Flow Control in PHP

Session 12

Objectives

- *Explain the use of loops*
- *Explain the use of jump statements*

Introduction

- Loops
  - Perform repetitive tasks such as:
    - Retrieving information stored in databases
    - Sending mails to multiple users
    - Reading contents of an array
- Loops provided by PHP are as follows:
  - While
  - Do-while
  - For

Working with Loops

- Executes a block of code repetitively
- Tests the specified condition
  - If `true`, the statements present in the body of the loop are executed repetitively
  - If `false`, the loop ends, and the control is transferred to the statement following the loop
- The continuous execution of statements inside the loop is called iteration

while Loop

- Validity of the condition is checked before the loop is executed:
  - If condition is `true`, statements are executed in the loop body
  - If condition is `false`, the body of the loop is not executed

Syntax

```
while(condition)
{
These statements are executed only if the condition is true;
}
These statements are executed irrespective of the condition;
```

Where,

- The `condition` is the test expression consisting of variables and operators

while Loop

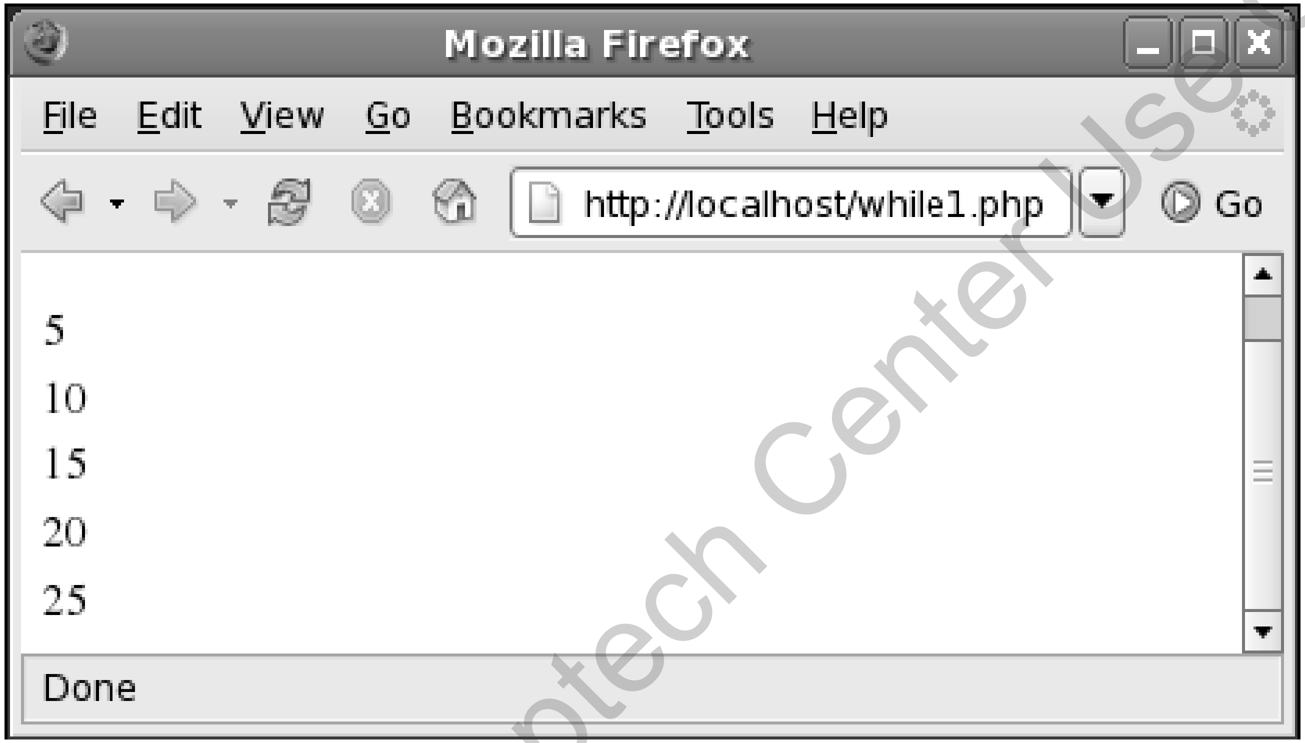
- Displaying the first five multiples of 5

Snippet

```
<?php
$counter=1;
$number=5;
while($counter <= 5)
{
$result=$number*$counter;
echo "<br>$result";
$counter=$counter+1;
}
?>
```

while Loop

Displays the following output:


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/while1.php
Go
Done

```
5
10
15
20
25
```

In the code, the result is displayed until the counter reaches 5.

