Easiest to Highest

Python

Chapter 4: Graphical User Interfaces

All the scripts considered so far are either dialog-based user interactive or non-user interactive. Dialog-based user interactive scripts guide users to respond to the prompted questions. One-step advanced user-interactive programs are to use graphic components. Examples of graphic components include buttons, textbox, labels, scroll bars, etc. The Python scripts that use graphic components for user interactions are called graphical user interfaces- (GUI-) based script.

This chapter describes various ways of GUI-based Python scripting.

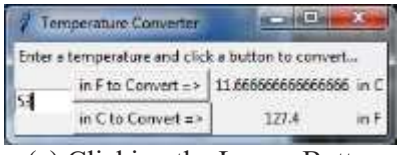
3.1 Graphic Components

The GUI module package provided by Python is called `tkinter`. The module `tkinter` should be imported in scripts and the function `Tk()` instantiates a top level graphic component. Before discussing about graphic components available in tkinter, we will consider a simple GUI example.

EXAMPLE 4.1: Recall a script, `My7loopTemp.py`, converting Celsius temperature to Fahrenheit temperature in EXAMPLE 2.6. A similar computation can be displayed on GUI. Write a GUI script, `gui1temp.py`, which can convert a given number to both Celsius and Fahrenheit.

(a) Initial GUI

(b) Click the Upper Button



(c) Clicking the Lower Button also

The script code is as follows:

Script `gui1temp.py`

```
(1)  from tkinter import *

(2)  def convert2c():
(3)    try:
(4)      f = float(tempIN.get())
(5)      c = (f -32)*5.0/9.0
(6)      tempOUTc.set(c)
(7)    except ValueError:
(8)      pass

(9)  def convert2f():
(10)   try:
(11)     c = float(tempIN.get())
(12)     f = c*9.0/5.0 +32
(13)     tempOUTf.set(f)
(14)   except ValueError:
(15)     pass

(16) root = Tk()
(17) f=root
(18) root.title("Temperature Converter")
(19)
```

```
(20) tempIN = StringVar()
(21) tempOUTc = StringVar()
(22) tempOUTf = StringVar()

(23) Label(f, text="Enter a temperature and click a button to
     convert...").grid(columnspan=4,row=0,sticky=W)
(24) Entry(f, textvariable=tempIN, width=7).grid(column=0, rowspan=2, row=1,
     sticky=W)
(25) Button(f, text="in F to Convert =>", command=convert2c).grid(column=1, row=1,
     sticky=W)
(26) Button(f, text="in C to Convert =>", command=convert2f).grid(column=1, row=2,
     sticky=W)
(27) Label(f, textvariable=tempOUTc).grid(column=2, row=1, sticky=(W, E)
(28) Label(f, textvariable=tempOUTf).grid(column=2, row=2, sticky=(W, E))
(29) Label(f, text="in C").grid(column=3, row=1, sticky=W)
(30) Label(f, text="in F").grid(column=3, row=2, sticky=W)

(31) root.mainloop()
```

As shown above, the module `tkinter` should be imported in line (1) to render the graphic components, the classes `Label`, `Entry`, `Button`, etc. Those classes of graphic component are already defined and provided. Note that Label is a Python class and `Label()` is in an object constructor form, in which form, the class `Label` can be instantiated. What we can do in this example is to instantiate those classes. As a class is instantiated, some specific options can be defined by using arguments. For example, `Button (root, fg="red", bg="blue")`, where foreground and background colors can be specified.

The module `tkinter` has classes, `Tk` and `Tcl`. The class `Tk()` can be instantiated with no argument as shown in line (16). The two functions shown in lines (2)-(16) have been discussed in EXAMPLE 2.6. An object of `Tk()` is called the `root` of this GUI, which is the master or parent window. The graphic components of `Label()`, `Entry()` and `Button()` are contained in the master window, root. The object of `Tk()` first invokes the `title()` method to display the title of the GUI window in line (18). Renaming of a `Tk()` object in line(17) is redundant if the object reference `root` is used instead.

