

Comment avoir une interface graphique, sans les widgets ?

*Disposer d'une interface graphique, sans utiliser les widgets comme tkinter ?
Utiliser une interface type web !*

Contenu

Introduction.....	2
Module http.....	3
Créer un serveur web rapidement en python.....	3
Serveur web python 2	3
Serveur web python 3	4
Créer une page web	4
Afficher les erreurs sur votre page web	5
Afficher les variables d'environnement	5
Create a simple http server with Python 3.....	5
Module cgi (et cgitb)	22
Web Forms with Python.....	22
Text Area	22
Text Input	23
Radio buttons	24
Check box	24
Drop down menu.....	25
Legend	26
Password field	26
More from this category:	27
Module Flask	58

Introduction

Tout est parti de là :

<http://stackoverflow.com/questions/7943751/what-is-the-python-3-equivalent-of-python-m-simplehttpserver>

<https://docs.python.org/3/library/http.server.html>

<https://docs.python.org/3/library/cgi.html>

<https://docs.python.org/3/library/subprocess.html>

<https://openclassrooms.com/forum/sujet/lancer-un-script-python-depuis-un-autre-script-pyt>

<http://www.pythonforbeginners.com/os/subprocess-for-system-administrators>

<https://docs.python.org/3/library/winreg.html> pour creation / modif ODBC dans registry

<https://pabitrapp.wordpress.com/2016/09/08/accessing-windows-registry-using-pythons-winreg-module/>

<http://effbot.org/librarybook/winreg.htm>

<http://www.programcreek.com/python/index/2100/winreg>

modules cgi et cgiitb <http://raspberrypiwebserver.com/cgiscripting/web-forms-with-python.html>

Juste avec socket <http://blog.scphillips.com/posts/2012/12/a-simple-python-webserver/>

??? <http://www.whatimade.today/data-junkies-part-1/>

Il existe des modules (Flask, CherryPy, ...), non livré avec la distribution de base, qui facilite la construction de ce type d'interface. En limitant le niveau d'exigence, on peut faire avec les modules livrés avec l'installation de base, les modules `http` et `cgi` pour Python3.

Module http

What is the Python 3 equivalent of `python -m SimpleHTTPServer`?

So, your command is `python3 -m http.server`

```
pushd ./dist; python -m http.server 9000; popd
```

```
import http.server
import socketserver

PORT = 8000

Handler = http.server.SimpleHTTPRequestHandler

httpd = socketserver.TCPServer(("", PORT), Handler)

print("serving at port", PORT)
httpd.serve_forever()
```

Créer un serveur web rapidement en python

<http://apprendre-python.com/page-python-serveur-web-cree-rapidement>

Il peut être intéressant, dans certains cas, d'implémenter un serveur web dans votre application. Cela permet notamment une communication entre vos programmes via un navigateur. En Python créer un serveur web, c'est quelques lignes de code :

Pour approfondir le sujet: [Doc python CGI](#)

Serveur web python 2

Voici le code pour créer un serveur web en python 2:

```
#!/usr/bin/python

import BaseHTTPServer
import CGIHTTPServer

PORT = 8888
server_address = ("", PORT)
```

```

server = BaseHTTPServer.HTTPServer
handler = CGIHTTPServer.CGIHTTPRequestHandler
handler.cgi_directories = ["/"]
print "Serveur actif sur le port :", PORT

httpd = server(server_address, handler)
httpd.serve_forever()

```

Serveur web python 3

```

#!/usr/bin/python3

import http.server

PORT = 8888
server_address = ("", PORT)

server = http.server.HTTPServer
handler = http.server.CGIHTTPRequestHandler
handler.cgi_directories = ["/"]
print("Serveur actif sur le port :", PORT)

httpd = server(server_address, handler)
httpd.serve_forever()

```

Créer une page web

Créez un fichier index.py à la racine de votre projet.

```

#!/usr/bin/python3
# -*- coding: utf-8 -*-

import cgi

form = cgi.FieldStorage()
print("Content-type: text/html; charset=utf-8\n")

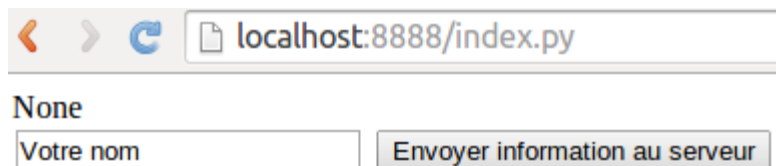
print(form.getvalue("name"))

html = """<!DOCTYPE html>
<head>
    <title>Mon programme</title>
</head>
<body>
    <form action="/index.py" method="post">
        <input type="text" name="name" value="Votre nom" />
        <input type="submit" name="send" value="Envoyer information au
serveur">
    </form>
</body>
</html>
"""

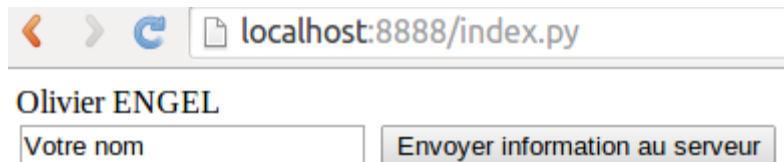
print(html)

```

Ouvrez votre navigateur et indiquez-lui l'url de votre serveur web, dans notre cas ce sera **localhost:8888/index.py**



Indiquez votre nom puis cliquez sur le bouton "Envoyer":



A noter que python ne fait pas de différences entre POST et GET, vous pouvez passer une variable dans l'url le résultat sera le même:

```
http://localhost:8888/index.py?name=Olivier%20ENGEL
```

Afficher les erreurs sur votre page web

Vous pouvez ajouter une fonction qui affichera les erreurs rencontrées par python (mode debug):

```
import cgi; cgi.enable()
```

Afficher les variables d'environnement

Vous pouvez afficher toutes les informations concernant votre serveur web / page en cours en appelant la méthode **test()**:

```
cgi.test()
```

Create a simple http server with Python 3

<https://daanlenaerts.com/blog/2015/06/03/create-a-simple-http-server-with-python-3/>

Knowing how to create a simple http server comes in very handy, especially when working on projects where your application has to be accessed by a remote device. It's not very difficult to create an http server with Python, so let's dive straight into it.

Python is a very good language to do this, because it runs on Windows, Mac OS X and Linux, which makes your little web server very universal. Especially in a development environment.

To get started, create a new Python script with the following contents:

```
#!/usr/bin/env python
```

```

from http.server import BaseHTTPRequestHandler, HTTPServer

# HTTPRequestHandler class
class testHTTPServer_RequestHandler(BaseHTTPRequestHandler):

    # GET
    def do_GET(self):
        # Send response status code
        self.send_response(200)

        # Send headers
        self.send_header('Content-type', 'text/html')
        self.end_headers()

        # Send message back to client
        message = "Hello world!"
        # Write content as utf-8 data
        self.wfile.write(bytes(message, "utf8"))
        return

def run():
    print('starting server...')

    # Server settings
    # Choose port 8080, for port 80, which is normally used for a http
    server, you need root access
    server_address = ('127.0.0.1', 8081)
    httpd = HTTPServer(server_address, testHTTPServer_RequestHandler)
    print('running server...')
    httpd.serve_forever()

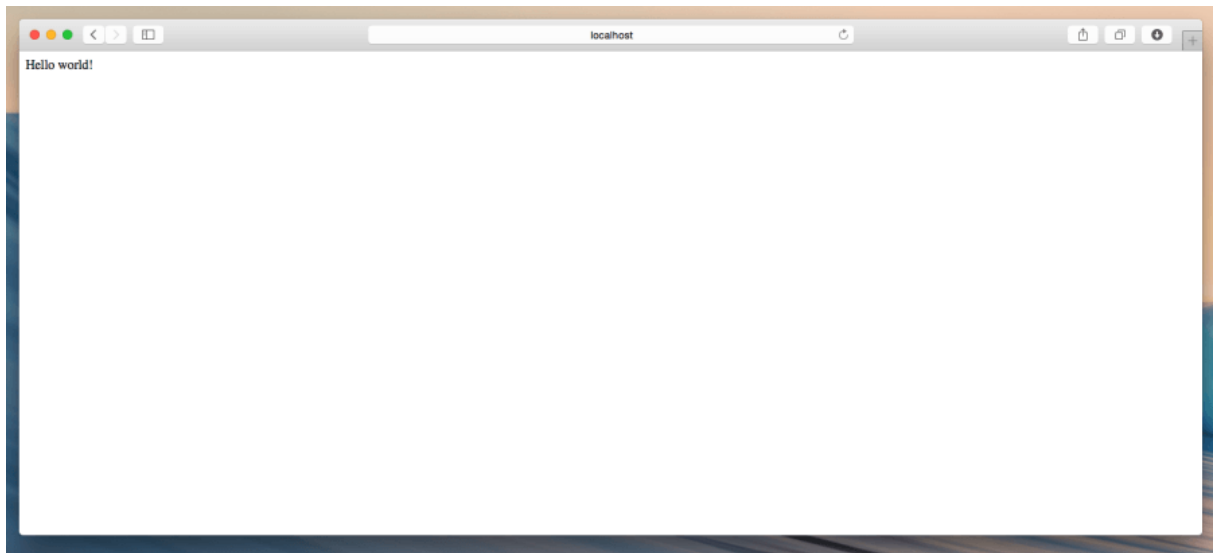
run()

```

When you run the script, the server will start on port 8081. If you want to use port 80, for example, you'll have to run your application with root permissions, so that's why I'm going with port 8081.

Whenever the server gets a GET request, it'll run the do_GET function, and return a 200 (success) status code header, as well as the text/html content type header to the client system. Along with an hello world message.

To test the http server, open your web browser and type "http://localhost:8081", you'll see this:



The `do_GET` method is a method that gets called automatically, through the `BaseHTTPRequestHandler` we're subclassing from, when a GET request arrives.

The same can be done with other HTTP methods like `do_POST` for example

How can I print the request string?

. You can do so by placing the following code in the `do_GET` definition:

```
print(self.path)
```

Write a simple HTTP server in Python

https://www.acmesystems.it/python_httpd

The default Python distribution has a built-in support to the HTTP protocol that you can use to make a simple stand-alone Web server.

The Python module that provides this support is called [BaseFTTPServer](#) and can be used in our programs just including it in our sources:

```
from BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer
```

Hello world !

Let's start with the first basic example. It just return "Hello World !" on the web browser.

[example1.py](#)

```
#!/usr/bin/python
from BaseHTTPServer import BaseHTTPRequestHandler,HTTPServer

PORT_NUMBER = 8080

#This class will handles any incoming request from
#the browser
class myHandler(BaseHTTPRequestHandler):

    #Handler for the GET requests
    def do_GET(self):
        self.send_response(200)
        self.send_header('Content-type','text/html')
        self.end_headers()
        # Send the html message
        self.wfile.write("Hello World !")
        return

try:
    #Create a web server and define the handler to manage the
    #incoming request
    server = HTTPServer(('', PORT_NUMBER), myHandler)
    print 'Started httpserver on port ' , PORT_NUMBER

    #Wait forever for incoming http requests
    server.serve_forever()

except KeyboardInterrupt:
    print '^C received, shutting down the web server'
    server.socket.close()
```

Run it by typing:

```
debarm:~/playground/python/httpserver# python example1.py
```

Then open your web browser on this URL: **http://fox_ip_address:8080**.

