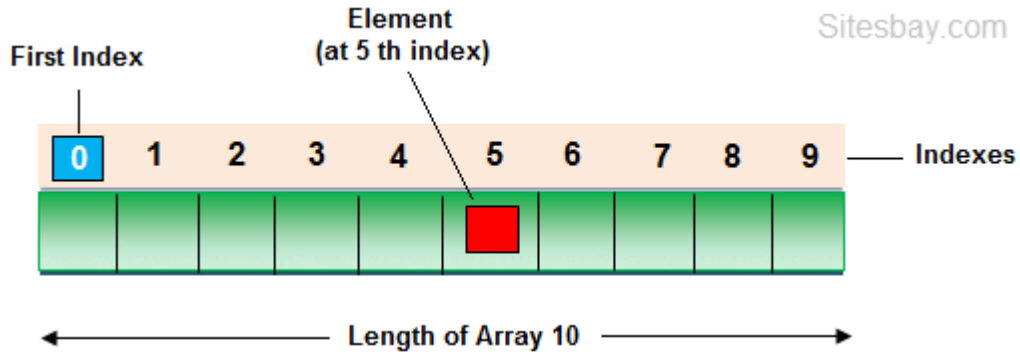
Unit 2: Arrays in PHP: Definition, Creating, Accessing Array**, Types of Arrays:** Indexed, Associative arrays, Multidimensional arrays, Accessing Array, Manipulating Arrays, displaying array, Array Functions.
Using Functions in PHP: Definition, Creating, invoking, user-defined functions, Formal parameters, actual parameters, Function and variable scope, Recursion, Library functions, Date and Time Functions.
Strings in PHP: Definition, Creating, Declaring, formatting strings, String Functions.

PHP Arrays
Arrays are one+ of the most important data structures in PHP. They allow you to store multiple values in a single variable. PHP arrays can hold values of different types, such as strings, numbers, or even other arrays. Understanding how to use arrays in PHP is important for working with data efficiently.

- PHP offers many built-in array functions for sorting, merging, searching, and more.
- PHP Arrays can store values of different types (e.g., strings, integers, objects, or even other arrays) in the same array.
- They are dynamically sized.
- They allow you to store multiple values in a single variable, making it easier to manage related data.

Array is used to store multiple values of same type in single variable. An array is a special variable, which can hold more than one value at a time.



Advantage of PHP Array:

1. **Less Code**: We don't need to define multiple variables.
2. **Easy to traverse**: By the help of single loop, we can traverse all the elements of an array.
3. **Sorting**: We can sort the elements of array.

What is an Array?

A variable is a storage area holding a number or text. The problem is, a variable will hold only one value.An array is a special variable, which can store multiple values in one single variable.If you have a list of items(a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1="Saab";

$cars2="Volvo";

$cars3="BMW";
```

Creating an array:

In PHP, you can create an array using the array() function or by using square brackets [] sincePHP 5.4. Here are examples of both methods:

Using the array() function:

$array1 = array(1, 2, 3, 4, 5); // Numeric array

$array2 = array("apple", "banana", "orange"); // Associative array

$array3 = array("a" => 1, "b" => 2, "c" => 3); // Associative arraywith keys

Using square brackets []:

$array4 = [1, 2, 3, 4, 5]; // Numeric array

$array5 = ["apple", "banana", "orange"]; // Associative array

$array6 = ["a" => 1, "b" => 2, "c" => 3]; // Associative array with keys

In PHP, arrays can hold any type of data, including other arrays, objects, and even a combination of different types of data.

Accessing array elements:

In PHP, you can access array elements using square brackets [] with the index of the element you want to access. Array indexing starts at 0 for the first element. Here are examples:

$array = array("apple", "banana", "orange", "grape");

echo $array[0]; // Outputs: apple

echo $array[1]; // Outputs: banana

echo $array[2]; // Outputs: orange

echo $array[3]; // Outputs: grape

You can also access elements in associative arrays using keys:

$assocArray = array("a" => "apple", "b" => "banana", "c" => "orange");

echo $assocArray["a"]; // Outputs: apple

echo $assocArray["b"]; // Outputs: banana

echo $assocArray["c"]; // Outputs: orange

If you try to access an element that does not exist, PHP will generate a notice or warning, depending on your error reporting settings. To avoid such notices, you can use **isset()**to check if the element exists before accessing it:

if (isset($array[4]))

{

echo $array[4];

}

Else

{

