PHP Flow Control

Conditional Statements, Loops,
Exit, Require

greenwich.edu.vn

UNIVERSITY of GREENWICH

Alliance with FPT Education

Table of Contents

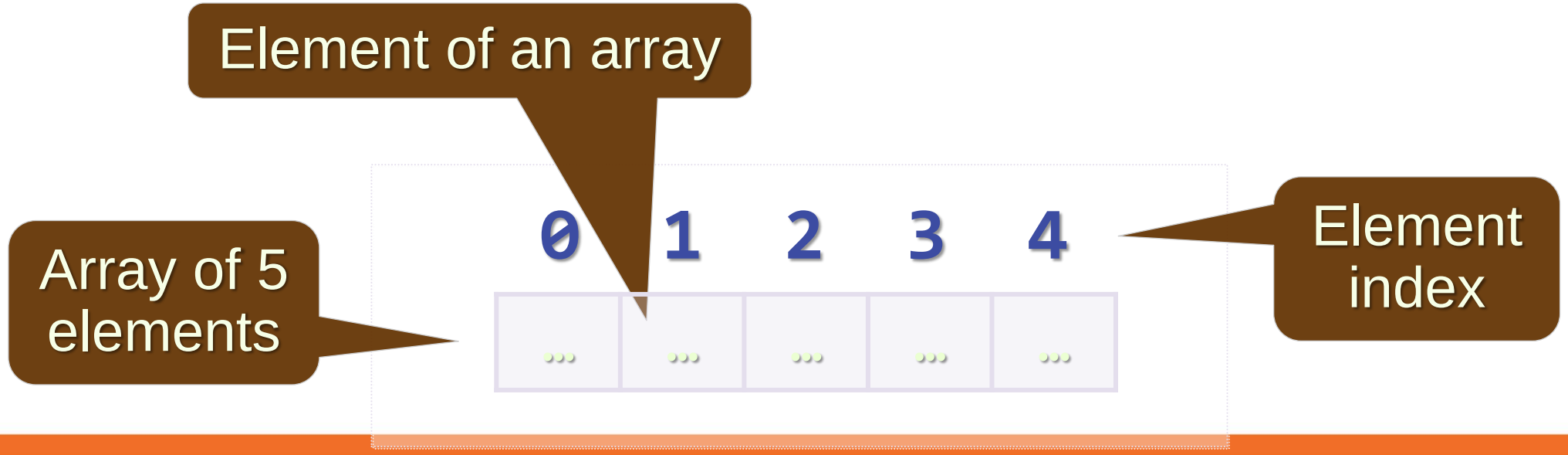
UNIVERSITY of GREENWICH

Alliance with FPT Education

ARRAYS IN PHP

What are Arrays?

- An array is a ordered sequence of elements
  - The order of the elements is fixed
  - Can get the current length (`count($array)`)
  - In PHP arrays can change their size at runtime (add / delete)



UNIVERSITY of GREENWICH

Alliance with FPT Education

CREATING ARRAYS

Initializing Arrays

- There are several ways to initialize an array in PHP:

array(elements)

```
$newArray = array(1, 2, 3); // [1, 2, 3]
```

[]:

```
$newArray = [7, 1, 5, 8]; // [7, 1, 5, 8]
```

array_fill($startIndex, $count, $value)

```
$newArray = array_fill(0, 3, "Hi"); // ["Hi", "Hi", "Hi"]
```

Initializing Arrays – Examples

```
// Creating an empty array
$emptyArray = array();

// Creating an array with 10 elements of value 0.0
$myArray = array_fill(0, 10, 0.0);

// Clearing an array
$myArray = array();

// Adding string elements
$colors = ['green', 'blue', 'red', 'yellow', 'pink', 'purple'];
```

UNIVERSITY of GREENWICH

Alliance with FPT Education

ACCESSING ARRAY ELEMENTS

Read and Modify Elements by Index

Accessing Array Elements

- Array elements are accessed by their **key** (index)
  - Using the **[]** operator
  - By default, elements are indexed from **0** to **count($arr)-1**

