An Introduction to Perl for bioinformatics

C. Tranchant-Dubreuil, F. Sabot

UMR DIADE, IRD

June 2015

Sommaire

Objectives

- To demonstrate how Perl can be used in bioinformatics

Objectives

- To demonstrate how Perl can be used in bioinformatics
- To empower you with the basic knowledge required to quickly and effectively create simple scripts to process biological data

Objectives

- To demonstrate how Perl can be used in bioinformatics
- To empower you with the basic knowledge required to quickly and effectively create simple scripts to process biological data
- **Write your own programs once time, use many times**

What is programming

- **Programs** : a set of instructions telling computer what to do

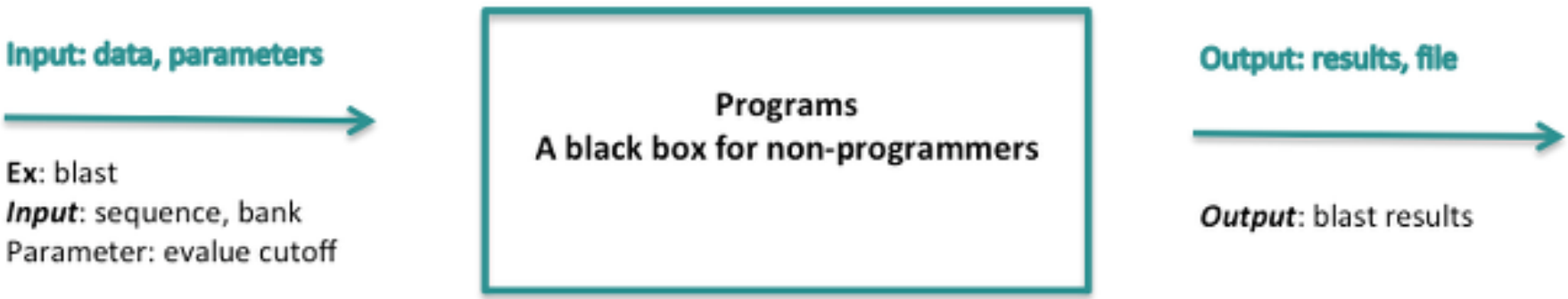
What is programming

- **Programs** : a set of instructions telling computer what to do
- **Programming languages** : bridges between human languages (A-Z) and machine languages (0&1)

