UNIT-4

UNIT - IV

GUI Programming: Introduction, Tkinter and Python Programming, Brief Tour of Other GUIs, Related Modules and Other GUIs.

WEB Programming: Introduction, Wed Surfing with Python, Creating Simple Web Clients, Advanced Web Clients, CGI-Helping Servers Process Client Data, Building CGI Application Advanced CGI, Web (HTTP) Servers

GUI Programming:

Introduction:

What Is A Graphical User Interface (GUI)?

Graphical User Interface (GUI) is nothing but a desktop application which helps you to interact with the computers. They are used to perform different tasks in the desktops, laptops and other electronic devices.

✓**GUI** apps like **Text-Editors** are used to create, read, update and delete different types of files.

✓**GUI** apps like Sudoku, Chess and Solitaire are games which you can play.

✓**GUI** apps like **Google Chrome, Firefox and Microsoft Edge** are used to browse through the **Internet**.

Python Libraries To Create Graphical User Interfaces:

Python provides several different **options/libraries** for writing GUI based programs. These are listed below:

1. Tkinter: It is the easiest among all libraries. It is Python's **standard GUI** (Graphical User Interface) package. It is the most commonly used **toolkit** for GUI Programming in Python.

2. JPython: It is the **Python platform for Java** that is providing **Python scripts** seamless access o Java class Libraries for the local machine.

3. wxPython: It is open-source, cross-platform GUI toolkit **written in C++.** It one of the alternatives to Tkinter, which is bundled with Python.

There are many other **interfaces(Kivy ,Python QT**) available for **GUI.** Among all of these, **Tkinter** is **preferred** by a lot of learners and developers just because of how simple and easy it is.

What Are Tcl, Tk, and Tkinter?

✓**Tkinter** is Python's **default GUI library**. It is based on the **Tk toolkit**, originally designed for the **Tool Command Language (Tcl).**

✓Due to **Tk's popularity,** it has been ported to a variety of other scripting languages, including **Perl** (Perl/Tk**), Ruby** (Ruby/Tk), and **Python** (Tkinter).

✓**tkinter** is an inbuilt **Python** module used to create simple **GUI** apps. It is the most commonly used module for **GUI** apps in the **Python**.

✓You don’t need to worry about installation of the **Tkinter** module as it comes with **Python** default.

>>> **import Tkinter----------python 2.x**

>>> **import tkinter----------python 3.x**

If your Python interpreter **was not compiled with Tkinter** enabled, the module import fails:

```
>>> import Tkinter

    Traceback (innermost last):
    File "<stdin>", line 1, in ?
    File "/usr/lib/python1.5/lib-tk/Tkinter.py", line 8, in ?
    import _tkinter # If this fails your Python may not
    be configured for Tk
    ImportError: No module named _tkinter
```

✓**You may have to recompile your Python interpreter to get access to Tkinter.**

Tkinter and Python Programming:

Creating a GUI application using **Tkinter** is an easy task. All you need to do is perform the following steps −

1. **importing tkinter**

   It is same as importing any other module in the python code. Note that the name of the module in Python 2.x is 'Tkinter' and in Python 3.x is 'tkinter'.

   import tkinter

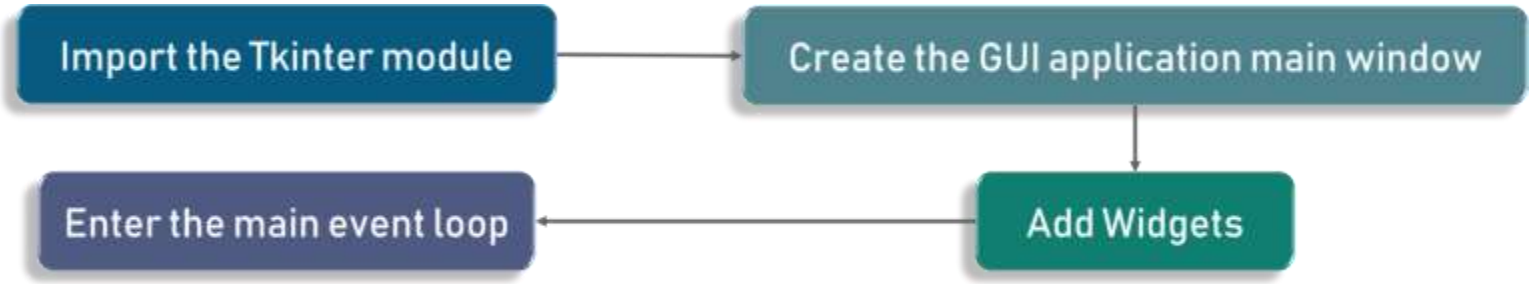
2. **Create the GUI application main window.**

   we are performing **operations and displaying visuals** and everything basically.

3. **Add one or more of the widgets to the GUI application.**

4. **Enter the main event loop to take action against each event triggered by the user.**

   An event loop is basically telling the code **to keep displaying the window** until we **manually** close it. It runs in an infinite loop in the back-end.



To create a simple window with the text **Hello World!**.

```
import tkinter
window = tkinter.Tk()
window.title("GUI")                    # to rename the title of the window
    # pack is used to show the object in the window
label = tkinter.Label(window, text = "Hello World!").pack()
window.mainloop()
```

Steps:-

1. import the module **tkinter**.
2. Initialize the window manager with the **tkinter.Tk()** method and assign it to a variable **window**. This method creates a blank window with **close, maximize and minimize buttons.**

   Tk(screenName=None, baseName=None, className='Tk', useTk=1)

   To create a main window, tkinter offers a method ,to change the **name of the window**, you can change the **className** to the desired one.

   The basic code used to create the main window of the application is-

   window = tkinter.Tk()

3.(optional) Rename the title of the window as you like with the

window.title(title_of_the_window).

4. **Label** is used to insert some objects into the **window**. Here, we are adding a **Label** with some text.
   - ✓ **pack()** attribute of the widget is used to display the **widget** in a size it requires.

5. Finally, the **mainloop()** method to display the **window** until you manually close it.

Tk Widgets:

Widgets are something like elements in the **HTML**. You will find different types of **widgets** to the different types of elements in the **Tkinter**.

1. **Button :**

It is used to add buttons in a Python application.

w = Button (master, option=value, ...)

master − This represents the parent window.

options − Here is the list of most commonly used options for this widget.

These options can be used as key-value pairs separated by commas.

Command--Function or method to be called when the button is clicked.

Activebackground--Background color when the button is under the cursor.

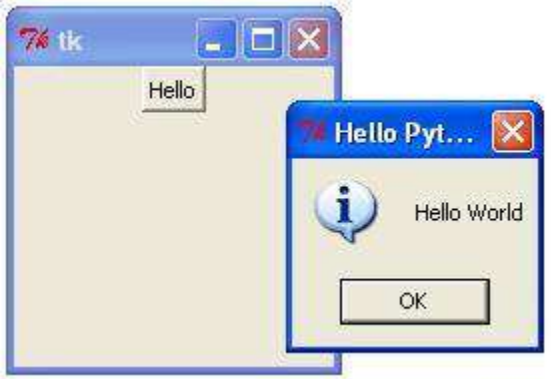
Activeforeground---Foreground color when the button is under the cursor.

Bd-----Border width in pixels. Default is 2.

Bg--Normal background color.

Ex:

```
 import tkinter
import tkinter.messagebox
top = tkinter.Tk()

def helloCallBack():
  messagebox.showinfo( "Hello Python", "Hello World")

B = tkinter.Button(top, text ="Hello", command = helloCallBack)

B.pack()
top.mainloop()
```



Ex:

```
 from tkinter import *
master = Tk()
var1 = IntVar()
Checkbutton(master, text='male', variable=var1).grid(row=0, sticky=W)
var2 = IntVar()
Checkbutton(master, text='female', variable=var2).grid(row=1, sticky=W)
mainloop()
```

Ex:

```
from tkinter import *
master = Tk()
Label(master, text='First Name').grid(row=0)
Label(master, text='Last Name').grid(row=1)
e1 = Entry(master)
e2 = Entry(master)
e1.grid(row=0, column=1)
e2.grid(row=1, column=1)
mainloop()
```

Ex:

```
from tkinter import *

top = Tk()
Lb = Listbox(top)
Lb.insert(1, 'Python')
Lb.insert(2, 'Java')
Lb.insert(3, 'C++')
Lb.insert(4, 'Any other')
Lb.pack()
top.mainloop()
```

2. Canvas:

The Canvas is **a rectangular area intended for drawing pictures or other complex layouts**. You can place graphics, text, widgets or frames on a Canvas.

Syntax;

w = **Canvas (master, option=value, ...)**

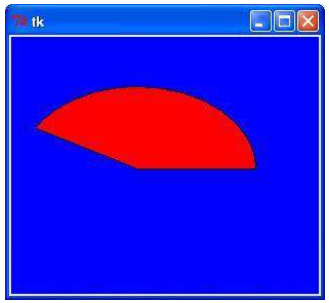
master − This represents the parent window.

options − Here is the list of most commonly used options for this widget. These options can be used as key-value pairs separated by commas.

bd--Border width in pixels. Default is 2.

bg--Normal background color.

confine--If true (the default), the canvas cannot be scrolled outside of the scrollregion.



