<table>
<thead>
<tr>
<th>Date</th>
<th>Contents</th>
<th>Lecturer/TA</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/18</td>
<td>DZPerl editor, debugging, and introduction to final project</td>
<td>楊永正、劉玉凡</td>
</tr>
<tr>
<td>2/25</td>
<td>Basic Perl review</td>
<td>張傳雄、劉玉凡</td>
</tr>
<tr>
<td>3/3</td>
<td>Hash, reference, and complex data structure</td>
<td>張傳雄、劉玉凡</td>
</tr>
<tr>
<td>3/10</td>
<td>Concept of object-oriented programming</td>
<td>張傳雄、劉玉凡</td>
</tr>
<tr>
<td>3/17</td>
<td>Use Perl and BioPerl modules</td>
<td>張傳雄、劉玉凡</td>
</tr>
<tr>
<td>3/24</td>
<td>Database (MySQL)</td>
<td>張傳雄、鄧詠文</td>
</tr>
<tr>
<td>3/31</td>
<td>CGI modules and web services</td>
<td>張傳雄、鄧詠文</td>
</tr>
<tr>
<td>4/7</td>
<td>Graphics: GD module</td>
<td>張傳雄、傅瓊玲</td>
</tr>
<tr>
<td>4/14</td>
<td>How to write Perl modules</td>
<td>張傳雄、劉玉凡</td>
</tr>
<tr>
<td>4/21</td>
<td>XML introduction</td>
<td>許鈞南、劉玉凡</td>
</tr>
<tr>
<td>4/28</td>
<td>XML: KGML, etc.</td>
<td>張佑誠、劉玉凡</td>
</tr>
<tr>
<td>5/5</td>
<td>Use YMBC modules</td>
<td>張傳雄、劉玉凡</td>
</tr>
<tr>
<td>5/12</td>
<td>Introduction to Ensembl</td>
<td>張傳雄、賴俊吉</td>
</tr>
<tr>
<td>5/19</td>
<td>Project discussion and progress report</td>
<td>張傳雄、劉玉凡</td>
</tr>
<tr>
<td>5/26</td>
<td>水上運動會停課一天</td>
<td></td>
</tr>
<tr>
<td>6/2</td>
<td>Project discussion and progress report</td>
<td>張傳雄、劉玉凡</td>
</tr>
<tr>
<td>6/9</td>
<td>Project discussion and progress report</td>
<td>張傳雄、劉玉凡</td>
</tr>
<tr>
<td>6/16</td>
<td>Project final presentation</td>
<td>張傳雄、楊永正</td>
</tr>
</tbody>
</table>
Reference book


- 天璣圖書有限公司
  - 台北市重慶南路一段107號
  - (02)2331-8868

Copyright 2004 © NYMU Bioinformatics
Reading in reference book

- Chapter 1. An Overview of Perl
- Chapter 2. Bits and Pieces
- Chapter 3. Unary and Binary Operators
- Chapter 4. Statements and Declarations
Operating environment and software

- Operating environment
  - Microsoft XP
  - Linux RedHat 9.0

- Software
  - Active Perl (freeware)
    http://www.activestate.com/Products/ActivePerl/
  - DZPerl (shareware)
    http://www.dzsoft.com/dzperl.htm
Write a program to count and display the different sites between the “TW.txt” and “HK.txt”

- Download sequences of “33411399” and “30023963” from NCBI Entrez database save as “TW.txt” and “HK.txt”, respectively.
- Display total nucleotide number for each sequence and the positions of different nucleotides between the “TW.txt” and “HK.txt”.
The format for homework

Student_ID: g389030
Homework_ID: 1 (第幾次)

