Perl - BioPerl

Université Henri Poincaré

Master Génomique et Informatique

Automne 2008

Plan

Rappels : types de données

- 3 types de données en Perl : **scalaires**, **tableaux**, et **tableaux associatifs (hash)**
- Les scalaires :
  - **nombres** (entiers ou réels)
  - **chaînes de caractères**
- Les nombres :
  - exemples : `12 +10 -34 +3.14 5e15`
  - opérateurs : `+ - * / ** %`
  - comparaison : `< > <= >= == !=`
- Exercice : créer un script perl permettant de calculer le reste de la division de deux entiers donnés en paramètres (NB: les paramètres sont stockés dans `@ARGV`)

Rappels : types de données (suite)

- Les chaînes de caractères :
  - délimitées par une simple quote (')
    $\longrightarrow$ contenu non-interprété
  - délimitées par une double quote (")
    $\longrightarrow$ contenu interprété
- Manipulation de chaînes :
  - exemples : `"Bonjour \n" 'Bonjour \n'`
    `"\"Bonjour \n"`
  - opérateur : `.` (concaténation)
  - comparaison : `lt gt le ge eq ne`
- Exercice : écrire un script perl prenant en paramètres un nom et un prénom et qui affiche "Bonjour Prénom Nom"

Rappels : types de données (suite)

- Définition de variables :
  - scalaires : `$x="ma chaine";`
  - de tableaux : `@t = (1, "oui");`
  - de hash : `%h = ("oui" => 1, "non" => 0);`
- Accès aux données d'un tableau :
  - `@t = (1, 2, 'ok');`
  - `$x = $t[0] + 1;`
  - `$t[1] = 3;`
  - `$t[$#t + 1] = 5;`
  - `@tt = @t[2..$#t];`

Rappels : types de données (suite)

- Fonctions de manipulation de scalaires :
  - `chomp` (retire le caractère fin de ligne)
  - `chr` (converti de charactère vers ASCII)
  - `split(/séparateur/, chaîne)` (découpe une chaîne)
- Fonctions de manipulation de tableaux :
  - `push` (ajout en fin)
  - `pop` (suppression en fin)
  - `unshift` (ajout en tête)
  - `shift` (suppression en tête)
  - `sort` (trie un tableau)
  - `reverse` (retourne un tableau)
  - `join` (concatène les éléments)

Rappels : types de données (suite)

- Accès aux données d'un hash :
  - `$h{'oui'} = 2;`
- Fonctions de manipulation de hash :
  - `keys` (clés d'un hash)
  - `values` (valeurs d'un hash)
  - `each` (couples clé,valeur d'un hash)
  - `delete` (suppression d'une valeur en fonction de la clé)

Plan

Rappels : structures de contrôle

Instruction `if else`

```
if (expression booléenne) {
  instructions si succès
}
else {
  instructions si échec
}
```

Rappels : structures de contrôle (suite)

Instruction `unless`

```
unless (expression booléenne) {
  instruction si échec
}
else {
  instruction si succès
}
```

Rappels : structures de contrôle (suite)

Instruction `while`

```
while (test) {
  ...
  instructions si succès
  ...
}
```

```
until (test) {
  ...
  instructions si succès
  ...
}
```

Rappels : structures de contrôle (suite)

Instruction `for`

```
for ($i = 0 ; $i < 10 ; $i++) {
   ...
   instructions si succès
   ...
}
```

Rappels : structures de contrôle (suite)

Instruction `foreach`

```
foreach $var (@tab) {
  ...
  instructions
  ...
}
```

Plan

Rappels : manipulation de fichiers

- Ouverture de fichiers (préalable à toute action sur le fichier) :
  - `open(FDESC,"<fichier") || die "Error: $!";` (lecture)
  - `open(FDESC,">fichier") || die "Error: $!";` (écriture)
  - `open(FDESC,">>fichier") || die "Error: $!";` (écriture avec ajout)

Rappels : manipulation de fichiers (suite)

- Lecture :

```
while ($ligne = <FDESC>) {
    ...
}
```

- Ecriture :

```
print FDESC "chaîne à stocker, avec $var1, $var2";
```

- Fermeture :

```
close(FDESC);
```

Plan

Rappels : expressions régulières

- Vérification de la présence d'un motif :

```
if ($v =~ m/expr_reg/)
{
   instructions
}
```

- Substitution d'un motif :

```
$v =~ s/expr_reg/remplacement/ ;

$w =~ s#expr_reg#remplacement# ;
```

- Regroupement au sein d'un motif :

```
if ($x =~ m/:([^:]+):/)
{
   print $1; # affiche ce qu'il y a entre :
}
```

Rappels : expressions régulières (suite)

- Ensembles de caractères :

```
[qjk]      # Soit q, soit j, soit k
[^qjk]     # Ni q, ni j, ni k
[a-z]      # Tout caractère compris entre a et z
[^a-z]     # Aucun caractère compris entre a et z
[a-zA-Z]   # Tous les caractères alpha. sans accent
[a-z]+     # Toute chaîne de a-z non vide
```

