Introduction to Perl programmation & one line of Perl program

BOCS Stéphanie
DROC Gaëtan
ARGOUT Xavier

Introduction

- What is Perl ?
  - PERL (Practical Extraction and Report Language) created in 1986 by Larry Wall
  - Programming language
  - Interpreted langage (no compilation)

- Why Perl ?
  - Portability (Unix, Windows, Mac)
  - Simplicity
  - Object
  - In biology, many libraries have been created (BioPerl)

- Which uses ?
  - Manipulate files (conversion of file format) and text (search of regular expressions)
  - Handling processes (system call) and run programs (workflow)

Scalar variable: $

- Single value (number or string)
- Declaring a variable ‘my’

```
my $name; # ‘#’ comment ‘;’ indicates end of an instruction
```

- Assigning a value ‘=‘

```
my $number = 4;
my $dna = "ATCGATAGACAT";
```

- Standard input ‘<>’

```
print " Please type a number : \n";
my $number = <STDIN>;
print " Your number is $number\n";
```

Operators, functions & numerical context

- Classical operators are available: +, -, /, *, %
- Shortcuts +=, -=, *=, /=

  `$x += 3;` is equivalent to `$x = $x + 3;`
- Auto-incrementers et Auto-decrementers

  `$a++;` is equivalent to `$a = $a + 1;`

  `$a--;`
- Some mathematical functions :
  - `abs($x);` returns absolute value of $x.
  - `sqrt($x);` returns the square root of $x.
  - `int($x/$y);` is the quotient of the integer division of $x by $y.
  - `sin($x); cos($x);` return the sine and the cosine of $x.
  - `exp($x); log($x);` return *e* power $x and logarithm in base *e* of $x.

Operators, functions & string context

- Concatenation

```
$c = 'ce' . 'rise';  (=> $c becomes 'cerise')
$c .= 's';  (=> $c becomes 'cerises')
```

- Replica

```
$b = 'a' x 5;  => 'aaaaa'
$b = 'jacqu' . 'adi' x 3;  => 'jacquadiadiadi'
```

- Some string functions
  - `length($x);` returns string length $x
  - `chop($x);` remove the last character of the string $x
  - `chomp($x);` remove the last character of the string $x if it is a newline (\n)
  - `reverse($x);` returns a string composed of $X characters but in the reverse order.
  - `substr($x,offset,length);` return the substring from position 'offset' and length 'length.
  - `index($string,$substring,$position);` returns the position of the first occurrence of $substring in $string.
  - `rindex($string,$substring,$position);` same as index but starting from the end of the string

Test & Boolean operators

- Test operators

context	numerical	string
equality	`==`	`eq`
difference	`!=`	`ne`
Lower than	`<`	`lt`
Greater than	`>`	`gt`
Lower or equal	`<=`	`le`
Greater or equal	`>=`	`ge`
comparison	`<=>`	`cmp`

- Boolean operators
  - `(expr1 && expr2)` is true if expr1 AND expr2 are true
  - `(expr1 || expr2)` is true if expr1 OR expr2 are true
  - `(!expr)` is true if expr is false.

Control structures (condition)

- Defined by the instruction: IF, ELSIF and ELSE

```
if( condition1 )
      {
        instructions1;
      }
   elsif( condition2 )
      {
        instructions2;
      }
   else
     {
       instructions3;
      }
```

- Ex:

```
 if( $x == $y )
   {
    print "\$x et \$y sont égaux\n";
   }
 else
   {
    print "\$x et \$y sont différents\n";
   }
```

Control structures (loop)

- Instructions: FOR and WHILE

```
for( initialisation; condition; incrément )
    {
       instructions;
    }

while( condition )
    {
      instructions;
    }
```

- Ex:

```
for( my $i=0; $i<=20; $i+=2 )
 {
    print "$i\n";
 }

my $i = 0;
while( $i <= 20 )
 {
   print "$i\n";
   $i+=2;
 }
```

Regular expressions: set and quantifier

- Syntax:

`.`: Any character

`[]`: Any character in the brackets: e.g. [AFTR] either A or F or T or R

`\d` : an interger

`\w` : an alphanumeric character

`\s` : a space

`*` : 0 or more times

`+` : 1 or more times

`?` : 0 or 1

`{n}` : n times

- Defined variable

`$1`, `$2`, etc.: correspond to substrings that match regular expression between parentheses

- Ex :

```
my $v = "za aa et tfe";
 if( $v =~ /(a+) et ([a-z])/ )
    {
      print "$1\n";  # 'aa'
      print "$2\n";  # 't'
    }
```