"Hello World !" will appear on your browser and two log messages on the FOX console:

```
Started httpserver on port 8080
10.55.98.7 - - [30/Jan/2012 15:40:52] "GET / HTTP/1.1" 200 -
10.55.98.7 - - [30/Jan/2012 15:40:52] "GET /favicon.ico HTTP/1.1" 200 -
```

Serve static files

Let's try an example that shows how to serve static files like index.html, style.css, javascript code and images:

[example2.py](#)

```
#!/usr/bin/python
from BaseHTTPServer import BaseHTTPRequestHandler,HTTPServer
from os import curdir, sep

PORT_NUMBER = 8080

#This class will handles any incoming request from
```



```

#the browser
class myHandler(BaseHTTPRequestHandler):

    #Handler for the GET requests
    def do_GET(self):
        if self.path=="/":
            self.path="/index_example2.html"

        try:
            #Check the file extension required and
            #set the right mime type

            sendReply = False
            if self.path.endswith(".html"):
                mimetype='text/html'
                sendReply = True
            if self.path.endswith(".jpg"):
                mimetype='image/jpg'
                sendReply = True
            if self.path.endswith(".gif"):
                mimetype='image/gif'
                sendReply = True
            if self.path.endswith(".js"):
                mimetype='application/javascript'
                sendReply = True
            if self.path.endswith(".css"):
                mimetype='text/css'
                sendReply = True

            if sendReply == True:
                #Open the static file requested and send it
                f = open(curdir + sep + self.path)
                self.send_response(200)
                self.send_header('Content-type',mimetype)
                self.end_headers()
                self.wfile.write(f.read())
                f.close()

            return

        except IOError:
            self.send_error(404,'File Not Found: %s' %
self.path)

    try:
        #Create a web server and define the handler to manage the
        #incoming request
        server = HTTPServer(('', PORT_NUMBER), myHandler)
        print 'Started httpserver on port ' , PORT_NUMBER

        #Wait forever for incoming http requests
        server.serve_forever()

    except KeyboardInterrupt:
        print '^C received, shutting down the web server'
        server.socket.close()

```

This new example:

- check the extension of the file requested file
- set the right mime type to give back to the browser
- open the static file requested
- send it back to the browser

Run it by typing:

```
debarm:~/playground/python/httpserver# python example2.py
```

then open your web browser on this URL: **http://fox_ip_address:8080**.

An HTML home page will appear on your browser.

Read data from a form

Now explore how to read incoming data from an HTML form.

[example3.py](#)

```
#!/usr/bin/python
from BaseHTTPServer import BaseHTTPRequestHandler, HTTPServer
from os import curdir, sep
import cgi

PORT_NUMBER = 8080

#This class will handles any incoming request from
#the browser
class myHandler(BaseHTTPRequestHandler):

    #Handler for the GET requests
    def do_GET(self):
        if self.path == "/":
            self.path = "/index_example3.html"

        try:
            #Check the file extension required and
            #set the right mime type

            sendReply = False
            if self.path.endswith(".html"):
                mimetype = 'text/html'
                sendReply = True
            if self.path.endswith(".jpg"):
                mimetype = 'image/jpg'
                sendReply = True
            if self.path.endswith(".gif"):
                mimetype = 'image/gif'
                sendReply = True
            if self.path.endswith(".js"):
                mimetype = 'application/javascript'
                sendReply = True
            if self.path.endswith(".css"):
                mimetype = 'text/css'
                sendReply = True

            if sendReply == True:
```

```

        #Open the static file requested and send it
        f = open(curdir + sep + self.path)
        self.send_response(200)
        self.send_header('Content-type',mimetype)
        self.end_headers()
        self.wfile.write(f.read())
        f.close()

    return

except IOError:
    self.send_error(404,'File Not Found: %s' %
self.path)

#Handler for the POST requests
def do_POST(self):
    if self.path=="/send":
        form = cgi.FieldStorage(
            fp=self.rfile,
            headers=self.headers,
            environ={'REQUEST_METHOD':'POST',
                'CONTENT_TYPE':self.headers['Content-
Type'],

            })

        print "Your name is: %s" % form["your_name"].value
        self.send_response(200)
        self.end_headers()
        self.wfile.write("Thanks %s !" %
form["your_name"].value)
        return

try:
    #Create a web server and define the handler to manage the
    #incoming request
    server = HTTPServer(('', PORT_NUMBER), myHandler)
    print 'Started httpserver on port ' , PORT_NUMBER

    #Wait forever for incoming http requests
    server.serve_forever()

except KeyboardInterrupt:
    print '^C received, shutting down the web server'
    server.socket.close()

```

Run this example:

```
debarm:~/playground/python/httpserver# python example3.py
```

and type your name in the "Your name" label.

Example: Simple HTTP Server (Python 3)

http://xlsxwriter.readthedocs.io/example_http_server3.html

Example of using Python and XlsxWriter to create an Excel XLSX file in an in memory string suitable for serving via SimpleHTTPRequestHandler or Django or with the Google App Engine.

Even though the final file will be in memory, via the BytesIO object, the module uses temp files during assembly for efficiency. To avoid this on servers that don't allow temp files, for example the Google APP Engine, set the `in_memory` constructor option to `True`.

For a Python 2 example see [Example: Simple HTTP Server \(Python 2\)](#).

```
#####  
###  
#  
# Example of using Python and XlsxWriter to create an Excel XLSX file in an  
# in  
# memory string suitable for serving via SimpleHTTPRequestHandler or Django  
# or  
# with the Google App Engine.  
#  
# Copyright 2013-2016, John McNamara, jmcnamara@cpan.org  
#  
  
# Note: This is a Python 3 example. For Python 2 see http_server_py2.py.  
  
import http.server  
import socketserver  
import io  
  
import xlsxwriter  
  
class Handler(http.server.SimpleHTTPRequestHandler):  
    def do_GET(self):  
        # Create an in-memory output file for the new workbook.  
        output = io.BytesIO()  
  
        # Even though the final file will be in memory the module uses temp  
        # files during assembly for efficiency. To avoid this on servers  
that  
the  
        # don't allow temp files, for example the Google APP Engine, set  
        # 'in_memory' constructor option to True:  
        workbook = xlsxwriter.Workbook(output, {'in_memory': True})  
        worksheet = workbook.add_worksheet()  
  
        # Write some test data.  
        worksheet.write(0, 0, 'Hello, world!')  
  
        # Close the workbook before streaming the data.  
        workbook.close()  
  
        # Rewind the buffer.  
        output.seek(0)
```

```

        # Construct a server response.
        self.send_response(200)
        self.send_header('Content-Disposition', 'attachment;
filename=test.xlsx')
        self.send_header('Content-type',
                          'application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet')
        self.end_headers()
        self.wfile.write(output.read())
        return

print('Server listening on port 8000...')
httpd = socketserver.TCPServer(('', 8000), Handler)
httpd.serve_forever()

```

docs.python.org :: http.server

<https://docs.python.org/3/library/http.server.html>

Source code: [Lib/http/server.py](#)

This module defines classes for implementing HTTP servers (Web servers).

One class, [HTTPServer](#), is a [socketserver.TCPServer](#) subclass. It creates and listens at the HTTP socket, dispatching the requests to a handler. Code to create and run the server looks like this:

```

def run(server_class=HTTPServer, handler_class=BaseHTTPRequestHandler):
    server_address = ('', 8000)
    httpd = server_class(server_address, handler_class)
    httpd.serve_forever()
class http.server.HTTPServer(server_address, RequestHandlerClass)

```

This class builds on the [TCPServer](#) class by storing the server address as instance variables named `server_name` and `server_port`. The server is accessible by the handler, typically through the handler's `server` instance variable.

The [HTTPServer](#) must be given a *RequestHandlerClass* on instantiation, of which this module provides three different variants:

```
class http.server.BaseHTTPRequestHandler(request, client_address, server)
```

This class is used to handle the HTTP requests that arrive at the server. By itself, it cannot respond to any actual HTTP requests; it must be subclassed to handle each request method (e.g. GET or POST). [BaseHTTPRequestHandler](#) provides a number of class and instance variables, and methods for use by subclasses.

The handler will parse the request and the headers, then call a method specific to the request type. The method name is constructed from the request. For example, for the request method SPAM, the `do_SPAM()` method will be called with no arguments. All of the relevant information is stored in instance variables of the handler. Subclasses should not need to override or extend the [__init__\(\)](#) method.

[BaseHTTPRequestHandler](#) has the following instance variables:

`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.

`requestline`

Contains the string representation of the HTTP request line. The terminating CRLF is stripped. This attribute should be set by [handle_one_request\(\)](#). If no valid request line was processed, it should be set to the empty string.

`command`

Contains the command (request type). For example, 'GET'.

`path`

Contains the request path.

`request_version`

Contains the version string from the request. For example, 'HTTP/1.0'.

`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.

`rfile`

An [io.BufferedReader](#) input stream, ready to read from the start of the optional input data.

`wfile`

Contains the output stream for writing a response back to the client. Proper adherence to the HTTP protocol must be used when writing to this stream.

Changed in version 3.6: This is an [io.BufferedReader](#) stream.

[BaseHTTPRequestHandler](#) has the following attributes:

`server_version`

Specifies the server software version. You may want to override this. The format is multiple whitespace-separated strings, where each string is of the form `name[/version]`. For example, `'BaseHTTP/0.2'`.

`sys_version`

Contains the Python system version, in a form usable by the [version_string](#) method and the [server_version](#) class variable. For example, `'Python/1.4'`.

`error_message_format`

Specifies a format string that should be used by [send_error\(\)](#) method for building an error response to the client. The string is filled by default with variables from [responses](#) based on the status code that passed to [send_error\(\)](#).

`error_content_type`

Specifies the Content-Type HTTP header of error responses sent to the client. The default value is `'text/html'`.

`protocol_version`

This specifies the HTTP protocol version used in responses. 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'`.

`MessageClass`

Specifies an [email.message.Message](#)-like class to parse HTTP headers. Typically, this is not overridden, and it defaults to `http.client.HTTPMessage`.

`responses`

This attribute contains a mapping of error code integers to two-element tuples containing a short and long message. For example, `{code: (shortmessage, longmessage) }`. The *shortmessage* is usually used as the *message* key in an error response, and *longmessage* as the *explain* key. It is used by [send_response_only\(\)](#) and [send_error\(\)](#) methods.

A [BaseHTTPRequestHandler](#) instance has the following methods:

`handle()`

Calls [handle_one_request\(\)](#) once (or, if persistent connections are enabled, multiple times) to handle incoming HTTP requests. You should never need to override it; instead, implement appropriate `do_*()` methods.

`handle_one_request()`

This method will parse and dispatch the request to the appropriate `do_*()` method. You should never need to override it.

