

# **Day 10: Modules**

Suggested Reading:  
Learning Perl (4<sup>th</sup> Ed.),  
Chapter 15: Perl Modules

# Reminders

- Turn in homework at **START** of class
- Writing code is **fun!**
  - Write at least a little every day
  - The more you do, the easier it gets
- When in doubt, ask questions!
- But only Zathras have no one to talk to. No one manages poor Zathras, you see. So Zathras talks to dirt. Sometimes talks to walls, or talks to ceilings. But dirt is closer. Dirt is used, through everyone walking on it. Just like Zathras, but we've come to like it. It is our role. It is our destiny in the universe. So, you see, sometimes dirt has insects in it. And Zathras likes insects. Not so good for conversation, but much protein for diet. Ha! Zathras fix now, this way.

# What exactly are modules?

- Perl: A self-contained chunk of \$language code that can be used by a \$language program or by other \$language modules.
  - Akin to a C library, or a C++ class.
  - We've been using 'strict' and 'warnings' throughout this class

## Language Comparisons: Perl

- Perl
  - Very large collection of built-in functions
  - Very large collection of standard modules
  - Many add-on modules available via CPAN
  - You've been using modules for a while:
    - `use <module>;`

# Language Comparisons: Python

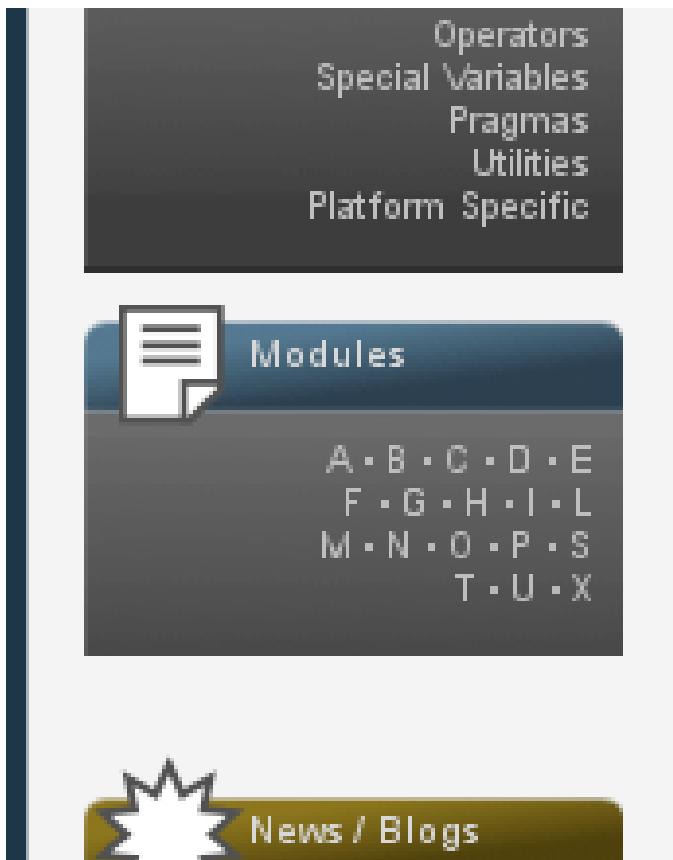
- Python
  - Minimalist set of built-in functions
  - Large collection of standard modules
  - Much functionality requires inclusion of modules:
    - `import <module>`
    - `from <module> import *`

# Language Comparisons

- Ruby
  - Large collection of built-in functions
  - Large collection of standard modules
  - Most modules are already imported for you
  - For others:
    - use <module>;

# Finding Perl Modules

- Look through the standard modules:
  - <http://perldoc.perl.org/>



If you are new to Perl, the [FAQ](#) section, which

## What's new?

As you can see, perl has been tweaked and updated

- **Improved**  
When you select a module, a breadcrumb trail will appear at the top of the page.
- **Pop-up info**  
Documentation for each module will now open in a pop-up window.
- **Improved**  
It's now even easier to get to the Getopt::Std documentation.

## Example #1 - File::Basename

```
#! /usr/bin/env perl
use strict;
use warnings;
use File::Basename;

die "usage: example-01 filepath\n" unless scalar(@ARGV);

my $full = shift( @ARGV );

my ($filename, $dir, $suffix);
($filename, $dir, $suffix) =
    File::Basename::fileparse( $full );
print "-> '$filename' '$dir', '$suffix'\n";

($filename, $dir, $suffix) =
    fileparse( $full, qr/.(py|pl)/ );
print "-> '$filename' '$dir', '$suffix'\n";
```

## Example #2 - File::Basename

```
#! /usr/bin/env perl
use strict;
use warnings;
use File::Basename;

die "usage: example-02 filepath\n" unless scalar(@ARGV);
my $full = shift( @ARGV );

sub fileparse { return ( shift(@_), ".txt", "", "" ); }

my ($filename, $dir, $suffix);
($filename, $dir, $suffix) =
    File::Basename::fileparse( $full );
print "-> '$filename' '$dir', '$suffix'\n";

($filename, $dir, $suffix) =
    fileparse( $full, qr/.(py|pl)/ );
print "-> '$filename' '$dir', '$suffix'\n";
```

