

Docente: **Matteo Re**

UNIVERSITÀ DEGLI  
STUDI DI MILANO



C.d.I. Informatica

# Bioinformatica

A.A. 2013-2014 semestre I

**p1**

## Introduzione

---

- **Programmazione PERL**
  - Soluzione problemi e Debugging
  - Lettura e scrittura di documentazione
  - Manipolazione dei dati: filtro e trasformazione
  - Pattern matching e data mining (esempi)
  - SQL
- **Biologia computazionale**
  - Analisi e manipolazione di biosequenze
  - Interazione con banche dati biologiche (NCBI, EnsEMBL, UCSC)
  - BioPERL

# Obiettivi

---





# Linee guida

- **Sistema operativo**
  - In aula di calcolo utilizzeremo windows

- **Installazione PERL**

**WIN:** <http://www.activestate.com/activeperl/downloads>  
**UNIX, MacOS:** normalmente già disponibile

- **Editor di testo**

- I programmi PERL vengono salvati come file di testo. Ci sono molte opzioni disponibili:
  - <http://www.vim.org> (UNIX like OS)
  - Notepad (Windows)
- Se siete interessati ad un **IDE**, questi sono adatti al lavoro che svolgeremo e sono free:
  - [www.eclipse.org](http://www.eclipse.org)
  - [www.activestate.com/komodo-ide](http://www.activestate.com/komodo-ide)



# File sequenze – Formato **FASTA**

```
>gi|40457238|HIV-1 isolate 97KE128 from Kenya gag gene, partial cds
CTTTTGAATGCATGGGTAAAAGTAATAGAAGAAAGAGGTTTCAGTCCAGAAGTAATACCCATGTTCTCAG
CATTATCAGAAGGAGCCACCCACAAGATTTAAATACGATGCTGAACATAGTGGGGGGACACCAGGCAGC
TATGCAAATGCTAAAGGATACCATCAATGAGGAAGCTGCAGAATGGGACAGGTTACATCCAGTACATGCA
GGGCCTATTCCGCCAGGCCAGATGAGAGAACCAAGGGGAAGTGACATAGCAGGAACTACTAGTACCCCTC
AAGAACAAGTAGGATGGATGACAAACAATCCACCTATCCCAGTGGGAGACATCTATAAAAGATGGATCAT
CCTGGGCTTAAATAAAATAGTAAGAATGTATAGCCCTGTTAGCATTTTGGACATAAAACAAGGGCCAAA
GAACCCTTTAGAGACTATGTAGATAGGTTCTTTAAACTCTCAGAGCCGAACAAGCTT
```

```
>gi|40457236| HIV-1 isolate 97KE127 from Kenya gag gene, partial cds
TTGAATGCATGGGTGAAAGTAATAGAAGAAAAGGCTTTCAGCCAGAAGTAATACCCATGTTCTCAGCAT
TATCAGAAGGAGCCACCCACAAGATTTAAATATGATGCTGAATATAGTGGGGGGACACCAGGCAGCTAT
GCAAATGTTAAAAGATAACCATCAATGAGGAAGCTGCAGAATGGGACAGGTTACATCCAATACATGCAGGG
CCTATTCCACCAGGCCAAATGAGAGAACCAAGGGGAAGTGACATAGCAGGAACTACTAGTACCCCTCAAG
AGCAAATAGGATGGATGACAAGCAACCCACCTATCCCAGTGGGAGACATCTATAAAAGATGGATAATCCT
GGGATTAAATAAAATAGTAAGAATGTATAGCCCTGTTAGCATTTTGGACATAAAACAAGGGCCAAAAGAA
CCTTTCAGAGACTATGTAGATAGGTTTTTTAAACTCTCAGAGCCGAACAAGCTT
```

```
>gi|40457234| HIV-1 isolate 97KE126 from Kenya gag gene, partial cds
CCTTTGAATGCATGGGTGAAAGTAATAGAAGAAAAGGCTTTCAGCCAGAAGTAATACCCATGTTTTTCAG
CATTATCAGAAGGAGCCACCCACAAGATTTAAATATGATGCTGAACATAGTGGGGGGGCACCAGGCAGC
TATGCAAATGTTAAAAGATAACCATCAATGAGGAAGCTGCAGAATGGGACAGGCTACATCCAGCACAGGCA
GGGCCTATTGCACCAGGCCAGATAAGAGAACCAAGGGGAAGTGATATAGCAGGAACTACTAGTACCCCTC
AAGAACAATAGCATGGATGACAGGCAACCCGCCTATCCCAGTGGGAGACATCTATAAAAGATGGATAAT
CCTGGGATTAAATAAAATAGTAAGAATGTATAGCCCTGTTAGCATTTTGGATATAAAACAAGGGCCAAA
GAACCATTCAGAGACTATGTAGACAGGTTCTTTAAACTCTCAGAGCCGAACAAGCTT
```