`handle_expect_100()`

When a HTTP/1.1 compliant server receives an `Expect: 100-continue` request header it responds back with a `100 Continue` followed by `200 OK` headers. This method can be overridden to raise an error if the server does not want the client to continue. For e.g. server can chose to send `417 Expectation Failed` as a response header and `return False`.

New in version 3.2.

`send_error(code, message=None, explain=None)`

Sends and logs a complete error reply to the client. The numeric *code* specifies the HTTP error code, with *message* as an optional, short, human readable description of the error. The *explain* argument can be used to provide more detailed information about the error; it will be formatted using the [error_message_format](#) attribute and emitted, after a complete set of headers, as the response body. The [responses](#) attribute holds the default values for *message* and *explain* that will be used if no value is provided; for unknown codes the default value for both is the string `???`. The body will be empty if the method is HEAD or the response code is one of the following: `1xx`, `204 No Content`, `205 Reset Content`, `304 Not Modified`.

Changed in version 3.4: The error response includes a Content-Length header. Added the *explain* argument.

`send_response(code, message=None)`

Adds a response header to the headers buffer and logs the accepted request. The HTTP response line is written to the internal buffer, followed by *Server* and *Date* headers. The values for these two headers are picked up from the [version_string\(\)](#) and [date_time_string\(\)](#) methods, respectively. If the server does not intend to send any other headers using the [send_header\(\)](#) method, then [send_response\(\)](#) should be followed by an [end_headers\(\)](#) call.

Changed in version 3.3: Headers are stored to an internal buffer and [end_headers\(\)](#) needs to be called explicitly.

```
send_header(keyword, value)
```

Adds the HTTP header to an internal buffer which will be written to the output stream when either [end_headers\(\)](#) or [flush_headers\(\)](#) is invoked. *keyword* should specify the header keyword, with *value* specifying its value. Note that, after the `send_header` calls are done, [end_headers\(\)](#) MUST BE called in order to complete the operation.

Changed in version 3.2: Headers are stored in an internal buffer.

```
send_response_only(code, message=None)
```

Sends the response header only, used for the purposes when 100 Continue response is sent by the server to the client. The headers not buffered and sent directly the output stream. If the *message* is not specified, the HTTP message corresponding the response *code* is sent.

New in version 3.2.

```
end_headers()
```

Adds a blank line (indicating the end of the HTTP headers in the response) to the headers buffer and calls [flush_headers\(\)](#).

Changed in version 3.2: The buffered headers are written to the output stream.

```
flush_headers()
```

Finally send the headers to the output stream and flush the internal headers buffer.

New in version 3.3.

```
log_request(code='-', size='-')
```

Logs an accepted (successful) request. *code* should specify the numeric HTTP code associated with the response. If a size of the response is available, then it should be passed as the *size* parameter.

```
log_error(...)
```

Logs an error when a request cannot be fulfilled. By default, it passes the message to [log_message\(\)](#), so it takes the same arguments (*format* and additional values).

```
log_message(format, ...)
```

Logs an arbitrary message to `sys.stderr`. This is typically overridden to create custom error logging mechanisms. The *format* argument is a standard printf-style format string, where the additional arguments to [log_message\(\)](#) are applied as inputs to the formatting. The client ip address and current date and time are prefixed to every message logged.

```
version_string()
```

Returns the server software's version string. This is a combination of the [server_version](#) and [sys_version](#) attributes.

```
date_time_string(timestamp=None)
```

Returns the date and time given by *timestamp* (which must be `None` or in the format returned by [time.time\(\)](#)), formatted for a message header. If *timestamp* is omitted, it uses the current date and time.

The result looks like 'Sun, 06 Nov 1994 08:49:37 GMT'.

```
log_date_time_string()
```

Returns the current date and time, formatted for logging.

```
address_string()
```

Returns the client address.

Changed in version 3.3: Previously, a name lookup was performed. To avoid name resolution delays, it now always returns the IP address.

```
class http.server.SimpleHTTPRequestHandler(request, client_address, server)
```

This class serves files from the current directory and below, directly mapping the directory structure to HTTP requests.

A lot of the work, such as parsing the request, is done by the base class [BaseHTTPRequestHandler](#). This class implements the [do_GET\(\)](#) and [do_HEAD\(\)](#) functions.

The following are defined as class-level attributes of [SimpleHTTPRequestHandler](#):

```
server_version
```

This will be "SimpleHTTP/" + `__version__`, where `__version__` is defined at the module level.

`extensions_map`

A dictionary mapping suffixes into MIME types. The default is signified by an empty string, and is considered to be `application/octet-stream`. The mapping is used case-insensitively, and so should contain only lower-cased keys.

The [SimpleHTTPRequestHandler](#) class defines the following methods:

`do_HEAD()`

This method serves the 'HEAD' request type: it sends the headers it would send for the equivalent GET request. See the [do_GET\(\)](#) method for a more complete explanation of the possible headers.

`do_GET()`

The request is mapped to a local file by interpreting the request as a path relative to the current working directory.

If the request was mapped to a directory, the directory is checked for a file named `index.html` or `index.htm` (in that order). If found, the file's contents are returned; otherwise a directory listing is generated by calling the `list_directory()` method. This method uses [os.listdir\(\)](#) to scan the directory, and returns a 404 error response if the [listdir\(\)](#) fails.

If the request was mapped to a file, it is opened and the contents are returned. Any [OSError](#) exception in opening the requested file is mapped to a 404, 'File not found' error. Otherwise, the content type is guessed by calling the `guess_type()` method, which in turn uses the `extensions_map` variable.

A 'Content-type:' header with the guessed content type is output, followed by a 'Content-Length:' header with the file's size and a 'Last-Modified:' header with the file's modification time.

Then follows a blank line signifying the end of the headers, and then the contents of the file are output. If the file's MIME type starts with `text/` the file is opened in text mode; otherwise binary mode is used.

For example usage, see the implementation of the [test\(\)](#) function invocation in the [http.server](#) module.

The [SimpleHTTPRequestHandler](#) class can be used in the following manner in order to create a very basic webserver serving files relative to the current directory:

```
import http.server
import socketserver

PORT = 8000

Handler = http.server.SimpleHTTPRequestHandler
```

```
with socketserver.TCPServer(("", PORT), Handler) as httpd:
    print("serving at port", PORT)
    httpd.serve_forever()
```

[http.server](#) can also be invoked directly using the `-m` switch of the interpreter with a `port` number argument. Similar to the previous example, this serves files relative to the current directory:

```
python -m http.server 8000
```

By default, server binds itself to all interfaces. The option `-b/--bind` specifies a specific address to which it should bind. For example, the following command causes the server to bind to localhost only:

```
python -m http.server 8000 --bind 127.0.0.1
```

New in version 3.4: `--bind` argument was introduced.

```
class http.server.CGIHTTPRequestHandler(request, client_address, server)
```

This class is used to serve either files or output of CGI scripts from the current directory and below. Note that mapping HTTP hierarchic structure to local directory structure is exactly as in [SimpleHTTPRequestHandler](#).

Note

CGI scripts run by the [CGIHTTPRequestHandler](#) class cannot execute redirects (HTTP code 302), because code 200 (script output follows) is sent prior to execution of the CGI script. This pre-empts the status code.

The class will however, run the CGI script, instead of serving it as a file, if it guesses it to be a CGI script. Only directory-based CGI are used — the other common server configuration is to treat special extensions as denoting CGI scripts.

The `do_GET()` and `do_HEAD()` functions are modified to run CGI scripts and serve the output, instead of serving files, if the request leads to somewhere below the `cgi_directories` path.

The [CGIHTTPRequestHandler](#) defines the following data member:

```
cgi_directories
```

This defaults to `['/cgi-bin', '/htbin']` and describes directories to treat as containing CGI scripts.

The [CGIHTTPRequestHandler](#) defines the following method:

```
do_POST()
```

This method serves the `'POST'` request type, only allowed for CGI scripts. Error 501, “Can only POST to CGI scripts”, is output when trying to POST to a non-CGI url.

Note that CGI scripts will be run with UID of user nobody, for security reasons.
Problems with the CGI script will be translated to error 403.

[CGIHTTPRequestHandler](#) can be enabled in the command line by passing the `--cgi` option:

```
python -m http.server --cgi 8000
```

Module cgi (et cgitb)

Web Forms with Python

<http://raspberrypiwebserver.com/cgiscripting/web-forms-with-python.html>

Web applications usually need get input from users at some point. Whether you're building a blogging site or a web UI for an embedded device, forms are a great way to allow users to interact with a web site.

A form is a set of input fields contained in form HTML tags. The following are types of form fields:

- text area
- text input
- radio buttons
- check boxes
- drop down menus
- legend
- file select
- password
- submit button
- reset button

The form tag contains two attributes, the action attribute which specifies which script should execute when the submit button is clicked, and the method used to pass the form to the server, GET or POST.

When the user fills in a form and clicks the submit button, the form data is sent to the server and processed by the script in the form attribute.

Text Area

Text areas allow users to enter a large amount of text like a blog post.

HTML code:

```
<form action="/cgi-bin/examples/textarea.py" method="POST">  
    <textarea name="post" cols=70 rows=5></textarea><br>
```

```
<input type="submit" value="Submit">
<input type="reset" value="Reset">

</form>
```

A screenshot of a web browser displaying a form. It features a large, empty rectangular text input field. Below the field are two buttons: "Submit" and "Reset".

Python code:

```
#!/usr/bin/env python

import cgi
import cgitb

cgitb.enable()

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

form=cgi.FieldStorage()

if "post" not in form:
    print "<h1>The text area was empty.</h1>"
else:
    text=form["post"].value
    print "<h1>Text from text area:</h1>"
    print cgi.escape(text)
```

Text Input

A text input field can be used to enter a single line of text.

```
<form action="/cgi-bin/examples/textinput.py" method="POST">

    <input type="text" size="70" name="data" value=""><br>

    <input type="submit" value="Submit">
    <input type="reset" value="Reset">

</form>
```

A screenshot of a web browser displaying a form. It features a single-line text input field. Below the field are two buttons: "Submit" and "Reset".

```
#!/usr/bin/env python

import cgi
import cgitb

cgitb.enable()
```

```

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

form=cgi.FieldStorage()

if "data" not in form:
    print "<h1>The text input box was empty.</h1>"
else:
    text=form["data"].value
    print "<h1>Text from text input box:</h1>"
    print cgi.escape(text)

```

Radio buttons

```

<form action="/cgi-bin/examples/radiobuttons.py" method="POST">

    <input type="radio" name="light" value="on">On<br>
    <input type="radio" name="light" value="off">Off
<br>
    <input type="submit" value="Submit">
    <input type="reset" value="Reset">

```

</form>

☐ On

☐ Off

```

#!/usr/bin/env python

import cgi
import cgiib

cgiib.enable()

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

form=cgi.FieldStorage()

if "light" not in form:
    print "<h1>Neither radio button was selected.</h1>"
else:
    text=form["light"].value
    print "<h1>Radio button chosen:</h1>"
    print cgi.escape(text)

```

Check box

Users can pick one or more options

```

<form action="/cgi-bin/examples/checkbox.py" method="POST">

```



```

        <input type="checkbox" name="setting" value="set" > Executable
    <br>
        <input type="submit" value="Submit">
        <input type="reset" value="Reset">

</form>

```

☐ Executable

Submit

Reset

```

#!/usr/bin/env python

import cgi
import cgitb

cgitb.enable()

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

form=cgi.FieldStorage()

if "setting" not in form:
    print "<h1>The box was not checked</h1>"
else:
    text=form["setting"].value
    print "<h1>The check box was ticked:</h1>"
    print tex

```

Drop down menu

Users can pick an option from a drop down menu.

```

<form action="/cgi-bin/examples/dropdown.py" method="POST">

    My favaurite programming language is
    <select name="language">
        <option value="c">C</option>
        <option value="python">Python</option>
        <option value="php">PHP</option>
        <option value="Java">Java</option>
    </select>
    <br>
    <input type="submit" value="Submit">
    <input type="reset" value="Reset">

</form>

```

My favaurite programming language is

Submit Reset

C

C

Python

PHP

```
#!/usr/bin/env python

import cgi
import cgi.b

cgi.b.enable()

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

form=cgi.FieldStorage()

if "language" not in form:
    print "<h1>No option chosen</h1>"
else:
    text=form["language"].value
    print "<h1>The option chosen was:</h1>"
    print text
```

Legend

```
<form action="/cgi-bin/examples/legend.py" method="POST">

    <fieldset>
        <legend>Legend</legend>
        <p>These paragraphs are surrounded by a legend.</p>
        <p>You can use legends to group things together like these two
paragraphs.</p>
    </fieldset>
</form>
```

Legend

These paragraphs are surrounded by a legend.

