Web Development with PHP
Practical - 3
PHP Array

Array :

In PHP, an array is a powerful and commonly used data structure that can store a collection of values or elements under a single variable name. An array can hold a mix of data types such as integers, strings, floats, and even other arrays, making it a versatile tool for handling and manipulating data. PHP provides numerous built-in array functions that can be used to sort, filter, and manipulate array data in various ways.
Arrays in PHP is a type of data structure that allows us to store multiple elements of similar data type under a single variable thereby saving us the effort of creating a different variable for every data. For example, you could create an array of integers, like [1, 2, 3, 4, 5] or of a mixed data type, like [false, 40, 'purple', $object, 'puppy'].

```
<?php
// Create an array with three elements
$fruits = ['apple', 'banana', 'orange'];

// Display the array
print_r($fruits);
?>

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

Advantages of PHP Array

- Flexibility:
  Arrays can store a mixture of different data types, including strings, numbers, and objects, making them a versatile tool for organizing and manipulating data.
- Efficient data storage:
  Arrays can store large amounts of data in a relatively small space, and they can be accessed quickly and easily using built-in PHP functions.
- Easy to use:
  PHP provides a wide range of functions for working with arrays, including functions for adding, deleting, and searching for elements. This makes it easy to manipulate array data and perform common operations quickly and efficiently.
- Iteration:
  Arrays can be easily iterated over using PHP's foreach loop, making it easy to loop through and process large sets of data.
- Customizable sorting:
  PHP provides a variety of built-in functions for sorting arrays, including sort(), asort(), and ksort(), which allow you to customize the way your data is sorted based on your needs.

- Support for multidimensional arrays:
  PHP allows you to create multidimensional arrays, which can store data in a hierarchical format, making it easier to organize and work with complex data sets.

The key can either be an int or a string. The value can be of any type. Additionally the following key casts will occur:

- Strings containing valid decimal ints, unless the number is preceded by a + sign, will be cast to the int type. E.g. the key "8" will actually be stored under 8. On the other hand "08" will not be cast, as it isn't a valid decimal integer.
- Floats are also cast to ints, which means that the fractional part will be truncated. E.g. the key 8.7 will actually be stored under 8.
- Bools are cast to ints, too, i.e. the key true will actually be stored under 1 and the key false under 0.
- Null will be cast to the empty string, i.e. the key null will actually be stored under "".
- Arrays and objects can not be used as keys. Doing so will result in a warning: Illegal offset type.
- If multiple elements in the array declaration use the same key, only the last one will be used as all others are overwritten.

Type of Array:

An array is created using an **array() fu**nction 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 arrays

Indexed Arrays : index will automatically be assigned, index is numeric .

Indexed or numeric arrays are the simplest and most common type of array in PHP. They use numeric keys to access and store values, where the first element of an indexed array has a key of 0, and subsequent elements have incremental numeric keys. Indexed arrays are commonly used in PHP to store and manipulate lists of related values, such as user data or product information. They are simple and efficient and can be easily manipulated using built-in array functions and operators. Here is an example of how to create and use an indexed array in PHP:

```
// Create an indexed array with four elements
$myArray = ['apple', 'banana', 'cherry', 'date'];

// Access elements of the array using their numeric keys
echo $myArray[0]; // Output: apple
echo $myArray[1]; // Output: banana
echo $myArray[2]; // Output: cherry
echo $myArray[3]; // Output: date

// Change the value of an element in the array
$myArray[1] = 'orange';
```

```
echo $myArray[1]; // Output: orange

// Add a new element to the end of the array
$myArray[] = 'berry';
echo $myArray[4]; // Output: berry

// Get the number of elements in the array
$numElements = count($myArray);
echo $numElements; // Output: 5

// Loop through the array and output each element
for ($i = 0; $i < count($myArray); $i++) {
  echo $myArray[$i] . ' ';
}
```

E.g

```
<?php
     // Define an indexed array
      $colors = array("White", "Yellow", "Red");
      //access individual element
      Echo $colors[1];

     //Looping through for loop echo "Looping using for:
     \n";
      for($n = 0; $n < 3; $n++) {
          echo $colors[$n], "\n";
      }
 ?>
```

Associative Array:

Associative arrays are a type of array in PHP that use named keys instead of numeric indices to access and store values. Each key is associated with a value, which can be of any data type. Associative arrays are useful for storing and retrieving data using human-readable keys, such as names or IDs. An array with strings as index. This stores element values in association with key values rather than in a strict linear index order.

To create an associative array in PHP, we need to assign values to named keys instead of numeric indices.

