Internet Applications

JAMOUM UNIVERSITY COLLEGE – COMPUTER SCIENCE DEPARTMENT

UMM AL-QURA UNIVERSITY

I. AMAL ALSHOMRANI

PHP Arrays

CHAPTER 9

Arrays

Background

An array is a data structure that

- Collects a number of related elements together in a single variable.
- Allows the set to be Iterated
- Allows access of any element

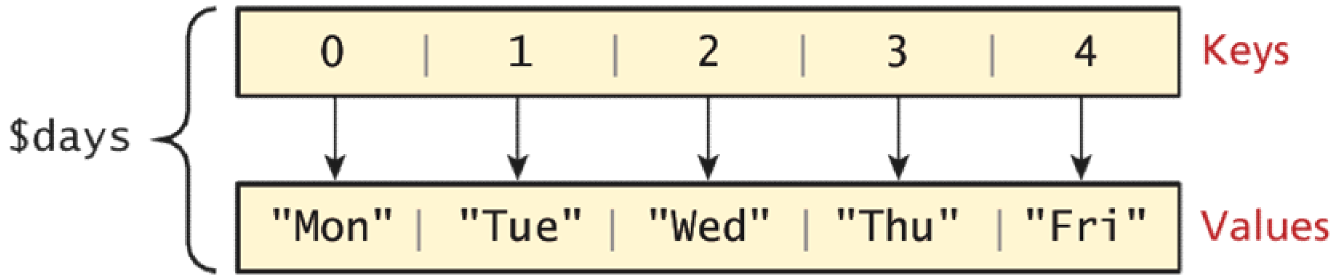
Since PHP implements an array as a dynamic structure:

- Add to the array
- Remove from the array

Arrays

Key Value

In PHP an array is actually an **ordered map**, which associates each value in the array with a key.



Arrays

Keys

Array keys are the means by which you refer to single elements in the array.

In most programming languages array keys are limited to integers, start at 0, and go up by 1.

In PHP, array keys *must* be either integers or strings and need not be sequential.

This means you cannot use an array or object as a key (doing so will generate an error)

- Don’t mix key types i.e. “1” vs 1
- If you don’t explicitly define them they are 0,1,…

Arrays

Values

Array values, unlike keys, are not restricted to integers and strings.

They can be any object, type, or primitive supported in PHP.

You can even have objects of your own types, so long as the keys in the array are integers and strings.

Arrays

Defining an array

The following declares an empty array named days:

$days = array();

You can also initialize it with a comma-delimited list of values inside the ( ) braces using either of two following syntaxes:

```
$days = array("Mon","Tue","Wed","Thu","Fri");

$days = ["Mon","Tue","Wed","Thu","Fri"]; // alternate
```

In these examples, because no keys are explicitly defined for the array, the default key values are 0, 1, 2, . . . , n.

Notice that you do not have to provide a size for the array: arrays are dynamically sized as elements are added to them.

Arrays

Defining an array

You can also declare each subsequent element in the array individually:

```
$days = array();

$days[0] = "Mon"; //set 0th key's value to "Mon"

$days[1] = "Tue";

$daysB = array(); // also alternate approach

$daysB[] = "Mon"; //set the next sequential value to
"Mon"

$daysB[] = "Tue";
```

