New Features of PHP 7

Session 3

Objectives

- *Explain the `intdiv()` function.*
- *Explain spaceship operator.*
- *Explain null coalescing operator.*
- *Describe Scalar Type Declarations.*
- *Explain Uniform Variable Syntax in PHP programs.*
- *Describe use operator.*
- *Explain closure call methods in the PHP programs.*
- *Explain Generator delegation via yield from.*
- *Explain Levels parameter of the `dirname()` function.*

Integer Division

- Is a division operator
- Is represented as /
- In earlier versions, always returned a `float` value and one had to use workarounds to get integers such as:

Snippet

```
<?php
$x = 10;
$y = 2;
$z = (int) ($x / $y);
echo $z;
?>
```

`$x` and `$y` contain float values. Therefore, using the division operator may not provide accurate results. Here, the value returned is cast to integer using int() function.

Integer Division

- In PHP 7, the integer division function accepts two parameters:
  - Dividend
  - Divisor
- It returns an integer result

Syntax

```
intdiv(m, n)
```

Where,

- **intdiv –** is the function
- **m** – is the dividend
- **n** – is the divisor

Integer Division

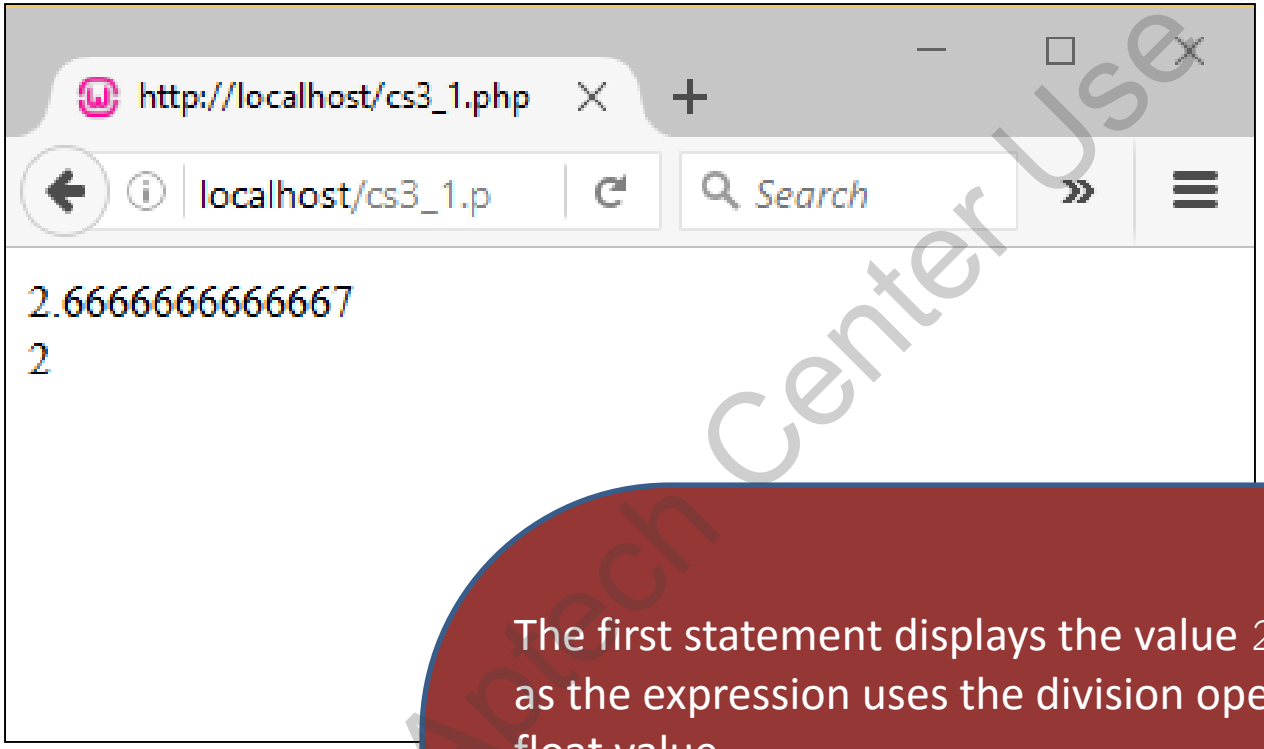
- Demonstrating the use of the `intdiv()` function and the behavior of operator '/'.

Snippet

```
<HTML>
<BODY>
<?php
 echo 8/3, "\n";
 echo intdiv(8, 3), "\n";
?>
</BODY>
</HTML>
```

Integer Division

- Displays the following output:

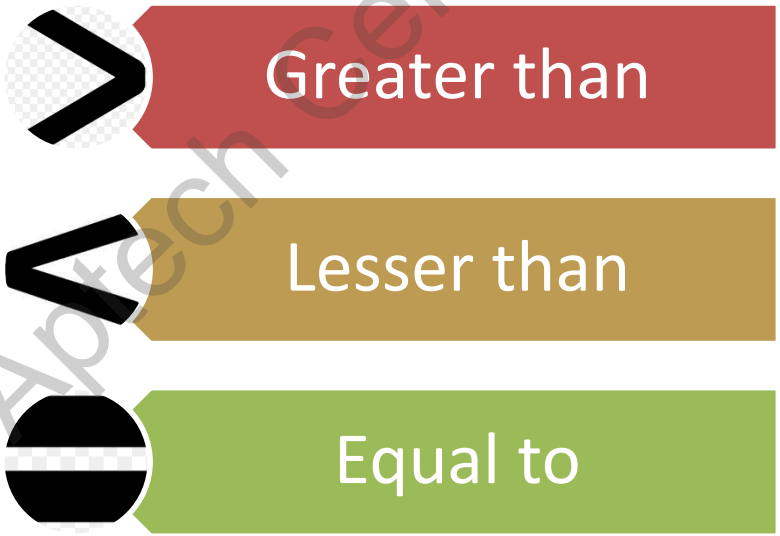


The first statement displays the value `2.6666666666667` as the expression uses the division operator that returns a float value.