```
echo "Element does not exist.";
}
```

This prevents PHP from throwing a notice if the element doesn't exist.

Types of arrays:

In PHP, there are three kind of arrays:

- Numeric/indexed array -An array with a numeric index
- Associative array -An array where each ID key is associated with a value
- Multidimensional array -An array containing one or more arrays

Numeric/indexed Arrays:

A numeric array stores each array element with a numeric index.

There are two methods to create a numeric array.

1. In the following example the index are automatically assigned (the index starts at 0):

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

2. In the following example we assign the index manually:

```
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
```

Example

In the following example you access the variable values by referring to the array name and index:

```
<?php
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
echo $cars[0] . " and " . $cars[1] . "are Swedish cars.";
?>
```

The code above will output:

Saab and Volvo are Swedish cars.

Associative Arrays

An associative array, each ID key is associated with a value.

When storing data about specific named values, a numerical array is not always the best way to do

it.

With associative arrays we can use the values as keys and assign values to them.

Example 1

In this example we use an array to assign ages to the different persons:

```
$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);
```

Example 2

This example is the same as example 1, but shows a different way of creating the array:

```
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
```

The ID keys can be used in a script:

```
<?php
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
echo "Peter is " . $ages['Peter'] . " years old.";
?>
```

The code above will output:

Peter is 32 years old.

Multidimensional Arrays

In a multidimensional array, each element in the main array can also be an array. And each element in the sub-array can be an array, and so on.

Example

In this example we create a multidimensional array, with automatically assigned ID keys:

```
$families = array
(
"Griffin"=>array
(
"Peter",
"Lois",
"Megan"
),
```

```
"Quagmire"=>array
(
"Glenn"
),
"Brown"=>array
(
"Cleveland",
"Loretta",
"Junior"
)
);
```

The array above would look like this if written to the output:

```
Array
(
[Griffin] => Array
(
[0] => Peter
[1] => Lois
[2] => Megan
)
[Quagmire] => Array
(
[0] => Glenn
)
[Brown] => Array
(
[0] => Cleveland
[1] => Loretta
[2] => Junior
)
)
```

Example 2

Displaying a single value from the array above:

echo "Is " . $families['Griffin'][2] .

" a part of the Griffin family?";

The code above will output:

Is Megan a part of the Griffin family?

Manipulating arrays:

In PHP, you can manipulate arrays in various ways. Here are some common operations you can perform:

- **Creating an Array:**You can create an array in PHP using square brackets [] or the array() function.

$array = [1, 2, 3, 4, 5]; // Using square brackets

$array = array(1, 2, 3, 4, 5); // Using array() function

- **Adding Elements to an Array**: You can add elements to an array using square bracket notation or by using the array_push() function.

$array[] = 6; // Adding element using square bracket notation

array_push($array, 7); // Using array_push() function

- **Removing Elements from an Array:**You can remove elements from an array using the unset() function or by using array functions like array_pop(), array_shift(), or array_splice().

unset($array[0]); // Removing element by index

array_pop($array); // Removing the last element

array_shift($array); // Removing the first element

array_splice($array, 2, 1); // Removing elements from a specific index

- **Accessing Array Elements:**You can access array elements by their index.
  echo $array[0]; //Accessing the first element

- **Iterating Through an Array:**You can iterate through arrays using loops like foreach, for, while, etc.

foreach ($array as $value) {

echo $value . ' ';

}

- **Merging Arrays:**You can merge arrays using the array_merge() function.

$mergedArray = array_merge($array1, $array2);

- **Sorting Arrays:**You can sort arrays using functions like sort(), rsort(), asort(), arsort(), ksort(), krsort(), etc.

In PHP, you can sort arrays using various functions depending on your requirements. Here are some

commonly used functions for sorting arrays:

- **sort():**Sorts an array in ascending order. The keys are not preserved.

```
$array = [3, 1, 4, 1, 5, 9, 2, 6];

sort($array);

print_r($array); // Output: [1, 1, 2, 3, 4, 5, 6, 9]
```

- **rsort():**Sorts an array in descending order. The keys are not preserved.The rsort() function sorts an indexed array in descending order.

```
$array = [3, 1, 4, 1, 5, 9, 2, 6];

rsort($array);

print_r($array); // Output: [9, 6, 5, 4, 3, 2, 1, 1]
```

- **asort():**Sorts an array in ascending order, preserving the keys.The asort() function sorts an associative array in ascending order, according to the value.