You can use legends to group things together like these two paragraphs.

Password field

Like a text input box, but characters are obscured as they're typed in.

```
<form action="/cgi-bin/examples/password.py" method="POST">
    Username: <input type="text" name="user_name" value="">
    Password: <input type="password" name="password">
<br>
    <input type="submit" value="Submit">
    <input type="reset" value="Reset">

</form>
```

Username: Password:

```
#!/usr/bin/env python

import cgi
import cgiib

cgiib.enable()

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

form=cgi.FieldStorage()

if "user_name" not in form:
    print "<h1>No username was entered</h1>"
else:
    text=form["user_name"].value
    print "<h1>Username:</h1>"
    print cgi.escape(text)

if "password" not in form:
    print "<h1>No password was entered</h1>"
else:
    text=form["password"].value
    print "<h1>Password:</h1>"
    print cgi.escape(text)
```

More from this category:

[CGI scripting](#)

- [Writing CGI scripts on a Raspberry Pi](#)
- [Writing CGI scripts in Python](#)
- [Using Google charts in Python CGI scripts](#)
- [Sending data to an HTTP server - get and post methods](#)
- [Web Forms with Python](#)
- [Using Python to set, retrieve and clear cookies](#)
- [Setting up Nginx and uWSGI for CGI scripting](#)

[Raspberry Pi Temperature Logger](#)

- [Building an SQLite temperature logger](#)
- [Building a web user interface for the temperature monitor](#)

Python 3 Programming CGI

http://www.w3ii.com/fr/python3/python_cgi_programming.html

Qu'est-ce que CGI?

Le Common Gateway Interface, ou CGI, est une norme pour les programmes de passerelles externes à l'interface avec les serveurs d'information tels que les serveurs HTTP.

La version actuelle est CGI / 1.1 et CGI / 1.2 est en cours de progressio

Navigation sur le Web

Pour comprendre le concept de CGI, nous allons voir ce qui se passe lorsque l'on clique un lien hyper pour parcourir une page web particulière ou URL.

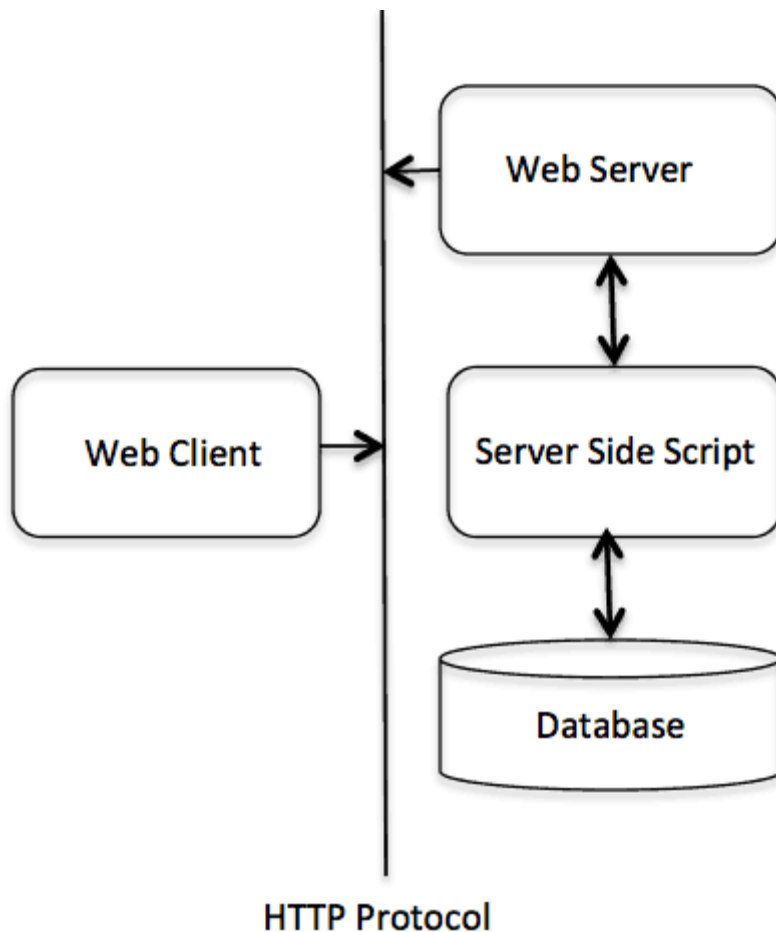
Vos contacts navigateur du serveur HTTP Web et les demandes de l'URL, à savoir, le nom de fichier.

Web Server analyse l'URL et recherche le nom du fichier. Si elle constate que le fichier puis l'envoie au navigateur, sinon envoie un message d'erreur indiquant que vous avez demandé un fichier erroné.

Le navigateur Web prend la réponse du serveur et affiche web soit le message de fichier ou d'erreur reçu.

Cependant, il est possible de configurer le serveur HTTP de sorte que chaque fois qu'un fichier dans un répertoire est demandé que le fichier ne soit pas renvoyé; à la place, il est exécuté en tant que programme, et quel que soit ce que les résultats du programme est renvoyé pour votre navigateur pour afficher. Cette fonction est appelée Common Gateway Interface ou CGI et les programmes sont appelés des scripts CGI. Ces programmes CGI peuvent être un script Python, PERL Script, Shell Script, C ou C ++ programme, etc.

CGI diagramme d'architecture



Premier programme de CGI

Voici un lien simple, qui est lié à un script CGI appelé [hello.py](#) . Ce fichier est stocké dans / var / www / cgi-bin et il a le contenu suivant. Avant l' exécution de votre programme CGI, assurez - vous que vous avez le changement mode de fichier en utilisant **chmod 755 hello.py** **commande UNIX pour rendre le fichier exécutable.**

```
#!/usr/bin/python3

print ("Content-type:text/html")
print ()
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>')
```

Remarque: La première ligne dans le script doit être chemin vers Python exécutable.Dans Linux, il devrait être `#!/usr/bin/python3`

Entrez URL suivante dans le navigateur

<http://localhost:8080/cgi-bin/hello.py>

En-tête HTTP

Le Content-type de ligne: text / html \ r \ n \ r \ n fait partie de tête HTTP qui est envoyé au navigateur pour comprendre le contenu. Tout l'en-tête HTTP sera sous la forme suivante -

HTTP Field Name: Field Content

For Example

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

Il y a peu d'autres en-têtes HTTP importantes, que vous utiliserez fréquemment dans votre programmation CGI.

Entête	La description
Content-Type:	Une chaîne MIME définissant le format du fichier retourné. Exemple est Type de contenu: text / html
Expire: Date	La date à laquelle l'information devient invalide. Il est utilisé par le navigateur de décider quand une page doit être actualisée. Une chaîne de date valide est au format 1 janvier 1998 12:00:00 GMT.
Lieu: URL	L'URL qui est retourné au lieu de l'URL demandée. Vous pouvez utiliser ce champ pour rediriger une requête vers un fichier.
Dernière modification: Date	La date de la dernière modification de la ressource.
Content-length: N	La longueur, en octets, des données étant retourné. Le navigateur utilise cette valeur pour signaler le temps de téléchargement estimé pour un fichier.
Set-Cookie: String	Définissez le cookie passé à travers <i>la chaîne</i>

Variables d'environnement CGI

Tous les programmes CGI ont accès aux variables d'environnement suivantes. Ces variables jouent un rôle important lors de l'écriture tout programme CGI.

Nom de la variable	La description
CONTENT_TYPE	Le type de données du contenu. Utilisé lorsque le client envoie le contenu connecté au serveur. Par exemple, le téléchargement de fichiers.
CONTENT_LENGTH	La longueur de l'information de requête. Il est disponible uniquement pour les requêtes POST.
HTTP_COOKIE	Retourne le jeu des cookies en forme de clé et la valeur paire.
HTTP_USER_AGENT	Le champ request-tête User-Agent contient des informations sur l'agent utilisateur origine de la demande. Il est le nom du navigateur Web.
PATH_INFO	Le chemin d'accès au script CGI.
CHAÎNE DE REQUÊTE	Les informations de code URL qui est envoyé à la demande de

	méthode GET.
REMOTE_ADDR	L'adresse IP de l'hôte distant qui fait la demande. Ceci est utile ou l'exploitation forestière pour l'authentification.
REMOTE_HOST	Le nom complet de l'hôte qui fait la demande. Si ces informations ne sont pas disponibles, alors REMOTE_ADDR peut être utilisé pour obtenir l'adresse IR.
REQUEST_METHOD	La méthode utilisée pour faire la demande. Les méthodes les plus courantes sont GET et POST.
SCRIPT_FILENAME	Le chemin complet du script CGI.
SCRIPT_NAME	Le nom du script CGI.
NOM DU SERVEUR	nom d'hôte ou l'adresse IP du serveur
SERVER_SOFTWARE	Le nom et la version du logiciel du serveur est en cours d'exécution.

