Unit-III Control Structures

Control Structure

- Control structure allows you to control the flow of code execution in application. Generally, a program is executed sequentially, line by line, and a control structure allows to alter that flow, usually depending on certain conditions.
- Control structures are core features of the PHP language that allow script to respond differently to different inputs or situations. This could allow script to give different responses based on user input, file contents, or some other data.
- PHP supports a number of different control structures:
    - Conditional Statements
        - if
        - else
        - elseif
        - switch
    - Looping Statements
        - while
        - do-while
        - for
        - foreach

if statement

- An if statement is a way of controlling the execution of a statement that follows it (that is, a single statement or a block of code inside braces).
- The if statement evaluates an expression between parentheses. If this expression results in a true value, the statement is executed.
- Otherwise, the statement is skipped entirely. This enables scripts to make decisions based on any number of factors

Syntax

```
if (expression) {
   Statements;
}
```

- As described in the section about expressions, expression is evaluated to its Boolean value.

- If expression evaluates to true, PHP will execute statements, and if it evaluates to false - it'll ignore it.
- The following example would display a is bigger than b if $a is bigger than $b

Example

```
<?Php
        If ($a > $b) {
                echo "a is bigger than b";
        }
?>
```

if...else statement

- Often you'd want to execute a statement if a certain condition is met, and a different statement if the condition is not met. This is what else is for.
- else extends the if statement to execute a statement in case the expression in the if statement evaluates to false.

Syntax

```
if (expression) {
  block-1
} else {
  block-2
}
```

- If expression evaluates to true, then block 1 is executed else block 2 is executed.
- For example, the following code would display a is bigger than b if $a is bigger than $b, and a is not bigger than b otherwise

Example

```
<?php
        If ($a > $b) {
                echo "a is bigger than b";
        } else {
                echo "a is not bigger than b";
        }
?>
```

elseif statement

- elseif, as its name suggests, is a combination of if and else.

- If the first expression does not evaluate to true, the first block of code is ignored.
- The else if clause then causes another expression to be evaluated.
- Once again, if this expression evaluates to true, the second block of code is executed.
- Otherwise, the block of code associated with the else clause is executed.
- You can include as many else if clauses as you want, and if you don't need a default action, you can omit the else clause.
- For example, the following code would display a is bigger than b, a equal to b or a is smaller than b

Example

```
<? php
        If ($a > $b) {
                echo "a is bigger than b";
        } else if ($a == $b) {
                echo "a is equal to b";
        } else {
                echo "a is smaller than b";
        }
?>
```

Switch

- The switch statement is similar to a series of if statements on the same expression.
- In many occasions, you may want to compare the same variable (or expression) with many different values, and execute a different piece of code depending on which value it equals to. This is exactly what the switch statement is for.

Syntax

```
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 any of the case does not execute.
}
```

Example

```
<? php
        switch ($x) {
                case 1: echo "number 1";                          break;
                case 2:  echo "number 2";                         break;
                case 3: echo "number 3";                          break;
                default: echo "number is not between 1 and 3";    break;
        }
?>
```

LOOPS

- Scripts can also decide how many times to execute a block of code. Loop statements are designed to enable you to achieve repetitive tasks.
- A loop will continue to operate until a condition is achieved, or you explicitly choose to exit the loop.

while loop

- while loops are the simplest type of loop in php.

Syntax

```
while (expression) {
   statements;
}
```

- The meaning of a while statement is simple. It tells php to execute the statement(s) repeatedly, as long as the while expression evaluates to true.
- The value of the expression is checked each time at the beginning of the loop, so even if this value changes during the execution of the nested statement(s), execution will not stop until the end of the iteration (each time php runs the statements in the loop is one iteration).
- Sometimes, if the while expression evaluates to false from the very beginning, the nested statement(s) won't even be run once.

Example

```
<? php
        $i = 1;
        while ($i <= 5) {
           echo $i++;
        }
?>
```

do...while loop

- Do…while loops are very similar to while loops, except the truth expression is checked at the end of each iteration instead of in the beginning.
- The main difference from regular while loops is that the first iteration of a do-while loop is guaranteed to run (the truth expression is only checked at the end of the iteration), whereas it's may not necessarily run with a regular while loop (the truth expression is checked at the beginning of each iteration, if it evaluates to false right from the beginning, the loop execution would end immediately).

