

**G22.3033-002**  
**Scripting Languages**  
 6/12/2008  
 Context and Modules (Perl)  
 Scripting as Glue

© Martin Hirzel      G22.3033-002 NYU 6/12/2008      1

**Outline**

- Regular expressions (continued)
- Type conversions
- Programming in the large
- Scripting as glue

© Martin Hirzel      G22.3033-002 NYU 6/12/2008      2

**Perl**

**Idiomatic Perl**

- Interpreter specification: #!
- Regular expressions extract data
  - Captured groups: \$1, \$2, ...
- Hashes and arrays store data
  - Nested references build up data structures
- Default operand: \$\_
- Parameter array: @\_
- String interpolations report results

© Martin Hirzel      G22.3033-002 NYU 6/12/2008      3

**Perl**

**Example from 5/22/2008, Revisited**

```
#!/usr/bin/perl -w
%cup2g = ( flour => 110, sugar => 225, butter => 225 );
%volume = ( cup => 1, tbsp => 16, tsp => 48, ml => 236 );
%weight = ( lb => 1, oz => 16, g => 453 );
while (<>) {
  my ($qty, $unit, $sing) = /([0-9.]+) (\w+) (\w+)/;
  if ($cup2g{$sing} && $volume{$unit}) {
    $qty = 1.0 * $qty * $cup2g{$sing} / $volume{$unit};
    $unit = 'g';
  } elsif ($volume{$unit}) {
    $qty = 1.0 * $qty * $volume{ml} / $volume{$unit};
    $unit = 'ml';
  } elsif ($weight{$unit}) {
    $qty = 1.0 * $qty * $weight{g} / $weight{$unit};
    $unit = 'g';
  }
  printf("%d $unit $sing\n", $qty + .5);
}
```

© Martin Hirzel      G22.3033-002 NYU 6/12/2008      4

**Concepts**

**Chomsky Hierarchy**

Type	Languages	Automata	Rules
0	Recursively enumerable	Turing machine	$\alpha \rightarrow \beta$
1	Context sensitive	Linear bounded Turing machine	$\alpha B \gamma \rightarrow \alpha \delta \gamma$
2	Context free	Pushdown automaton	$A \rightarrow \beta$ (BNF)
3	Regular	Finite state machine	$A \rightarrow b, C \rightarrow dE$ (regexp)

- Formal regexp only has “essentials”
- Perl adds shortcuts and non-regular features

© Martin Hirzel      G22.3033-002 NYU 6/12/2008      5

**Perl**

**Non-Regular Perl Regex Features**

- Non-regular = not essential feature, and can not be emulated by essential features
  - Backtracking engine, may take exponential time
- Backreferences in same pattern: \1, \2, ...
- Zero-width assertions:
  - Look-ahead positive (?=...), negative (?!...)
  - Look-behind positive (?<=...), negative (?<!...)
- Match-time code execution: (?{...})
- Match-time interpolation: (??{...})
- Backtracking suppression: (?!...)

© Martin Hirzel      G22.3033-002 NYU 6/12/2008      6

**Perl**

### Non-Regular Perl Regex Examples

Description	Regex	Matched
Square (repeated)	<code>(\d+)x\1</code>	<code>123x123</code>
Palindrome (mirrored)	<code>(\w+)\w(?:{reverse \$1})</code>	<code>redivider</code>
Balanced parentheses	<code>(&lt;+&gt;x(?:{' ' x length \$1})</code>	<code>&lt;&lt;&lt;x&gt;&gt;&gt;</code>

© Martin Hirzel G22.3033-002 NYU 6/12/2008 7

**Perl**

### Outline

- Regular expressions (continued)
- Type conversions
- Programming in the large
- Scripting as glue

© Martin Hirzel G22.3033-002 NYU 6/12/2008 8

**Concepts**

### Natural and Artificial Languages

English		Perl	
Concept	Examples	Feature	Examples
Noun	fish	Variable	<code>\$line</code>
Pronoun	it, they	Default variable	<code>\$_, @_</code>
Inflection	...s	Sigil	<code>@...</code>
Singular	apple	Scalar	<code>\$i</code>
Plural	oranges	Array	<code>@row</code>
Verb	cook	Function	<code>sort</code>
Irregular verb	read	Operator	<code>&lt;...&gt;</code>
Infinitive	to be	Function reference	<code>&amp;cmp</code>
Context	go fish	Context	<code>if (@a) ...</code>

© Martin Hirzel G22.3033-002 NYU 6/12/2008 9

**Perl**

### Context

- When you use a value in a different context than the value's type, Perl converts it
  - E.g., using array as scalar -> int length
- Context provided by:
  - Operator: e.g., `string . string`
  - Function: e.g., `print list`
  - Assignment: e.g., `@a = list`
  - Statement: e.g., `if (boolean) {...}`
- Functions provide context for parameters, and can react to context for return value