0	1	2	3	4
Apple	Pear	Peach	Banana	Melon

  - Values can be accessed / changed by the **[]** operator

```
$fruits = ['Apple', 'Pear', 'Peach', 'Banana', 'Melon'];
echo $fruits[0]; // Apple
echo $fruits[3]; // Banana
```

Accessing Array Elements (2)

- Changing element values

```
$cars = ['BMW', 'Audi', 'Mercedes', 'Ferrari'];
echo $cars[0]; // BMW
$cars[0] = 'Opel';
print_r($cars); // Opel, Audi, Mercedes, Ferrari
```

- Iterating through an array

```
$teams = ['FC Barcelona', 'Milan', 'Manchester United',
    'Real Madrid', 'Loko Plovdiv'];
for ($i = 0; $i < count($teams); $i++) {
    echo $teams[$i];
}
```

Append to Array

- Arrays in PHP are dynamic (dynamically-resizable)
  - Their size can be changed at runtime through append / insert / delete
- Appending elements at the end:
  - **`array_push($array, $element1, $element2, …)`**
  - Alternative syntax: **`$cars[] = 'Lada';`**

```
$months = array();
array_push($months, 'January', 'February', 'March');
$months[] = 'April';
// ['January', 'February', 'March', 'April']
```

Delete from Array

- **`unset($array[$index])`** – removes element at given position
  - Does ***NOT*** reorder indexes

```
$array = array(0, 1, 2, 3);
unset($array[2]);
print_r($array); // prints the array

// Array ([0] => 0 [1] => 1 [3] => 3)
```

Indices remain unchanged

  - Use **`array_splice()`** in case proper ordering is important

Delete / Insert in Array

- `array_splice($array, $startIndex, $length)` – removes the elements in the given range

```
$names = array('Maria', 'John', 'Richard', 'George');
array_splice($names, 1, 2); // ['Maria', 'George']
```

- `array_splice($array, $startIndex, $length, $element)` – removes the elements in given range and inserts an element

```
$names = array('Jack', 'Melony', 'Helen', 'David');
array_splice($names, 2, 0, 'Don');
// ['Jack', 'Melony', 'Don', 'Helen', 'David']
```

Displaying Arrays

- There are several ways of displaying the entire content of an array:

```
$names = ['Maria', 'John', 'Richard', 'Hailey'];
```

  – **`print_r($names)`** – prints the array in human-readable form

```
Array ( [1] => Maria [2] => John [3] => Richard [4] => Hailey )
```

  – **`var_export($names)`** – prints the array in array form

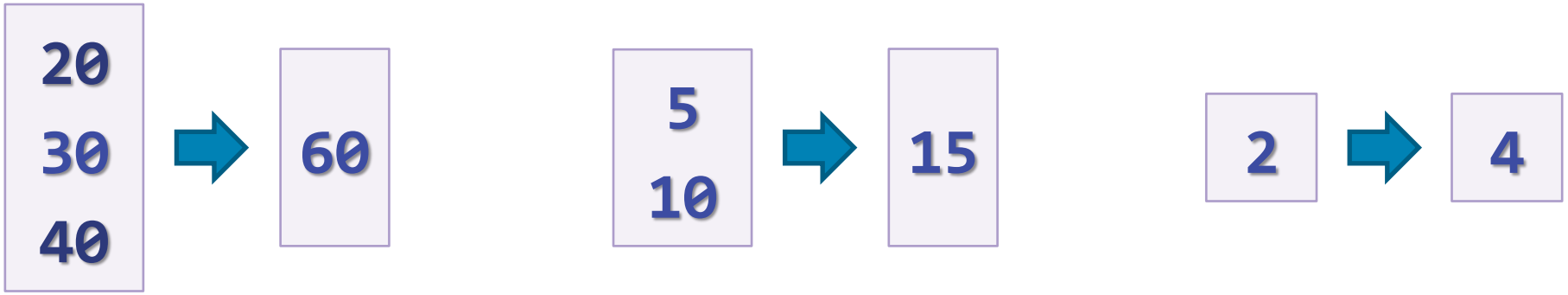
```
array ( 1 => 'Maria', 2 => 'John', 3 => 'Richard', 4 => 'Hailey', )
```