# GenBank Record

LOCUS AK091721 2234 bp mRNA linear PRI 20-JAN-2006

DEFINITION Homo sapiens cDNA FLJ34402 fis, clone HCHON2001505.

ACCESSION AK091721

VERSION AK091721.1 GI:21750158

KEYWORDS oligo capping; fis (full insert sequence).

SOURCE Homo sapiens (human)

ORGANISM Homo sapiens

Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi;  
Mammalia; Eutheria; Euarchontoglires; Primates; Catarrhini;  
Hominidae; Homo.

TITLE Complete sequencing and characterization of 21,243 full-length  
human cDNAs

JOURNAL Nat. Genet. 36 (1), 40-45 (2004)

FEATURES Location/Qualifiers

source 1..2234  
/organism="Homo sapiens"  
/mol\_type="mRNA"

CDS 529..1995  
/note="unnamed protein product"  
/codon\_start=1  
/protein\_id="BAC03731.1"  
/db\_xref="GI:21750159"  
/translation="MVAERSPARSPGSWLFPGWLWLLVLSGPGGLLRAQEQPSCRRAFD  
...  
RLDALWALLRRQYDRVSLMRPQEGDEGRCINFSRVPSQ"

ORIGIN

1 gttttcggag tgcggagga gttggggccg ccggaggaga agagtctcca ctctagttt  
61 gttctgccgt cgccgcgtcc cagggacccc ttgtcccgaa ggcacggca gcggggggaa

...



# Perchè Perl?

- **Ampiamente utilizzato in biologia computazionale**
  - Bioperl
    - [http://www.bioperl.org/wiki/Main\\_Page](http://www.bioperl.org/wiki/Main_Page)
- **Relativamente semplice da programmare**
  - Ottimo per problemi di pattern matching
  - Semplice creazione di pipelines (controllo di altri programmi)
  - Facile da apprendere (relativamente)
- **Rapida prototipizzazione**
  - Molti problemi possono essere risolti con poche righe di codice
- **Portabilità**
  - Disponibile su *Unix, Windows, Macs*
- **Mentalità Open source radicata da tempo**
  - Molte fonti di informazioni ( provare: `%perldoc perldoc`)
    - `%perldoc -f print`
  - <http://perldoc.perl.org/index-tutorials.html>
  - Molti moduli disponibili ( <http://www.cpan.org/> )

# L'interprete PERL

PERL è un linguaggio interpretato.

L'interprete PERL si occupa di **tradurre** una istruzione scritta in un linguaggio comprensibile per un essere umano in una corrispondente istruzione scritta in un linguaggio comprensibile ad un calcolatore.

Esegue l'istruzione appena tradotta

Continua fino a quando non rimane nessuna istruzione da tradurre

Esiste comunque la possibilità di compilare il codice in un unico file costituito interamente da istruzioni comprensibili al calcolatore generando un **eseguibile** (maggiore velocità di esecuzione).

---



Docente: **Matteo Re**

UNIVERSITÀ DEGLI  
STUDI DI MILANO



C.d.I. Informatica

# Bioinformatica

A.A. 2013-2014 semestre I

**p2**

**BASI DI PERL**

---



I **tipi di variabile** PERL sono indicati dall **simbolo iniziale** presente nel loro nome:

`$var` contiene uno scalare (singola stringa o numero)

```
$x = 10;  
$s = "ATTGCGT";  
$x = 3.1417;
```

`@var` contiene un array (lista di valori)

```
@a = (10, 20, 30);  
@a = (100, $x, "Jones", $s);  
print "@a\n"; # prints "100 3.1417 Jones ATTGCGT"
```

`%var` contiene un hash (array associativo)

```
%ages = { John => 30, Mary => 22, Lakshmi => 27 };  
print $age{"Mary"}, "\n"; # prints 22
```