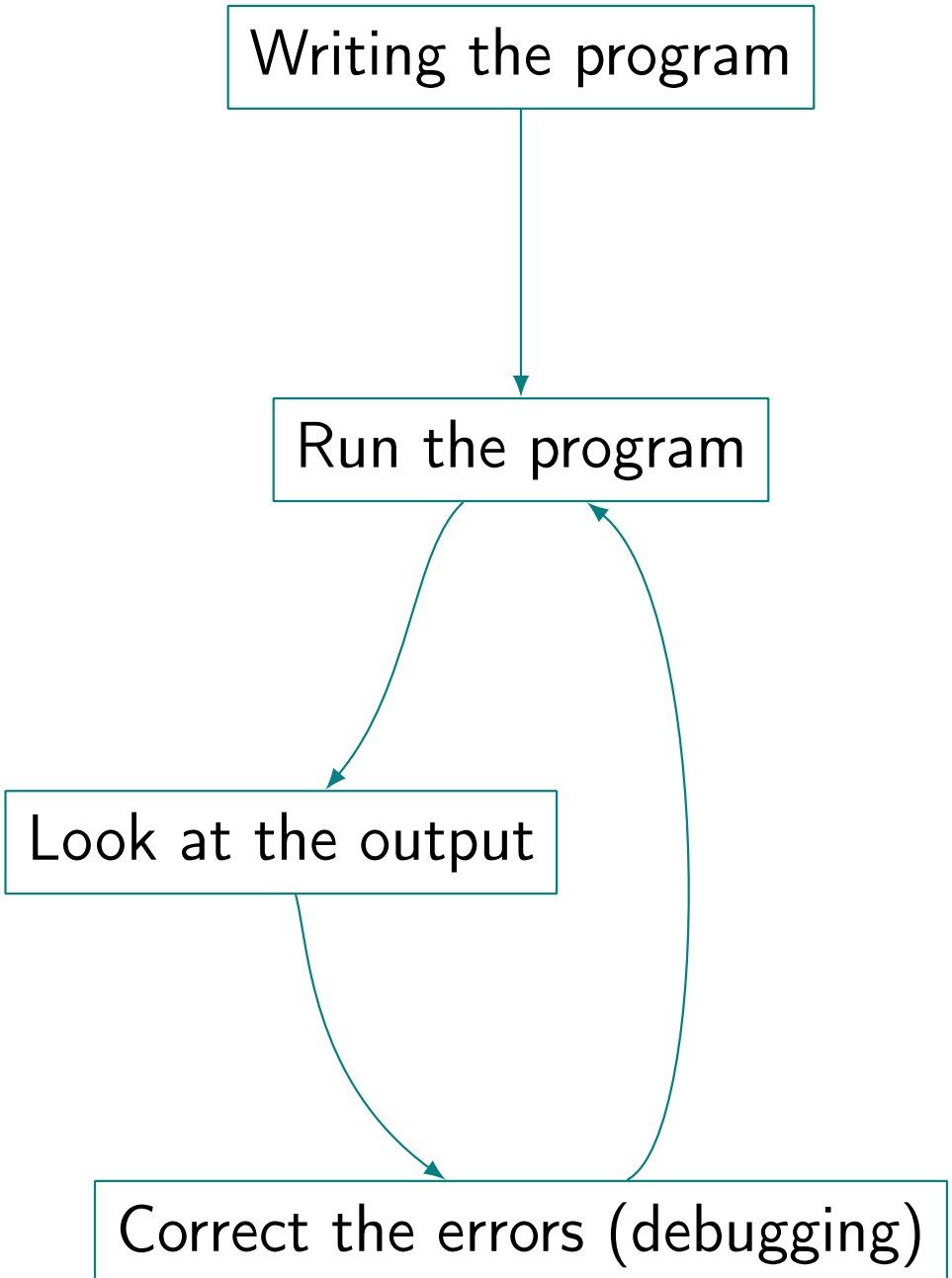
What is programming

- **Programs** : a set of instructions telling computer what to do
- **Programming languages** : bridges between human languages (A-Z) and machine languages (0&1)
- **Compilers** convert programming languages to machine languages

What is a program



Methods for programming



Perl, Practical Extraction and Report Language

- Created in 1987 by Larry Wall

Perl, Practical Extraction and Report Language

- Created in 1987 by Larry Wall
- Optimized for **scanning text files and extracting information** from them

Perl, Practical Extraction and Report Language

- Created in 1987 by Larry Wall
- Optimized for **scanning text files and extracting information** from them
- A massive library of reusable code (`http://www.cpan.org/`)

Perl, Practical Extraction and Report Language

- Created in 1987 by Larry Wall
- Optimized for **scanning text files and extracting information** from them
- A massive library of reusable code (`http://www.cpan.org/`)
- Scripts often **faster/easier to write** than full compiled programs

Perl, Practical Extraction and Report Language

- Created in 1987 by Larry Wall
- Optimized for **scanning text files and extracting information** from them
- A massive library of reusable code (`http://www.cpan.org/`)
- Scripts often **faster/easier to write** than full compiled programs

Syntax sometimes confusing to new users : There's more than one way to do it

Choose a language based on your needs

Perl is NOT suitable for

- significant computation
- sophisticated data structures that use large amounts of memory

Choose a language based on your needs

Perl is NOT suitable for

- significant computation
- sophisticated data structures that use large amounts of memory

Perl is suitable for

- Quick and dirty solutions (prototyping)
- Text processing
- Almost anything if performance is not an issue

Lincoln Stein (1996) www.perl.org

Lincoln Stein (1996) www.perl.org

- Open Source project that has recruited developers from all over the world

Perl in Bioinformatics

How Perl saved the Human Genome Project

Lincoln Stein (1996) www.perl.org

- Open Source project that has recruited developers from all over the world
- A project to produce modules to process all known forms of biological data

Perl in Bioinformatics

How Perl saved the Human Genome Project

Lincoln Stein (1996) www.perl.org

- Open Source project that has recruited developers from all over the world
- A project to produce modules to process all known forms of biological data
- Perl allowed various genome centers to effectively communicate their data with each other

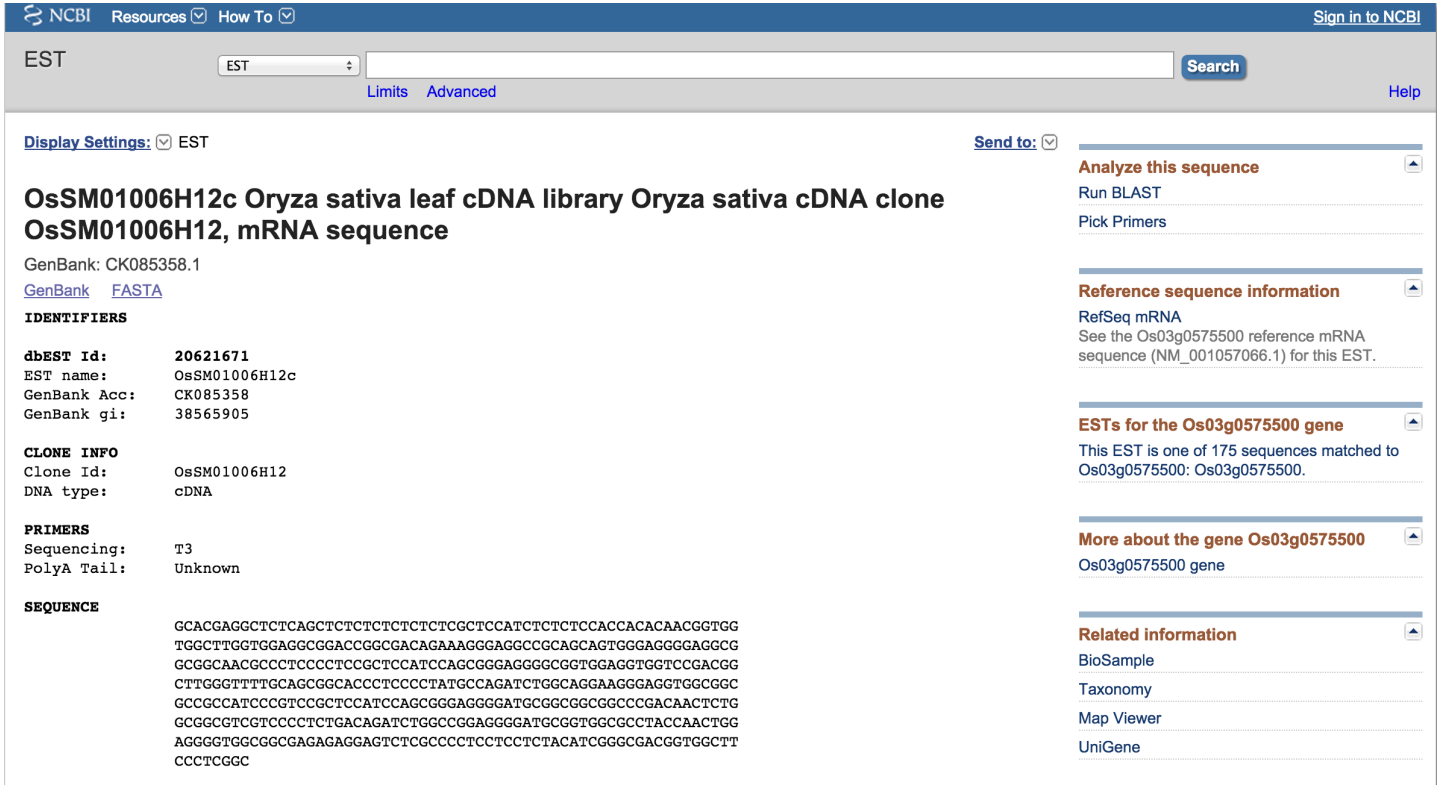
$\Longrightarrow$ **Bioperl project** – `www.bioperl.org`

