

FCM162 - HowTo - Python in the REAL World - Part 110

By Greg Walters

Last month, a PAGE user came to both Don and myself for some help with a fairly new Python library that implements a spreadsheet like table widget written completely in Python. He was fairly new to PAGE and was having trouble getting the table to initialize properly. I thought the library has a tremendous amount of possibilities, so I thought I'd share the information with you. Just so you know, you don't have to use PAGE to implement the sheet widget, but you do have to use Tkinter at least.

To get started, you need to install the library and as usual, we can use Pip to do it.

```
$ pip install tksheet
```

Once you've done that, you might want to consider downloading the entire source code from the github repository at <https://github.com/ragardner/tksheet> . I suggest this since there are so many options that this library has, you might not catch some without digging into the source code. The documentation for the library is currently being written, so it's easy to suss out how to do some of the options by having the code available.

We'll use a modified version of the demo program to start with, which does not require PAGE. You can find the full version on the homepage of the repository.

```
from tksheet import Sheet
import tkinter as tk
```

As almost always, we want to import the important libraries, in this case, the tksheet and tkinter libraries.

Now we'll create a class called demo. This will hold all of the code for the demo program that deals with the sheet widget. As you go through the code, you will see that many of the functions later on are simply there to allow you to customize the builtin functions that the library has.

```
class demo(tk.Tk):
    def __init__(self):
        tk.Tk.__init__(self)
        self.grid_columnconfigure(0, weight=1)
        self.grid_rowconfigure(0, weight=1)
        self.frame = tk.Frame(self)
        self.frame.grid_columnconfigure(0, weight=1)
        self.frame.grid_rowconfigure(0, weight=1)
```

As you already know, the __init__ function sets up various parameters and defaults. In the next lines, the sheet object is simply an instance of Sheet.

```
        self.sheet = Sheet()
```

```

self.frame,
page_up_down_select_row=True,
column_width=120,
startup_select=(0, 1, "rows"),

```

Here is where you load the dummy data into the sheet.

```

data=[[
    f"Row {r}, Column {c}\nnewline1\nnewline2" for c in range(50)
] for r in range(1000)], #to set sheet data at startup

height=500, #height and width arguments are optional
width=1200 #For full startup arguments see DOCUMENTATION.md
)

```

Next, we enable the various bindings that are already coded in the library.

```

self.sheet.enable_bindings((
    "single_select", #"single_select" or "toggle_select"
    "column_select",
    "row_select",
    "arrowkeys",
    "row_height_resize",
    "double_click_row_resize",
    "right_click_popup_menu",
    "rc_select",
    "rc_insert_column",
    "rc_delete_column",
    "rc_insert_row",
    "rc_delete_row",
    "edit_cell"))

```

The next two lines insert the frame and sheet objects into a Tkinter grid.

```

self.frame.grid(row=0, column=0, sticky="nswe")
self.sheet.grid(row=0, column=0, sticky="nswe")

```

As you can tell from the commented code, the next set of lines show ways to control various things like the theme, highlighting columns, rows and cells, etc.

```

""" _____ EXAMPLES _____ """
""" _____ """

# _____ CHANGING THEME _____

#self.sheet.change_theme("light green")

# _____ DISPLAY SUBSET OF COLUMNS _____

self.sheet.display_subset_of_columns(indexes=[0, 1, 2, 3, 4, 5],
                                     enable=True)

self.sheet.insert_column(idx=0)

```

```

self.sheet.insert_columns(columns=5,
                           idx=10,
                           mod_column_positions=False)

# _____ HIGHLIGHT / DEHIGHLIGHT CELLS _____

self.sheet.highlight_cells(row=5, column=5, fg="red")
self.sheet.highlight_cells(row=5, column=1, fg="red")
self.sheet.highlight_cells(row=5,
                           bg="#ed4337",
                           fg="white",
                           canvas="row_index")
self.sheet.highlight_cells(column=0,
                           bg="#ed4337",
                           fg="white",
                           canvas="header")

# _____ CELL / ROW / COLUMN ALIGNMENTS _____

self.sheet.align_cells(row=1, column=1, align="e")
self.sheet.align_rows(rows=3, align="e")
self.sheet.align_columns(columns=4, align="e")

# _____ ADDITIONAL BINDINGS _____

