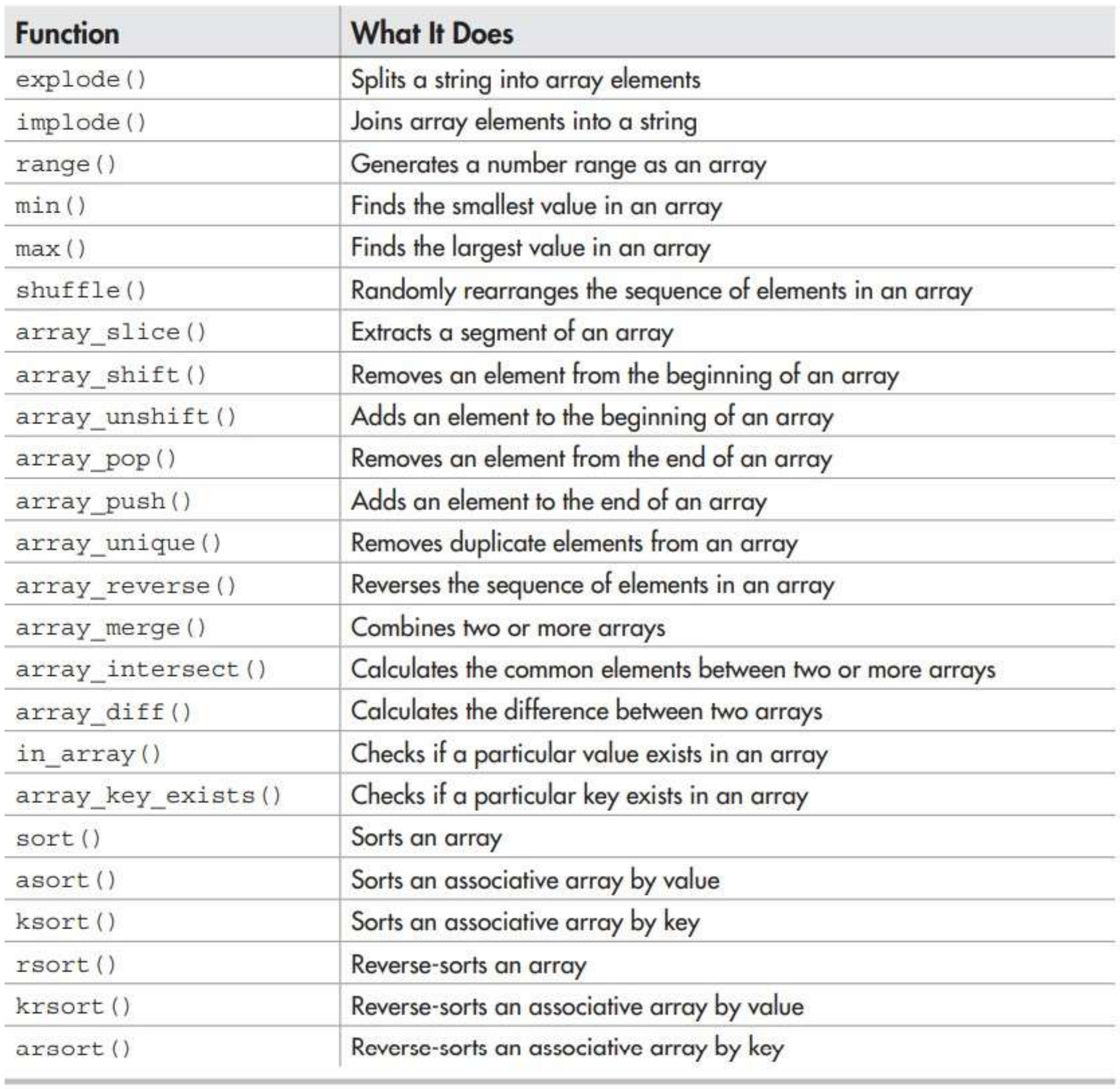
Unit-III

Lecture-III

Working with Array Functions

PHP has numerous built-in array manipulation functions, supporting operations ranging from array search and comparison to sorting and conversion operations. Some of the functions are as below:



Function	What It Does
`explode()`	Splits a string into array elements
`implode()`	Joins array elements into a string
`range()`	Generates a number range as an array
`min()`	Finds the smallest value in an array
`max()`	Finds the largest value in an array
`shuffle()`	Randomly rearranges the sequence of elements in an array
`array_slice()`	Extracts a segment of an array
`array_shift()`	Removes an element from the beginning of an array
`array_unshift()`	Adds an element to the beginning of an array
`array_pop()`	Removes an element from the end of an array
`array_push()`	Adds an element to the end of an array
`array_unique()`	Removes duplicate elements from an array
`array_reverse()`	Reverses the sequence of elements in an array
`array_merge()`	Combines two or more arrays
`array_intersect()`	Calculates the common elements between two or more arrays
`array_diff()`	Calculates the difference between two arrays
`in_array()`	Checks if a particular value exists in an array
`array_key_exists()`	Checks if a particular key exists in an array
`sort()`	Sorts an array
`asort()`	Sorts an associative array by value
`ksort()`	Sorts an associative array by key
`rsort()`	Reverse-sorts an array
`krsort()`	Reverse-sorts an associative array by value
`arsort()`	Reverse-sorts an associative array by key

Converting between strings and arrays:

PHP lets you convert a string into an array, by splitting the string on a user-defined separator and assigning the resulting segments to an array.

