

G22.3033-002
Scripting Languages
 6/5/2008
 Textual data processing (Perl)

© Martin Hirzel G22.3033-002 NYU 6/5/2008 1

Outline

- Perl Basics
- Regular Expressions
- At 7:30pm: Quiz 1

© Martin Hirzel G22.3033-002 NYU 6/5/2008 2

Perl

About Perl

- Practical Extraction and Reporting Language
 - Regular expressions
 - String interpolation
 - Associative arrays
- TIMTOWDI
 - There is more than one way to do it
 - Make easy jobs easy
 - ... without making hard jobs impossible
 - Pathologically Eclectic Rubbish Lister

© Martin Hirzel G22.3033-002 NYU 6/5/2008 3

Concepts

Orthogonality

Definition of orthogonal	Language design principle	Violation of orthogonality
At right angles (unrelated)	Uniform rules for feature interaction	VBA object assignments
Not redundant	Few, but general, features	VBA while vs. do while loop

Perl is diagonal rather than orthogonal:
"If I walk from one corner of the park to another, I don't walk due east and then due north. I go northeast." [Larry Wall]
 => shortcut features even when not orthogonal

© Martin Hirzel G22.3033-002 NYU 6/5/2008 4

Perl

Related Languages

- Predecessors: C, sed, awk, sh
- Successors:
 - PHP ("PHP 1" = collection of Perl scripts)
 - Python, JavaScript (different languages, inspired by Perl's strengths + weaknesses)
- Perl 5 (current version, since 1994)
- Perl 6
 - Larry Wall has been talking about it since 2001
 - Release unclear; not backward compatible

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

Perl

How to Write + Run Code

- `perl [-w] -e 'perl code'`
 - "-w" flag produces warnings
- `perl [-w] script.pl`
- `script.pl`
 - Write the file in Vi or Emacs or ...
 - `chmod u+x script.pl`
Makes script executable
 - `#!/usr/bin/perl -w`
In first line of script specifies interpreter
- `perl [-w] -d -e 42`
 - Edit-eval-print loop (debug the script "42")

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

Perl

Lexical Peculiarities

- “term” = token
- Single-line comments: #
- Semicolon required after statements unless {last; in; block}
- Quotes around certain strings (bare words) optional in certain cases (e.g., as hash key)
- v-string: `v13.10 = "\x{13}\x{10}"`
- Interpolation; pick-your-own-quotes; POD; Heredocs

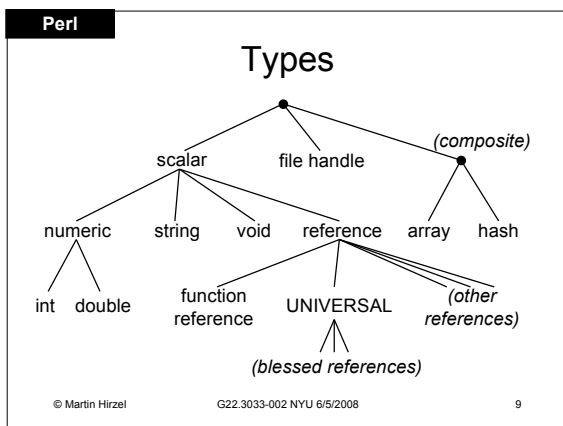
© Martin Hirzel G22.3033-002 NYU 6/5/2008 7

Perl

Interpolation

- Expansion of values embedded in string
- Single-quoted string literal `'abcde'`
 - Only interpolate `\'` and `\\`
- Double-quoted string literal `"abcde"`
 - More escape sequences, e.g., `\n`
 - Variables only, starting with `@` or `$`
 - Use curly braces to delimit: `"time ${hours}h"`
- Trick to interpolate arbitrary expressions
 - `"... @{{expr}} ..."` or `"... @{{scalar expr}} ..."`

© Martin Hirzel G22.3033-002 NYU 6/5/2008 8



Perl

Sigils, a.k.a. “Funny Characters”

- Symbol that must appear in front of variable, showing its type
 - `$`=scalar, `@`=array, `%`=hash, `&`=function, `*`=typeglob
 - E.g., `$a[0]` is element 0 of array `@a`
- Unlike shell, Perl requires sigil also on left-hand side of assignment
- `${id}` is the same as `$id`
- Function sigil `&` not required for call

© Martin Hirzel G22.3033-002 NYU 6/5/2008 10

Perl

Variable Declarations

Implicit	<code>print \$a + 1; \$b = 5;</code>	Read <code>undef</code> if non-existent
Local, lexical scope	<code>my \$c; my (\$d,\$e)=(3,4);</code>	
Global used locally	<code>sub f{ our \$g; print \$g++ }</code>	Hides locals; unlimited lifetime
Global, dynamic scope	<code>sub h{print \$i;} sub k{ local \$i=5; h }</code>	Can also localize single array/hash item