Program_Name: homework1.pl
__Start__
#!/usr/bin/perl -w

__End__

Program_Name: homework2.pl
__Start__
#!/usr/bin/perl -w

__End__
The results of Test1

Taiwan SARS: 29727 bp
HongKong SARS: 29742 bp

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1782</td>
<td>T =&gt; C</td>
<td></td>
</tr>
<tr>
<td>2601</td>
<td>T =&gt; C</td>
<td></td>
</tr>
<tr>
<td>3852</td>
<td>C =&gt; T</td>
<td></td>
</tr>
<tr>
<td>7930</td>
<td>G =&gt; A</td>
<td></td>
</tr>
<tr>
<td>8387</td>
<td>G =&gt; C</td>
<td></td>
</tr>
<tr>
<td>8417</td>
<td>G =&gt; C</td>
<td></td>
</tr>
<tr>
<td>11493</td>
<td>T =&gt; C</td>
<td></td>
</tr>
<tr>
<td>13494</td>
<td>G =&gt; A</td>
<td></td>
</tr>
<tr>
<td>13495</td>
<td>T =&gt; G</td>
<td></td>
</tr>
<tr>
<td>18065</td>
<td>G =&gt; A</td>
<td></td>
</tr>
<tr>
<td>25569</td>
<td>T =&gt; A</td>
<td></td>
</tr>
<tr>
<td>26477</td>
<td>C =&gt; T</td>
<td></td>
</tr>
<tr>
<td>26600</td>
<td>C =&gt; T</td>
<td></td>
</tr>
<tr>
<td>29728</td>
<td>=&gt; A</td>
<td></td>
</tr>
<tr>
<td>29729</td>
<td>=&gt; A</td>
<td></td>
</tr>
<tr>
<td>29730</td>
<td>=&gt; A</td>
<td></td>
</tr>
<tr>
<td>29731</td>
<td>=&gt; A</td>
<td></td>
</tr>
<tr>
<td>29732</td>
<td>=&gt; A</td>
<td></td>
</tr>
<tr>
<td>29733</td>
<td>=&gt; A</td>
<td></td>
</tr>
<tr>
<td>29734</td>
<td>=&gt; A</td>
<td></td>
</tr>
<tr>
<td>29735</td>
<td>=&gt; A</td>
<td></td>
</tr>
<tr>
<td>29736</td>
<td>=&gt; A</td>
<td></td>
</tr>
<tr>
<td>29737</td>
<td>=&gt; A</td>
<td></td>
</tr>
<tr>
<td>29738</td>
<td>=&gt; A</td>
<td></td>
</tr>
<tr>
<td>29739</td>
<td>=&gt; A</td>
<td></td>
</tr>
<tr>
<td>29740</td>
<td>=&gt; A</td>
<td></td>
</tr>
<tr>
<td>29741</td>
<td>=&gt; A</td>
<td></td>
</tr>
<tr>
<td>29742</td>
<td>=&gt; A</td>
<td></td>
</tr>
</tbody>
</table>
The answer of Test1

```
#!/usr/bin/perl

open TW, "C:/TW.txt";
while ($line = <TW>) {
    chomp($line);
    $TW_seq .= $line;
}
close TW;
@list_TW = split('//', $TW_seq);

open HK, "C:/HK.txt";
while ($line = <HK>) {
    chomp($line);
    $HK_seq .= $line;
}
close HK;
@list_HK = split('//', $HK_seq);

for my $pos (0..$#list_HK) {
    if ($list_TW[$pos] ne $list_HK[$pos]) {
        $real_pos = $pos+1;
        print $real_pos,"\t",$list_TW[$pos],"
";
    }
}
```

## Concept of Test1

### @list_TW

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>T</td>
<td>A</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>A</td>
<td>G</td>
<td>G</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>A</td>
<td>C</td>
<td>C</td>
<td>T</td>
<td>A</td>
<td>C</td>
</tr>
</tbody>
</table>

### @list_HK

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>T</td>
<td>A</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>A</td>
<td>G</td>
<td>G</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>A</td>
<td>C</td>
<td>C</td>
<td>T</td>
<td>A</td>
<td>C</td>
</tr>
</tbody>
</table>

![Diagram](image)

Not equal

20 C => A

Copyright 2004 © NYMU Bioinformatics
# Kimura two-parameter

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Transition</td>
<td>Transition</td>
<td>Transversion</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Transversion</td>
<td>Transversion</td>
<td>Transition</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>Transition</td>
<td>Transversion</td>
<td>Transversion</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>Transversion</td>
<td>Transition</td>
<td>Transversion</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.6</td>
<td>0.1</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td>C</td>
<td>0.1</td>
<td>0.6</td>
<td>0.1</td>
<td>0.2</td>
</tr>
<tr>
<td>G</td>
<td>0.2</td>
<td>0.1</td>
<td>0.6</td>
<td>0.1</td>
</tr>
<tr>
<td>T</td>
<td>0.1</td>
<td>0.2</td>
<td>0.1</td>
<td>0.6</td>
</tr>
</tbody>
</table>

