Quick answers to common problems

Python GUI Programming Cookbook

Over 80 object-oriented recipes to help you create mind-blowing GUIs in Python

Burkhard Meier

[PACKT] open source* community experience distilled PUBLISHING

Python GUI Programming Cookbook

Over 80 object-oriented recipes to help you create mind-blowing GUIs in Python

Burkhard A. Meier

[PACKT] PUBLISHING open source community experience distilled

BIRMINGHAM - MUMBAI

Python GUI Programming Cookbook

Copyright © 2015 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: November 2015

Production reference: 1261115

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78528-375-8

www.packtpub.com

Credits

Author
Burkhard A. Meier

Reviewers
Joy Bindroo
Peter Bouda
Joseph Rex

Commissioning Editor
Nadeem Bagban

Acquisition Editor
Vivek Anantharaman

Content Development Editor
Sumeet Sawant

Technical Editor
Pramod Kumavat

Copy Editor
Janbal Dharmaraj

Project Coordinator
Shweta H. Birwatkar

Proofreader
Safis Editing

Indexer
Priya Sane

Graphics
Kirk Dpenha

Production Coordinator
Komal Ramchandani

Cover Work
Komal Ramchandani

About the Author

Burkhard A. Meier has more than 15 years of professional experience working in the software industry as a software tester and developer, specializing in software test automation development, execution, and analysis. He has a very strong background in SQL relational database administration, the development of stored procedures, and debugging code.

While experienced in Visual Studio .NET C#, Visual Test, TestComplete, and other testing languages (such as C/C++), the main focus of the author over the past two years has been developing test automation written in Python 3 to test the leading edge of FLIR ONE infrared cameras for iPhone and Android smart phones as well as handheld tablets.

Being highly appreciative of art, beauty, and programming, the author developed GUIs in C# and Python to streamline everyday test automation tasks, enabling these automated tests to run unattended for weeks, collecting very useful data to be analyzed and automatically plotted into graphs and e-mailed to upper management upon completion of nightly automated test runs.

His previous jobs include working as a senior test automation engineer and designer for InfoGenesis (now Agilysys), QAD, InTouch Health, and presently, FLIR Systems.

You can get in touch with him through his LinkedIn account, `https://www.linkedin.com/pub/burkhard-meier/5/246/296`.

I would like to thank all truly great artists such as Leonardo Da Vinci, Charles Baudelaire, Edgar Poe, and so many more for bringing the presence of beauty into our human lives. This book is about creating very beautiful GUIs written in the Python programming language, and it was inspired by these truly great artists.

I would like to thank all of the great people that made this book possible. Without any of you, this book would only exist in my mind. I would like to especially thank all of my editors at Packt Publishing: Vivek, Arwa, Sumeet, Pramod, Nikhil and so many more. I would also like to thank all of the reviewers of the code of this book. Without them this book would be harder to read and apply to real-world problems. Last but not least, I like to thank my wife, our daughter, and our parents for the emotional support they provided so successfully during the writing of this book. I also like to give thanks to the creator of this very beautiful and powerful programming language that Python truly is. Thank you Guido.

About the Reviewers

Joy Bindroo holds a bachelor's degree in computer science and engineering. He is currently pursuing his post-graduate studies in the field of information management. He is a creative person and enjoys working on Linux platform and other open source technologies. He enjoys writing about Python and sharing his ideas and skills on his website, `http://www.joybindroo.com/`. He likes to sketch, write poems, listen to music, and have fun with his friends in his free time.

> I would like to thank my family, teachers, and friends for always encouraging and supporting me. I'm most thankful to Lord Shiva who enables me to achieve greater heights.

Peter Bouda works as a senior web developer for MAJ Digital and is a specialist in full stack JavaScript applications based on LoopBack and AngularJS. He develops Python GUIs for companies and research projects since 2003 and wrote a German book, *PyQt und PySide – GUI- und Anwendungsentwicklung mit Python und Qt*, on Python GUI development, which was published in 2012. Currently, he is getting crazy with embedded and open hardware platforms and is working on a modular game console based on open hardware.

Joseph Rex is a full stack developer with a background in computer security. He has worked on Python GUI products and some CLI programs to experiment with information security. He came out of security to web development and developed a passion for rails and JavaScript MVC frameworks after working on several projects using jQuery. He has been in the web industry for 3 years, building web applications and mobile apps. He has also written articles on security for InfoSec Institute and has written some scripts to back them up. He has to his credit several personal experimental projects written in Python.

www.PacktPub.com

Support files, eBooks, discount offers, and more

For support files and downloads related to your book, please visit `www.PacktPub.com`.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

PACKTLiB™

`https://www2.packtpub.com/books/subscription/packtlib`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Free Access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view 9 entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface v

Chapter 1: Creating the GUI Form and Adding Widgets 1

- Introduction 1
- Creating our first Python GUI 2
- Preventing the GUI from being resized 4
- Adding a label to the GUI form 6
- Creating buttons and changing their text property 7
- Text box widgets 9
- Setting the focus to a widget and disabling widgets 11
- Combo box widgets 12
- Creating a check button with different initial states 14
- Using radio button widgets 16
- Using scrolled text widgets 18
- Adding several widgets in a loop 20

