Master II Oncology

« High-Throughput Technologies in Oncology »

Introduction To Algorithms with Perl

Jean-Philippe Villemin - Bioinformatics, Software Engineer -

Centre Léon Bérard, Cheney C

28 rue Laennec,

69373 Lyon cedex 08, France

JeanPhilippe.VILLEMIN@lyon.unicancer.fr

- An algorithm is a method for solving a problem, with or without a computer.
- It will execute your list of commandments.
- To organize your commandments, you will use control structures (If...else, for, while...).
- The data traited will be handled inside scalars, arrays and hashes. ($scalar, @array,%hash).

- First Perl Program in Perl : Execute in the console 'Perl Tuto.pl'

TuTo.pl

```
1  #!/usr/bin/perl
2
3  use strict;
4  use warnings;
5  #-------------------------------------------------------------#
6  #                        Introduction To perl                 #
7  #-------------------------------------------------------------#
8
9  print "HELLO, I'M A COMPUTER SPEAKING, DON'T FREAK OUT";
10
```

- Shebang line to let the computer know it's a perl script
- Stuffs to check syntax and good programming methodology

This is just a simple code to print a message in the console.

- We will show how to declare variables and play with them using operators :

  Scalars

  Arrays

  Hashes

- In the next episode, we'll study some control structures in a context of pure algorithm & in real context of progamming with perl.

  Conditionnal Statments

  Loop

- Then, we will see how to pass arguments to your program and to create clean code using subroutines.

- Finally, let's take a look at Perl Regular Expressions to give you some serious headaches…

Perl has three basic data types: scalars, arrays of scalars, and hashes of scalars, also known as associative arrays. Here is little detail about these data types.

S.N.	Types and Description
1	**Scalar:** Scalars are simple variables. They are preceded by a dollar sign ($). A scalar is either a number, a string, or a reference. A reference is actually an address of a variable which we will see in upcoming chapters.
2	**Arrays:** Arrays are ordered lists of scalars that you access with a numeric index which starts with 0. They are preceded by an "at" sign (@).
3	**Hashes:** Hashes are unordered sets of key/value pairs that you access using the keys as subscripts. They are preceded by a percent sign (%).

SCALARS

Here is a simple example of using scalar variables:

```
#!/usr/bin/perl

$age = 25;              # An integer assignment
$name = "John Paul";    # A string
$salary = 1445.50;      # A floating point

print "Age = $age\n";
print "Name = $name\n";
print "Salary = $salary\n";
```

This will produce following result:

```
Age = 25
Name = John Paul
Salary = 1445.5
```

Scalar Operations

You will see a detail of various operators available in Perl in a separate chapter but here I'm going to list down few numeric and string operations.

```
#!/usr/bin/perl

$str = "hello" . "world";       # Concatenates strings.
$num = 5 + 10;                  # adds two numbers.
$mul = 4 * 5;                   # multiplies two numbers.
$mix = $str . $num;             # concatenates string and number.

print "str = $str\n";
print "num = $num\n";
print "mix = $mix\n";
```

This will produce following result:

```
str = helloworld
num = 15
mix = helloworld15
```

SCALARS – Arithmetic Operators

Operator	Description	Example
+	Addition - Adds values on either side of the operator	$a + $b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	$a - $b will give -10
*	Multiplication - Multiplies values on either side of the operator	$a * $b will give 200
/	Division - Divides left hand operand by right hand operand	$b / $a will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	$b % $a will give 0
**	Exponent - Performs exponential (power) calculation on operators	$a**$b will give 10 to the power 20

ARRAYS

PERL

```
my @ages = (25, 30, 40);
my @names = ("John", "Lisa", "Kumar");
print $ages[0] ; #This will produce 25
print $names[0]; #This will produce John
```

An array is a variable that stores an ordered list of scalar values.
Array variables are preceded by an "at" (@) sign. To refer to a single element of an array, you will use the dollar sign ($) with the variable name followed by the index of the element in square brackets.Arrays can contains numeric or string values.

ARRAYS – add or remove values

Adding and Removing Elements in Array

Perl provides a number of useful functions to add and remove elements in an array. You may have a question what is a function? So far you have used **print** function to print various values. Similarly there are various other functions or sometime called sub-routines which can be used for various other functionalities.

S.N.	Types and Description
1	**push @ARRAY, LIST** Pushes the values of the list onto the end of the array.
2	**pop @ARRAY** Pops off and returns the last value of the array.
3	**shift @ARRAY** Shifts the first value of the array off and returns it, shortening the array by 1 and moving everything down
4	**unshift @ARRAY, LIST** Prepends list to the front of the array, and returns the number of elements in the new array.

