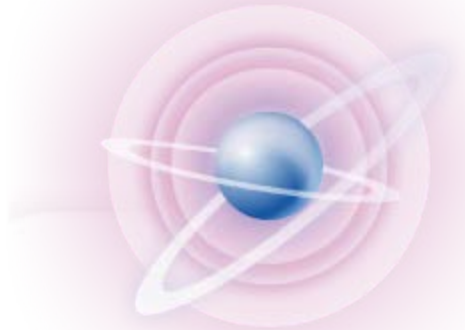
程式語言

PROGRAMMING LANGUAGE

1. Module & Package
2. GUI Development

陳炳志
國立成功大學 物理系



#11, 2021/12/08

模組

- 模組 (module) 是一個檔名為 *modulename*.py的Python檔案，裡面定義了一些資料、函式或類別，當要使用模組所提供的功能時，必須使用import指令進行匯入，其語法如下：

  import *modulename*

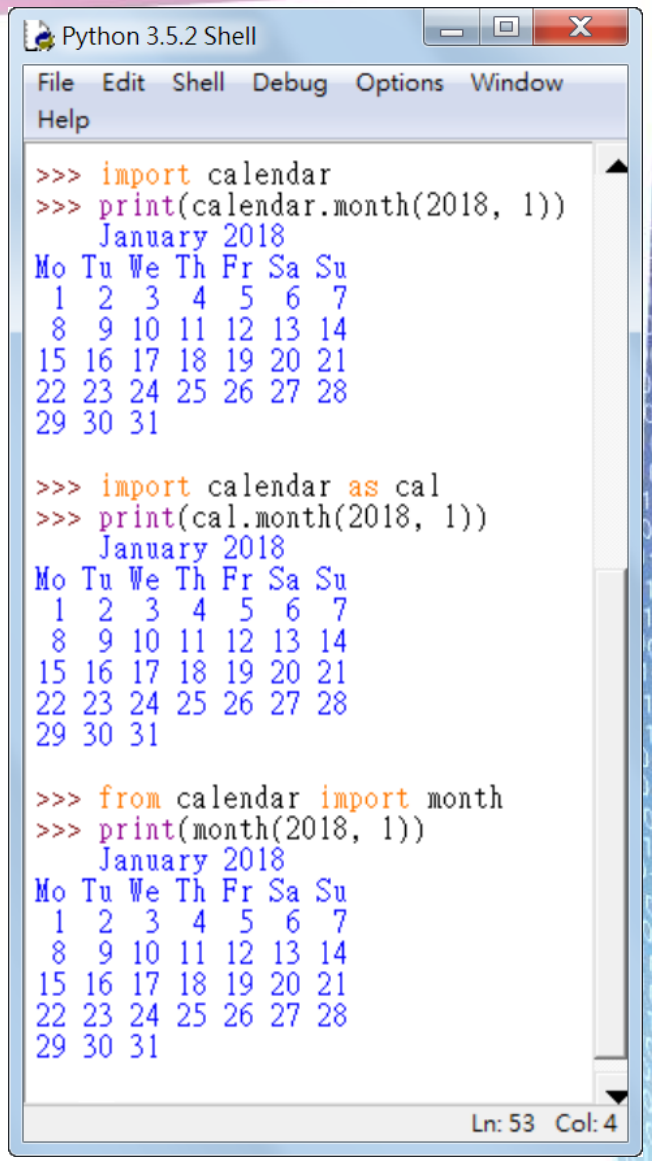
- 也可以在匯入模組的同時加上as替模組取個別名，其語法如下：

  import *modulename* **as** *alias*

- 也可以使用from … import … 從模組匯入特定的類別或函式，其語法如下：**form** *modulename* **import** *classname/functionname*

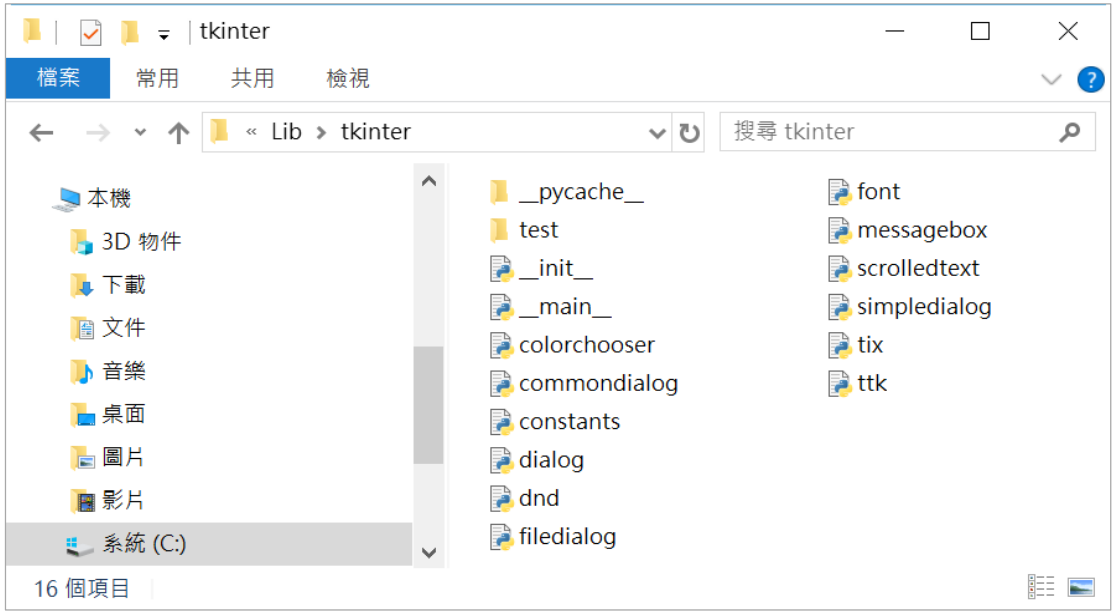
- 可以用 dir(*module_name*)查看模組內容。

- 可以用 help(*function*)查看模組內函式內容。

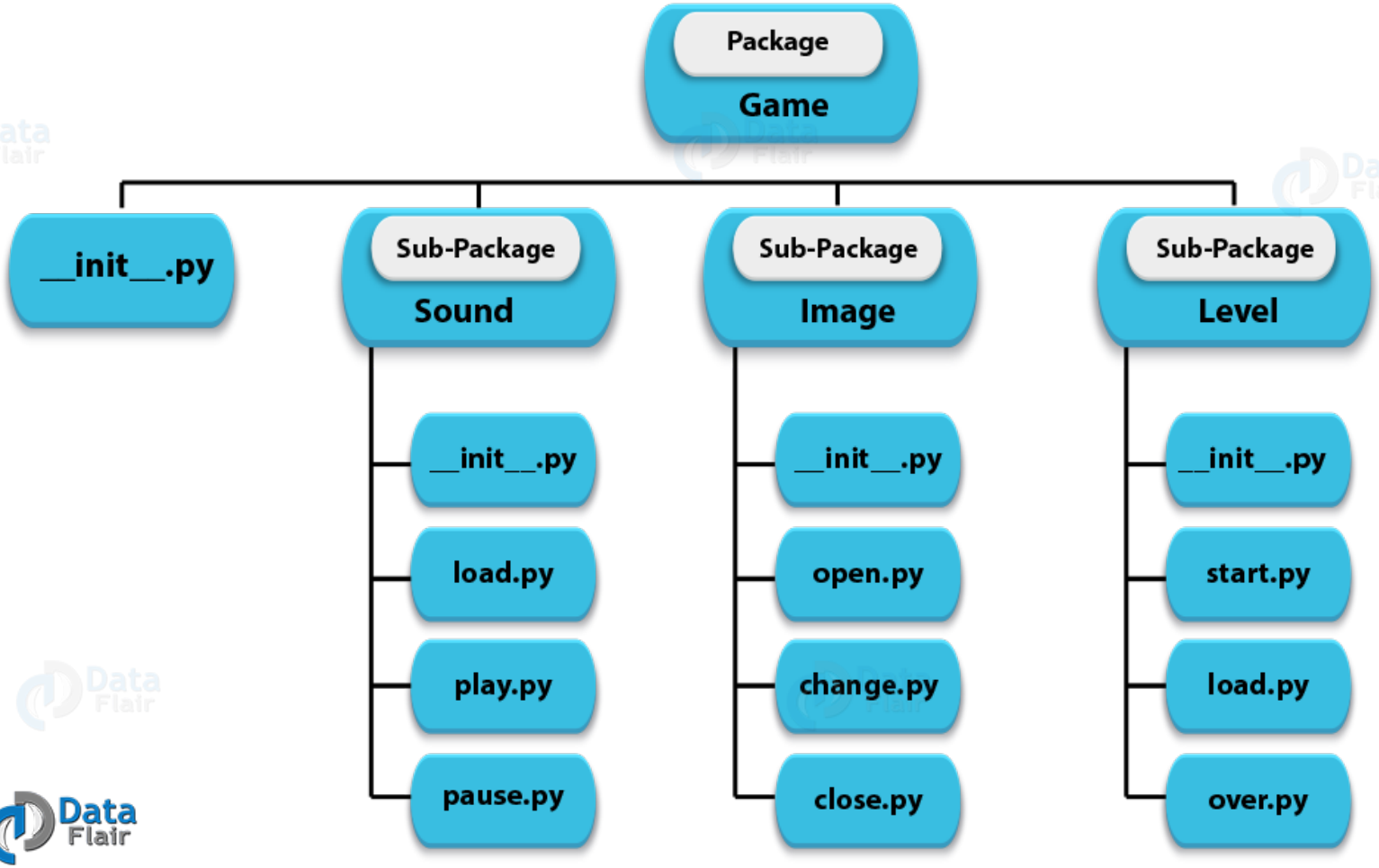


套件

套件 (package) 是存放了數個模組，就像一個資料夾。原則上，只要是包含 _ _init_ _.py檔案的資料夾就會被視為Python的一個套件，例如下圖為tkinter套件。



Package Module Structure



第三方套件(1/3)

相較於內建的模組與套件是在安裝Python時一併安裝，第三方套件(third-party package) 則是要另外安裝的套件。常見的第三方套件如下：

- NumPy：陣列與資料運算。
- Matplotlib：2D視覺化工具。
- SciPy：科學計算。
- pandas：資料處理與分析。
- Django、Pyramid、Web2py、Flask：web框架。
- Kivy、Flexx、Pywin32、PyQt、WxPython：GUI程式開發。
- BeautifulSoup：HTML/XML解析器。
- Pillow：圖形處理。
- PyGame：多媒體與遊戲軟體開發。
- Requests：存取網際網路資料。
- Scrapy：網路爬蟲套件。
- SciKit-Learn、TensorFlow、Keras：機器學習套件。

絕大多數Anaconda都已經內建安裝了！

第三方套件(2/3)

