

Perl Basics
7 Mar 2009
#fedora-classroom

Doran Barton
<fozz@iodynamics.com>

LEARNING RESOURCES

- Books
 - Learning Perl (Llama book)
 - Programming Perl (Camel book)
 - A whole bunch of other O'Reilly books
 - Perl Best Practices (highly recommended)
- Sites
 - PerlMonks.org
- Organizations
 - PerlMongers – user groups (pm.org)

WHAT IS PERL?

- Interpreted, Compiled on-the-fly
- Cross-platform
 - Unix, Windows, Mac, VMS, you name it
- "Natural" language
- Free (as in speech and as in beer)
- Object oriented
- TMTOWTDI
 - *“There's more than one way to do it.”*

COMPARED TO...

- PHP
- Python
- Java
- C/C++

DOCUMENTATION

- Includes POD (plain old documentation) data
 - Read with 'perldoc'
 - Ex: 'perldoc -f printf' or 'perldoc perltoot'
 - Also accessible online
 - <http://www.perl.com/pub/q/documentation>

HELLO WORLD

- The Goodbye, Cruel World script:

```
#!/usr/bin/perl  
print "Goodbye, cruel world.\n";
```

RUNNING PERL SCRIPTS

- As argument to interpreter
 - `$ perl cruel.pl`
- As standalone executable
 - Include magic line (often called the "she-bang")
 - `#!/usr/bin/perl`
 - Set execute permissions
 - `$ chmod a+x cruel.pl`
 - Run it
 - `$ cruel.pl`

FUNDAMENTAL RULES

- All Perl code is case-sensitive
- Most commands end with a semi-colon
- Comments begin with a pound (#) character
 - Magic / she-bang is special exception
- Perl code can be ugly... but doesn't have to be.
- Use -w interpreter option to get more verbose warnings
 - `#!/usr/bin/perl -w`

SCALAR VARIABLES

- Can contain singular data (integers, strings, booleans)
- Fundamental variable type in Perl
- Variable names preceded by \$
 - `$month = get_month();`
`print "Today's month is $month\n";`

LIST VARIABLES

- Ordered, indexed collection of scalars
- It's an array, really
- Variable names preceded by @
- Individual list items accessed via \$-name.
 - `@now = localtime();`
`print "The current hour is $now[4]\n";`

MORE ON LISTS

- Use **scalar** to view list in scalar context which reveals number of items in list
 - ```
if(scalar @friends == 0) {
 print "You need some friends, dude.\n";
}
```
- Use **shift/unshift** and **pop/push** to treat list as pseudo-queue or pseudo-stack
  - ```
push @names, "John Doe";
```

HASH VARIABLES

- An unordered, indexed collection of data.
 - Similar to a set
 - Key-value pairs
 - Variable names preceded by %
 - Individual hash values accessed via \$-name.
 - ```
%student = (
 first_name => 'Doran',
 last_name => 'Barton',
 student_no => 512);
print "First name is $student{'first_name'}\n";
```

# MORE ON HASHES

- Use the `keys` function to get list of key values in hash
  - ```
@fields = keys %student_rec;  
foreach $hash_field (@fields) {  
    print "$hash_field: $student_rec{$hash_field}\n";  
}
```

DEFAULT VARIABLES

- Perl uses a default scalar (`$_`) and a default list (`@_`)
 - Ex: reading from a file
 - ```
while(<SOMEFILE>) {
 print "LINE: $_";
}
```
  - Ex: Command-line arguments
    - ```
if(scalar @_ != 3) {  
    print "Improper usage!\n";  
}
```

QUOTING

- Quoting methods in Perl
 - Double quotes (")
 - Single quotes (')
 - Back quotes (`)
- Rules are similar to that of Unix shell script

DOUBLE QUOTES

- Variables inside double quotes are interpolated
 - `$string = "Today's month is $month";`
 - Value of `$string` might be
Today's month is 2

SINGLE QUOTES

- No interpolation inside single quotes
 - `$string = 'Hot dogs: $1.00';`
 - Value of string: Hot dogs: \$1.00

BACK QUOTES

- Substitutes output of external command given
- Quick and dirty way to grab data from other programs
 - `$date = `/bin/date`;`
 - Value of `$date`: Fri Feb 9 01:41:20 MST 2001

IF-ELSE-ELSIF

- Testing conditions in Perl...

- `if($name eq "admin") {
 log_attempt();
} else {
 login_normal();
}`
 - `if($x == 1) {
 ...
} elsif($x == 2) {
 ...
}`

GIVEN...

- Historically, no switch/case structure.
- Perl 5.10 provides the **given** structure
 - ```
given($gender) {
 when($_ eq "male") { $life_exp = 75; }
 when($_ eq "female") { $life_exp = 80; }
 default { $life_exp = 0; }
}
```

# TESTS

- Testing equality/inequality
  - == Test for equality between numbers
  - != Test for inequality between numbers
  - eq Test for equality between strings
  - ne Test for inequality between strings
- Comparisons
  - <,>,<=,>= Tests for numbers
  - gt,ge,lt,le Tests for strings
- See 'perlop' POD

# FOR LOOP

- Like C
- Iterative loop
  - ```
for($i=0;$i<50;$i++) {  
    process_state($states[$i]);  
}
```

FOREACH

- Execute codeblock for each item in a list
 - ```
foreach $student (@allstudents) {
 compute_grade($student);
}
```
  - Functionally equivalent to:
    - ```
for($i=0;$i<scalar @allstudents;$i++) {  
    $student = $allstudents[$i];  
    compute_grade($student);  
}
```

WHILE/UNTIL

- Provides looping based on conditional expression
 - `while(<STDIN>) {`
 `chomp;`
 `$length += length;`
 `}`
 - `do {`
 `process_thingie($x);`
 `} until(check_status($x)) ;`

BUILT-IN FUNCTIONS

- See `perlfunc` POD for documentation
- All parameters are passed (and received) as list
 - Can use parentheses (C-style) or not
 - `print "Hello ", $name, ". How are you?\n";`
 - `...` is the same as `...`
 - `print("Hello", $name, ". How are you?\n");`