K2P model
Homework 1

- Download sequences of “33411399” and “30023963” from NCBI Entrez database save as “TW.txt” and “HK.txt”, respectively.
- Please calculate the ratio of transition and transversion between the two sequences base on the pair-wised alignment

\[ R = \frac{\# \text{ transition}}{\# \text{ transversion}} \]

Email: perl@ym.edu.tw
Numbers

- All numbers are the same Format internally
- Floating-Point Literals
  - 1.25
  - 255.000
  - 7.25e45
  - -1.2e-23
- Integer Literals
  - 0
  - 2003
  - -40
  - 61298040283768 (61_298_040_283_768)
- Nondecimal integer literals
  - 0377 (octal same as 255 decimal)
  - 0xff (hexadecimal, also 255 decimal)
  - 0b11111111 (binary, also 255 decimal)
Numeric operators

- addition (2 + 3)
- subtraction (5.1 - 2.4)
- multiplication (3 * 12)
- division (14 / 2)
- modulus (10 % 3)
- exponentiation (2**3)
Strings

- Typical strings are printable sequence of letters and digits and punctuation in the ASCII32 to ASCII126 range.

- Single-Quoted String Literals
  - 'fred'
  - 'Don\'t let an apostrophe end this string prematurely!'
  - 'hello world\n'
  - '\''

- Double-Quoted String Literals
  - "barney"
  - "hello world\n"
  - "coke\tsprite"

- Double-Quoted String Backslash escape
<table>
<thead>
<tr>
<th>Escape</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\n</td>
<td>Newline</td>
</tr>
<tr>
<td>\r</td>
<td>Carriage Return</td>
</tr>
<tr>
<td>\t</td>
<td>Tab</td>
</tr>
<tr>
<td>\f</td>
<td>Formfeed</td>
</tr>
<tr>
<td>\b</td>
<td>Backspace</td>
</tr>
<tr>
<td>\v</td>
<td>Vertical Tab</td>
</tr>
<tr>
<td>\a</td>
<td>Bell</td>
</tr>
<tr>
<td>\e</td>
<td>Escape</td>
</tr>
<tr>
<td>\001</td>
<td>Octal ASCII value (here Ctrl-A)</td>
</tr>
<tr>
<td>\x20</td>
<td>Hex ASCII value (here space)</td>
</tr>
<tr>
<td>\cD</td>
<td>Control character (here Ctrl-D)</td>
</tr>
<tr>
<td>\</td>
<td>Backslash</td>
</tr>
<tr>
<td>&quot;</td>
<td>Double Quote</td>
</tr>
<tr>
<td>\l</td>
<td>Lowercase next letter</td>
</tr>
<tr>
<td>\L</td>
<td>Lowercase all following letters until \E</td>
</tr>
<tr>
<td>\u</td>
<td>Uppercase next letter</td>
</tr>
<tr>
<td>\U</td>
<td>Uppercase all following letters until \E</td>
</tr>
<tr>
<td>\E</td>
<td>Terminate \L or \U</td>
</tr>
</tbody>
</table>
String operators

- String concatenation ("Hello"."Liu")
- String repetition ("mcu" x 3)
- Automatic conversion between numbers and string
Scalar variables

- A variable is a name for a container that holds one or more values.
- Scalar variable names begin with a dollar sign ($) followed by what we'll call a Perl identifier:
  - A letter or underscore, and then possible more letters, or digits, or underscores
    - $fred
    - $a_very_long_variable_that_ends_in_1
    - $_name
    - $123abc     # wrong identifier
- Choosing good variable names - most variable name prefer lowercase:
  - $r                  # not very descriptive
  - $super_bow # better name
  - $ARGV         # is special identifier for internal using
Scalar assignment

- The most common operation on a scalar variable is assignment, which is the way to give a value to a variable.
  - \$fred = 17;
  - \$barney = 'hello';
  - \$barney = \$fred + 3;
  - \$barney = \$barney *2;

- Binary assignment operators
  - \$fred = \$fred +5;
  - \$fred += 5;
  - \$barney = \$barney * 2;
  - \$barney *= 2
  - \$str = \$str."hello";
  - \$str .= "hello";
Interpolation of Scalar Variables into strings

