Loops and Arrays

1PHPD

1PHPD – Loops and Arrays

Course Objectives

By the end of the course, students should:
- Know about arrays in PHP
- Be able to loop over data

Time:
- course: 2h
- exercises: 3h

Summary

1. Loops
2. Arrays
3. Performance

1. Loops

1. Loops

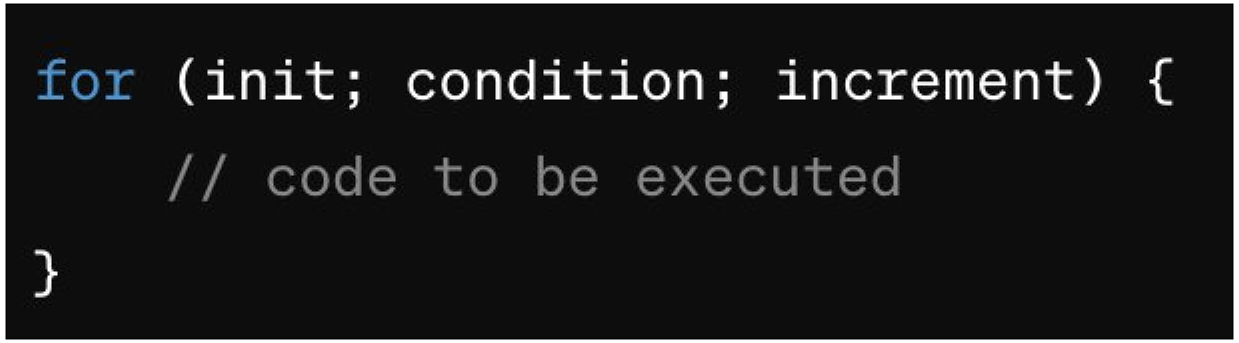
Loops in PHP are used to execute the same block of code a specified number of times or as long as a specified condition is true.

They are essential for tasks that require repetitive execution, such as generating HTML tables, processing arrays, or doing calculations.

PHP supports several types of loops, each suited for different scenarios

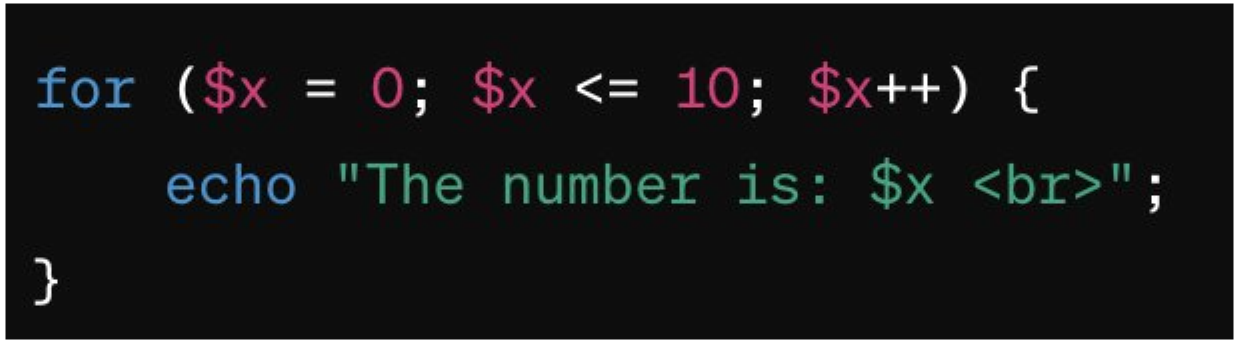
1. Loops

The for loop is used when you know in advance how many times you want to execute a statement or a block of statements.



```
for (init; condition; increment) {
    // code to be executed
}
```

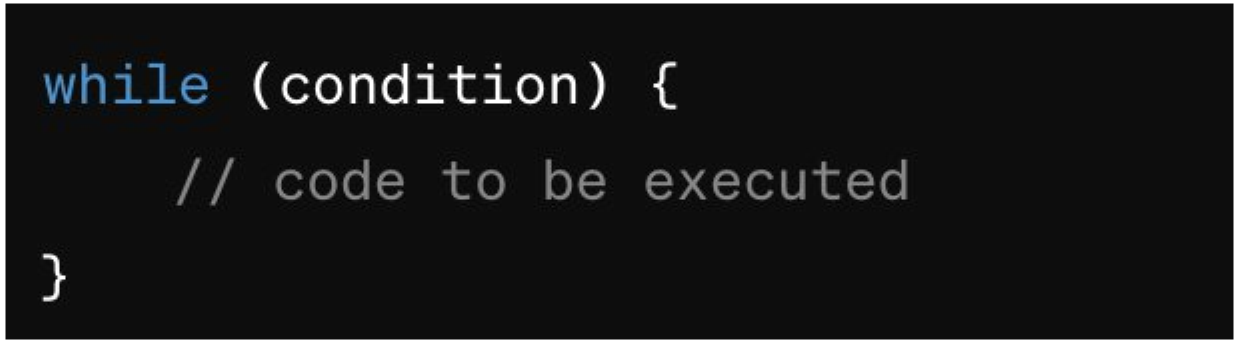
1. Loops



```php
for ($x = 0; $x <= 10; $x++) {
    echo "The number is: $x <br>";
}
```

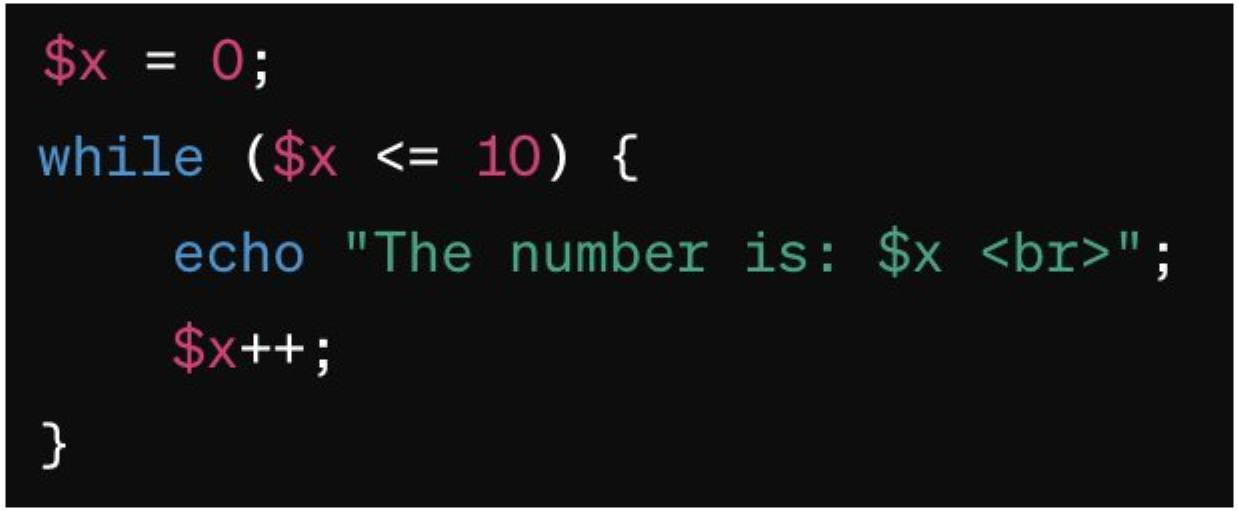
1. Loops

The while loop executes a block of code as long as the specified condition is true.



```
while (condition) {
    // code to be executed
}
```

