*Python\_Tkinter\_Programming\_c03*

***CHAPTER 3***

*Building an application*

*3.1 Calculator example: key features 21  
3.2 Calculator example: source code 21  
3.3 Examining the application structure 27  
 3.4 Extending the application 28*

*Most books on programming languages have followed Kernigan and Ritchie’s example and have presented the obligatory “Hello World” example to illustrate the ease with which that language may be applied. Books with a GUI component seem to continue this tradition and present a “Hello GUI World” or something similar. Indeed, the three-line example presented on page 13 is in that class of examples.*

*La mayoría de los libros sobre lenguajes de programación han seguido el ejemplo de Kernigan y Ritchie y han presentado el obligatorio ejemplo "Hola mundo" para ilustrar la facilidad con la que se puede aplicar ese lenguaje. Los libros con un componente GUI parecen continuar esta tradición y presentan un “Hola Mundo GUI” o algo similar. De hecho, el ejemplo de tres líneas presentado en la página 13 pertenece a esa clase de ejemplos.*

*There is a growing trend to present a calculator example in recent publications. In this book I am going to start by presenting a simple calculator (you may add the word obligatory, if you wish) in the style of its predecessors. The example has been written to illustrate several Python and Tkinter features and to demonstrate the compact nature of Python code.*

*Existe una tendencia creciente a presentar un ejemplo de calculadora en publicaciones recientes. En este libro voy a comenzar presentando una calculadora sencilla (puede agregar la palabra obligatoria, si lo desea) al estilo de sus predecesoras. El ejemplo ha sido escrito para ilustrar varias características de Python y Tkinter y demostrar la naturaleza compacta del código Python.*

*The example is not complete because it accepts only mouse input; in a full example, we would expect keyboard input as well. However, it does work and it demonstrates that you do not need a lot of code to get a Tkinter screen up and running. Let’s take a look at the code that supports the screen:*

*El ejemplo no está completo porque sólo acepta entradas del mouse; en un ejemplo completo, también esperaríamos entrada del teclado. Sin embargo, funciona y demuestra que no se necesita mucho código para poner en funcionamiento una pantalla de Tkinter. Echemos un vistazo al código que admite la pantalla:*

***calc1.py***

*from Tkinter import \**

*def frame(root, side):*

*w = Frame(root)*

*w.pack(side=side, expand=YES, fill=BOTH)*

*return w*

*def button(root, side, text, command=None):*

*w = Button(root, text=text, command=command)*

*w.pack(side=side, expand=YES, fill=BOTH)*

*return w*

*class Calculator(Frame):*

*def \_\_init\_\_(self):*

*Frame.\_\_init\_\_(self)*

*self.pack(expand=YES, fill=BOTH)*

*self.master.title('Simple Calculator')*

*self.master.iconname("calc1")*

*display = StringVar()*

*Entry(self, relief=SUNKEN,*

*textvariable=display).pack(side=TOP, expand=YES,*

*fill=BOTH)*

*for key in ("123", "456", "789", "-0."):*

*keyF = frame(self, TOP)*

*for char in key:*

*lambda w=display, c=char: w.set(w.get()+' '+c+' '))*

*clearF = frame(self, BOTTOM)*

*button(clearF, LEFT, 'Clr', lambda w=display: w.set(''))*

*button(keyF, LEFT, char,*

*lambda w=display, s=' %s '%char: w.set(w.get()+s))*

*opsF = frame(self, TOP)*

*for char in "+-\*/=":*

*if char == '=':*

*btn = button(opsF, LEFT, char)*

*btn.bind('<ButtonRelease-1>',*

*lambda e, s=self, w=display: s.calc(w), '+')*

*else:*

*btn = button(opsF, LEFT, char,*

*def calc(self, display):*

*try:*

*display.set(`eval(display.get())`)*

*except ValueError:*

*display.set("ERROR")*

*if \_\_name\_\_ == '\_\_main\_\_':*

*Calculator().mainloop()*

***Code comments***

1. *We begin by defining convenience functions to make the creation of frame and button wid- gets more compact. These functions use the pack geometry manager and use generally useful values for widget behavior. It is always a good idea to collect common code in compact func- tions (or classes, as appropriate) since this makes readability and maintenance much easier.*