© Martin Hirzel G22.3033-002 NYU 6/5/2008 11

Concepts

Static vs. Dynamic Scoping

Static scoping	Dynamic scoping
Bound in closest nesting scope in program text	Bound in closest calling function at runtime
<pre>#!/usr/bin/perl -w \$x = 's'; sub g { my \$x = 'd'; return h() } sub h { return \$x } print g(), "\n"; #s print \$x, "\n"; #s</pre>	<pre>#!/usr/bin/perl -w \$x = 's'; sub g { local \$x = 'd'; return h() } sub h { return \$x } print g(), "\n"; #d print \$x, "\n"; #s</pre>

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

Perl	Operator	Precedence	Category	Description
(), " ", ' ', \, ., print, sort, ...		L	L	Terms, function call, quoting, list operators (leftward)
>		2	L	Dereference and member access
++, --		1	N	Auto-increment, auto-decrement
**		2	R	Exponentiation
!, ~, \, *, -		1	R	Negation (!, ~, -), reference (\), no-op (+)
m, /		2	L	Binding to regular expression pattern match
*, /, %, x		2	L	Multiplicative (x is string repetition)
+, -, .		2	L	Additive (. is string concatenation)
<<, >>		2	L	Bitwise shift
eval, sqrt, -f, -e, ...		1	N	Named unary operators, file test operators
<, >, <=, >=, lt, gt, le, ge		2	N	Relational (lt, gt, le, ge is for strings)
==, !=, <=>, eq, ne, cmp		2	N	Equality (eq, ne, cmp is for strings) (<=>, cmp yield -1/0/1)
&, , ^		2	L	Bitwise (not all same precedence)
&&		2	L	Logical and (short-circuit)
, //		2	L	Logical or (), Defined-or (//) (short-circuit)
... ..		2	N	Range (in list context) or bistable (in scalar context)
?, : ..		3	R	Ternary conditional
=, +=, -=, *=, ...		2	R	Assignment: return l-value of target
,, ,, =>		2	L	List (in list context) or sequencing (in scalar context)
print, sort, ...		N	L	List operators (rightward)
not, and, or, xor		2	R	Logical (short-circuit; not all same precedence)

© Martin Hirzel G22.3033-002 NYU 6/5/2008 13

Perl Operators: List vs. Named Unary

- Different precedence rules
- List operator (most user-defined functions)
 - High leftward, low rightward precedence
 - @a = (1,5,sort 9,2); print @a; #1529
- Named unary operator
 - Lower than arithmetic, higher than comparison
 - @a = (1,5,sqrt 9,2); print @a; #1532
- Call either one with parentheses
 - Highest precedence
 - @a = (1,5,sort(3+6),2); print @a; #1592

© Martin Hirzel G22.3033-002 NYU 6/5/2008 14

Perl Input and Output

- Output
 - print "Hello, world!";
 - print STDERR "boo!";
 - printf "sqrt(%.2f)=%.2f\n", 2, sqrt(2);
- Input
 - \$lineFromStdIn = <>;
 - open MYFILE, '<recipe' or die "\$!";
 - \$lineFromMyFile = <MYFILE>;
 - @allLines = <MYFILE>;

© Martin Hirzel G22.3033-002 NYU 6/5/2008 15

Perl Arrays

- Resizable
- Literals: list @a = (1, 3, 5), range @b = 2..4
- Indexing: e.g. \$a[1]
 - Zero-based; negative index counts from end
 - \$#a returns last index of @a, in this case, 2
 - Write to non-existent index auto-vivifies
- Free: undef @a, truncate: \$#a=1
- Array slice: using multiple indices, e.g., @a[0,2] or @a[1..2]
- Using array in scalar context: returns length
 - scalar(@a); # 3 = size

© Martin Hirzel G22.3033-002 NYU 6/5/2008 16

Perl Hashes

- Associative arrays, using hash tables
- Literals: none; use list literals instead
 - %h = (lb => 1, oz => 16, g => 453);
- Indexing: string, bare-word quotes optional
 - \$h{"oz"}, \$h{oz}
 - Write to non-existing index auto-vivifies
- Free: undef %h; delete: undef \$h{oz};
- Hash slice: returns array, e.g., @a = @h{lb, g}; returns (1, 453)
- Using hash in scalar context: returns string
 - scalar(%h); # "3/8" = size/capacity