透過pip程式安裝第三方套件

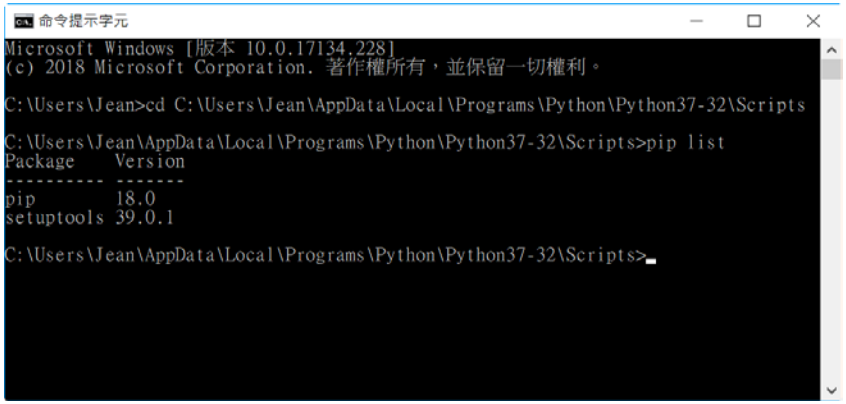
- **pip list指令用來列出目前安裝的套件與版本**，例如左下圖。

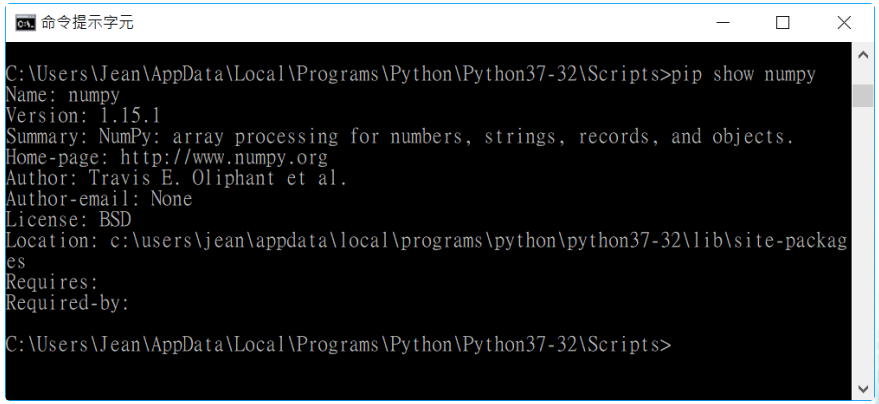
pip install指令用來安裝套件，例如下面的敘述會安裝NumPy套件：

C:\Users\Alfred>pip install numpy

- pip show指令用來查詢已經安裝的套件，例如右下圖。
- pip uninstall指令用來解除安裝套件，例如下面的敘述會解除安裝NumPy套件：

C:\Users\Alfred>pip uninstall numpy

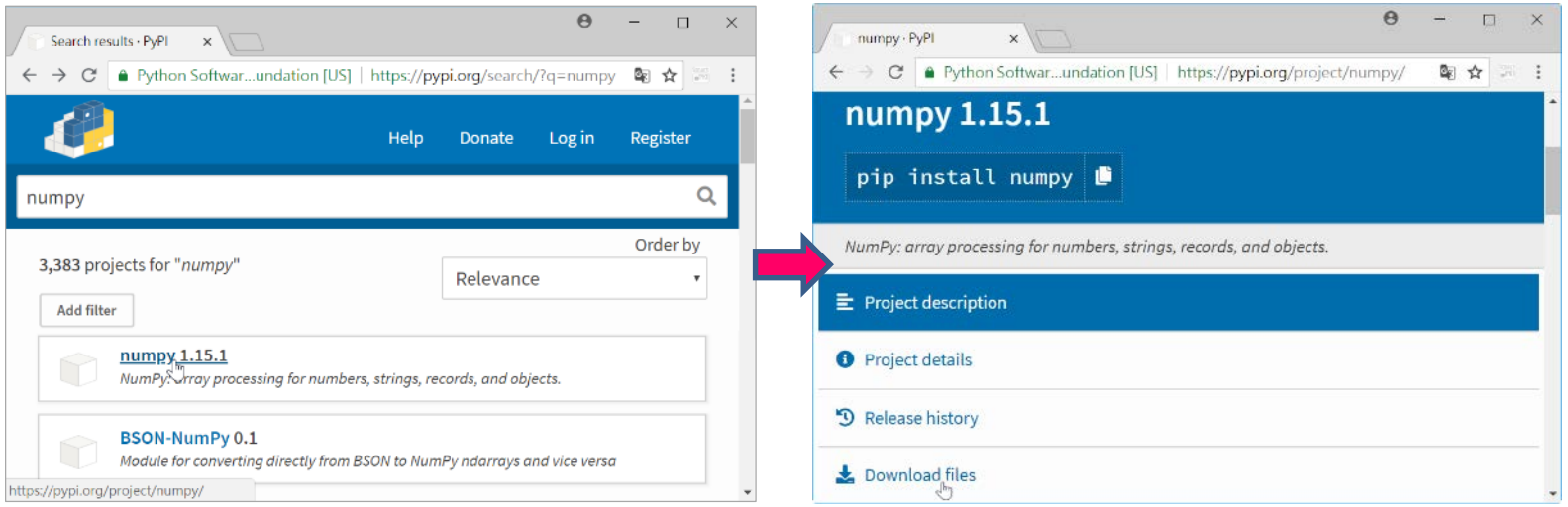




第三方套件(3/3)

透過PyPI網站安裝第三方套件

PyPI - the Python Package Index網站 (https://pypi.python.org/pypi) 登錄了數萬個第三方套件，只要輸入套件的名稱進行搜尋，例如numpy，就能找到相關的檔案，然後將檔案下載並安裝到電腦即可。



不要重新發明輪子！

使用TKINTER開發GUI程式

Graphical User Interface, GUI
圖形使用者介面

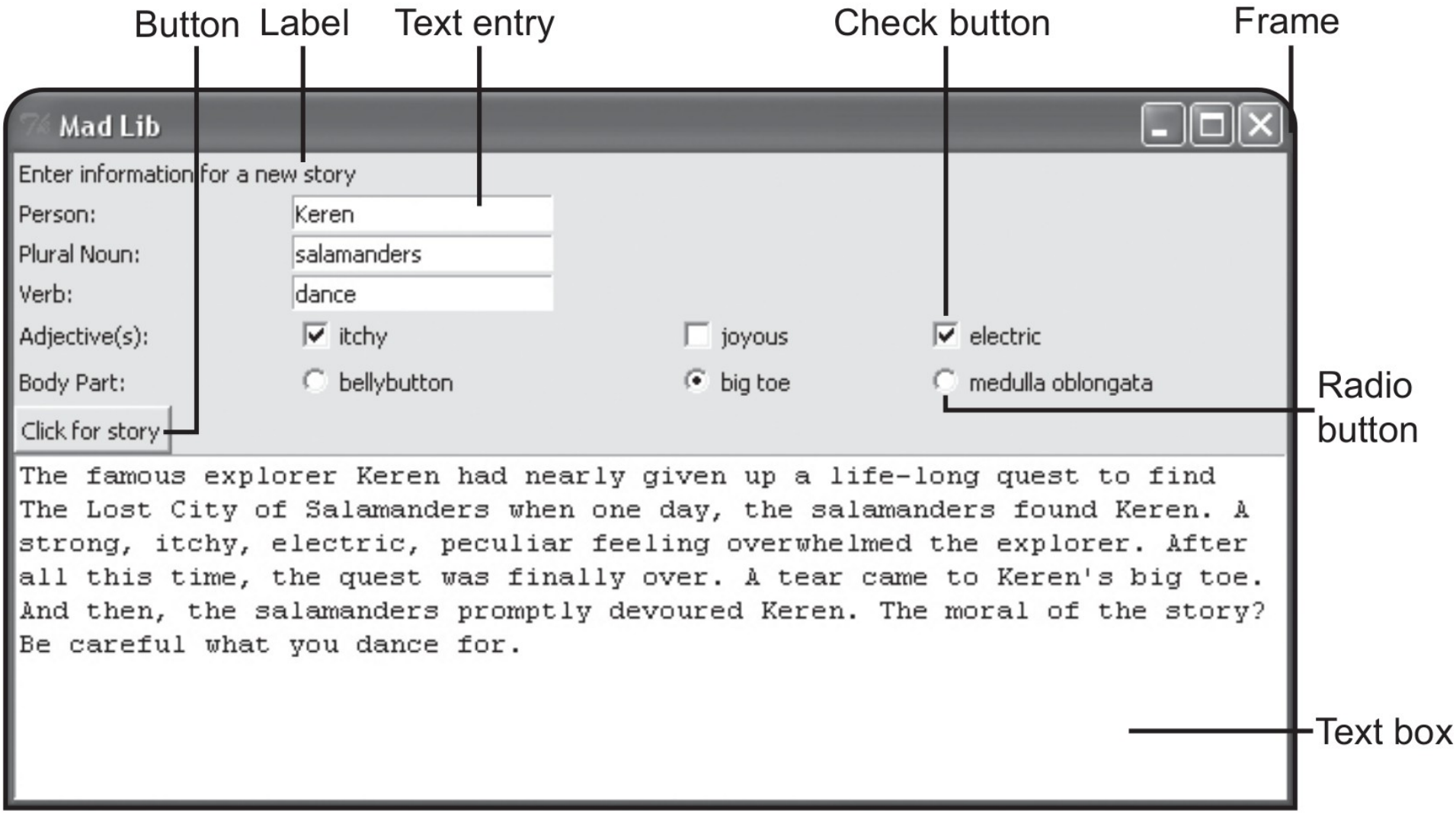
- 採用圖形方式顯示的電腦操作使用者介面。與早期電腦使用的命令列介面(Command Line Interface, CLI)相比，除了降低使用者的操作負擔之外，對於新使用者而言，圖形介面對於使用者來說在視覺上更易於接受，學習成本大幅下降，也讓電腦的大眾化得以實現。
- 雖然圖形使用者介面已經成為現代電腦的主要介面，然而這介面必定要透過在顯示器的特定位置，以「各種美觀、而不單調的視覺訊息」提示使用者「狀態的改變」，勢必得比簡單的文字訊息呈現，花上更多的電腦運算能力，計算「要改變顯示器哪些光點，變成哪些顏色」，功能命令的設計也比較複雜，現代作業系統的圖形複雜程度更遠超早期的GUI。

GUI 組成

- 桌面
- 視窗
  - 單一檔案介面（Single Document Interface）
  - 多檔案介面（Multiple Document Interface）
    - 標籤
- 清單
- 圖示
- 按鈕
- ….

Examining A GUI

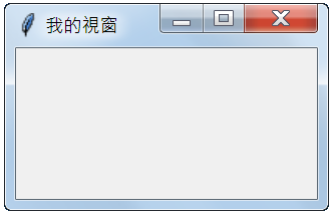
- Define all of the GUI (Graphical User Interface) elements you'll meet:

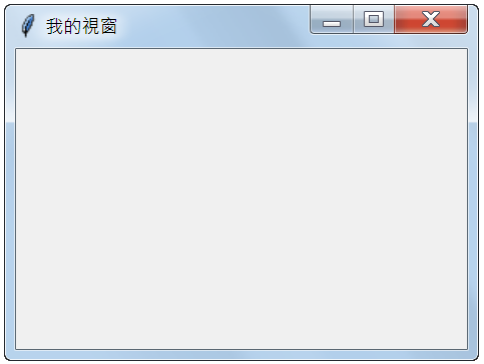


認識tkinter套件

tkinter (唸做tk-inter) 是一個跨平台的GUI套件，能夠在UNIX、Linux、Windows、Mac等平台開發GUI程式，下面是一個例子。

```
from tkinter import *
window = Tk()
window.title("我的視窗")
window.geometry("200x100")
window.maxsize(300, 200)
window.mainloop()
```





GUI元件(1/12)

window gadgets

- Tkinter提供了許多GUI元件 (widget)，例如：
  - Frame (視窗區域)
  - LabelFrame (標籤式視窗區域)
  - Label (文字標籤)
  - Entry (文字方塊)
  - Text (文字區域)
  - Button (按鈕)
  - Checkbutton (核取按鈕)
  - Radiobutton (選項按鈕)
  - Listbox (清單方塊)
  - Menu (功能表)
  - Menubutton (功能表按鈕)
  - Scrollbar (捲軸)
  - Scale (滑桿)
  - Spinbox (調整鈕)
  - messagebox (對話方塊)
  - PhotoImage (圖形)

- tkinter提供的元件都有各自的類別，如欲在視窗上面放置元件，只要根據元件類別建立物件即可，其語法如下：

  元件類別(*父物件*, *選擇性參數1* = *值1*, …)

- 例如：

```
btn1 = Button(window, text = "確定", bg = "yellow")
btn1["fg"] = "red"
btn1["bg"] = "blue"
btn1.config(fg = "red", bg = "blue")
```

GUI元件(2/12)

Label (文字標籤)

◆Label (文字標籤) 可以用來顯示無法由使用者編輯的文字，我們可以使用如下語法建立文字標籤：

Label(父物件, 選擇性參數1 = 值1, …)

◆例如：

- 常用的選擇性參數如下：
  - text
  - width
  - height
  - bg或background
  - fg或foreground
  - bd或borderwidth
  - padx
  - pady
  - justify


```
from tkinter import *
window = Tk()
label1 = Label(window, text = "文字標籤1", width = 30, bg = "lightyellow")
label2 = Label(window, text = "文字標籤2", width = 30, bg = "lightblue")
label3 = Label(window, text = "文字標籤3", width = 30, bg = "lightgray")
label1.pack()
label2.pack()
label3.pack()
window.mainloop()
```

GUI元件(3/12)

設定位置的第一種方法 - pack()

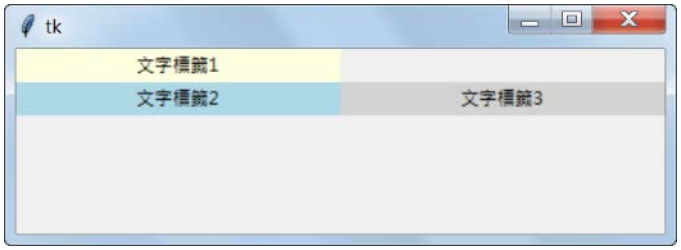
將前一個例子第06 ~ 08行改寫成如下：

```
label1.pack(side = BOTTOM)
label2.pack(side = BOTTOM)
label3.pack(side = BOTTOM)
```


```
label1.grid(row = 0, column = 0)
label2.grid(row = 1, column = 0)
label3.grid(row = 1, column = 1)
```

設定位置的第二種方法 - grid()

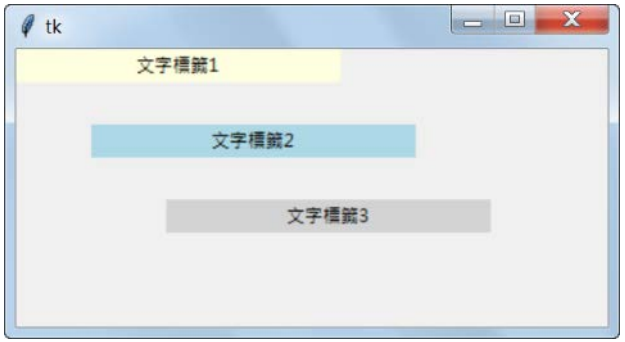
將前一個例子第06 ~ 08行改寫成如下：



設定位置的第三種方法 - place()

將前一個例子第06 ~ 08行改寫成如下：

```
label1.place(x = 0, y = 0)
label2.place(x = 50, y = 50)
label3.place(x = 100, y = 100)
```



GUI元件(4/12)

Button (按鈕)

◆Button (按鈕) 可以用來執行、終止或中斷動作，我們可以使用如下語法建立按鈕：

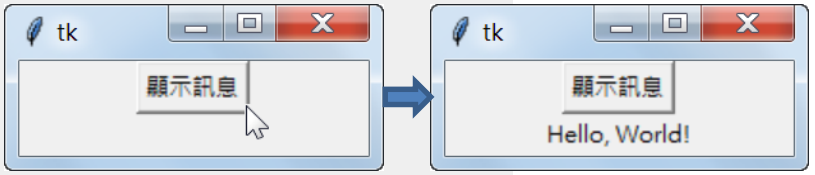
Button(父物件, 選擇性參數1 = 值1, ...)

