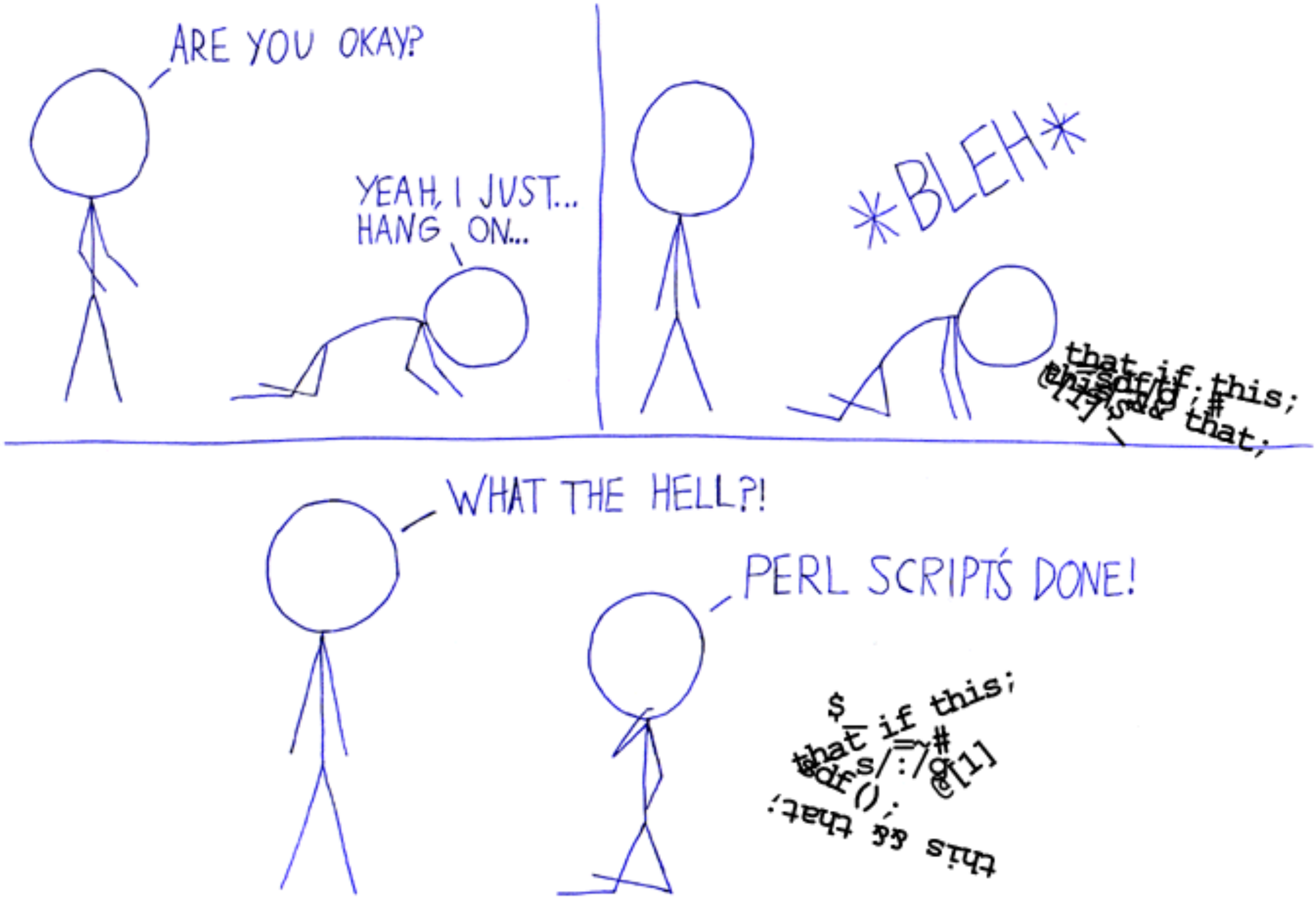
Tutto Perl in 2 ore

The CRIBI Crash Course

Andrea Telatin e Giovanni Birolo
Bioinformatics Unit, CRIBI



from: http://www.zoitz.com/archives/13

Cosa faremo oggi

- Visioneremo i primi assaggi di Perl (lab02)
- Cercheremo di capire come si traduce una «necessità» in programma
- Affronteremo le principali strutture di Perl
  - Gestione dati (variabili scalari, array e hash)
  - Sintassi (if, for, while, subroutines)
  - Leggere un file

