Unit-4 (web programming)

Introduction:

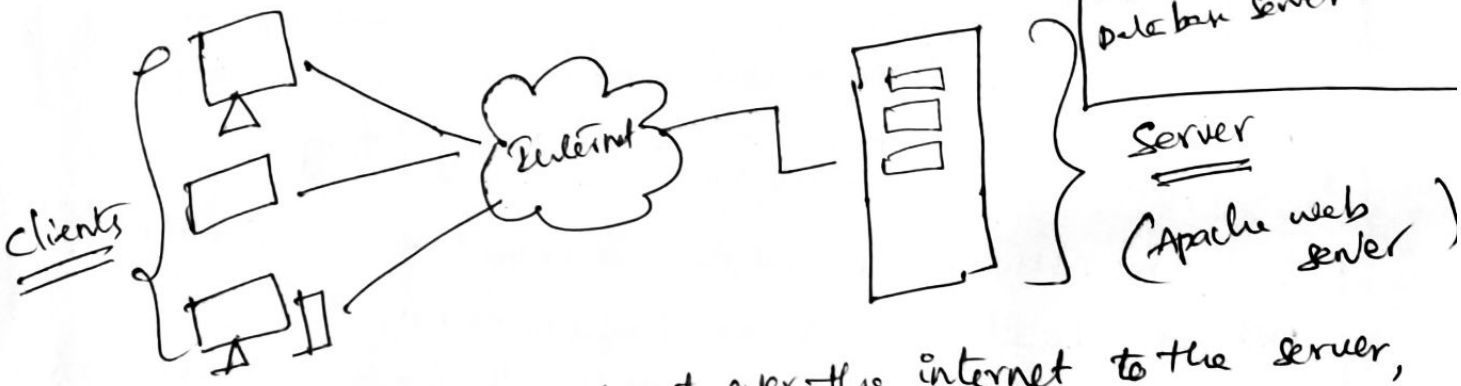
→ web surfing :- [client/server]

Server advantages
Backup, security & storage.

web clients → web clients are browsers, applications that allow users to seek documents on the world wide web.

web servers: web servers are processes that run on an information provider's host computer. These servers wait for clients and their document requests, process them, and return the requested data. These are designed to run forever.

types of servers
File server, Application, mail servers, web server, database server.



→ A client sends a request out over the internet to the server, which then responds with the requested data back to the client

→ client may issue a variety of requests to servers which may include obtaining a web page for viewing (or) submitting a form with data for processing. The request is then serviced by the web server, and the reply comes back to the client in a special format for display purposes.

→ The standard protocol user for web communication is HTTP (HyperText Transfer protocol). It is written on top of the TCP/IP protocol.

HTTP is known as a 'stateless' protocol & connectionless protocol. It allows to transfer the data in the form of plain text, hypertext, audio, video and so on.

.HTTP, FTP, POP, MIME (Multipurpose Internet mail extension.

protocol: It is a set of rules which are used in digital communication to connect two devices and exchange information between them.

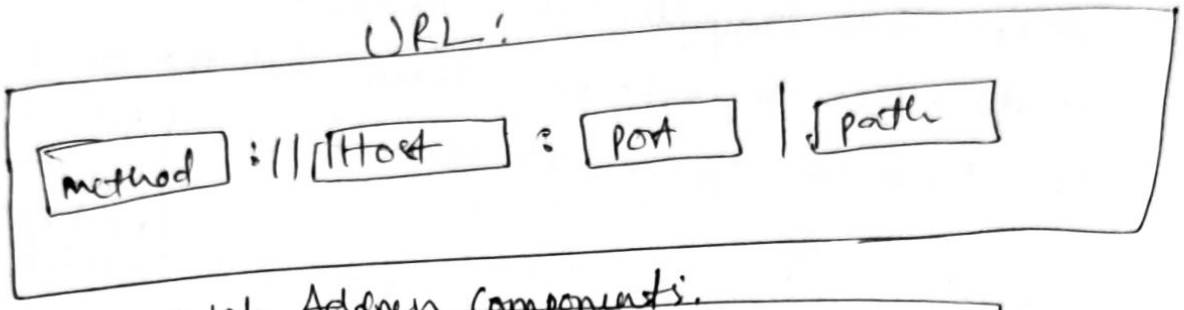
Uniform Resource Locator (URL) ← web surfing with python

→ A client that wants to accen the document in an internet needs an address and to facilitate the accen of document the HTTP uses the concept of URL.

