What is CGI in Python?

CGI stands for **Common Gateway Interface** in Python which is a set of standards that explains how information or data is exchanged between the web server and a routine script.

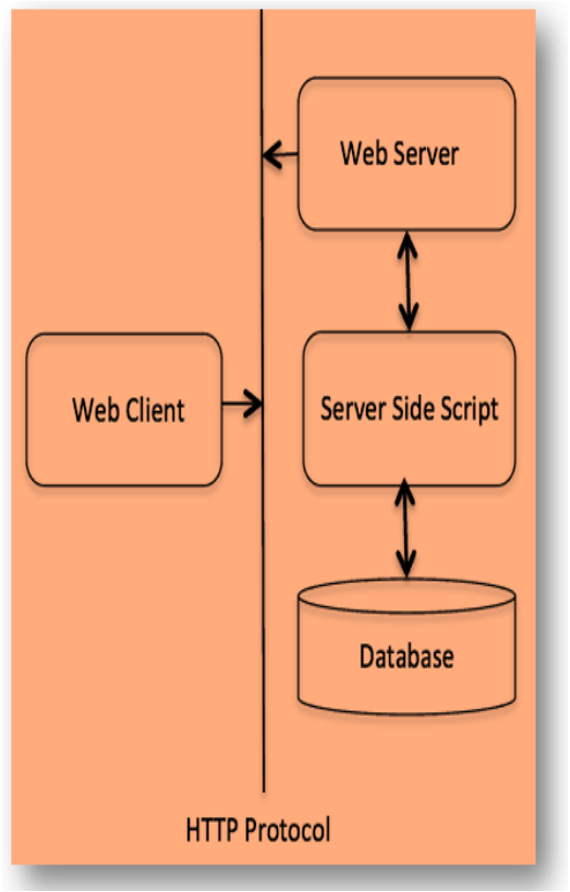
This interface is used by web servers to route information requests supplied by a browser or we can say that CGI is customary for external gateway programs to interface with information servers such as HTTP servers.

A CGI script is invoked by an HTTP server, usually to course user input which is submitted through an HTML <FORM> or an <ISINDEX> element.

Concept of CGI

Whenever we click on a hyperlink to browse a particular web page or URL, our browser interacts with the HTTP web server and asks for the same URL (or filename). Web Server then parses the URL and looks for the same filename. If that file is found, then that file is sent back to the browser, otherwise, an error message is sent indicating that we are demanding the wrong file. Web browser takes the response from a web server and displays it, then whether it is the received file from the webserver or an error message. But, conversely, it is possible to set up the HTTP server so that whenever a specific file is requested, then that file is not sent back, but instead, it is executed as a program, and whatever that program output is, that is sent back to our browser for display. This same function is called the **Common Gateway Interface (or CGI)** and the programs which are executed are called CGI scripts. In python, these CGI programs are Python Script.

Architecture of CGI:




```
#!/usr/bin/python

# Import CGI and CGIT module

import cgi, cgitb

# to create instance of FieldStorage

# class which we can use to work

# with the submitted form data

form = cgi.FieldStorage()

your_name = form.getvalue('your_name')

# to get the data from fields

comapny_name =
form.getvalue('company_name')

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

print ("<html>")

print ("<head>")

print ("<title>First CGI Program</title>")

print ("</head>")
```

```
print ("<body>")
print ("<h2>Hello, %s is working in %s</h2>"
        % (your_name, company_name))
print ("</body>")
print ("</html>")
```

Web Surfing with Python: Creating Simple Web ClientsOne thing to keep in mind is that a browser is only one type of Web client. Anyapplication that makes a request for data from a Web server is considered a "client."Yes, it is possible to create other clients that retrieve documents or data off the Internet.

One important reason to do this is that a browser provides only limited capacity, i.e., it isused primarily for viewing and interacting with Web sites. A client program, on the otherhand, has the ability to do moreit can not only download data, but it can also store it,manipulate it, or perhaps even transmit it to another location or application.

