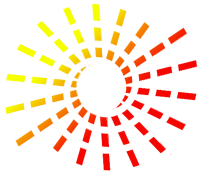
Introduction to PERL III

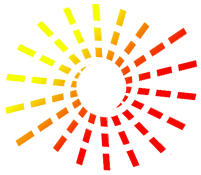
March 13, 2014

Noe Fernandez & Lukas Mueller



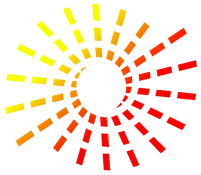
Class Content

- File I/O
- Arguments
- Regular expressions
- Tab-delimited files parsing
- Subroutines
- Modules
- BioPerl
- Documenting code
- Two files comparison



Subroutines

- Subroutines are pieces of code useful to organize the script structure and avoid repeated tasks
- Code that is duplicated in a script may be better in a subroutine



Subroutines

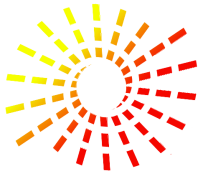
BTI

```
sub foo {
  my @params = @_;

  # do something and return result
  return $result;
}

my $result = foo($param1, $param2);
```

Note: subroutines parameters and returned values when using arrays or hashes need to use arrayrefs or hashrefs

Subroutines

```
#!/usr/bin/perl
use strict;
use warnings;

sub format_seq {
    my $old_seq = shift @_;
    my $fasta_width = shift @_;
    my $new_seq = "";

    # format sequence to the chosen width
    for (my $i=0; $i<length($old_seq); $i=$i+$fasta_width) {
        $new_seq = $new_seq.substr($old_seq,$i,$fasta_width)."\n";
    }
    chomp($new_seq);
    return $new_seq;
}

my $formatted_seq = format_seq($seq,$width);
```

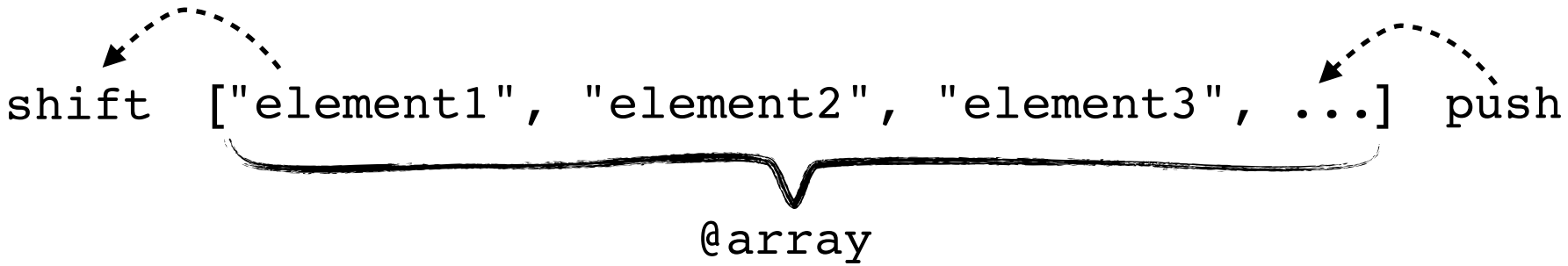


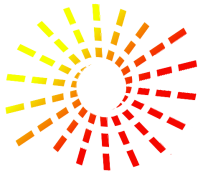
Add and get elements from arrays

BTI

```
my @array = ();
my $element = "foo";

# add element at the end of the array
push(@array,$element);
```

```
# get first element from array
my $element = shift(@array);
```



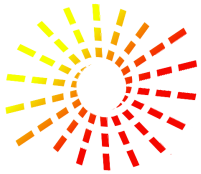


Modules

- Modules are re-usable code that scripts can load and use
- Subroutines that are used from different scripts are better placed into modules

Comprehensive Perl Archive Network

http://search.cpan.org



Modules examples

Math

- Math::Complex - complex numbers and associated mathematical functions.
- Math::BigInt - Arbitrary size integer/float math package.

File

- File::Temp - return name and handle of a temporary file safely.
- File::Basename - Parse file paths into directory, filename and suffix.
- File::Spec - portably perform operations on file names.

Test

- Test::More - yet another framework for writing test scripts.

Modules examples

```
Options (defaults in parentheses):

 Input:
  -q                 query input files are FASTQ .fq/.fastq (default)
  --qseq             query input files are in Illumina's qseq format
  -f                 query input files are (multi-)FASTA .fa/.mfa
  -r                 query input files are raw one-sequence-per-line
  -c                 <m1>, <m2>, <r> are sequences themselves, not files
  -s/--skip <int>    skip the first <int> reads/pairs in the input (none)
  -u/--upto <int>    stop after first <int> reads/pairs (no limit)
  -5/--trim5 <int>   trim <int> bases from 5'/left end of reads (0)
  -3/--trim3 <int>   trim <int> bases from 3'/right end of reads (0)
  --phred33          qualities are Phred+33 (default)
  --phred64          qualities are Phred+64
  --int-quals        qualities encoded as space-delimited integers
```

Get options

Getopt::Std

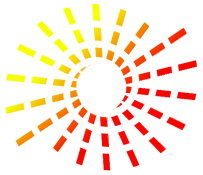
Getopt::Long - Extended processing of command line options.

