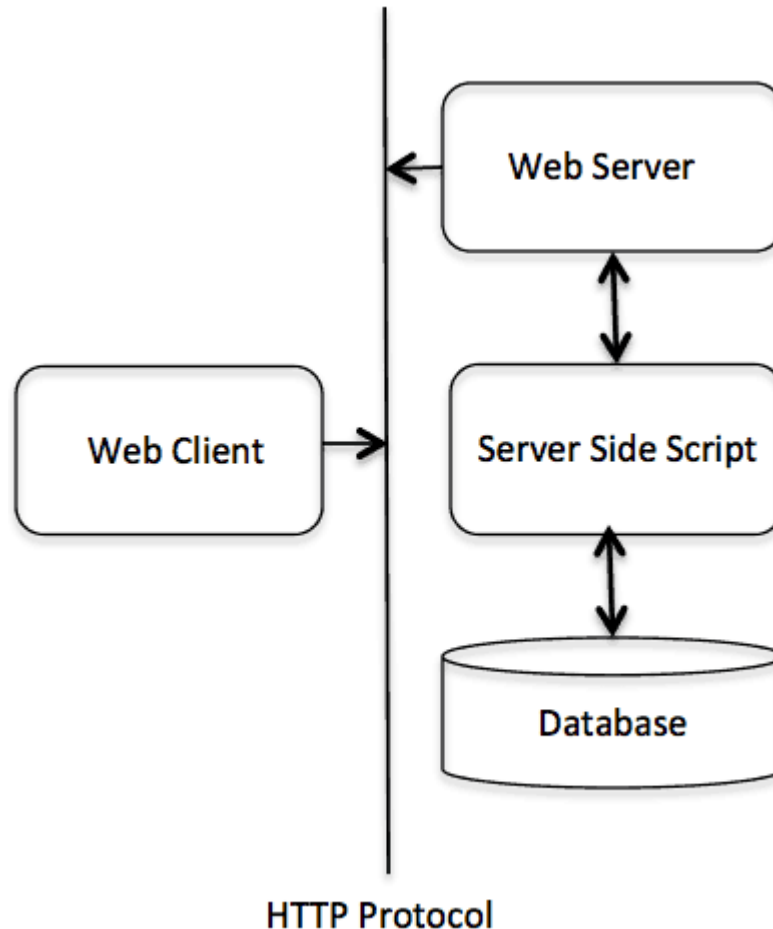


# What is CGI?

- The Common Gateway Interface (CGI)
  - It defines how information is exchanged between the web server and a custom script.
  - It is a standard for external gateway programs to interface with information servers such as HTTP servers.
- The current version is CGI/1.1 and CGI/1.2 is under progress.
- Web Browsing
  - Your browser contacts the HTTP web server and demands for the URL i.e., filename.
  - Web Server will parse the URL and will look for the filename in if it finds that file then sends it back to the browser, otherwise sends an error message indicating that you have requested a wrong file.

# CGI Architecture Diagram

- Web browser takes response from web server and displays either the received file or error message.



# Web Server Support & Configuration

- Please 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.
  - This directory is called CGI Directory and by convention it is named as `/var/www/cgi-bin`

```
<Directory "/var/www/cgi-bin">
  AllowOverride None
  Options ExecCGI
  Order allow,deny
  Allow from all
</Directory>

<Directory "/var/www/cgi-bin">
Options All
</Directory>
```

# First CGI Program

```
#!/usr/bin/python

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**

- There is one important and extra feature available which is first line to be printed **Content-type:text/html\r\n\r\n**.
- This line is sent back to the browser and specify the content type to be displayed on the browser screen.

# HTTP Header

- All the HTTP header will be in the following form:

```
HTTP Field Name: Field Content

For Example
Content-type: text/html\r\n\r\n
```

- There are few other important HTTP headers, which you will use frequently in your CGI Programming.

Header	Description
Content-type:	A MIME string defining the format of the file being returned. Example is Content-type:text/html <b>MIME</b> , <b>Multipurpose Internet Mail Extensions</b>
Expires: Date	The date the information becomes invalid. This should be used by the browser to decide when a page needs to be refreshed. A valid date string should be in the format 01 Jan 1998 12:00:00 GMT.
Location: URL	The URL that should be returned instead of the URL requested. You can use this field to redirect a request to any file.
Last-modified: Date	The date of last modification of the resource.
Content-length: N	The length, in bytes, of the data being returned. The browser uses this value to report the estimated download time for a file.
Set-Cookie: String	Set the cookie passed through the <i>string</i>

# CGI Environment Variables

- All the CGI program will have access to the following environment variables.

Variable Name	Description
CONTENT_TYPE	The data type of the content. Used when the client is sending attached content to the server. For example, file upload, etc.
CONTENT_LENGTH	The length of the query information. It's available only for POST requests.
HTTP_COOKIE	Returns the set cookies in the form of key & value pair.
HTTP_USER_AGENT	The User-Agent request-header field contains information about the user agent originating the request. Its name of the web browser.
PATH_INFO	The path for the CGI script.
QUERY_STRING	The URL-encoded information that is sent with GET method request.
REMOTE_ADDR	The IP address of the remote host making the request. This can be useful for logging or for authentication purpose.
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.
REQUEST_METHOD	The method used to make the request. The most common methods are GET and POST.
SCRIPT_FILENAME	The full path to the CGI script.
SCRIPT_NAME	The name of the CGI script.
SERVER_NAME	The server's hostname or IP Address
SERVER_SOFTWARE	The name and version of the software the server is running.

# Example

