Elements of Programming in Perl

<H16-4/5>

Scalar Variables and Operators

Josep F. Abril

jabril@imim.es

$calars in Perl

$ ⇨ **$calar values**

```
$dnaseq = "ATGGGTA"; # Strings
$counter = 1;        # Integers
$factor = 0.005;     # Real numbers
```

Double quotes: Variable interpolation

```
$string = "Factor is $factor";
print $string,"\n"; #-> Factor is 0.005
```

Single quotes: Literal values

```
$string = 'Factor is $factor';
print $string,"\n"; #-> Factor is $factor
```

\ ⇨ **References**

```
$sequence = \$dnaseq;
                               $$sequence eq $dnaseq
                             ▶ ${$sequence} eq $dnaseq
```

Working with String $calars (I)

```
$a = "to bit or not to bit\n";
$b = "to bit or not to bit";
```

chop scalar

```
$c = chop($a); # $a eq "to bit or not to bit\n" && $c eq "\n"
$c = chop($b); # $b eq "to bit or not to bi"    && $c eq "t"
```

chomp scalar

```
chomp($a); # $a eq "to bit or not to bit"
chomp($b); # $b eq "to bit or not to bit"
```

reverse scalar

```
$c = reverse $b; # $c eq "tib ot ton ro tib ot"
```

substr scalar, offset, length, replacement

```
$c = substr($b, 0, 6);      # $c eq "to bit"
$d = substr($b, -10);       # $d eq "not to bit"
substr($b,6,0) = ".$c.";    # $b eq "to bit.to bit. or not to bit"
substr($b,6,8,".$c.");      # $b eq "to bit.to bit.to bit"
substr($b,6) = ", $c, $c"; # $b eq "to bit, to bit, to bit"
```

Working with String $calars (II)

```
$a = "to bit or not to bit";
```

split pattern, scalar

```
@A = split //,     $a; # @A = ("t", "o", "␣", "b", "i", "t", ... )
@B = split /\s+/, $a; # @B = ("to", "bit", "or", "not", "to", "bit")
```

join string, list_of_scalars

```
$b = join('',  @A); # $b = "to bit or not to bit";
$b = join('␣', @B); # $b = "to bit or not to bit";
```

Concatenation operators: **.** **.=**

```
$b1 = "to bit"; $b2 = "not ".$b1;
$c = $b1." or ".$b2; # $c  eq $a
$b1 .= " or ".$b2;   # $b1 eq $a
```

Repetition operators: **x** **x=**

```
$c = '=' x 10; # $c eq '=========='
$c .= ':';     # $c eq '==========:'
$c x= 4;       # $c eq '==========:==========:==========:==========:'
```

String Quoting Mechanisms

Single quotes: `$str_s = 'Factor is $factor';`

Double quotes: `$str_d = "Factor is $factor";`

Here documents:

```
$str_d = <<EOS; # EOS eq "EOS", ne 'EOS'
 Factor is $factor
EOS
# $factor = 0.005 => $str_d eq 'Factor is 0.005\n'
```

Single quoting: `$str_s =  q(Factor is $factor);`

Double quoting: `$str_d = qq(Factor is $factor);`

Quoting words:

```
@A = qw/ A C G T /;
   # @A = ("A", "C", "G", "T");
```

Case Conversion

```
$dna_seq = "atgggta";
```

Uppercase → Lowercase:

```
$DNASEQ = uc($dna_seq); # $DNASEQ eq "ATGGGTA"
```

Lowercase → Uppercase:

```
$dnaseq = lc($DNASEQ);  # $dnaseq eq "atgggta"
```

First Character:

```
$DNAseq = ucfirst($dnaseq); # $DNAseq eq "Atgggta"
$dnaSEQ = lcfirst($DNASEQ); # $dnaSEQ eq "aTGGGTA"
```