```

The rest of the code overrides the default events and bindings and provides examples of how to handle them in your own code.

```

def all_extra_bindings(self, event):
    print(event)

def begin_edit_cell(self, event):
    print(event) # event[2] is keystroke
    return event[
        2] # return value is the text to be put into cell edit window

def window_resized(self, event):
    pass
    #print (event)

def deselect(self, event):
    print(event, self.sheet.get_selected_cells())

def rc(self, event):
    print(event)

def ctrl_a(self, response):
    print(response)

def row_select(self, response):
    print(response)

def column_select(self, response):

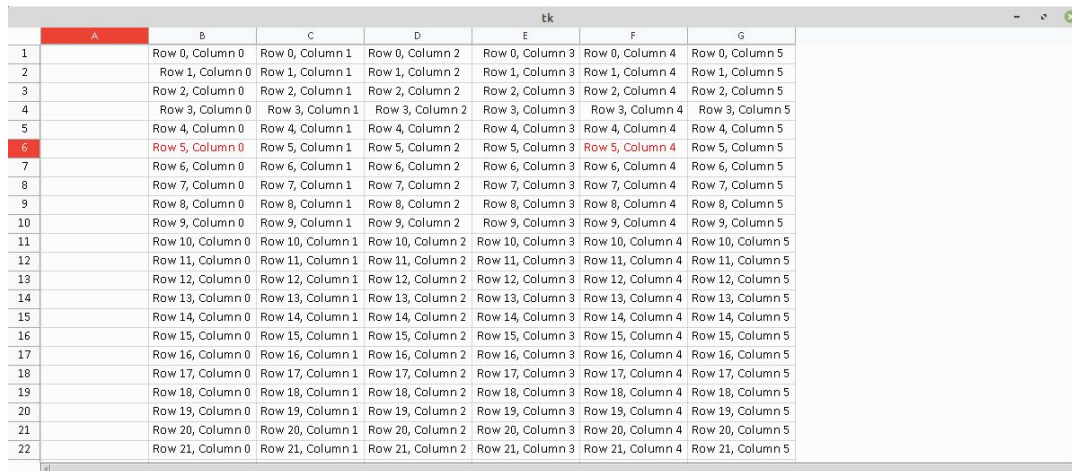
```

```
print(response)
```

Finally, the required instantiation of the demo class object and the call to the Tkinter mainloop.

```
app = demo()
app.mainloop()
```

When you fire up the program, you should see something that looks like this:



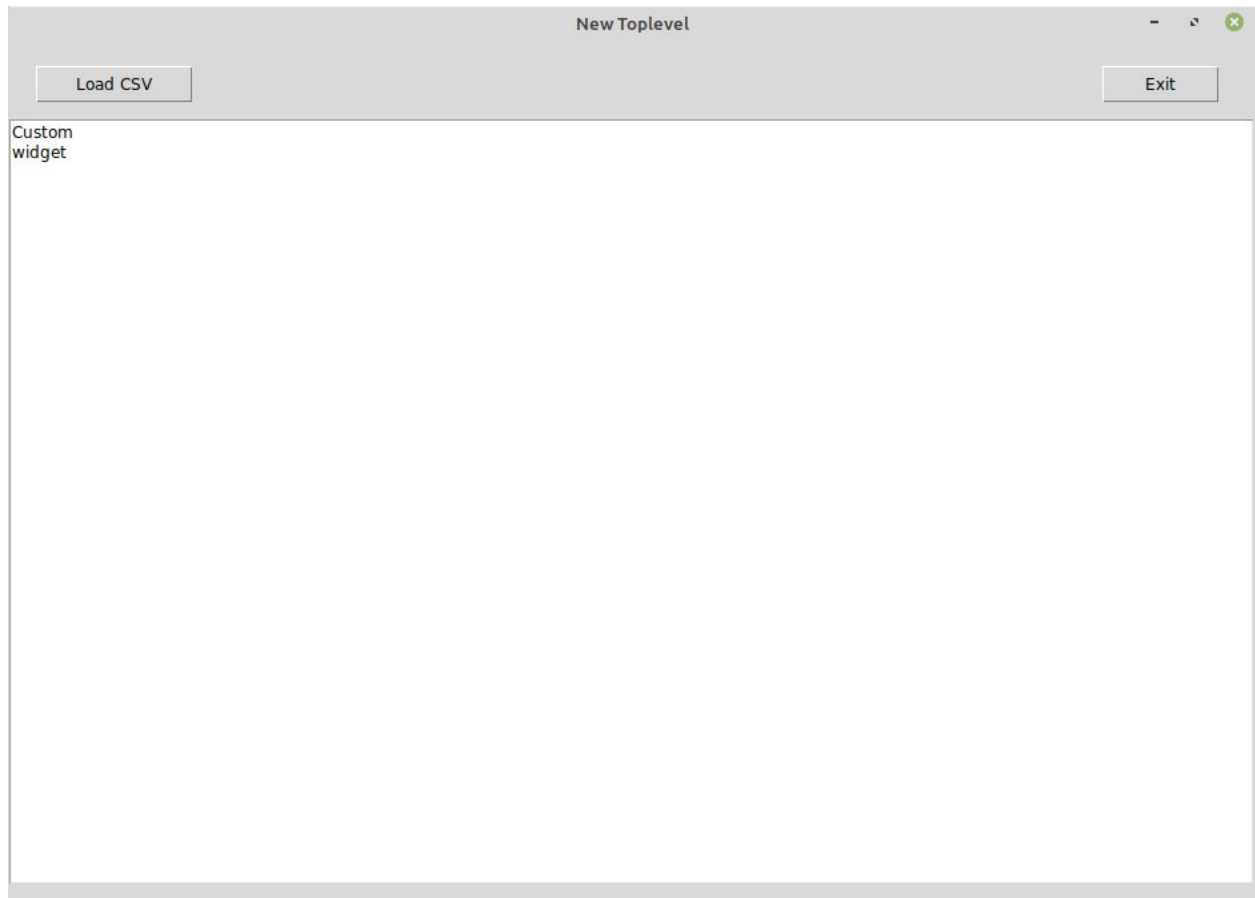
	A	B	C	D	E	F	G
1		Row 0, Column 0	Row 0, Column 1	Row 0, Column 2	Row 0, Column 3	Row 0, Column 4	Row 0, Column 5
2		Row 1, Column 0	Row 1, Column 1	Row 1, Column 2	Row 1, Column 3	Row 1, Column 4	Row 1, Column 5
3		Row 2, Column 0	Row 2, Column 1	Row 2, Column 2	Row 2, Column 3	Row 2, Column 4	Row 2, Column 5
4		Row 3, Column 0	Row 3, Column 1	Row 3, Column 2	Row 3, Column 3	Row 3, Column 4	Row 3, Column 5
5		Row 4, Column 0	Row 4, Column 1	Row 4, Column 2	Row 4, Column 3	Row 4, Column 4	Row 4, Column 5
6		Row 5, Column 0	Row 5, Column 1	Row 5, Column 2	Row 5, Column 3	Row 5, Column 4	Row 5, Column 5
7		Row 6, Column 0	Row 6, Column 1	Row 6, Column 2	Row 6, Column 3	Row 6, Column 4	Row 6, Column 5
8		Row 7, Column 0	Row 7, Column 1	Row 7, Column 2	Row 7, Column 3	Row 7, Column 4	Row 7, Column 5
9		Row 8, Column 0	Row 8, Column 1	Row 8, Column 2	Row 8, Column 3	Row 8, Column 4	Row 8, Column 5
10		Row 9, Column 0	Row 9, Column 1	Row 9, Column 2	Row 9, Column 3	Row 9, Column 4	Row 9, Column 5
11		Row 10, Column 0	Row 10, Column 1	Row 10, Column 2	Row 10, Column 3	Row 10, Column 4	Row 10, Column 5
12		Row 11, Column 0	Row 11, Column 1	Row 11, Column 2	Row 11, Column 3	Row 11, Column 4	Row 11, Column 5
13		Row 12, Column 0	Row 12, Column 1	Row 12, Column 2	Row 12, Column 3	Row 12, Column 4	Row 12, Column 5
14		Row 13, Column 0	Row 13, Column 1	Row 13, Column 2	Row 13, Column 3	Row 13, Column 4	Row 13, Column 5
15		Row 14, Column 0	Row 14, Column 1	Row 14, Column 2	Row 14, Column 3	Row 14, Column 4	Row 14, Column 5
16		Row 15, Column 0	Row 15, Column 1	Row 15, Column 2	Row 15, Column 3	Row 15, Column 4	Row 15, Column 5
17		Row 16, Column 0	Row 16, Column 1	Row 16, Column 2	Row 16, Column 3	Row 16, Column 4	Row 16, Column 5
18		Row 17, Column 0	Row 17, Column 1	Row 17, Column 2	Row 17, Column 3	Row 17, Column 4	Row 17, Column 5
19		Row 18, Column 0	Row 18, Column 1	Row 18, Column 2	Row 18, Column 3	Row 18, Column 4	Row 18, Column 5
20		Row 19, Column 0	Row 19, Column 1	Row 19, Column 2	Row 19, Column 3	Row 19, Column 4	Row 19, Column 5
21		Row 20, Column 0	Row 20, Column 1	Row 20, Column 2	Row 20, Column 3	Row 20, Column 4	Row 20, Column 5
22		Row 21, Column 0	Row 21, Column 1	Row 21, Column 2	Row 21, Column 3	Row 21, Column 4	Row 21, Column 5