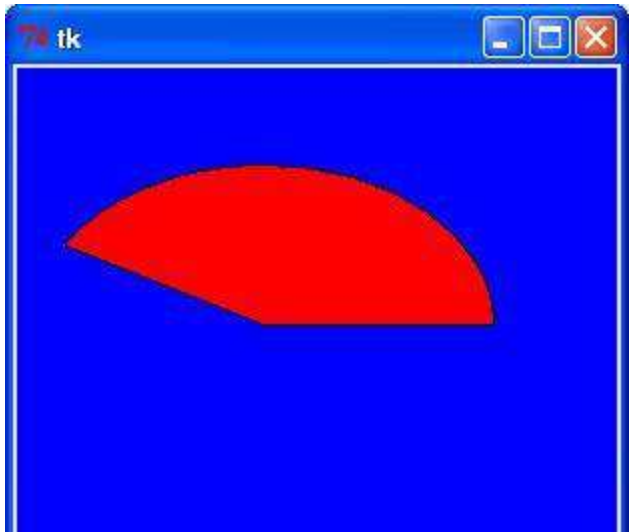
Cursor--Cursor used in the canvas like *arrow, circle, dot etc.*

height--Size of the canvas in the Y dimension.

Highlightcolor--Color shown in the focus highlight.

Ex:

```
import tkinter

top = tkinter.Tk()

C = tkinter.Canvas(top, bg="blue", height=250, width=300)

coord = 10, 50, 240, 210

arc = C.create_arc(coord, start=0, extent=150, fill="red“)

C.pack()

top.mainloop()
```



3. Scale widget:

✓ Scale widget is used **to implement the graphical slider to the python application** so that the user can **slide t**hrough the range of values shown on the slider and **select the one among** them.

✓We can control the **minimum and maximum values** along with the resolution of the scale.

Syntax

w = Scale(top, options)

A list of possible options is given below:

from_: It is used to represent **one end** of the widget range.

to: It represents a **float or integer value** that specifies the **other end of the range** represented by the scale.

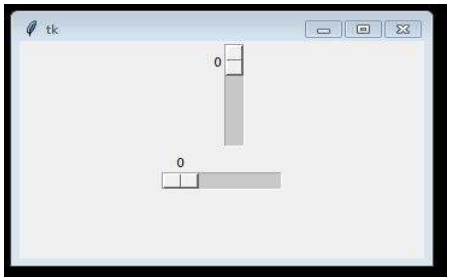
orient: It can be set to horizontal or vertical depending upon the type of the scale.

bd: The border size of the widget. The default is 2 pixel.

bg: The background color of the widget.

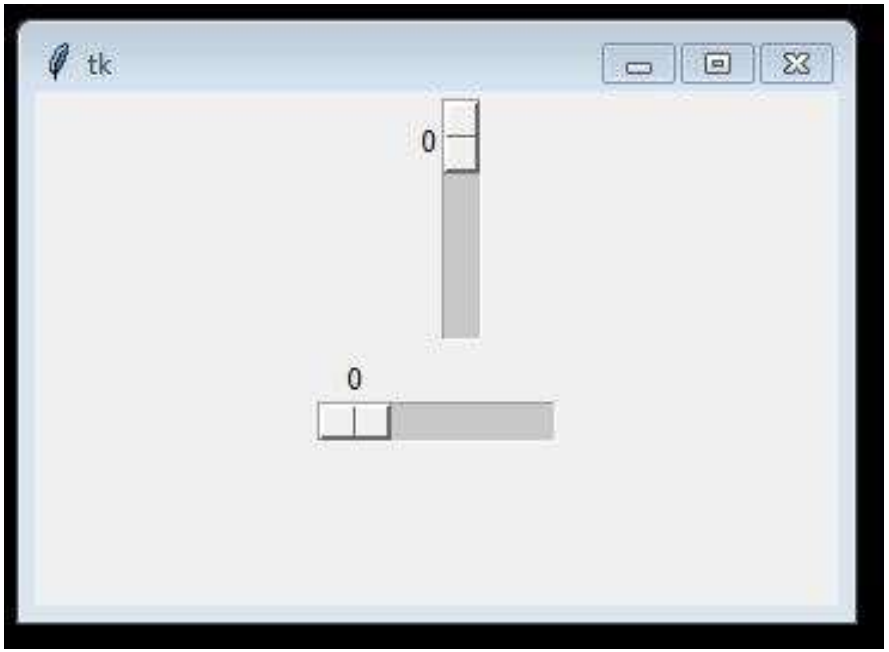
font: The font type of the widget text.

fg: The foreground color of the text



Ex:

```
from tkinter import *
master = Tk()
w = Scale(master, from_=0, to=100)
w.pack()
w = Scale(master, from_=0, to=200, orient=HORIZONTAL)
w.pack()
mainloop()
```



4. Label:

✓The Label is used **to specify the container box** where we can place the **text or images**. This widget is used to **provide the message to the user** about other widgets used in the python application.

✓There are the various options which can be specified to configure the text or the part of the text shown in the Label.

Syntax:

w = **Label (master, options)**

- **master** is the parameter used to represent the parent window.
- There are number of options which are used to change the format of the widget.

bg: to set he normal background color.

bg to set he normal background color.

command: to call a function.

font: to set the font on the button label.

image: to set the image on the button.

width: to set the width of the button.

height" to set the height of the button.

Ex:

```
from tkinter import *
 root = Tk()
 var = StringVar()
 label = Label( root, textvariable=var, relief=RAISED )
 var.set("Hey!? How are you doing?")
label.pack()
root.mainloop()
```

relief

Specifies the appearance of a decorative border around the label. The default is FLAT.

5. Checkbutton:

This widget is used to display a **number of options** to a user as toggle buttons. The user can then **select one or more options** by clicking the button corresponding to each option**. (similar to HTML checkbox input)**

Ex: checkbutton

```
from tkinter import *
import messagebox
import tkinter
top = tkinter.Tk()
CheckVar1 = IntVar()
CheckVar2 = IntVar()
C1 = Checkbutton(top, text = "Music", variable = CheckVar1,  onvalue = 1, offvalue = 0, height=5, width = 20)
C2 = Checkbutton(top, text = "Video", variable = CheckVar2, onvalue = 1, offvalue = 0, height=5, width = 20)
C1.pack()  C2.pack()     top.mainloop()
```

6. Entry

✓The Entry widget is used to **provide the single line text-box to the user to accept a value from the user.**

✓If you want to display **multiple lines of text** that can be edited, then you should use the ***Text* widget.**

✓If you want to display **one or more lines of text** that cannot be modified by the user, then you should use the ***Label* widget.**

Ex: Entry Widget

```
from tkinter import *
top = Tk()
 L1 = Label(top, text="User Name")
L1.pack( side = LEFT)
E1 = Entry(top, bd =5)
 E1.pack(side = RIGHT)
top.mainloop()
```

7. Frame:

It is used to **organize the group of widgets.** It acts like a container which can be used to **hold the other widgets.** The rectangular areas of the screen are used to **organize the widgets** to the python application.

8. Listbox:

The Listbox widget is used to **display a list of items** from which a user can select a number of items.

9. Menu:

The goal of this widget is to allow us **to create all kinds of menus** that can be used by our applications. The core functionality provides ways to create three menu types: **pop-up, top-level and pull-down**.

Ex: frame widget

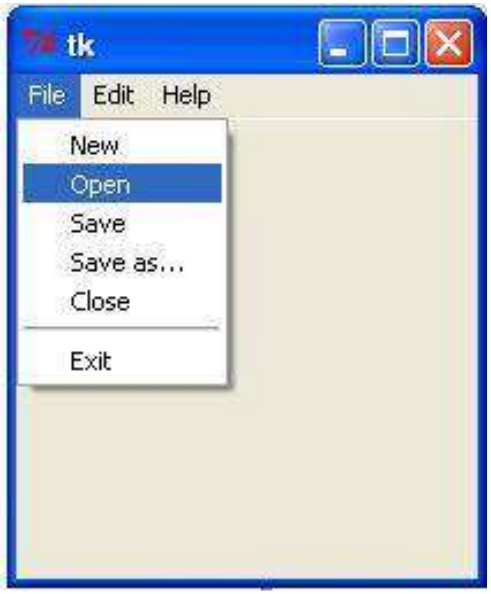
```
from Tkinter import *

root = Tk()
frame = Frame(root)
frame.pack()

bottomframe = Frame(root)
bottomframe.pack( side = BOTTOM )

redbutton = Button(frame, text="Red", fg="red")
redbutton.pack( side = LEFT)

greenbutton = Button(frame, text="Brown", fg="brown")
greenbutton.pack( side = LEFT )

bluebutton = Button(frame, text="Blue", fg="blue")
bluebutton.pack( side = LEFT )

blackbutton = Button(bottomframe, text="Black", fg="black")
blackbutton.pack( side = BOTTOM)

root.mainloop()
```

Ex: Listbox widget

```
from tkinter import *
import messagebox
import tkinter
top = Tk()
Lb1 = Listbox(top)
Lb1.insert(1, "Python")
Lb1.insert(2, "Perl")
Lb1.insert(3, "C")
Lb1.insert(4, "PHP")
Lb1.insert(5, "JSP")
Lb1.insert(6, "Ruby“)
Lb1.pack()
top.mainloop()
```

Ex: Menu

```
from tkinter import *
import messagebox
import tkinter
top = Tk()
mb=  Menubutton ( top, text="File", relief=RAISED )
mb.grid()
mb.menu =  Menu ( mb, tearoff = 0 )
mb["menu"] =  mb.menu
Var1 = IntVar()
Var2 = IntVar()
mb.menu.add_checkbutton ( label="New“, variable=Var1 )
mb.menu.add_checkbutton ( label="open“,  variable=Var2 )
mb.menu.add_checkbutton ( label="Save",  variable=Var2 )
mb.menu.add_checkbutton ( label="Save as",  variable=Var2 )
mb.menu.add_checkbutton ( label="Close“, variable=Var2 )
mb.menu.add_checkbutton ( label="Exit",   variable=Var2 )
mb.pack()
top.mainloop()
```



10. Menubutton

It can be defined as the **drop-down menu t**hat is shown to the user all the time. It is used to provide the user a option **to select the appropriate choice** exist within the application.