# Variabili

---

## use strict;

- Inserire **use strict** all'inizio di un sorgente PERL fa sì che quest'ultimo vi "punisca" con un'interruzione di esecuzione quando infrangete certe regole.
- Vi obbliga a dichiarare tutte le variabili
- Evita che le variabili vengano create per errori di battitura
- Le variabili si possono dichiarare usando **my**, **our** o **local**
- Per ora usiamo solo **my**:

```
my $a;                                # value of $a is undef
my ($a, $b, $c); # $a, $b, $c are all undef
my @array;                             # value of @array is ()
```

- E' possibile combinare dichiarazione ed inizializzazione:

```
my @array = qw/A list of words/;
my $a = "A string";
```

# Dichiarazione di variabili

---



- Tutte le operazioni in PERL sono valutate in **contesto lista** o **scalare**. Comportamento differente a seconda del contesto.

```
@array = ('one', 'two', 'three');  
$a = @array;    # scalar context for assignment, return size  
print $a;      # prints 3
```

```
($a) = @array;    # list context for assignment  
print $a;        # prints 'one'
```

```
($a, $b) = @array;  
print "$a, $b";    # prints 'one, two'  
($a, $b, $c, $d) = @array;    # $d is undefined
```

# Contesto lista e scalare

---

- Concatenamento di stringhe

```
$DNA1 = "ATG";  
$DNA2 = "CCC";  
$DNA3 = $DNA1 . $DNA2;      # concatenation operator  
$DNA3 = "$DNA1$DNA2";      # string interpolation  
print "$DNA3";              # prints ATGCC
```

```
$DNA3 = '$DNA1$DNA2';      # no string interpolation  
print "$DNA3";              # prints $DNA1$DNA2
```

# Operazioni su stringhe

---



# Arrays

Un array contiene una lista ordinata di scalari:

```
@gene_array = ('EGF1', 'TFEC', 'CFTR', 'LOC1691');  
print "@gene_array\n";
```

**Output:**

```
EGF1 TFEC CFTR LOC1691
```

```
# there's more than one way to do it (see previous slide  
  on declaring variables)  
@gene_array = qw/EGF1 TFEC CFTR LOC1691/;
```

The 'quote word' function `qw()` is used to generate a list of words. It takes a string such as:

```
tempfile tempdir
```

and returns a quoted list:

```
'tempfile', 'tempdir'
```

[http://www.perlmeme.org/howtos/perlfunc/qw\\_function.html](http://www.perlmeme.org/howtos/perlfunc/qw_function.html)

Un array contiene una lista ORDINATA di scalari:

```
@a = ('one', 'two', 'three', 'four');
```

Ad ogni elemento dell'array è associato un indice. Gli indici partono da 0:

```
print "$a[1] $a[0] $a[3]\n";
```

stampa:

```
two one four
```

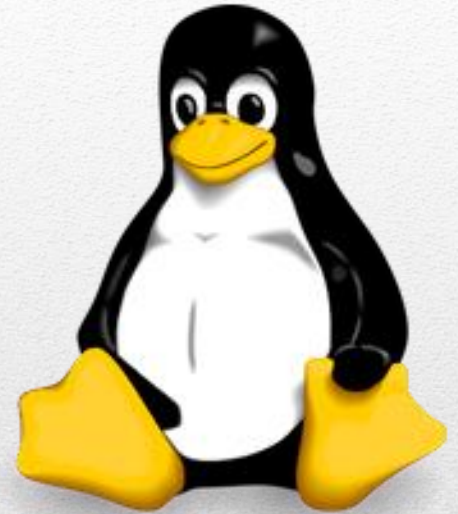
Nota bene:  **$a[i]$**  è uno scalare poichè abbiamo utilizzato il contesto  **$a$**  per riferirci alla variabile.

# Arrays

---



**Diverse modalità a seconda del sistema operativo:** il punto cruciale è come far capire al sistema che vogliamo invocare l'interprete PERL per eseguire lo script.