Bioperl code example

- Retrieve a FASTA sequence from a remote sequence database by accession

Bioperl code example

- Retrieve a FASTA sequence from a remote sequence database by accession


NCBI Resources How To Sign in to NCBI

EST

EST Search

Limits Advanced Help

Display Settings: EST Send to:

OsSM01006H12c Oryza sativa leaf cDNA library Oryza sativa cDNA clone OsSM01006H12, mRNA sequence

GenBank: CK085358.1

GenBank FASTA

IDENTIFIERS

dbEST Id: 20621671
EST name: OsSM01006H12c
GenBank Acc: CK085358
GenBank gi: 38565905

CLONE INFO
Clone Id: OsSM01006H12
DNA type: cDNA

PRIMERS
Sequencing: T3
PolyA Tail: Unknown

SEQUENCE

GCACGAGGCTCTCAGCTCTCTCTCTCTCTCGCTCCATCTCTCTCCACCACACAACGGTGG
TGGCTTGGTGGAGGCGGACCGGCGACAGAAAGGGAGGCCGCAGCAGTGGGAGGGGAGGCG
GCGGCAACGCCCTCCCCTCCGCTCCATCCAGCGGGAGGGGCGGTGGAGGTGGTCCGACGG
CTTGGGTTTTGCAGCGGCACCCTCCCCTATGCCAGATCTGGCAGGAAGGGAGGTGGCGGC
GCCGCCATCCCGTCCGCTCCATCCAGCGGGAGGGGATGCGGCGGCGGCCCGACAACTCTG
GCGGCGTCGTCCCCTCTGACAGATCTGGCCGGAGGGGATGCGGTGGCGCCTACCAACTGG
AGGGGTGGCGGCGAGAGAGGAGTCTCGCCCCTCCTCCTCTACATCGGGCGACGGTGGCTT
CCCTCGGC

Analyze this sequence
Run BLAST
Pick Primers

Reference sequence information
RefSeq mRNA
See the Os03g0575500 reference mRNA sequence (NM_001057066.1) for this EST.

ESTs for the Os03g0575500 gene
This EST is one of 175 sequences matched to Os03g0575500: Os03g0575500.

More about the gene Os03g0575500
Os03g0575500 gene

Related information
BioSample
Taxonomy
Map Viewer
UniGene

First steps

- Perl programs start with : **#!/usr/bin/perl -w**

First steps

- Perl programs start with : **#!/usr/bin/perl -w**
- Perl statements end with a semicolon‘ ;’

First steps

- Perl programs start with : **#!/usr/bin/perl -w**
- Perl statements end with a semicolon' ;'
- # means comment
  - Anything is ignored after a # in a line
  - Comments are free
  - Helps you and others understand your code

4 lines of code

Listing 1 – retrieve-accession.pl

```
1 #!/usr/bin/perl
2
3 #retrieve_accession.pl
4
5 #Load bioperl library used
6 use Bio::DB::GenBank;
7
8 $gb=Bio::DB::GenBank->new();
9 $seq=$gb->get_Seq_by_version('CK085358.1'); #GI ↩
      Number
10
11 #Write the sequence into a file
12 $out=Bio::SeqIO->new('-file' => ">CK085358.1.fa");
13 $out->write_seq($seq);
```

Running Perl program

1. placing the perl path interpreter in the first line of the script

```
1 #!/usr/local/bin/perl
```

Running Perl program

1. placing the perl path interpreter in the first line of the script

```
1 #!/usr/local/bin/perl
```

2. Don't forget make your program executable !

```
chmod +x your_program.pl
```

Running Perl program

1. placing the perl path interpreter in the first line of the script

```
1 #!/usr/local/bin/perl
```

2. Don't forget make your program executable !

   `chmod +x your_program.pl`

3. Run your program !

   `/path2program/your_program.pl`

Practice

Task P1.1

1. Open a terminal
2. Connect on the distant server (bioinfo-inter@ird.fr)
3. Change to the directory /home/formation1/SCRIPTS
4. Run the program retrieve-accession.pl

Practice

Task P1.2

1. Change the accession
2. Run the program retrieve-accession.pl

Sommaire

Plan

Write your first program

The traditional first program

Listing 2 – helloWorld.pl

```
1 # helloWorld.pl
2
3 print "Hello world ... or not\n";
```

print	write to the terminal

Write your first program

The traditional first program

Listing 3 – helloWorld.pl

```
1 # helloWorld.pl
2
3 print "Hello world ... or not\n";
```

print	write to the terminal
\n	signify a newline

Practice

Run your first program

Task P2.1

1. Write your script helloWorld.pl and save it into SCRIPTS directory
2. Run it

Practice

Run your first program

Task P2.2

- Modify the program to output some other text.
- Add a few more print statements and experiment with what happens if you omit or add extra newlines.

