

Generator Delegation and Throwable Interface

Session 26



Objectives

- ◆ *Explain generators and generator return expressions.*
- ◆ *Describe Generator Delegation.*
- ◆ *Explain exception handling and changes in exceptions in PHP 7.*
- ◆ *Explain the use of `E_ERROR` and `E_RECOVERABLE_ERROR`.*
- ◆ *Explain the `Throwable` Interface.*

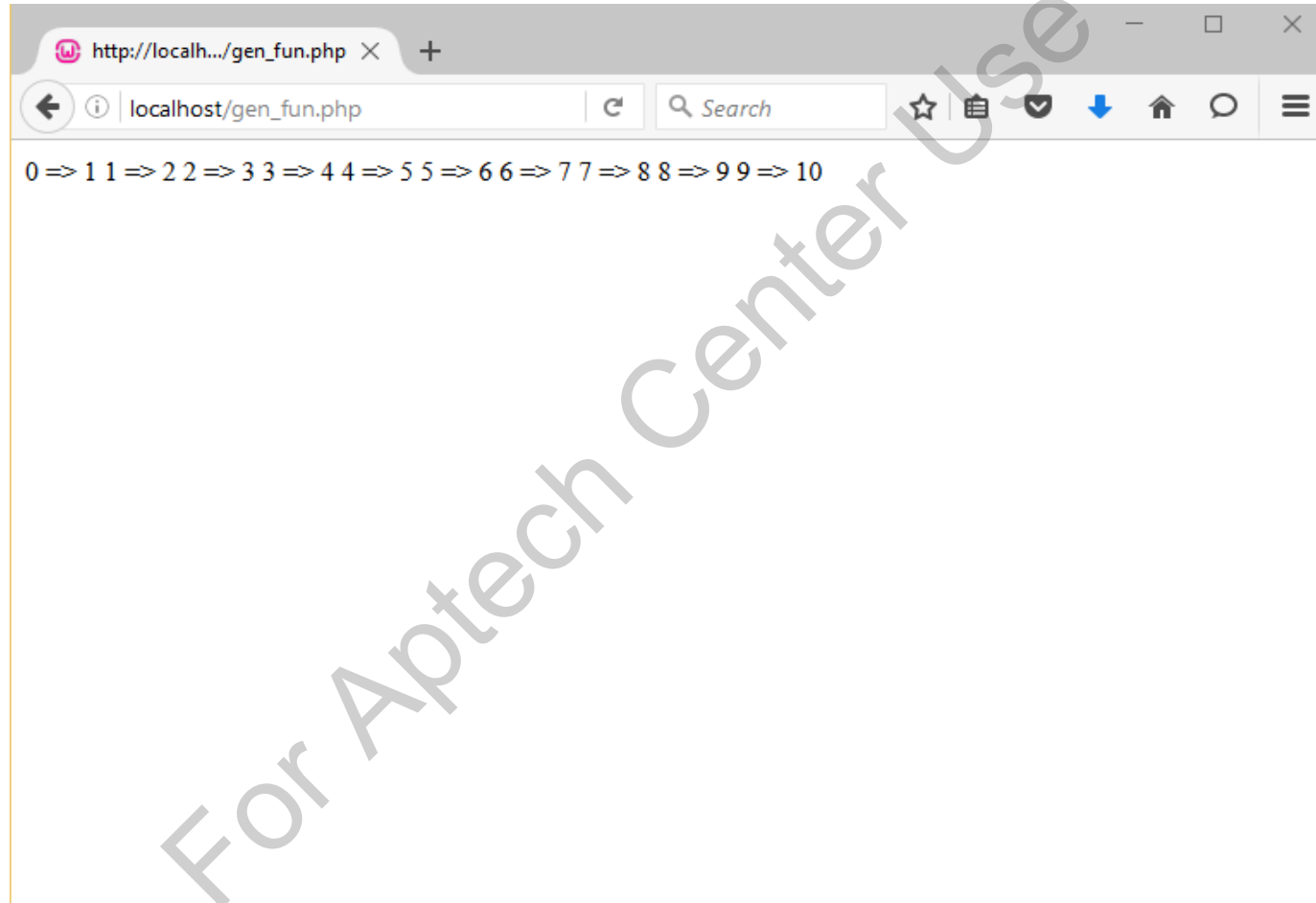
- ◆ Returns/yields values on demand
- ◆ Return multiple values from a function
- ◆ Use the `yield` keyword to return data