© Martin Hirzel G22.3033-002 NYU 6/5/2008 17

Perl Array and Hash References

- References are scalars, only scalars can be stored in other arrays/hashes
- Literals: "anonymous array/hash composers"
 - \$ar=[1,3,5]; \$hr={x=>2,y=>0};
- Indexing: e.g., \$ar->[1]; \$hr->{y};
 - Dereference operator -> optional in nested chain, e.g., \$rarr->[123]{abc}
 - Write to non-existent index auto-vivifies all indices on the way
- Using reference in string context: returns string describing the type and address
 - print \$ar; # "ARRAY(0x18bef54)"

© Martin Hirzel G22.3033-002 NYU 6/5/2008 18

Perl

Data Structures

```

$recipe = [
  {
    qty => 2,
    unit => "cup",
    ing => "flour",
  },
  {
    qty => 4,
    unit => "oz",
    ing => "sugar",
  },
  {
    qty => 3,
    unit => "tbsp",
    ing => "milk",
  },
];
    
```

© Martin Hirzel G22.3033-002 NYU 6/5/2008 19

Perl

Control Statements

Proper statements (man perlsyn)	<pre> if (expr) { ... } [elsif (expr) { ... } ...] [else { ... }] [label:] while (expr) { ... } [continue { ... }] [label:] until (expr) { ... } [continue { ... }] [label:] for (expr; expr; expr) { ... } [label:] foreach [var] (list) { ... } [continue { ... }] [label:] for [var] (list) { ... } [continue { ... }] [label:] { ... } [continue { ... }] </pre>
Modifiers	<i>stmt</i> (if unless while until foreach) <i>expr</i>
Operators (man perlfunc)	<pre> do eval sub continue goto last next redo return die dump exit </pre>

© Martin Hirzel G22.3033-002 NYU 6/5/2008 20

Perl

What is Truth

- Strings "" and "0" are false, everything else is true
- File read <...> returns "" at end-of-file
- Number 0 converted to string "0"
- Array in scalar context 0 iff empty
- Hash in scalar context "0" iff empty
- Reference "" iff undef

© Martin Hirzel G22.3033-002 NYU 6/5/2008 21

Concepts

Blocks as Function Arguments

- do { ... }, eval { ... }, sub { ... } look like control statements, but aren't
 - They are functions with block argument
 - Other examples: grep, map, sort
 - You can write such functions yourself
- This is a lightweight form of callback
- SmallTalk language does same, only more
 - SmallTalk has no control statements at all
 - Method on boolean with block argument
 - (x<y) ifTrue: [mx:=y] ifFalse: [mx:=x]

© Martin Hirzel G22.3033-002 NYU 6/5/2008 22

Perl

Writing Subroutines

- Declaration
 - sub [id] [proto] [attrs] [{...}]
 - No id: anonymous; no proto: list operator; no block: declaration without definition
 - To return a value: return expr;
 - If no explicit return: value of last expression
- Arguments
 - Array @_
 - Call-by-reference (\$_[i] is alias of actual i)
 - Common idiom: copy arguments into named locals, e.g., my (\$a, \$b) = @_;

© Martin Hirzel G22.3033-002 NYU 6/5/2008 23

Perl

Default Variable \$_

- Only angle operator in while condition: read line from file handle into \$_
- No variable in foreach: use \$_ as iteration variable
- No binding in pattern match: use \$_
- No explicit operand for certain functions: use \$_, e.g., print, chop, chomp, split, ...
- Don't confuse \$_ with @_ (arguments) or _ (file handle of previous test)

© Martin Hirzel G22.3033-002 NYU 6/5/2008 24

Outline

- Perl Basics
- Regular Expressions
- At 7:30pm: Quiz 1

© Martin Hirzel G22.3033-002 NYU 6/5/2008 25

Perl Example of m// Matching

Input	<pre>-rw-r--r-- 1 hirzel ibm 9404 May 30 10:37 hw01.html -rw-r--r-- 1 hirzel ibm 59469 May 30 10:37 hw01.pdf -rw-r--r-- 1 hirzel ibm 9142 Jun 1 19:21 index.html</pre>
Script	<pre>#!/usr/bin/perl -w while(\$line = <>) { if (\$line =~ m/(\d+)\s\w{3}[\d\s:]+(\S+)/) { printf "\$2 contains \$1 bytes\n"; } }</pre>
Output	<pre>hw01.html contains 9404 bytes hw01.pdf contains 59469 bytes index.html contains 9142 bytes</pre>