◆例如：

- 常用的選擇性參數如下：
  - text
  - width
  - height
  - bg或background
  - fg或foreground
  - bd或borderwidth
  - padx
  - pady
  - justify
  - image
  - textvariable
  - underline
  - command

```
from tkinter import *
def showMsg():
    label1["text"] = "Hello, World!"

window = Tk()
btn1 = Button(window, text = "顯示訊息", command = showMsg)
label1 = Label(window)
btn1.pack()
label1.pack()
window.mainloop()
```

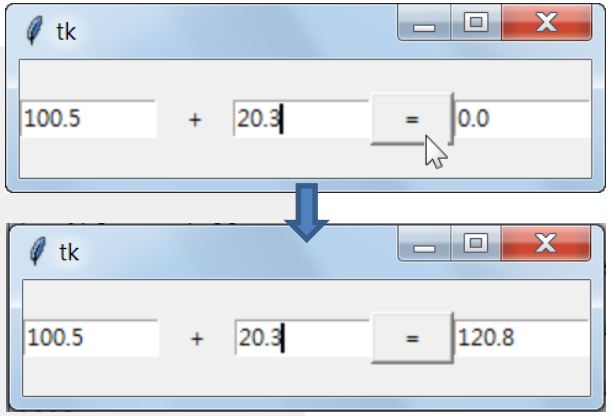


GUI元件(5/12)

Entry (文字方塊)

◆Entry (文字方塊) 可以用來取得使用者輸入的資料，我們可以使用如下語法建立文字方塊：

- 常用的選擇性參數如下：
  - width
  - bg或background
  - fg或foreground
  - state
  - show
  - textvariable




```
from tkinter import *
def add():
    result.set(num1.get() + num2.get())

window = Tk()
num1 = DoubleVar()
num2 = DoubleVar()
result = DoubleVar()
Entry(window, width = 10, textvariable = num1).pack(side = LEFT)
Label(window, width = 5, text = "+").pack(side = LEFT)
Entry(window, width = 10, textvariable = num2).pack(side = LEFT)
Button(window, width = 5, text = "=", command = add).pack(side = LEFT)
Entry(window, width = 10, textvariable = result).pack(side = LEFT)
window.mainloop()
```

GUI元件(6/12)

Text (文字區域)

◆Text (文字區域) 可以用來顯示與編輯具有格式的文字，我們可以使用如下語法建立文字區域：

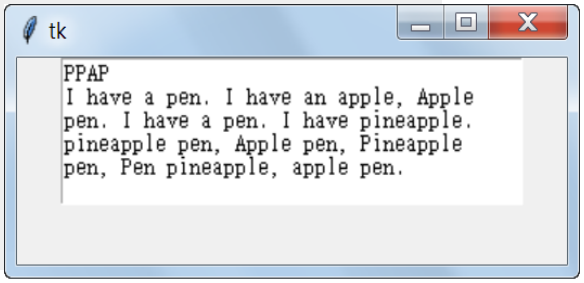
Text(父物件, 選擇性參數1 = 值1, ...)

- 常用的選擇性參數如下：
  - width
  - height
  - bg或background
  - fg或foreground
  - bd或borderwidth
  - padx
  - pady
  - state
  - wrap
  - xscrollcommand
  - yscrollcommand

◆例如：

```
from tkinter import *

song = "I have a pen. I have an apple, Apple pen. I have a pen. I have pineapple. \
pineapple pen, Apple pen, Pineapple pen, Pen pineapple, apple pen."
window = Tk()
text1 = Text(window, width = 40, height = 6, wrap = WORD)
text1.insert(END, "PPAP\n")
text1.insert(END, song)
text1.pack()
window.mainloop()
```



GUI元件(7/12)

Scrollbar (捲軸)

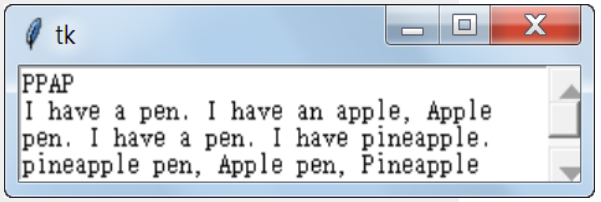
◆Scrollbar (捲軸) 可以用來在Text (文字區域)、Listbox (清單方塊) 顯示捲軸，我們可以使用如下語法建立捲軸：

Scrollbar(父物件, 選擇性參數1 = 值1, ...)

- 常用的選擇性參數如下：
  - width
  - bg或background
  - bd或borderwidth
  - highlightbackground
  - highlightcolor
  - activebackground
  - orient
  - command

◆例如：

```
from tkinter import *
 song = "I have a pen. I have an apple, Apple pen. I have a pen. I have pineapple. \
pineapple pen, Apple pen, Pineapple pen, Pen pineapple, apple pen."
 window = Tk()
sbar1 = Scrollbar(window)
text1 = Text(window, width = 40, height = 4, wrap = WORD)
text1.insert(END, "PPAP\n")
text1.insert(END, song)
sbar1.pack(side = RIGHT, fill = Y)
text1.pack(side = LEFT, fill = Y)
sbar1["command"] = text1.yview
text1["yscrollcommand"] = sbar1.set
window.mainloop()
```



GUI元件(8/12)

messagebox (對話方塊)

tkinter套件的messagebox模組提供下列數個方法可以用來顯示對話方塊：

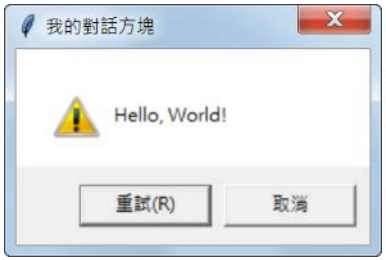
- askokcancel(*title*, *message*, *options*)
- askquestion(*title*, *message*, *options*)
- askretrycancel(*title*, *message*, *options*)
- askyesno(*title*, *message*, *options*)
- showerror(*title*, *message*, *options*)
- showinfo(*title*, *message*, *options*)
- showwarning(*title*, *message*, *options*)

例如：

```
from tkinter import messagebox
messagebox.askokcancel("我的對話方塊", "Hello, World!")
```

例如：

```
messagebox.askretrycancel("我的對話方塊", "Hello, World!")
```



GUI元件(9/12)

Checkbutton (核取按鈕)

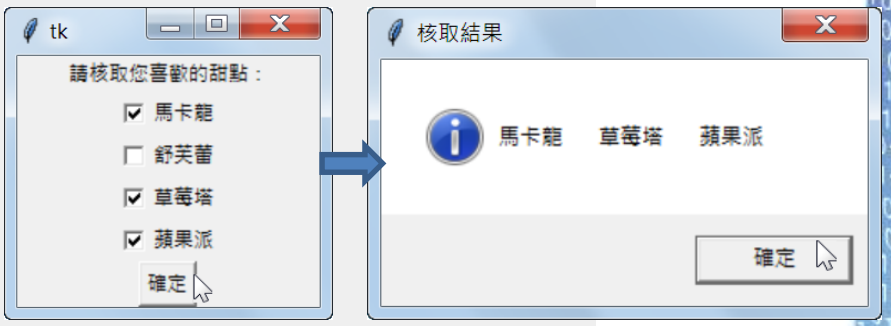
◆Checkbutton (核取按鈕) 允許使用者核取多個選項， 我們可以使用如下語法建立核取按鈕：

Checkbutton (父物件, 選擇性參數1 = 值1, …)

- 常用的選擇性參數如下：
  - text
  - width
  - height
  - bg或background
  - textvariable
  - variable
  - command

```
from tkinter import *
def showMsg():
   result = ''
   for i in checkvalue:
      if checkvalue[i].get() == True:
         result = result + dessert[i] + '\t'
   messagebox.showinfo("核取結果", result)

window = Tk()
label1 = Label(window, text = "請核取您喜歡的甜點：").pack()
dessert = {0 : "馬卡龍", 1 : "舒芙蕾", 2 : "草莓塔", 3 : "蘋果派"}
checkvalue = {}
for i in range(len(dessert)):
   checkvalue[i] = BooleanVar()
   Checkbutton(window, variable = checkvalue[i], text = dessert[i]).pack()
Button(window, text = "確定", command = showMsg).pack()
window.mainloop()
```



GUI元件(10/12)

Radiobutton (選項按鈕)

◆Radiobutton (選項按鈕) 允許使用者選取一個選項， 我們可以使用如下語法建立選項按鈕：

Radiobutton (父物件, 選擇性參數1 = 值1, …)

◆例如：

- 常用的選擇性參數如下：
  - text
  - width
  - height
  - Textvariable
  - value
  - variable
  - command


```
from tkinter import *

def showMsg():
    i = radiovalue.get()
    messagebox.showinfo("選取結果", dessert[i])

window = Tk()
label1 = Label(window, text = "請選取您最喜歡的甜點：").pack()
dessert = {0 : "馬卡龍", 1 : "舒芙蕾", 2 : "草莓塔", 3 : "蘋果派"}
radiovalue = IntVar()
radiovalue.set(0)
for i in range(len(dessert)):
    Radiobutton(window, text = dessert[i], variable = radiovalue, value = i).pack()
Button(window, text = "確定", command = showMsg).pack()
window.mainloop()
```

GUI元件(11/12)

Menu (功能表)

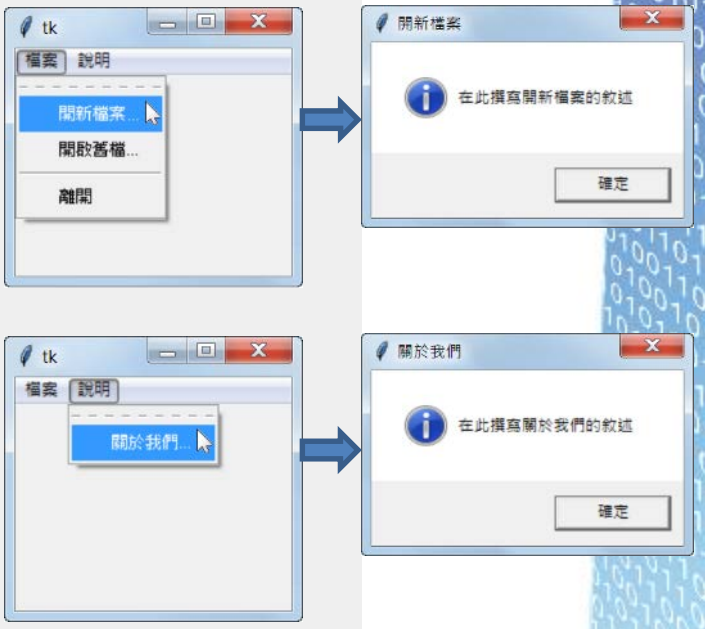
◆我們可以使用如下語法建立Menu (功能表)：

Menu (父物件, 選擇性參數1 = 值1, …)

◆例如：

```
from tkinter import *
def newFile():
    messagebox.showinfo("開新檔案", "在此撰寫開新檔案的敘述")
def openFile():
    messagebox.showinfo("開啟舊檔", "在此撰寫開啟舊檔的敘述")
def about():
    messagebox.showinfo("關於我們", "在此撰寫關於我們的敘述")
window = Tk()
menu = Menu(window)
window["menu"] = menu
filemenu = Menu(menu)
menu.add_cascade(label = "檔案", menu = filemenu)
filemenu.add_command(label = "開新檔案...", command = newFile)
filemenu.add_command(label="開啟舊檔...", command = openFile)
filemenu.add_separator()
filemenu.add_command(label="離開", command = window.destroy)
helpmenu = Menu(menu)
menu.add_cascade(label = "說明", menu = helpmenu)
helpmenu.add_command(label="關於我們...", command = about)
window.mainloop()
```

- 常用的選擇性參數與方法如下：
  - bg或background
  - fg或foreground
  - bd或borderwidth
  - activebackground
  - tearoff
  - add_cascade(*options*)
  - add_command(*options*)
  - add_separator()



GUI元件(12/12)

PhotoImage (圖形)

- 我們可以使用PhotoImage (圖形) 類別在視窗加入圖形，其語法如下：

  PhotoImage(file = "*GIF或PGM/PPM圖檔*"))