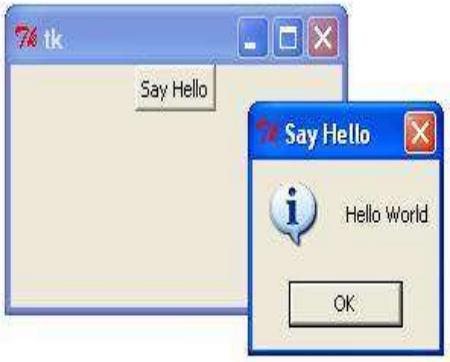
11. Message:

✓It is used to **show the message** to the user regarding the behavior of the python application.

✓It shows the **text messages to the user which can not be edited.**

✓The message text contains more than one line. However, the message can only be shown in the single font.

12. MessageBox:



✓It is used to **display message boxes** in your applications.

✓It provides a **number of functions** that you can use to display an appropriate message.

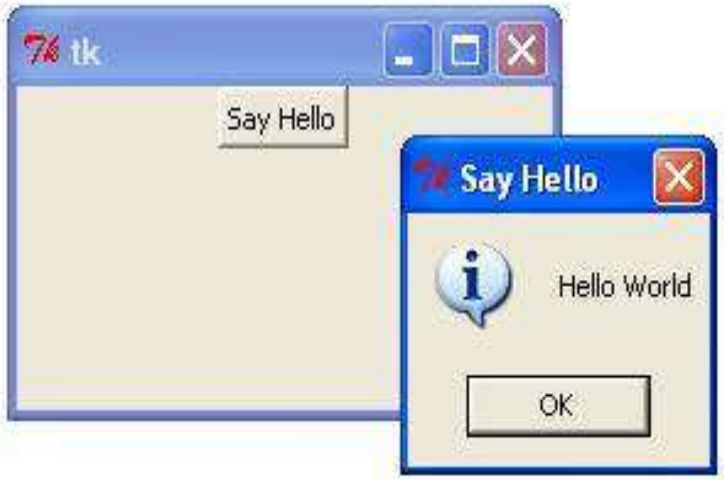
✓Some of these functions are **showinfo**, showwarning, showerror, askquestion, askokcancel, askyesno, and askretryignore.

Ex: Message widget

```
from tkinter import *

root = Tk()

var = StringVar()

label = Message( root, textvariable=var, relief=RAISED )

var.set("Hey!? How are you doing?")

label.pack()

root.mainloop()
```

Ex: Message box widget

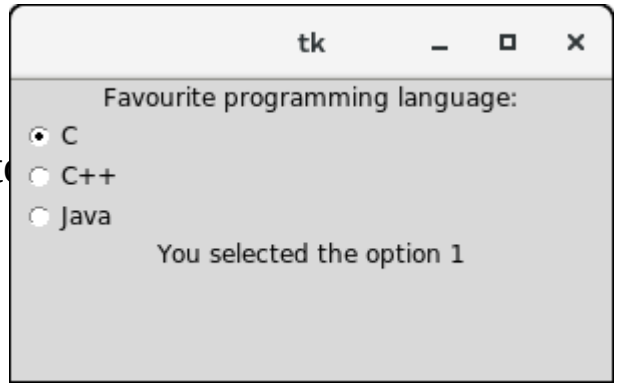
```
import tkinter

import messagebox

top = tkinter.Tk()

def hello():

   messagebox.showinfo("Say Hello", "Hello World“)

B1 = Tkinter.Button(top, text = "Say Hello", command = hello)

B1.pack()

top.mainloop()
```



13. Radiobutton:

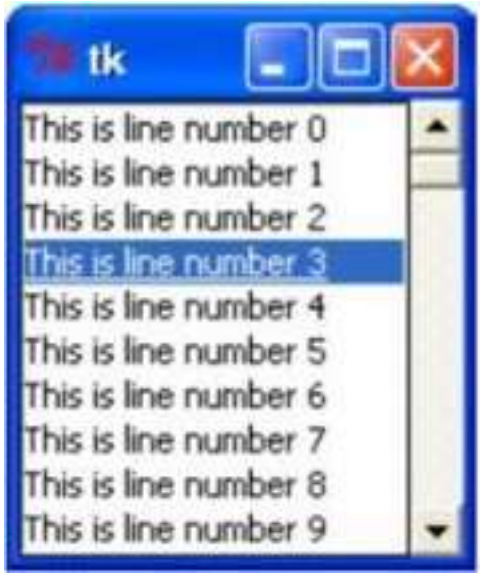
Set of buttons of which **only one can be "pressed"** (similar t

HTML radio input)



14. Scrollbar:

The scrollbar widget is used to **scroll down the content of the other widgets like listbox, text, and canvas**. However, we can also create the horizontal scrollbars to the Entry widget.

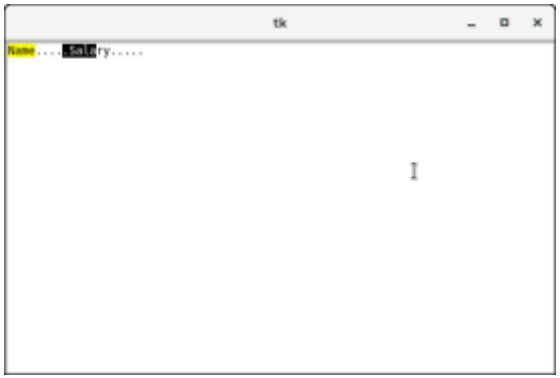


15. Text

✓It is used to **show the text data** on the Python application. However, Tkinter provides us the **Entry widget which is used to implement the single line text box.**

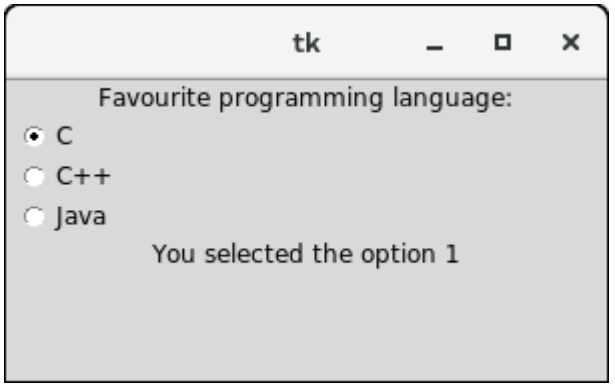
✓The Text widget is used to **display the multi-line formatted text with various styles and attributes**.

✓The Text widget is used to provide the text editor to the user.



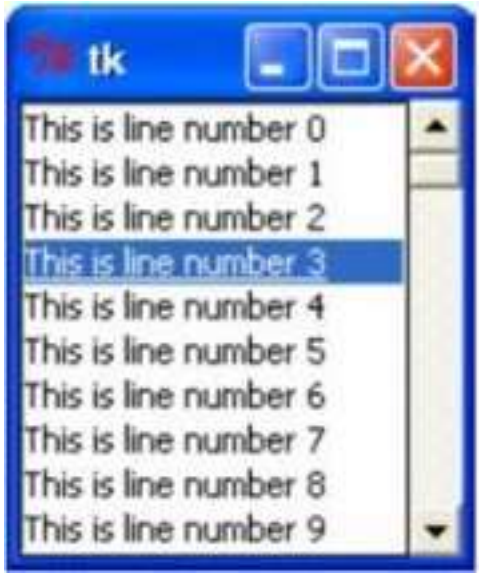
Ex: Radio button

```
from tkinter import *

def sel():
   selection = "You selected the option " + str(var.get())
   label.config(text = selection)

root = Tk()
var = IntVar()
R1 = Radiobutton(root, text="C", variable=var, value=1,
                 command=sel)
R1.pack( anchor = W )

R2 = Radiobutton(root, text="C++", variable=var, value=2,
                 command=sel)
R2.pack( anchor = W )

R3 = Radiobutton(root, text="PYTHON", variable=var, value=3,
                 command=sel)
R3.pack( anchor = W)

label = Label(root)
label.pack()
root.mainloop()
```



Ex: scroll bar:

```
from tkinter import *

root = Tk()

scrollbar = Scrollbar(root)

scrollbar.pack( side = RIGHT, fill = Y )

mylist = Listbox(root, yscrollcommand = scrollbar.set )

for line in range(50):

   mylist.insert(END, "This is line number " + str(line))

mylist.pack( side = LEFT, fill = BOTH )

scrollbar.config( command = mylist.yview )

mainloop()
```



Ex: text

```
from tkinter import *

def onclick():
   pass

root = Tk()

text = Text(root)

text.insert(INSERT, "Hello.....")

text.insert(END, "Bye Bye.....")

text.pack()

text.tag_config(background="yellow", foreground="blue")

text.tag_config(background="black", foreground="green")

root.mainloop()
```

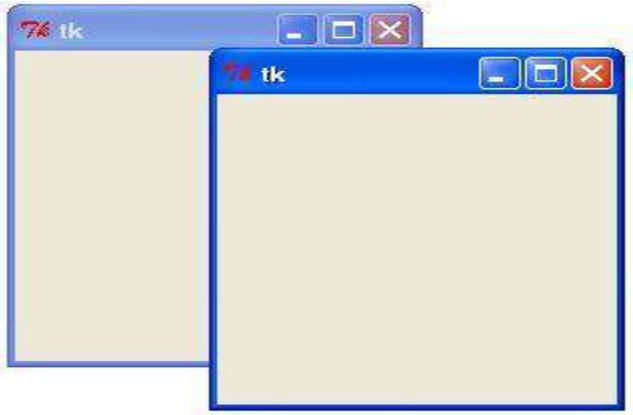
16. Toplevel:

✓The Toplevel widget is used to **create and display the toplevel windows** which are directly managed by the **window manager.**

✓The toplevel widget may or may not have **the parent window** on the top of them.

