Lecture 6
Python CGI

Lecturer: Pieter De Bleser

Bioinformatics Core Facility, IRC

GHENT UNIVERSITY

VIB-UGENT CENTER FOR INFLAMMATION RESEARCH

Slides derived from: Andrew C.R. Martin (andrew.martin@ucl.ac.uk)

The Internet and WWW

The Internet:

Hardware

Physical connection between different computers

Software (TCP/IP)

Allows communication between the computers.

Network

Once connected to one point on the net, can potentially access the entire net.

Communication Types on the Internet

E-mail

Send and receive messages.

Telnet/<u>ssh</u> (safer version that uses encryption)

Login to one computer from another one.

FTP (File Transfer Protocol)

Copy a file from one computer to another.

HTTP (HyperText Transport Protocol)

Retrieve hypertext files present in special repositories on other computers.

Hypertext Files and The World Wide Web

Hypertext Files

Text containing links to other documents (e.g. hypertexts, graphics, sound etc.), in the form of highlighted keywords.

The World Wide Web (WWW)

The collection of all hypertext and hypermedia resources on the Internet.

What is a web client?

Any program that retrieves data from a web server using the HTTP protocol

Examples:

- web browsers – contact web servers using HTTP protocol and display HTTP responses
- web crawlers – traverse the web automatically to gather information
- web service clients (service requester) – request and process data from a web service provider; the web service provider responds using some web service protocol such as RSS (Rich Site Summary) or RPC (remote procedure call)

Client Programs

1. For E-mail: Microsoft Outlook, Mozilla Thunderbird
2. For FTP: FileZilla, FireFTP, winSCP, cyberDuck...
3. For HTTP: Web Browsers (Firefox, Edge,...)

The Internet and WWW

Accessing Internet resources via a Web browser

URL (Uniform Resource Locator)

An addressing system that locates documents on the Internet.
Examples:
https://ubidots.com/
https://academic.oup.com/

The URL contains the internet protocol, host computer name, directory path, filename.

mailto:pieterdb@irc.vib-ugent.be
The "mailto" URL contains userid and host name.

Client - Server Architecture on the Web

Web Server

A repository of hypertext / hypermedia files and a software package that allows their access by Web browsers using the HTTP protocol.

```
                    HTTP request ---------------> Web browser
Web Server (Client) (Server) <--------------- HTML document
```

HTML (Hyper Text Markup Language)

Code defining the structure, appearance and linking to other documents of a hypertext document.

Static and Dynamic Web Pages

Static HTML page

Stored as an HTML file on the Web server.

Dynamic HTML page

Created "on the fly" by a computer program at the server side.

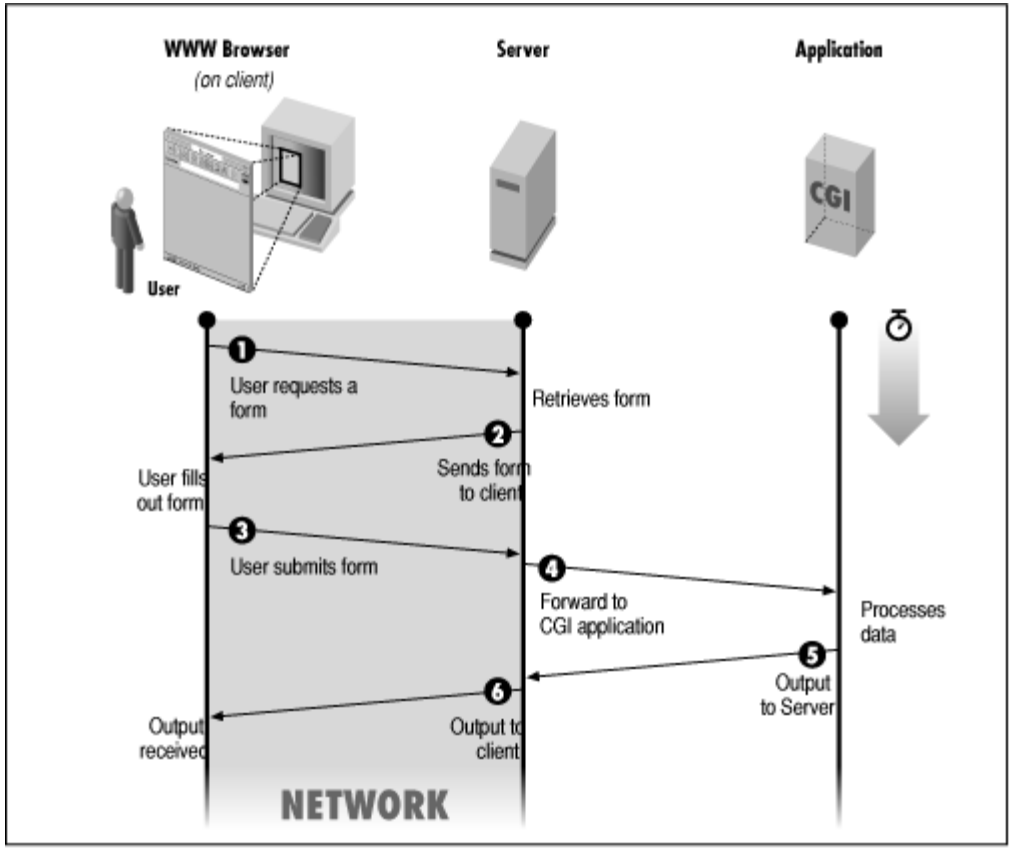
CGI (Common Gateway Interface)

The mechanism that connects between computer programs and the Web server.

CGI-Programming

CGI (Common Gateway Interface) is the mechanism that connects between computer programs and the Web server.
It allows you to provide a Web-interface to your program, or, in other words, to run your program from a Web browser and get the output as a Web page.

• Form interaction with CGI



Web Server Support and Configuration

Does your Web Server supports CGI and it is configured to handle CGI Programs?

- CGI programs are kept in a pre-configured directory, the CGI directory.
- Conventions:
  - Located at /var/www/cgi-bin.
  - CGI files have extension as .cgi,
    - but you can keep your files with python extension .py as well.

```
<Directory "/var/www/cgi-bin">
   AllowOverride None
   Options ExecCGI
   Order allow,deny
   Allow from all
</Directory>
<Directory "/var/www/cgi-bin">
Options All
</Directory>
```

- The Linux server is configured to run only the scripts in the cgi-bin directory in /var/www. If not OK modify the httpd.conf file

Testing your CGI script

If you want to experiment some python code as CGI script to serve by a HTTP server, you can get started by these steps:

1. Create a cgi-bin directory.
2. Write a script (‘hello.py’) and save it in the cgi-bin directory.
3. Make the script executable:

```
> chmod a+x hello.py
```

4. Start up a local http server:

```
> python3 -m http.server --cgi
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

5. Test it on your browser with: `http://localhost:8000/cgi-bin/hello.py`.
   Hit CTRL+C to stop the server.

Testing your CGI script

Hello World! Your custom CGI script is working. Here are your current Python local variables.

__annotations__
__builtins____cached____doc____file____loader____name____package____spec__localvars_table

NOTE: If you want to write useful CGI script, try the Python 'cgi' module. See cgitest.py script.

local_vars.py

