CS3157: Advanced Programming

Lecture #6
Feb 12
Shlomo Hershkop
shlomo@cs.columbia.edu
Announcement

- Monday 2/19
  - Review for midterm, wrapping up perl

- Wednesday
  - 2/21
  - MEET IN THIS CLASS
  - MIDTERM, open book/notes, no computers/phones
Today

- Cgi stuff
- Object oriented perl
CGI

- Hopefully you all feel comfortable with CGI
- Anyone do anything beyond the lab ??
- Types of data
Serving mp3 files

open(MP3FILE,"....") || die ....

my $buffer;
print "Content-type: audio/mp3\n\n";
binmode STDOUT;
while( read(MP3FILE, $buffer, 16384)){
  print $buffer;
}
Example

- http://..../cgi-bin/mp3server.cgi/Song.mp3
Argument passing

- Say you have a cool program which you can hook to the web....
  - Give a cell phone
  - Give a message
  - Will send the cell phone a message
<HTML><HEAD>
<TITLE>Cool</TITLE>
</HEAD>
<BODY>
<form action="cgi-bin/cool.cgi" method="GET">
<p>Enter cell phone to use: 
<input type="text" name="cellphone"></p>
<p>Enter Message: 
<input type="text" name="message"></p>
<input type="submit"/>
</form>
</BODY></HTML>
Use CGI;
my $coolp = '/usr/local/bin/cellmsg';

my $q = new CGI;
my $cell = $q->param(“cellphone”);
my $msg = $q->param(“message”);
# error checking here
open PIPE, “$coolp $cell $message |” or die “Can not open cellphone program”; 
print $q->header( “text/plain”);
print while <PIPE>
close PIPE;
What can go wrong?
When executing command can in theory pass in the following arguments

Something ; rm –rf *.*
Perl Taint mode

- T
  - Taints all data references (incoming)

- #!/usr/bin/perl -wT

- Flags data to make sure perl doesn’t do anything insecure
Tainted?

- STDIN
- CGI

- If variables/values are tainted
- Tainted follows it around with assignments

```perl
Sub is_tainted {
    my $var = shift;
    my $blank = substr($var,0,0);
    return not eval {
        eval "1 || $blank" || 1;
    };
}
```
Why

- Why would you want to keep track of tainted data?
Getting out of taint

- Match related patterns ($1,$2 ..)
- Idea: would check for security problems and then allow it

- Reminder: only in taint mode if set
References

- Reminder: A reference is a way of talking about a variable

- Symbolic Reference: a reference which we need to look up what it means in some other place

- Hard Reference: direct value of other object
Creating References

- Backslash (covered in the past)
  - $foo = 200;
  - $ref_foo = \$foo;
  - $constref = \312;
  - $sub_ref = \&somesub;
References to Arrays

- $ref = [0,1,2,3,];
  - how would you print the 1?

- $ref = [ [-1,1],45,78,[ ‘s’,’r’] ];
  - how would you print the ‘s’?
Hashes

- Remember that when you stick an anonymous list into a hash value (why not key?)

- How to use it?
sub references

- $subref = sub(print "hello\n");

- &$subref;
Arrow Operator

- -> infix operator for referencing
  - arrays
  - hashes
  - subroutines

- $$\text{arrayref}[2]$$ can be $$\text{arrayref}->[2]$$
- $$\&\text{subref}()$$ can be $$\text{subref}->()$$
Modules

- Idea: take a piece of code we are using a lot, and package it up so anyone can reuse it

- Advantages ??

- Disadvantages ?
Modules

- File with code:
  - something.pm

- Code which wants to use it:
  - use something;

- really::something::else
  - would be really/something/else.pm

- The last line of your module needs to be 1;
  - That will return a true
  - So that use yourmodule will evaluate correctly
Subroutines in Modules

- `something.pm` which defines `foo();`
- either
  - `something::foo();`
- or export it correctly in the module file
  - `@EXPORT = qw(foo);`
Warning

- Be careful about exporting!!

- Does anyone know the difference between overloading and overriding ??
Bad Example

- say your package defined a sub called isdir ......

- If you export it what will happen ?
Versions

- can also define a $VERSION scalar which tells Perl what your version is

- use Function 3.2.1.2;

- would check the Function.pm for version 3.2.1.2 or later
Objects

- Perl can be programmed in an object oriented fashion with objects/classes etc
Class declaration

- package Person;

- Will define the scope until the end of the current file
Perl OOP

- No way of doing strict encapsulation
- Expectation of good behavior

- Recycles concepts to get OOP
- you end up using references!
- Class is just a package
- Bless!! ties a reference with a class!
summary

- Objects = references
- Class = package
- Method = subroutine
Note

- Object’s sub get an extra initial argument passed in:
  - name of the class

- sub package resolved during compile stage

- method resolved during runtime
Method calls

- $name->foo()
  foo $name ()
  - example of instance method being called
- $result = Math->pi();
  - class method being called
$instance = new Math
  can have problems if local new or Math sub defined

$instance = new Math::
$instance = Math::new;
Construction

- objects are references
- not all references are objects
- marking a reference with package is called BLESSING

```perl
$obj = { }; bless($obj);  # current package
bless($obj, Math);
```
Person Example

any questions from last lab ??
How would we do a Fraction class?
methods

- Accessor and modifier methods are defined in the current package

- Use arrow notation to access methods

- Object reference => name of method()
constructor

```perl
sub new {
    my $self = {
        _firstName => undef;
        _lastName => undef;
    };
    bless $self, 'Person';
    return $self;
}
```
sub print {
    my ($self) = @_;

    #print info
    print $self->firstName . " " . $self->lastName;
}
Instantiating

- `my $shlomo = new Person();`
accessing

sub firstName {
    my ( $self, $firstName) = @_;

    $self->{_firstName} = $firstName if defined ($firstName);

    return $self->{_firstName};
}

Default sub

- AUTOLOAD
  - special sub which exists in the undefined package
  - when call an unknown sub will pass args and call AUTOLOAD
  - great for classes, if call something undefined
sub AUTOLOAD {
our $AUTOLOAD;
warn "attemp to call $AUTOLOAD failed\n";
}

- so can access regular @_
Inheritance

- Classes can be used to extend the functionality of a class using a process called inheritance

- @ISA
  - perl keyword showing inheritance
Inheritance

- Process of relating classes
- if you create an instance of class Z which is derived from X
- Zinstance->foo
  - will look in Z
  - then look in X
  - then ??
Multiple inheritance

- We can have X->Y->Z relationship (single inheritance)
- Can have mulitple
  - A->(B,C,D)
    - B->E
    - C->(F,G)
- So how to resolve foo when called on A??
- Current
- Parent
  - Left->Right
  - Recursively
  - Deep first
- Universal

Then same thing for AUTOLOAD for all above