UNIT – V GUI AND WEB

UI design: Tkinter – Events – Socket Programming – Sending email – CGI: Introduction to CGI Programming, GET and POST Methods, File Upload.

GUI Programming in Python

Python provides various options for developing graphical user interfaces (GUIs).

- **Tkinter** – Tkinter is the Python interface to the Tk GUI toolkit shipped with Python.
- **wxPython** – This is an open-source Python interface for wxWindows
- **JPython** – JPython is a Python port for Java which gives Python scripts seamless access to Java class libraries on the local machine

Tkinter Programming

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with tkinter outputs the fastest and easiest way to create the GUI applications. Creating a GUI using tkinter is an easy task.

To create a tkinter:

1. Importing the module – tkinter
2. Create the main window (container)
3. Add any number of widgets to the main window
4. Apply the event Trigger on the widgets.
5. Enter the main event loop to take action against each event triggered by the user.

Importing tkinter is same as importing any other module in the python code.

```
import tkinter
```

There are two main methods used you the user need to remember while creating the Python application with GUI.

1. Tk(screenName=None, baseName=None, className='Tk', useTk=1): To create a main window, tkinter offers a method 'Tk(screenName=None, baseName=None, className='Tk', useTk=1)'. 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:
   ```
   m=tkinter.Tk() where m is the name of the main window object
   ```
2. mainloop(): There is a method known by the name mainloop() is used when you are ready for the application to run. mainloop() is an infinite loop used to run the application, wait for an event to occur and process the event till the window is not closed.

```
m.mainloop()
```

Tkinter Widgets

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.

There are currently 15 types of widgets in Tkinter.

S.No.	Operator & Description
1	**Button :** The Button widget is used to display buttons in your application.
2	**Canvas :** The Canvas widget is used to draw shapes, such as lines, ovals, polygons and rectangles, in your application.
3	**Checkbutton :** The Checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time.
4	**Entry:** The Entry widget is used to display a single-line text field for accepting values from a user.
5	**Frame :** The Frame widget is used as a container widget to organize other widgets.
6	**Label :** The Label widget is used to provide a single-line caption for other widgets. It can also contain images.
7	**Listbox:** The Listbox widget is used to provide a list of options to a user.
8	**Menubutton :** The Menubutton widget is used to display menus in your application.
9	**Menu :** The Menu widget is used to provide various commands to a user. These commands are contained inside Menubutton.
10	**Message :** The Message widget is used to display multiline text fields for accepting values from a user.
11	**Radiobutton :** The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time.
12	**Scale :** The Scale widget is used to provide a slider widget.

13	**Scrollbar :** The Scrollbar widget is used to add scrolling capability to various widgets, such as list boxes.
14	**Text :** The Text widget is used to display text in multiple lines.
15	**Toplevel :** The Toplevel widget is used to provide a separate window container.
16	**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.
17	**PanedWindow :** A PanedWindow is a container widget that may contain any number of panes, arranged horizontally or vertically.
18	**LabelFrame :** A labelframe is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts.
19	**MessageBox :** This module is used to display message boxes in your applications.

Geometry Management

All Tkinter widgets have access to specific geometry management methods, which have the purpose of organizing widgets throughout the parent widget area. Tkinter exposes the following geometry manager classes: pack, grid, and place.

- The *pack()* Method – This geometry manager organizes widgets in blocks before placing them in the parent widget.
- The *grid()* Method – This geometry manager organizes widgets in a table-like structure in the parent widget.
- The *place()* Method – This geometry manager organizes widgets by placing them in a specific position in the parent widget.

Creation of Button

Button:To add a button in your application, this widget is used.
The general syntax is:
w=Button(master, option=value)

master is the parameter used to represent the parent window.
There are number of options which are used to change the format of the Buttons. Number of options can be passed as parameters separated by commas. Some of them are listed below.

- **activebackground**: to set the background color when button is under the cursor.

- **activeforeground**: to set the foreground color when button is under the cursor.
- **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.

```
import tkinter as tk
r = tk.Tk()
r.title('Counting Seconds')
button = tk.Button(r, text='Stop', width=25, command=r.destroy)
button.pack()
r.mainloop()
```

Event Handler in Python

A Tkinter application runs most of its time inside an event loop, which is entered via the mainloop method. It waiting for events to happen. Events can be key presses or mouse operations by the user.

