

Lecture 4: Perl Tutorial

- Tutorial on Programming in Perl

Common Programming Language Constructs

- Variables and Assignment
- Variable types: Arrays, Hashes, and Scalars
- Arithmetic Operators
- Logical Operators
- Conditional Statements
- Loops
- Input/Output
- Array and Hash Functions
- Regular Expressions
- Database Functions

Why Perl?

§ Perl = **P**ractical **E**xtraction and **R**eport Language

§ Advantages:

- Scripting Language: fast to write
- Relatively easy first language to learn
- Built-in tools: Regular Expressions, etc.
- Runs on most operating systems: Linux, Mac, Windows
- Perl community: CPAN, modules.
- Bioinformatics tools: Bioperl
- Web programming: CGI.pm or mod_perl

§ To learn more about Perl:

- *Learning Perl* by Randal L. Schwartz and Tom Christiansen
- *Beginning Perl for Bioinformatics* by James Tisdall
- *Programming Perl* by Larry Wall and Tom Christiansen
- www.perl.com/perl and www.perl.com/cpan

Perl: Basic Syntax Rules

§ Statements are terminated by a semi-colon

- `print ("Hello!\n");`

§ Text blocks are demarcated by curly brackets

- `if ($a == $b) {
 print ("a = b!\n");
}`

§ Comments are indicated by a sharp sign (rest of line is a comment)

- `$a = 10; # Set $a equal to ten.`

§ Separate variable names and tokens with spaces, otherwise space has no meaning.

- `$a + $b;` is the same as `$a + $b;`

§ Common conventions: `\n` = newline, `\t` = tab, `" "` = string

§ Order matters: statements are evaluated in descending order

Perl: Assignment

Assignment:

§ Equal sign represents variable assignment

- `$text = "B";`

§ Binary assignment operators:

- `$numA = $numA + 5; => $numA += 5;`
- `$B = $B - 6; => $B -= 6;`

Perl: Variable types

§ Dollar-sign (\$) variable represents a scalar; a scalar can be a number or a string

- `$DNA_length = 20;`
- `$DNA_sequence = "ATTAGCCGAATTGGCCAAGG";`

§ At-sign (@) variable represents an array

Dollar-sign (\$) sign represents individual array element

- `@DNA = ("A","T","T","A","G","C","C","G","A","A","T","T","G","G","C","C","A","A","G","G");`
- `$DNA[0]` is equal to "A"; `$DNA[1]` is equal to "T"; etc.

§ Percent sign (%) variable represents a hash.

Dollar-sign (\$) represents individual hash element

- `%TF = ("GAL4" => "CGGNNNNNNNNNNCCG", "HSE" => "NGAAN");`
- `$TF{"HSE"}` is equal to "NGAAN";

Perl: Arithmetic Operators

Arithmetic Operators:

§ Addition: `$a = 5 + 6;` # \$a equals 11

§ Subtraction: `$a = 6 - 4;` # \$a equals 2

§ Multiplication: `$a = 3 * 2;` # \$a equals 6

§ Division: `$a = 6 / 2;` # \$a equals 3

§ Modulus: `$a = 3 % 2;` # \$a equals 1

§ Auto-increment: `$a = 0; $a++;` # \$a is equal to 1

Perl: Logical Operators

Logical Operators:

§ Identity test: if (`$a == 6`) # \$a is equal to 6?

§ Not equals test: if (`$a != 6`) # \$a is not equal to 6?

§ Less than, greater than: `$a < 6; $b > 5`

§ AND operator: `$a == 6 && $b > 5;` # \$a is equal to 6 AND \$b is greater than 5

§ OR operator: `$a == 6 || $b < 4;` # \$a is equal to 6 OR \$b is less than 4

§ Comparing strings: if (`$a eq "Hello"`) # \$a is the string "Hello"?

§ Comparing strings: if (`$a ne "Hi"`) # \$a is not equal to the string "Hi"?

Perl: String Functions

String functions:

§ String concatenate operator is a period (.)

- `$a = "Hello"; $b = "World";`
- `$c = $a . $b; # $c is "HelloWorld"`

To space between words: `$d = $a . " " . $b;`

§ String length: `$length = length($string);`

- `$text_length = length($c); # $text_length is 10`

§ String reverse operator: `$rev_string = reverse($string);`

- `$rev_c = reverse ($c); # $rev_c is "dlroWolleH"`

Perl: Array Functions

§ Split function: splits a string into an array of letters

- `$seq = "ATAGCCAT";`
`@DNA = split(//, $seq); # $DNA[0] is "A", $DNA[1] is "T", etc.`

§ Push/Pop: Push adds value to end of array; pop removes last value of array

- `push (@DNA, "G"); # @DNA is ["A","T","A","G","C","C","A","T","G"]`
- `$last = pop (@DNA); # $last is "G", @DNA is ["A","T","A","G","C","C","A","T"]`