*Comenzamos definiendo funciones de conveniencia para hacer más compacta la creación de widgets de marcos y botones. Estas funciones utilizan el administrador de geometría del paquete y utilizan valores generalmente útiles para el comportamiento del widget. Siempre es una buena idea recopilar código común en funciones compactas (o clases, según corresponda), ya que esto facilita mucho la legibilidad y el mantenimiento.*

1. *We call the Frame constructor to create the toplevel shell and an enclosing frame. Then, we set titles for the window and icon.*

*Llamamos al constructor Frame para crear el caparazón de nivel superior y un marco envolvente. Luego, configuramos títulos para la ventana y el ícono.*

1. *Next, we create the display at the top of the calculator and define a Tkinter variable which provides access to the widget’s contents:*

*A continuación, creamos la pantalla en la parte superior de la calculadora y definimos una variable Tkinter que proporciona acceso al contenido del widget:*

***display*** *= StringVar()*

*Entry(self.master, relief=SUNKEN,*

*textvariable=variable).pack(side=TOP, expand=YES,*

*fill=BOTH)*

1. *Remember that character strings are sequences of characters in Python, so that each of the subsequences is really an array of characters over which we can iterate:*

*Recuerde que las cadenas de caracteres son secuencias de caracteres en Python, por lo que cada una de las subsecuencias es en realidad una matriz de caracteres sobre la que podemos iterar:*

*for key in ("123", "456", "789", "-0."):*

*keyF = frame(self, TOP)*

*for char in key:*

*We create a frame for each row of keys.*

*Creamos un marco para cada fila de claves.*

1. *We use the convenience function to create a button, passing the frame, pack option, label and callback:*

*button(keyF, LEFT, cahr,*

*lambda w = display, c=char: w.set(w.get() + c))*

*Don’t worry about the lambda form of the callback yet, I will cover this in more detail later. Its purpose is to define an inline function definition.*

1. *The = key has an alternate binding to the other buttons since it calls the calc method when the left mouse button is released:*

*btn.bind('<****ButtonRelease-1****>',*

*lambda e, s=self, w=display: s.****calc****(w))*

1. *The calc method attempts to evaluate the string contained in the display and then it replaces the contents with the calculated value or an ERROR message:*

*display.set(`eval(display.get())`)*

*Personally, I don’t like the calculator, even though it demonstrates compact code and will be quite easy to extend to provide more complete functionality. Perhaps it is the artist in me, but it doesn’t look like a calculator!*

*Let’s take a look at a partly-finished example application which implements a quite sophisticated calculator. It has been left unfinished so that curious readers can experiment by adding functionality to the example (by the time you have finished reading this book, you will be ready to build a Cray Calculator!). Even though the calculator is unfinished, it can still be put to some use. As we will discover a little later, some surprising features are hidden in the reasonably short source code.*

*Let’s start by taking a look at some of the key features of the calculator.*

***3.1 Calculator example: key features***

*The calculator example illustrates many features of applications written in Python and Tkinter, including these:*

* *GUI application structure Although this is a simple example, it contains many of the elements of larger applications that will be presented later in the book.*
* *Multiple inheritance It is simple in this example, but it illustrates how it may be used to simplify Python code.*
* *Lists, dictionaries and tuples As mentioned in chapter 1, these language facilities give Python a con- siderable edge in building concise code. In particular, this example illustrates the use of a dictionary to dis- patch actions to methods. Of particular note is the use of lists of tuples to define the content of each of the keys. Unpacking this data generates each of the keys, labels and associated bindings in a compact fashion.*
* *Pmw (Python megawidgets) The scrolled text widget is implemented with Pmw. This example illustrates set- ting its attributes and gaining access to its components.*
* *Basic Tkinter operations Creating widgets, setting attributes, using text tags, binding events and using a geometry manager are demonstrated.*
* *eval and exec functions The example uses eval to perform many of the math functions in this example. However, as you will see later in this chapter, eval can- not be used to execute arbitrary Python code; exec is used to execute single or multiple lines of code (and multiple lines of code can include control flow structures).*