```
$array = ['b' => 3, 'a' => 1, 'c' => 2];

asort($array);

print_r($array); // Output: ['a' => 1, 'c' => 2, 'b' => 3]
```

- **arsort():**Sorts an array in descending order, preserving the keys.The arsort() function sorts an associative array in descending order, according to the value

```
$array = ['b' => 3, 'a' => 1, 'c' => 2];

arsort($array);

print_r($array); // Output: ['b' => 3, 'c' => 2, 'a' => 1]
```

- **ksort():**Sorts an array by keys in ascending order.The ksort() function sorts an associative array in ascending order, according to the key.

```
$array = ['b' => 3, 'a' => 1, 'c' => 2];

ksort($array);

print_r($array); // Output: ['a' => 1, 'b' => 3, 'c' => 2]
```

- **krsort():**Sorts an array by keys in descending order.The ksort() function sorts an associative array in ascending order, according to the key.

```
$array = ['b' => 3, 'a' => 1, 'c' => 2];

krsort($array);

print_r($array); // Output: ['c' => 2, 'b' => 3, 'a' => 1]
```

These functions allow you to sort arrays based on different criteria while preserving or reordering keys as needed.

Displaying arrays:

In PHP, you can display the contents of an array using various methods, depending on how you want

to present the data. Here are a few common ways to display arrays:

- **Using print_r():**The print_r() function is commonly used to display the contents of an array in a human-readable format.

```
$array = [1, 2, 3, 4, 5];
print_r($array);
```

Output:

```
Array
(
[0] => 1
[1] => 2
[2] => 3
[3] => 4
[4] => 5
)
```

- **Using var_dump():**The var_dump() function provides more detailed information about the array, including data types and lengths

```
$array = [1, 2, 3, 4, 5];
var_dump($array);
```

Output:

```
array(5) {
[0]=>int(1)
[1]=>int(2)
[2]=>int(3)
[3]=>int(4)
[4]=>int(5)
}
```

- **Using foreach Loop:**You can iterate through the array using a foreach loop and display each element individually.

```
$array = [1, 2, 3, 4, 5];
foreach ($array as $value) {
echo $value . ' ';
}
```

Output:

1 2 3 4 5

- **Using implode():**You can use the implode() function to join array elements into a string and then display the string.

$array = [1, 2, 3, 4, 5];

echo implode(', ', $array);

Output:

1, 2, 3, 4, 5

These methods offer flexibilityin how you display array data, depending on your specific

requirements and the context in which you are working.

Using array functions

PHP provides a variety of array functions that you can use to manipulate arrays efficiently. Here are some commonly used array functions in PHP:

- **array():** Creates an array.

$array = array('foo', 'bar', 'baz');

- **count():**Returns the number of elements in an array.

$count = count($array);

- **array_push():**Pushes one or more elements onto the end of an array.

array_push($array,'qux');

- **array_pop():**Pops the element off the end of array.

$lastElement = array_pop($array);

- **array_shift():**Shifts the first value of the array off and returns it.

$firstElement = array_shift($array);

- **array_unshift():**Prepend one or more elements to the beginning of an array.

array_unshift($array, 'quux');

- **array_merge():**Merge one or more arrays.

$mergedArray = array_merge($array1, $array2);

- **array_slice():**Extract a slice of the array.

$slice = array_slice($array, 2, 2);

- **array_splice():**Remove a portion of the array and replace it with something else.

array_splice($array, 2, 1, 'replacement');

- **array_search():**Searches the array for a given value and returns the corresponding key if successful.

$key = array_search('bar', $array);

Find Length of an Array in PHP

Using count() function you can find length of an array in php.

Example

```
<?php
$student = array("Harry", "Varsha", "Gaurav"); echo
"Length of Array: ";
echo count($student);
?>
```

Output: Length of Array: 3

array_change_key_case()

Changes all keys in an array to lowercase or uppercase

Syntax:

array_change_key_case(array $array[, int $case = CASE_LOWER])

Example

```
<?php
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"2000
00");
print_r(array_change_key_case($salary,CASE_UPPER));
?>
```

Output:

Array ([SONOO] => 550000 [VIMAL] => 250000 [RATAN] => 200000)

array_chunk(): Splits an array into chunks of arrays

Syntax: array_chunk (array $array , int $size)

Example