- ◆ Demonstrating the use of a generator that prints a range of values

Snippet

```
<?php
function print_range_of_values($min_val, $max_val)
{
    for ($i= $min_val; $i <= $max_val; $i++)
    {
        yield $i;
    }
}
foreach(print_range_of_values(1,10) as $key=>$value)
{
    echo "$key => $value", PHP_EOL;
}
?>
```

- ◆ Displays the following output:



◆ Demonstrating the use of range () function

Snippet

```
<?php
function Sim_range($start_num, $max_limit,
    $increment_by = 1) {
    if ($start_num < $max_limit) {
        if ($increment_by <= 0) {
            throw new LogicException('Step must be +ve');
        }
        for ($x = $start_num; $x <= $max_limit; $x +=
            $increment_by) {
                yield $x;
            }
        }
    }
}
```

```
else {
    if ($increment_by >= 0) {
        throw new LogicException('Step must be -ve');
    }
    for ($x = $start_num; $x >= $max_limit; $x +=
$increment_by) {
        yield $x;
    }
}

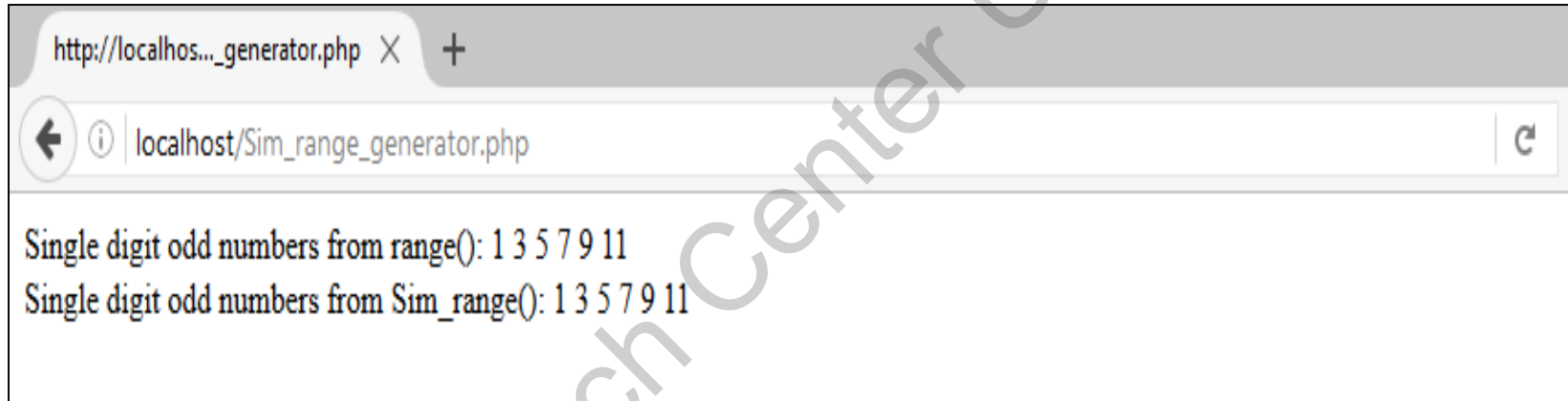
}

echo 'Single digit odd numbers from range():<br> ';
foreach (range(1, 11, 2) as $num) {
    echo "$num ";
}
echo "<br>";

echo 'Single digit odd numbers from Sim_range():
<br>';
foreach (Sim_range(1, 11, 2) as $num) {
    echo "$num ";}

?>
```