Arrays

Access values

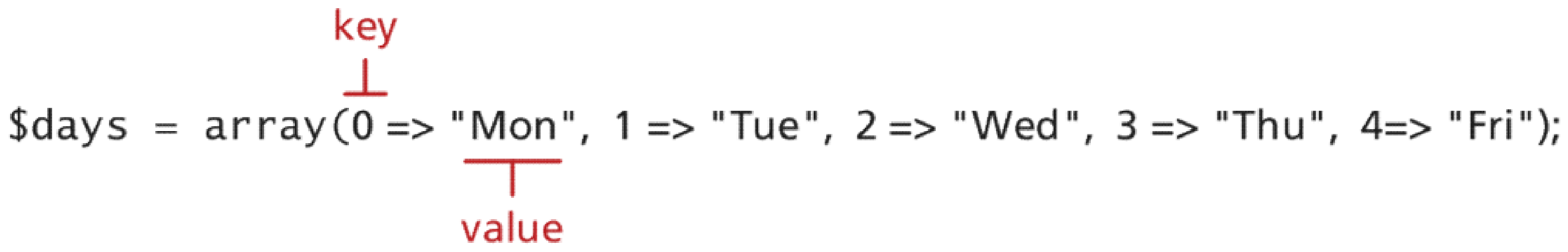
To access values in an array you refer to their key using the square bracket notation.

```
echo "Value at index 1 is ". $days[1];
```

Keys and Values

In PHP, you are also able to explicitly define the keys in addition to the values.

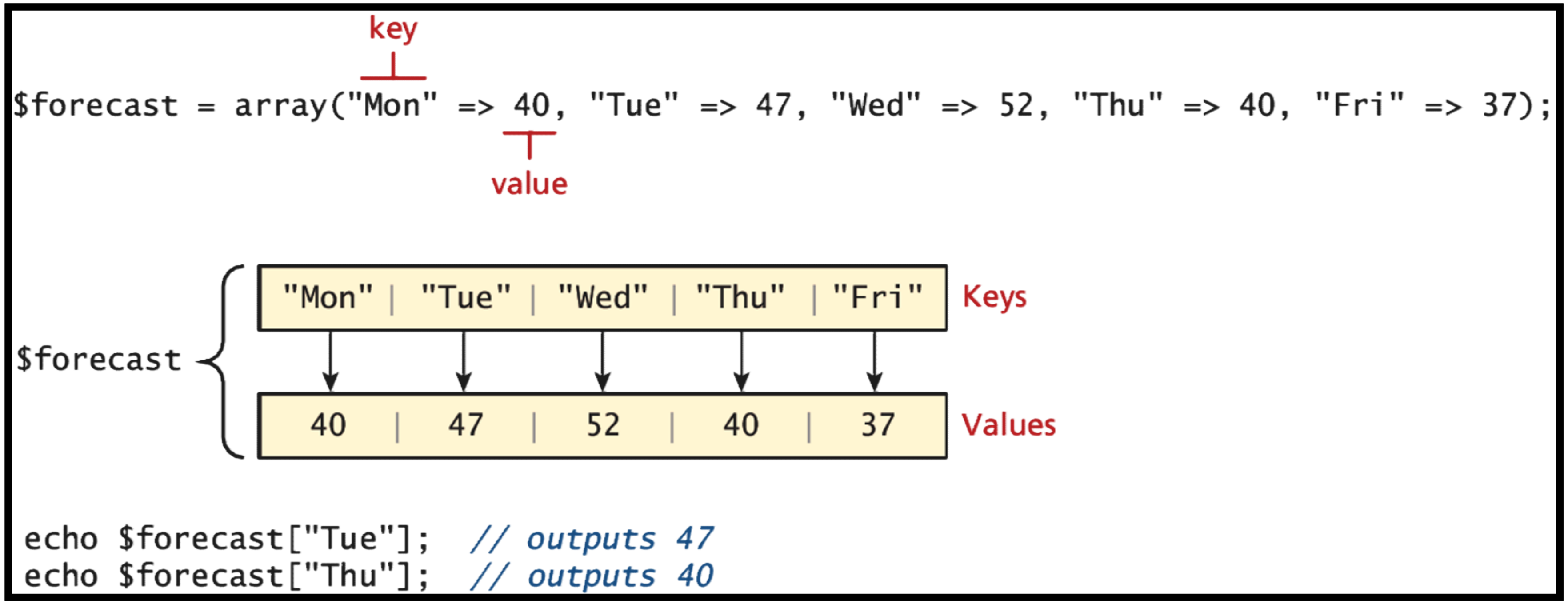
This allows you to use keys other than the classic 0, 1, 2, . . . , n to define the indexes of an array.



```
$days = array(0 => "Mon", 1 => "Tue", 2 => "Wed", 3 => "Thu", 4=> "Fri");
```

