

# Web programming with CGI.pm

Wednesday, October 24, 12

1

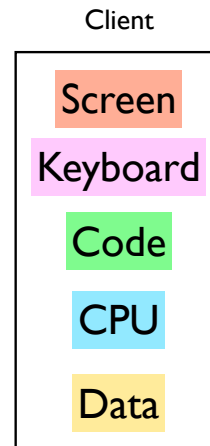
## What is CGI all about?

- Somewhat dry and abstract definition of CGI from Wikipedia: The Common Gateway Interface (CGI) is a standard method for web servers to delegate the generation of web pages to executable files. Such files are known as CGI scripts; they are programs, often stand-alone applications, usually written in a scripting language.
- Until now, you have run scripts from the command line.
  - Your scripts are run from a directory such as `~/perl/` or `~/scripts/` and the output is printed on the screen or to a file
- In web programming, scripts go in a directory that a web server can run scripts in like `~username/Sites/cgi-bin/` or `/Library/Webserver/CGI-Executables/` (on a Mac) and on a web server (depends on how the webserver is set up)
  - The output of scripts is text or HTML and is sent to a browser running on a client machine where it is rendered as a web page.
  - This set up allows you to create dynamic web pages that are generated in response to user input e.g. if the user enters a search query on a form on a web page, the search terms are sent to a CGI script which runs on the web server and returns the results of the search as HTML which is then displayed in the web browser exactly as if it was a web page.

Wednesday, October 24, 12

2

## Running scripts on your workstations.



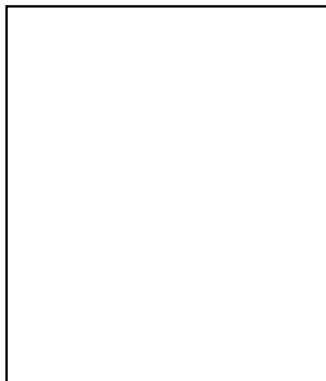
Wednesday, October 24, 12

3

## Running CGI scripts

Let's draw this diagram together

Web server



Client



Wednesday, October 24, 12

4

## Setting up and executing CGI scripts on your computers

- The web server Apache is running on the server.
- We will be writing CGI scripts in perl. They run as normal perl scripts, but have to go in a special directory.
- We have configured `~/Sites/cgi-bin/` to be the directory for CGI scripts (remember your home directories are on the (web)server too).

```
sprochnik% cd Sites/cgi-bin/
```

```
[coursemain:~/Sites/cgi-bin] sprochnik% ls -la
```

```
drwxr-xr-x+ 4 sprochnik  staff   136B Oct 23 14:49 .
drwxr-xr-x+ 7 sprochnik  staff   238B Oct 23 11:56 ..
-rwxr-xr-x+ 1 sprochnik  staff   474B Oct 23 14:49 boo.pl
```

- Save your CGI scripts in this directory.
- This directory has to be executable by 'other'. You can use `chmod +755 <dirname>` to do this.
- Your web scripts also have to be executable by 'other'. You can do this with `chmod +755 myscript.pl`

## Mapping a CGI script to a URL

- You will execute a CGI script by typing a URL that points to the script into a web browser and 'navigating' to the script.
- Let's look an example of how you translate a path on the file server into a URL in a web browser
- If you have a CGI script with the following path:  
`~sprochnik/Sites/cgi-bin/myCGIscrip.pl`
- (or, in full, `/Network/Servers/coursemain.cshl.edu/Volumes/cmain/INFOUsers/sprochnik/Sites/cgi-bin/myCGIscrip.pl`)
- can be executed by typing the following URL into a browser  
`http://coursemain.cshl.edu/~sprochnik/cgi-bin/myCGIscrip.pl`

## A CGI Script that Creates Plain Text

```
#!/usr/bin/perl
# file: plaintext.pl

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

print "When that Aprill with his shoures soote\n";
print "The droghte of March hath perced to the roote,\n";
print "And bathed every veyne in swich licour\n";
print "Of which vertu engendered is the flour...\n";
```

