COMP519 Web Programming
Lecture 18: CGI Programming
Handouts

Ullrich Hustadt

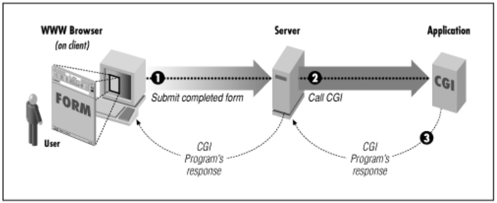
Department of Computer Science
School of Electrical Engineering, Electronics, and Computer Science
University of Liverpool

Contents

Common Gateway Interface — CGI

The Common Gateway Interface (CGI) is a standard method for web servers to use an external application, a CGI program, to dynamically generate web pages

1. A web client generates a client request, for example, from an HTML form, and sends it to a web server
2. The web server selects a CGI program to handle the request, converts the client request to a CGI request, executes the program
3. The CGI program then processes the CGI request and the server passes the program's response back to the client



Client requests

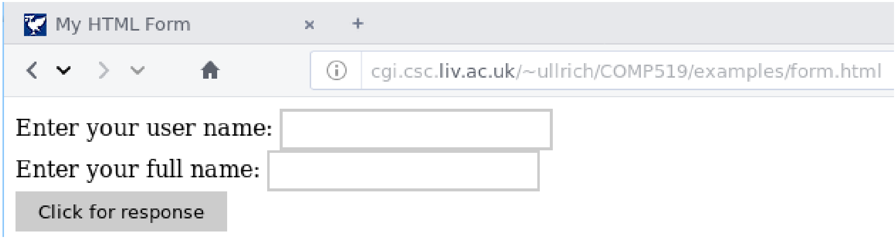
In the following we focus on client requests that are generated using HTML forms

```
<html lang="en-GB">
  <head><title>My HTML Form</title></head>
  <body>
    <form action=
"http://student.csc.liv.ac.uk/cgi-bin/cgiwrap/uh/process"
          method="post">
      <label>Enter your user name:
        <input type="text" name="username">
      </label><br>
      <label>Enter your full name:
        <input type="text" name="fullname">
      </label><br>
      <input type="submit" value="Click for response">
    </form>
  </body>
</html>
```

Client requests

In the following we focus on client requests that are generated using HTML forms

```
<html lang="en-GB">
  <head><title>My HTML Form</title></head>
  <body>
    <form action="http://student.csc.liv.ac.uk/cgi-bin/cgiwrap/uh/process"
          method="post">
      <label>Enter your user name:<input type="text" name="username"></label><br>
      <label>Enter your full name:<input type="text" name="fullname"></label><br>
      <input type="submit" value="Click for response">
    </form>
  </body>
</html>
```



Encoding of input data

- Input data from an HTML form is sent URL-encoded as sequence of key-value pairs: `key1=value1&key2=value2&...`

```
username=dave&fullname=David%20Davidson
```

- Keys may not be unique (for example, in the case of checkboxes)
- Form controls without name do not appear
- All characters except `A-Z`, `a-z`, `0-9`, `-`, `_`, `.`, `~` (unreserved characters) are encoded
- ASCII characters that are not unreserved characters are represented using ASCII codes (preceded by %)
  - A space is represented as %20 or +
  - + is represented as %2B
  - % is represented as %25

```
username=cath&fullname=Catherine+O%27Donnell
```

Request methods: GET versus POST

The two main request methods used with HTML forms are GET and POST:

- GET:
  - Form data is appended to the URI in the request (limited to 1KB to 8KB characters depending on both browser and server)

    ```
    <scheme> "://" <server-name> ":" <server-port>
    <script-path> <extra-path> "?" <query-string>
    ```

  - Form data is accessed by the CGI program via environment variables, name/value pairs that are part of the environment in which a process/programs is run by the operating system
  - Requests remain in the browser history and can be bookmarked
  - Requests should not be used for sensitive data, e.g. passwords

```
GET /cgi-bin/cgiwrap/uh/process?username=dave&fullname=David
    ↪+Davidson HTTP/1.1
Host: student.csc.liv.ac.uk
```