The second statement displays the value 2 as the expression is using the function `intdiv()` that returns an integer value after division.

Spaceship Operator (<=>)

- Is a single comparison operator
- Is mainly used for sorting
- Is represented as <=>
- Compares operands against three rules, namely:



Spaceship Operator (<=>)

- Demonstrating the use of spaceship operator

Syntax

```
$x <=> $y
```

- This expression displays the result:
  - -1 if $x is smaller than $y
  - 0 if $x is equal to $y
  - 1 if $x is greater than $y

Spaceship Operator (<=>)

- Demonstrating the use of spaceship operator on numbers

Snippet

```
<HTML>
<BODY>
<?php
$x = 1;
$y = 2;
echo $x.'<=>'.$y,' Returns ', $x <=> $y;
// This will output -1
echo '</br>';
$x = 10;
$y = 10;
echo $x.'<=>'.$y,' Returns ',$x <=> $y;
// This will output 0
```

Spaceship Operator (<=>)


```
echo '</br>';
$x = 10;
$y = 5;
echo $x.'<=>'.$y,' Returns ', $x <=> $y;
// This will output 1
?>
</BODY>
</HTML>
```

Spaceship Operator (<=>)

- Displays the output:

Spaceship Operator (<=>)

- Demonstrating the use of spaceship operator on strings

Snippet

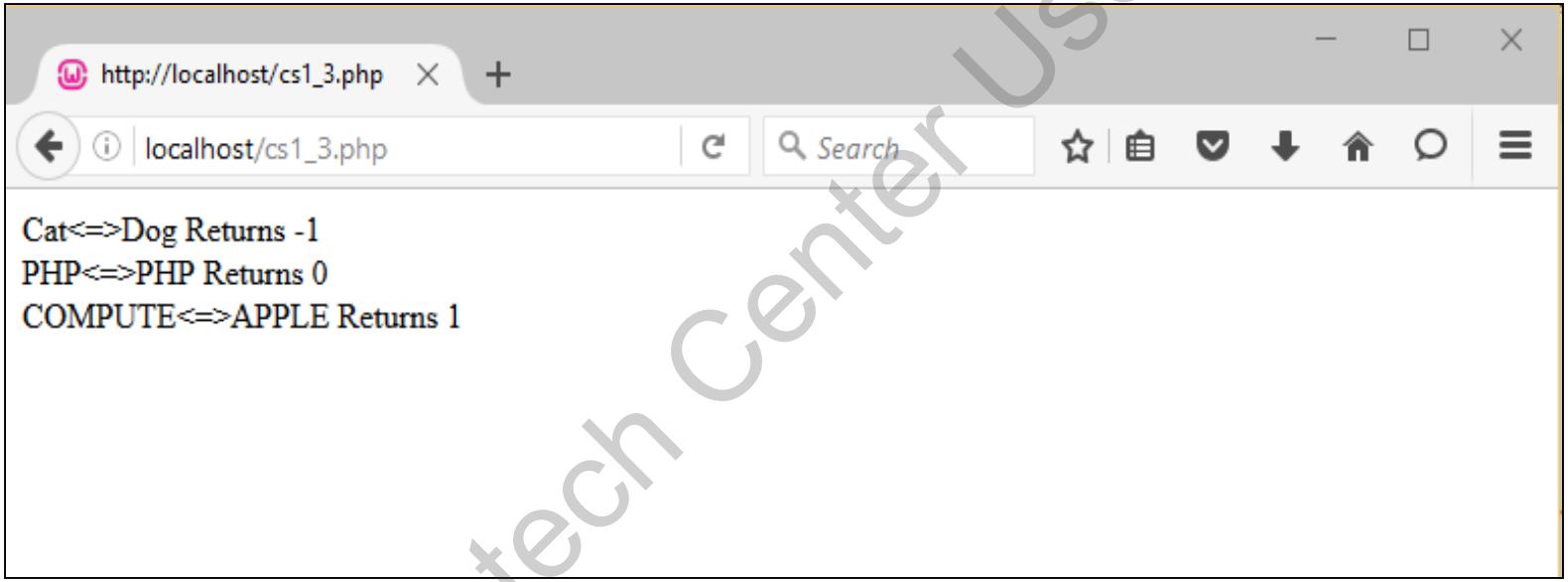
```
<HTML>
<BODY>
<?php
$x = "Cat";
$y = "Dog";
echo $x.'<=>'.$y,' // Returns ', $x <=> $y;
// This will output -1 because Cat is
// less than Dog.
 echo '</br>';
 $x = "PHP";
 $y = "PHP";
 echo $x.'<=>'.$y,' // Returns ',$x <=> $y;

// This will output 0 because both strings have
// same value.
```

Spaceship Operator (<=>)


```
echo '</br>';
 $x = "COMPUTE";
 $y = "APPLE";
 echo $x.'<=>'.$y,' // Returns ',$x <=> $y;
// This will output 1 because "COMPUTE" is
greater than APPLE.
 echo '</br>';
?>
</BODY>
</HTML>
```