<http://mckay.cshl.edu/cgi-bin/course/plaintext.pl>

## A CGI Script that Creates HTML

```
#!/usr/bin/perl
# file: chaucer.pl

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

print "<html><head><title>Chaucer</title></head><body>\n";
print "<h1>Chaucer Sez</h1>\n";

print "When that Aprill with his shoures soote<br>\n";
print "The droghte of March hath perced to the roote,<br>\n";
print "And bathed every veyne in swich licour<br>\n";
print "Of which vertu engendered is the flour...<p>\n";

print "<cite>-Geoffrey Chaucer</cite>\n";
print "<hr>\n";
print "</body></html>\n";
```

<http://mckay.cshl.edu/cgi-bin/course/chaucer.pl>

## A CGI Script that Does Something Useful

A CGI script can do anything a Perl script can do, such as opening files and processing them. Just print your results to STDOUT.

```
#!/usr/bin/perl -w
# file: process_cosmids.pl
use strict;

my @GENES   = qw/act-1 dpy-5 unc-13 let-653 skn-1 C02D5.1/;
my $URL      = 'http://www.wormbase.org/db/gene/gene?name=';

print "Content-type: text/html\n\n";
print "<html><head><title>Genes</title></head><body>\n";
print "<h1>Genes</h1>\n";
print "<ol>\n";

for my $gene (@GENES) {
    print qq(<li><a href="$URL$gene">$gene</a>\n);
}

print "</ol>\n";
print "</body></html>\n";
```

[http://mckay.cshl.edu/cgi-bin/course/process\\_genes.pl](http://mckay.cshl.edu/cgi-bin/course/process_genes.pl)

## 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://stein.cshl.org/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:

Choose a Motif:

## Creating Fill-Out Forms II

---

### 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
- 

## Creating Fill-Out Forms III

### <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.

## Creating Fill-Out Forms IV

Examples:

<code>&lt;input type="text" name="motif1" value="TATTAT"&gt;</code>	<input type="text" value="TATTAT"/>
<code>&lt;input type="checkbox" name="motif2" value="TATTAT"&gt;</code>	<input type="checkbox"/>
<code>&lt;input type="radio" name="motif3" value="TATTAT" checked&gt;</code> <code>&lt;input type="radio" name="motif3" value="GGGGGG"&gt;</code>	<input checked="" type="radio"/> <input type="radio"/>
<code>&lt;input type="hidden" name="settings" value="PRIVACY MODE ON"&gt;</code>	
<code>&lt;input type="submit" name="search" value="SEARCH!"&gt;</code>	<input type="submit" value="SEARCH!"/>

## Creating Fill-Out Forms V

### **<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.





## What is CGI.pm?

1. Standard module in Perl distribution ( $\geq 5.004$ )
2. Emits correct HTTP headers
3. HTML shortcuts
4. Parses CGI parameters
5. "Sticky" form fields
6. Creates & processes cookies
7. File uploads

## Make HTML Beautiful

CGI.pm defines functions that emit HTML. The page is easier to read and write than raw HTML \*

```
<h1>
  Eat Your Vegetables
</h1>
<ol>
  <li>peas</li>
  <li>broccoli</li>
  <li>cabbage</li>
  <li>
    peppers
    <ul>
      <li>red</li>
      <li>yellow</li>
      <li>green</li>
    </ul>
  </li>
</ol>
<hr>
```

```
#!/usr/bin/perl
# Script: vegetables1.pl

use CGI ':standard';

print header,
  start_html('Vegetables'),
  h1('Eat Your Vegetables'),
  ol(
    li('peas'),
    li('broccoli'),
    li('cabbage'),
    li('peppers',
      ul(
        li('red'),
        li('yellow'),
        li('green')
      )
    ),
  ),
  hr,
  end_html;
```

\* if you speak Perl!