$ perl -h

Usage: `perl [switches] [--] [programfile] [arguments]`

```
  -0[octal]           specify record separator (\0, if no argument)
  -a                  autosplit mode with -n or -p (splits $_ into @F)
  -C[number/list]     enables the listed Unicode features
  -c                  check syntax only (runs BEGIN and CHECK blocks)
  -d[:debugger]       run program under debugger
  -D[number/list]     set debugging flags (argument is a bit mask or alphabets)
  -e program          one line of program (several -e's allowed, omit programfile)
  -f                  don't do $sitelib/sitecustomize.pl at startup
  -F/pattern/         split() pattern for -a switch (//'s are optional)
  -i[extension]       edit <> files in place (makes backup if extension supplied)
  -Idirectory         specify @INC/#include directory (several -I's allowed)
  -l[octal]           enable line ending processing, specifies line terminator
  -[mM][-]module              execute "use/no module..." before executing program
  -n                  assume "while (<>) { ... }" loop around program
```

Exercice1: one line of Perl program

Convert a file in fasta format (2 lines per sequence) to a excel file (1 line per sequence)

Read the /usr/local/bioinfo/training/Perl/exercice1.txt

- `od -c`
- `dos2unix`
- `perl -ne`
- `Tail`
- `echo & cat`

Special characters

- `\n`: new line
- `\r`: carriage return
- `\t`: tabulation
- `\f`: new page
- `\e`: escape

Exercice2: substitution

- Functionalities : correspondence et substitution

- Syntax:

`m/pattern/`: correspondence

`s/pattern/string/`: substitution

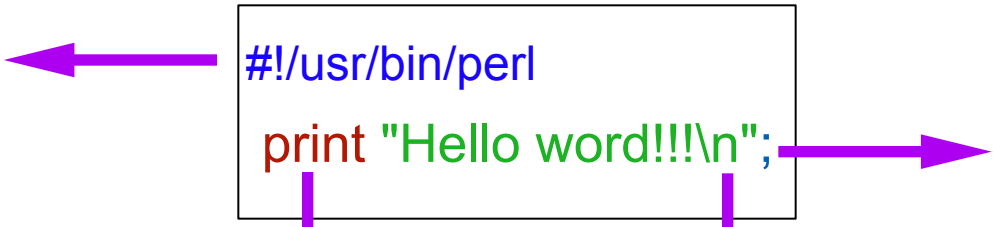
- Syntax example:

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

- Read the /usr/local/bioinfo/training/Perl/exercice2.txt

First steps to create a program

- Create a program: hello.pl



Shebang
Specifies to the operating system where is the Perl executable.
Always at the first line of the file.

The character ';' indicates the end of a instruction in the perl file (like the end of a sentence in a text)

Predefined function that will display on the screen the string "Hello world"

New line operator

Exercice3: Create a program hello.pl

- Create a program: hello_yourname.pl

```
#!/usr/bin/perl
print "Hello word!!!\n";
```

- Make the program executable

```
chmod +x hello.pl
./hello_yourname.pl
```

- Give parameters to the programm

```
#!/usr/bin/perl
my $name = shift;
print "Hello $name!!!\n";
```

- Run the program

```
$ ./hello_yourname.pl Toto
```

List and array: @

- A list is a suite of scalar values
- An array is variable which can contain a list
- Declaration

```
my @tab;
```

- A list is represented by the values that it should contain enclosed in parenthesis

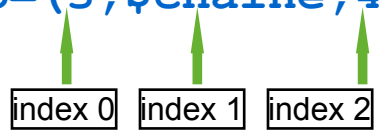
```
(2,'age',$variable)
```

- To assign a list to an array

```
my @tab=(3,$chaine,4);
```

- Access to a value in the table : index concept

```
my @tab=(3,$chaine,4);
```



$tab[0] → 3 $tab[1] → valeur de $chaine $tab[2] → 4

Functions to manipulate tables

```
my @t = (1, 2, 3, 4);
```

- **Unshift**: uses an array and a list and add the list to the top of the table

ex: `unshift(@t, 5, 6);` -> @t is (5,6,1,2,3,4)

- **shift**: removes the first element of the tab and returns it

ex: `$v = shift(@t);` -> $v is 1 and @t is (2,3,4)

- **push**: uses an array and a list and add the list to the end of the table

ex: `push(@t, 5, 6);` -> @t is (1,2,3,4,5,6)

- **Pop**: removes the last element of the tab and returns it

Ex: `$v = pop(@t);` -> $v is 4 and @t is (1,2,3)

- **Reverse**: use an array and returns a reversed list without modifying the tab

Ex: `@s = reverse(@t);` -> @s is (4,3,2,1) and @t is not modified

Hash: %

- A hash is a data type to associate a value with a key. E.g. we will be able to associate phone numbers to person:

```
Paul: 01.23.45.67.89
Virginie: 06.06.06.06.06
Pierre: 06.28.33.55.66
```

- Déclaration :

```
my %phone_hash ;
```

- Pour assigner des valeurs à une table de hachage :

```
my %phone_hash = ( "Paul"     => "01.23.45.67.89",
                   "Virginie" => "06.06.06.06.06",
                   "Pierre"   => "06.28.33.55.66" );
```