→ The URL is a standard way of specifying any kind of information on the internet.

It defines four parts: method, host computer, port and path

URL:



web Address components:

prot_sch://net_loc/path;params?query#frag

prot_sch: Network protocol (or) download scheme (ex: http, https, ftps, ...

net_loc: Location of server (refers to web page)

path: slash (/) delimited path to file (or) CGI application (refers to file)

params: optional parameters

query: Ampersand (&) delimited set of "key=value" pairs

frag: Fragment to a specific anchor within document.

(It is internal page reference which refers to a section within the web page.)

Network Location components

user:passwd@host:port

HTTPS:
HyperText Transfer protocol secure

user: user name (or) login

passwd: user password

host: Name (or) address of machine running web server

port: port number [if not 80, the default]

https://www.example.co.uk:443/blog/article/search?docid=720&hl=en#dayfragment

- https: scheme
- www: subdomain
- example: domain
- co.uk: top-level domain
- 443: port number
- /blog/article/search: path
- ?: query string separator
- docid=720&hl=en: query string parameters
- #dayfragment: fragment

urllib module (urlparse)

url schemes : ftp, http, https, hdl, news, imap, mms, snews, telnet, ws, wss. . . .

① urlparse() : It parses a URL into six components, returning a 6-tuple. This corresponds to the general structure of a URL. Each tuple is a string.

Attribute	Index	Value
scheme	0	URL scheme specifier
netloc	1	Network location part
path	2	Hierarchical path
params	3	parameters for last path element
query	4	Query component
fragment	5	Fragment identifier

url parse module

```
from urllib.parse import urlparse
url = 'https://mail.google.com/mail/u/0/?tab=rm#inbox'
t = urlparse(url)
print(t)
```

```
ParseResult(scheme='https', netloc='mail.google.com', path='/mail/u/0/',
            params='', query='tab=rm', fragment='inbox')
```

② urlunparse(parts) ⇒ It constructs a URL from a tuple as returned by urlparse().

```
from urllib.parse import urlunparse
url = urlunparse(t)
print(url)
```

```
o/p : https://mail.google.com/mail/u/0/?tab=rm#inbox.
```

③ urlsplit() : It is similar to urlparse() but does not split the params from the URL. It returns a 5-tuple: (addressing scheme, network location, path, query, fragment id)

```
from urllib.parse import urlsplit
url = 'https://mail.google.com/mail/u/0/?tab=rm#inbox'

t = urlsplit(url)
print(t)
SplitResult (scheme='https', netloc='mail.google.com', path='/mail/u/0/',
             query='tab=rm', fragment='inbox').
```

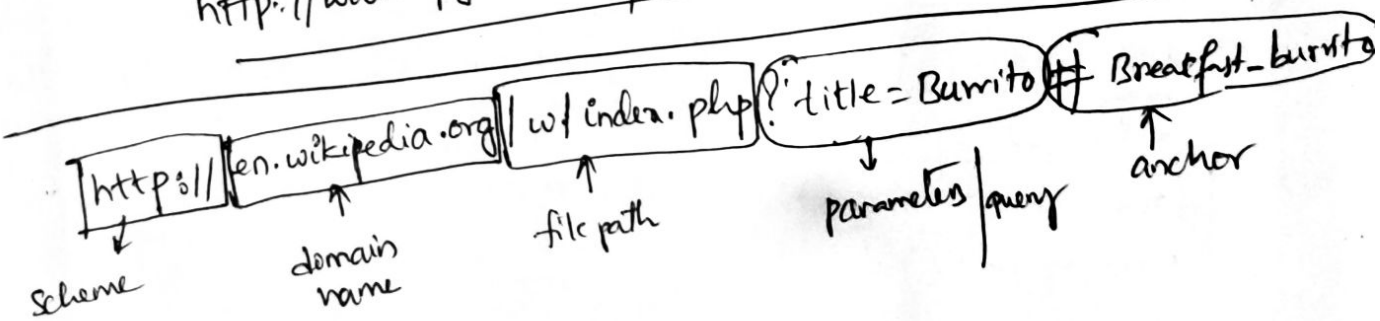
④ urlparse.urljoin()

It is useful in cases where many related URLs are needed. for ep: the urls for a set of pages to be generated for a website: urljoin(base url, new url, allow_fragments=None)

```
t = urljoin('http://www.python.org/doc/FAQ.html',
            'current/lib/lib.html')
```

↓↓

http://www.python.org/doc/current/lib/lib.html



urlparse module functions

① urlparse (urlstr, defprotsch = None, allowFrag = None)

→ parses urlstr into separate components, using defProtsch if the protocol (or) scheme is not given in urlstr; allowFrag determines whether a URL fragment is allowed.

