COMP519
Web Programming
Autumn 2015
CGI Programming
These lectures notes are designed to:

- Teach you how to use CGI in server-side programming
- Use environmental variables in Python
- Access the input from a form with "post" or "get" methods
- Make a basic counter for your web page
What is CGI?

The **Common Gateway Interface** (CGI) is a standard (protocol) for interfacing external applications with information servers, such as HTTP or Web servers. A CGI program is executed in real-time, so that it can output dynamic information.

The Web server can call up a program, while passing user-specific data to the program.

The program then processes that data and the server passes the program’s response back to the Web browser.
What is CGI? (cont.)

CGI can be used for various web programming applications and in various contexts.

**Forms**
Shopping, Booking

**Gateways**
Search Engine, Database

**Virtual Documents**
Guestbook, Chat, Bulletin Board, Dictionary
Internal Workings of CGI

How does this process work? (What I describe can differ from web server to web server, I specifically describe what is needed for this to work on the CS Department server.)

Idea 1. CGI programs:
▶ A special directory (e.g. http://cgi.csc.liv.ac.uk/cgi-bin/cgiwrap/martin/script)
▶ A certain file extension(????) Certain versions of browsers seem to require a certain file extension on the scripts to work properly. Having said that, I have experienced no problems with recent testing of Python scripts with the "py" extension.

Idea 2. HTTP requests (get, post):
▶ Specify URL
▶ Specify protocol (e.g. HTTP/1.1)
▶ Specify accept formats
▶ Specify user-agent
▶ Specify user's data
Internal Workings of CGI

How does this process work? (What I describe can differ from web server to web server, I specifically describe what is needed for this to work on the CS Department server.)

Idea 1. CGI programs:

- A special directory (e.g. http://cgi.csc.liv.ac.uk/cgi-bin/cgiwrap/martin/script)
- A certain file extension(?????) Certain versions of browsers seem to require a certain file extension on the scripts to work properly. Having said that, I have experienced no problems with recent testing of Python scripts with the ".py" extension.
Internal Workings of CGI

How does this process work? (What I describe can differ from web server to web server, I specifically describe what is needed for this to work on the CS Department server.)

Idea 1. CGI programs:

▸ A special directory (e.g. http://cgi.csc.liv.ac.uk/cgi-bin/cgiwrap/martin/script)
▸ A certain file extension(????) Certain versions of browsers seem to require a certain file extension on the scripts to work properly. Having said that, I have experienced no problems with recent testing of Python scripts with the ".py" extension.

Idea 2. HTTP requests (get, post):

▸ Specify URL
▸ Specify protocol (e.g. HTTP/1.1)
▸ Specify accept formats
▸ Specify user-agent
▸ Specify user’s data
Internal Workings of CGI (cont.)

Idea 3. Web servers:

- Recognize it is a CGI program
- Execute the program
- Makes user’s info available to the CGI program along with some server info
Internal Workings of CGI (cont.)

Idea 3. Web servers:
- Recognize it is a CGI program
- Execute the program
- Makes user’s info available to the CGI program along with some server info

Idea 4. Input to CGI program:
- From user (in Python, easiest to utilize the `cgi` module)
- From Unix environmental variables (`os.environ` “dictionary” in Python)
Internal Workings of CGI (cont.)

Idea 3. Web servers:
- Recognize it is a CGI program
- Execute the program
- Makes user’s info available to the CGI program along with some server info

Idea 4. Input to CGI program:
- From user (in Python, easiest to utilize the cgi module)
- From Unix environmental variables (os.environ “dictionary” in Python)

Idea 5. Output from CGI program:
- Generally this will be in the form of a webpage.
  HTTP header + a blank line + HTML webpage
Configuring Everything...

Before you can run CGI programs on your server, certain parameters in the server configuration files must be modified (this has been done already for us).

For the UoL server you should proceed as follows. CGI scripts must go in the directory HOME/public_html/cgi-bin/ (so mine are in martin/public_html/cgi-bin/) and are accessed using the URL: http://cgi.csc.liv.ac.uk/cgi-bin/cgiwrap/{user}/{script}

If the scripts don't seem to work, try replacing "cgiwrap" with "cgiwrapd" to see debugging output.