- Substitution of a scalar variable reference with its value done inside a double-quoted string literal.
- Can use `{}` around the name of a variable to delimit it.

```plaintext
$name = "Bob Tarr";
$str1 = "My name is $name";  # $str1 is My name is Bob Tarr
$str2 = "My name is $names";  # $str2 is My name is
$str3 = "My name is ${name}s";  # $str3 is My name is Bob Tarrs

$x = '$name';
y = "$x";  # $y is $name
```
Operator precedence and associativity

- 2+3*4
- (2+3)*4

Homework 2: The problem of associativity

- 4**3**2
  - [A]: 64**2
  - [B]: 4**9

Email: perl@ym.edu.tw

Copyright 2004 © NYMU Bioinformatics
Comparison operators

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Numeric</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equal</td>
<td>==</td>
<td>eq</td>
</tr>
<tr>
<td>Not equal</td>
<td>!=</td>
<td>ne</td>
</tr>
<tr>
<td>Less than</td>
<td>&lt;</td>
<td>lt</td>
</tr>
<tr>
<td>Greater than</td>
<td>&gt;</td>
<td>gt</td>
</tr>
<tr>
<td>Less than or equal to</td>
<td>&lt;=</td>
<td>le</td>
</tr>
<tr>
<td>Greater than or equal to</td>
<td>&gt;=</td>
<td>ge</td>
</tr>
</tbody>
</table>

- 35 != 30 + 5          # false
- 35 == 35.0            # true
- '35' eq '35.0'       # false
- 'fred' lt 'barney'    # false
- 'fred' eq 'fred'      # true

Copyright 2004 © NYMU Bioinformatics
The **if else** control structure

- Once you can compare two values, you’ll probably want your program to make decisions based upon that comparison.

```perl
#!/C:/Perl/bin/perl.exe
#
# The if Control structure
#
if ($name eq 'fred') {
    print "Hello world!".$name."\n";
}
if ($name eq 'fred') {
    print "Hello world!".$name."\n";
} else {
    print "The name is not fred!\n";
}
```
The **chomp** operator

- If the string ends in a newline character, chomp can get rid of the newline.

```perl
#!/C:\Perl\bin\perl.exe
#
# The chomp operator a
#
$text = "a line of text\n";
chomp($text);
```
The **while** control structure

- The while loop repeats a block of code as long as a condition is true

```perl
#!/C:\Perl\bin\perl.exe
#
# The while operator
#
$count = 0;
while ($count < 10) {
    $count += 1;
    print "count is now $count\n";
}
```
Get user **input**

- You can get the value or string from keyboard into Perl program by using “line-input” operator `<STDIN>`

```perl
#!/C:/Perl/bin/perl.exe
#
# The if Control structure
# Run in command prompt
#
print "Enter your name? ";
$name = <STDIN>;

if ($name eq "\n") {
    print "That was just a blank line\n";
} else {
    print "Hello world! ".$name."\n";
}
```
Homework 3

- Write a program to do complementary sequence
  - Function 1: Could let user key in the sequence (either upper or lower case)
  - Function 2: Error check (not ATGC bases)
  - Function 3: Display the user key in sequence and complementary sequence

Email: perl@ym.edu.tw
What is Lists?

- A list is a sequence of scalar values enclosed in parentheses.
- A list is an ordered collection of scalars.
- Each element of an array or list is a separate scalar variable with an independent scalar value.
- Example - (35, 12.4, "hello", 1.72e30, "bye\n")

<table>
<thead>
<tr>
<th>Element number</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>35</td>
</tr>
<tr>
<td>1</td>
<td>12.4</td>
</tr>
<tr>
<td>2</td>
<td>&quot;hello&quot;</td>
</tr>
<tr>
<td>3</td>
<td>1.72e30</td>
</tr>
<tr>
<td>4</td>
<td>&quot;bye\n&quot;</td>
</tr>
</tbody>
</table>
Accessing elements of an Array

Fred = ("yabba", "dabba", "doo", "hello", "world");

Fred[0] = "yabba";
Fred[1] = "dabba";
Fred[2] = "doo";

Homework 4
Fred[3] = ?
Fred[4] = ?