© Martin Hirzel G22.3033-002 NYU 6/12/2008 10

**Perl**

### Primitive Conversions

Value	Context			
	Boolean	Number	String	Reference
Number	<code>0</code> false	identity	<code>" "</code>	undefined
	<code>!=0</code> false		<code>"value"</code>	
String	<code>" "</code> false	0	identity	symbolic reference
	<code>"0"</code> false	0		
	Other true	prefix		
Undefined	false	0	<code>" "</code>	undefined
Reference	true	addr	<code>"typ (addr) "</code>	identity

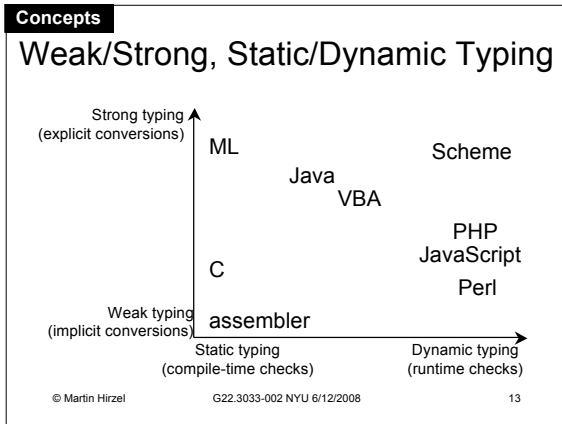
© Martin Hirzel G22.3033-002 NYU 6/12/2008 11

**Perl**

### List Conversions

Value	Context	
	Scalar	List
Scalar	identity	1-element list
List literal	last element	flattened sublists
Array	length	identity
Hash	empty <code>"0"</code>	empty list
	non-empty <code>"size/buckets"</code>	<code>k<sub>1</sub>, v<sub>1</sub>, k<sub>2</sub>, v<sub>2</sub>, ...</code>
<code>m/ /</code>	true iff match	all groups
<code>&lt;...&gt;</code>	end of file <code>" "</code>	empty array
	not EOF	line
		all lines

© Martin Hirzel G22.3033-002 NYU 6/12/2008 12



**Perl**

### Functions and Context

- Prototype provides context for parameters
  - `sub [id] [proto] [attrs] [{...}]`
  - No prototype: list operator
  - `proto = (protoChar*)`
  - `$` scalar, `@` list, `%` hash, `*` file handle
  - `&` subroutine (in first slot: bare block, no comma)
  - `;` separates mandatory from optional arguments
  - `\` adds backslashes on actuals
- `wantarray` checks context for return value
  - `true = list, false = array, undefined = void context`

© Martin Hirzel G22.3033-002 NYU 6/12/2008 14

**Perl**

### Prototype Example

- Example from page 227 of "Programming Perl", 3rd ed. Wall, Christiansen, Orwant. O'Reilly, 2000.
- See also the Error module on CPAN.

Function definitions	Example usage
<pre>sub try(&amp;\$) {   my (\$try, \$catch) = @_;   eval { &amp;\$try };   if (\$?) {     local \$_ = \$@;     &amp;\$catch;   } } sub catch (&amp;) { \$_[0] }</pre>	<pre>try {   die "phooey"; } #not end of the call! catch {   /phooey/ and   print "unphooey\n"; };</pre>

© Martin Hirzel G22.3033-002 NYU 6/12/2008 15

### Outline

- Regular expressions (continued)
- Type conversions
- Programming in the large
- Scripting as glue

© Martin Hirzel G22.3033-002 NYU 6/12/2008 16

**Soap-box**

### Readable Perl

- Use pragmata: `warnings`, `strict`, maybe also `diagnostics`, `sigtrap`
- Avoid default `$_`
- Name your subroutine parameters
- Use `my`, avoid `local`
- Use `/x` modifier on regular expressions
  - Add whitespace, line breaks, comments
- Use modules for large projects

© Martin Hirzel G22.3033-002 NYU 6/12/2008 17

**Perl**

### 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

© Martin Hirzel G22.3033-002 NYU 6/12/2008 18

**Perl**

### 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 ('qz' 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";
```

© Martin Hirzel G22.3033-002 NYU 6/12/2008 19

**Perl**

### 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

© Martin Hirzel G22.3033-002 NYU 6/12/2008 20

**Perl**

### 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, $show) = @_;
    return $show . "d " . pluck($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"
```