The loop stops once the counter exceeds 5.

while Loop

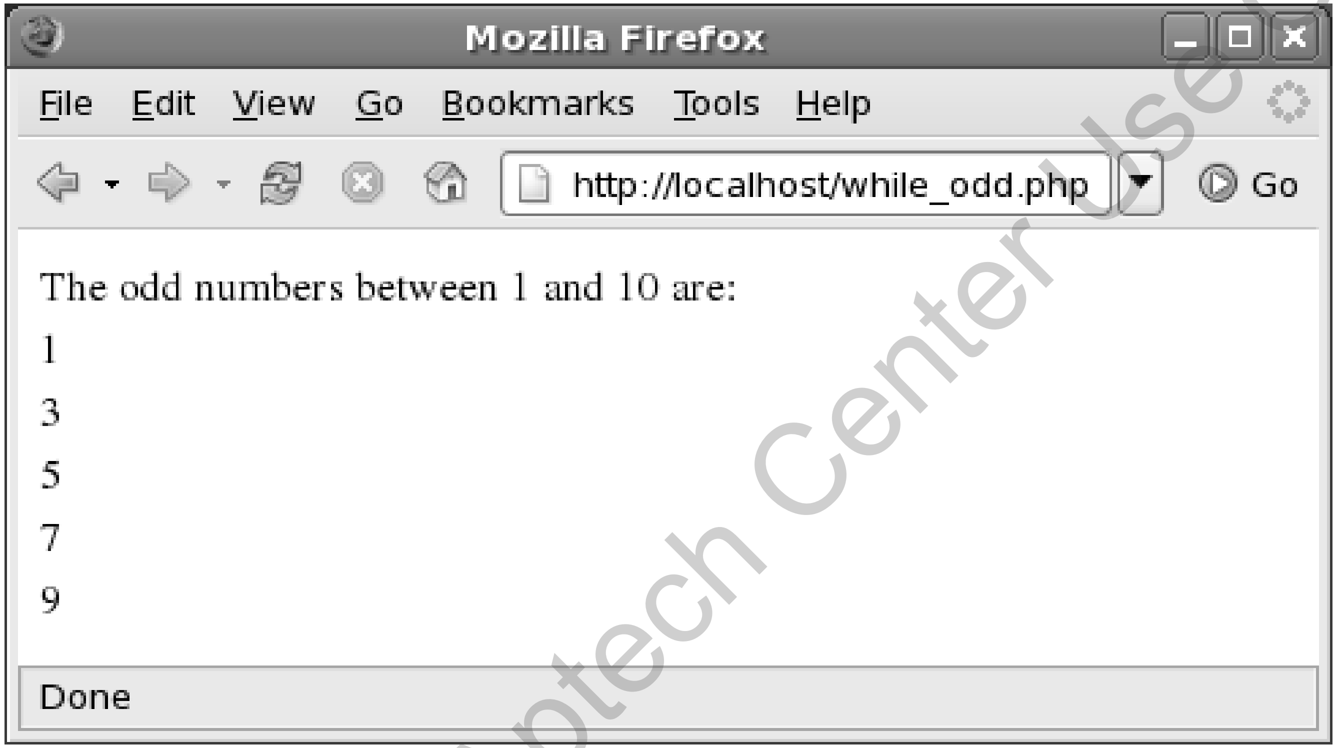
- Displaying the odd numbers between 1 to 10

Snippet

```
<?php
$number=1;
echo "The odd numbers between 1 and 10 are:";
while($number <= 10)
{
echo "<br>$number";
$number=$number+2;
}
?>
```

while Loop

Displays the following output:


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/while_odd.php
Go
Done

```
The odd numbers between 1 and 10 are:
1
3
5
7
9
```

In the code, the number is always incremented by 2 until the number reaches 10.