Voici petit programme CGI pour lister toutes les variables CGI. Cliquez sur ce lien pour voir le résultat [Obtenir Environnement](#)

```
#!/usr/bin/python3

import os

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

Méthodes GET et POST

Vous devez avoir rencontré de nombreuses situations où vous devez passer des informations à partir de votre navigateur vers le serveur Web et, finalement, à votre programme CGI. Le plus souvent, le navigateur utilise deux méthodes de deux laissez-passer ces informations au serveur Web. Ces méthodes sont la méthode GET et POST Méthode.

Transmission des informations en utilisant la méthode GET

La méthode GET envoie les informations d'utilisateur encodées annexée à la demande de page. La page et les informations codées sont séparés par le caractère « ? » comme suit: -

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

La méthode GET est la méthode par défaut pour transmettre des informations à partir du navigateur vers le serveur Web et il produit une longue chaîne qui apparaît dans l'emplacement de votre navigateur: case. Ne jamais utiliser la méthode GET si vous avez un mot de passe ou d'autres informations sensibles à transmettre au serveur. La méthode GET a la taille limitation: seulement 1024 caractères peuvent être envoyés dans une chaîne de demande. La méthode GET envoie des informations en utilisant la tête de QUERY_STRING et sera accessible dans votre programme CGI par QUERY_STRING variable d'environnement.

Vous pouvez transmettre des informations par simple concaténation des paires de clés et de valeurs ainsi que toute URL ou vous pouvez utiliser HTML balises <form> pour transmettre des informations en utilisant la méthode GET.

Voici une simple URL, qui passe deux valeurs à hello_get.py programme en utilisant la méthode GET.

```
/cgi-bin/hello_get.py?first_name=Malhar&last_name=Lathkar

#!/usr/bin/python3

# 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")
print()
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>")
```

FORMULAIRE Simple Exemple: méthode GET

Cet exemple passe deux valeurs en utilisant le formulaire HTML et le bouton soumettre. Nous utilisons même hello_get.py script CGI pour gérer cette entrée.

```
<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>
```

Voici la sortie réelle du formulaire ci-dessus, vous entrez Nom et prénom puis cliquez sur le bouton soumettre pour voir le résultat.

Prénom:

Nom de famille:

Transmission des informations Utilisation de la méthode POST

Une méthode généralement plus fiable de transmettre des informations à un programme CGI est la méthode POST. Ce emballe les informations exactement de la même manière que les méthodes GET,

mais au lieu de l'envoyer comme une chaîne de texte après « ? » dans l'URL, il l'envoie comme un message séparé. Ce message vient dans le script CGI sous la forme de l'entrée standard.

Ci-dessous est le même script `hello_get.py` qui gère GET ainsi que la méthode POST.

```
#!/usr/bin/python3

# 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")
print()
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>")
```

Reprenons même exemple que ci-dessus qui passe deux valeurs en utilisant le formulaire HTML et le bouton soumettre. Nous utilisons même `hello_get.py` script CGI pour gérer cette entrée.

```
<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>
```

Prénom:

Nom de famille:

Passant Checkbox données au programme CGI

```
<form action="/cgi-bin/checkbox.py" 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>
```

☒ Mathématiques ☐ La physique

```
#!/usr/bin/python3

# 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")
print()
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>")

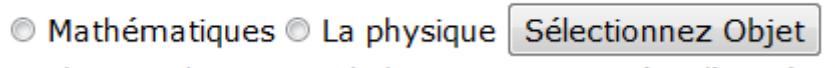
```

Passer des données des boutons de radio au programme CGI

```

<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>

```



```

#!/usr/bin/python3

# 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"

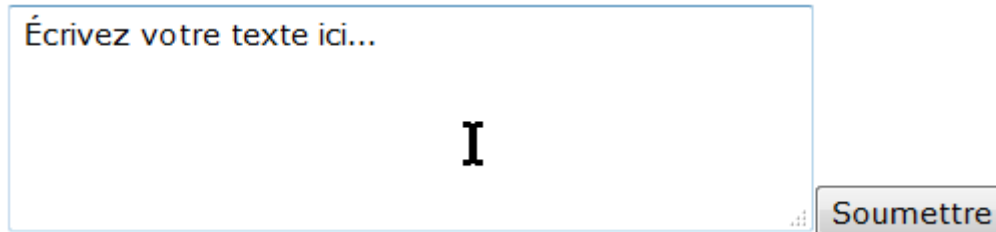
print "Content-type:text/html"
print()
print("<html>")
print("<head>")
print("<title>Radio - Fourth CGI Program</title>")
print("</head>")
print("<body>")
print("<h2> Selected Subject is %s</h2>" % subject)

```

```
print ("</body>")
print ("</html>")
```

Passing Zone de données texte au programme CGI

```
<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>
```



Écrivez votre texte ici...

I

Soumettre

```
#!/usr/bin/python3

# Import modules for CGI handling
import cgi, cgitb

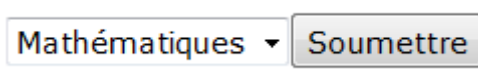
# Create instance of FieldStorage
form = cgi.FieldStorage()

# Get data from fields
if form.getvalue('textcontent'):
    text_content = form.getvalue('textcontent')
else:
    text_content = "Not entered"

print "Content-type:text/html"
print()
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>")
```

Passant Drop Down Box données au programme CGI

```
<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>
```



Mathématiques ▾ Soumettre

```
#!/usr/bin/python3

# 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"
print()
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>")

```

Utilisation des cookies dans CGI

le protocole HTTP est un protocole sans état. Pour un site commercial, il est nécessaire de maintenir des informations de session entre les différentes pages. Par exemple, un enregistrement de l'utilisateur se termine après avoir terminé de nombreuses pages. Comment maintenir les informations de session de l'utilisateur sur l'ensemble des pages web?

Dans de nombreuses situations, en utilisant des cookies est la méthode la plus efficace de la mémoire et le suivi des préférences, des achats, des commissions et autres informations nécessaires pour une meilleure expérience du visiteur ou du site statistiques.

Comment cela fonctionne?

Votre serveur envoie des données au navigateur du visiteur sous la forme d'un cookie. Le navigateur peut accepter le cookie. Si elle le fait, elle est stockée comme un enregistrement de texte brut sur le disque dur du visiteur. Maintenant, lorsque le visiteur arrive sur une autre page sur votre site, le cookie est disponible pour la récupération. Une fois récupéré, votre serveur sait / se souvient ce qui a été stocké.

Les cookies sont un enregistrement brut de données de texte de 5 champs de longueur variable:

Expire: La date à laquelle le cookie expirera. Si cette zone est vide, le cookie expirera lorsque le visiteur quitte le navigateur.

Domaine: Le nom de domaine de votre site.

Chemin: Le chemin d'accès à la page d'annuaire ou web qui définit le cookie. Cela peut être vide si vous souhaitez récupérer le cookie d'un répertoire ou d'une page.

Sécurité: Si ce champ contient le mot «sûr», alors le cookie ne peut être récupéré avec un serveur sécurisé. Si ce champ est vide, aucune restriction existe.

Nom = Valeur: Les cookies sont définis et récupérés sous la forme de paires de clés et de valeurs.

Configuration de cookies

Il est très facile d'envoyer des cookies au navigateur. Ces cookies sont envoyés avec en-tête HTTP avant de champ Content-Type. En supposant que vous souhaitez définir UserID et mot de passe que les cookies. Réglage des témoins se fait comme suit: -

```
#!/usr/bin/python3

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.w3ii.com;\r\n")
print ("Set-Cookie:Path=/perl;\n")
print ("Content-type:text/html\r\n\r\n")
.....Rest of the HTML Content....
```

De cet exemple, vous devez avoir compris comment configurer les cookies. Nous utilisons **Set-Cookie en-tête HTTP pour définir les cookies**.

Elle est facultative pour définir les cookies attributs comme Expire, Domaine et Chemin. Il est à noter que les cookies sont mis en avant l'envoi ligne magique **Type de contenu ": text / html \ r \ n \ r \ n**.

Récupération cookies

Il est très facile de récupérer tous les set cookies. Les cookies sont stockés dans un environnement de CGI variables HTTP_COOKIE et ils auront la forme suivante -

```
key1=value1;key2=value2;key3=value3....
```

```
#!/usr/bin/python3

# 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)
```

Fichier Upload Exemple

Pour télécharger un fichier, le formulaire HTML doit avoir l'attribut enctype **multipart / form-data**. La balise d'entrée avec le type de fichier crée un bouton "Parcourir".

```
<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>
```

Fichier:

No file selected.

```
#!/usr/bin/python3

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,))
```

Si vous exécutez le script ci-dessus sous Unix / Linux, alors vous devez prendre soin de remplacer le séparateur de fichier comme suit, sinon sur votre machine Windows ci-dessus ouverte déclaration () devrait fonctionner correctement.

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

Comment élever un "Téléchargement de fichier" boîte de dialogue?

Parfois, il est souhaitable que vous vouliez donner l'option où un utilisateur peut cliquer sur un lien et il fera apparaître une "Téléchargement de fichier" boîte de dialogue à l'utilisateur au lieu d'afficher le

contenu réel. Ceci est très facile et peut être réalisée par en-tête HTTP. Cet en-tête HTTP est d'être différent de l'en-tête mentionné dans la section précédente.

Par exemple, si vous voulez faire un téléchargement de fichier **FileName** à partir d'un lien donné, sa syntaxe est la suivante -

```
#!/usr/bin/python3

# 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()
```

Apprendre à programmer avec Python/Applications web

https://fr.wikibooks.org/wiki/Apprendre_%C3%A0_programmer_avec_Python/Applications_web

Python pour partager vos répertoires via un serveur web / HTTP

<https://www.parlonsgeek.com/python-partager-vos-repertoires-via-serveur-web-http/>

Si vous voulez partager vos dossiers et fichiers via un serveur HTTP et que pour une raison ou pour une autre vous ne voulez pas vous salir les mains avec la configuration de Apache qui peut s'avérer assez compliquée et peut vous faire perdre pas mal de temps, alors croyez-moi c'est Python que vous cherchez, il vient avec un script très simple à utiliser qui vous permettra de transformer n'importe quel répertoire / dossier en serveur web vous permettant ainsi de partager son contenu avec d'autres personnes, ça leur donnera la possibilité de télécharger et d'uploader des fichiers directement depuis / sur ce répertoire.

Le seul et unique logiciel que vous aurez besoin d'installer est Python (qui n'est pas vraiment un logiciel mais un compilateur de scripts codés en Python – le langage de programmation –). Python est disponible en deux versions, Python2 et Python3, je vous conseille d'installer les deux côte à côte afin de pouvoir exécuter tous les scripts sans problèmes, peu importe la version de Python qu'ils utilisent... Cliquez ici pour télécharger Python2 et Python3 depuis le site officiel.

L'installation sur Windows est très simple, suivant suivant suivant et HOP vous avez Python installé et fonctionnel, Si vous utilisez Linux ou Mac, il y a de grandes chances que vous ayez déjà Python2 pré-installé, vous pouvez vérifier en exécutant l'une des commandes suivantes dans le terminal :

Le port utilisé par défaut est **8000**, vous pouvez le changer en ajoutant un espace et le port que vous voulez utiliser après la commande. Ex : `$ python2 -m SimpleHTTPServer 4444`

Téléchargement et Upload :

Pour permettre l'upload en plus du téléchargement via un serveur web, c'est un peu différent mais ça reste très simple. Il suffit de copier le code suivant qui est une version modifiée du module par défaut de SimpleHTTPServer que nous avons utilisé ci-dessus, et le coller dans un fichier **et assurez-vous bien de lui donner l'extension .py !** moi je choisis de le nommer « **serveurHTTP.py** ».