② urlunparse (urltup):

Unparses a tuple of URL data (urltup) into a single URL string

③ urljoin (baseurl, newurl, allowFrag = None)

Merges the base part of the baseurl URL with newurl to form a complete URL; allowFrag is the same as for urlparse()

---

urllib module

It provides functions to download data from given URLs as well as encoding and decoding strings to make them suitable for including as part of valid URL strings.

Functions: urlopen(), urlretrieve(), quote(), unquote(), quote_plus(), unquote_plus() and urlencode().

urllib.urlopen()

It opens a web connection to the given URL string and returns a file-like object.

Syntax: urlopen (urlstr, postQueryData = None) → opens the url pointed to by urlstr.

For all HTTP requests, the normal request type is "GET"
If the "POST" request method is desired, then the query string should be placed in the postQueryData variable.

→ when a successful connection is made, urlopen() returns a file-like object as if the destination was a file opened in read mode.

methods: f.read(), f.readline(), f.readlines(), f.close(), f.fileno()

File like object methods

f.read([bytes]) → Reads all (or) bytes from f
f.readline() → Reads a single line from f
f.readlines() → Reads all lines from f into a list
f.close() → closes URL connection for f
f.fileno() → returns filenumber of f
f.info() → Gets MIME headers of f
(Multipurpose Internet Mail Extension)
f.geturl() → returns true URL opened for f

urllib.urlretrieve()

urlretrieve(urlstr, localfile=None, downloadstatushook=None)
↳ It will simply download the entire HTML file located at urlstr to your local disk. It will store the downloaded data into local file if given. The downloadstatushook is a function that is called after each block of data has been downloaded and delivered. It has three arguements: no of blocks read so far, the block size in bytes, and the total size of the file. (useful for download status.)

urllib.quote)

The quote*() functions take URL data and "encodes" them so that they are fit for inclusion as part of a URL string. Certain special characters that are unprintable (or) cannot be part of valid urls acceptable to a webserver must be converted.

quote(urldata, safe='/')

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

The disallowed characters are changed to their hexadecimal ~~~~ ordinal equivalents prepended with a '%' i.e %xx where 'xx' is the hexadecimal representation of a character's ASCII value.

The safe string should contain a set of characters which should also not be converted.

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

```
>>> url = 'http://www/~foo/cgi-bin/s.py?name=joe&mama&num=6'
>>> urllib.quote(url)
'http%3a//www/%7efoo/cgi-bin/s.py%3fname%3djoe%26mama%26num%3d6'
>>> urllib.quote_plus(url)
'http%3a//www/%7efoo/cgi-bin/s.py%3fname%3djoe+mama%26num%3d6'
```

urllib functions

urlopen(urlstr) ⇒ opens a URL urlstr

urlretrieve(URLstr) ⇒ downloads a file located at the urlstr URL to local file.

quote(urldata) ⇒ Encodes invalid URL characters of urldata

quote_plus(urldata) ⇒ Same as quote() except encodes spaces as plus (+) signs

unquote(urldata) → Decodes encoded characters of urldata

unquote_plus(urldata) ⇒ Same as unquote() but converts plus signs to spaces.

urlencode(dict) ⇒ Encodes a key-value pairs of dict into a valid string for CGI queries.

HTML

- Hyper Text Markup Language
- used for creating webpages
- It describes the structure of a webpage
- It consists of a series of HTML elements which tells the browser how to display the content. These elements label pieces of content such as "this is a heading", "this is a paragraph", ...etc.