In order to refer the variables that are used in the logic of temperature conversions, the variables shown in graphic components are declared in lines (20)-(22). `tempIN`, `tempOUTf`, and `tempOUTc` are the variables holding the user input temperature, the conversion output to Fahrenheit, and the conversion output to Celsius, respectively. Those graphic components are then organized in grid layout, 4 columns and 3 rows.

On the first row, the text appears over those all four columns in left alignment by the statement `sticky=W`. Columns are spanned by `columnspan=4` in line (23). The second and third rows have four columns, the input textbox, button to execute a conversion method, the output text, and the label to display the temperature unit, from left to right. User input can be received by the Entry() class in line (24). Here the variable `tempIN` is defined in this graphic component by `textvariable=tempIN`, which will be used in conversion processes as shown in lines (4) and (11).

Then, two buttons are created in lines (25) and (26). As a button is clicked, a corresponding method is executed as specified by `command=convert2c` and `command=convert2f`. In lines (27) and (28), the output `tempOUTf` or `tempOUTc` of those temperature conversion methods is displayed. The value held in such a variable is centralized in horizon alignment by `sticky=(W,E)`. Finally, the units of temperature are shown in the `Label()` objects.

All of these graphic components are continued to run by `root.mainloop()`. Detailed description about `tkinter` can be found at https://docs.python.org/3.4/library/tkinter.html.

EXERCISE 4.1: Write a Python script, `grid1temp2.py`, which has a few additions to the script, `grid1temp.py` in EXAMPLE 4.1. A sample run illustrates below:

The initial GUI shown above can be expanded as the conversion outcome is displayed.

Hint: Font and color can be defined as the class `Entry` and `Label` are instantiated. Note that the font can defined for font face (e.g., `Helvetica`), size (e.g., `24`) and style (e.g., `bald`), and the color can be changed by adding `fg="#xxxxxx"` for foreground color setting, which is a RGB code in Hex.

3.2 Menus

The module `tkinter` provides extended widgets, e.g., `Menu`, referring to http://www.tutorialspoint.com/python/tk_menu.htm. The following example illustrates a pulldown menu where a specific menu is selected from the menu pulled-down and a corresponding method is executed.

EXAMPLE 4.2: Write a script, `menu2pulldown0.py`, that displays pull-down menus for file editing. The following shows a sample coding and run.

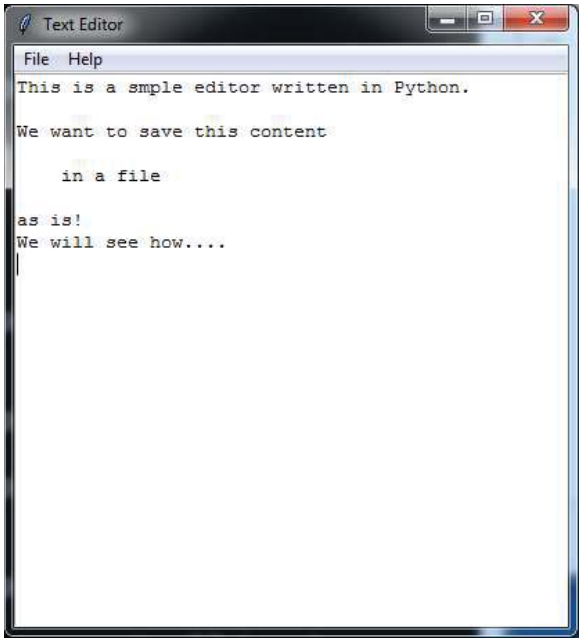
Script `menu2pulldown0.py`

```
(1)  from tkinter import *
```

