Giovedì 18 Dicembre ore 18:30
@ Toolbox Coworking Torino

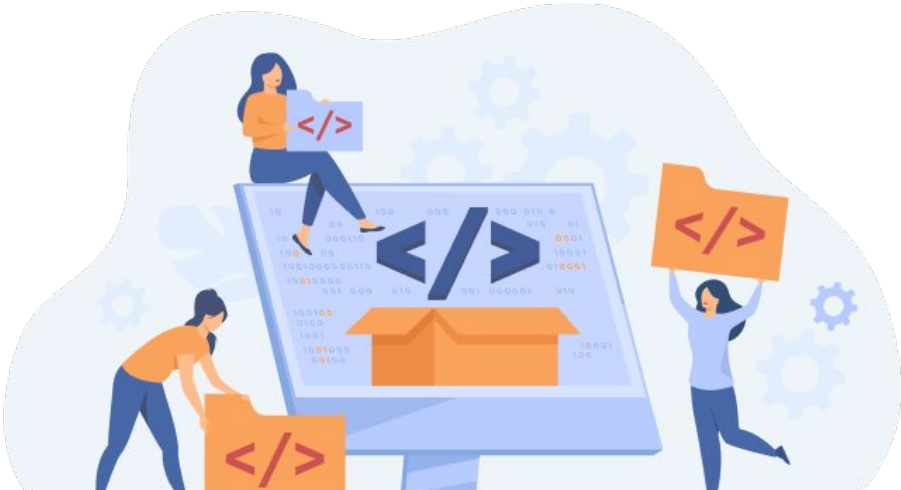
# What's new in php 8.5 ? Let's discuss together