Prima riga  
esecuzione

	MAC	WIN	*NIX
Prima riga	<code>#!/usr/bin/perl</code>		<code>#!/usr/bin/perl</code>
esecuzione	<code>./nomescript (*)</code>	<code>perl nomescript</code>	<code>./nomescript (*)</code>

# Eseguire script PERL

---

Sistemi UNIX, MAC : nella  
shell scrivere **whereis perl**

```
#!/usr/bin/perl -w
# Example 1   Storing DNA in a variable, and printing it out

# First we store the DNA in a variable called $DNA
$DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';

# Next, we print the DNA onto the screen
print $DNA;

# Finally, we'll specifically tell the program to exit
exit;

-- Changing permissions
% chmod +x example1.pl

-- Running a perl script
% ./example1.pl   (UNIX,MAC)
> perl example1.pl (WIN)
```

Sistemi UNIX, MAC :  
rendere **eseguibile** lo script

# Eseguire script PERL



# Lavorare in PERL

1. Creare una **directory**
2. Aprire una **shell** (\*NIX, MAC) o il **prompt** dei comandi (WIN) e posizionarsi **nella directory** appena creata (comando **cd**)
3. Aprire un **editor** (Notepad (WIN), VIM, KOMOD, ECLIPSE ...)
4. Scrivere un programma in PERL
5. **Salvare il file nella directory** appena creata
6. Tornare alla finestra della **shell** (o al **prompt** dei comandi)
7. **Eseguire** lo script perl

Se ci sono errori, tornare all'editor, modificare lo script, **salvare**, tornare al prompt/shell e rieseguire lo script.

---

- Operatori di Match
- Sostituzione
- Translitterazione
- Funzioni per stringhe
  - length, reverse
- Funzioni per arrays
  - scalar, reverse, sort
  - push, pop, shift, unshift
- Cicli
  - while, foreach, for
- Split e join
- Input/Output

# Argomenti

---



```
$dna = "ATGCATTT";
if ($dna =~ /ATT/) {
    print "$dna contains ATT\n";
}
else {
    print "$dna doesn't contain ATT\n";
}
```



**Output:**

**ATGCATTT contains ATT**

# matching a pattern

```
$dna = "ATGAAATTT";
```

```
$pattern = "GGG";
```

```
if ($dna =~ /$pattern/) {
    print "$dna contains $pattern\n";
}
```

```
else {
    print "$dna doesn't contain $pattern\n";
}
```

```
print "\n";
```

**ATGAAATTT doesn't contain GGG**

# Operatore Match

```
print "substitution example:\n";
$dna = "ATGCATTT";
print "Old DNA: $dna\n";
$dna =~ s/TGC/gggagc/;
print "New DNA: $dna\n\n";
```

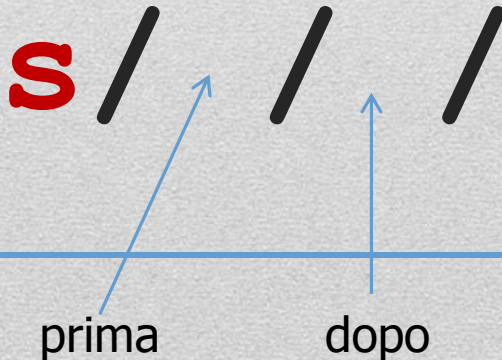
substitution example:  
Old DNA: ATGCATTT  
New DNA: AgggagcATT

```
print "single substitution:\n";
$dna = "ATGCATTT";
print "Old DNA: $dna\n";
$dna =~ s/T/t/;
print "New DNA: $dna\n\n";
```

single substitution:  
Old DNA: ATGCATTT  
New DNA: AtGCATTT

```
print "global substitution:\n";
$dna = "ATGCATTT";
print "Old DNA: $dna\n";
$dna =~ s/T/t/g;
print "New DNA: $dna\n\n";
```

global substitution:  
Old DNA: ATGCATTT  
New DNA: AtGCAtt



# Sostituzione



```
print "removing white space\n";
$dna = "ATG CATTT   CGCATAG";
print "Old DNA: $dna\n";
$dna =~ s/\s//g;
print "New DNA: $dna\n\n";
```

removing white space

Old DNA: ATG CATTT CGCATAG

New DNA: ATGCATTTCGCATAG

```
print "substitution ignoring
      case\n";
$dna = "ATGCAttT";
print "Old DNA: $dna\n";
$dna =~ s/T/U/gi;
print "New DNA: $dna\n\n";
```

substitution ignoring case

Old DNA: ATGCAttT

New DNA: AUGCAUUU

# Sostituzione

# Calcolo della strand complementare di DNA (con **baco**)

```
#!/usr/bin/perl -w
# Calculating the complement of a strand of DNA (with bug)

# The DNA
$strand1 = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
print "strand1: $strand1 \n";

# Copy strand1 into strand2
$strand2 = $strand1;

# Replace all bases by their complements: A->T, T->A, G->C, C->G
$strand2 =~ s/A/T/g;
$strand2 =~ s/T/A/g;
$strand2 =~ s/G/C/g;
$strand2 =~ s/C/G/g;

print "strand2: $strand2 \n";
exit;

% complement1
strand1: ACGGGAGGACGGGAAAATTACTACGGCATTAGC
strand2: AGGGGAGGAGGGGAAAAAAGAAGGGGAAAAGG
```