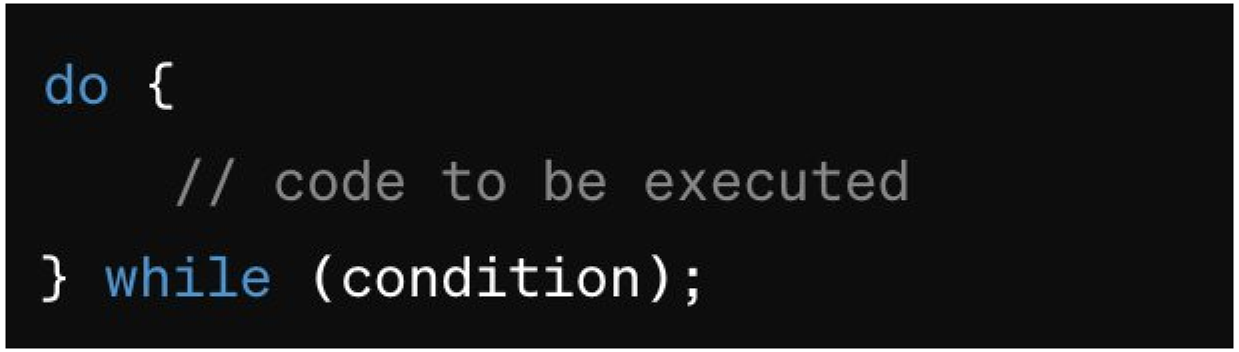
1. Loops



```php
$x = 0;
while ($x <= 10) {
    echo "The number is: $x <br>";
    $x++;
}
```

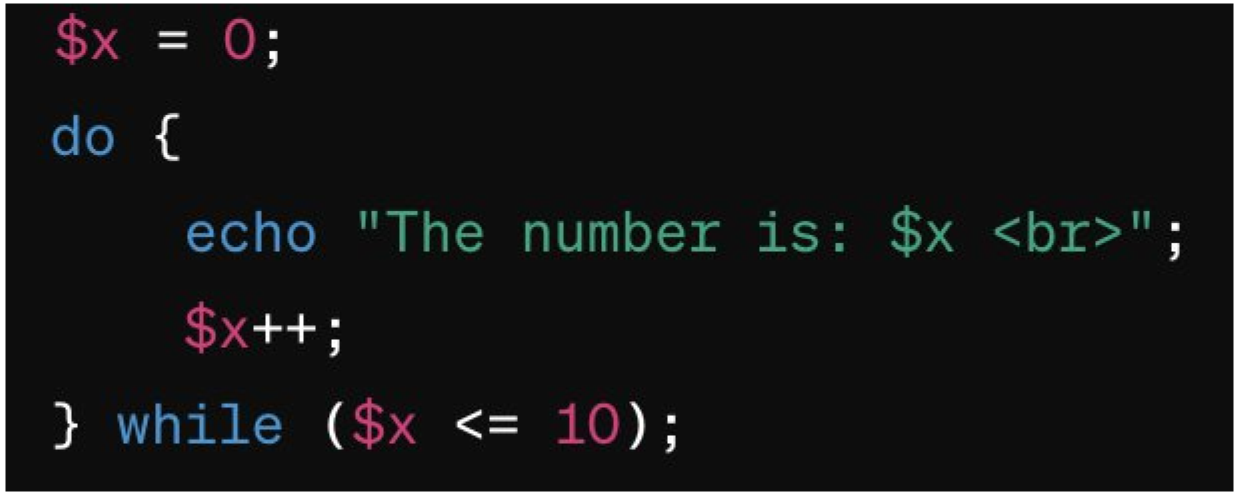
1. Loops

The do...while loop will execute the block of code once, before checking if the condition is true, then it will repeat the loop as long as the specified condition is true.



```
do {
    // code to be executed
} while (condition);
```

1. Loops



```php
$x = 0;
do {
    echo "The number is: $x <br>";
    $x++;
} while ($x <= 10);
```

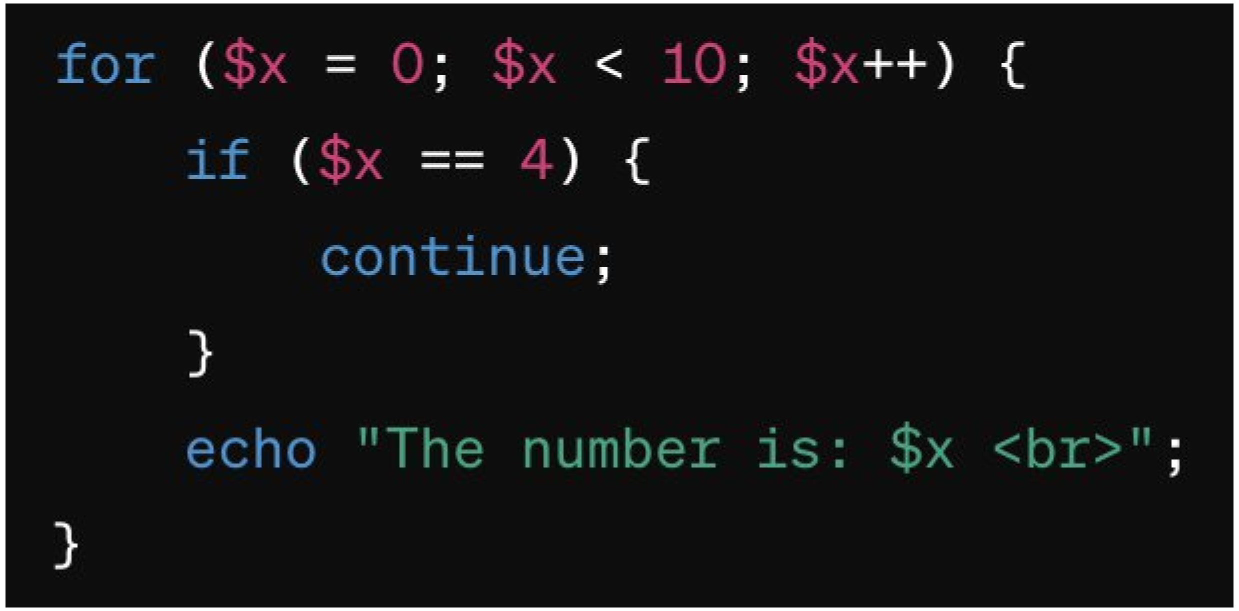
1. Loops

Part of the loop ecosystem, you may encounter two keywords

- continue: Used within any loop to skip the rest of the current loop iteration and continue execution at the condition evaluation and then the beginning of the next iteration.
- break: Exits the loop immediately, regardless of the loop's condition.

