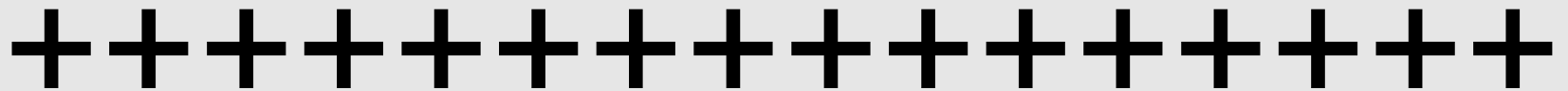




Lesson #5: CGI/Perl II



Environment variables

Scalar variables

Selection structures

Loops and arrays



Environment variables

- Environment variables are set each time a CGI script is run.
- They contain information about the Web server, its configuration and about the request made by the browser.
- Environment variables vary from server to server. They are stored in the %ENV hash.



HTTP_USER_AGENT

- The HTTP_USER_AGENT contains the name and version of the visitor's browser, the computer platform.

```
print $ENV{ 'HTTP_USER_AGENT' } ;
```

Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;
SV1)

Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.7) Gecko/20040803 Firefox/0.9.3



Viewing available variables

- Here a simple program that shows all the available environment variables on a server.

```
foreach (keys %ENV) {  
    print "Key: $_; Value:  
    $ENV{$_}";  
}
```

- See it in action here:

<http://www2.scs.ryerson.ca/~dhamelin/cgi-bin/env.cgi>



Getting single-valued form data

- The form (saved in public_html directory):

```
<form action="cgi-bin/name.cgi" method="post">
```

```
Name: <input type="text" name="name" size="35" />
```

```
<input type="submit" />
```

```
</form>
```



Getting single-valued form data

- The name.cgi script (saved in cgi-bin directory with 755 protection):

```
#!/usr/bin/perl
use CGI ':standard';
print "Content-type:
text/html\n\n";

$name = param ('name');
print "Hello $name";
```



Getting multiple-valued form data

- The form (saved in public_html directory):

```
<form action="cgi-bin/ckbox.cgi"
  method="post">
```

```
Select your favorite dish (you may pick
  more than one:<br />
```

```
<input type="checkbox" name="dish"
  value="pizza" />Pizza
```

```
<input type="checkbox" name="dish"
  value="tandoori" />Tandoori Chicken
```

```
<input type="checkbox" name="dish"
  value="steak" />Steak
```

```
<input type="submit" />
```

```
</form>
```



Getting multiple-valued form data

- The ckbox.cgi script (saved in cgi-bin directory with 755 protection):

```
#!/usr/bin/perl
use CGI ':standard';
print "Content-type:
      text/html\n\n";

@food = param ('dish');
print "Your favorite dishes are:
      @food";
```



Getting the form's names

- The `param` function without arguments supplies an array containing the form items' names.

```
@names = param();
```

- To get the values as well, use a `foreach` loop.

```
foreach my $name (param()) {  
    my @values = param($name);
```



Scalars

- Here are examples of scalars:
- 'elephant'
- 3
- \$animal
- \$animal[2]
- \$animal{'habitat'}



Scalar Operators

- `=:` assignment operator.
- `+`, `-`, `*`, `/`, `%` are like in C.
- Operator precedence rules are similar to those in C.
- `**` is the exponent operator. It takes precedence over `*`, `/` and `%`.
- Dot (`.`) is the string concatenation operator.
- The math functions *sqrt*, *abs*, *int*, *atan2*, *cos*, *sin*, *log* and *exp* are available.
- Shortcuts like `$a+=3;` and `++$i` are permitted.



Comparison Operators

- `<`, `<=`, `>`, `>=`, `==` and `!=` are used to compare numbers.
- Strings use a different set of operators:
- **eq** for equal, **ne** for not equal, **gt** for greater than, **ge** for greater or equal, **lt** for less than and **le** for less or equal.
- A condition is true if not equal to zero, empty or undefined.



if statement

- Some examples of if statements:

```
if ($food eq "spinach")
{
    print "Good! You will be
strong and healthy!";
}
else
{
    print "You should eat your
spinach!";
}
```



if statement

```
if ($food eq "spinach") {  
    print "Good! You will be  
strong and healthy!";  
}  
elsif ($food eq "broccoli"){  
    print "Excellent! Broccoli  
is good for you!";  
}  
else {  
    print "You should eat more  
vegetables!";  
}
```



unless statement

- The unless statement is used for a false condition to simplify a program.

```
unless ($food eq "spinach")  
{  
    print "You should eat your  
    spinach!";  
}
```



foreach statement

- The `foreach` statement is used to loop through all the elements of an array.

```
foreach $creature (@prey)
{
    print "<div>$creature</div>";
}
```

- In the previous code, we take the first element of the `@prey` array, assign it to the `$creature` variable. We do that for all the elements of the array.



while statement

- The while statement repeats a block while the condition remains true.

```
$i = 1;  
while ($i <= 100) {  
  print "$i ";  
  ++$i;  
}
```



until statement

- The `until` statement repeats a block while the condition remains false or until it is true.

```
$i = 1;  
until ($i > 100) {  
  print "$i ";  
  ++$i;  
}
```



do-while statement

- The do-while statement repeats a block once before testing the condition.

```
$i = 1;  
do {  
    print "$i ";  
    ++$i;  
}while ($i <= 100);
```



for statement

- The for statement repeats a block a given number of times.

```
for ($i=1; $i<=100; ++$i)
{
    print "$i ";
}
```



The localtime function

- The **localtime** function returns the actual local time of the hosting server. It is very practical to store the result in an array. The **gmtime** function returns the Greenwich Mean Time (GMT).

```
@time = localtime;  
  
foreach $elem (@time) {  
    print "$elem ";  
}
```

```
59 56 11 4 9 106 3 276 1  
Sec min hour day mth year wday yday dst
```



Array items

- The loop on the previous page can be done by referencing individual elements:

```
@time = localtime;  
$i = 0;  
while ($i < @time) {  
print "$time[$i] ";  
++$i;  
}
```

Length of array `@time`
Can also use `$#time`,
the last subscript

```
59 56 11 4 9 106 3 276 1  
Sec min hour day mth year wday yday dst
```



Other array operations

- You can copy the first element of an array into a scalar.

```
($variable) = @array;
```

- Or the first two into 2 scalars:

```
($var1, $var2) = @array;
```

- Or the first two into scalars, the rest into another array.

```
($var1, $var2, @other) = @array;
```



Other array operations

- You can copy selected items into another array:

```
@choice = @array[2,5,$i];
```

- You can add an element to the end of the array:

```
push (@array, $new_element);
```

- You can add an element at the beginning of the array:

```
unshift (@array, $new_element);
```



Other array operations

- You can remove the last item of an array:

```
$removed = pop (@array);
```

- You can remove the first item of an array:

```
$removed = shift (@array);
```

- You can sort an array by ASCII order:

```
@array = sort (@array);
```

- You can reverse an array:

```
@array = reverse (@array);
```



Array sorting

- The simple sort function sorts by ASCII values. There are variations.
- To sort by reverse ASCII order:

```
@array = sort{$b cmp $a} (@array);
```

- To sort numerically in ascending order:

```
@array = sort{$a <=> $b} (@array);
```

- To sort numerically in descending order:

```
@array = sort{$b <=> $a} (@array)
```



End of lesson