HTML page structure

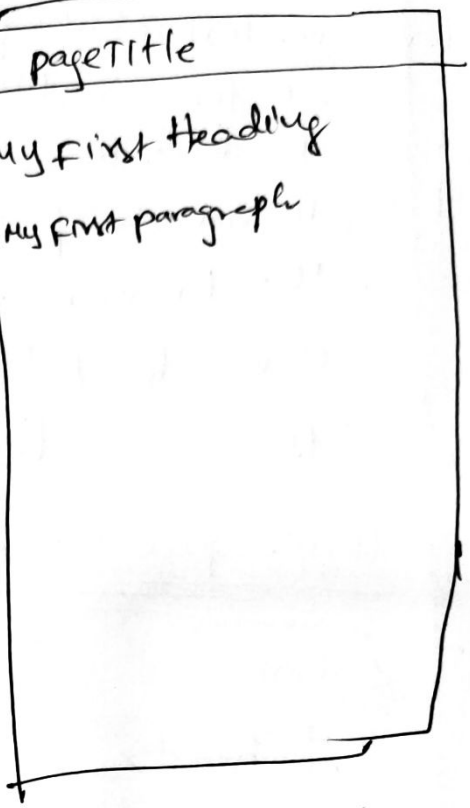
```
<html>
    <head>
        <title> page title </title>
    </head>
    <body>
        <h1> This is a heading </h1>
        <p> This is a paragraph. </p>
        <p> This is another paragraph </p>
    </body>
</html>
```

<!DOCTYPE html> defines that this is an HTML5 document
<html> element is the root element of an HTML page
<head> element contains meta information about the HTML page
<title> Specifies a title for the HTML page
<body> defines the document's body (headings, paragraphs, images...)
<h1> defines a large heading
<p> defines a paragraph