Index files

Lucy::Simple - Basic search engine.

Parallelize process

threads - Perl interpreter-based threads



Bioperl

http://www.bioperl.org/

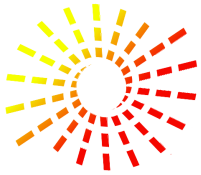
http://search.cpan.org/~cjfields/BioPerl-1.6.923/BioPerl.pm

BioPerl is a toolkit of perl modules useful in building bioinformatics solutions in Perl. It is built in an object-oriented manner so that many modules depend on each other to achieve a task. It includes parsers for most of the biological file formats and tools to work with biological programs and databases

List of Bioperl Modules:
http://search.cpan.org/~cjfields/BioPerl-1.6.923/

Bioperl

Bio::SeqIO - Handler for SeqIO Formats.

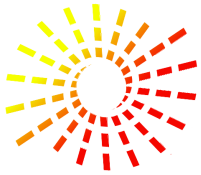


Bioperl

BTI

Read FASTA files and write output

```
#!/usr/bin/perl

use strict;
use warnings;
use Bio::SeqIO;

# open file to read the fasta file using Bioperl
my $in = Bio::SeqIO->new(-file => "input.fasta", -format => 'Fasta');

# open file to write the output
my $out = Bio::SeqIO->new( -file => ">output.fasta", -format => 'Fasta');


# read fasta file sequence by sequence
while(my $seq = $in->next_seq()) {
  # print seq
  $out->write_seq($seq);
}
```

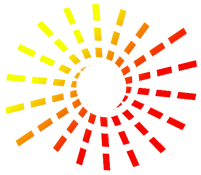


Bioperl

BTI

get sequence information from a FASTA file

```
#!/usr/bin/perl

use strict;
use warnings;
use Bio::SeqIO;

# open file to read the fasta file using Bioperl
my $in = Bio::SeqIO->new(-file => "input.fasta", -format => 'Fasta');

# read fasta file sequence by sequence
while(my $seq = $in->next_seq()) {

  # get sequence data
  my $seq_name = $seq->id();
  my $seq_length = $seq->length();
  my $seq_fasta = $seq->seq();
  my $seq_desc = $seq->desc();
}
```

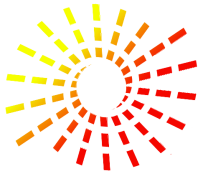


Documenting code: POD

Plain Old Documentation format

```
#!/usr/bin/perl

=head1 NAME

  script.pl
  Perl class example script

=head1 DESCRIPTION

  perl script.pl <file.fasta> <gene_list.txt>

  example:
  perl script.pl ITAG2.3_proteins.fasta gene_list.txt

=head1 AUTHOR

  Noe Fernandez
  (nf232@cornell.edu)

=cut
```

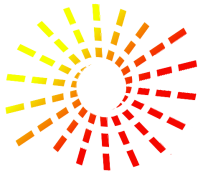
perldoc script.pl



Documenting code: POD

POD format for subroutines

```
=head2 format_seq

  Example: format_seq($seq,$seq_width)
  Description: format sequence width to a chosen length
  Parameters: sequence (string), width (integer)
  Returns: a sequence in a string

=cut

sub format_seq {
  my $seq = shift;
  my $seq_width = shift;
  ...
  return $new_seq;
}
```

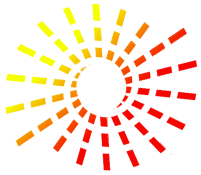


Two files comparison

Step 1

open the smallest file and store the useful information in a hash

```
# declare a hash to save the useful info
my %hash = ();

# open file
open (my $fh, "<", $smaller_input_file);

# read input file line by line
while (my $line = <$fh>) {
  chomp($line); # remove new line

  # parse file
  ...
  # fill hash data
  $hash{$key} = $value;
}
```



Two files comparison

Step 2

open the largest file, read it line by line, and make comparisons with the data from the other file stored in the hash

```
# open largest file
open (my $other_fh, "<", "largest_file.txt");

while (my $line = <$other_fh>) {
  chomp($line); # remove new line

  # parse data
  ...

  # make comparisons
  if ($hash{$key}) {
    print "$result\n";
  }

}
```

Do not nest 2 while loops to read each line of both file!