

Perl Riferimento Rapido

Variabili, regex, I/O su file, riferimenti, moduli essenziali

Basi

Hello World

```
#!/usr/bin/perl
use strict;
use warnings;
print "Hello, World!\n";
say "Hello, World!"; # con use feature 'say';
```

Eseguire Perl

```
perl script.pl # esegui un file
perl -e 'print "hi\n"' # esegui inline
perl -ne 'print' file # elabora file riga per riga
```

Commenti e Documentazione

```
# commento su una riga
=pod
Documentazione POD multi-riga
=cut
```

Variabili

Sigil

\$scalar	Valore singolo (stringa, numero, riferimento)
@array	Lista ordinata di scalari
%hash	Coppie chiave-valore
\$array[0]	Accede a singolo elemento array (contesto scalare)
\$hash{key}	Accede a singolo valore hash (contesto scalare)

Variabili Scalari

```
my $name = "Perl"; # stringa
my $version = 5.40; # numero
my $count = 42; # intero
my $undef; # non definito (undef)
my $combined = "$name v$version"; # interpolazione
```

Contesto

```
my @arr = (1, 2, 3);
my $count = @arr; # contesto scalare: 3
my @copy = @arr; # contesto lista: (1, 2, 3)
my $len = scalar @arr; # forza contesto scalare
```

Variabili Speciali

\$_	Variabile di default (topic)
@_	Argomenti della subroutine
\$_!	Messaggio di errore di sistema
\$_@	Errore da eval
\$0	Nome del programma
@ARGV	Argomenti della riga di comando
%ENV	Variabili d'ambiente

Operatori

Operatori di Confronto

==, !=, <, >, <=, >=	Confronto numerico
eq, ne, lt, gt, le, ge	Confronto su stringhe
<=>	Spaceship numerico (restituisce -1, 0, 1)
cmp	Spaceship su stringhe
=~	Corrispondenza / binding regex
!~	Corrispondenza regex negata

Operatori su Stringhe

```
my $full = "Hello" . " " . "World"; # concatenazione
my $line = "-" x 40; # ripetizione
my $len = length($full); # 11
```

Operatori Logici

&& / and	AND logico (bassa precedenza: and)
 / or	OR logico (bassa precedenza: or)
//	Defined-or (restituisce il sinistro se definito)
! / not	NOT logico
? :	Condizionale ternario

Flusso di Controllo

Condizionali

```
if ($x > 0) { print "positivo\n"; }
elsif ($x == 0) { print "zero\n"; }
else { print "negativo\n"; }
print "si\n" if $condition; # if postfisso
print "no\n" unless $condition; # unless postfisso
```

Cicli

```
for my $i (0..9) { print "$i\n"; }
foreach my $item (@array) { print "$item\n"; }
while ($line = <STDIN>) { chomp $line; }
until ($done) { last if check(); }
```

Controllo dei Cicli

next	Salta alla prossima iterazione (come continue)
last	Esce dal ciclo (come break)
redo	Ripristina l'iterazione corrente
next LABEL	Salta all'iterazione successiva del ciclo etichettato
last LABEL	Esce dal ciclo etichettato

Given / When

```
use feature 'switch';
given ($status) {
    when ("ok") { say "successo"; }
    when ("error") { say "fallito"; }
    default { say "sconosciuto"; }
}
```

Subroutine

Subroutine di Base

```
sub greet {
    my ($name) = @_;
    return "Hello, $name!";
}
my $msg = greet("Alice");
```

Parametri con Default e Nominati

```
sub connect {
    my (%opts) = @_;
    my $host = $opts{host} // "localhost";
    my $port = $opts{port} // 5432;
    return "$host:$port";
}
connect(host => "db.example.com", port => 3306);
```

Riferimenti a Subroutine

```
my $double = sub { return $_[0] * 2; };
print $double->(5); # 10
my @sorted = sort { $a <=> $b } @nums;
```

Prototipi e Firme

```
use feature 'signatures';
sub add($a, $b) { return $a + $b; }
sub greet($name, $greeting = "Hello") {
    return "$greeting, $name!";
}
```

Regex

Corrispondenza

```
if ($str =~ /pattern/) { print "trovato\n"; }
if ($str =~ /\d+/) { print "numero: $1\n"; }
my @matches = ($str =~ /\w+/g); # tutte le corrispondenze
```

Sostituzione

```
$str =~ s/old/new/; # prima occorrenza
$str =~ s/old/new/g; # globale (tutte le occorrenze)
$str =~ s/^\s+|\s+$//g; # rimuovi spazi iniziali/finali
(my $clean = $str) =~ s/\W//g; # copia non distruttiva
```