1. Loops

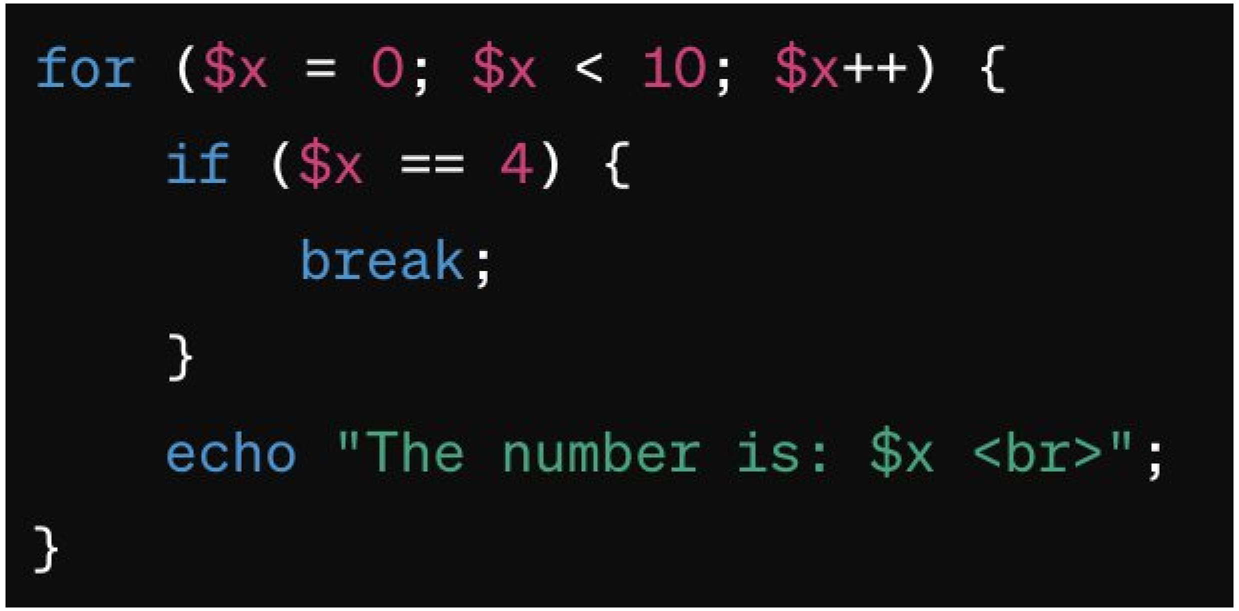
Continue:



```
for ($x = 0; $x < 10; $x++) {
    if ($x == 4) {
        continue;
    }
    echo "The number is: $x <br>";
}
```

1. Loops

Break:



```php
for ($x = 0; $x < 10; $x++) {
    if ($x == 4) {
        break;
    }
    echo "The number is: $x <br>";
}
```

1. Loops

Here is a quick tips on which solution to choose

- Use a for loop when the number of iterations is known.
- Use a while loop when the number of iterations is not known, and you want the loop to continue as long as a condition is true.
- Use a do...while loop when the block of code should execute at least once, but then continue only if a condition is true.

Exercises

2. Arrays

2. Arrays

Arrays in PHP are powerful and flexible data structures that allow you to store multiple values in a single variable.

They can hold values of any type, such as integers, strings, or even other arrays, and PHP supports both numeric (indexed) arrays and associative arrays.

2. Arrays

Arrays are ideal for storing collections of related data under a single variable name.

This can range from simple lists, like user names or product IDs, to more complex data structures, like records from a database.

2. Arrays

Arrays are often used to store configurations and settings for applications.

This allows for a centralized place where settings can be easily modified and accessed.

Arrays are used to collect and process data submitted via web forms.

PHP automatically organizes submitted form data into associative arrays, making it easy to work with input data.

2. Arrays

Arrays play a crucial role in working with JSON data. PHP provides functions like json_encode() and json_decode() to convert arrays to JSON format and vice versa, facilitating data interchange between servers and web applications or APIs.

2. Arrays

Session variables in PHP are stored as arrays, which makes arrays an integral part of managing user sessions in web applications.

This allows for the temporary storage of user data across multiple pages.

2. Arrays

Arrays provide the foundation for sorting and filtering data, which is essential for organizing and presenting data in web applications, such as sorting a list of products by price or filtering a list of items based on user input.

2. Arrays

These use cases highlight the versatility and importance of arrays in PHP programming.

Whether you're handling simple data lists or implementing complex data structures and algorithms, arrays offer a powerful way to manage and manipulate data in your applications.

2. Arrays

In PHP, we can find three types of arrays

- Indexed Arrays
- Associative Arrays
- Multidimensional Arrays

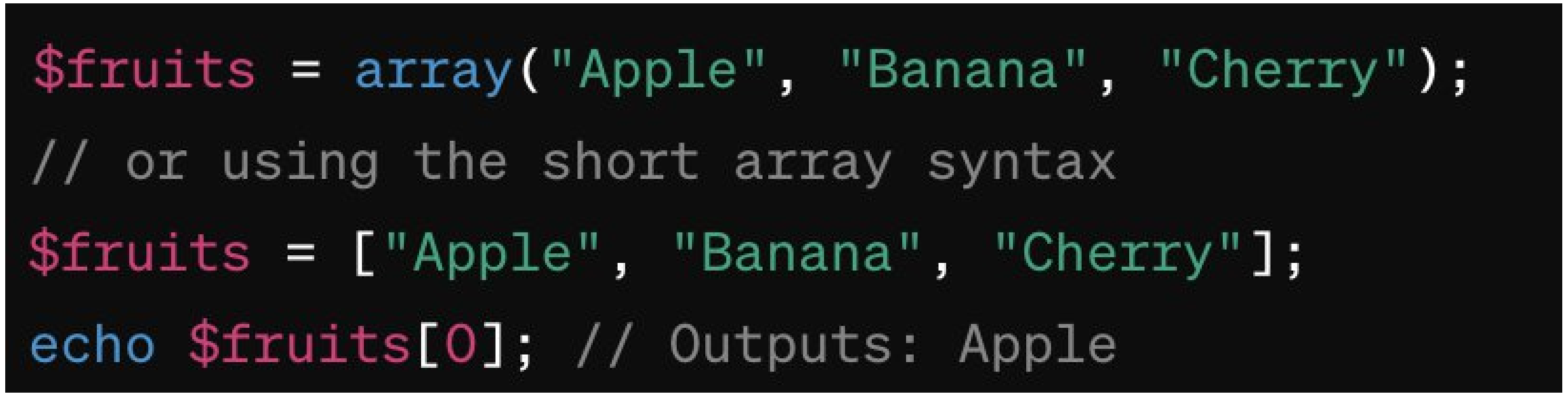
They all have their specific use case, and particularity.

2. Arrays

Indexed Arrays

These arrays use numeric indexes (starting from 0) to access their elements.
This is the most basic type of array.

