Arrays (Lists)

An *array* is a sequence of scalars, indexed by position (0,1,2,...)
The whole array is denoted by `@array`
Individual array elements are denoted by $array[index]
$#array gives the *index of the last element*.
Example: *a*[0] = "*firststring*";a[1] = "2nd string"; *a*[2] = 123;
or, equivalently,
@a = ("first string", "2nd string", 123);
print "Index of last element is *a*"; *print*" *Numberofelementsis*",a+1,
"";

Arrays (Lists)

@a = ("abc", 123, 'x');

numeric context ... gives list length $n = $@$a$;n == 3

string context ... gives space-separated elems $s = $"@$a$";s eq "abc 123 x"

scalar context ... gives list length $t = $@$a$."";t eq "3"

print context ... gives joined elems print @a; displays "abc123x"

In Perl, interpretation is context-dependent.

Arrays (Lists)

Arrays do not need to be declared, and they grow and shrink as needed.
"Missing" elements are interpolated, e.g. $abc[0] = "abc"$;abc[2] = "xyz"; reference to $abc[1] returns""$
Can assign *to* a whole array; can assign *from* a whole array, e.g.
@numbers = (4, 12, 5, 7, 2, 9); (a,b, c,d) = @numbers;
Since assignment of list elements happens in parallel ... (x,y) = (y,x); swaps values of x,y

Arrays (Lists)

Array *slices*, e.g. @list = (1, 3, 5, 7, 9); print "@list[0,2]"; displays "1 5" print "@list[0..2]"; displays "1 3 5" print "@list[4,2,3]"; displays "9 5 7" print "@list[0..9]"; displays "1 3 5 7 9"

Array values interpolated into array literals: @a = (3, 5, 7); @b = @a; @b = (3,5,7); @c = (1, @a, 9); @c = (1,3,5,7,9); @a == (@a) == ((@a)) ...

Arrays (Lists)

Arrays can be accessed element-at-a-time using the `for` loop:
@nums = (23, 95, 33, 42, 17, 87); $sum = 0; for$(i = 0;
$i <$ @$nums$;i++)   @nums gives length $sum+ =$nums[i];sum = 0;
foreach num(@$nums$)$sum+ =$num;
`push` and `pop` act on the "right-hand" end of an array: Value of
@a @a = (1,3,5); (1,3,5) push @a, 7; (1,3,5,7)
$x = pop$@$a; (1, 3, 5, 7)$,x == 7 $y = pop$@$a; (1, 3, 5)$,y == 5

Arrays (Lists)

Other useful operations on arrays:

`@b = sort(@a)`	returns sorted version of `@a`
`@b = reverse(@a)`	returns reversed version of `@a`
`shift(@a)`	like `pop(@a)`, but from left-hand end
`unshift(@a,x)`	like `push(@a,x)`, but at left-hand end

Lists as Strings

Recall the marks example from earlier on; we used `"54,67,88"` to effectively hold a list of marks.

Could we turn this into a real list if e.g. we wanted to compute an average?

The *split* operation allows us to do this:

Syntax: `split(/`*pattern*`/,`*string*`)` returns a list

The *join* operation allows us to convert from list to string:

Syntax: `join(`*string*`,`*list*`)` returns a string

(Don't confuse this with the `join` filter in the shell. Perl's `join` acts more like `paste`.)

Lists as Strings

Examples: $marks = "99, 67, 85, 48, 77, 84";$
@listOfMarks = split(/,/,
$marks); assigns (99, 67, 85, 48, 77, 84) to @listOfMarks$
$sum = 0; foreach$m (@listOfMarks) $sum + =$m;
$newMarks = join(':', @listOfMarks); assigns$"99 : 67 : 85 : 48 : 77 :
84" tonewMarks

Lists as Strings

Complex splits can be achieved by using a full regular expression rather than a single delimiter character.
If part of the regexp is parenthesised, the corresponding part of each delimiter is retained in the resulting list.
split(/[@]+/,'ab@cd@@e'); gives (ab,c,d,e)
split(/([@]+)/,'ab@cd@@e');gives (ab,@,c,,d,@@,e)
split(/([@])+/,'ab@cd@@e');gives (ab,,c,,d,@,e)
And as a specially useful case, the empty regexp is treated as if it matched between every character, splitting the string into a list of single characters:
split(//, 'hello'); gives (h, e, l, l, o)

Associative Arrays (Hashes)

As well as arrays indexed by numbers, Perl supports arrays indexed by strings: *hashes*.
Conceptually, as hash is a set (not list) of (*key*, *value*) pairs.
We can deal with an entire hash at a time via %*hashName*, e.g.
Key Value
"Mon" =¿ "Monday", "Tue" =¿ "Tuesday", "Wed" =¿ "Wednesday", "Thu" =¿ "Thursday", "Fri" =¿ "Friday", "Sat" =¿ "Saturday");

Associative Arrays (Hashes)

Individual components of a hash are accessed via
$\$ hashName\{keyString\}$
Examples: $days"Sun"returns"Sunday"$ days"Fri" returns "Friday"
$days"dog"isundefined(interpretedas"")$days0 is undefined
(interpreted as "")
inserts a new (key,value)
$daysdog = "DogDayAfternoon"; barewordOKaskey$
replaces value for key "Sun"
$days"Sun" = Soonday; barewordOKasvalue$

Associative Arrays (Hashes)

Consider the following two assignments: @f = ("John", "blue", "Anne", "red", "Tim", "pink");

The first produces an array of strings that can be accessed via position, such as `$f[0]`

The second produces a lookup table of names and colours, e.g. `$g{"Tim"}`.

(In fact the symbols => and comma have identical meaning in a list, so either right-hand side could have been used. However, always use the arrow form exclusively for hashes.)

Associative Arrays (Hashes)

Consider iterating over each of these data structures:

foreach $x(@f)print$"x";
John blue Anne red Tim pink

foreach $x(keysprint$"x =¿ gx";
Anne =¿ red Tim =¿ pink John =¿ blue

The data comes out of the hash in a fixed but arbitrary order (due to the hash function).

Associative Arrays (Hashes)

There are several ways to examine the $(key, value)$ pairs in a hash: foreach $key(keysprint$"(key, $myHash$key)";
or, if you just want the values without the keys foreach $val(valuesprint$"(?,val)";
or, if you want them both together while ((key,val) = each print "(key,val)";
Note that each method produces the keys/values in the same order. It's illegal to change the hash within these loops.

Associative Arrays (Hashes)

Example (collecting marks for each student):

- a data file of (*name*, *mark*) pairs, space-separated, one per line
- out should be (*name*, *marksList*), with comma-separated marks

while (¡¿) chomp; remove newline (*name*,mark) = split; separate data fields *marks*name .= ",*mark*"; *accumulatemarksforeach*name (keys *marks*name = s/,//; remove comma prefix print "*name*marks*name*";

Associative Arrays (Hashes)

The `delete` function removes an entry (or entries) from an associative array.
To remove a single pair: delete *days*"*Mon*";"*Idon′tlikeMondays*"
To remove multiple pairs: delete @days ("Sat","Sun") ; Oh noes - no weekend!
To clean out the entire hash: foreach *d*(*keysor*, *moresimply*