Spaceship Operator (<=>)

- Displaying the output:



Spaceship Operator (<=>)

- Demonstrating the use of spaceship operator on arrays

Snippet

```
<HTML>
<BODY>
<?php
$x = array();
$y = array();
echo 'array()'.'<=>'.'array()'.' Returns ', $x <=> $y;
// This will output 0
echo '</br>';
$m = array(1,2, 3);
$n = array(1,2, 3);
$p = array(1,2, 1);
$q = array(1,2, 4);
echo 'array(1,2,3)'.'<=>'.'array(1,2,3)'.' Returns ',$m
<=> $n;
// This will output 0
```

Spaceship Operator (<=>)

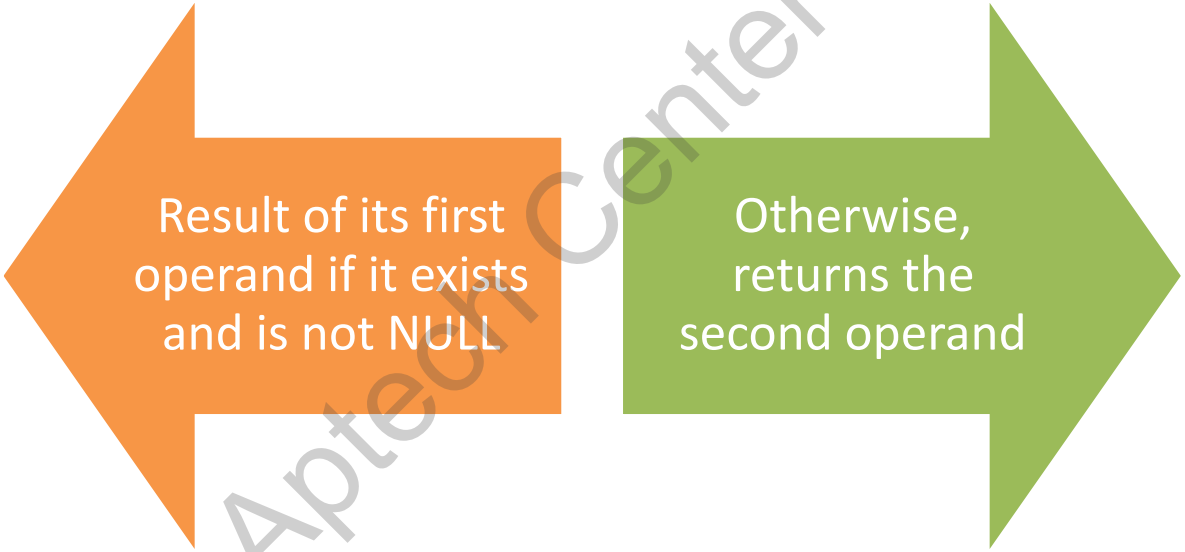

```
echo '</br>';
echo 'array(1,2,3)'.'<=>'.'array()'.' Returns ',
$m <=> $x;
// This will output 1
echo '</br>';
echo 'array(1,2,3)'.'<=>'.'array(1,2,4)'.'
Returns ',$m <=> $q;
// This will output -1
?>
</BODY>
</HTML>
```

Spaceship Operator (<=>)

- Displaying the output:

Null Coalescing Operator (??)

- Is used to check for a NULL value
- Is represented as `??`
- Returns:



Null Coalescing Operator (??)

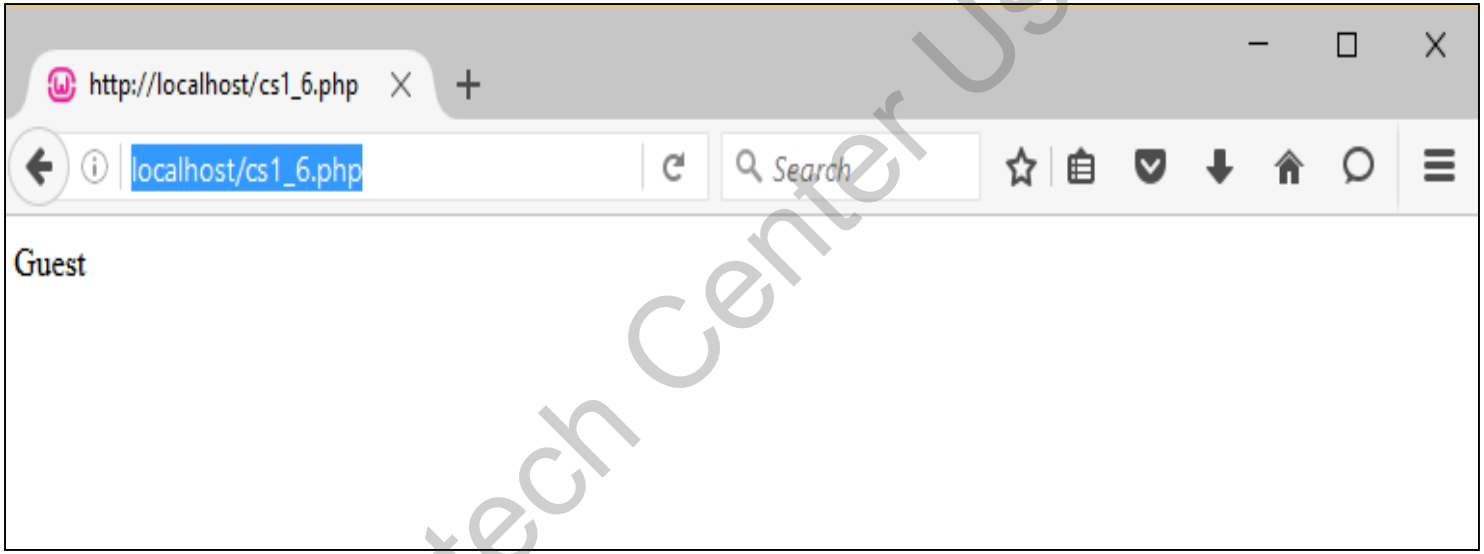
- Demonstrating the use of null coalescing operator