```
<?php
$salary=array("Sonoo"=>"550000","Vimal"=>"250000","Ratan"=>"2000
00");
print_r(array_chunk($salary,2));
?>
```

Output:

```
Array (
[0] => Array ( [0] => 550000 [1] => 250000 )
```

[1] => Array ([0] => 200000)

)

ARRAY FUNCTIONS

Function	Description
array()	Creates an array
array_combine()	Creates an array by using the elements from one "keys" array and one "values" array
array_count_values()	Counts all the values of an array
array_diff()	Compare array and return the differences (compare values only)
array_diff_assoc()	Compare array and return the differences (compare keys and values)
array_diff_key()	Compare array and return the differences (compare keys only)
array_fill()	Fills an array with values
array_fill_keys()	Fills an array with values, specifying keys
array_filter()	Filters the value of an array using a callback function
array_flip()	Flips/Exchanges all keys with their associated values in an array
array_intersect()	Compare arrays and returns the matches

Example

```
<?php
$name1=array("sonoo","john","vivek","smith");
$name2=array("umesh","sonoo","kartik","smith");
$name3=array_intersect($name1,$name2)
;    foreach( $name3 as $n )
{
  echo "$n<br />";
}
?>
```

Output: sonoo

smith

Function	*Description*
array_intersect_assoc()	Compare arrays and returns the matches (compare keys and values)
array_intersect_key()	Compare arrays, and returns the matches (compare keys only)
array_intersect_uassoc( )	Compare arrays, and returns the matches(compare keys and values, using a user-defined key comparison function)

array_intersect_ukey()	Compare arrays, and returns the matches(compare keys only, using a user-defined key comparison function)
array_key_exists()	Checks if the specified key exists in the array
array_keys()	Returns all the keys of an array
array_map()	Sends each value of an array to a user-made function, which returns new values
array_merge()	Merges one or more arrays into one array
array_merge_recursive()	Merges one or more arrays into one array recursively
array_multisort()	Sorts multiple or multi-dimensional arrays
array_pad()	insert a specified number of items, with a specified value, to an array
array_pop()	Deletes the last element of an array
array_product()	Calculates the product of the values in an array
array_push()	Inserts one or more elements to the end of an array
array_rand()	Returns one or more random keys from an array
array_reduce()	Returns an array as a string, using a user defined function
array_replace()	Replaces the values of the first array with the values from following arrays
array_replace_recursive( )	Replaces the values of the first array with the values from following arrays recursively
array_reverse()	Returns an array in the reverse order

Example

```
<?php
$season=array("summer","winter","spring","autumn");
$reverseseason=array_reverse($season); foreach( $reverseseason as $s )
{
echo "$s<br />";
}
?>
```

Output: autumn spring winter summer

array_search(): Searches an array for a given value and returns the key

Example

```
<?php
$season=array("summer","winter","spring","autumn");
$key=array_search("spring",$season); echo $key;
```

?>

Output: **2**

Function	Description
array_unique()	Removes duplicate values from an array
array_unshift()	add one or more elements to the beginning of an array
array_values()	Returns all the values of an array
count()	Returns the number of elements in an array
current()	Returns the current element in an array
each()	Returns the current key and value pair from an array
end()	Sets the internal pointer of an array to its last element
extract()	Imports variables into the current symbol table from an array
in_array()	Checks if a specified value exists in an array
key()	Fetches a key from an array
list()	Assigns variables as if they were an array
range()	Creates an array containing a range of elements
reset()	Sets the internal pointer of an array to its first element
shuffle()	Shuffles an array
sizeof()	Alias of count()
uksort()	Sorts an array by keys using a user-defined comparison function
usort()	Sorts an array using a user-defined comparison function

natsort(): The natsort() function is used to sorts an array using a "natural order" algorithm. The function implements a sort algorithm but maintains original keys/values.

This function implements a sort algorithm that orders alphanumeric strings in the way a human being would while maintaining key/value associations.

Syntax: natsort(array_name)

Example:

```
<?php
$php_files = array("code12.php", "code22.php", "code2.php", "Code3.php", "code1.php");
natsort($php_files);
echo "List of file using natural order: "; print_r($php_files);
?>
```

Output: List of file using natural order:

```
(
[3] => Code3.php
```

[4] => code1.php

[2] => code2.php

[0] => code12.php

[1] => code22.php

)

natcasesort(): The natcasesort() function is used to sort an array using a case insensitive "natural array" algorithm. The function implements a sort algorithm but maintains original keys/values.

natcasesort() is a case insensitive version of natsort()

Syntax: natcasesort(array_name)

Example:

```
<?php
$php_files = array("code12.php", "code22.php", "code2.php", "Code3.php", "code1.php");
natcasesort($php_files);
echo "List of file using natural order: "; print_r($php_files);
?>
```

Output: List of file using natural order:

Array

(

[4] => code1.php

[2] => code2.php

[3] => Code3.php

[0] => code12.php

[1] => code22.php

)

usort():

The usort() function sorts an array by a user defined comparison function. This function assigns new keys for the elements in the array. Existing keys will be removed.

Syntax

```
usort ( $array, $cmp_function )
```

Parameters

Sr.No	Parameter & Description
1	**array(Required)** It specifies an array.
2	**cmp_function(Required)** Useful defined function to compare values and to sort them. If a = b, return 0 If a > b, return 1 If a < b, return -1

Example

```
<?php
function cmp_function($a, $b) { if ($a == $b) return 0;
return ($a > $b) ? -1 : 1;
}
$fruits = array("d"=>"lemon", "a"=>"orange", "b"=>"banana" );
usort($fruits, "cmp_function"); print_r($fruits);
?>
```

Output: Array ([0] => orange [1] => lemon [2] => banana)

extract():it converts array keys into variable names and array values into variable value.

Syntax: extract(array,extract_rules,prefix) Example:

```
<?php
$state = array("AS"=>"ASSAM", "OR"=>"ORRISA", "KR"=>"KERELA");
extract($state);
echo" $AS is $AS\n $KR is $KR\n $OR is $OR";
?>
```

Output: $AS is ASSAM

$KR is KERELA

$OR is ORRISA

Example:

```
<?php
$AS="Original";
```

```
$state = array("AS"=>"ASSAM", "OR"=>"ORRISA", "KR"=>"KERELA");
extract($state, EXTR_PREFIX_SAME, "dup");
echo"\$AS is $AS\n $KR is $KR\n $OR if $OR \n $dup_AS = $dup_AS";
?>
```

Output: $AS is Original

$KR is KERELA

$OR is ORRISA

$dup_AS = ASSAM

current(): The current() function is used to fetch the value of the current element in an array.

Syntax: current(array_name)

Example:

```
<?php
$subject= array('Language','English','Math','Science');
$result = current($subject); echo $result;
?>
```

Output: Language

in_array(): The in_array() function is used to check whether a value exists in an array or not.

Syntax: in_array(search_value, array_name, mode)

Parameters:

Name	Description	Required / Optional	Type
search_value	Value to search in the array.	Required	Mixed*
array_name	Specifies the array to search.	Required	Array
mode	If it is set to true, the function checks the type of the search_value.	Optional	Boolean

```
<?php
$number_list = array('16.10', '22.0', '33.45', '45.45'); if (in_array(22.0, $number_list))
```

```
{
echo "'22.0' found in the array";
}
?>
```

Output: '22.0' found in the array

next(): The next() function is used to advance the internal array pointer. next() behaves like current(), with one difference. It advances the internal array pointer one place forward before returning the element value.

Syntax: next(array_name)

Example:

```
<?php
$val1 = array('Language', 'Math', 'Science', 'Geography');
$cval = current($val1); echo "$cval <br />";
$cval = next($val1); echo "$cval <br />";
$cval = next($val1); echo "$cval <br />";
$cval= prev($val1); echo "$cval <br />";
$cval= end($val1); echo "$cval <br />";
?>
```

Output: Language Math Science Math Geography

Prev: The prev() function is used to fetch the array value in the previous place, pointed by the internal array pointer.

prev() function behaves just like next(), except it rewinds the internal array pointer one place instead of advancing it.

Syntax: prev(array_name)

```
<?php
$val1 = array('Language', 'Math', 'Science', 'Geography');
$cval = current($val1); echo "$cval <br />";
$cval = next($val1); echo "$cval <br />";
$cval = next($val1); echo "$cval <br />";
```