Come lo faremo

- Oggi un lungo e faticoso **tour de force** nel linguaggio

- Rilasceremo piccoli esercizi sul blog per tenervi attivi fino al prossimo incontro. Fateli.

- La teoria serve meno della pratica!

http://www.html.it/guide/guida-perl/

Come lo faremo

Serve la voglia di **provare**,
e a volte qualche piccolo atto di fede

alla fine vedrete che sarete in grado
di programmare autonomamente!

Come lo faremo

Serve la voglia di **provare**,
e a volte qualche piccolo atto di fede

alla fine vedrete che sarete in grado
di programmare autonomamente!

= se non fate esercizio da soli,
siete **spacciati**

Homeworks

fatti?

Cosa voglio fare?

- Dato un allineamento in formato SAM **(INPUT)**
- Calcolare quante read sono allineate in ciascun cromosoma **(TASK)**
- E stampare una “tabella” con tre colonne: cromosoma, numero di allineamenti, percentuale **(OUTPUT)**

A matter of quotes

“ and ”, as ‘ or ’ **have** to be intended as " and '

we’ll meet the weird `backticks` somewhere

Un programma Perl

```
1  # A volte meglio mettere dei commenti
2
3  print “Hello World!\n”;
4  if (2 < 3) {
5      print “Due è minore di tre.\n”;
6  }
7
8
9
10
11
12
13
```

Un programma Perl

```
1  # A volte meglio mettere dei commenti
2
3  print “Hello World!\n”;
4  if (2 < 3) {
5      print “Due è minore di tre.\n”;
6  }
7
8
9
10
11
12
13
```

Un programma Perl

```
1  # A volte meglio mettere dei commenti
2
3  print “Hello World!\n”;
4  if (2 < 3) {
5      print “Due è minore di tre.\n”;
6  }
7
8
9
10
11
12
13
```

Un programma Perl

```
1  # A volte meglio mettere dei commenti
2
3  print “Hello World!\n”;
4  if (2 < 3) {
5      print “Due è minore di tre.\n”;
6  }
7
8
9
10
11
12
13
```

Un programma Perl

```
1  # A volte meglio mettere dei commenti
2
3  print “Hello World!\n”;
4  if (2 < 3) {
5      print “Due è minore di tre.\n”;
6  }
7
8
9
10
11
12
13
```

Tipi di dati

- **Variabili scalari** (testo o numeri) come $nome
  - contengono 1 nome o numero $nome = ‘pino’;
- **Array** di scalari (…) come @array
  - elenco di scalari @nome = (‘lino’, ‘giovanni’, $nome);
- **Array associativi**, od hash %hash
  - chiave e valore come %eta = (‘lino’ => 21, ‘momi’ => 2);

Variabili scalari

- Le variabili, in generale, sono delle scatoline, hanno un **nome proprio** e un **contenuto**.
- Il contenuto, come si intuisce, puo' essere modificato
- Le variabili scalari, o scalari, possono contenere 1 numero o 1 stringa (numeriche o testuali)

Variabili scalari

- **Creazione (opzionale)**
  my $nome;

- **Assegnazione di un valore**
  $nome = ‘Pina’;
  $colore_pref = ‘ciano’;
  $age = 21;

- **Alterazione del valore**
  $age = $age + 3;
  $colore_pref = $colore_pref . ‘ chiaro’;
  $age++;

Array

- Sono **liste** di scalari

- **Creazione di un array vuoto:**
  my @appello;

- **Inizializzazione di un array:**
  @appello = (‘Giov’, ‘Andre’, ‘Nick’, ‘Risso’, ‘Chuck’);
  @prova = (‘Mio’, ‘Tuo’, 21, 22.2, ‘Tombola’);

Array e indici

- dato l'array:
  @array = ('Uno', 'due', 'tre', $jolly, $altro);

- ogni elemento ha un **indice** numerico da 0 a *n.*

- l'ultimo indice è $#array. Quanti elementi in tutto?

