CSCI-GA.3033.003 Scripting Languages

9/20/2013

Context and Modules (Perl)
Scripting as Glue

Outline

- Programming in the large
- Scripting as glue

Modules

- Module = file **`p/q.pm`** that starts with declaration **`package p::q`**
  - Group library functions for reuse
  - E.g., **`Math::BigInt`**, arbitrary precision arithmetic
- Module import: access with unqualified names
  - Compile-time **`use p;`** runtime **`require p;`**
  - Disable locally: **`no p;`**
- Pragma = pragmatic module
  - Changes Perl behavior in fundamental way
  - E.g., **`strict`**, forces variable declarations

Module Example

```
use strict; use warnings;                          # put this code in a
package apple;                                     # file apple.pm
sub create {
  my ($weight, $color) = @_;
  return { WEIGHT => $weight, COLOR => $color };
}
sub pluck {
  my $ref_to_hash = $_[0];
  return $ref_to_hash->{COLOR} . " apple";
}
sub prepare {
  my ($ref_to_hash, $how) = @_;
  return $how . "d " . pluck($ref_to_hash);
}
1                                                  # return true = success
```

```
#!/usr/bin/perl
use strict; use warnings; use apple;               # note the "use apple"
our $fruit = apple::create(150, "red");            # from a different file
print apple::prepare($fruit, "slice"), "\n";       # "sliced red apple"
```

Pluggable Type Systems

- User chooses between multiple alternative optional type systems, e.g., in Perl:
  - **`use strict "vars";`** no implicit declarations
  - **`use strict "refs";`** no string→ref conversion
  - **`use strict "subs";`** no bareword strings
  - **`use warnings;`** same as **`perl -w`**
  - **`perl -T`** taint-flow checking
- Motivation: trade-off between ease of writing, maintainability, robustness, and security
  - Similar to "Option Explicit" in VBA

Structure of a Perl Application

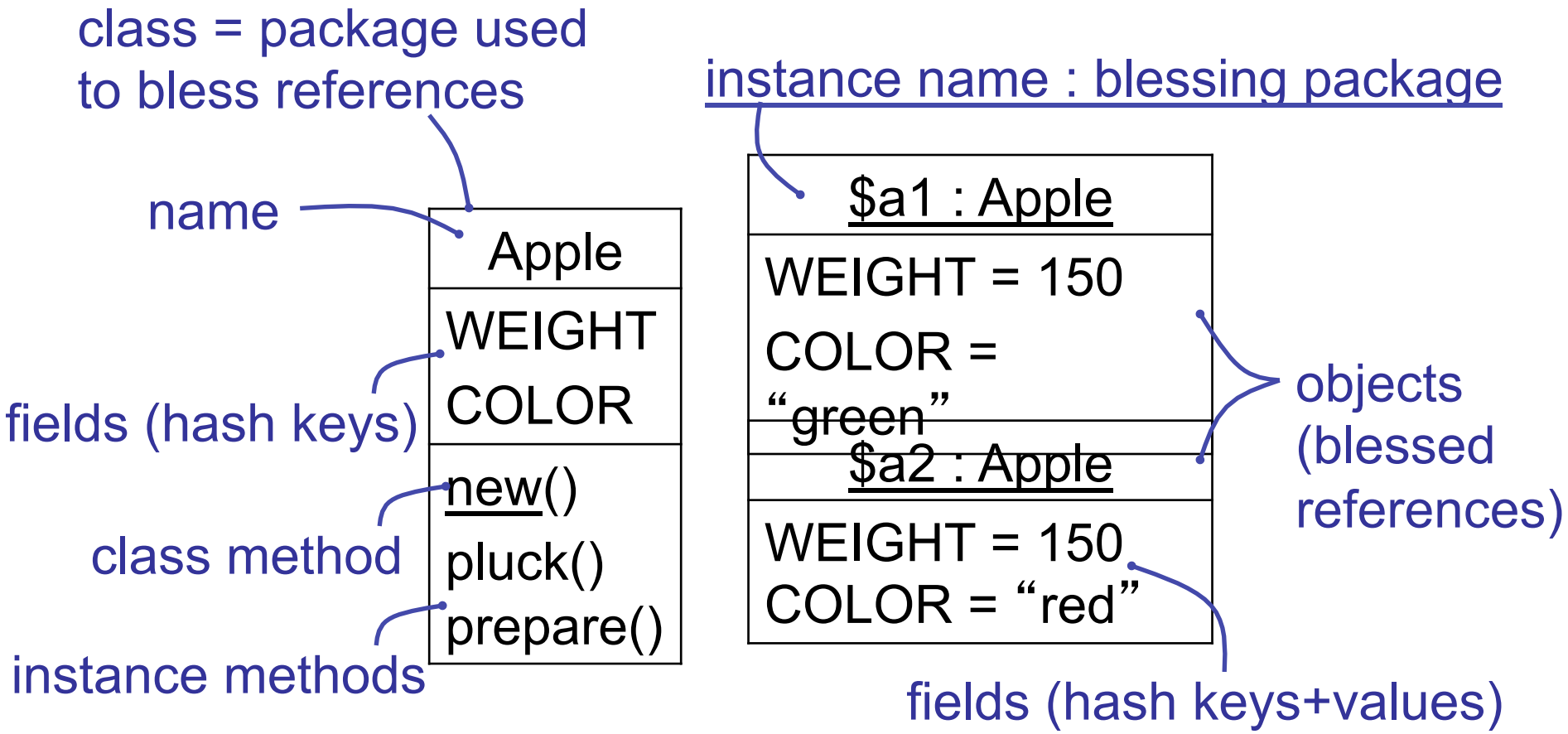
Compilation unit	Usually, file; also, **`-e`**, **`eval`** arg
Package	Named global namespace may vary by block; usually: one per file
Module	**`p/q.pm`** file with package **`p::q`**
Pragma	Module that changes language
Class	Package used to bless reference
Subroutine	Named subroutines don't nest
Statements	In subroutine or directly in file