Daniele Barbaro

Enrico Zimuel

GRUSP

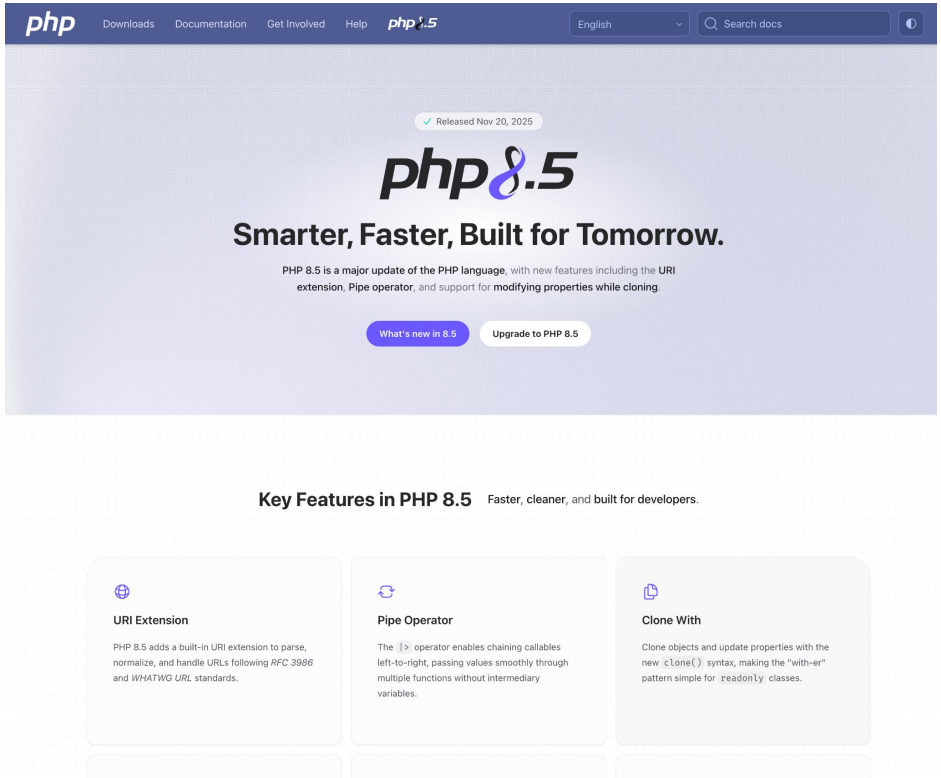
TOOLBOX COWORKING



# Programmer User Group Torino

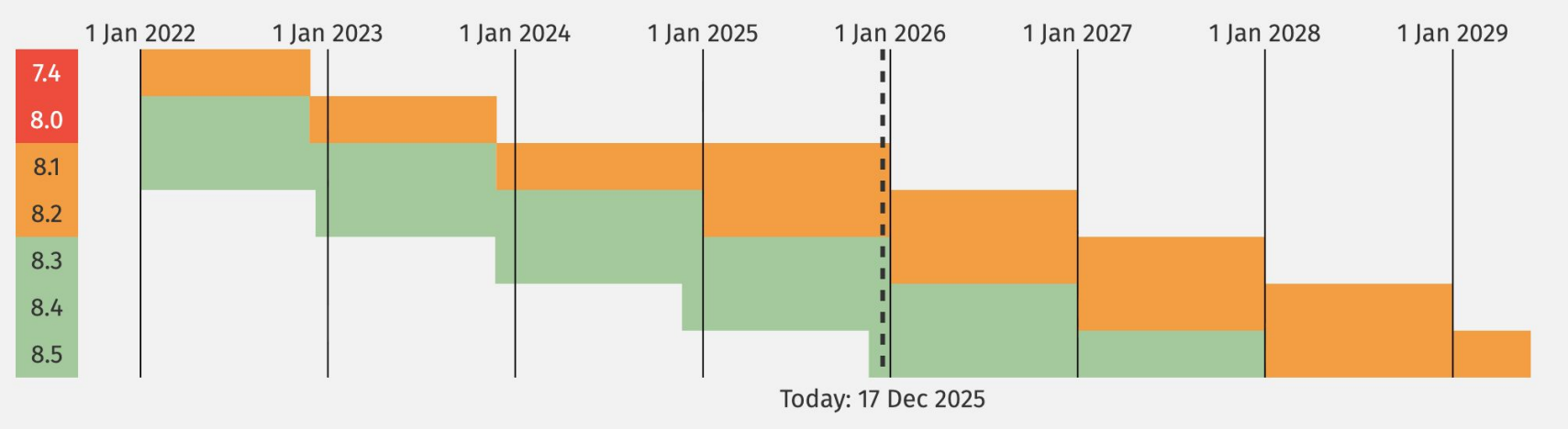
- Il Programmer User Group (PUG) di Torino è un gruppo di appassionati di programmazione per la condivisione di conoscenze e codice
- E' attivo dal 2011 sul linguaggio PHP, recentemente ha allargato i suoi orizzonti verso tutti i linguaggi di programmazione
- Organizzato conferenze nazionali come il PHP.TO.START nel 2011 e lo Zend Framework Day nel 2014
- Ha al suo attivo più di 800 membri su meetup.com

# PHP 8.5 - un nuovo sito!




Votazione - https://github.com/php/web-php/issues/1563

# Versioni supportate del PHP



# Repository con esempi

https://github.com/danielebarbaro/pug-php8.5

# Pipe operator

- Il pipe operator rende il codice lineare, leggibile e **step-by-step** (vedi RFC)
- Utilizza il simbolo |> (pipe) per passare il risultato allo step successivo:
  - ***mixed* |> *callable***

```php
$input = " Hello World  ";
// Method 1: Nested function calls
$result = str_shuffle(strtoupper(trim($input)));
printf("Result: %s\n", $result);

// Method 2: Using temporary variables
$result = $input;
$result = trim($result);
$result = strtoupper($result);
$result = str_shuffle($result);
printf("Result: %s\n", $result);
```

```php
// Method 3: Using the pipe operator
$result = $input
    |> trim(...)
    |> strtoupper(...)
    |> str_shuffle(...);
printf("Result: %s\n", $result);
```

# Pipe operator: proprietà

- Il pipe operator è *left associative*

```
// sono equivalenti
$result = 5 + 2 |>  someFunc(...);
$result = (5 + 2) |>  someFunc(...);
```

- Il pipe operator è tradotto in fase di compilazione, non ci sono overhead
- L'aggiunta di *closure* addizionali può comportare un (minimo) overhead

# Pipe: esempi con closure

```php
$numbers = [1, 2, 3, 4, 5];
$result = array_sum(
    array_filter(
        array_map(fn($n) => $n * 2, $numbers),
        fn($n) => $n > 5
    )
);
```

```php
$numbers = [1, 2, 3, 4, 5];
$result = $numbers
    |> (fn($arr) => array_map(fn($n) => $n * 2, $arr))
    |> (fn($arr) => array_filter($arr, fn($n) => $n > 5))
    |> array_sum(...);
```