Request methods: GET versus POST

The two main request methods used with HTML forms are `GET` and `POST`:

- `POST`:
  - Form data is appended to end of the request (after headers and blank line)
  - There is no limit on the length/size of the form data
  - Form data can be accessed by the CGI program via standard input
  - Form data is not necessarily URL-encoded (but URL-encoding is the default)
  - Requests do not remain in the browser history and cannot be bookmarked
  - Requests are suitable for the transfer of sensitive data, e.g. passwords

```
POST /cgi-bin/cgiwrap/uh/process HTTP/1.1
Host: student.csc.liv.ac.uk

username=dave&fullname=David+Davidson
```

Environment variables: GET

Env variable	Meaning
QUERY_STRING	The query information passed to the program
REQUEST_METHOD	The request method that was used
PATH_INFO	Extra path information passed to a CGI program
PATH_TRANSLATED	Translation of PATH_INFO from virtual to physical path
SCRIPT_NAME	The relative virtual path of the CGI program
SCRIPT_FILENAME	The physical path of the CGI program

```
GET http://student.csc.liv.ac.uk/cgi-bin/cgiwrap/uh/demo/more/dirs?
    username=dave&fullname=David+Davidson
```

```
QUERY_STRING       username=dave&fullname=David+Davidson
REQUEST_METHOD     GET
PATH_INFO          /more/dirs
PATH_TRANSLATED    /users/www/external/docs/more/dirs
SCRIPT_NAME        /cgi-bin/cgiwrap/uh/demo
SCRIPT_FILENAME    /users/loco/uh/public_html/cgi-bin/demo

STDIN             # empty
```

Environment variables: GET

Env variable	Meaning
QUERY_STRING	The query information passed to the program
REQUEST_METHOD	The request method that was used
PATH_INFO	Extra path information passed to a CGI program
PATH_TRANSLATED	Translation of PATH_INFO from virtual to physical path
SCRIPT_NAME	The relative virtual path of the CGI program
SCRIPT_FILENAME	The physical path of the CGI program

```
GET http://student.csc.liv.ac.uk/cgi-bin/cgiwrap/uh/process/more/dirs?
    username=2%60n+d%2Bt+e+s%27t&fullname=Peter+Newton
```

```
QUERY_STRING       username=2%60n+d%2Bt+e+s%27t&fullname=Peter+Newton
REQUEST_METHOD     GET
PATH_INFO          /more/dirs
PATH_TRANSLATED    /users/www/external/docs/more/dirs
SCRIPT_NAME        /cgi-bin/cgiwrap/uh/process
SCRIPT_FILENAME    /users/loco/uh/public_html/cgi-bin/process

STDIN             # empty
```

Environment variables: POST

Env variable	Meaning
QUERY_STRING	The query information passed to the program
REQUEST_METHOD	The request method that was used
SCRIPT_NAME	The relative virtual path of the CGI program
SCRIPT_FILENAME	The physical path of the CGI program

```
POST /cgi-bin/cgiwrap/uh/demo
Host: student.csc.liv.ac.uk

username=2%60n+d%2Bt+e+s%27t&fullname=Peter+Newton
```

```
QUERY_STRING        # empty
REQUEST_METHOD      POST
SCRIPT_NAME         /cgi-bin/cgiwrap/uh/demo
SCRIPT_FILENAME     /users/loco/uh/public_html/cgi-bin/demo

STDIN               username=2%60n+d%2Bt+e+s%27t&fullname=Peter+Newton
```

More environment variables

Env variable	Meaning
HTTP_ACCEPT	A list of the MIME types that the client can accept
HTTP_REFERER	The URL of the document that the client points to before accessing the CGI program
HTTP_USER_AGENT	The browser the client is using to issue the request
REMOTE_ADDR	The remote IP address of the user making the request
REMOTE_HOST	The remote hostname of the user making the request
SERVER_NAME	The server's hostname
SERVER_PORT	The port number of the host on which the server is running
SERVER_SOFTWARE	The name and version of the server software