```
#!/usr/bin/env python3
localvars_table = '<table>'

for x in dir():
    localvars_table += '<tr><td>%s</td></tr>' % x
    localvars_table += '</table>'

print("Content-type: text/html")
print("")
print("""<html><body>
<p>Hello World! Your custom CGI script is working. Here are your current Python local variables.</p>
%s
<p>NOTE: If you want to write useful CGI script, try the Python 'cgi' module. See cgitest.py script.</p>
</body></html>""" % (localvars_table))
```

MIME Types - Notes

Each web document has a **MIME** type that tells the browser how the document should be displayed:

- **text/plain**: To create a text/plain file simply create a file in your web directory with the extension .txt and put some text in it. The browser presents the text from such a file without any modification or formatting.
- the MIME type of a document is mostly determined from its extension. For example, JPEG images have MIME type image/jpeg and have the extensions jpeg, jpg, etc.
- Some of the other mime types that you are likely to encounter as a web user are **text/html**, **image/gif**, and **application/pdf**.
- Some mime types must be sent to a ***plug-in*** or ***helper application*** (plug-in's are somewhat more integrated than helper applications). For example, you need a pdf reader to view application/pdf files.

  Recommendation: To provide maximum hassle free usability of your web site stick to MIME types that are built in to Netscape!

MIME: Multipurpose Internet Mail Extensions

Introduction to HTML

- HTML (HyperText Markup Language) (MIME type text/html, file extensions html, htm) is the central MIME type for web documents. Core HTML provides formatting,hyperlinks and "forms“.
- Core HTML allows you to provide a huge amount of functionality for very little effort.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN<93>
"http://www.w3.org/TR/REC-html40/loose.dtd">
<HTML>
<HEAD>
<TITLE>A Simple HTML Document, simple.html</TITLE>
</HEAD>
<BODY>
<H1>This is a title</H1>
<P>
And here is some ordinary text.
</P>

<form action=http://yoursite/cgi-bin/test-cgi.pl method="POST">
Choose a Motif: <input type="text" name="motif" value="TATTAT">
<br>
<input type="submit" name="search" value="Search!"> <br> </form>

</BODY>
</HTML>
```

Introduction to HTML – simple.html

CGI Scripts – script location

```
<form method="post" action="/cgi-bin/clustalw/clustalw.py">
```

- Note the location of the script
  - the CGI script will reside on the same machine as the web page
  - can also use a full URL

post and get

- ‘get’
  - Used where small amounts of data are to be sent
  - Data are sent as part of the URL

```
<form method="get" action="/cgi-bin/clustalw/clustalw.py">
```

https://www.ncbi.nlm.nih.gov/gene/?term=TP53

post and get

- ‘post’
  - Used where larger amounts of data are to be sent
  - Data sent separately from the URL

```
<form method="post" action="/cgi-bin/clustalw/clustalw.py">
```

Introduction to HTML - Notes

Comments	<!...>
Document "outline"	<html><head><title>...</title></head><body>...</body></html>
Pairs of *tags*	<*whatever*> matches </*whatever*> (but not every <*whatever*> requires a </*whatever*>)
Paired tags must nest	E.g. <head><title>...</title></head> **NOT** <head><title>...</head></title>
Center format instruction	...
Heading format instructions	<H1>This is a title</H1>

Hyperlinks

Genome Informatics

In this example the URL refers to the file index.html in the parent directory of to the current directory.
(Files named index.html often have a special role; web servers are often configured so that e.g. the URL http://bush1/ refers to http://bush1/index.html.)
The text Genome Informatics between and is called an *anchor*, and gives the reader something to click on.
Hyperlinks can also refer to an entirely different web site, for example this is a link the Apache web site:

Apache (Apache).

The *href* can be either relative (i.e. a path relative to the protocol, host, and path of the current document), or can specify a full protocol, host, and (optional) path.

Other Essential HTML Tags

Start a new paragraph. (No need for .)

<hr>

Print a "horizontal rule":

(No need for </hr>.)

...

Use a "strong" (e.g. bold) font, **like this**.

<pre>...</pre>

Leave the line breaks and whitespace the way they are. For example, <pre> These line breaks are not wrapped. </pre> gets presented like this: These line breaks are not wrapped.

```
<pre> These
   line    breaks are
not wrapped.
</pre>
```

gets presented like this:

```
These
  line    breaks are
not wrapped.
```

Creating Fill-Out Forms

HTML includes about a half-dozen elements for creating fill-out form elements. A form must begin with <FORM> and end with </FORM>:

Code:

```
<form action=http://yoursite/cgi-bin/test-cgi.pl method="POST">
Choose a Motif: <input type="text" name="motif" value="TATTAT"> <br>
<input type="submit" name="search" value="Search!"> <br> </form>
```

Result:

The <FORM> Tag

Attributes:

action (required)

CGI script to submit contents of form to.

method (required)

Submission method. Depends on CGI script. One of:

- POST
- GET

encoding

Required by certain scripts that accept file uploads. One of:

- application/x-www-form-urlencoded
- multipart/form-data

<INPUT> Elements

Used for text fields, buttons, checkboxes, radiobuttons. Attributes:

type

Type of the field. Options:

- submit
- radio
- checkbox
- text
- password
- hidden
- file

name

Name of the field.

value

Starting value of the field. Also used as label for buttons.

size

Length of text fields.

checked

Whether checkbox/radio button is checked.

<INPUT> Elements - examples

```
<input type="text"   name="motif1" value="TATTAT">
<input type="checkbox" name="motif2" value="TATTAT">
<input type="radio" name="motif3" value="TATTAT" checked>
<input type="radio" name="motif3" value="GGGGGG">
<input type="hidden" name="settings" value="PRIVACY MODE ON">
<input type="submit" name="search" value="SEARCH!">
```

TATTAT

SEARCH!

<SELECT> Element

Used to create selection lists.
Attributes:
name
Name the field.
size
Number of options to show simultaneously.
multiple
Allow multiple options to be shown simultaneously.

<OPTION> Element
Contained within a >SELECT> element. Defines an option:
>option>I am an option</option>
Attributes:
selected
Whether option is selected by default.
value
Give the option a value different from the one displayed.

<SELECT> Element - examples

```
<select name="motif1">
  <option>GATTTAA</option>
  <option selected>GGGTTTTC</option>
  <option>TTTTTAAAAA</option>
  <option>TATATATAT</option>
  <option value="Tricky!">GGCCGGTTA</option>
</select>

<select name="motif2" size=6>
  <option>GATTTAA</option>
  <option selected>GGGTTTTC</option>
  <option>TTTTTAAAAA</option>
  <option>TATATATAT</option>
  <option>GGCCGGTTA</option>
</select>

<select name="motif3" size=6 multiple>
  <option>GATTTAA</option>
  <option selected>GGGTTTTC</option>
  <option>TTTTTAAAAA</option>
  <option>TATATATAT</option>
  <option>GGCCGGTTA</option>
</select>

<input type="submit" name="search" value="SEARCH!">
```

GGGTTTTC

GATTTAA
GGGTTTTC
TTTTTAAAAA
TATATATAT
GGCCGGTTA

GATTTAA
GGGTTTTC
TTTTTAAAAA
TATATATAT
GGCCGGTTA

SEARCH!