- Quantificateurs :

```
.   # Tout caractère sauf le retour à la ligne
^   # Le début d'une ligne ou d'une chaîne
$   # La fin d'une ligne (\n non compris) ou d'une chaîne
+   # Une occurrence ou plus de l'expr. précédente
*   # Zéro ou plusieurs occurrences
?   # Zéro ou une occurrence
```

Rappels : expressions régulières (suite)

- Classes de caractères :

```
\n   # Retour à la ligne
\t   # Tabulation
\w   # Tout caractère d'un mot [a-zA-Z0-9_]
\W   # Aucun caractère d'un mot
\d   # Tout chiffre
\D   # Aucun chiffre
\s   # Tout séparateur
\S   # Aucun séparateur
```

Plan

Rappels : définition de fonctions

- Définition d'une fonction :

```
sub ma_fonction {
    instruction 1;
    instruction 2;
    ...
}
```

- Arguments d'une fonction contenues dans la variable `@_`
- Valeur de retour d'une fonction précédée de l'instruction `return`

  Exemple :

```
sub moyenne2 {
    my ($a, $b)=@_;
    return ($a + $b) / 2;
}
```

Rappels : définition de fonctions (suite)

- Appel d'une fonction au moyen du symbole &
- Exemple :

```
#!/usr/bin/perl

use strict;

sub moyenne2 {
   local($a, $b)=@_;
   return ($a + $b) / 2;
}

my ($a,$b) = (4,3);

my $res = &moyenne2($a,$b);
print "La moyenne de 4 et 3 est $res \n";
```

Plan

Références

- Problème : lorsqu'on passe des listes à une fonction, celles-ci sont **concaténées** et stockées dans `@_`
- Comment distinguer les deux listes ?
  → utiliser des **pointeurs** (références) sur les listes
- Création d'une référence en perl :

Règle 1 placer un \ devant la variable

```
$ref_tab = \@tab;
$ref_hash = \%hash;
```

Règle 2 utiliser des références anonymes ([] ou {})

```
$ref_tab = ["toto", 2, "mot"];
$ref_hash = {"titre" => "Tintin",
             "auteur" => "Hergé"}
```

Références (suite)

- Manipulation d'une référence :

  Règle 1 remplacer le nom de la variable par celui de la référence au format {$nom_ref}

  `@{$ref_tab}` au lieu de `@tab`

  `%{$ref_hash}` au lieu de `%hash`

  Règle 2 utiliser la notation `->` au lieu de `${...}`

  `$ref_tab->[0]` au lieu de `${$ref_tab}[0]`

  `$ref_hash->{'toto'}`
  au lieu de `${$ref_hash}{'toto'}`

- Faire exercices 3 et 4 feuille jointe.

Plan

Modules

- Un module est un fichier perl regroupant un ensemble de variables et de fonctions
- Le fichier doit se nommer `Nom_du_module.pm`
- Le fichier doit se trouver dans un répertoire contenu dans la variable `@INC`
- Structure d'un module :

```
# -- fichier Monmodule.pm

package Monmodule;
sub toto { ...
}
sub tata { ...
}
1;
```

Modules (suite)

- Utilisation d'un module dans un programme perl :

```
#!/usr/bin/perl

use strict;
use Monmodule;

Monmodule::toto("$ARGV[0]","$ARGV[1]");
```

- Possibilité de structurer des modules en répertoires :

```
# -- fichier Repertoire/Monmodule.pm

package Repertoire::Monmodule;
```

Plan

Programmation objets en Perl

- Définition de **classes d'objets** $\approx$ programmes autonomes contenant des variables (attributs) et des fonctions (méthodes)
- Intérêt des classes : cacher la complexité d'un traitement derrière une **interface relativement "simple"**
- En perl, une classe correspond à un **module** spécial
- Un **objet** (instance d'une classe) correspond à une **référence** sur une structure complexe contenant attributs et méthodes
- Il existe des méthodes de classes (indépendantes d'un objet) et des méthodes d'objets (appelées par l'objet)

Programmation objets en Perl (suite)

Création d'une classe en perl

1. Choix du nom de la classe (définition du module correspondant)
2. Définition des attributs de la classe
3. Définition des **constructeurs** (méthodes spéciales)
4. Définition des autres méthodes de la classe (interface entre les objets et le programme perl de l'utilisateur)

Programmation objets en Perl (suite)

- Un constructeur retourne une référence sur un objet, référence créée par la fonction `bless` :

```
#!/usr/bin/perl
use strict;
package Personne;
sub new {
    my($classe,$un_nom,$un_age) = @_;
    my $self = {};
    $self->{NOM} = $un_nom;
    $self->{AGE} = $un_age;
    $self->{PRENOMS} = [];
    bless($self,$classe);
    return($self);
}
1; # à ne pas oublier
```

- Le constructeur `new()` est une méthode de classe

```
$moi = Personne->new("Eric",24);
```