CGI programs and Python

- CGI programs need to process input data from environment variables and STDIN, depending on the request method
  - $\rightsquigarrow$ preferably, the input data would be accessible by the program in a uniform way
- CGI programs need to process input data that is encoded
  - $\rightsquigarrow$ preferably, the input data would be available in decoded form
- CGI programs need to produce HTML markup/documents as output
  - $\rightsquigarrow$ preferably, there would be an easy way to produce HTML markup

In Python, we can use

- the `cgi` module to process inputs
- the `environ` dictionary of the `os` module to access environment variables
- `print` statements to produce HTML markup

Python: Basic Syntax

- A Python program/script consists of one or more statements and comments
- One-line comments start with # and run to the end of the line
- Multi-line comments simply consist of several one-line comments
- Statements are delimited by newlines except where a newline is escaped (by a backslash \)
- On Unix/Linux systems, Python scripts begin with #! (called 'hash bang' or 'she bang') and the location of the Python interpreter/compiler

```
#!/usr/bin/python3
# HelloWorld.py
# Our first Python script

print("Hello World")
```

Python: Basic Syntax

- Strictly speaking, in Python one assigns a (variable) name to a value, not the other way round
  ⇝ a (variable) name does not exist before the first assignment
- But, the syntax for an assignment is the same as in JavaScript

```
age = 23
```

- The first assignment to a variable defines that variable
- Python supports the standard binary assignment operators

```
age += 10
```

- Python uses static scoping

- Blocks of statements, called suites are delimited with indentation
  ⇝ each time the level of indentation is increased, a new block starts
  ⇝ each time the level of indentation is decreased, a block has ended
- A colon : separates the header of block from the rest of the suite

Python: Type System

- Python is a dynamically typed language:
  a variable declaration does not include a type and
  a variable can hold values of different types over time

  ```
  x = "Hello"
  x = 42
  ```

  is a valid sequence of statements
- Python is a (mostly) strongly typed language:
  values are not automatically converted from 'unrelated' types

  ```
  y = "Hello" + 42
  ```

  will cause an error in Python
- However, quite a number of types are considered to be 'related'

  ```
  z = 42 and True  # z --> True
  ```

  will not cause an error in Python although a boolean operator is applied to a number

Python: Type System: Strings

- A string literal is a sequence of characters surrounded by single-quotes, double-quotes, or triple-quotes

`'chars'`	single-quoted string
`"chars"`	double-quoted string
`'''chars'''` `"""chars"""`	triple-quoted string, can span several lines and contain single and double quotes, but not at the start or end

```
'''This is a triple-quoted 'string' containing "quotes"
   and spanning more than one line'''
```

- In all these forms \ acts as escape character

Python: Type System: Dictionaries

- A Python dictionary is a mapping of keys to values (aka associative array or hash table)
- A dictionary literal is a comma-separated list of key-value pairs consisting of a key and a value separated by a colon : surrounded by curly brackets

```
{ 'name': 'Dave', 'age': 23, 'height': '185cm' }
```

- Elements of any immutable type, e.g. strings, can be used as keys
- The value associated with a specific key `key` in a dictionary `dict` can be accessed (and modified) using

```
dict[key]
```

```
dct = { 'name': 'Dave', 'age': 23, 'height': '185cm' }
print(dct['name'])          # prints 'Dave'
dct['height']  = '190cm'    # 'height'  now maps to '190cm'
dct['age']    += 1          # 'age'     now maps to 24
dct['surname'] = 'Shield'   # 'surname'     maps to 'Shield'
```

Python: Conditional Statements

Python conditional statements take the following form

```
if condition:
    suite
elif condition:
    suite
else:
    suite
```

- The else-clause is optional and there can be at most one
- The elif-clause is optional and there can be more than one
- None of the *suite* blocks of statements can be empty
  $\leadsto$ the statement `pass` can be used as a ‘null statement’

```
if x == 0:
   # We'll come up with a solution for x == 0 later
   pass
else:
   y = y / x
```