2. Arrays



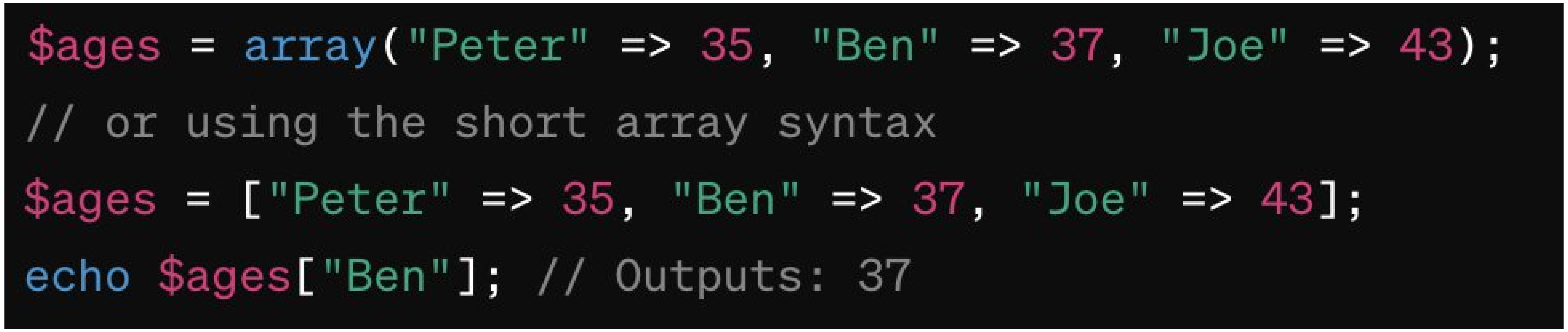
```php
$fruits = array("Apple", "Banana", "Cherry");
// or using the short array syntax
$fruits = ["Apple", "Banana", "Cherry"];
echo $fruits[0]; // Outputs: Apple
```

2. Arrays

Associative Arrays

These arrays use named keys that you assign to them to access their elements.

2. Arrays



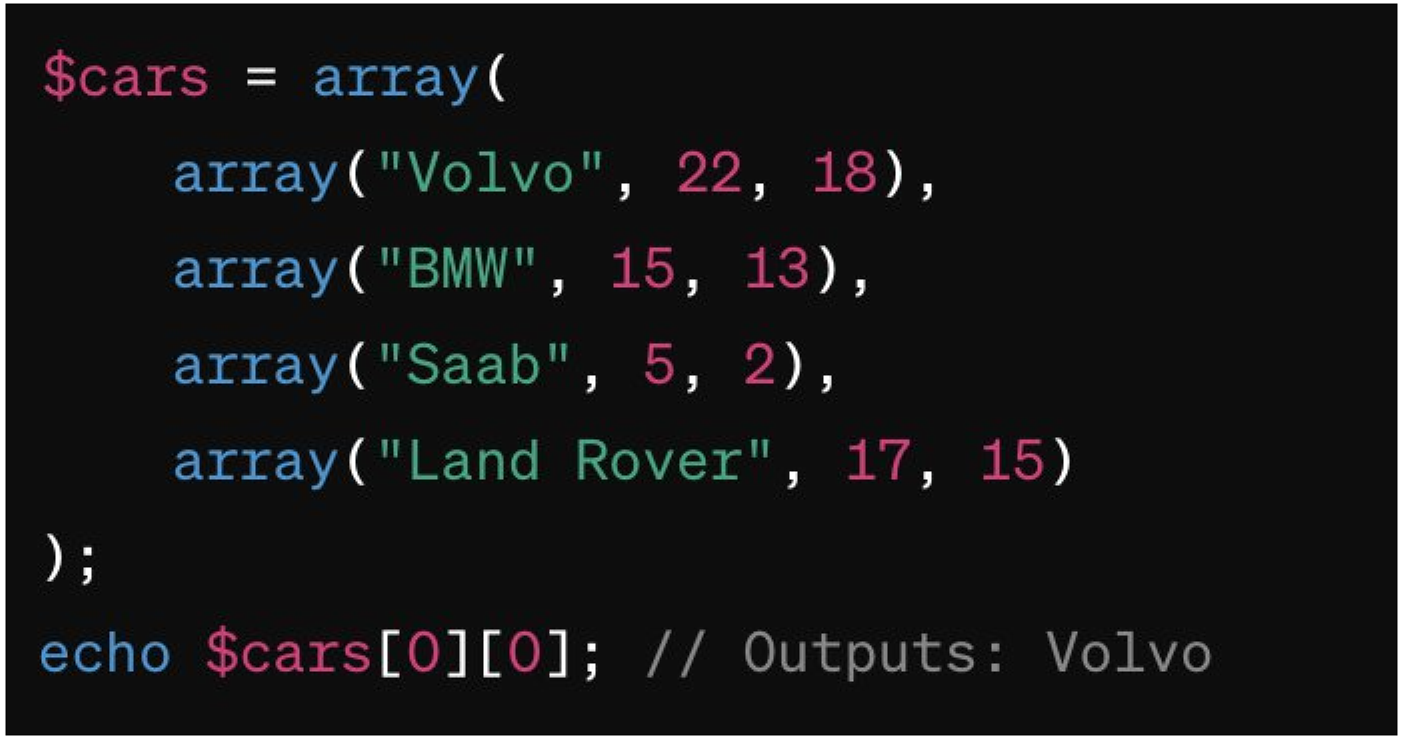
```php
$ages = array("Peter" => 35, "Ben" => 37, "Joe" => 43);
// or using the short array syntax
$ages = ["Peter" => 35, "Ben" => 37, "Joe" => 43];
echo $ages["Ben"]; // Outputs: 37
```

2. Arrays

Multidimensional Arrays

Arrays that contain one or more arrays.

2. Arrays



```
$cars = array(
    array("Volvo", 22, 18),
    array("BMW", 15, 13),
    array("Saab", 5, 2),
    array("Land Rover", 17, 15)
);
echo $cars[0][0]; // Outputs: Volvo
```

2. Arrays

We just saw how we can create a new array. But this is not the full extend of functionalities allowed by PHP.

As you can expected, you want to be able to count the number of items in an array, add a specific values or remove one.

For all of that, PHP provides some core functions to let you do that.

2. Arrays

The first and more important is to see how we can access one specific value in an array.

You may have seen in the previous example that there is some output. Arrays are working thanks to index. Index can be a number in the case of an indexed array, or a string in an associative array.

2. Arrays

Accessing the value of an indexed array

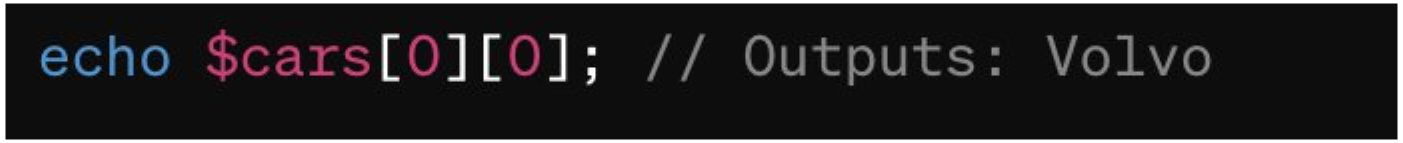
```
echo $fruits[0]; // Outputs: Apple
```

Accessing the value of an associative array

```
echo $ages["Ben"]; // Outputs: 37
```

2. Arrays

If you are working with multidimensional arrays, you can also drill down to access a value of a sub array

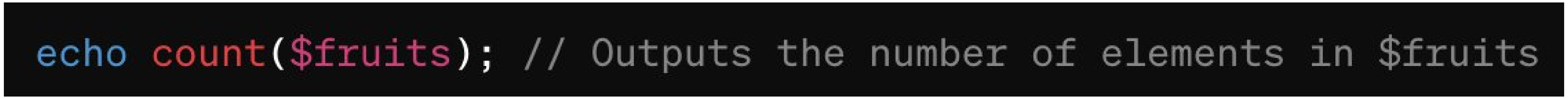


```php
echo $cars[0][0]; // Outputs: Volvo
```

The first [0] will select the first array stored in the cars variable while the second [0] will select the first item of the previously selected array.