Modificatori

/i	Case-insensitive
/g	Globale (tutte le corrispondenze)
/m	Multi-riga (^ e \$ corrispondono ai confini di riga)
/s	Riga singola (. corrisponde a newline)
/x	Esteso (consente spazi e commenti)

Pattern Comuni

\d, \D	Cifra / non-cifra
\w, \W	Carattere parola / non-parola
\s, \S	Spazio / non-spazio
\b	Confine di parola
(?: ...)	Gruppo non-catturante
(?<name> ...)	Cattura nominata (accedi con \${name})

I/O su File

Apri e Leggi

```
open(my $fh, '<', 'data.txt') or die "Impossibile aprire: $!";
while (my $line = <$fh>) {
    chomp $line;
    print "$line\n";
}
close($fh);
```

Scrivi e Aggiungi

```
open(my $fh, '>', 'out.txt') or die "Impossibile aprire: $!";
print $fh "Hello\n";
close($fh);
open(my $fh, '>>', 'log.txt') or die "Impossibile aprire: $!";
print $fh "entry\n";
close($fh);
```

Leggiintero File

```
use File::Slurp;
my $content = read_file('data.txt');
my @lines = read_file('data.txt', chomp => 1);
```

Test sui File

-e \$path	Il file esiste
-f \$path	È un file normale
-d \$path	È una directory
-r / -w / -x	Leggibile / scrivibile / eseguibile
-s \$path	Dimensione del file in byte (0 se vuoto)
-z \$path	Il file ha dimensione zero

Perl Riferimento Rapido

Array e Hash

Array

```
my @arr = (1, 2, 3, 4, 5);
push @arr, 6;           # aggiunge in fondo
my $last = pop @arr;   # rimuove l'ultimo
my $first = shift @arr; # rimuove il primo
unshift @arr, 0;       # aggiunge in testa
my @slice = @arr[1..3]; # fetta
```

Funzioni su Array

scalar @arr	Numero di elementi
push / pop	Aggiunge/rimuove dalla fine
shift / unshift	Rimuove/aggiunge dall'inizio
splice(@a, 2, 1)	Rimuove 1 elemento all'indice 2
sort @arr	Ordina alfabeticamente
reverse @arr	Ordine inverso
grep { /pat/ } @arr	Filtra per pattern
map { \$_ * 2 } @arr	Trasforma ogni elemento
join(',', @arr)	Unisce in stringa

Hash

```
my %user = (name => "Alice", age => 30);
$user{email} = "a@b.com"; # aggiunge coppia
delete $user{age};        # rimuove coppia
my @keys = keys %user;
my @vals = values %user;
```

Iterazione Hash

```
while (my ($k, $v) = each %hash) {
    print "$k => $v\n";
}
for my $key (sort keys %hash) {
    print "key: $hash{$key}\n";
}
```

Riferimenti

Creazione di Riferimenti

```
my $scalar_ref = \$name;
my $array_ref  = \@arr;
my $hash_ref   = \%hash;
my $anon_arr   = [1, 2, 3]; # ref array anonimo
my $anon_hash  = {a => 1, b => 2}; # ref hash anonimo
```

Dereferenziazione

```
print $$scalar_ref; # dereferenzia scalare
print $array_ref->[0]; # notazione freccia
print $hash_ref->{key}; # notazione freccia
my @copy = @$array_ref; # dereferenzia in array
my %copy = %$hash_ref; # dereferenzia in hash
```

Strutture Dati Complesse

```
my @users = (
    { name => "Alice", age => 30 },
    { name => "Bob",   age => 25 },
);
print $users[0]->{name}; # "Alice"
```

Funzione ref()

ref(\$r) eq 'SCALAR'	Riferimento a scalare
ref(\$r) eq 'ARRAY'	Riferimento ad array
ref(\$r) eq 'HASH'	Riferimento a hash
ref(\$r) eq 'CODE'	Riferimento a subroutine

Moduli

Usare i Moduli

```
use strict;
use warnings;
use List::Util qw(sum max min);
use File::Basename;
use Cwd qw(abs_path);
```

Creare un Modulo

```
# MyModule.pm
package MyModule;
use Exporter 'import';
our @EXPORT_OK = qw(helper);
sub helper { return "help"; }
1; # il modulo deve restituire vero
```

Moduli Core Comuni

List::Util	sum, max, min, reduce, any, all
File::Basename	basename, dirname, fileparse
File::Path	make_path, remove_tree
Getopt::Long	Parsing opzioni riga di comando
JSON	encode_json, decode_json
LWP::Simple	get(\$url) — client HTTP semplice
Data::Dumper	Debug dump di strutture dati
Carp	croak, confess — messaggi di errore migliori

CPAN

```
cpan install Module::Name # installa da CPAN
cpanm Module::Name       # cpanminus (più veloce)
perldoc Module::Name     # leggi documentazione modulo
```