```
#!/usr/bin/env python
```

```
"""Simple HTTP Server With Upload.
```

```
This module builds on BaseHTTPServer by implementing the standard GET and HEAD requests in a fairly straightforward manner.
```



```

"""

__version__ = "0.1"
__all__ = ["SimpleHTTPRequestHandler"]
__author__ = "bones7456"
__home_page__ = "http://li2z.cn/"

import os
import posixpath
import BaseHTTPServer
import urllib
import cgi
import shutil
import mimetypes
import re
try:
    from cStringIO import StringIO
except ImportError:
    from StringIO import StringIO

class SimpleHTTPRequestHandler(BaseHTTPServer.BaseHTTPRequestHandler):

    """Simple HTTP request handler with GET/HEAD/POST commands.

    This serves files from the current directory and any of its
    subdirectories. The MIME type for files is determined by
    calling the .guess_type() method. And can receive file uploaded
    by client.

    The GET/HEAD/POST requests are identical except that the HEAD
    request omits the actual contents of the file.

    """

    server_version = "SimpleHTTPWithUpload/" + __version__

    def do_GET(self):
        """Serve a GET request."""
        f = self.send_head()
        if f:
            self.copyfile(f, self.wfile)
            f.close()

    def do_HEAD(self):
        """Serve a HEAD request."""
        f = self.send_head()
        if f:
            f.close()

    def do_POST(self):
        """Serve a POST request."""
        r, info = self.deal_post_data()
        print r, info, "by: ", self.client_address
        f = StringIO()
        f.write('<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">')
        f.write("<html>\n<title>Upload Result Page</title>\n")
        f.write("<body>\n<h2>Upload Result Page</h2>\n")
        f.write("<hr>\n")
        if r:

```

```

        f.write("<strong>Success:</strong>")
    else:
        f.write("<strong>Failed:</strong>")
    f.write(info)
    f.write("<br><a href=\"%s\">back</a>" % self.headers['referer'])
    f.write("<hr><small>Powerd By: bones7456, check new version at ")
    f.write("<a
href=\"http://li2z.cn/?s=SimpleHTTPServerWithUpload\">")
    f.write("here</a>.</small></body>\n</html>\n")
    length = f.tell()
    f.seek(0)
    self.send_response(200)
    self.send_header("Content-type", "text/html")
    self.send_header("Content-Length", str(length))
    self.end_headers()
    if f:
        self.copyfile(f, self.wfile)
        f.close()

def deal_post_data(self):
    boundary = self.headers.plisttext.split("=")[1]
    remainbytes = int(self.headers['content-length'])
    line = self.rfile.readline()
    remainbytes -= len(line)
    if not boundary in line:
        return (False, "Content NOT begin with boundary")
    line = self.rfile.readline()
    remainbytes -= len(line)
    fn = re.findall(r'Content-Disposition.*name="file";
filename="(.)"', line)
    if not fn:
        return (False, "Can't find out file name...")
    path = self.translate_path(self.path)
    fn = os.path.join(path, fn[0])
    line = self.rfile.readline()
    remainbytes -= len(line)
    line = self.rfile.readline()
    remainbytes -= len(line)
    try:
        out = open(fn, 'wb')
    except IOError:
        return (False, "Can't create file to write, do you have
permission to write?")

    preline = self.rfile.readline()
    remainbytes -= len(preline)
    while remainbytes > 0:
        line = self.rfile.readline()
        remainbytes -= len(line)
        if boundary in line:
            preline = preline[0:-1]
            if preline.endswith('\r'):
                preline = preline[0:-1]
            out.write(preline)
            out.close()
            return (True, "File '%s' upload success!" % fn)
        else:
            out.write(preline)
            preline = line
    return (False, "Unexpect Ends of data.")

```

```

def send_head(self):
    """Common code for GET and HEAD commands.

    This sends the response code and MIME headers.

    Return value is either a file object (which has to be copied
    to the outputfile by the caller unless the command was HEAD,
    and must be closed by the caller under all circumstances), or
    None, in which case the caller has nothing further to do.

    """
    path = self.translate_path(self.path)
    f = None
    if os.path.isdir(path):
        if not self.path.endswith('/'):
            # redirect browser - doing basically what apache does
            self.send_response(301)
            self.send_header("Location", self.path + "/")
            self.end_headers()
            return None
        for index in "index.html", "index.htm":
            index = os.path.join(path, index)
            if os.path.exists(index):
                path = index
                break
        else:
            return self.list_directory(path)
    ctype = self.guess_type(path)
    try:
        # Always read in binary mode. Opening files in text mode may
        cause
        # newline translations, making the actual size of the content
        # transmitted *less* than the content-length!
        f = open(path, 'rb')
    except IOError:
        self.send_error(404, "File not found")
        return None
    self.send_response(200)
    self.send_header("Content-type", ctype)
    fs = os.fstat(f.fileno())
    self.send_header("Content-Length", str(fs[6]))
    self.send_header("Last-Modified",
self.date_time_string(fs.st_mtime))
    self.end_headers()
    return f

def list_directory(self, path):
    """Helper to produce a directory listing (absent index.html).

    Return value is either a file object, or None (indicating an
    error). In either case, the headers are sent, making the
    interface the same as for send_head().

    """
    try:
        list = os.listdir(path)
    except os.error:
        self.send_error(404, "No permission to list directory")
        return None
    list.sort(key=lambda a: a.lower())
    f = StringIO()

```

```

        displaypath = cgi.escape(urllib.unquote(self.path))
        f.write('<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">')
        f.write("<html>\n<title>Directory listing for %s</title>\n" %
displaypath)
        f.write("<body>\n<h2>Directory listing for %s</h2>\n" %
displaypath)
        f.write("<hr>\n")
        f.write("<form ENCTYPE=\"multipart/form-data\" method=\"post\">")
        f.write("<input name=\"file\" type=\"file\"/>")
        f.write("<input type=\"submit\" value=\"upload\"/></form>\n")
        f.write("<hr>\n<ul>\n")
        for name in list:
            fullname = os.path.join(path, name)
            displayname = linkname = name
            # Append / for directories or @ for symbolic links
            if os.path.isdir(fullname):
                displayname = name + "/"
                linkname = name + "/"
            if os.path.islink(fullname):
                displayname = name + "@"
            # Note: a link to a directory displays with @ and links
with /
                f.write('<li><a href=\"%s\">%s</a>\n'
                        % (urllib.quote(linkname), cgi.escape(displayname)))
        f.write("</ul>\n<hr>\n</body>\n</html>\n")
        length = f.tell()
        f.seek(0)
        self.send_response(200)
        self.send_header("Content-type", "text/html")
        self.send_header("Content-Length", str(length))
        self.end_headers()
        return f

def translate_path(self, path):
    """Translate a /-separated PATH to the local filename syntax.

    Components that mean special things to the local file system
    (e.g. drive or directory names) are ignored. (XXX They should
    probably be diagnosed.)

    """
    # abandon query parameters
    path = path.split('?',1)[0]
    path = path.split('#',1)[0]
    path = posixpath.normpath(urllib.unquote(path))
    words = path.split('/')
    words = filter(None, words)
    path = os.getcwd()
    for word in words:
        drive, word = os.path.splitdrive(word)
        head, word = os.path.split(word)
        if word in (os.curdir, os.pardir): continue
        path = os.path.join(path, word)
    return path

def copyfile(self, source, outputfile):
    """Copy all data between two file objects.

    The SOURCE argument is a file object open for reading
    (or anything with a read() method) and the DESTINATION
    argument is a file object open for writing (or

```

```

anything with a write() method).

The only reason for overriding this would be to change
the block size or perhaps to replace newlines by CRLF
-- note however that this the default server uses this
to copy binary data as well.

"""
shutil.copyfileobj(source, outputfile)

def guess_type(self, path):
    """Guess the type of a file.

    Argument is a PATH (a filename).

    Return value is a string of the form type/subtype,
    usable for a MIME Content-type header.

    The default implementation looks the file's extension
    up in the table self.extensions_map, using application/octet-stream
    as a default; however it would be permissible (if
    slow) to look inside the data to make a better guess.

    """

    base, ext = posixpath.splitext(path)
    if ext in self.extensions_map:
        return self.extensions_map[ext]
    ext = ext.lower()
    if ext in self.extensions_map:
        return self.extensions_map[ext]
    else:
        return self.extensions_map['']

if not mimetypes.inited:
    mimetypes.init() # try to read system mime.types
extensions_map = mimetypes.types_map.copy()
extensions_map.update({
    '': 'application/octet-stream', # Default
    '.py': 'text/plain',
    '.c': 'text/plain',
    '.h': 'text/plain',
})

def test(HandlerClass = SimpleHTTPRequestHandler,
        ServerClass = BaseHTTPServer.HTTPServer):
    BaseHTTPServer.test(HandlerClass, ServerClass)

if __name__ == '__main__':
    test()

```

Ce script est comme je vous avais dit basé sur le module SimpleHTTPServer de python, donc automatiquement lui aussi il utilise 8000 comme port par défaut pour le serveur web, et oui vous l'avez deviné pour ce script aussi on peut indiquer un port différent après la commande.

Aperçu de la CGI avec Python

<https://openclassrooms.com/courses/aperçu-de-la-cgi-avec-python>

Vous aimeriez rendre votre site interactif, par exemple pour mettre en place un compteur de visites, un formulaire de contact, un petit jeu...

Pour ça, tout le monde utilise le PHP. Mais vous faites partie d'une élite éclairée et vous aimeriez le faire avec Python. C'est faisable !

Voici un aperçu de la [CGI](#) avec ce langage.

Juste une chose avant de commencer, si vous n'avez pas d'hébergeur qui vous permette d'utiliser Python sur vos pages web, regardez donc [cette liste](#). Si vous hébergez votre site vous-même, alors sachez qu'il vous suffit d'installer Apache et Python.

Les exemples contenus dans ce cours fonctionnent avec la version 2 de Python. Si vous utilisez la version 3 ou supérieure, n'oubliez pas de remplacer les `print ...` par des `print(...)`

Préparation

Pour que vos scripts soient exploitables, il faut tout d'abord configurer Apache.

Créons un fichier `.htaccess`, que vous placerez à la racine du site (en effet, ce fichier fonctionne récursivement : tous les répertoires situés dans le répertoire contenant votre `.htaccess` seront assujettis aux règles que vous définirez dans celui-ci). Le voici :

```
AddHandler cgi-script .py
Options +ExecCGI
```

Ces deux lignes indiquent que les fichiers `.py` doivent être interprétés avant d'être envoyés au client. Le comportement par défaut serait d'envoyer le fichier comme s'il s'agissait d'un fichier texte ordinaire.

Une bonne chose à faire est de configurer Apache pour considérer vos fichiers `index.py` comme les indexes de vos répertoires. Pour cela, vous pouvez ajouter la ligne suivante à votre `.htaccess` :