- $terzo = $array[2];
  print "Il terzo elemento è $terzo (= $array[2])\n";
  print "L'ultimo elemento ha indice $#array,
  ed è $array[-1]\n";

Funzioni per array

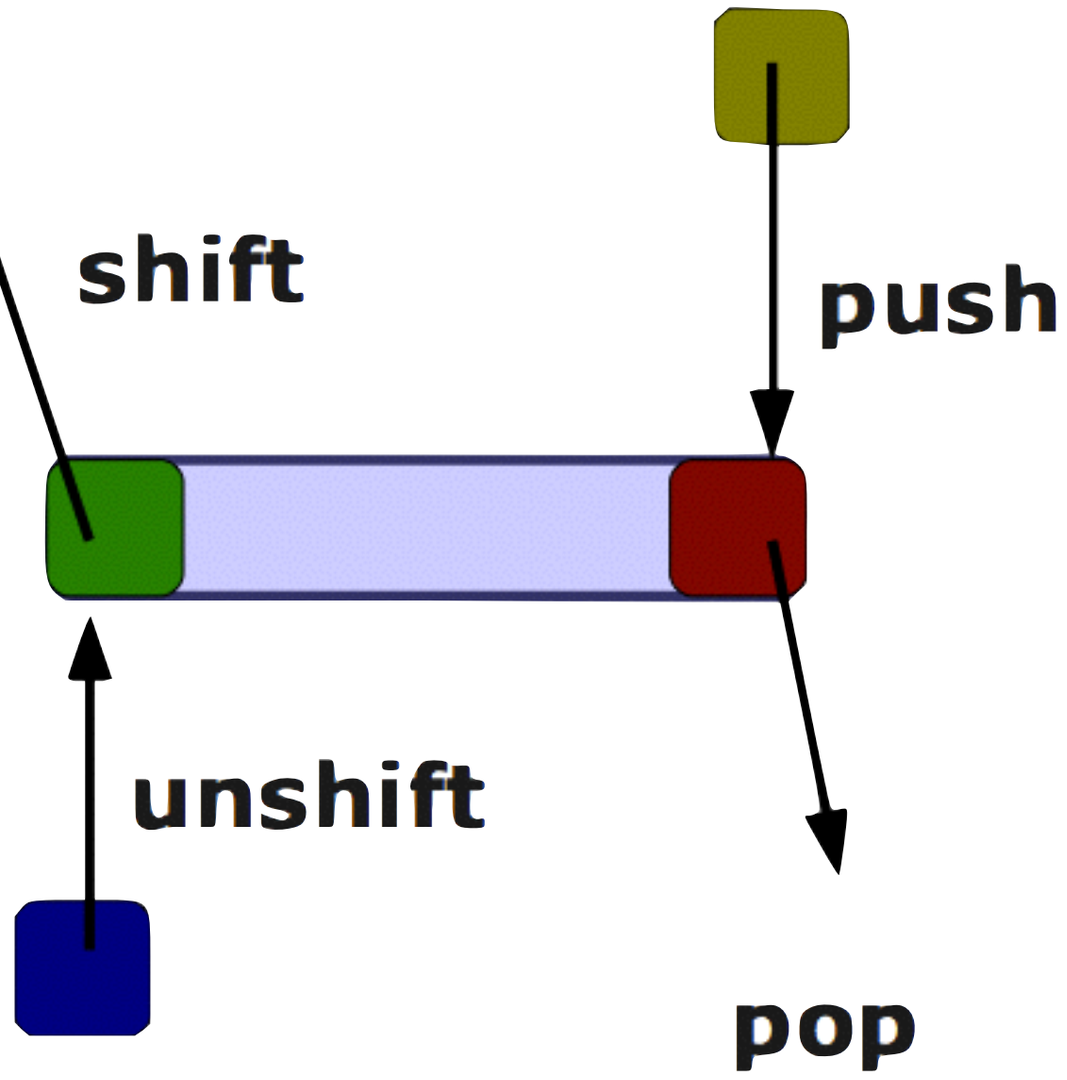
- **aggiungi una variabile: push**
  push(@array, ‘Nuovo elemento’);
  push(@array, ‘Uno’, ‘due’, ‘tre’, $jolly, $altro);

