CS24210: Patterns in Perl

Edel Sherratt

22 October 2000

This practical is about using pattern matching in Perl programs to implement a simple sales assistant program. The sales assistant is accessed via the web, and makes use of the Perl module CGI.pm.

1 Setting up your web-access

1.1 `public_html`

Some of you will already have a `public_html` directory, and an index.html page in that directory. If you already have these, then add the following anchor point to your `index.html` file:

```
<a href="http://www.aber.ac.uk/cgi-bin/user/your_login/sales_assistant.pl">
A simple sales assistant</a>
```

where `your_login` is replaced by your own login.

If you do not already have a `public_html` directory, then create a directory called `public_html` in your home directory, and set its permissions so that it is executable by everyone.

Use `mkdir public_html` to create the directory, and `chmod 711 public_html` to make it executable by everyone.

Now download the default `index.html` file from the CS24210 web site, and place it in your new `public_html` directory. You'll have to change its name from `default_index.html` to `index.html` before it will work properly for you. **Only do all this if you don't already have an `index.html` file in your `public_html` directory.**

1.2 cgi-bin

Unless you've already got one, create a directory called `cgi-bin` in your home directory. Set its permissions so that it is executable by yourself; `chmod 700 cgi-bin` will do this.

2 A simple sales assistant

2.1 The greeting

Create a file in your cgi-bin directory, `egbert.pl`, containing the following Perl program.

```
#!/usr/local/bin/perl

use CGI qw/:standard/;

sub greet {
  print start_html,
    h4("How do you do.",
      p("I'm Egbert, your friendly sales assistant."),
      p("How may I help you?")),
  end_html;
}

print header;
greet;
```

Use `chmod 700 egbert.pl` to make this executable; then try running it from your browser window.

To do this, go you your home page,

`http://users.aber.ac.uk/your login/`

and click on the link 'A simple sales assistant'.

2.2 A more interesting assistant

Modify `egbert.pl` so that it looks like this:

```
#!/usr/local/bin/perl

use CGI qw/:standard/;

sub getOrder {
  print start_html,
    start_form,
      textarea(-name=>'order',-rows=>5,-cols=>60),
      p,
      submit(-name=>'submit',
        -value=>'Press here to submit your order.'),
    end_form,
  end_html;
}
```

```
sub processForm {
  print start_html,
    h4("You said: ", param('text'),
      p("Will that be all?")),
    startform,
       submit(-name=>'finished',-value=>'yes, thanks.'),
    endform,
    h4("Or would you like to change your order?"),
  end_html;
}

sub greet {
  print start_html,
    h4("How do you do.",
      p("I'm Egbert, your friendly sales assistant."),
      p("How may I help you?")),
  end_html;
}

print header;
if (param('submit')) {
  processForm;
} else {
  greet;
}
getOrder;
```

Run it as before.

Notice how when you press a submit button, the program is run again. (You could run a different program, but in this practical, the same program is run again).

Read the program carefully, and try to work out how the program 'knows' the button whose name is 'order' has been pressed.

Now try and work out how the text the user typed into the textbox in getOrder is made available to processForm during the next execution of the script (look at `param('text')` in processForm).

2.3 The Shopping List

Look at the file `egbert3.pl` on the CS24210 web pages.

Look at the subroutine shoppingList. Notice how a reference to the shopping_list hash table is passed as a parameter whenever shoppingList is called. How is this reference represented?

See if you can spot when shoppingList returns a value to its caller, and when it calls itself again.

See if you can describe in your own words what shoppingList does.

Add some new categories to the pricelist; for example, you might add 'confectionery' with an appropriate price. Modify shoppingList so that it charges an appropriate amount for, say, toffee and chocolate. You might also add the category 'dairy'; this could include things like milk, butter, eggs and cheese.

Add any other categories you think might be fun – but don't add anything that you wouldn't like your mother/father/employer to read!

3 Distinguishing between 'I want' and 'I don't want'

The subroutine `shoppingList` has an obvious peculiarity. If you type 'I don't want any tea', it will immediately add tea to your list!

Try to create a new version of Edgar that takes account of negatives.

Probably the simplest way to do this is to replace the assignment to `$shopping_list` by a call to a subroutine that adds an item to the shopping list only after checking that the item doesn't immediately follow a pattern like 'don't want a', 'don't need a', 'no', etc. The inputs to the subroutine include the string before the word just matched (`` $` ``), the word just matched (`$&`)l, its class (`$class`), and a pointer to the shopping list.

4 Further improvements

Think about you you might improve Edgar further.

Here are a few suggestions – simplest first.

You could make your program print out a message when the 'yes thanks' button is pressed.

You could make Edgar offer related items, when an order for an item has been recognised.

You could keep track of a stock level, and offer alternatives when an item is out of stock.

You could keep track of how many items are ordered, and add more than one of a given kind of item to the shopping list when the customer asks for several of the item.

You might also print a suitable message when a customer asks for something that isn't in the pricelist. (This is fairly tricky – you'll have to think about which words should trigger this response).

Have fun, but don't go completely overboard; full blown natural language processing is well beyond the scope of CS24210!

5 Important Note:

There are lots of security issues with CGI. Check out

`http://www.lanl.gov/projects/ia/library/bits/bits0396.html`

to find out more about this topic.