Using Objects

`#!/usr/bin/perl` `use strict; use warnings; use Apple;`	*Import class module*
`our $a1 = new Apple(150, "green");` `our $a2 = Apple->new(150, "green");`	*Constructor call forms (indirect obj vs. arrow)*
`$a2->{COLOR} = "red";`	*Set hash entry*
`print $a1->prepare("slice"), "\n";` `print $a2->prepare("squeeze"), "\n";`	*Method call (arrow form)*

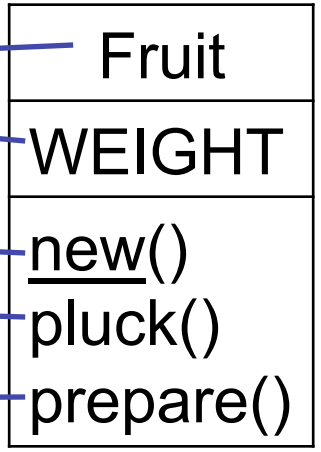
Concepts

Classes and Objects



Defining Classes

```
use strict; use warnings; package Fruit;
sub new {
  my ($cls, $weight) = @_;
  return bless({ WEIGHT => $weight }, $cls);
}
sub pluck {
  my $self = $_[0];
  return "fruit(" . $self->{WEIGHT} . "g)";
}
sub prepare {
  my ($self, $how) = @_;
  return $how . "d " . $self->pluck();
}
1
```

