbox.grid(row=6, column=2, columnspan=2, sticky='nsew')
        tk.Label(self.frame, text=' ').grid(row=7, column=1, sticky=tk.W)
```

We create a list of aps file names by looking in the project Vista folder

Split creates a list for each word in the filename string; we check the last item in the list to check if the filename suffix is aps. *Join* recombines the list into a string; we could have avoided this be simply using a different variable name on line 114 but it is useful to see both methods. If it is an aps file we add it to the list of available aps files

We create a list box and populate it with the list of aps filenames

We configure the listbox; select_set sets the default pick

```
        # Widgets for Excel filename
        # add labels
        excel_label = tk.Label(self.frame, text=('Overheating Results will be added to'
        ' an Excel sheet that will be saved into the main project folder'))
        excel_label.grid(row=8, column=2, columnspan=2, sticky=tk.W)
        name_xls_label = tk.Label(self.frame, text='Name the Excel file below:')
        name_xls_label.grid(row=9, column=2, columnspan=2, sticky=tk.W)

        # add entry box
        self.save_file_entry_box = tk.Entry(self.frame)
        self.save_file_entry_box.insert(0, self.save_file_name)
        self.save_file_entry_box.grid(row=10, column=2, columnspan=2, sticky='ew')
        tk.Label(self.frame, text=' ').grid(row=11, column=5, sticky=tk.W)

        # Widgets for Excel filename
        # add button; this calls the run_calc function when pressed
        calc_button = tk.Button(self.frame, text="Run Calculation", command=self.run_calc)
        calc_button.grid(row=12, column=2, sticky=tk.W)
```

We create a label widget from the Tk module; it is a child of self.frame

We create an entry box widget from the Tk module; it is a child of self.frame

We create a button widget from the Tk module; it is a child of self.frame. It calls the self.run_calc function

```
    def create_grouping(self):
        """
        Function called by the 'Create Grouping Scheme' button. It tests to see if the
        grouping scheme already exists. If it does not exist, it creates it. If it does
        exist a popup message is displayed to the user
        """

        schemes = self.room_groups.get_grouping_schemes()
        new_grouping_scheme_needed = True
        for scheme in schemes:
            if scheme['name'] == 'Overheating Analysis':
                new_grouping_scheme_needed = False

        if new_grouping_scheme_needed:
            scheme_index = self.room_groups.create_grouping_scheme('Overheating Analysis')
            self.room_groups.create_room_group(scheme_index, 'Analyze Overheating Results')
            self.room_groups.create_room_group(scheme_index, 'Do Not Analyze')
            tk.messagebox.showinfo('Grouping scheme', ('Manually select the rooms you'
            ' wish to be analyzed and add them to the group \'Analyze Overheating\''))
        else:
            tk.messagebox.showinfo('Grouping scheme', 'Grouping scheme already exists')
```

We create the function to set up the grouping scheme

Check if the grouping scheme already exists

If the grouping scheme does not exist create the grouping scheme and pop-up a messagebox with instructions

If the grouping scheme already exists pop-up a messagebox

```
    def calc(self, results_reader_file, rooms_to_be_analyzed):
        """
        Function calculates overheating hours above 25 & 28 degC whilst occupied for all
        rooms to be analyzed

        Args
        ----
        results_reader_file   :   results reader object for selected aps file
        rooms_to_be_analyzed  :   list of rooms

        """
```

We create a separate function for just calculating the overheating results called from run_calc; this avoids the run-calc function from becoming too long and hard to read

```
        # Create an empty list for the results
        overheating_data = []
        # Get a list of all the rooms in the model
        rooms = results_reader_file.get_room_list()

        # loop through every room in the model
        for room_number, room in enumerate(rooms):
            # Unpack the tuples of room data that is returned from the ResultsReader
            # function. the variables 'a' and'b' and not used here but had to be assigned
            name, room_id, a, b = room
```

Returns a list of *(room name, room ID, room area, room volume)* tuples for all rooms in the results file

We loop through all rooms in the model; we use enumerate to give us the count & the value of the list item (a tuple)

We unpack the tuple into separate variables

```
            # Check if the current room is part of the 'rooms_to_be_analyzed' list and
            # only perform the calculations if this is true
            if room_id in rooms_to_be_analyzed:
                np_room_temp = results_reader_file.get_room_results(room_id,
                'Comfort temperature', 'Dry resultant temperature', 'z', 1, 365)
                np_occupancy = results_reader_file.get_room_results(room_id,
                'Number of people', 'Number of people', 'z', 1, 365)