<TEXTAREA> Elements

Used to create big text elements.
Attributes:

name

name of field

rows

rows of text

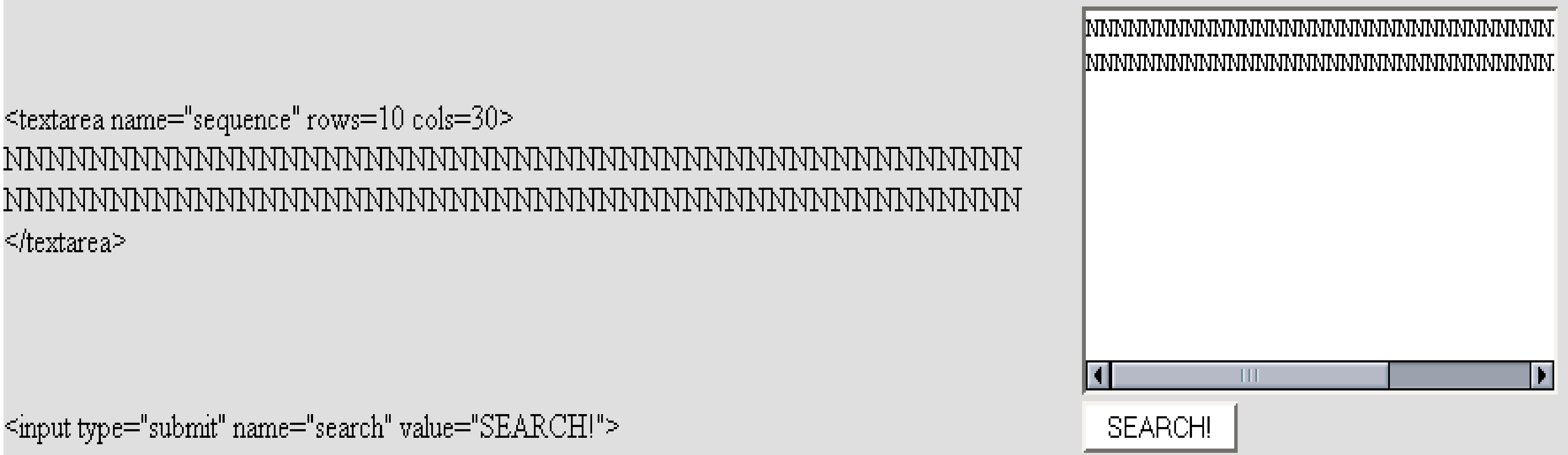
cols

columns of text

wrap

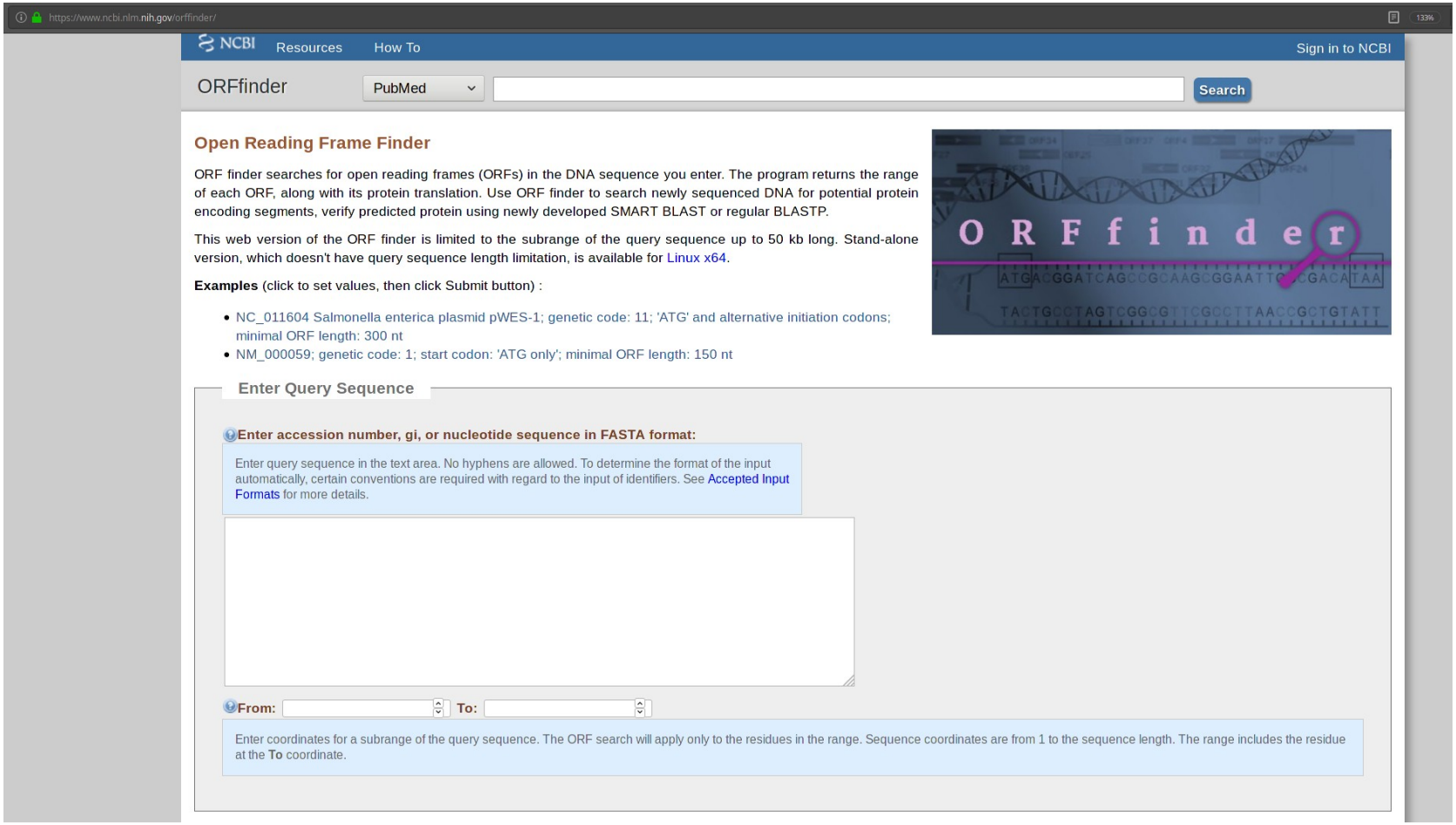
type of word wrapping

<TEXTAREA> Elements - examples



```
<textarea name="sequence" rows=10 cols=30>
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
</textarea>

<input type="submit" name="search" value="SEARCH!">
```

Examples for uses in Bioinformatics



Examples for uses in Bioinformatics

Creating a Web page "on the fly" by a CGI-program

```
#!/usr/bin/env python3

print("Content-type:text/html\r\n\r\n" )
print('<html>')
print('<head>')
print('<title>Hello Word - First CGI Program</title>')
print('</head>')
print('<body>')
print('<h2>Hello Word! This is my first CGI program</h2>')
print('</body>')
print('</html>')
```

Make the script executable:

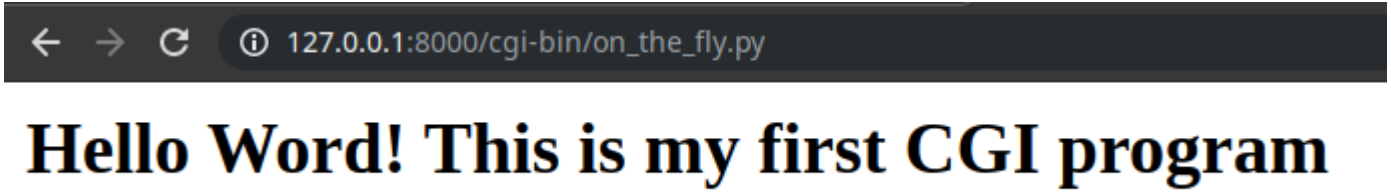
```
> chmod a+x on_the_fly.py
```

Start up a local http server:

```
> python3 -m http.server --cgi
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Test it on your browser with: `http://localhost:8000/cgi-bin/on_the_fly.py`.
Hit CTRL+C to stop the server.

Creating a Web page "on the fly" by a CGI-program



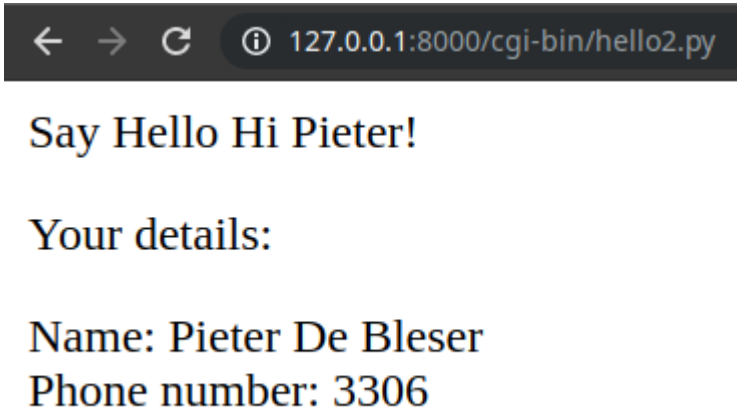
Guidelines for HTML page generation by a CGI-program

- In order for your program to send its standard output to the Web it should be placed in a special directory on the Web server, usually the cgi-bin directory.
- You can sometimes get the same effect if you append the name of your program with .cgi and place it in your public_html directory.
- Consult your system administrator or Internet provider for that.
- If your program output is a text file (e.g. in HTML format), you must include in your program the following print command before any other print commands: print "content-type: text/html\n\n";
- Remember to use <BR> or <P> tags to mark end of text lines.
  \n "newlines" will not be visible on Web pages.
- In quoted strings, remember to place a backslash before quotes. e.g. print "<A HREF=\"home.html\">Back to Home Page</A>";
- Do not forget to make the file containing your program (e.g. hello.cgi) executable by "the world" using the Unix command: chmod a+x hello.cgi
- To execute your program from a Web browser, use the following URL (assuming that your program is in your public_html directory): http://host_computer_name/~userid/hello.cgi Where userid is your own userid.

Receiving CGI-program arguments from the URL (I)

hello2.py

```
#!/usr/bin/env python3

f_name = "Pieter"
l_name = "De Bleser"
phone  = 3306

print("Content-type:text/html\r\n\r\n")
print("<html>")
print("<head>")
print("<title>Say Hello</title>")
print("</head>")
print("<body")
print("<h2>Say Hello</h2>")
print("Hi %s!<p>" % (f_name))
print("Your details:<p>")
print("Name: %s %s<br>" % (f_name, l_name))
print("Phone number: %s" % (phone))
print("</body>")
print("</html>")
```



Receiving CGI-program arguments from the URL (II)

The URL
You can send arguments to a CGI-program by appending a **query string** to the URL of the program, having the following format: `http://.../program_name?name1=value1&name2=value2&name3=value3`

```
                                    ----------------------------------------
                                                 query string
```

name1, value1
name2, value2
name3, value3 (and you can add more)
are called **name-value pairs**, each defining a variable name and its value (content).

The program
The CGI-program receives the query string as an environmental variable, which can be accessed by the class **FieldStorage** to work with the submitted form data.

The CGI module
The Python CGI module handles situations and helps debug scripts. With the latest addition, it also lends us support for uploading files from a form.

Using CGI for parsing the query string

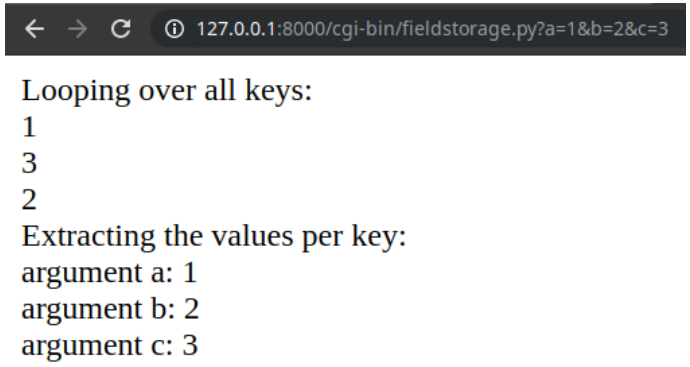
In order for your Python program to be able to use the functions included in the CGI module, write the following commands somewhere near the top of your program:

```
import cgi
import cgitb # Optional; for debugging only
```

`cgi.FieldStorage()` returns a dictionary with key as the field and value as its value.

fieldstorage.py

```
#!/usr/bin/env python3
import cgi
import cgitb; cgitb.enable() # Optional; for debugging only

print("Content-Type: text/html")
Print("")

arguments = cgi.FieldStorage()

print("Looping over all keys:<br>")
for i in arguments.keys():
    print("%s<br>" % (arguments[i].value))

print("Extracting the values per key:<br>")
print("argument a: %s<br>" % (arguments['a'].value))
print("argument b: %s<br>" % (arguments['b'].value))
print("argument c: %s<br>" % (arguments['c'].value))
```



Form generation – CGI version

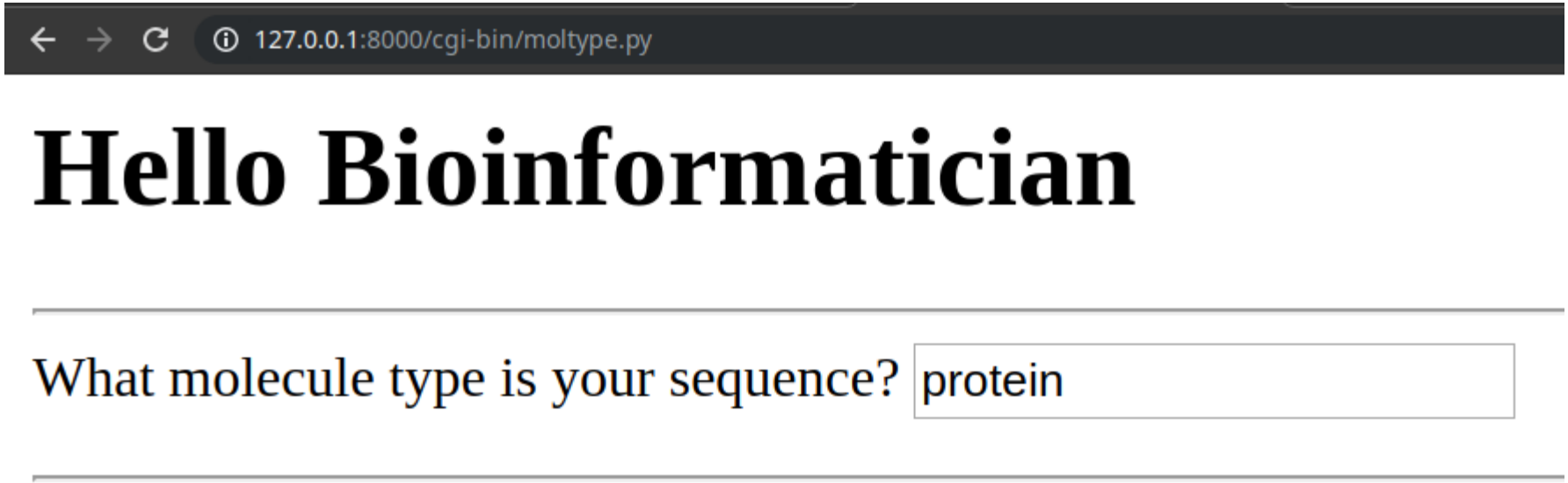
The same CGI program can:
1.produce the HTML page with the form
2.respond to the user's form input

Implementation:
divide the program into two parts, which do different things depending on whether or not the program was invoked with arguments.
If no arguments were received, then the program sends the empty form to the browser.
Otherwise, the arguments contain a user's input to the previously sent form, and the program returns a response to the browser based on that input.

Form generation – CGI version