Email: perl@ym.edu.tw
Special Array Indices

- $rocks[0] = 'bedrock';
- $rocks[1] = 'slate';
- $rocks[2] = 'lava';
- $rocks[3] = 'crushed rock';
- $rocks[99] = 'schist';

- $end = $#rocks;              # 99 last element’s index
- $number_of_rocks = $#rock+1  # 100
- $#rocks = 2                  # forget all rocks after ‘slate’
- $#rocks = 99                 # add 97 undef elements
- $rocks[$#rocks] = 'hard rock';
- $rock[-1] = 'hard rock';
- $dead_rock = $rocks[-100]    # gets ‘bedrock’
- $rocks[-200] = 'crystal'     # fatal error
Practice by yourself - random number generator

```
#!/C:/Perl/bin/perl.exe

# collect the random numbers
$count = 1;
while ($count <= 100) {
    $randnum = int( rand(10) ) + 1;
    $randtotal[$randnum] += 1;
    $count += 1;
}

# print the total of each number
$count = 1;
print ("Total for each number:\n");
while ($count <= 10) {
    print ("\tnumber $count: $randtotal[$count]\n");
    $count += 1;
}
```
List literals

- (1,2,3)  # list of three values 1, 2, and 3
- (1,2,3,)  # the same three values (the trailing comma is ignored)
- (“fred”, 4.5)  # two values, “fred” and 4.5
- ()  # empty list – zero element
- (1..100)  # list of 100 integers
- (1..5)  # same as (1,2,3,4,5)
- (1.7..5.7)  # same as (1,2,3,4,5)
- (5..1)  # empty list
- (1, 2..6, 10, 12)  # same as (1,2,3,4,5,6,10,12)
- ($a..$b)  # range determine by current values of $a and $b
The qw shoutcut

qw stands for “quoted words” or “quoted by whitespace”

Example

qw /fred barney betty wilma dino/
eq(“fred”, “barney”, “betty”, “wilma”, “dino”)
List Assignment

- ($fred, $barney, $dino) = ("flintstone", "rubble", undef)

- ($fred, $barney) = ($barney, $fred)  # swap those values
- ($fred, $barney) = qw/flintstone rubble slate granite/

- @rocks = qw/talc mica feldspar quartz/
- ($rocks[0], $rocks[1], $rocks[2], $rocks[3]) = qw/talc mica feldspar quartz/

- @tiny = (1..3)
- @stuff = (6,9)
- @all = (@tiny, @stuff)
- @all = (1,2,3,6,9)

- @copy = @quarry  # copy a list from one array to another
Homework 5 - List Assignment

- \( @\text{tiny} = (1..16) \)
- \( @\text{stuff} = (\text{"nymu"}, \text{"taigen"}, \text{"mcu"}) \)
- \( @\text{all} = (@\text{tiny}, @\text{stuff}) \)

Please answer following questions

- \$\text{all}[1] = ? \)
- \$\text{all}[-1] = ? \)
- \$\text{all}[12] = ? \)
- \$\text{all}[14] = ? \)

Email: perl@ym.edu.tw
The pop and push operators

- You could remove from the right-side of the list by pop operator
  - `@array = (5,6,7,8,9)`
  - `$fred = pop(@array)`  # $fred gets 9, @array now has (5,6,7,8)
  - `$barney = pop(@array)`  # $barney gets 8, @array now has (5,6,7)
  - `pop(@array)`  # @array now has (5,6) (7 is discarded)

- You could add new items to the end of an array by push operator
  - `@array = (5,6)`
  - `push (@array, 0)`  # @array now has (5,6,0)
  - `push (@array, 1..4)`  # @array now has (5,6,0,1,2,3,4)
  - `@others = (9, 10, 11)`
  - `push(@array, @others)`  # @array now has (5,6,0,1,2,3,4,9,10,11)
The shift and unshift operators

- You could remove from the left-side of the list by shift operator

  ```perl
  @array = qw(dino fred barney)
  $a = shift(@array) # $a get “dino”, @array now has (“fred”, “barney”)
  $b = shift @array # $b get “fred”, @array now has (“barney”)
  shift(@array) # @array is now empty
  ```

- You could add new items to the end of an array by unshift operator

  ```perl
  @array = (1,2);
  unshift(@array,5) # @array now has (5,1,2)
  unshift @array 6 # @array now has (6,5,1,2)
  @others = (3..4)
  unshift(@array, @others) # @array now has (3,4,6,5,1,2)
  ```
The foreach control structure