*3.2 Calculator example: source code*

***calc2.py***

*from Tkinter import \**

*import Pmw*

*1* ***Python MegaWidgets***

*class SLabel(Frame):*

*""" SLabel defines a 2-sided label within a Frame. The*

*left hand label has blue letters; the right has white letters. """*

*def \_\_init\_\_(self, master, leftl, rightl):*

*Frame.\_\_init\_\_(self, master, bg='gray40')*

*self.pack(side=LEFT, expand=YES, fill=BOTH)*

*Label(self, text=leftl, fg='steelblue1',*

*font=("arial", 6, "bold"), width=5, bg='gray40').pack(*

*side=LEFT, expand=YES, fill=BOTH)*

*Label(self, text=rightl, fg='white',*

*font=("arial", 6, "bold"), width=1, bg='gray40').pack(*

*class Key(Button):*

*def \_\_init\_\_(self, master, font=('arial', 8, 'bold'),*

*fg='white',width=5, borderwidth=5, \*\*kw):*

*kw['font'] = font*

*kw['fg'] = fg*

*kw['width'] = width*

*kw['borderwidth'] = borderwidth*

*apply(Button.\_\_init\_\_, (self, master), kw)*

*self.pack(side=LEFT, expand=NO, fill=NONE)*

*class Calculator(Frame):*

*def \_\_init\_\_(self, parent=None):*

*Frame.\_\_init\_\_(self, bg='gray40')*

*self.pack(expand=YES, fill=BOTH)*

*self.master.title('Tkinter Toolkit TT-42')*

*self.master.iconname('Tk-42')*

*self.calc = Evaluator() # This is our evaluator*

*self.buildCalculator() # Build the widgets*

*2*

*# This is an incomplete dictionary - a good exercise! self.actionDict = {'second': self.doThis, 'mode': self.doThis,*

*'delete': self.doThis, 'alpha': self.doThis,*

*'stat': self.doThis, 'math': self.doThis,*

*'matrix': self.doThis, 'program': self.doThis,*

*22*

*CHAPTER 3*

*BUILDING AN APPLICATION*

*'vars': self.doThis, 'clear':*

*'sin': self.doThis, 'cos':*

*'tan': self.doThis, 'up':*

*'X1': self.doThis, 'X2':*

*'log': self.doThis, 'ln':*

*'store': self.doThis, 'off':*

*'neg': self.doThis, 'enter':*

*}*

*self.current = ""*

*def doThis(self,action):*

*print '"%s" has not been implemented' % action*

*def turnoff(self, \*args):*

*self.quit()*

*def clearall(self, \*args):*

*self.current = ""*

*self.display.component('text').delete(1.0, END)*

*def doEnter(self, \*args):*

*self.display.insert(END, '\n')*

*result = self.calc.runpython(self.current)*

*if result:*

*self.clearall,*

*self.display.insert(END, '%s\n' % result, 'ans')*

*self.current = ""*

*4*

*5 6*

*8*

*def doKeypress(self, event):*

*key = event.char*

*if key != '\b':*

*self.current = self.current + key*

*7*

*self.doThis, 3 self.doThis, self.doThis, self.doThis,*

*self.turnoff*

*self.doEnter,*

*lse:*

*self.current = self.current[:-1]*

*def keyAction(self, key):*

*self.display.insert(END, key)*

*self.current = self.current + key*

*def evalAction(self, action):*

*try:*

*self.actionDict[action](action)*

*except KeyError:*

*pass*

***Code comments***

1. *Pmw (Python MegaWidgets) widgets are used. These widgets will feature prominently in this book since they provide an excellent mechanism to support a wide range of GUI requirements and they are readily extended to support additional requirements.*
2. *In the constructor for the Key class, we add key-value pairs to the kw (keyword) dictionary and then apply these values to the Button constructor.  
      
   def \_\_init\_\_(self, master, font=('arial', 8, 'bold'), fg='white',width=5, borderwidth=5,* ***\*\*kw****):* ***kw['font'] = font***

*...* ***apply(Button.\_\_init\_\_, (self, master), kw)***

* *This allows us a great deal of flexibility in constructing our widgets.*

1. *The Calculator class uses a dictionary to provide a dispatcher for methods within the class.  
      
