Perl 3 – References and dereferencing, functions, and perl Modules
BIOC 8142 Feb 25, 2013
Bill Pearson wrp@virginia.edu

- References and dereferencing – multi-dimensional @arrays, %hashes
- Functions (subroutines)
  - defining sub {}
  - arguments are (flattened) array
- Perl Modules
  - Getopt::Long - command line options
  - LWP - perl's version of curl/wget
  - Interacting with the NCBI
- CPAN

1

What should you do to reinforce the lecture material?

- Beginner's introduction to Perl
  www.perl.com/pub/2000/10/begperl1.html
- Curtis Poe's Beginning Perl: (ch. 6, 7, 11)
  proquest.safaribooksonline.com/book/programming/perl/9781118235638 (In intro to references, '\' are missing)
- Chromatic's Modern Perl (ch. 3, references, ch. 5)
  http://onyxneon.com/books/modern_perl/
- `man perldsc` – perl datastructures (references)
- `man perlref` – references and data structures
- `man perllol` – manipulating arrays of arrays
- `man perlsub` – subroutines (functions)

2

Arrays of arrays (and hashes of hashes) Variable dereferencing

```
       {qseqid}        {sseqid}    {percid}          . . .      {evalue} {bts}
[0] sp|GSTM1_HUMAN   sp|GSTM1_HUMAN 100.00 218  0    0  1  218 1 218 7e-127 452
[1] sp|GSTM1_HUMAN   sp|GSTM4_HUMAN  86.70 218 29    0  1  218 1 218 3e-112 403
[2] sp|GSTM1_HUMAN   sp|GSTM1_MACFA  85.78 218 31    0  1  218 1 218 3e-110 397
[3] sp|GSTM1_HUMAN   sp|GSTM2_PONAB  85.78 218 31    0  1  218 1 218 1e-109 395
[4] sp|GSTM1_HUMAN   sp|GSTM2_MACFA  85.78 218 31    0  1  218 1 218 1e-109 395
[5] sp|GSTM1_HUMAN   sp|GSTM5_HUMAN  87.61 218 27    0  1  218 1 218 1e-109 395
```

- Perl @arrays and (%hashes) are always one-dimensional, but data is usually (at least) two-dimensional.
- How do we build data structures that have multiple dimensions?

```
hit[1]{percid}==86.70
```

3

Variable dereferencing

To build multi-dimensional (complex) data structures, perl provides variable references, which are scalar (simple) values:

```
\$string_ref, \@array_ref, \%hash_ref

$month_name_ref = \@month_name;
@{$month_name_ref} == @month_name;

$month_hash_ref = \%month_hash;
%{$month_hash_ref} == %month_hash
```

```
DB<> x %month_hash
0  'Sep'
1  31
2  'May'
3  31
4  'Jul'
5  31
6  'Jun'
7  30
8  'Jan'
9  31
...
```

```
DB<9> x \%month_hash
0  HASH(0xfc35d40)
   'Apr' => 30
   'Aug' => 31
   'Dec' => 31
   'Feb' => 28
   'Jan' => 31
   'Jul' => 31
   'Jun' => 30
   'Mar' => 31
   'May' => 31
```

reference

referred type data

4

Variable dereferencing

Since references are scalars, they can be put into @arrays and %hashes to make 2-D structures

```
DB<10> @new_array = (\@months, \@month_days, \%month_hash);
DB<11> x @new_array
0  ARRAY(0xfb512a0)
   0  'Jan'
   1  'Feb'
 ...
  11  'Dec'
1  ARRAY(0xfc361a8)
   0  31
   1  28
 ...
  11  31
2  HASH(0xfc35d40)
   'Apr' => 30
   'Aug' => 31
   'Dec' => 31
```

```
DB<12> x $new_array[0][4]
0  'May'
DB<13> x $new_array[0]->[4]
0  'May'
DB<14> x $new_array[1][4]
0  31
DB<15> x $new_array[2]->{May}
0  31
```