This is because **`$number`** is initialized at 1 and every alternate number is odd.

do-while Loop

- Condition is checked at the end of the loop
- Executes the loop body at least once
- Works similar to the `while` loop

Syntax

```
do{
<These statements are executed if the condition is true;>
}while(condition)
<These statements are executed irrespective of the condition;>
```

Where,

- The loop body is followed by the `while` keyword and the condition in parenthesis

do-while Loop

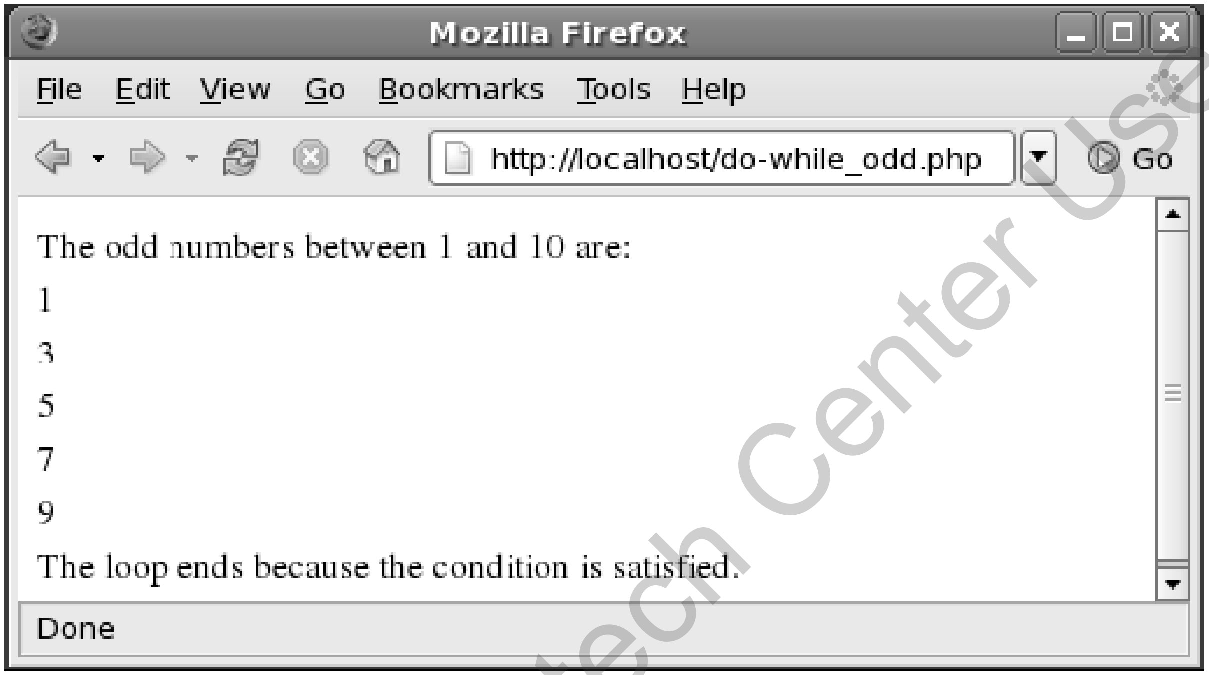
- Displaying the odd numbers between 1 to 10 using `do-while` loop

Snippet

```
<?php
$number=1;
echo "The odd numbers between 1 and 10 are:";
do{
    echo "<br>$number";
    $number=$number+2;
}
while($number <= 10);
echo "<br>The loop ends because the condition is satisfied.";
?>
```

do-while Loop

Displays the following output:


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/do-while_odd.php
Go
Done

```
The odd numbers between 1 and 10 are:
1
3
5
7
9
The loop ends because the condition is satisfied.
```

In the code, **`$number`** is initialized at 1 and is incremented by 2, since the odd numbers are required to be displayed.