                # Create and zero counter variables for the range test
                range_test_28_deg = 0
                range_test_25_deg = 0
                occupied_hours = 0

                # Convert numpy arrays to python lists
                room_temp = np_room_temp.tolist()
                occupancy = np_occupancy.tolist()
```

If room is to be analyzed we get the results for Ta and occupancy

Setup counters

Convert the results to lists

```
                # Test if room is above 25 and 28 deg C while occupied, if it is
                # increment the appropriate counter variables. Test 28 degC first
                # so that occupied_hours does not double counted
                for num, temp in enumerate(room_temp):
                    if temp > 28 and occupancy[num] > 0:
                        range_test_28_deg += 1
                        range_test_25_deg += 1
                        occupied_hours += 1
                    elif temp > 25 and occupancy[num] > 0:
                        range_test_25_deg += 1
                        occupied_hours += 1
                    elif occupancy[num] > 0:
                        occupied_hours += 1
```

Use enumerate so we have a counter that we can use as an index in the Ta & occupancy lists

Test both lists at the same; if in occupancy AND Ta exceeds test increment counters for range test and occupied hours. If occupied and not overheating increment occupied hours

```
                # Calculate the % of hours above the thresholds by dividing the result
                # by the total occupied hours
                if occupied_hours > 0:
                    percent_25_deg = str(round(range_test_25_deg / occupied_hours *
                    100, 2)) + '%'
                    percent_28_deg = str(round(range_test_28_deg / occupied_hours *
                    100, 2)) + '%'
                else:
                    percent_25_deg = '0%'
                    percent_28_deg = '0%'

                # Create a list of the results for the room
                room_overheating_data = [name, range_test_25_deg, percent_25_deg,
                range_test_28_deg, percent_28_deg]

                # Append the room results list to the output
                overheating_data.append(room_overheating_data)

        return overheating_data
```

If occupied calculate % hours overheating

Gather the result in to a list that we can return from the function

```
    def run_calc(self):
        """
        Function called by the 'Run calc' button. It runs the tests and writes the
        results to a new excel sheet
        """

        # Search grouping schemes for the overheating scheme
        schemes = self.room_groups.get_grouping_schemes()
        scheme_handle = False
        for scheme in schemes:
            if scheme['name'] == 'Overheating Analysis':
                scheme_handle = scheme['handle']

        # Check if the grouping scheme exists
        if scheme_handle == False:
            tk.messagebox.showinfo('Grouping scheme', ('Create grouping scheme and'
            ' assign rooms before running calculation'))
            return
```

We create the function called by the calc_button; this contains

Check if the overheating grouping scheme exists; if it exists set the flag to the group handle attribute

If the overheating grouping scheme does not exist show a messagebox and exit the function

```
        # Get the rooms in the overheating grouping scheme
        overheating_group = self.room_groups.get_room_groups(scheme_handle)
        rooms_to_be_analyzed = []
        for group in overheating_group:
            if group['name'] == 'Analyze Overheating Results':
                rooms_to_be_analyzed = group['rooms']

        # Check if the grouping scheme is empty
        if not rooms_to_be_analyzed:
            tk.messagebox.showerror('Room group error', ('You must manually add some'
            ' rooms to the room group \'Analyze Overheating Results\''))
            return

        # Get the aps file name that has been selected in the listbox
        aps_file_name = self.listbox.get(tk.ACTIVE)
        if not aps_file_name:
            tk.messagebox.showinfo('APS error', ('No APS file selected. Please select'
            ' an APS file.'))
            return

        # Get the Excel filename that shas been entered in the entry box
        self.save_file_name = self.save_file_entry_box.get()
```

If the overheating grouping scheme exists get a list of the rooms to be analyzed

Check if the list of rooms is empty; if empty show a messagebox and exit the function

Get the aps filename that has been selected in the dialog, check if it is empty; if empty show a messagebox and exit the function

Get the Excel filename from the dialog

```
        # Create excel workbook
        workbook = xlsxwriter.Workbook(self.project_folder + '\\' + self.save_file_name
        + '.xlsx')

        # Create excel work sheet
        sheet1 = workbook.add_worksheet('sheet1')

        # Get results
        results_reader_file = self.results_reader.open(aps_file_name)

        # Run calculation
        print('Running Calculations')
        overheating_data = self.calc (results_reader_file, rooms_to_be_analyzed)