While that wasn't too hard at all, I believe that using PAGE makes the process much easier. I've created a VERY simple PAGE GUI to show how quickly and easy it is.

Create a folder to hold your project and start up PAGE within that folder. Expand the default Toplevel form just a bit and place two buttons near the very top of the form, one on the left and one on the right. The left one should have the text "Load CSV" and the one on the right "Exit". In the command attribute for the left button enter "on_btnLoad" and for the right button "on_btnExit". (***We've covered PAGE programming so often, this should be very obvious, but if it isn't, look at one of my previous articles about using PAGE. My article in FCM #155 should be a good easy reference.***)

Next, place a frame in the form that takes up pretty much of the rest of the form. Then place a custom widget into that frame and expand it to fill the frame. (Mouse-3 (Right click for most people)) | Widget | Fill Container). Save your project as **tksheetGUI.tcl** and generate your python code.

This is what your project hopefully looks like. If not, that's ok. You get the general idea.



Now open `tksheet_support.py` in your IDE or editor and let's get to work.

PAGE only creates a single import line, since that's all that you need to get started. Add the following lines to the import section of the code to support our needs. The line that PAGE gives us is not in bold.

```
import sys
import platform
import os

# Third party libraries
from tksheet import Sheet
import pandas as pd
```

We also need to modify the import section that PAGE provides for us. Normally, you would not need to do this, but we are going to add support for the Tkinter messagebox and the filedialog sub-systems.

```
try:
    import Tkinter as tk
    import tkinterFileDialog as filedialog
```

```

import tkMessageBox as messagebox
except ImportError:
    import tkinter as tk
    from tkinter import messagebox
    from tkinter import filedialog

```

Note that we actually don't do anything with the messagebox but it's there for your possible future development.

Now we need to add a couple of lines to the init function, that again, PAGE provides for us.

```

def init(top, gui, *args, **kwargs):
    global w, top_level, root
    w = gui
    top_level = top
    root = top
    startup()
    init_custom()

```

As I'm sure you are aware, this will call the startup() and init_custom() functions before the form is shown to the user. Let's look at the startup function first.

```

def startup():
    global version
    pv = platform.python_version()
    print(f"Running under Python {pv}")
    # Set the path of the program
    global proppath
    proppath = os.getcwd()
    print(proppath)
    version = "0.0.1"
    print(f"Version: {version}")
    # =====
    # Some default global values
    # =====
    global defaultColumns, defaultRows, useBoldFont, defFont
    defaultColumns = 30
    defaultRows = 50
    useBoldFont = False
    defFont = "Arial"

```

Yes, there is some code that aren't strictly needed, but I like to add them. Basically, we get and provide the python version, program path and program version and print all of that to the terminal window. Then we set up some global default values for use with the tksheet library.