✓The toplevel widget is used when a python application needs **to represent some extra information, pop-up, or the group of widgets on the new window.**

✓The toplevel windows have **the title bars, borders, and other window decorations**.



17. spinbox:

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

Ex:

```
from tkinter import *
master = Tk()
w = Spinbox(master, from_=0, to=10)
w.pack()
mainloop()
```

Brief Tour of Other GUIs

four of the more popular and available toolkits out there:

1. **Tix** (Tk Interface eXtensions)
2. **Pmw** (Python MegaWidgets Tkinter extension)
3. **wxPython** (Python binding to wxWidgets) and
4. **PyGTK** (Python binding to GTK+).

✓The **Tix** module is already available in the **Python standard library**. You must download the other toolkit, which are third party.

✓**Pmw** is just an extension to Tkinter, it is the easiest to install (just extract into
✓your site packages).

✓**wxPython and PyGTK** involve the download of more than one file and building.

✓we would like to introduce the **Control or SpinButton and ComboBox.**

Application using various GUIs under Win32

Tix

PyGTK

Pmw

wxPython

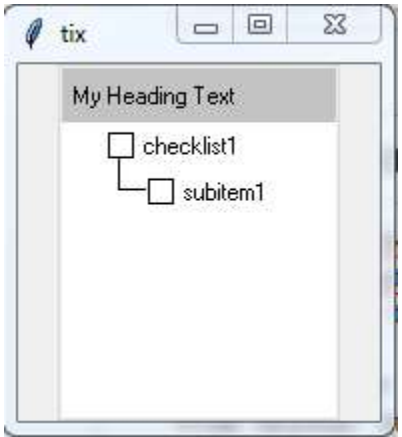
1. Tix (Tk Interface eXtensions):

✓The Tix module provides an **additional** rich set of widgets.

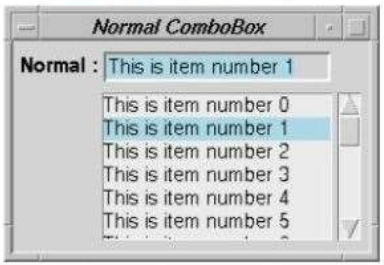
✓The **Tix library** provides most of the commonly needed widgets that are missing from standard Tk: **HList, ComboBox, Control (SpinBox)** and an assortment of **scrollable** widgets.

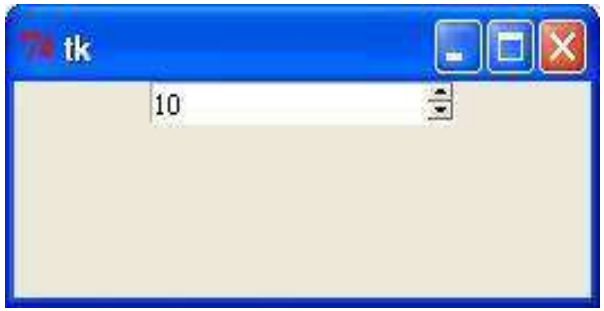
✓**Tix** also includes many more widgets that are generally useful in a wide range of applications: **NoteBook, FileEntry, PanedWindow, etc.**

✓The **HList widget** can be used to display any data that have a **hierarchical structure,** for example, file system directory trees



- TixComboBox = entry + listbox
- The `-dropdown` option:



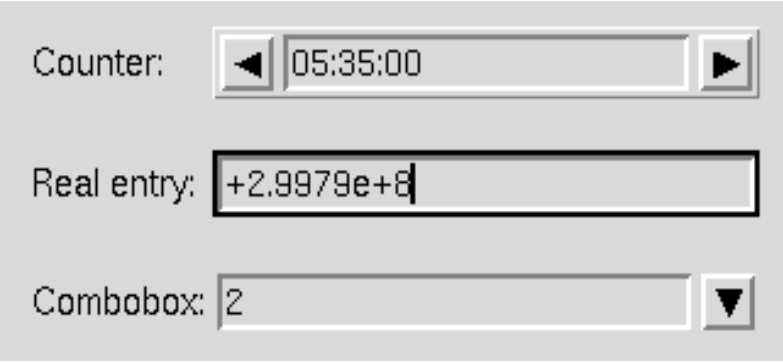


2. Pmw (Python MegaWidgets Tkinter extension):

- ✓ It is a toolkit for building **high-level compound widgets** in Python using the **tkinter package.**

- ✓ It consists of a set of base classes and a library of flexible and extensible megawidgets built on this foundation.

- ✓ These megawidgets include **notebooks, comboboxes, selection widgets, paned widgets, scrolled widgets, dialog windows,** etc.



3. wxPython (Python binding to wxWidgets):

✓It allows Python programmers to **create programs with a robust**, highly functional graphical user interface, simply and easily.

✓wxPython is **Open Source**, which means that it is **free for anyone to use and the source code is available** for anyone to look at and modify. And anyone can contribute fixes or enhancements to the project.

✓wxPython is a **cross-platform** toolkit. This means that the same program will run on multiple platforms without modification.

✓Currently **Supported platforms** are Microsoft Windows, Mac OS X and macOS, and Linux or other unix-like systems with GTK2 or GTK3 libraries.

✓ In most cases the **native widgets** are used on each platform to provide a 100% native look and feel for the application.

4. PyGTK (Python binding to GTK+):

- ✓ GTK are **open-source cross-platform ,User Interface toolkits and development framework**s.

- ✓ It is popular frameworks in use for **Linux** because they are open-source and give developers a powerful toolkit to design Graphical User Interfaces.

- ✓ **PyGTK** is a set of Python **wrappers** for the GTK+ graphical user interface library.

- ✓ GTK is just widget toolkit for the **GNOME applications** , **LXDE** and **Xfce** Desktop Environments.

- ✓ **GNOME** is a collection of approximately 30 applications that are packaged as part of the standard free and open-source GNOME desktop environment.

Related Modules and Other GUIs

There are other GUI development systems that can be used with Python.

GUI Module or System	Description
Tk-Related Modules	
`Tkinter`	TK INTERface: Python's default GUI toolkit http://wiki.python.org/moin/TkInter
`Pmw`	Python MegaWidgets (Tkinter extension) http://pmw.sf.net
`Tix`	Tk Interface eXtension (Tk extension) http://tix.sf.net
`TkZinc (Zinc)`	Extended Tk canvas type (Tk extension) http://www.tkzinc.org
`EasyGUI (easygui)`	Very simple non-event-driven GUIs (Tkinter extension) http://ferg.org/easygui
`TIDE + (IDE Studio)`	Tix Integrated Development Environment (including IDE Studio, a Tix-enhanced version of the standard IDLE IDE) http://starship.python.net/crew/mike

wxWidgets-Related Modules	
wxPython	Python binding to wxWidgets, a cross-platform GUI framework (formerly known as wxWindows)http://wxpython.org
Boa Constructor	Python IDE and wxPython GUI builder http://boa-constructor.sf.net
PythonCard	wxPython-based desktop application GUI construction kit (inspired by HyperCard) http://pythoncard.sf.net
wxGlade	another wxPython GUI designer (inspired by Glade, the GTK+/GNOME GUI builder) http://wxglade.sf.net
GTK+/GNOME-Related Modules	
PyGTK	Python wrapper for the GIMP Toolkit (GTK+) library http://pygtk.org
GNOME-Python	Python binding to GNOME desktop and development libraries http://gnome.org/start/unstable/bindings http://download.gnome.org/sources/gnome-python
Glade	a GUI builder for GTK+ and GNOME http://glade.gnome.org
PyGUI(GUI)	cross-platform "Pythonic" GUI API (built on Cocoa [MacOS X] and GTK+ [POSIX/X11 and Win32]) http://www.cosc.canterbury.ac.nz/~greg/python_gui

