Passing two arrays to a function

sub @_ \ ARRAY

Passing two scalars to a function, and accepting them inside the function is easy. What happens if you try to pass two arrays to a function as in this example `f(@a, @b)`?

Perl will flatten and unite the contents of the two arrays, and inside the function you won't be able to tell where does the first end, and where does the second start. `@_`, the array of parameters will be just one long list of values.

Let's see a very simple example:

Adding two numbers

The simple subroutine adding two numbers work well.

```perl
#!/usr/bin/perl
use strict;
use warnings;

sub add {
    my ($x, $y) = @_;
    return $x+$y;
}

print add(2, 3), "\n";
```

Adding two vectors

What if you would like to create another subroutine that would accept two arrays and add the values pair-wise:

(2, 3) + (7, 8, 5) = (9, 11, 5)

```perl
my @first  = (2, 3);
my @second = (7, 8, 5);
```

```
add(@first, @second);

sub add {
    print "@_\n";  # 2 3 7 8 5
    ....
}
```

Unfortunately, inside the subroutine `@_` will hold the list of all the values in one flat array.

Array References

That's one of the major uses of references in Perl: Passing complex data structures to subroutines.

If you have an array called `@names`, you can get a reference to his array by preceding it with a back-slash: `\@names`. You can assign this reference to a scalar variable: `my $names_ref = \@names;`. (I only use the _ref to make it cleared in this article. Usually you would not use such names.)

If you try to print the content of this new variable: `print $names_ref;` you will get an output like this: **ARRAY(0x703dcf2)**. You don't have much to do with this string, but if you see such output from a code, you know, someone has forgotten to **de-reference** an array. (De-referencing is the word to use when you want to get back the real something from a reference of the real something.)

About the only legitimate thing you can do with the reference is to de-reference it, to get back the original content of the array. For this you'd put a `@` in-front of the reference: `@$names_ref`.

For better readability you might want to add a pair of curly braces around the variable like this: `@{$names_ref}`, or even with spaces like this: `@{ $names_ref }`.

Check out this examples:

```
#!/usr/bin/perl
use strict;
use warnings;

my @names = qw(Foo Bar Baz);

my $names_ref  = \@names;
print "$names_ref\n";          # ARRAY(0x703dcf2)

print "@$names_ref\n";         # Foo Bar Baz
print "@{ $names_ref }\n";     # Foo Bar Baz
```

Passing two array references

The solution then to the original problem is to pass two references of the two arrays:

```perl
#!/usr/bin/perl
use strict;
use warnings;

my @first  = (2, 3);
my @second = (7, 8, 5);
add(\@first, \@second);  # passing two references

sub add {
    my ($one_ref, $two_ref) = @_;
    my @one = @{ $one_ref };       # dereferencing and copying each array
    my @two = @{ $two_ref };
```