- It’s handy to be able to process an entire array or list, so Perl provides a control structure to do just that.
- The `foreach` loop steps through a list of values, executing one iteration (time through the loop) for each value.

```perl
#!/C:/Perl/bin/perl.exe
@rocks = qw/ bedrock slate lava /;
foreach $rock (@rocks) {
    $rock = "\t$rock";
    $rock .= "\n";
}
print "The rocks are: \n", @rocks;
```
Perl’s Favorite Default : 

- If you omit the control variable from the beginning of the `foreach` loop, Perl uses its favorite default variable, `$_`.
- Example

```perl
#!/C:\Perl\bin\perl.exe

foreach (1..10) {
    print "I can count to \$_!\n";
}
```

Output:
```
I can count to 1!
I can count to 2!
I can count to 3!
I can count to 4!
I can count to 5!
I can count to 6!
I can count to 7!
I can count to 8!
I can count to 9!
I can count to 10!
```
The reverse operator

- The reverse operator takes a list of values (which may come from an array) and returns the list in the opposite order.

- \texttt{@fred} = 6..10
- \texttt{@barney} = reverse (@fred)  \# gets 10, 9, 8, 7, 6
- \texttt{@wilma} = reverse 6..10  \# gets 10, 9, 8, 7, 6
- \texttt{@fred} = reverse @fred  \# gets 6, 7, 8, 9, 10

Copyright 2004 © NYMU Bioinformatics
The sort operator

- The sort operator takes a list of values (which may come from an array) and sorts them in the internal character ordering. For ASCII strings, that would be ASCII betical order.

```
@rocks = qw/ bedrock slate rubble granite/

@sorted = sort (@rocks)  # gets “bedrock”, “granite”, “rubble”, “slate”

@back = reverse sort @rocks  # gets “slate”, “rubble”, “granite”, “bedrock”

@rocks = sort @rocks

@number = sort (97..102)  # gets 100, 102, 102, 97, 98, 99
```
Sort numerically ascending or descending

- sort numerically ascending
  ```perl
  #!/C:/Perl/bin/perl.exe
  @list = (5, 10, 12, 34, 12);
  @sorted = sort { $a <=> $b } @list
  ```

- sort numerically descending
  ```perl
  #!/C:/Perl/bin/perl.exe
  @list = (5, 10, 12, 34, 12);
  @sorted = sort { $b <=> $a } @list
  ```

Copyright 2004 © NYMU Bioinformatics
More advance feature of If control structure

- simple *if statement* in Perl has the following syntax.

```perl
#!/C:\Perl\bin\perl.exe

test_var = 0;

if ( test_var < 1 ) {
    print "test_var is less than one\n";
}
```

- In this example, Perl executes a print statement if `$test_var` is less than one. In this case the condition is true so the statement is printed. But what if the *if statement* had evaluated to false? Well, nothing would have happened.
More advance feature of If control structure

- If you want to find out how $test_var is being evaluated, add an `else` block, which divides the `if statement` into two sections: one for `true`, and one for `false`.

```perl
#!/C:\Perl\bin\perl.exe
$test_var = 6;
if ( $test_var < 1 ) {
    print "test_var is less than one\n";
} else {
    print "test_var is greater than one\n";
}
```
More advance feature of If control structure

- You can optionally expand on this logic by adding an `elsif` block. If the condition evaluates as `false`, then Perl drops to the `elsif` block and evaluates another conditional. If that is also `false`, Perl will drop to the `else` block and execute the code there.
The **split** operator for transfer string into list

- `@fields = split /separator/, string`

**example:**

- `@fields = split /:/, "abc:def:g:h"`
- `@fields = ("abc","def","g",f");`
Homework 6

- Write a program to count the A, T, G, C number of key in sequence
  - Function 1 - user key in sequence (either upper or lower cases)
  - Function 2 - error check
  - Function 3 - Display the total nucleotides and A, T, G, C numbers

```
Please key in your DNA sequence? ATGCATGCAATGCATT
You key in DNA sequence is: ATGCATGCAATGCATT
The length of sequence is: 16 nucleotides
Total 4 are A symbol.
Total 6 are T symbol.
Total 3 are G symbol.
Total 3 are C symbol.
您按任意键繼續... 
```

Email: perl@ym.edu.tw