Python Crash Course: GUIs with Tkinter

Do you train for passing tests
or do you train for creative inquiry?
Noam Chomsky

Steffen Brinkmann

Max-Planck-Institut für Astronomie, Heidelberg

IMPRESS, 2016

The ingredients

- A working Python installation
- Internet connection
- Passion for Python

If anything of the above is missing, please say so now!

Outline

Motivation

- To GUI or not to GUI

Getting started

- Terminology
- Hello World

The Widgets

- Overview
- Widget Properties and Placement

Putting it together

- More complexity

Outline

Motivation
To GUI or not to GUI

Getting started
Terminology
Hello World

The Widgets
Overview
Widget Properties and Placement

Putting it together
More complexity

Why use GUIs?

- Users are used to GUIs
- Easy interaction
- Display images, animations etc.
- Visualise workflow
- Display complex Data

Why not use GUIs?

- Faster development
- Faster execution
- Easier to test
- Can be included in scripts (pipelines)
- less dependences

Why Tkinter?

- Tkinter is the standard Python interface to the Tk GUI toolkit.
- No need to install additional packages
- Simple enough to get quick results
- Powerful enough for most applications
- Alternatives: gi (GTK), pyside (Qt), wx (wxWidgets)

Get help

- http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html
- http://www.tutorialspoint.com/python/python_gui_programming.htm

Outline

Motivation
To GUI or not to GUI

Getting started
Terminology
Hello World

The Widgets
Overview
Widget Properties and Placement

Putting it together
More complexity

Terminology

- **Widget:** Visual element of interaction. Can be a container (see below), control (Button, slider, . . .), a display (label, canvas,. . .) or a combination of both (Text field, spin box)
- **Box, Container:** Widget that can contain one or more other widgets.

Terminology

- **Event:** Events are created by the program itself (destroy, resize, . . .) or the user via controls (click button, mouse move,. . .).
- **Event handler:** Function or method dedicated to react to events. Typically called on_*event*.
- **Event loop:** separate execution thread or process which checks for new events.

Outline

Motivation
To GUI or not to GUI

Getting started
Terminology
Hello World

The Widgets
Overview
Widget Properties and Placement

Putting it together
More complexity

Very simple

```
#!/bin/env python

from __future__ import print_function
import sys

if sys.version_info >= (3,):
    import tkinter as tk
else:
    import Tkinter as tk

top = tk.Tk()

top.mainloop()
```

- The Tkinter module is called differently in Python 2 and 3
- The class `Tk` represents the Tkinter framework and is necessary for the event loop, exiting cleanly etc.
- The function `mainloop` starts the event loop. It stops when `Tk().destroy` is called.

A little less simple

```
class Hello(tk.Frame):
    def __init__(self, master=tk.Tk()):
        tk.Frame.__init__(self, master)
        self.master=master
        self.grid()
        self.createWidgets()

    def createWidgets(self):
        self.button1 = tk.Button(self,
                                 text="click me",
                                 command=self.say_hello)
        self.button1.grid(row=10, column=10)

        self.button_quit = tk.Button(self,
                                     text="QUIT",
                                     fg="red",
                                     command=self.master.destroy)
        self.button_quit.grid(row=20, column=10)

    def say_hello(self):
        print("hello there, everyone!")

app = Hello()
app.mainloop()
```

A little less simple

- It is convenient to create an application class, derived from `tk.Frame`
- Widgets can be added to the application window by `grid()`, `pack()` or `place()`.

Outline

Motivation
To GUI or not to GUI

Getting started
Terminology
Hello World

The Widgets
Overview
Widget Properties and Placement

Putting it together
More complexity

Overview

Frame and Buttons

Frame The Frame widget is used as a container widget to organize other widgets.

Button The Button widget is used to display buttons in your application.

Checkbutton The Checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time.

Radiobutton The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time.

---

http://www.tutorialspoint.com/python/python_gui_programming.htm

Overview

Text and Displays

Label The Label widget is used to provide a single-line caption for other widgets. It can also contain images.

Message The Message widget is used to display multiline text fields.

Entry The Entry widget is used to display a single-line text field for accepting values from a user.

Spinbox The Spinbox widget is a variant of the standard Tkinter Entry widget, which can be used to select from a fixed number of values.

Text The Text widget is used to display text in multiple lines.

Canvas The Canvas widget is used to draw shapes, such as lines, ovals, polygons and rectangles, in your application.

---

http://www.tutorialspoint.com/python/python_gui_programming.htm

Overview

Menus and more choices and controls

Menubutton The Menubutton widget is used to display menus in your application.

Menu The Menu widget is used to provide various commands to a user.These commands can be contained inside Menubutton or set as the main menu.

Listbox The Listbox widget is used to provide a list of options to a user.

Scale The Scale widget is used to provide a slider widget.

Scrollbar The Scrollbar widget is used to add scrolling

---

http://www.tutorialspoint.com/python/python_gui_programming.htm

Overview

Containers

Toplevel The Toplevel widget is used to provide a separate window container.