  – **`echo json_encode($names)`** – prints the array as JSON string

```
["Maria","John","Richard","Hailey"]
```

Problem: Sum First and Last Array Elements

- You are given array of strings holding numbers
  - Calculate and print the sum of the first and the last elements




```
$array = [20, 30, 40];
$array_num = count($array);
echo $arr[0] + $arr[$array_num - 1];
```

Problem: Print Array Elements

- Enter array elements from html form - array of strings holding numbers - comma separated
  - Print the array as follows

```
20
30
40
50
60
70
```

→

```
20 70
30 60
40 50
```

```
7
3
4
5
10
```

→

```
7 10
3 5
4
```

```
2
```

→

```
2
```

UNIVERSITY of GREENWICH

Alliance with FPT Education

IF AND IF-ELSE

Implementing Conditional Logic

Conditional Statements: `if-else`

- PHP implements the classical **if** / **if-else** statements:

```
$number = 5;

if ($number % 2 == 0) {
    echo "This number is even.";
} else {
    echo "This number is odd.";
}
```

Alternative If Syntax

- PHP offers an alternative syntax for the **if**-statement:

```
<?php
    $a == 5;
    if ($a == 5):
        echo "a equals 5";
        echo "...";
    elseif ($a == 6):
        echo "a equals 6";
        echo "!!!";
    else:
        echo "a is neither 5 nor 6";
    endif;
?>
```

Problem: Odd / Even

- Check if a number is odd or even or invalid

```
$num = $_GET['num'];
$rem = fmod($num, 2);
if ($rem == 0) {
    echo "even";
} else if ($rem == round($rem)) {
    echo "odd";
} else {
    echo "invalid";
}
```

5	→	odd
8	→	even
-3	→	odd
1.5	→	invalid
0	→	even

SWITCH-CASE

Making Several Comparisons at Once

The `switch-case` Statement

- Selects for execution a statement from a list depending on the value of the **switch** expression

```
switch ($day) {
    case 1: echo('Monday'); break;
    case 2: echo('Tuesday'); break;
    case 3: echo('Wednesday'); break;
    case 4: echo('Thursday'); break;
    case 5: echo('Friday'); break;
    case 6: echo('Saturday'); break;
    case 7: echo('Sunday'); break;
    default: echo('Error!'); break;
}
```

Alternative Switch Syntax

- PHP offers an alternative syntax for **switch** constructs:

```
$variable = 2;
switch ($variable):
    case 1:
        echo "<div>News</div>";
        break;
    case 2:
        echo "<div>Forum</div>";
        break;
endswitch;
```

Problem: Fruit or Vegetable

- Print "fruit", "vegetable" or "unknown" depending on the input string
  - Fruits are: banana, apple, kiwi, cherry, lemon, grapes, peach
  - Vegetable are: tomato, cucumber, pepper, onion, garlic, parsley
  - All others are unknown

`lemon`	→	`fruit`	`peach`	→	`fruit`
`onion`	→	`vegetable`	`pizza`	→	`unknown`

Solution: Fruit or Vegetable

```
$word = $argv[1];
  switch ($word) {
    case 'banana':
    case 'apple':
    case 'kiwi':
    case 'cherry':
    case 'lemon':
    case 'grapes':
    case 'peach':
      echo 'fruit';
      break;
    case 'tomato':
    case 'cucumber':
    case 'pepper':
    case 'onion':
    case 'parsley':
    case 'garlic':
      echo 'vegetable';
      break;
    default:
      echo 'unknown';
  }
```

