

Perl Arrays

How to use an array in Perl

Page Resource

[CGI Resources](#) | [Perl Tutorials](#)

[Home](#)/[CGI](#)/[Perl](#)/[Arrays](#)

Now it is time to get into some more fun, this section is on using arrays. An array is basically a way to store a whole bunch of values under one name. These values usually have something in common, and the array makes the values easier to access and manipulate- especially if you are using loops.

Define an Array

Let's take a look at how an array is defined in Perl:

```
@arrayname = ("element1", "element2");
```

Notice that an array name begins with an "@" symbol. This is how Perl know it is an array and not something else. You can have as many elements as you need, I just listed two above to keep from running on. The elements can be numbers or strings depending on what you need. If you use plain numbers, the string quotes wouldn't be needed here, for example:

```
@one_two_three = (1, 2, 3);
```

This would define an array named @one_two_three with three elements: the numbers 1, 2, and 3. Since the elements are only numbers, they need no quotes around them. On the other hand, if you wanted an array of string values, you'll need the quotes so the interpreter sees the elements as strings:

```
@browser = ("NS", "IE", "Opera");
```

This time we have an array named @browser with three string elements: NS, IE and Opera. The question now is how do we make use of these elements later?

How to Access Array Elements

Suppose we wanted to grab the element "IE" out of the browser array above. You'll notice "IE" is the second element in the array, but look at the code we will use to get it carefully:

```
$browser[1]
```

You'll see that we get the element by using a regular variable (the \$ sign) with the same name as the array (browser). Then we use brackets with a number to get to a certain element within the array. Each array is ordered, but not starting at the number one like we would like. Arrays instead begin ordering at the number zero! So, the first element of an array is always accessed using:

```
$arrayname[0]
```

So, this is why we used `$browser[1]` to get "IE" above. It is the second element, but its list number is 1. So, be sure to remember this when working with arrays. They start the number list at zero, so be careful when you try to get an element or you might have a hard time figuring out where things went wrong in your script (believe me, I have done this a few times).

In a similar way, we can get the other elements for the array and use them. Let's say we wanted to write a little sentence about the browsers. We could use the array elements as part of the sentence:

```
print "I like to use $browser[0], $browser[1], and sometimes $browser[2].";
```

Yes, a pretty simple sentence indeed- but here is the result:

```
I like to use NS, IE, and sometimes Opera.
```

Great, but not much more useful to us than 3 variables. But if you use a loop, an array can be very useful either to save typing or to manipulate a set of similar values.

Arrays and Loops

This is where an array can be more powerful, if you want to print the three browser names you could now use a loop rather than a separate print line with variable names. This is really handy if the list is a long one, but to keep things short we'll stick to our three element browser array. The following could be used to print the three browser names, one per line:

```
@browser = ("NS", "IE", "Opera");

print "My Favorite Browsers List:\n\n";

foreach $browser (@browser)
{
    print "$browser\n";
}
```

Notice how we used the foreach loop we looked at in the last section. As you can see, it is very handy with arrays. It loops through enough times to grab each element in `@browser` and print it. Also, using the foreach loop, we didn't have to add the brackets to access the array element. You'll notice we just used `$browser` inside the loop. This is because the foreach loop knows which elements to grab- all of them. So it gives back the next element in order each time.

In the next section, we'll have even more fun with arrays. Until then, enjoy creating your own arrays!

Well, that's it for now. Let's go on to: [Manipulating Arrays](#).

Partners

[CoolHomepages](#) | [Web Design Library](#) | [Website Content](#)

The tutorials and articles on these pages are © 1997-2005 by John Pollock and may not be reposted without written permission from the author, and may not be reprinted for profit.



[Previous](#)

By: [John Pollock](#)



[Next](#)

[Main Page](#) | [HTML](#) | [JavaScript](#) | [Graphics](#) | [DHTML/Style Sheets](#) | [ASP/PHP](#)
[PutWeb/FTP](#) | [CGI/Perl](#) | [Promotion](#) | [Java](#) | [Design Articles](#)
[Support Forums](#) | [Site Search](#) | [FAQs](#) | [Privacy](#) | [Contact](#)

Copyright © 1997-2005 [The Web Design Resource](#). All rights reserved.