**Sapete trovare il baco?**



```
print "transliteration operator\n";
$dna = "ATGCAttT";
print "Old DNA: $dna\n";
$dna =~ tr/T/U/;
print "New DNA: $dna\n\n";
```

transliteration operator

Old DNA: ATGCAttT

New DNA: AUGCAttU

```
print "tr on multiple
characters\n";
$dna = "ATGCAttT";
print "Old DNA: $dna\n";
$dna =~ tr/Tt/Uu/;
print "New DNA: $dna\n\n";
```

tr on multiple characters

Old DNA: ATGCAttT

New DNA: AUGCAuuU

tr / / /

# Operatore di Translitterazione

```
print "DNA complement strand\n";  
$dna = "ATGCAttT";  
$complement = $dna;  
$complement =~ tr/AaTtGgCc/TtAaCcGg/;  
print "$dna\n";  
print "$complement\n\n";
```

```
DNA complement strand  
ATGCAttT  
TACGTaaA
```

# Strand Complementare di DNA



# Calcolo della strand complementare di DNA (senza **baco**)

```
#!/usr/bin/perl -w
# Calculating the complement of a strand of DNA

# The DNA
$strand1 = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
print "strand1: $strand1\n";

# Copy strand1 into strand2
$strand2 = $strand1;

# Replace all bases by their complements: A->T, T->A, G->C, C->G
# tr replaces each char in first part with char in second part
$strand2 =~ tr/ATGC/TACG/;

print "strand2: $strand2 \n";
exit;

% complement2
strand1: ACGGGAGGACGGGAAAATTACTACGGCATTAGC
strand2: TGCCCTCCTGCCCTTTTAATGATGCCGTAATCG
```

Come è stato eliminato il baco?

```
print "length function\n";  
$dna = "ATGCAttT";  
$size = length($dna);  
print "DNA $dna has length $size\n\n";
```

length function

DNA ATGCAttT has length 8

# Funzione Length

---



```
print "reverse function\n";
$dna = "ATGCAttT";
$reverse_dna = reverse($dna);
print "DNA: $dna\n";
print "Reverse DNA:
    $reverse_dna\n\n";
```

**reverse function**

**DNA: ATGCAttT**

**Reverse DNA: TttACGTA**

```
print "reverse complement\n";
$dna = "ATGCAttT";
$rev_comp = reverse($dna);
$rev_comp =~
    tr/AaTtGgCc/TtAaCcGg/;
print "$dna\n";
print "$rev_comp\n\n";
```

**reverse complement**

**ATGCAttT**

**AaaTGCAT**

# Funzione Reverse

# Funzioni su Array : scalar, reverse, sort

```
print "array of gene names\n";  
@genes = ("HOXB1", "ALPK1", "TP53");  
$size = scalar @genes;  
print "A list of $size genes: @genes\n";  
@genes = reverse @genes;  
print "Reversed list of $size genes: @genes\n";  
@genes = sort @genes;  
print "Sorted list of $size genes: @genes\n\n";
```

array of gene names

A list of 3 genes: HOXB1 ALPK1 TP53

Reversed list of 3 genes: TP53 ALPK1 HOXB1

Sorted list of 3 genes: ALPK1 HOXB1 TP53

---



# Aggiunta di elementi alla fine di un array

```
print "Appending to an array\n";
@genes = ("HOXB1", "ALPK1", "TP53");
push @genes, "ZZZ3";
$size = scalar @genes;
print "There are now $size genes: @genes\n";
push @genes, ("EGF", "EFGR");
$size = scalar @genes;
print "There are now $size genes: @genes\n\n";
```