Applications that use theurllibmodule to download or access information on the Web[using eitherurllib.urlopen()orurllib.urlretrieve()] can be considered a simpleWeb client. All you need to do is provide a valid Web address.20.2.1. Uniform Resource LocatorsSimple Web surfing involves using Web addresses called URLs (Uniform ResourceLocators). Such addresses are used to locate a document on the Web or to call a CGIprogram to generate a document for your client. URLs are part of a larger set ofidentifiers known as URIs (Uniform Resource Identifiers).

This superset was created inanticipation of other naming conventions that have yet to be developed. A URL is simplya URI which uses an existing protocol or scheme (i.e., http, ftp, etc.) as part of itsaddressing. To complete this picture, we'll add that non-URL URIs are sometimes knownas URNs (Uniform Resource Names), but because URLs are the only URIs in use today,you really don't hear much about URIs or URNs, save perhaps as XML identifiers.Like street addresses, Web addresses have some structure.

An American street addressusually is of the form "number street designation," i.e., 123 Main Street. It differs fromother countries, which have their own rules. A URL uses the format:prot_sch://net_loc/path;params?query#frag

Advanced Web Clients

Web browsers are basic Web clients. They are used primarily for searching and downloading documents from the Web. Advanced clients of the Web are those applications which do more than download single documents from the Internet.

One example of an advanced Web client is a crawler (a.k.a. spider, robot). These are programs which explore and download pages from the Internet for different reasons, some of which include:

- Indexing or cataloging into a large search engine such as Google, Alta Vista, or Yahoo!,
- Offline browsing—downloading documents onto a local hard disk and rearranging hyperlinks to create almost a mirror image for local browsing,
- Downloading and storing for historical or archival purposes, or
- Web page caching

- Web browsers are basic Web clients. They are used primarily for searching and downloading documents from the Web. Advanced clients of the Web are those applications that do more than download single documents from the Internet.
- One example of an advanced Web client is a *crawler* (aka *spider*, *robot*). These are programs that explore and download pages from the Internet for different reasons, some of which include:
- Indexing into a large search engine such as Google or Yahoo!
- Offline browsing—downloading documents onto a local hard disk and rearranging hyperlinks to create almost a mirror image for local browsing
- Downloading and storing for historical or archival purposes, or
- Web page caching to save superfluous downloading time on Web site revisits.
- The crawler we present below, crawl.py, takes a starting Web address (URL), downloads that page and all other pages whose links appear in succeeding pages, but only those that are in the same domain as the starting page. Without such limitations, you will run out of disk space!

Helping Web Servers Process Client Data

1. Introduction to CGI

The Web was initially developed to be a global online repository or archive of documents (mostly educational and research-oriented). Such pieces of information generally come in the form of static text and usually in HTML.

HTML is not as much a language as it is a text formatter, indicating changes in font types, sizes, and styles. The main feature of HTML is in its hypertext capability. This refers to the ability to designate certain text (usually highlighted in some fashion) or even graphic elements as *links* that point to other “documents” or locations on the Internet and Web that are related in context to the original. Such a document can be accessed by a simple mouse click or other user selection mechanism. These (static) HTML documents live on the Web server and are sent to clients when requested.

As the Internet and Web services evolved, there grew a need to process user input. Online retailers needed to be able to take individual orders, and online banks and search engine portals needed to create accounts for individual users. Thus fill-out forms were invented; they were the only way a Web site could get specific information from users (until Java applets came along). This, in turn, required that the HTML be generated on the fly, for each client submitting user-specific data.

But, Web servers are only really good at one thing: getting a user request for a file and returning that file (i.e., an HTML file) to the client. They do not have the “brains” to be able to deal with user-specific data such as those which come from fields. Given this is not their responsibility, Web servers farm out such requests to external applications which create the dynamically generated HTML that is returned to the client.

The entire process begins when the Web server receives a client request (i.e., GET or POST) 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, which then (finally) forwards it back to the user. This process of the server receiving a form, contacting an external application, and receiving and returning the HTML takes place through the CGI. An overview of how CGI works is presented in Figure 10-1, which shows you the execution and data flow, step-by-step, from when a user submits a form until the resulting Web page is returned.

