**Python training Notes:**

**Course Name:** **SCRIPT 502 Intermediate Python**

**Day 2: 21 Mar 2020**

**Expectation Setting : ASL (Assisted Self-Learning) 4Hrs session daily**

**Prerequisite course learning :** [**SCRIPT 302 Basic Python**](https://knowledgecenter.persistent.co.in/ViewCourse/Python302)

**Kindly ensure that Basic Python Module is completed before joining the Intermediate Session on Monday, 20th April 2020**

**And then do self-study and hands on assignments form below learning course link of –**

[**SCRIPT 502 Intermediate Python**](https://knowledgecenter.persistent.co.in/ViewCourse/pmoc)**:**

<https://knowledgecenter.persistent.co.in/ViewCourse/pmoc>

***Please visit the following URL to view the collaborative learning group*** <https://persistentuniversity.persistent.co.in/CollaborativeLearningGroup/View.aspx?SkillID=10543>

You can also view my Python Video recording as a part Smart India Hackathon 2018 at -

<https://www.youtube.com/watch?v=zF-0vJeIxt8&feature=youtu.be>

**\*\*\*Installation Guide:**

1. Python 3.x on windows  :Install **Python 3.x** on windows using below link : <https://www.python.org/downloads/>

**OR**

**Anaconda installation on Windows**

Use the link : <https://www.anaconda.com/distribution/#download-section>

Install Python 3.7 version

1. Text Editor—Notepad/Notepad++ or IDLE editor or PyCharm or Eclipse or Anaconda, Jupyter Notebook
2. MySQL 5.5
3. Please install Apache server from : <http://archive.apache.org/dist/httpd/binaries/win32/httpd-2.2.16-win32-x86-no_ssl.msi>

**Topics covered:**

**OOP …continued…**

Static method

Class method

Inheritance , multiple

Method overloading, method overriding

Special methods : \_\_str\_\_() \_\_repr\_\_() \_\_add\_\_()

**Functional programming: high order functions- map , filter , reduce**

For loop…

List comprehension

Map() function call

lambda anonymous functions

lambda with parameters

lambda with map()

filter() reduce() calls

complex sorting

**Database Programming**

**XML Parsing**

**CGI programming**

**Some miscellaneous modules - Miscellaneous Modules covered as listed in the folder – “Day 2\** **some modules” at CLG**

**Linux execution**

**Python execution in Eclipse**

**Jupyter Notebook – Anaconda Software**

**\*\*\*\*\*To Do for Day 2:**

**Learn all nuggets from below KC course link of -**

[**SCRIPT 502 Intermediate Python**](https://knowledgecenter.persistent.co.in/ViewCourse/pmoc)**:**

<https://knowledgecenter.persistent.co.in/ViewCourse/pmoc>

**Nugget 1 : Overview of Basic Python**

**Nugget 2 : Regular expressions**

**Nugget 3 : Introduction to OOP**

**Execute all codes from the folder for the mentioned folders from** [**CLG**](https://persistentuniversity.persistent.co.in/CollaborativeLearningGroup/View.aspx?SkillID=10543) **link-**

**Day2 –**

**Functional Programming**

**some modules**

**XML**

**CGIDemos**

**Database**

**Python Basics\_Notebooks**

**Notes**

## **Database Programming**

import sqlite3

conn = sqlite3.connect('test')

cursor = conn.cursor()

print cursor

#cursor.execute("create table studentdata(name text, count integer)")

cursor.execute("insert into student (name, count) values (?, ?)",("Jill", 15))

result = cursor.execute("select \* from student")

print(result.fetchall())

conn.commit()

conn.close()

Here is the list of available Python database interfaces: [Python Database Interfaces and APIs](http://wiki.python.org/moin/DatabaseInterfaces) .You must download a separate DB API module for each database you need to access. For example, if you need to access an Oracle database as well as a MySQL database, you must download both the Oracle and the MySQL database modules.

The DB API provides a minimal standard for working with databases using Python structures and syntax wherever possible. This API includes the following:

* Importing the API module.
* Acquiring a connection with the database.
* Issuing SQL statements and stored procedures.
* Closing the connection

We would learn all the concepts using MySQL, so let us talk about MySQLdb module.