UNIVERSITY of GREENWICH

Alliance with FPT Education

LOOPS

While Loop

- While loops repeat a block while a certain condition is true:

```
while (expr) {
    statement;
}
```

```
while (expr):
    statement;
endwhile;
```

```
while ($count < 10) {
    $count++;
    echo $count;
}
```

```
while ($count < 10):
    $count++;
    echo $count;
endwhile;
```

While Loop – Example

- Printing the numbers from 1 to 10:

```
$counter = 1;
while ($counter <= 10) {
  echo "<p>$counter</p>";
  $counter++;
}
```

While Loop – Alternative Syntax

- While loops have alternative syntax, without { }
- Printing the Unicode characters from &#0; to ᎈ

```
<?php
$charCode = 0;
while ($charCode <= 5000) : ?>
    <div style="display: inline-block; width:80px">
    <?= $charCode++ ?> -> &#<?= $charCode ?>;
    </div>
<?php endwhile; ?>
```

<?= … ?> is like
<?php echo … ?>

Do-While Loop

- Do-while loops repeat a code block until some condition breaks
  - The loop body executes at least once

```
$i = 10;
do {
    echo $i . " ";
    $i--;
} while ($i > 0);
```

```
10 9 8 7 6 5 4 3 2 1
```

```
$i = -6;
do {
    echo $i . " ";
    $i--;
} while ($i > 0);
```

```
-6
```

For Loops

- The classical **for**-loop syntax is:

```
for (initialization; test; update) {
    statements;
}
```

- Example:

```
for ($i=0; $i < 10; $i++) {
    echo $i;
}
```

For Loop – Examples

- A simple **for**-loop to print the numbers 0...9:

```
for ($number = 0; $number < 10; $number++) {
    echo $number . " ";
}
```

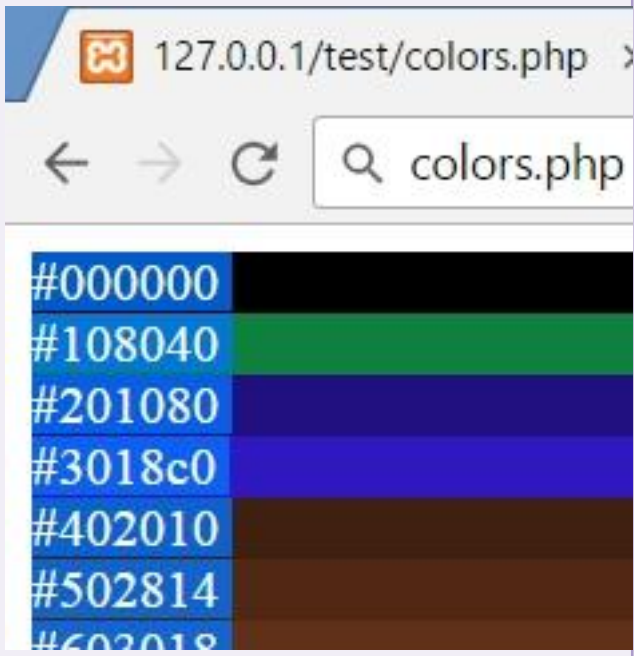
- A simple **for**-loop to calculate n!:

```
$n = 5; $factorial = 1;
for ($i = 1; $i <= $n; $i++) {
    $factorial *= $i;
}
```

For-Loop – Alternative Syntax

- Printing blocks of different colors:

```
<?php
for ($r=0, $g=0, $b=0; $r < 256; $r+=16, $g+=8, $b+=4) :
    $color = "#" .
    str_pad(dechex($r), 2, '0') .
    str_pad(dechex($g), 2, '0') .
    str_pad(dechex($b), 2, '0');
?>
  <div style="width:400px;
   background-color:<?= $color ?>">
    <?= $color ?>
  </div>
<?php endfor; ?>
```



