# Why should the goto term be used sparingly?

The goto operator was introduced in PHP 5.3 It allows for jumping around within the same file. The target point is specified by a label and then colon, for example:

<?php
 goto process1;
 echo “Goto”;

 process1:
 echo “Example”;
?>

It should be noted that the goto operator is restricted to jumping within the current method or function. While it can provide additional flexibility, overuse can result in difficult to read code as a less logical flow would be followed.

# Write a script that emulates the function call str\_pad( $myString, $desiredLength).

 *In other words, take a string, and add space characters to the right of it until the string reaches the desired length. Display both the original and padded string in the page.*

The str\_pad function allows for the adding of space or padding to the left or right of a string. The example below demonstrates this concept using periods to pad the left and right of the string.

<html doctype=”html”>

 <head>

 <title>My String Padding Script</title>

 </head>

 <body>

 <h1>String Padding Example</h1>

 <?php

 $OriginalText = "This is our string variable";

 $PaddedText = str\_pad($OriginalText, 40, ".");

 $PaddedTextLeft = str\_pad($OriginalText, 40, ".", STR\_PAD\_LEFT );

 echo "<p><strong>Oiginal text:</strong> $OriginalText</p>";

 echo "<p><strong>Padded 40 on right:</strong> $PaddedText</p>";

 echo "<p><strong>Padded 40 on left:</strong> $PaddedTextLeft</p>";

 ?>

 </body>

</html>

**The above code can be seen running here: http://www.carnaghan.net/string.php**

# How does the weak typing of PHP enable more flexibility within the language and what does this mean for complex software applications?

PHP uses weak typing or loosely typed variables which means that it doesn’t require each variable set to a specific datatype upon initialization. In addition to this, PHP will dynamically change the data type of variables depending on their context. For example an integer could have a float value added to it, which would in turn convert it to a float data type. This is both advantageous and problematic.

One of the major benefits of loosely typed variables is flexibility. Other languages such as C# and Java only allow for strongly typed variables and therefore once a variable type is set, it must always remain of that type. PHP on the other hand can reuse variables in different situations that require different data types.

The disadvantage with the loosely typed approach is that PHP will automatically do the conversions and this can lead to a situation where a potential wrong type is being passed to a function or method expecting something different. These types of errors can be difficult to track down since variables can change depending on the context of the process.

# What is the difference between $variable = 1 and $variable == 1? Why is this an important distinction to understand?

In PHP the equals = operator is used to set a variable value. For example $variable1 = 1 would set the value 1 to variable1. The double equals == is a comparison operator which is used to see if two variables contain the same value. In PHP if $variable1 == $variable2, true is returned, likewise if they do not match false is returned. The comparison operator is used to determine if two variables match in the example below:

<?php

 $variable1 = 1;

 $variable2 = 1;

 if ($variable1 == $variable2)

 {

 echo "The variables match";

 }

 else

 {

 echo "The variables do not match";

 }

?>

In this code I have set variable1 to 1 and variable2 to 1. Since they are both the same value, the code will display “The variables match”. It is important to draw this distinction because if the == was switched out for an = the statement $variable1 = $variable2 would set the value of variable1 to that of variable2.

In PHP there is another comparison operator === which not only checks if the value is the same, but also checks if the data types match. For example if variable1 contained a string “1” and variable2 contained an integer 1, the comparison would return false.

# How do you decide when to use an if statement vs. a loop vs. a switch statement and why?

If statements (and switch statements) are generally used for decision making, while loops are used to process repetitive tasks such as grabbing and setting values for a number of items. If statements can be written using the basic if, elseif, else format or by using the switch statement. I created an example that demonstrates very basic user authentication on a hypothetical blogging or article based system. The example assumes that prior access levels have been set and once the user logs into the system a check is made to authenticate system privileges as an admin, editor or reader.

if ($access\_level == “admin”)
 {
 set $grant\_access = “1”; // set access level privileges to administrator
 }
elseif ($access\_level == “editor”)
 {
 set $grant\_access = “2”; // set access level privileges to editor
 }
elseif ($access\_level == “reader”)
 {
 set $grant\_access = “3”; // set access level to read only
 }
else
 {
 set $grant\_access = “0”; // do not allow access to editor or administrator functions
 }

If else is fine if there are only a few values and outcomes, however in situations where there are a greater number of decisions, a switch statement would be a better choice. As you can see from the same scenario using a switch statement, the code is much cleaner and if there were a greater number of different outcomes, it would be a better choice:

switch ( $access\_level )
{
 case “admin”:
 set $grant\_access = “1”; // set access level privileges to administrator
 break;
 case “admin”:
 set $grant\_access = “1”; // set access level privileges to administrator
 break;
 case “admin”:
 set $grant\_access = “1”; // set access level privileges to administrator
 break;
 default:
 set $grant\_access = “0”; // do not allow access to editor or administrator functions
 break;
}

It should be noted that once PHP finds an access level that matches, it will not only process the code in the current switch statement, but it will continue to process the rest of the cases. This is why I have included breaks at the end of each statement, which tells PHP to stop and leave the switch once one of the decisions has been processed.

Loops allow for repeating of tasks based on a condition. For example if I wanted to list all available records in a database I could use a loop to grab each record and display it to the user. There are three main types of loops which include while, do while and for. The simplest type of loop is the while loop, which runs a piece of code repeatedly while some condition is true. A while loop requires a condition change to be set within the loop so it doesn’t repeat indefinably. For example the while loop could check if number of items left is greater than zero and inside the loop the number of items would need to be decremented each time. A while loop checks the condition first and then processes the code. A ‘do while’ loop on the other hand processes the loop and then checks the condition. A for loop allows for cleaner code by requiring an initializer, loop test and count within the for statement. This means that no further incrementing / decrementing is required inside the loop. Some basic examples follow:

*While Loop*

while ( $items > 0 )
{
 // code
 $items --;
}

*Do While Loop*

do
{
 // code & counting
}
while {expression}

*For Loop*

for ( expression1; expression2; expression3 ) // initializer, loop test, count
{
 // code
}

Loops are used when a number of items need to be processed as opposed to decision making processes using if and switch statements. Both are commonly used together, for example you may need to loop over a number of items which need to be checked for validity using an if statement.

As I mentioned earlier, if statements are commonly used for decisions PHP must make based on a current set of variables or values. In my earlier example I demonstrated how an if statement could be used to authenticate a user. Loops are commonly used to process a single piece of code multiple times. They are useful for grabbing data from arrays and database records and are often used together with if statements (example: if user authenticates and requires a database listing).