© Martin Hirzel G22.3033-002 NYU 6/12/2008 21

**Concepts**

### 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

© Martin Hirzel G22.3033-002 NYU 6/12/2008 22

**Perl**

### Structure of a Perl Application

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

© Martin Hirzel G22.3033-002 NYU 6/12/2008 23

**Perl**

### Using Objects

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

© Martin Hirzel G22.3033-002 NYU 6/12/2008 24

### Concepts

## Classes and Objects

class = package used to bless references

name

fields (hash keys)

class method

instance methods

instance name : blessing package

objects (blessed references)

fields (hash keys+values)

```

class Apple
  WEIGHT
  COLOR
  new()
  pluck()
  prepare()

object $a1 : Apple
  WEIGHT = 150
  COLOR = "green"

object $a2 : Apple
  WEIGHT = 150
  COLOR = "red"
  
```

© Martin Hirzel G22.3033-002 NYU 6/12/2008 25

### Perl

## 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, $show) = @_;
  return $show . "d " . $self->pluck();
}
1
  
```

•Invocant (class name or object) is first argument

•**bless** (*ref*, *pkg*) associates object with class

© Martin Hirzel G22.3033-002 NYU 6/12/2008 26

### Perl

## 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, $show) = @_;
  return $show . "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
  
```

© Martin Hirzel G22.3033-002 NYU 6/12/2008 27

### 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, $show) = @_;
  return $show . "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
  
```

Use class of reference to decide which method to call

© Martin Hirzel G22.3033-002 NYU 6/12/2008 28

### Perl

## 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"

© Martin Hirzel G22.3033-002 NYU 6/12/2008 29

## Outline

- Regular expressions (continued)
- Type conversions
- Programming in the large
- Scripting as glue

© Martin Hirzel G22.3033-002 NYU 6/12/2008 30

**Perl**

### Gluing Perl using Bash: Pipes

- Bash (Bourne again shell) is an interactive command prompt for Unix or Linux
- At the bash shell prompt, *prog<sub>1</sub>* | *prog<sub>2</sub>* pipes STDOUT of *prog<sub>1</sub>* to STDIN for *prog<sub>2</sub>*
- 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+)+){3}(S+)$/;print "$1 $3\n"} | sort -rn | head -5
```
- Good for one-time use, bad for readability

© Martin Hirzel G22.3033-002 NYU 6/12/2008 31

**Perl**

### 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  `$?`

© Martin Hirzel G22.3033-002 NYU 6/12/2008 32

**Perl**

### 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

© Martin Hirzel G22.3033-002 NYU 6/12/2008 33

**Perl**

### 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(<<id1, <<id2)`; stacked heredocs

© Martin Hirzel G22.3033-002 NYU 6/12/2008 34

**Perl**

### Common Perl Mistakes

Detected	Message	Typical example
Compiler	Missing curlyes	<code>if(\$x) print "x"</code>
User	<code>==</code> instead of <code>eq</code>	<code>die if("1x"=="1y")</code>
Compiler	No comma allowed after filehandle	<code>print STDOUT, "hi"</code>
Interpreter	Modification of a read-only value	<code>\$x=10; \$\$x=20;</code>
	<i>And many more ...</i>	

© Martin Hirzel G22.3033-002 NYU 6/12/2008 35

**Soap-box**

### Evaluating Perl

<p><u>Strengths</u></p> <ul style="list-style-type: none"> <li>• String processing                     <ul style="list-style-type: none"> <li>– Regular expressions</li> <li>– Interpolation</li> </ul> </li> <li>• One-liners                     <ul style="list-style-type: none"> <li>– Many operators</li> <li>– Implicit operands</li> </ul> </li> <li>• Vibrant community</li> <li>• Portability</li> </ul>	<p><u>Weaknesses</u></p> <ul style="list-style-type: none"> <li>• Write-only language</li> <li>• Lack of orthogonality                     <ul style="list-style-type: none"> <li>– Large language</li> <li>– Many gotchas</li> </ul> </li> <li>• Grafted-on features                     <ul style="list-style-type: none"> <li>– Object-orientation</li> <li>– Exception handling</li> </ul> </li> <li>• Language specified by implementation</li> </ul>
--	--

© Martin Hirzel G22.3033-002 NYU 6/12/2008 36

Perl

## Perl Culture Perl mongers

- Perl haikus and other poetry

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

```
$_ = "wftedskaebjgdpjgidbsmngc";  
tr/a-z/oh, turtleneck Phrase Jar!//;  
print;
```
- perl.org, cpan.org, perlmonks.org, pm.org

© Martin Hirzel G22.3033-002 NYU 6/12/2008 37

Perl

## Suggestions for Perl Practice

- hw04 gets you points, but you may want to do more at your own leisure
- 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

© Martin Hirzel G22.3033-002 NYU 6/12/2008 38

Administrative

## Last Slide

- Poll: changing deadline for homeworks
- Pick up your graded hw01
- Quiz will be graded by next week
- Today's lecture
  - Context
  - Modules
  - Object-oriented Perl
  - Scripting as glue
- Next lecture
  - HTML, HTTP
  - Server-side scripting
  - PHP

© Martin Hirzel G22.3033-002 NYU 6/12/2008 39