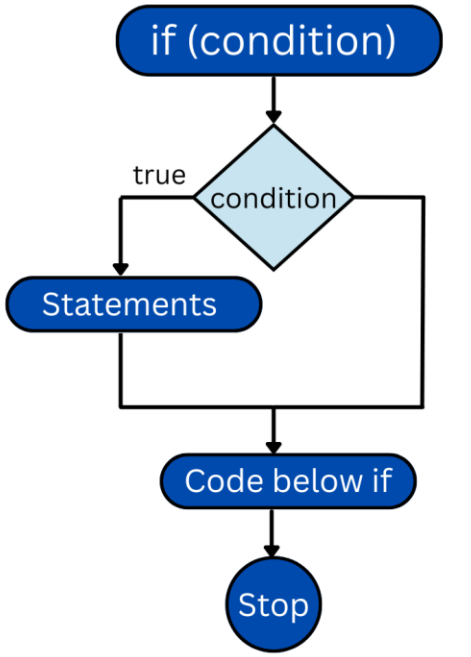
Condition Statement, Looping Statement Arrays

Uday M
Assistant Professor
Department of Computer Science
Seshadripuram Degree College

Conditional Statements

- if statement
- if...else statement
- if...elseif...else
- switch statement



Behaviour of an if statement

The if Statement

The if statement is used to execute a block of code only if the specified condition evaluates to true.

Syntax

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

Example:

```
<?php
$today = "Friday";
 if ($today == "Friday") {
echo "Happy Friday!";
 }
?>
```

The if...else Statement

The if...else statement executes one block of code if the specified condition is evaluated as to true and another block of code if it is evaluated to be false.

Syntax

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

Example:

```
<?php

$mytime = 12;

if ($mytime < 20) {
   echo "Good day!";
} else {
   echo "Good night!";
}

?>
```

The if...elseif...else Statement

The if...elseif...else statement executes different codes for more than two conditions.

Syntax

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

Example:

```
<?php

$mytime = 8;

if ($mytime < 10) {
   echo "Good morning!";
} elseif ($mytime < 20) {
   echo "Good day!";
} else {
   echo "Good night!";
}
?>
```

The PHP switch Statement

The switch statement is used to select one of many blocks of code to be executed. This is an alternative to the if...elseif...else statement.

Syntax

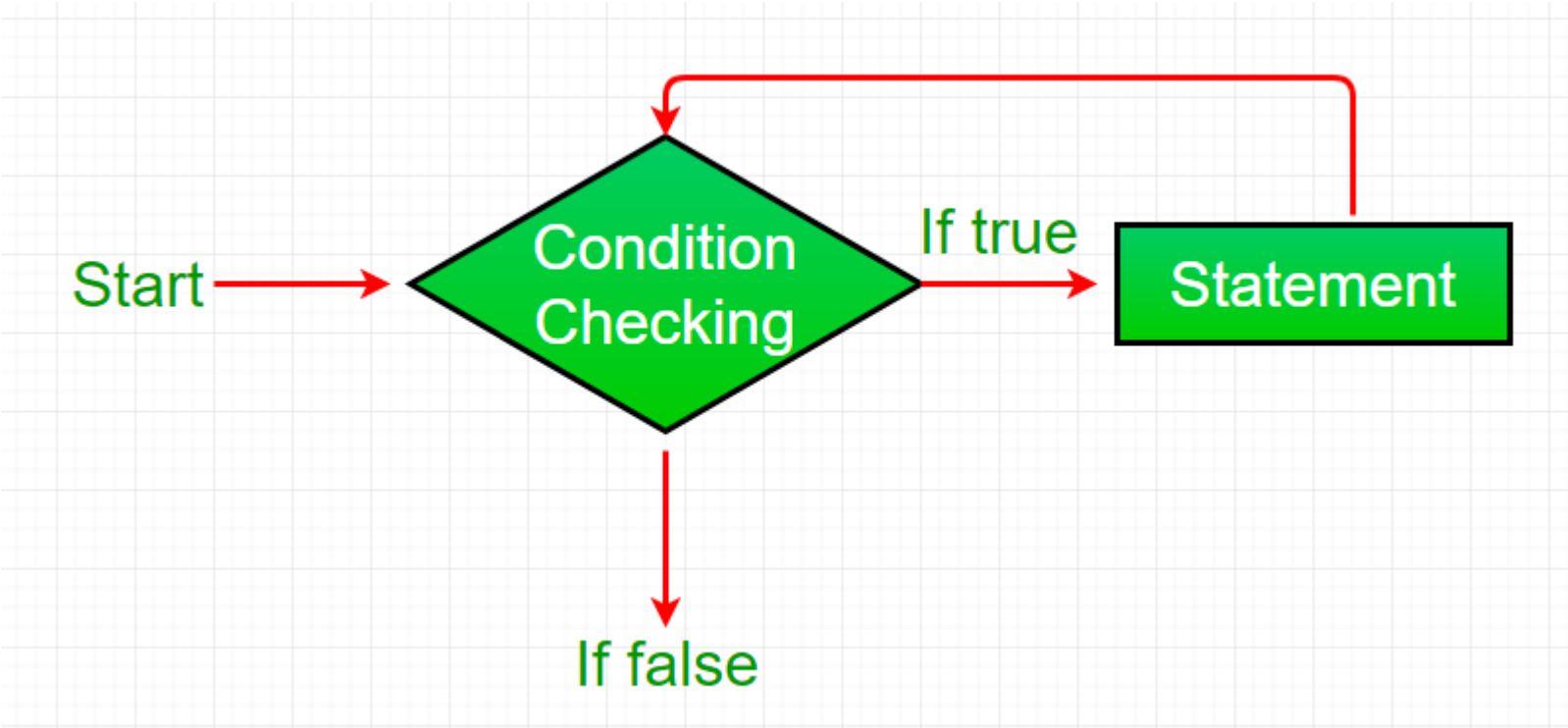
```
switch (expression) {
  case label1:
    //code block
    break;
  case label2:
    //code block;
    break;
  case label3:
    //code block
    break;
  default:
    //code block
}
```

