codecademy

PHP Arrays

Accessing and Adding Elements

In PHP, we can access the value that a given key points to using square brackets (`[]`) and the key. To add new elements to an *associative array*, we can use the assignment operator (`=`). We can also change existing elements using the same syntax that adds new array elements.

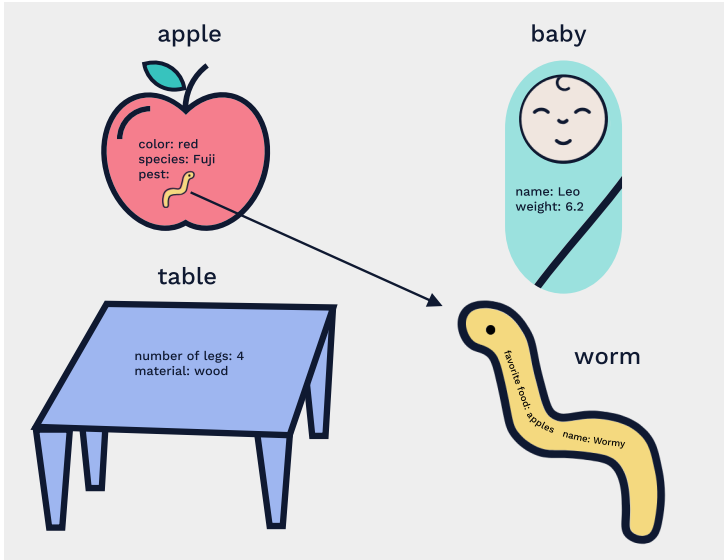
```
$nums = ["2" => 2,"4" => 3, "6" => 6];

echo $nums["2"]; // Accesses value and
Prints: 2

$nums["8"] = 8;  // Adds new value 8
echo $nums["8"]; // Prints: 8

$nums["4"] = 4;  // Changes value of key
"4"
echo $nums["4"]; // Prints: 4
```

PHP Associative Array

In PHP, *associative arrays* are map-like structures, where keys are associated with values. When we need to access a specific value, we can use the *associated* key to find it. In a PHP ordered array, the index locations are the keys. However, the PHP array type also allows us to assign meaningful keys to values. These data structures are called *associative arrays*.

For example, in the following associative array `table`, the key `"num_legs"` points to the value `4`, and the key `"material"` points to the value `"wood"`:

```
$table = ["num_legs" => 4, "materi
```



PHP Ordered Arrays

In PHP, an ordered array is a data structure representing a list of ordered, stored data. The location of an element in the array is known as its index. The elements in an ordered array are arranged in ascending numerical order starting with zero.

```
$array_of_integers = array(3, 2, 1);
$array_of_strings = ["one", "2",
"twenty"];
$array_of_mixed_content = ["two", 0, 1.0];
echo $array_of_integers[0]; // 3
echo $array_of_strings[2]; // "twenty"
echo $array_of_mixed_content[1] // 0
```

PHP array function

In PHP, an ordered array can be constructed with the built-in PHP function: `array()`. The `array()` function returns an array. Each of the arguments with which the function was invoked becomes an element in the array (in the order they were passed in).
Pass in arguments by including them, comma-separated, between the parentheses.

```
$array_of_strings = array("Ab", "C#", "D",
"Fm");
$mixed_content = array("Shelby", 5, "GTO",
1964.5);
```

PHP Nested Arrays

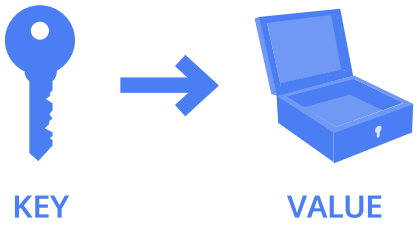
In PHP, nested arrays are arrays that contain other arrays as elements. Chained operations can be used to access and/or change elements within a nested array.

```
$array_of_arrays = ["a", "Not that lucky",
2, ["smoke detectors", "d", ["boring
street in", "U.S."], "nothing happens"],
"."];
echo $array_of_arrays[3][2][1] //Returns
"U.S."
$array_of_arrays[3][2][1] = "America"; //
"U.S." changed to "America"
echo $array_of_arrays[3][2][1] //Returns
"America"
```

Create Associative Array

In PHP, *associative arrays* are collections of *key => value* pairs, where the key must be either a string or an integer and the value can be of any type.
The `=>` operator is used to associate a key with its value.
For example:

```
$grades = ["John" => 91, "Sally" =
```



The associative array `grades` is defined using short array syntax. The key `"John"` points to the value `91`, and the key `"Sally"` points to the value `94`.
We can also define this associative array using `array()`.
For example:

```
$grades = array(
    "John" => 91,
    "Sally" => 94
);
```

PHP Accessing Array Values

In PHP, the individual elements in an array can be accessed using the array variable's name, and the location index surrounded by square brackets `[]`.
Remember that in PHP the location index starts at zero.

```
$suit_up = ["top hat", "&", "tails"];
echo $suit_up[0]; // top hat

$index_value = 2;
$odd_choices = [99, "red", "balloons"];
echo $odd_choices[$index_value]; //
balloons
```