dechex - Converts decimal to hexadecimal

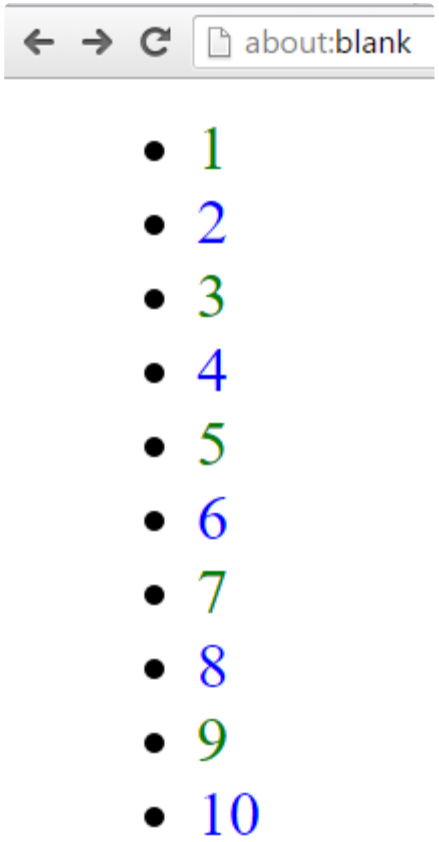
Problem: Colorful Numbers 1 … n

- Print all the numbers from 1 to n
  - Return a string holding HTML list `<ul><li>…</li></ul>`
  - Display the odd lines in `blue`, even lines in `green`

```
<ul>
  <li><span style='color:blue'>1</span></li>
  <li><span style='color:green'>2</span></li>
  <li><span style='color:blue'>3</span></li>
  …
</ul>
```

Solution: Colorful Numbers 1 … n

```
$n = 10;
$html = '<ul>';
for ($i = 1; $i <= $n; $i++) {
  $color = 'blue';
  if ($i % 2 != 0) {
    $color = 'green';
  }
  $html .= "  <li><span style='color:
     $color'>$i</span></li>";
}
$html .= '</ul>';
```



Foreach (1)

- The **foreach** construct iterates over arrays and objects

```
foreach (array_expression as $value) {
    statements;
}
```

- Print all items of an array:

```
$colors = ["red", "green", "blue"];
foreach ($colors as $value) {
    echo "$value <br>";
}
```

Foreach (2)

- Iterate over the key-value pairs in associative array:

```
foreach (array_expression as $key => $value) {
    statements;
}
```

- Print all items of an array and their keys:

```
$colors = ["one" => "red", "two" => "green"];
foreach ($colors as $key => $value) {
    echo "k-> $key  v-> $value <br>";
}
```

Foreach – Alternative Syntax

- Iterating over object properties:

```
<?php
$colors = (object)[];
$colors->red = "#F00";
$colors->slateblue = "#6A5ACD";
$colors->orange = "#FFA500";

foreach ($colors as $key => $value) : ?>
    <p style="background-color:<?= $value ?>">
        <?= $key ?> -> <?= $value ?>
    </p>
<?php endforeach; ?>
```

Continue

- `continue` skips to the next loop iteration
- Print all elements of array except 'second'

```
$stack = ['first', 'second', 'third'];
foreach ($stack as $value) {
    if ($value == 'second') {
      continue;
    }
    echo $value.'<br>';
}
```

Break

- break terminates the execution of the loop
- Terminate if 'third' is in array

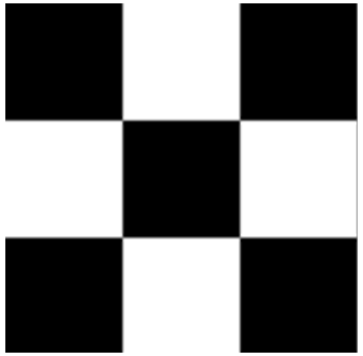
```
$stack = ['first', 'second', 'third'];
foreach ($stack as $value) {
    if ($value == 'third') {
      break;
    }
    echo $value.'<br>';
}
```