- **estrai la prima variabile (e toglila):**
  $primo = shift(@array);
  shift(@array);

- **estrai l’ultima variabile (e toglila):**
  $ultima = pop(@array);

Funzioni per array

- **in sintesi**



Blocco di codice

- **Un set di istruzioni da essere eseguite si chiama blocco, ed in Perl è un listato racchiuso da graffe.**

```
{
    blocco
    di
    comandi…
}
```

If . . .

- **Eseguo un blocco di codice se si verifica una condizione**

```
1 if ($nome eq ‘Giovanni’) {
2     # eseguo tutto quello che è fra { e }
3     print “CIAO!\n”;
4 } else {
5     # eseguo tutto quest’altro blocco
6     print “E allora, chi sei?\n”;
7 }
8
9
10
```

If . . .

- **Eseguo un blocco di codice se si verifica una condizione**

```
1 if ($nome eq ‘Giovanni’) {
2     # eseguo tutto quello che è fra { e }
3     print “CIAO!\n”;
4 } else {
5     # eseguo tutto quest’altro blocco
6     print “E allora, chi sei?\n”;
7 }
8
9
10
```

Condizione che deve essere VERA

If . . .

- **Eseguo un blocco di codice se si verifica una condizione**

```
1 if ($nome eq ‘Giovanni’) {
2     # eseguo tutto quello che è fra { e }
3     print “CIAO!\n”;
4 } else {
5     # eseguo tutto quest’altro blocco
6     print “E allora
7 }
8
9
10
```

Blocco di codice eseguito SE la condizione è verificata

If . . .

- **Eseguo un blocco di codice se si verifica una condizione**

La condizione deve poter essere “vera” o “falsa”

Esempi:
La riga corrente è di intestazione FASTA?
—> Ovvero il primo carattere è un “>”

```
1 if ($nome eq 'Giovanni') {
2     # eseguo tutto quello che è fra { e }
3     print “CIAO!\n”;
4 } else {
5     # eseguo tutto quest'altro blocco
6     print “E allora?\n”;
7 }
8
9
10
```

Blocco di codice eseguito SE la condizione è verificata

While …

- **Eseguo un blocco di codice fin tanto che una condizione è vera (ovvero finché non diventa falsa)**

```
1 while (piove) {
2     print “Sono stanco della pioggia…\n”
3 }
4 print “ FINALMENTE IL SOLEEEEE\n”;
5
6
7
8
9
10
```

While …

- **Eseguo un blocco di codice fin tanto che una condizione è vera (ovvero finché non diventa falsa)**

```
1 while ($risultato < 1000) {
2     $risultato = $risultato**2;
3     print “Siamo arrivati a $risultato\n”;
4 }
5 print “ FINE: $risultato >= 1000\n”;
6
7
8
9
10
```

While …

- **Eseguo un blocco di codice fin tanto che una condizione è vera (ovvero finché non diventa falsa)**

```
1 while ($risultato < 1000) {
2     $risultato = $risultato**2;
3     print “Siamo arrivati a $risultato\n”;
4 }
5 print “ R                    1000\n”;
6
7
8
9
10
```