The execution of the loop continues until the counter reaches 10. The loop stops execution once the condition is satisfied.

for Loop

- Executes block of code repetitively for a fixed number of times
- Statements in the loop body are executed as long as the condition is satisfied
- Stops the execution only when the condition is not satisfied

for Loop

Syntax

```
for (expr1; expr2; expr3)
{
These statements are executed if the condition is true;
}
These statements are executed irrespective of the condition;
```

Where,

- **expr1** - is an initialization expression that initializes the value of the counter
- **expr2** – is a test expression that is evaluated for each loop iteration
- **expr3** – is a re-initialization expression that increases or decreases the value in the counter variable

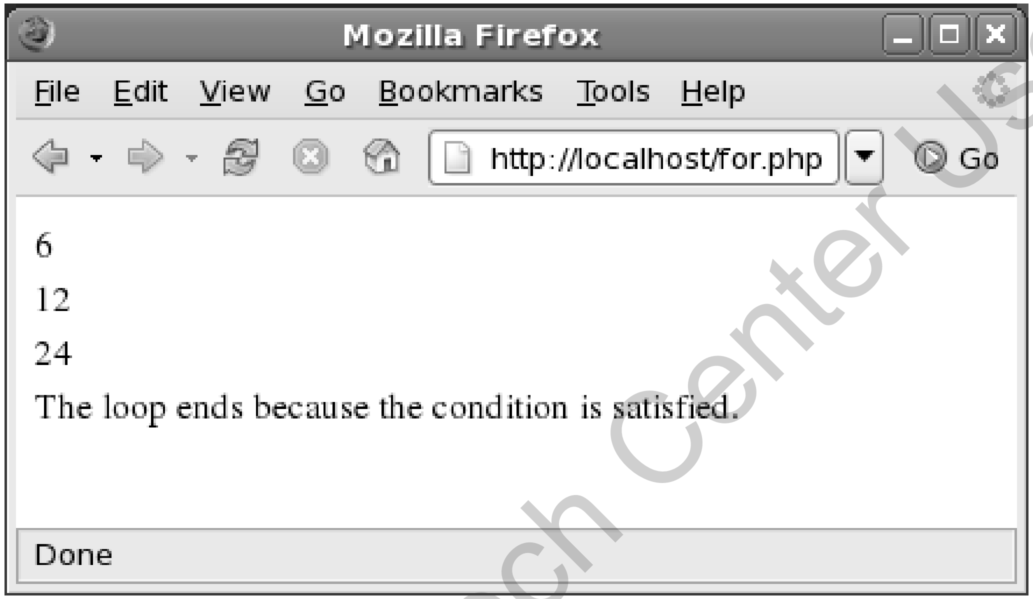
for Loop

- Displaying the double of the given number using `for` loop

Snippet

```
<?php
$number=6;
for($counter=1; $counter <= 3; $counter++)
{
  echo "$number<br>";
  $number=$number*2;
  }
  echo "The loop ends because the condition is
  satisfied.";
  ?>
```

Displays the following output:



```
6
12
24
The loop ends because the condition is satisfied.
```

The variable, **`$number`** is initialized with a value of 6.

When the loop starts, 6 is multiplied by 2 And the value is stored in the variable, **`$number`** is 12.

The loop executes thrice since the terminating condition has been set when the counter value reaches 3.

Once the counter value reaches 3, the loop stops executing.

