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] = "first string";
$a[1] = "2nd string";
$a[2] = 123;

# or, equivalently,
@a = ("first string", "2nd string", 123);

print "Index of last element is $#a\n";
print "Number of elements is ", $#a+1, "\n";
Arrays (Lists)

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

# scalar 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"

# list 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.

```perl
$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.

```perl
@numbers = (4, 12, 5, 7, 2, 9);
($a, $b, $c, $d) = @numbers;
```

Since assignment of list elements happens in parallel …

```perl
($x, $y) = ($y, $x);  # swaps values of $x, $y
```
Arrays (Lists)

Array **slices**, e.g.

```perl
@list = (1, 3, 5, 7, 9);
print "@list[0,2]\n";    # displays "1 5"
print "@list[0..2]\n";   # displays "1 3 5"
print "@list[4,2,3]\n";  # displays "9 5 7"
print "@list[0..9]\n";   # displays "1 3 5 7 9"
```

Array values interpolated into array literals:

```perl
@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:

```perl
@nums = (23, 95, 33, 42, 17, 87);
$sum = 0;
# @nums in scalar context gives length
for ($i = 0; $i < @nums; $i++) {
    $sum += $nums[$i];
}
$sum = 0;
foreach $num (@nums) { sum += $num; }
```

`push` and `pop` act on the “right-hand” end of an array:

```perl
@a = (1,3,5);  # Value of @a
push @a, 7;   # (1,3,5)
$x = pop @a;  # (1,3,5,7), $x == 7
$y = pop @a;  # (1,3,5), $y == 5
```
Arrays (Lists)

Other useful operations on arrays:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@b = sort @a</td>
<td>returns sorted version of @a</td>
</tr>
<tr>
<td>@b = reverse @a</td>
<td>returns reversed version of @a</td>
</tr>
<tr>
<td>shift @a</td>
<td>like pop(@a), but from left-hand end</td>
</tr>
<tr>
<td>unshift @a, x</td>
<td>like push(@a, x), but at left-hand end</td>
</tr>
</tbody>
</table>
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` function allows us to do this:

\[
\text{split}(\text{/pattern/}, \text{string}) \quad \text{returns a list}
\]

The `join` function allows us to convert from list to string:

\[
\text{join}(\text{string}, \text{list}) \quad \text{returns a string}
\]

- Don’t confuse this with the `join` filter in the shell.
  - Perl’s `join` acts more like `paste`
Lists as Strings

Examples:

```perl
$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" to $newMarks
```
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.

```javascript
# returns (ab,c,d,e)
split(/[#@]+/, 'ab##@#c#d@@e');
# returns (ab,##@#,c,#,d,@@,e)
split(/(#@+)/, 'ab##@#c#d@@e');
# returns (ab,#,c,#,d,@,e)
split(/(#@)+/, '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:

```javascript
# returns (h, e, l, l, o)
split(//, 'hello');
```
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.

```perl
%days = (  "Sun" => "Sunday",
           "Mon" => "Monday",
           "Tue" => "Tuesday",
           "Wed" => "Wednesday",
           "Thu" => "Thursday",
           "Fri" => "Friday",
           "Sat" => "Saturday" );
```
Individual components of a hash are accessed via `\$hashName\{keyString\}`

Examples:

```perl
$days{"Sun"}   # returns "Sunday"
$days{"Fri"}   # returns "Friday"
$days{"dog"}   # is undefined (interpreted as "")
$days{0}       # is undefined (interpreted as "")

# inserts a new (key,value)
$days{"dog"} = "Dog Day Afternoon";

# replaces value for key "Sun"
$days{"Sun"} = "Soonday";
```
Consider the following two assignments:

```perl
@f = ("John", "blue", "Anne", "red", "Tim", "pink");
"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.
Consider iterating over each of these data structures:

```perl
foreach $x (@f) {
    print "$x
";
}
```

John
blue
Anne
red
Tim
pink

```perl
foreach $x (keys %g) {
    print "$x => $g{$x}
";
}
```

Anne => red
Tim => pink
John => blue

- The data comes out of the hash in arbitrary order.
- This order changes with each execution to make security exploits harder.
- use sort if you need a fixed order
There are several ways to examine the \((key, value)\) pairs in a hash:

```perl
foreach $key (keys %myHash) {
    print "($key, $myHash{$key})\n";
}
```

or, if you just want the values without the keys

```perl
foreach $val (values %myHash) {
    print "(?, $val)\n";
}
```

or, if you want them both together

```perl
while (($key,$val) = each %myHash) {
    print "($key, $val)\n";
}
```

- Note that each method produces the keys/values in the same order.
- It’s illegal to change the hash within these loops.
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

```perl
while (<>)
{
    chomp;       # remove newline
    ($name, $mark) = split;   # separate data fields
    $marks{$name} .= ",$mark"; # accumulate marks
}

foreach $name (keys %marks)
{
    $marks{$name} =~ s/,//;    # remove comma prefix
    print "$name $marks{$name}\n";
}
```
The `delete` function removes an entry (or entries) from an associative array.

```perl
# to remove a single pair:
delete $days{"Mon"};  # "I don't like Mondays"

# to remove multiple pairs:
delete @days{ ("Sat","Sun") };  # No weekend!

# to clean out the entire hash:
foreach $d (keys %days) {
    delete $days{$d};
}

# or, more simply
```