Tkinter provides a mechanism to let the programmer deal with events. For each widget, it's possible to bind Python functions and methods to an event.

widget.bind(event, handler)

If the defined event occurs in the widget, the "handler" function is called with an event object. describing the event.

```
from tkinter import *
def hello(event):
    print("Single Click, Button-l")
def quit(event):
    print("Double Click, so let's stop")
    import sys; sys.exit()
widget = Button(None, text='Mouse Clicks')
```

```
widget.pack()
widget.bind('<Button-1>', hello)
widget.bind('<Double-1>', quit)
widget.mainloop()
```

Let's have another simple example, which shows how to use the motion event, i.e. if the mouse is moved inside of a widget:

```
from tkinter import *
def motion(event):
  print("Mouse position:" (event.x, event.y))
  return
master = Tk()
whatever_you_do = "Whatever you do will be insignificant, but it is very important that you do it.\n(Mahatma Gandhi)"
msg = Message(master, text = whatever_you_do)
msg.config(bg='lightgreen', font=('times', 24, 'italic'))
msg.bind('<Motion>',motion)
msg.pack()
mainloop()
```

Every time we move the mouse in the Message widget, the position of the mouse pointer will be printed. When we leave this widget, the function motion() is not called anymore

Python Socket Programming

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server.

To understand python socket programming it is necessary to know about following topics

Socket.

Socket is the endpoint of a bidirectional communications channel between server

and client. Sockets may communicate within a process, between processes on the same machine, or between processes on different machines. For any communication with a remote program, we have to connect through a socket port.

Server

A server is a software that waits for client requests and serves or processes them accordingly.

Client

A client is requester of this service. A client program request for some resources to the server and server responds to that request.

The *socket* Module

To create a socket, it is necessary to use the *socket.socket()* function available in *socket* module, which has the general syntax –

s = socket.socket (socket_family, socket_type, protocol=0)

Here is the description of the parameters –

- **socket_family** – This is either AF_UNIX or AF_INET, as explained earlier.
- **socket_type** – This is either SOCK_STREAM or SOCK_DGRAM.
- **protocol** – This is usually left out, defaulting to 0.

Once you have *socket* object, then you can use required functions to create your client or server program.

Server Socket Methods

S.No.	Method & Description
1	**s.bind()** This method binds address (hostname, port number pair) to socket.
2	**s.listen()** This method sets up and start TCP listener.
3	**s.accept()** This passively accept TCP client connection, waiting until connection arrives

	(blocking).

Client Socket Methods

S.No.	Method & Description
1	**s.connect()** This method actively initiates TCP server connection.

General Socket Methods

S.No.	Method & Description
1	**s.recv()** This method receives TCP message
2	**s.send()** This method transmits TCP message
3	**s.recvfrom()** This method receives UDP message
4	**s.sendto()** This method transmits UDP message
5	**s.close()** This method closes socket
6	**socket.gethostname()** Returns the hostname.

A Simple Server

To write Internet servers, we use the **socket** function available in socket module to create a socket object. A socket object is then used to call other functions to setup a socket server.

A server has a bind() method which binds it to a specific ip and port so that it can listen to incoming requests on that ip and port. A server has a listen() method which puts the server into listen mode. This allows the server to listen to incoming connections. And last a server has an accept() and close() method. The accept method initiates a connection with the client and the close method closes the connection with the client.

```
import socket               # Import socket module

s = socket.socket()         # Create a socket object
host = socket.gethostname() # Get local machine name
port = 12345                # Reserve a port for your service.
s.bind((host, port))        # Bind to the port

s.listen(5)                 # Now wait for client connection.
while True:
   c, addr = s.accept()     # Establish connection with client.
   print('Got connection from', addr)
   c.send('Thank you for connecting')
   c.close()                # Close the connection
```

- First of all we import socket which is necessary.
- Then we made a socket object and reserved a port on our pc.
- After that we binded our server to the specified port. Passing an empty string means that the server can listen to incoming connections from other computers as well. If we would have passed 127.0.0.1 then it would have listened to only those calls made within the local computer.
- After that we put the server into listen mode.5 here means that 5 connections are kept waiting if the server is busy and if a 6th socket trys to connect then the connection is refused.
- At last we make a while loop and start to accept all incoming connections and close those connections after a thank you message to all connected sockets.