PanedWindow A PanedWindow is a container widget that may contain any number of panes, arranged horizontally or vertically.

LabelFrame A labelframe is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts.

---

http://www.tutorialspoint.com/python/python_gui_programming.htm

Overview

Dialogues

tkMessageBox This module is used to display message boxes in your applications.

tkFileDialog This module provides two different pop-up windows you can use to give the user the ability to find existing files or create new files.

tkColorChooser Module featuring a dialogue to select a colour.

---

http://www.tutorialspoint.com/python/python_gui_programming.htm

Outline

Motivation
To GUI or not to GUI

Getting started
Terminology
Hello World

The Widgets
Overview
Widget Properties and Placement

Putting it together
More complexity

Setting Widget Properties

During Initialisation

```
button = tk.Button(parent,
                   text='button label',
                   command=self.on_button)
```

- The only positional argument `parent` specifies in whose grid the widget will be displayed.
- Options are set as keyword arguments: `option=value`

Setting Widget Properties

Using `config()`

```
button = tk.Button(parent)
button.config(text='button label')
button.config(command=self.on_button)
```

- `parent` still has to be set when constructing the widget.
- Options are set as keyword arguments in method `config`: `config(option=value)`
- calling `config()` without arguments returns a dictionary of all the widget's current options.
- The method `cget(option)` returns the value of a single option.

Setting Widget Properties

Using Widget as Dictionary

```
button = tk.Button(parent)
button['text'] = 'button label'
button['command'] = self.on_button
```

- `parent` still has to be set when constructing the widget.
- Options are set as values using the string `'option'` as key.

Postioning Widgets

Overview

- `grid()`: Arrange widget on e rectangular grid, i.e. in rows and columns. Recommended.
- `pack()`: Arrange widgets by stacking them vertically or horizontally. Very limited.
- `place()` Place widgets at specific coordinates. Most (too much) freedom.

Postioning Widgets

```
class Hello(tk.Frame):
    def __init__(self, master=tk.Tk()):
        tk.Frame.__init__(self, master)
        self.master=master
        self.grid()
        self.createWidgets()

    def createWidgets(self):
        self.button1 = tk.Button(self,
                                 text="click me",
                                 command=self.say_hello)
        self.button1.grid(row=10, column=10)

        self.button_quit = tk.Button(self,
                                     text="QUIT",
                                     fg="red",
                                     command=self.master.destroy)
        self.button_quit.grid(row=20, column=10)

    def say_hello(self):
        print("hello there, everyone!")

app = Hello()
app.mainloop()
```

Postioning Widgets

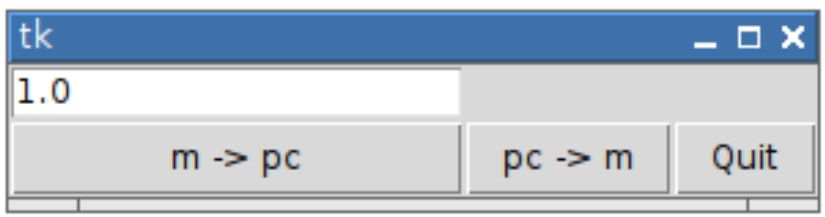
grid()

- `grid()` places the widget in tk's geometry manager.
- Use grid in the top level widget (e.g. `Frame` or other container) to initialise the grid.
- Call grid for every widget with the keywords `row` and `column`.
- Cover more rows and/or columns: `columnspan`, `rowspan`
- Use `sticky` argument to align (e.g. `tk.E`, `tk.NW`) or stretch (e.g. `tk.N+tk.S`) the widget.

Exercise

Conversion tool

Write a program that converts meter to parsec and vice versa:



- Hint: You will need `tk.Entry` and `tk.StringVar`.

Outline

Motivation
To GUI or not to GUI

Getting started
Terminology
Hello World

The Widgets
Overview
Widget Properties and Placement

Putting it together
More complexity

Control Variables

- Control variables can be
  - `tk.StringVar`
  - `tk.IntVar`
  - `tk.DoubleVar`
- Get and set values with `get()` and `set()`.
- Use them to store the Data independently from the visual representation. I.e. Keep **model** and **view** separated.