Case conversion within a string:

```
"\U...\E" ~ uc(...)             "\L...\E" ~ lc(...)
"\u...\E" ~ uc(...)             "\l...\E" ~ lcfirst(...)
print "Sequence [\u\L$DNASEQ\E]\n"; #-> "Sequence [Atgggta]"
```

Transliteration:

```
($dnaseq = $DNASEQ) =~ tr/A-Z/a-z/; # $dnaseq eq "atgggta"
```

Working with Numeric $calars

Numeric operators:

Addition	+	+=	++
Substraction	-	-=	--
Multiplication	*	*=	
Division	/	/=	
Modulus	%	%=	
Exponentiation	**	**=	

Numeric Context

- If a number is used as a string, the conversion is straight forward.
  `853 → "853"`
- If a string is used as a number, Perl will convert the string based on the first character(s):
  - If first character is numeric (ie, number, period (decimal), or negative (hyphen)), converted number reads from start to first non-numeric character.
    `"-534.4ab32" → -534.4`
  - If first character is non-numeric, converted number is 0.
    `"a4332.5" → 0`
  - Force numeric context by adding 0.

```
$x = "123x";
$y = +$x; $z = $x + 0;
print "y => $y, z => $z\n";
    # output is: y => 123x, z => 123
```

- If a scalar is used in a conditional (`if`, `while`), it is treated as a boolean value.

Numeric functions

abs EXPR	absolute value of EXPR
int EXPR	integer portion of EXPR
sqrt EXPR	square root of EXPR
log EXPR	natural logarithm (base e)
exp EXPR	e to the power of EXPR
sin EXPR	sine of EXPR in radians
cos EXPR	cosine of EXPR in radians
atan2 Y/X	arctangent of Y/X **(**-p .. +p**)**
srand [EXPR]	sets seed value for pseudo-random number generation
rand [EXPR]	returns a pseudo-random floating point value in the range of 0 up to EXPR

Comparison Operators

Numeric Comparisons		String Comparisons
==	equal	eq
!=	not equal	ne
<	less than	lt
<=	less than or equal	le
>	greater than	gt
>=	greater than or equal	ge
<=>	comparison -1 / 0 / 1	cmp

Logical Operators

&&
and

A	B	TEST
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

||
or

A	B	TEST
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

xor

A	B	TEST
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

!
not

Operand	TEST
TRUE	FALSE
FALSE	TRUE

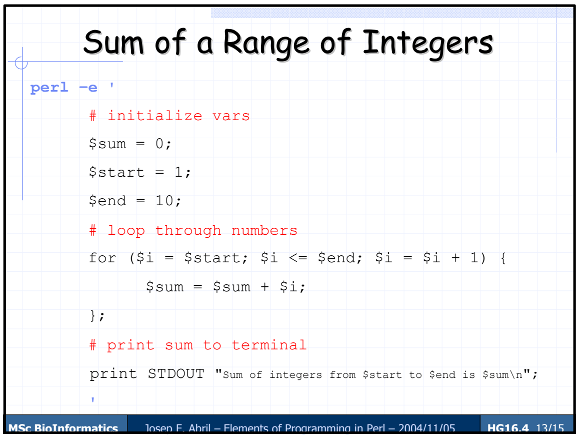
More on Operators: `http://www.perl.com/doc/manual/html/pod/perlop.html`

Default Input/Output Streams

```
perl -e '
    print STDERR "Running test...\n";
    while (<STDIN>) {
        chomp;
        ($_ eq "quit") && last;
        print STDOUT "$_\n";
        print STDERR "Read: $_\n";
    };
    print STDERR "Test finished...\n";
  '  <SHELL_INPUT  >SHELL_OUTPUT  2>SHELL_ERROR
```