- 例如：

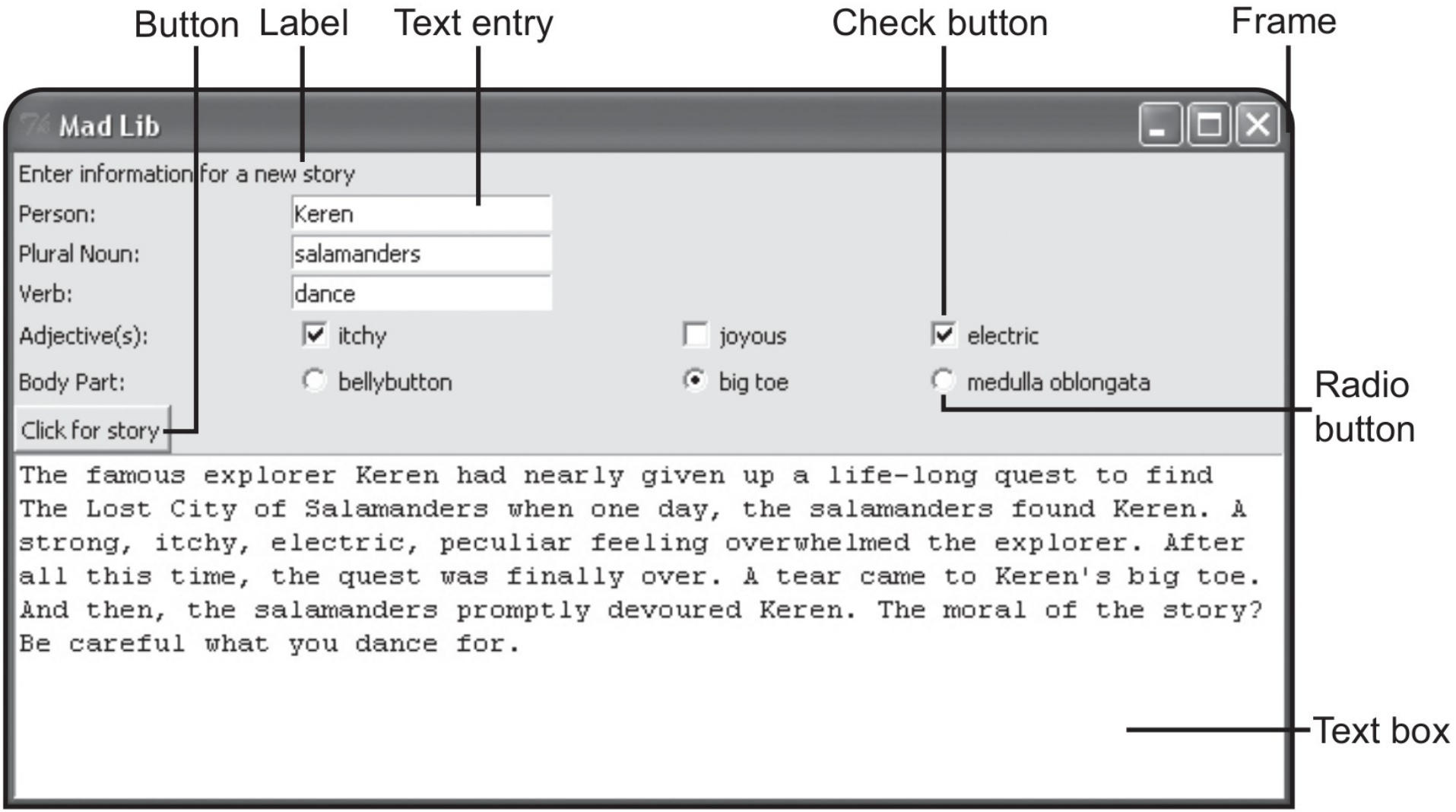
```
from tkinter import *

def showMsg():
    i = radiovalue.get()
    if i == 0:
        messagebox.showinfo("選取結果", "繡球花")
    else:
        messagebox.showinfo("選取結果", "鬱金香")

window = Tk()
image1 = PhotoImage(file = "flower1.gif")
image2 = PhotoImage(file = "flower2.gif")
label1 = Label(window, text = "請選取您最喜歡的花：").pack()
radiovalue = IntVar()
radiovalue.set(0)
Radiobutton(window, image = image1, variable = radiovalue, value = 0).pack()
Radiobutton(window, image = image2, variable = radiovalue, value = 1).pack()
Button(window, text = "確定", command = showMsg).pack()
window.mainloop()
```

Examining A GUI

Define all of the GUI (Graphical User Interface) elements you'll meet in this chapter.



- To create a GUI with Python, we need to use a GUI toolkit.
- There are many to pick from, but we use Tkinter, or TK, a popular cross-platform toolkit, here.
- We create GUI elements by instantiating objects from classes of the tkinter module, part of the Tkinter toolkit.

Element	tkinter	Class Description
Frame	Frame	Holds other GUI elements
Label	Label	Displays uneditable text or icons
Button	Button	Performs an action when the user activates it
Text entry	Entry	Accepts and displays one line of text
Text box	Text	Accepts and displays multiple lines of text
Check button	Checkbutton	Allows the user to select or not select an option
Radio button	Radiobutton	Allows, as a group, the user to select one option from several

- No need to memorize all of these tkinter classes.

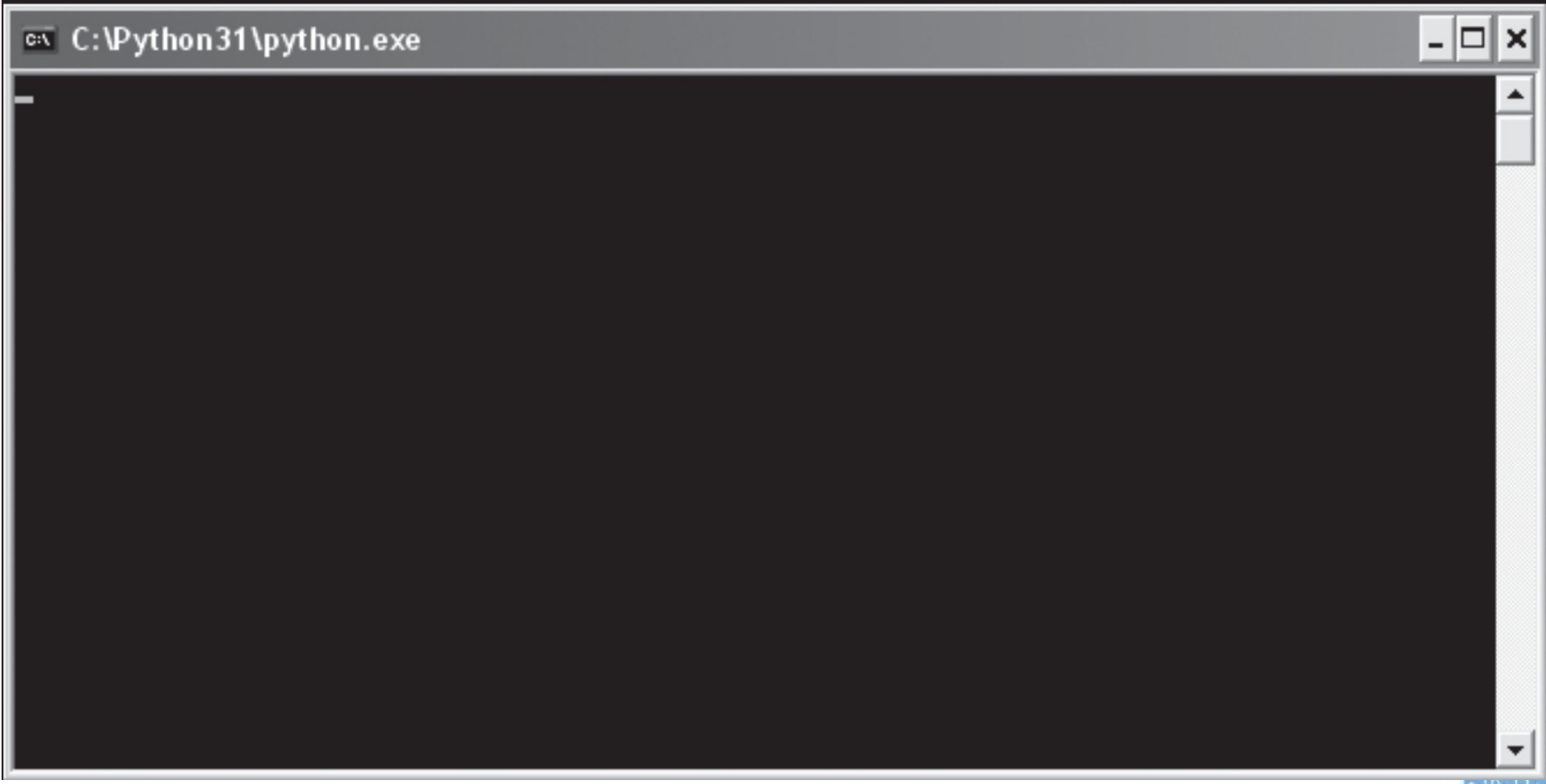
Understanding Event-Driven Programming

- GUI programs are *event-driven*, meaning they respond to actions regardless of the order in which they occur.
- For an event-driven program, you *bind* (associate) *events* (things that can happen involving the program's objects) with *event handlers* (code that runs when the events occur).
- By defining all of your objects, events, and event handlers, you establish how your program works. Then, you kick off the program by entering an event loop, where the program waits for the events to occur. When any of those events do occur, the program handles them, just as you've laid out.

Introducing the Simple GUI Program

- The batch file: simple_gui.bat

```
simple_gui.py
pause
```



- In addition to the window just showed, Simple GUI may generate another window (depending upon your system): the familiar console window.

- On a Windows machine, the easiest way to suppress the accompanying console window is to change the extension of your program from py to pyw.

simple_gui.py

```
# Simple GUI
# Demonstrates creating a window

from tkinter import *

# create the root window
root = Tk()

# modify the window
root.title("Simple GUI")
root.geometry("200x100")

# kick off the window's event-loop
root.mainloop()
```

Importing the tkinter Module

from tkinter import *

- The code imports all of tkinter directly into the program's global scope.
- Normally, you want to avoid doing something like this; however, tkinter is designed to be imported in this way.

Creating a Root Window

- To create a root window, we instantiate an object of the tkinter class **Tk**:

```
root = Tk()
```

- Notice that we didn't have to prefix the module name, tkinter, to the class name, **Tk**.

- In fact, we can now directly access any part of the tkinter module, without having to use the module name. This saves a lot of typing and makes code easier to read.

- You can have only one root window in a Tkinter program.

Modifying a Root Window

- Modify the root window using a few of its methods:

```
root.title("Simple GUI")
root.geometry("200x100")
```

- **title()** sets the title of the root window. All you have to do is pass the title you want displayed as a string.

- **geometry()** sets the size of the root window, in pixels. The method takes a **string** (and not integers) that represents the window's width and height, separated by the "x" character.

Entering a Root Window's Event Loop

- Start up the window's event loop by invoking root 's **mainloop()**:

```
root.mainloop()
```

- As a result, the window stays open, waiting to handle events.

Using Labels

- GUI elements are called *widgets*, short for window gadgets.
- The simplest widget is the Label widget, which is uneditable text and/or icons.
- A Label widget labels part of a GUI. It's often used to label other widgets. And labels aren't interactive.

Introducing the Labeler Program

- The batch file: labeler.bat

labeler.py
pause

labeler.py

```
# Labeler
# Demonstrates a label

from tkinter import *

# create the root window
root = Tk()
root.title("Labeler")
root.geometry("200x50")

# create a frame in the window to hold other widgets
app = Frame(root)
app.grid()

# create a label in the frame
lbl = Label(app, text = "I'm a label!")
lbl.grid()

# kick off the window's event loop
root.mainloop()
```

Creating a Frame

- A Frame is a widget that can hold other widgets (eg, Label widget). You use it as a base on which to place other things:

```
app = Frame(root)
```

- When you create a new widget, you must pass its *master* (the thing that will contain the widget) to the constructor of the new object. Here, we pass root to the **Frame** constructor.

- Invoke the **grid()** method of the new object:

```
app.grid()
```

- **grid()** is a method that all widgets have. It's associated with a *layout manager*, which lets you arrange widgets.

Creating a Label

- create a **Label** widget by instantiating an object of the **Label** class:

```
lbl = Label(app, text = "I'm a label!")
```

- By passing app to the **Label** object's constructor, we make the frame that app refers to the master of the **Label** widget. As a result, the label is placed in the frame.

- By passing "I'm a label!" to the text parameter, we set the widget's text option to that string.

- Invoke the object's **grid()** method:

```
lbl.grid()
```

ensures that the label will be visible.

Introducing the Lazy Buttons Program

- The batch file: lazy.buttons.bat

lazy_buttons.py
pause

lazy_buttons.py

```
# Lazy Buttons
# Demonstrates creating buttons

from tkinter import *

# create a root window
root = Tk()
root.title("Lazy Buttons")
root.geometry("200x85")

# create a frame in the window to hold other widgets
app = Frame(root)
app.grid()

# create a button in the frame
bttn1 = Button(app, text = "I do nothing!")
bttn1.grid()
```

```
# create a second button in the frame
bttn2 = Button(app)
bttn2.grid()
bttn2.configure(text = "Me too!")

# create a third button in the frame
bttn3 = Button(app)
bttn3.grid()
bttn3["text"]= "Same here!"

# kick off the root window's event loop
root.mainloop()
```

Creating Buttons

- Create a **Button** widget by instantiating an object of the **Button** class:

```
bttn1 = Button(app, text = "I do nothing!")
bttn1.grid()
```

- These lines create a new button with I do nothing! The button's master is the frame we created earlier, which means that the button is placed in the frame.

- You can create a widget and set all of its options in one line, or you can create a widget and set or alter its options later:

```
bttn2 = Button(app)
bttn2.grid()
```

- The only value we pass to the object's constructor is app, so all we have done is add a blank button to the frame.

- We can modify a widget after we create it, using the object's **configure()** method:

```
bttn2.configure(text = "Me too!")
```

- You can use a widget's **configure()** for any widget option (and any type of widget). You can even use the method to change an option that you've already set.

- Create a 3rd button:

```
bttn3 = Button(app)
bttn3.grid()
```

- Set the button's text option, using a different interface:

```
bttn3["text"]= "Same here!"
```

- We access the button's text option through a dictionary-like interface. I set the text option to "Same here!" , which puts the text Same here! on the button.

Introducing the Lazy Buttons 2 Program

- The batch file: lazy.buttons2.bat

lazy_buttons2.py
pause

lazy_buttons2.py

```
# Lazy Buttons 2
# Demonstrates using a class with Tkinter
from tkinter import *
class Application(Frame):
    """ A GUI application with three buttons. """
    Def __init__(self, master):
        """ Initialize the Frame. """
        super(Application, self).__init__(master)
        self.grid()
        self.create_widgets()

    def create_widgets(self):
        """ Create three buttons that do nothing. """
        # create first button
        self.bttn1 = Button(self, text = "I do nothing!")
        self.bttn1.grid()
```

```
        # create second button
        self.bttn2 = Button(self)
        self.bttn2.grid()
        self.bttn2.configure(text = "Me too!")

        # create third button
        self.bttn3 = Button(self)
        self.bttn3.grid()
        self.bttn3["text"] = "Same here!"

# main
root = Tk()
root.title("Lazy Buttons 2")
root.geometry("200x85")
app = Application(root)
root.mainloop()
```

Defining the Application Class

- Create a new class, Application, based on Frame:

```
class Application(Frame):
    """ A GUI application with three buttons. """
```

- Instead of instantiating a Frame object, we'll end up instantiating an Application object to hold all of the buttons.