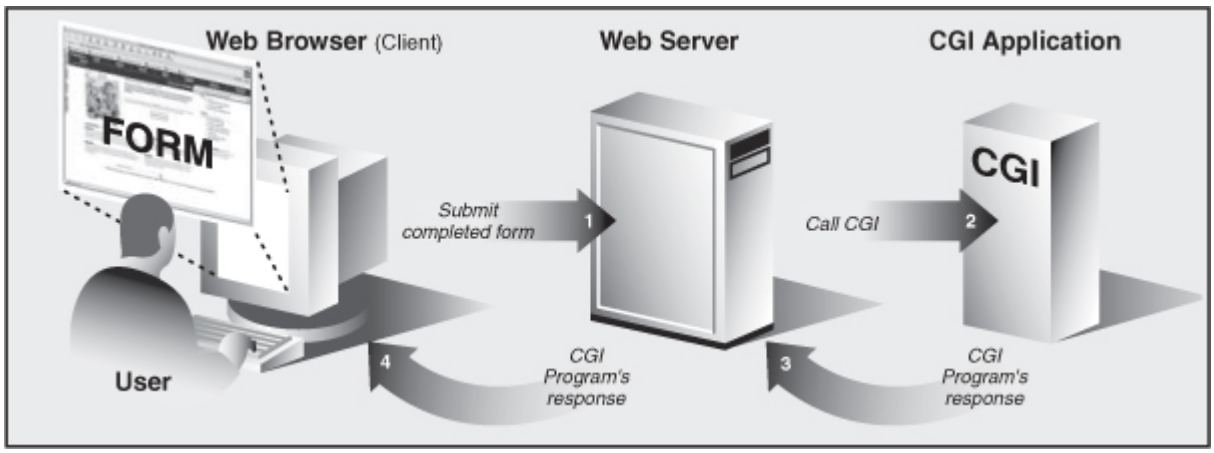


Figure 10-1. Overview of how CGI works. CGI represents the interaction between a Web server and the application that is required to process a user’s form and generate the dynamic HTML that is eventually returned.

Forms input on the client and sent to a Web server can include processing and perhaps some form of storage in a back-end database. Just keep in mind that any time a Web page contains items that require user input (text fields, radio buttons, etc.) and/or a Submit button or image, it most likely involves some sort of CGI activity.

CGI applications that create the HTML are usually written in one of many higher-level programming languages that have the ability to accept user data, process it, and then return HTML back to the server. Before we take a look at CGI, we have to issue the caveat that

Furthermore, there are a good number of Web application development frameworks out there as well as content management systems, all of which make building CGI a relic of past. However, beneath all the fluff and abstraction, they must still, in the end, follow the same model that CGI originally provided, and that is being able to take user input, execute code based on that input, and then provide valid HTML as its final output for the client. Therefore, the exercise in learning CGI is well worth it in terms of understanding the fundamentals required to develop effective Web services.

In this next section, we will look at how to create CGI applications in Python, with the help of the `cgi` module.

2. CGI Applications

A CGI application is slightly different from a typical program. The primary differences are in the input, output, and user interaction aspects of a computer program. When a CGI script starts, it needs to retrieve the user-supplied form data, but it has to obtain this data from the

Web client, not a user on the server computer or a disk file. This is usually known as *the request*.

The output differs in that any data sent to standard output will be sent back to the connected Web client rather than to the screen, GUI window, or disk file. This is known as *the response*. The data sent back must be a set of valid headers followed by HTML-tagged data. If it is not and the Web client is a browser, an error (specifically, an Internal Server Error) will occur because Web clients understand only valid HTTP data (i.e., MIME headers and HTML).

Finally, as you can probably guess, there is no user interaction with the script. All communication occurs among the Web client (on behalf of a user), the Web server, and the CGI application.

3. The cgi Module

There is one primary class in the `cgi` module that does all the work: the `FieldStorage` class. This class reads in all the pertinent user information from the Web client (via the Web server); thus, it should be instantiated when a Python CGI script begins. Once it has been instantiated, it will consist of a dictionary-like object that contains a set of key-value pairs. The keys are the names of the input items that were passed in via the form. The values contain the corresponding data.