(I believe that your cgi-bin directory and scripts must be executable by yourself (and possibly everyone) in order to be executable by the web server.)
Configuring Everything...

Before you can run CGI programs on your server, certain parameters in the server configuration files must be modified (this has been done already for us).

For the UoL server you should proceed as follows.

**CGI scripts must go in the directory**

HOME/public_html/cgi-bin/ (so mine are in martin/public_html/cgi-bin/)

and are accessed using the URL:

http://cgi.csc.liv.ac.uk/cgi-bin/cgiwrap/{user}/{script}
Configuring Everything...

Before you can run CGI programs on your server, certain parameters in the server configuration files must be modified (this has been done already for us).

For the UoL server you should proceed as follows.

**CGI scripts must go in the directory**

HOME/public_html/cgi-bin/ (so mine are in martin/public_html/cgi-bin/)

and are accessed using the URL:

http://cgi.csc.liv.ac.uk/cgi-bin/cgiwrap/{user}/{script}

If the scripts don’t seem to work, try replacing "cgiwrap" with "cgiwrapd"

to see debugging output.

(I believe that your cgi-bin directory and scripts must be executable by yourself (and possibly everyone) in order to be executable by the web server.)
Programming in CGI

Now that I know how CGI works, what programming language can I use?

You can use whatever language you want, but it should ideally include:

▶ Ability to access environmental variables (in UNIX)
▶ Ability to interface with other software libraries and utilities
▶ Ease of text manipulation

Perl, Python, and Ruby are widely used for CGI programming!

▶ Highly portable and readily available
▶ Powerful string manipulation operators (especially Perl)
▶ Very simple and concise constructs
▶ Calling shell commands, and useful equivalents of certain UNIX system functions
▶ Extensions on top of Python for specialized functions
Programming in CGI

Now that I know how CGI works, what programming language can I use?

You can use whatever language you want, but it should ideally include

- Ability to access environmental variables (in UNIX)
- Ability to interface with other software libraries and utilities
- Ease of text manipulation

Perl, Python, and Ruby are widely used for CGI programming!

- Highly portable and readily available
- Powerful string manipulation operators (especially Perl)
- Very simple and concise constructs
- Calling shell commands, and useful equivalents of certain UNIX system functions
- Extensions on top of Python for specialized functions
Programming in CGI

Now that I know how CGI works, what programming language can I use?

You can use whatever language you want, but it should ideally include

▶ Ability to access environmental variables (in UNIX)
▶ Ability to interface with other software libraries and utilities
▶ Ease of text manipulation

Perl, Python, and Ruby are widely used for CGI programming!

▶ Highly portable and readily available
▶ Powerful string manipulation operators (especially Perl)
▶ Very simple and concise constructs
▶ Calling shell commands, and useful equivalents of certain UNIX system functions
▶ Extensions on top of Python for specialized functions
The CGI input information can be roughly broken into three groups:

Information about the client, server, and user
  ▶ available in CGI environmental variables

See a trend here? Environmental variables are available in Python through the use of the os module.
The CGI input information can be roughly broken into three groups:

- Information about the client, server, and user
  - available in CGI environmental variables

- Form data that the user supplied
  - available either in a CGI environmental variable, or
  - via the Python `cgi` module
The CGI input information can be roughly broken into three groups:

- Information about the client, server, and user
  - available in CGI environmental variables

- Form data that the user supplied
  - available either in a CGI environmental variable, or
  - via the Python `cgi` module

- Additional pathname information
  - available in CGI environmental variables

See a trend here? Environmental variables are available in Python through the use of the `os` module.
The CGI input information can be roughly broken into three groups:

Information about the client, server, and user
  ▶ available in CGI environmental variables

Form data that the user supplied
  ▶ available either in a CGI environmental variable, or
  ▶ via the Python `cgi` module

Additional pathname information
  ▶ available in CGI environmental variables

See a trend here?
CGI input

The CGI input information can be roughly broken into three groups:

Information about the client, server, and user
   ▶ available in CGI environmental variables

Form data that the user supplied
   ▶ available either in a CGI environmental variable, or
   ▶ via the Python cgi module

Additional pathname information
   ▶ available in CGI environmental variables

See a trend here?