Super Explicit

Array declaration with string keys, integer values



```
$forecast = array("Mon" => 40, "Tue" => 47, "Wed" => 52, "Thu" => 40, "Fri" => 37);
```

```
echo $forecast["Tue"];   // outputs 47
echo $forecast["Thu"];   // outputs 40
```

These types of arrays in PHP are generally referred to as associative arrays. You can see in this Figure an example of an associative array and its visual representation.

Multidimensional Arrays

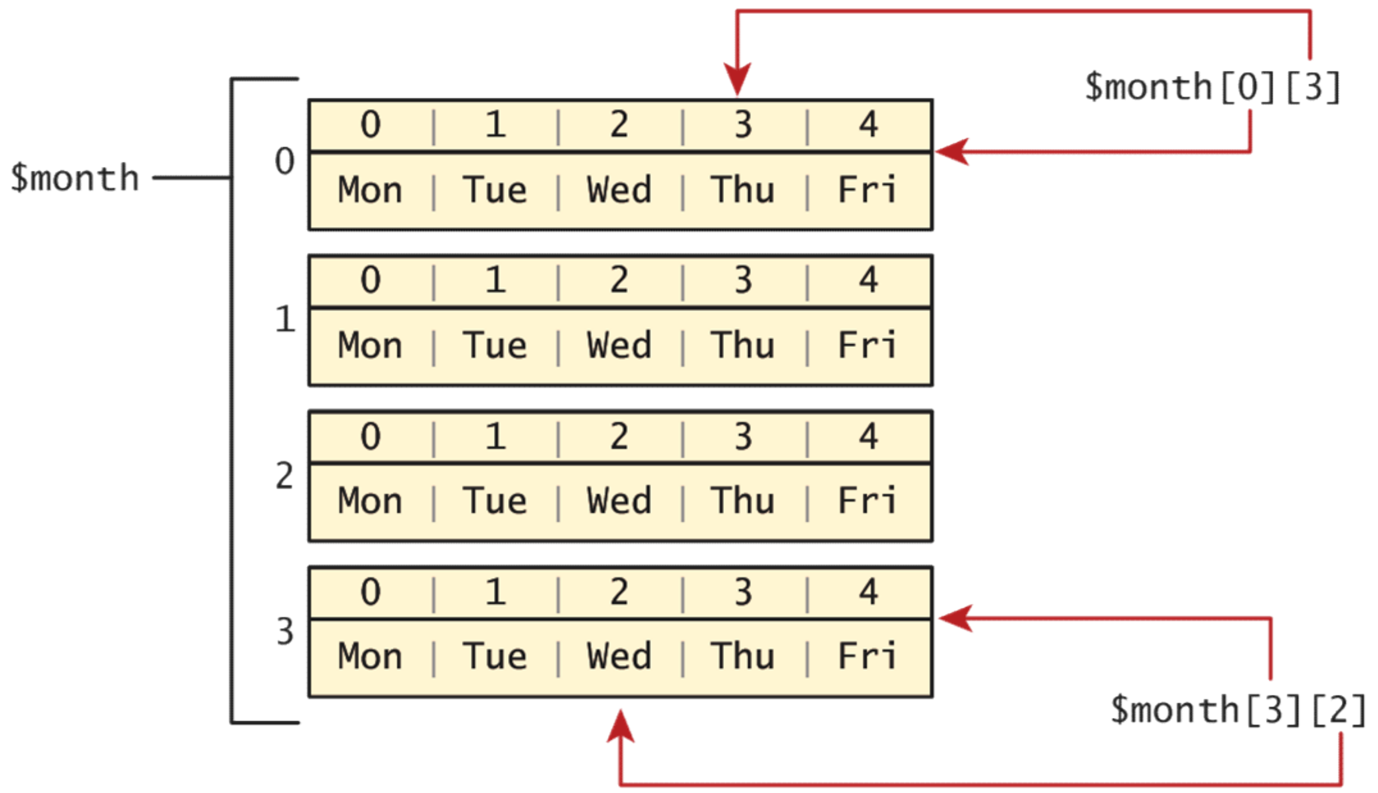
Creation

PHP also supports multidimensional arrays. Recall that the values for an array can be any PHP object, which includes other arrays