Practice

Run your first program

Task P2.3

- Make a few deleterious mutations to your program. For example, leave off the semicolon or ". Observe the error messages.

One of the most important aspects of programming is debugging. Probably more time is spent debugging than programming, so it's a good idea to start recognizing errors now.

What's a variable ?

- Variables are used to store information to be referenced and manipulated in a computer program.
- The value of the variable can change, depending on conditions or on information passed to the program.

Variables

- Three types of variable
- **scalar** :

Variables

- Three types of variable
- **scalar** :
  - The most basic kind of variable
  - Single value, can be string, number
  - prefixed with $ e.g. $dna

Variables

- Three types of variable
- **scalar** :
  - The most basic kind of variable
  - Single value, can be string, number
  - prefixed with $ e.g. $dna
- **array** : list of values, prefixed with @
- **hash** : paired values (key and value) prefixed with %

Assigning values

- = means "assign a value to a variable"

```
1 $dna='ATATACGACGAGACATAG'
```

- No formal declarations of variables necessary

Assigning values

- = means "assign a value to a variable"

```
1 $dna='ATATACGACGAGACATAG'
```

- No formal declarations of variables necessary
  - Good practice to "**use strict**" : forces variable declaration

```
1 #!/usr/local/bin/perl
2
3 use strict;
4
5 my $variable ; # a var declared with my
```

Variable naming

- You can use (almost) anything for your variable names : a-z, A-Z, 0-9,_
- no space
- casse sensitive
- try to use names which are descriptive and not too long.

```
1 $a = "ATCAGGG";    # $a obscur
2 $dna_sequence_variable;    # too long
3 $sequence = "ATCAGGG";  # $sequence is better
4 $dna = "ATCAGGG";  # $dna is even better
```

Scalar : string

Listing 4 – scalar-string.pl

```
1 #!/usr/bin/perl
2
3 #scalar-string.pl
4
5 use strict;
6 use warnings;
7
8 # Variable declaration and initialization
9 my $number= "12";
10
11 my $composite= "There␣are␣$number␣sequences";
12 print $composite, "\n";
```

Scalar : string

Listing 5 – scalar-string.pl

```
1 #!/usr/bin/perl
2
3 #scalar-string.pl
4
5 use strict;
6 use warnings;
7
8 # Variable declaration and initialization
9 my $number= "12";
10
11 my $composite= "There␣are␣$number␣sequences";
12 print $composite, "\n";
```

```
[login@node1]$./scalarstring.pl
There are 12 sequences
```

Manipulate strings

Function	Meaning
substr	takes characters out of a string

Manipulate strings

Function	Meaning
substr	takes characters out of a string

Example

- Getting 5 letters from one position in a string variable

```
1 $letter = substr($dnaSeq, $position, 5);
```

Manipulate strings

Function	Meaning
substr	takes characters out of a string
length	get length of a string

Example

```
1 print length($dnaSeq);
```

Manipulate strings

Function	Meaning
substr	takes characters out of a string
length	get length of a string
.	concatenate two string

Example

```
1 $accession=">myseq\n";
2 $sequence="ATACGTCAGCTAGCTACTGCCCT\n";
3 $mySeq=$accession.$sequence;
```

Practice

Use variables

Task P3.1

- Modify the program retrieve-accession.pl by adding a variable $accession and $fastaFile
- Print on the terminal, once the sequence downloaded, the following generic message : The sequence CK085358.1 has been downloaded and saved in the file CK085358.1.fasta

Practice

Use variables

Task P3.2

- Mutate your program. Delete a $, remove my and see what error message you get.

Task P3.3

- Modify the program by changing the contents of the variables. Observe the output. Try experimenting by creating more variables.

Scalar : number

Listing 6 – scalar-number.pl

```
1 my $x=3;
2 my $y=2;
3
4 print "$x + $y is ", $x + $y, "\n";
5 print "$x - $y is ", $x - $y, "\n";
6 print "$x * $y is ", $x * $y, "\n";
7 print "$x / $y is ", $x / $y, "\n";
```

Scalar : number

- Numbers can be incremented by one with '++'

Scalar : number

- Numbers can be incremented by one with '++'

```
1 $x = 2;
2 $x++;
3 print "$x\n"; # 3
```

Scalar : number

- Numbers can be incremented by one with '++'

```
1 $x = 2;
2 $x++;
3 print "$x\n"; # 3
```

- Numbers can be decremented by one with '–'

Scalar : number

- Numbers can be incremented by one with '++'

```
1 $x = 2;
2 $x++;
3 print "$x\n"; # 3
```

- Numbers can be decremented by one with '–'

```
1 $x = 2;
2 $x--;
3 print "$x\n"; # 1
```

Practice

Use variables scalar and array

Task P4.1

- Write a new program called average.pl that calculates the average of three numeric variables x, y, z ; stores the result into a variable and print the result.

```
1  my $average = ($x + $y) / 2;
```