Chapter 2: Layout Management 23

- Introduction 23
- Arranging several labels within a label frame widget 24
- Using padding to add space around widgets 26
- How widgets dynamically expand the GUI 28
- Aligning the GUI widgets by embedding frames within frames 32
- Creating menu bars 36
- Creating tabbed widgets 41
- Using the grid layout manager 46

Chapter 3: Look and Feel Customization 49
Introduction 49
Creating message boxes – information, warning, and error 50
How to create independent message boxes 53
How to create the title of a tkinter window form 56
Changing the icon of the main root window 57
Using a spin box control 58
Relief, sunken, and raised appearance of widgets 61
Creating tooltips using Python 63
How to use the canvas widget 67
Chapter 4: Data and Classes 69
Introduction 69
How to use StringVar() 69
How to get data from a widget 73
Using module-level global variables 75
How coding in classes can improve the GUI 79
Writing callback functions 85
Creating reusable GUI components 86
Chapter 5: Matplotlib Charts 91
Introduction 91
Creating beautiful charts using Matplotlib 92
Matplotlib – downloading modules using pip 94
Matplotlib – downloading modules with whl extensions 98
Creating our first chart 100
Placing labels on charts 102
How to give the chart a legend 106
Scaling charts 109
Adjusting the scale of charts dynamically 112
Chapter 6: Threads and Networking 117
Introduction 117
How to create multiple threads 118
Starting a thread 121
Stopping a thread 125
How to use queues 128
Passing queues among different modules 133
Using dialog widgets to copy files to your network 136
Using TCP/IP to communicate via networks 145
Using URLOpen to read data from websites 147

Chapter 7: Storing Data in Our MySQL Database via Our GUI 153
Introduction 153
Connecting to a MySQL database from Python 154
Configuring the MySQL connection 157
Designing the Python GUI database 161
Using the SQL INSERT command 168
Using the SQL UPDATE command 172
Using the SQL DELETE command 177
Storing and retrieving data from our MySQL database 181
Chapter 8: Internationalization and Testing 187
Introduction 187
Displaying widget text in different languages 188
Changing the entire GUI language all at once 191
Localizing the GUI 196
Preparing the GUI for internationalization 201
How to design a GUI in an agile fashion 204
Do we need to test the GUI code? 208
Setting debug watches 212
Configuring different debug output levels 216
Creating self-testing code using
Python's __main__ section 220
Creating robust GUIs using unit tests 224
How to write unit tests using the Eclipse PyDev IDE 229
Chapter 9: Extending Our GUI with the wxPython Library 235
Introduction 235
How to install the wxPython library 236
How to create our GUI in wxPython 239
Quickly adding controls using wxPython 244
Trying to embed a main wxPython app in a main tkinter app 251
Trying to embed our tkinter GUI code into wxPython 253
How to use Python to control two different GUI frameworks 256
How to communicate between the two connected GUIs 260
Chapter 10: Creating Amazing 3D GUIs with PyOpenGL and PyGLet 265
Introduction 265
PyOpenGL transforms our GUI 266
Our GUI in 3D! 270
Using bitmaps to make our GUI pretty 275
PyGLet transforms our GUI more easily than PyOpenGL 279
Our GUI in amazing colors 283
Creating a slideshow using tkinter 286

Chapter 11: Best Practices 291
Introduction 291
Avoiding spaghetti code 291
Using __init__ to connect modules 298
Mixing fall-down and OOP coding 305
Using a code naming convention 310
When not to use OOP 314
How to use design patterns successfully 317
Avoiding complexity 320
Index 327

Preface

In this book, we will explore the beautiful world of graphical user interfaces (GUIs) using the Python programming language.

Along the way, we will talk to networks, queues, the OpenGL graphical library, and many more technologies.

This is a programming cookbook. Every chapter is self-contained and explains a certain programming solution.

We will start very simply, yet throughout this book we will build a working program written in Python 3.

We will also apply some design patterns and use best practices throughout this book.

The book assumes that the reader has some basic experience using the Python programming language, but that is not really required to use this book.

If you are an experienced programmer in any programming language, you will have a fun time extending your skills to programming GUIs using Python!

Are you ready?

Let's start on our journey…

What this book covers

Chapter 1, *Creating the GUI Form and Adding Widgets*, explains the steps to develop our first GUI in Python. We will start with the minimum code required to build a running GUI application. Each recipe then adds different widgets to the GUI form.

Chapter 2, *Layout Management*, explores how to arrange widgets to create our Python GUI. The grid layout manager is one of the most important layout tools built into tkinter that we will be using.

Chapter 3, *Look and Feel Customization*, shows several examples of how to create a good "look and feel" GUI. On a practical level, we will add functionality to the **Help** | **About** menu item we created in one of the recipes.