        # Define column headings for excel worksheet
        heading = ['Room Name', 'Hours > 25\u2070C', '% Hours > 25\u2070C',
        'Hours > 28\u2070C', '% Hours > 28\u2070C']
```

Create a new Excel workbook (note xlsxwriter cannot open existing workbooks)

Create a worksheet

Using the aps filename get a resultsreader object

Call the calculation function; passing the arguments resultsreader object & the list of rooms to be analyzed. The value (list) returned is assigned to overheating_data

Create a list of column headings; note the use of escape characters

```
        # write data to excel worksheets
        print('Writing results to Excel Sheet')
        # write column headings
        sheet1.write_row(0, 0, heading)
        # write results data; start at row 1
        y = 1
        for row in overheating_data:
            sheet1.write_row(y, 0, row,)
            y += 1

        # set column widths
        sheet1.set_column('A:E', 15)

        # Save the excel workbook
        try:
            workbook.close()
        except PermissionError as e:
            print("Couldn't close workbook: ", e)
        os.startfile(self.project_folder + '\\' + self.save_file_name + '.xlsx')

        # close script
        self.parent.destroy()


    def close(self):
        # close script when the X button is pressed
        self.parent.destroy()
```

Write the column headings on to the worksheet

Write the data a row (a room) at a time on to the worksheet

Configure the column widths so the data is fully visible

Save the workbook; we use a *try* statement to handle the situation of the workbook being opened by another app since being created

Open the Workbook in Excel

The destroy method kills the widget; in this it kills the parent widget and thus ends the script

```
def main():
    """
    Function that creates a root window, dialog and runs the event loop
    The event loop cycles waiting for user input until the window is closed
    """

    # Create root window and configure it so it is always is visible
    # Root is the top object in the dialog object hierarchy
    root = tk.Tk()
    root.geometry("550x450")
    root.deiconify()  # restore if minimized
    root.lift()  # make the window active

    # Create an instance of the class Window using the root window
    app = Dialog(root)

    # Start the event loop
    root.mainloop()

if __name__ == '__main__':
    run = main()
```

The code in this function could easily be after line 354, but making a function makes for easy-to-read structured code

We create a ‘root’ object that will be the top of the dialog object tree; everything will be a child of root

We set some attributes of root

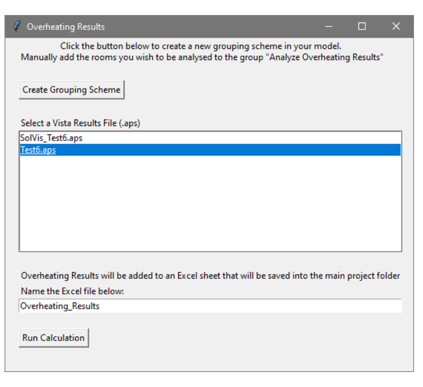
We call the class to create an instance; we pass in the argument root – this will be the parent object

We start the event loop; this loops waiting for the user to do something on the dialog

We use the __name__ conditional to ensure we only run the script when it is the main program. We call our main function to start the script

Sample output:

Tk dialog ...



Overheating Results

Click the button below to create a new grouping scheme in your model.
Manually add the rooms you wish to be analysed to the group "Analyze Overheating Results"

Create Grouping Scheme

Select a Vista Results File (.aps)

SolVis_Test6.aps
Test6.aps

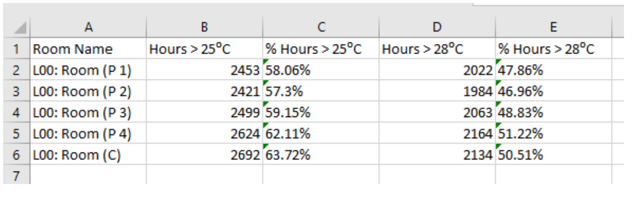
Overheating Results will be added to an Excel sheet that will be saved into the main project folder

Name the Excel file below:

Overheating_Results

Run Calculation

Workbook ...



	A	B	C	D	E
1	Room Name	Hours > 25°C	% Hours > 25°C	Hours > 28°C	% Hours > 28°C
2	L00: Room (P 1)	2453	58.06%	2022	47.86%
3	L00: Room (P 2)	2421	57.3%	1984	46.96%
4	L00: Room (P 3)	2499	59.15%	2063	48.83%
5	L00: Room (P 4)	2624	62.11%	2164	51.22%
6	L00: Room (C)	2692	63.72%	2134	50.51%
7					