## **What is MySQLdb?**

MySQLdb is an interface for connecting to a MySQL database server from Python. It implements the Python Database API v2.0 and is built on top of the MySQL C API.

## **How do I Install MySQLdb?**

Before proceeding, you make sure you have MySQLdb installed on your machine. Just type the following in your Python script and execute it:

#!/usr/bin/python

import MySQLdb

If it produces the following result, then it means MySQLdb module is not installed:

Traceback (most recent call last):

File "test.py", line 3, in <module>

import MySQLdb

ImportError: No module named MySQLdb

To install MySQLdb module, download it from [MySQLdb Download](http://sourceforge.net/projects/mysql-python" \t "_blank) page and proceed as follows:

$ gunzip MySQL-python-1.2.2.tar.gz

$ tar -xvf MySQL-python-1.2.2.tar

$ cd MySQL-python-1.2.2

$ python setup.py build

$ python setup.py install

**Note:** Make sure you have root privilege to install above module.

**OR**

<https://dev.mysql.com/downloads/connector/python/1.2.html>

Use **Windows (x86, 32-bit), MSI Installer  
Python 3.5**

## **Database Connection**

Before connecting to a MySQL database, make sure of the followings −

* You have created a database TESTDB.
* You have created a table EMPLOYEE in TESTDB.
* This table has fields FIRST\_NAME, LAST\_NAME, AGE, SEX and INCOME.
* User ID "testuser" and password "test123" are set to access TESTDB.
* Python module MySQLdb is installed properly on your machine.
* You have gone through MySQL tutorial to understand [MySQL Basics.](https://www.tutorialspoint.com/mysql/index.htm)

### Example

Following is the example of connecting with MySQL database "TESTDB"

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using *cursor()* method

cursor = db.cursor()

# execute SQL query using *execute()* method.

cursor.execute("SELECT VERSION()")

# Fetch a single row using *fetchone()* method.

data = cursor.fetchone()

print "Database version : %s " % data

# disconnect from server

db.close()

While running this script, it is producing the following result in my Linux machine.

Database version : 5.0.45

If a connection is established with the datasource, then a Connection Object is returned and saved into **db** for further use, otherwise **db** is set to None. Next,**db** object is used to create a **cursor** object, which in turn is used to execute SQL queries. Finally, before coming out, it ensures that database connection is closed and resources are released.

## **Creating Database Table**

Once a database connection is established, we are ready to create tables or records into the database tables using **execute** method of the created cursor.

### Example

Let us create Database table EMPLOYEE:

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using *cursor()* method

cursor = db.cursor()

# Drop table if it already exist using *execute()* method.

cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")

# Create table as per requirement

sql = """CREATE TABLE EMPLOYEE (

FIRST\_NAME CHAR(20) NOT NULL,

LAST\_NAME CHAR(20),

AGE INT,

SEX CHAR(1),

INCOME FLOAT )"""

cursor.execute(sql)

# disconnect from server

db.close()

## **INSERT Operation**

It is required when you want to create your records into a database table.

### Example

The following example, executes SQL *INSERT* statement to create a record into EMPLOYEE table −

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using *cursor()* method

cursor = db.cursor()

# Prepare SQL query to INSERT a record into the database.

sql = """INSERT INTO EMPLOYEE(FIRST\_NAME,

LAST\_NAME, AGE, SEX, INCOME)

VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""

try:

# Execute the SQL command

cursor.execute(sql)

# Commit your changes in the database

db.commit()

except:

# Rollback in case there is any error

db.rollback()

# disconnect from server

db.close()

Above example can be written as follows to create SQL queries dynamically −

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using *cursor()* method

cursor = db.cursor()

# Prepare SQL query to INSERT a record into the database.

sql = "INSERT INTO EMPLOYEE(FIRST\_NAME, \

LAST\_NAME, AGE, SEX, INCOME) \

VALUES ('%s', '%s', '%d', '%c', '%d' )" % \

('Mac', 'Mohan', 20, 'M', 2000)

try:

# Execute the SQL command

cursor.execute(sql)

# Commit your changes in the database

db.commit()

except:

# Rollback in case there is any error

db.rollback()

# disconnect from server

db.close()

### Example

Following code segment is another form of execution where you can pass parameters directly −

..................................

user\_id = "test123"

password = "password"

con.execute('insert into Login values("%s", "%s")' % \

(user\_id, password))

..................................

