Control Structures

Conditional Statements

Conditional statements perform different actions for different decisions.
The **if else** statement is used to execute a certain code if a condition is **true**, and another code if the condition is **false**.
Syntax:

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

You can also use the **if** statement without the **else** statement, if you do not need to do anything, in case the condition is **false**.

If Else

The example below will output the greatest number of the two.

```
<?php
$x = 10;
$y = 20;
if ($x >= $y) {
  echo $x;
} else {
  echo $y;
}

// Outputs "20"
?>
```

Try It Yourself

Tap **Try It Yourself** to play around with the code!

The Elseif Statement

Use the **if...elseif...else** statement to specify a **new** condition to test, if the first condition is **false**.

Syntax:

```
if (condition) {
  code to be executed if condition is true;
} elseif (condition) {
```

```
  code to be executed if condition is true;
} else {
  code to be executed if condition is false;
}
```

You can add as many **elseif** statements as you want. Just note, that the **elseif** statement must begin with an **if** statement.

The Elseif Statement

For example:

```
<?php
$age = 21;

if ($age<=13) {
  echo "Child.";
} elseif ($age>13 && $age<19) {
  echo "Teenager";
} else {
  echo "Adult";
}

//Outputs "Adult"
?>
```

Try It Yourself

We used the **logical AND** (&&) operator to combine the two conditions and check to determine whether $age is between 13 and 19.
The curly braces can be omitted if there only one statement after the **if**/**elseif**/**else**.
For example:

```
if($age<=13)
echo "Child";
else
echo "Adult";
```

Loops

When writing code, you may want the same block of code to run over and over again. Instead of adding several almost equal code-lines in a script, we can use **loops** to perform a task like this.

The while Loop

The **while** loop executes a block of code as long as the specified condition is **true**.
Syntax:

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

If the condition never becomes **false**, the statement will continue to execute indefinitely.

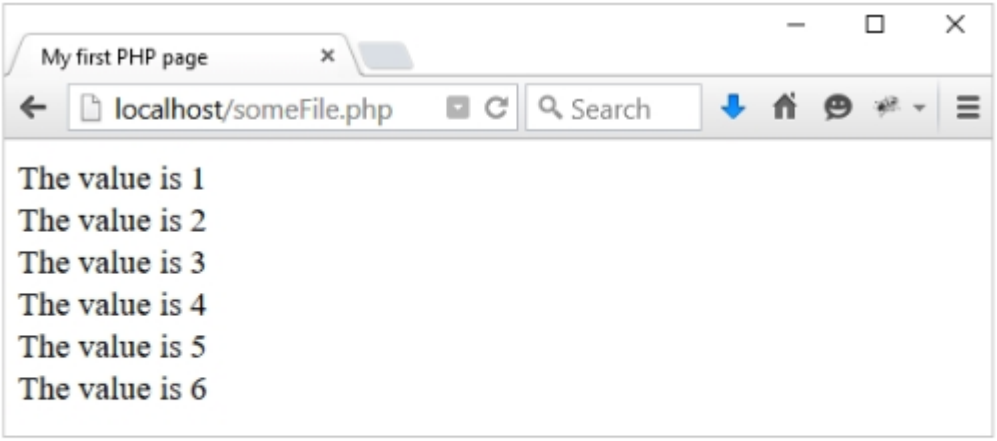
The while Loop

The example below first sets a variable $i to one ($i = 1). Then, the **while** loop runs as long as $i is less than seven ($i < 7). $i will increase by one each time the loop runs ($i++):

```
$i = 1;
while ($i < 7) {
  echo "The value is $i <br />";
  $i++;
}
```

Try It Yourself

This produces the following result:



Tap **Try It Yourself** to play around with the code!

The do...while Loop

The **do...while** loop will always execute the block of code once, check the condition, and repeat the loop as long as the specified condition is true.

Syntax:

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

Regardless of whether the condition is **true** or **false**, the code will be executed at least **once**, which could be needed in some situations.

The do...while Loop

The example below will write some output, and then increment the variable $i by one. Then the condition is checked, and the loop continues to run, as long as $i is less than or equal to 7.

```php
$i = 5;
do {
  echo "The number is " . $i . "<br/>";
  $i++;
} while($i <= 7);

//Output
//The number is 5
//The number is 6
//The number is 7
```

Try It Yourself

Note that in a **do while** loop, the condition is tested AFTER executing the statements within the loop. This means that the **do while** loop would execute its statements at least once, even if the condition is false the first time.