Reassign Array Element

In PHP, individual elements in an array can easily be reassigned. To do this, we access the location of an element in the array using square brackets (`[]`), and then assign a new value using the assignment operator (`=`).

```
$num_array = [1, 2, 3];

$num_array[1] = 4; // Reassign value of second element

echo $num_array[1]; // Prints: 4
```

PHP Apending Arrays

In PHP, elements can be added to the end of an array by attaching square brackets ([]) at the end of the array's name, followed by typing the assignment operator (=), and then finally by typing the element to be added to the array.

```
$string_array = ["first element", "second element"];
$string_array[] = "third element";
// $string_array is now:  ["first element", "second element", "third element"]
```

PHP array_push function

The PHP built-in array_push() function takes an array as its first argument. The arguments that follow are elements to be added to the array.
The function adds each of the elements to the end of the array and returns the new number of elements in the array.
To use the function place an array or a variable with an array as its value in between the parentheses.

```
$some_numbers = ["1", 2, 3.0];
echo array_push($some_numbers, "four");
//Returns: 4
// $some_numbers is now: ["1", 2, 3.0, "four"]
```

PHP Count Function

The built-in PHP `count()` function takes an array as its argument and returns the number of elements in that array.
To use the function place an array or a variable with an array as its value in between the parentheses.

```
$devices = array("watch", "phone", "tablet", "laptop");
$number_of_devices = count($devices);
echo $number_of_devices; //Returns: 4
```

PHP Square Bracket Arrays

In PHP, arrays can also be constructed using short array syntax. To use this syntax, simply wrap comma-separated elements in a set of square brackets.

```
$one_to_three = [1, 2, 3];
$three_to_five = ["3", "four", 5.0];
```

PHP array_pop function

The PHP built-in `array_pop()` function takes an array as its argument. It permanently removes the last element of an array and returns the removed element.
To use the function place an array or a variable with an array as its value in between the parentheses.

```
$some_numbers = ["1", 2, "three"];
echo array_pop($some_numbers); // "three"
// $some_numbers is now: ["1", 2]
```

PHP unshift function

The PHP built-in `array_unshift()` function takes an array as its first argument. The arguments that follow are elements to be added to the array.
The function adds each of the elements to the beginning of the array and returns the new number of elements in the array.
To use the function, place an array or a variable with an array as its value in between the parentheses.

```
$joke = [7, "ate", "nine!"];
$scared = array_unshift($joke, "Why is 6",
"afraid of 7?", "Because");
echo $scared; // 6
// $joke is now: ["Why is 6", "afraid of
7?", "Because", 7, "ate", "nine!"]
```

PHP implode function

The built-in `implode()` function makes a string from an array. The function takes two arguments: a string to place between each element of an array and the array to be joined together.
The function returns a single string with the string argument, known as the `$glue`, inserted between each element in the array, known as the `$pieces`.

```
$pieces = [1, "2", "three", 4.0];
$glue = " and a ";
echo implode($glue, $pieces); // 1 and a 2
and a three and a 4

//Note - the arrays do not have to always
be labelled "$pieces" & "$glue"
```

PHP array_shift function

The PHP built-in `array_shift()` function takes an array as its argument, permanently removes the first element from the array, and returns the removed element. All the elements in the array will be shifted to an index one smaller than their previous index.
To use the function, place an array or a variable with an array as its value in between the parentheses.

```
$some_numbers = ["1", 2, "three"];
echo array_shift($some_numbers); //
Returns "1"
// $some_numbers is now: [2, "three"]
```

Assign by Value or by Reference

PHP arrays can be assigned or passed by value or by reference. PHP arrays that are passed by reference use the reference sigil (`&`). Here are some of the important differences between the two:

- Passed by value: this creates two variables which hold copies of the same value but remain independent entities.
- Passed by reference: this creates two variable names (aliases) which point to the same space in memory. They cannot be modified separately!

```
$favorites = ["food"=>"pizza",
"person"=>"myself", "dog"=>"Tadpole"];
$copy = $favorites;
$alias =& $favorites;
$favorites["food"] = "NEW!";

echo $favorites["food"]; // Prints: NEW!
echo $copy["food"]; // Prints: pizza
echo $alias["food"]; // Prints: NEW!
```

Joining Arrays in PHP

In PHP, the union (`+`) operator takes two array operands and returns a new array with any unique keys from the second array appended to the first array.

```
$my_array = ["panda" => "very cute",
"lizard" => "cute", "cockroach" => "not
very cute"];

$more_rankings = ["capybara" => "cutest",
"lizard" => "not cute", "dog" => "max
cuteness"];

$animal_rankings = $my_array +
$more_rankings;

/*
$animal_rankings will have each of the
key=>value pairs from $my_array and the
key=>value pairs from $more_rankings.

$my_array will retain the value of "cute"
for $animal_rankings["lizard"] since
"lizard is not a unique key"
*/
```

Removing Elements in Associative Array

A `key=>value` pair in a PHP associative array can be removed entirely using the PHP `unset()` function.
If the key used does not exist in the array, then nothing happens.

```
$nums = ["one" => 1,"two"=> 2];

unset($nums["one"]); // Removes the "one"
=> 1 pair.

echo implode(", ", $nums); // Prints: 2
```

Numerical Keys in Associative Arrays

Associative arrays can use integers or strings as keys. In PHP, ordered arrays are just arrays in which integer keys have been assigned to the values automatically. For example, the first element is associated with key `0`, the second with `1`, etc. Even though ordered arrays are the same structure as associative arrays, it's recommended that you treat the two separately.