## **READ Operation**

READ Operation on any database means to fetch some useful information from the database.

Once our database connection is established, you are ready to make a query into this database. You can use either **fetchone()** method to fetch single record or **fetchall()** method to fetech multiple values from a database table.

* **fetchone():** It fetches the next row of a query result set. A result set is an object that is returned when a cursor object is used to query a table.
* **fetchall():** It fetches all the rows in a result set. If some rows have already been extracted from the result set, then it retrieves the remaining rows from the result set.
* **rowcount:** This is a read-only attribute and returns the number of rows that were affected by an execute() method.

### Example

The following procedure queries all the records from EMPLOYEE table having salary more than 1000 −

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using *cursor()* method

cursor = db.cursor()

# Prepare SQL query to INSERT a record into the database.

sql = "SELECT \* FROM EMPLOYEE \

WHERE INCOME > '%d'" % (1000)

try:

# Execute the SQL command

cursor.execute(sql)

# Fetch all the rows in a list of lists.

results = cursor.fetchall()

for row in results:

fname = row[0]

lname = row[1]

age = row[2]

sex = row[3]

income = row[4]

# Now print fetched result

print "fname=%s,lname=%s,age=%d,sex=%s,income=%d" % \

(fname, lname, age, sex, income )

except:

print "Error: unable to fecth data"

# disconnect from server

db.close()

This will produce the following result −

fname=Mac, lname=Mohan, age=20, sex=M, income=2000

## **Update Operation**

UPDATE Operation on any database means to update one or more records, which are already available in the database.

The following procedure updates all the records having SEX as **'M'**. Here, we increase AGE of all the males by one year.

### Example

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using *cursor()* method

cursor = db.cursor()

# Prepare SQL query to UPDATE required records

sql = "UPDATE EMPLOYEE SET AGE = AGE + 1

WHERE SEX = '%c'" % ('M')

try:

# Execute the SQL command

cursor.execute(sql)

# Commit your changes in the database

db.commit()

except:

# Rollback in case there is any error

db.rollback()

# disconnect from server

db.close()

## **DELETE Operation**

DELETE operation is required when you want to delete some records from your database. Following is the procedure to delete all the records from EMPLOYEE where AGE is more than 20 −

### Example

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using *cursor()* method

cursor = db.cursor()

# Prepare SQL query to DELETE required records

sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)

try:

# Execute the SQL command

cursor.execute(sql)

# Commit your changes in the database

db.commit()

except:

# Rollback in case there is any error

db.rollback()

# disconnect from server

db.close()

Python XML Processing

## **What is XML?**

The Extensible Markup Language (XML) is a markup language much like HTML or SGML. This is recommended by the World Wide Web Consortium and available as an open standard.

XML is extremely useful for keeping track of small to medium amounts of data without requiring a SQL-based backbone.

The Python standard library provides a minimal but useful set of interfaces to work with XML.

The two most basic and broadly used APIs to XML data are the SAX and DOM interfaces.

* **Simple API for XML (SAX) :**Here, you register callbacks for events of interest and then let the parser proceed through the document. This is useful when your documents are large or you have memory limitations, it parses the file as it reads it from disk and the entire file is never stored in memory.
* **Document Object Model (DOM) API :**This is a World Wide Web Consortium recommendation wherein the entire file is read into memory and stored in a hierarchical (tree-based) form to represent all the features of an XML document.

## **XML Parser Architectures and APIs**

The Python standard library provides a minimal but useful set of interfaces to work with XML.

The two most basic and broadly used APIs to XML data are the SAX and DOM interfaces.