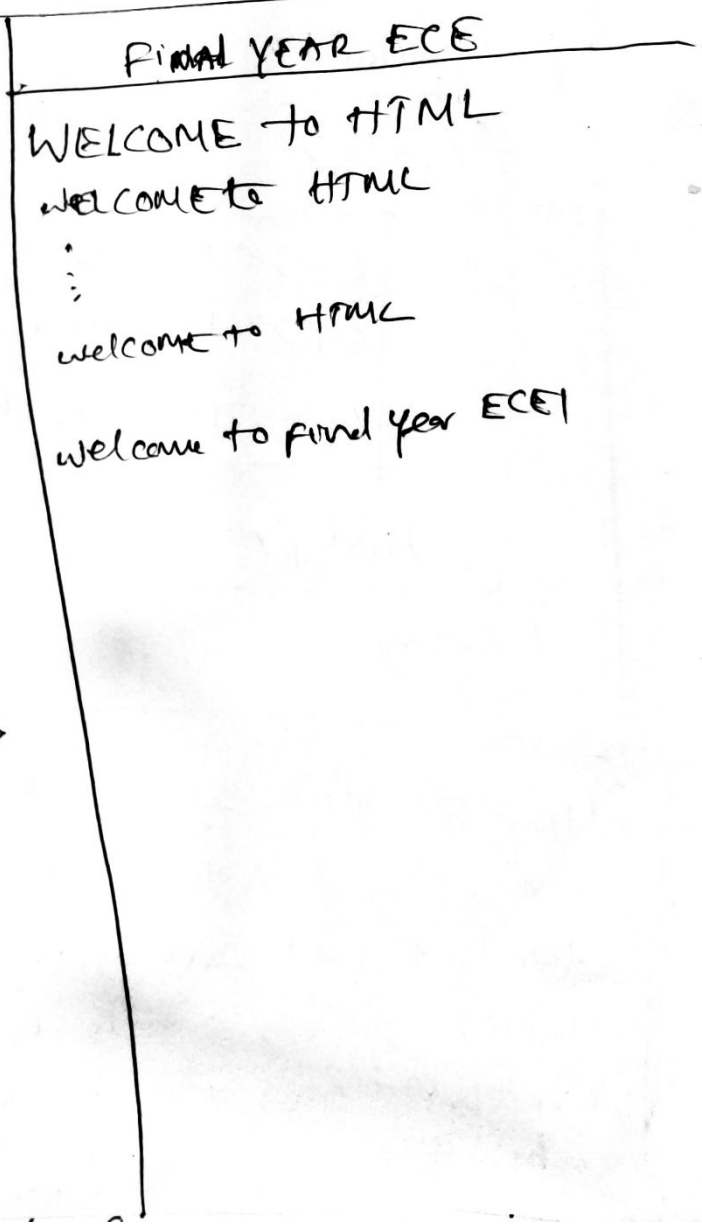
A simple HTML DOCUMENT

Python
print ("Content-type: text/html\r\n\r\n")

```
<!DOCTYPE html>
<html>
 <head>
  <title> page Title </title>
 </head>
 <body>
  <h1> My First Heading </h1>
  <p> My first paragraph. </p>
 </body>
</html>
```




```
<!DOCTYPE html>
<html>
 <head>
  <title> FINAL YEAR ECE </title>
 </head>
 <body>
  <h1> welcome to html </h1>
  <h2> welcome to html </h2>
   :
  <h6> welcome to html </h6>
  <p> welcome to Final year ECE! </p>
 </body>
</html>
```



CGI programming

CGI → Common Gateway Interface, is a set of standards that define how information is exchanged between the webserver and a custom script. The CGI specs are currently maintained by the NCSA [National Center for Supercomputing Applications] → one of the earliest web browsers.

→ CGI is a standard for external gateway programs to interface with information servers such as HTTP servers.

web browsing:

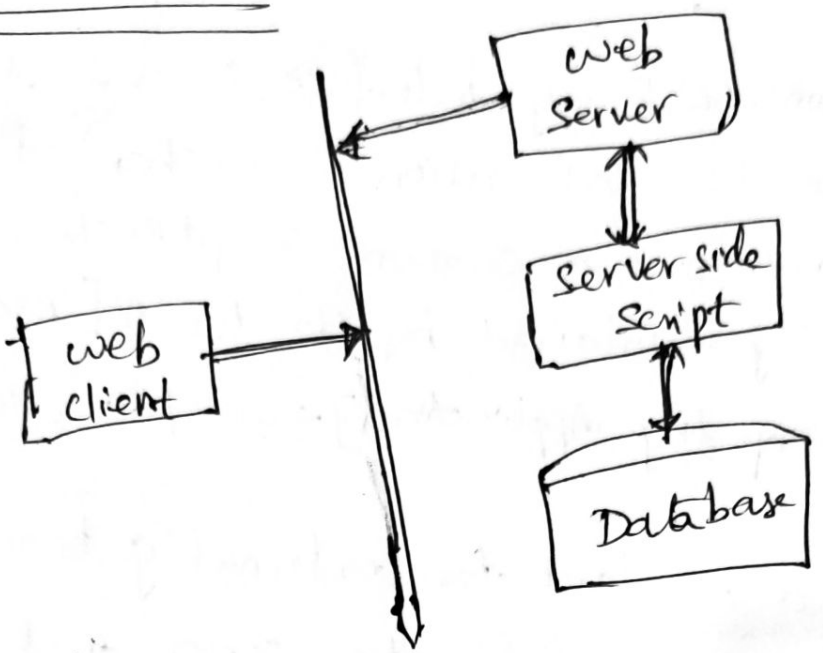
① your browser contacts the HTTP web server and demands for the URL i.e filename.

② 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.

③ 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 CGI and the programs are called CGI scripts. These scripts can be a python script, Perl script, Shell scripts, (or) C / C++ program...

CGI architecture



→ web server support and configuration:

Make sure that your web server supports CGI and it is configured to handle CGI programs. All the CGI programs to be executed by the HTTP server are kept in a pre-configured directory, known as CGI directory (cgi-bin)

```
#! usr/bin/ python

print ("Content-type : text/html\r\n\r\n")
print ('<html>')
print ('<head>')
print ('<title> Hello world - First program </title>')
print ('</head>')
print ('<body>')
print ('<h2> Helloworld! This is my first CGI program </h2>')
print ('</body>')
print ('</html>')
```

XAMPP/htdocs/PyCgi/one.py

O/P

Helloworld! this is my first CGI program

HTTP header.

HTTP FieldName : Field content

↳ Content-type: text/html\r\n\r\n

CGI Header Description

① Content-type: A MIME string defining the format of the file being returned. Ex: Content-type: text/html.

② Expires: Date : A valid date string in the format 01 Jan 2021 12:00:00 GMT

③ Location : URL : The URL that is returned instead of the URL requested:

④ Last-modified: Date: The date of last modification of the resource.