- ◆ Displays the following output:



- ◆ Enables to yield values from other generator, arrays, and traversable objects

Syntax

```
yield from <thing_to_iterate>
```

- ◆ Demonstrating the use of generator delegation

Snippet

```
<?php
function DisplayHello() {
    yield "Hello";
    yield " ";
    yield "World!";
    yield from DisplayGoodbye();
}

function DisplayGoodbye() {
    yield "Goodbye";
    yield " ";
    yield "Mars!";
}
```

```
$gen = DisplayHello();  
foreach ($gen as $value) {  
    echo $value;  
    echo '</br>';  
}  
?>
```

- ◆ Displays the following output:



- ◆ Are a mechanism for error handling
- ◆ An exception handling codes include:

Code	Description
try block	If exceptions do not occur, then the code executes and exists successfully. OR If exceptions occur, then the exception is caught by the catch block and action is taken to handle to exception.
catch block	If exceptions occur, then action statements are included in the block to handle the exception.

Code	Description
throw	A method to manually trigger an exception. However, each throw should be associated with an corresponding catch.
finally	A block usually executed after the <code>try</code> and <code>catch</code> blocks. The code within this block is executed irrespective of whether an exception has been thrown or before normal execution continues.

- ◆ Demonstrating the use of try-catch-finally statements

Snippet

```
<?php
function computeDiv($num) {
    if (!$num) {
        throw new Exception('Division by zero.');
```

```
try {
    echo computeDiv(0). '<br><br>';
    echo 'Result of division';
} catch (Exception $e) {
    echo 'Caught exception:', $e->
    getMessage(), '<br><br>';
} finally {
    echo "Now in finally block.<br><br>";
}

// Continue execution
echo "Program execution
continues<br><br>";
?>
```


- ◆ Displays the following output:

```
Caught exception: Division by zero.  
Now in finally block.  
Program execution continues
```

A fatal runtime error.

A non recoverable error.

The script or program terminates prematurely when this error occurs.

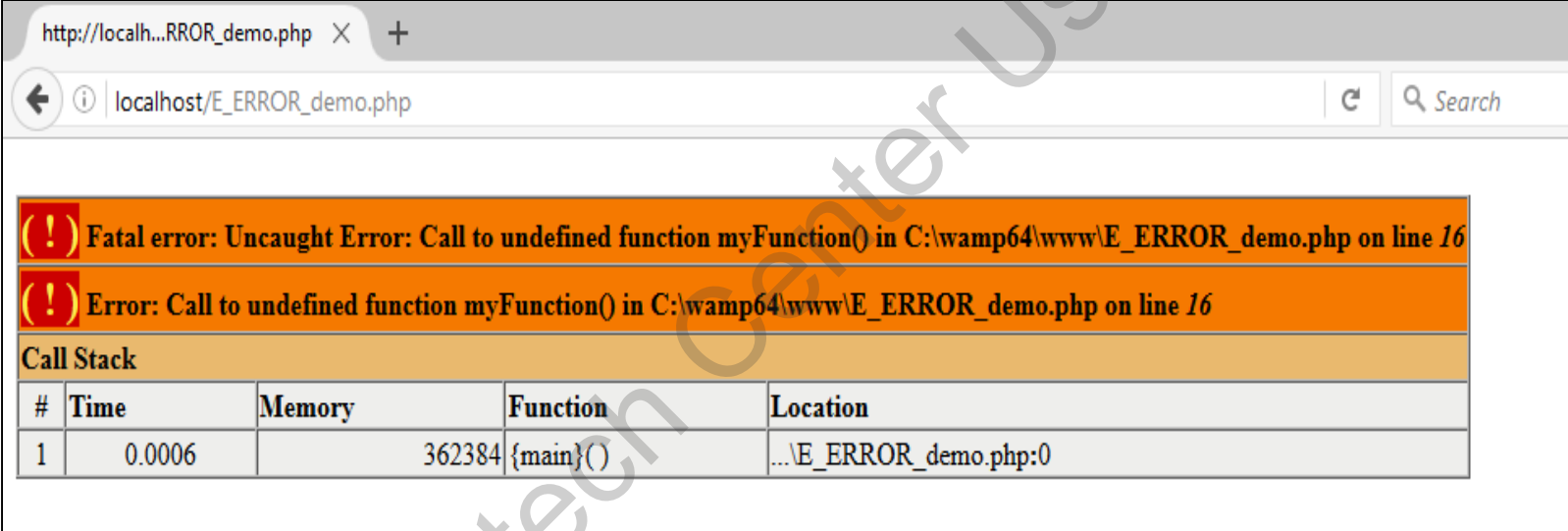
The error enables backward compatibility.