```
#!/usr/bin/env python3
# Import modules for CGI handling
import cgi, cgitb

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

seqtype = form.getvalue('moltype')

print("Content-type:text/html\r\n\r\n")
print("<html>")
print("<head>")
print("<title>moltype.py CGI Program</title>")
print("</head>")
print("<body>")
print("<h1> Hello Bioinformatician </h1>" )
print("</body>")

if seqtype:
    print("Your sequence is %s" % (seqtype))
else:
    print("<hr>")
    print(r'<form method="post" action="moltype.py" enctype="multipart/form-data">')
    print(r'What molecule type is your sequence? <input type="text" name="moltype" value="protein" /></form>')
    print("<hr>")

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

Form generation – CGI version



Form generation – CGI version

protcharge.html

```
<!DOCTYPE html>
<html lang="en">
 <head><meta charset="utf-8">
  <title>Protein Charge Calculator</title>
  <link href="css/bootstrap.min.css" rel="stylesheet">
 </head>
 <body style="background-color:#e7f5f5;">
 <div class="container"><h2>Protein Charge Calculator</h2>
  <form action='/cgi-bin/protcharge.py' method='post'>
   <div class="row">
   <div class="col-sm-8">
   <div class="form-group">
    <label for="aaseq">Enter the amino-acid sequence:</label>
    <textarea name="aaseq" rows="5" cols="40"></textarea>
   </div>
   </div>
   </div>
   <div class="row">
   <div class="col-sm-8">
   <div class="form-group">
    <label for="prop">Do you want to see the proportion of
       charged amino-acid?</label>
    <div class="radio">
     <label>
      <input type="radio" name="prop" value="y">Yes
     </label>
    </div>
    <div class="radio">
     <label>
      <input type="radio" name="prop" value="n">No
     </label>
    </div>
    <label for="title">Job title (optional):</label>
    <input type="text" size="30" name="title" value="">
    <br>
    <button type="submit" class="btn btn-primary">Send
    </button>
   </div>
   </div>
   </div>
  </form>
 </div>
 </body>
</html>
```

Protein Charge Calculator HTML

```
#!/usr/bin/env python
import cgi, cgitb

def chargeandprop(aa_seq):
    protseq = aa_seq.upper()
    charge = -0.002
    cp = 0
    aa_charge = {'C':-.045,'D':-.999,'E':-.998,'H':.091,
                 'K':1,'R':1,'Y':-.001}
    for aa in protseq:
        charge += aa_charge.get(aa, 0)
        if aa in aa_charge:
            cp += 1
    prop = float(cp)/len(aa_seq)*100
    return (charge, prop)

cgitb.enable()
print('Content-Type: text/html\n')
form = cgi.FieldStorage()
#seq = form.getvalue('seq', 'QWERTYYTREWQRTYEYTRQWE')
seq = form.getvalue('aaseq')
prop = form.getvalue('prop', 'n')
jobtitle = form.getvalue('title','No title')
charge, propvalue = chargeandprop(seq)
print('<html><body>Job title:{0}<br/>'.format(jobtitle))
print('Your sequence is:<br/>{0}<br/>'.format(seq))
print('Net charge: {0}<br/>'.format(charge))
if prop == 'y':
 print('Proportion of charged AA: {0:.2f}<br/>'
       .format(propvalue))
print('</body></html>')
```

Protein Charge Calculator Python CGI

Exercise

Write a Python CGI script that accepts a DNA sequence and returns the sequence reverse complemented.

Accessing external programs

Often need to access another program (e.g. BLAST) from your CGI script

Run a program from a Python script:

```
import subprocess

result = subprocess.check_output(“prog args”, shell=True)
result = str(result, 'utf-8')
```

Note:

In Perl you can just do:

```
$retval = `cmd args`;
```

Accessing external programs

sub_process.py

```
#!/usr/bin/env python3
import subprocess

result = subprocess.check_output("ls -l", shell=True)
result = str(result, 'utf-8')

print(result)
```

```
> python sub_process.py
total 84
-rwxrwxr-x 1 pieterdb pieterdb 854 Oct 29 15:39 adder.py
-rwxrwxr-x 1 pieterdb pieterdb 422 Oct 24 14:21 cgi_1.py
-rwxrwxr-x 1 pieterdb pieterdb  44 Oct 24 14:21 cgitest.py
-rwxrwxr-x 1 pieterdb pieterdb 482 Oct 29 13:29 fieldstorage.py
-rwxrwxr-x 1 pieterdb pieterdb 422 Oct 24 14:21 getData.py
-rwxrwxr-x 1 pieterdb pieterdb 415 Oct 29 11:07 hello2.py
-rwxrwxr-x 1 pieterdb pieterdb 634 Oct 29 13:10 hello3.py
-rwxrwxr-x 1 pieterdb pieterdb 488 Oct 28 19:22 hello_get.py
-rwxrwxr-x 1 pieterdb pieterdb 251 Oct 28 20:17 hello.py
-rwxrwxr-x 1 pieterdb pieterdb 206 Oct 28 20:09 here.py
-rwxrwxr-x 1 pieterdb pieterdb 224 Oct 28 16:25 http_get_arguments.py
-rwxrwxr-x 1 pieterdb pieterdb 469 Oct 24 14:21 local_vars.py
-rwxrwxr-x 1 pieterdb pieterdb 708 Oct 29 14:27 moltype.py
-rwxrwxr-x 1 pieterdb pieterdb 278 Oct 29 08:34 on_the_fly.py
-rwxrwxr-x 1 pieterdb pieterdb 945 Oct 28 20:59 protcharge.py
-rwxrwxr-x 1 pieterdb pieterdb 595 Oct 29 16:04 reversec.py
-rw-rw-r-- 1 pieterdb pieterdb 358 Oct 24 14:21 simple_form.html
-rwxrwxr-x 1 pieterdb pieterdb 205 Oct 28 17:37 test_contents.py
-rwxrwxr-x 1 pieterdb pieterdb 518 Oct 29 10:26 textarea.py
-rw-rw-r-- 1 pieterdb pieterdb 192 Oct 29 13:58 ulify.py
-rwxrwxr-x 1 pieterdb pieterdb 854 Oct 24 14:21 web_app_1.py
```

When you don't need output...

```
import os
os.system(“cmd args”)
```

✔ Shell commands, redirection
✘ Escape special chars
✘ Deprecated

```
import subprocess
subprocess.call(“cmd args”, shell=True)
subprocess.call([“cmd”,”arg”])
```

- ✔ Lots of flexibility – recommended way to do it!

When you need the output...- os.popen

os_popen.py

```
#!/usr/bin/env python3
import os

stream = os.popen('ls -1')
my_files = stream.read()
stream.close()

print(my_files)
```

```
> ./os_popen.py
cgi-bin
Lecture6_py4bio.pdf
Lecture6_py4bio.pptx
os_popen.py
protcharge.html
python-notes-cds.ppt
reversec.html
simple_form.html
simple.html
sub_process.py
textarea.html
```

- ✔ As os.system() but stream is a file handle that can be used in the usual way

When you need the output...- subprocess

sub_process.py

```
#!/usr/bin/env python3
import subprocess

result = subprocess.check_output("ls -l", shell=True)
result = str(result, 'utf-8')

print(result)
```

✔ Lots of flexibility!