⑤ Content-length: N → The length, in bytes, of the data being returned. (to report the estimated download time)

⑥ Set-Cookie : String → set the cookie passed through the string.

CGI Environment Variables

① CONTENT-TYPE: The data type of the content

② CONTENT-LENGTH: The length of the query information (available for only POST requests)

③ HTTP-COOKIE: returns the set cookies in the form of key & value pair.

④ HTTP-USER-AGENT: It is name of the web browser

⑤ PATH-INFO: The path of the CGI script

⑥ QUERY-STRING: The URL-encoded information that is sent with GET method request.

⑦ REMOTE-ADDR: Ip address of the remote host

⑧ REMOTE-HOST: The fully qualified name of the host making the request.

⑨ REQUEST-METHOD: It is used to make the request. common methods are GET and POST.

⑩ SCRIPT-FILENO: The full path to the CGI script

⑪ SCRIPT-NAME: The name of the CGI script

⑫ SERVER-NAME: Server's host name or Ip.Address

⑬ SERVER-SOFTWARE: The name and version of the sfw

passing information using GET method

It sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? character as follows:

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

It is the default method to pass information from browser to the webserver and it produces a long string that appears in the browser's Location box. It has size limit: 1024 chars

It sends information using QUERY-STRING header and will be accessible in your CGI program through QUERY-STRING environment variable.

```
#!/usr/bin/python
import cgi, cgitb
form = cgi.FieldStorage()
firstname = form.getvalue('firstname')
lastname = form.getvalue('lastname')
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>" % (firstname, lastname))
print("</body>")
print("</html>")
```

xampp/htdocs/pycgi/hello.py

```
#!/usr/bin/python
print("content-type:text/html\r\n\r\n")
print("<form action = "/pycgi-bin/hello.py" method = "get">")
print("First Name: <input type = "text" name = "firstname"><br/>")
print("Lastname: <input type = "text" name = "lastname"/>")
print("<input type = "submit" value = "Submit" />")
print("</form>")
```

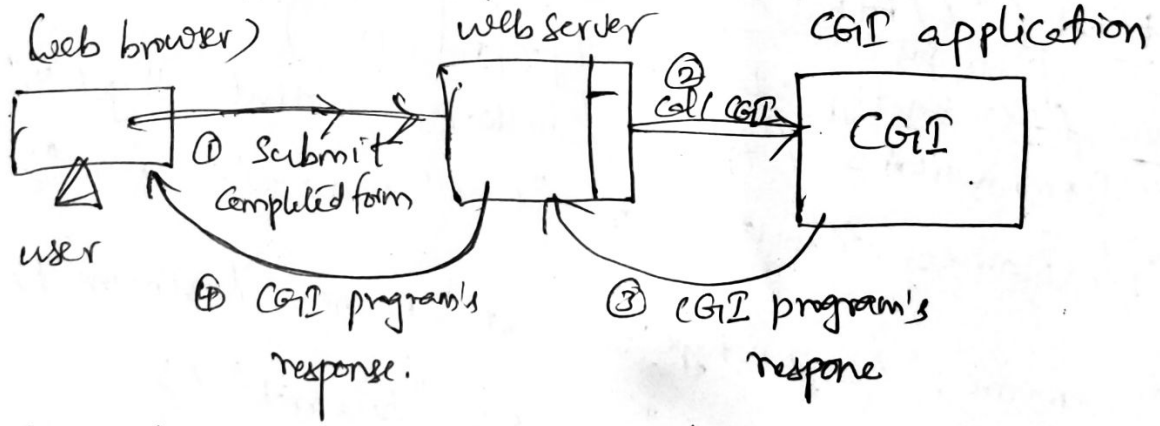
hello.py

xampp | htdocs | pycgi | test.py

c:\xampp\htdocs\pycgi\test.py

Field storage → store a sequence of fields, reading multipart/form-data. This class provides naming, typing, files stored on disk and more. At the top level, it is accessible like a dictionary, whose keys are the field names.

→ The entire process of web browsing begins when the web server receives a client request and calls the appropriate application. It then waits for the resulting HTML meanwhile, the client also waits. once the application has completed, it passes the dynamically generated HTML back to the server, who then forwards it back to the user. This process of the server receiving a form, contacting an external application, and receiving and returning the newly-generated HTML takes place through what is called the web server's CGI.