When adding an element to an array without specifying a key, PHP associates it with the "next" key. If no integer keys have been used, it will associate it with the key `0`, otherwise it will associate it one more than the largest integer used so far.

```
$num_array = [1000 => "one thousand", 100 => "one hundred", 600 => "six hundred"];
echo $num_array[1000]; // Prints: one thousand

$ordered = [99, 1, 7, 8];
$ordered["special"] = "Cool!";
echo $ordered[3]; // Prints: 8
echo $ordered["special"]; // Prints: Cool!

$num_array = [1000 => "one thousand", 100 => "one hundred", 600 => "six hundred"];
$num_array[] = "New Element in \$num_array";
echo $num_array[1001]; // Prints: New Element in $num_array

$animals_array = ["panda"=>"very cute", "lizard"=>"cute", "cockroach"=>"not very cute"];
array_push($animals_array, "New Element in \$animals_array");
echo $animals_array[0]; // Prints: New Element in $animals_array
```

PHP print_r function

The built-in `print_r()` function outputs arrays in a human readable format.
To use the function, place an array or a variable with an array as its value in between the parentheses.

```
$my_array = array(25, "or", 6, "to", 4);
print_r($my_array);
/*
Array
(
    [0] => 25
    [1] => or
    [2] => 6
    [3] => to
    [4] => 4
)
*/
```

codecademy

PHP Loops

PHP break keyword

In PHP, `break` can be used to terminate execution of a `for`, `foreach`, `while` or `do…while` loop.
One downside of heavy usage of break statements is that code can become less readable.

```
// We can use the break statement to end
the loop once the count reaches 4
$count = 1;
while ($count < 10)
{
  echo "The count is: " . $count . "\n";
  if ($count === 4) {
    break;
  }
  $count += 1;
}
```

PHP do while loops

In PHP, `do…while` loops are very similar to `while` loops. The main difference is that `do...while` loops always execute their code block at least once and continue execution as long as their conditional statement is true. Even if the conditional is false, the code block will execute one time.
The syntax for a `do…while` loop is:

```
do {
    #code block
} while (/*conditional*/);
```

```
// This loop counts from 0 to 100
$count = 0;
do {
  echo "The count is: " . $count . "\n";
  $count += 1;
} while ($count <= 100);
```

PHP for loop

In PHP, a `for` loop is commonly used to execute a code block a specific number of times. The syntax makes use of three expressions:

```
for (#expression 1; #expression 2;
{
  # code block
}
```

- The first is evaluated only one time before the first iteration
- The second is evaluated before each iteration. If it is TRUE, the code block is executed. Otherwise, the loop terminates.
- The third is evaluated after each iteration.

```
// This for loop counts from 1 to 50
for ($count = 1; $count < 51; $count++)
{
  echo "The count is: " . $count . "\n";
}
```

PHP while loops

In PHP, `while` loops repeat execution of their code block as long as their conditional statement is true. The syntax for a `while` loop is:

```
while (/*conditional*/) {
  #code block
}
```

```
// This while loop counts from 0 to 100
$count = 0;
while ($count <= 100)
{
  echo "The count is: " . $count . "\n";
  $count += 1;
}
```

PHP foreach loop

In PHP, the `foreach` loop is used for iterating over an array. The code block is executed for every element in the array and the value of that element is available for use in the code block.
The syntax is:

```
foreach ($array as $value) {
  #code block
}
```

On each iteration, a $value from $array is available for usage in the code block.
To access the keys as well as values in an array, the syntax can be modified to:

```
foreach ($array as $key => $value)
  #code block
}
```

```
// This foreach loop counts from 1 to 5
$nums = [1, 2, 3, 4, 5];
foreach ($nums as $num) {
  echo "The num is: " . $num . "\n";
}
```

PHP continue keyword

In PHP, `continue` can be used to terminate execution of a loop iteration during a `for`, `foreach`, `while` or `do…while` loop. The code execution continues with the next iteration of the loop.
The `continue` keyword is similar to `break` except it only ends the current iteration early, not the entire loop.

```
// This code counts from 1 to 10 but skips over 5
$count = 1;
while ($count < 11)
{
  if ($count === 5) {
    $count += 1;
    continue;
  }
  echo "The count is: " . $count . "\n";
  $count += 1;
}
```