10 Flow Control

Content

- if, if/else, switch
- for, while, do/while

Conditional Statements

- Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.
- In PHP we have two conditional statements:

1. **if (...else) statement** - use this statement if you want to execute a set of code when a condition is true (and another if the condition is not true)
2. **switch statement** - use this statement if you want to select one of many sets of lines to execute

if Statement

- If you want to execute some code if a condition is true and another code if a condition is false, use the if....else statement.
- Syntax

```
if (condition)
code to be executed if condition is true;
else
code to be executed if condition is false;
```

Example

- The following example will output "Have a nice weekend!" if the current day is Friday, otherwise it will output "Have a nice day!":

```
<html>
<body><?php
$d=date("D");
if ($d=="Fri")
echo "Have a nice weekend!";
else
echo "Have a nice day!";
?></body>
</html>
```

if Statement

- If more than one line should be executed when a condition is true, the lines should be enclosed within curly braces:

```
<html>
<body><?php
$x=10;
if ($x==10)
{
echo "Hello<br />";
echo "Good morning<br />";
}
?></body>
</html>
```

```
if ($x==10){
…
}
else
if ($x==11){

}
else
if ($x==12){

}
else
if…
```

switch Statement

- If you want to select one of many blocks of code to be executed, use the Switch statement.

```
switch (expression) {
case label1:
  code to be executed if expression = label1;
  break;
case label2:
  code to be executed if expression = label2;
  break;
default:
  code to be executed
  if expression is different
  from both label1 and label2;
}
```

Example

```
<html>
<body><?php
switch ($x)
{
case 1:
  echo "Number 1";
  break;
case 2:
  echo "Number 2";
  break;
case 3:
  echo "Number 3";
  break;
default:
  echo "No number between 1 and 3";
}
?></body>
</html>
```

- ►This is how it works: First we have a single expression (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure.
- ►If there is a match, the block of code associated with that case is executed.
- ►Use break to prevent the code from running into the next case automatically.
- ►The default statement is used if none of the cases are true.

Looping

- Very often when you write code, you want the same block of code to run a number of times. You can use looping statements in your code to perform this.
- In PHP we have the following looping statements:

1. ***while*** - loops through a block of code if and as long as a specified condition is true
2. ***do...while*** - loops through a block of code once, and then repeats the loop as long as a special condition is true
3. ***for*** - loops through a block of code a specified number of times
4. ***foreach*** - loops through a block of code for each element in an array

while Statement

- The while statement will execute a block of code if and as long as a condition is true.
- Syntax

```
while (condition)
code to be executed;
```

Example

- The following example demonstrates a loop that will continue to run as long as the variable ***i*** is less than, or equal to ***5***.
- ***i*** will increase by 1 each time the loop runs:

```
<html>
<body><?php
$i=1;
while($i<=5) {
echo "The number is " . $i . "<br />";
$i++;
}
?></body>
</html>
```

do...while Statement

- The do...while statement will execute a block of code at least once - it then will repeat the loop as long as a condition is true.
- Syntax

```
do
{
code to be executed;
}
while (condition);
```

Example

- The following example will increment the value of i at least once, and it will continue incrementing the variable i while it has a value of less than 5:

```
<html>
<body><?php
$i=0;
do
{
$i++;
echo "The number is " . $i . "<br />";
}
while ($i<5);
?></body>
</html>
```

for Statement

- The for statement is used when you know how many times you want to execute a statement or a list of statements.
- Syntax

```
for (initialization; condition; increment)
{
    code to be executed;
}
```

for Statement

- The for statement has three parameters.
- The first parameter is for initializing variables,
- The second parameter holds the condition,
- The third parameter contains any increments required to implement the loop.
- If more than one variable is included in either the initialization or the increment section, then they should be separated by commas.
- The condition must evaluate to true or false.

Example

- The following example prints the text "Hello World!" five times:

```
<html>
<body><?php
for ($i=1; $i<=5; $i++)
{
echo "Hello World!<br />";
}
?></body>
</html>
```

```
<html>
<head><title>Loop Example # 1</title></head>
<body>
<?php
$i=2;
while ($i<=10)
{
    echo “$i<br>” ;
    $i=$i+2 ;
}
?>
</body>
</html>
```

```
<html>
<head><title> Loop Example # 2</title></head>
<body>
<?php
for ($i=2; $i<=10 ; $i=$i+2)
{
    echo “$i<br>” ;
}
?>
</body>
</html>
```

```
<html>
<head><title>Loop Example # 3</title></head>
<body>
<?php
$i=1;
while ($i<=10)
{
    if (($i%2)==0){
       echo "$i<br>" ;
    }
    $i=$i+1 ;
}
?>
</body>
</html>
```

```
<html>
<head><title> Loop Example # 4</title></head>
<body>
<?php
$i=3;
while ($i<=100)
{
    if ((($i%3)==0)||(($i%7)==0)){
       echo "$i<br>" ;
    }
$i=$i+1 ;
}
?>
```

```
<html>
<head><title>Prime Test</title></head>
<body>
<?php
$number=3457;
$isPrime=true;
for ($i=2; $i*$i<=$number ; $i=$i+1)
{
    if ($number%$i==0){
        echo "$number is not prime.";
        $isPrime=false;
        break;
    }
}
if ($isPrime)
  echo "$number is prime";
?>
</body>
</html>
```

```
<html>
<head><title>Prime Factorization</title></head>
<body>
<?php
$number=3615;
$first=true;
echo "$number=";
for ($i=2; $number>1 ;$i++) {
    while ($number%$i==0){
            $number /= $i;
            if ($first){
               echo "$i";
               $first=false;
             }
           else
              echo "*$i" ;
    }
}
?>
</body>
</html>
```