```
$person = [
  'name' => 'FName LName',
  'age' => 24,
  'email' => 'student@ddu.ac.in',
];
```

```
#---------------------------------
$person = [];
$person['name'] = 'FName LName';
$person['age'] = 24;
$person['email'] = 'student@ddu.ac.in';
#---------------------------------
<?php
 /* First method to create an associate array. */
 $cities = array("Jack" => London, "Tom" => Paris,
 "Mona" => Delhi);

 echo "Location of Jack is ". $cities['Jack'] .
 "<br />"; echo "Location of Tom is ".
 $cities['Tom']. "<br />"; echo "Location of
 Mona is ". $cities['Mona']. "<br />";

 /* Second method to create an array. */
 $cities['Jack'] = "UK";
 $cities['Tom'] = "Europe";
 $cities['Mona'] = "Asia";
 echo "Location of Jack is ". $cities['Jack'] . "<br
 />";
 echo "Location of Tom is ". $cities['Tom']. "<br
 />";
 echo "Location of Mona is ". $cities['Mona']. "<br
 />";

 // Using for each loop
 foreach($cities as $key => $value)
  { echo $key . "=>". "$value" ."<br>";
 }
?>
```

Manipulating an Associative Array:

We can manipulate an associative array using built-in PHP functions and operators. Here are some common operations:

- Adding a new key-value pair:

```
$myArray = [
  'name' => 'Student',
  'age' => 30,
];
$myArray['email'] = 'student@ddu.ac.in';
```

- Updating an existing value:

```
$myArray = [
```

```
  'name' => 'Student',
  'age' => 30,
];
$myArray['age'] = 31;
```

- Removing a key-value pair:

```
$myArray = [
  'name' => 'Student',
  'age' => 30,
];
unset($myArray['age']);
```

- Checking if a key exists:

```
$myArray = [
  'name' => 'Student',
  'age' => 30,
];
if (array_key_exists('name', $myArray)) {
  echo "The 'name' key exists in the array.";
}
```

- Looping through an associative array:

```
$myArray = [
  'name' => 'Student',
  'age' => 30,
];
foreach ($myArray as $key => $value) {
  echo $key. ' => ' . $value. '<br>';
}
```

Multidimensional Arrays

Multidimensional arrays in PHP are arrays that contain other arrays as elements. They are useful for storing complex data structures, such as tables or matrices, and allow us to access elements using multiple keys, which represent the indices of the nested arrays.

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. Values in the multi-dimensional array are accessed using multiple indexes. A multi-dimensional array can be indexed or associated or mixed.

```
$data = [
                ['HI', 20, 'X'],
                ['HELLO', 30, 'Y'],
                ['WORLD', 40, 'Z']
          ];
```

To access an element in a multidimensional array, we use multiple indices separated by brackets. For example, to access yash's age, we can use the following code:

```
echo $data[0][1]; // Output: 20
```

We can also use loops to iterate through the elements of a multidimensional array. For example, to print all the students' names and grades, we can use a nested foreach loop like this:

```
foreach ($data as $d) {
  foreach ($d as $value) {
    echo $value. ' ';
  }
  echo '<br>';
}
```

E.g.