Defining a Constructor Method

- Define Application's constructor:

```
def __init__(self, master):
    """ Initialize the Frame. """
    super(Application, self).__init__(master)
    self.grid()
    self.create_widgets()
```

- The 1st thing to do is call the superclass constructor. We pass along the Application object's master, so it gets set as the master.

- Finally, we invoke the Application object's create_widgets(), defined next.

Defining a Method to Create the Widgets

- create_widgets() creates all three buttons:

```
def create_widgets(self):
    # create first button
    self.bttn1 = Button(self, text = "I do nothing!")
    self.bttn1.grid()

    self.bttn2 = Button(self)          # create 2nd button
    self.bttn2.grid()
    self.bttn2.configure(text = "Me too!")

    self.bttn3 = Button(self)          # create 3rd button
    self.bttn3.grid()
    self.bttn3["text"] = "Same here!"
```

- Here bttn1, bttn2, bttn3 are attributes of an Application object. And we use self as the master for the buttons so that the Application object is their master.

Creating the Application Object

- In the main section of code, we create a root window:

```
root = Tk()
root.title("Lazy Buttons 2")
root.geometry("200x85")
```

- Then instantiate an Application object with the root window as its master:

```
app = Application(root)
```

- The Application object's constructor invokes the object's create_widgets(). This method then creates the 3 buttons, with the Application object as their master.

- Finally, we invoke the root window's event loop:

```
root.mainloop()
```

Introducing the Click Counter Program

- The batch file: click_counter.bat

```
click_counter.py
pause
```

click_counter.py

```
# Click Counter
# Demonstrates binding events with an event handler

from tkinter import * class

Application(Frame):
    """ GUI application which counts button clicks. """
    Def __init__(self, master):
        """ Initialize the frame. """
        super(Application, self). __init__(master)
        self.grid()
        # the number of button clicks
        self.bttn_clicks = 0
        self.create_widget()
```

```
    def create_widget(self):
        """ Create button displaying number of clicks. """
        self.bttn = Button(self)
        self.bttn["text"]= "Total Clicks: 0"
        self.bttn["command"] = self.update_count
        self.bttn.grid()

    def update_count(self):
        """ Increase click count and display new total. """
        self.bttn_clicks += 1
        self.bttn["text"]= "Total Clicks: " \
                           + str(self.bttn_clicks)
# main
root = Tk()
root.title("Click Counter")
root.geometry("200x50")

app = Application(root)

root.mainloop()
```

Setting Up the Program

- start the Application class definition:

```
class Application(Frame):
    """ GUI application which counts button clicks. """
        def __init__(self, master):
        """ Initialize the frame. """
        super(Application, self).__init__(master)
        self.grid()
        self.bttn_clicks = 0        # the number of button clicks
        self.create_widget()
```

- We creates an object attribute self.bttn_clicks to keep track of the number of times the user clicks the button.

Binding the Event Handler

- In the create_widget() method, we create a single button:

```
def create_widget(self):
    """ Create button displaying number of clicks. """
    self.bttn = Button(self)
    self.bttn["text"]= "Total Clicks: 0"
    self.bttn["command"] = self.update_count
    self.bttn.grid()
```

- We set the Button widget's **command** option to the update_count() method. So when the user clicks the button, the method is invoked.

- Technically, what we've done is bind an event (the clicking of Button widget) to an event handler (ie, update_count()).

- In general, you set a widget's **command** option to bind the activation of the widget with an event handler.

Creating the Event Handler

- update_count() handles the event of the button being clicked:

```
def update_count(self):
    """ Increase click count and display new total. """
    self.bttn_clicks += 1
    self.bttn["text"]= "Total Clicks: " \
                       + str(self.bttn_clicks)
```

- This method increments the total number of button clicks and then changes the text of the button to reflect the new total.

Wrapping Up the Program

- The main part of the code:

```
# main
root = Tk() root.title("Click
Counter")
root.geometry("200x50")

app = Application(root)

root.mainloop()
```

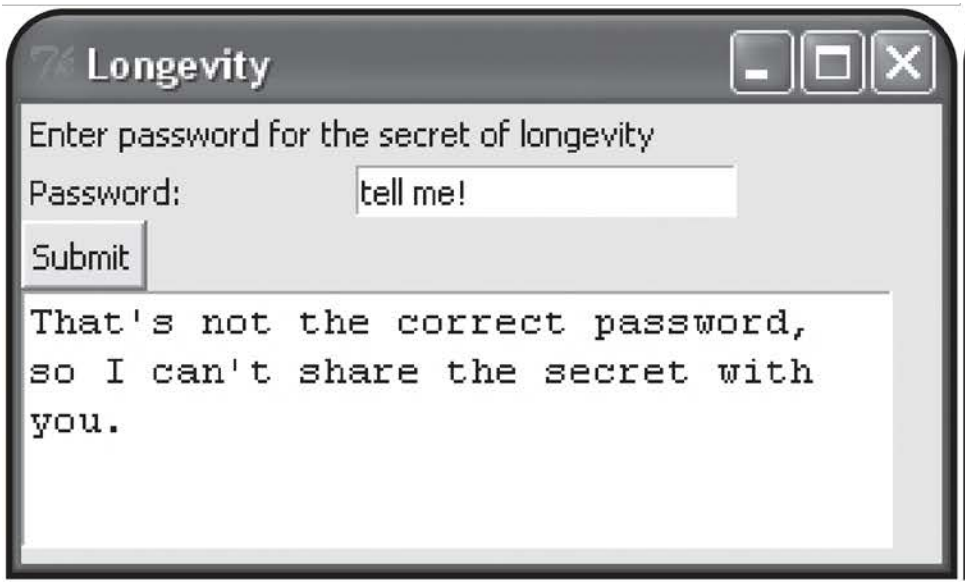
- We create a root window and set its title and dimensions.

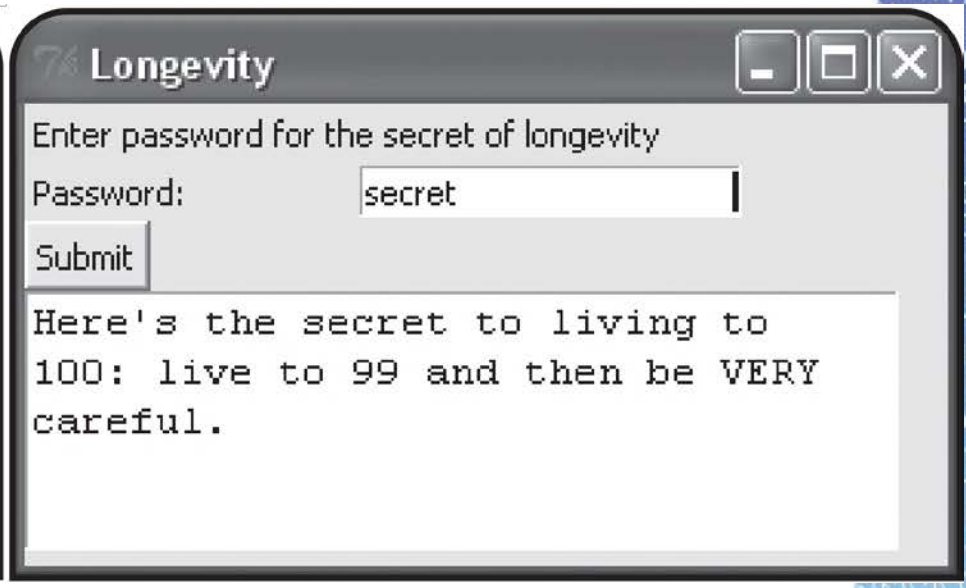
Then I instantiate a new Application object with the root window as its master. Lastly, I start up the root window's event loop to bring the GUI to life on the screen.

Introducing the Longevity Program

- The batch file: longevity.bat

```
longevity.py
pause
```





longevity.py

```
# Longevity
# Demonstrates text, entry widgets, grid layout manager

from tkinter import * class

Application(Frame):
    """ GUI application reveals the secret of longevity. """
    def __init__(self, master):
            super(Application, self).__init__(master)
            self.grid()
            self.create_widgets()

def create_widgets(self):
    # create instruction label
    self.inst_lbl = Label(self, text="Enter password”+\
                                    “ for the secret of longevity")
    self.inst_lbl.grid(row = 0, column = 0,
                               columnspan = 2, sticky = W)
```

```
# create label for password
self.pw_lbl = Label(self, text = "Password: ")
self.pw_lbl.grid(row = 1, column = 0, sticky = W)

# create entry widget to accept password
self.pw_ent = Entry(self)
self.pw_ent.grid(row = 1, column = 1, sticky = W)

# create submit button
self.submit_bttn = Button(self, text = "Submit",
                          command = self.reveal)
self.submit_bttn.grid(row=2, column=0, sticky=W)

# create text widget to display message
self.secret_txt = Text(self, width = 35, height = 5,
                       wrap = WORD)
self.secret_txt.grid(row = 3, column = 0,
                     columnspan = 2, sticky = W)
```

```
    def reveal(self):
        """ Display message based on password. """
        contents = self.pw_ent.get()
        if contents == "secret":
            message="Here's the secret to living to 100:”+\ “
                            live to 99 and then be VERY careful."
        else:
            message = "That's not the correct password,”+\
                             “so I can't share the secret with you."
        self.secret_txt.delete(0.0, END)
        self.secret_txt.insert(0.0, message)

# main
root = Tk()
root.title("Longevity")
root.geometry("300x150")

app = Application(root)

root.mainloop()
```

Placing a Widget with the Grid Layout Manager

- Start create_widgets() and create a label that provides instructions to the user:

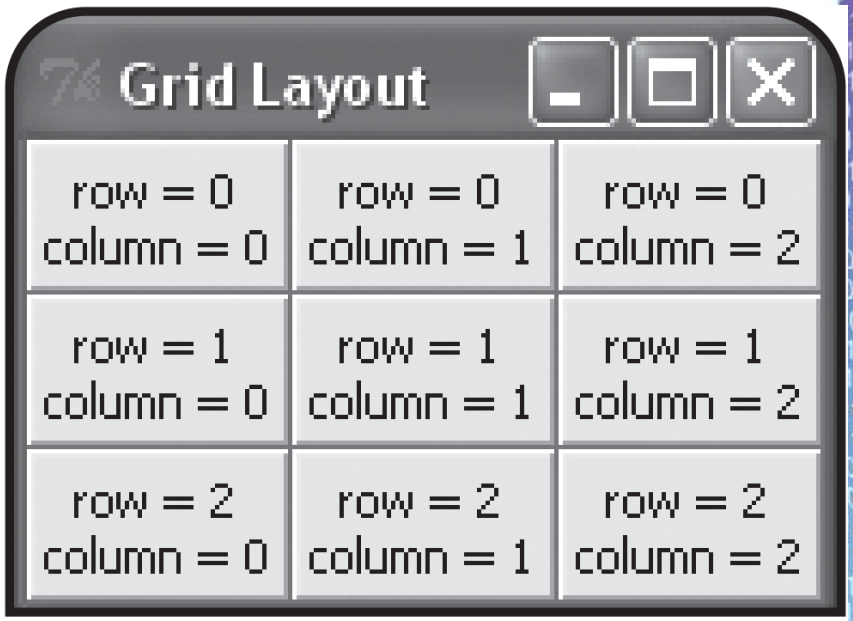
```
def create_widgets(self):
    self.inst_lbl=Label(self, text="Enter password”+\
                                “ for the secret of longevity")
```

- Use the grid layout manager to be specific about the placement of this label:

```
self.inst_lbl.grid(row = 0, column = 0,
                   columnspan = 2, sticky = W)
```

- A widget object’s grid() method can take values for many different parameters, but we only use 4 of them: **row**, **column**, **columnspan**, and **sticky**.

- The **row** and **column** parameters take integers and define where an object is placed within its master widget.
- Imagine the frame in the root window as a grid, divided into rows and columns. At each row and column intersection is a cell, where you can place a widget.
- For our Label widget, we pass 0 to **row**, 0 to **column**, which puts the label in the upper-left corner of the frame.
- If a widget is wide, you may want to allow the widget to span more than one cell so that your other widgets are correctly spaced.



- **columnspan** lets you span a widget over more than one column. We pass 2 to this parameter to allow the long label to span 2 columns.

- So the label takes up 2 cells, the one at row 0, column 0, and the other at row 0, column 1.
- You can also use the **rowspan** parameter to allow a widget to span more than one row.
- You can justify the widget within the cell by using **sticky**, which takes directions as values, including **N**, **S**, **E**, and **W**.
- Since we pass **W** to **sticky** for the Label object, the label is forced to the west (left).
- Create a label that appears in the next row, left-justified:

```
self.pw_lbl = Label(self, text = "Password: ")
self.pw_lbl.grid(row = 1, column = 0, sticky = W)
```

Creating an Entry Widget

- Create a new type of widget, an Entry widget:

```
self.pw_ent = Entry(self)
self.pw_ent.grid(row = 1, column = 1, sticky = W)
```

creates a text entry where the user can enter a password, and position the **Entry** in the cell next to the password label.

- Create a button to submit the password:

```
self.submit_bttn = Button(self, text = "Submit",
                          command = self.reveal)
self.submit_bttn.grid(row=2,column=0, sticky=W)
```

bind the activation of the button with reveal(), which reveals the longevity secret, if the user has entered the correct password. And place the button in the next row, all the way to the left.