```
$month = array(

        array("Mon","Tue","Wed","Thu","Fri"),

        array("Mon","Tue","Wed","Thu","Fri"),

        array("Mon","Tue","Wed","Thu","Fri"),

        array("Mon","Tue","Wed","Thu","Fri")

);

echo $month[0][3]; // outputs Thu
```

Multidimensional Arrays

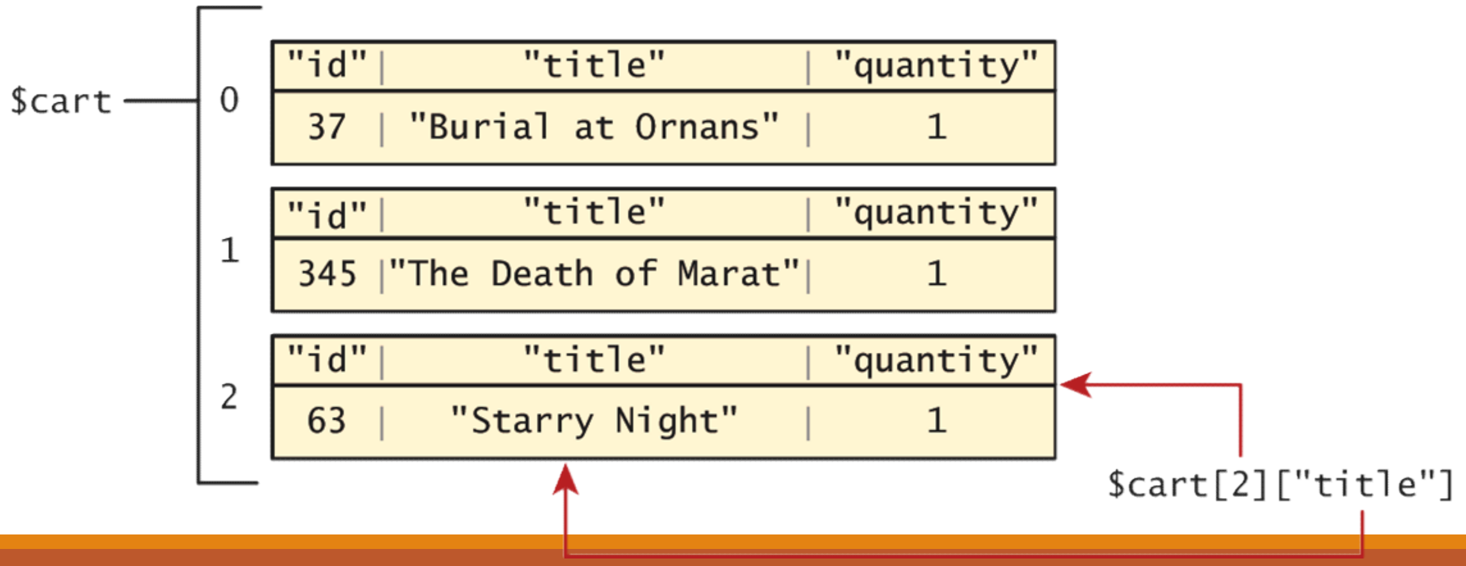
Access



Multidimensional Arrays

Another example

```
$cart = array();

$cart[] = array("id" => 37, "title" => "Burial at Ornans", "quantity" => 1);

$cart[] = array("id" => 345, "title" => "The Death of Marat", "quantity" => 1);

$cart[] = array("id" => 63, "title" => "Starry Night", "quantity" => 1);
```