* **Simple API for XML (SAX) :**Here, you register callbacks for events of interest and then let the parser proceed through the document. This is useful when your documents are large or you have memory limitations, it parses the file as it reads it from disk and the entire file is never stored in memory.
* **Document Object Model (DOM) API :**This is a World Wide Web Consortium recommendation wherein the entire file is read into memory and stored in a hierarchical (tree-based) form to represent all the features of an XML document.

SAX obviously cannot process information as fast as DOM can when working with large files. On the other hand, using DOM exclusively can really kill your resources, especially if used on a lot of small files.

*movies.xml*

<collection shelf="New Arrivals">

<movie title="Enemy Behind">

<type>War, Thriller</type>

<format>DVD</format>

<year>2003</year>

<rating>PG</rating>

<stars>10</stars>

<description>Talk about a US-Japan war</description>

</movie>

<movie title="Transformers">

<type>Anime, Science Fiction</type>

<format>DVD</format>

<year>1989</year>

<rating>R</rating>

<stars>8</stars>

<description>A schientific fiction</description>

</movie>

<movie title="Trigun">

<type>Anime, Action</type>

<format>DVD</format>

<episodes>4</episodes>

<rating>PG</rating>

<stars>10</stars>

<description>Vash the Stampede!</description>

</movie>

<movie title="Ishtar">

<type>Comedy</type>

<format>VHS</format>

<rating>PG</rating>

<stars>2</stars>

<description>Viewable boredom</description>

</movie>

</collection>

## **Parsing XML with SAX APIs**

SAX is a standard interface for **event-driven XML parsing**. Parsing XML with SAX generally requires you to create your own **ContentHandler by subclassing xml.sax.ContentHandler.**

Your *ContentHandler* handles the particular tags and attributes of your flavor(s) of XML. A ContentHandler object provides methods to handle various parsing events. Its owning parser calls ContentHandler methods as it parses the XML file.

The methods ***startDocument***and ***endDocument*** are called at the start and the end of the XML file. The method *characters(text)* is passed character data of the XML file via the parameter text.

The ContentHandler is called at the start and end of each element. If the parser is not in namespace mode, the methods ***startElement(tag, attributes)* and*endElement(tag)*** are called; otherwise, the corresponding methods*startElementNS* and *endElementNS* are called. Here, tag is the element tag, and attributes is an Attributes object.

Here are other important methods to understand before proceeding –

## ***make\_parser* Method**

Following method creates a new parser object and returns it. The parser object created will be of the first parser type the system finds.

xml.sax.make\_parser( [parser\_list] )

Here is the detail of the parameters:

* **parser\_list:**The optional argument consisting of a list of parsers to use which must all implement the make\_parser method.

## **The *parse* Method**

Following method creates a SAX parser and uses it to parse a document.

xml.sax.parse( xmlfile, contenthandler[, errorhandler])

Here is the detail of the parameters −

* **xmlfile:**This is the name of the XML file to read from.
* **contenthandler:**This must be a ContentHandler object.
* **errorhandler:** If specified, errorhandler must be a SAX ErrorHandler object.

## **The *parseString* Method**

There is one more method to create a SAX parser and to parse the specified**XML string**.

xml.sax.parseString(xmlstring, contenthandler[, errorhandler])

Here is the detail of the parameters −

* **xmlstring:**This is the name of the XML string to read from.
* **contenthandler:**This must be a ContentHandler object.
* **errorhandler:** If specified, errorhandler must be a SAX ErrorHandler object.

### Example

#!/usr/bin/python

import xml.sax

class MovieHandler( xml.sax.ContentHandler ):

def \_\_init\_\_(self):

self.CurrentData = ""

self.type = ""

self.format = ""

self.year = ""

self.rating = ""

self.stars = ""

self.description = ""

# Call when an element starts

def startElement(self, tag, attributes):

self.CurrentData = tag

if tag == "movie":

print "\*\*\*\*\*Movie\*\*\*\*\*"

title = attributes["title"]

print "Title:", title

# Call when an elements ends

def endElement(self, tag):

if self.CurrentData == "type":

print "Type:", self.type

elif self.CurrentData == "format":

print "Format:", self.format

elif self.CurrentData == "year":

print "Year:", self.year

elif self.CurrentData == "rating":

print "Rating:", self.rating

elif self.CurrentData == "stars":

print "Stars:", self.stars

elif self.CurrentData == "description":

print "Description:", self.description

self.CurrentData = ""