```
#!/usr/bin/python

import os

print "Content-type: text/html\r\n\r\n";
print "<font size=+1>Environment</font><\br>";
for param in os.environ.keys():
    print "<b>%20s</b>: %s<\br>" % (param, os.environ[param])
```

## 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 to pass this information to web server.
  - ✓ **GET**
  - ✓ **POST**

# 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.



# Passing Information using GET Method

- 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 will pass two values to hello\_get.py program using GET method.
- [/cgi-bin/hello\\_get.py?first\\_name=ZARA&last\\_name=ALI](/cgi-bin/hello_get.py?first_name=ZARA&last_name=ALI)

# Example

- 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>"
```

Hello ZARA ALI

# Simple FORM Example: GET Method

- Here is a simple example which passes two values using HTML FORM and submit button.
- We are going to 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:

# 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>"
```

# POST Method

- Let us take again same example as above which passes two values using HTML FORM and submit button.
- We are going to 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:

# Passing Checkbox Data to CGI Program

- Checkboxes are used when more than one option is required to be selected.
- Here is example HTML code for a form with two checkboxes:

```
<form action="/cgi-bin/checkbox.cgi" method="POST" target=" blank">  
<input type="checkbox" name="maths" value="on" /> Maths  
<input type="checkbox" name="physics" value="on" /> Physics  
<input type="submit" value="Select Subject" />  
</form>
```

- The result of this code is the following form:

Maths  Physics

- Below is checkbox.cgi script to handle input given by web browser for checkbox button.

# Checkbox Data

```
#!/usr/bin/python

# Import modules for CGI handling
import cgi, cgitb

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

# Get data from fields
if form.getvalue('maths'):
    math_flag = "ON"
else:
    math_flag = "OFF"

if form.getvalue('physics'):
    physics_flag = "ON"
else:
    physics_flag = "OFF"

print "Content-type:text/html\r\n\r\n"
print "<html>"
print "<head>"
print "<title>Checkbox - Third CGI Program</title>"
print "</head>"
print "<body>"
print "<h2> CheckBox Maths is : %s</h2>" % math_flag
print "<h2> CheckBox Physics is : %s</h2>" % physics_flag
print "</body>"
print "</html>"
```

# Using Cookies in CGI

- HTTP protocol is a stateless protocol.
  - But for a commercial website, it is required to maintain session information among different pages.
- For example, one user registration ends after completing many pages.
  - But 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.



# Using Cookies in CGI

- Your 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:
  1. **Expires** : The date the cookie will expire. If this is blank, the cookie will expire when the visitor quits the browser.
  2. **Domain** : The domain name of your site.
  3. **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.
  4. **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.
  5. **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 will be sent along with HTTP Header before to Content-type field.
- Assuming you want to set UserID and Password as cookies. So cookies setting will be done as follows:

```
#!/usr/bin/python
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\r\n"
print "Set-Cookie:Domain=www.tutorialspoint.com;\r\n"
print "Set-Cookie:Path=/perl;\r\n"
print "Content-type:text/html\r\n\r\n"
.....Rest of the HTML Content.....
```

- We use **Set-Cookie** HTTP header to set cookies.
- Here, it is optional to set cookies attributes like Expires, Domain and Path. It is notable that cookies are set before sending magic line "**Content-type:text/html\r\n\r\n**".

# Retrieving Cookies

- Cookies are stored in CGI environment variable `HTTP_COOKIE` and they will have following form:
  - **key1=value1;key2=value2;key3=value3....**
- Here is an example of how to retrieve cookies.

```
#!/usr/bin/python

# 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
```

```
User ID = XYZ
Password = XYZ123
```

# 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 will create 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:  No file chosen

# Example

- Here is the script `save_file.py` to handle file upload:

```
#!/usr/bin/python

import cgi, os
import cgi; cgi.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,)
```

If you are running above script on Unix/Linux, then you would have to take care of replacing file separator as follows, otherwise on your windows machine above `open()` statement should work fine.

```
fn = os.path.basename(fileitem.filename.replace("\\", "/"))
```

# How To Raise a "File Download" Dialog Box ?

- A user will click a link and it will pop up a "File Download" dialogue box to the user instead of displaying actual content.
- This is very easy and will be achieved through HTTP header.
- This HTTP header will be different from the header mentioned in previous section.
- For example, if you want make a **FileName** file downloadable from a given link, then its syntax will be as follows:

```
#!/usr/bin/python

# HTTP Header
print "Content-Type:application/octet-stream; name=\"FileName\"\\r\\n";
print "Content-Disposition: attachment; filename=\"FileName\"\\r\\n\\n";

# Actual File Content will go hear.
fo = open("foo.txt", "rb")

str = fo.read();
print str

# Close opened file
fo.close()
```