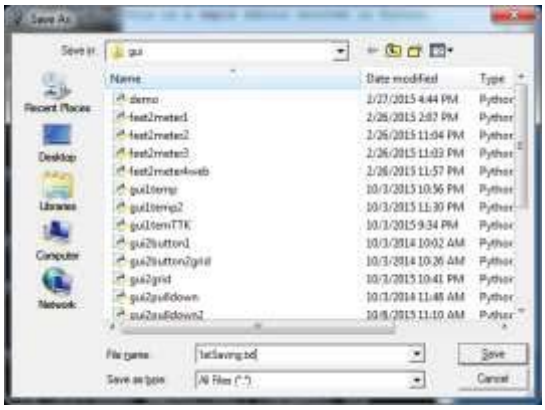
```
(2)  import tkinter as tk
(3)  from tkinter.filedialog import asksaveasfile
(4)  from tkinter.filedialog import askopenfile

(5)  root = tk.Tk()

(6)  fram = tk.Frame(root, width=400, height=400)
(7)  fram.pack_propagate(0)
(8)  fram.pack(expand=YES, fill=BOTH)

(9)  editor = tk.Text(fram)
(10) editor.pack_propagate (0)
(11) editor.pack(expand=YES, fill=BOTH)

(12) root.title("Text Editor")

(13) def file_save():
(14)   global editor
(15)   f = asksaveasfile(mode='w', defaultextension=".txt")
(16)   if f is None: # asksaveasfile return `None` if dialog closed with "cancel".
(17)       return
(18)   text2save = str(editor.get(1.0, END)) # starts from `1.0`, not `0.0`
(19)   f.write(text2save)
(20)   f.close()

(21) def createFile():
(22)   print ("implement this")

(23) def openFile():
(24)   print ("implement this")

(25) def getOut():
(26)   root.quit()

(27) def explainHelp():
(28)   helpwin = Toplevel (root)
(29)   button = Button (helpwin, text ="This is my first menu-driven Python program
     \n" \
(30) " written for IASP 505.")
(31)   button.pack()

(32) myMenu = Menu(root)

(33) fileMenu = Menu (myMenu, tearoff=0)
(34) fileMenu.add_command(label="New", command=createFile, accelerator="Ctrl+N")
(35) fileMenu.add_command(label="Open...", command=openFile, accelerator="Ctrl+O")
(36) fileMenu.add_command(label="Save", command=file_save, accelerator="Ctrl+S")
(37) fileMenu.add_command(label="Save As...", command=file_save)
(38) fileMenu.add_separator()
(39) fileMenu.add_command(label="Exit", command=getOut)

(40) myMenu.add_cascade(label="File", menu=fileMenu)

(41) helpMenu = Menu (myMenu, tearoff=0)
(42) helpMenu.add_command(label="View Help", command=createFile)
(43) helpMenu.add_separator()
(44) helpMenu.add_command(label="About Notepad", command=explainHelp)
(45) myMenu.add_cascade(label="Help", menu=helpMenu)

(46) root.config (menu=myMenu)
(46) root.mainloop ()
```

Output from c:\python34\python `menu2pulldown0.py`

Initial Windows with “Saving As” menu



Then, the following is popped up to create and save a file:



Then, the file is created and saved.

The top level master window, called root, is instantiated by Tk(). From the master windows, Frame()is instantiated in line (6). Text() is instantiated within the frame instance in line (9), meaning that a Text instance, `editor`, is contained in a Frame instance, `fram`. Similarly, Menu is instantiated in line (32). The instance `myMenu` has `fileMenu` in line (40) and `helpMenu` in line (45) in a cascading alignment, and then `myMenu` object is contained in the master window where `Fram` is available in line (46). Each menu, fileMenu and helpMenu is created in lines (33) and (41), respectively. Pulldown menus are listed in each menu. For example, submenus about files are added to fileMenu as shown in lines (34)-(37) and line (39). In between, a separate line is added as shown in line (38).

The screen of a sample run is captured. The left screenshot shows how to edit and save it in a file. As the submenu “Save As” is selected, the windows explorer is popped up where you can change directory and name a file to save. In an example, the file saved is 1stSaving.txt in the upper right screenshot, which appears in the windows explorer in the lower right screenshot.