5

Variable dereferencing

Since references are scalars, they can be put into @arrays and %hashes to make 2-D structures

```
DB<20> %new_hash = (
  names=>\@months,
  length=>\@month_days
  hash =>\%month_hash);
DB<21> x \%new_hash
0  HASH(0x100d7798)
   'days' => ARRAY(0xfc361a8)
      0  31
      1  28
     11  31
   'hash' => HASH(0xfc35d40)
      'Apr' => 30
      'Aug' => 31
      'Sep' => 31
   'months' => ARRAY(0xfb512a0)
      0  'Jan'
      1  'Feb'
      11  'Dec'
```

```
DB<21> x $new_hash{months}[1]
0  'Feb'
DB<22> x $new_hash{hash}{Apr}
0  30
DB<23> x $new_hash{hash{Mar}}
Can't locate object method "hash"...
DB<24> x $new_hash{hash->{Mar}}
0  undef # because hash->{Mar}==31
DB<25> x $month_hash{Mar}
0  31
```

6

Subroutines (functions) in Perl

```
#!/usr/bin/perl -w
use strict;
my $name = getname();
print "my name is: $name\n";
exit; # not necessary, only a visual cue
sub getname {
    print "Enter your name: ";
    my $line = <STDIN>;
    chomp($line);
    return $line;
}
```

Passing Parameters

```
#!/usr/bin/perl -w
use strict;

my $fname = getname('first');
my $lname = getname('last');
print "Your name is: $fname $lname\n";

sub getname {
    my ($type) = @_;
    print "Please enter your $type name: ";
    my $line = <STDIN>; chomp($line);
    return $line;
}
```

Parameter arrays are 1-D!

```
#!/usr/bin/perl -w
use strict;
my @a = qw(a b c d);
my @b = 1..10;

print "Parameter Count : ", countparam('foo', @a, @b), "\n";
# prints 15, not 3!


sub countparam {
    # arguments to functions are transferred in @_
    # my $var = shift(@_);  my ($var1, $var2, $var3) = @_
    return scalar @_;
}
```

When you need to send an @array of data, use an \@array reference.

```
sub sum_arrays {
   my ($arr1_ref, $arr2_ref);
   my @result= ();
   for (my $i=0; $i < scalar(@$array1_ref); $i++) {
       $result[$i]=$arr1_ref->[$i] + $arr2_ref->[$i];
   }
   return @result;
}
```

Variable "scope"

```
my $var1 = 0; # global, available everywhere
sub mysub { # the open brace opens a new scope
  my $var2 = 1; # $var2 only available within 'mysub'
  local $var1 = 2; # make our own private copy of $var1,
                   # leaving $var1 unchanged outside of
                   # 'mysub'; useful with global vars
                   # that you want to temporarily change
} # end of this scope

{ local $var1 = 1; dostuff(); }
print "$var1\n"; # prints 0;

sub dostuff { print "$var1\n"; } # prints 1, not 0 nor 2.
```

Packaging useful subroutines

```
# MyMath.pm
package MyMath;

sub cuberoot {
    my $y = shift @_;
    return exp(log($y)/3);
}

sub nthroot {
    my ($y, $n) = @_;
    return exp(log($y)/$n);
}

1;
```

```
#!/usr/bin/perl -w
use strict;

use lib "./"; # where to find it
use MyMath;

$cube = MyMath::cuberoot(10);
$nth = MyMath::nthroot(10, 5);
```

Modules - prepackaged methods

```
#!/usr/bin/perl
use strict;
use LWP::Simple 'get'; # import the 'get' subroutine

my $url = "http://www.weather.com/weather/local/22908";
my $html = get($url);
my ($temp) = $html =~ m/(\d+)\&deg\;F/m;
print "Current temperature: $temp F\n";
```

Modules and Object Oriented Programming

```
#!/usr/bin/perl -w
use strict;

use LWP::UserAgent;
use HTTP::Common;

# $ua and $req are objects built with "new" method:
my $ua = new LWP::UserAgent;
my $req = new HTTP::Common
  POST => 'www.ncbi.nlm.nih.gov/genome/guide/gquery.cgi',
  [ db => 1, term => 'gtm1_human' ];

# "request" is a method of $ua;
# it returns a "response" object
my $res = $ua->request($req);

# "content" is a method of $res:
my $html = $res->content();
# do stuff with $html
```