2. Arrays

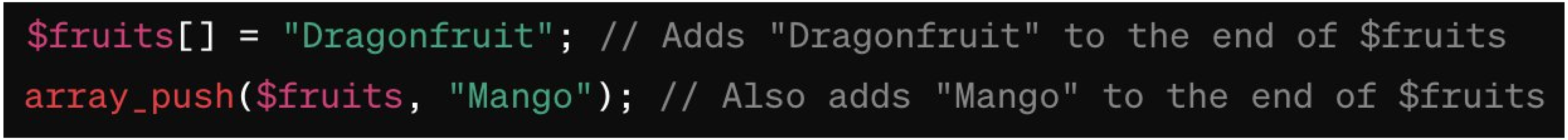
Counting: Use count() to get the number of elements in an array.



```php
echo count($fruits); // Outputs the number of elements in $fruits
```

2. Arrays

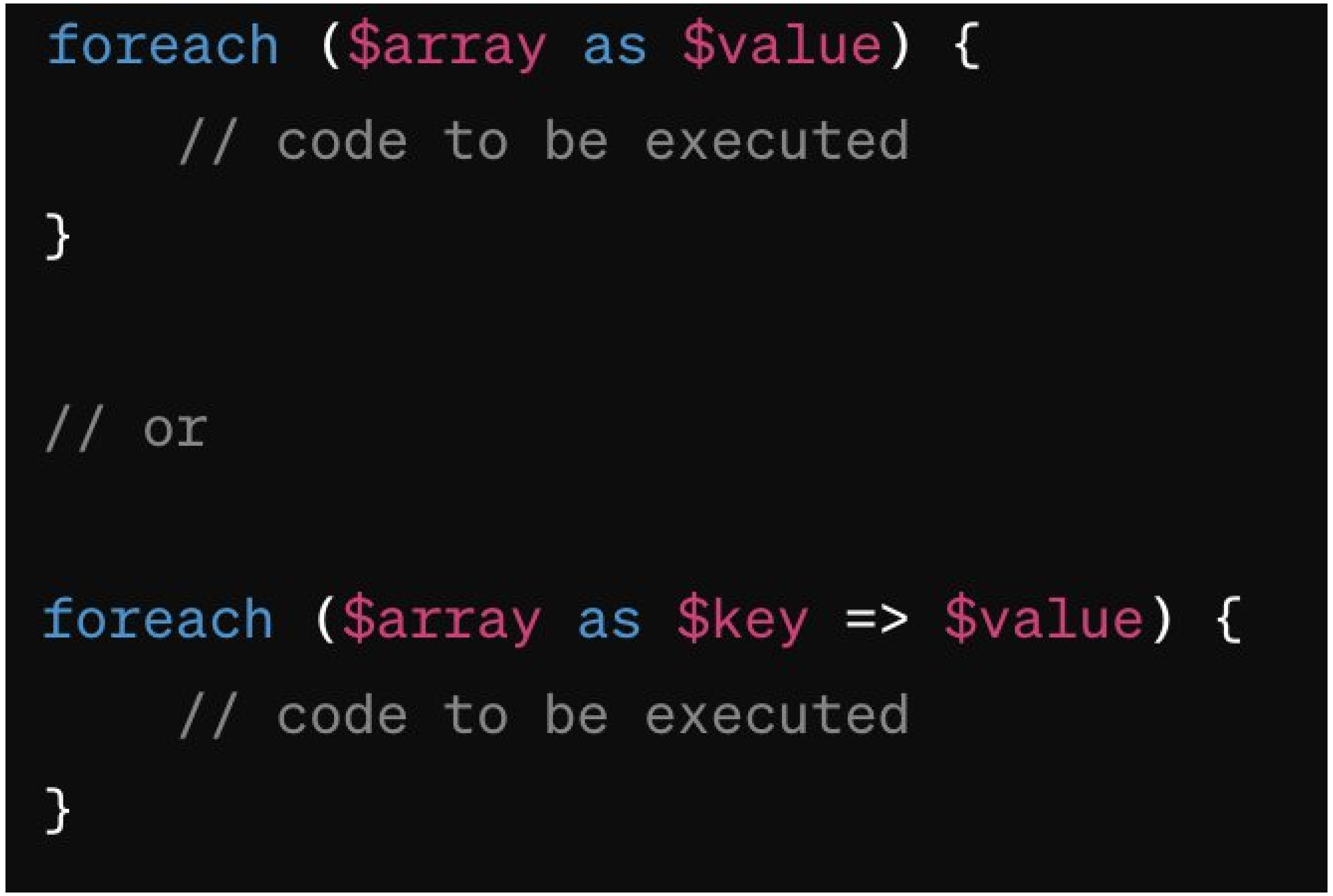
Adding Elements: You can add elements to an array using [] or array_push().



```
$fruits[] = "Dragonfruit"; // Adds "Dragonfruit" to the end of $fruits
array_push($fruits, "Mango"); // Also adds "Mango" to the end of $fruits
```

2. Arrays

Iterating: Loop through an array with foreach, for, or while. This is another method that can be used to loop over a set of data

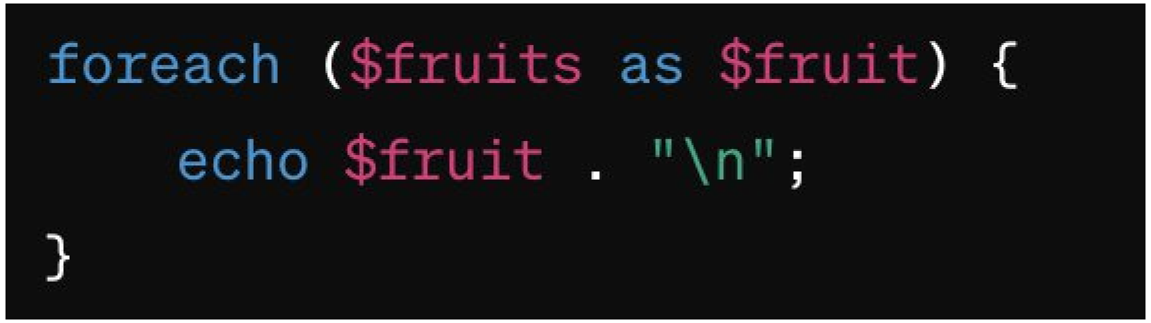


```
foreach ($array as $value) {
    // code to be executed
}


// or


foreach ($array as $key => $value) {
    // code to be executed
}
```

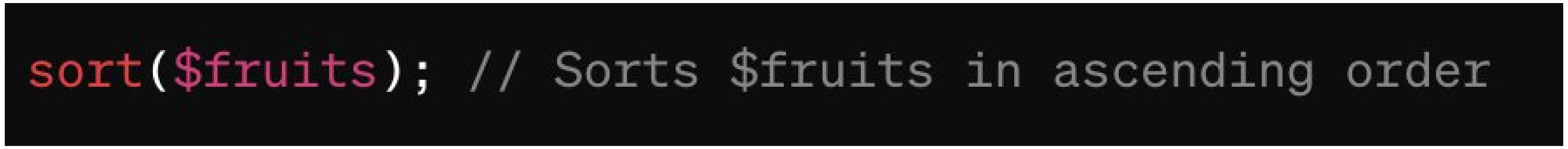
2. Arrays



```
foreach ($fruits as $fruit) {
    echo $fruit . "\n";
}
```