Syntax

```
do {
        statements;
} while (expr);
```

Example

```
<? php
        $i = -5;
        do {
           echo $i;
        } while ($i > 0 );
?>
```

- The above loop would run one time exactly, since after the first iteration, when truth expression is checked, it evaluates to false ($i is not bigger than 0) and the loop execution ends.

for loop

- The variable controlling loop is initialized outside the while statement. The while statement then tested the variable in its expression. The variable was incremented within the code block.
- The for statement allows you to achieve this on a single line. This allows for more compact code and makes it less likely that you will forget to increment a counter variable, thereby creating an infinite loop

Syntax

```
for (initialization expression; test expression; modification expression) {
// code to be executed
}
```

- The expressions of the loop are separated by semicolons.
    - First expression initializes a counter variable (starting of loop)
    - Second expression is the test condition to stop execution (ending of loop)
    - Third expression increments/decrements the counter (to reach start to end)
- Consider the following examples. All of them display numbers from 1 to 10

```
<? php
        /* example 1 */
        for ($i = 1; $i <= 10; $i++) {
           echo $i;
        }
        /* example 2 */
        for ($i = 1; ; $i++) {
           if ($i > 10)
                  break;
           echo $i;
        }
        /* example 3 */
        $i = 1;
        for (; ; ) {
           if ($i > 10) {  break;    }
           echo $i;
           $i++;
        }
        /* example 4 */
        for ($i = 1; $i <= 10; print $i, $i++);
        ?>
```

- Of course, the first example appears to be the nicest one (or perhaps the fourth), but you may find that being able to use empty expressions in for loops comes in handy in many occasions.

foreach loop

- 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;
}
```

Example

```
<?php
        $arr = array(1, 2, 3, 4);
        foreach ($arr as $value) {
           echo $value. "<br>";
}
```

break

- Break ends execution of the current for, foreach, while, do-while or switch structure.

Example

```
<?php
        $i=1;
        while(1) {
        switch($i) {
                        case 1: echo "<br>$i - first case";             break;
                        case 2: echo "<br>$i - second case";            break;
                        case 5: echo "<br>$i - breaking from while.";   break;
                        default: echo "<br>$i - default case.";         break;
                }
                $i++;
        }
        echo "<br> Program over.";
?>
```

continue

- Continue is used within looping structures to skip the rest of the current loop iteration and continue execution at the condition evaluation and then the beginning of the next iteration.

Example

```
<?php
   for($i=1;$i<=2;$i++){
```

```
            for($j=1;$j<=2;$j++){
              if($j==2) continue;
              echo "J = $j<br>";
            }
            echo "I = $i<br>";
          }
        ?>
```

exit() Function

- exit() function prints a message and exits the application. It's often used to print a different message in the event.
- Use exit() when there is not an error and have to stop the execution.

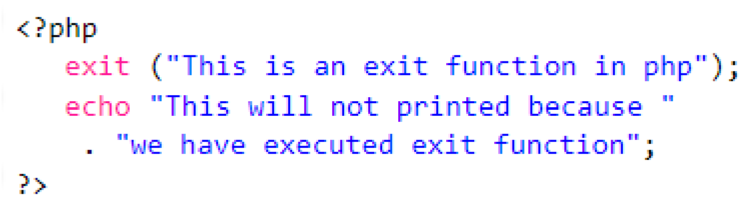
Syntax:

exit ("your message");

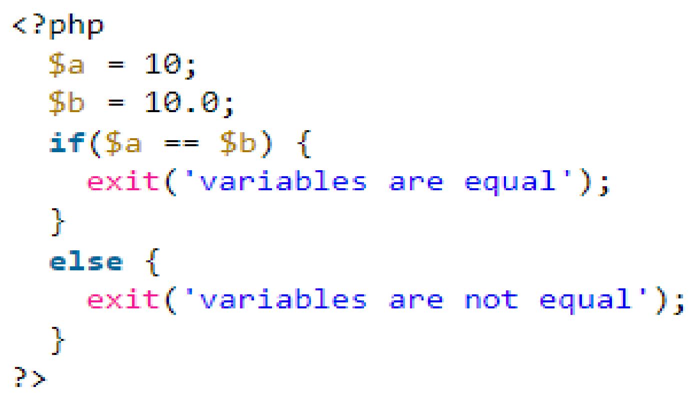
or

exit ();

Example:



```
<?php
   exit ("This is an exit function in php");
   echo "This will not printed because "
    . "we have executed exit function";
?>
```