Chapter 4, *Data and Classes*, discusses saving the data our GUI displays. We will start using object-oriented programming (OOP) in order to extend Python's built-in functionality.

Chapter 5, *Matplotlib Charts*, explains how to create beautiful charts that visually represent data. Depending upon the format of the data source, we can plot one or several columns of data within the same chart.

Chapter 6, *Threads and Networking*, explains how to extend the functionality of our Python GUI using threads, queues, and network connections. This will show us that our GUI is not limited at all to the local scope of our PC.

Chapter 7, *Storing Data in Our MySQL Database via Our GUI*, shows us how to connect to a MySQL database server. The first recipe in this chapter will show how to install the free MySQL Server Community Edition, and in the following recipes we will create databases, tables, and then load data into those tables as well as modify these data. We will also read the data back out from the MySQL server into our GUI.

Chapter 8, *Internationalization and Testing*, shows how to internationalize our GUI by displaying text on labels, buttons, tabs, and other widgets in different languages. We will start simple and then explore how we can prepare our GUI for internationalization at the design level. We will also explore several ways to automatically test our GUI using Python's built-in unit testing framework.

Chapter 9, *Extending Our GUI with the wxPython Library*, introduces another Python GUI toolkit that currently does not ship with Python. It is called wxPython, and we will be using the Phoenix version of wxPython which was designed to work well with Python 3.

Chapter 10, *Creating Amazing 3D GUIs with PyOpenGL and PyGLet*, shows how to transform our GUI by giving it true three-dimensional capabilities. We will use two Python third-party packages. PyOpenGL is a Python binding to the OpenGL standard, which is a graphics library that comes built-in with all major operating systems. This gives the resulting widgets a native look and feel. PyGLet is one such binding that we will explore in this chapter.

Chapter 11, *Best Practices*, explores different best practices that can help us to build our GUI in an efficient way and keep it both maintainable and extendible. Best practices are applicable to any good code and our GUI is no exception to designing and implementing good software practices.

What you need for this book

All required software for this book is available online and is free of charge. This starts with Python 3 itself, and then extends to Python's add-on modules. In order to download any required software, you will need a working Internet connection.

Who this book is for

This book is for programmers who wish to create a graphical user interface (GUI). You might be surprised by what we can achieve by creating beautiful, functional, and powerful GUIs using the Python programming language. Python is a wonderful, intuitive programming language, and is very easy to learn.

I like to invite you to start on this journey now. It will be a lot of fun!

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, and user input are shown as follows: "Using Python, we can create our own classes using the `class` keyword instead of the `def` keyword."

A block of code is set as follows:

```
import tkinter as tk     # 1
win = tk.Tk()            # 2
win.title("Python GUI")  # 3
win.mainloop()           # 4
```

Any command-line input or output is written as follows:

```
pip install numpy-1.9.2+mkl-cp34-none-win_amd64.whl
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Next, we will add functionality to the menu items, for example, closing the main window when clicking the **Exit** menu item and displaying a **Help** | **About** dialog."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail `feedback@packtpub.com`, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files from your account at `http://www.packtpub.com` for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

1

Creating the GUI Form and Adding Widgets

In this chapter, we start creating amazing GUIs using Python 3:

- Creating our first Python GUI
- Preventing the GUI from being resized
- Adding a label to the GUI form
- Creating buttons and changing their text property
- Text box widgets
- Setting the focus to a widget and disabling widgets
- Combo box widgets
- Creating a check button with different initial states
- Using radio button widgets
- Using scrolled text widgets
- Adding several widgets in a loop

Introduction

In this chapter, we will develop our first GUI in Python. We start with the minimum code required to build a running GUI application. Each recipe then adds different widgets to the GUI form.

In the first two recipes, we show the entire code, consisting of only a few lines of code. In the following recipes we only show the code to be added to the previous recipes.

By the end of this chapter, we will have created a working GUI application that consists of labels, buttons, text boxes, combo boxes, and check buttons in various states, as well as radio buttons that change the background color of the GUI.

Creating our first Python GUI

Python is a very powerful programming language. It ships with the built-in tkinter module. In only a few lines of code (four, to be precise) we can build our first Python GUI.

Getting ready

To follow this recipe, a working Python development environment is a prerequisite. The IDLE GUI that ships with Python is enough to start. IDLE was built using tkinter!

All the recipes in this book were developed using Python 3.4 on a Windows 7 64-bit OS. They have not been tested on any other configuration. As Python is a cross-platform language, the code from each recipe is expected to run everywhere.

If you are using a Mac, it does come built-in with Python, yet it might be missing some modules such as tkinter, which we will use throughout this book.

We are using Python 3 and the creator of Python intentionally chose not to make it backwards compatible with Python 2.

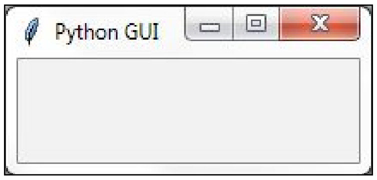
If you are using a Mac or Python 2, you might have to install Python 3 from `www.python.org` in order to successfully run the recipes in this book.

How to do it...

Here are the four lines of Python code required to create the resulting GUI:

```
import tkinter as tk      # 1
win = tk.Tk()             # 2
win.title("Python GUI")   # 3
win.mainloop()            # 4
```

Execute this code and admire the result:



How it works...

In line 1, we import the built-in `tkinter` module and alias it as `tk` to simplify our Python code. In line 2, we create an instance of the `Tk` class by calling its constructor (the parentheses appended to `Tk` turn the class into an instance). We are using the alias `tk` so we don't have to use the longer word `tkinter`. We are assigning the class instance to a variable named `win` (short for a window). As Python is a dynamically typed language, we did not have to declare this variable before assigning to it and we did not have to give it a specific type. *Python infers the type from the assignment of this statement*. Python is a strongly typed language, so every variable always has a type. We just don't have to specify its type beforehand like in other languages. This makes Python a very powerful and productive language to program in.

A little note about classes and types:

In Python every variable always has a type. We cannot create a variable without assigning it a type. Yet, in Python, we do not have to declare the type beforehand, as we have to do in the C programming language.

Python is smart enough to infer the type. At the time of writing, C# also has this capability.

Using Python, we can create our own classes using the `class` keyword instead of the `def` keyword.

In order to assign the class to a variable, we first have to create an instance of our class. We create the instance and assign this instance to our variable.

```
class AClass(object):
    print('Hello from AClass')

classInstance = AClass()
```

Now the variable `classInstance` is of the type `AClass`.

If this sounds confusing, do not worry. We will cover OOP in the coming chapters.

In line 3, we use the instance variable of the class (`win`) to give our window a title via the `title` property. In line 4, we start the window's event loop by calling the `mainloop` method on the class instance `win`. Up to this point in our code, we created an instance and set one property *but the GUI will not be displayed until we start the main event loop*.

An event loop is a mechanism that makes our GUI work. We can think of it as an endless loop where our GUI is waiting for events to be sent to it. A button click creates an event within our GUI or our GUI being resized also creates an event.

We can write all of our GUI code in advance and nothing will be displayed on the user's screen until we call this endless loop (`win.mainloop()` in the code shown above).

The event loop ends when the user clicks the red **X** button or a widget that we have programmed to end our GUI. When the event loop ends, our GUI also ends.

There's more...

This recipe used a minimum amount of Python code to create our first GUI program. However, throughout this book, we will use OOP when it makes sense.

Preventing the GUI from being resized

Getting ready

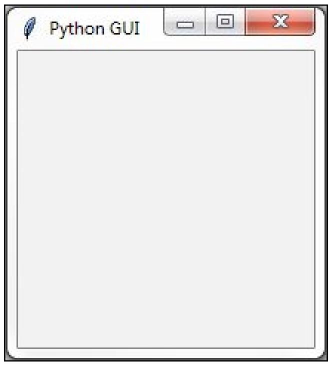
This recipe extends the previous one. Therefore, it is necessary to have typed Recipe 1 yourself into a project of your own or downloaded the code from `https://www.packtpub.com/support`.

How to do it...

We are preventing the GUI from being resized.

```
import tkinter as tk        # 1 imports

win = tk.Tk()               # 2 Create instance
win.title("Python GUI")     # 3 Add a title

win.resizable(0, 0)         # 4 Disable resizing the GUI

win.mainloop()              # 5 Start GUI
```

Running the code creates this GUI:



How it works...

Line 4 prevents the Python GUI from being resized.

Running this code will result in a GUI similar to the one we created in Recipe 1. However, the user can no longer resize it. Also, notice how the maximize button in the toolbar of the window is grayed out.

Why is this important? Because, once we add widgets to our form, resizing can make our GUI look not as good as we want it to be. We will add widgets to our GUI in the next recipes.

`Resizable()` is a method of the `Tk()` class and, by passing in `(0, 0)`, we prevent the GUI from being resized. If we pass in other values, we hard-code the x and y start up size of the GUI, *but that won't make it nonresizable*.

We also added comments to our code in preparation for the recipes contained in this book.

In visual programming IDEs such as Visual Studio .NET, C# programmers often do not think of preventing the user from resizing the GUI they developed in this language. That creates inferior GUIs. Adding this one line of Python code can make our users appreciate our GUI.

Adding a label to the GUI form

Getting ready

We are extending the first recipe. We will leave the GUI resizable, so don't use the code from the second recipe (or comment the `win.resizable` line 4 out).

How to do it...

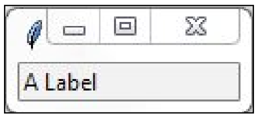
In order to add a `Label` widget to our GUI, we are importing the `ttk` module from `tkinter`. Please note the two import statements.

```
# imports                    # 1
import tkinter as tk         # 2
from tkinter import ttk      # 3
```

Add the following code just above `win.mainloop()` located at the bottom of recipes 1 and 2.