2. Arrays

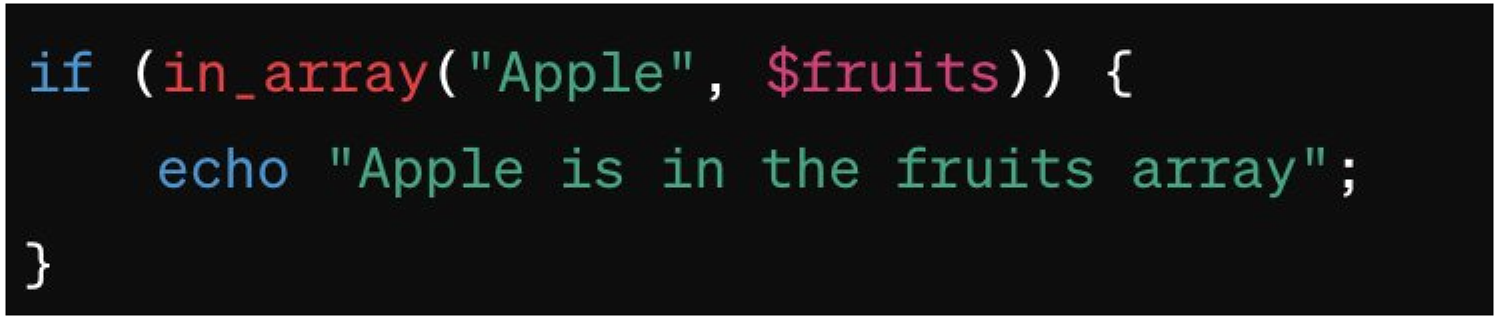
Sorting: PHP provides several functions to sort arrays, such as sort(), rsort(), asort(), ksort(), arsort(), and krsort().



```php
sort($fruits); // Sorts $fruits in ascending order
```

2. Arrays

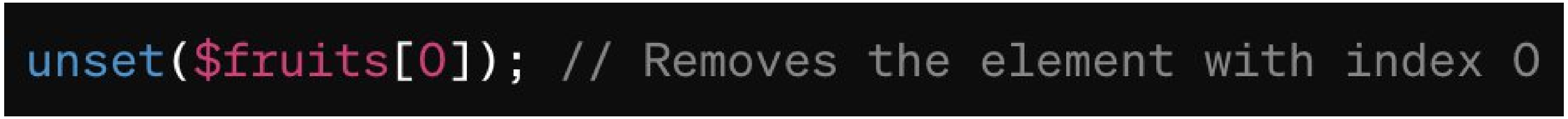
Searching: Use in_array() to check if a value exists in an array, and array_search() to find the key of a value.



```
if (in_array("Apple", $fruits)) {
    echo "Apple is in the fruits array";
}
```

2. Arrays

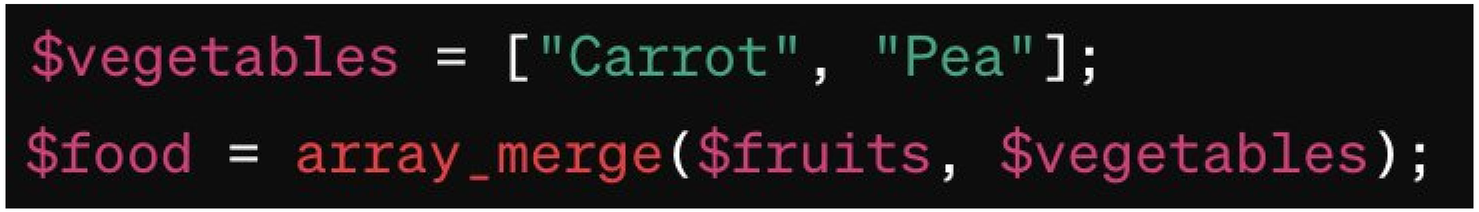
Removing Elements: Use unset() to remove an element by its key.



```php
unset($fruits[0]); // Removes the element with index 0
```

2. Arrays

Merging Arrays: Combine arrays with array_merge().



```php
$vegetables = ["Carrot", "Pea"];
$food = array_merge($fruits, $vegetables);
```

2. Arrays

Technically, like most of language, a string is a simple array of chars. Meaning that you can use some of the previous method to work with a string.

2. Arrays

Arrays in PHP are dynamic, meaning they can grow and shrink as needed, providing a lot of flexibility for storing and manipulating collections of data.

Whether you're handling simple lists of values or complex datasets, PHP's array functions and features offer the tools you need to manage and process your data efficiently.

2. Arrays

To display the full content of an array in PHP, including its structure and all the values it holds, you can use the print_r(), var_dump(), or var_export() functions.

Each of these functions provides a way to output the array's contents, but they differ slightly in how they present the information.

You cannot directly use “echo” to display an array, but you can use it and loop over the array, displaying each line one by one.

2. Arrays

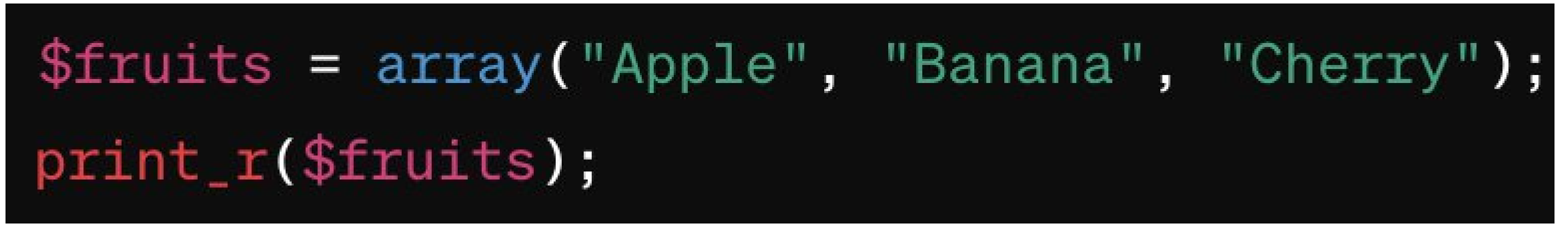
The print_r() function prints human-readable information about a variable.

It's useful for printing the values held in an array, including nested arrays, but it does not show type information of the variable.

```
print_r($array);
```

2. Arrays

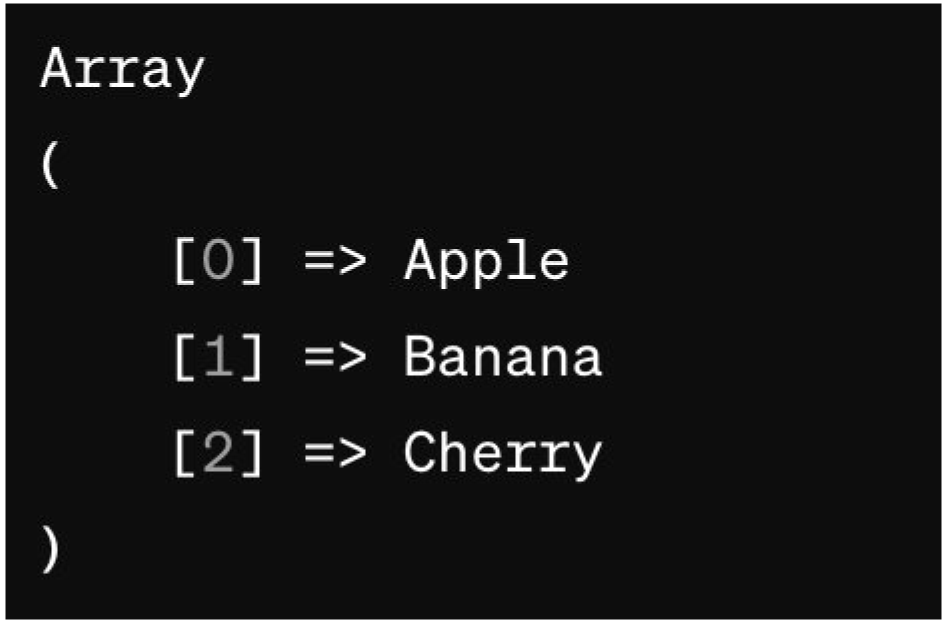
Example



```php
$fruits = array("Apple", "Banana", "Cherry");
print_r($fruits);
```