Python: Functions

Functions are elements of type function and can be defined as follows:

```
def identifier(param1,param2, ...):
    docstring
    suite
```

- The function name *identifier* is case-sensitive
- The function name must be followed by parentheses
- A function has zero, one, or more parameters that are variables
- Parameters are not typed
- *docstring* is a string describing the function and will be returned by `help(identifier)` or `identifier.__doc__`
- *suite* is a non-empty sequence of statements

A function is called by using the function name followed by a list of arguments in parentheses

```
... identifier(arg1, arg2,...) ... # Function call
```

Python: Modules

- A lot of functionality of Python is contained in modules
- The statement

  ```
  import module1[, module2[,... moduleN]]
  ```

  makes all functions from modules *module1*, *module2* available, but all function names must be prefixed by the module name

  ```
  import math
  print math.factorial(5)
  ```

- The statement

  ```
  from module1 import fun1[, fun2[,... funN]]
  ```

  imports the named functions from module *module1* and makes them available without the need for a prefix

  ```
  from math import factorial
  print factorial(5)
  ```

Python: The re module

The re module of Python provides functions that use regular expressions

`re.match(`*regexp*`,`*string* `[,`*flags*`])`

attempts to find a match for *regexp* at the start of *string*

returns a match object when *regexp* is found and None otherwise

```
re.match(r'[mM][rs]',"Mr Dobbs")    # MatchObject
re.match(r'[mM][rs]',"Hi Mr Dobbs") # None
```

`re.search(`*regexp*`,`*string* `[,`*flags*`])`

attempts to find a match for *regexp* anywhere in *string*

returns a match object when *regexp* is found and None otherwise

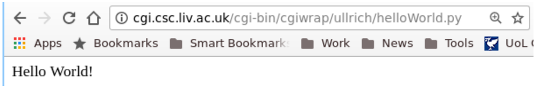
```
re.search(r'[mM][rs]',"Hi Mr Dobbs") # MatchObject
re.search(r'[mM][rs]',"Miss Dobbs")  # None
```

Hello World CGI Program

```
#!/usr/bin/python3

print('''\
Content-type: text/html

<!DOCTYPE html>
<html lang="en-GB">
  <head>
    <meta charset="utf-8">
    <title>Hello World</title>
  </head>
  <body>
    Hello World
  </body>
</html>''')
```



Hello World!

User-Defined Functions for an HTML Wrapper

It makes sense to define functions that print out the
initial (up to body start) and final HTML markup (from body end)

```
#!/usr/bin/python3
# htmlUH.py placed in ~ullrich/public_html/cgi-bin/

def start_html(title):
  print('''\
Content-type: text/html

<!DOCTYPE html>
<html lang="en-GB">
  <head>
    <meta charset="utf-8">
    <link rel="stylesheet" type="text/css"
          href="/~ullrich/COMP519/examples/table.css">
    <title>''' + title + '''</title>
  </head><body>''')
def end_html():
  print('''  </body></html>''')
```

Processing Environment Variables

- The module `os` provides the `environ` dictionary
- The `environ` dictionary maps a script's environmental variables as keys to the values of those variables

`os.environ['SERVER_ADDR']`	`10.128.0.103`
`os.environ['SERVER_NAME']`	`cgi.csc.liv.ac.uk`
`os.environ['SERVER_PROTOCOL']`	`HTTP/1.1`
`os.environ['SERVER_SOFTWARE']`	`Apache/2.4.34 ... Python` ↪`/3.6 PHP/7.2.10`
`os.environ['HTTP_USER_AGENT']`	`... Chrome/78.0.3904.85 ...`
`os.environ['REMOTE_ADDR']`	`212.159.116.53`
`os.environ['REQUEST_METHOD']`	`GET`
`os.environ['REQUEST_SCHEME']`	`https`
`os.environ['SCRIPT_URI]`	`https://cgi.csc.liv.ac.uk/` ↪`cgi-bin/cgiwrap/ullrich` ↪`/python14A.py`
`os.environ['QUERY_STRING']`	
`os.environ['SCRIPT_NAME']`	`/LOCAL/www/html/ullrich/` ↪`python14A.py`