<http://mckay.cshl.edu/cgi-bin/course/vegetables.pl>

## Make HTML Concise

### Tag Functions are Distributive

```
print li('hi','how','are','you')

<LI>hi how are you</LI>

@items=('hi','how','are','you'); print li(\@items)

<LI>hi</LI>
<LI>how</LI>
<LI>are</LI>
<LI>you</LI>

print li(['hi','how','are','you'])

<LI>hi</LI>
<LI>how</LI>
<LI>are</LI>
<LI>you</LI>
```

### Add HTML Attributes Using Hash References

```
%opts=(-type=>'square'); @items=('hi','how','are','you'); print li(\%opts,\@items)

<LI TYPE="square">hi</LI>
<LI TYPE="square">how</LI>
<LI TYPE="square">are</LI>
<LI TYPE="square">you</LI>

print li({-type=>'square'},['hi','how','are','you'])

<LI TYPE="square">hi</LI>
<LI TYPE="square">how</LI>
<LI TYPE="square">are</LI>
<LI TYPE="square">you</LI>
```

```
#!/usr/bin/perl
# Script: vegetables2.pl

use CGI ':standard';

print header,
  start_html('Vegetables'),
  h1('Eat Your Vegetables'),
  ol(
    li(['peas',
        'broccoli',
        'cabbage',
        'peppers' .
        ul(['red','yellow','green']),
        'kolrabi',
        'radishes'])
  ),
  hr,
  end_html;
```

<http://mckay.cshl.edu/cgi-bin/course/vegetables2.pl>

## Using CGI.pm for the Genes Script

```
#!/usr/bin/perl -w
# file: process_genes2.pl

use strict;
use CGI ':standard';

my @GENES = qw/act-1 dpy-5 unc-13 let-653 skn-1 C02D5.1/;
my $URL = 'http://www.wormbase.org/db/gene/gene?name=';

my @list_items;
for my $gene (@GENES) {
  push @list_items, a({-href=>"$URL$gene"}, $gene);
}

print header(),
  start_html('Genes'),
  h1('Genes'),
  ol(
    li(\@list_items)
  ),
  end_html;
```

[http://mckay.cshl.edu/cgi-bin/course/process\\_genes2.pl](http://mckay.cshl.edu/cgi-bin/course/process_genes2.pl)

## Setting & Retrieving CGI Parameters

You can set and retrieve CGI parameters easily. In these examples, the CGI field string is:

```
banana=yellow&squash=green&tomato=red&tomato=green
```

### Retrieve a Single-Valued Field Named "Tomato":

Call *param()* with the name of the field and assign it to a scalar.

```
my $banana_color = param('banana');  
# yields "yellow"
```

This works for both GET and POST types, including *multipart/form-data*.

### Retrieve a Multi-Valued Field Named "Tomatoes":

Call *param()* with the name of the field and assign it to an array:

```
my @tomatoes = param('tomato');  
# yields ('red','green')
```

## Finding out What Parameters are Available

Call *param()* without any arguments. Will return the names of each of the parameters:

```
my @fields = param;  
# yields ('banana','squash','tomato')
```

## Setting Single- and Multivalued Fields:

```
param(-name=>'tomato', -value=>'red');  
param(-name=>'tomatoes',-value=>['red','green','blue']);
```

Parameters set in this way will be used as default values for fill-out form fields and hidden fields.

```
#!/usr/bin/perl
# file: final_exam.pl

use CGI 'standard';

print header;
print start_html("Your Final Exam"),
      h1("Your Final Exam"),
      start_form,
      "What's your name? ", textfield(-name=>'first_name'),
      p,
      "What's the combination?",
      p,
      checkbox_group(-name => 'words',
                    -values => ['eenie','meenie','minie','moe'],
                    -defaults => ['eenie','minie']),
      p,
      "What's your favorite color? ",
      popup_menu(-name => 'color',
                -values => ['red','green','blue','chartreuse']),
      p,
      submit,
      end_form,
      hr;

if (param()) {
    print
      "Your name is: ", param('first_name'),
      p,
      "The keywords are: ", join(" ", param('words')),
      p,
      "Your favorite color is: ", param('color'),
      hr;
}

print end_html;
```

## A Simple Form

### Your Final Exam

What's your name?