A Simple Client

Let us write a very simple client program which opens a connection to a given port 12345 and given host. This is very simple to create a socket client using Python's *socket* module function.

The **socket.connect(hosname, port)** opens a TCP connection to *hostname* on the *port*. Once you have a socket open, you can read from it like any IO object. When done, remember to close it, as you would close a file.

The following code is a very simple client that connects to a given host and port, reads any available data from the socket, and then exits −

```
import socket               # Import socket module

s = socket.socket()         # Create a socket object
host = socket.gethostname() # Get local machine name
port = 12345                # Reserve a port for your service.

s.connect((host, port))
print s.recv(1024)
s.close()                   # Close the socket when done
```

Now run this server.py in background and then run above client.py to see the result.

```
# Following would start a server in background.
$ python server.py &

# Once server is started run client as follows:
$ python client.py
```

Output

```
Got connection from ('127.0.0.1', 48437)
Thank you for connecting
```

Sending email

Python, being a powerful language don't need any external library to import and offers a native library to send emails- "SMTP lib". "smtplib" creates a Simple Mail Transfer Protocol client session object which is used to send emails to any valid email id on the internet.

Different websites use different port numbers. If Gmail account is used to send a mail, then its port number is used. Port number used here is '587'. if other mail is used for sending information, it is necessary to get the corresponding information.

Steps to send mail from Gmail account:

1. First of all, "smtplib" library needs to be imported.
2. After that, to create a session, we will be using its instance SMTP to encapsulate an SMTP connection.
   s = smtplib.SMTP('smtp.gmail.com', 587)

   In this, you need to pass the first parameter of the server location and the second parameter of the port to use. For Gmail, we use port number 587.
3. For security reasons, now put the SMTP connection in the TLS mode. TLS (Transport Layer Security) encrypts all the SMTP commands. After that, for security and

authentication, you need to pass your Gmail account credentials in the login instance. The compiler will show an authentication error if you enter invalid email id or password.

4. Store the message you need to send in a variable say, message. Using the sendmail() instance, send your message. sendmail() uses three parameters: **sender_email_id, receiver_email_id and message_to_be_sent**. The parameters need to be in the same sequence.

Program:

```
# Python code to illustrate Sending mail from your Gmail account
import smtplib
# creates SMTP session
s = smtplib.SMTP('smtp.gmail.com', 587)
# start TLS for security
s.starttls()
# Authentication
s.login("sender_email_id", "sender_email_id_password")
# message to be sent
message = "Message_you_need_to_send"

# sending the mail
s.sendmail("sender_email_id", "receiver_email_id", message)
# terminating the session
s.quit()
```

Sending same message to multiple people

```
# Python code to illustrate Sending mail
# to multiple users
# from your Gmail account
import smtplib

# list of email_id to send the mail
li = ["xxxxx@gmail.com", "yyyyy@gmail.com"]

for i in range(len(li)):
    s = smtplib.SMTP('smtp.gmail.com', 587)
    s.starttls()
    s.login("sender_email_id", "sender_email_id_password")
    message = "Message_you_need_to_send"
    s.sendmail("sender_email_id", li[i], message)
    s.quit()
```

Adding Attachments Using the email Package

In order to send binary files to an email server that is designed to work with textual data, they need to be encoded before transport.

```
import email, smtplib, ssl
from email import encoders
from email.mime.base import MIMEBase
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

subject = "An email with attachment from Python"
body = "This is an email with attachment sent from Python"
sender_email = "my@gmail.com"
receiver_email = "your@gmail.com"
password = input("Type your password and press enter:")

# Create a multipart message and set headers
message = MIMEMultipart()
message["From"] = sender_email
message["To"] = receiver_email
message["Subject"] = subject
message["Bcc"] = receiver_email  # Recommended for mass emails
# Add body to email
message.attach(MIMEText(body, "plain"))
filename = "document.pdf"  # In same directory as script
# Open PDF file in binary mode
with open(filename, "rb") as attachment:
    # Add file as application/octet-stream
    # Email client can usually download this automatically as attachment
    part = MIMEBase("application", "octet-stream")
    part.set_payload(attachment.read())

# Encode file in ASCII characters to send by email
encoders.encode_base64(part)

# Add header as key/value pair to attachment part
part.add_header(
    "Content-Disposition",
    f"attachment; filename= {filename}",
)
```

```
# Add attachment to message and convert message to string
message.attach(part)
text = message.as_string()
# Log in to server using secure context and send email
context = ssl.create_default_context()
with smtplib.SMTP_SSL("smtp.gmail.com", 465, context=context) as server:
    server.login(sender_email, password)
    server.sendmail(sender_email, receiver_email, text)
```