```
<?php

    $cars = array (array("Volvo",22,18),
    array("BMW",15,13), array("Saab",5,2),  array("Land
    Rover",17,15));

    echo  $cars[0][0].": In stock: ".$cars[0][1].",

    sold:        ".$cars[0][2].".<br>";        echo

    $cars[1][0].":   In   stock:   ".$cars[1][1].",

    sold:        ".$cars[1][2].".<br>";        echo

    $cars[2][0].":   In   stock:   ".$cars[2][1].",

    sold:        ".$cars[2][2].".<br>";        echo

    $cars[3][0].":   In   stock:   ".$cars[3][1].",

    sold: ".$cars[3][2].".<br>";


?>

<?php

    for($row = 0; $row < 4; $row++){

      echo "<p><b>Row number $row</b></p>";

      echo "<ul>";

      for ($col = 0; $col < 3; $col++) {
```

```
        echo "<li>".$cars[$row][$col]."</li>";

       }

       echo "</ul>";

     }

?>
```

E.g

```
<?php

       $studrect = array(

            'roll1' => array( 'rollno' => 1 , 'physics' => 35 ,
            'Maths' => 30 ),

            'roll2' => array( 'rollno' => 2 , 'physics' => 32 ,
            'Maths' => 35 ) ,

           'roll3' => array( 'rollno' => 3 , 'physics' => 20 ,
           'Maths' => 25 )

        );

      echo "Marks for Physics is ".
     $studrect['roll1']['physics']."<br>";

?>

<?php

         // Accessing using for loop
        foreach ($studrect as $row)
         {
        foreach($row as $col) {
          echo $col ."<br>";
          }
         echo "\n";
         }

 ?>
<?php
        $keys = array_keys($studrect);
        $size = count($studrect);
        echo $size . "<br>";
```

```
        print_r($keys);

        for($i = 0 ; $i< $size ; $i++)
         {
            echo '<p style ="color:fuchsia;"> keys
    of' .$keys[$i].'   </p>';
            foreach($studrect[$keys[$i]] as $key => $value )
             echo $key . " = >". $value . "<br>";
         }

?>
```

Sparse Arrays

Sparse arrays are arrays that have gaps or missing keys between the existing elements. They are not commonly used in PHP but can be useful for storing large amounts of data with many empty elements.

```
$sparse_array = [1 => 'one', 3 => 'three', 5 => 'five'];
```

Dynamic Arrays

Dynamic arrays are arrays that can change size dynamically at runtime. In PHP, all arrays are dynamic by default, which means we can add or remove elements as needed using built-in functions and operators.

```
$dynamic_array = [];

$dynamic_array[] = 'apple';

$dynamic_array[] = 'banana';

$dynamic_array[] = 'cherry';
```

Array Functions:

The array function is a built-in feature in PHP, so installing it separately is unnecessary. Arrays are an essential data type in PHP, used to store a collection of values in a single variable. They can be created using the array() function or shorthand [] notation. Once created, you can perform various operations on the array, such as adding, removing, or modifying elements, sorting, filtering, and searching for values.

Functions:

- `array()` - creates a new array.
- `count()` - returns the number of elements in an array.
- `in_array()` - checks if a value exists in an array.
- `array_push()` - adds one or more elements to the end of an array.
- `array_pop()` - removes and returns the last element of an array.
- `array_shift()` - removes and returns the first element of an array.
- `array_unshift()` - adds one or more elements to the beginning of an array.
- `array_reverse()` - reverses the order of the elements in an array.
- `array_merge()` - merges two or more arrays into one array.
- `array_slice()` - extracts a slice of an array.
- `array_splice()` - removes and replaces elements from an array.
- `array_keys()` - returns an array of the keys of an array.
- `array_values()` - returns an array of the values of an array.
- `array_flip()` - flips the keys and values of an array.
- `array_unique()` - removes duplicate values from an array.
- `array_walk()` - applies a user-defined function to each element of an array.
- `array_map()` - applies a user-defined function to each element of an array and returns a new array with the results.
- `array_filter()` - filters an array using a user-defined function and returns a new array with the filtered values.
- `array_rand()` - returns one or more random keys from an array.
- `array()` - creates a new array.
- `count()` - returns the number of elements in an array.
- `in_array()` - checks if a value exists in an array.
- `array_push()` - adds one or more elements to the end of an array.

- array_pop() - removes and returns the last element of an array.
- array_shift() - removes and returns the first element of an array.
- array_unshift() - adds one or more elements to the beginning of an array.
- array_reverse() - reverses the order of the elements in an array.
- array_merge() - merges two or more arrays into one array.
- array_slice() - extracts a slice of an array.
- array_splice() - removes and replaces elements from an array.
- array_keys() - returns an array of the keys of an array.
- array_values() - returns an array of the values of an array.
- array_flip() - flips the keys and values of an array.
- array_unique() - removes duplicate values from an array.
- array_search() - searches for a value in an array and returns its key.
- array_walk() - applies a user-defined function to each element of an array.
- array_map() - applies a user-defined function to each element of an array and returns a new array with the results.
- array_filter() - filters an array using a user-defined function and returns a new array with the filtered values.
- array_reduce() - reduces an array to a single value using a user-defined function.
- array_column() - returns an array of values from a single column of a multidimensional array.
- array_intersect() - returns an array of the values that exist in two or more arrays.
- array_diff() - returns an array of the values that exist in one array but not in another.

Some Functions:

- `array array_pad ( array $input, int $pad_size mixed $pad_value )`
    - input - Initial array of values to pad.
    - pad_size - New size of the array.
    - pad_value-Value to pad if input is less than pad_size.

      e.g