What's the combination?

☒ eenie ☐ meenie ☒ minie ☐ moe

What's your favorite color?

---

Your name is: Sheldon

The keywords are: eenie, minie

Your favorite color is: red

---

## Form Generating Functions I

Run *perldoc CGI* for details:

**start\_form**

Start the form (returns the <form> tag with default attributes).

**end\_form**

End the form by returning the </form> tag.

**textfield(-name=>\$name,-value=>\$starting\_value)**

Create a text field.

**password\_field(-name=>\$name,-value=>\$starting\_value)**

Create a password field.

**textarea(-name=>\$name,-value=>\$starting\_value,-rows=>\$rows,-cols=>\$cols)**

Create a multiline text input area.

**checkbox(-name=>\$name,-value=>\$value,-checked=>1)**

Create a single checkbox.

**checkbox\_group(-name=>\$name,-value=>\@values,-default=>\@on)**

Create a group of checkboxes sharing the same name. @values gives the list of checkbox values, and @on gives the list of those that are initially on.

## Form Generating Functions II

`radio_group(-name=>$name,-value=>\@values,-default=>$on)`  
 Create a group of radio buttons sharing the same name. @values gives the list of radio values, and \$on indicates which one is on to start with.

`popup_menu(-name=>$name,-value=>\@values,-default=>$on)`  
 Create a popup menu. @values gives the list of items, and \$on indicates which one is initially selected.

`scrolling_list(-name=>$name,-value=>\@values,-default=>$on)`  
 Create a scrolling list. @values gives the list of items, and \$on indicates which one (if any) is initially selected.

`submit(-name=>$name,-value=>$value)`  
 Creates a submit button. \$value optionally sets the button label.

```
#!/usr/bin/perl
# file: reversec.pl

use CGI 'standard';

print header;
print start_html('Reverse Complementation'),
      h1('Reverse Complementation'),
      start_form,
      "Enter your sequence here:",br,
      textarea(-name=>'sequence',-rows=>5,-cols=>60),
      submit('Reverse Complement'),
      end_form,
      hr;

if ( param ) {
    my $sequence = param('sequence');

    my $reversec = do_reverse($sequence);
    $reversec =~ s/(.{60})/$1\n/g; # do word wrap

    print h2('Reverse complement');
    print pre($reversec);
}

print end_html;

sub do_reverse {
    my $seq = shift;
    $seq =~ s/\s//g;          # strip whitespace
    $seq =~ tr/gatcGATC/ctagCTAG/; # complement
    $seq = reverse $seq;      # and reverse
    return $seq;
}
```

A reverse complementation script

### Reverse Complementator

Enter your sequence here:

AAAAAAAGATTGTGCTTCTCATCTCTCTC

Reverse Complement

### Reverse complement

GAGAGAGATGAGAAGCACAAATCTTTTTT

## File Uploading

HTML: `<INPUT TYPE="FILE">`      CGI.pm: `filefield()`

### Annoying complication:

You have to start the form with `start_multipart_form()` rather than `start_form()`.