Values can be one of three objects. The first are `FieldStorage` objects (instances). The second are instances of a similar class called `MiniField Storage`, which is used in cases for which no file uploads or multiple-part form data is involved. `MiniFieldStorage` instances contain only the key-value pair of the name and the data. Lastly, they can be a list of such objects. This occurs when a form contains more than one input item with the same field name.

For simple Web forms, you will usually find all `MiniFieldStorage` instances. All of our examples that follow pertain only to this general case.

4. The cgitb Module

As we mentioned earlier, a valid response back to the Web server (which would then forward it to the user/browser) must contain valid HTTP headers and HTML-tagged data. Have you thought about the returned data if your CGI application crashes? What happens when you run a Python script that results in an error? That's right: a traceback occurs. Would the text of a traceback be considered as valid HTTP headers or HTML? No.

A Web server receiving a response it doesn't understand will just throw up its hands and give up, returning a "500 error." The 500 is an HTTP response code that means an internal Web server error has occurred, most likely from the application that is being executed. The output on the browser doesn't aid the developer either, as the screen is either blank or shows "Internal Server Error," or something similar.

When our Python programs were running on the command-line or in an *integrated development environment* (IDE), errors resulted in a traceback, upon which we could take action. Not so in the browser. What we really want is to see the Web application's traceback on the browser screen, not "Internal Server Error." This is where the `cgitb` module comes in.

To enable a dump of tracebacks, all we need to do is to insert the following import and call in our CGI applications:

```
import cgitb
```

cgitb.enable()

You'll have plenty of opportunity as we explore CGI for the first half of this chapter. For now, just leave these two lines out as we undertake some simple examples. First, I want you to see the "Internal Server Error" messages and debug them the hard way. Once you realize how the server's not throwing you a bone, you'll add these two lines religiously, on your own.

Web (HTTP) Servers

If Google Chrome, Mozilla Firefox, Microsoft Internet Explorer, and Opera are among the most popular Web clients, then what are the most common Web servers? They are Apache, lighHTTPD, Microsoft IIS, LiteSpeed Technologies LiteSpeed, and ACME Laboratories thttpd. For situations in which these servers might be overkill for your desired application, Python can be used to create simple yet useful Web servers.

Note that although these servers are simplistic and not meant for production, they can be very useful in providing development servers for your users. Both the Django and Google App Engine development servers are based on the `BaseHTTPServer` module described in the next section.

`client_address`
: Contains a tuple of the form `(host, port)` referring to the client's address.

`server`
: Contains the server instance.

`close_connection`
: Boolean that should be set before `handle_one_request()` returns, indicating if another request may be expected, or if the connection should be shut down.

`path`
: Contains the request path. If query component of the URL is present, then `path` includes the query. Using the terminology of **RFC 3986**, `path` here includes `hier-part` and the `query`.

`headers`
: Holds an instance of the class specified by the `MessageClass` class variable. This instance parses and manages the headers in the HTTP request. The `parse_headers()` function from `http.client` is used to parse the headers and it requires that the HTTP request provide a valid **RFC 2822** style header.

`protocol_version`
: Specifies the HTTP version to which the server is conformant. It is sent in responses to let the client know the server's communication capabilities for future requests. If set to `'HTTP/1.1'`, the server will permit HTTP persistent connections; however, your server *must* then include an accurate `Content-Length` header (using `send_header()`) in all of its responses to clients. For backwards compatibility, the setting defaults to `'HTTP/1.0'`.

1. Simple Web Servers in Python

The base code needed is already available in the Python standard library—you just need to customize it for your needs. To create a Web server, a base server and a *handler* are required.

The base Web server is a boilerplate item—a must-have. Its role is to perform the necessary HTTP communication between client and server. The base server class is (appropriately) named `HTTPServer` and is found in the `BaseHTTPServer` module.

The handler is the piece of software that does the majority of the Web serving. It processes the client request and returns the appropriate file, whether static or dynamically generated. The complexity of the handler determines the complexity of your Web server. The Python Standard Library provides three different handlers.