EXERCISE 4.2: Write a Python script, `menu2pulldn.py` to complete methods, `createFile()` and `openFile()`. Note that do not worry about the menus, Edit, Format, View, but complete File only.

EXAMPLE 4.3: Write a script, `search_tkText.py`, that takes a keyword from Entry(), and find all matches from the bottom text box. All matched words in the text box are now highlighted in red. The following shows a sample code and run.

Script `search_tkText.py`

```
(1)     from tkinter import *

(2)     root = Tk()

(3)     def match():
(4)         article.tag_remove('matched', '1.0', END)
(5)         article.tag_configure('matched', foreground='red', background='yellow')
(6)         kk = key.get()
(7)         if kk:
(8)            lastIndx = '1.0'
(9)            while True:
(10)             indx = article.search(kk, lastIndx, nocase=1, stopindex=END)
(11)             if not indx: break
(12)             lastIndx = '%s+%dc' % (indx, len(kk))
(13)             article.tag_add('matched', indx, lastIndx)

(14)    fram = Frame(root)
(15)    Label(fram,text='Keyword:').pack(side=LEFT)
(16)    key = Entry(fram)
(17)    key.pack(side=LEFT, fill=BOTH, expand=1)
(18)    btnMatch = Button(fram, text='Match')
(19)    btnMatch.pack(side=RIGHT)
(20)    fram.pack(side=TOP)

(21)    article = Text(root)
(22)    article.insert('1.0',"You may enter/override an article here ...")
(23)    article.pack(side=BOTTOM)

(24)    btnMatch.config(command=match)
(25)    root.mainloop()
```

Output from c:\python34\python `search_tkText.py`

Initial Windows after a few sentences are added

Highlighted matched words

In this example, a Text component is created and located bottom in the window as shown in lines (21) and (23). An initial text is given in line (22).

The key coding block in this example is the function for search and highlight in lines (3)-(13). Entry and Text are instantiated to object references, key and article, to read keyword and to read and match a full text, respectively. In line (6), a keyword which is entered by users is read. Note that a Text instance can invoke the method (or function), `search()` in line (10), which will then return the index (or the position) of a matched word. A text object can invoke the method `search()` by arguments starting a search keyword, a start position followed by variable and value pairs, for example, `nocase=0` for case sensitive; `nocase=1` for case nonsensitive; `stopindex=END` for searching all the way to the end of the text. Then, the method returns the start column position of the matched word. This position number is held by a variable `indx` in line (10) for example.

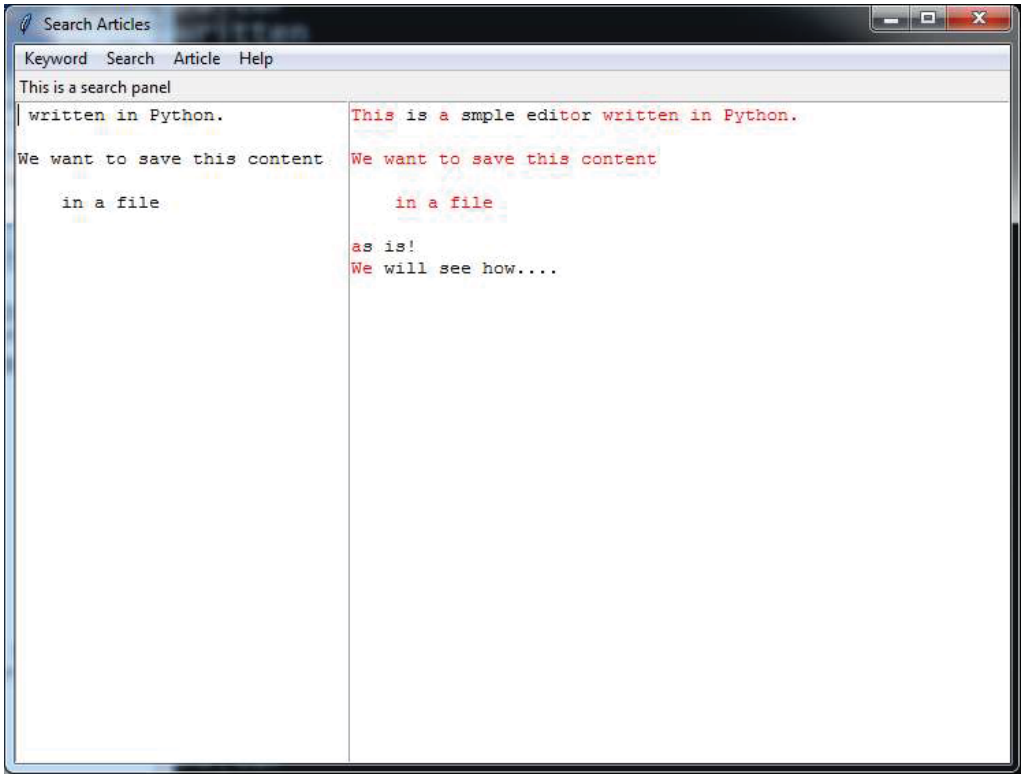
If no matched, meaning that `indx` is null, then the while loop stops.