Problem: Chessboard

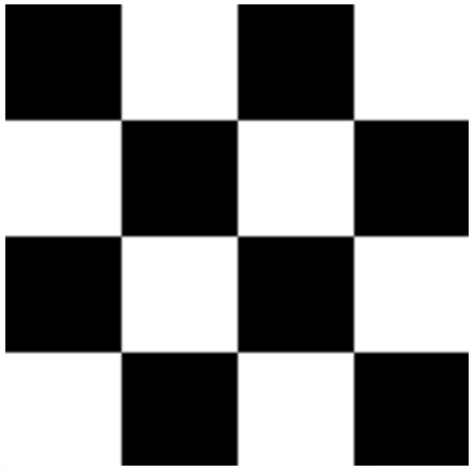
- Print a chessboard of size **n**. Examples:

```
<div class="chessboard">
  <div>
    <span class="black"></span>
    <span class="white"></span>
  </div>
  <div>
    <span class="white"></span>
    <span class="black"></span>
  </div>
  <div>…</div>
</div>
```

n = **3** ➡



n = **4** ➡



Solution: Chessboard

```
$size = 5;
$html = '<div class="chessboard">';
for ($row = 0; $row < $size; $row++) {
  $html .= '  <div>';
  $color = ($row % 2 == 0) ? 'black' : 'white';
  for ($col = 0; $col < $size; $col++) {
     $html .= "    <span class=\"$color\"></span>";
     $color = ($color == 'white') ? 'black' : 'white';
  }
  $html .= '  </div>';
}
$html .= '</div>';
echo $html;
```

Exit

- The **`exit`** statement ends the PHP script execution immediately
  - Used for fatal errors, e.g. missing file / resource / database
- If the statement is a number, that value will be used as the exit status and not printed
- If it is a string, the value is printed before the process terminates

```
$filename = '/path/to/data-file';
$file = fopen($filename, 'r')
    || exit("Unable to open file: $filename");
```

Die

- The function **die()** is an alias for the **exit** statement:

```
die(message);
```

  – The message is required

- The **die()** function prints a message and exits the current script:

```
$db = mysql_connect("localhost", $username, $password);
if ( ! $db) {
  die("Could not connect to database");
}
```

INCLUDE AND REQUIRE

Including a Script from Another Script

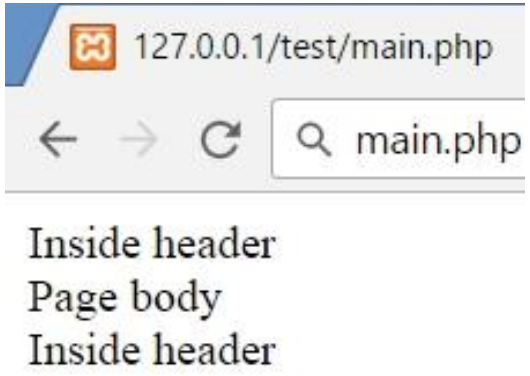
Include and Require

- `include` and `require` load and evaluate a file holding PHP code

main.php
`require "header.php";` `echo "page body ";` `include "footer.php";`

header.php
`echo….`

footer.php
`echo….`



- Difference between `include` and `require`:
  - If file is not found `include` produces a warning
    - `require` produces a fatal error

include_once and require_once

- With include and require you can include one file many times and each time it is evaluated

main.php

```
require "header.php";
echo "Page body<br>";
include "header.php";
```

header.php

```
function test();
```

footer.php

```
function test();
```

  – With include_once and require_once if file is already included, nothing happens

```
Fatal error: Cannot redeclare test()…
```

Summary

- If-else statements are as in C#, Java and C++
  - Alternative syntax: **`if-elseif-endif`**
- Switch-case statement are similar to Java / C#
- PHP supports the classical loop statements
  - **`While`**, **`do…while`**, **`for`**, **`foreach`**
- Including PHP code in another PHP code
  - **`include`** / **`include_once`** / **`require`** / **`require_once`**