```
# Adding a Label             # 4
ttk.Label(win, text="A Label").grid(column=0, row=0) # 5
```

Running the code adds a label to our GUI:



How it works...

In line 3 of the above code, we are importing a separate module from `tkinter`. The `ttk` module has some advanced widgets that make our GUI look great. In a sense, `ttk` is an extension within `tkinter`.

We still need to import `tkinter` itself, but we have to specify that we now want to also use `ttk` from `tkinter`.

`ttk` stands for 'themed tk". It improves our GUI look and feel.

Line 5 above adds the label to the GUI, just before we call `mainloop` (not shown here to preserve space. See recipes 1 or 2).

We are passing our window instance into the `ttk.Label` constructor and setting the text property. This becomes the text our `Label` will display.

We are also making use of the *grid layout manager*, which we'll explore in much more depth in *Chapter 2*, *Layout Management*.

Note how our GUI suddenly got much smaller than in previous recipes.

The reason why it became so small is that we added a widget to our form. Without a widget, `tkinter` uses a default size. Adding a widget causes optimization, which generally means using as little space as necessary to display the widget(s).

If we make the text of the label longer, the GUI will expand automatically. We will cover this automatic form size adjustment in a later recipe in *Chapter 2*, *Layout Management*.

There's more...

Try resizing and maximizing this GUI with a label and watch what happens.

Creating buttons and changing their text property

Getting ready

This recipe extends the previous one. You can download the entire code from the Packt Publishing website.

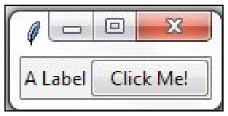
How to do it...

We are adding a button that, when clicked, performs an action. In this recipe, we will update the label we added in the previous recipe, as well as updating the text property of the button.

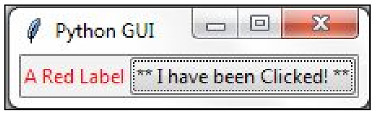
```
# Modify adding a Label                                        # 1
aLabel = ttk.Label(win, text="A Label")                        # 2
aLabel.grid(column=0, row=0)                                   # 3


# Button Click Event Callback Function                         # 4
```

```
def clickMe():                                               # 5
    action.configure(text="** I have been Clicked! **")
    aLabel.configure(foreground='red')

# Adding a Button                                            # 6
action = ttk.Button(win, text="Click Me!", command=clickMe)  # 7
action.grid(column=1, row=0)                                 # 8
```

Before clicking the button:



After clicking the button, the color of the label has been changed, and so has the text of the button. Action!



How it works...

In line 2 we are now assigning the label to a variable and in line 3 we use this variable to position the label within the form. We will need this variable to change its properties in the `clickMe()` function. By default, this is a module-level variable so we can access it inside the function as long as we declare the variable above the function that calls it.

Line 5 is the event handler that is being invoked once the button gets clicked.

In line 7, we create the button and bind the command to the `clickMe()` function.

GUIs are event-driven. Clicking the button creates an event. We bind what happens when this event occurs in the callback function using the command property of the `ttk.Button` widget. Notice how we do not use parentheses; only the name `clickMe`.

We also change the text of the label to include `red` as in the printed book, this might otherwise not be obvious. When you run the code you can see that the color did indeed change.

Lines 3 and 8 both use the grid layout manager, which will be discussed in the following chapter. This aligns both the label and the button.

There's more...

We will continue to add more and more widgets to our GUI and we will make use of many built-in properties in other recipes in the book.

Text box widgets

In `tkinter`, the typical textbox widget is called `Entry`. In this recipe, we will add such an `Entry` to our GUI. We will make our label more useful by describing what the `Entry` is doing for the user.

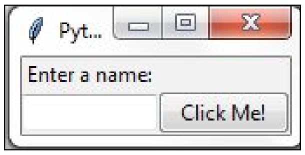
Getting ready

This recipe builds upon the *Creating buttons and changing their text property* recipe.

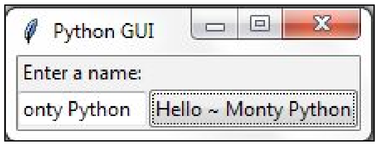
How to do it...

```
# Modified Button Click Function   # 1
def clickMe():                     # 2
    action.configure(text='Hello ' + name.get())