Programmation objets en Perl (suite)

- Une méthode est une fonction permettant de modifier la structure référencée par un objet :

```
sub nommer {
    my($Homme,$nouveau_nom) = @_;
    $Homme->{NOM} = $nouveau_nom;
}
sub dire_age {
    my $Homme = shift(@_);
    return ($Homme->{AGE});
}
```

- NB: Ces méthodes sont des méthodes d'objet
- Le 1er argument d'une méthode de classe est la classe, celui d'une méthode d'objet est l'objet

Plan

BioPerl

- Bioperl est en ensemble de modules de perl qui sont programmés en langage objet
- Installation de bioperl (ubuntu) :
  `sudo apt-get install bioperl`
- Documentation des modules bioperl :
  `http://doc.bioperl.org/bioperl-live/`
- Modules principaux (pour ce cours) :

  `Bio::Perl` (fonctions générales)

  `Bio::Seq` (représentation de séquences)

  `Bio::SeqIO` (entrées-sorties de séquences)
- Accéder au code source d'un module :
  `perldoc -m Bio::SeqIO::fasta | less`

BioPerl (suite)

- Récupération d'une séquence *à partir d'un fichier* :
  - `read_sequence(file, format)`
    (module `Bio::Perl`)
    → **fonction** retournant un objet `Bio::Seq` !
  - `Bio::SeqIO->new(-file=>"<fichier",-format=>format)`
    (module `Bio::SeqIO`)
    → **méthode** créant un objet `Bio::SeqIO` !
- Récupération d'une séquence *à partir d'internet* :
  - `get_sequence(format, identifiant)`
    (module `Bio::Perl`)
    → **fonction** retournant un objet `Bio::Seq` !
  - `Bio::DB::GenBank->new()->get_Seq_by_acc(id)`
    (module `Bio::DB::GenBank`)
    → **méthodes** créant un objet `Bio::DB::GenBank`
    puis recupérant une séquence !

BioPerl (suite)

- Ecriture d'une séquence *dans un fichier* :
  - `write_sequence(file, format, sequence)`
    (module `Bio::Perl`)
    → **fonction** ne retournant rien !
  - `Bio::SeqIO->new(-file=>">fichier",-format=>format)`
    (module `Bio::SeqIO`)
    → **méthode** créant un objet `Bio::SeqIO` !
- Parcourir les séquences contenues dans un fichier :

```
my $seq_in = Bio::SeqIO->new(-file=>"<$file",
                             -format=>'genbank');

while (my $seq=$seq_in->next_seq()) {
   write_sequence(">$fileout",'fasta', $seq);
}
```

BioPerl (suite)

- La classe `Bio::Seq` :

  (voir `http://doc.bioperl.org/releases/bioperl-current/bioperl-live/Bio/Seq.html`)

  - `$seq_obj -> seq();` (string)
  - `$seq_obj -> subseq(5,10);` (string)
  - `$seq_obj -> accession_number();` (identifiant)
  - `$seq_obj -> alphabet();` ('dna','rna' ou 'protein')
  - `$seq_obj -> seq_version();`
  - `$seq_obj -> keywords();`
  - `$seq_obj -> length();`
  - `$seq_obj -> desc();` (description)
  - `$seq_obj -> primary_id();` (identifiant unique)
  - `$seq_obj -> display_id();` (identifiant)
  - `$seq_obj -> revcom;` (complément)
  - `$seq_obj -> get_SeqFeatures();` (caractéristiques)

BioPerl (suite)

- Recherche d'une caractéristique dans une séquence :

```
foreach $feat ( $seq_obj->get_SeqFeatures() ) {
   if( $feat->primary_tag eq 'exon' ) {
         print STDOUT "Location ",$feat->start,":",
         $feat->end," GFF[",$feat->gff_string,"]\n";
   }
}
```

- Exécuter une requête sur une base de données :
  - `$req=Bio::DB::Query::GenBank->new(...)`
    (création de requête sur base GenBank)
  - `$stream=new Bio::DB::GenBank->`
    `get_Stream_by_query($req);`
    (récupération des résultats de la requête)
  - `$stream->next_seq`
    (parcours des résultats, séquence par séquence)

BioPerl (suite)

- Utiliser la documentation des classes du paquet `BioPerl` !

`http://doc.bioperl.org/releases/bioperl-current/bioperl-live/`

Ressources

- les pages man :
  - `man perldata` (Perl structures de données)
  - `man perlsyn` (Perl syntaxe)
  - `man perlop` (Perl operateurs et précédence)
  - `man perlre` (Perl expressions régulières)
  - `man perlrun` (Perl exécution et options)
  - `man perlfunc` (Perl fonctions prédéfinies)
  - `man perlref` (Perl références)
- Tutoriel Perl (Sylvain Lhullier) : `http://www.lhullier.org/publications/perl.html`
- Remerciements: ce cours a été réalisé à partir du support de Catherine Eng.