The most basic, plain, vanilla handler, `BaseHTTPRequestHandler`, is found in the `BaseHTTPServer` module, along with the base Web server. Other than taking a client request, no other handling is implemented at all, so you have to do it all yourself, such as in our `myhttpd.py` server coming up.

The `SimpleHTTPRequestHandler`, available in the `SimpleHTTP-Server` module, builds on `BaseHTTPRequestHandler` by implementing the standard GET and HEAD requests in a fairly straightforward manner. Still nothing sexy, but it gets the simple jobs done.

Finally, we have the `CGIHTTPRequestHandler`, available in the `CGIHTTPServer` module, which takes the `SimpleHTTPRequestHandler` and adds support for POST requests. It has the ability to call common gateway interface (CGI) scripts to perform the requested processing and can send the generated HTML back to the client. In this chapter, we're only going to explore a CGI-processing server; the next chapter will describe to you why CGI is no longer the way the world of the Web works, but you still need to know the concepts.

3.x

To simplify the user experience, consistency, and code maintenance, these modules (actually their classes) have been combined into a single module named `server.py` and installed as part of the `http` package in Python 3. (Similarly, the Python 2 `httplib` [HTTP client] module has been renamed to `http.client` in Python 3.) The three modules, their classes, and the Python 3 `http.server` umbrella package are summarized in Table 9-6.

Table 9-6. Web Server Modules and Classes

Module	Description
`BaseHTTPServer`[a]	Provides the base Web server and base handler classes, `HTTPServer` and `BaseHTTPRequestHandler`, respectively
`SimpleHTTPServer`[a]	Contains the `SimpleHTTPRequestHandler` class to perform GET and HEAD requests
`CGIHTTPServer`[a]	Contains the `CGIHTTPRequestHandler` class to process POST requests and perform CGI execution
`http.server`[b]	All three Python 2 modules and classes above combined into a single Python 3 package.

a. Removed in Python 3.0.
b. New in Python 3.0.

More Power, Less Code: A Simple CGI Web Server

The previous example is also weak in that it cannot process CGI requests. `BaseHTTPServer` is as basic as it gets. One step higher, we have the `SimpleHTTPServer`. It provides the `do_HEAD()` and `do_GET()` methods on your behalf, so you don't have to create either, such as we did with the `BaseHTTPServer`.

The highest-level (take that with a grain of salt) server provided in the standard library is `CGIHTTPServer`. In addition to `do_HEAD()` and `do_GET()`, it defines `do_POST()`, with which you can process form data. Because of these amenities, a CGI-capable development

server can be created with just two real lines of code (so short we're not even bothering making it a code example in this chapter, because you can just recreate it by typing it up on your computer now):

Click here to view code image

```
#!/usr/bin/env python
import CGIHTTPServer
CGIHTTPServer.test()
```

Note that we left off the check to quit the server by using Ctrl+C and other fancy output, taking whatever the `CGIHTTPServer.test()` function gives us, which is a lot. You start the server by just invoking it from your shell. Below is an example of running this code on a PC—it's quite similar to what you'll experience on a POSIX machine:

Click here to view code image

```
C:\py>python cgihttpd.py
Serving HTTP on 0.0.0.0 port 8000 ...
```

It starts a server by default on port 8000 (but you can change that at runtime by providing a port number as a command-line argument:

Click here to view code image

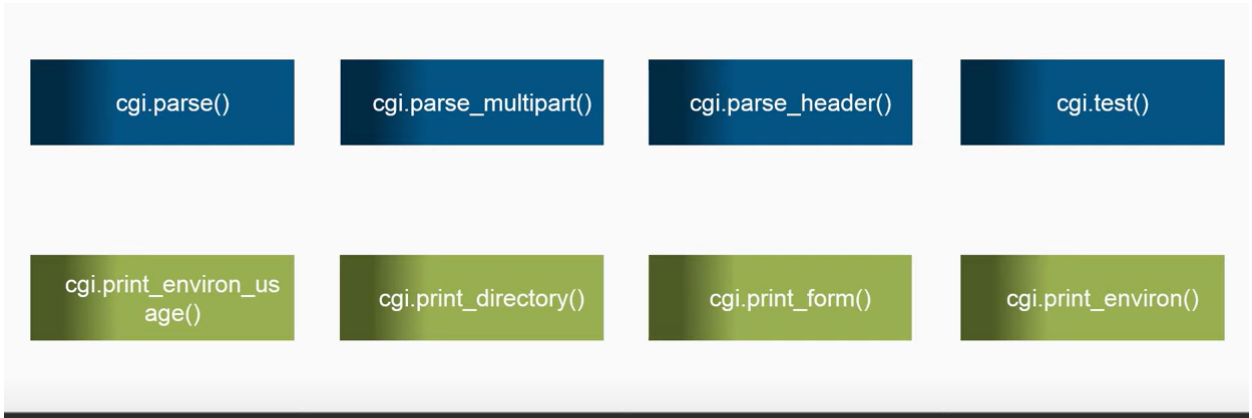
```
C:\py\>python cgihttpd.py 8080
Serving HTTP on 0.0.0.0 port 8080 ...
```

To test it out, just make sure that a `cgi-bin` folder exists (with some CGI Python scripts) at the same level as the script. There's no point in setting up Apache, setting CGI handler prefixes, and all that extra stuff when you just want to test a simple script. We will show you how to write CGI scripts in Chapter 10, "Web Programming: CGI and WSGI," as well as tell you why you should avoid doing so.

As you can see, it does not take much to have a Web server up and running in pure Python. Again, you should not be writing servers all the time. Generally, you're creating Web applications that run on Web servers. These server modules are meant only to create servers that are useful during development, regardless of whether you develop applications or Web frameworks.

In production, your live service will instead be using servers that are production-worthy such as Apache, ligHTTPD, or any of the others listed at the beginning of this section. However, we hope this section will have enlightened you such that you realize doing complex tasks can be simplified with the power that Python gives you.

