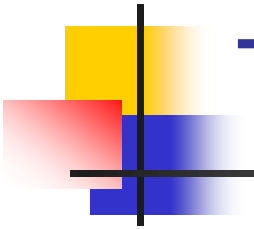


Topics

- Logical expressions and Conditional statements
- string functions: substr and index
- hashes

Logical Value of Expressions

- Any expression in Perl can be interpreted as a logical value
  - true or false
- Scalar context:
  - false if 0, "", or undefined
  - otherwise true

```
my $x; # or: my $x = 0;
if ($x) { $x++ }
else { $x = 17 }
print "$x\n";
17

my $x = 2;
while ($x) {
    print $x--, "\n";
}
2
1
```

- List context:
  - false if () or undefined
  - true otherwise

```
my @a = ("A", "T");
while (@a) {
     print shift @a, "\n";
}
A
T
```

Logical Operators

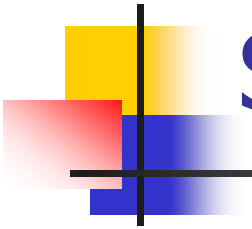
Logical operator "short-circuit": only evaluate second argument if necessary

`$a and $b`	**`$a` if `$a` is false, `$b` otherwise**
`$a or $b`	**$a if `$a` is true, `$b` otherwise**
`not $a`	**true if `$a` is not true**
`$a xor $b`	**True if `$a` or `$b` is true, but not both**

Example:

```
open(GRADES, "grades") or die "Can't open file grades\n";
```

See Wall (p 109-110) for discussion of C-style `&&`, `||` and `!` operators



String Functions: `substr`

substr *EXPR, OFFSET, LENGTH, REPLACEMENT*
Return substring of string *EXPR* at position *OFFSET* and length *LENGTH*

```
$s = "hello, world!";
$x = substr $s, 1, 1;              # $x = "e"
$x = substr $s, 0, 3;              # $x = "hel"
$x = substr $s, 7;                 # $x = "world!"

# The substring is replaced by REPLACEMENT if used:
substr($s,0,5,"goodbye"); # $s = "goodbye, world!";

# This does the same thing:
$s = "hello, world!";
substr($s,0,5) = "goodbye";
```

Note: Predefined Perl functions may be used with or without parentheses around their arguments

String Functions: `index`

```
index STR, SUBSTR, OFFSET
    Return the position of the first occurrence of SUBSTR in STR; If OFFSET
    is given, skip this many letters before looking
    Returns -1 if SUBSTR not found

$dna = "GATGCCATGAAATGC";
$pos = index $dna, "ATG";
print "ATG found at position $pos\n";      # answer: 1

pos = -1;
while (($pos = index($dna, "ATG", $pos)) > -1) {
    print "ATG found at position $pos\n";
    $pos++;
}
OUTPUT:
ATG found at position 1
ATG found at position 6
ATG found at position 11
```

Hashes (Associative Arrays)

- A Hash is a collection of zero or more pairs of scalar values, called keys and values

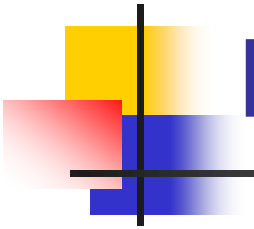
- Hash variable names begin with a percent sign (`%`)

```
%genes = ( "gene1", "ATTCGT", "gene2", "CTGCCATGA");
```

- The *values* are indexed by the *keys*
  - Given a key, the hash returns the corresponding value

```
$seq = $genes{"gene2"};  #  $seq = "CTGCCATGA"
```

  - Note that `$genes{"gene2"}` is a scalar, so it starts with `$`



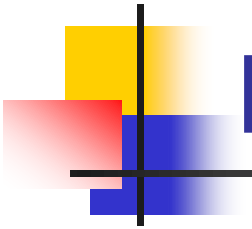
Hashes

- Hashes can be assigned values use key=>value notation:

```
%genes = ( "gene1", "ATTCGT", "gene2", "CTGCCATGA");
%genes = ( gene1=>"ATTCGT", gene2=>"CTGCCATGA");
```

- Hash elements can be created/altered by assignment statements:

```
$genes{"gene1"} = "ATTCGT";
$genes{gene2} = "CTGCCATGA";  # note: no quotes in key
```



Hashes (Associative Arrays)

```
%genomes = ( );   # creates an empty hash

# two ways to do the same thing:
%genomes = ( "virus", 31, "bacteria", 89, "plants", 5 );
%genomes = ( virus => 31, bacteria => 89, plants => 5 );

$genomes{mammals} = 2;  # adds a new pair to the hash

@genome_list = keys %genomes;
# @genome list is now ("plants" , "mammals", "bacteria", "virus")

@genome_counts = values %genomes;
# @genome_counts is now (5, 2, 89, 31)

# keys and values are not guaranteed to return the data is same order
# as it was entered, but they are guaranteed to return the data in the
# same order as each other.
```

Hashes

The **keys** function returns a list of all keys in a hash (in some random order)

```
%genes = (gene2=>"CTGCCATGA", gene1=>"ATTCGT");
@key_list = keys(%genes);
print "@key_list\n";   # prints: gene1 gene2

# often used to loop through a hash:
foreach $key (@key_list) {
   print "The value of $key is $genes{$key}\n";
}
```

Output:

```
The value of gene1 is ATTCGT
The value of gene2 is CTGCCATGA
```

Hashes (Associative Arrays)

- The **each** function returns a two-element list containing one key from the hash and its associated value
- Subsequent calls to **each** will return another pair, until all pairs have been returned (at which point an empty array is returned)

```
while ( ($genome, $count) = each %genomes ) ) {
     print "$genome $count\n";
}

OUTPUT: (possibly not in this order)
plants 5
virus 31
bacteria 89
mammals 2
```

More on Hashes (Associative Arrays)

- Assigning the return value from **values** or **keys** to a scalar gives the number of pairs in a hash:

```
$genome_count = keys %genomes;  #  $genome_count is now 4
$genome_count = values %genomes;  #  $genome_count is now 4
```

- The **delete** function removes a pair from a hash

```
delete $genomes{bacteria};
$genome_count = keys %genomes;  #  $genome_count is now 3
```

Hashes

```
% cat testfile2
a text file with lots of words
some words occur once and some
words occur more than once

% wordcount.pl testfile2
a 1
and 1
file 1
lots 1
more 1
occur 2
of 1
once 2
some 2
text 1
than 1
with 1
words 3
```