CGI Programming

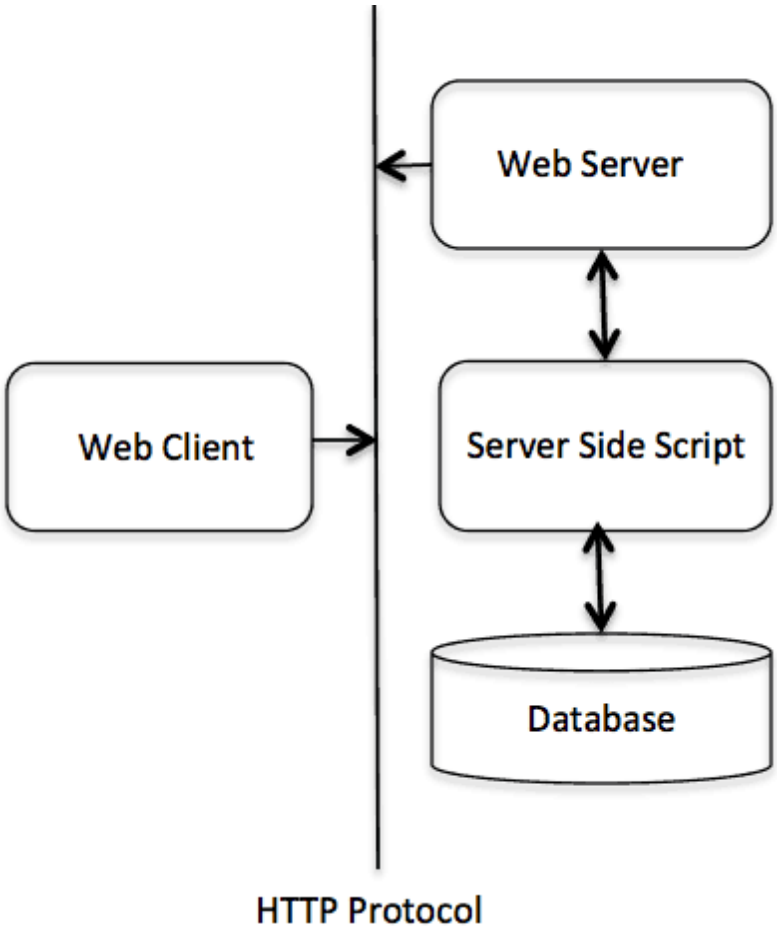
The Common Gateway Interface, or CGI, is a set of standards that define how information is exchanged between the web server and a custom script.

To understand the concept of CGI, let us see what happens when we click a hyper link to browse a particular web page or URL.

- Your browser contacts the HTTP web server and demands for the URL, i.e., filename.
- Web Server parses the URL and looks for the filename. If it finds that file then sends it back to the browser, otherwise sends an error message indicating that you requested a wrong file.
- Web browser takes response from web server and displays either the received file or error message.

However, it is possible to set up the HTTP server so that whenever a file in a certain directory is requested that file is not sent back; instead it is executed as a program, and whatever that program outputs is sent back for your browser to display. This function is called the Common Gateway Interface or CGI and the programs are called CGI scripts. These CGI programs can be a Python Script, PERL Script, Shell Script, C or C++ program, etc.

CGI Architecture Diagram



First CGI Program

Install apache2 on your system can we will run 'hello.py' on host '127.0.0.1'

```
print("content-type:text/html\r\n\r\n")
print('<html>')
print('<head>')
print('<title>hello word - first cgi program</title>')
print('</head>')
print('<body>')
print('<h2>hello word! this is my first cgi program</h2>')
print('</body>')
print('</html>')
```

If you click hello.py, then this produces the following output

Hello Word! This is my first CGI program

CGI Environment Variables

All the CGI programs have access to the following environment variables. These variables play an important role while writing any CGI program.