Condizione: entro nel ciclo WHILE solo se è VERA (altrimenti lo salto)

While …

- **Eseguo un blocco di codice fin tanto che una condizione è vera (ovvero finché non diventa falsa)**

```
1 while ($risultato < 1000) {
2     $risultato = $risultato**2;
3     print “Siamo arrivati a $risultato\n”;
4 }
5 print “ FINE: $risultato >= 1000\n”;
6
7
8
9
10
```

Questo blocco di codice viene eseguito FINO A CHE la condizione è vera.
Attenzione: anche all’infinito se non cambia qualcosa durante l’esecuzione

While …

- **Eseguo un blocco di codice fin tanto che una condizione è vera (ovvero finché non diventa falsa)**

```
1 while (il file non è finito) {
2     $riga = leggi_una_riga();
3     $contaRighe++;
4 }
5 print “ Il file ha $contaRighe righe\n”;
6
7
8
9
10
```

L’uso tipico che faremo è leggere un file riga per riga.
La condizione che verifichiamo è che non sia l’ultima riga.
Non sappiamo a priori quante righe sono, ma sappiamo che dopo l’ultima dobbiamo uscire :)

For …

- **Eseguo un blocco di codice un certo numero di volte, usando una variabile “contatore”**

```
1 for ($i=10; $i<20; $i++) {
2     # eseguo tutto quello che è fra { e }
3     print “Ciao, $i volta su 20!\n”;
4 }
5
6
7
8
9
10
```

For …

- **Eseguo un blocco di codice un certo numero di volte, usando una variabile “contatore”**

```
1 for ($i=10; $i<20; $i++) {
2     # eseguo tutto quello che è fra { e }
3     print “Ciao, $i volta su 20!\n”;
4 }
5
6
7
8
9
10
```

Inizializzo il contatore

For …

- **Eseguo un blocco di codice un certo numero di volte, usando una variabile “contatore”**

```
1 for ($i=10; $i<20; $i++) {
2     # eseguo tutto quello che è fra { e }
3     print “Ciao, $i volta su 20!\n”;
4 }
5
6
7
8
9
10
```

Eseguo fintanto che questa condizione è VERA

For …

- **Eseguo un blocco di codice un certo numero di volte, usando una variabile “contatore”**

```
1 for ($i=10; $i<20; $i++) {
2     # eseguo tutto quello che è fra { e }
3     print “Ciao, $i volta su 20!\n”;
4 }
5
6
7
8
9
10
```

Eseguo questa operazione AD OGNI CICLO
(ovvero incremento il contatore)

For …

- **Eseguo un blocco di codice un certo numero di volte, usando una variabile “contatore”**

```
1 for ($i=10; $i<20; $i++) {
2     # eseguo tutto quello che è fra { e }
3     print “Ciao, $i volta su 20!\n”;
4 }
5
6
7
8
9
10
```

Eseguo questa operazione AD OGNI CICLO
(ovvero incremento il contatore)

For …

- **Eseguo un blocco di codice un certo numero di volte, usando una variabile “contatore”**

```
1 for ($i=10; $i<20; $i++) {
2     # eseguo tutto quello che è fra { e }
3     print “Ciao, $i volta su 20!\n”;
4 }
5
6
7
8
9
10
```

C’è un ultimo ciclo, lo vedremo dopo

pausa

Un po’ di funzioni

Lessico

valore = funzione(argomenti)

- Una funzione puo’ richiedere o meno dei parametri, si chiamano **argomenti**

- Se una funzione emette un risultato, si dice che “**restituisce**”, o **ritorna**, un valore.