ARRAYS - examples

```
#!/usr/bin/perl

# create a simple array
@coins = ("Quarter","Dime","Nickel");
print "1. \@coins  = @coins\n";

# add one element at the end of the array
push(@coins, "Penny");
print "2. \@coins  = @coins\n";

# add one element at the beginning of the array
unshift(@coins, "Dollar");
print "3. \@coins  = @coins\n";

# remove one element from the last of the array.
pop(@coins);
print "4. \@coins  = @coins\n";

# remove one element from the beginning of the array.
shift(@coins);
print "5. \@coins  = @coins\n";
```

This will produce following result:

```
1. @coins = Quarter Dime Nickel
2. @coins = Quarter Dime Nickel Penny
3. @coins = Dollar Quarter Dime Nickel Penny
4. @coins = Dollar Quarter Dime Nickel
5. @coins = Quarter Dime Nickel
```

HASHES

PERL

```
my %age =  ('John' => 45, 'Lisa' => 30, 'Kumar' => 40);

print $age{'John'}" #This will produce 45
```

A hash is a set of key/value pairs. Hash variables are preceded by a percent (%) sign.
To refer to a single element of a hash, you will use the hash variable
name preceded by a "$" sign and followed by the "key" associated
with the value in curly brackets.

HASHES

Add & Remove Elements in Hashes

Adding a new key/value pair can be done with one line of code using simple assignment operator. But to remove an element from the hash you need to use **delete** function as shown below in the example:

```
#!/usr/bin/perl

%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);
@keys = keys %data;
$size = @keys;
print "1 - Hash size:  is $size\n";

# adding an element to the hash;
$data{'Ali'} = 55;
@keys = keys %data;
$size = @keys;
print "2 - Hash size:  is $size\n";

# delete the same element from the hash;
delete $data{'Ali'};
@keys = keys %data;
$size = @keys;
print "3 - Hash size:  is $size\n";
```

This will produce following result:

```
1 - Hash size: is 3
2 - Hash size: is 4
3 - Hash size: is 3
```

HASHES

Extracting Keys and Values

You can get a list of all of the keys from a hash by using **keys** function which has the following sy

```
keys %HASH
```

This function returns an array of all the keys of the named hash. Following is the example:

```
#!/usr/bin/perl

%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);

@names = keys %data;

print "$names[0]\n";
print "$names[1]\n";
print "$names[2]\n";
```

This will produce following result:

```
Lisa
John Paul
Kumar
```

Similarly you can use **values** function to get a list of all the values. This function has following syntax:

```
values %HASH
```

This function returns a normal array consisting of all the values of the named hash. Following is the example:

```
#!/usr/bin/perl

%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);

@ages = values %data;

print "$ages[0]\n";
print "$ages[1]\n";
print "$ages[2]\n";
```

This will produce following result:

```
30
45
40
```

HASHES

Checking for Existence

If you try to access a key/value pair from a hash that doesn't exist, you'll normally get the **undefined** value, and if you have warnings switched on, then you'll get a warning generated at run time. You can get around this by using the **exists** function, which returns true if the named key exists, irrespective of what its value might be:

```
#!/usr/bin/perl

%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);

if( exists($data{'Lisa'} ) ){
   print "Lisa is $data{'Lisa'} years old\n";
}
else{
   print "I don't know age of Lisa\n";
}
```

Getting Hash Size

You can get the size - that is, the number of elements from a hash by using scalar context on either keys or values. Simply saying first you have to get an array of either the keys or values and then you can get size of array as follows:

```
#!/usr/bin/perl

%data = ('John Paul' => 45, 'Lisa' => 30, 'Kumar' => 40);

@keys = keys %data;
$size = @keys;
print "1 - Hash size:  is $size\n";

@values = values %data;
$size = @values;
print "2 - Hash size:  is $size\n";
```

- We will show how to declare variables and play with them using operators :

  Scalars

  Arrays

  Hashes

- In the next episode, we'll study some control structures in a context of pure algorithm & in real context of progamming with perl.

  Conditionnal Statments

  Loop

- Then, we will see how to pass arguments to your program and to create clean code using subroutines.