2. Arrays

Output

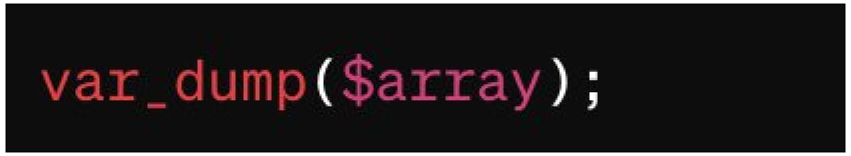


```
Array
(
    [0] => Apple
    [1] => Banana
    [2] => Cherry
)
```

2. Arrays

The var_dump() function provides more detailed information than print_r(), including the data type and size of each element in the array.

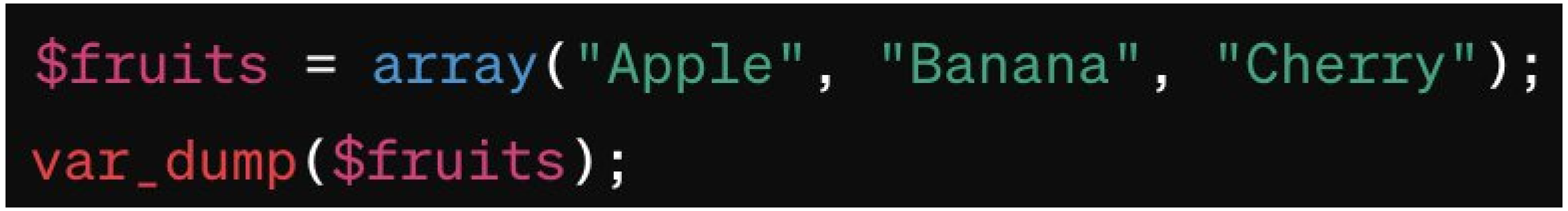
This function is particularly useful for debugging.



```
var_dump($array);
```

2. Arrays

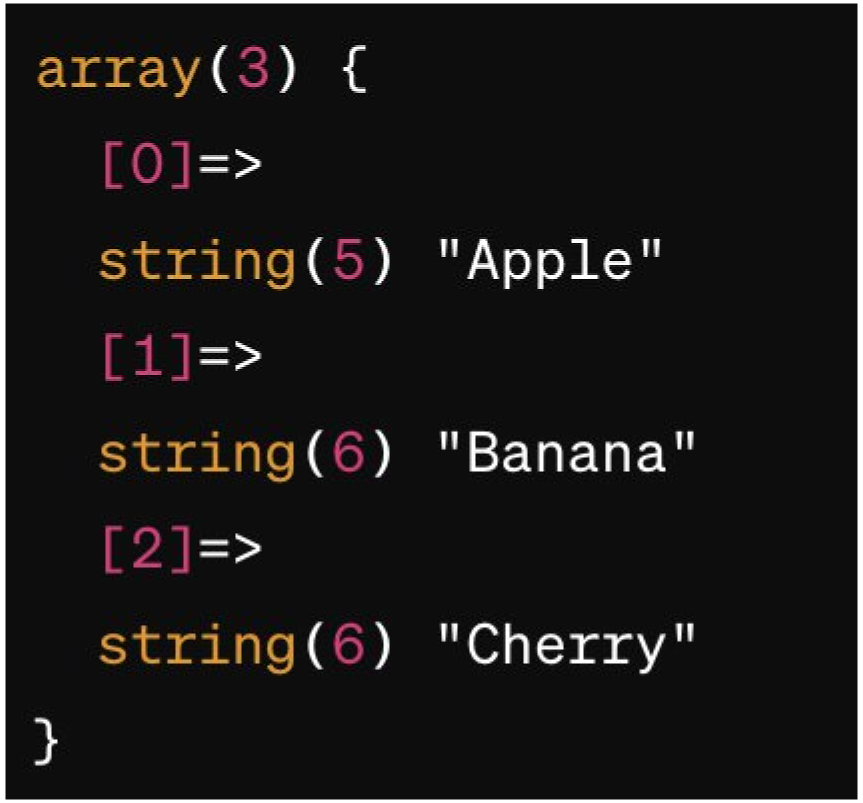
Example



```php
$fruits = array("Apple", "Banana", "Cherry");
var_dump($fruits);
```

2. Arrays

Output



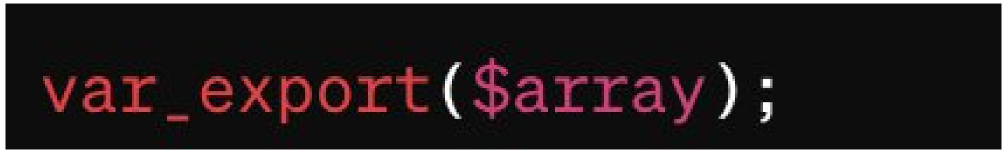
```
array(3) {
  [0]=>
  string(5) "Apple"
  [1]=>
  string(6) "Banana"
  [2]=>
  string(6) "Cherry"
}
```

2. Arrays

The var_export() function outputs or returns a parsable string representation of a variable.

This is similar to var_dump() but the output is valid PHP code.

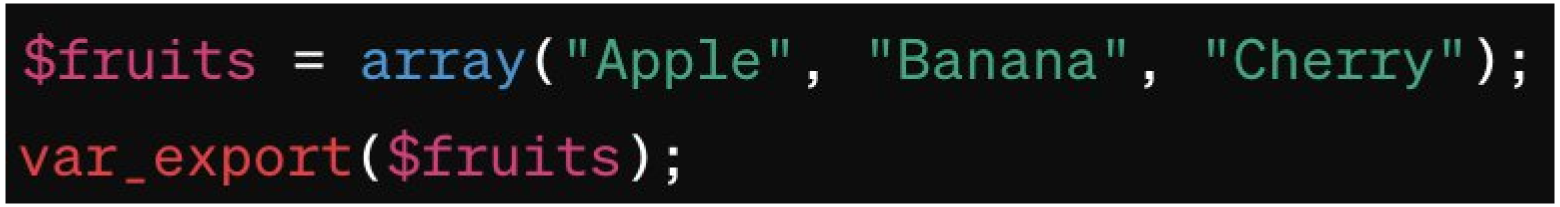
It's useful when you need to see a representation of the array that can be used directly in PHP scripts.



```php
var_export($array);
```