§ Reverse: reverses order of the array

- `@DNA = reverse (@DNA); # @DNA is ["T","A","C","C","G","A","T","A"]`

§ Length of array: `scalar @array`

- `$size_of_array = scalar @DNA; # $size_of_array is 8`

Perl: Conditional statements

§ if/else statements:

- `if (statement) {do if statement is true}`
- `else {do if statement is false}`
- `if ($DNA eq "A") {
 print ("DNA is equal to A\n");
}`
- `elsif ($DNA eq "T") {
 print ("DNA is equal to T\n");
}`
- `else {
 print ("DNA is not A or T\n");
}`

Perl: Loops

§ for statements:

- `for (initial expression; test expression; increment expression) {
 do statement}`
`# @DNA is ["T","A","C","C","G","A","T","A"]`
- `for ($i = 0; $i < 8; $i++) {
 print ("position $i equals: $DNA[$i]\n");
}`

Output of loop:

```
position 0 equals: T
position 1 equals: A
position 2 equals: C
position 3 equals: C
position 4 equals: G
position 5 equals: A
position 6 equals: T
position 7 equals: A
```

Perl Loops (continued)

§ while statements:

- **while** (statement) {do if statement is true, evaluate while again}
\$i = 0;
while (\$i < \$size_of_array) {
 print ("position \$i equals: \$DNA[\$i]\n");
 \$i++;
}

§ foreach statements:

- **foreach** \$i (@some_list) {do statement}
- foreach \$i (@DNA) {
 print ("\$i\n");
}

Output: T
A
C
G
A
T
A

Perl: Input/Output

§ Standard input: <STDIN>

- \$a = <STDIN>;
 # \$a equals line of text inputted by user

§ Standard output: print statement

- **print** ("Hello World!"); # prints: Hello World!
- **print** "Value of a: \$a\n";
 # prints: Value of a: [value of \$a] to the screen

§ Opening File: open (FILEHANDLE, "filename");

- **open** (DNASEQ, "dnaseq.txt");

§ Reading File: single line: \$DNA_seq = <FILEHANDLE>; whole file: @DNA = <FILEHANDLE>;

- \$DNA_seq = <DNASEQ>;
 # Reads single line from dnaseq.txt file

Perl: Regular Expressions

§ Matching: \$string =~ /pattern/

- \$DNA = "ATATAAAGA";
if (\$DNA =~ /TATA/) {
 print ("Contains TATA element\n");
}

§ Substitution: \$string =~ s/pattern/replacement pattern/(g)

(note: **g** results in global replacement, otherwise just replaces first match)

- \$DNA =~ s/TATA/GGGG/g; # \$DNA is now "AGGGGAAGA"

§ Transliteration: \$string =~ tr/ATCG/TAGC/

- Will replace "A" with "T", "T" with "A", "C" with "G", and "G" with "C"
- \$DNA =~ tr/ATCG/TAGC/; # \$DNA is now "TCCCCTTCT";

Perl: Regular Expressions (continued)

§ Wildcards:

- [ATGC] matches A or T or G or C
- [^0-9] matches all non-digit characters
- A{1,5} matches a stretch of 1 to 5 "A" characters

Example:

- \$DNA = "TCCCCTTCT";
- \$DNA =~ /[AT]C{1,4}T/; # Does this find a match?

Example Perl Program: ReverseComp.pl

§ Want to write a Perl program that will calculate the reverse complement of a DNA sequence provided by the software user.

§ Program tasks:

- (1) Obtain DNA sequence from user.
- (2) Calculate the complement of the DNA sequence.
 - A => T; T => A; G => C; C => G
- (3) Reverse the order of the DNA sequence.
- (4) Output the calculated reverse complement DNA sequence to user.

Implementing Reverse Complement Algorithm in Perl

```
# ReverseComp.pl => takes DNA sequence from user
# and returns the reverse complement

print ("Please input DNA sequence:\n");
$DNA = <STDIN>;

$DNA =~ tr/ATGC/TACG/; # Find complement of DNA sequence

$DNA = reverse($DNA);      # Reverse DNA sequence

print ("Reverse complement of sequence is:\n");

print $DNA . "\n";
```

Exact Sequence Matching Algorithm in Perl

Perl code:

```
# @DNA_Q is an array containing each nucleotide of the query sequence as an
# array element (from user)
# @DNA_T is an array containing each nucleotide of the chromosome sequence as
# an array element.

# Find length of Query and Template (Chromosome) arrays
$length_Q = scalar @DNA_Q;
$length_T = scalar @DNA_T;

# Initialize number of matches counting variable
$num_matches = 0;

# Search for sequence match and print position of match if found.
for ($i = 0; $i <= ($length_T - $length_Q); $i++) {
    for ($j = 0; $j < $length_Q && $DNA_Q[$j] eq $DNA_T[($i + $j)]; $j++) {
        if ($j == ($length_Q - 1)) {
            print ("Found match at position $i in chromosome\n");
            $num_matches++;
        }
    }
}
```