Introduction to Perl & Bioperl (I) Basic Perl Programming

Xuefeng Zhao
L. H. Baker Center, ISU

BCB 444/544X

Objectives

- Get an overview of basic Perl programming
- Write simple perl scripts to manipulate DNA/RNA sequences

Outline

- Why Perl?
- Install Perl/Bioperl on Windows
- Run a Hello perl script
- Data Types, Variables and Built-in Functions
- Control Structures
- Basic IO
- Subroutines & Functions
- More String Manipulation

Why Perl?

- Perl: Practical Extraction and Report Language
- Easy to learn
- Good for string manipulation and File IO
- Open source for all OS's
- A lot of Bioinformatics tools available

Install Perl/Bioperl

Ref: Install Note on www.bioperl.org

1. Unix/Linux: Pre-installed

2. Mac: http://www.macperl.org

3. Windows
   Download the current ActivePerl from www.activeperl.com. The windows installer package file is ActivePerl-5.8.6.811-MSWin32-x86-122208.msi
   - Install ActivePerl using the default values by double-clicking the msi file, using c:\perl as the Perl home
   - *** To install GD.pm.
     ppm> install http://theoryx5.uwinnipeg.ca/ppms/GD.ppd
   - To install Bioperl
     ppm>rep add Bioperl http://bioperl.org/DIST
     ppm>search bioperl
     1. bioperl [1.2.3] bioinformatics tool kits
     2. bioperl [1.2.1] non
     ...
     8. bioperl-1.4 [1.4] BioPerl 1.4 PPM3 Archive
     ....
     ppm>install 8
     .....
     Successfully installed Bioperl-1.4 version 1.4 in ActivePerl 5.8.6.811

Run a Hello Perl script

- Download all Perl scripts for the class from www.bioinformatics.iastate.edu/BBSI/ to your USB disk, and run hello.pl
- To run: perl hello.pl or
- To run: hello.pl

Script:Hello.pl

```
#!/usr/local/bin/perl

#hello perl script

use strict;
print "Hello, ISU-BCBSI\n";
```

Basic Syntax:
1. the first #! line indicates the location of perl, used for Unix/Linux OS.
2. Free form, case-sensitive
3. Each statement ends with a semicolon (;)
4. A comment line starts with a pound sign (#), you have to comment line by line.

Data type, Variables and Built-in functions

Data Types:

1. Scalar: string and number.
2. Array: list arrays (array) and associative arrays(hash).

Variables:

1. Scalar variables: start with $. *Ex.* $DNAstr, $numNT
2. Array: start with @. *Ex.* @nameAA
3. Hash: start with %. *Ex.* %RestricteEnzym

Some Perl built-in functions:

length	substr	index	rindex
push	pop	keys	sort
open	close		
die	exit	print	

Scalar variables: number and string

```
Number:  $numVar=EXPRESSION;
    $counter=20;
    $Tm=37.5;

String variables: $strVar=EXPRESSION;

#using single quote('), no variable expansion takes place,
$str1='isu-bcbsi' ;
$strSupport='$$$:NIH-NSF';          # $strSupport: $$$:NIH-NSF

#using double quote("), a variable expansion takes place,
#sepcial characters needs to be escape-ed
$str2="$str1,  ames, iowa' ;        # $str2: "isu-bcbsi, ames, iowa";

#using backstick(`), the variable is assigned the return values from the command
# line
$strDateNow=`date /t`;              # $strDateNow: "Mon 06/06/2005"
```

Operators and Functions for Scalar variables

Arithmetic Operators: =, +, -, *, /, %, **

Notation: $counter++; $counter= $counter+1; $counter += 1;
$counter--; $counter= $counter-1; $counter -= 1;

Operator and function for strings

Operator/Function	Example	Desc
Dot (.) .=	$str1="DNA"; $str2="RNA"; $str3=$str1." and ".$str2; $str1 .= $str2;	Concatenation: $str3: DNA and RNA Appending: $str1: DNARNA
length(STRING)	$len=length("ACGT")	$len: 4
index(STRING,SEARCH)	$idx=index("ACGTACGT", "C");	$idx: 1
rindex(STRING,SEARCH)	$ridx=index("ACGTACGT", "C");	$ridx: 5
substr(STRING,OFFSET, LEN, REPLACEMENT)	$msubstr=substr("ACGTACGT", 2, 3);	$msubstr: GTA

chop(STRING)
chomp(STRING)
	$str1="DNA";
	chop($str1);
	#matching chop, remove "\n";
	chomp($str1);
		$str1: DN
		
		$str1: DN

Array

Array:

```
@NTlist = ("A","T","G","C");     # assign 4 elements to the array, included in ();
$nt_2 = $NTlist[1];              # the array index starts 0, the index number is include in [];
$last_idx = $#NTlist;                    # last_idx: 3
$num_ele = scalar @NTlist;   # num_element: 4
```

Operator/Function	Example	Desc
push(@arr, element)	push (@NTlist, ("A","T")):	@NTlist: A, T, G, C, A, T
pop(@arr)	$m_nt= pop(@NTlist);	$m_nt: T; @NTlist: A, T, G, C, A
shift(@arr)	shift((@NTlist)	@NTlist: T, G, C, A
unshift(@arr, element)	unshift((@NTlist, "G");	@NTlist: G, T, G, C, A
delete($arr[$idx])	delete $NTlist[1];	@NTlist: G, G, C, A