Let's modify `reversec.pl` to support file uploads:

- First part (script too big for one page), print the form

```
#!/usr/bin/perl
# file: sequpload.pl

use CGI ':standard';

print header;
print start_html('Reverse Complementation'),
      h1('Reverse Complementator'),
      start_multipart_form,
      "Enter your sequence here: ".br,
      textarea(-name=>'sequence',-rows=>5,-cols=>60).br,
      'Or upload a sequence here: ' . filefield(-name=>'uploaded_sequence'),
      submit('Reverse Complement'),
      end_form,
      hr;
```

### sequpload.pl continued...

```
if ( param ) {
    my $sequence;

    # look for the uploaded sequence first...
    if ( my $upload = param('uploaded_sequence') ) {
        print h2("Reverse complement of $upload");

        while (my $line = <$upload>) {
            chomp $line;
            next unless $line =~ /^[gatcnGATCN]/;
            $sequence .= $line;
        }

    } else { # ... not found, so read it from the text field
        print h2('Reverse complement');
        $sequence = param('sequence');
    }

    $reversec = do_reverse($sequence);
    $reversec =~ s/(.{60})/$1\n/g; # do word wrap
    print pre($reversec);
}

print end_html;

sub do_reverse {
    my $seq = shift;
    $seq =~ s/\s//g;           # strip whitespace
    $seq =~ tr/gatcGATC/ctagCTAG/; # complement
    $seq = reverse $seq;       # and reverse
    return $seq;
}
```

If `param()` returns true, that means that we have some user input

### Reverse Complementator

Enter your sequence here:

Or upload a sequence here: `smckay/Desktop/myseq.txt`

### Reverse complement of myseq.txt

GAGAGAGATGAGAAGCACAACTCTTTTTT

<http://mckay.cshl.edu/cgi-bin/course/sequpload.pl>

## Adding Cascading Stylesheets

```
#!/usr/bin/perl -w
# Script: veggies_with_style.pl
use CGI ':standard';

my $css = <<END;
<style type="text/css">
li.yellow { color: yellow }
li.green { color: green }
li.red { color: red }
ol {
background-color: gainsboro;
padding: 5px;
margin-left: 200px;
width: 150px;
}
ul { background-color: black }
</style>
END

print header,
start_html( -title => 'Vegetables',
            -head => $css );

print
h1('Eat Your Vegetables'),
ol(
li(['broccoli', 'peas', 'cabbage']),
li('peppers',
ul(
li({-class => 'red'}, 'red'),
li({-class => 'yellow'}, 'yellow'),
li({-class => 'green'}, 'green')
)
)
),
hr,
end_html;
```

## Eat Your Vegetables

1. broccoli
2. peas
3. cabbage
4. peppers

◦ red  
◦ yellow  
◦ green

[http://mckay.cshl.edu/cgi-bin/course/veggies\\_with\\_style.pl](http://mckay.cshl.edu/cgi-bin/course/veggies_with_style.pl)

## External stylesheet

```
#!/usr/bin/perl -w
# Script: veggies_with_style.pl
use CGI ':standard';

my $css = '/css/veggies.css';

print header,
start_html( -title => 'Vegetables',
            -style => $css );

print
h1('Eat Your Vegetables'),
ol(
li(['broccoli', 'peas', 'cabbage']),
li('peppers',
ul(
li({-class => 'red'}, 'red'),
li({-class => 'yellow'}, 'yellow'),
li({-class => 'green'}, 'green')
)
)
),
hr,
end_html;
```

[http://mckay.cshl.edu/cgi-bin/course/veggies\\_with\\_style2.pl](http://mckay.cshl.edu/cgi-bin/course/veggies_with_style2.pl)



### CGI Exercises

#### Problem #1

Write a CGI script that prompts the user for his or her name and age. When the user presses the submit button, convert the age into "dog years" (divide by 7) and print the result.

#### Problem #2

Accept a DNA sequence and break it into codons.

Extra credit: Translate the codons into protein.

## Tools for grabbing web content

- perl module LWP.pm
- from the command line: wget, curl
  - `wget -o html.log -O googlecontent.html http://www.google.com`
  - `-o` is the log file
  - `-O` is the HTML output text file
  - `curl http://www.google.com > google.html`
  - redirect STDOUT to an output file