Environmental variables are available in Python through the use of the os module.
Here is a webpage to show the environmental variables available to a Python web script.

```python
#!/usr/bin/python
#    pCGI02.py
import os

def start_html(title):
    print "Content-type: text/html\n"
    print "<!DOCTYPE html>\n"
    print "<html>\n<meta charset="utf-8">"
    print "<html><head><title>" + title + "</title><head><head>\n"
    print "<body>"

def end_html():
    print "</body>\n</html>"

start_html("Environmental variables")
print "<h1>Unix environmental variables</h1>"
for k in sorted(os.environ.keys()):
    print "<p>" + k + " = " + os.environ[k] + "</p>"
end_html()
```
Checking the Client’s Browser

You can check the client’s operating system and/or browser type, and then display specific content based on that information.

How can you do this?

1. Get the name of the browser from the HTTP_USER_AGENT environmental variable.
2. Identify the operating system using a regular expression (for example).
3. Print the appropriate content.

You could, for example, even send back different (HTML) files based on the information obtained about the client.
#!/usr/bin/python
# pCGI03.py COMP519
import os
import re
def start_html(title):
    .......

def end_html():
    .......

start_html("Who are you?")
print "<h1>I see you!</h1>"

user = os.environ["HTTP_USER_AGENT"]
print "<p>HTTP_USER_AGENT: " + user + "</p>"

if re.search("Windows",user):
    print "<p>Are you a friend of Microsoft?</p>"
elif re.search("Macintosh|iPad|iPhone",user):
    print "<p>Hello, Apple user.</p>"
else:
    print "<p>Hello, stranger!</p>"
end_html()
Restricting Access for Specified Domains

You can restrict access to your documents based on the client’s domain.

How can you do this?

1. Find the IP (numerical) address for the remote user. Use the REMOTE_ADDR environmental variable.
2. Convert the IP address to its corresponding domain name. (You can use a function in the Python module called socket.)
3. Check that the domain name is “allowed”.
4. Reply to the user as appropriate.
#!/usr/bin/python
# pCGI04.py COMP519
import os
import socket
import re
from utility import start_html, end_html # external file

start_html("Where are you coming from?")
print "<h2>Where are you coming from?</h2>"

user_ip = os.environ["REMOTE_ADDR"]
try:
    host = socket.gethostbyaddr(user_ip)[0] # socket module
except:
    host = "Not resolvable"

print "<p>Client’s IP address: " + user_ip + "</p>"
print "<p>Client’s hostname: " + host + "</p>"
if re.search("liv.ac.uk", host):
    print "<p>Welcome, university user!</p>"
else:
    print "<p><b>Sorry, please come back when you are on a university machine!</b></p>"

end_html()}
Where Did You Come From?

You can get information from where a remote user came from. This utilizes the **HTTP_REFERER** environmental variable.

Note that not all browsers set this variable.

If a user accesses your server by using a bookmark or just typing in the URL, this environmental variable may not be set, or may be meaningless.
#!/usr/bin/python
# pCGI05.py COMP519
import os
import re
from utility import start_html, end_html
start_html("Where did you come from?"

print "<h1>I see (some other things about you) you!</h1>"

user_ip = os.environ["REMOTE_ADDR"]
print "<p>Hello user from IP address: ", user_ip, "</p>"

if "HTTP_REFERER" in os.environ.keys():
    print "<p>The last site you visited was ", os.environ["HTTP_REFERER"
else:
    print "<p>I don’t know the last site you visited.</p>"

print "<p>Go <a href="http://cgi.csc.liv.ac.uk/~martin/test-ref.html" to test this out.""}
Accessing Form Input

A CGI program can access form input from a user.
Accessing Form Input

The main idea is that an HTML form has an action property that can refer to a CGI script. When the form is submitted, the input on the form is packaged up (in an appropriate way) and sent to the web server, with a request to execute the CGI script.

The Python `cgi` module provides objects that give easy access to the information that has been submitted.