Processing Environment Variables: Example

```
#!/usr/bin/python3
import os,  sys, re, codecs, locale
from htmlUH import start_html,end_html

# Make sure output uses UTF-8 encoding
sys.stdout = codecs.getwriter("utf-8")(sys.stdout.detach())

start_html("Where are you coming from?")

user_ip = os.environ["REMOTE_ADDR"]

print('<div>Clients IP address: ' + user_ip + '</div>')

if re.match(r'138\.253\.', user_ip):
  print('<p>Welcome, university user!</p>')
  print('''<p>Lots more content only available
           to university users</p>''')
else:
  print('<div><b>Sorry, please come back\
               when you are on a uni computer</b></div>')
end_html()
```

Accessing and Processing Form Data

The module `cgi` provides methods to access the input data of HTML forms in a two step process:

1. Create an instance of the `FieldStorage` class and assign it to a variable

   ```
   variable = cgi.FieldStorage()
   ```

   This reads the form data from standard input or the environment variable `QUERY_STRING`

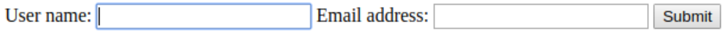
2. Then
   - `variable['name'].value`
   - `variable.getvalue('name')`
   - `variable.getfirst('name', default=None)`
   - `variable.getlist('name')`

   return the value/values entered for the form control with name *name*

Processing Form Data: Example

We want to create a CGI script that both creates the following form and validates its inputs

```
# The form
<form method="POST">
  <label>User name:     <input type="text" name="user"></label>
  <label>Email address: <input type="text" name="email"></label>
  <input type="submit" name="submit" />
</form>
```



The form itself will be created by the following function

```
def printForm():
  print('''<form method="POST">
     <label>User name:     <input type="text" name="user"></label>
     <label>Email address: <input type="text" name="email"></label>
     <input type="submit" name="submit" />
   </form>''')
```

Processing Form Data: Example

Validation will be done by the following two functions

```
def validateName(field):
  if field == "" or field == None:
    return "No username entered.\n"
  elif len(field) < 5:
    return "Username too short.\n"
  elif re.search(r'[^a-zA-Z0-9_-]',field):
    return "Invalid character in username.\n"
  else:
    return ""

def validateEmail(field):
  if field == "" or field == None:
    return "No email entered.\n"
  elif not ((field.find(".") > 0) and (field.find("@") > 0)):
    return "Symbol @ or . missing.\n"
  elif re.search(r'[^a-zA-Z0-9\.\@\_\-]',field):
    return "Invalid character in email.\n"
  else:
    return ""
```

Accessing and Processing Form Data: Example

```
#!/usr/bin/python3
import cgi, os, sys, socket, re, codecs, locale
from htmlUH import start_html,end_html
sys.stdout = codecs.getwriter("utf-8")(sys.stdout.detach())

start_html("Form Processing")
inputs = cgi.FieldStorage()

if inputs.getvalue('submit'):
  err = validateName(inputs.getvalue('user'))
  err = err + validateEmail(inputs.getvalue('email'))
  if err:
    print('<div class="error"> Error:',err,' Try again.</div>')
    printForm()
  else:
    print('<div class="ok">All inputs are OK!</div>')
    # Some further processing follows now
else:
  printForm()

end_html()
```

Revision and Further Reading

- To get familiar with Python start at
  Python Software Foundation: Our Documentation.
  Python.org, 29 Oct 2019. `https://www.python.org/doc/`
  [accessed 29 Oct 2019]
- For information on modules for CGI programming in Python see
  - `cgi`: Common Gateway Interface support
    `https://docs.python.org/3/library/cgi.html`
  - `os`: Miscellaneous operating system interfaces
    `https://docs.python.org/3/library/os.html`

  of Python Software Foundation: The Python Standard Library.
  Python.org, 29 October 2019.
  `https://docs.python.org/3/library`
  [accessed 29 October 2019]