```
DirectoryIndex index.py
```

Néanmoins, il reste une chose **très importante** à faire : rendre votre script exécutable. Je dois reconnaître que je me suis souvent fait avoir par ce genre d'erreur. C'est assez pervers car vous ne savez pas pourquoi votre script échoue : vous vous retrouvez face à une erreur 500 fort peu explicite.

Pour ce faire, sous UNIX, il faut utiliser la commande suivante :

```
chmod +x fichier.py
```

Vous **devez** le faire sur chacun de vos scripts.

Maintenant, nous allons voir comment rédiger nos fichiers Python.

La rédaction des scripts

Lorsque vous utilisez Python pour votre site web, vous générez du HTML.

Il vous faut le préciser : vous pourriez très bien renvoyer une image, aussi faut-il informer le client de la nature du document renvoyé.

Pour ce faire, il faut tout simplement utiliser print :

```
#!/usr/bin/python

print 'Content-type: text/html'
print
# Là commence votre code.
```

La première ligne est importante : il s'agit du [shebang](#). Il indique quel interpréteur Python utiliser.

La seconde (en réalité, la troisième car j'ai sauté une ligne) indique le type de document envoyé.

La dernière écrit une ligne vide. En effet, le client et le serveur communiquent par le protocole HTTP. Ce dernier est composé d'un en-tête, dans lequel des informations comme le nom et la version du client, ceux du serveur, le poids du fichier, l'état de la requête (vous savez, 200 OK, 404 Not Found, 500 Internal Server Error...), etc., et du corps de la requête, qui contient, dans le cas d'une requête du serveur vers le client, le fichier envoyé. Ces deux blocs (en-tête et corps) sont séparés par une ligne vide. Pour plus d'informations, regardez donc [l'article sur Wikipedia](#).

Le type de fichier (que j'ai spécifié à la deuxième ligne avec *'Content-type: text/html'*), fait partie de l'en-tête de la requête. Or, la page que vous souhaitez envoyer se trouve dans le corps. C'est pourquoi vous devez les séparer à l'aide de cette ligne vide.

Votre script est maintenant prêt. Vous pouvez désormais écrire votre code HTML, toujours à l'aide de print :

```
#!/usr/bin/python

print 'Content-type: text/html'
print
print '<html><head><title>...'
```

Vous pouvez, plutôt que de vous esquinter à écrire ligne par ligne :

```
print '<html>'
print '<head>'
print '<title>Mon super site en Python qui powne tout XdXdDXDXd</title>'
print '</head>'
print '<body>'
# ... Je vous épargne la suite de cette horreur.
```

Utiliser la syntaxe suivante, qui n'est pas toujours connue :

```
print '''
<html>
    <head>
        <title>Mon super site en Python qui powne tout XdXDxDXDXd</title>
    </head>
    <body>
        <p>Vous êtes jaloux, hein ?</p>
    </body>
</html>
'''
```

Vous constatez qu'à l'intérieur des `'''`, vous pouvez utiliser l'indentation que vous voulez. Il est bien entendu important de rendre son site conforme aux normes de la W3C.

Si vous avez eu l'occasion de faire une erreur dans votre code, vous avez sûrement pu admirer une erreur 500. Et, vous l'avez remarqué, c'est très gênant pour le dépiage d'erreurs. Le module `cgitb` va pouvoir vous aider.

Pour cela, il faut l'importer, puis l'activer :

```
import cgitb
cgitb.enable()
```

C'est fait ! Maintenant, les erreurs s'afficheront sur la page. Ces deux lignes vous seront certainement utiles !

Mise en pratique : un compteur de visites

Maintenant que vous connaissez le strict *minimum*, pourquoi ne pas l'employer ? Nous allons réaliser un compteur de visites.

Il serait judicieux de tenter de le faire seul. Néanmoins, je vous donne ma méthode.

Tout d'abord, je tente d'ouvrir un fichier en mode lecture (`'r'` pour *read*). Je stocke son contenu dans la variable `nbr_visiteurs`, en le transformant en un nombre (car il s'agit pour l'instant d'une chaîne). Si cela ne marche pas (pour la bonne raison que mon fichier n'existe pas, ou que son contenu n'est pas un nombre), alors `nbr_visiteurs = 0`.

Cela donne :

```
try: # Pour essayer le code qui suit. Si ledit code ne fonctionne pas,
    alors except: est appelé. Autrement, on continue.
    fichier = open('compteur','r') # Lecture dans le fichier, appelé
    'compteur'.
    nbr_visiteurs = int(fichier.read()) # int() transforme son argument (à
    savoir, le contenu du fichier retourné par fichier.read()) en un nombre.
except Exception:
    nbr_visiteurs = 0 # Si le fichier est inexistant (= page jamais
    visitée) ou incorrect, on repart de 0.
```

Maintenant, nous allons ouvrir le fichier avec l'option `'w'` pour write. S'il existe, il sera supprimé et recréé vide. Sinon, il sera tout simplement créé. Dans les deux cas, nous n'aurons plus qu'à écrire dedans la valeur `nbr_visiteurs + 1` (car il faut compter le visiteur qui charge la page :-), en la transformant en une chaîne avec la fonction `str()`.

```
fichier = open('compteur','w')
fichier.write(str(nbr_visiteurs+1))
```


Finalement, on indique au visiteur le nombre de visites :

```
print nbr_visiteurs+1, 'visites \n'
```

Ce qui, avec le code minimal, donne :

```
#!/usr/bin/python

print 'Content-type: text/html'
print

try:
    fichier = open('compteur', 'r')
    nbr_visiteurs = int(fichier.read())
except Exception:
    nbr_visiteurs = 0
    fichier = open('compteur', 'w')
fichier.write(str(nbr_visiteurs+1))
print nbr_visiteurs+1, 'visites \n'
```

Traiter un formulaire

Un des points inévitables en CGI est le traitement de formulaires. En Python, c'est aussi faisable.

Nous travaillons maintenant avec des données envoyées par l'utilisateur. Nous allons notamment les afficher à l'écran. Or, si le visiteur décide de mettre du HTML dans les données qu'il envoie, cela peut présenter une faille.

Prenons un cas concret. Si le visiteur entre le code suivant :

```
<script type="text/javascript">alert('LOLOL CMT G T PWNEED');</script>
```

Une alerte s'affichera sur l'écran du visiteur ! Et rien ne l'empêche de se faire passer pour vous « Votre session va expirer. Entrez votre mot de passe à nouveau. », de le récupérer puis de se le faire envoyer par mail, le tout en JavaScript !

Pour pallier ce problème, nous allons utiliser une fonction fort pratique : *cgi.escape()*. C'est un peu l'équivalent de *htmlspecialchars()* en PHP : elle remplace les caractères < et > par **<** et **>**. De ce fait, le code JavaScript cité plus haut s'affichera tel quel, au lieu d'être exécuté. Notez que si vous utilisez une version de python supérieure à la 3.2 il vous faudra utiliser *html.escape()* à la place de *cgi.escape()*, car cette dernière est dépréciée.

Aussi, **prenez l'habitude de sécuriser toutes les données envoyées par le visiteur à l'aide de cette fonction.**

Pour traiter un formulaire, nous devons tout d'abord importer le module CGI. Vous savez comment faire :

```
import cgi
```

À partir de là, ça se corse. 🤖 On va créer une instance de la classe *cgi.FieldStorage()*, qui contiendra le formulaire envoyé. Puis, on pourra en extraire les informations avec *instance.getvalue('nom')*. Cela vous semble tordu ? En vérité, c'est très simple.

```
#!/usr/bin/python

import cgi

print 'Content-type: text/html'
print
formulaire = cgi.FieldStorage()
if formulaire.getvalue('nom') == None:
    print '''
Veuillez remplir le formulaire :
<form action="formulaire.py" method="post">
<input type="text" name="nom" />
<input type="submit"></form>
'''
else:
    print 'Ainsi, vous vous
appelez',cgi.escape(formulaire.getvalue('nom')), ' ?' # N'oubliez pas de
sécuriser le code !
```

Ce code, enregistré dans un fichier *formulaire.py* (autrement, il vous faudra le modifier, car le formulaire renvoie vers *formulaire.py*), créera une instance de *cgi.FieldStorage()* dans la variable *formulaire*.

Si *formulaire.getvalue('nom') == None*, alors le champ '*nom*' était vide, voire non envoyé. On présente donc le formulaire au client. Autrement, on récupère la valeur du champ '*nom*' avec *formulaire.getvalue('nom')*, et on l'affiche tout en prenant soin de le sécuriser avec *cgi.escape()*.

Si le champ s'était appelé '*prenom*', vous comprenez bien que l'on aurait récupéré sa valeur avec *formulaire.getvalue('prenom')*.

Vous voulez une bonne nouvelle ? Pour un formulaire envoyé avec GET (c'est-à-dire que les valeurs des champs se retrouvent dans l'URL, dans le style

page.py?var=valeur&var2=valeur2), c'est exactement pareil.

Ainsi, le script suivant enregistré sous *essai.py* et appelé avec *essai.py?var=1&var2=2* affichera 'Var = 1 et Var2 = 2'.

```
#!/usr/bin/python

import cgi

print 'Content-type: text/html'
print

form = cgi.FieldStorage()
print 'Var = ',cgi.escape(form.getvalue('var')), ' et Var2 = ',cgi.escape(form.getvalue('var2'))
```

Mise en pratique : un livre d'or

Le livre d'or est une des fonctionnalités que l'on se plaît à implémenter sur son site. Grâce aux formulaires et à la manipulation de fichiers, c'est tout à fait faisable en Python.

Nous allons tout d'abord créer un formulaire tout simple, puis le traiter. Ensuite, nous verrons comment lire le fichier dans lequel seront enregistrés les messages.

Le formulaire ressemblera à ça :

```
<form action="enregistrement.py" method="post">
```

```

    <p>Vous avez une remarque, un commentaire, un conseil ? Signez le livre
d'or !</p>
    <p>
        Pseudonyme : <input type="text" name="pseudo" /><br />
        Site web : <input type="text" name="site" /><br />
        Message :<br />
        <textarea name="message" cols="20" rows="2"></textarea><br />
        <input type="submit" />
    </p>
</form>

```

À vous de le faire correspondre à vos besoins.
Pour la gestion du formulaire :

```

#!/usr/bin/python

import cgi

print 'Content-type: text/html'
print

print '<html><head><title>Mon super site</title></head><body>'
formulaire = cgi.FieldStorage()
if formulaire.getvalue('message') != None:
    print '<p>Merci d\'avoir particip&eacute; au livre d\'or. Vous
pouvez le visiter <a href="livreor.py">ici</a>.</p>'
    message = formulaire.getvalue('message')
    site = formulaire.getvalue('site')
    pseudo = formulaire.getvalue('pseudo')
    try:
        fichier = open('livreor', 'r')
        livreor = fichier.read()
    except IOError:
        livreor = ''
    message_poste =
'<message><auteur>'+cgi.escape(pseudo)+'</auteur><site>'+cgi.escape(site)+'
</site><contenu>'+cgi.escape(message)+'</contenu></message>\n'
    fichier = open('livreor', 'w')
    fichier.write(message_poste+livreor)
else:
    print '''Erreur : vous n'avez pas rempli le formulaire.'''
print '</body></html>'

```

C'est tout simple :

- on charge le formulaire dans la variable *formulaire* avec *cgi.FieldStorage* ;
- on teste si le formulaire a été rempli, en vérifiant qu'un message est présent. Vous pouvez bien sûr vérifier si tous les champs sont remplis ;
- on charge les éléments du formulaire dans des variables ;
- on essaie de lire le contenu du fichier *livreor*. Si cela rate (avec une *IOError*), c'est que le fichier n'existe pas, auquel cas le contenu de *livreor* sera une chaîne vide ;
- on ouvre le fichier *livreor* en mode écriture. On écrit dedans le contenu du nouveau message.

L'écriture est terminée. Chaque message est écrit dans le format :

```
<message><auteur>Auteur</auteur><site>Site</site><contenu>Contenu</contenu>
</message>
```

Nous n'avons plus qu'à le lire !

