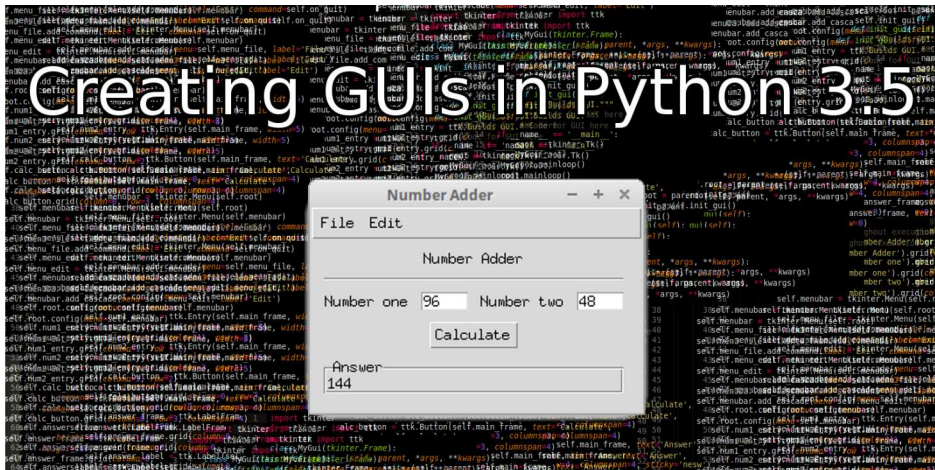


[Home](#)[Top Resources](#)[What Is This Blog?](#)[Who Are You?](#)[Got Something To Say?](#)[HOME](#)[TOP RESOURCES](#)[WHAT IS THIS BLOG?](#)[WHO ARE YOU?](#)[GOT SOMETHING TO SAY?](#)

How to Build a GUI in Python 3.5

BY JAMAL MOIR

10 COMMENTS

Looking around the community, at other people products and such I get the feeling that Python is being used more for services and scientific applications rather than desktop applications. That being said, python desktop applications are by no means dead and lots of people want to create them (me included).

In order to create a desktop application we need to use a GUI library to help us build it. Now, if you are a Python 2.7, or even 3.3 and below user then your choices are plentiful. With Python 3.5 and above however, at the time of writing this post, we are quite limited.

THE CURRENT STATE OF PYTHON 3.5 GUI LIBRARIES

After doing some research into what is available to me I was quite disappointed if I'm quite honest. A lot of popular libraries just haven't been ported forward to 3.5 yet, especially the 64 bit version.

I was looking for a native-looking, cross-platform option and neither wxPython-Phoenix (Python's wx binding) nor PySide (a Python QT binding) seemed to work with Python 3.5 64 bit version and these seemed to be the popular choices with the most relaxed licenses. They seemed to work with Python 3.3 and I believe wxPython works with 32 bit Python 3.5 from what people have been saying online, but has problems compiling with Python 3.5 64 bit version.

UPDATE: According to Opaque, you can actually get wxPython-Phoenix working using the following method:

“ You can get a version of wxpython phoenix (<https://github.com/wxWidgets>

[/Phoenix](#) working with 64bit python 3.5.1 by installing a wheel from <http://wxpython.org/Phoenix/snapshot-builds/> I've been using it for the last few months without problems.

So what do we do? Well, we go for Tkinter. Tkinter is a GUI library built on top of tcl/Tk, it also happens to be included in Python's standard library and so very convenient.

Where possible, we will be using tkinter.ttk widgets. The difference between vanilla tkinter and tkinter.ttk widgets is that tkinter.ttk widgets use style to define their appearance. They are less customisable than the plain tkinter widgets, but give a more 'native' look.

THE BASIC STRUCTURE OF A TK GUI

During this post we will be creating a very simple program that adds two numbers to outline the basics of GUI programming with tkinter and ttk in Python 3.5.

```
1 import tkinter
2 from tkinter import ttk
3
4 class Adder(ttk.Frame):
5     """The adders gui and functions."""
6     def __init__(self, parent, *args, **kwargs):
7         ttk.Frame.__init__(self, parent, *args, **kwargs)
8         self.root = parent
9         self.init_gui()
10
11     def init_gui(self):
12         """Builds GUI."""
13         self.root.title('Number Adder')
14
15 if __name__ == '__main__':
16     root = tkinter.Tk()
17     Adder(root)
18     root.mainloop()
```

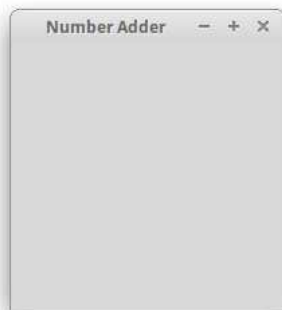
This is an object orientated approach to GUI building and seems as Tkinter is an object orientated GUI framework, it works well. It's simple enough, first of all we import tkinter and ttk so we can access the GUI tools we need.