recommended way to do it!

```
> python sub_process.py
total 84
-rwxrwxr-x 1 pieterdb pieterdb 854 Oct 29 15:39 adder.py
-rwxrwxr-x 1 pieterdb pieterdb 422 Oct 24 14:21 cgi_1.py
-rwxrwxr-x 1 pieterdb pieterdb  44 Oct 24 14:21 cgitest.py
-rwxrwxr-x 1 pieterdb pieterdb 482 Oct 29 13:29 fieldstorage.py
-rwxrwxr-x 1 pieterdb pieterdb 422 Oct 24 14:21 getData.py
-rwxrwxr-x 1 pieterdb pieterdb 415 Oct 29 11:07 hello2.py
-rwxrwxr-x 1 pieterdb pieterdb 634 Oct 29 13:10 hello3.py
-rwxrwxr-x 1 pieterdb pieterdb 488 Oct 28 19:22 hello_get.py
-rwxrwxr-x 1 pieterdb pieterdb 251 Oct 28 20:17 hello.py
-rwxrwxr-x 1 pieterdb pieterdb 206 Oct 28 20:09 here.py
-rwxrwxr-x 1 pieterdb pieterdb 224 Oct 28 16:25 http_get_arguments.py
-rwxrwxr-x 1 pieterdb pieterdb 469 Oct 24 14:21 local_vars.py
-rwxrwxr-x 1 pieterdb pieterdb 708 Oct 29 14:27 moltype.py
-rwxrwxr-x 1 pieterdb pieterdb 278 Oct 29 08:34 on_the_fly.py
-rwxrwxr-x 1 pieterdb pieterdb 945 Oct 28 20:59 protcharge.py
-rwxrwxr-x 1 pieterdb pieterdb 595 Oct 29 16:04 reversec.py
-rw-rw-r-- 1 pieterdb pieterdb 358 Oct 24 14:21 simple_form.html
-rwxrwxr-x 1 pieterdb pieterdb 205 Oct 28 17:37 test_contents.py
-rwxrwxr-x 1 pieterdb pieterdb 518 Oct 29 10:26 textarea.py
-rw-rw-r-- 1 pieterdb pieterdb 192 Oct 29 13:58 ulify.py
-rwxrwxr-x 1 pieterdb pieterdb 854 Oct 24 14:21 web_app_1.py
```

Accessing external programs

- CGI scripts and the programs they spawn run as the 'nobody' user.
  - Search path and environment variables may well not be what you expect!

Accessing external programs

- Set any environment variables you need in your CGI script:

```
import os

os.environ[“varname”]=”value”
```

- Use the full path to any external programs
  - (possible exception of standard Unix-like commands)

Temporary files

- Often need to create temporary working files
- Must ensure that the filename is unique
  - More than one person could hit your web server at the same time!
- Use the process ID to ensure a unique filename

getpid.py

```
import os
filename = "/tmp/cgifile_" + str(os.getpid())

print("tempfile: %s" % (filename))
```

```
> python getpid.py
tempfile: /tmp/cgifile_56619
```

Temporary files

- May need to create a temporary file to return to the user
- Most web servers provide a directory in which such files can be written

Summary

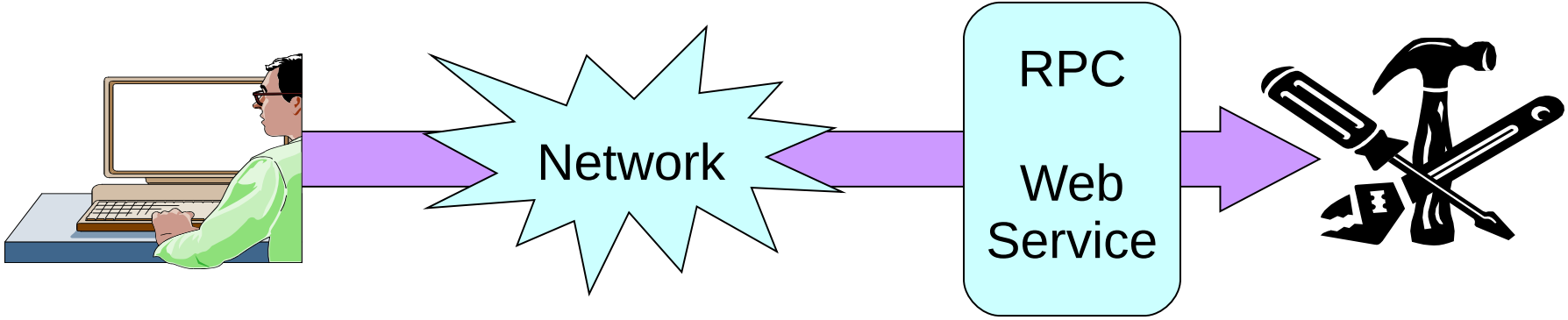
- Forms are used to send data to the web server
- GET and POST methods for transferring data
- CGI scripts can simply serve web pages
  - no data obtained from a form
- CGI scripts can obtain data from a page and run external programs

Remote Procedure Calling

Aims and objectives

- Understand the concepts of remote procedure calling and web services
- To be able to describe different methods of remote procedure calls
- Understand the problems of ‘screen scraping’
- Know how to write code using urllib.request

What is RPC?



A network accessible interface to application functionality using standard Internet technologies

Why do RPC?

- distribute the load between computers
- access to other people's methods
- access to the latest data

Ways of performing RPC

- screen scraping
- simple cgi scripts (REST)
- custom code to work across networks
- standardized methods
  - (e.g. CORBA, SOAP, XML-RPC)

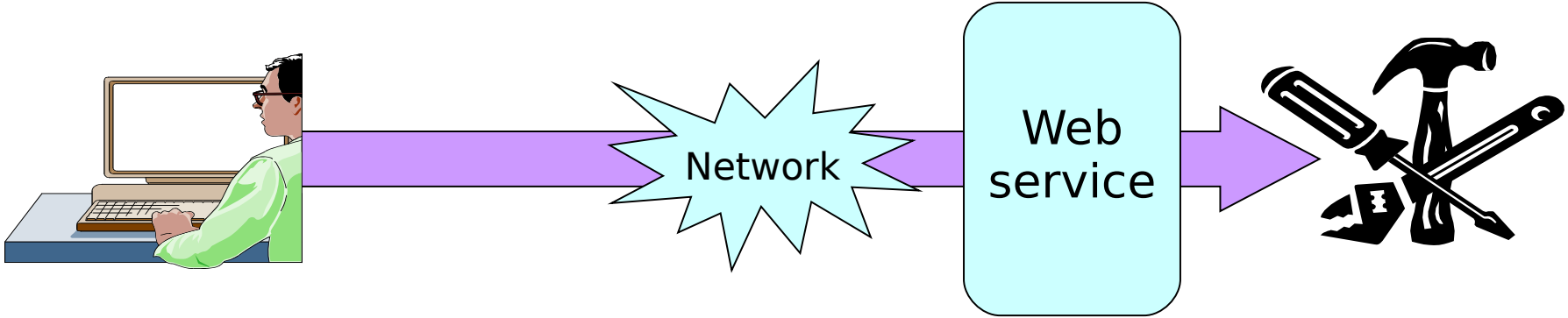
Web services

- RPC methods which work across the internet are often called “Web Services”
- Web Services can also
  - be self-describing (WSDL*)
  - provide methods for discovery (UDDI)

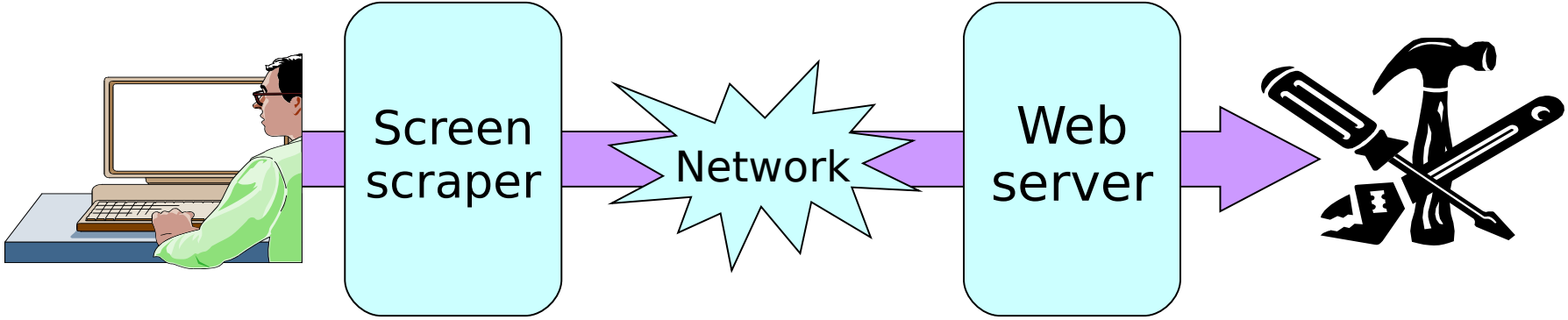
WSDL: Web Services Description Language
UDDI:Universal Description, Discovery, and Integration.