Iterating through an array

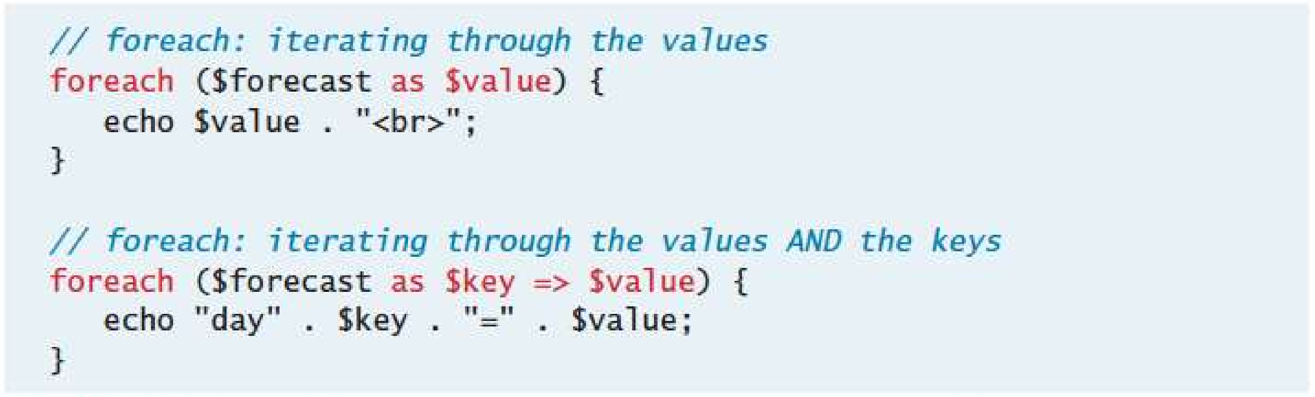
```php
// while loop
$i=0;
while ($i < count($days)) {
   echo $days[$i] . "<br>";
   $i++;
}


// do While loop
$i=0;
do {
   echo $days[$i] . "<br>";
   $i++;
} while ($i < count($days));


// for loop
for ($i=0; $i<count($days); $i++) {
   echo $days[$i] . "<br>";
}
```

LISTING 9.2 Iterating through an array using while, do while, and for loops

Iterating through an array

Foreach loop is pretty nice

The challenge of using the classic loop structures is that when you have nonsequential integer keys (i.e., an associative array), you can’t write a simple loop that uses the $i++ construct. To address the dynamic nature of such arrays, you have to use iterators to move through such an array.



```
// foreach: iterating through the values
foreach ($forecast as $value) {
   echo $value . "<br>";
}

// foreach: iterating through the values AND the keys
foreach ($forecast as $key => $value) {
   echo "day" . $key . "=" . $value;
}
```

LISTING 9.3 Iterating through an associative array using a foreach loop

PRO TIP

In practice, arrays are printed in web apps using a loop as shown in Listing 9.2 and Listing 9.3. However, for debugging purposes, you can quickly output the content of an array using the `print_r()` function, which prints out the array and shows you the keys and values stored within. For example,

```
print_r($days);
```

Will output the following:

```
Array ( [0] => Mon [1] => Tue [2] => Wed [3] => Thu [4] => Fri )
```

Adding to an array

An element can be added to an array simply by using a key/index that hasn’t been used

```
$days[5] = "Sat";
```

Since there is no current value for key 5, the array grows by one, with the new key/value pair added. If the key had a value already, the same style of assignment replaces the value at that key.