◆ Demonstrating the use of E_ERROR error

Snippet

```
<?php
    error_reporting(E_ERROR);
    function handle_error($err_no,
    $err_str,$err_file,$err_line) {
        echo "<b>Error:</b> [$err_no] $err_str -
    $err_file:$err_line";
        echo "<br>";
        echo "Terminating PHP Script";
        die();
    }
    //set error handler
    set_error_handler("handle_error", E_ERROR);
    //trigger error
    my_function();
?>
```

- ◆ Displays the following output:



The screenshot shows a web browser window with the address bar displaying `http://localhost/E_ERROR_demo.php`. The page content displays two error messages in orange boxes:

- Fatal error:** Uncaught Error: Call to undefined function myFunction() in C:\wamp64\www\E_ERROR_demo.php on line 16
- Error:** Call to undefined function myFunction() in C:\wamp64\www\E_ERROR_demo.php on line 16

Below the error messages is a section titled "Call Stack" containing a table with the following data:

#	Time	Memory	Function	Location
1	0.0006	362384	{main}()	...E_ERROR_demo.php:0



◆ Demonstrating the use of E_RECOVERABLE_ERROR

Snippet

```
<?php
error_reporting(E_RECOVERABLE_ERROR);
function handle_error($errno,
    $errstr,$error_file,$error_line) {
    echo "<b>Error:</b> [$errno] $errstr -
    $error_file:$error_line";
    echo "<br>";
    echo "Terminating PHP Script";
    die();
}

//set error handler
set_error_handler("handleError", E_RECOVERABLE_ERROR);

//trigger error
myFunction();
?>
```

- ◆ Displays the following output:



- ◆ Benefits of handling fatal errors as exceptions are:

Enables to handle fatal errors using the try-catch block

Helps handle catchable errors easily

Helps easy debugging

- ◆ Exception and Error classes implement the `Throwable` interface.
- ◆ The root of the hierarchy was `\Exception` in the previous versions of PHP.

◆ Demonstrating the hierarchy of classes and interfaces.