    'matrix': self.doThis, 'program': self.doThis,*

*'vars': self.doThis, 'clear': self.clearall,*

*'sin': self.doThis, 'cos': self.doThis,*

*Remember that dictionaries can handle much more complex references than the rela- tively simple cases we need for this calculator.*

1. *We use a Pmw ScrolledText widget, which is a composite widget. To gain access to the contained widgets, the component method is used.  
      
   self.display.****component****('text').delete(1.0, END)*

* *5 When the ENTER key is clicked, the collected string is directed to the calculator’s evaluator:  
     
  result = self.calc.runpython(self.current)  
     
  The result of this evaluation is displayed in the scrolled text widget.*
* *6 The final argument in the text insert function is a text tag 'ans' which is used to change the foreground color of the displayed text.  
     
  self.display.insert(END, '%s\n' % result,* ***'ans'****)*
* *7 doKeypress is a callback bound to all keys.Theeventargumentinthecallbackprovidesthe client data for the callback. event.char is the key entered; several attributes are available in the client data, such as x-y coordinates of a button press or the state of a mouse operation (see “Tkinter events” on page 98). In this case we get the character entered.*
* *8 A simple exception mechanism to take action on selected keys is used.*

*def buildCalculator(self): FUN =1*

*KEY =0*

*KC1 = 'gray30'*



*KC2 = 'gray50'*



*KC3 = 'steelblue1'*



*KC4 = 'steelblue'*



*keys = [*

*# A Function #AKey*

*# Dark Keys*

*# Light Keys*

*# Light Blue Key # Dark Blue Key*

*9*

*[('2nd', '', '', KC3, FUN, 'second'), # Row 1*

*('Mode', 'Quit', '', KC1, FUN, 'mode'),*

*('Del', 'Ins', '', KC1, FUN, 'delete'),*

*('Alpha','Lock', '', KC2, FUN, 'alpha'),*

*('Stat', 'List', '', KC1, FUN, 'stat')],*

*[('Math', 'Test', 'A', KC1, FUN, 'math'),*

*('Mtrx', 'Angle','B', KC1, FUN, 'matrix'),*

*('Prgm', 'Draw', 'C', KC1, FUN, 'program'),*

*('Vars', 'YVars','', KC1, FUN, 'vars'),*

*('Clr', '', '', KC1, FUN, 'clear')],*

*# Row 2*

*[('X-1', 'Abs', 'D', KC1, FUN, 'X1'),*

*('Sin', 'Sin-1','E', KC1, FUN, 'sin'),*

*('Cos', 'Cos-1','F', KC1, FUN, 'cos'),*

*('Tan', 'Tan-1','G', KC1, FUN, 'tan'),*

*('^', 'PI', 'H', KC1, FUN, 'up')],*

*[('X2', 'Root', 'I', KC1, FUN, 'X2'),*

*(',', 'EE', 'J', KC1, KEY, ','),*

*('(', '{', 'K', KC1, KEY, '('),*

*(')', '}', 'L', KC1, KEY, ')'),*

*('/', '', 'M', KC4, KEY, '/')],*

*[('Log', '10x', 'N', KC1, FUN, 'log'),*

*('7', 'Un-1', 'O', KC2, KEY, '7'),*

*('8', 'Vn-1', 'P', KC2, KEY, '8'),*

*('9', 'n', 'Q', KC2, KEY, '9'),*

*('X', '[', 'R', KC4, KEY, '\*')],*

*[('Ln', 'ex', 'S', KC1, FUN, 'ln'),*

*('4', 'L4', 'T', KC2, KEY, '4'),*

*('5', 'L5', 'U', KC2, KEY, '5'),*

*('6', 'L6', 'V', KC2, KEY, '6'),*

*('-', ']', 'W', KC4, KEY, '-')],*

*# Row 3*

*# Row 4*

*# Row 5*

*# Row 6*

*[('STO', 'RCL', 'X', KC1, FUN, 'store'), # Row 7*

*('1', 'L1', 'Y', KC2, KEY, '1'),*