Modules - prepackaged methods

```
#!/usr/bin/perl
use strict;
use Getopt::Long;
my $url="";
my $display_cnt=10;
GetOptions("help"=>\&display_help, "url=s"=>\$url, "cnt=i"=>\$display_cnt);
if ($url) { ... }
...
for (my $i=0; $i < $display_cnt; $i++) { ... }
```

perldoc Getopt::Long

```
Getopt::Long(3)       User Contributed Perl Documentation      Getopt::Long(3)
NAME
       Getopt::Long - Extended processing of command line options
SYNOPSIS
         use Getopt::Long;
         my $data   = "file.dat";
         my $length = 24;
         my $verbose;
         $result = GetOptions ("length=i" => \$length,    # numeric
                               "file=s"   => \$data,      # string
                               "verbose"  => \$verbose);  # flag
DESCRIPTION
       The Getopt::Long module implements an extended getopt function called GetOptions().
       This function adheres to the POSIX syntax for command line options, with GNU
       extensions. In general, this means that options have long names instead of single
       letters, and are introduced with a double dash "--". Support for bundling of command
       line options, as was the case with the more traditional single-letter approach, is
       provided but not enabled by default.
```

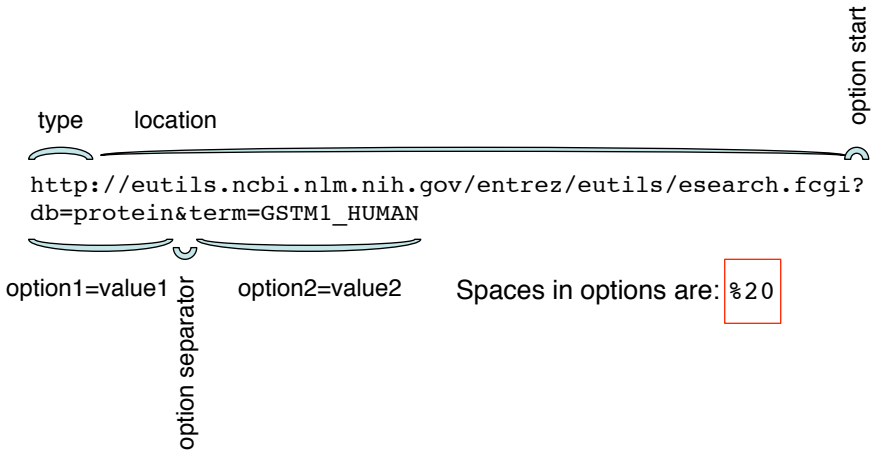
Modules – `LWP::Simple` (Perl's curl/wget)

```
#!/usr/bin/perl
use strict;
use LWP::Simple 'get'; # import the 'get' subroutine

my $url = "http://www.weather.com/weather/local/22908";
my $html = get($url);
my ($temp) = $html =~ m/(\d+)\&deg\;F/m;
print "Current temperature: $temp F\n";
```

LWP::Simple 'get' – perl's wget

```
#!/usr/bin/perl -w
use strict;

use LWP::Simple 'get';    # get information from a URL
use IO::String;  # needed to do I/O from a string, instead of a file

my $search_url = "http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?";
my $search_args = "db=protein&term=GSTM%20AND%20human[organism]&rettype=uilist";

my $search_html = get($search_url . $search_args);

my $search_io = IO::String->new($search_html);
my @search_lines = <$search_io>;

my @search_ids = grep { /<Id>(\d+)<\/Id>/; $_ = $1} @search_lines;

# now we have a list of IDs, get the sequences
my $seq_url = "http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?";
my $seq_args = "db=protein&id=" . join(",",@search_ids) ."&rettype=fasta";

my $seq_html = get($seq_url . $seq_args);

print $seq_html, "\n";
```