Creating a Text Widget

- Create a new type of widget, a Text widget:

```
self.secret_txt = Text(self, width = 35, height = 5,
                       wrap = WORD)
self.secret_txt.grid(row = 3, column = 0,
                     columnspan = 2, sticky = W)
```

- We pass values to **width** & **height** to set the text box. Then we pass a value to the parameter **wrap**, which could be

  * **WORD** wraps entire words when you reach the right edge of the text box.
  * **CHAR** wraps character, meaning that when you get to the right edge of the text box, the next character simply appears on the following line.
  * **NONE** means no wrapping. As a result, you can only write text on the 1st line of the text box.

Getting and Inserting Text with Text- Based Widgets

- reveal() tests if the user has entered the correct password. If so, the method displays the secret to a long life. Otherwise the user is told that the password is incorrect.
- Firstly get the text in the Entry widget by invoking its **get() def**

```
reveal(self):
    contents = self.pw_ent.get()
```

- **get()** returns the text in the widget. Both Entry and Text objects have a **get()** method.
- Check if the text is equal to "secret". If so, set message to the string describing the secret to living to 100. Otherwise, set message to the string that tells the user that he entered the wrong password:

```
if contents == "secret":
    message="Here's the secret to living to 100:”+\
                    “ live to 99 and then be VERY careful."
else:
    message = "That's not the correct password,”+\
                    “so I can't share the secret with you."
```

- Now we have the string that we want to show to the user, we need to insert it into the Text widget.
- Firstly, delete any text already in the Text widget by invoking its **delete()**:

```
self.secret_txt.delete(0.0, END)
```

- **delete()** can delete text from text-based widgets. It can take a single index, or a beginning and an ending point.

- Pass floating-point numbers to represent a row and column number pair where the digit to the left of the decimal point is the row number and the digit to the right of the decimal point is the column number.

- Pass 0.0 as the starting point, meaning that the method should delete text starting at row 0, column 0 (the absolute beginning) of the text box.

- **END** means the end of the text. So, this line of code deletes everything from the 1st position in the text box to the end.
  Both Text and Entry widgets have a **delete()** method.

- Insert the string we want to display into the Text widget:

```
self.secret_txt.insert(0.0, message)
```

- **insert()** inserts a string into a text-based widget. The method takes an insertion position and a string.

- We pass 0.0 as the insertion position, meaning the method should start inserting at row 0, column 0. We pass message as the 2nd value, so that the appropriate message shows up in the text box.

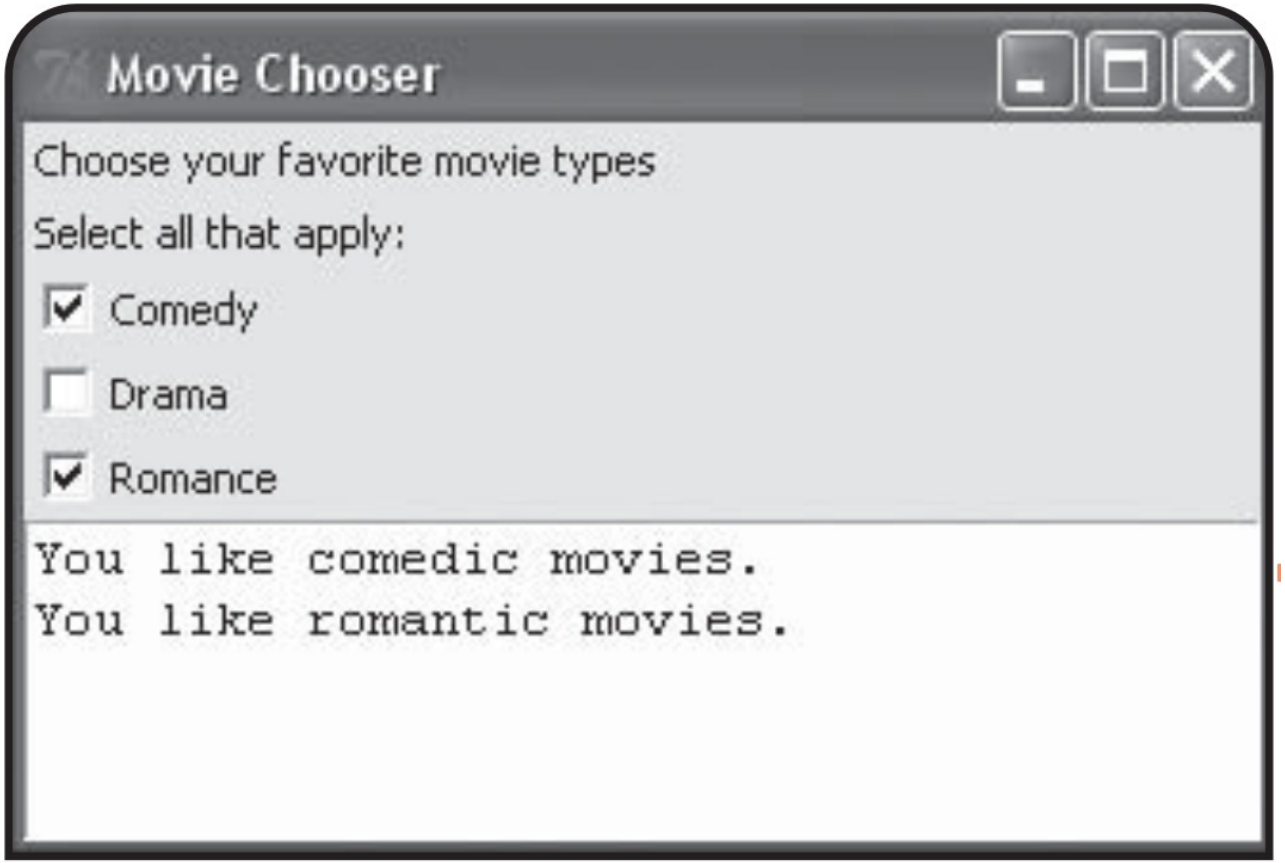
- Both Text and Entry widgets have an **insert()** method.

- **insert()** doesn't replace the text in a text-based widget—it simply inserts it. If you want to replace the existing text with new text, first call the text-based widget's **delete()** method.

Introducing the Movie Chooser Program

The batch file: movie_chooser.bat

```
movie_chooser.py
pause
```



movie_chooser.py

```
# Movie Chooser
# Demonstrates check buttons

from tkinter import * class

Application(Frame):
    """ GUI Application for favorite movie types. """
    def __init__(self, master):
            super(Application, self).__init__(master)
            self.grid()
            self.create_widgets()

    def create_widgets(self):
    """ Create widgets for movie type choices. """
    # create description label
    Label(self,
             text = "Choose your favorite movie types"\
            ).grid(row = 0, column = 0, sticky = W)
```

```
# create instruction label
Label(self, text = "Select all that apply:"\
      ).grid(row = 1, column = 0, sticky = W)

# create Comedy check button
self.likes_comedy = BooleanVar()
Checkbutton(self, text = "Comedy",
            variable = self.likes_comedy,
            command = self.update_text\
            ).grid(row = 2, column = 0, sticky = W)

# create Drama check button
self.likes_drama = BooleanVar()
Checkbutton(self, text = "Drama",
            variable = self.likes_drama,
            command = self.update_text \
            ).grid(row = 3, column = 0, sticky = W)
```

```
    # create Romance check button
    self.likes_romance = BooleanVar()
    Checkbutton(self, text = "Romance",
                variable = self.likes_romance,
                command = self.update_text\
                ).grid(row = 4, column = 0, sticky = W)

    # create text field to display results
    self.results_txt = Text(self, width=40, height = 5,
                                                wrap = WORD)
    self.results_txt.grid(row=5, column=0,
                                        columnspan=3)

def update_text(self):
    """ Update and display user's favorite movie. """
    likes = ""

    if self.likes_comedy.get():
        likes += "You like comedic movies.\n"
```

```
        if self.likes_drama.get():
            likes += "You like dramatic movies.\n"

        if self.likes_romance.get():
            likes += "You like romantic movies."

        self.results_txt.delete(0.0, END)
        self.results_txt.insert(0.0, likes)

# main
root = Tk() root.title("Movie
Chooser") app =
Application(root)
root.mainloop()
```

Allowing a Widget's Master to Be Its Only Reference

- Create a label that describes the program:

```
def create_widgets(self):
    Label(self, text="Choose your favorite movie types"\
        ).grid(row = 0, column = 0, sticky = W)
```

- The important difference between this label and the earlier: we don't assign the resulting Label object to a variable. This would usually be a mistake, rendering the object useless because it isn't connected to the program in any way.

- With tkinter, a Label object is connected to the program, like all GUI elements, by its master.

- So if we won't need to directly access a widget, we don't need to assign the object to a variable. The main benefit of this approach is shorter, cleaner code.

- So the master of the Label object is the only reference to it.

- Create another label in much the same way:

```
# create instruction label
Label(self, text = "Select all that apply:"\
        ).grid(row = 1, column = 0, sticky = W)
```

to provides instructions, telling the user that he can select as many movie types as apply.

Creating Check Buttons

- Create the check buttons, one for each movie type.
- Every check button needs a special object associated with it that automatically reflects the check button's status.
- The special object must be an instance of the **BooleanVar** class from the tkinter module. A Boolean *variable* is a special kind of variable that can be only True or False.
- We instantiate a BooleanVar object and assign it to a new object attribute, likes_comedy, before we create the Comedy check button:

```
self.likes_comedy = BooleanVar()
```

- Next, we create the check button itself:

```
Checkbutton(self, text = "Comedy",
            variable = self.likes_comedy,
            command = self.update_text\
            ).grid(row = 2, column = 0, sticky = W)
```

creates a new check button with the text Comedy.

- By passing self.likes_comedy to **variable**, we associate the check button's status (selected or unchecked) with the likes_comedy attribute.

- By passing self.update_text() to command, we bind the activation of the check button with update_text(). Whenever the user selects or clears the check button, the update_text() method is invoked.

- We don't assign the resulting **Checkbutton** object to a variable because what we care about is the status of the button, which I can access from the likes_comedy attribute.

- Create the next 2 check buttons in the same way:

```
self.likes_drama = BooleanVar()
Checkbutton(self, text = "Drama",
            variable = self.likes_drama,
            command = self.update_text\
            ).grid(row = 3, column = 0, sticky = W)

self.likes_romance = BooleanVar()
Checkbutton(self, text = "Romance",
            variable = self.likes_romance,
            command = self.update_text\
            ).grid(row = 4, column = 0, sticky = W)
```

- Whenever the user selects or clears the Drama or Romance check buttons, update_text() is invoked.

- Even though we don't assign the resulting **Checkbutton** objects to any variables, we can always see the status of the Drama/Romance check button through the likes_drama/ likes_romance attribute.

- Finally, we create the text box to show the results of the user's selections:

```
# create text field to display results
self.results_txt = Text(self, width = 40, height = 5,
                                                wrap = WORD)
self.results_txt.grid(row=5,column=0,columnspan=3)
```

Getting the Status of a Check Button

- update_text() updates the text box to reflect the check buttons the user has selected:

```
def update_text(self):
    likes = ""

    if self.likes_comedy.get():
        likes += "You like comedic movies.\n"
    if self.likes_drama.get():
        likes += "You like dramatic movies.\n"
    if self.likes_romance.get():
        likes += "You like romantic movies."

    self.results_txt.delete(0.0, END)
    self.results_txt.insert(0.0, likes)
```

- You can't access the value of a BooleanVar object directly. Instead, you must invoke the object's get() method.

- We use get() of the BooleanVar object referenced by likes_comedy to get the object's value.

- If the value evaluates to True, so the Comedy check button is selected, and we add "You like comedic movies.\n" to the string we are building to display in the text box.

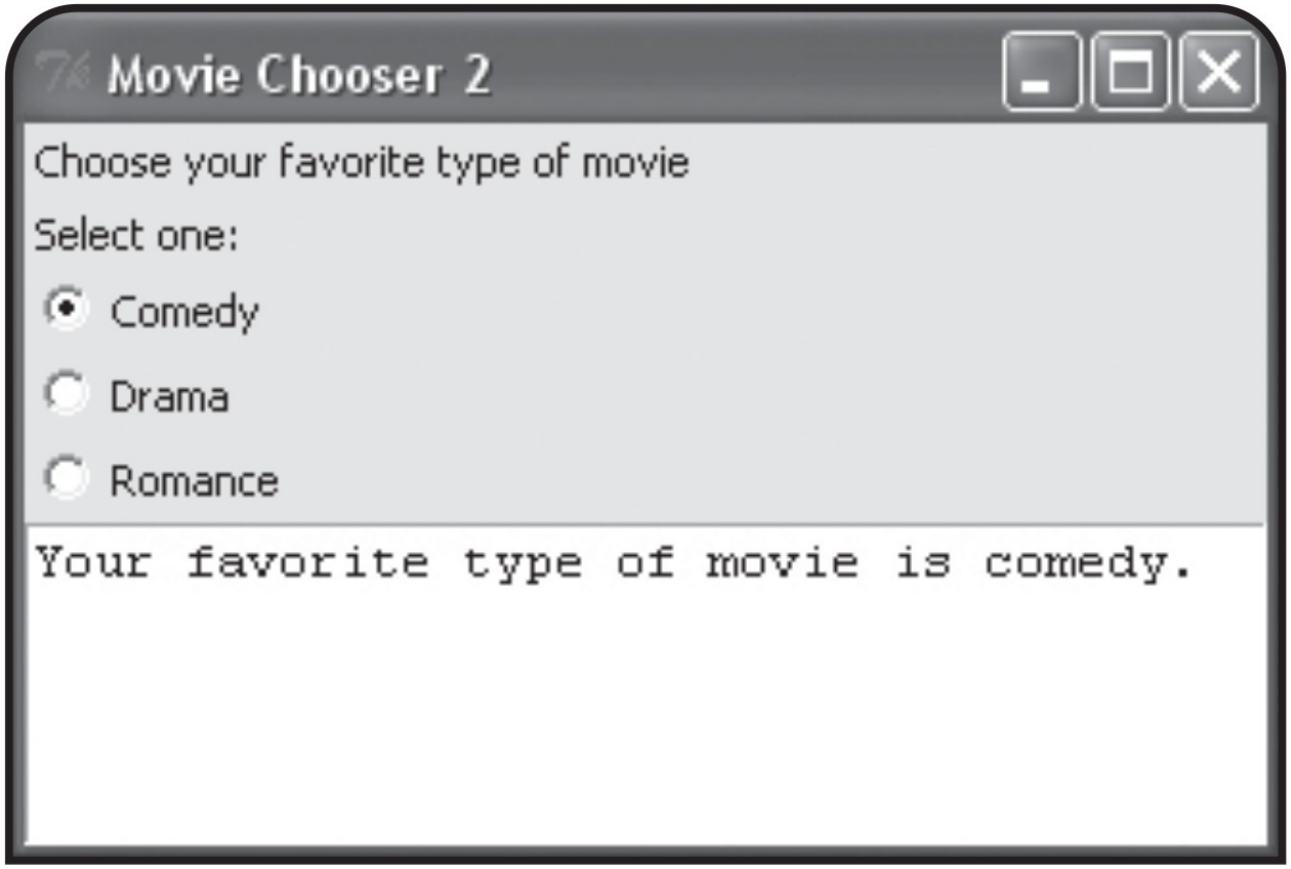
- Perform similar operations based on the status of the Drama and Romance check buttons.