```
$cval = prev($val1); echo "$cval <br />";
?>
```

Output: Language Math Science Math

range();The range() function used to create an array containing a range of elements.

Syntax: range(low_value, high_value, step)

Parameters:

Name	Description	Required/ Optional	Type
low_value	Specify the lowest value.	Required	Mixed*
high_value	Specify the highest value.	Required	Mixed*
step	It is used as increment between elements. The default value is 1.	Optional	Integer/Float

Example:

```
<?php
$number_list = range(11,19); print_r ($number_list);
echo "<br /> ";
$number_list_step = range(11,19,2); print_r ($number_list_step);
echo "<br /> ";
$letter_list = range("u","z"); print_r ($letter_list);
?>
```

Output:

Array ([0] => 11 [1] => 12 [2] => 13 [3] => 14 [4] => 15 [5] => 16 [6] => 17 [7] => 18 [8] => 19) Array ([0] => 11 [1] => 13 [2] => 15 [3] => 17 [4] => 19) Array ([0] => u [1] => v [2] => w [3] => x [4] => y [5] => z)

reset(): The reset() function rewinds array's internal pointer to the first element and returns the value of the first array element

Syntax: reset ($array); Example

```
<?php
$input = array('foot', 'bike', 'car', 'plane');
$mode = end($input); print "$mode <br />";

$mode = reset($input); print "$mode <br />";
$mode = current($input); print "$mode <br />";
?>
```

Output: plane foot foot

end(): The end() function advances array's internal pointer to the last element, and returns its value.

Syntax: end ($array);

Example

```
<?php
$transport = array('foot', 'bike', 'car', 'plane');
$mode = current($transport); print "$mode <br />";
$mode = next($transport); print "$mode <br />";
$mode = current($transport); print "$mode <br />";
$mode = prev($transport); print "$mode <br />";
$mode = end($transport); print "$mode <br />";
$mode = current($transport); print "$mode <br />";
?>
```

Output : foot bike bike foot plane plane

Including and requiring files-use of include() and require()

PHP code can get cluttered, which means that if you want to later change something, it becomes a hard task to do. Include and require statements are two almost identical statements that help in an important aspect of coding, the organization of code, and making it more readable and flexible. The include/require statement copies all text, code or any other markup from one existing file to the file using the statement. In a simple viewpoint, do consider these statements like this:

The include and require statements are the same, except upon failure of code execution where:

• require will produce a fatal error (E_COMPILE_ERROR) and stop the script from executing

• include will only produce a warning (E_WARNING) and the script will continue

The syntax of these two statements is as follows:

```
<?php
include 'file.php'; // in case of include
require 'file.php'; // in case of require
?>
```

Let's now see a real-world example where we use a header and footer using include and require into our main file:

header.php

```
<header>
<nav>
<ul>
<li><a href="#">Home</a></li>
<li><a href="#">Profile</a></li>
<li><a href="#">About</a></li>
<li><a href="#">Contact</a></li>
</ul>
</nav>
</header>
// styling to make the menu look right
<style type="text/css">
ul, li {
list-style-type: none;
display: inline-block;
margin-right: 1em;
padding: 0;
}
</style>
```

footer.php

```
<footer>All Rights Reserved. Do not reproduce this page.</footer>
```

In our main file, we'll use require for the header.php and include for the footer.php file:

index.php

```
<?php require 'header.php'; ?>
<body>
<h2>Main Content Goes Here</h2>
</body>
<?php include 'footer.php' ?>
```

Implicit and Explicit Casting

In PHP, implicit and explicit casting refer to the conversion of data from one data type to another.

Implicit Casting: Implicit casting, also known as automatic type conversion, occurs when PHP automatically converts data from one type to another without any explicit instructions from the programmer. This typically happens when performing operations between different data types.

For example:

```
$num = 10; // integer
$str = "5"; // string
$result = $num + $str; // Implicitly casts "5" to integer and adds it to $num
echo $result; // Output: 15
```

In the above example, the string "5" is implicitly castto an integer when adding it to the integer variable **$num**.

Explicit Casting: Explicit casting, also known as manual type conversion, occurs when the programmer explicitly instructs PHP to convert data from one type to another using casting functions or type-casting syntax.

For example:

```
$str = "10"; // string
$num = (int)$str; // Explicitly cast $str to an integer
echo $num; // Output: 10
```

Here, **(int)**is used to explicitly cast the string **$str**to an integer. Similarly, you can use **(float)**, **(string)**, **(array)**, **(object)**, and **(bool)**for explicit casting to float, string, array, object, and boolean respectively.