# Call when a character is read

def characters(self, content):

if self.CurrentData == "type":

self.type = content

elif self.CurrentData == "format":

self.format = content

elif self.CurrentData == "year":

self.year = content

elif self.CurrentData == "rating":

self.rating = content

elif self.CurrentData == "stars":

self.stars = content

elif self.CurrentData == "description":

self.description = content

if ( \_\_name\_\_ == "\_\_main\_\_"):

# create an XMLReader

parser = xml.sax.make\_parser()

# turn off namepsaces

parser.setFeature(xml.sax.handler.feature\_namespaces, 0)

# override the default ContextHandler

Handler = MovieHandler()

parser.setContentHandler( Handler )

parser.parse("movies.xml")

For a complete detail on SAX API documentation, please refer to standard[Python SAX APIs](http://docs.python.org/library/xml.sax.html).

This would produce following result −

\*\*\*\*\*Movie\*\*\*\*\*

Title: Enemy Behind

Type: War, Thriller

Format: DVD

Year: 2003

Rating: PG

Stars: 10

Description: Talk about a US-Japan war

\*\*\*\*\*Movie\*\*\*\*\*

Title: Transformers

Type: Anime, Science Fiction

Format: DVD

Year: 1989

Rating: R

Stars: 8

Description: A schientific fiction

\*\*\*\*\*Movie\*\*\*\*\*

Title: Trigun

Type: Anime, Action

Format: DVD

Rating: PG

Stars: 10

Description: Vash the Stampede!

\*\*\*\*\*Movie\*\*\*\*\*

Title: Ishtar

Type: Comedy

Format: VHS

Rating: PG

Stars: 2

Description: Viewable boredom

## **Parsing XML with DOM APIs**

The Document Object Model ("DOM") is a cross-language API from the World Wide Web Consortium (W3C) for accessing and modifying XML documents.

The DOM is extremely useful for random-access applications. SAX only allows you a view of one bit of the document at a time. If you are looking at one SAX element, you have no access to another.

Here is the easiest way to quickly load an XML document and to create a minidom object using the xml.dom module. The minidom object provides a simple parser method that quickly creates a DOM tree from the XML file.

The sample phrase calls the parse( file [,parser] ) function of the minidom object to parse the XML file designated by file into a DOM tree object.

### Example

#!/usr/bin/python

from xml.dom.minidom import parse

import xml.dom.minidom

# Open XML document using minidom parser

DOMTree = xml.dom.minidom.parse("movies.xml")

collection = DOMTree.documentElement

if collection.hasAttribute("shelf"):

print "Root element : %s" % collection.getAttribute("shelf")

# Get all the movies in the collection

movies = collection.getElementsByTagName("movie")

# Print detail of each movie.

for movie in movies:

print "\*\*\*\*\*Movie\*\*\*\*\*"

if movie.hasAttribute("title"):

print "Title: %s" % movie.getAttribute("title")

type = movie.getElementsByTagName('type')[0]

print "Type: %s" % type.childNodes[0].data

format = movie.getElementsByTagName('format')[0]

print "Format: %s" % format.childNodes[0].data

rating = movie.getElementsByTagName('rating')[0]

print "Rating: %s" % rating.childNodes[0].data

description = movie.getElementsByTagName('description')[0]

print "Description: %s" % description.childNodes[0].data

This would produce the following result −

Root element : New Arrivals

\*\*\*\*\*Movie\*\*\*\*\*

Title: Enemy Behind

Type: War, Thriller

Format: DVD

Rating: PG

Description: Talk about a US-Japan war

\*\*\*\*\*Movie\*\*\*\*\*

Title: Transformers

Type: Anime, Science Fiction

Format: DVD

Rating: R

Description: A schientific fiction

\*\*\*\*\*Movie\*\*\*\*\*

Title: Trigun

Type: Anime, Action

Format: DVD

Rating: PG

Description: Vash the Stampede!