Appending to an array

There are now 4 genes: HOXB1 ALPK1 TP53 ZZZ3

There are now 6 genes: HOXB1 ALPK1 TP53 ZZZ3 EGF EFGR

---

# Rimuovere elementi alla fine di un array

```
print "Removing items from end of array\n";
@genes = ("HOXB1", "ALPK1", "TP53", "EGF");
$size = scalar @genes;
print "A list of $size genes: @genes\n";
pop @genes;
$size = scalar @genes;
print "There are now $size genes: @genes\n";
$gene = pop @genes;
$size = scalar @genes;
print "There are now $size genes: @genes\n";
print "There gene removed was $gene\n\n";
```

Removing items from end of array

A list of 4 genes: HOXB1 ALPK1 TP53 EGF

There are now 3 genes: HOXB1 ALPK1 TP53

There are now 2 genes: HOXB1 ALPK1

There gene removed was TP53

---



# Rimuovere elementi dall' inizio di un array

```
print "Removing items from front of array\n";
@genes = ("HOXB1", "ALPK1", "TP53", "EGF");
$size = scalar @genes;
print "A list of $size genes: @genes\n";
shift @genes;
$size = scalar @genes;
print "There are now $size genes: @genes\n";
$gene = shift @genes;
$size = scalar @genes;
print "There are now $size genes: @genes\n";
print "There gene removed was $gene\n\n";
```

Removing items from front of array

A list of 4 genes: HOXB1 ALPK1 TP53 EGF

There are now 3 genes: ALPK1 TP53 EGF

There are now 2 genes: TP53 EGF

There gene removed was ALPK1

```
@genes = ("HOXB1", "ALPK1", "TP53");
while (scalar @genes > 0) {
    $gene = shift @genes;
    print "Processing gene $gene\n";
    # put processing code here
}
```

```
Processing gene HOXB1
Processing gene ALPK1
Processing gene TP53
```

```
@genes = ("HOXB1", "ALPK1", "TP53");
while (@genes) {
    $gene = shift @genes;
    print "Processing gene $gene\n";
    # put processing code here
}
$size = scalar @genes;
print "There are now $size genes in
the list: @genes\n";
```

```
Processing gene HOXB1
Processing gene ALPK1
Processing gene TP53
There are now 0 genes in the list:
```

# Ciclo `while` per il processamento di liste

---



```
print "foreach loop to process all items from a list\n";
@genes = ("HOXB1", "ALPK1", "TP53");
foreach $gene (@genes) {
    print "Processing gene $gene\n";
    # put processing code here
}
$size = scalar @genes;
print "There are still $size genes in the list: @genes\n";
```

foreach loop to process all items from a list

Processing gene HOXB1

Processing gene ALPK1

Processing gene TP53

There are still 3 genes in the list: HOXB1 ALPK1 TP53

# Ciclo foreach per il processamento di liste

---

```
print "another for loop to process a list\n";
@genes = ("HOXB1", "ALPK1", "TP53");
$size = scalar @genes;
for (my $i = 0; $i < $size; $i++) {
    $gene = $genes[$i];
    print "Processing gene $gene\n";
    # put processing code here
}
$size = scalar @genes;
print "There are still $size genes in the list: @genes\n";
```

another for loop to process a list

Processing gene HOXB1

Processing gene ALPK1

Processing gene TP53

There are still 3 genes in the list: HOXB1 ALPK1 TP53

# Ciclo `for` per il processamento di liste

---



```
print "converting array to
string\n";
@genes = ("HOXB1", "ALPK1",
"TP53");
$string = join(" ", @genes);
print "String of genes:
$string\n";
$size = length $string;
print "String has length:
$size\n";
```

converting array to string  
String of genes: HOXB1 ALPK1 TP53  
String has length: 16

```
print "join with empty
separator\n";
@genes = ("HOXB1", "ALPK1",
"TP53");
$string = join("", @genes);
print "String of genes:
$string\n";
$size = length $string;
print "String has length:
$size\n";
```

join with empty separator  
String of genes: HOXB1ALPK1TP53  
String has length: 14