Accessing this information is done in slightly different manners, depending upon the method that the information is sent to the server.

The two main methods are “get” and “post”.

Also, information in a query string can be accessed either directly through an environmental variable, or using the `cgi` module.
Query Strings

Information can be appended to the URL directly, after a question mark.

```python
#!/usr/bin/python
# pCGI06.py COMP519
import os
from utility import start_html, end_html
start_html("Finding a query string")
print "<h1>I see (something else about you) you!</h1>"

query_string = os.environ["QUERY_STRING"]

if query_string == "fortune":
    print "<h2>FORTUNE</h2>"
elif query_string == "finger":
    print "<h2>POKE!</h2>"
else:
    print "<h2>Blah, blah, blah</h2>"

end_html()
```

http://cgi.csc.liv.ac.uk/cgi-bin/cgiwrap/martin/pCGI06.py?fortune
http://cgi.csc.liv.ac.uk/cgi-bin/cgiwrap/martin/pCGI06.py?stuff
A Simple Form (Using “get”)

A form can solicit information from the user. The **action** on the form specifies a script to execute on the server. The **method** specifies the method to send the information.

```html
<!DOCTYPE html>
<html>
<!-- COMP519 form01-CGI.html 2015/11/10 -->
<head>
<title>Simple Form!</title>
<meta charset="utf-8">
</head>
<body>
<h1>Simple Form!</h1>
<hr>
<form method = "get"
action="http://cgi.csc.liv.ac.uk/cgi-bin/cgiwrap/martin/pCGI07.py">
<p>
Command: <input type="text" name="command" size="40"> <br> <br>
<input type="submit" value="Submit Form!"> &nbsp;&nbsp;&nbsp;&nbsp;
<input type="reset" value="Clear Form">
</p>
</form>
<hr>
</body> </html>
```
Manipulating the Query String

#!/usr/bin/python
#    pCGI07.py COMP519
import os
import re
from utility import start_html, end_html

start_html("Using the query string")

query_string = os.environ[‘QUERY_STRING’]
print "<p>The raw query string is <b>" + query_string + "</b></p>"

if re.search("fortune", query_string):
    print ‘<p>GOLD!!!</p>’
elif re.search("finger", query_string):
    print ‘<p>POKE!</p>’
else:
    print ‘<p>How dull...</p>’

end_html()

http://cgi.csc.liv.ac.uk/ martin/teaching/comp519/PYTHON/form01-CGI.html
Manipulating the Query String (cont.)

Note how the name of the input field is combined with the value the user inputs to form the query string.
Manipulating the Query String (cont.)

Note how the name of the input field is combined with the value the user inputs to form the query string.

If there is more than one input value submitted, each name,value pair is formed in a similar fashion, and these pairs are then concatenated, separated by & symbols, to form the full query string.
Note how the name of the input field is combined with the value the user inputs to form the query string.

If there is more than one input value submitted, each name,value pair is formed in a similar fashion, and these pairs are then concatenated, separated by & symbols, to form the full query string.

To recover the user input values, you would have to process the string, for example, splitting on the & symbols, then splitting on the = signs to get the values.
Manipulating the Query String (cont.)

Note how the **name** of the input field is combined with the **value** the user inputs to form the query string.

If there is more than one input value submitted, each **name,value** pair is formed in a similar fashion, and these pairs are then concatenated, separated by & symbols, to form the full query string.

To recover the user input values, you would have to process the string, for example, splitting on the & symbols, then splitting on the = signs to get the values.

This is where Python string methods and/or regular expressions can help out.

Alternatively, you can use the Python **cgi** module.
Using the `cgi` module

The Python `cgi` module provides a method for getting all user input, regardless of whether the “get” or “post” method was used, and regardless of whether it was submitted via a form, or as a direct link to a URL.

We modify the `action` of the previous page to point to a new Python script (and that is the only change necessary there).

We then use the `cgi.FieldStorage()` constructor. This returns a `FieldStorage` object that contains all name-value pairs provided as input to your script.
Using the **cgi** module (cont.)