- Finally, let's take a look at Perl Regular Expressions to give you some serious headaches…

CONDITIONAL STATEMENTS

PURE ALGORITHM

```
IF Test
    Instruction 1
END IF
Instruction 2
```

IF

PERL

```
my $variable = 2 ;

If($variable > 2) {
      Print « superior to 2 » ;
}
print « treatment
completed» ;
```

CONDITIONAL STATEMENTS

PURE ALGORITHM

```
IF Test
    Instruction 1
ELSE
   Instruction 2
END IF
Instruction 3
```

IF...ELSE

PERL

```
my $variable = 2 ;

If($variable > 2) {
      print « superior to 2 » ;
}
else {print « something else » ;}

print « treatment completed» ;
```

CONDITIONAL STATEMENTS

PURE ALGORITHM

```
IF Test
   Instruction 1
ELSE IF
   Instruction 2
END IF
Instruction 3
```

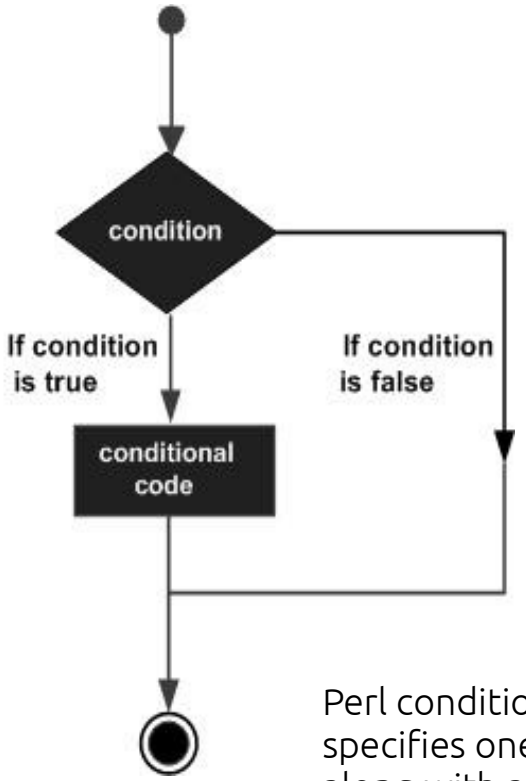
IF…
ELSIF
…

PERL

```
my $variable = 2 ;

If($variable > 2) {
      Print « superior to 2 » ;
}
elsif($variable > 1){
      Print «Superior to » ;
}
Print « treatment
terminated » ;
```

CONDITIONAL STATEMENTS



Statement	Description
if statement	An **if statement** consists of a boolean expression followed by one or more statements.
if...else statement	An **if statement** can be followed by an optional **else statement**.
if...elsif...else statement	An **if statement** can be followed by an optional **elsif statement** and then by an optional **else statement**.
unless statement	An **unless statement** consists of a boolean expression followed by one or more statements.
unless...else statement	An **unless statement** can be followed by an optional **else statement**.
unless...elsif..else statement	An **unless statement** can be followed by an optional **elsif statement** and then by an optional **else statement**.
switch statement	With latest versions of Perl, you can make use of **switch** statment which allows a simple way of comparing a variable value against various conditions.

Perl conditional statements helps in decision making which require the programmer specifies one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

CONDITIONAL STATEMENTS – Logical Operators

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then then condition becomes true.	($a and $b) is false.
&&	C-style Logical AND operator copies a bit to the result if it exists in both operands.	($a && $b) is false.
or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	($a or $b) is true.
||	C-style Logical OR operator copies a bit if it exists in eather operand.	($a || $b) is true.
	Called Logical NOT Operator. Use to reverses the	

You will need this operators to test complex statements as follow :

PERL
my $value = 70 ;
If (($value ≥ 5) and ($value ≤ 10)){ print « Yep » ;}

CONDITIONAL STATEMENTS – Equality Operators

For strings

Operator	Description	Example
lt	Returns true if the left argument is stringwise less than the right argument.	($a lt $b) is true.
gt	Returns true if the left argument is stringwise greater than the right argument.	($a gt $b) is false.
le	Returns true if the left argument is stringwise less than or equal to the right argument.	($a le $b) is true.
ge	Returns true if the left argument is stringwise greater than or equal to the right argument.	($a ge $b) is false.
eq	Returns true if the left argument is stringwise equal to the right argument.	($a eq $b) is false.
ne	Returns true if the left argument is stringwise not equal to the right argument.	($a ne $b) is true.
cmp	Returns -1, 0, or 1 depending on whether the left argument is stringwise less than, equal to, or greater than the right argument.	($a cmp $b) is -1.