Snippet

```
<HTML>
<BODY>
<?php
$name = $first_name ?? "Guest";
echo $name;
?>
</BODY>
</HTML>
```

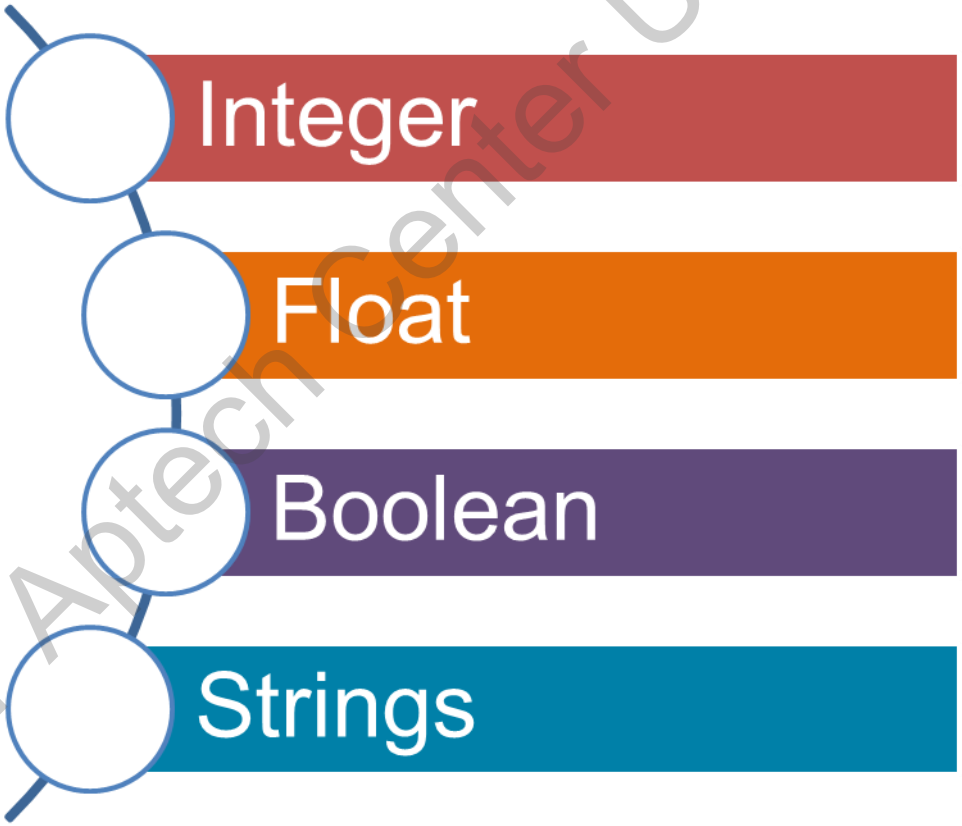
Null Coalescing Operator (??)

- Displaying the output:



Scalar Type Declarations

- Data types that hold single data type are known as scalar data types.
- These can be:



Scalar Type Declarations

- Refers to specifying the data type of a parameter in a function
- Also referred to as type hinting
- Provides hints to a function
- Enforces to input the parameter data type, that can be either strict or coercive

Uniform Variable Syntax

- Allows to use multiple operators in a given expression
- Follows a left-to-right evaluation order
- Allows backward compatibility in old evaluation technique
- Allows nesting of dereferencing operations

- Demonstrating the use of uniform variable syntax

Snippet

```
<?php
 function e() {
 echo "This is e() \n";
 };
 function f() {
 echo "This is f() \n";
 return e;
 };
 function g() {
 echo "This is g()\n";
 return f;
};
 g();
 echo "***********\n";
 g()();
 echo "***********\n";
 g()()();
?>
```

- Displays the following output:

```
This is g()
***********
This is g()
This is f()
***********
This is g()
This is f()
This is e()
```

Use Operator

1-4

- Is used to achieve aliasing
- Allows grouping of multiple declarations
- Enables to group classes, functions, and constants imported from the same namespace

Use Operator

- Demonstrating the use of `use` operator

Snippet

```
<?php
 namespace aptech;
 class Boston {
function say()
{
echo "Boston\n";
}
}
class NewYork {
function say()
{
echo "NewYork\n";
}
}
function foo1()
{
echo "This is foo1()\n";
}
function foo2()
{
echo "This is foo2()\n";
}
?>
```

Save the file as myfile.php.