# Position Button in second row, second column (zero-based)
action.grid(column=1, row=1)

# Changing our Label               # 3
ttk.Label(win, text="Enter a name:").grid(column=0, row=0) # 4

# Adding a Textbox Entry widget    # 5
name = tk.StringVar()              # 6
nameEntered = ttk.Entry(win, width=12, textvariable=name) # 7
nameEntered.grid(column=0, row=1)  # 8
```

Now our GUI looks like this:



After entering some text and clicking the button, there is the following change in the GUI:



How it works...

In line 2 we are getting the value of the `Entry` widget. We are not using OOP yet, so how come we can access the value of a variable that was not even declared yet?

Without using OOP classes, in Python procedural coding we have to physically place a name above a statement that tries to use that name. So how come this works (it does)?

The answer is that the button click event is a callback function, and by the time the button is clicked by a user, the variables referenced in this function are known and do exist.

Life is good.

Line 4 gives our label a more meaningful name, because now it describes the textbox below it. We moved the button down next to the label to visually associate the two. We are still using the grid layout manager, to be explained in more detail in *Chapter 2*, *Layout Management*.

Line 6 creates a variable `name`. This variable is bound to the `Entry` and, in our `clickMe()` function, we are able to retrieve the value of the `Entry` box by calling `get()` on this variable. This works like a charm.

Now we see that while the button displays the entire text we entered (and more), the textbox `Entry` widget did not expand. The reason for this is that we had hard-coded it to a width of 12 in line 7.

Python is a dynamically-typed language and infers the type from the assignment. What this means is if we assign a string to the variable `name`, the variable will be of the type string, and if we assign an integer to `name`, this variable's type will be integer.

Using tkinter, we have to declare the variable `name` as the type `tk.StringVar()` before we can use it successfully. The reason is this that Tkinter is not Python. We can use it from Python but it is not the same language.

Setting the focus to a widget and disabling widgets

While our GUI is nicely improving, it would be more convenient and useful to have the cursor appear in the `Entry` widget as soon as the GUI appears. Here we learn how to do this.

Getting ready

This recipe extends the previous recipe.

How to do it...

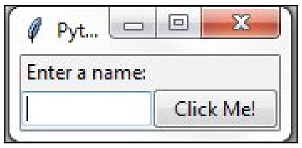
Python is truly great. All we have to do to set the focus to a specific control when the GUI appears is call the `focus()` method on an instance of a `tkinter` widget we previously created. In our current GUI example, we assigned the `ttk.Entry` class instance to a variable we named `nameEntered`. Now we can give it the focus.

Place the following code just above the bottom of the module that starts the main windows event loop, just like in previous recipes. If you get some errors, make sure you are placing calls to variables below the code where they are declared. We are not using OOP as of yet, so this is still necessary. Later, it will no longer be necessary to do this.

```
nameEntered.focus()            # Place cursor into name Entry
```

On a Mac, you might have to set the focus to the GUI window first before being able to set the focus to the `Entry` widget in this window.

Adding this one line of Python code places the cursor into our text `Entry` box, giving the text `Entry` box the focus. As soon as the GUI appears, we can type into this text box without having to click it first.

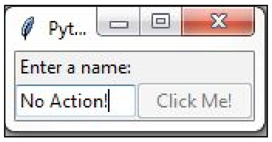


Note how the cursor now defaults to residing inside the text `Entry` box.

We can also disable widgets. To do that, we set a property on the widget. We can make the button disabled by adding this one line of Python code:

```
action.configure(state='disabled')    # Disable the Button Widget
```

After adding the above line of Python code, clicking the button no longer creates any action!



How it works...

This code is self-explanatory. We set the focus to one control and disable another widget. Good naming in programming languages helps to eliminate lengthy explanations. Later in this book, there will be some advanced tips on how to do this while programming at work or practicing our programming skills at home.

There's more...

Yes. This is only the first chapter. There is much more to come.

Combo box widgets

In this recipe, we will improve our GUI by adding drop-down combo boxes that can have initial default values. While we can restrict the user to only certain choices, at the same time, we can allow the user to type in whatever they wish.

Getting ready

This recipe extends the previous recipes.

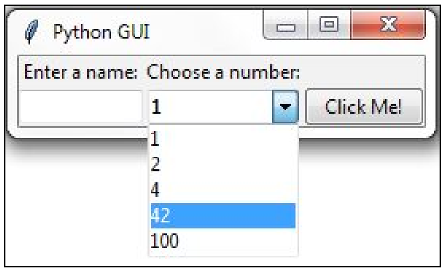
How to do it...

We are inserting another column between the `Entry` widget and the `Button` using the grid layout manager. Here is the Python code.

```
ttk.Label(win, text="Choose a number:").grid(column=1, row=0)  # 1
number = tk.StringVar()                                # 2
numberChosen = ttk.Combobox(win, width=12, textvariable=number) #3
```

```
numberChosen['values'] = (1, 2, 4, 42, 100)      # 4
numberChosen.grid(column=1, row=1)               # 5
numberChosen.current(0)                          # 6
```

This code, when added to previous recipes, creates the following GUI. Note how, in line 4 in the preceding code, we assign a tuple with default values to the combo box. These values then appear in the drop-down box. We can also change them if we like (by typing in different values when the application is running).



How it works...

Line 1 adds a second label to match the newly created combo box (created in line 3). Line 2 assigns the value of the box to a variable of a special `tkinter` type (`StringVar`), as we did in a previous recipe.

Line 5 aligns the two new controls (label and combo box) within our previous GUI layout, and line 6 assigns a default value to be displayed when the GUI first becomes visible. This is the first value of the `numberChosen['values']` tuple, the string `"1"`. We did not place quotes around our tuple of integers in line 4, but they got casted into strings because, in line 2, we declared the values to be of type `tk.StringVar`.

The screenshot shows the selection made by the user (**42**). This value gets assigned to the `number` variable.

There's more...

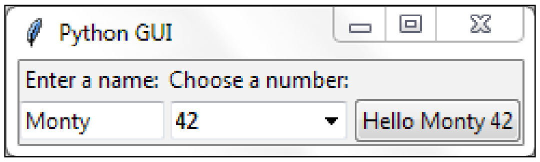
If we want to restrict the user to only be able to select the values we have programmed into the `Combobox`, we can do that by passing the *state property* into the constructor. Modify line 3 in the previous code to:

```
numberChosen = ttk.Combobox(win, width=12, textvariable=number,
state='readonly')
```

Now users can no longer type values into the `Combobox`. We can display the value chosen by the user by adding the following line of code to our Button Click Event Callback function:

```
# Modified Button Click Callback Function

def clickMe():

    action.configure(text='Hello ' + name.get()+ ' ' +
        numberChosen.get())
```

After choosing a number, entering a name, and then clicking the button, we get the following GUI result, which now also displays the number selected:



Creating a check button with different initial states

In this recipe, we will add three `Checkbutton` widgets, each with a different initial state.

Getting ready

This recipe extends the previous recipes.

How to do it...

We are creating three `Checkbutton` widgets that differ in their states. The first is disabled and has a checkmark in it. The user cannot remove this checkmark as the widget is disabled.

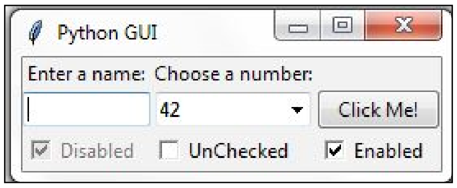
The second `Checkbutton` is enabled and, by default, has no checkmark in it, but the user can click it to add a checkmark.

The third `Checkbutton` is both enabled and checked by default. The users can uncheck and recheck the widget as often as they like.

```
# Creating three checkbuttons    # 1
chVarDis = tk.IntVar()           # 2
check1 = tk.Checkbutton(win, text="Disabled", variable=chVarDis, state
='disabled')                     # 3
check1.select()                  # 4
```

```
check1.grid(column=0, row=4, sticky=tk.W) # 5

chVarUn = tk.IntVar()            # 6
check2 = tk.Checkbutton(win, text="UnChecked", variable=chVarUn)
check2.deselect()                # 8
check2.grid(column=1, row=4, sticky=tk.W) # 9

chVarEn = tk.IntVar()            # 10
check3 = tk.Checkbutton(win, text="Enabled", variable=chVarEn)
check3.select()                  # 12
check3.grid(column=2, row=4, sticky=tk.W) # 13
```

Running the new code results in the following GUI:



How it works...

In lines 2, 6, and 10, we create three variables of type `IntVar`. In the following line, for each of these variables we create a `Checkbutton`, passing in these variables. They will hold the state of the `Checkbutton` (unchecked or checked). By default, that is either 0 (unchecked) or 1 (checked) so the type of the variable is a `tkinter` integer.

We place these `Checkbutton` widgets in our main window so the first argument passed into the constructor is the parent of the widget; in our case `win`. We give each `Checkbutton` a different label via its `text` property.

Setting the sticky property of the grid to `tk.W` means that the widget will be aligned to the west of the grid. This is very similar to Java syntax and it means that it will be aligned to the left. When we resize our GUI, the widget will remain on the left side and not be moved towards the center of the GUI.

Lines 4 and 12 place a checkmark into the `Checkbutton` widget by calling the `select()` method on these two `Checkbutton` class instances.

We continue to arrange our widgets using the grid layout manager, which will be explained in more detail in *Chapter 2*, *Layout Management*.

Using radio button widgets

In this recipe, we will create three `tkinter Radiobutton` widgets. We will also add some code that changes the color of the main form depending upon which `Radiobutton` is selected.

Getting ready

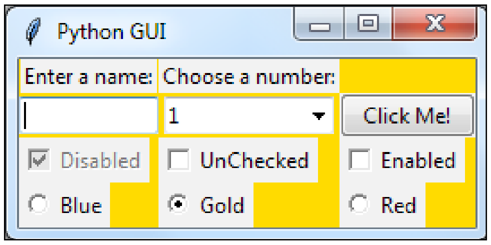
This recipe extends the previous recipes.

How to do it...

We are adding the following code to the previous recipe:

```
# Radiobutton Globals   # 1
COLOR1 = "Blue"         # 2
COLOR2 = "Gold"         # 3
COLOR3 = "Red"          # 4

# Radiobutton Callback  # 5
def radCall():          # 6
   radSel=radVar.get()
   if   radSel == 1: win.configure(background=COLOR1)
   elif radSel == 2: win.configure(background=COLOR2)
   elif radSel == 3: win.configure(background=COLOR3)