- Delete all of the text in the text box and then insert the new string, likes.

Introducing the Movie Chooser 2 Program

The batch file: movie_chooser2.bat

```
movie_chooser2.py
pause
```



movie_chooser2.py

```
# Movie Chooser 2
# Demonstrates radio buttons

from tkinter import * class

Application(Frame):
    """ GUI Application for favorite movie type. """
    def __init__(self, master):
    """ Initialize Frame. """
    super(Application, self).__init__(master)
    self.grid()
    self.create_widgets()

def create_widgets(self):
    """ Create widgets for movie type choices. """
    # create description label
    Label(self, text = "Choose your favorite type of \
    movie").grid(row = 0, column = 0, sticky = W)
```

```
# create instruction label
Label(self, text = "Select one:"\
      ).grid(row = 1, column = 0, sticky = W)

# variable for single, favorite type of movie
self.favorite = StringVar()
self.favorite.set(None)

# create Comedy radio button
Radiobutton(self, text = "Comedy",
            variable = self.favorite, value = "comedy.",
            command = self.update_text\
            ).grid(row = 2, column = 0, sticky = W)

# create Drama radio button
Radiobutton(self, text = "Drama",
             variable = self.favorite, value = "drama.",
             command = self.update_text\
             ).grid(row = 3, column = 0, sticky = W)
```

```
        # create Romance radio button
        Radiobutton(self, text = "Romance",
                    variable=self.favorite, value="romance.",
                    command = self.update_text\
                    ).grid(row = 4, column = 0, sticky = W)

        # create text field to display result
        self.results_txt = Text(self, width=40, height=5,
                                wrap = WORD)
        self.results_txt.grid(row=5, column=0,
                                columnspan=3)

def update_text(self):
    """ Update & display user's favorite movie type""" message
    = "Your favorite type of movie is " message +=
    self.favorite.get()

    self.results_txt.delete(0.0, END)
    self.results_txt.insert(0.0, message)
```

```
# main
root = Tk()
root.title("Movie Chooser 2")
app = Application(root)
root.mainloop()
```

Creating Radio Buttons

- Since only one radio button in a group can be selected at one time, there's no need for each radio button to have its own status variable, as required for check buttons.

- A group of radio buttons share one, special object that reflects which of the radio buttons is selected. This object can be an instance of the **StringVar** class from the tkinter module, which allows a string to be stored and retrieved.

- Before we create the radio buttons, we create a single **StringVar** object for all of the radio buttons to share, assign it to the attribute favorite, and set its initial value to None using the object's **set()** method:

```
self.favorite = StringVar()
self.favorite.set(None)
```

- Create the Comedy radio button:

```
Radiobutton(self, text = "Comedy",
            variable = self.favorite, value = "comedy.",
            command = self.update_text\
            ).grid(row = 2, column = 0, sticky = W)
```

- A radio button's variable option defines the special variable associated with the radio button, while a radio button's value option defines the value to be stored by the special variable when the radio button is selected.

- By setting this radio button's variable to self.favorite and its **value** to "comedy." , we're saying that when the Comedy radio button is selected, the **StringVar** referenced by self.favorite should store the string "comedy."

- Create the other 2 radio buttons:

```
Radiobutton(self, text = "Drama",
            variable = self.favorite, value = "drama.",
            command = self.update_text\
            ).grid(row = 3, column = 0, sticky = W)

Radiobutton(self, text = "Romance",
            variable = self.favorite, value="romance.",
            command = self.update_text\
            ).grid(row = 4, column = 0, sticky = W)
```

- By setting the Drama/Romance radio button's variable to self.favorite and its **value** to "drama."/"romance.", when the Drama/Romance radio button is selected, the StringVar referenced by self.favorite should store "drama."/"romance."

- Create the text box to display the results:

```
self.results_txt = Text(self, width = 40, height = 5,
                                          wrap = WORD)
self.results_txt.grid(row=5,column=0,columnspan=3)
```

Getting a Value from a Group of Radio Buttons

- Getting a value from a group of radio buttons is to invoke the get() method of the StringVar object that they all share:

```
def update_text(self):
    message = "Your favorite type of movie is "
    message += self.favorite.get()
```

- When the Comedy/Drama/Romance radio button is selected, self.favorite.get() returns "comedy."/"drama."/ "romance.".

- Delete any text in the text box and insert the string which declares the user's favorite movie type:

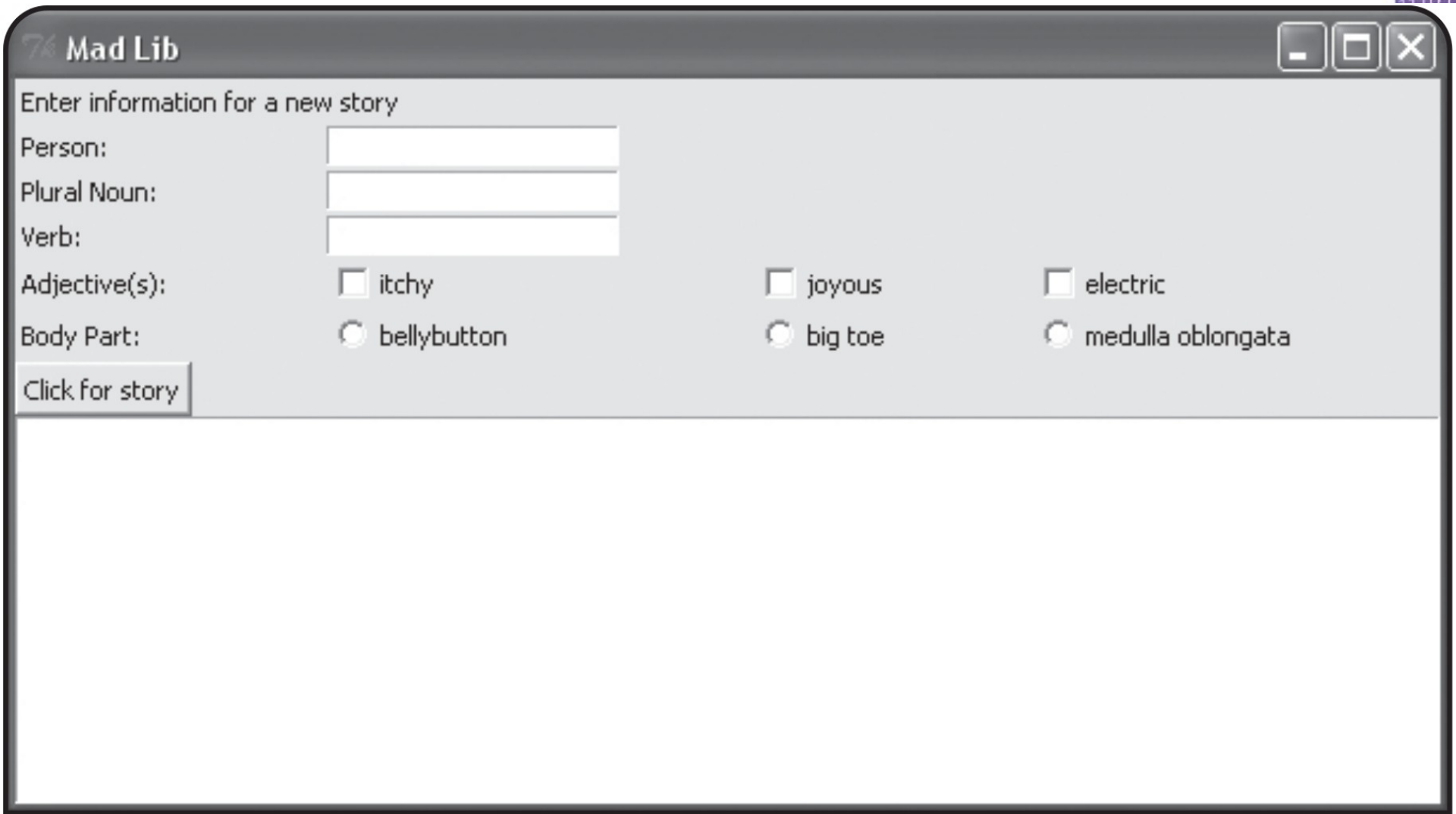
```
    self.results_txt.delete(0.0, END)
    self.results_txt.insert(0.0, message)
```