# array_first - array_last

- **array_first()** e **array_last()** restituiscono il primo e l'ultimo valore di un array (vedi RFC)
- funzionano anche con array associativi e rendono inutile i vecchi **reset()** e **end()**.

```php
$names = [
    'primo' => 'Dani', 'secondo' => 'Sole', 'terzo' => 'Ceci',
];

$first = array_first($names);
$last = array_last($names);
```

```php
$first = array_values($array)[0];
$last  = array_values($array)[count($array) - 1];

// Alternativa vintage
$first = reset($array);
$last  = end($array);
```

# Attributes on constants

- E’ possibile applicare attributi su costanti (vedi RFC)
- Es. è possibile applicare #[\Deprecated]

```
class ConstantsExample {
  #[\Deprecated("use NEW_CONST instead")]
  public const OLD__CONST = 'foo';
  public const NEW__CONST = 'bar';
}
printf("OLD_CONST: %s\n", ConstantsExample::OLD__CONST);
```

```
PHP Deprecated:  Constant ConstantsExample::OLD__CONST is deprecated, use NEW_CONST
instead in /…/pug-php8.5/examples/attribute-constant.php on line 10
OLD_CONST: foo
```

# Final property promotion

- contratti più forti (vedi RFC)
- oggetti più sicuri

```php
class Product
{
    public function __construct(
        public final string $sku,      // non sovrascrivibile
        public final float $price,     // regola di business
        private string $name           // privata ma modificabile
    ) {}

    public function rename(string $name): void
    {
        $this->name = $name;
    }
}
class DiscountedProduct extends Product
{
    // ❌ ERRORE
    public float $price;
}
```

***final*** → *blocca l'override nelle classi figlie* | ***readonly*** → *blocca la modifica dopo la costruzione*

# php --ini=diff

- mostra solo le differenze rispetto ai default
- debug veloce

```
Non-default INI settings:

html_errors: "1" -> "0"
implicit_flush: "0" -> "1"
max_execution_time: "30" -> "0"
max_memory_limit: "256M" -> "1G"
memory_limit128M
```

# FILTER_THROW_ON_FAILURE

- filter_var() può generare un’eccezione in caso di fallimento (vedi RFC)

```php
$email = 'email-non-valida';
try {
    $res = filter_var(
        $email,
        FILTER_VALIDATE_EMAIL,
        FILTER_THROW_ON_FAILURE
    );

    echo "Email valida: $res\n";
} catch (\Filter\FilterFailedException $e) {
    echo "Errore: " . $e->getMessage() . "\n\n";
}
```

# PHP_BUILD_DATE constant

- Mostra la data di build della versione PHP
- Utile per diagnostica e tracciabilità
- Migliora determinazione esatta della build in esecuzione

```php
echo 'PHP version: '  . PHP_VERSION  . PHP_EOL;
echo 'Build date: '   . PHP_BUILD_DATE  . PHP_EOL;
// PHP version: 8.5.0
// Build date: Dec  8 2025 22:42:23
```

# NoDiscard: return deve essere utilizzato

Il valore di ritorno di questa funzione **non deve essere ignorato** (vedi RFC).

❌ non è un'eccezione

❌ non ferma l'esecuzione

❌ non cambia la logica di runtime

```php
#[\NoDiscard("Il risultato deve essere salvato")]
function calculateHash(string $data): string {
    return hash('sha256', $data);
}
```

# IntlListFormatter

- International list: virgole, congiunzione finale (e), spaziatura corretta

```
$list = new IntlListFormatter('it_IT');
echo $list->format(['mele', 'pere', 'banane']);
// mele, pere e banane
```

- Formattazione dei numeri decimali SHORT e LONG

```
$total = new NumberFormatter('it_IT', NumberFormatter::DECIMAL_COMPACT_SHORT);
printf("%s\n", $total->format(12345));
// 12.345
$total = new NumberFormatter('it_IT', NumberFormatter::DECIMAL_COMPACT_LONG);
printf("%s\n", $total->format(12345));
// 12 mila
```

