Elements of Programming in Perl

<H16-14/15>

Functions and Perl Modules

Josep F. Abril

jabril@imim.es

Functions in Perl

& ⇨ &unctions

```
sub reverse_seq() {
  my $seq = shift;
  $revseq = reverse $seq;
  return $revseq;
}
&reverse_seq($seq)
```

Arguments list: @_

Prototyping ⇨ `sub reverse_seq($);`

\ ⇨ References

```
$funct = \&reverse_seq;
$anon_funct = sub { ...statements... };
```

```
&{$funct}($seq)
$funct->($seq)
```


Passing Arguments to Functions

By copy

```
...
sub do_loop($@); # prototype
...
sub do_loop() {  # implementation
  my ($seq, @ary) = @_;
  foreach my $q (@ary) {
    print STDOUT "$seq : $q\n";
  };
} # do_loop
...
my $seq = "ATG";
my @N = (1..10);
my @M = ('A'..'Z');
&do_loop($seq, @N, @M);
...
```

By reference

```
...
sub do_loop($$$); # prototype
...
sub do_loop() {  # implementation
  my ($seq, $ary1, $ary2) = @_;
  foreach my $q (@$ary1) {
    print STDOUT "A: $seq : $q\n";
  };
  foreach my $q (@$ary2) {
    print STDOUT "B: $seq : $q\n";
  };
} # do_loop
...
my $seq = "ATG";
my @N = (1..10);
my @M = ('A'..'Z');
&do_loop($seq, \@N, \@M);
...
```

Working with Functions

```
sh$ ./seq_analysis.pl single_seq.fa
    sh$ ./seq_analysis_subs.pl single_seq.fa

================================================================================

SEQUENCE: dna_seq

--> STATS:
  Length: 500bp
     G+C: 61.20%

--> FORWARD Translations:
   Phase 0 : TRVSSLDPGNRVHIESGANVRSPSLPFPPPLPRPRPRFPPCAAPSELAAAGELFPSIPPSFTQDVHIRTSASAGFHGRSLLFW
             GGDHLRNPTRGRSRRPPKIPNVNTHR*KMYAKTGVESLGRRVDDRAPAEGLGARFPPLPVPPSDVGSPSIHSASIKAPAEAPT
   Phase 1 : PGSHPLTLGTGSTLNQVRMFARLLCLSRLPSPGRGPGSPPALHPQSWLQPASCSRQSLPPLHRMSILGHLRQQVSTAGPCCSG
             GGTISEILHAEGLGDPLRSQM*TLIGERCMRRRGLKAWGVELTTERPQRALGRASPPFQFRPVT*EVHPFTALL*RRQLRRLL
   Phase 2 : PGLIP*PWEPGPH*IRCECSLAFSAFPASPPPAAAPVPPLRCTLRVGCSRRAVPVNPSLLYTGCPY*DICVSRFPRPVPVVLG
             GGPSPKSYTRKV*ETP*DPKCEHS*VKDVCEDGG*KPGA*S*RQSARRGPWGALPPPSSSAQ*RRKSIHSQRFYKGAS*GAYY

--> REVERSE Translations:
   Phase 0 : VVGASAGAFIEAL*MDGLPTSLGGTGRGGKRAPRPSAGALSSTLRPRLSTPVFAYIFHL*VFTFGILGGLLDLPRVGFRRWSP
             PQNNRDRPWKPADADVLIWTSCVKEGGIDGNSSPAAANSEGAAQGGNRGRGRGRGGGKGREGERTFAPDSMWTRFPGSRDETR
   Phase 1 : **APQLAPL*KRCEWMDFLRHWAELEGGGSAPQGPLRALCRQLYAPGFQPPSSHTSFTYECSHLGS*GVS*TFRV*DFGDGPP
             PRTTGTGRGNLLTQMS*YGHPV*RREGLTGTARRLQPTLRVQRRGGTGAAAGGGEAGKAEKASEHSHLIQCGPGSQGQGMRPG
   Phase 2 : SRRLSWRLYRSAVNGWTSYVTGRNWKGGEARPKALCGRSVVNSTPQAFNPRLRIHLSPMSVHIWDLRGSPRPSACRISEMVPP
             PEQQGPAVETC*RRCPNMDILCKGGRD*REQLAGCSQL*GCSAGGEPGPRPGEGRRERQRRRANIRT*FNVDPVPRVKG*DPG

================================================================================
```


Packages

The `package` keyword defines the beginning of a new namespace.
All global identifiers (names of variables, subroutines, filehandles, formats and directory handles) appearing after this statement "belong" to that package.

```
$seq = "??: NNNNNNNNNNNN";

package human;
$seq = "Hs: AGCTGATTGACG";

package mouse;
$seq = "Mm: GCAGTTAGTCGA"; # this $seq is different from previous one

package human; # make human package the current package
print "$seq\n"; # prints "Hs: AGCTGATTGACG"

package main; # this is the default current package for our script
print "$seq\n";          # prints "??: NNNNNNNNNNNN"
print "$mouse::seq\n"; # prints "Mm: GCAGTTAGTCGA"
```