The PHP function to accomplish this task is:

- **explode():**
    - accepts two arguments- the separator and the source string and returns an array.

```
<?php

//define string

$str = 'tinker, tailor, soldier, spy';

//convert string to array

//output: ('tinker', 'tailor', 'soldier', 'spy')

$arr = explode (',' , $str);

print_r($arr);

?>
```

print_r() → this function prints the information about some variables in a more human-readable form.

e.g.

```
<html>

<body>

<?php

$a = array("red", "green", "blue");

print_r($a);

echo "<br>";


$b = array("Anil"=>"35", "Vinay"=>"37", "Abhi"=>"43");
```

```
print_r($b);
?>
</body>
</html>
```

Output:

```
Array ( [0] => red [1] => green [2] => blue )
Array ( [Anil] => 35 [Vinay] => 37 [Abhi] => 43 )
```

implode(): It is also possible to reverse the process i.e. joining the elements of an array into a single string using user-supplied “glue” using implode ().

e.g.

```
<php
//define array
$arr = array( ‘one’, ‘two’, ‘three’, ‘four’);
// convert array to string
// output: ‘one and two and three and four’
$str = implode(‘ and ’, $arr);
print_r($str);
?>
```

Working with number ranges:

range() : this function offers a convenient alternative to manually entering each value. This function accepts two end points and returns an array containing all the numbers between those end points. e.g.

```
<?php
//define array
$arr = range(1,1000);
```

```
print_r($arr);

?>
```

the above example will generate an array containing all the values between 1 and 1000.

PHP's **min() and max()** functions can be used to accept an array of numbers and return the smallest and largest values in the array respectively.

Extracting Array Segments:

array_slice() : PHP allows one to slice an array into smaller parts with the array_slice() function, which accepts three arguments:

- the original array,
- the index position(offset) at which to start slicing and
- the number of elements to return from the starting offset.

e.g.

```
<?php

//define array

$rainbow = array ('violet', 'indigo', 'blue', 'green', 'yellow', 'orange', 'red');

//extract 3 central values

// output: ('blue', 'green', 'yellow')

$arr = array_slice($rainbow, 2, 3);

print_r($arr);

?>
```

To extract a segment from the end of an array, pass array_slice() a negative offset.

Adding and Removing Array Elements:

PHP comes with four functions to allow you to add or remove elements from the beginning or end of an array:

- **array_unshift():** adds an element to the beginning of an array.
- **array_shift():** removes the first element of an array.
- **array_push():** adds an element to the end of an array.

- **array_pop():** removes the last element of an array.

Example:

```
<?php
// define array
$movies = array('The Lion King', 'Cars', 'A Bug\'s Life');
// remove element from beginning of array
array_shift($movies);
// remove element from end of array
array_pop($movies);
// add element to end of array
array_push($movies, 'Ratatouille');
// add element to beginning of array
array_unshift($movies, 'The Incredibles');
// print array
// output: ('The Incredibles', 'Cars', 'Ratatouille')
print_r($movies);
?>
```

The array_unshift(), array_shift(), array_push() and array_pop() functions should be used only with numerically indexed arrays and not with associative arrays. Each of these functions automatically re-indexes the array to account for the value(s) added or removed during its operation.

Removing Duplicate Array Elements:

- **array_unique():** PHP lets you strip an array of duplicate values with its array_unique() function, which accepts an array and returns a new array containing only unique values.
  e.g.:
  ```
  <?php
  // define array
  ```

```
$duplicates = array('a', 'b', 'a', 'c', 'e', 'd', 'e');
// remove duplicates
// output: ('a', 'b', 'c', 'e', 'd')
$uniques = array_unique($duplicates);
print_r($uniques);
?>
```

Randomizing and Reversing Arrays:

- **shuffle():** PHP's shuffle function re-arranges the elements of an array in random order.
- **array_reverse():** reverses the order of an array's elements.

e.g.:

```
<?php

// define array

$rainbow = array('violet', 'indigo', 'blue', 'green', 'yellow',

 'orange', 'red');

// randomize array

shuffle($rainbow);

print_r($rainbow);

// reverse array

// output: ('red', 'orange', 'yellow', 'green', 'blue',

// 'indigo', 'violet')

$arr = array_reverse($rainbow);

print_r($arr);

?>
```

Searching Arrays:

- **in_array():** function looks through an array for a specified value and returns true if found.
- **array_key_exists():** function looks for a match to the specified search term among an array's keys.

Sorting Arrays:

- **sort():** this function lets you sort numerically indexed arrays alphabetically or numerically from lowest to highest value.

e.g.

```
<?php

//define array

$data = array(15,81,14,74,2);

sort($data);

print_r($data);

?>
```

- **assort():** If you have to sort an associative array, use assort(), which maintains the correlation between keys and values while sorting.
- **ksort():** also related to associative arrays, which uses keys instead of values when performing the sorting.

e.g.

```
<?php
// define array
$profile = array(
 "fname" => "Ajay",
 "lname" => "Sharma",
 "sex" => "male",
 "sector" => "Asset Management"
);
// sort by key
// output: ('fname' => 'Ajay',
// 'lname' => 'Sharma',
// 'sector' => 'Asset Management',
// 'sex' => 'male')
ksort($profile);
print_r($profile);
?>
```