Qt/KDE-Related Modules	
PyQt	Python binding for the Qt GUI/XML/SQL C++ toolkit from Trolltech (partially open source [dual-license]) http://riverbankcomputing.co.uk/pyqt
PyKDE	Python binding for the KDE desktop environment http://riverbankcomputing.co.uk/pykde
eric	Python IDE written in PyQt using QScintilla editor widget http://die-offenbachs.de/detlev/eric3 http://ericide.python-hosting.com/
PyQtGPL	Qt (Win32 Cygwin port), Sip, QScintilla, PyQt bundle http://pythonqt.vanrietpaap.nl
Other Open Source GUI Toolkits	
FXPy	Python binding to FOX toolkit (http://fox-toolkit.org) http://fxpy.sf.net
pyFLTK (`fltk`)	Python binding to FLTK toolkit (http://fltk.org) http://pyfltk.sf.net
PyOpenGL (`OpenGL`)	Python binding to OpenGL (http://opengl.org) http://pyopengl.sf.net
Commercial	
`win32ui`	Microsoft MFC (via Python for Windows Extensions) http://starship.python.net/crew/mhammond/win32
`swing`	Sun Microsystems Java/Swing (via Jython) http://jython.org

Web Programming:

Introduction:

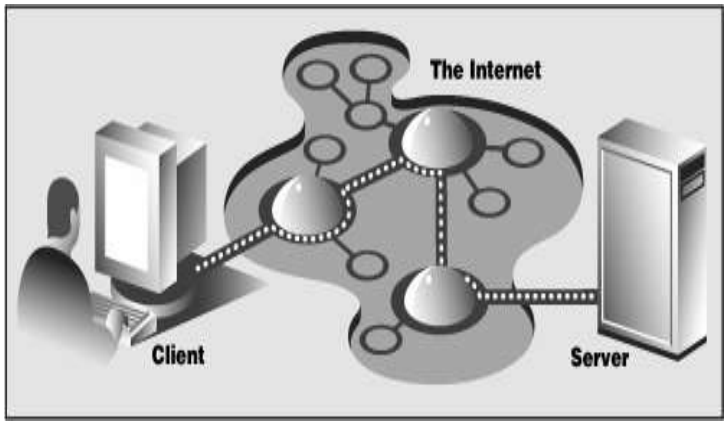
1. Web Surfing: Client/Server Computing (Again?!?):

- ✓ Web surfing falls under the same **client/server architecture** umbrella that we have seen repeatedly.

✓This time, **Web *clients are browsers***, *applications that allow users to find documents on the World Wide* Web.

✓ On the other side are **Web *servers***, *processes that run on an information provider's host* computers. These servers wait for clients and their document requests, process them, and return the requested data.

✓Here, **a user runs a Web client program** such as a browser and makes a connection to a Web server elsewhere on the Internet to obtain information.



Web Surfing with Python: Creating Simple Web Clients:

✓One thing to keep in mind is that a **browser is only one type of Web client**. Any application that makes a request for data from a Web server is considered a "**client.“** It is possible to create other clients that retrieve documents or data off the Internet.

✓One important reason to do this is that a **browser provides only limited capacity**, i.e., it is used primarily for **viewing and interacting with Web sites.**

✓on the other hand , **A client program,, has the ability to do more it can not only download data, but it can also store it, manipulate it, or perhaps even transmit it to another location or application.**

✓Applications that use the **urllib module** to download or access information on the Web [using either urllib.urlopen() or urllib.urlretrieve()] can be considered a simple Web client. All you need to do is provide a valid **Web address.**

Uniform Resource Locators:

✓Simple Web surfing involves using **Web addresses called *URLs*** *(Uniform Resource Locators). Such* addresses are used to locate a document on the Web or to call a CGI program to generate a document for your client.

✓**URLs are part of a larger set of identifiers** known as *URIs (Uniform Resource Identifiers).*

✓A URL is simply a URI which uses an existing protocol or scheme (i.e., http, ftp, etc.) as part of its addressing.

✓Like street addresses, **Web addresses have some structure**. An American street address usually is of the form "number street designation," i.e., 123 Main Street. It differs from other countries, which have their own rules.

A URL uses the format:

prot_sch://net_loc/path;params?query#fra*g*

URL Component	Description
prot_sch	Network protocol or download scheme
net_loc	Location of server (and perhaps user information)
path	Slash (/) delimited path to file or CGI application
params	Optional parameters
query	Ampersand (&) delimited set of "key=value" pairs
frag	Fragment to a specific anchor within document

Again**, net_loc** can be **broken down** into several more components, some required, others optional. The net_loc string looks like this:

user:passwd@host:port

net_loc Component	Description
user	User name or login
passwd	User password
host	Name or address of machine running Web server [required]
port	Port number (if not 80, the default)

✓Of the four, the **host name** is the most important for running web server.

✓ The **port number** is necessary only if the Web server is running on a different port number from the default.

✓**User names and passwords** are used **only when making FTP connections.**

To deal with **URLs** in completely different functionality and capacities. Python supplies two different **modules,**. One is **urlparse**, and the other is **urllib.**

1. urlparse Module:

✓The urlparse module provides **basic functionality** with which to manipulate **URL strings.**

These functions include-

1. urlparse(),
2. urlunparse(), and
3. urljoin()

1. **urlparse.urlparse():**

✓urlparse() **breaks up a URL string into some of the major components**(prot_sch, net_loc, path, params, query, frag).

It has the following syntax:

urlparse(urlstr, defProtSch=None, allowFrag=None)

✓urlparse() parses urlstr into a **6-tuple (prot_sch, net_loc, path, params, query, frag)**

✓**defProtSch** indicates a default network protocol or download scheme in case one is not provided in **urlstr.allowFrag.**

✓**urlstr.allowFrag** is a flag that signals whether or not a fragment part of a URL is allowed.

✓ Here is what urlparse() outputs when given a URL:

>>> **urlparse.urlparse('http://www.python.org/doc/FAQ.html')**

ParseResult(scheme='http', netloc='www.python.org', path='/doc/FAQ.html', params='', query='', fragment='')

ii. urlparse.urlunparse():

urlunparse() does the exact **opposite** of urlparse() **it merges a 6-tuple** (prot_sch, net_loc, path, params, query, frag) urltup, which could be the output of urlparse(), into a single URL string and returns it.

Syntax:

urlunparse(urlparse(urlstr))

Ex:

Urlunparse(urlparse('http://www.python.org/doc/FAQ.html'))

Output:

http://www.python.org/doc/FAQ.html

✓In this case, **parameters, query, and fragment are all missing in the original URL**. The **new URL** does not look the same as the **original**, but is equivalent according to the standard.

iii) urlparse.urljoin():

The urljoin() function is useful in cases where many related URLs are needed,

for example, the URLs for a set of pages to be generated for a Web site.

syntax for urljoin() is:

urljoin(baseurl, newurl, allowFrag=None)

urljoin() takes baseurl and joins its base path **(net_loc plus the full path up to, but not including, a file at the end)** with newurl.

For example:

```
>>> urlparse.urljoin('http://www.python.org/doc/FAQ.html', \
... 'current/lib/lib.htm')
    'http://www.python.org/doc/current/lib/lib.html'
```

urllib Module:

✓urllib is a Python module that can be used for opening URLs(Uniform Resource Locators). It defines functions and classes to help in URL actions.

Urllib is a package that collects several modules for working with URLs, such as:

- ✓**urllib.request** for opening and reading.
- ✓**urllib.parse** for parsing URLs
- ✓**urllib.erro**r for the exceptions raised
- ✓**urllib.robotparse**r for parsing robot.txt files

✓If urllib is not present in your environment, execute the below code to install it.

pip install urllib

urllib.urlopen()

urlopen() opens a Web connection to the given URL string and returns a file-like object.

syntax:

urlopen(*urlstr, postQueryData=None)*

✓urlopen() opens the URL pointed to by *urlstr. If no protocol or download scheme is given, or if a "file“* scheme is passed in, urlopen() will open a local file.

✓For all HTTP requests, the normal request type is "GET." In these cases, the query string provided to the Web, should be given as part of *urlstr.*

✓If the "POST" request method is desired, then the query string should be placed in the *postQueryData variable.*

✓GET and POST requests are the two ways to "upload" data to a Web server. When a successful connection is made, **urlopen() returns a file-like object** as if the destination was a file opened in read mode.

✓ for example, if our file object is f, then our "handle" would support the expected read methods such as **f.read(), f.readline(), f.readlines(), f.close(), and f.fileno().**

`urlopen()` Object Methods	Description
`f.read([bytes])`	Reads all or `bytes` bytes from `f`
`f.readline()`	Reads a single line from `f`
`f.readlines()`	Reads a all lines from `f` into a list
`f.close()`	Closes URL connection for `f`
`f.fileno()`	Returns file number of `f`
`f.info()`	Gets MIME headers of `f`
`f.geturl()`	Returns true URL opened for `f`

urllib.urlretrieve():

Syntax:

urlretrieve(*urlstr, localfile=None, downloadStatusHook*=None)

✓Rather than reading from the URL like urlopen() does, **urlretrieve()** will **simply download the entire HTML file located** at urlstr to your local disk.

✓It will store the downloaded data into localfile if given or a temporary file if not.

✓If the file has already been copied from the Internet or if the file is local, no subsequent downloading will occur.

✓The **downloadStatusHook,** if provided, is a function that is called after each block of data has been downloaded and delivered.

✓ It is called with the following **three arguments:**

1. number of blocks read so far,
2. the block size in bytes, and
3. the total (byte) size of the file.

✓This is very useful if you are implementing "**download status**" information to the user in a text-based or graphical display.

- ✓ **urlretrieve()** returns a 2-tuple, (filename, mime_hdrs).
  - ❑ **filename** is the name of the local file containing the downloaded data.
  - ❑ **mime_hdrs** is the set of MIME headers returned by the responding Web server.

urllib.quote() and urllib.quote_plus():

The quote() functions take URL data and "encodes" them so that they are "fit" for inclusion as part of a URL string.