- Una funzione può restituire scalari, array o altro…

una prima funzione: int

- la funzione int() rimuove i decimali di un numero, che li abbia o meno
- **int($numero)** restituisce il numero troncato, ma non altera il suo argomento

- $intero = int(9/8);
  $intero = int($fraction);

```
print “Ciao!\n”;
$stud = 29;
$bisc = 291;

# si aggiunge uno
# studente!
$stud++;

$bps = $bisc/$stud;

$tondi = int($bps);

print “Ogni studente
avrà $tondi biscotti\n”;
```

un’altra funzione: sqrt

- la funzione sqrt() ritorna la radice quadrata dell’argomento.

- **sqrt($numero)** restituisce un numero, potenzialmente con la virgola

- $radice = sqrt(9/8);
  $root = sqrt($valore*2);

```
print “Lato quadrato\n”;

$area = 1000;
$lato = sqrt($area);

print “ Area: $area cm2
Lato: $lato cm\n”;
```

casualità: rand

- la funzione rand() ritorna un numero casuale.

- **sqrt()** ritorna da 0 a 1.

- Se specifico un argomento: rand(10) da 0 a 9.99.

- $num = rand();
  $num = int(rand(91));

```
# Programma lancia dadi
print “Lancio n dadi\n”;

$num_dadi = 20;
for ($i=1; $i<=$num_dadi; $i++) {
    $valore = int(rand(7));
    print “Dado n. $i: $valore\n”;
}

print “Ho finito.\n”;
```

ancora su print

- **print** stampa una stringa da qualche parte :)
- print DOVE “Stringa da stampare”.
- Dove è un “filehandle” che avete creato voi, o uno standard come STDOUT o STDERR.

```
print “Questo va in output”;      # sottinteso STDOUT
print STDERR “Attenzione!\n”; # Importante per messaggi
print O “Stampa sul file…”;       #vedi oltre
```

ancora su print

- **Abituatevi fin da subito a usare**
  print STDERR “Messaggio\n”;
  per i messaggi informativi

- **E usare**
  print “Valori e risultati...\n”;
  per l’output vero e proprio del programma

funzioni per stringhe

- **uc($stringa)** converte una stringa in maiuscolo
- **lc($stringa)**, in minuscolo
- **length($stringa)** riporta la lunghezza in caratteri
- $NOME = uc($nome);
  $size = length($dna);

```
print “Ciao!\n”;
$stud = 29;
$bisc = 291;
# si aggiunge uno
# studente!
$stud++;
$bps = $bisc/$stud;
print “Ogni studente avrà
$bps biscotti\n”;
$tondo = int($bps);
print “Intero = $tondo\n”;
```

Togliere \n da stringhe

- la funzione **chop()** toglie (e restituisce) l'ultimo carattere di una stringa:
  $last = chop($string);

- la funzione chomp() toglie l'ultimo carattere **solo** se è un a capo.
  chomp($riga);

```
print "Ciao!\n";
$stud = 29;
$bisc = 291;
# si aggiunge uno
# studente!
$stud++;
$bps = $bisc/$stud;
print "Ogni studente avrà
$bps biscotti\n";
$tondo = int($bps);
print "Intero = $tondo\n";
```

Holy grail: substr

- “**substring**”: a function to retrieve a slice of a string

- $stringa = ‘pipi e pupu’;
  $slice = substr($stringa, 3, 10);

- Tutte le funzioni accettano variabili come argomenti:
  $slice = substr($stringa, $from, $span);

- Può fare cose più complicate! —> Leggi il manuale

Un primo esempio

```
$sequenza =
'ACGATCGTACGTAGCTGACTGATCGTACGTAACGTACGTAGCT
GACTGATCGATCGTAGCTAGCAGCTGATCGACTGACTGACTGAT
CGATCGTACGTAGCTAGCTGACTGATC';
```

```
$lunghezza = length($sequenza);
$lunghezza_read = 12;
$posizione = int(rand($lunghezza - $lunghezza_read);
$seq = substr($sequenza, $posizione, $lunghezza_read);
```

```
print ">Random\n$seq\n";
```