*('2', 'L2', 'Z', KC2, KEY, '2'),*

*('3', 'L3', '', KC2, KEY, '3'),*

*('+', 'MEM', '"', KC4, KEY, '+')],*

*[('Off', '', '', KC1, FUN, 'off'),*

*# Row 8*

*('0', '', '', KC2, KEY, '0'),*

*('.', ':', '', KC2, KEY, '.'),*

*('(-)', 'ANS', '?', KC2, FUN, 'neg'),*

*('Enter','Entry','', KC4, FUN, 'enter')]]*

*self.display = Pmw.ScrolledText(self, hscrollmode='dynamic', - vscrollmode='dynamic', hull\_relief='sunken',*

*hull\_background='gray40', hull\_borderwidth=10,*

*text\_background='honeydew4', text\_width=16, text\_foreground='black', text\_height=6, text\_padx=10, text\_pady=10, text\_relief='groove', text\_font=('arial', 12, 'bold'))*

*self.display.pack(side=TOP, expand=YES, fill=BOTH)*

*self.display.tag\_config('ans', foreground='white') self.display.component('text').bind('<Key>', self.doKeypress) self.display.component('text').bind('<Return>', self.doEnter)*

*!*

*@*

*for row in keys:*

*rowa = Frame(self, bg='gray40')*

*rowb = Frame(self, bg='gray40')*

*for p1, p2, p3, color, ktype, func in row:*

*if ktype == FUN:*

*a = lambda s=self, a=func: s.evalAction(a)*

*else:*

*a = lambda s=self, k=func: s.keyAction(k)*

*SLabel(rowa, p2, p3)*

*Key(rowb, text=p1, bg=color, command=a)*

*rowa.pack(side=TOP, expand=YES, fill=BOTH)*

*rowb.pack(side=TOP, expand=YES, fill=BOTH)*

*class Evaluator:*

*def \_\_init\_\_(self):*

*self.myNameSpace = {}*

*self.runpython("from math import \*")*

*def runpython(self, code):*

*try:*

*return `eval(code, self.myNameSpace, self.myNameSpace)` # except SyntaxError:*

*try:*

*exec code in self.myNameSpace, self.myNameSpace*

*except:*

*return 'Error'*

*Calculator().mainloop()*

***Code comments (continued)***

*9. A number of constants are defined. The following data structure is quite complex. Using con- stants makes it easy to change values throughout such a complex structure and they make the code much more readable and consequently easier to maintain.*

*FUN = 1*

*KEY = 0*

*KC1 = 'gray30'*

*KC2 = 'gray50'*

*# A Function*

*# A Key*

*# Dark Keys*

*# Light Keys*

*These are used to populate a nested list of lists, which contains tuples. The tuples store three labels, the key color, the function or key designator and the method to bind to the key’s cmd (activate) callback.*

*10. We create the Pmw ScrolledText widget and provide values for many of its attributes.*

*self.display = Pmw.ScrolledText(self, hscrollmode='dynamic',*

*vscrollmode='dynamic', hull\_relief='sunken',*

*hull\_background='gray40', hull\_borderwidth=10,*

*text\_background='honeydew4', text\_width=16,*

*Notice how the attributes for the hull (the container for the subordinate widgets within Pmw widgets) and the text widget are accessed by prefixing the widget.*

*11. We define a text tag which is used to differentiate output from input in the calculator’s screen.*

*self.display.tag\_config('ans', foreground='white') We saw this tag in use earlier in the text insert method.*