In the init_custom() function we will be initializing various settings of the tksheet for our use. It took me a fair amount of time to narrow down the actual settings needed for proper demo use.

There are many other settings as you saw in the earlier demo program, but for us, these are needed. One thing to note is the theme setting. There are four default themes defined by default. The first demo used the **light green** theme. For the PAGE version, we will use the **dark blue** theme.

```
def init_custom():
    w.Custom1.page_up_down_select_row = True
    w.Custom1.column_width = 120
    w.Custom1.startup_select = (0, 1, "rows")
    w.Custom1.theme = "dark blue"
    w.Custom1.insert_column(idx=0)
    w.Custom1.insert_columns(columns=5, idx=10, mod_column_positions=False)
```

Now we need to enable the bindings that we want so that the functionality is what you would normally expect for a spreadsheet demo. These are things like selecting a single cell, a row or column and using the arrow keys as well as allowing for “right click” (mouse 3) context menu support.

```
w.Custom1.enable_bindings(
    (
        "single_select", # "single_select" or "toggle_select"
        # To support column/row select
        "column_select",
        "row_select",
        # To support column/row resize
        "column_width_resize",
        "double_click_column_resize",
        "row_width_resize",
        "column_height_resize",
        "arrowkeys",
        "row_height_resize",
        "Double_click_row_resize",
```

The next few lines allow you to use the popup context menu. Some of them are pretty obvious, but the last 6 lines aren't as clear why they would be needed.

```
        "right_click_popup_menu",
        "rc_select",
        "rc_insert_column",
        "rc_delete_column",
        "rc_insert_row",
        "rc_delete_row",
        "hide_columns",
        # The following 5 lines must be included for popup support
        "copy",
        "cut",
        "paste",
        "delete",
        "undo",
```

```

        "edit_cell",
    )
)

```

The last part of the `init_custom` function deals with binding the Mouse-3 button to the “rc” routine and sets extra bindings for us to be able to override the default functions within the library.

```

w.Custom1.bind("<3>", rc)
w.Custom1.extra_bindings(
    [
        ("cell_select", on_cellSelect),
        ("column_select", on_columnSelect),
        ("row_select", on_rowSelect),
    ]
)

```

Now we’ll create the four callback functions that we just defined. I must admit that I created the first three for a different demo that printed the information to a status label, but I felt that for this very simple demo, printing to the terminal was ok.

```

def on_columnSelect(p1):
    print(f"on_columnSelect Triggered {p1}")
    print(p1)
    selcol = p1[1]
    colname = headers[p1[1]]
    disp = f"Column Selected - Col: {selcol}    ColName: {colname}"
    print(disp)

def on_rowSelect(p1):
    print(f"on_rowSelect Triggered {p1}")
    print(p1)
    selrow = p1[1]
    disp = f"Row Selected - {selrow}"
    print(disp)

def on_cellSelect(p1):
    global headers
    print(f"on_cellSelect triggered: {p1}")
    selrow = p1[1] + 1
    selcol = p1[2] + 1
    colname = headers[p1[2]]
    disp = f"Cell Selected - Row: {selrow}    Col: {selcol}    ColName: {colname}"
    print(disp)

def rc(event):
    print(event)

```


I created a simple class to handle the actual tksheet as a custom control.

```
class sheet(Sheet):
    def __init__(self, parent, **kw):
        Sheet.__init__(
            self,
            parent,
            page_up_down_select_row=True,
            column_width=120,
            font=("Arial", 12, "bold"),
            theme="dark blue",
            height=500, # height and width arguments are optional
            width=1200, # For full startup arguments see DOCUMENTATION.md
        )
```

