Announcements

- please make sure you are making progress on the homework
  - any questions?
  - again, except md5, no cpan modules in general
  - try to organize your code for efficiency, TAs can help you with this on giving you feedback on your ideas (A or B faster etc)
Today

- wrap up Perl
  - packages and modules
  - OOP
  - Random stuff
- Please acquire a C book (Deitel & Deitel)

- reading:
  - perl object and packages

IMPORTANT

- if you miss something (might be glossing over some things)
  - please contact me about it
  - try to also find it in the index of your book

- Hard to learn unless you ask
Issues

- character encoding issues
- cgi isssues

Side Note: Line Endings

- Carriage return \r
- Line Feed \n
CRLF
- Unix – LF (\n) CR (\r)

- print “Content-type: text/html\n\n”
- Why not \n\r\n\r ????
Updates

- Ranges are only in the positive direction

- \([ 5 .. 1]\)
  - null list returned
  - So how to get it?

Using Perl Libraries
Digest::HMAC - Perl implementation of RFC 2104 HMAC algorithm

SYNOPSIS

```
# Functional style
use Digest::HMAC;  
my $hash = Digest::HMAC::hash('md5', 'my secret key', 'my data');

# XOR style
use Digest::HMAC::xor;  
my $xor = Digest::HMAC::xor('md5', 'my secret key', 'my data');
```

DESCRIPTION

This module provides a functional interface to the RFC 2104 HMAC algorithm, allowing for easy integration into Perl scripts. The HMAC (Hash-based Message Authentication Code) is a cryptographic construction that combines a cryptographic hash function with a symmetric key cipher. It is commonly used for message authentication to ensure data integrity and authenticity.

The module supports multiple hash algorithms, including MD5, SHA-1, SHA-256, and SHA-512, allowing for flexibility in choosing the appropriate level of security for different applications. The HMAC algorithm is particularly useful for scenarios where both the integrity and authenticity of messages need to be verified, such as in secure communication protocols and data exchange operations.

For more detailed information and examples on how to use the module, refer to the official documentation or the CPAN distribution.
GUI

- There are easy ways to make graphics in Perl
- Will not cover in this course
  - But will have enough knowledge to pick this up on your own if you choose
  - Better way: will see later today

---

Graphics

```
#!/c: \perl\bin
use Tk;

my $mwin = MainWindow->new;

$mwin->Button(-text => "Hello World!", -command => sub{exit})->pack;
MainLoop;
```
Graphics

- Good to know about
- Might need to one day debug someone else’s code (GASP!)

Computer Security

- System and theory of ensuring the confidentiality, integrity, availability, and control of electronic information and systems.
  - Network
  - Host
  - Data
For host based security

- Want to ensure permission system
  - X should only be allowed to do A, B, and C
- Want to ensure accountability
  - If Y does something not allowed, should be noted
- Want to be able to track
  - If something has been tampered with, how can we locate it
  - Both preventative and reactionary

Homework Project

- Assuming you are a system administrator or just paranoid
- Take chronological snapshots of your system to compare and find changes
  - Many changes by system
  - Many changes by valid user
  - Might locate malicious user/system changes
- Want to search filenames
- Want to organize snapshots of system
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

- $arrayref[2] can be $arrayref->[2]
- &$subref() can be $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
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-&gt;foo()
  
  foo $name ()
  
  example of instance method being called

- $result = Math-&gt;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

$obj = { };
bless($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()
sub print {
    my ($self) = @_;

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

---

Instantiating

my $shlomo = new Person();
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

- 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

**Garbage Collection**

- High level languages do the garbage collection for you
- When leave a scope, all local variables are freed and their memory recycled
- How do you decide who to free ??
Reference counting

- An object can keep track of how many references are pointing to it
  - How can we do this??
  - used on unix file systems

Problem

```perl
{  my ($a, $b);
    $a = \$b;
    $b = \$a;
}
```
Worse

```perl
{  
    my $a;
    $a = \$a;
}

solutions ??
```

Switch

- Back to CGI

- review: What is CGI?

- How do we write CGI applications?
Interacting

- GET
  - HTTP request directly to the cgi script by appending the URL
- POST
  - HTTP request in content of message, i.e. it is stdin to your script
- Format of GET (default):
  - Value=key separated by &
  - Space replaced by +
  - URL conversion characters

Input Tag

- Each field is in an input tag
- Type
  - Text
  - Radio button
  - Checkbox
  - Pull down menus
  - etc
- Name
  - Symbolic name (so can recognize it)
- Value
  - Default value, or what the user will end up typing
Note: Encoding

- Spaces are turned to +
- & separates field
- Special characters are turned into %?? (hex)
  - "(" is %28
  - So “class is great” = “class+is+great”

others

- Submit buttons
  - `<input type="submit">`
- Reset buttons
  - `<input type="reset">`

- Value will change the default name on the button
- try not to trick user....
Decoding Form Input

1. \$ENV\{QUERY\_STRING\}
2. If( \$ENV\{REQUEST\_METHOD\} eq POST) 
   { read \$ENV\{CONTENT\_LENGTH\}\}
3. Split pairs around &
4. Split keys and values
5. Decode URL
6. Remember key,values

Drawback

- A lot of work
- Pain if we have multiple values associated with one key
- Must be easier way.....
- CGI.pm
  - Included after 5.003_07+
CGI.pm

- Allows you to handle cgi in a standard format
- Can save and load key,value pairs to standard file
- Helps in creating html documents to the server by streamlining certain operations and keeping it in an object oriented design
The bad news

- Can’t use it in this class for our labs, since I want to teach how it’s done on the low level
- feel free to write your own test stuff with it
- understand what is happening on object level
- Want you to practice doing it the manual way...better for learning and later CGI + C/CPP

Summary: CGI

- Minimum the web server needs to provide to allow an external process to create WebPages.