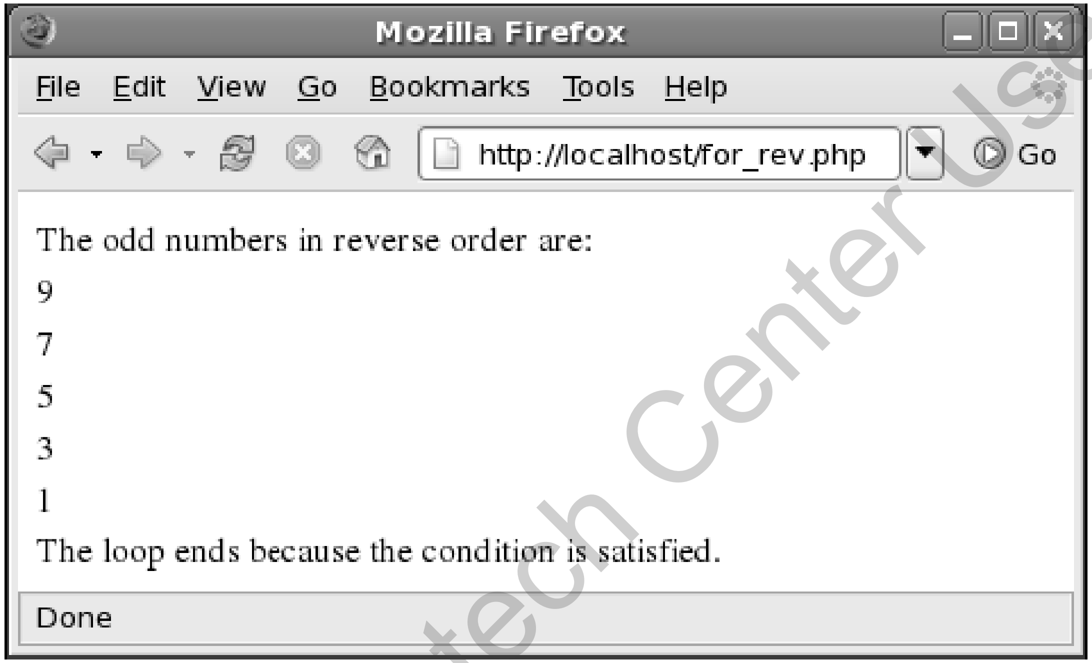
for Loop

- Displaying the first five odd numbers in the reverse order using `for` loop

Snippet

```
<?php
echo "The odd numbers in reverse order are:";
for($i=5;$i>=1;$i--)
{
  $number=$i * 2 - 1;
  echo "<br>$number";
  }
  echo  "<br>The  loop  ends  because  the  condition  is
  satisfied.";
  ?>
```

Displays the following output:


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/for_rev.php
Go
Done

```
The odd numbers in reverse order are:
9
7
5
3
1
The loop ends because the condition is satisfied.
```

In the code, the for loop declares a counter variable, which is initialized at 5.

The re-initialization expression decrements the counter every time the for loop is executed.

Jump Statements

- Control the execution of the loop and conditional statements

- PHP provides the following jump statements:
  - `break`
  - `continue`
  - `exit`

break Statement

- Stops the execution of the loops and conditional statements
- The control is then transferred either to the beginning of the next loop or to the statement following the loop
- Can be used with the `if` statement, `switch` statement, `for` loop, `while` loop, and `do-while` loop