Hash

Hash

```
# indexed by strings. Brace {} for key, percent sign  % for entire array
# assign 4 elements to the array,

%AAlist=(Ala=>"A",Gly=>"G",His=>"H", Phe=>"F");

$aa_gly=$AAlis{Gly};
```

Operator/function	example	Desc
keys(%ARRAY)	@aa_keys=keys (%AAlist);	@aa_keys: Ala,Gly,His,Phe Not ordered
values(%ARRAY)	@aa_vals=values (%AAlist);	@aa_vals: A, G, H, F, Not ordered
each(%ARRAY)	each((%AAlist)	Return pair by pair (key, value) =>(Ala, A), used for loop
delete($ARRAY{KEY})	delete($AAlist{Ala});	Remove the pair(Ala, A)
	$AAlis{Val}="V";	Add one element

Control Structures: IF ELSIF ESE

```
If (condition){
        statements;….
}
elsif {
        statements;….
}
else {
        statements;….
}
```

```
If (condition){
        statements;….
}
elsif {
        statements;….
}
#use elsif only, no switch in
Perl, use elsif to get around
```

```
If (condition){
        statements
;….
}

#use if only
```

comparison	number	string
Great than	>	gt
Great than or equal	>=	ge
Equal	==	eq
Less than	<	lt
Less or equal	<=	le
Not equal	!=	le
Comparison returns -1,0,1	<=>	cmp

Control Structures: IF ELSIF ELSE

```
If ($m_nt eq “A”)  { $num_A++; }          # A counter
elsif ($m_nt eq “T”) { $num_T++; }    # T counter
elsif ($m_nt eq “G”){ $num_G++; }   # G counter
elsif ($m_nt eq “C”){ $num_C++; }   #C counter
else { $num_error++; }                      # error
```

Loop Structures: WHILE, DO WHILE, FOR

```
-- WHILE Loop
while (condition){
        statements;….
}

-- DO-WHILE Loop: do at least once
Do{
        statements;….
}while(condition)

-- FOR Loop
For (initial values; test; increment)
{  statement;
}
```

Two control statements for loop:

last	Declare that this is the last statement in the loop
next	Start a new iteration of the loop

GC_counter.pl

Write a perl script to count GC content in a DNA sequence

Basic IO

Input	Keyboard (STDIN)	$m_input=<STDIN>; # wait for the input unt the new line character
Input	file	Open(MYFILEHANDLE, “<mySeqFileName”): @all_lines=<MYFILEHANDLE>; Close(MYFILEHANDLE);
Output	Screen	print STDOUT “Hello, ISU-BCBSI class!”; print “Hello, ISU-BCBSI class!”;
Output	File	Open(MYFILEHANDLE, “>mySeqFileName”); print MYFILEHANDLE $m_line; Close(MYFILEHANDLE);

File IO	Desc
<	read
>	write
+>	read and write
>>	append
| CMD	open a pipe to CMD
CMD |	open a pipe from CMD

File Test Operator	desc
-r	readable
-x	executable
-e	exists
-d	a directory?

Out2File_FASTA.pl

Output the DNA sequence to a file in FASTA format. The FASTA format description is:
http://ngfnblast.gbf.de/docs/fasta.html

Subroutine & Functions

1. No difference for subroutine and functions in perl. Use subroutine all the time.
2. How to call sub: & is used before the sub name
3. Arguments are passed in the subroutine by a special array @_.

```
#demo calls subroutine

$g_msg=“calling from outside”;
&get_msg($g_msg);
print $g_msg, “\n”;

sub get_msg()
{
   $m_msg= $_[0];     # Arguments are listed  in @_
   print $m_msg, “\n”;
   $g_msg =“hello from sub”; # the global variable is updated here!!!! Scope issue
}
```

Anti-bugging

```
#all variables must be declared before being used.
use strict;

# give warning msg
use warnings;

# a little bit more info than use warnings
use diagnostics;

# Limit the scope of variables
 my(): lexical scope, visible in the scope of the current block that defines it
    only.

 local(): does not create a private variable, but let the global variable has
    a temp value and be restored to the old values when the variable is
    out of scope.
```

debugging

1. print out or comment out

2 run perl debugger

perl -d your script.pl

db> h

db>n # next step

db> x $your_var # to exam the value

More String Manipulation:split, match & regular expression

Ref Ch7, by Michael Moorhouse and Paul Barry

```
Split:
$NTStr = "A:T:G:C";
@NTlist= split(/:/, $NTStr );


Matching:
$NTStr="ATGCAAAAAAA";
$NTStr =~ /GC/;

Substitute:
$NTStr="ATGCAAAAAAA";
$NTStr  =~ s/AAA/A/

Translation: character-by-character
translation

$NTStr = "ATGCAAAAAAA";
$NTStr =~ tr/ATGC/TACG/ ;
```

summary

Data Types, Variables and Operations
Control Structures
Basic IO
Subroutines & Functions
More String Manipulation
Debug