With the starting position of a matched word, line (12) computes the ending position of the matched word. In line (13), these two numbers, starting and ending positions, a new tag which is highlighted according to the configuration in line (5).

For detailed explanation on the Text wedget, please check out the website: http://effbot.org/tkinterbook/text.htm.

It is possible that the menu and graphic components are contained in the same root windows. The following example illustrates how they can be combined together to display GUI more effectively.

EXERCISE 4.3: Write a Python script, `menu2pulldn4search.py` to extend `menu2pulldown.py` to render an image file. A sample run is as follows:

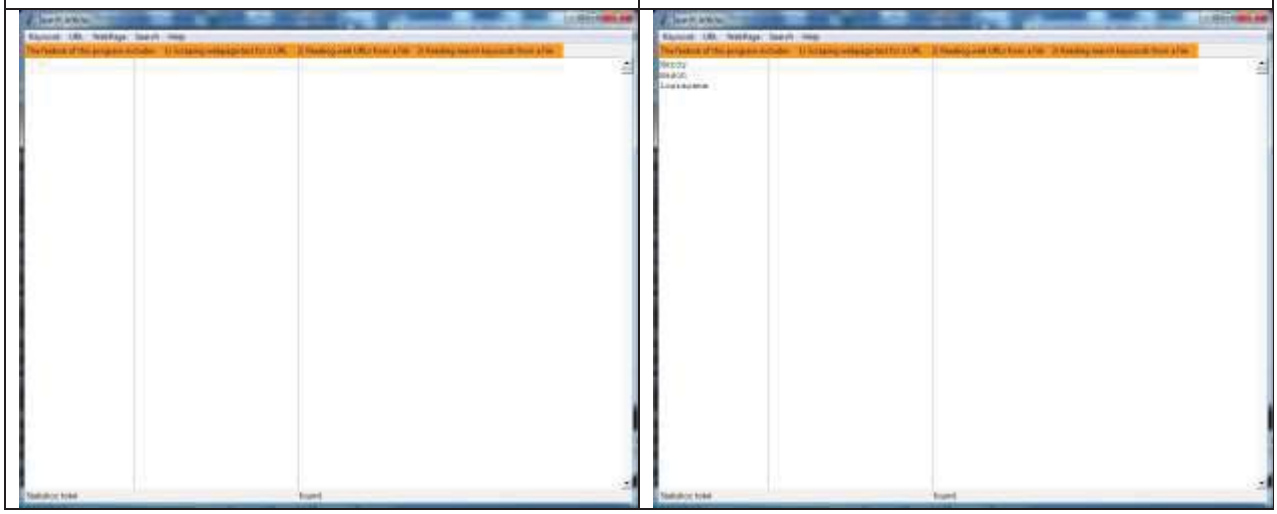


There are four pulldown menus: Keyword, Search, Article and Help. Both Keyword and Article can open two different files. The Search menu read the text from the left text box. Each keyword in the left text box is searched from the text in the right text box. If matched, the word in the right text box is highlighted in red.