Writing Perl Modules

```
package yourmodule; # module name must match its filename
use strict;
use vars qw( @ISA @EXPORT @EXPORT_OK %EXPORT_TAGS $VERSION ...EXPORTED_VARS... );
use Exporter; # mandatory
$VERSION = 1.00;
@ISA = qw(Exporter); # also mandatory
@EXPORT = qw( $yourmodule_exported_var &yourmodule_exported_function );
@EXPORT_OK = qw( ); # to set functions/vars to be exported on demand
%EXPORT_TAGS = ();  # to set groups of functions/vars being exported on demand
###
### Here starts your implementation of exported/local variables and functions
$yourmodule_exported_var = 'HELLO WORLD';
      # they do not require 'my' if they were predeclared at 'use vars'
$yourmodule_global_var = 1;
sub yourmodule_exported_function() {
    my $local_var = shift;
    &yourmodule_local_function() if $yourmodule_global_var;
    print STDOUT "$local_var\n";
    &yourmodule_local_function() if $yourmodule_global_var;
}
sub yourmodule_local_function() {
    print STDOUT ("#"x60)."\n";
}
### Here finishes your module implementation...
### The following line is mandatory and warns perl that the module
### has finished (more or less... ;^D), just put as shown here
1;
```

Working with Modules

```
sh$ perl -e 'print "@INC\n"'
sh$ export PERL5LIB='.'  ## or ##  sh$ export PERL5OPT='-I.'
sh$ perl -e 'print "@INC\n"'
sh$ ./seq_analysis_modules.pl single_seq.fa
```

```
==============================================================================

SEQUENCE: dna_seq

--> STATS:
  Length: 500bp
     G+C: 61.20%

--> FORWARD Translations:
   Phase 0 : TRVSSLDPGNRVHIESGANVRSPSLPFPPPLPRPRPRFPPCAAPSELAAAGELFPSIPPSFTQDVHIRTSASAGFHGRSLLFW
             GGDHLRNPTRGRSRRPPKIPNVNTHR*KMYAKTGVESLGRRVDDRAPAEGLGARFPPLPVPPSDVGSPSIHSASIKAPAEAPT

   Phase 1 : PGSHPLTLGTGSTLNQVRMFARLLCLSRLPSPGRGPGSPPALHPQSWLQPASCSRQSLPPLHRMSILGHLRQQVSTAGPCCSG
             GGTISEILHAEGLGDPLRSQM*TLIGERCMRRRGLKAWGVELTTERPQRALGRASPPFQFRPVT*EVHPFTALL*RRQLRRLL

   Phase 2 : PGLIP*PWEPGPH*IRCECSLAFSAFPASPPPAAAPVPPLRCTLRVGCSRRAVPVNPSLLYTGCPY*DICVSRFPRPVPVVLG
             GGPSPKSYTRKV*ETP*DPKCEHS*VKDVCEDGG*KPGA*S*RQSARRGPWGALPPPSSSAQ*RRKSIHSQRFYKGAS*GAYY

--> REVERSE Translations:
   Phase 0 : VVGASAGAFIEAL*MDGLPTSLGGTGRGGKRAPRPSAGALSSTLRPRLSTPVFAYIFHL*VFTFGILGGLLDLPRVGFRRWSP
             PQNNRDRPWKPADADVLIWTSCVKEGGIDGNSSPAAANSEGAAQGGNRGRGRGRGGGKGREGERTFAPDSMWTRFPGSRDETR

   Phase 1 : **APQLAPL*KRCEWMDFLRHWAELEGGGSAPQGPLRALCRQLYAPGFQPPSSHTSFTYECSHLGS*GVS*TFRV*DFGDGPP
             PRTTGTGRGNLLTQMS*YGHPV*RREGLTGTARRLQPTLRVQRRGGTGAAAGGGEAGKAEKASEHSHLIQCGPGSQGQGMRPG

   Phase 2 : SRRLSWRLYRSAVNGWTSYVTGRNWKGGEARPKALCGRSVVNSTPQAFNPRLRIHLSPMSVHIWDLRGSPRPSACRISEMVPP
             PEQQGPAVETC*RRCPNMDILCKGGRD*REQLAGCSQL*GCSAGGEPGPRPGEGRRERQRRRANIRT*FNVDPVPRVKG*DPG

==============================================================================
```