Syntax:

quote(*urldata, safe='/')*

✓**Characters that are never converted** include commas, underscores, periods, and dashes, and alphanumerics. All others are subject to conversion.

✓In particular, the **disallowed characters(?,!, spaces) are changed to their hexadecimal ordinal equivalents** prepended with a percent sign **(%), i.e.,** "%xx" where "xx" is the hexadecimal representation of a character's ASCII value.

Ex:

```
>>> import urllib
>>> urllib.quote('Hello World@Python2')
        'Hello%20World%40Python2‘
```

✓ When calling quote*(), the ***urldata string is*** **converted to an equivalent string** that can be part of a URL string.

✓Note that, the quote() function considers **/ character *safe* by default**. That means, It doesn't **encode / character .**

✓The quote() function accepts a named parameter called ***safe*** whose default value is /

✓. If you want to **encode /** character as well, then you can do so by supplying an empty string in the *safe* parameter like this-

```
>>> urllib.parse.quote('/', safe='')
        '%2F‘
```

✓**quote_plus() is similar to quote()** except that it also encodes spaces to plus signs (+).

✓**Ex:**

```
>>> import urllib
 >>> urllib.quote('Hello World@Python2')
            'Hello+World%40Python2‘
```

urllib.urlencode()

✓urlencode() takes a **dictionary of key-value pairs and encodes them** to be included as part of a query in a CGI request URL string.

✓The pairs are in **"key=value" format** and are **delimited by ampersands (&).**

Ex:

```
>>> aDict = { 'name': 'Georgina Garcia', 'hmdir': '~ggarcia' }
>>> urllib.urlencode(aDict)
```

'**name=Georgina+Garcia&hmdir=%7eggarcia**'

urllib2 Module:

✓**urllib2** is a Python module that can be used for fetching URLs.

✓The urllib2 module defines **functions and classes** which help in opening URLs (mostly HTTP) in a complex world (basic and digest authentication, redirections, cookies and more)

Difference between urlib and urlib2

✓While both modules do URL request related stuff, they have different functionality.

✓ **urllib2** can accept a Request object to set the headers for a URL request, **urllib** accepts only a URL.

✓ **urllib** provides the urlencode method which is used for the generation of GET query strings, **urllib2** doesn't have such a function.

Advanced Web Clients

✓Web browsers are **basic Web clients**. They are used primarily for searching and downloading documents from the Web.

✓**Advanced clients** of the Web are those applications that do **more than download single documents from the Internet.**

✓We can explore and download pages from the Internet for different reasons, some of which include:

- Indexing into a large search engine such as Google or Yahoo!
- Offline browsing downloading documents onto a local hard disk and rearranging hyperlinks to create almost a mirror image for local browsing
- Downloading and storing for historical or archival purposes, or
- Web page caching to save superfluous downloading time on Web site revisits.

CGI: Helping Web Servers Process Client Data

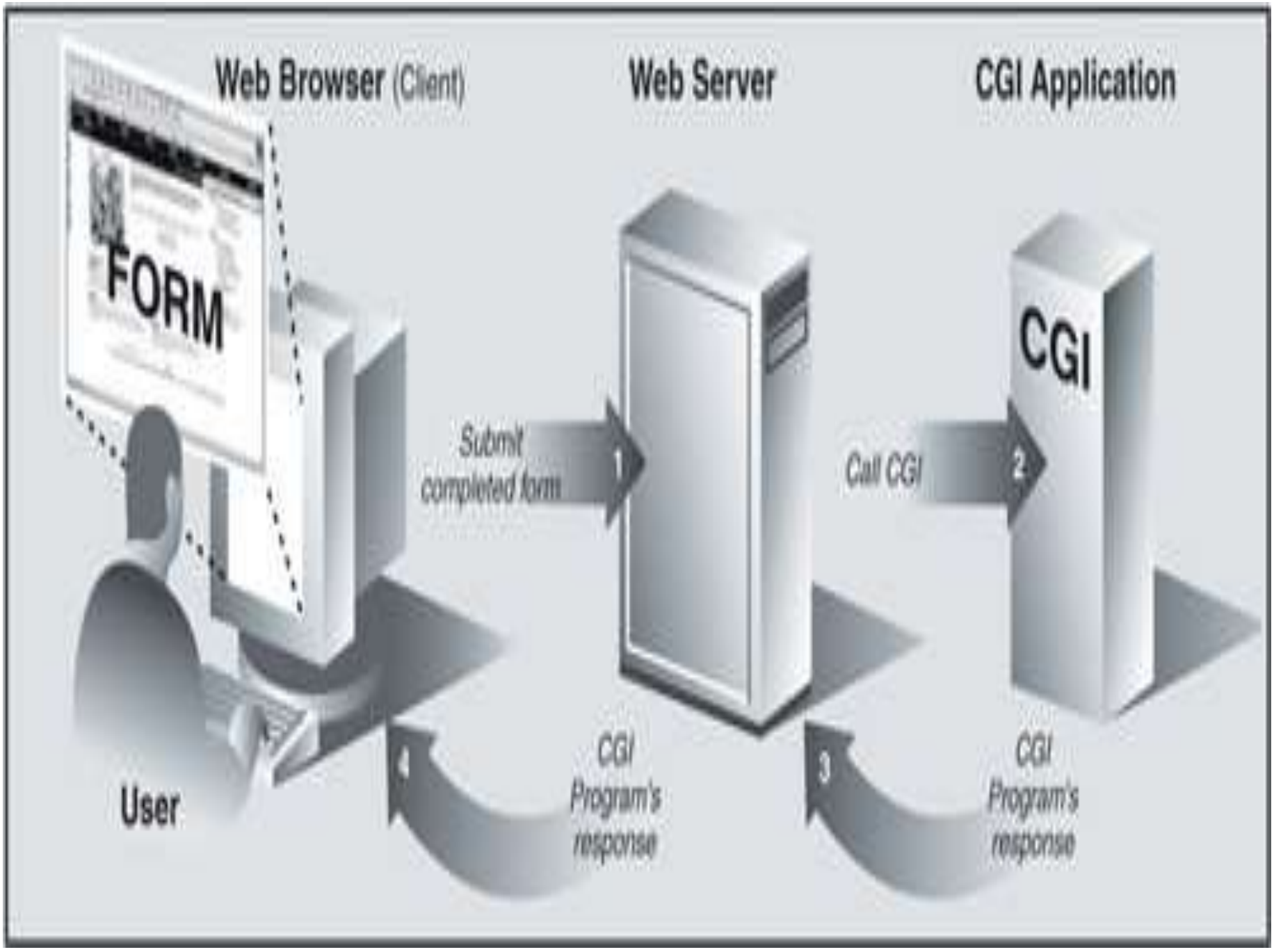
✓**CGI**(**C**ommon **G**ateway **I**nterface) is a **web technology and protocol** that defines a way for a web server (HTTP server) to interact with external applications, e.g. PHP.

✓CGI is used as an **interface between the web server** and the additionally installed applications generating dynamic web content. These applications are called **CGI scripts** and are written in different script and programming languages such as PHP, Perl, Python, etc.

✓ CGI allows CGI programs **process HTML forms** or other data coming from clients, and then it allows the CGI programs send a response back to the client

✓.

✓The **response** can be HTML documents, GIF files, video clips, or any data the client browser can view. This makes your web pages **interactive** with the user.


Web Browser (Client)
Web Server
CGI Application
FORM
CGI
Submit completed form
1
Call CGI
2
User
4
CGI Program's response
3
CGI Program's response

✓When a client **requests** a document from a server, the server finds the file and sends it to the client.

✓However, if a client requests a **CGI program**, the **server** simply acts as an **intermediary** between the client and the CGI program.

what happens when a client requests a CGI process?

1. The client sends a request to the server running on your machine. The request might be for a document, or like the contents of an HTML form.

✓If the request is for a **regular** document (such as an HTML document or a .GIF file), the **server** sends that document **directly** back to the client.

✓If the request is data intended for an **external application**, then the server needs to **use CGI** to run that application.

✓ **For example**, the client's request might be to **search a database.** The CGI application takes the search criteria, searches the database, then sends the results back to the client.

2. **When a server receives a request that must be handled by an external application (a *CGI request*) that server creates a copy of itself.**
   - ✓ This second process is called the ***CGI process*** because it is the process in which the CGI program will run.
   - ✓ The CGI process has all the **same communication pathways** that the server process has. The only purpose for the CGI process is **to set up communications between the CGI program and the server.**

3. **The CGI program responds to the client**.
   - ✓ The CGI program **takes the data** that the server provides through **environment variables, standard input**, or command-line arguments.
   - ✓ It processes the data, contacts any external services it needs to, and then **sends a response** to the server by way of the data pathways using **standard output.**
   - ✓ **The server** then takes the program's response and sends **it back to the client** software.
   - ✓ Your program can **output** any type of data it needs to, including HTML, GIFs, or JPEGs.

CGI applications:

✓CGI applications that **create the HTML** are usually written in one of many higher-level programming languages that have the **ability to accept user data**, process it, and **return HTML back to the server.**

✓**CGI application is slightly different from a typical program.**

✓The primary differences are in the **input, output, and user interaction aspects of a computer program.**

✓When a **CGI script starts**, it needs to retrieve the **user-supplied form data**, but it has to obtain this data from the **Web client,** not a user on the server machine nor a disk file.