# Clone with (vedi RFC)

```php
readonly class Book
{
    public function __construct(
        public string $title,
        public string $description,
        public string $author = 'Sconosciuto',
    ) {}
    public function withTitle(string $title): self
    {
        return clone ($this, ['title' => $title]);
    }
    public function withAuthor(string $author): self
    {
        return clone ($this, ['author' => $author]);
    }
}
```

```php
$libro = new Book(
    title: 'PHP 8.5 Guide',
    description: 'Una guida completa a PHP 8.5',
    author: 'Mario Rossi'
);

echo "Originale: {$libro->title} - {$libro->author}\n";

$copia = $libro->withTitle('PHP 8.5 Advanced Guide');
echo "Copia: {$copia->title} - {$copia->author}\n";

$copia2 = $copia->withAuthor('Luigi Verdi');
echo "Copia2: {$copia2->title} - {$copia2->author}\n";
```

**QUIZ:** qual'è il titolo e l'autore di $copia2?

# Fatal error backtraces (vedi RFC)

```php
ini_set('memory_limit', '2M');

function my_heavy_function(): string {
    return str_repeat('A', 1024 * 1024 * 5);
}

my_heavy_function();
// PHP Fatal error:  Allowed memory size of 2097152 bytes exhausted
// (tried to allocate 5242912 bytes) in test.php on line 12
// Stack trace:
// #0 test.php(12): str_repeat()
// #1 test.php15): my_heavy_function()
// #2 {main}
```

# Persistent cURL Share Handles

```
$sh = curl_share_init_persistent([
   CURL_LOCK_DATA_DNS,
   CURL_LOCK_DATA_CONNECT,
]);
$ch = curl_init('https://php.net/');
curl_setopt($ch, CURLOPT_SHARE, $sh);
curl_setopt($ch, CURLOPT_RETURNTRANSFER,  1);

// This may now reuse the connection from an earlier SAPI request
$response = curl_exec($ch);
$end = microtime(true);
printf("Duration: %.3f seconds\n", $end - $start);
var_dump($response);
```

# Handler introspection

- Nuova funzione **get_error_handler(): ?callable**

```php
$handler = function (int $errno, string $errstr, ?string $errfile, ?int $errline) {
    echo "Error: " . $errstr . "\n";
};
set_error_handler($handler);
var_dump(get_error_handler() === $handler); // bool(true)
```

- Nuova funzione **get_exception_handler(): ?callable**

```php
$handler = function (Throwable $ex) {
    echo "Exception: " . $ex::class . ": " . $ex->getMessage() . "\n";
};
set_exception_handler($handler);
var_dump(get_exception_handler() === $handler); // bool(true)
```

# URI Extension

- Aggiunta una nuova classe **Uri\Rfc3986\Uri** per l'estensione URI (vedi RFC)
- Utilizzo delle librerie uriparser e lexbor (articolo interessante da leggere)

```php
use Uri\Rfc3986\Uri;

# PHP < 8.5
$components = parse_url('https://php.net/releases/8.4/en.php');
var_dump($components['host']);
// string(7) "php.net"

# URI parsing con PHP 8.5
$uri = new Uri('https://php.net/releases/8.5/en.php');
var_dump($uri->getHost());
// string(7) "php.net"
```

# Closures in constant expressions (vedi RFC)

```
function my_array_filter(
    array $array,
    Closure $callback = static function ($item) {return !empty($item); },
) {
    $result = [];
    foreach ($array as $item) {
        if ($callback($item)) {
            $result[] = $item;
        }
    }
    return $result;
}
var_dump(my_array_filter([
    0, 1, 2,
    '', 'foo', 'bar',
]));
```

# Closures in constant expressions (2)

```php
class MyObject
{
   public function __construct(
     private Closure $callback
   ) {}
}

const Foo = new MyObject(
  static function () {
     return 'foo';
  }
);
```

```php
function foo(
   string $input,
   array $callbacks = [
       static function ($value) {
           return \strtoupper($value);
       },
       static function ($value) {
           return \preg_replace('/[^A-Z]/', '', $value);
       },
   ]
) {
   foreach ($callbacks as $callback) {
       $input = $callback($input);
   }
   return $input;
}
var_dump(foo('Hello, World!')); // string(10) "HELLOWORLD"
```

# grapheme_levenshtein (vedi RFC)

- Distanza di Levenshtein: metrica che misura quanto due stringhe siano diverse, contando il numero minimo di modifiche su un singolo carattere (inserimenti, eliminazioni o sostituzioni) necessario per trasformare una stringa nell'altra
- Esempio:
  - levenshtein("casa", "caso") = 1
  - levenshtein("casa", "casa") = 0
  - levenshtein("casa", "cera") = 2
- Introdotta una nuova funzione **grapheme_levenshtein()** che supporta UTF-8:
  - levenshtein("göthe", "gothe") = 2 che è errato!
  - grapheme_levenshtein("göthe", "gothe") = 1