# create three Radiobuttons   # 7
radVar = tk.IntVar()          # 8
rad1 = tk.Radiobutton(win, text=COLOR1, variable=radVar, value=1,
command=radCall)              # 9
rad1.grid(column=0, row=5, sticky=tk.W)  # 10

rad2 = tk.Radiobutton(win, text=COLOR2, variable=radVar, value=2, comm
and=radCall)                             # 11
rad2.grid(column=1, row=5, sticky=tk.W)  # 12

rad3 = tk.Radiobutton(win, text=COLOR3, variable=radVar, value=3, comm
and=radCall)                             # 13
rad3.grid(column=2, row=5, sticky=tk.W)  # 14
```

Running this code and selecting the `Radiobutton` named **Gold** creates the following window:



How it works...

In lines 2-4 we create some module-level global variables, which we will use in the creation of each radio button as well as in the callback function that creates the action of changing the background color of the main form (using the instance variable `win`).

We are using global variables to make it easier to change the code. By assigning the name of the color to a variable and using this variable in several places, we can easily experiment with different colors. Instead of doing a global search-and-replace of a hard-coded string (which is prone to errors), we just need to change one line of code and everything else will work. This is known as the **DRY principle**, which stands for **Don't Repeat Yourself**. This is an OOP concept that we will use in later recipes of the book.

> The names of the colors we are assigning to the variables (`COLOR1`, `COLOR2` ...) are `tkinter` keywords (technically, they are *symbolic names*). If we use names that are not `tkinter` color keywords, then the code will not work.

Line 6 is the *callback function* that changes the background of our main form (`win`) depending upon the user's selection.

In line 8 we are creating a `tk.IntVar` variable. What is important about this is that we are creating only one variable to be used by all three radio buttons. As can be seen from the above screenshot, no matter which `Radiobutton` we select, all the others will automatically be unselected for us.

Lines 9 to 14 create the three radio buttons, assign them to the main form, and pass in the variable to be used in the callback function that creates the action of changing the background of our main window.

> While this is the first recipe that changes the color of a widget, quite honestly, it looks a bit ugly. A large portion of the following recipes in this book explain how to make our GUI look truly amazing.

There's more...

Here is a small sample of the available symbolic color names that you can look up at the official tcl manual page:

`http://www.tcl.tk/man/tcl8.5/TkCmd/colors.htm`

Name	Red	Green	Blue
alice blue	240	248	255
AliceBlue	240	248	255
Blue	0	0	255
Gold	255	215	0
Red	255	0	0

Some of the names create the same color, so `alice blue` creates the same color as `AliceBlue`. In this recipe we used the symbolic names `Blue`, `Gold`, and `Red`.

Using scrolled text widgets

`ScrolledText` widgets are much larger than simple `Entry` widgets and span multiple lines. They are widgets like Notepad and wrap lines, automatically enabling vertical scrollbars when the text gets larger than the height of the `ScrolledText` widget.

Getting ready

This recipe extends the previous recipes. You can download the code for each chapter of this book from the Packt Publishing website.

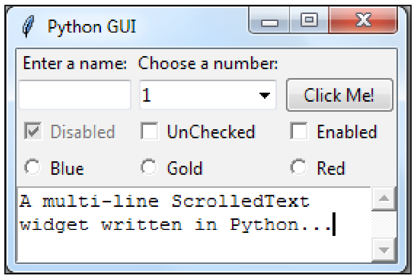
How to do it...

By adding the following lines of code, we create a `ScrolledText` widget:

```
# Add this import to the top of the Python Module    # 1
from tkinter import scrolledtext      # 2

# Using a scrolled Text control       # 3
scrolW  = 30                          # 4
scrolH  =  3                          # 5
```

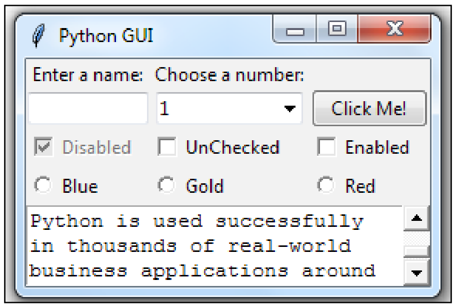
```
scr = scrolledtext.ScrolledText(win, width=scrolW, height=scrolH,
wrap=tk.WORD)                           # 6
scr.grid(column=0, columnspan=3)        # 7
```

We can actually type into our widget, and if we type enough words, the lines will automatically wrap around!



```
A multi-line ScrolledText
widget written in Python...
```

Once we type in more words than the height of the widget can display, the vertical scrollbar becomes enabled. This all works out-of-the-box without us needing to write any more code to achieve this.



```
Python is used successfully
in thousands of real-world
business applications around
```

How it works...

In line 2 we are importing the module that contains the `ScrolledText` widget class. Add that to the top of the module, just below the other two `import` statements.

Lines 4 and 5 define the width and height of the `ScrolledText` widget we are about to create. These are hard-coded values we are passing into the `ScrolledText` widget constructor in line 6.

These values are *magic numbers* found by experimentation to work well. You might experiment by changing `srcolW` from 30 to 50 and observe the effect!