```
$scores = array(3, 10);
padded = array_pad($scores, 5, 0);
 // $padded is now array(3, 10, 0, 0, 0)
 padded = array_pad($scores, -5, 0);
// $padded is now array(0, 0, 0,3,10)
```

- `array_count_values($array)` : return the count of each value in the arra

```
<?php
 $a=array("A","Cat","Dog","A","Dog");
 print_r(array_count_values($a));
?>
Output :
Array ( [A] => 2 [Cat] => 1 [Dog] => 2 )
```

- `array_search($value, $array, strict_parameter)`: used to search and locate a specific value in the given array. If it successfully finds the specific value, it returns its corresponding key value
    - value (required):This parameter represents the value that the user wishes to search in the given array.
    - $array (required):This parameter represents the original array, in which the user wants to search the element.
    - strict_parameter (optional): It is an optional parameter that can be set either to TRUE or FALSE. It represents the strictness of the array search. By default, this parameter is set to Boolean FALSE.
        - If strict_parameter is set to TRUE, then the function looks for similar values in the array, i.e., a string 200 will not be considered same as integer 200. Hence, both the values are different.
        - If strict_parameter is set to Boolean FALSE, strictness is not retained, i.e., a string 200 will be considered same as integer 200.

```
E.g
<?php
$array_name = array("red ", "blue", "green", "white",
"Rohit");
$search_value = "white";
$x= array_search($search_value, $array_name)
 print_r("$search_value is at position $x);
?>
```

- `in_array()` : check whether value is in array or not
    - syntax : in_array($value,$array)
    - Value is found in an array then returns true otherwise false.

Create array using range() :

range(): function creates an array of consecutive integer or character values between the two values

```
array range ($start , $limit [, number $step = 1 ] )
```

Parameters

- Start - First value of the sequence.
- Limit - The sequence is ended upon reaching the limit value.
- step value is given, it will be used as the increment between elements in the sequence. step should be given as a positive number. If not specified, step will default to 1.

E.g

```
<?php
        $numbers= range(1,5);
        print_r($numbers);
?>
```

E.g

```
<?php
        $chars= range(1,15,3);
        foreach($chars as $var)
           print( " {$var} " );
?>
```

Exercise

1. Write a php script to create an associative array with 5 different elements and display its key and value.
2. Write a php script which creates an array of even numbers starting from 50 to 200. Calculate the average, sum, min and max from the numbers stored in the array.
3. Write a php script which having multidimensional array for storing information like

Item	Price	Quantity
Item1	100	12
Item2	250	10

   ➔ Add 5 items information using associative array, Display the information in tabular format
4. Write a php script which demonstrates the function of array padding.
5. Write a php script which checks if the array is present or not , if value is present then it searches the value from the array and prints the index of search value .
6. Write a script which creates an array called marks , containing student name as key and mark as value. (e.g “Pratik”=> 34 ,”reema”= 36) .
   ➔ Display the array according to marks in ascending order.
   ➔ Display the array according to the name in descending order
7. Write a php script which creates an array[3][2] matrix and displays the matrix.