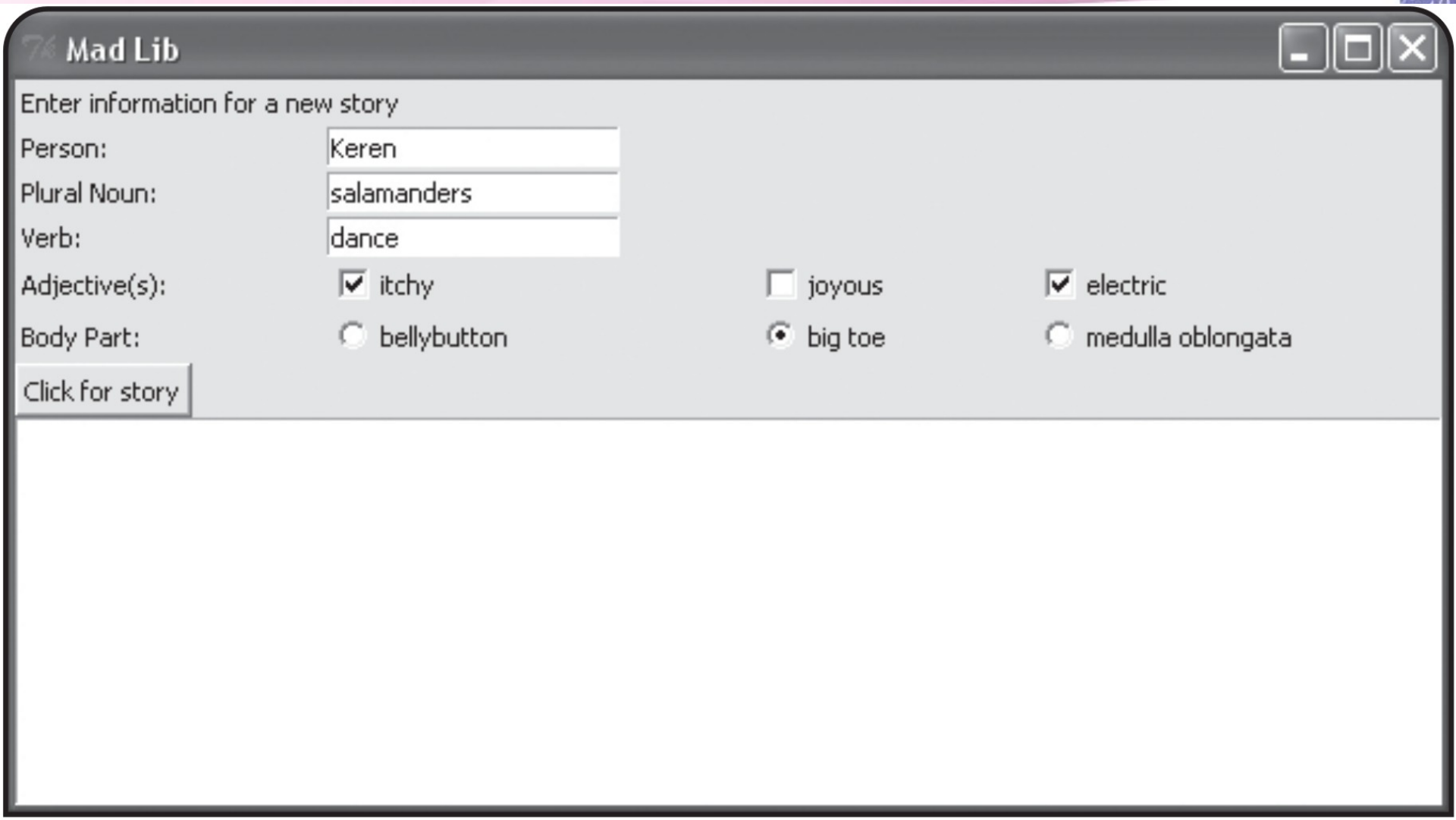
Introducing the Mad Lib Program

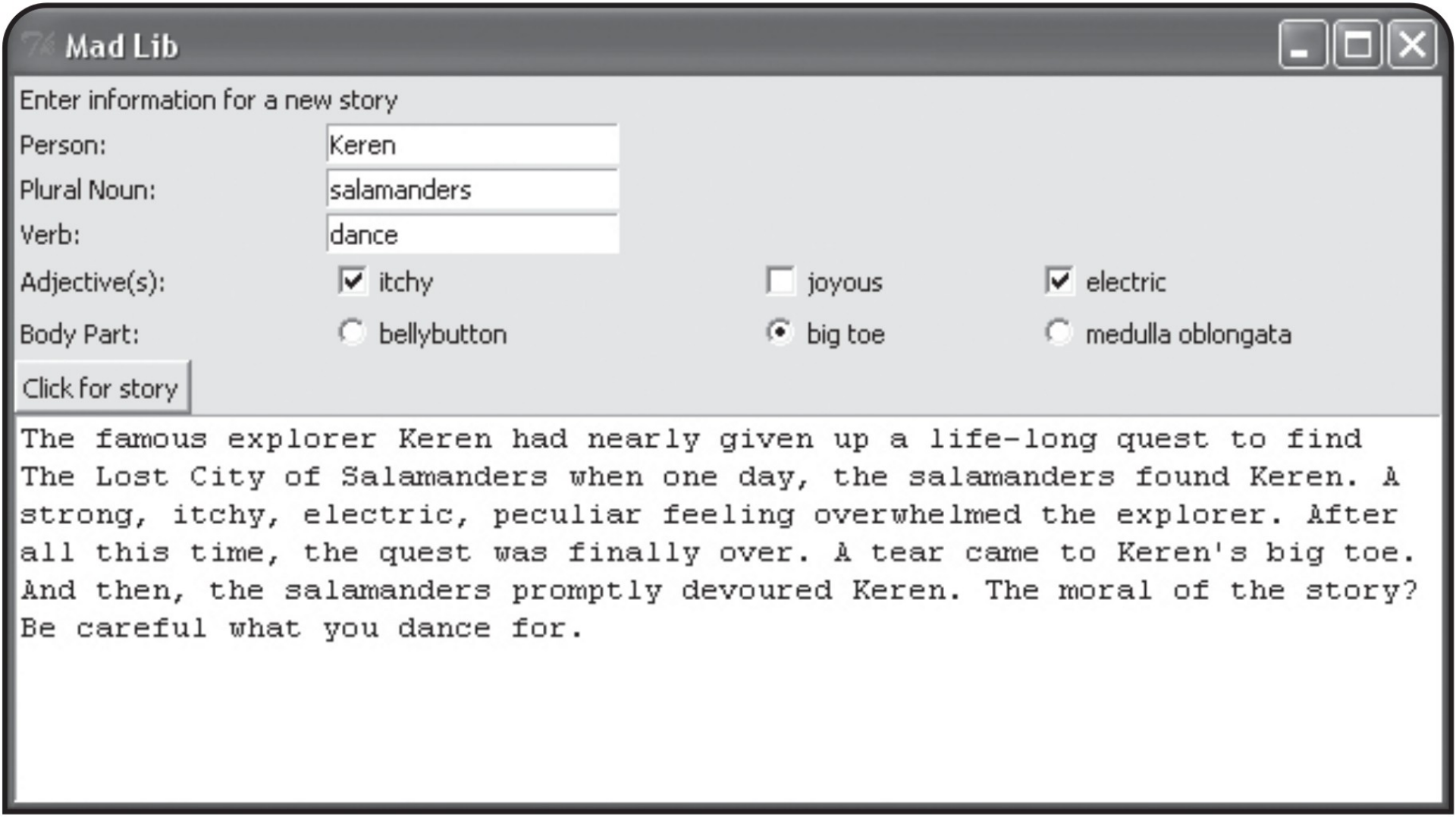
The batch file: mad_lib.bat

mad_lib.py

pause




Mad Lib
Enter information for a new story
Person:
Keren
Plural Noun:
salamanders
Verb:
dance
Adjective(s):
itchy
joyous
electric
Body Part:
bellybutton
big toe
medulla oblongata
Click for story


Mad Lib
Enter information for a new story
Person:
Keren
Plural Noun:
salamanders
Verb:
dance
Adjective(s):
itchy
joyous
electric
Body Part:
bellybutton
big toe
medulla oblongata
Click for story
The famous explorer Keren had nearly given up a life-long quest to find The Lost City of Salamanders when one day, the salamanders found Keren. A strong, itchy, electric, peculiar feeling overwhelmed the explorer. After all this time, the quest was finally over. A tear came to Keren's big toe. And then, the salamanders promptly devoured Keren. The moral of the story? Be careful what you dance for.

mad_lib.py

```
# Mad Lib
# Create a story based on user input

from tkinter import * class

Application(Frame):
    """ GUI app creates a story based on user input. """
    def __init__(self, master):
            """ Initialize Frame. """
            super(Application, self). init(master)
            self.grid()
            self.create_widgets()
    def create_widgets(self):
        """Create widgets for story info & display. """
        # create instruction label
        Label(self, text="Enter information for a new story"\
                ).grid(row=0,column=0,columnspan=2,sticky=W)
```

```
# create a label & text entry for the person's name.
Label(self, text = "Person: "\
      ).grid(row = 1, column = 0, sticky = W)
      self.person_ent = Entry(self)
      self.person_ent.grid(row = 1, column=1, sticky = W)

# create a label and text entry for a plural noun
Label(self, text = "Plural Noun:"\
      ).grid(row = 2, column = 0, sticky = W)
self.noun_ent = Entry(self)
self.noun_ent.grid(row = 2, column = 1, sticky = W)

# create a label and text entry for a verb
Label(self, text = "Verb:"\
      ).grid(row = 3, column = 0, sticky = W)
self.verb_ent = Entry(self)
self.verb_ent.grid(row = 3, column = 1, sticky = W)

# create a label for adjectives check buttons
Label(self, text = "Adjective(s):"\
      ).grid(row = 4, column = 0, sticky = W)
```

```
# create itchy check button
self.is_itchy = BooleanVar()
Checkbutton(self,text="itchy",variable=self.is_itchy\
        ).grid(row = 4, column = 1, sticky = W)

# create joyous check button
self.is_joyous = BooleanVar()
Checkbutton(self, text = "joyous",
        variable = self.is_joyous\
        ).grid(row = 4, column = 2, sticky = W)

# create electric check button
self.is_electric = BooleanVar()
Checkbutton(self, text = "electric",
        variable = self.is_electric\
        ).grid(row = 4, column = 3, sticky = W)

# create a label for body parts radio buttons
Label(self, text = "Body Part:"\
    ).grid(row = 5, column = 0, sticky = W)
```

```
# create variable for single, body part
self.body_part = StringVar()
self.body_part.set(None)

# crSeate body part radio buttons
body_parts = ["bellybutton", "big toe",
                        "medulla oblongata"]
column = 1
for part in body_parts:
    Radiobutton(self, text=part,
    variable=self.body_part, value=part).grid(row=5,
    column=column, sticky=W)
    column += 1

# create a submit button
Button(self, text = "Click for story",
        command = self.tell_story\
        ).grid(row = 6, column = 0, sticky = W)
self.story_txt=Text(self,width=75,height=10,
                                        wrap=WORD)
self.story_txt.grid(row=7,column=0, columnspan=4)
```

```
def tell_story(self):
    """Fill text box with story based on user input. """
    # get values from the GUI
    person = self.person_ent.get()
    noun = self.noun_ent.get()
    verb = self.verb_ent.get()
    adjectives = ""
    if self.is_itchy.get():
        adjectives += "itchy, "
    if self.is_joyous.get():
        adjectives += "joyous, "
    if self.is_electric.get():
        adjectives += "electric, "
    body_part = self.body_part.get()

    # create the story
    story = "The famous explorer " story
    += person
    story += " had nearly given up a life-long quest” + \ “ to find
                    The Lost City of "
    story += noun.title()
```

```
story += " when one day, the "
story += noun
story += " found "
story += person + ". "
story += "A strong, "
story += adjectives
story +="weird feeling overwhelmed the explorer. "
story +="After all this time, the quest was “ +\
            “finally over. A tear came to "
story += person + "'s "
story += body_part + ". "
story += "And then, the "
story += noun
story += " promptly devoured "
story += person + ". The "
story += "moral of the story? Be careful what you"
story += verb
story += " for."
```

```
        # display the story
        self.story_txt.delete(0.0, END)
        self.story_txt.insert(0.0, story)

# main
root = Tk()
root.title("Mad Lib")
app = Application(root)
root.mainloop()
```

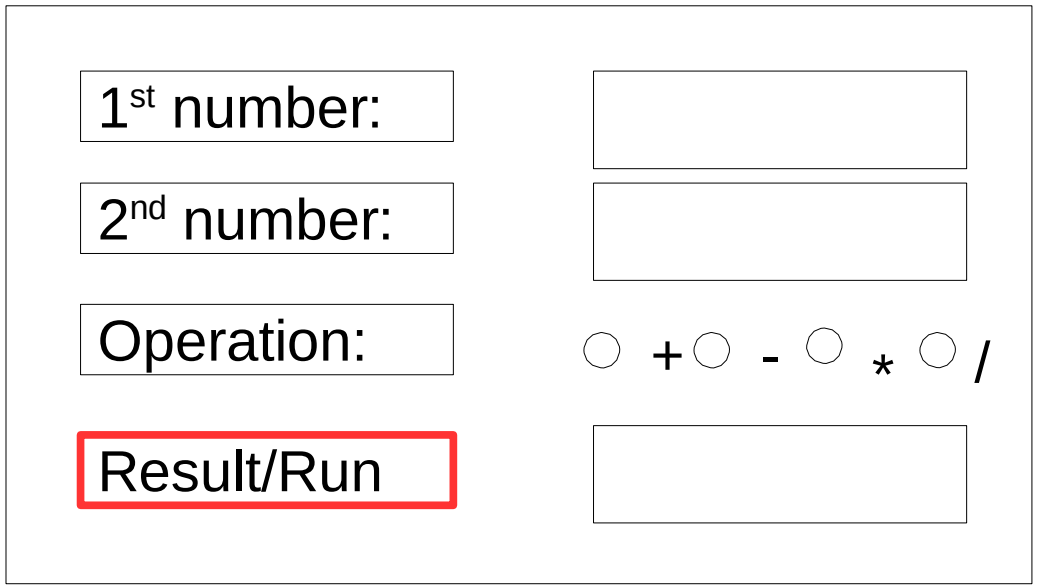
The Application Class's create_widgets()

- This class creates all of the widgets in the GUI. The only new thing is to create all 3 radio buttons in a loop by moving through a list of strings for each radio button's text and value options. (See codes.)

The Application Class's tell_story()

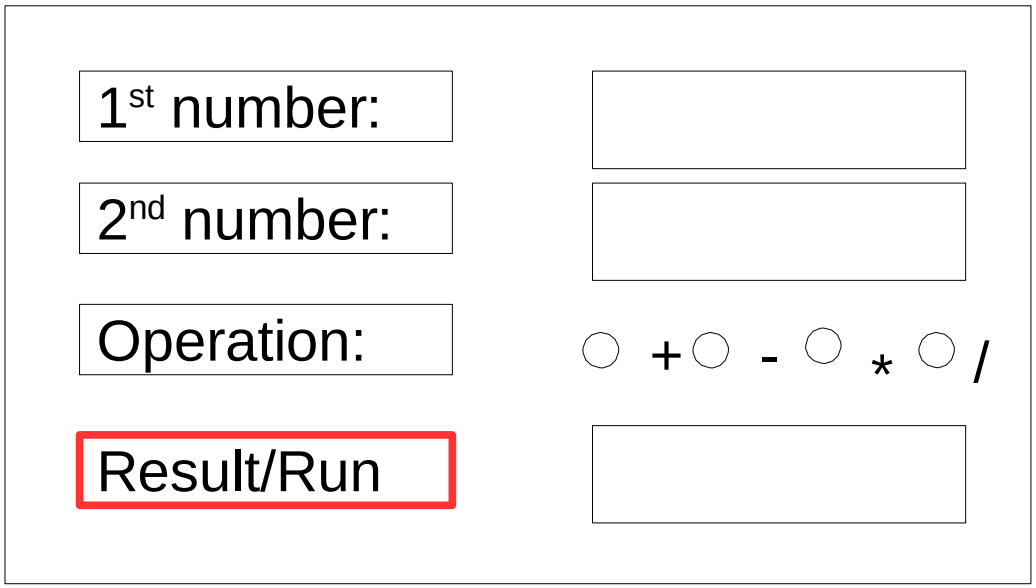
- In this method, we get the values the user has entered and use them to create the one, long string for the story. Then, we delete any text in the text box and insert the new string to show the user the story he or she created. (See codes.)

Quiz 11: Use the tkinter module to design a simplistic calculator. For example,



Homework 10

Use the tkinter module to design a simplistic calculator.
For example:



Homework 10

- Use Tkinter and matplot to design a GUI with the user inputs: amplitude A, frequency f, phase_shift ϕ, t_min, t_max, # of points, to generate corresponding sine plot.

$$f(t) = A \cdot cos(2\pi f \cdot t + \varphi)$$

Hint:

- Please refer FigureCanvasTkAgg
- The following examples may be useful:
  - https://fishark.pixnet.net/blog/post/47333844
  - https://www.twblogs.net/a/5e53154bbd9eee2116822a3b