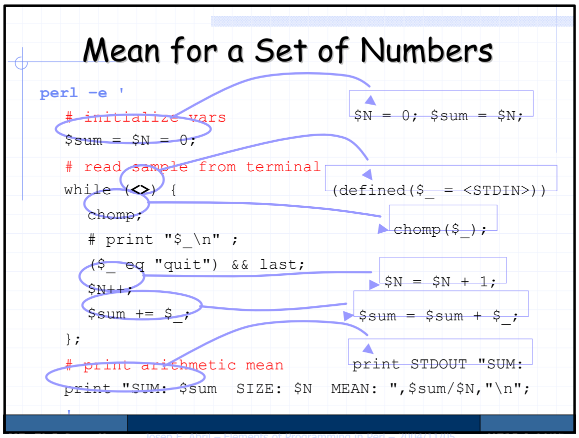
The **Diamond** operator: **`<>`** is a shortcut for **`<STDIN>`**

```
$line = <>;  # scalar context -> reading a single record
@lines = <>; # list context -> reading whole input records
($first,$second,@lines,$last) = <>; # $last eq ''
$last = pop @lines; # $last eq $lines[$#lines]
```



Sum of a Range of Integers

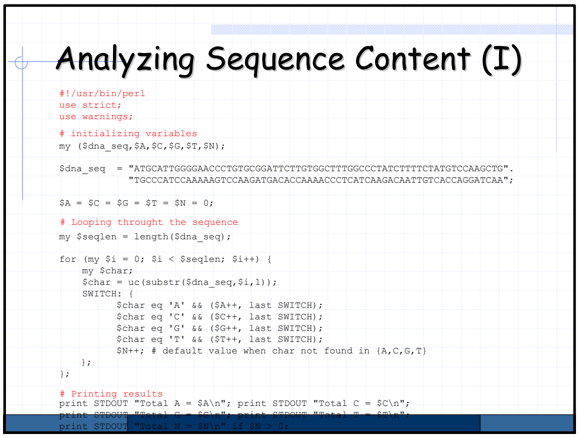
```
perl -e '
        # initialize vars
        $sum = 0;
        $start = 1;
        $end = 10;
        # loop through numbers
        for ($i = $start; $i <= $end; $i = $i + 1) {
                $sum = $sum + $i;
        };
        # print sum to terminal
        print STDOUT "Sum of integers from $start to $end is $sum\n";
        '
```



Mean for a Set of Numbers

```
perl -e '
    # initialize vars
    $sum = $N = 0;
    # read sample from terminal
    while (<>) {
        chomp;
        # print "$_\n" ;
        ($_ eq "quit") && last;
        $N++;
        $sum += $_;
    };
    # print arithmetic mean
    print "SUM: $sum  SIZE: $N  MEAN: ",$sum/$N,"\n";
    '
```

```
$N = 0; $sum = $N;
(defined($_ = <STDIN>))
chomp($_);
$N = $N + 1;
$sum = $sum + $_;
print STDOUT "SUM:
```



Analyzing Sequence Content (I)

```
#!/usr/bin/perl
use strict;
use warnings;
# initializing variables
my ($dna_seq,$A,$C,$G,$T,$N);

$dna_seq  = "ATGCATTGGGGAACCCTGTGCGGATTCTTGTGGCTTTGGCCCTATCTTTTCTATGTCCAAGCTG".
            "TGCCCATCCAAAAAGTCCAAGATGACACCAAAACCCTCATCAAGACAATTGTCACCAGGATCAA";

$A = $C = $G = $T = $N = 0;

# Looping throught the sequence
my $seqlen = length($dna_seq);

for (my $i = 0; $i < $seqlen; $i++) {
    my $char;
    $char = uc(substr($dna_seq,$i,1));
    SWITCH: {
          $char eq 'A' && ($A++, last SWITCH);
          $char eq 'C' && ($C++, last SWITCH);
          $char eq 'G' && ($G++, last SWITCH);
          $char eq 'T' && ($T++, last SWITCH);
          $N++; # default value when char not found in {A,C,G,T}
    };
};

# Printing results
print STDOUT "Total A = $A\n"; print STDOUT "Total C = $C\n";
print STDOUT "Total G = $G\n"; print STDOUT "Total T = $T\n";
print STDOUT "Total N = $N\n" if $N > 0;
```