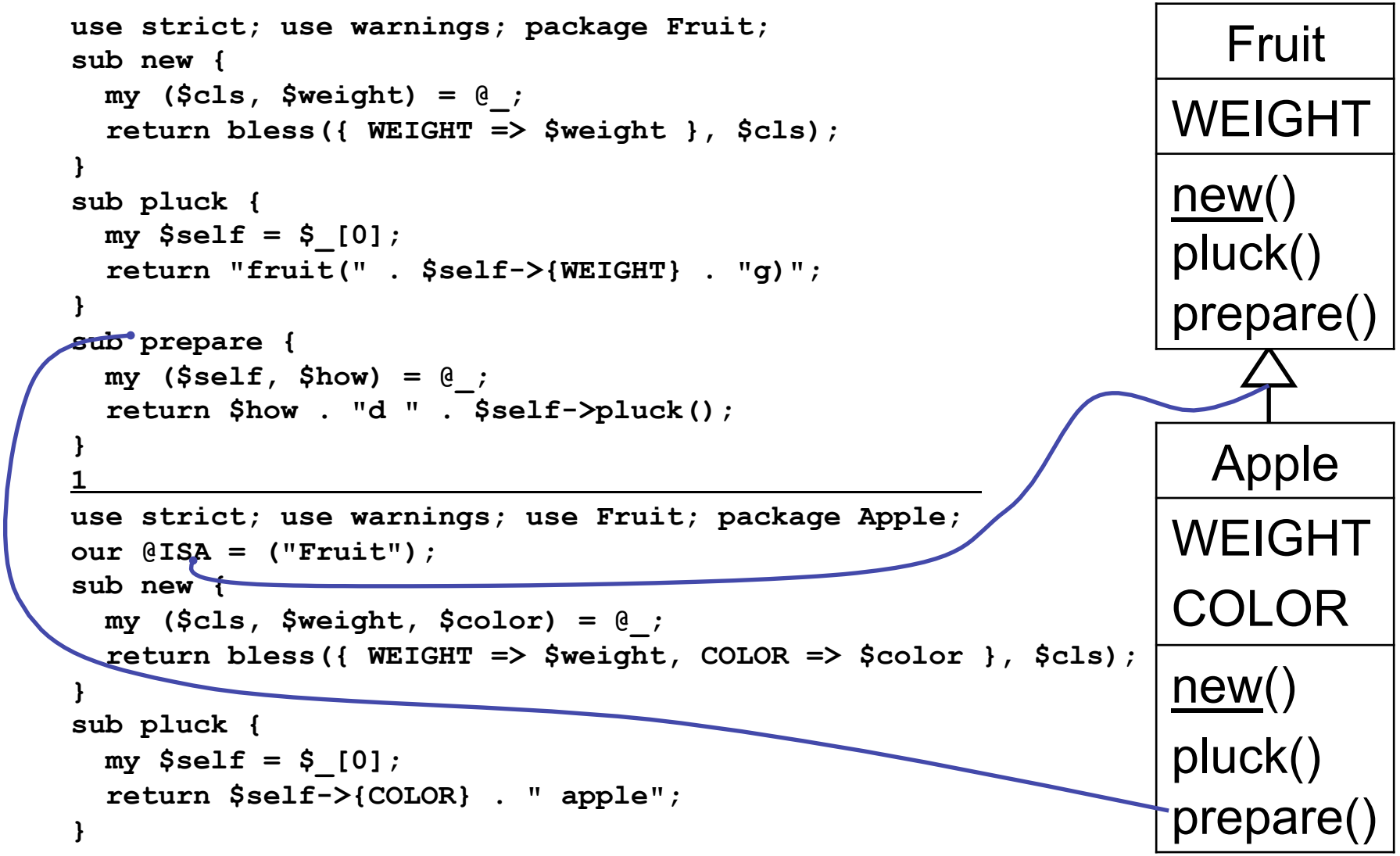


- Invocant (class name or object) is first argument
- **`bless(`***ref*`,`*pkg*`)`** associates object with class

Inheritance in Perl

```
use strict; use warnings; package Fruit;
sub new {
  my ($cls, $weight) = @_;
  return bless({ WEIGHT => $weight }, $cls);
}
sub pluck {
  my $self = $_[0];
  return "fruit(" . $self->{WEIGHT} . "g)";
}
sub prepare {
  my ($self, $how) = @_;
  return $how . "d " . $self->pluck();
}
1
```

```
use strict; use warnings; use Fruit; package Apple;
our @ISA = ("Fruit");
sub new {
  my ($cls, $weight, $color) = @_;
  return bless({ WEIGHT => $weight, COLOR => $color }, $cls);
}
sub pluck {
  my $self = $_[0];
  return $self->{COLOR} . " apple";
}
1
```