```
#!/usr/bin/python

def trouver(chaine, chaine1, chaine2):
    position_chaine1 = chaine.index(chaine1) + len(chaine1)
    position_chaine2 = chaine.index(chaine2, position_chaine1)
    return chaine[position_chaine1:position_chaine2]

def isoler_message(chaine):
    dico = {}
    try:
        message = trouver(chaine, '<message>', '</message>')
        dico['auteur'] = trouver(message, '<auteur>', '</auteur>')
        dico['site'] = trouver(message, '<site>', '</site>')
        dico['contenu'] = trouver(message, '<contenu>', '</contenu>')
        return dico
    except Exception:
        return 0

def supprimer(chaine):
    return
chaine.replace('<message>'+trouver(chaine, '<message>', '</message>')+'</message>', '')

print 'Content-type: text/html'
print
print '''
<html>
<head>
<title>Le livre d'or</title>
</head>
<body>
'''
try:
    fichier = open('livreor', 'r')
    contenu = fichier.read()
    continuer = True
    while continuer:
        message = isoler_message(contenu)
        if not message:
            continuer = False
        else:
            print '<p>De : <b><a
href="', message['site'], '">', message['auteur'], '</a></b></p>'
            print '<p>', message['contenu'], '</p>'
            print '<br /><br />'
            contenu = supprimer(contenu)
except Exception:
    print 'Il n\'y a aucun message dans le livre d\'or'
print '</body></html>'
```

Ma première fonction, *trouver()*, retourne la première chaîne trouvée entre *chaine1* et *chaine2* dans *chaine*. Son fonctionnement est simple, lisez le code si vous souhaitez le comprendre.

La seconde, *isoler_message()*, utilise *trouver()* pour trouver le premier message (la première chaîne entre *<message>* et *</message>*), puis extrait les différentes informations (l'auteur, le site et le contenu). Finalement, elle retourne le dictionnaire dans lequel elle a tout enregistré.

La troisième, *supprimer(chaine)*, supprime le premier message (le texte entre *<message>* et *</message>*, ainsi que les deux balises), pour que la lecture puisse être recommencée sur le message suivant.

On commence par afficher le début du HTML de la page. Puis, on essaie de lire le fichier *livreor*, puis d'isoler le premier message, de l'afficher, et de le supprimer de la liste, avant de recommencer, jusqu'à ce qu'il n'y ait plus de messages.

Si cela ne fonctionne pas, on affiche qu'il n'y a pas de message dans le livre d'or.

C'est tout à fait rudimentaire. À vous d'ajouter ce que vous voulez : datez les messages, faites des pages lors de l'affichage...

Vous voyez que, en travaillant des fichiers et en traitant des formulaires, vous pouvez déjà aller loin. Légèrement modifié, ce livre d'or peut vous fournir un système de news.

Afficher son code source

Une dernière astuce dont j'aimerais vous faire part est l'affichage de la source. En effet, il est agréable de pouvoir avoir le code d'un site web sous les yeux, pour pouvoir s'en inspirer, ou même donner des conseils au webmaster.

Ce genre de chose ne doit PAS être implémenté sur des pages sensibles comme l'administration, à moins que vous ne soyez absolument certain(e) de la sécurité de votre code.

Pour afficher la source, le client n'aura qu'à entrer l'URL de la page ainsi :

<http://site.web/index.py?source=1>

Il nous faudra donc utiliser *cgi.FieldStorage()*, et tester si *source=1*. Dans ce cas, on lit le fichier, et on l'affiche entre *<pre>* et *</pre>*, pour que le code HTML et les sauts à la ligne apparaissent. Puis, on s'arrête là avec *sys.exit(0)*.

Autrement, on envoie la page comme d'habitude.

Voici mon code :

```
#!/usr/bin/python

import sys
import cgi

form = cgi.FieldStorage()
if form.getvalue('source') == '1':
    print """Content-type: text/plain
"""
    print open('livreor.py').read()
    sys.exit(0)

print """Content-type: text/html
"""
# Suite du code
```

C'est enfantin mais pratique. 😊

Ce tutoriel n'est ici que pour solliciter votre enthousiasme. Pour pouvoir mettre en place un site web plus complexe, utilisant une base de données par exemple, vous devez vous tourner vers la documentation.

J'espère vous avoir intéressés et donné des idées.

Bonne continuation avec Python !

Merci à delroth et plus généralement à tout #python pour l'aide qui m'a été apportée lors de la création de mon propre site, et sans qui ce tutoriel n'aurait pas vu le jour.

Laurent Tichit Programmation ???

<http://www.dil.univ-mrs.fr/~tichit/web/>

1. Attention !! L'interpréteur Python3 installé sur le serveur Web possède un bug : les scripts CGI écrits en python3 ne fonctionneront pas s'ils contiennent des caractères accentués.

Deux solutions pour cette année :

1. enlever les caractères accentués (write in english for instance :))
2. basculer vers python2 (remplacer `#!/usr/bin/python3` par `#!/usr/bin/python`).

Préférez la première solution, ça vous évitera de gérer les incompatibilités python2/python3

2. Attention aux retours à la ligne différents entre Windows et le reste du monde. Les éditeurs de texte gedit ... enregistrer sous ou (sous Windows) Notepad++ peuvent détecter le type de retour à la ligne et passer de l'un à l'autre. Si vous voyez le caractère ^M, c'est que vous avez des retours à la ligne à la Windows. Votre script ne fonctionnera pas sous Linux ! La commande UNIX `dos2unix monScript.py` permet de convertir les retours à la ligne.
3. Dans les premières versions des fichiers `upload.py` et `uploadCreate.py`, il faut remplacer la fonction `file()` par un classique `open()`. En effet, `file()` était toléré jusqu'à python3.1. Il ne l'est plus depuis python3.2.

CGI / Python

- Le cours sur [les formulaires](#)
- Le [sujet](#) du TP sur les CGI et les formulaires
 - Partie 1 - les scripts à télécharger : [cgi](#)
 - Partie 2 - le formulaire HTML à télécharger : [form.html](#)
 - Partie 4 :
 - les formulaires HTML à télécharger : [forms](#)
 - les scripts à télécharger : [cgi2](#)
 - Autres scripts intéressants : [cgi3](#)
- Comment donner à Apache la permission de créer des fichiers dans votre `public_html` ?
 - Intérêts :
 - Pour uploader un fichier, et que ce fichier soit accessible ultérieurement.

- Si l'un de vos scripts crée dynamiquement un fichier (une image grâce à [matplotlib](#), un fichier texte résultat, etc.), sur lequel on mettra un lien hypertexte.
 - Créez un sous-répertoire `download` dans votre répertoire web, grâce à une commande du style : `mkdir ~/public_html/download`
 - Donnez les permissions à tous (dont Apache) de créer des fichiers dans ce répertoire : `chmod a+w ~/public_html/download`
 - Notre script ouvrira un fichier en écriture, de nom "`../download/arbre.nwk`" (votre script étant dans `~/public_html/cgi-bin`), puis créera dans la page HTML résultant un lien (`` pour permettre à l'utilisateur de le télécharger.
 - Problème : si 1000 utilisateurs se connectent et créent leur propre fichier résultat, il faudrait pour chacun d'eux des noms différents. Pour ça, créez des nombres aléatoires (en utilisant `random`, `rand`, `randint`, etc.) et nommez vos fichiers de la façon suivante par exemple : `protein134581356.fasta`.
 - La façon propre est d'utiliser les *sessions* (voir cours de Magali Contensin).
 - Il faudra aussi trouver un moyen de supprimer ces fichiers créés sinon votre disque dur finira par déborder.
 - Cette partie est également valable pour PHP (en l'occurrence, vous verrez les sessions en PHP).
- **Travail à rendre pour le dimanche 9 février 23h :**
 - Pour ceux ayant fait le projet PS1 : votre projet de PS1 porté en application web avec python-CGI.
 - Pour les autres, un projet conséquent en CGI, en sachant que le support CGI pour Perl, Python2, Python3, Shell, et langages compilés (C, C++, Fortran) est installé.

Évaluation :

- Utilisez les validateurs HTML et CSS (w3.org) : pas conforme au standard XHTML 1.0 Strict ou HTML5 = pas la moyenne. Je ne veux aucune erreur ; les warnings sont tolérés.
- Le script CGI doit fonctionner et ne pas générer d'exceptions (testez le avant, hors-ligne == en ligne de commande, puis avec `cgitb` activé).
- A l'instar de TradConcept, faites une vraie séparation entre le CSS (aspect visuel) et HTML (contenu). Je visualiserai avec et sans CSS.
- Vous devez tester sur les machines du CRFB. Si ça ne passe pas sur celles-ci : pas la moyenne.
- Bien entendu il faut quand même un code conséquent. Si votre code est trop simpliste, creusez-vous un peu la tête pour faire quelque chose d'un peu sympa côté web.

Liens utiles CGI

Cours Python et cours CGI :

- La page de Henri Garreta consacrée à [Python BBSG](#).

- Le chapitre [21.2](#) de la Python Library Reference.
- [Site](#) sur les CGI en langage C, ainsi qu'un exemple modifié tiré de ce cours : [formulaire](#) et [CGI](#)

Divers

Deux petites choses rigolotes en Python :

- Lancer le navigateur web par défaut à partir de python : [browser.py](#)
- Lancer un mini serveur web qui publie les fichiers du répertoire courant sur internet : [webServer.py](#). Pour tester le serveur, exécuter le script, puis ouvrez un navigateur web à l'adresse : localhost:8987

Browser.py

```
#!/usr/bin/python

import webbrowser
webbrowser.open('http://python.org')
```

webServer.py

```
#!/usr/bin/python

import BaseHTTPServer, SimpleHTTPServer

server = BaseHTTPServer.HTTPServer(('', 8987),
SimpleHTTPServer.SimpleHTTPRequestHandler)
server.serve_forever()
```

Module Flask

<https://www.hackster.io/sankarCheppali/setting-up-python-web-server-for-raspberry-pi-260deb>

<https://www.raspberrypi.org/learning/python-web-server-with-flask/>

<http://www.instructables.com/id/Python-Web-Server-for-your-Raspberry-Pi/>

contrôle GPIO <http://randomnerdtutorials.com/raspberry-pi-web-server-using-flask-to-control-gpios/>