*12. Again, we must use a lambda expression to bind our callback function.*

*13. Python exceptions are quite flexible and allow simple control of errors. In the calculator’s evaluator (runpython), we first run eval.*

*try:*

*return `eval(code, self.myNameSpace, self.myNameSpace)`*

*This is used mainly to support direct calculator math. eval cannot handle code sequences, however, so when we attempt to eval a code sequence, a SyntaxError exception is raised.*

*14. We trap the exception:*

*except SyntaxError:*

*try:*

*exec code in self.myNameSpace, self.myNameSpace*

*except:*

*return 'Error'*

*and then the code is exec’ed in the except clause. Notice how this is enclosed by another try... except clause.*

*Figure 3.2 shows the results of clicking keys on the cal- culator to calculate simple math equations. Unlike many cal- culators, this displays the input and output in different colors. The display also scrolls to provide a history of calcu- lations, not unlike a printing calculator. If you click on the display screen, you may input data directly. Here is the sur- prise: you can enter Python and have exec run the code.*

*La figura 3.2 muestra los resultados de hacer clic en las teclas de la calculadora para calcular ecuaciones matemáticas simples. A diferencia de muchas calculadoras, esta muestra la entrada y la salida en diferentes colores. La pantalla también se desplaza para proporcionar un historial de cálculos, similar a una calculadora impresa. Si hace clic en la pantalla de visualización, puede ingresar datos directamente. Aquí está la sorpresa: puedes ingresar Python y hacer que exec ejecute el código.*

*Figure 3.3 shows how you can import the sys module and access built-in functions within Python. Technically, you could do almost anything from this window (within the constraint of a very small display window). However, I don’t think that this calculator is the much-sought Interactive Development Environment (IDE) for Python! (Readers who subscribe to the Python news group will understand that there has been a constant demand for an IDE for Python. Fortunately, Guido Van Rossum has now released IDLE with Python.)*

*La Figura 3.3 muestra cómo puede importar el módulo sys y acceder a funciones integradas dentro de Python. Técnicamente, puedes hacer casi cualquier cosa desde esta ventana (dentro de la limitación de una ventana de visualización muy pequeña). Sin embargo, no creo que esta calculadora sea el tan buscado entorno de desarrollo interactivo (IDE) para Python. (Los lectores que se suscriban al grupo de noticias de Python comprenderán que ha habido una demanda constante de un IDE para Python. Afortunadamente, Guido Van Rossum ahora lanzó IDLE con Python).*

*When you press ENTER after dir(), you will see output similar to figure 3.4. This list of built-in symbols has scrolled the display over several lines (the widget is only 16 characters wide, after all).*

*Because we are maintaining a local namespace, it is possible to set up an interactive Python session that can do some use- ful work. Figure 3.5 shows how we are able to set variables within the namespace and manipulate the data with built-ins.*

*Debido a que mantenemos un espacio de nombres local, es posible configurar una sesión interactiva de Python que puede realizar algún trabajo útil. La Figura 3.5 muestra cómo podemos establecer variables dentro del espacio de nombres y manipular los datos con funciones integradas.*

*Figure 3.6 is yet another example of our ability to gain access to the interpreter from an interactive shell. While the exam- ples have been restricted to operations that fit within the lim- ited space of the calculator’s display, they do illustrate a potential for more serious applications. Note how Python allows you to create and use variables within the current namespace.*

*La Figura 3.6 es otro ejemplo más de nuestra capacidad para obtener acceso al intérprete desde un shell interactivo. Si bien los ejemplos se han restringido a operaciones que caben dentro del espacio limitado de la pantalla de la calculadora, ilustran un potencial para aplicaciones más serias. Observe cómo Python le permite crear y usar variables dentro del espacio de nombres actual.*

***Note***

*When developing applications, I generally hide a button or bind a “secret” key sequence to invoke a GUI which allows me to execute arbitrary Python so that I can examine the namespace or modify objects within the running system. It is really a miniature debugger that I always have access to during development when something unusual happens. Sometimes restarting the application for a debug session just does not get me to the solution. An example of one of these tools is found in “A Tkinter explorer” on page 334.*