➤ Interface Throwable

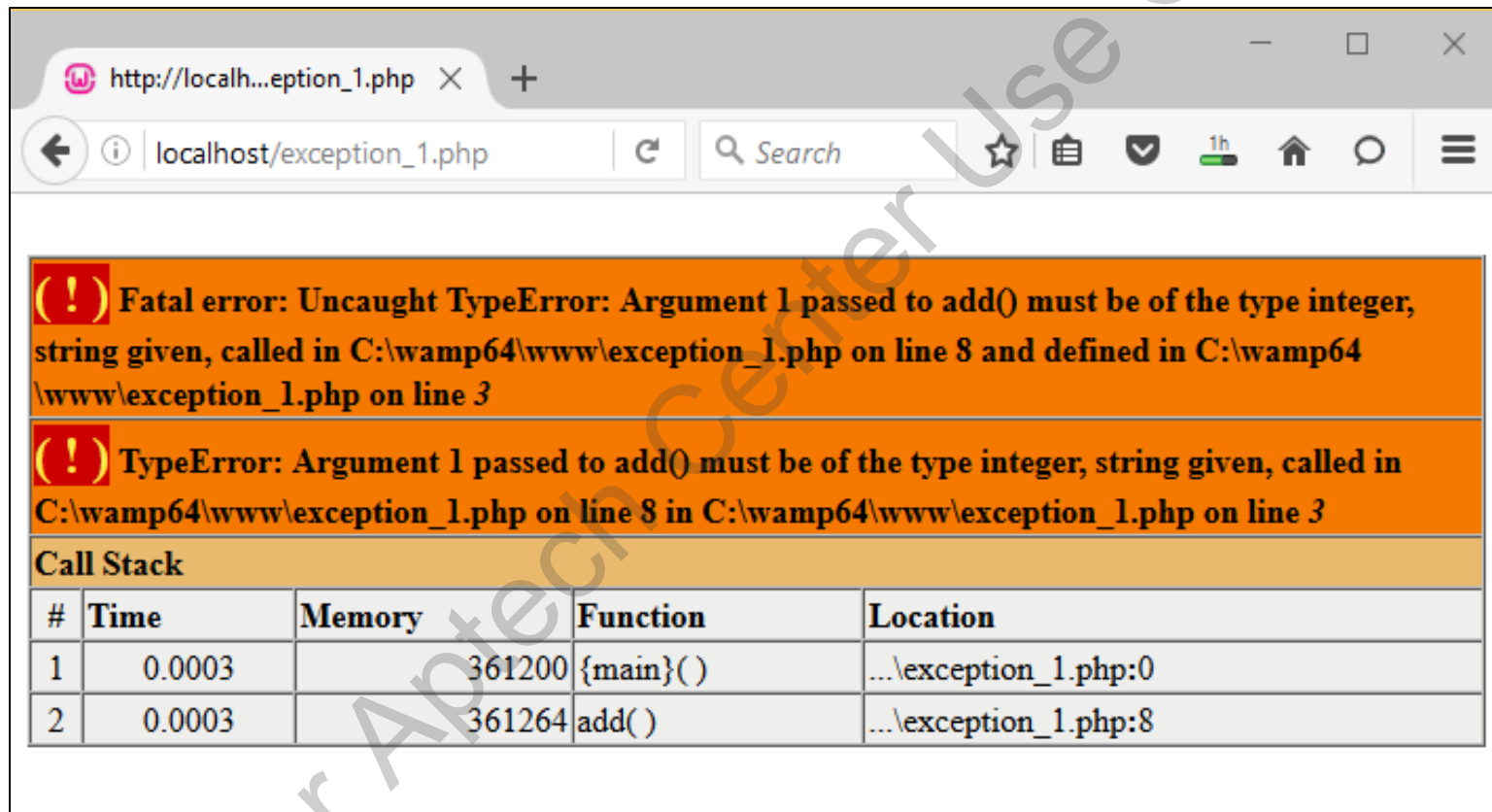
- Exception **implements** Throwable
 - RuntimeException (**extends** Exception)
 - IllegalArgumentException (**extends** RuntimeException)
 - ...
- Error **implements** Throwable (Replaces EngineException)
 - TypeError **extends** Error
 - ParseError **extends** Error
 - ...

- ◆ Demonstrating a fatal error where exception is not caught and error is thrown.

Snippet

```
<?php
function add(int $left_op, int $right_op) {
    return $left_op + $right_op;
}
try {
    echo add('two', 'three');
} catch (Exception $e) {
    // Handle or log exception.
    echo "Error occurred in the program due to
parameter datatype mismatch";
}
?>
```

- ◆ Displays the following output:



(!) Fatal error: Uncaught TypeError: Argument 1 passed to add() must be of the type integer, string given, called in C:\wamp64\www\exception_1.php on line 8 and defined in C:\wamp64\www\exception_1.php on line 3

(!) TypeError: Argument 1 passed to add() must be of the type integer, string given, called in C:\wamp64\www\exception_1.php on line 8 in C:\wamp64\www\exception_1.php on line 3