```
<?php
  $a = 10;
  $b = 10.0;
  if($a == $b) {
    exit('variables are equal');
  }
  else {
    exit('variables are not equal');
  }
?>
```

die() Function

- die() is the same as exit(). A program's result will be an empty screen. Use die() when there is an error and have to stop the execution.

Syntax:

die ("your message"); or die ();

exit()	**die()**
Exits the process without exceptions	Can throw in an exception
Exit script and prints message	Prints a message and end a process

RETURN

- return statement or keyword immediately terminates the execution of a function when it is called from within that function.
- If return is called from a global scope, the script stops the execution of the current script. If the current script file was included using include () or required (), then control goes back to the calling file.

Syntax:

return [expression]

Example:

```
<?php
    function square($x) {
        return $x**2;
    }
    $num=5;
    echo "calling function with argument $num";
    $result=square($num);
    echo "function returns square of $num = $result";
?>

// This is greeting.php file
<?php

    echo "Welcome every one...!!!";
    return;
?>

// This is printmessage.php file
<?php
    include("greeting.php");
    echo "Thank you so much...!!! Have nice Day...!!!";
?>
```

ARRAYS

- You know that these variables are used to store values. They can store only one value at a time
- But arrays are special types of variables that enable you to store as many values as you want.
- An array in PHP is actually an ordered map. A map is a type that maps values to keys.

- The value is whatever value you associate with that position – a string, an integer, or whatever you want.
- PHP mainly offers two types of arrays:
    - Indexed Array
    - Associative Array

<u>Creating arrays</u>

- You can create an array using
    - either the array () function (convenient for multiple value)
    - or the array operator [] (for a few elements or changing single value)
- The array () function is usually used when you want to create a new array and populate it with more than one element at the same time.
- The array operator is used when you want to create a new array with a few elements, or when you want to add to an existing element.

<u>Indexed Arrays</u>

- Arrays are indexed, which means that each entry is made up of a key and a value.
- The key is the index position, beginning with 0.

Syntax

```
$arr_var = array(val1,val2,…);          // array() function
                OR
$arr_var[index]  = "value";             // array operator []
```

Examples

1. $rainbow = array("red", "orange", "yellow", "green", "blue", "indigo", "violet");
2.
```
$rainbow[] = "red";
$rainbow[] = "orange";
$rainbow[] = "yellow";
$rainbow[] = "green";
$rainbow[] = "blue";
$rainbow[] = "indigo";
$rainbow[] = "violet";
```
3.
```
$rainbow[0] = "red";
$rainbow[1] = "orange";
$rainbow[2] = "yellow";
$rainbow[3] = "green";
$rainbow[4] = "blue";
$rainbow[5] = "indigo";
$rainbow[6] = "violet";
```

- All the three examples create a seven-element array called $rainbow, with values starting at index position 0 and ending at index position 6.
- Note: In PHP it is not compulsory to provide ordered index. You can also have unordered index numbers for the elements of array. Moreover, you can store multiple datatypes of value in a single array.

Example

```
<?php
        $a[5] = "First element";
        $a[4] = 4.25;
        $a[-10] = true;
        echo $a[5],"<br>" ,$a[4] ,"<br>",$a[-10];
?>
```

Associative Arrays

- Whereas numerically indexed arrays use an index position as the key, while associative arrays utilize actual named keys.

Syntax

```
$arr_var = array(key1=>val1, key2=>val2,…);          // array() function
                OR
$arr_var[key]  = “value”;                            // array operator []
```

Example

```
<?php
        $arr = array("Name"=>"Rajanikant",
                        "Occupation" => "Superhero",
                        "Age" => 0.00,
                        "Speciality" => "There is nothing that Rajni kan't." );
        $arr["Salary"] = -9999999;
        echo $arr[“Speciality”]. “<br>”;
        foreach($arr as $key => $val) {
                echo "$key => $val<br>";
        }
?>
```

Multidimensional Arrays

- Multidimensional array holds other arrays. It is like array of arrays.
- If each set of key/value pairs constitutes a dimension, a multidimensional array holds more than one series of these key/value pairs.

Example

```
<?php
        $arr = array(
                        array("Name"=>"Shaktiman",
                                "Occupation" => "Superhero",
                                "Age" => 10,
                                "Speciality" => "Can be invisible."),
                        array("Name"=>"Krish",
                                "Occupation" => "Superhero",
                                "Age" => 40,
                                "Speciality" => "Can fly." ),
                        array("Name"=>"Rajanikant",
                                "Occupation" => "Superhero",
                                "Age" => 0.00,
                                "Speciality" => "There is nothing that Rajni kan't." )
                        );
  echo $arr[2];               //array & notice
  echo $arr[1]['Name'];       //Krish
  foreach($arr as $element){
    foreach($element as $val){
      echo "<br>$val";
    }
    echo "<br>";
  }
?>
```