*Cuando desarrollo aplicaciones, generalmente oculto un botón o vinculo una secuencia de claves "secreta" para invocar una GUI que me permite ejecutar Python arbitrario para poder examinar el espacio de nombres o modificar objetos dentro del sistema en ejecución. En realidad, es un depurador en miniatura al que siempre tengo acceso durante el desarrollo cuando sucede algo inusual. A veces, reiniciar la aplicación para una sesión de depuración simplemente no me lleva a la solución. Un ejemplo de una de estas herramientas se encuentra en “Un explorador de Tkinter” en la página 334.*

***3.3 Examining the application structure***

*The calculator example derives its compact code from the fact that Tkinter provides much of the structure for the application. Importing Tkinter establishes the base objects for the system and it only requires a little extra code to display a GUI. In fact, the minimal Tkinter code that can be written is just four lines:*

*El ejemplo de la calculadora deriva su código compacto del hecho de que Tkinter proporciona gran parte de la estructura de la aplicación. La importación de Tkinter establece los objetos base para el sistema y solo requiere un poco de código adicional para mostrar una GUI. De hecho, el código Tkinter mínimo que se puede escribir es de sólo cuatro líneas:*

*from Tkinter import \**

*aWidget = Label(None, text=’How little code does it need?’)*

*aWidget.pack()*

*aWidget.mainloop()*

*In this fragment, the label widget is realized with the pack method. A mainloop is nec- essary to start the Tkinter event loop. In our calculator example, the application structure is a little more complex:*

*En este fragmento, el widget de etiqueta se implementa con el método pack. Es necesario un bucle principal para iniciar el bucle de eventos de Tkinter. En nuestro ejemplo de calculadora, la estructura de la aplicación es un poco más compleja:*

*from Tkinter import \**

*...*

*define helper classes*

*...*

*class Calculator:*

*Calling Calculator.mainloop() creates a calculator instance and starts the mainloop.*

*Llamar a Calculator.mainloop() crea una instancia de calculadora e inicia el bucle principal.*

*As we develop more applications, you will see this structure repeatedly. For those of us that tend to think spatially, the diagram shown in figure 3.7 may help.*

*A medida que desarrollemos más aplicaciones, verá esta estructura repetidamente. Para aquellos de nosotros que tendemos a pensar espacialmente, el diagrama que se muestra en la figura 3.7 puede ayudar.*

*All we have to do is fill in the blocks and we’re finished! Well, nearly finished. I believe that the most important block in the structure is the last one: “Test Code.” The purpose of this section is to allow you to test a module that is part of a suite of modules without the whole application structure being in place. Writing Python code this way will save a great deal of effort in integrating the com- ponents of the application. Of course, this approach applies to any implementation.*

*¡Todo lo que tenemos que hacer es completar los bloques y listo! Bueno, casi terminado. Creo que el bloque más importante de la estructura es el último: “Código de prueba”. El propósito de esta sección es permitirle probar un módulo que forma parte de un conjunto de módulos sin que toda la estructura de la aplicación esté implementada. Escribir código Python de esta manera ahorrará una gran cantidad de esfuerzo en la integración de los componentes de la aplicación. Por supuesto, este enfoque se aplica a cualquier implementación.*

***3.4 Extending the application***

*I leave you now with an exercise to extend the calculator and complete the functions that have been left undefined. It would be a simple task to modify the keys list to remove unnecessary keys and produce a rather more focused calculator. It would also be possible to modify the keys to provide a business or hex calculator.*

*Os dejo ahora con un ejercicio para ampliar la calculadora y completar las funciones que han quedado sin definir. Sería una tarea sencilla modificar la lista de claves para eliminar claves innecesarias y producir una calculadora bastante más enfocada. También sería posible modificar las claves para proporcionar una calculadora comercial o hexadecimal.*

*In subsequent examples, you will see more complex manifestations of the application structure illustrated by this example.*

*En ejemplos posteriores, verá manifestaciones más complejas de la estructura de la aplicación ilustrada en este ejemplo.*