Getting Data from the WWW: URL's
URL – universal resource locator



Learning about Modules: perldoc module

```
d-128-54-138 108% perldoc LWP
NAME
    LWP - The World-Wide Web library for Perl
SYNOPSIS
      use LWP;
      print "This is libwww-perl-$LWP::VERSION\n";
DESCRIPTION
    The libwww-perl collection is a set of Perl modules which provides a
    simple and consistent application programming interface (API) to the
    World-Wide Web. The main focus of the library is to provide classes and
    functions that allow you to write WWW clients. The library also contain
    modules that are of more general use and even classes that help you
    implement simple HTTP servers.
    Most modules in this library provide an object oriented API. The user
    agent, requests sent and responses received from the WWW server are all
    represented by objects. This makes a simple and powerful interface to
    these services. The interface is easy to extend and customize for your
    own needs.
```

How to find data: NCBI

www.ncbi.nlm.nih.gov/books/NBK25500/
www.ncbi.nlm.nih.gov/books/NBK25497/

ESearch (text searches)

eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi

Responds to a text query with the list of matching UIDs in a given database (for later use in ESummary, EFetch or ELink), along with the term translations of the query.

EPost (UID uploads)

eutils.ncbi.nlm.nih.gov/entrez/eutils/epost.fcgi

Accepts a list of UIDs from a given database, stores the set on the History Server, and responds with a query key and web environment for the uploaded dataset.

ESummary (document summary downloads)

eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi

Responds to a list of UIDs from a given database with the corresponding document summaries.

EFetch (data record downloads)

eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi

Responds to a list of UIDs in a given database with the corresponding data records in a specified format.

20

LWP::Simple at the NCBI

```
#!/usr/bin/perl -w
use strict;

use LWP::Simple 'get';    # get information from a URL
use IO::String;  # needed to do I/O from a string, instead of a file

my $search_url = "http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?";
my $search_args = "db=protein&term=GSTM%20AND%20human[organism]&rettype=uilist";

my $search_html = get($search_url . $search_args);

my $search_io = IO::String->new($search_html);
my @search_lines = <$search_io>;

my @search_ids = grep { /<Id>(\d+)<\/Id>/; $_ = $1} @search_lines;

# now we have a list of IDs, get the sequences
my $seq_url = "http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?";
my $seq_args = "db=protein&id=" . join(",",@search_ids) ."&rettype=fasta";

my $seq_html = get($seq_url . $seq_args);

print $seq_html, "\n";
```

NCBI &retmode, &rettype

www.ncbi.nlm.nih.gov/books/NBK25499/table/chapter4.chapter4_table1/?report=objectonly

Record Type	&rettype	&retmode
	db = gene	
text ASN.1	null	asn.1, default
XML	null	xml
Gene table	gene_table	text
	db = nuccore, nucest, nucgss, protein or popset	
text ASN.1	null	text, default
binary ASN.1	null	asn.1
Full record in XML	native	xml
FASTA	fasta	text
	db = pubmed	
text ASN.1	null	asn.1, default
XML	null	xml
MEDLINE	medline	text
PMID list	uilist	text
Abstract	abstract	text
	db = taxonomy	
XML	null	xml, default
TaxID list	uilist	text or xml

22

www.ncbi.nlm.nih.gov/books/NBK25498/

```
use LWP::Simple;
# Download PubMed records that are indexed in MeSH for both asthma and
# leukotrienes and were also published in 2009.

$db = 'pubmed';  $query = 'asthma[mesh]+AND+leukotrienes[mesh]+AND+2009[pdat]';

#assemble the esearch URL
$base = 'http://eutils.ncbi.nlm.nih.gov/entrez/eutils/';
$url = $base . "esearch.fcgi?db=$db&term=$query&usehistory=y";
#post the esearch URL
$output = get($url);

#parse WebEnv and QueryKey
$web = $1 if ($output =~ /<WebEnv>(\S+)<\/WebEnv>/);
$key = $1 if ($output =~ /<QueryKey>(\d+)<\/QueryKey>/);

### include this code for ESearch-ESummary
#assemble the esummary URL
$url = $base . "esummary.fcgi?db=$db&query_key=$key&WebEnv=$web";
#post the esummary URL
$docsums = get($url);
print "$docsums";

### include this code for ESearch-EFetch
#assemble the efetch URL
$url = $base . "efetch.fcgi?db=$db&query_key=$key&WebEnv=$web";
$url .= "&rettype=abstract&retmode=text";

#post the efetch URL
$data = get($url);
print "$data";
```