Example:

```
<?php

switch (n) {
    case label1:
        code to be executed if n=label1;
        break;
    case label2:
        code to be executed if n=label2;
        break;
    case label3:
        code to be executed if n=label3;
        break;
    ...
    default:
        code to be executed if n is different from all
labels;
}
?>
```

Permanently Affiliated to the University of Mysore, Approved by AICTE
ISO 9001:2015 Certified | NAAC Accredited B++

- while
- do...while
- for
- foreach



The PHP while Loop

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

Syntax

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

Example:

```
<?php

$i = 1;

while($i <= 10) {
    echo " $i <br>";
    $i++;
}

?>
```

The PHP DO...WHILE LOOP

The do...while loop is a variant of while loop. It evaluates the condition at the end of each loop iteration. Thus, a do-while loop the block of code is executed once before the condition is evaluated (this is its difference from the while loop). If the condition is true, the block of code is repeated as long as the specified condition is true.

Syntax

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

Example:

```
<?php

$i = 1;

do {
    echo $i . "<br>";
    $i++;
} while ($i <= 10);

?>
```

The PHP FOR LOOP

The for loop repeatedly executes a block of code as long as a certain condition is met. It is used mostly to execute a block of code for certain number of times.

Syntax

```
for (init counter; test counter; increment counter) {
    code to be executed for each iteration;
}
```

Example:

```
<?php

for ($i = 0; $i <= 10; $i++) {
    echo "The number is: $i <br>";
}

?>
```

The PHP FOREACH LOOP

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

Syntax

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

Example:

```
<?php

$colors = array("red", "blue", "yellow");

foreach ($colors as $value) {
    echo "$value <br>";
}

?>
```

Permanently Affiliated to the University of Mysore, Approved by AICTE
ISO 9001:2015 Certified | NAAC Accredited B++

Break

The break statement can be used to exit out of a loop.

Example:

```
<?php

for ($i = 0; $i <= 10; $i++) {
    if ($i == 5) {
        break;
    }
    echo "The number is: $i <br>";
}

?>
```

Continue

The continue statement skips one iteration of the loop, if a specified condition occurs, and continues with the next iteration in the loop.

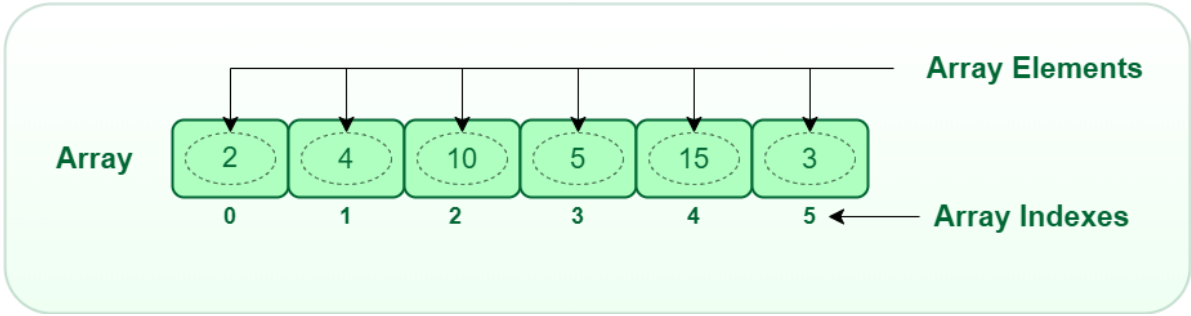
Example:

```
<?php

for ($i = 0; $i < 10; $i++) {
    if ($i == 5) {
        continue;
    }
    echo " $i <br>";
}

?>
```