Call Stack

#	Time	Memory	Function	Location
1	0.0003	361200	{main}()	...\exception_1.php:0
2	0.0003	361264	add()	...\exception_1.php:8

◆ Demonstrating to catch an Exception and Error

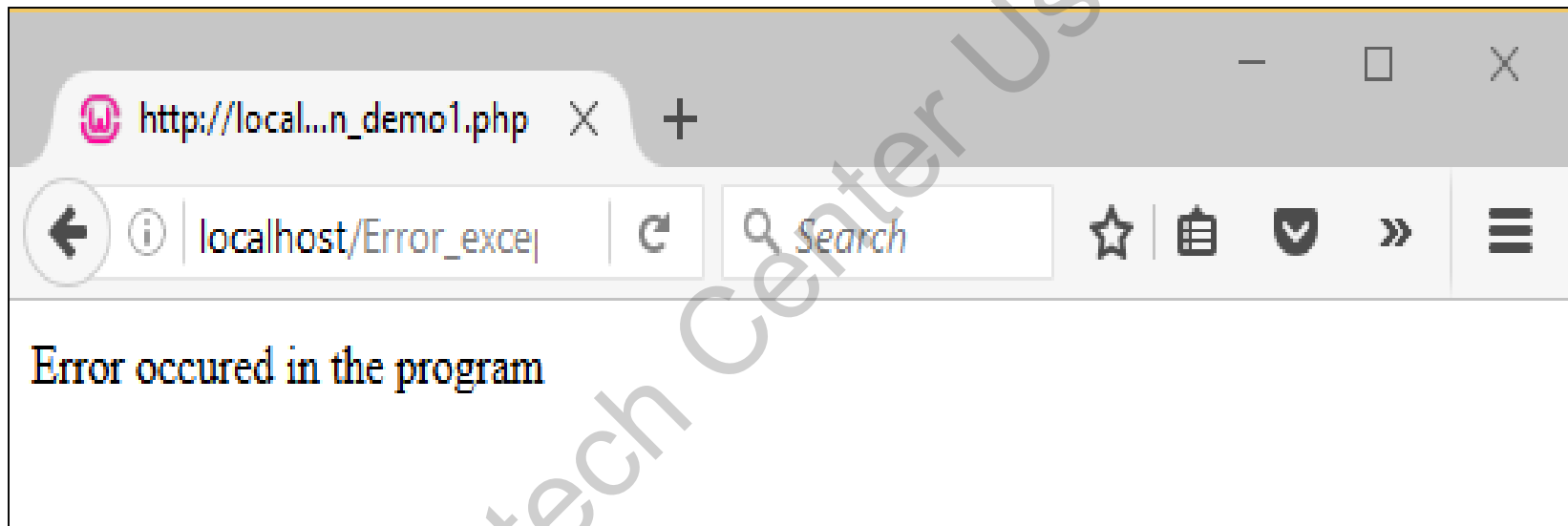
Snippet

```
<?php

function add(int $left_op, int $right_op) {
    return $left_op + $right_op;
}

try {
    echo add('two', 'three');
} catch (Exception $e) {
    // Handle or log exception.
    echo "Exception occurred in the program";
}
catch (Error $e) { // Log error and end gracefully
    echo "Error occurred in the program";
}
?>
```

- ◆ Displays the following output:



- ◆ Generator return expressions is a new feature in PHP 7 that allows you to 'return' a value on completion of a generator using `return` keyword and `Generator::getReturn()` method.
- ◆ Generator Delegation promotes reusability and cleaner code.
- ◆ Generator Delegation allows getting/yielding values from other generators using `yield from` clause.

- ◆ PHP supports error and exception handling mechanisms that allow programs to continue or exit gracefully instead of exiting abruptly.
- ◆ In PHP 7, all errors including fatal errors are engine exceptions.
- ◆ `Exception` and `Error` classes implement the new `Throwable` interface, which is newly introduced in PHP 7.