We then create our own class which extends the ttk.Frame class. This is where the GUI building will go on. Pretty much everything to do with our GUI will be handled in here.

The init_gui() method is where the definition of our GUI will be, for now we will just set the title of the window. We do this by accessing the root of our application and setting its title.

Then, down in the if __name__ == '__main__' section we create our top level widget; the main window of the application and then create an instance of our class and start its mainloop.

Our GUI now looks like this:



One thing you should note here, is that we don't do this:

```
1 from tkinter import *
```

Most tutorials and documentation online, including the official [Python documentation](#) tell you to do this. Don't, from * import * is the devil. Keep your code concise, easy to follow and avoid naming

SHARES

BASIC WIDGETS

```

1 def init_gui(self):
2     """Builds GUI."""
3     self.root.title('Number Adder')
4
5     self.num1_entry = ttk.Entry(self, width=5)
6
7     self.num2_entry = ttk.Entry(self, width=5)
8     self.calc_button = ttk.Button(self, text='Calculate')
9
10    self.answer_frame = ttk.LabelFrame(self, text='Answer',
11                                       height=100)
12
13    self.answer_label = ttk.Label(self.answer_frame, text='')
14
15    # Labels that remain constant throughout execution.
16    ttk.Label(self, text='Number Adder')
17    ttk.Label(self, text='Number one')
18    ttk.Label(self, text='Number two')
19
20    ttk.Separator(self, orient='horizontal')

```

You will notice that widget creation is simply instantiating a new object. The first argument you need to enter when creating a new widget is its parent. Its parent is the widget that will contain the new one you are making, in this case we are putting all our widgets inside the main all encompassing frame; our custom class that we made, and we reference it via 'self'. You may be wondering what the parent of the topmost widget is. Well, if you take a look at the code we wrote in the previous section you will see.

```

1 root = tkinter.Tk()

```

That is our top level widget and so clearly doesn't have a parent.

If you run this code you will notice that nothing appears. The GUI is hasn't changed at all. That's because although we have created new widgets, we haven't placed them anywhere in our GUI, they are just floating about somewhere. In the next two sections we will see two ways of making them appear.

With Tkinter there are lots of widgets at our disposal and we only use a few here, for a full list of tkinter widgets [click here](#), and for a full list of tkinter.ttk widgets [click here](#).

PACK

The first way we can make our widgets appear is by using pack(). This is the quickest and easiest way, but gives you less freedom when defining the way your GUI is layed out.

```

1 def init_gui(self):
2     """Builds GUI."""
3     self.root.title('Number Adder')
4     self.pack(fill='both', padx=10, pady=5)
5
6     ttk.Label(self, text='Number Adder').pack(pady=5)
7
8     ttk.Label(self, text='Number one').pack(side='left', padx=5, pady=5)
9     self.num1_entry = ttk.Entry(self, width=5)
10    self.num1_entry.pack(side='left', padx=5, pady=5)
11
12    ttk.Label(self, text='Number two').pack(side='left', padx=5, pady=5)
13    self.num2_entry = ttk.Entry(self, width=5)
14    self.num2_entry.pack(side='left', padx=5, pady=5)

```

Using pack() without any arguments will arrange your widgets in a column, you can change this behaviour with the optional argument 'side' where you can define 'top', 'bottom', 'left' or 'right'. The default is 'top', which will align widgets vertically. To align things horizontally, use 'left'. You can also make a widget fill it's parent element along the x or y axis using the optional argument 'fill'. You can make it fill the x or y axis by specifying 'x' or 'y' and can make it fill both by specifying 'both'.

The padding for a widget can be set via the 'padx', 'pady', 'ipadx' and 'ipady' where padx and pady are the external padding and ipadx and ipady is internal padding.

Our GUI should now look like this:



Using `pack()` however, is only practical for very simple GUIs, for more complicated ones, it's best to use the grid layout.