A new element can be added to the end of any array

```
$days[ ] = "Sun"; // The advantage to this approach is that we don’t have to worry about skipping an index key.
```

Adding to an array

And quickly printing

PHP is more than happy to let you “skip” an index

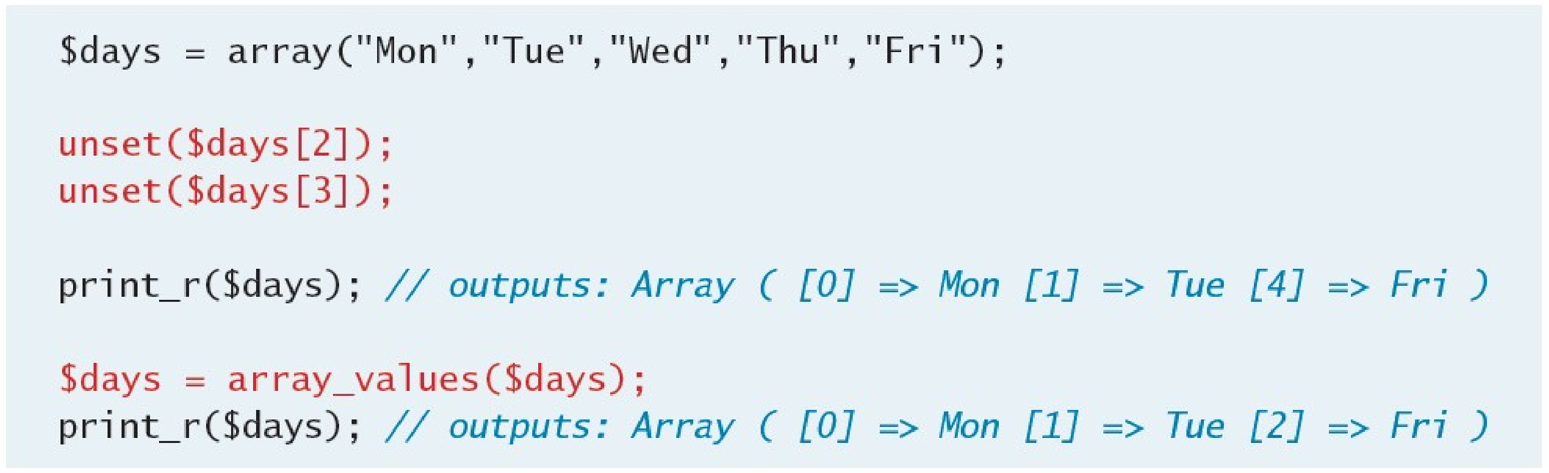
```
$days = array("Mon","Tue","Wed","Thu","Fri");

$days[7] = "Sat";

print_r($days);
```

```
Array ([0] => Mon [1] => Tue [2] => Wed [3] => Thu [4] => Fri [7] => Sat)’
```

If we try referencing $days[6], it will return a **NULL** value which is a special PHP value that represents a variable with no value.

Deleting from an array

You can explicitly delete array elements using the unset() function, this will create "gaps"



```
$days = array("Mon","Tue","Wed","Thu","Fri");

unset($days[2]);
unset($days[3]);

print_r($days); // outputs: Array ( [0] => Mon [1] => Tue [4] => Fri )

$days = array_values($days);
print_r($days); // outputs: Array ( [0] => Mon [1] => Tue [2] => Fri )
```

LISTING 9.4 Deleting elements

Deleting from an array

you can remove “gaps” in arrays (which really are just gaps in the index keys) using the array_values() function, which reindexes the array numerically.