✓The **output differs** in that any data sent to standard output will be **sent back** to the connected **Web client** rather than to the screen, GUI window, or disk file.

✓The data sent back must be a **set of valid headers followed by HTML**. If it is not ,**an error will occur** because Web clients is a browser, it understand only valid HTTP data (i.e., MIME headers and HTML).

cgi Module

✓There is one primary class in the **cgi module** i.e., **the FieldStorage class**.

✓This class should be **instantiated** when a Python CGI script **begins**, as it will read in all the relevant user information from the **Web client** (via the Web server).

✓Once this **object has been instantiated**, it will consist of **a dictionary-like object that has a set of key-value pairs.** The **keys** are the **names of the form items** that were passed in through the form while the **values** contain the **corresponding data.**

✓These values themselves can be one of **three objects**. They can be

- ✓**FieldStorage objects** (instances)
- ✓**MiniFieldStorage**, which is used in cases where **no file uploads** or **multiple-part form data is involved.** MiniFieldStorage instances contain only the **key-value pair** of the name and the data.
- ✓**Lastly, they can be a list of such objects.** This occurs when a form contains **more than one input item with the same field name.**

✓For simple Web forms, you will usually find all **MiniFieldStorage instances**.

Building CGI Applications:

1. **Setting Up a Web Server**
2. **Creating the Form Page**
3. **Generating the Results Page**
4. **Fully Interactive Web sites**

1. Setting Up a Web Server:

✓**To work with** CGI development in Python, you need to first **install a Web server**, configure it for handling Python CGI requests, and then give the Web server access to your CGI scripts.

✓If you want a **real Web server**, you will likely download and install **Apache.** There are Apache plug-ins or modules for **handling Python CGI.**

✓If you want to just start up the most basic Web server, just execute it directly with Python:

$ python -m CGIHTTPServer

✓This will start a Web server on port 8000 on your current machine from the current directory. Then you can just create a **Cgi-bin** right underneath the directory from which you started the server and put your Python CGI scripts in there.

✓Put some HTML files in that directory and perhaps some .py CGI scripts in Cgi-bin, and you are ready to "surf" directly to this Web site with addresses looking something like these:

http://localhost:8000/friends.htm

http://localhost:8000/cgi-bin/friends2.py

2. Creating the Form Page

This HTML file **(friends.html)** presents a form to the user with an empty field for the user's name and a set of radio buttons for the user to choose from.

```
1 <HTML><HEAD><TITLE>
2 Friends CGI Demo (static screen)
3 </TITLE></HEAD>
4 <BODY><H3>Friends list for: <I>NEW USER</I></H3>
5 <FORM ACTION="/cgi-bin/fsriend1.py">
6 <B>Enter your Name:</B>
7 <INPUT TYPE=text NAME=person VALUE="NEW USER" SIZE=15>
8 <P><B>How many friends do you have?</B>
9 <INPUT TYPE=radio NAME=howmany VALUE="0" CHECKED> 0
10 <INPUT TYPE=radio NAME=howmany VALUE="10"> 10
11 <INPUT TYPE=radio NAME=howmany VALUE="25"> 25
12 <INPUT TYPE=radio NAME=howmany VALUE="50"> 50
13 <INPUT TYPE=radio NAME=howmany VALUE="100"> 100
14 <P><INPUT TYPE=submit></FORM></BODY></HTML>
```

✓As you can see in the code, the form contains two input variables: person and how many. The values of these two fields will be passed to our CGI script, friends1.py.

friends.htm in a client

Friends form page in IE6 on Win32 (friends.htm)

3.Generating the Results Page:

✓The input is entered by the user and the "Submit" button is pressed.

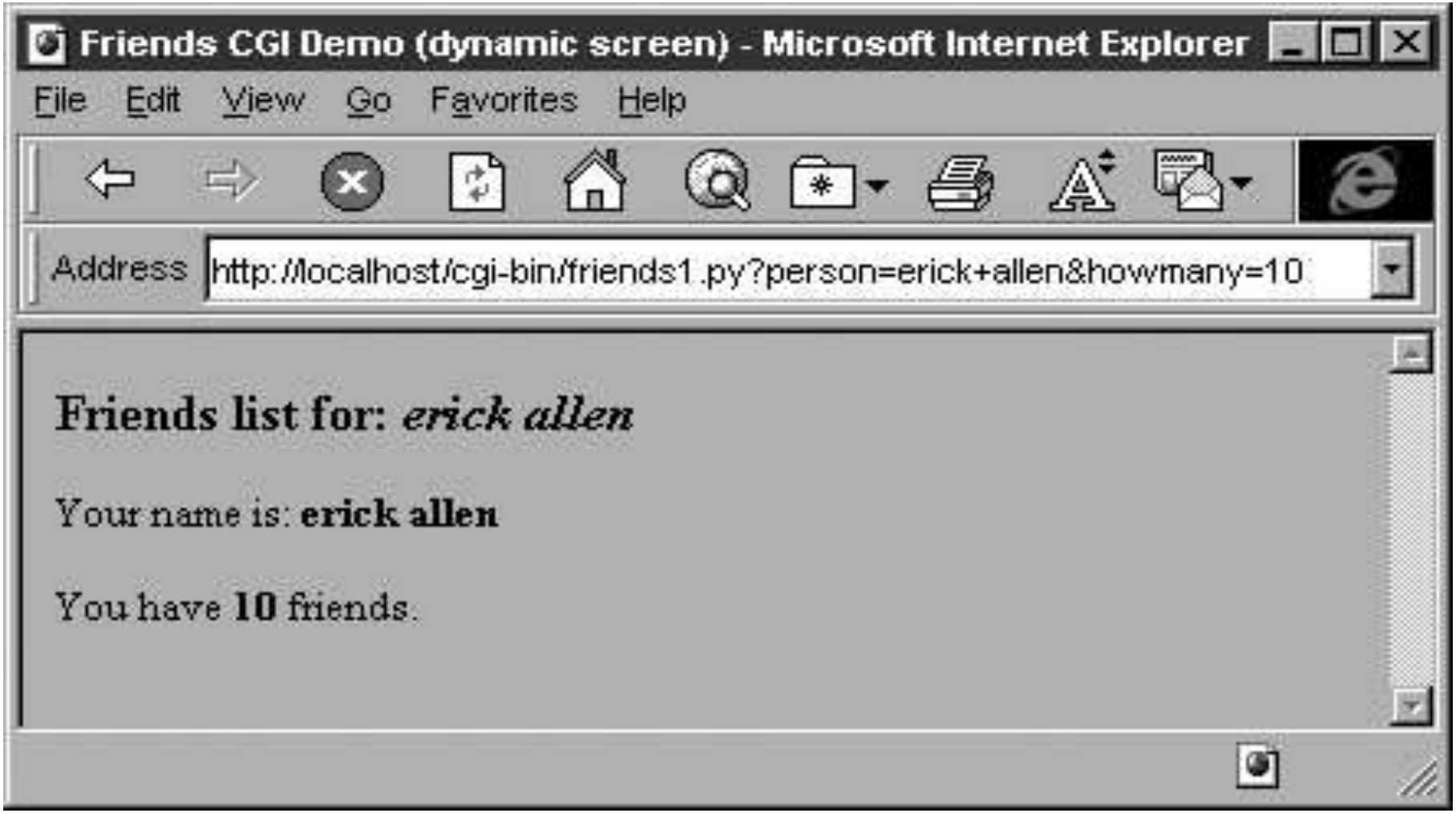
✓When this occurs, the script in, friends1.py, is executed via CGI.

✓This CGI script grabs the person and how many fields from the form and uses that data to create the dynamically generated results screen.

```
1 #!/usr/bin/env python(friends1.py)
2
3 import cgi
4
5 reshtml = '''Content-Type: text/html\n
6 <HTML><HEAD><TITLE>
7 Friends CGI Demo (dynamic screen)
8 </TITLE></HEAD>
9 <BODY><H3>Friends list for: <I>%s</I></H3>
10 Your name is: <B>%s</B><P>
11 You have <B>%s</B> friends.
12 </BODY></HTML>'''
13
14 form = cgi.FieldStorage()
15 who = form['person'].value
16 howmany = form['howmany'].value
17) print reshtml % (who, who, howmany
```

✓This script contains all the programming power to read the form input and process it, as well as return the resulting HTML page back to the user.

✓assuming the user typed in "erick allen" as the name and clicked on the "10 friends" radio button.



4. Fully Interactive Web sites:

✓A user enters his or her information from the form page. We then process the data and output a results page.

✓Now, we will add a link to the results page that will allow the user to go back to the form page, but rather than presenting a blank form, we will fill in the data that the user has already provided.

Advanced CGI

✓These advanced CGI include the **use of cookies cached data saved on the client side multiple values for the same CGI field and file upload** using multipart form submissions.

Using Cookies in CGI

✓For a commercial website, it is required to maintain **session information among** different pages.

✓ For example, one user registration ends after completing many pages. How to maintain user's session information across all the web pages?

✓In many situations, using cookies is the most efficient method of remembering and tracking preferences, purchases, commissions, and other information required for better visitor experience or site statistics.

How It Works?

- ✓ When **server** sends some data to the visitor's browser **in the form of a cookie**. The browser may **accept the cookie**. If it does, it is stored as a **plain text record** on the visitor's hard drive.
- ✓ Now, when the visitor arrives at another page on your site, the cookie is available for retrieval. Once retrieved, your server knows/remembers what was stored.