3.3 Applications

In this section, one of the GUI programs that can access webpages and searches keywords is illustrated. Let's first see the steps of a run.

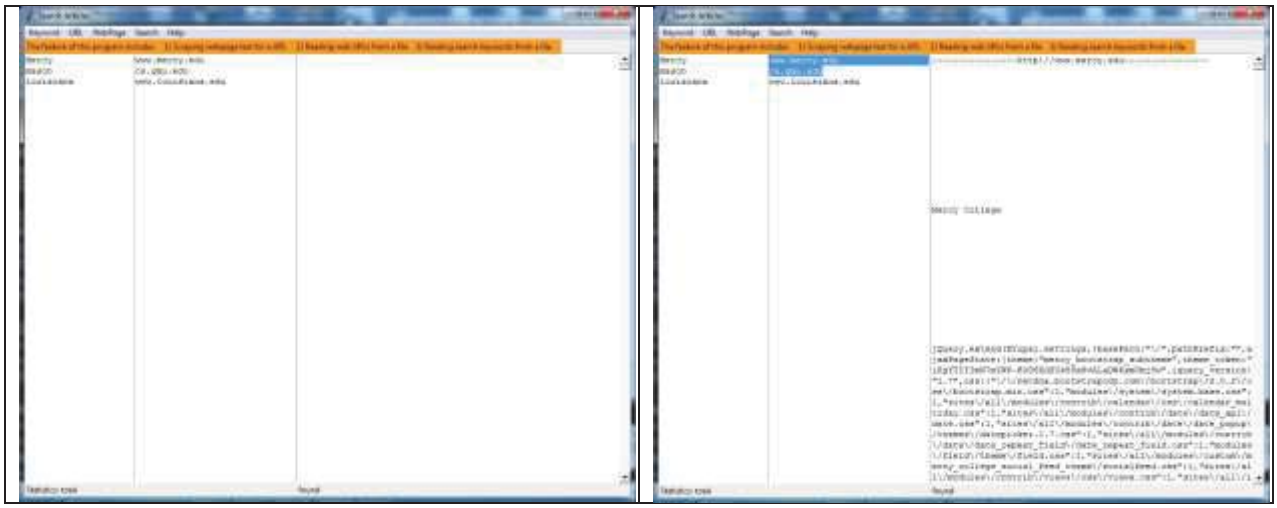
The given initial GUI has four of the pulldown menus on top, which appears on the left. The first keyword menu can open a keyword file and display a set of keywords on the right below.