Array variables

- a named list of information

Array variables

- a named list of information
  - Each array element can be any scalar variable

Array variables

- a named list of information
  - Each array element can be any scalar variable
  - Array is indexed by integers beginning with zero.
    The first element of an array is therefore the zero-th element

```
1 my @gene_list=('CK085358.1','CK085357.1','CK085356↩
      .1');
2 print "$gene_list[1]\n"; # CK085357.1
3 print "$gene_list[0]\n"; # CK085358.1
```

Practice

Use array variables

Task P4.2

- Create and run the following program

Listing 7 – array.pl

```
1 my @animals = ('cat', 'dog', 'pig');
2 print "1st animal in array is: $animals[0]\n";
3 print "2nd animal in array is: $animals[1]\n";
4 print "Entire animals array contains: @animals\↩
    n";
```

Practice

Use array variables

Task P4.3

- Mutate your program as below and see what error message you get.

```
1 print "@animals[0]\n";
```

Array variables

Functions

- Perl arrays are dynamic
- add or remove entries/element from list is easy

Array variables

Functions

- Perl arrays are dynamic
- add or remove entries/element from list is easy

Function	Meaning	Example
push	to add to end	push(@array, "apple")

Array variables

Functions

- Perl arrays are dynamic
- add or remove entries/element from list is easy

Function	Meaning	Example
push	to add to end	push(@array, "apple")
pop	to remove from end	$pop_val = pop(@array)

Array variables

Functions

- Perl arrays are dynamic
- add or remove entries/element from list is easy

Function	Meaning	Example
push	to add to end	push(@array, "apple")
pop	to remove from end	$pop_val = pop(@array)
unshift	add to front	unshift(@array, "some value")

Array variables

Functions

- Perl arrays are dynamic
- add or remove entries/element from list is easy

Function	Meaning	Example
push	to add to end	push(@array, "apple")
pop	to remove from end	$pop_val = pop(@array)
unshift	add to front	unshift(@array, "some value")
shift	remove from front	$shift_val = shift(@array)

Array variables

Functions

- example with push method

Listing 8 – array.pl

```
1
2 push @animals, "fox"; # the array is now longer
3 my $length = @animals;
4 print "The array now contains $length elements\↩
      n";
5 print "Entire animals array contains: @animals\↩
      n";
```

Practice

Use array variables

Task P5.1

- Experiment with the array functions by adding some new lines to array.pl.
- Rather than just adding a text string to an array, try to see if you can use the push() or unshift() functions to add variables.
- For the shift() and pop() functions, try to see what happens if you don't assign the popped or shifted value to a variable.

```
1 my $value = pop(@array);
2 pop(@array);
```

Array variables

From strings to arrays and back

Function	Meaning
join	to create a string from an array

Array variables

From strings to arrays and back

Function	Meaning
join	to create a string from an array

Listing 10 – string-array.pl

```
1  #!/usr/bin/perl
2
3  # string-array.pl
4
5  use strict;
6  use warnings;
7
8  my @gene_names = ("unc-10","cyc-1","act-1","let-7",↩
       "dyf-2");
9  my $joined_string = join(", ", @gene_names);
10 print "$joined_string\n";
```

Array variables

From strings to arrays and back

Function	Meaning
join	to create a string from an array
split	to divides a string into an array

Array variables

From strings to arrays and back

Function	Meaning
join	to create a string from an array
split	to divides a string into an array

Array variables

From strings to arrays and back

Function	Meaning
join	to create a string from an array
split	to divides a string into an array

```
1 my $line="....." ;
2 my $separator="\t";
3
4 my @col = split($separator,$line);
```

Practice

Use split function

Task P6.1

- Running the program string-array.pl
- Add the following lines to this program and use the split function to digest the DNA sequence by the EcoRI enzyme. Print the result of the digestion.

```
my $dna = "aaaaGAATTCttttttGAATTCgggggggg";
my $EcoRI = "GAATTC";
```

Array variables

Sorting

Task P7.1

- Create the following program and run it

Listing 11 – sorting-array.pl

```
1 #!/usr/bin/perl
2
3 # sorting-array.pl
4 use strict;
5 use warnings;
6
7 my@list=("c","b","a","C","B","A","a","b","c"↩
      ,3,2,1);#an unsorted list
8 my @sorted_list = sort @list;
9 print "default␣sorting:␣@sorted_list\n";
```

Array variables

Sorting

Task 7.2

- Change the last program with an array of number and run it
- Try sorting with the numeric comparison operator <=> and print the array sorted

```
1 @sorted_list = sort {$a <=> $b} @list;
```

Array variables

Sorting

Task P7.3

- Change the last program with an array of number and run it
- Try sorting in reverse direction

```
1 @sorted_list = sort {$b <=> $a} @list;
```

Array variables

The famous array @ARGV

- When you execute a perl script, you can pass any arguments.

```
./detect_SNP.pl bam_directory vcf_file_out
```

Array variables

The famous array @ARGV

- When you execute a perl script, you can pass any arguments.

  ```
  ./detect_SNP.pl bam_directory vcf_file_out
  ```

  How know which values were passed ?

Array variables

The famous array @ARGV

- When you execute a perl script, you can pass any arguments.

  `./detect_SNP.pl bam_directory vcf_file_out`

  How know which values were passed ?
- By using the array **@ARGV** created automatically by perl

Array variables

The famous array @ARGV

- When you execute a perl script, you can pass any arguments.

  ```
  ./detect_SNP.pl bam_directory vcf_file_out
  ```

  How know which values were passed ?
- By using the array **@ARGV** created automatically by perl
- @ARGV holds all the values from the command line.

