PHP Looping – Advanced Breaking of Loops

The course has already examined using a rudimentary means of 'escaping' loops using the 'break' command, this part of the section examines how Developers can control how far the loops will break if the loops are nested.

An explanation is needed regarding nesting loops, the foreach demonstration used nesting but it must be pointed out that loops can be written within another loop and that too can have a loop within it, in this example the Developer has created three loops …

PHP Looping – Advanced Breaking of Loops

```
for ($loopa = 1; $loopa < 5; $loopa++)
      {

       for ($loopb = 1; $loopb < 5; $loopb++)
          {

          for ($loopc = 1; $loopc < 5; $loopc++)
             {
               echo $loopa.$loopb.$loopc;
             }

          echo "End of Loop B<br>";

          }
       echo "End of Loop A<br>";

      }

echo "Processing continues after the loop";
```

PHP Looping – Advanced Breaking of Loops

The output is as follows …

111112113114End of Loop B
121122123124End of Loop B
131132133134End of Loop B
141142143144End of Loop B
End of Loop A
211212213214End of Loop B
221222223224End of Loop B
231232233234End of Loop B
241242243244End of Loop B
End of Loop A

…

Processing continues after the loop

PHP Looping – Advanced Breaking of Loops

The previous example will continue processing until all the loops have been exhausted, if the Developer places a break in the third level loop (loopc) this will have the effect of stopping that loop and continuing the loop which surrounds it …

```
for ($loopc = 1; $loopc < 5; $loopc++)
        {
          echo $loopa.$loopb.$loopc;
          if ($loopc = 2) break;
        }
```

The result is given on the next slide …

PHP Looping – Advanced Breaking of Loops

111End of Loop B
121End of Loop B
131End of Loop B
141End of Loop B
End of Loop A
211End of Loop B
221End of Loop B
231End of Loop B
241End of Loop B
End of Loop A
…
End of Loop A
Processing continues after the loop

Notice that only the third loop is 'broken'. If the Developer wished to break up to higher in the loop then that needs to be specified.

PHP Looping – Advanced Breaking of Loops

By default the break command assumes a number of 1, this means it breaks one level, therefore if break is given a number of 2 it will break through two levels (notice End of Loop B is never echoed) …

```
for ($loopb = 1; $loopb < 5; $loopb++)
        {
        for ($loopc = 1; $loopc < 5; $loopc++)
           {
            echo $loopa.$loopb.$loopc;
            if ($loopc = 2) break 2;
           }
        echo "End of Loop B<br>";
        }
```

PHP Looping – Advanced Breaking of Loops

This outputs the following …

111End of Loop A
211End of Loop A
311End of Loop A
411End of Loop A
Processing continues after the loop

If the Developer had used ***break 3*** the processing would have ended very quickly !

111Processing continues after the loop

PHP Looping – Using Continue with Loops

The Continue statement allows the Developer to 'skip' the current iteration of the loop returning it to the beginning again … the following is the example code used …

```
for ($loopa = 1; $loopa < 5; $loopa++)
      {
       for ($loopb = 1; $loopb < 5; $loopb++)
           {
           for ($loopc = 1; $loopc < 5; $loopc++)
              {
                echo $loopa.$loopb.$loopc;
              }
           echo "End of Loop B<br>";
           }
       echo "End of Loop A<br>";
      }
```

PHP Looping – Using Continue with Loops

In this first example, the Developer will 'skip' when the value of loopc is 2 (note its not printed) …

```
for ($loopc = 1; $loopc < 5; $loopc++)
            {
              if ($loopc == 2) continue;
              echo $loopa.$loopb.$loopc;
            }
```

111113114End of Loop B
121123124End of Loop B
131133134End of Loop B
141143144End of Loop B
…
Processing continues after the loop

PHP Looping – Using Continue with Loops

Similar to break, Continue can be used to specify the number of loops to 'skip', this example skips 2 loops …

```
for ($loopc = 1; $loopc < 5; $loopc++)
        {
          if ($loopc == 2) continue 2;
          echo $loopa.$loopb.$loopc;
        }
```

111121131141End of Loop A
211221231241End of Loop A
311321331341End of Loop A
411421431441End of Loop A
Processing continues after the loop

Continue 3 would have gone back to the top loop each time …

111211311411Processing continues after the loop