UNDERSTANDING THE GRID LAYOUT

The grid layout is a powerful tool for defining the layout of your GUI, much more so than `pack` and is the method that I have come to favour. Obviously, the grid layout is, well, a grid. A collection of rows and columns that you can insert widgets into to structure your GUI.

```

1 def init_gui(self):
2     """Builds GUI."""
3     self.root.title('Number Adder')
4
5     self.grid(column=0, row=0, sticky='nsew')
6
7     self.num1_entry = ttk.Entry(self, width=5)
8     self.num1_entry.grid(column=1, row=2)
9
10    self.num2_entry = ttk.Entry(self, width=5)
11    self.num2_entry.grid(column=3, row=2)
12
13    self.calc_button = ttk.Button(self, text='Calculate')
14    self.calc_button.grid(column=0, row=3, colspan=4)
15
16    self.answer_frame = ttk.LabelFrame(self, text='Answer',
17                                       height=100)
18    self.answer_frame.grid(column=0, row=4, colspan=4, sticky='nesw')
19
20    self.answer_label = ttk.Label(self.answer_frame, text='')
21    self.answer_label.grid(column=0, row=0)
22
23    # Labels that remain constant throughout execution.
24    ttk.Label(self, text='Number Adder').grid(column=0, row=0,
25                                              colspan=4)
26    ttk.Label(self, text='Number one').grid(column=0, row=2,
27                                             sticky='w')
28    ttk.Label(self, text='Number two').grid(column=2, row=2,
29                                             sticky='w')
30
31    ttk.Separator(self, orient='horizontal').grid(column=0,
32                                                  row=1, colspan=4, sticky='ew')
33
34    for child in self.winfo_children():
35        child.grid_configure(padx=5, pady=5)

```

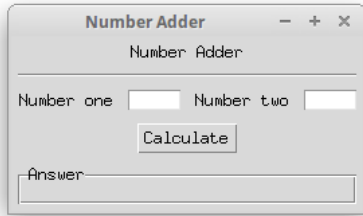
When using the grid layout you should first have a good idea of what you want your GUI to look like. Then using a combination of placing widgets in specified rows and columns, making them span across multiple rows and columns and setting sides for them to stick to we can create complex GUIs.

Using the grid manager is incredibly easy, simply use the `grid()` method to place your widget in the desired row and column on the grid. Note that the grid starts at 0 and not entering a row or column will result in them defaulting to this value.

When you place your widget on the grid, by default it will be centred within the cell. To change this you can enter a 'sticky' argument in the `grid()` method. You can pass it in a string containing any of 'n', 'e', 's', 'w' and can combine them together too.

Another useful feature of the grid layout is the ability to make widgets span across multiple rows and columns using the optional arguments 'rowspan' and 'colspan'. Remember that if your last column is '4' then you actually have five columns, so if you want something to span across the whole grid, that is something to remember.

Our GUI now looks like this:



An important point to note here is that `grid()` returns `None`, so when you need to access the widget at a later date, first create the widget and assign it to a variable, then call the `grid` method on that variable.

CREATING A MENU BAR

I feel creating a menu bar in Tkinter is not as simple as it could be, that being said, it's not difficult either.

```
1 self.root.option_add('*tearOff', 'FALSE')
```

First what we do is turn off tearing off. What this does is prevent you from being able to detach the menu bar from the main window. When it is turned on you can drag the menu bar off the main application window into a window of its own. We don't want this behaviour, so we will turn it off.

```
1 self.menubar = tkinter.Menu(self.root)
2
3 self.menu_file = tkinter.Menu(self.menubar)
4
5 self.menu_edit = tkinter.Menu(self.menubar)
```

We then create the main menu bar itself and set its parent to the root element. On the menu bar we will have 'File' and 'Edit' menus, so we create those exactly as before, except we set their parents to the main menu bar.

```
1 self.menu_file = tkinter.Menu(self.menubar)
2 self.menu_file.add_command(label='Exit',)
```

Now, under the 'File' menu we will create an item that exits the application when you click it as an example of how to add things to menus. We can do this via the `add_command()` method. We will see later how to make this do something.

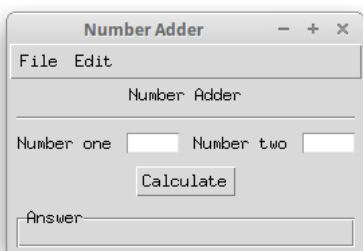
```
1 self.menubar.add_cascade(menu=self.menu_file, label='File')
2 self.menubar.add_cascade(menu=self.menu_edit, label='Edit')
```

Now we have three separate menu bars, but we want to add the file menu and the edit menu to the main menu. We do this using the `add_cascade()` method, which will add the two menus to the menu bar as cascading menus, ie. drop downs. You can also add cascades to the file and edit menus.

```
1 self.root.config(menu=self.menubar)
```

The final thing we need to do is set the root's menu to our newly created menu bar and we are done.