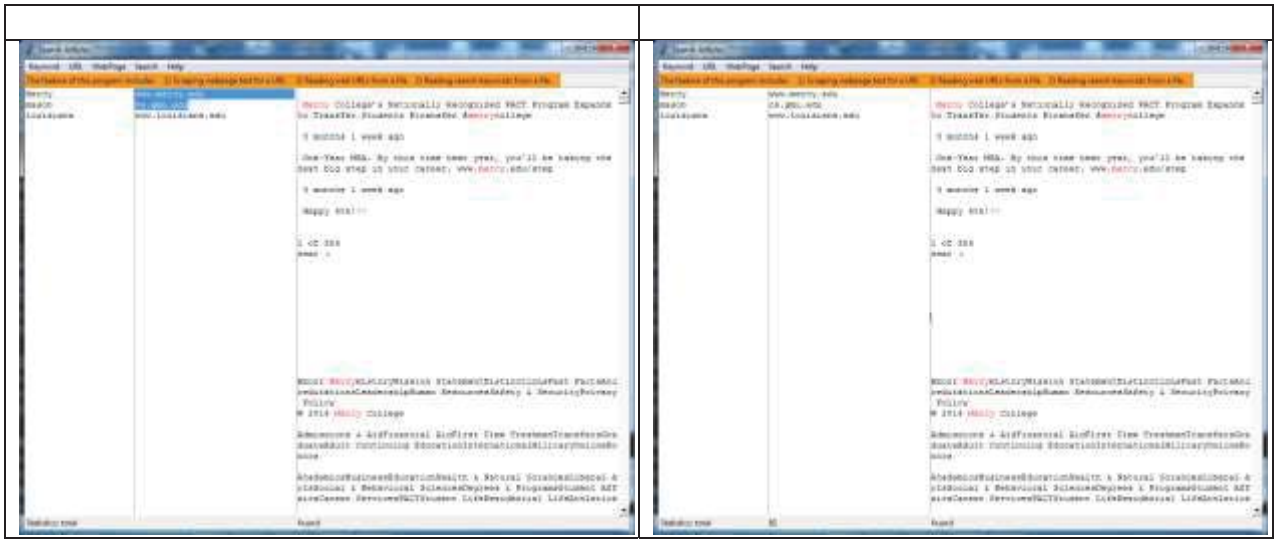
Similarly, the second pulldown menu can list the URLs from a file that is selected from the windows explorer. Of the list, only highlighted URLs are accessed and the webpages are fetched. To access a given web site, the module lib.parse needs to be imported. Then, a given URL, a method, urllib.request.urlopen(a url) returns the HTML content of the url.

Since webpages are written in HTML, they can be converted into plaintext. Web contents in text are then displayed on the right.

We may import a module, BeautifulSoup4 to convert HTML code to text. Of a few ways, you may download and install using pip install BeautifulSoup4. BeautifulSoup is instantiated with a given HTML file will be an object holding the converted text.



Then, against the web contents on the right, a search with the keyword will be conducted. Only matched words are highlighted in the web contents on the left. In the pulldown menu of Search, the Statistics is shown on the bottom row as shown on the right.



The third column content can then be stored in a file.