Cookies are a plain text data record of 5 variable-length fields −

Expires − The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.

Domain − The domain name of your site.

Path − The path to the directory or web page that sets the cookie. This may be blank if you want to retrieve the cookie from any directory or page.

Secure − If this field contains the word "secure", then the cookie may only be retrieved with a secure server. If this field is blank, no such restriction exists.

Name=Value − Cookies are set and retrieved in the form of key and value pairs.

Setting up Cookies

✓It is very easy to send cookies to browser. These **cookies are sent along with HTTP Header** before to Content-type field. Assuming you want to set UserID and Password as cookies.

Setting

```
print "Set-Cookie:UserID = XYZ;\r\n"
print "Set-Cookie:Password = XYZ123;\r\n"
print "Set-Cookie:Expires = Tuesday, 31-Dec-2007 23:12:40 GMT";\r\n"
print "Set-Cookie:Domain = www.tutorialspoint.com;\r\n"
print "Set-Cookie:Path = /perl;\n"
print "Content-type:text/html\r\n\r\n"
.....Rest of the HTML Content....
```

✓We use **Set-Cookie** HTTP header to set cookies.

✓It is optional to set cookies attributes like Expires, Domain, and Path.

✓ It is notable that cookies are set before line **"Content-type:text/html\r\n\r\n**.

Retrieving Cookies:

✓It is very easy to retrieve all the set cookies. Cookies are stored in CGI environment variable **HTTP_COOKIE** and they will have following form −

key1 = value1;key2 = value2;key3 = value3....

```
# Import modules for CGI handling
from os import environ
import cgi, cgitb

if environ.has_key('HTTP_COOKIE'):
   for cookie in map(strip, split(environ['HTTP_COOKIE'], ';')):
      (key, value ) = split(cookie, '=');
      if key == "UserID":
         user_id = value

      if key == "Password":
         password = value

print "User ID  = %s" % user_id
print "Password = %s" % password
```

This produces the following result for the cookies set by above script −

User ID = XYZ

Password = XYZ123

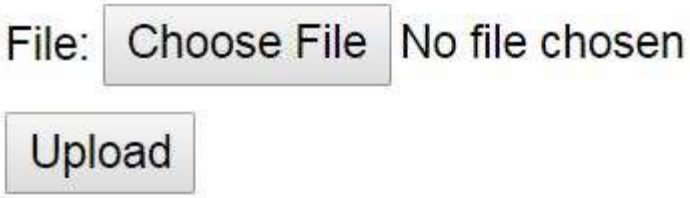
File Upload:

To upload a file, the HTML form must have the **enctype attribute** set to **multipart/form-data**.

The input tag with the file type creates a "Browse" button.

```
<html>
<body>
   <form enctype = "multipart/form-data"
                     action = "save_file.py" method = "post">
   <p>File: <input type = "file" name = "filename" /></p>
   <p><input type = "submit" value = "Upload" /></p>
   </form>
</body>
</html>
```

The result of this code is the following form −



Here is the script **save_file.py** to handle file upload −

```
import cgi, os
import cgitb; cgitb.enable()

form = cgi.FieldStorage()

# Get filename here.
fileitem = form['filename']

# Test if the file was uploaded
if fileitem.filename:
   # strip leading path from file name to avoid
   # directory traversal attacks
   fn = os.path.basename(fileitem.filename)
   open('/tmp/' + fn, 'wb').write(fileitem.file.read())

   message = 'The file "' + fn + '" was uploaded successfully'

else:
   message = 'No file was uploaded'

print """\
Content-Type: text/html\n
<html>
<body>
   <p>%s</p>
</body>
</html>
""" % (message,)
```

Multivalued Fields:

how to process fields with multiple values.

✓when you have a set of **checkboxes** allowing a user to select from various choices.

✓Each of the checkboxes is labeled with the **same field name**, but to differentiate them, **each will have a different value associated with a particular checkbox.**

✓As you know, the data from the user are sent to the server in **key-value pairs** during form submission.

✓**When more than one checkbox is submitted**, you will **have multiple values** associated with the **same key**. In these cases, rather than being given a single **MiniFieldStorage instance** for your data, **the cgi module will create a list of such instances** that you will iterate over to obtain the different values.

Web (HTTP) Servers:

An HTTP web server is nothing but a process that is running on your machine and does exactly two things:

1- Listens for incoming http requests on a specific TCP socket address (IP address and a port number)

2- Handles this request and sends a response back to the user.

Creating Web Servers in Python:

✓**In python 2,**To create a Web server, **a base server** and a **"handler"** are required.

✓The **base (Web) server** is a boilerplate item, a must have. Its role is to perform the necessary **HTTP communication between client and server**.

✓ The base server class is (appropriately) named **HTTPServe**r and is found in the **BaseHTTPServe**r module.

✓The **handler** is the piece of software that does the majority of the **"Web serving."**

It processes the client request and returns the appropriate file, whether static or dynamically generated by CGI.

✓ The complexity of the handler determines the **complexity of your Web server**.

The Python standard library provides three different handlers:

1. **BaseHTTPRequestHandler** is found in the **BaseHTTPServer** module, along with the **base Web server.**

2. The **SimpleHTTPRequestHandler,** available in the **SimpleHTTP-Server** module, builds on **BaseHTTPRequestHandle**r by implementing the standard **GET and HEAD** requests.

3. Finally, we have the **CGIHTTPRequestHandler,** available in the **CGIHTTPServe**r module, which takes the **SimpleHTTPRequestHandler** and adds support for **POST requests.** It has the ability to call CGI scripts to perform the requested processing and can send the generated HTML back to the client.

Web Server Modules and Classes:

Module	Description
1. **BaseHTTPServer**	Provides the **base Web server** and **base handler classes**, HTTPServer and BaseHTTPRequestHandler, respectively.
2. SimpleHTTPServer	Contains the **SimpleHTTPRequestHandler class** to perform GET and HEAD requests.
3. CGIHTTPServer	Contains the **CGIHTTPRequestHandler class** to process POST requests and perform CGI execution

✓In order to create a web server in **Python 3**, you will need to import two modules: **http.server** and **socketserver.**

✓in **Python 2**, there was a module named **SimpleHTTPServer.** This module has been merged into **http.server** in **Python 3.**

✓I mentioned earlier, **First a web server** is a process that listens to incoming requests on specific TCP address. **TCP address** is identified by an **ip address and a port number**.

✓**Second, a web server** also needs to be told **how to handle incoming requests.**

- ✓These incoming requests are handled by **special handlers.** You can think of a web server as a **dispatcher**, a request comes in, the **http server inspects the request** and dispatches it to a **designated handler.**

Ex: Creating a webserver

```
import http.server
import socketserver
PORT = 8080
Handler = http.server.SimpleHTTPRequestHandler
 with socketserver.TCPServer(("", PORT), Handler) as httpd:
      print("serving at port", PORT)
      httpd.serve_forever()
```

✓**http.server.SimpleHTTPRequestHandle**r is: a simple HTTP request handler that serves files from **the current directory** and **any of its subdirectories**.

✓**socketserver.TCPServer** class: An instance of **TCPServer** describes a server that uses the TCP protocol to send and receive messages.

To instantiate a **TCP Server**, we need two things:

1- The TCP address (IP address and a port number)

2- The handler

socketserver.TCPServer(("", PORT), Handler)

✓ **TCP address is passed as a tuple of (ip address, port number)**

✓We are Passing an **empty string as the ip address** means that the **server will be listening** on any network interface (all available IP addresses).

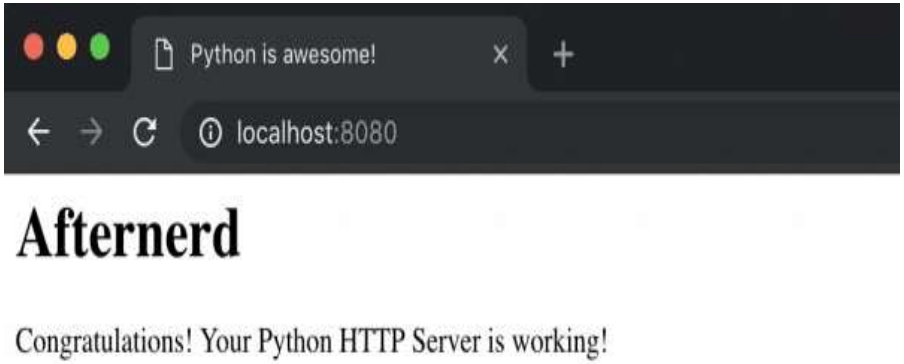
✓Handler is **http.server.SimpleHTTPRequestHandler**

✓ **PORT** stores the value of 8080, then the server will be listening on incoming requests on that port.

✓**Serve_forever** is a method on the **TCPServer instance** that starts the server and begins listening and responding to incoming requests.

✓save this file as **server.py** in *the same directory* as **index.html**

```
<html>
<head>
    <title>Python is awesome!</title>
</head>
<body>
    <h1>Afternerd</h1>
    <p>Congratulations! The HTTP Server is working!</p>
</body>
</html>
```



✓In that directory, start the web server:

$ python server.py

serving at port 8080

✓Now we have an **HTTP server** that is listening on any interface at **port 8080** waiting for **incoming http requests**.

✓Open your browser and type **localhost:8080** in the address bar.