Our GUI should now have a menu bar:



BINDING FUNCTIONS

In our example program we have 'Exit' in the file menu and the calculate button, but neither of them do anything. We need to give them functionality, which in Tkinter is laughably simple.

```
1 # Inside the Adder class but not inside init_gui()
2 def on_quit(self):
3     """Exits program."""
4     quit()
5
6 def calculate(self):
7     """Calculates the sum of the two inputted numbers."""
8     num1 = int(self.num1_entry.get())
9     num2 = int(self.num2_entry.get())
10    num3 = num1 + num2
11    self.answer_label['text'] = num3
```

First what we need to do is create methods inside our Adder class (our Frame class) to take care of the functionality that we need.

```
1 # Inside init_gui()
2 self.menu_file.add_command(label='Exit', command=self.on_quit)
3
4 self.calc_button = ttk.Button(self, text='Calculate',
5     command=self.calculate)
```

Then all we do is add the 'command' argument to our exit button in the file menu and our calculate button and point them to the methods we created. Simple.

A FULL EXAMPLE

```
1 """
2 adder.py
3 ~~~~~
4
5 Creates a simple GUI for summing two numbers.
6 """
7
8 import tkinter
9 from tkinter import ttk
10
11 class Adder(ttk.Frame):
12     """The adders gui and functions."""
13     def __init__(self, parent, *args, **kwargs):
14         ttk.Frame.__init__(self, parent, *args, **kwargs)
15         self.root = parent
16         self.init_gui()
17
18     def on_quit(self):
19         """Exits program."""
20         quit()
21
22     def calculate(self):
23         """Calculates the sum of the two inputted numbers."""
24         num1 = int(self.num1_entry.get())
25         num2 = int(self.num2_entry.get())
26         num3 = num1 + num2
27         self.answer_label['text'] = num3
28
29     def init_gui(self):
30         """Builds GUI."""
31         self.root.title('Number Adder')
32         self.root.option_add('*tearOff', 'FALSE')
33
34         self.grid(column=0, row=0, sticky='nsew')
35
36         self.menubar = tkinter.Menu(self.root)
37
38         self.menu_file = tkinter.Menu(self.menubar)
39         self.menu_file.add_command(label='Exit', command=self.on_quit)
40
41         self.menu_edit = tkinter.Menu(self.menubar)
42
43         self.menubar.add_cascade(menu=self.menu_file, label='File')
44         self.menubar.add_cascade(menu=self.menu_edit, label='Edit')
45
46         self.root.config(menu=self.menubar)
47
48         self.num1_entry = ttk.Entry(self, width=5)
49         self.num1_entry.grid(column=1, row=2)
50
51         self.num2_entry = ttk.Entry(self, width=5)
52         self.num2_entry.grid(column=3, row=2)
53
54         self.calc_button = ttk.Button(self, text='Calculate',
55             command=self.calculate)
56         self.calc_button.grid(column=0, row=3, columnspan=4)
57
58         self.answer_frame = ttk.LabelFrame(self, text='Answer',
59             height=100)
```

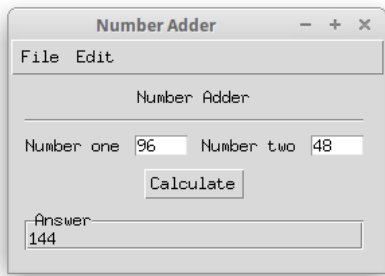
SHARES

```

62     self.answer_label = ttk.Label(self.answer_frame, text='')
63     self.answer_label.grid(column=0, row=0)
64
65     # Labels that remain constant throughout execution.
66     ttk.Label(self, text='Number Adder').grid(column=0, row=0,
67         columnspan=4)
68     ttk.Label(self, text='Number one').grid(column=0, row=2,
69         sticky='w')
70     ttk.Label(self, text='Number two').grid(column=2, row=2,
71         sticky='w')
72
73     ttk.Separator(self, orient='horizontal').grid(column=0,
74         row=1, columnspan=4, sticky='ew')
75
76     for child in self.winfo_children():
77         child.grid_configure(padx=5, pady=5)
78
79 if __name__ == '__main__':
80     root = tkinter.Tk()
81     Adder(root)
82     root.mainloop()

```

The above code should produce this GUI that calculates the sum of the two input numbers and exits when you press the exit button in the file menu:



With this we can now create complex GUIs in python 3.5 using tkinter. Hopefully in the near future other GUI tool kits will be ported forward, giving us more choice.