STEP BY STEP PROCEDURE

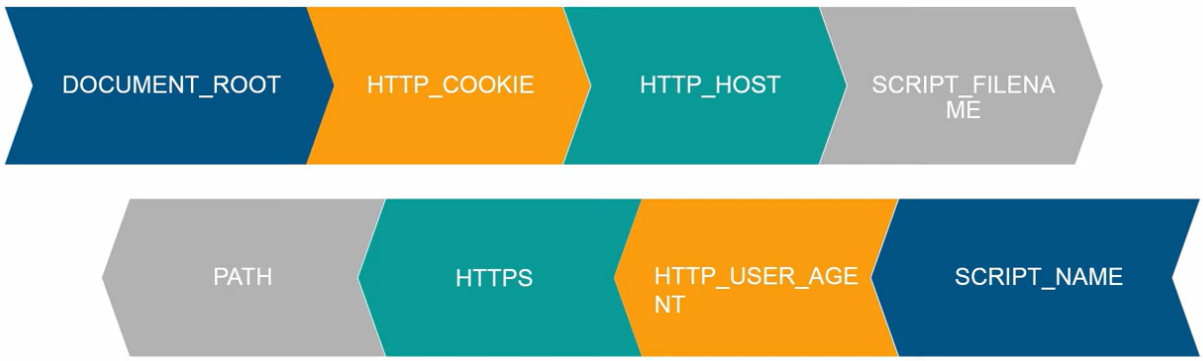


CGI Functions

The above are a few function that we have in CGI programming

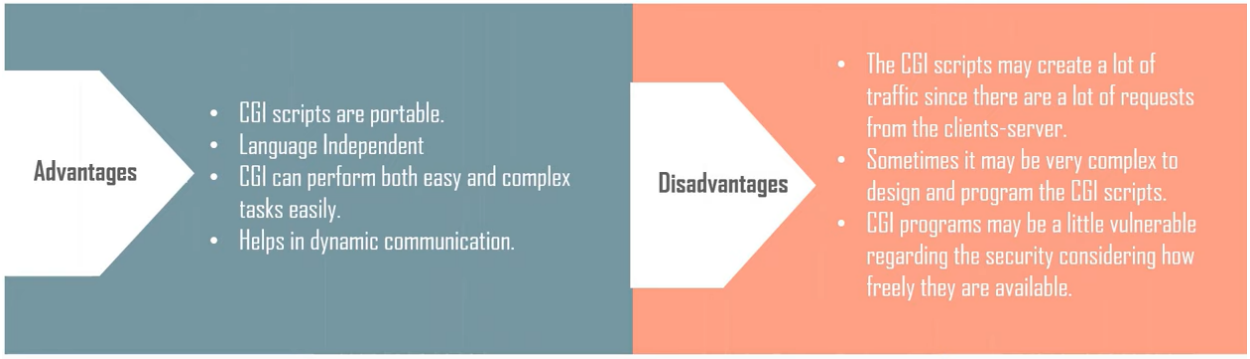
HTTP Headers

Some of the most used HTTP headers in CGI programming



Environment Variables

Environment variables are a series of hidden values that the web server sends to every CGI program you run.



Advantages And Disadvantages

Some of the advantages and disadvantages of CGI programming

First CGI program using Python

Follow the above steps to run first CGI script

Why to Use Python For Web Development?

Undoubtedly, Python has become one of the most dominant programming languages today. ***As per stats by major survey sites, Python has ranked among the top coding language for the past few years now.*** There are tons of reasons for choosing Python as the primary language if compared with others such as Java, C++, or PHP. Since we're here to talk about web development, surprisingly even in web development, Python has reached its peak with tons of features, and improvements, and over the period it is becoming popular every day.

What's the most crucial point from a developer's point of view is to pick the right language that can deliver the desired results with ease, especially when we talk about web development, there are certain factors to consider that include ***management of database, security, and data retrieval*** and so on. Now, how these factors fit in with Python is still clueless for most programmers because they've been using Java, PHP, etc. for web development but today, even big tech giants like ***Netflix, Google, and NASA have been actively using Python for web development.*** So, in this article, we will see *why Python can be considered for web development* and became famous among the top programming language over these years.

5 Reasons to Choose Python for Web Development:

1. Multi-Purpose Programming Language

In contrast, Multi-purpose can simply be referred to as a multi-functionality having capabilities to operate in multiple ways. Surprisingly a developer can do wonders and can easily develop software by implementing easy methods.

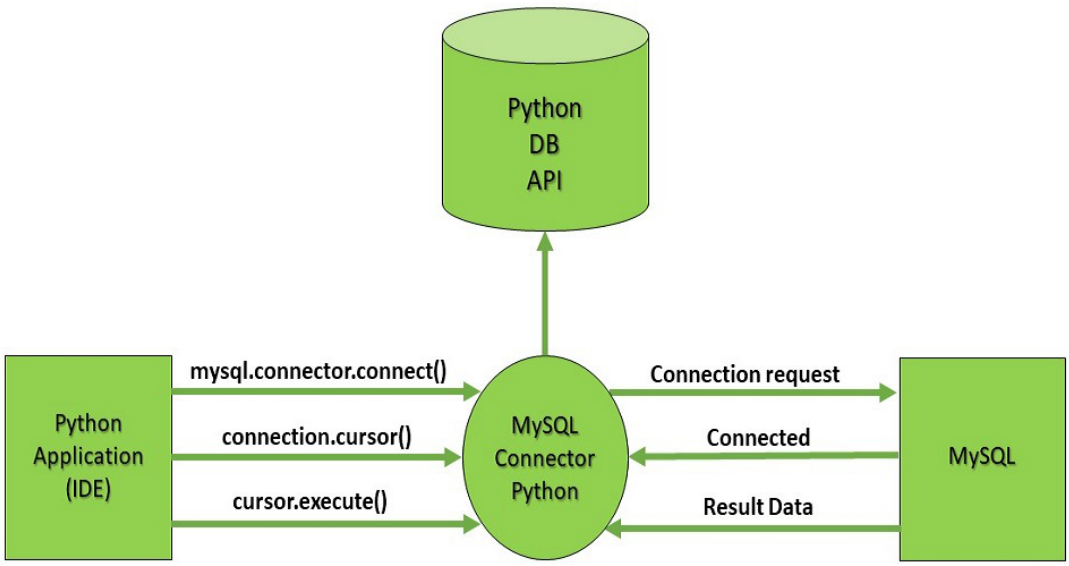
Being an interpreted language (smoothness in dev. process), this platform is absolutely free and open for all, it also offers platform independency, meaning their (Python's) code can run on any platform without making any changes (such as Linux, macOS, etc.)