\*\*\*\*\*Movie\*\*\*\*\*

Title: Ishtar

Type: Comedy

Format: VHS

Rating: PG

Description: Viewable boredom

For a complete detail on DOM API documentation, please refer to standard[Python DOM APIs](http://docs.python.org/library/xml.dom.html).

# **Python CGI Programming**

\*\*\*Please execute all codes given in Apache folder on allocated VM.

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. The CGI specs are currently maintained by the NCSA.

## **What is CGI?**

* The Common Gateway Interface, or CGI, 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**

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**



## **Web Server Support and Configuration**

Before you proceed with CGI Programming, 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. By convention, CGI files have extension as.**cgi,** but you can keep your files with python extension **.py** as well.

By default, the Linux server is configured to run only the scripts in the cgi-bin directory in /var/www.

Here, we assume that you have Web Server up and running successfully and you are able to run any other CGI program like Perl or Shell, etc.

## **First CGI Program**

Here is a simple link, which is linked to a CGI script called [hello.py](https://www.tutorialspoint.com/cgi-bin/hello.py). This file is kept in /var/www/cgi-bin directory and it has following content. Before running your CGI program, make sure you have change mode of file using**chmod 755 hello.py** UNIX command to make file executable.

#!/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, or access it as <http://localhost/cgi-bin/Hello.py> in the browser, then this produces the following output −

|  |
| --- |
| **Hello Word! This is my first CGI program** |

**You can write a html file to access this CGI scipt-**

**<!-- put it in /htdocs dir-->**

**<html>**

**<body>**

**<form method="get" action="/cgi-bin/Hello.py">**

**<input type = "submit" name="b1" value ="Click Me">**

**</form>**

**</body>**

**</html>**

This hello.py script is a simple Python script, which writes its output on STDOUT file, i.e., screen. 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 it specifies the content type to be displayed on the browser screen.

By now you must have understood basic concept of CGI and you can write many complicated CGI programs using Python. This script can interact with any other external system also to exchange information such as RDBMS.

## **HTTP Header**

The line **Content-type:text/html\r\n\r\n** is part of HTTP header which is sent to the browser to understand the content. 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

## **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 limtation: 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](https://www.tutorialspoint.com/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 –

**hello\_get.py**

#!/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>

## **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.

Top of Form

First Name: 

Last Name:  

Bottom of Form

## **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.

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>

## **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:

Top of Form

 Maths  Physics 

Bottom of Form

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

#!/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>"

## **Passing Radio Button Data to CGI Program**

Radio Buttons are used when only one option is required to be selected.

Here is example HTML code for a form with two radio buttons −

<form action="/cgi-bin/radiobutton.py" method="post" target="\_blank">

<input type="radio" name="subject" value="maths" /> Maths

<input type="radio" name="subject" value="physics" /> Physics

<input type="submit" value="Select Subject" />

</form>

The result of this code is the following form −

Top of Form

 Maths  Physics 

Bottom of Form

Below is radiobutton.py script to handle input given by web browser for radio button:

#!/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('subject'):

subject = form.getvalue('subject')

else:

subject = "Not set"

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

print "<html>"

print "<head>"

print "<title>Radio - Fourth CGI Program</title>"

print "</head>"

print "<body>"

print "<h2> Selected Subject is %s</h2>" % subject

print "</body>"

print "</html>"

## **Passing Text Area Data to CGI Program**

TEXTAREA element is used when multiline text has to be passed to the CGI Program.

Here is example HTML code for a form with a TEXTAREA box −

<form action="/cgi-bin/textarea.py" method="post" target="\_blank">

<textarea name="textcontent" cols="40" rows="4">

Type your text here...

</textarea>

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

</form>

The result of this code is the following form −

Top of Form

 

Bottom of Form

Below is textarea.cgi script to handle input given by web browser −

#!/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('textcontent'):

text\_content = form.getvalue('textcontent')

else:

text\_content = "Not entered"

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

print "<html>"

print "<head>";

print "<title>Text Area - Fifth CGI Program</title>"

print "</head>"

print "<body>"

print "<h2> Entered Text Content is %s</h2>" % text\_content

print "</body>"

## **Passing Drop Down Box Data to CGI Program**

Drop Down Box is used when we have many options available but only one or two will be selected.