Array variables

The famous array @ARGV

- When you execute a perl script, you can pass any arguments.

  ```
  ./detect_SNP.pl bam_directory vcf_file_out
  ```

  How know which values were passed ?
- By using the array **@ARGV** created automatically by perl
- @ARGV holds all the values from the command line.
  - If there are no parameters, the array will be empty.

Array variables

The famous array @ARGV

- When you execute a perl script, you can pass any arguments.

  ```
  ./detect_SNP.pl bam_directory vcf_file_out
  ```

  How know which values were passed ?
- By using the array **@ARGV** created automatically by perl
- @ARGV holds all the values from the command line.
  - If there are no parameters, the array will be empty.
  - If there is one parameter passed, that value will be the only element in @ARGV

Practice

Use variables scalar and array

Task P8.1

- In the program called average.pl, the average of three numeric variables x, y, z is calculated and the result is printed on the screen.
- Instead of assigning the variables inside the code, three numeric variables will be given as argument by the user when the script will be executed. Save the script as average-arg.pl.

What's a control structure ?

- block of programming that analyzes variables and chooses a direction in which to go based on given parameters.
- Different types
  - IF /ELSE : a simple control that tests whether a condition is true or false

What's a control structure ?

- block of programming that analyzes variables and chooses a direction in which to go based on given parameters.
- Different types
  - IF /ELSE : a simple control that tests whether a condition is true or false
  - Loop structure : repeats a bunch of function until it is done.
    - WHILE, FOR, FOREACH

IF - ELSE

- control expression that
  - IF the condition is true, one statement block is executed,
  - ELSE a different statement block is exected (ifelse.pl).

IF - ELSE

- control expression that
  - IF the condition is true, one statement block is executed,
  - ELSE a different statement block is exected (ifelse.pl).

```
1  if (control_expression is TRUE)
2  {
3        do this;
4        and this;
5  }
6  else
7  {
8        do that;
9        and that;
10 }
```

IF - ELSE

- If you want to test mutltiple statements you can combine else and if to make 'elsif'

IF - ELSE

- If you want to test mutltiple statements you can combine else and if to make 'elsif'

```
1 if (condition1 is TRUE)
2 {
3       do this;
4 }
5 elsif  (condition2 is TRUE)
6 {
7       do that;
8 }
9 else { do whatever; } #all tests are failed
```

Numerical comparison operator

Operator	Meaning	Example
==	equal to	if (x ==y)
!=	not equal to	if (x ! = y)
>	greater than	if (x >y)
<	less than	if (x <y)
>=	greater than or equal to	if (x >=y)
<=	less than or equal to	if (x <=y)

Numerical comparison operator

Operator	Meaning	Example
==	equal to	if (x ==y)
!=	not equal to	if (x ! = y)
>	greater than	if (x >y)
<	less than	if (x <y)
>=	greater than or equal to	if (x >=y)
<=	less than or equal to	if (x <=y)

- Be careful ! ! !
  - equality is tested with two equals signs !
  - = used to assign a variable

String operator

Operator	Meaning	Example
eq	equal to	if ($x eq $y)
ne	not equal to	if ($x ne $y)
.	concatenation	$z = $x . $y

Multiple comparisons

Logical/boolean operators

AND

- something is overall true only if both of the conditions are true.

Multiple comparisons

Logical/boolean operators

AND

- something is overall true only if both of the conditions are true.

Example

- I'll go if Peter goes AND Paul goes.
- "Peter goes" and "Paul goes" must both be true before I'll go

Multiple comparisons

Logical/boolean operators

AND

- something is overall true only if both of the conditions are true.

Example

- I'll go if Peter goes AND Paul goes.
- "Peter goes" and "Paul goes" must both be true before I'll go

```
1 if (condition 1 AND condition2)
2 {
3    ...
4 }
```

Multiple comparisons

Logical/boolean operators

OR

- something is overall true if there exists any condition that is true

Multiple comparisons

Logical/boolean operators

OR

- something is overall true if there exists any condition that is true

Example

- I'll go if Peter goes OR Paul goes.

Multiple comparisons

Logical/boolean operators

OR

- something is overall true if there exists any condition that is true

Example

- I'll go if Peter goes OR Paul goes.

```
1 if (condition1 OR condition2)
2 {
3    ...
4 }
```

Practice

Use if / elsif / else

Task P9.1

- Perform the script average-arg.pl : Print a different message according if the average is lower than 10, or comprised between 10 and 14 or greater than 14.

String matching

- One of the most useful features of Perl and common task

String matching

- One of the most useful features of Perl and common task
- to find a pattern within another string.

String matching

- One of the most useful features of Perl and common task
- to find a pattern within another string.
- characteristics
  - contained in slashes,
  - matching occurs with the =~operator

String matching

- One of the most useful features of Perl and common task
- to find a pattern within another string.
- characteristics
  - contained in slashes,
  - matching occurs with the =~operator

Listing 15 – matching.pl

```
1 if ($sequence =~ m/GAATTC/)
2 {
3     print "EcoRI site found\n";
4 }
5 else
6 {
7     print "no EcoRI site found\n";
8 }
```

Practice

Use matching operators

Task P10.1

- Add the following lines and observe what happens when you use the substitution operator.

```
1 $sequence =~ s/GAATTC/gaaTtc/;
2 print "$sequence\n";
```

- Add the following lines and find out what happens to $sequence

```
1 $sequence =~ s/A/adenine/;
2 print "$sequence\n";
3 $sequence =~ s/C//;
4 print "$sequence\n";
```

Practice

Use matching operators

Task P10.2

- With the last program, we have only replaced the first occurrence of the matching pattern.
- If you wanted to replace all occurrences, add a letter 'g' to the end of the regular expression (global option)

```
1 $sequence =~ s/C//g;
```