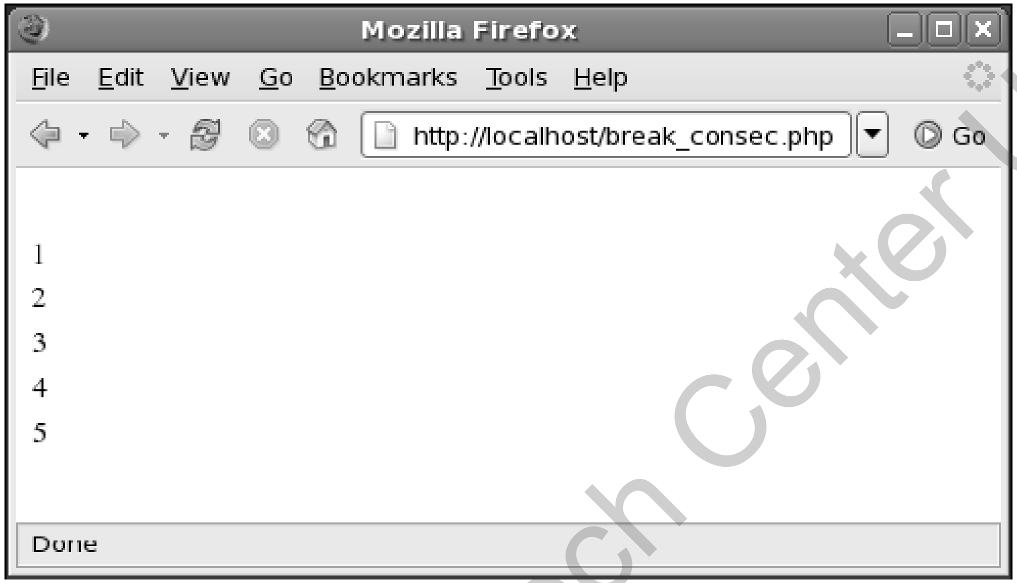
break Statement

- Displaying consecutive numbers from 1 to 5 using `break` statement

Snippet

```
<?php
for($i=1;;$i++) {
if($i>5)
{
   break;
}
echo "<br>$i";
}
?>
```

Displays the following output:


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/break_consec.php
Go
Done

```
1
2
3
4
5
```

The `break` statement is used within the for loop.

The `for` loop does not include any terminating condition.

The terminating condition is specified within the `if` statement using the `break` statement.

If the `break` statement is not used, it will become an infinite loop.

break Statement

- Checking whether the alphabet is a vowel using `switch` statement

Snippet

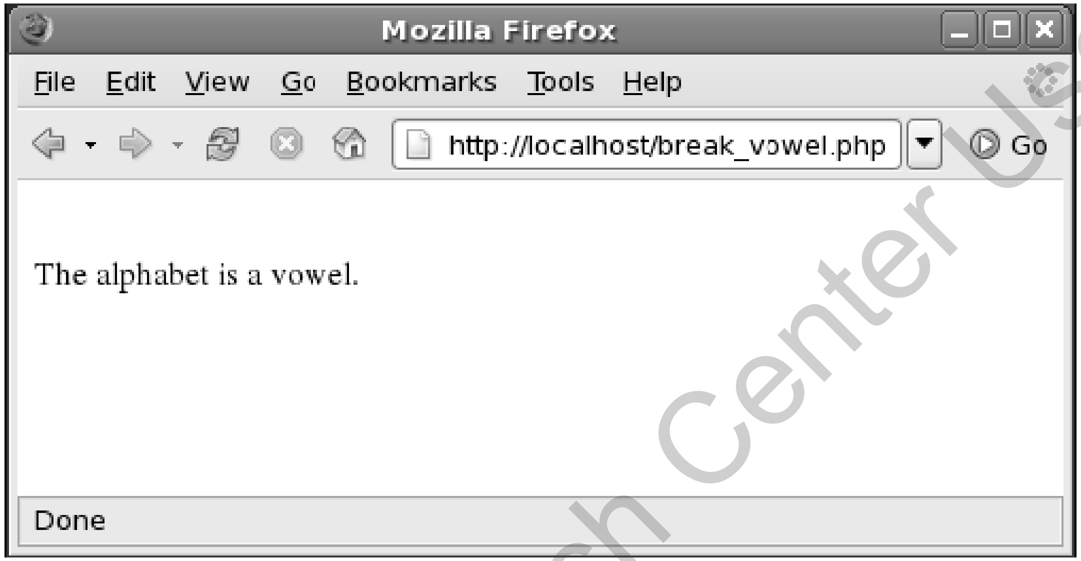
```
<?php
$alphabet='u';
switch($alphabet) {
case 'a':
echo "<br>The alphabet is a vowel.";
break;
case 'A':
                  echo "<br>The alphabet is a vowel.";
                  break;
case 'e':
         echo "<br>The alphabet is a vowel.";
         break;
case 'E':
         echo "<br>The alphabet is a vowel.";
         break;
case 'i':
```

break Statement

```
          echo "<br>The alphabet is a vowel.";
          break;
case 'I':
          echo "<br>The alphabet is a vowel.";
          break;
case 'o':
          echo "<br>The alphabet is a vowel.";
          break;
case 'O':
          echo "<br>The alphabet is a vowel.";
          break;
case 'u':

          echo "<br>The alphabet is a vowel.";
                   break;
case 'U':
                   echo "<br>The alphabet is a vowel.";
                   break;
default:
                   echo   "<br>The   alphabet   is   not   a
vowel.";
}?>
```

Displays the following output:


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/break_vowel.php
Go
Done

The alphabet is a vowel.