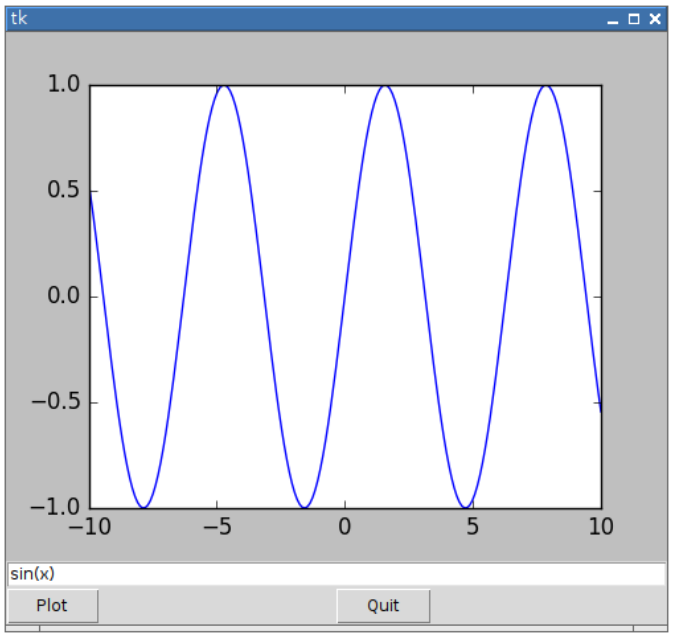
Matplotlib in tkinter

- For simple plot, `tk.Canvas` may suffice
- For displaying Matplotlib plots in tkinter programs, use `FigureCanvasTkAgg`:

```
import matplotlib
matplotlib.use('TkAgg')
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
```

- Use `FigureCanvasTkAgg` as Canvas

Matplotlib in tkinter



Matplotlib in tkinter

```
from numpy import *

import matplotlib
matplotlib.use('TkAgg')

from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure

class Fitting(tk.Frame):
    def __init__(self, master=tk.Tk()):
        tk.Frame.__init__(self, master)
        self.master=master
        self.grid()
        self.createVariables()
        self.createWidgets()

    def createVariables(self):
        self.e_input_text = tk.StringVar()
        self.e_input_text.set('sin(x)')
```

...

Matplotlib in tkinter

```
    def createWidgets(self):
        # matplotlib FigureCanvasTkAgg
        f = Figure(figsize=(5, 4), dpi=100)
        self.sp = f.add_subplot(111)
        self.x = linspace(-10,10,200)
        x = self.x
        y = eval(self.e_input_text.get())
        self.sp.plot(x, y)
        self.canvas = FigureCanvasTkAgg(f, master=self)
        self.canvas.show()
        self.canvas.get_tk_widget().grid(row = 2, column=10,
                                         columnspan=20, sticky=tk.NSEW)
        # entry
        self.e_input = tk.Entry(self, textvariable=self.e_input_text)
        self.e_input.grid(row=5, column=10, columnspan=20, sticky=tk.EW)
        # buttons
        self.b_plot = tk.Button(self, text="Plot", command=self.on_plot)
        self.b_plot.grid(row=10, column=10, sticky=tk.EW)
        self.b_quit = tk.Button(self, text="Quit", command=self.master.destroy)
        self.b_quit.grid(row=10, column=20, sticky=tk.EW)

    def on_plot(self):
        x = self.x
        y = eval(self.e_input_text.get())
        self.sp.plot(x, y)
        self.canvas.show()

app = Fitting()
app.mainloop()
```

Loading files conveniently

- Let the user choose the file to load using the `tkFileDialog`

```
if sys.version_info >= (3,):
        from  tkinter.filedialog import askopenfilename
else:
        from tkFileDialog import askopenfilename
filename_with_path = askopenfilename()
data = np.loadtxt(filename_with_path, unpack=True)
```

- To let the user choose a filename for saving data (i.e. creating a file) use `asksaveasfilename()`

Menus in tkinter

- Create menus with `tk.Menu`
- Add items to the menu using `add_command()`
- Add a submenu using `add_cascade()`.
- Add a separator using `add_separator()`.
- Finally set the menu as the programs main menu with `master.config(menu=themenu)`

Menus in tkinter

```
class JustMenu(tk.Frame):
    def __init__(self, master=tk.Tk()):
        tk.Frame.__init__(self, master)
        self.master=master
        self.config(width=300)
        self.config(height=200)
        self.grid()
        self.createMenu()

    def createMenu(self):
        menubar = tk.Menu(self.master)
        filemenu = tk.Menu(menubar, tearoff=1)
        filemenu.add_command(label="New", command=self.donothing, underline=0)
        filemenu.add_command(label="Open", command=self.donothing)
        filemenu.add_command(label="Save", command=self.donothing)
        filemenu.add_command(label="Save as...", command=self.donothing)
        filemenu.add_command(label="Close", command=self.donothing)
        filemenu.add_separator()
        filemenu.add_command(label="Exit", command=self.master.quit)
        menubar.add_cascade(label="File", menu=filemenu, underline=0)

        self.master.config(menu=menubar)

    def donothing(self):
            print('do nothing')

app = JustMenu()
app.mainloop()
```

Exercise

- Download the data file: www.mpia.da/~brinkmann/PythonCrash/fitting.dat
- Write a program that implements a button to load and display the data in a plot.
- Extend this program to feature three slide controls (`tk.Scale`), which control a, b and c in this function

```
def model(x,a,b,c):
    return a*x**2 + b*x + c
```

- Plot the parabola in the same axes as the data. It should update whenever the slide controls are changed.

For Further Reading I


J. E. Grayson
Python and Tkinter Programming.
Manning Publications, 2000.


B. Chaudhary
Tkinter GUI Application Development.
Packt Publishing, 2013.