For numerics

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	($a == $b) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	($a != $b) is true.
<=>	Checks if the value of two operands are equal or not, and returns -1, 0, or 1 depending on whether the left argument is numerically less than, equal to, or greater than the right argument.	($a <=> $b) returns -1.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	($a > $b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	($a < $b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	($a >= $b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	($a <= $b) is true.

You can use also several ELSE IF Instruction with a final ELSE Instruction

Try to code something with that idea .

THE LOOP - while

PURE ALGORITHM

```
WHILE Conditon is True
   Instruction 1
END WHILE
```

WHILE

PERL

```
my $counter =0 ;
while (my $counter > 4) {

      print $counter;
      $counter ++;
}
Print « done » ;
```

The while loop has a condition, in our case checking if the variable $counter is larger than 4, and then a block of code wrapped in curly braces.
When the execution first reaches the beginning of the while loop it checks if the condition is true or false.
If it is FALSE the block is skipped and the next statement, in our case printing 'done' is executed.
If the condition of the while is TRUE, the block gets executed, and then the execution goes back to the condition again.
It is evaluated again. If it is false the block is skipped and the 'done' is printed.
If it is true the block gets executed and we are back to the condition ...

THE LOOP – foreach & for

FOREACH

```
PERL

my @list = (1,2,3,4) ;

foreach my $value(@liste)
{
      print $value;
}
```

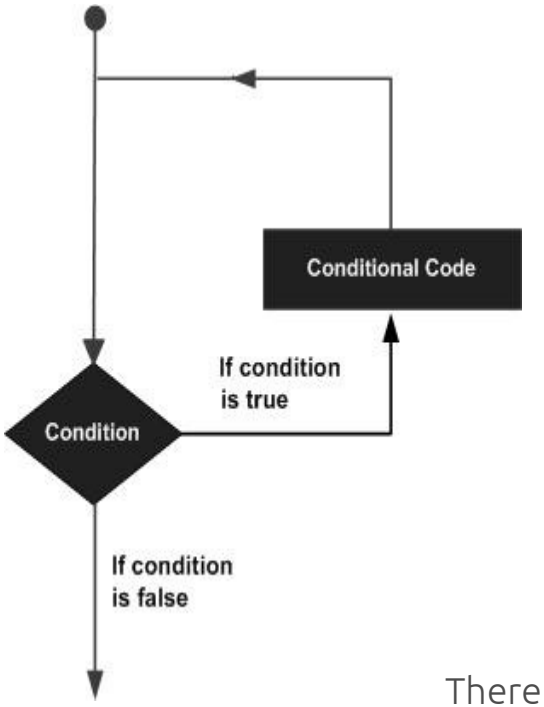
FOR

```
PERL

for (my $i=0; $i<4; $i++)
{
     print "$i\n";
}
```

/n is used to print on a new line.

THE LOOP

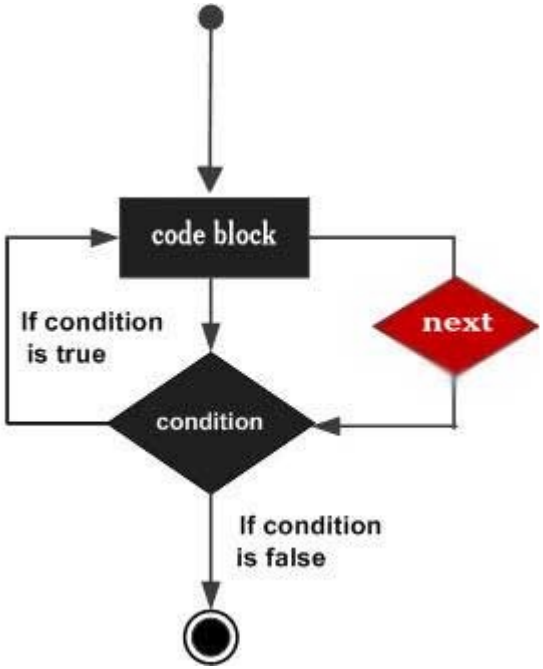


Perl programming language provides following types of loop to handle looping requirements. Click the following links to check their detail.

Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
until loop	Repeats a statement or group of statements until a given condition becomes true. It tests the condition before executing the loop body.
for loop	Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
foreach loop	The foreach loop iterates over a normal list value and sets the variable VAR to be each element of the list in turn.

There may be a situation when you need to execute a block of code several number of times.
In general statements are executed sequentially:
The first statement in a function is executed first, followed by the second, and so on.