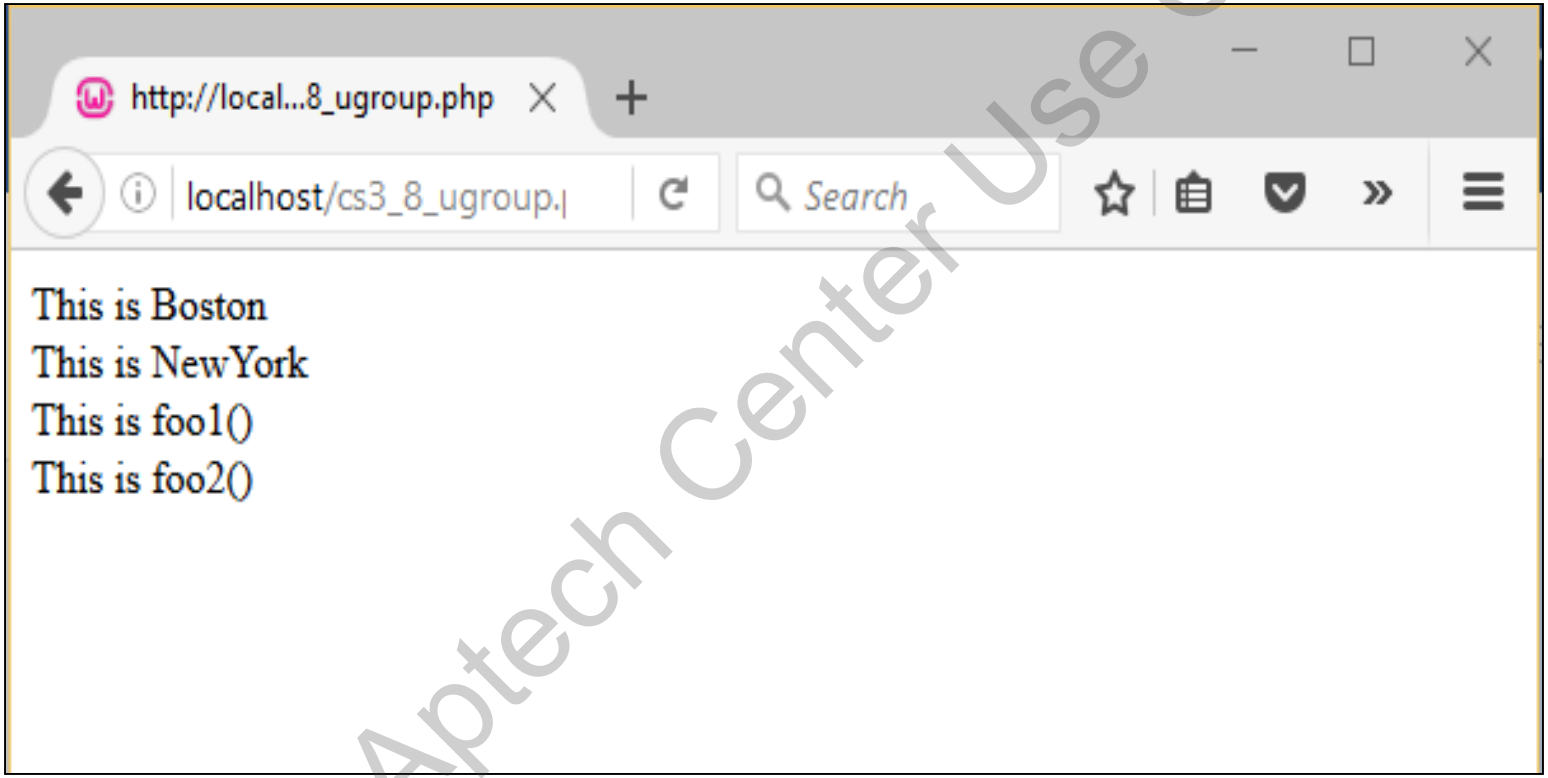
Use Operator

- Include myfile.php in ugroup.php as shown:

Snippet

```
<?php

 include 'myfile.php';

 use aptech\{Boston, NewYork};
 use function aptech\{foo1, foo2};

 $d = new Boston();
 $d->say();

 $n = new NewYork();
 $n->say();

 foo1();
 foo2();
 ?>
```

- Displays the following output:



Closure call () Method

- A closure record stores a function together with an environment.
- A closure method allows the functions to access the captured variables.
- Adding `$this` parameter to the closure method results in two new methods:
  - The Instance method - `Closure->bindTo()`
  - The Static method - `Closure::bind()`