Practice

Use matching operators

Task P10.3

- Add the following lines to the script and try to work out what happens when you add an 'i' to the to matching operator

```
my $protein = "
    MVGGKKKTKICDKVSHEEDRISQLPEPLISEILFHLSTKDLWQSVPGL
    ";
print "Protein contains proline\n" if ($protein
    =~ m/p/i);
```

Practice

Use matching operators

Task P10.4

- The substitution operator can also be used to count how many changes are made. Add the following lines to your script.

```
1 my $sequence = "AACTAGCGGAATTCCGACCGT";
2 my $g_count = ($sequence =~ s/G/G/);
3 print "The␣letter␣G␣occurs␣$g_count␣times␣in␣↩
      $sequence\n";
```

Practice

Use matching operators

Task P10.4

Explanation

- Perl performs the code inside the parentheses first and this performs the substitution.
- The result is that lots of G->G substitutions are made which leaves $sequence unchanged.
- The substitution operator counts how many changes are made and the count is assigned to a variable (as in this example).

Matching operator

Operator	Meaning	Example
=~ m//	match	if ($seq =~ m/GAATTC/)

Matching operator

Operator	Meaning	Example
=~ m//	match	if ($seq =~ m/GAATTC/)
!~ m//	no match	if ($seq !~ m/GAATTC/)

Matching operator

Operator	Meaning	Example
=~ m//	match	if ($seq =~ m/GAATTC/)
!~ m//	no match	if ($seq !~ m/GAATTC/)
=~ s//	substitution	$seq =~ s/thing/other/

Project 1 : DNA composition

- At this point, we know enough Perl to write our first useful program.

How to write your program

- Think in terms of input, process and output
- Identify what each of these are. For example, input might be "three numbers given as argument", variable might be "average", process is "calculate average of x, y, z" and output is "return the average".
- Figure out how to turn the algorithm into code. Express "instructions in words/equations" as "operations in code".

Project 1 : DNA composition

- Your program will read a sequence stored in a variable and report the following :
  - The length of the sequence
  - The total number of A, C, G and T nucleotides
  - The fraction of A,C, G, and T nucleotides
  - The GC fraction

Loops

For

- generally iterates over integers, usually from zero to some other number.

Loops

For

- generally iterates over integers, usually from zero to some other number.
- 3 steps :
  - **initialization** : provide some starting value for the loop counter
    An initial expression is set up ($count=0)

Loops

For

- generally iterates over integers, usually from zero to some other number.
- 3 steps :
  - **initialization** : provide some starting value for the loop counter
    An initial expression is set up ($count=0)
  - **validation** : provide a condition for when the loop should end.
    This test expression is tested at each iteration ($count < 10).

Loops

For

- generally iterates over integers, usually from zero to some other number.
- 3 steps :
  - **initialization** : provide some starting value for the loop counter An initial expression is set up ($count=0)
  - **validation** : provide a condition for when the loop should end. This test expression is tested at each iteration ($count < 10).
  - **update** : how should the loop counter be changed in each loop cycle. ($count ++).

Loops

For

- generally iterates over integers, usually from zero to some other number.
- 3 steps :
  - **initialization** : provide some starting value for the loop counter An initial expression is set up ($count=0)
  - **validation** : provide a condition for when the loop should end. This test expression is tested at each iteration ($count < 10).
  - **update** : how should the loop counter be changed in each loop cycle. ($count ++).

```
1 for ($init_val=0; $init_val < 10; $init_val++)
2 {
3       do this;
4       do that;
5 }
```

Practice

Use variables scalar and array

Task P10.1

- Run the loops-for.pl program
- Try looping backwards from 50 to 40
- Try skipping 10 by 10, from 0 to 100

Practice

Use variables scalar and array

Task P10.2

- Write a program calculateSum.pl that computes the sum of integers from 1 to n, with n is a number given as argument.

Practice

Use variables scalar and array

Task P10.3

- Update the script retrieve-accession.pl :
  - several sequence accessions will be given by the user as arguments
  - A for loop will be used to parse the array that contains the accession and to get fasta sequence

Practice

Use variables scalar and array

Task P10.4

- One of the most common operations you will do as a programmer is to loop over arrays. To make it interesting, we will loop over two arrays simultaneously :

```
1 my @animals = ("cat","dog","cow");
2 my @sounds = ("Meow","Woof","Moo");
3 for (my $i = 0; $i < @animals; $i++)
4 {
5       print "$i) $animals[$i] $sounds[$i]\n";
6 }
```

- Write and run this program

While

- to do a series of actions while some condition is true

```
1 while (expression is true)
2 {
3         do this;
4         do that;
5         ...
6 }
```

Perl input/output

How your program talk to the rest of the "world"

- **input** : getting information into your program

Perl input/output

How your program talk to the rest of the "world"

- **input** : getting information into your program
- **output** : getting informations out of your program
  - the terminal by default

Perl input/output

How your program talk to the rest of the "world"

- **input** : getting information into your program
- **output** : getting informations out of your program
  - the terminal by default

Example

- open a file and read its contents
- write your results to a file

Reading files

Opening a file

open() function : open a file for reading or writing

Reading files

Opening a file

open() function : open a file for reading or writing

```
1 open(IN, MODE, $fileName);
```

- Three arguments
  - The name of a filehandle : a special variable used to refer to a file once it has been opened
  - Open mode
  - The name of the file to open