Sr.No.	Variable Name & Description
1	**CONTENT_TYPE** The data type of the content. Used when the client is sending attached content to the server. For example, file upload.
2	**CONTENT_LENGTH** The length of the query information. It is available only for POST requests.
3	**HTTP_COOKIE** Returns the set cookies in the form of key & value pair.
4	**HTTP_USER_AGENT** The User-Agent request-header field contains information about the user agent originating the request. It is name of the web browser.
5	**PATH_INFO** The path for the CGI script.
6	**QUERY_STRING** The URL-encoded information that is sent with GET method request.
7	**REMOTE_ADDR** The IP address of the remote host making the request. This is useful logging or for authentication.
8	**REMOTE_HOST** The fully qualified name of the host making the request. If this information is not available, then REMOTE_ADDR can be used to get IR address.
9	**REQUEST_METHOD** The method used to make the request. The most common methods are GET and POST.

10	**SCRIPT_FILENAME** The full path to the CGI script.
11	**SCRIPT_NAME** The name of the CGI script.
12	**SERVER_NAME** The server's hostname or IP Address
13	**SERVER_SOFTWARE** The name and version of the software the server is running.

GET and POST Methods

You must have come across many situations when you need to pass some information from your browser to web server and ultimately to your CGI Program. Most frequently, browser uses two methods two pass this information to web server. These methods are GET Method and POST Method.

Passing Information using GET method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? character as follows –

http://www.test.com/cgi-bin/hello.py?key1=value1&key2=value2

The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use GET method if you have password or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be sent in a request string. The GET method sends information using QUERY_STRING header and will be accessible in your CGI Program through QUERY_STRING environment variable.

You can pass information by simply concatenating key and value pairs along with any URL or you can use HTML <FORM> tags to pass information using GET method.

Simple URL Example:Get Method

Here is a simple URL, which passes two values to hello_get.py program using GET method.

/cgi-bin/hello_get.py?first_name=ZARA&last_name=ALI

Below is **hello_get.py** script to handle input given by web browser. We are going to use **cgi** module, which makes it very easy to access passed information −

```
#!/usr/bin/python

# Import modules for CGI handling
import cgi, cgitb

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
first_name = form.getvalue('first_name')
last_name  = form.getvalue('last_name')

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Hello - Second CGI Program</title>"
print "</head>"
print "<body>"
print "<h2>Hello %s %s</h2>" % (first_name, last_name)
print "</body>"
print "</html>"
```

This would generate the following result −

Hello ZARA ALI

Simple FORM Example:GET Method

This example passes two values using HTML FORM and submit button. We use same CGI script hello_get.py to handle this input.

```
<form action = "/cgi-bin/hello_get.py" method = "get">
First Name: <input type = "text" name = "first_name">  <br />

Last Name: <input type = "text" name = "last_name" />
<input type = "submit" value = "Submit" />
</form>
```

Here is the actual output of the above form, you enter First and Last Name and then click

submit button to see the result.

First Name:

Last Name: Submit

Passing Information Using POST Method

A generally more reliable method of passing information to a CGI program is the POST method. This packages the information in exactly the same way as GET methods, but instead of sending it as a text string after a ? in the URL it sends it as a separate message. This message comes into the CGI script in the form of the standard input.

Below is same hello_get.py script which handles GET as well as POST method.

```
#!/usr/bin/python

# Import modules for CGI handling
import cgi, cgitb

# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
first_name = form.getvalue('first_name')
last_name  = form.getvalue('last_name')

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Hello - Second CGI Program</title>"
print "</head>"
print "<body>"
print "<h2>Hello %s %s</h2>" % (first_name, last_name)
print "</body>"
print "</html>"
```

Let us take again same example as above which passes two values using HTML FORM and submit button. We use same CGI script hello_get.py to handle this input.

```
<form action = "/cgi-bin/hello_get.py" method = "post">
First Name: <input type = "text" name = "first_name"><br />
```

```
Last Name: <input type = "text" name = "last_name" />

<input type = "submit" value = "Submit" />
</form>
```

Here is the actual output of the above form. You enter First and Last Name and then click submit button to see the result.

First Name:

Last Name: Submit

File Upload Example

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 –

File: Browse... No file selected.

Upload