If, say, \( f \) is a FieldStorage instance, you can use this method:

\[
\text{f.getvalue(cName, default=None)}
\]

If \( f \) has one value for the control name cName, this method returns that value as a string. If there are multiple values, the method returns them as a list of strings.

If \( f \) has no values for the control name cName, it returns the default value (or \texttt{None} if no default value is given).

It is standard practice to test whether something has been assigned to an input (e.g. typically testing whether the value is not \texttt{None}), as trying to use string methods to compare, test, or extract some part of it will result in an error (which could be trapped as an exception if desired).
Changing the Script to use FieldStorage

We modify the Python script to the following:

```python
#!/usr/bin/python
# pCGI07-alt.py COMP519
import os
import re
import cgi
from utility import start_html, end_html
start_html("Using the query string")

form = cgi.FieldStorage()
command = form.getvalue('command')
print "<p>The raw query string is <b>" + os.environ[‘QUERY_STRING’] + "</b></p>
if command:
    print "<p>The command extracted through FieldStorage is <b>" + command + "</b></p>
    if re.search("fortune", command):
        print ' <p>GOLD!!!</p>'
    elif re.search("finger", command):
        print ' <p>POKE!</p>'
    else:
        print ' <p>How dull...</p>'
else:
    print "<p>Nothing was entered in the input!</p>
end_html()
```
“get” and “post” are the two methods for passing data to the web server.

The essential differences are:

▶ “get” encodes the data in the URL (accessible via the query string), while “post” sends the data to standard input in the server (available by reading from standard input).

▶ There is a limit of 1024 characters using the “get” method. With “post” there is no limit (in principle).
Using the Python `FieldStorage` object (in the `cgi` module), there is no real difference between the “get” and “post” methods in terms of accessing the data.

The basic procedure is to initialize a `FieldStorage` object via the constructor, and then can query that object (via its method) to obtain the user input by passing the method the (HTML) name that was assigned to a particular input field, textarea, checkboxes, set of radio buttons, etc.
Another form example

Here is another small example of a form, using the “post” method.

```html
<!DOCTYPE html>
<html>
<!-- COMP519  form02-CGI.html  2015/11/10 -->
<head>
  <title>When’s your birthday?</title>
  <meta charset="utf-8">
</head>
<body>
<h1>What are your name and birthday?</h1>
<form action="http://cgi.csc.liv.ac.uk/cgi-bin/cgiwrap/martin/pCGI08.py" method="post">
  <p>
    Name (Given Name):
    <input type="text" name="id" size="40">
    <br>
    Birthday (mm/dd/yyyy):
    <input type="text" name="birthday" size="40">
    <br>
    <input type="submit" value="Submit Form!">
    &nbsp;&nbsp;&nbsp;&nbsp;
    <input type="reset" value="Clear Form">
  </p>
</form>
</body>
</html>
```
#!/usr/bin/python
# pCGI08.py COMP519
import cgi
import datetime
from utility import start_html, end_html
start_html("About the User")
form = cgi.FieldStorage()  # initialize the "data store"

if "id" in form and "birthday" in form:
    first = form["id"].value.strip()  # call "strip" fcn on string
    print "<h1>Thank you " + first + "!</h1>"