Closure call () Method

Instance Method

- Called at closure instance
- Accepts two arguments

Static Method

- Passed as the first argument to the closure ()
- Accepts two arguments

Closure call () Method

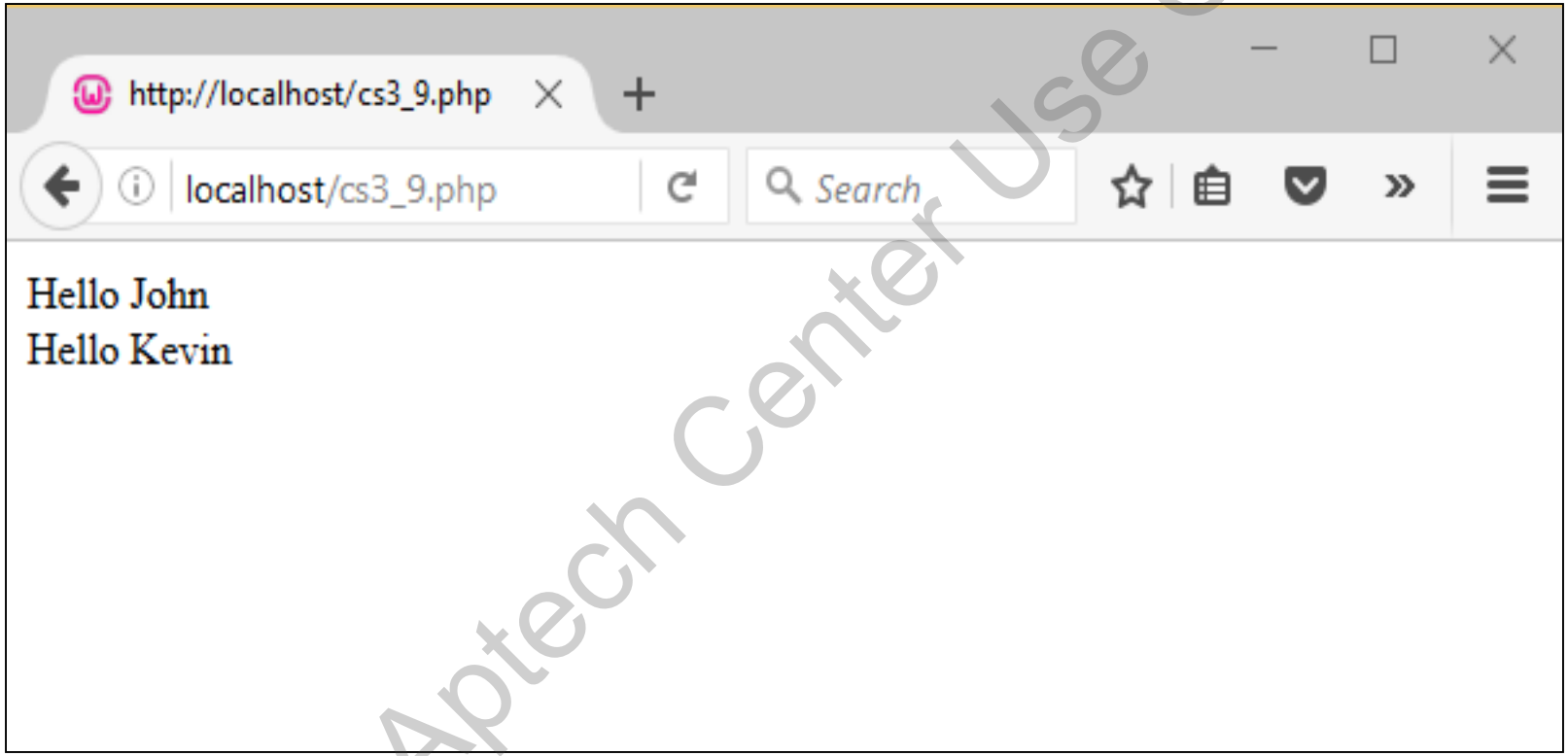
- Demonstrating the use of closure ()

Snippet

```
<?php

 class Greetings {

 private $word = "Hello";
 }

 $closure = function($whom) {

 echo "$this->word $whom\n";
 };

 $obj = new Greetings();

 $closure->call($obj, 'John');
 $closure->call($obj, 'Kevin');

?>
```

Closure call () Method

- Displays the following output



Generator Return Expressions

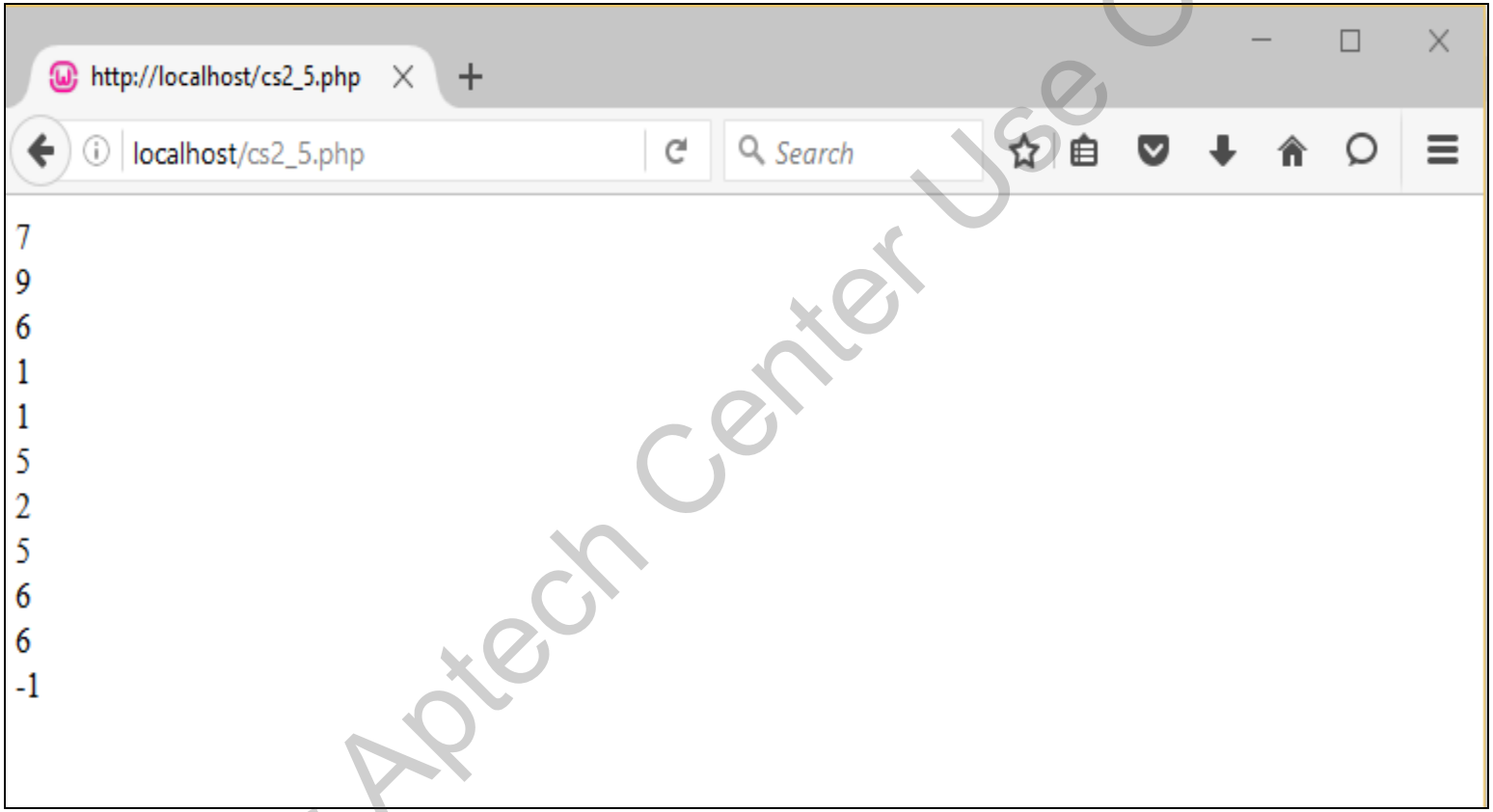
- A statement used within a generator to return the final expression.
- The value can be retrieved using the `getReturn()` method.
- The method should be used only after the generator has finished yielding results.