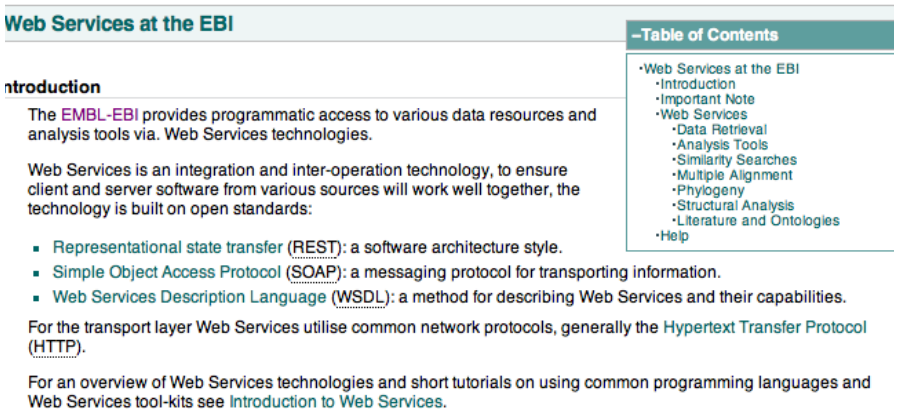
www.ncbi.nlm.nih.gov/books/NBK25498/

```
use LWP::Simple;

# Download protein records corresponding to a list of GI numbers.

$db = 'protein';
$id_list = '194680922,50978626,28558982,9507199,6678417';

#assemble the epost URL
$base = 'http://eutils.ncbi.nlm.nih.gov/entrez/eutils/';
$url = $base . "epost.fcgi?db=$db&id=$id_list";

#post the epost URL
$output = get($url);

#parse WebEnv and QueryKey
$web = $1 if ($output =~ /<WebEnv>(\S+)<\/WebEnv>/);
$key = $1 if ($output =~ /<QueryKey>(\d+)<\/QueryKey>/);

### include this code for EPost-ESummary
#assemble the esummary URL
$url = $base . "esummary.fcgi?db=$db&query_key=$key&WebEnv=$web";

#post the esummary URL
$docsums = get($url);
print "$docsums";

### include this code for EPost-EFetch
#assemble the efetch URL
$url = $base . "efetch.fcgi?db=$db&query_key=$key&WebEnv=$web";
$url .= "&rettype=fasta&retmode=text";

#post the efetch URL
$data = get($url);
print "$data";
```

Note: *To post a large number (more than a few hundred) UIDs in a single URL, please use the HTTP POST method for the EPost call (see Application 4).*

How to find data: EBI/EMBL

www.ebi.ac.uk/Tools/webservices/



Web Services at the EBI

–Table of Contents

- Web Services at the EBI
  - Introduction
  - Important Note
  - Web Services
    - Data Retrieval
    - Analysis Tools
    - Similarity Searches
    - Multiple Alignment
    - Phylogeny
    - Structural Analysis
    - Literature and Ontologies
  - Help

ntroduction

The EMBL-EBI provides programmatic access to various data resources and analysis tools via. Web Services technologies.

Web Services is an integration and inter-operation technology, to ensure client and server software from various sources will work well together, the technology is built on open standards:

- Representational state transfer (REST): a software architecture style.
- Simple Object Access Protocol (SOAP): a messaging protocol for transporting information.
- Web Services Description Language (WSDL): a method for describing Web Services and their capabilities.

For the transport layer Web Services utilise common network protocols, generally the Hypertext Transfer Protocol (HTTP).

For an overview of Web Services technologies and short tutorials on using common programming languages and Web Services tool-kits see Introduction to Web Services.