© Martin Hirzel G22.3033-002 NYU 6/5/2008 26

Perl Explanation of m// Matching

```

if ($line =~ m/(\d+)\s\w{3}[\d\s:]+(\S+)/) {
    printf "$2 contains $1 bytes\n";
}
    
```

Annotations:

- `string to search in` points to `$line`
- `true if matched` points to the `if` statement
- `binding operator` points to `=~`
- `m//: match operator` points to `m/`
- `() : grouping+capturing` points to the parentheses in the pattern
- `$1, $2: captured groups` points to the backreferences in the code
- `(\d+)\s\w{3}[\d\s:]+(\S+): regular expression` points to the pattern

© Martin Hirzel G22.3033-002 NYU 6/5/2008 27

Perl Inside a Regular Expression

Kind	Construct	Syntax	Example	
			Pattern	Matches
Essentials	Character	Itself	<code>b</code>	<code>b</code>
	Concatenation	<code>e₁e₂</code>	<code>bc</code>	<code>bc</code>
	Alternative (or)	<code>e₁ e₂</code>	<code>a bc</code>	<code>a, bc</code>
	Repetition (≥0)	<code>e*</code>	<code>a*</code>	<code>, a, aa, aaa</code>
Quantifier	Optional	<code>e?</code>	<code>(a b)?</code>	<code>, a, b</code>
	Repetition (≥1)	<code>e+</code>	<code>a+</code>	<code>a, aa, aaa</code>
Char class	Custom class	<code>[...]</code>	<code>[a-c]</code>	<code>a, b, c</code>
	Wildcard character	<code>.</code>	<code>.</code>	<code>a, 3, :</code>
	Shortcut class	<code>\...</code>	<code>\d</code>	<code>0,1,2,...,9</code>
Assertion	Zero-width anchor	<code>\$/, ^, \w, ...</code>	<code>\w</code>	(word boundary)

© Martin Hirzel G22.3033-002 NYU 6/5/2008 28

Perl Explanation of s/// substitution

```

#!/usr/bin/perl -w
while($line = <>) {
    if ($line =~ m/(\d+)\s\w{3}[\d\s:]+(\S+)/) {
        s/_+\s(\d+)\s\w{3}[\d\s:]+(\S+)/$2 contains $1 bytes/;
        print $line;
    }
}
    
```

Annotations:

- `string to search and modify` points to `$line`
- `true if matched` points to the `if` statement
- `binding operator` points to `=~`
- `s///: substitution operator` points to `s/`
- `regular expression` points to the pattern in the `s/` command
- `$2 contains $1 bytes: replacement text` points to the replacement text

© Martin Hirzel G22.3033-002 NYU 6/5/2008 29

Perl Outside a Regular Expression

- Match: `m//` or simply `//`; substitute: `s///`
- Binding: `=~` positive, `!~` negative
 - Or when missing: bound to `$_`
- Modifiers: e.g. `s///g` global substitute
- Return value: true if success, invert for `!~`
 - Or when assigned to list: groups, e.g., `my ($a,$b) = $s =~ /(\w+) (\d+)/;`
- Output: `$&` entire match, `$1, ...` groups
- Find out more: “man perlre”

© Martin Hirzel G22.3033-002 NYU 6/5/2008 30

Perl

Library Functions

- String: chomp, chop, length, substr, uc
- RegExp: m//, s//, split
- Math: abs, cos, exp, sin, sqrt
- Array: pop, push, shift, unshift
- Control: die, do, eval, last, next, sub
- Scoping: local, my, our, package, use
- Files: chdir, chmod, fork, printf, qx//
- And many more (“man perfunc”)

© Martin Hirzel G22.3033-002 NYU 6/5/2008 31

Perl

Perl Documentation

- Included: manpages perl, perldata, perlsyn, perlop, perlre, perfunc, ...
- Book: Programming Perl, 3rd edition. Larry Wall, Tom Christiansen, and Jon Orwant. O'Reilly, 2000 [safari].
- Online:
 - CPAN = Comprehensive Perl Archive Network, <http://www.cpan.org>
 - <http://www.perl.com>

© Martin Hirzel G22.3033-002 NYU 6/5/2008 32

Administrative

Last Slide

- Homework grader: Robert Soulé
- Example solutions available
- “Code in ppt file” doesn't mean “on slide”
- Today's lecture
 - Associative arrays
 - Regular expressions
 - Basics of Perl
- Next lecture
 - Context
 - Typeglobs
 - Object-oriented Perl

© Martin Hirzel G22.3033-002 NYU 6/5/2008 33

Outline

- Perl Basics
- Regular Expressions
- At 7:30pm: Quiz 1

© Martin Hirzel G22.3033-002 NYU 6/5/2008 34