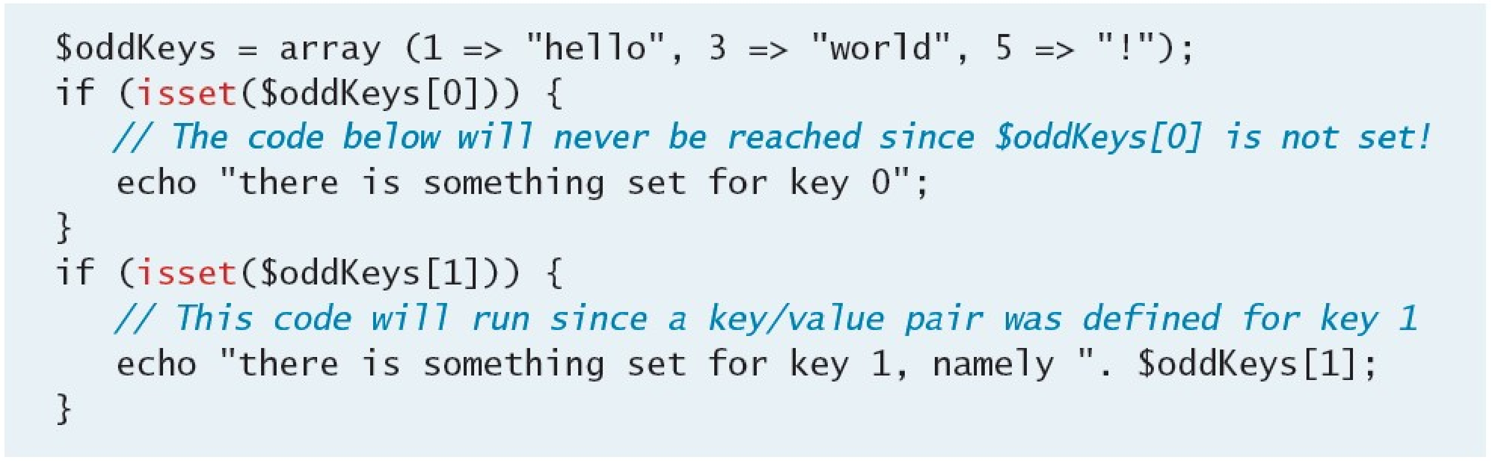
```
$days = array("Mon","Tue","Wed","Thu","Fri");

unset($days[2]);
unset($days[3]);

print_r($days); // outputs: Array ( [0] => Mon [1] => Tue [4] => Fri )

$days = array_values($days);
print_r($days); // outputs: Array ( [0] => Mon [1] => Tue [2] => Fri )
```

LISTING 9.4 Deleting elements

Checking for a value

Since array keys need not be sequential, and need not be integers, you may run into a scenario where you want to check if a value has been set for a particular key.

To check if a value exists for a key, you can therefore use the isset() function, which returns true if a value has been set, and false otherwise



```
$oddKeys = array (1 => "hello", 3 => "world", 5 => "!");
if (isset($oddKeys[0])) {
   // The code below will never be reached since $oddKeys[0] is not set!
   echo "there is something set for key 0";
}
if (isset($oddKeys[1])) {
   // This code will run since a key/value pair was defined for key 1
   echo "there is something set for key 1, namely ". $oddKeys[1];
}
```

LISTING 9.5 Illustrating nonsequential keys and usage of isset( )

Array Sorting

Sort it out

There are many built-in sort functions, which sort by key or by value. To sort the $days array by its values you would simply use:

```
sort($days);
```

As the values are all strings, the resulting array would be:

```
Array ([0] => Fri [1] => Mon [2] => Sat [3] => Sun [4] => Thu [5] => Tue [6] => Wed)
```

However, such a sort loses the association between the values and the keys! A better sort, one that would have kept keys and values associated together, is:

```
asort($days);
```

```
Array ([4] => Fri [0] => Mon [5] => Sat [6] => Sun [3] => Thu [1] => Tue [2] => Wed)
```

More array operations

Too many to go over in depth here…

- array_keys($someArray)
- array_values($someArray)
- array_rand($someArray, $num=1)
- array_reverse($someArray)
- array_walk($someArray, $callback, optionalParam)
- in_array($needle, $haystack)
- shuffle($someArray)
- …

Loop Through an Associative Array

Exercise

- Write .php script that defined an array of 3 girls with their ages (ages are the keys, names are the values), Print girls info
- Sort array , According to Value
- Print girls info
- Delete the second girl info then print the array details
- Reindexes the array numerically