## Example #3: Time / Date Parsing

```
#! /usr/bin/env perl
use strict;
use warnings;
use Time::ParseDate;

my $TimeStr = scalar(@ARGV) ? shift(@ARGV) : `/bin/date`;
chomp $TimeStr;
my $Seconds = parsedate( $TimeStr );
print "$TimeStr = $Seconds\n";
print "= " . localtime( $Seconds ) . "\n";
```

# Much much more

- Trying to find a module to solve a particular problem???
  - Google is your friend

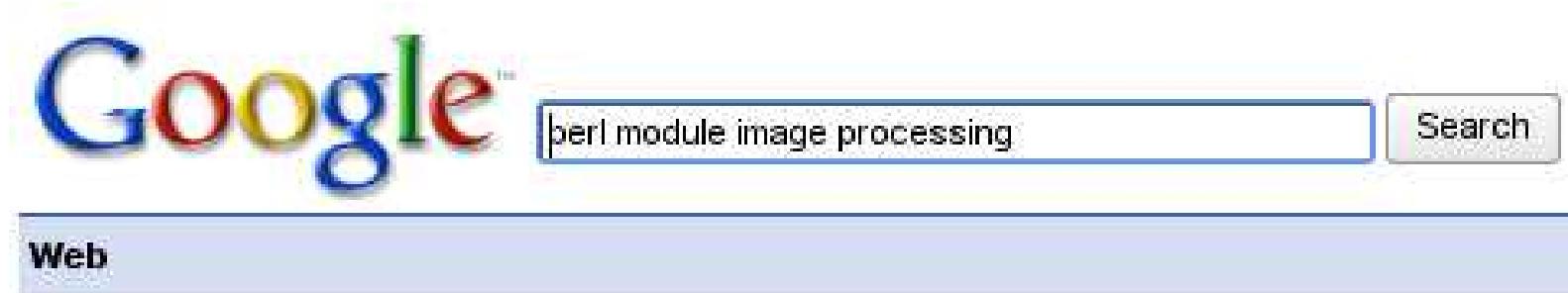


image processing using perl

in Perl without using any extra **modules** like Image::Magick or GD. ... **Image**

manipulation generally requires quite a bit of **processing** so, ...

[http://www.perlmonks.org/%3Fnode\\_id%3D635016](http://www.perlmonks.org/%3Fnode_id%3D635016) - 30k - Cached - Similar pages

Image processing in Perl graphic applications

# Simple Script

- Dumps out the IP address of all interfaces

```
#!/usr/bin/env perl
use strict;
use warnings;

open( IFCONFIG, "/sbin/ifconfig|" )
    or die "Can't run ifconfig";
my $Adaptor = "";
while( <IFCONFIG> ) {
    if( /^(\w+)\s+Link/ ) {
        $Adaptor = $1;
    }
    elsif ( /inet addr:(\d+\.\d{3}\.\d+)/ ) {
        print "$Adaptor: $1\n";
    }
}
```

# Subroutine

```
sub GetInterfaces()
{
    open(IFCONFIG, "/sbin/ifconfig|") or die "Can't run ifconfig";
    my $Adapter = "";
    my %Adapters;
    while( <IFCONFIG> ) {
        if( /^(\w+)\s+Link/ ) {
            $Adapter = $1;
        }
        elsif( $Adapter ne "" and /inet addr:(\d+\.\d{3})\d+/ ) {
            $Adapters{$Adapter} = $1;
        }
    }
    return \%Adapters;
}
```

# The Main Program...

```
#! /usr/bin/env perl
use strict;
use warnings;

die "usage: example-05 fname" unless scalar(@ARGV) == 1;
my $Name = $ARGV[0];
my $All = GetInterfaces();
if( exists $All->{$Name} ) {
    print "$Name => $All->{$Name}\n";
}
else {
    print "$Name not found\n";
}

sub GetInterfaces() ....
```

# Subroutine

```
sub GetInterfaces()
{
    open(IFCONFIG, "/sbin/ifconfig|") or die "Can't run ifconfig";
    my $Adapter = "";
    my %Adapters;
    while( <IFCONFIG> ) {
        if( /^(\w+)\s+Link/ ) {
            $Adapter = $1;
        }
        elsif( $Adapter ne "" and /inet addr:(\d+\.\d{3})\d+/ ) {
            $Adapters{$Adapter} = $1;
        }
    }
    return \%Adapters;
}
```

# Making a Module

```
#! /usr/bin/env perl
use strict;
use warnings;

my %Adapters;
sub InitializeInterfaces()
{
    open(IFCONFIG, "/sbin/ifconfig|") or die "Can't run ifconfig";
    my $Adapter = "";
    while( <IFCONFIG> ) {
        if( /^(\w+)\s+Link/ ) {
            $Adapter = $1;
        }
        elsif( $Adapter ne "" and /inet addr:(\d+\.\{3}\d+)/ ) {
            $Adapters{$Adapter} = $1;
        }
    }
}
sub GetInterfaces()
{
    return \%Adapters;
}
1;
```

# Using your new Module

- A program to use this new module:

```
#!/usr/bin/env perl
use strict;
use warnings;
use example_06;

die "usage: example-06 ifname" unless scalar(@ARGV) == 1;
my $Name = $ARGV[0];
InitializeInterfaces();
my $All = GetInterfaces();
if( exists $All->{$Name} ) {
    print "$Name => $All->{$Name}\n";
}
else {
    print "$Name not found\n";
}
```