foreach Statement

- Loops over the array given by the parameter. On each loop, the value of the current element is assigned to $value and the array pointer is advanced by one - so on the next loop, you'll be looking at the next element.
- Syntax

```
foreach (array as value)
{
    code to be executed;
}
```

PHP Arrays


```
<?
$array[0] = 1;
$array[5] = "This is element 5";
$array[200] = 300;
$array['one'] = "One";

$b = $array;
$b[5] = 'Only in b';

foreach ($array as $elem)
{
    echo "Element is $elem<br>";
}
?>
```

$b does not overwrite $array

No numerical index

PHP Arrays

- Arrays in PHP are quite versatile
  - We can use them as we use traditional arrays, indexing on integer values
  - We can use them as hashes, associating a key with a value in an arbitrary index of the array
  - In either case we access the data via subscripts
    - In the first case the subscript is the integer index
    - In the second case the subscript is the key value
  - We can even mix the two if we'd like

PHP Arrays

- PHP Arrays can be created in a number of ways
  - Explicitly using the array() construct
  - Implicitly by indexing a variable
    - Since PHP has dynamic typing, you cannot identify a variable as an array except by assigning an actual array to it
    - If the variable is already set to a scalar, indexing will have undesirable results – indexes the string!
    - However, we can unset() it and then index it
      - We can test a variable to see if it is set (isset() and if it is an array (is_array()) among other things
  - Size will increase dynamically as needed

PHP Arrays

- Accessing Arrays – can be done in many ways
  - We can use **direct access** to obtain a desired item
    - Good if we are using the array as a hash table or if we need direct access for some other reason
  - For **sequential access**, the **foreach** loop was designed to work with arrays
    - Iterates through the items in two different ways

```
foreach ($arrayvar as $key => $value)
```

» Gives both the key and value at each iteration

```
foreach ($arrayvar as $value)
```

» Gives just the next value at each iteration

PHP Arrays

- Be careful – in both cases data is iterated by the order it has been generated, not by index order
- Items accessed in the arrays using foreach are copies of the data, not references to the data
  - So changing the loop control variable in the foreach loop in PHP does NOT change the data in the original array
  - To do this we must change the value using indexing

– A regular for loop can also be used, but due to the non-sequential requirement for keys, this does not often give the best results

PHP Arrays

- The data in the array is not contiguous, so incrementing a counter for the next access will not work correctly unless the array index values are used in the "traditional" way

– We can also use other iterators such as next and each to access the array elements

- next gives us the next value with each call
  - It moves to the next item, then returns it, so we must get the first item with a separate call (ex: use current())

```
$curr = current($a1);
while ($curr):
      echo "\$curr is $curr <BR />\n";
      $curr = next($a1);
endwhile;
```

PHP Arrays

- each returns an array of two items:
  - A key field for the current key
  - A value field for the current value
  - It returns the next (key,value) pair, then moves, so the first item is no longer a special case

```
while ($curr = each($a1)):
    $k = $curr["key"];
    $v = $curr["value"];
    echo "key is $k and value is $v <BR />\n";
endwhile;
```

  - This function is preferable to next() if it is possible that FALSE or an empty string or 0 could be in the array
    - The loop on the previous slide will stop for any of those values

PHP Arrays

- Both of these iteration functions operate similar to the ***Iterator*** interface in Java
  - Iterate through the data in the collection without requiring us to know how that data is actually organized
  - Allow multiple iterations to proceed concurrently on the same underlying collection
  - However, **unlike in Java**, if the array is changed during the iteration process, the current iteration continues
    - Since new items are always added at the "end" of the array adding a new item during an iteration does not cause any data validity problems

Example

- The following example demonstrates a loop that will print the values of the given array:

```
<html>
<body><?php
$arr=array("one", "two", "three");
foreach ($arr as $value)
{
echo "Value: " . $value . "<br />";
}
?></body>
</html>
```