- Screen scraping

Web service

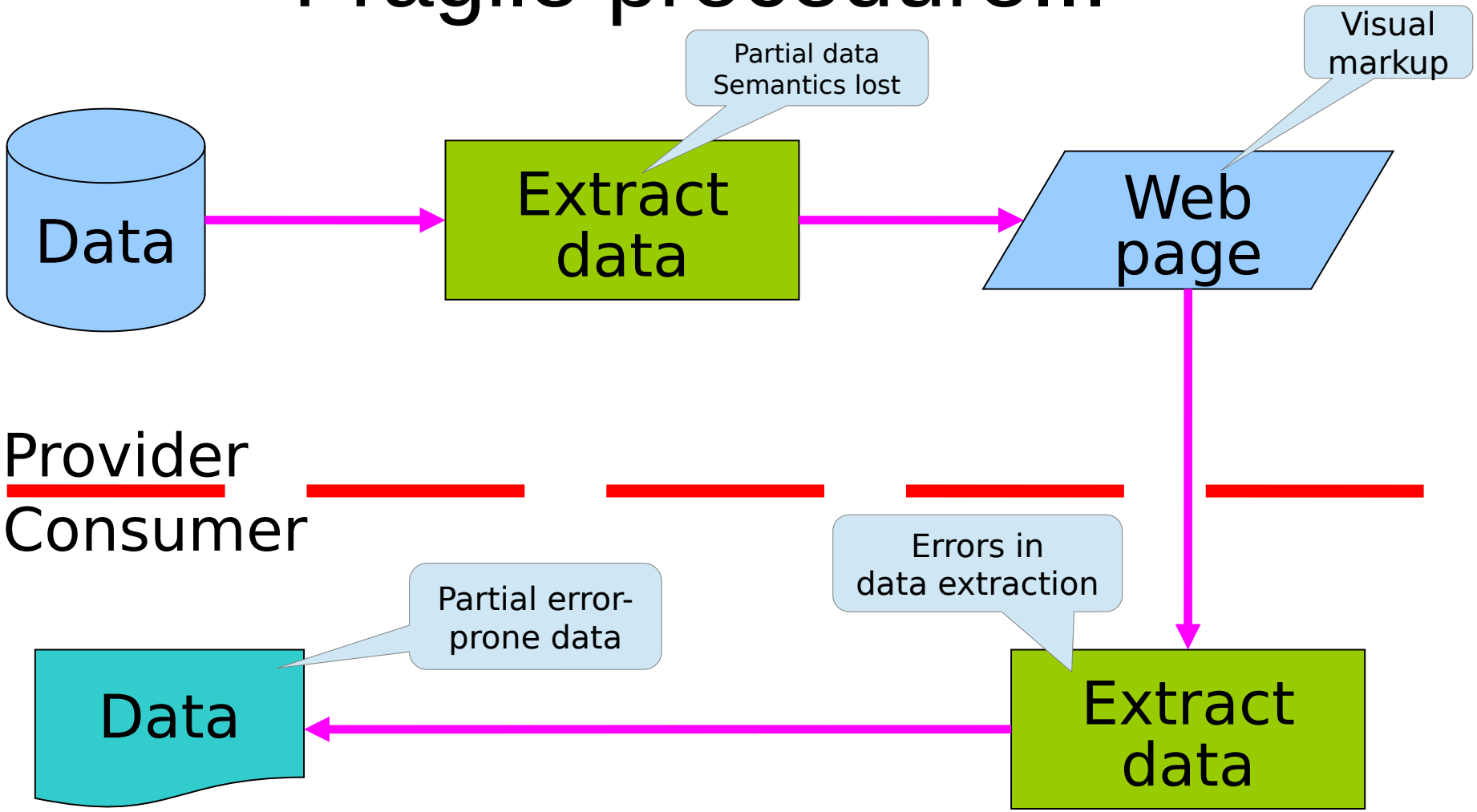


Screen scraping



- Extracting content from a web page

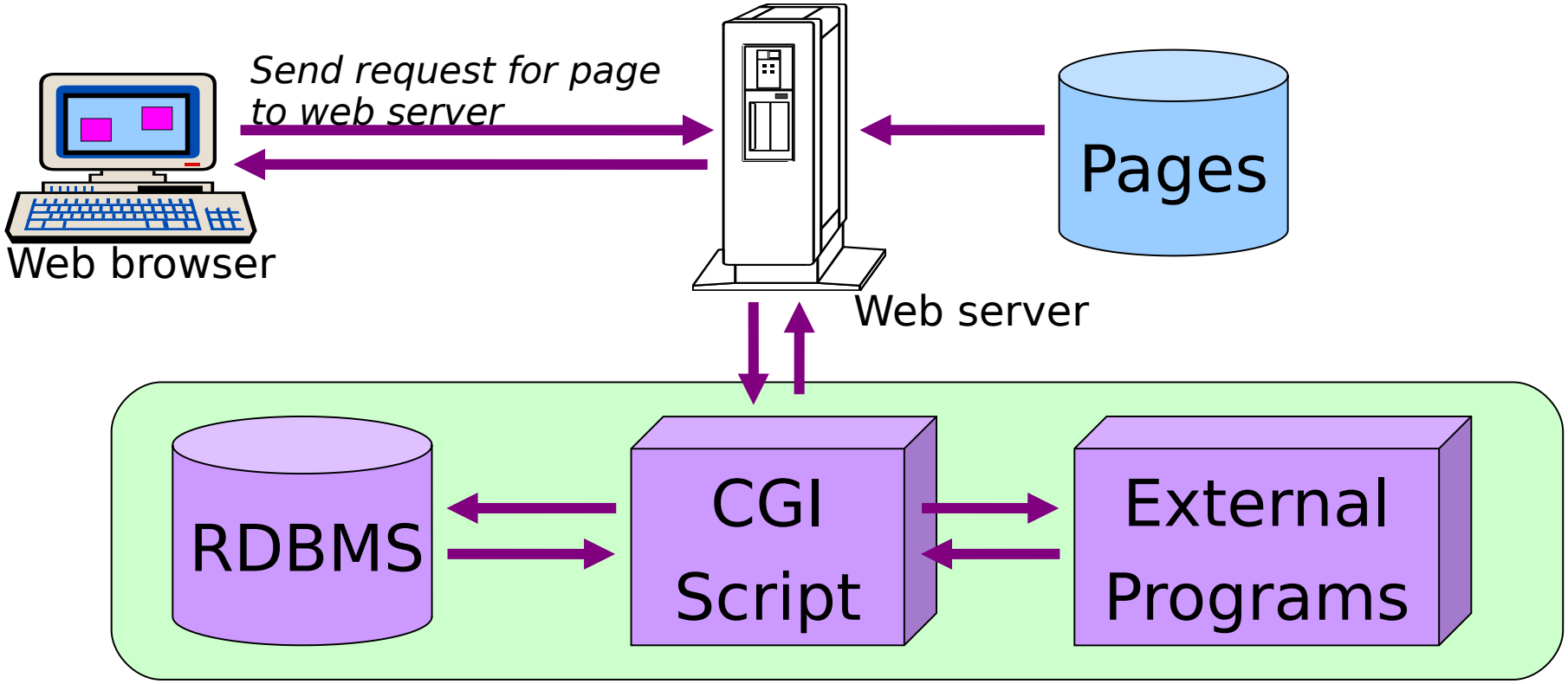
Fragile procedure...