Almost at the end now. We still have to assign the library as the custom control. PAGE supplies a line that says "Custom = tk.Frame", but we need to **replace tk.Frame** with our external tkinter control library. You can add the following line and comment the 'Custom = tk.Frame' line out.

```
Custom = sheet
```

Finally, we need to provide the callback and support functions for the buttons that we defined when we designed the GUI. PAGE generated the first two functions for us as skeletons. The final function helps the import of the CSV files. You already have seen the proper way to end a PAGE application, but I've included it here for your convenience.

```
def on_btnExit():
    print("tksheet_support.on_btnExit")
    sys.stdout.flush()
    destroy_window()
```

Now we have the callback for the load csv file button. It's a little bloated, but I thought it would be easier for you to follow the code this way rather than write it more elegantly.

```
def on_btnLoad():
    print("tksheetdemo1_support.on_btnLoadCSV")
    sys.stdout.flush()
    global headers, prospath
    filename = filedialog.askopenfilename(
        initialdir=prospath,
        title="Select file",
        filetypes=(("CSV files", "*.csv"), ("all files", "*.*")),
```

```

)
data = load_csv_file(filename)
# print(headers)
w.Custom1.headers(headers)
# Now load data into sheet
w.Custom1.set_sheet_data(data)
# Get the number of rows and columns
global totalrows, totalcols
totalrows = w.Custom1.total_rows()
totalcols = w.Custom1.total_columns()
# Print the information to the terminal
print(f"Total Rows: {totalrows}")
print(f"Total Cols: {totalcols}")

```

Last but not least, here is the `load_csv_file` function. Here we are using pandas to read the CSV file and convert it into a list. That way, the `tksheet` library can utilize it.

```

def load_csv_file(filename):
    global headers
    # Load the local csv file into a pandas dataframe
    df = pd.read_csv(filename)
    headers = list(df.columns)
    # Convert it into a list
    dl = df.values.tolist()
    # Return it to the calling function
    return dl

```

Be sure to save your file.

Last but not least, you need to know where a good CSV file is. For this demo, I have used the `titanic3.csv` file which is provided with the source code for the Pandastable Library. I'm certain it's been released into the public domain. I copied it and put it into the development folder for quick access.

Once you run the program, it should look something like this:

New Toplevel									
Load CSV		Exit							
	Unnamed: 0	pclass	survived	sex	age	sibsp	parch		
1	0	1	1	female	29.0	0	0	2416	
2	1	1	1	male	0.9167	1	2	1137	
3	2	1	0	female	2.0	1	2	1137	
4	3	1	0	male	30.0	1	2	1137	
5	4	1	0	female	25.0	1	2	1137	
6	5	1	1	male	48.0	0	0	1995	
7	6	1	1	female	63.0	1	0	1350	
8	7	1	0	male	39.0	0	0	1120	
9	8	1	1	female	53.0	2	0	1176	
10	9	1	0	male	71.0	0	0	PC 1	
11	10	1	0	male	47.0	1	0	PC 1	
12	11	1	1	female	18.0	1	0	PC 1	
13	12	1	1	female	24.0	0	0	PC 1	
14	13	1	1	female	26.0	0	0	1987	
15	14	1	1	male	80.0	0	0	2704	
16	15	1	0	male	nan	0	0	PC 1	
17	16	1	0	male	24.0	0	1	PC 1	
18	17	1	1	female	50.0	0	1	PC 1	
19	18	1	1	female	32.0	0	0	1181	
20	19	1	0	male	36.0	0	0	1305	
21	20	1	1	male	37.0	1	1	1175	
22	21	1	1	female	47.0	1	1	1175	
23	22	1	1	male	26.0	0	0	1113	

I hope that you enjoyed this month's project as much as I did creating it. This library has a lot of potential and will be great for quick display of things in table or sheet format. However, there is no code to write back to a file (that's simple enough for you to work out) and there are no calculation functions at this point. But this is a fantastic library and something that has been needed for a very long time!

As I did once before, I've created a repository on github to hold the code and images from this article. You can find it at <https://github.com/gregwa1953/FCM162> .

As always, until next time; stay safe, healthy, positive and creative!

Greg