# join: convert array in string

```
print "join with newline separator\n";
@genes = ("HOXB1", "ALPK1", "TP53");
$string = join "\n", @genes;
print "String of genes: $string\n";
$size = length $string;
print "String has length: $size\n\n";
```

join with newline separator

String of genes: HOXB1

ALPK1

TP53

String has length: 16

# join con separatore **newline**



```
print "converting string to array\n";
$dna = "ATGCATTT";
@bases = split "", $dna;
print "dna = $dna\n";
$size = scalar @bases;
print "The list of $size bases: @bases\n\n";
```

converting string to array

dna = ATGCATTT

The list of 8 bases: A T G C A T T T

**split: conversione  
di stringhe in array**

---

```
print "split on white space\n";
$string = "HOXB1 ALPK1 TP53";
@genes = split " ", $string;
print "$string\n@genes\n\n";
```

split on white space

HOXB1 ALPK1 TP53

HOXB1 ALPK1 TP53

```
print "split on 'P'\n";
$string = "HOXB1 ALPK1 TP53";
@genes = split "P", $string;
print "$string\n";
foreach $gene (@genes) {
    print "|$gene|\n";
}
```

split on 'P'

HOXB1 ALPK1 TP53

|HOXB1 AL|

|K1 T|

|53|

# split: utilizzo separatori

---



L'array **@ARGV** è la lista degli **argomenti** che vengono passati al programma:

```
% myprogram.pl hello 73 abcdef
```

è equivalente a:

```
@ARGV = ("hello", 73, "abcdef");
```

**@ARGV : un array speciale**

**Per ha due modi principali di aprire I files:**

- alla maniera della shell (prompt) per convenienza
- Alla maniera del linguaggio C per precisione

**Apertura files: la via della shell:**

```
% myprogram file1 file2 file3
% myprogram < inputfile
% myprogram > outputfile
% myprogram >> outputfile
% myprogram | otherprogram
% otherprogram | myprogram
```

---

# Apertura files



```
#!/usr/bin/perl
use strict;
use warnings;
# File: readname.pl

# Read in name and age
print "Enter your name: ";
my $name = <>;          # "<>" reads one line from "standard input"
chomp $name;           # chomp deletes any newlines from end of
                        string

print "Enter your age: ";
my $age = <>;
chomp $age;

print "Hello, $name! ";
print "On your next birthday, you will be ", $age+1, ".\n";
exit;
```

```
% readname.pl
Enter your name: Joe Smith
Enter your age: 20
Hello, Joe Smith! On your next birthday, you will be 21.
```

# Input interattivo da tastiera

```
% cat infile
Joe Smith
20
```



**NBB: variabili separate  
da newline**

```
% readname.pl < infile
Enter your name: Enter your age: Hello, Joe Smith!  On your
next birthday, you will be 21.
%
```

# Input da file

( Comodo per leggere **file parametri** )



```
#!/usr/bin/perl
use strict;
use warnings;
```

**Commentate ( # ) tutte le chiamate  
print PRIMA della lettura del file**

```
# Read in name and age
# print "Enter your name: ";
my $name = <>;           # "<>" reads one line from "standard input"
chomp $name;           # chomp deletes any newlines from end of string
```

```
# print "Enter your age: ";
my $age = <>;
chomp $age;
```

```
print "Hello, $name! ";
print "On your next birthday, you will be ", $age+1, ".\n";
exit;
```

```
% readname.pl < infile
```

```
Hello, Joe Smith! On your next birthday, you will be 21.
```

```
%
```

# Input da file file

( visto dall'**interno** dello script)

- La funzione **open** richiede **due** argomenti:
  - Un **filehandle**
  - Una **stringa** che dice a PERL **cosa** aprire e **come**.
- Open ritorna 1 quanto tutto è andato bene o mette la descrizione dell'errore nella variabile speciale **\$!** e poi blocca il programma
- Se un filehandle con lo stesso nome è già aperto **lo chiude** implicitamente

```
open INFO, "< datafile" or die "can't open datafile: $!";  
open RESULTS,"> runstats" or die "can't open runstats: $!";  
open LOG, ">> logfile " or die "can't open logfile: $!";
```

```
<      modalità lettura  
>      Modalità scrittura  
>>    Modalità append (aggiunta in coda)
```

# Funzione **open**

---



- La funzione **close** richiede **un** argomento:
  - Il **filehandle** aperto in precedenza dalla funzione open

```
open INFO, "< datafile" or die "can't open datafile: $!";
```

```
# ... fare qualcosa con il file
```

```
close INFO;
```

---

# Funzione **close**

Perl.com

( <http://www.perl.com> )

Perl.it

( <http://www.perl.it> )



**Buone fonti di informazioni**

( manuali, tutorials, discussioni, recensioni ... )