Field storage class → this class should be instantiated when a python CGI scripts begins, as it will read the user information from the web client. 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.

C:\xampp\htdocs\pycgi\test.py

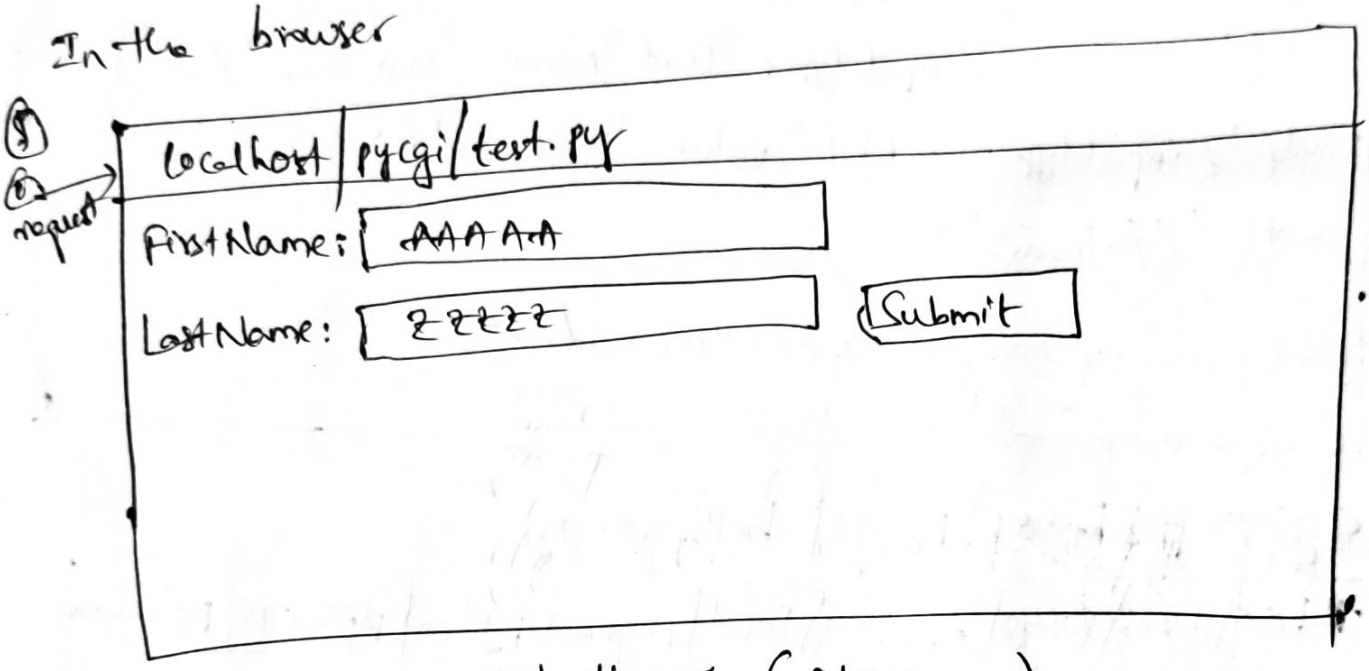
```
#!C:\Users\Divya\AppData\Local\Programs\Python\Python310\python.exe
print ("Content-type: text/html\r\n\r\n")
print ('<form action = "/pycgi/hello_get.py" method = "get">')
print ('First Name : <input type = "text" name = "first_name"> <br/>')
print ('Last Name : <input type = "text" name = "last_name" />')
print ('<input type = "submit" value = "Submit" />')
print ('</form>')
```