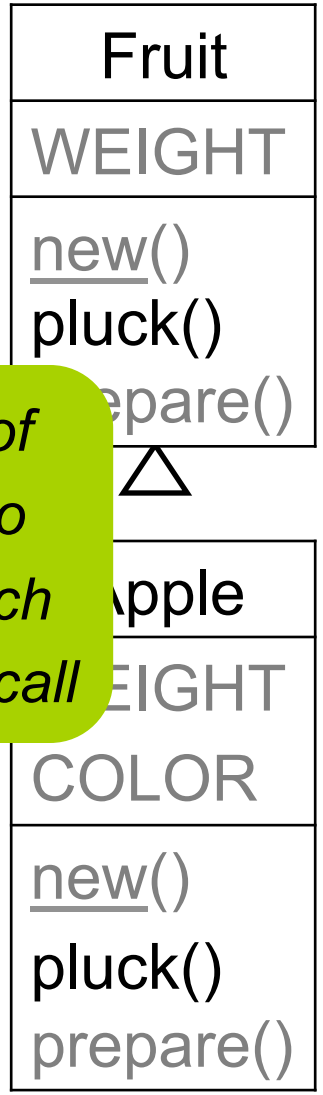
Concepts

Virtual Dispatch

```
use strict; use warnings; package Fruit;
sub new {
  my ($cls, $weight) = @_;
  return bless({ WEIGHT => $weight }, $cls);
}
sub pluck {
  my $self = $_[0];
  return "fruit(" . $self->{WEIGHT} . "g)";
}
sub prepare {
  my ($self, $how) = @_;
  return $how . "d " . $self->pluck();
}
1
```

```
use strict; use warnings; use Fruit; package Ap
our @ISA = ("Fruit");
sub new {
  my ($cls, $weight, $color) = @_;
  return bless({ WEIGHT => $weight, COLOR => $color }, $cls);
}
sub pluck {
  my $self = $_[0];
  return $self->{COLOR} . " apple";
}
1
```

Use class of reference to decide which method to call



Outline

- Programming in the large
- Scripting as glue

Gluing Perl using Bash: Pipes

- Bash (Bourne again shell) is an interactive command prompt for Unix or Linux
- At the bash shell prompt, $prog_1$ | $prog_2$ pipes STDOUT of $prog_1$ to STDIN for $prog_2$
- Perl scripts work well in pipelines: read from STDIN, write to STDOUT
- Idiomatic Perl facilitates one-liners, e.g.:

```
ls -l | perl -e'while(<>){/(\d+)\s(\S+\s+){3}(\S+)$/;print "$1 $3\n"}' | sort -rn | head -5
```

- Good for one-time use, bad for readability

Gluing Bash using Perl: `…`

- Perl provides many common Unix commands as library functions
  - **`chdir`**, **`chmod`**, **`kill`**, **`mkdir`**, **`rmdir`**, …
- Like bash, Perl has file test operators
  - **`-d`**, **`-e`**, **`-f`**, **`-r`**, **`-w`**, **`-x`**, …
- Perl programs can embed shell commands with `…`
  - Returns output as string
  - Error code goes in **`$?`**

Gluing Perl using Perl: **`eval`**

- Perl scripts can interpret embedded Perl code using **`eval`**

```
#!/usr/bin/perl
print "please enter an operator, e.g., '+':\n";
$op = <>;
chomp $op;
$command = 'print (42 ' . $op . ' 7)';
print "running the command '$command':\n";
eval $command;
print "\n";
```

- Module **`Safe`** provides restricted **`eval`** (**`reval`**) as sandbox for untrusted code

Gluing Text using Perl: Heredocs

- Heredoc = multi-line text embedded in Perl

```
#!/usr/bin/perl
print "-Error-\n", <<POEM, "-David Dixon-";
Three things are certain:
Death, taxes, and lost data.
Guess which has occurred.
POEM
```

- Variations:
  - **<<'***id***'** suppress interpolation
  - **<<***space* lines up to blank line
  - **print(<<***id*$_1$**, <<***id*$_2$**);** stacked heredocs

Common Perl Mistakes

Detected	Message	Typical example
Compiler	Missing curleys	`if($x) print "x"`
User	`==` instead of `eq`	`die if("1x"=="1y")`
Compiler	No comma allowed after filehandle	`print STDOUT, "hi"`
Interpre-ter	Modification of a read-only value	`$x=10; $$x=20;`
	And many more	*…*

Soap-box

Evaluating Perl

Strengths

- String processing
  - Regular expressions
  - Interpolation
- One-liners
  - Many operators
  - Implicit operands
- Vibrant community
- Portability

Weaknesses

- Write-only language
- Need references to nest arrays/hashes
- Grafted-on features
  - Object-orientation
  - Exception handling
- Language specified by implementation

Perl Culture

- Perl haikus and other poetry

```
print STDOUT q,
Just another Perl hacker,
unless $spring
```

- Obfuscated Perl contest

```
$_ = "wftedskaebjgdpjgidbsmnjgc";
tr/a-z/oh, turtleneck Phrase Jar!/;
print;
```