THE LOOP - Control Statements



Control Statement	Description
next statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
last statement	Terminates the **loop** statement and transfers execution to the statement immediately following the loop.

The loop will be useful when you will parse files.

- We will show how to declare variables and play with them using operators :

  Scalars

  Arrays

  Hashes

- In the next episode, we'll study some control structures in a context of pure algorithm & in real context of progamming with perl.

  Conditionnal Statments

  Loop

- Then, we will see how to pass arguments to your program and to create clean code using subroutines.

- Finally, let's take a look at Perl Regular Expressions to give you some serious headaches…

PASS ARGUMENTS TO YOUR PROGRAM

In console : perl test.pl Hello

PERL

```
#!/usr/bin/perl

use strict;
use warnings;

my ($message) = @ARGV;

print "$message\n";
```

This will print Hello.

SUBROUTINES - 1

```
#!/usr/bin/perl

# Function definition
sub Hello{
   print "Hello, World!\n";
}

# Function call
Hello();
```

When above program is executed, it produces following result:

```
Hello, World!
```

PERL

```
my @list = (1,2,3) ;

my $listSize = callRoutineForSize(@list) ;
print $listSize ; #This will print 3 ;

sub callRoutineForSize
{
 my (@list) = @_;
 my $size = @list;

return $size ;
};
```

keyword annotation for recovering sent arguments.

Keyword to return a result.

SUBROUTINES - 2

PERL

```
my $value1 = 1 ;
my $value2 = 2 ;

my $sum = callRoutineForSum($value1,$value2) ;
print $sum; #This will print 3 ;

sub callRoutineForSum
{
 my ($value1,$value2) = @_;
 my $sum = $value1+$value2;

return $sum;
};
```

Use subroutines to produce a clean code when you go into large developpment.

- We will show how to declare variables and play with them using operators :

  Scalars

  Arrays

  Hashes

- In the next episode, we'll study some control structures in a context of pure algorithm & in real context of progamming with perl.

  Conditionnal Statments

  Loop

- Then, we will see how to pass arguments to your program and to create clean code using subroutines.

- Finally, let's take a look at Perl Regular Expressions to give you some serious headaches…

Perl Regular Expressions

Useful tool to look for patterns

```
PERL

my @list = (1,2,'3RE','fds4fsd') ;

foreach my $value(@list)
{
      if($value =~ m/^\d+/){

            print «  match okk» ;
      }
}
```

This will print 'match ok' only when the value begins(because of ^) by a number.

```
PERL

my @list = ('Revolver','Revolt','Stuff') ;

foreach my $value(@list)
{
      if($value =~ m/Revo/){

            print « match ok » ;
      }
}
```

This will print 'match ok' only when the value is equal to Revolver or Revol because the pattern 'Revo' is found.

Perl Regular Expressions

Quick (Incomplete) Reference

Metacharacters

These need to be escaped to be matched.

\ . ^ $ * + ? { } [ ] ( ) |

Escape sequences for pre-defined character classes

\d - a digit - [0-9]
\D - a nondigit - [^0-9]
\w - a word character (alphanumeric including underscore) - [a-zA-Z_0-9]
\W - a nonword character - [^a-zA-Z_0-9]
\s - a whitespace character - [\t\n\r\f]
\S - a non-whitespace character - [^ \t\n\r\f]

Assertions

Assertions have zero width.

^ - Matches the beginning of the line
$ - Matches the end of the line
(or before a newline at the end)
\B - Matches everywhere except between a word character and non-word character
\b - Matches between word character and non-word character
\A - Matches only at the beginning of a string

Supplementary help for the tutorial

To read a file. You will need to pass the file name as an argument to the program via the shell.
Then you will handle the lines inside an array.
Finally you read the content of your array and you can do any treatment you want on each line to filter the informations you need.

PERL

```
open(FILE,$path) or die "Can't read to file \n";
open(FILE,"<".$path);
my @lines = <FILE>;
foreach my $line (@lines){
print $line ;
}
close(FILE)  # Don't forget to close !!
```

Use '<' to read, '>' to write in a file.

PERL RULES

There's more than one way to do it

Easy things should be easy and hard things should be possible

Internet will be your best friend to lean more…

https://www.perl.org/books/beginning-perl/
http://perl-begin.org/tutorials/perl-for-newbies/part1/
http://www.tutorialspoint.com/perl/
https://www.youtube.com/watch?v=pXaMLeSS9Eg