Fragile procedure...

- Trying to interpret semantics from display-based markup!
- If the presentation changes, the screen scraper breaks!

Web servers…


Send request for page
to web server
Web browser
Pages
Web server
RDBMS
CGI
Script
External
Programs

Screen scraping

- Straightforward in Python!
- Python urllib.request module
  - trivial to write a web client
- Pattern matching and string handling routines

Example:
You have a human gene symbol e.g. SMAD3 and you want the list of reference sequence Ids (RefSeQ Ids) associated with it...

https://www.ncbi.nlm.nih.gov/gene/?term=SMAD3

https://www.ncbi.nlm.nih.gov/gene/?term=SMAD3

NCBI Resources How To

Gene | Gene | SMAD3

Create RSS Save search Advanced

Gene sources
Genomic

Categories
Alternatively spliced
Annotated genes
Non-coding
Protein-coding
Pseudogene

Sequence content
CCDS
Ensembl
RefSeq
RefSeqGene

Status clear
✓ **Current**

Clear all

Show additional filters

Tabular 20 per page Sort by Relevance

GENE

SMAD3 – SMAD family member 3

Homo sapiens (human)

Also known as: HSPC193, HsT17436, JV15-2, LDS1C, LDS3, MADH3

GeneID: 4088

RefSeq transcripts (6) RefSeq proteins (6) RefSeqGene (1) PubMed (807)

Orthologs | Genome Browser | BLAST | Download

RefSeq Sequences

Search results

Items: 1 to 20 of 1782

See also 7 discontinued or replaced items.

Example scraper…

Program must:

1. connect to web server
2. submit the gene symbol
3. obtain the results and extract data

Examine the source for the page…

Example scraper…

```
from urllib import request

def get_refseq_page(gene_symbol):
    url    = "https://www.ncbi.nlm.nih.gov/gene/"
    params = "term=" + gene_symbol
    fullurl= url + "?" + params

    result = request.urlopen(fullurl).read()
    result = str(result, encoding='utf-8')

    if(result != ''):
        return(result)
    else:
        sys.stderr.write("Nothing was returned\n")

    return("")

web_page = get_refseq_page("SMAD3")
print(web_page)
```

Example scraper…

Specify URL and parameters

```
from urllib import request

def get_refseq_page(gene_symbol):
    url    = "https://www.ncbi.nlm.nih.gov/gene/"
    params = "term=" + gene_symbol
    fullurl= url + "?" + params

    result = request.urlopen(fullurl).read()
    result = str(result, encoding='utf-8')

    if(result != ''):
        return(result)
    else:
        sys.stderr.write("Nothing was returned\n")

    return("")

web_page = get_refseq_page("SMAD3")
print(web_page)
```

Example scraper…

```
from urllib import request

def get_refseq_page(gene_symbol):
    url    = "https://www.ncbi.nlm.nih.gov/gene/"
    params = "term=" + gene_symbol
    fullurl= url + "?" + params

    result = request.urlopen(fullurl).read()
    result = str(result, encoding='utf-8')

    if(result != ''):
        return(result)
    else:
        sys.stderr.write("Nothing was returned\n")

    return("")

web_page = get_refseq_page("SMAD3")
print(web_page)
```

Retrieve data and convert from byte string

Example scraper…

```
from urllib import request

def get_refseq_page(gene_symbol):
    url    = "https://www.ncbi.nlm.nih.gov/gene/"
    params = "term=" + gene_symbol
    fullurl= url + "?" + params

    result = request.urlopen(fullurl).read()
    result = str(result, encoding='utf-8')

    if(result != ''):
        return(result)
    else:
        sys.stderr.write("Nothing was returned\n")

    return("")

web_page = get_refseq_page("SMAD3")
print(web_page)
```

return the HTML source code

Examine the source for the page…

```
…
<li>
    <a href="https://www.ncbi.nlm.nih.gov/nuccore/NM_005902.4,NM_001145102.1,NM_001145103.1,NM_001145104.1,XM_011521559.3,XM_011521560.1"
id="gene_refseqtranscripts" data-ga-action="click_feat_suppl" data-ga-label="RefSeq transcripts"
ref="discoId=KnownItemSensor&amp;itemType=gene">RefSeq transcripts</a>




         <span class="ncbi-text-light">(6)</span>


</li>


<li>
    <a href="https://www.ncbi.nlm.nih.gov/protein/NP_005893.1,NP_001138574.1,NP_001138575.1,NP_001138576.1,XP_011519861.1,XP_011519862.1"
id="gene_refseqproteins" data-ga-action="click_feat_suppl" data-ga-label="RefSeq proteins"
ref="discoId=KnownItemSensor&amp;itemType=gene">RefSeq proteins</a>




         <span class="ncbi-text-light">(6)</span>


</li>
…
```

Example scraper…

```
import re

def parse_refseq_page(web_page):
    html_lines = web_page.splitlines()
    for line in html_lines:
        if re.search(r'<a href="https://www.ncbi.nlm.nih.gov/nuccore/(.*)" id="gene_refseqtranscripts"',line):
            result = re.search(r'<a href="https://www.ncbi.nlm.nih.gov/nuccore/(.*)" id="gene_refseqtranscripts"',line).group(1)
    return(result)
```

Example scraper…

Create and match regular expression

```
import re

def parse_refseq_page(web_page):
    html_lines = web_page.splitlines()
    for line in html_lines:
        if re.search(r'<a href="https://www.ncbi.nlm.nih.gov/nuccore/(.*)" id="gene_refseqtranscripts"',line):
            result = re.search(r'<a href="https://www.ncbi.nlm.nih.gov/nuccore/(.*)" id="gene_refseqtranscripts"',line).group(1)
    return(result)
```

Example scraper…

```
import re

def parse_refseq_page(web_page):
    html_lines = web_page.splitlines()
    for line in html_lines:
        if re.search(r'<a href="https://www.ncbi.nlm.nih.gov/nuccore/(.*)" id="gene_refseqtranscripts"',line):
            result = re.search(r'<a href="https://www.ncbi.nlm.nih.gov/nuccore/(.*)" id="gene_refseqtranscripts"',line).group(1)
    return(result)
```

Extract the () group match

Pros and cons

- Advantages
  - 'service provider' doesn’t do anything special
- Disadvantages
  - screen scraper will break if format changes
  - may be difficult to determine semantic content

Simple CGI scripts

REST:
Representational State Transfer

http://en.wikipedia.org/wiki/REST

Simple CGI scripts

Extension of screen scraping
relies on service provider to provide a script designed specifically for remote access

Client identical to screen scraper
but guaranteed that the data will be parsable (plain text or XML)

Simple CGI scripts

Server's point of view

- provide a modified CGI script which returns plain text
- May be an option given to the CGI script

Simple CGI scripts

'**Entrez** programming utilities'
https://www.ncbi.nlm.nih.gov/books/NBK25501/

Search using EUtils is performed in
2 stages:
1.specified search string returns a set of PubMed Ids
2.fetch the results for each of these PubMed IDs in turn.

Check also biopython!!!

https://krother.gitbooks.io/biopython-tutorial/content/ncbi_eutils.html