Permanently Affiliated to the University of Mysore, Approved by AICTE
ISO 9001:2015 Certified | NAAC Accredited B++

In PHP, an array is a special variable that can hold many values under a single name, and you can access the values by referring to an index number or a name

In PHP, there are three types of arrays:



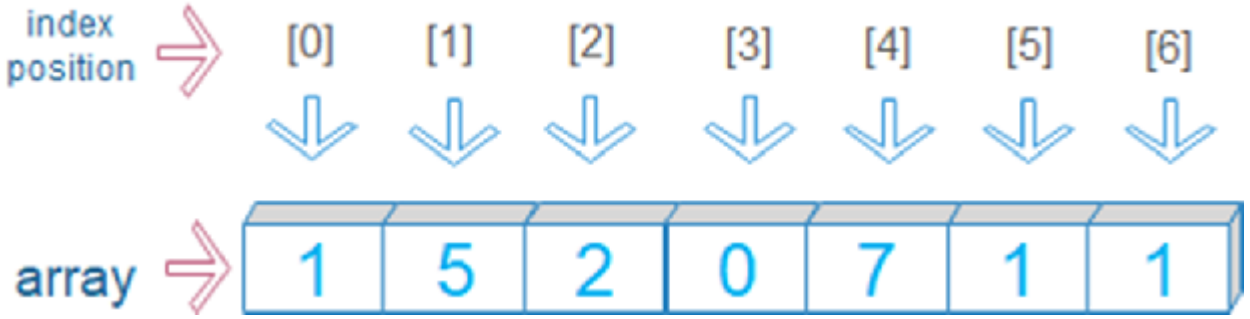
Indexed arrays - Arrays with a numeric index

Associative arrays - Arrays with named keys

Multidimensional arrays - Arrays containing one or more array

Indexed Arrays

Indexed arrays use numeric indexes starting from 0. These arrays are ideal when you need to store a list of items where the order matters

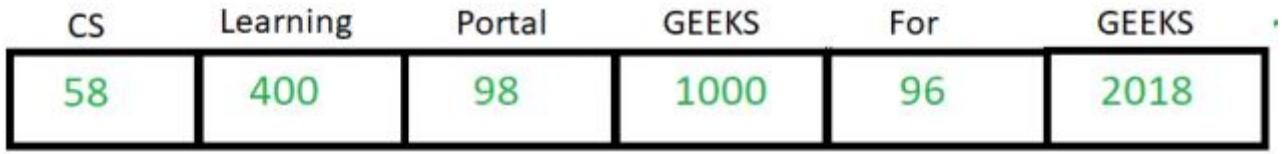


Example:

```
<?php

$cars = array("Volvo", "BMW", "Toyota");
var_dump($cars);

?>
```

Permanently Affiliated to the University of Mysore, Approved by AICTE
ISO 9001:2015 Certified | NAAC Accredited B++

Associative arrays use named keys, which are useful when you want to store data with meaningful identifiers instead of numeric indexes.