C:\xampp\htdocs\pycgi\hello_get.py

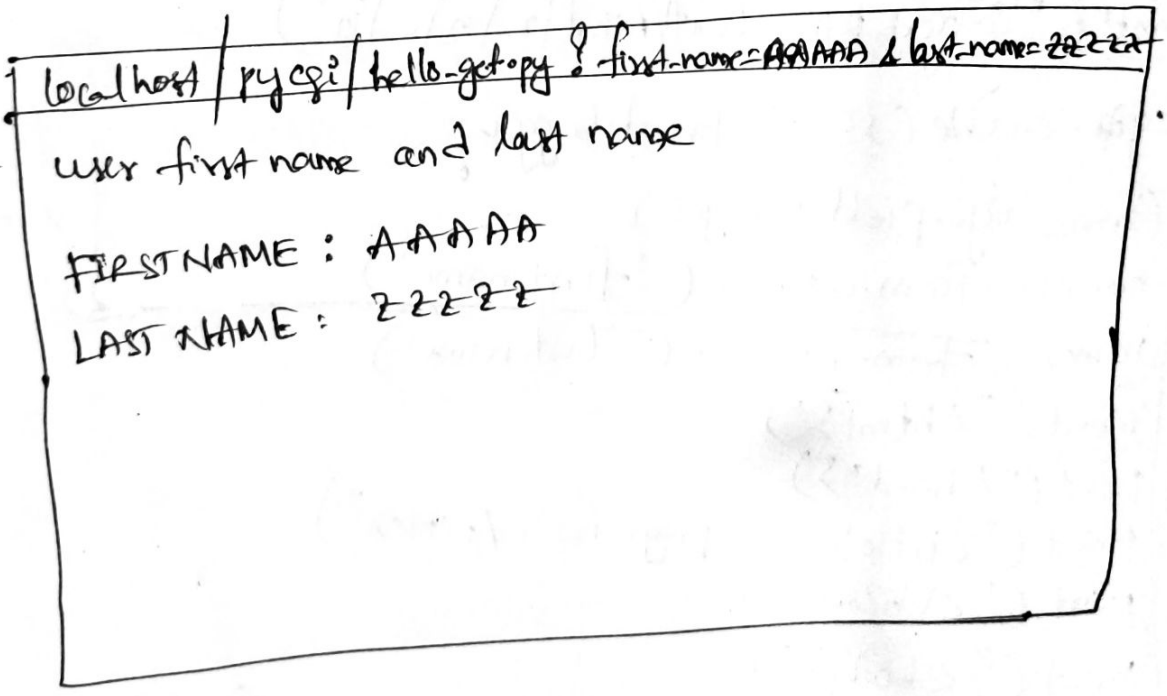
```
#!C:\Users\Divya\AppData\Local\Programs\Python\Python310\python.exe
import cgi, cgitb
print ("Content-type: text/html\r\n\r\n")
cgitb.enable()  #for debugging
form = cgi.FieldStorage()
name1 = form.getvalue('first_name')
name2 = form.getvalue('last_name')
print ("<html>")
print ("<head")
print ("<title> CGI-program2 </title>")
print ("</head>")
print ("<body>")
print ("<h3> user first name and last name </h3>")
print ("<br>")
```

```
print(" FIRST NAME : ", name1)
print ("<br>")
print ("LASTNAME : ", name2)
print ("</body>")
print ("</html>")
```

In the browser



③ Click on submit button ↓ (opens)



C:\xampp\htdocs\pycgi\cal.py

```
#!C:\Users\DIVYA\AppData\Local\Programs\Python\Python310\python.exe
print ("Content-type: text/html\r\n\r\n")
print ('<form action="/pycgi/cal.py' method="get">')
print ('num1: <input type="text" name="n1"> <br/>')
print ('num2: <input type="text" name="n2" />')
print ('<input type="submit" value="+" name="sub" />')
print ('<input type="submit" value="-" name="sub"/>')
print ('<input type="submit" value="*" name="sub"/>')
print ('<input type="submit" value="/" name="sub"/>')
print ('</form>')
```

C:\xampp\htdocs\pycgi\cal.py

```
#!C:\Users\DIVYA\AppData\Local\Programs\Python\Python310\python.exe
Print ("Content-type: text/html\r\n\r\n")
print()
import cgi, cgitb
cgitb.enable() #for debugging
form = cgi.FieldStorage()
sn1 = form.getvalue("n1")
sn2 = form.getvalue("n2")
a = int(sn1)
b = int(sn2)
```

```
op= form.getvalue('sub')
print("<html>")
print("<body>")
print("<h3> SIMPLE CALCULATOR </h3>")
if(op=="+"):
    print("ADDITION:", (a+b))
elif(op=="-"):
    print("SUBTRACTION:", (a-b))
elif(op=="*"):
    Print("MULTIPLICATION:", (a*b))
else:
    print("DIVISION:", (a/b))
print("</body>")
print("</html>")
```