Reading files

Opening a file

open() function : open a file for reading or writing

- To read from a file

```
1 open(IN, "<" , $fileName);
```

Reading files

Opening a file

open() function : open a file for reading or writing

- To read from a file

```
1 open(IN, "<" , $fileName);
```

- To create a file an write to a file

```
1 open(IN, ">", $fileName");
```

Reading files

Opening a file

open() function : open a file for reading or writing

- To read from a file

```
1 open(IN, "<" , $fileName);
```

- To create a file an write to a file

```
1 open(IN, ">", $fileName");
```

- To append to a file

```
1 open(IN, ">>", $fileName");
```

die function

- open returns 0 on failure and a non-zero value on success

die function

- open returns 0 on failure and a non-zero value on success
- die kills your script safely and prints a message
- often used to prevent doing something regrettable - e.g. running your script on a file that doesn't exist or overwriting an existing file.

```
1 open(IN, "<", $fileName) or die ("Can't␣open␣the␣↩
      File␣$fileName") ;
```

Reading files

Closing a file

close() function : close file handles

Reading files

Closing a file

close() function : close file handles

```
1 close(IN);
```

Reading files

Reading one line

- In a scalar context, the input operator <> reads one line from the specified file handle

```
1 $line = <IN>
```

- Note that, in the above examples, $line still contains the trailing newline character ; it can be removed with the chomp function

```
1 chomp $line;
```

Writing files

- While is often used to reads one line at a time from one file

Listing 16 – readFasta.pl

```
1  open(IN, "<", $ARGV[0]) or die "error reading $ARGV[0] for reading";
2
3  while (my $line=<IN>)
4  {
5      chomp $line;
6      if ($line =~ /^>/)
7      {
8          print "Sequence accession : $line";
9      }
10 }
11 close IN;
```

Practice

Use variables scalar and array

Task P11.1

- Update the script retrieve-accession.pl :
  - a filename will be given by the user as arguments
  - this file contains a list of accessions
  - A while loop will be used to read the file

Loops

Takes a list of values and assigns them to a scalar variable, which executes a block of code

```
1 foreach $element (@list)
2 {
3         do this;
4         do that; #until no more $element's
5 }
```

Hash

First steps

- similar to the array
- instead of indexing with integers, the hash is indexed with text

Hash

First steps

- similar to the array
- instead of indexing with integers, the hash is indexed with text

Listing 18 – hash.pl

```
1 my %genetic_code = ('ATG' => 'Met',
2                     'AAA' => 'Lys',
3                     'CCA' => 'Pro');
4
5 print "$genetic_code{'ATG'}\n";
```

Hash

Keys and values

- iterate over the keys with foreach loop and keys() function
- keys() : returns an array of keys

Hash

Keys and values

- iterate over the keys with foreach loop and keys() function
- keys() : returns an array of keys

Listing 20 – hash.pl

```
1 foreach my $key (keys %genetic_code)
2 {
3     print "$key␣$genetic_code{$key}\n";
4 }
```

Practice

Use hash variable

Task P12.1

- Write the following program

Listing 21 – hash.pl

```
1  my %genetic_code = ('ATG' => 'Met',
2                      'AAA' => 'Lys',
3                      'CCA' => 'Pro');
4
5  print "$genetic_code{'ATG'}\n";
6
7  foreach my $key (keys %genetic_code)
8  {
9      print "$key $genetic_code{$key}\n";
10 }
```

Practice

Use values() function

Task P12.2

- use the **values()** function that returns an array of values
- Add the following lines to your last program

```
1 my @vals = values(%genetic_code);
2 print "values: @vals\n";
```

Hash

Adding, Removing and Testing

- To add one value :

```
1 $genetic_code{CCG} = 'Pro';
2 $genetic_code{AAA} = 'Lysine';
```

- To remove both a key and its associated value from a hash :

```
1 delete $genetic_code{AAA};
```

Hash

Adding, Removing and Testing

- Sometimes, it's necessary to test if a particular key already exists in a hash, for example, before overwriting something.
- **exists()** function

```
1 if (exists $genetic_code{AAA})
2 {
3       print "AAA codon has a value : $genetic_code{AAA}\n";
4 }
5 else
6 {
7       print "No value set for AAA codon\n";
8 }
```

Practice

Use values() function

Task P12.3

- Write instructions adding a new key and value to %genetic_code
- Add lines removing a key
- Add lines testing if a particular key already exists in a hash

Hash variables

Summary of hash-related functions

Function	Meaning
keys %hash	return an array of keys
values %hash	return an array of values
exists $hashkey	returns true if the key exists
delete $hashkey	removes the key and value from the hash

EXERCICE

Project 2 : Counting codon

- Your program will read a nucleic sequence given by user as argument and report the codon usage for the sequence
- How to write your program ?
  - what input ?
  - what output ?
  - what variables used ? scalar, hash
  - List of main operations
  - List of instructions

Interacting with other program

2 ways

- The simplest one is the backticks operator ``, the output will be returned to you in an array or scalar
- The second one is with the **system()** function

Interacting with other program

backticks operator

Listing 22 – system.pl

```
1 my @files = `ls` or die "Problem with ls command";
2 print "@files\n";
3 my $file_count = `ls | wc`;
4 print "$file_count\n";
```

Practice

Use variables scalar and array

Task P13.1

- Write and run the script system.pl
- Add the directory name to list as argument given by the user

Interacting with other program

system

```
1 system("ls ") == 0 or die "Command failed\n";
```

Practice

Use variables scalar and array

Task P13.2

- In the script system.pl, run a blastx analysis on the fasta files (against the bank bank.fasta)

Module

Keep calm ! It's your turn !