That's it for this post, I hope now you feel a little more confident producing GUIs for your applications written in Python 3.5. Make sure you give this post a share so others can read it too.

Don't forget to share and follow!

Remember, don't forget to share this post so that other people can see it too! Also, make sure you [subscribe to this blog's mailing list](#), follow me on [Twitter](#) and add me on [Google+](#) so that you don't miss out on any useful posts!

I read all comments, so if you have something to say, something to share or questions and the like, leave a comment below!



Share this:



Related

[How to Use Google's Python Client Library to Authorise Your Desktop Application With OAuth 2.0](#)
March 11, 2016
In "Programming"

[\[Video Series\] Taking Your Python Skills to the Next Level With Pythonic Code - Introduction](#)
July 22, 2016
In "Programming"

[How to Write Beautiful Code with Pythonic Idioms | Notable Others](#)
February 23, 2016
In "Programming"

FILED UNDER: PROGRAMMING
TAGGED WITH: GUI, PYTHON, TKINTER

SHARES

10 Comments Data Dependence

 Login Recommend  Share

Sort by Best



Join the discussion...

**Opaque** • a year ago

You can get a version of wxpython phoenix (<https://github.com/wxWidget...> working with 64bit python 3.5.1 by installing a wheel from <http://wxpython.org/Phoenix...> I've been using it for the last few months without problems.

1 ^ | v • Reply • Share ›

**Jamal** Mod → Opaque • a year ago

Oh fantastic! Thank you for sharing this. I'll update this post right away.

^ | v • Reply • Share ›

**Pooja Bhalode** • a month ago

Hi, I had a question regarding creating an explorer bar in Tkinter window. I am creating an application and was wondering if there is a way to display the explorer tab in the LHS side of the root window in Tkinter?

Please let me know.

^ | v • Reply • Share ›

**Jamal** Mod → Pooja Bhalode • a month ago

Sorry, what do you mean by an explorer bar?

^ | v • Reply • Share ›

**Matthew Lauria** • a year ago

As much as I too despise the `from blah import *` technique, tkinter is the one exception I make as it makes your code that much more readable (e.g. `grid(sticky=W)` looks way better than `grid(sticky=tkinter.W)`) and practicality beats purity

^ | v • Reply • Share ›

**Jamal** Mod → Matthew Lauria • a year ago

I would still argue that it's best not to do it when thinking about the benefits not doing it brings and the drawbacks of doing it. Also instead of `sticky=tkinter.W`, you can just do `sticky='w'`.

But yes, typing that would be a little messy, but it still helps people know, where everything came from.

^ | v • Reply • Share ›

**Todd Fiske** → Jamal • a year ago

The middle-ground solution is to do: `import tkinter as tk`

Then your constants will look like: `grid(sticky=tk.W)`

They will be scoped and made more explicit than just string constants without too much extra typing.

^ | v • Reply • Share ›

**Jamal** Mod → Todd Fiske • a year ago

Yeah, I've seen quite a few people do that. That's definitely better than the `import *` in my opinion.

^ | v • Reply • Share ›

**yedpodtrzitko** • a year ago

I don't see Toga mentioned anywhere, I think it would be definitely worth trying:

<https://github.com/pybee/toga>

^ | v • Reply • Share ›

**Jamal** Mod → yedpodtrzitko • a year ago

Oh this looks interesting! I've actually never heard of it before, but it looks promising. It states that its windows support is iffy at the moment though, so might not be best for everyone.

As a linux user though this is something I will definitely have to look into. Thanks.

SHARES

Search this website ...

FOLLOW ME

SUBSCRIBE

Don't miss out on any useful posts. Subscribe!


Email Address

SUBSCRIBE


POPULAR POSTS

 [An Introduction to Scientific Python – Pandas](#)

 [How to Build a GUI in Python 3.5](#)

 [Creating Attractive and Informative Map Visualisations in Python with Basemap](#)

 [An Introduction to Scientific Python – Matplotlib](#)

 [Write Pythonic Code Like a Seasoned Developer: The Course Everyone New to Python Desperately Needs to Take](#)

Copyright © 2017 · Metro Pro Theme on Genesis Framework · WordPress · [Log in](#)

SHARES