Ultimo ciclo: foreach

- **ripeti un blocco di codice per ciascun elemento di un array**

- Proviamo senza:

```
@lista = (‘ciao’, ‘mondo’, ‘crudele’);
for ($i=0; $i<=$#lista; $i++) {
  $elemento = $lista[$i];
  print “ -> $elemento\n”;
}
```

Ultimo ciclo: foreach

- **ripeti un blocco di codice per ciascun elemento di un array**

- Proviamo con:

```
@lista = (‘ciao’, ‘mondo’, ‘crudele’);
foreach $elemento (@lista) {
    print “ - $elemento\n”;
}
```

- In italiano vuol dire scorri l’array e ad ogni ciclo metti l’elemento corrente dentro una comoda variabile.

dividere stringe: split

- **split** spezza una stringa ogni volta che trova un *pattern* (il pattern viene rimosso).
- I pezzi possono essere tanti quindi restituisce un array!

- ```
  @campi = split(“\t”, $riga_sam);
  $num_campi = $#campi + 1;
  print “Nome sequenza: $campi[0],
  cromosoma: $campi[2]\n”;
  ```
  ```

unire stringhe: join

- al contrario di split che prende una stringa e la spezza (creando un array), **join** prende un array e lo unisce in una stringa

- Il *pattern* in questo caso funge da “separatore”.

- $nuova_riga_sam = join(@pezzi, “\t”);
  $elenco = join(@nomi, “,”);

mettere in ordine: sort

- restituisce un array ordinato alfabeticamente, non modifica l'originale

```
@sorted = sort(@appello);
```

- con dei trucchi si ordina secondo altri criteri, ad esempio:

```
@numeri = sort {$a <=> $b} (@lista_numeri);
```

hash e i suoi array: keys

- un hash è un doppio array, un array “chiavi” in relazione con un array “valori”.

```
@chiavi = keys(%hash);
@valori = values(%hash);
```

- In questo modo possiamo usare le funzioni già note.

pausa

Un array speciale @ARGV

- @ARGV contiene tutti i parametri passati dalla riga di comando (shell)

- esempio:

```
$ perl tuoprogr.pl ciao caro!          (comando)
```

```
print “$ARGV[0] $ARGV[1]\n”;          (tuoprog.pl)
```

Leggere un file

- Consideriamo di avere un file chiamato “lista.txt” nella directory /home/geno/files/.

- Aprire un file significa permettere a Perl di leggerlo una riga alla volta.

- Comando per aprire un file:

- open(F, “nomefile”);

Leggere un file

- Come si usa il comando **open**? È una funzione che restituisce vero se riesce o falso se non.

```
if (open(I, “filename”)) {
    print “Ho aperto il file.\n”;
} else {
    die “ Non riesco ad aprire il file filename.\n”;
}
```

- Come si legge il file poi?

Leggere un file

- if (open(SAM, “filename”) == 0) {
  die “ Non riesco ad aprire il file filename.\n”;
  }

- Come si legge il file poi?
  while ($riga = <SAM>) {
  print “$riga”;
  }

- Attenzione: $riga contiene sempre l’a-capo alla fine!

Leggere un file: *cat.pl*

- if (open(I, “$filename”) == 0) {
  die “ Non riesco ad aprire il file \“$filename\”.\n”;
  }

```
while ($riga = <I>) {
   chomp($riga);
   $c++;
   print “$riga\n”;
}
print STDERR “Ho finito di leggere il file: ha $c righe totali.
\n”;
```

Leggere un file: *cat.pl*

- ```
  $filename = $ARGV[0];
  if ($filename eq ‘’) {
     die “ ERROR: Questo programma richiede il nome di un file!\n”;
  }
  ```

```
if (open(I, “$filename”) == 0) {
   die “ Non riesco ad aprire il file \“$filename\”.\n”;
}

while ($riga = <I>) {
   chomp($riga);
   $c++;
   print “$riga\n”;
}
print STDERR “Ho finito di leggere il file: ha $c righe totali.\n”;
```
```

pausa