The for Loop

The **for** loop is used when you know in advance how many times the script should run.

```php
for (init; test; increment) {
  code to be executed;
}
```

Parameters:
init: Initialize the loop counter value
test: Evaluates each time the loop is iterated, continuing if evaluates to **true**, and ending if it evaluates to **false**
increment: Increases the loop counter value

Each of the parameter expressions can be empty or contain multiple expressions that are separated with **commas**.
In the **for** statement, the parameters are separated with **semicolons**.

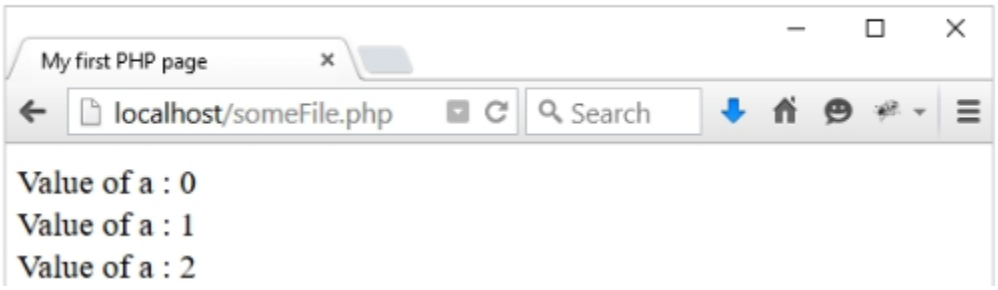
The for Loop

The example below displays the numbers from 0 to 5:

```php
for ($a = 0; $a < 6; $a++) {
  echo "Value of a : ". $a . "<br />";
}
```

Try It Yourself

Result:



```
Value of a : 3
Value of a : 4
Value of a : 5
```

The **for** loop in the example above first sets the variable **$a** to 0, then checks for the condition ($a < 6). If the condition is true, it runs the code. After that, it increments $a ($a++).

The foreach Loop

The **foreach** loop works only on arrays, and is used to loop through each key/value pair in an array.

There are two syntaxes:

```
foreach (array as $value) {
  code to be executed;
}
//or
foreach (array as $key => $value) {
   code to be executed;
}
```

The first form loops over the array. On each iteration, the value of the current element is assigned to **$value**, and the array pointer is moved by one, until it reaches the last array element.
The second form will additionally assign the current element's key to the **$key** variable on each iteration.

The following example demonstrates a loop that outputs the values of the **$names** array.

```
$names = array("John", "David", "Amy");
foreach ($names as $name) {
   echo $name.'<br />';
}

// John
// David
// Amy
```

Try It Yourself

Tap **Try It Yourself** to play around with the code!

The switch Statement

The **switch** statement is an alternative to the **if-elseif-else** statement.
Use the **switch** statement to select one of a number of blocks of code to be executed.

Syntax:

```
switch (n) {
  case value1:
    //code to be executed if n=value1
    break;
```

```
  case value2:
    //code to be executed if n=value2
    break;
  ...
  default:
    // code to be executed if n is different from all labels
}
```

First, our single expression, **n** (most often a variable), is evaluated once. Next, the value of the expression is compared with the value of each case in the structure. If there is a match, the block of code associated with that case is executed.

> Using **nested if else statements** results in similar behavior, but switch offers a more elegant and optimal solution.

Switch

Consider the following example, which displays the appropriate message for each day.

```
$today = 'Tue';

switch ($today) {
  case "Mon":
    echo "Today is Monday.";
    break;
  case "Tue":
    echo "Today is Tuesday.";
    break;
  case "Wed":
    echo "Today is Wednesday.";
    break;
  case "Thu":
    echo "Today is Thursday.";
    break;
  case "Fri":
    echo "Today is Friday.";
    break;
  case "Sat":
    echo "Today is Saturday.";
    break;
  case "Sun":
    echo "Today is Sunday.";
    break;
  default:
    echo "Invalid day.";
}
//Outputs "Today is Tuesday."
```

Try It Yourself

> The **break** keyword that follows each case is used to keep the code from automatically running into the next case. If you forget the **break;** statement, PHP will automatically continue through the next case statements, even when the case doesn't match.

---

default

The **default** statement is used if no match is found.

```
$x=5;
switch ($x) {
  case 1:
```

```
    echo "One";
    break;
  case 2:
    echo "Two";
    break;
  default:
    echo "No match";
}

//Outputs "No match"
```

Try It Yourself

The **default** statement is optional, so it can be omitted.