- perl.org, cpan.org, perlmonks.org, pm.org

Suggestions for Perl Practice

- hw04 gets you points, but you may want to do more at your own leasure
- Sort CSV file by user-specified column
- Pretty-print C code as HTML
- Turn PID and PPID from ps into tree
- Use Perl to translate gnuplot into VBA code that draws a Powerpoint slide
- Write a Perl poem

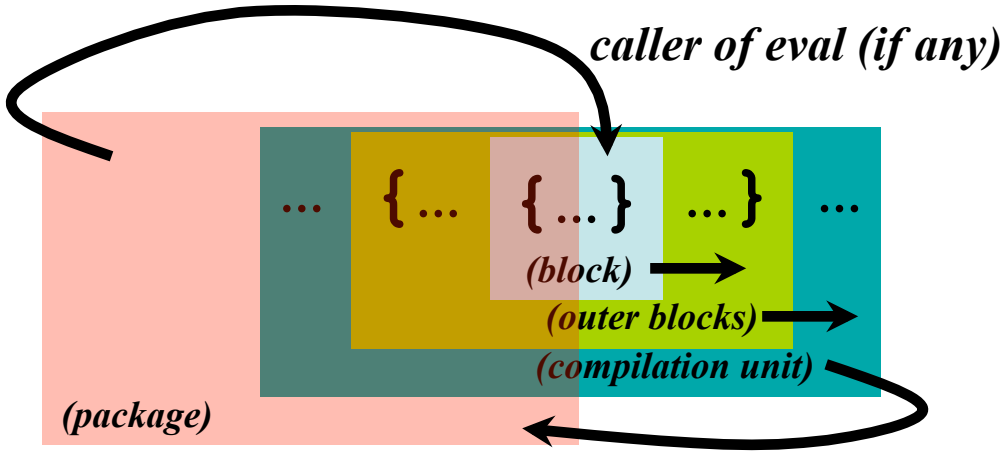
Administrative

Last Slide

- Pick up your graded homework and quiz
- Today's lecture
  - Context
  - Modules
  - Object-oriented Perl
  - Scripting as glue
- Next lecture
  - AWK
  - Prelim review

Scopes and Visibility

Name lookup:



- Compilation unit = file, **`-e`**, or **`eval`** argument
- Package = global named namespace
  - Can declare same package in multiple blocks, or even in multiple compilation units
  - Only one package in effect at any point

Package Example

```
#!/usr/bin/perl
package p;
our $x = 'px';
{
  package q;
  our ($x, $y) = ('qx', 'qy');
  print   "x       ('qx'  from block)                  $x      \n";
  print   "p::x    ('px'  qualified access)            $p::x   \n";
  print   "q::x    ('qx'  qualified access)            $q::x   \n";
}
{
  package r::s; # name can include multiple qualifier levels
  our $x = 'rsx';
  {
    package q; # can reopen in different block, even different file
    our $z = 'qz';
    print "x       ('rsx' block hides package)         $x      \n";
    print "y       ('qy'  disallowed by 'use strict')  $y      \n";
    print "z       ('qz'  from block)                  $z      \n";
    print "q::x    ('qx'  qualified access)            $q::x   \n";
    print "r::s::x ('rsx' qualified access)            $r::s::x\n";
  }
}
print     "x       ('px'  from compilation unit)       $x      \n";
print     "q::z    ('qz'  qualified access)            $q::z   \n";
print     "r::s::x ('rsx' qualified access)            $r::s::x\n";
```

Typeglobs and Symbol Tables

- Package **`p::q`** uses symbol table **`%p::q::`**
  - Symbol table entry **`$p::{'x'}`** is typeglob **`*p::x`**
- Typeglob = symbol table entry
  - E.g., **`*x`** holds **`$x`**, **`@x`**, **`%x`**, …
  - **`*x{SCALAR}`** is the same as **`\$x`**, and similarly for **`ARRAY`**, **`HASH`**, **`CODE`**, **`IO`**
  - **`*x{GLOB}`** is the same as **`\*x`**
  - **`*x{PACKAGE}`** returns package containing **`x`**
  - **`*x{NAME}`** returns name of typeglob
- Perl internally manipulates typeglobs to implement features, e.g., function "**`use`**"