In the code, the `break` statement is used in the `switch` statement.

The `break` statement moves the control to the statements following the `switch` statement.

If the `break` statement is not used, PHP will execute all the statements including the statements present in the following case statement.

continue Statement

- Used within the loop statements
- Skips the code following the continue statement in the loop body and executes the next iteration of the loop
- Can be used with the `if` statement, `for` loop, `while` loop, and `do-while` loop

continue Statement

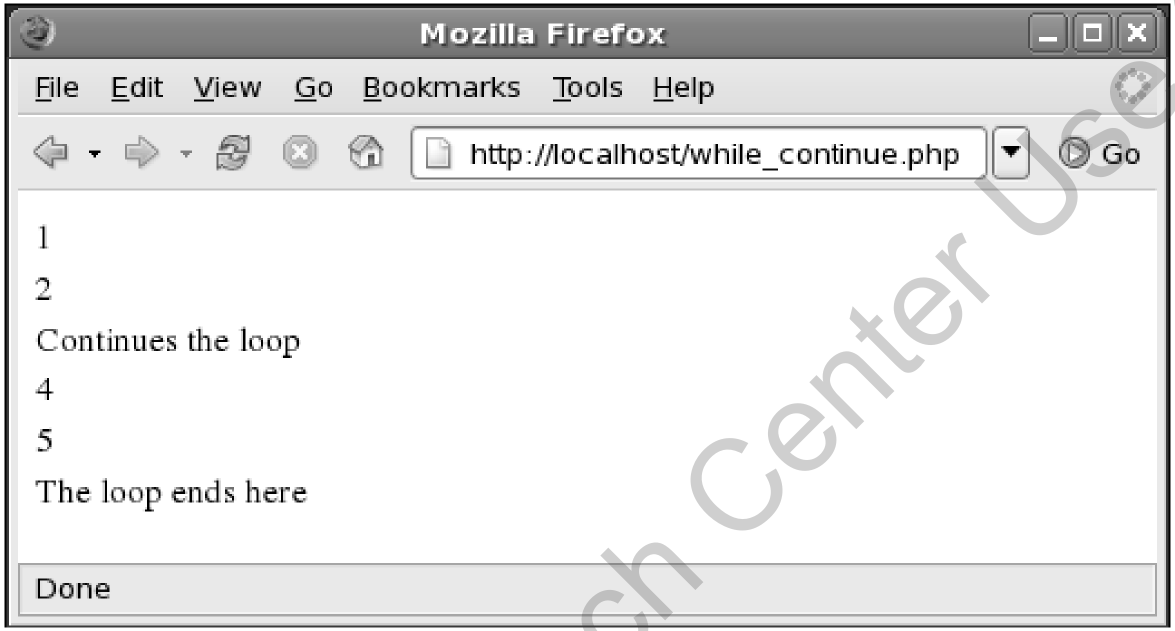
- Displaying the consecutive numbers from 1 to 5 using the `while` loop

Snippet

```
<?php
$counter = 0;
while($counter<5)
{
  $counter++;
  if($counter==3)
  {
    echo "Continues the loop<br>";
    continue;
  }
  echo "$counter<br>";
  }
  echo "The loop ends here";
  ?>
```

continue Statement

Displays the following output:


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/while_continue.php
Go
Done

```
1
2
Continues the loop
4
5
The loop ends here
```

In the code, the `continue` statement is used in the `if` statement.

Here, the counter is initialized to 0. The loop continues until the counter reaches 3.