- Demonstrating the use of `getReturn()` expression

Snippet

```
<?php
srand();
function random_numbers($k) {
 for ($i=0; $i<$k; $i++) {

 $r = rand(1, 10);

 yield $r;
 }

 return -1;
 }
 $rns = random_numbers(10);

 foreach ($rns as $r) {
 echo "$r";
 echo '</br>';
 }
 echo $rns->getReturn() . PHP_EOL;
 ?>
```

- Displaying the output:



Generator Delegation via yield from 1-2

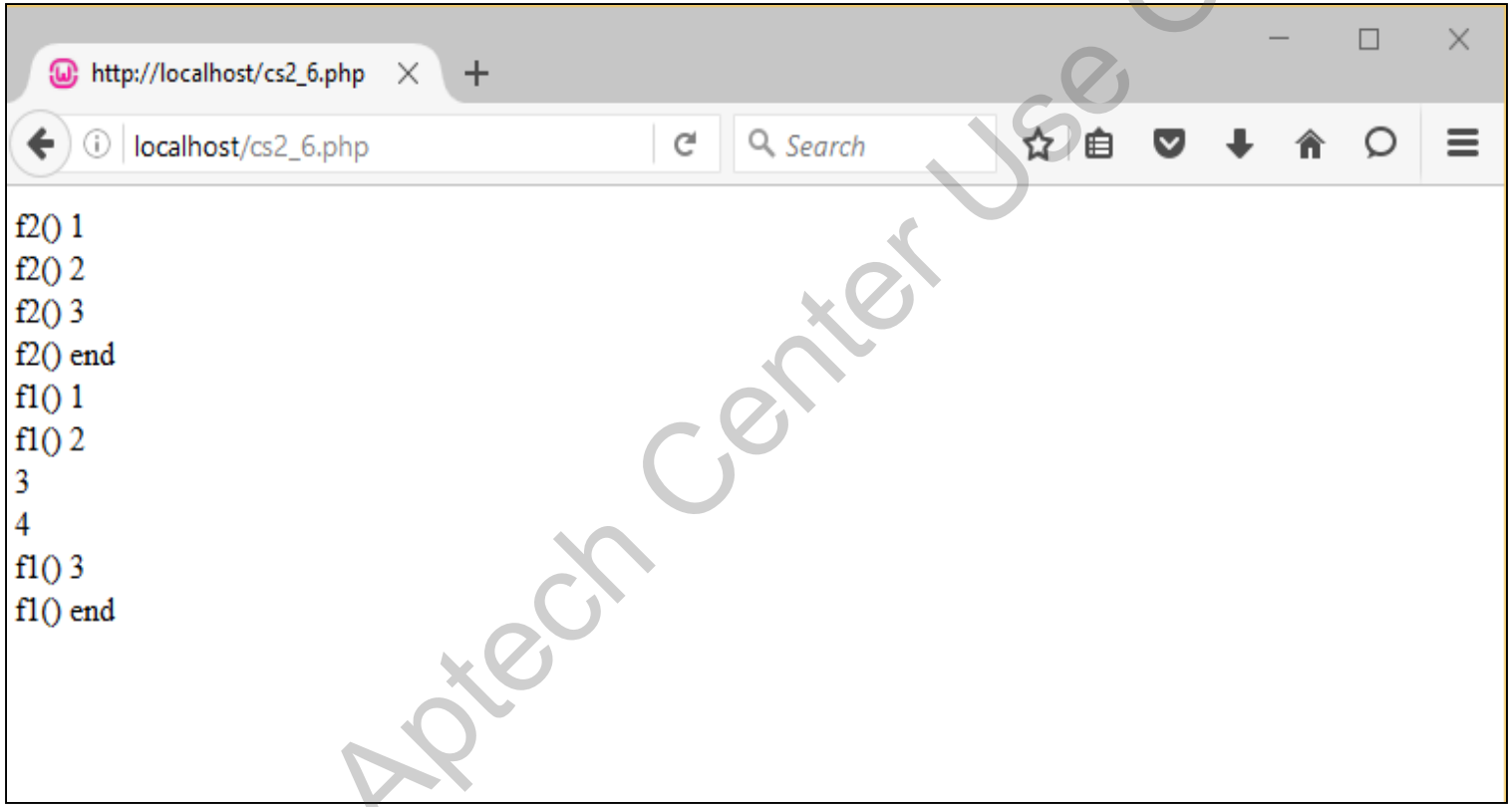
- Demonstrating the use of `yield from` keyword