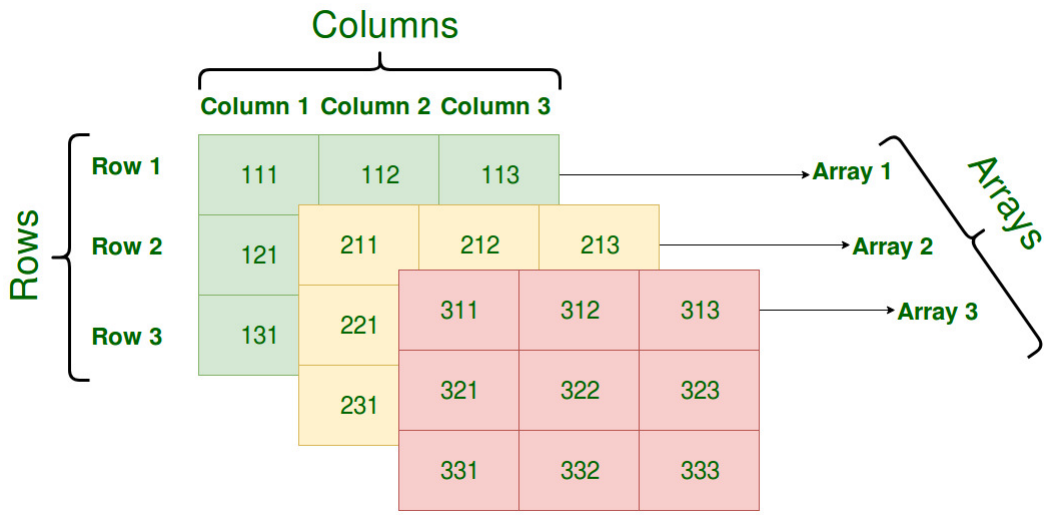


Example:

```
<?php

    $person = array("name" => "GFG", "age" => 30,
"city" => "New York");
echo $person["name"];

?>
```

Permanently Affiliated to the University of Mysore, Approved by AICTE
ISO 9001:2015 Certified | NAAC Accredited B++

Multidimensional arrays are arrays that contain other arrays as elements. These are used to represent more complex data structures, such as matrices or tables.



Example:

```
<?php

$students = array(
    "Anjali" => array("age" => 25, "grade" => "A"),
    "GFG" => array("age" => 22, "grade" => "B")
);
echo $students["GFG"]["age"];

?>
```

Permanently Affiliated to the University of Mysore, Approved by AICTE
ISO 9001:2015 Certified | NAAC Accredited B++

You can access individual elements in an array using their index (for indexed arrays) or key (for associative arrays).

Accessing Indexed Array:

```
$fruits = ["apple", "banana", "cherry"];
echo $fruits[0]; // Outputs: apple
```

Accessing Associative Array:

```
$person = ["name" => "GFG", "age" => 30];
echo $person["name"]; // Outputs: GFG
```

You can modify an existing element by assigning a new value to a specific index or key.

Modifying Indexed Array:

```
$fruits = ["Apple", "Banana", "Cherry"];
$fruits[1] = "Mango";  // Changes "Banana" to "Mango"
echo $fruits[1];  // Outputs: Mango
```

Modifying Associative Array:

```
$person = ["name" => "GFG", "age" => 25];
$person["age"] = 26;  // Updates the age to 26
echo $person["age"];  // Outputs: 26
```

array_push(): Adds elements to the end of an indexed array.

```
$fruits = ["apple", "banana"];
array_push($fruits, "cherry"); // Adds "cherry" to the end
```

array_unshift(): Adds elements to the beginning of an indexed array.

```
$fruits = ["apple", "banana"];
array_push($fruits, "cherry"); // Adds "cherry" to the end
```

Direct assignment: Adds an element to an associative array.

```
$person["city"] = "New York";
```

array_pop(): Removes the last element from an indexed array.

```
array_pop($fruits);
```

array_shift(): Removes the first element from an indexed array.

```
array_shift($fruits);
```

unset(): Removes a specific element from an array by key or index.

```
unset($fruits[2]); // Removes the element with index 2
```

Array Merge: The array_merge() function combines two or more arrays into one.

```
$array1 = [1, 2, 3];
$array2 = [4, 5, 6];
$merged = array_merge($array1, $array2);
print_r($merged);  // Outputs: [1, 2, 3, 4, 5, 6]
```

Array Search: The in_array() function checks if a specific value exists in an array.

```
$fruits = ["Apple", "Banana", "Cherry"];
if (in_array("Banana", $fruits)) {
    echo "Banana is in the array!";
}
```

Array Sort: The sort() function sorts an indexed array in ascending order.

```
$numbers = [3, 1, 4, 1, 5];
sort($numbers);
print_r($numbers);  // Outputs: [1, 1, 3, 4, 5]
```

Array_count_values() : The array_count_values() function counts all the values of an array.

```
$a=array("A","Cat","Dog","A","Dog");
print_r(array_count_values($a));
```