# grapheme_levenshtein (2)

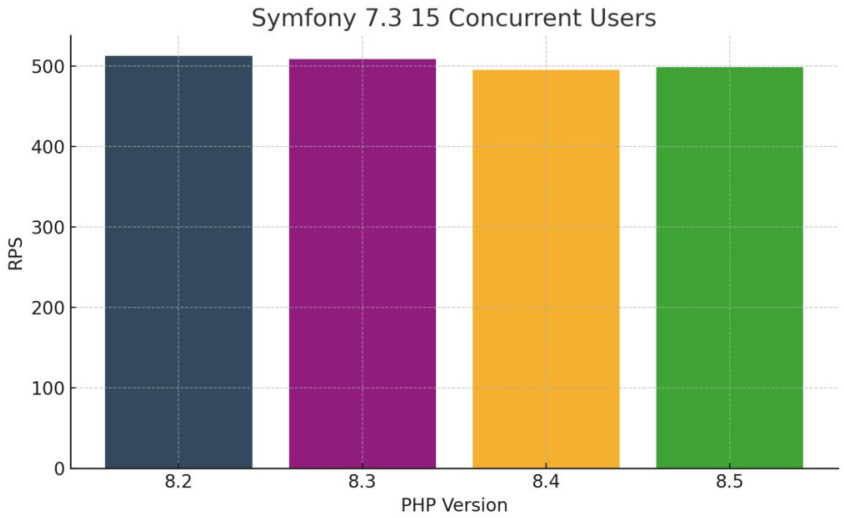
- **grapheme_levenshtein()** lavora su grafemi, ossia la più piccola unità distintiva della scrittura (una lettera o combinazione di lettere) che rappresenta un fonema o una parte di esso in una lingua
- Utile per caratteri UNICODE

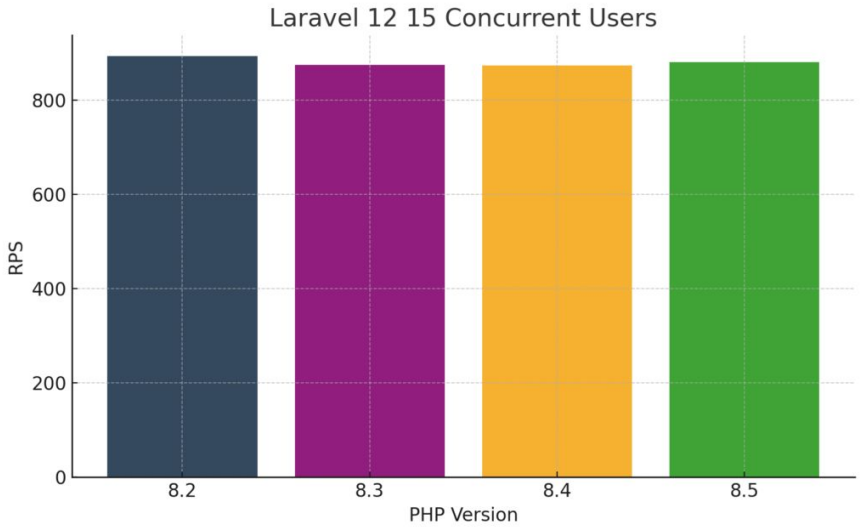
```
# Altro esempio con caratteri Unicode combinati
$string1 = "\u{00e9}"; // 'é' come singolo carattere
$string2 = "\u{0065}\u{0301}"; // 'e' + accento acuto come due caratteri
$lev = levenshtein($string1, $string2);
$graphemeLev = grapheme_levenshtein($string1, $string2);

printf("Levenshtein tra '%s' e '%s': %d\n" , $string1, $string2, $lev);
printf("grapheme_levenshtein: %d \n" , $string1, $string2, $graphemeLev);
```

# Performance

- Le performance sono praticamente le stesse da PHP 8.2
- Fonte benchmark: [tideways.com]



# Repo Info

Repo:

- https://github.com/danielebarbaro/pug-php8.5

Fonti:

- https://www.php.net/releases/8.5/en.php
- https://stitcher.io/blog/new-in-php-85
- https://benjamincrozat.com/php-85
- https://php.watch/versions/8.5

# Grazie!

Per informazioni:
https://www.meetup.com/it-it/pugto-php-user-group-torino/