25

How to find data: EBI/EMBL

www.ebi.ac.uk/Tools/webservices/

Service	Clients	Description
ArrayExpress		Microarray data searching with ArrayExpress.
ChEBI Web Services	ChEBI Web Services	Entry retrieval from the ChEBI database.
ChEMBL Web Services	ChEMBL Web Services	Search data in, and retrieve data from the ChEMBL database
EB-Eye	EB-eye	Database search using the EB-eye search engine.
ENA Browser		Retrieval of sequence and associated records from ENA
Gene Expression Atlas API		Enriched database of summary statistics over a curated subset of ArrayExpress Archive
MartService		Database search and data retrieval using BioMart.
PSICQUIC		Standardised access to molecular interaction databases, including ChEMBL, Reactome and IntAct.
Rhea		Manually annotated database of chemical reactions
SRS		Database search and data retrieval using SRS@EBI.
UniProt.org		The Universal Protein Resource (UniProt) a comprehensive resource for protein sequence and annotation data.
WSDbfetch (REST)	WSDbfetch (REST)	Identifier based entry retrieval for various up-to-date biological databases.
WSDbfetch (SOAP)	WSDbfetch (SOAP)	Identifier based entry retrieval for various up-to-date biological databases.

26

How to find data: EBI/EMBL

www.ebi.ac.uk/Tools/webservices/

REST Service	SOAP Service	Description
FASTA (REST)	FASTA (SOAP)	Fast protein or nucleotide comparison using the FASTA suite. Includes Smith and Waterman local-local (SSEARCH), global-local (GLSEARCH) and global-global (GGSEARCH) alignment searches.
FASTM (REST)	FASTM (SOAP)	Peptide fragment searches using the FASTF, FASTM or FASTS programs from the FASTA suite.
NCBI BLAST (REST)	NCBI BLAST (SOAP)	Compare a sequence with those contained in nucleotide and protein databases using NCBI BLAST.
PSI-BLAST (REST)	PSI-BLAST (SOAP)	Position Specific Iterative BLAST (PSI-BLAST), guided mode
PSI-Search (REST)	PSI-Search (SOAP)	Iterative Smith and Waterman using a PSI-BLAST strategy
WU-BLAST (REST)	WU-BLAST (SOAP)	Compare a novel sequence with those contained in nucleotide and protein databases using WU-BLAST

27

How to find modules: CPAN

www.cpan.org

CPAN

Comprehensive Perl Archive Network

2011-02-08 online since 1995-10-26
8719 MB 258 mirrors
8728 authors 19305 modules

Welcome to CPAN! Here you will find All Things Perl.

Searching

- Perl core documentation (perldoc.perl.org; Jon Allen)
- Perl core and CPAN modules documentation (Randy Kobes)
- CPAN modules, distributions, and authors (search.cpan.org)

Browsing

- Perl modules
- Perl scripts
- Perl binary distributions ("ports")
- Perl source code
- Perl recent arrivals
- recent Perl modules
- CPAN sites list

FAQ etc

- CPAN Frequently Asked Questions
- Perl FAQ
- Perl Mailing Lists

For this course, modules will be pre-installed on franklin.achs

28

Homework - Modules

1. Write program that takes a list of PubMed ID's (that you would get from Entrez) and downloads the references from PubMed into a file. Download the text in "Medline" format, removing <html> tags. (For information on downloading information from Entrez and Pubmed, see:
   www.ncbi.nlm.nih.gov/books/NBK25500/
   www.ncbi.nlm.nih.gov/books/NBK25499/
2. Implement the download part of your program in a function() with three arguments:
   ncbi_fetch(\@uid_list, $database, $rettype) where $database can be a valid NCBI database (e.g. "protein", "pubmed", etc), and $rettype is a valid NCBI return type ("fasta", "medline", etc; NCBI retmode=text always). The function should either return an array of results (i.e. one fasta entry per array element) or and empty list (if no results are found)
3. Build another function, "`ncbi_search()`", that uses "`ncbi_fetch()`" but takes as arguments `ncbi_search($db, $search_term, $rettype)` and returns an array of entries (fasta, medline, etc)

Homework 2 - References

Distantly related proteins can often be shown to be homologous by finding an intermediate protein that shares significant similarity to each of the two homologous, but not significantly similar, proteins.

1. Write a script that performs a BLASTP search against SwissProt, then, for each of the non-significant matches (evalue > 0.1), runs a search (against SwissProt) with the subject sequence, and then compares the results from the first search to the "reverse" searches, attempting to find an intermediate protein that had significant similarity to the queries in both the first and second searches.
2. (optional) Modify the script to ensure that the regions of the proteins that align in the two searches overlap.

30