2. Arrays

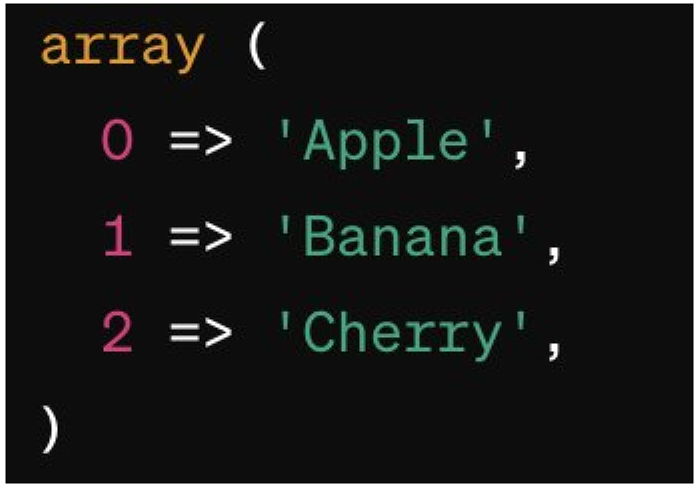
Example



```php
$fruits = array("Apple", "Banana", "Cherry");
var_export($fruits);
```

2. Arrays

Output



```
array (
  0 => 'Apple',
  1 => 'Banana',
  2 => 'Cherry',
)
```

Exercises

3. Performance

3. Performance

When working with loops and arrays in PHP, performance considerations become particularly important as the size of the data grows.

The efficiency of your code can significantly impact execution time, memory usage, and overall performance of your PHP application.

3. Performance

Minimize Work Inside Loops

The more work done inside a loop, the slower the loop executes, especially if the loop iterates many times. To optimize:

- Move calculations or code that does not depend on the loop iteration outside of the loop.
- Avoid unnecessary function calls within the loop, especially heavy operations like database queries or complex calculations.

3. Performance

Use Appropriate Loop Constructs

Different loop constructs (for, foreach, while, do-while) have slightly different performance characteristics, but the differences are often negligible.

- foreach is generally the most efficient way to iterate over arrays because it's specifically optimized for that purpose in PHP.
- For simple iteration over a range of numbers, a for loop is equally efficient.

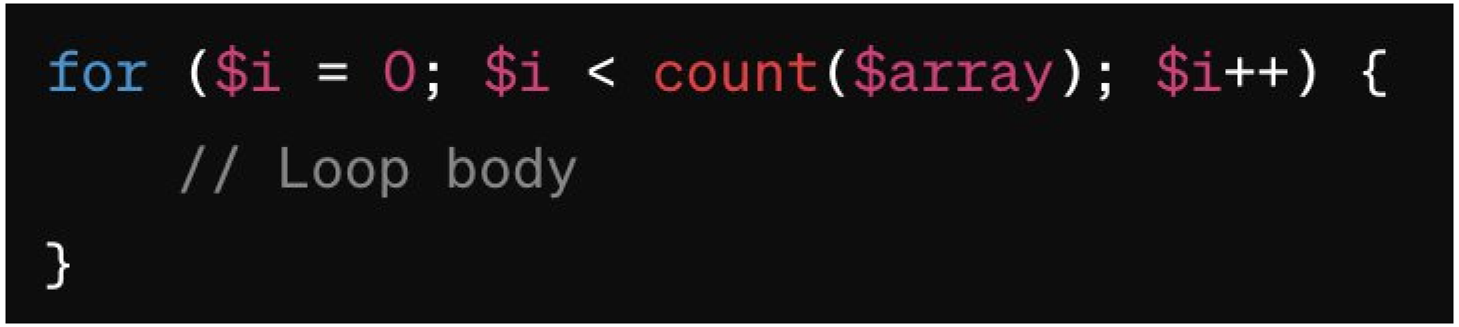
3. Performance

Count Outside Loops

If you're using a for loop to iterate over an array, avoid calling functions like count() in the condition part of the loop, as it will be executed on every iteration.

3. Performance

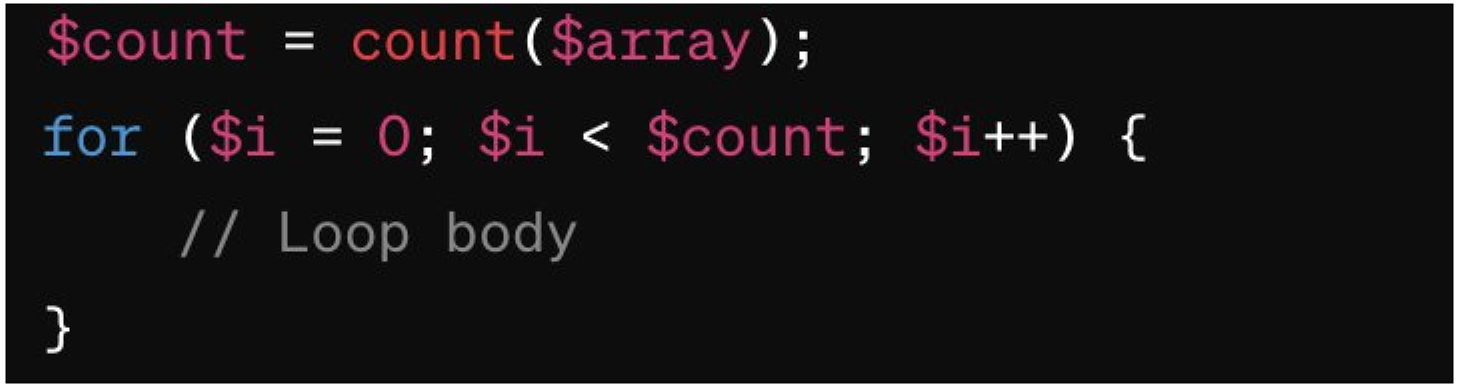
Less efficient



```
for ($i = 0; $i < count($array); $i++) {
    // Loop body
}
```

3. Performance

More efficient



```
$count = count($array);
for ($i = 0; $i < $count; $i++) {
    // Loop body
}
```

3. Performance

Reference Large Arrays & Objects If Necessary

When working with very large arrays or objects within loops, consider using references to avoid copying the entire data structure.

However, be cautious with references as they can lead to harder-to-debug code.

3. Performance

Avoid Modifying Arrays Inside Loops

Modifying the structure of an array (e.g., adding or removing elements) while iterating over it can lead to unexpected behavior and performance issues. If you need to modify the array, consider:

- Making a copy of the array solely for iteration.
- Collecting changes during iteration and applying them after the loop.

3. Performance

Preallocate Array Size

If the final size of an array is known before the loop starts, preallocating the array size can sometimes improve performance by reducing the number of memory reallocations needed as the array grows.

3. Performance

Use Built-in Functions

PHP's built-in array functions are often faster and more memory-efficient than manually iterating over arrays and implementing the same functionality.

For example, use array_map(), array_filter(), array_reduce(), sort(), etc., where appropriate.

3. Performance

Optimizing loops and arrays in PHP is about writing clean, efficient code and knowing when and how to use PHP's built-in functions to your advantage.

While PHP's performance has greatly improved with recent versions, especially with PHP 7.x and PHP 8.x, applying these best practices can further enhance your application's responsiveness and efficiency.

3. Performance

Remember, the key to optimization is measuring; always benchmark and test your optimizations to ensure they have the desired effect.

Exercises