Besides this, Python can also be used in various ways:

- **For Web Applications:** Python is acing of the web development field and offers many frameworks to work with. Some of the most popular tools are Django, Flask, etc.
- **Desktop Applications:** It is being used to create fascinating desktop applications by many companies these days and some of the widely used tools are Tkinter, PyGUI, Kivy, etc.
- **Cybersecurity:** For malware analysis, developers are actively using highly secured tools to prevent any cyber attacks, tools like NumPy, Pandas, etc. are considered the perfect choices for it.
- **Scientific Calculation & Computing:** The simplicity of Python allows developers to write more reliable systems and working on complex algorithms is much easier with this.

2. Database Connectivity

Establishing database connectivity is pretty simple with Python and accessing (including implementation) can easily be done on major databases such as Oracle, MySQL, PostgreSQL, etc and they can be called up by their respective **APIs** when required. Mention a demographic image below for best reference on how connectivity is being established.



Some of the most common used DB connections are:

- .cursor()
- .commit()
- .rollback()

- .close(), etc.

3. Simplifies -> Debugging – Deployment – Prototyping

For Testing

There are 2 major testings that can be performed within python programming:

- **Doctest:** It offers execution that starts with >>> while comparing to the desired outputs.

Unit testing: It's a technique used by developers to perform testing of a particular segment (of course to check for any bugs) and it provides testing of any individual unit for spotting and fixing errors.

4. Bunch of Frameworks

There's a list of some influential frameworks that help in building sites and can easily fit into your project. And that too by offering enhanced security for either simple or complex websites

5. Growing Community Base

As per the recent survey, over **10 million developers** are helping in making it a strong base and actively offering their inputs whenever someone gets stuck. The number is high and is primarily for rectifying the errors that exist during designing the language.

Features of CGI:

- It is a very well defined and supported standard.
- CGI scripts are generally written in either Perl, C, or maybe just a simple shell script.
- CGI is a technology that interfaces with HTML.
- CGI is the best method to create a counter because it is currently the quickest
- CGI standard is generally the most compatible with today's browsers

Advantages of CGI:

- The advanced tasks are currently a lot easier to perform in CGI than in Java.
- It is always easier to use the code already written than to write your own.
- CGI specifies that the programs can be written in any language, and on any platform, as long as they conform to the specification.
- CGI-based counters and CGI code to perform simple tasks are available in plenty.

Disadvantages of CGI:

There are some disadvantages of CGI which are given below:

- In Common Gateway Interface each page load incurs overhead by having to load the programs into memory.
- Generally, data cannot be easily cached in memory between page loads.
- There is a huge existing code base, much of it in Perl.
- CGI uses up a lot of processing time.

Python - HTTP Client

In the http protocol, the request from the client reaches the server and fetches some data and metadata assuming it is a valid request. We can analyze this response from the server

using various functions available in the python requests module. Here the below python programs run in the client side and display the result of the response sent by the server.

Get Session Object Response

The Session object allows you to persist certain parameters across requests. It also persists cookies across all requests made from the Session instance. If you're making several requests to the same host, the underlying TCP connection will be reused.

```
import requests
s = requests.Session()

s.get('http://httpbin.org/cookies/set/sessioncookie/31251425')
r = s.get('http://httpbin.org/cookies')

print(r.text)
```

When we run the above program, we get the following output –

```
{"cookies":{"sessioncookie":"31251425"}}
```

Handling Error

In case some error is raised because of issue in processing the request by the server, the python program can gracefully handle the exception raised using the timeout parameter as shown below. The program will wait for the defined value of the timeout error and then raise the time out error.

```
requests.get('http://github.com', timeout=10.001)
```