Retrieving Command-line Options

```
#!/usr/bin/perl -s
# '-s' -> rudimentary built-in switch parsing
defined($h) && do {
  print STDERR <<'EOHelp';
# USAGE:
#   ./script.pl -v -h -ofile=file.out ...
EOHelp
  exit(1);
};
$verbose = defined($v) ? 1 : 0;
defined($ofile) && open(OFH, ">$ofile");
...
```

```
use Getopt::Std;

my ($opt_v, $opt_H, $opt_o) = (0,0,undef);
   # declare vars when using strict
getopts("vHo:");
   # sets $opt_v, $opt_H and $opt_o
$opt_H && &help();

my %args = (); # declare when using strict
getopts("vHo:", \%args);
   # sets $args{v}, $args{H} and $args{o}
defined($args{H}) && &help();

sub help() {
  print STDERR <<'EOHelp';
# USAGE:
#   ./script.pl -v -H -o file.out ...
EOHelp
  exit(1);
} # help
...
```

```
use Getopt::Long;

my ($verbose,$outfile) = (0,undef);

GetOptions(
    "v|verbose"   => \$verbose,
    "o|outfile=s" => \$outfile,
    "h|help"      => \&help,
);

if (defined($outfile)) {
  open(OFH, "> $outfile") || die("ERROR...");
  $ofh = \*OFH;
} else {
  $ofh = \*STDOUT;
};

sub help() {
  print STDERR <<'EOHelp';
# USAGE:
#   ./script.pl [options] ...
# OPTIONS:
#   -v | --verbose
#       Reporting program execution.
#   -o | --outfile "file.out"
#       Setting output file (default STDOUT).
#   -h | --help
#       Print this help.
EOHelp
  exit(1);
} # help
...
```


CGI: Common Gateway Interface


```
# CGI script that creates a fill-out form
# and echoes back its values.

use CGI qw/ :standard /;

print header,
      start_html('A Simple Example'),
      h1('A Simple Example'),
      start_form,
      "What's your name? ",textfield('name'),p,
      "What's the combination?", p,
      checkbox_group(-name=>'words',
                     -values=>['eenie','meenie','minie','moe'],
                     -defaults=>['eenie','minie']), p,
      "What's your favorite color? ",
      popup_menu(-name=>'color',
                 -values=>['red','green','blue','chartreuse']),p,
      submit,
      end_form,
      hr;

if (param()) {
    print "Your name is",em(param('name')),p,
          "The keywords are: ",em(join(", ",param('words'))),p,
          "Your favorite color is ",em(param('color')),
          hr;
}
exit(0);
```

DBI: DataBase Independent Interface

```
use DBI;
use strict;

$dbh = DBI->connect($dsn, $user, $password,
      { RaiseError => 1, AutoCommit => 0 });

$sth = $dbh->prepare(
         "SELECT foo, bar FROM table WHERE baz=?");

$sth->execute( $baz );

while ( @row = $sth->fetchrow_array ) {
    print "@row\n";
};
```


BioPerl

http://bioperl.org/

```
use Bio::Seq;
use Bio::SeqIO;
$seqin  = Bio::SeqIO->new(-file   => "human.gb",
                          -format => 'GenBank'   );
$seqout = Bio::SeqIO->new(-file   => ">forward.fa",
                          -format => 'Fasta'     );
$seqtuo = Bio::SeqIO->new(-file   => ">reverse.fa",
                          -format => 'Fasta'     );
while ( my $seq = $seqin->next_seq() ) {
        $id = $seq->display_id();
        print STDERR "Working on: $id\n";
        $seqout->write_seq($seq);
        $id = "$id.rev";
        $rev = $seq->revcom();
        $rev->display_id($id);
        $seqtuo->write_seq($rev);
};
```


Inline

Embeeding code from other languages (Assembler, C, C++, Java, Python, Awk, ...).

Handles for us all the compilation steps required for those languages before binding the new functions into our script.

Easy access to the Advanced Programming Interface (API) to reuse perl internal inplemented functions in C.

```
use Inline::Files;
use Inline C;

$filename = $ARGV[0];
die "Usage: perl vowels.pl filename\n" unless -f $filename;

$text = join ??, <>;          # slurp input file
$vp = vowel_scan($text);      # call our function
$vp = sprintf("%03.1f", $vp * 100);  # format for printing
print "The letters in $filename are $vp% vowels.\n";

__END__
__C__

/* Find percentage of vowels to letters */
double vowel_scan(char* str) {
    int letters = 0;
    int vowels = 0;
    int i = 0;
    char c;
    char normalize = ?a? ^ ?A?;
    /* normalize forces lower case in ASCII; upper in EBCDIC */
    char A = normalize ? ?a?;
    char E = normalize ? ?e?;
    char I = normalize ? ?i?;
    char O = normalize ? ?o?;
    char U = normalize ? ?u?;
    char Z = normalize ? ?z?;

    while(c = str[i++]) {
        c |= normalize;
        if (c >= A && c <= Z) {
            letters++;
            if (c == A ?? c == E ?? c == I ?? c == O ?? c == U)
                vowels++;
        }
    }

    return letters ? ((double) vowels / letters) : 0.0;
}
```