- Goal: responding to queries and presenting dynamic content via HTTP.
File handling

- We covered basic file handling

- How does this change over the web?

File Locking

use Fcntl "flock";

open FILE, "?????.txt" or die $!

#one of these
flock FILE, LOCK_EX;
flock FILE, LOCK_SH;
.....
flock FILE, LOCK_UN;
Serving mp3 files

```perl
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
<HTML>
<HEAD>
<TITLE>Cool</TITLE>
</HEAD>
<BODY>
<form action="cgi-bin/cool.cgi" method="GET">
<p>Enter cell phone to use:</p>
<input type="text" name="cellphone"/>
<p>Enter Message:</p>
<input type="text" name="message"/>
<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 $msg |" 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

Other issues

- Remember with each user, your perl script is being instantiated and executed

- In general might want to be able to run alongside yourself (not only in web context).
  - How do we share a variable between instances (to pass information)?
Command shell

- A better way of executing command shell arguments to a program is to divide the work
- Create an instance of the program you want to run
- Pass arguments directly to it, instead of using the command shell (where can combine multiple commands)

fork/exec

my $pid = open PIPE, "-|";
die "problem forking $!" unless defined $pid;

unless($pid) {
  exec COOL, $message or die "cant open pipe $!";
}
Some more background

- When you work with CGI, many times you have to work with specific formats and files
- Need to know how it will be handled on client side
- One such common file, is graphics..

Graphics

- Formats:
  - GIF (Graphic Interchange Format)
    - 256 colors
    - LZW compression
    - Animation
    - Transparent bit
  - PNG (Portable Network Graphic)
    - 256 color / 16-bit gray / 48-bit true color
    - NOT LZW
    - Alpha channels
    - Interlacing algorithms
• JPEG (Joint Photographic Expert Group)
  ■ 24-bit color
  ■ Lossy compression
  ■ No animation/transparency

• PDF (Portable Document Format)
  ■ Postscript language for document layout

---

**Image manipulation**

• Many packages in perl to work with image data

• GD
  ■ Lightweight package
  ■ Port of c graphics library
  ■ Manipulation routines for PNG
CGI

- CGI is a common framework
- Perl is not the only player
- We will also be doing CGI + PERL|C|CPP

Alternatives

- ASP
  - Created by Microsoft for its servers
  - Mix code into html
  - Visual basic/javascript
- PHP
  - Apache webserver
  - Similar to perl
  - Embed code in html
Alt II

- Coldfusion
  - Webserver interprets std coldfusion call embedded in html, and can add code to run custom functions
  - Windows, and linux
- Java servelts
  - Compiled java classes invoked by web client
  - Code creates documents
- FastCGI
  - Threaded instance of perl continuously running to help cgi perl run faster
- Mod_perl
  - Appache server perl thread to make perl cgi faster

Text handling

- One of the exciting developments in the last decade of computer science is data processing/mining/learning
- Many other area in and out of CS need data to be analyzed or presented in some (controlled but arbitrary fashion)
Handling data

- Using chiseled stone
- By hand (literally copy paste)
- Early mechanics (typwriters)
- Take 3157 😊

Outputting text

- Many times will have multiple fields per line
- Arbitrary delimiters:
  - Comma
  - Tabs
  - Pipe |
- Make sure whatever you choose
  - Is either not/can’t be present in the data
  - What if it is? How to represent these delimiters ??
**Approach**

- Memory vs disk based handling
- Brute force
- Divide and conquer
- Regexp is your friend

**Ahead!**

- Because CGI/Internet involves network based thinking, I will illustrate a quick example now.
- we will be doing this in next lab
Socket

- In order to communicate across computer networks (or between processes on the same computer)
- Need to decide on rules of communication
  - Language of protocol
  - Directional vs bi-directional
  - Throwaway or continuous
  - Life time of communication
  - Overhead
  - Priority
  - Location

IO:Socket client

see: http://search.cpan.org/~gbarr/IO-1.2301/IO/Socket/INET.pm

Use IO::Socket::INET;

$socket = IO::Socket::INET->new(
  PeerAddr => $remote_host,
  PeerHost => $remote_port,
  Proto => "tcp",
  Type => SOCK_STREAM) or die...
#writing out
print $socket "hello World";

#notice treatment of handle
$answer = <socket>;

close($socket);

Server version

my $server = IO::Socket::INET->new(
    LocalPort=> $portnum,
    LocalAddr => 'localhost',
    Proto => 'tcp',
    Reuse => "1",
    Listen => "10")
    or die "could not start server on port $portnum ....\n
while($client = $server->accept()) {
    ...
}
Next

- understand slides
- start homework
- have fun on the lab