    birthday = form["birthday"].value.strip()
    d = birthday.split('/')
    d = [int(x) for x in d]
    bd = datetime.date(d[2], d[0], d[1])
    today = datetime.date.today()
    days_old = today - bd
    print "<h2>Your birthday is " + bd.strftime("%d %B %Y") + ".</h2>"
    print "<h3>You are " + str(days_old.days) + " days old."</h3>"
else:
    print "<h2>Oops, some value is missing from your form! Go back and retry!</h2>"
end_html()
Other HTML input elements

```html
<!DOCTYPE html>
<html>
<!-- COMP519  form03-CGI.html  2015/11/16 -->

<head>
  <title>When’s your birthday?</title>
  <meta charset="utf-8">
</head>

<body>
<h1>What are your name and birthday?  (Version 2)</h1>
<hr>
<form action="http://cgi.csc.liv.ac.uk/cgi-bin/cgiwrap/martin/pCGI09.py"
  method="post">
  <p>
    Name (Given Name):
    <input type="text" name="id" size="40">
    <br><br>
    Birthday:  <br>
  </p>
</form>
</body>
</html>
```
Other HTML input elements (cont.)

```html
<div style="float:left; padding:0 4em 0 0">
<span style="text-decoration: underline">Month</span> <br>
<input type="radio" name="month" value="1" id="jan">
<label for="jan">January</label> <br>

......
</div>

<div style="float: left; padding:0 4em 0 0">
<span style="text-decoration: underline">Date</span><br>
<input type="text" name="date" size="10">
</div>

<div style="clear: right">
<span style="text-decoration: underline">Year</span><br>
<select name="year">
<script>
for (var i = 2015; i >= 1900; i--)
    { document.write(’<option value="’+i+’’>'+i+'</option>’); } </script>
</select><br>
</div>

<div style="clear: both"> 
<input style="clear: both" type="submit" value="Submit Form!">
&nbsp;&nbsp;&nbsp;&nbsp; <input type="reset" value="Clear Form">
</div></form> </body> </html>
```
The altered Python script

```python
#!/usr/bin/python
# pCGI09.py COMP519
import cgi
import datetime
from utility import start_html, end_html
start_html("About the User")
form = cgi.FieldStorage()
if ("id" in form and "month" in form and "date" in form
    and "year" in form):
    first = form["id"].value.strip()  # call "strip" fcn on string
    print "<h1>Thank you " + first + "!\</h1>"

    month = int(form.getvalue("month"))
    date = int(form.getvalue("date"))
    year = int(form.getvalue("year"))
    bd = datetime.date(year, month, date)
    today = datetime.date.today()
    days_old = today - bd

    print "<h2>Your birthday is " + bd.strftime("%d %B %Y") + ".\</h2>"
    print "<h3>You are " + str(days_old.days) + " days old."\</h3>"
else:
    print "<h2>Oops, some value is missing from your form! Go back and retry!\</h2>"
end_html()
```
Output from CGI

The most basic output for a CGI program is a simple document in either plain text or HTML, which the browser displays as it would any document on the Web. However, there are other things you can do, such as:

▶ Return graphics and other binary data.
▶ Tell the browser whether to cache the virtual document.
▶ Send special HTTP status codes to the browser.
▶ Tell the server to send an existing document or other file to the client.

Each of these techniques involves knowing a little bit about returning additional types of headers from the CGI program (which I am not planning on talking about here).
Extra Path Information

Another way we can pass additional information as part of the URL is to give extra path information. As might be expected, this information will show up in an environmental variable.

http://cgi.csc.liv.ac.uk/cgi-bin/cgiwrap/martin/pCGI10.py/data/text.dat

```python
#!/usr/bin/python
#        pCGI10.py COMP519
import os
import sys

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

file_name = "." + os.environ["PATH_INFO"]

try:
    f = open(file_name, "r")
except:
    print "Oops, there was an error with the file!"
    sys.exit()

for x in f:
    print x,

f.close()
```
Counting Visitors

Suppose I want to count the number of visitors to my webpage.

How can I do that?

You basically have all of the tools necessary to do this already. Use a file to keep track of this counter that you are tracking. Update the file whenever a visitor accesses your webpage.
A CGI Counter

#!/usr/bin/python
#pCGI11.py COMP519
import os
from utility import start_html, end_html
start_html("Rack 'em Up!")

file_name = "counter.dat"
if not os.path.exists(file_name):
    f = open(file_name, "w")
    f.write("0")
    f.close()
f = open(file_name, "r+")
count = int(f.read())
count += 1
f.seek(0,0)
f.write(str(count))
f.close()
print "<h3>Welcome visitor #" + str(count) + "</h3>"
print "<p>Isn’t my webpage grand?</p>"
print "<img src="http://cgi.csc.liv.ac.uk/~martin/teaching/comp519/HTML/Cathedral.jpg" title="Liverpool’s Anglican Cathedral" alt="Picture of Liverpool’s Anglican Cathedral">" end_html()

http://cgi.csc.liv.ac.uk/cgi-bin/cgiwrap/martin/pCGI11.py