Snippet

```
<?php
 function f1() {
 yield from f2();
 yield "f1() 1";
 yield "f1() 2";
 yield from [3, 4];
 yield "f1() 3";
 yield "f1() end";
 }
 function f2() {
 yield "f2() 1";
 yield "f2() 2";
 yield "f2() 3";
 yield "f2() end";
 }
 $f = f1();
 foreach ($f as $val) {
 echo "$val\n";
 }
?>
```

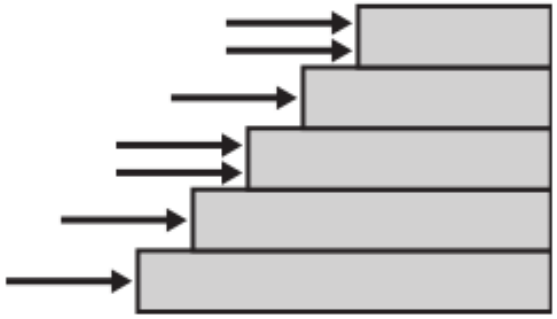
Generator Delegation via yield from 2-2

- Displaying the following output:



`levels` Parameter of `dirname()`

- The `dirname()` functions returns the parent's directory path.
- The `dirname()` now accepts a second parameter that specify the number of levels a parent directory can go up.
- The `levels` parameter tells how many parent directories can go up.



levels Parameter of dirname()

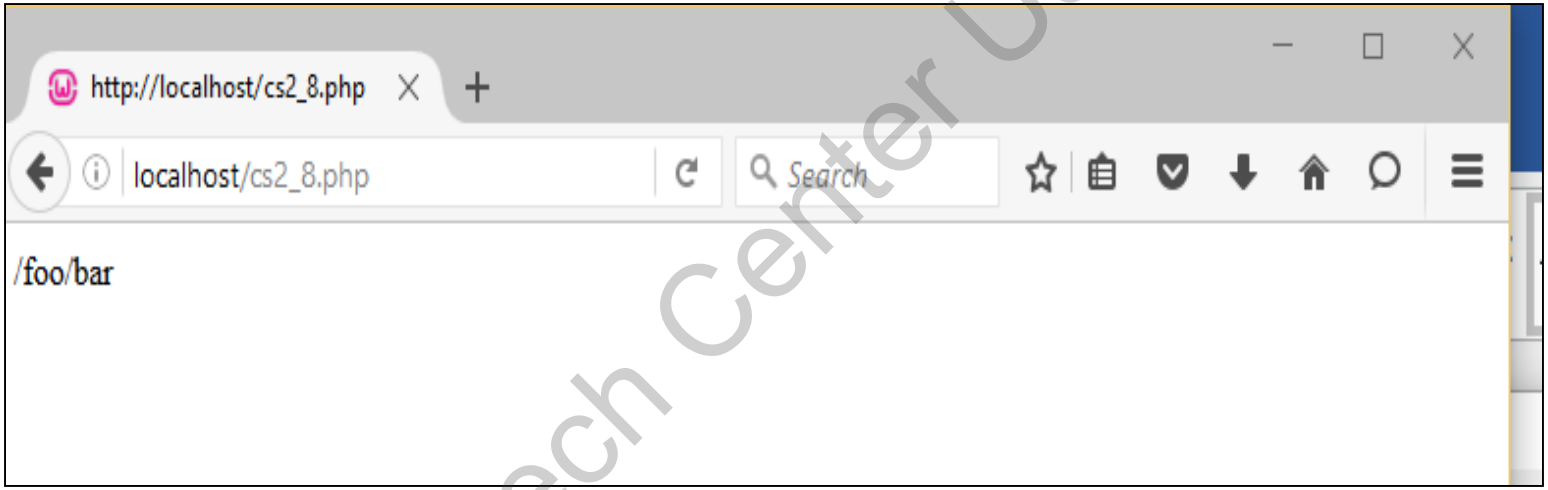
- Demonstrating the use of `dirname()` function

Snippet

```
<?php
$path = '/foo/bar/bat/baz';
echo dirname($path, 2);
?>
```

levels Parameter of dirname()

- Displays the following output:



Summary

- The new intdiv() performs integer division and is considered the inverse of the modulo operator %.
- Spaceship operator also known as combined comparison operator, returns three distinct values, -1, 0, or +1.
- The null coalesce operator ?? returns the left operand if it is not null; otherwise, it returns the right operand.

Summary

- Group ‘use’ declarations allow to deduplicate common prefixes in 'use' statements and specify unique parts within a block {}.
- Generator->getReturn() method allows you to retrieve value returned by any return statement inside the generator.
- With addition of $this, closure has gained two methods, the instance method Closure->bindTo() and the static method Closure::bind().
- The dirname() function can now accept a second parameter to set how many levels up it will go.