- Accéder à un élément

`$phone_hash{"Paul"}` is 01.23.45.67.89

Exercice4: Print hash structure: Data::Dumper

```
$ more printhash_yourname.pl
#!/usr/bin/perl -w
use strict;
use Data::Dumper;

my %hash = ("pcr" => {n1 => "10", n2 => "3"},
        "alex" => {n1 => "03", n2 => "6"},
        "zorro"  => {n1 => "5", n2 => "4"});

print Dumper(\%hash);
```

```
$ ./printhash_stefi.pl
$VAR1 = {
        'zorro' => {
                     'n2' => '4',
                     'n1' => '5'
                   },
        'pcr' => {
                    'n2' => '3',
                    'n1' => '10'
                  },
        'alex' => {
                     'n2' => '6',
                     'n1' => '03'
                   }
      };
```

Give a reference to the Dumper function of the Data::Dumper module

To know more see links in comment

Exercice5: multiple Sort

```
$ more sort.txt
pur   10   3
alex 03   6
zero 5     4
par   04   5
pcr   03   7
pir    01   3
zorro       01   20
```

```
$ sort  sort.txt
alex 03   6
par   04   5
pcr   03   7
pir    01   3
pur   10   3
zero 5     4
zorro       01   20
```

Specify the column number

```
$ sort -k 2 -n sort.txt
pir    01   3
zorro       01   20
alex 03   6
pcr   03   7
par   04   5
zero 5     4
pur   10   3
```

Reverse

```
$ sort -k 2 -n -r sort.txt
pur   10   3
zero 5     4
par   04   5
pcr   03   7
alex 03   6
zorro       01   20
pir    01   3
```

Sort on several fields

```
$ sort -k2n -k1r sort.txt
zorro       01   20
pir    01   3
pcr   03   7
alex 03   6
par   04   5
zero 5     4
pur   10   3
```

Exercice6: multiple Sort with hash

```
$ tail sorthash_stefi.pl
…
print "\nsort by key\n";
foreach my $key (sort (keys(%hash))) {
    print join("\t",$key,$hash{$key}->{n1}, $hash{$key}->{n2}),"\n";
}

print "\nsort by value\n";
foreach my $key(sort {$hash{$a}->{n1} <=> $hash{$b}->{n1} || $hash
{$b}->{n2} <=> $hash{$a}->{n2}} keys(%hash)) {
    print join("\t",$key,$hash{$key}->{n1}, $hash{$key}->{n2}),"\n";
}
```

Returns a list consisting of all the keys of the named hash, or the indices of an array (In scalar context, returns the number of keys or indices).

sort by key

alex	03	6
par	04	5
pcr	03	7
pir	01	3
pur	10	3
zero	5	4
zorro	01	20

sort by value

zorro	01	20
pir	01	3
pcr	03	7
alex	03	6
par	04	5
zero	5	4
pur	10	3

Manipulate files

- Operators :

`open` : open file

`while (<>)` : browse file

`close` : close file

- Syntax ex:

```
open  F, "data.txt";
while (<F>)
    {
     instructions;
    }
close F;
```

- You can open a file in read or write mode

```
open  F, "data.txt";     : read
open  F, ">data.txt";    : overwrite
Open  F, ">>data.txt";   : write by adding at the end of the file
```

Exercice7: read a file

```
#!/usr/bin/perl
my $fichier = "/home/sidibebocs/work/training_20140527/perl/name.txt";
open(F, "<$fichier”) or die "ce fichier n'existe pas\n”;
while (<F>)
{
    chomp;
    my $ligne=$_;
    if ($ligne eq "Xavier")
    {
        print "$ligne Diplome : Master\n";
    }
    elsif ($ligne eq ("Stephanie" || "Manuel"))
    {
        print "$ligne Diplome : These\n";
    }
    else
    {
        print "$ligne Diplome : Inconnu\n";
    }
}
```

Name.txt

Manuel
Xavier
Esteban
Stephanie
Jean Pierre

Exercice8: write an output file

```
$ multiple2oneline.pl
#!/usr/bin/perl
use strict;

my $file = shift;
my $file_out = $file . ".oneline";
open(OUT,">$file_out");
open(GENE,$file);

my %hash;

while(<GENE>){
 chomp;
 my ($gene, $GO) = (split(/\t/,$_));

 push @{$hash{$gene}},$GO;
}
close GENE;

foreach my $gene (keys %hash) {
     print OUT join("\t", $gene, @{$hash{$gene}}), "\n";
}
close OUT;
```

```
$ ./multiple2oneline.pl prot2go_test.txt
$ wc -l prot2go_test.txt*
 1000 prot2go_test.txt
  331 prot2go_test.txt.oneline
```

```
$ more prot2go_test.txt
GSMUA_Achr10T03460_001  GO:0005525
GSMUA_Achr10T03460_001  GO:0005622
GSMUA_Achr10T03460_001  GO:0007264
GSMUA_Achr10T03460_001  GO:0015031
GSMUA_Achr10T03520_001  GO:0005524
```

```
$ more prot2go_test.txt.oneline
GSMUA_Achr10T03460_001  GO:0005525 GO:0005622 GO:0007264 GO:0015031
```

Good programming practices

CIRAD_BIOS_DAP_ID-perl_coding_coventions_sum_up-fr

CIRAD_BIOS_DAP_ID-perl_coding_coventions-fr