Script `gui1temp.py`

```
(1)  from tkinter import *
(2)  import tkinter as tk
(3)  from tkinter.filedialog import asksaveasfile
(4)  from tkinter.filedialog import askopenfile
(5)  import urllib.request
(6)  from bs4 import BeautifulSoup as bs

(7)  root = tk.Tk()

(8)  root.title("Search Articles")

(9)  Label(root,text="The feature of this program includes  \
(10)    1) Scraping webpage text for a URL  \
```

```
(11)    2) Reading web URLs from a file \
(12)    3) Reading search keywords from a file \
(13)    ", bg="#ff9911").grid(columnspan=4,row=0,sticky=W)

(14) search = tk.Text(root, width=20, height=40)
(15) search.grid(column=0,row=1,sticky=W)

(16) url = tk.Text(root, width=30, height=40)
(17) url.grid(column=1,row=1,sticky=(W,E))

(18) article = tk.Text(root, width=60, height=40)
(19) article.grid(column=2,row=1)#,sticky=W)

(20) cntOUT=StringVar()
(21) tk.Label(root,text="Statistics: total").grid(column=0,row=2, sticky=W)
(22) tk.Label(root, textvariable=cntOUT).grid(column=1,row=2,sticky=W)
(23) tk.Label(root,text="found").grid(column=2,row=2, sticky=W)

     # scroll bar can be defined here

(24) def appendKey():
          # implement here

(25) def createKey():
(26)   search.delete(1.0,END)

(27) def openKey():
(28)   global search
(29)   f = askopenfile(mode='r')
(30)   if f is None:
(31)      return
(32)   text2open = str(f.read())
(33)   search.insert(INSERT,text2open)

(34) def createURL():
(35)   url.delete(1.0,END)

(36) def openURL():
(37)   global aurl
(38)   f = askopenfile(mode='r')
(39)   if f is None:
(40)      return
(41)   text2open = str(f.read())
(42)   url.delete(1.0,END)
(43)   url.insert(INSERT,text2open)

(44) def createFile():
(45)   article.delete(1.0,END)

(46) def file_save():
          # implement here

(47) def readAll():
          # implement here

(48) def readOne():
(49)   global article
(50)   article.delete(1.0,END)
(51)   readAddr = str(url.selection_get())
(52)   for ur in readAddr.split():
(53)     urladdr = "http://"+ur
(54)     print (urladdr)
(55)     urladd = "----------------" + urladdr + "----------------"
(56)     article.insert(END,urladd)
(57)     htmlFile = urllib.request.urlopen(urladdr)
(58)     soup = bs(htmlFile)
(59)     article.insert(END,soup.get_text())

(60) def caseHighlight():
          # implement here
```

```
(61) def nocaseHighlight():
          # implement here

(62) def searchStatistic():
(63)   global cnt
(64)   try:
(65)     cntOUT.set(cnt)
(66)   except ValueError:
(67)     pass

(68) def getOut():
(69)   print("Good-bye")
(70)   root.quit()

(71) def explainHelp():
(72)   helpwin = Toplevel (root)
(73)   button = Button (helpwin, text ="This is a sample program \n" \
(74) " written for IASP 505.")
(75)   button.pack()

(76) myMenu = Menu(root)

(77) keyMenu= Menu (myMenu, tearoff=0)
(78) keyMenu.add_command(label="New", command=createKey, accelerator="Ctrl+N")
(79) keyMenu.add_command(label="Open...", command=openKey, accelerator="Ctrl+O")
(80) keyMenu.add_separator()
(81) keyMenu.add_command(label="Exit", command=getOut)
(82) myMenu.add_cascade(label="Keyword", menu=keyMenu)

(83) urlMenu= Menu (myMenu, tearoff=0)
(84) urlMenu.add_command(label="New", command=createURL, accelerator="Ctrl+N")
(85) urlMenu.add_command(label="Open...", command=openURL, accelerator="Ctrl+O")
(86) myMenu.add_cascade(label="URL", menu=urlMenu)

(87) artiMenu= Menu (myMenu, tearoff=0)
(88) artiMenu.add_command(label="Fetch Selected", command=readOne, accelerator="Ctrl+O")
(89) artiMenu.add_command(label="Fetch All", command=readAll, accelerator="Ctrl+S")
(90) artiMenu.add_command(label="Save As...", command=file_save)
(91) artiMenu.add_separator()
(92) artiMenu.add_command(label="Page Setup...", command=createFile)
(93) artiMenu.add_command(label="Print...", command=createFile, accelerator="Ctrl+P")
(94) artiMenu.add_separator()
(95) artiMenu.add_command(label="Exit", command=getOut)
(96) myMenu.add_cascade(label="WebPage", menu=artiMenu)

(97) srchMenu= Menu (myMenu, tearoff=0)
(98) srchMenu.add_command(label="Case-sensitive", accelerator="Ctrl+io", command=caseHighlight)
(99) srchMenu.add_command(label="All Matches", accelerator="Ctrl+io", command=nocaseHighlight)
(100)   srchMenu.add_separator()
(101)   srchMenu.add_command(label="Statistics", accelerator="Ctrl+ix", command=searchStatistic)
(102)   myMenu.add_cascade(label="Search", menu=srchMenu)

(103)   helpMenu= Menu (myMenu, tearoff=0)
(104)   helpMenu.add_command(label="View Help", command=createFile)
(105)   helpMenu.add_separator()
(106)   helpMenu.add_command(label="About Notepad", command=explainHelp)
(107)   myMenu.add_cascade(label="Help", menu=helpMenu)

(108)   root.config (menu=myMenu)
(109) root.mainloop ()
```

EXERCISE 4.4: Complete the Python script, `menu2pulldn4web.py`, whose sample run and coding list were discussed above.