Array_fill(): The array_fill() function fills an array with values.

```
$a1=array_fill(3,4,"blue");
print_r($a1);
```

array_intersect() : The array_intersect() function compares the values of two (or more) arrays, and returns the matches.

```
$a1=array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow");
$a2=array("e"=>"red","f"=>"green","g"=>"blue");

$result=array_intersect($a1,$a2);
print_r($result);
```

array_replace() : The array_replace() function replaces the values of the first array with the values from following arrays.

```
$a1=array("red","green");
$a2=array("blue","yellow");
print_r(array_replace($a1,$a2));
```

array_search() : The array_search() function search an array for a value and returns the key.

```
$a=array("a"=>"red","b"=>"green","c"=>"blue");
echo array_search("red",$a);
```

array_slice(): The array_slice() function returns selected parts of an array.

```
$a=array("red","green","blue","yellow","brown");
print_r(array_slice($a,2));
```

array_sum(): The array_sum() function returns the sum of all the values in the array.

```
$a=array(5,15,25);
echo array_sum($a);
```

array_unique(): The array_unique() function removes duplicate values from an array. If two or more array values are the same, the first appearance will be kept and the other will be removed.

```
$a=array("a"=>"red","b"=>"green","c"=>"red");
print_r(array_unique($a));
```

Permanently Affiliated to the University of Mysore, Approved by AICTE
ISO 9001:2015 Certified | NAAC Accredited B++

count() : The count() function returns the number of elements in an array or in a countable object.

```
$cars=array("Volvo","BMW","Toyota");
echo count($cars);
```

pos() **:** The pos() function returns the value of the current element in an array.

```
$people = array("Peter", "Joe", "Glenn", "Cleveland");
echo pos($people) . "<br>";
```

reset()**:** The reset() function moves the internal pointer to the first element of the array.

```
$people = array("Peter", "Joe", "Glenn", "Cleveland");
echo current($people) . "<br>";
echo next($people) . "<br>";
echo reset($people);
```

sizeof()**:** The sizeof() function returns the number of elements in an array.

```
$cars=array("Volvo","BMW","Toyota");
echo sizeof($cars);
```

Require()

The **require()** function in PHP is basically used to include the contents/code/data of one PHP file to another PHP file. During this process, if there are any kind of errors then this **require()** function will pop up a warning along with a fatal error and it will immediately stop the execution of the script.

```
<html>
<body>
    <h1>Welcome to geeks for geeks!</h1>
    <p>Myself, Gaurav Gandal</p>
    <p>Thank you</p>
    <?php include 'GFG.php'; ?>
</body>
</html>
```

```
<?php

   echo "<p>Visit Again; " . date("Y") . " Geeks for geeks.com</p>";

?>
```

Include()

The **include()** function in PHP is basically used to include the contents/code/data of one PHP file to another PHP file. During this process if there are any kind of errors then this **include()** function will pop up a warning but unlike the **require()** function, it will not stop the execution of the script rather the script will continue its process.

```
<html>
<body>
    <h1>Welcome to geeks for geeks!</h1>
    <p>Myself, Gaurav Gandal</p>
    <p>Thank you</p>
    <?php include 'GFG.php'; ?>
</body>
</html>
```

```
<?php

   echo "<p>Visit Again; " . date("Y") . " Geeks for geeks.com</p>";

?>
```

Include vs Require

include()	require()
The include() function does not stop the execution of the script even if any error occurs.	The require() function will stop the execution of the script when an error occurs.
The include() function does not give a fatal error.	The require() function gives a fatal error
The include() function is mostly used when the file is not required and the application should continue to execute its process when the file is not found.	The require() function is mostly used when the file is mandatory for the application.
The include() function will only produce a warning (E_WARNING) and the script will continue to execute.	The require() will produce a fatal error (E_COMPILE_ERROR) along with the warning and the script will stop its execution.
The include() function generate various functions and elements that are reused across many pages taking a longer time for the process completion.	The require() function is more in recommendation and considered better whenever we need to stop the execution incase of availability of file, it also saves time avoiding unnecessary inclusions and generations.

Type Casting

Type casting is a concept in programming where you change the data type of a variable from one type to another. It's like changing a piece of clay from one shape to another.

There are two types of type casting: implicit and explicit.

1. Implicit Type Casting (Coercion)
2. Explicit Type Casting (Casting)

Implicit Type Casting

In implicit type casting, the programming language automatically converts data from one type to another if needed.

```
$x = 3 + "10 items"; // "10 items" is converted to the number 10. Result: 13 (int)
$y = 5 * "2";        // "5" and "2" are converted to integers. Result: 10 (int)
$z = 5 + null;       // null is converted to 0. Result: 5 (int)
```

Explicit Type Casting

Explicit type casting, also known as type conversion or type coercion, occurs when the programmer explicitly converts a value from one data type to another.

The PHP casting operators are:

(string) - Converts to data type String

(int) - Converts to data type Integer

(float) - Converts to data type Float

(bool) - Converts to data type Boolean

(array) - Converts to data type Array

```
$a = 5;
$a = (string) $a;
```