When the counter reaches the value of 3, the loop skips the `if` body and executes the next iteration of the loop.

The loop continues until the condition becomes `false`.

exit Statement

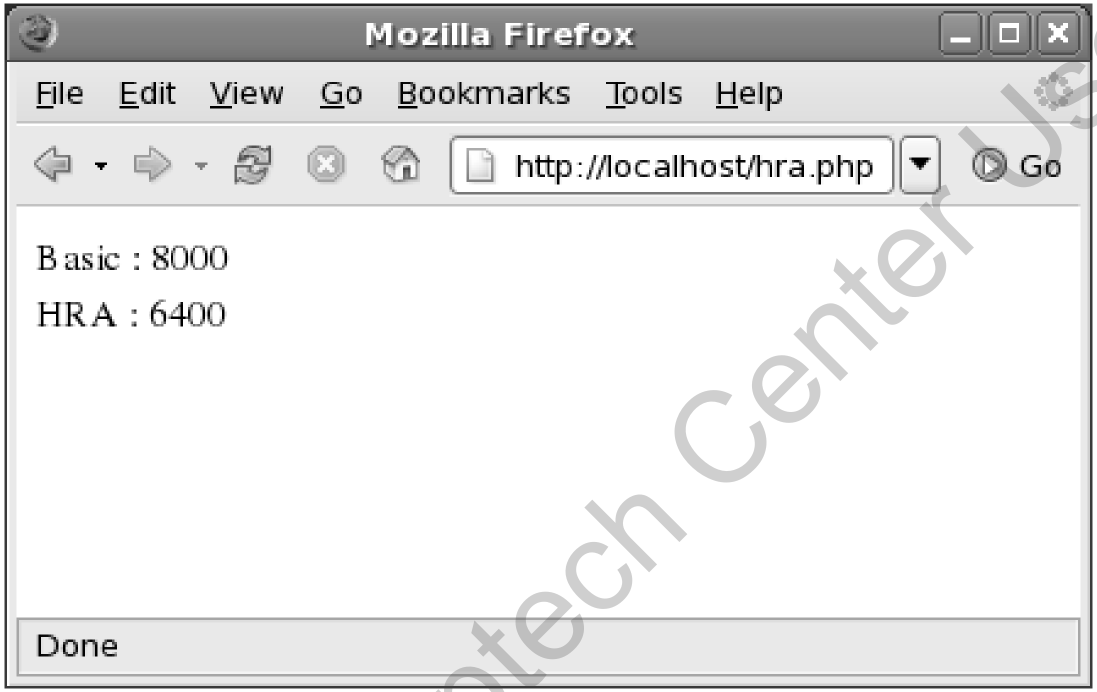
- Ends the loop and the control is transferred to the statement following the loop body
- Following code calculates the HRA using `exit` statement:

Snippet

```
<?php
$salary=8000;
if($salary<6000)
{
   echo "Basic : $salary<br>";
   echo "Salary below 6000 is not entitled for HRA.";
   exit;
   }
else
{
   echo "Basic : $salary<br>";
   $hra=$salary * 0.8;
   echo "HRA : $hra";
   }
   ?>
```

exit Statement

Displays the following output:



```
Basic : 8000
HRA : 6400
```

In the code, HRA is calculated based on the basic salary.

If the basic salary is less than 6000, the `if` statement exits.

If the basic salary is greater than or equal to 6000, HRA is calculated.

Summary

- A loop executes a block of code repetitively
- A while loop executes the statements in the loop body as long as the condition is true
- The do-while loop is similar to the while loop. In this loop structure the condition is placed at the end of the loop
- A for loop enables the execution of a block of code repetitively for a fixed number of times
- The jump statements control the execution of the loop statements

Summary

- The break statement stops the execution of the loop. The control is then passed either to the beginning of the next loop or to the statement following the loop
- The continue statement skips the code following the continue statement in the loop body and executes the next iteration of the loop
- The exit statement ends the loop and the control is passed to the statement following the loop body