Here is example HTML code for a form with one drop down box −

<form action="/cgi-bin/dropdown.py" method="post" target="\_blank">

<select name="dropdown">

<option value="Maths" selected>Maths</option>

<option value="Physics">Physics</option>

</select>

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

</form>

The result of this code is the following form −

Top of Form

 

Bottom of Form

Below is dropdown.py script to handle input given by web browser.

#!/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('dropdown'):

subject = form.getvalue('dropdown')

else:

subject = "Not entered"

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

print "<html>"

print "<head>"

print "<title>Dropdown Box - Sixth CGI Program</title>"

print "</head>"

print "<body>"

print "<h2> Selected Subject is %s</h2>" % subject

print "</body>"

print "</html>"

## **Using Cookies in CGI**

HTTP protocol is a stateless protocol. 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?**

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:

* **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 the cookies is 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"

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....

From this example, you must have understood how to set cookies. 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 sending magic 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....

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

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

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 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 −

Top of Form

File:



Bottom of Form

Above example has been disabled intentionally to save people uploading file on our server, but you can try above code with your server.

Here is the script **save\_file.py** to handle file upload −

#!/usr/bin/python

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,)

If you run the above script on Unix/Linux, then you need 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("\\", "/" ))

**Apache Server:**

**Please install Apache server from :** [**http://archive.apache.org/dist/httpd/binaries/win32/httpd-2.2.16-win32-x86-no\_ssl.msi**](http://archive.apache.org/dist/httpd/binaries/win32/httpd-2.2.16-win32-x86-no_ssl.msi)

**\*\*\*\*Eclipse Settings**

1. **To add python development plugin**

**windows🡪preferences**

**or help->install new software**

**click on ADD**

**window : Name : enter as Python 2**

**search pydev download in google**

**put this location** [**http://pydev.org/updates**](http://pydev.org/updates)

[http://update-production-pydev.s3.amazonaws.com/pydev/updates](http://update-production-pydev.s3.amazonaws.com/pydev/updates/site.xml)

**More Info at:** **http://www.pydev.org/download.html**

**2)**

**then preferences , windows option-preferences**

**interpreter**

**selete c:\python\python.exe**

**Assignments to do:**

1. Given temperatures in a list like-

Celsius = [39.2, 36.5, 37.3, 37.8]

1. Convert all these temperature values into Fahrenheit and store in a target list – Fahrenheit =[]

Use map and lambda. Hint : formula - (float(9)/5)\*x + 32

b) Convert above Fahrenheit list into Celsius list.

Use map and lambda. Hint : formula - (float(5)/9)\*(x-32),

1. Define a list as below reduce 🡨

List1 = [47,11,42,102,13]

Find maximum number

3. Let’s say I give you a list saved in a variable: a = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100].

Write one line of Python that takes this list a and makes a new list b

that has only the even elements of this list in it. (Use filter and lambda)

4.

* Write Python script for x`database connectivity using sqlite3 or MySQLdb module.Create a table from the script, please drop it if it already exists.

CREATE TABLE users(id INTEGER PRIMARY KEY, name TEXT,

phone TEXT, email TEXT unique, password TEXT)

Accept some values for varaibles-

name1 ,phone1 ,email1, password1 , e.g.

name1 = 'Ravi Verma'

phone1 = '9923849335'

email1 = 'ravi\_verma@gmail.com'

password1 = 'tough@password!!'

* Insert these values in table users.
* Read the data back and display
* Accept name from user, display the details for that user if it exists in table users.

5. Read Employee database table from mysql and instantiate the Employee object. Display all employee details (Name, empID, Age, Sal).

Use Employee class

6. Read given XML file movies.xml. Print the total count of movie details stored in it. Also display all Movie details

Create a folder: Assignments \Day2 \ Q1.py, Q2.py, Q3.py