Switch

Failing to specify the **break** statement causes PHP to continue to executing the statements that follow the **case**, until it finds a **break**. You can use this behavior if you need to arrive at the same output for more than one case.

```
$day = 'Wed';

switch ($day) {
  case 'Mon':
    echo 'First day of the week';
    break;
  case 'Tue':
  case 'Wed':
  case 'Thu':
    echo 'Working day';
    break;
  case 'Fri':
    echo 'Friday!';
    break;
  default:
    echo 'Weekend!';
}

//Outputs "Working day"
```

Try It Yourself

The example above will have the same output if **$day** equals 'Tue', 'Wed', or 'Thu'.

The break Statement

As discussed in the previous lesson, the **break** statement is used to break out of the **switch** when a case is matched.
If the break is absent, the code keeps running. For example:

```
$x=1;
switch ($x) {
```

```
  case 1:
    echo "One";
  case 2:
    echo "Two";
  case 3:
    echo "Three";
  default:
    echo "No match";
}

//Outputs "OneTwoThreeNo match"
```

Try It Yourself

Break can also be used to halt the execution of **for**, **foreach**, **while**, **do-while** structures.

> The **break** statement ends the current **for**, **foreach**, **while**, **do-while** or **switch** and continues to run the program on the line coming up after the loop.
> A **break** statement in the outer part of a program (e.g., not in a control loop) will stop the script.

The continue Statement

When used within a looping structure, the **continue** statement allows for skipping over what remains of the current loop iteration. It then continues the execution at the condition evaluation and moves on to the beginning of the next iteration.

The following example skips the even numbers in the **for** loop:

```
for ($i=0; $i<10; $i++) {
  if ($i%2==0) {
    continue;
  }
  echo $i . ' ';
}

//Output: 1 3 5 7 9
```

Try It Yourself

> You can use the **continue** statement with all looping structures.

---

include

The **include** and **require** statements allow for the insertion of the content of one PHP file into another PHP file, before the server executes it.
Including files saves quite a bit of work. You can create a standard header, footer, or menu file for all of your web pages. Then, when the header is requiring updating, you can update the header include file only.

Assume that we have a standard header file called **header.php**.

```
<?php
  echo '<h1>Welcome</h1>';
?>
```

Use the **include** statement to include the header file in a page.

```
<html>
 <body>

 <?php include 'header.php'; ?>

 <p>Some text.</p>
 <p>Some text.</p>
 </body>
</html>
```

The include and require statements allow for the insertion of the content of one PHP file into another PHP file, before the server executes it.

---

include

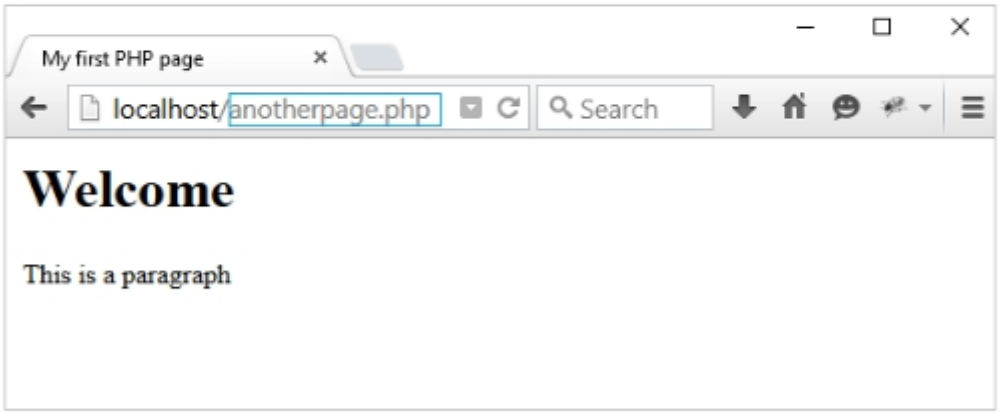
Using this approach, we have the ability to include the same **header.php** file into multiple pages.

```
<html>
 <body>

 <?php include 'header.php'; ?>

 <p>This is a paragraph</p>
 </body>
</html>
```

Result:



Files are included based on the file **path**.
You can use an **absolute** or a **relative** path to specify which file should be included.

---

include vs require

The **require** statement is identical to include, the exception being that, upon failure, it produces a fatal error.
When a file is included using the **include** statement, but PHP is unable to find it, the script continues to execute.
In the case of **require**, the script will cease execution and produce an error.

Use **require** when the file is required for the application to run.
Use **include** when the file is not required. The application should continue, even when the file is not found.

End.