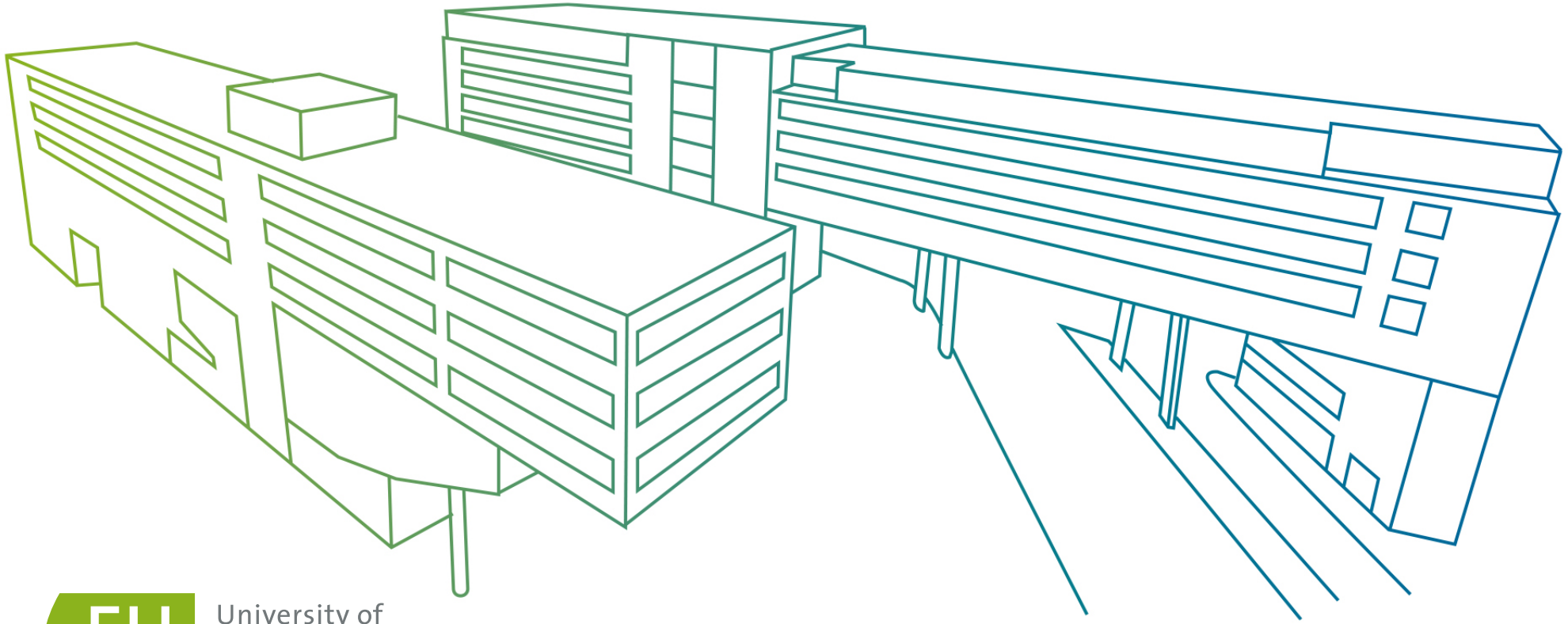


FH University of Applied Sciences
TECHNIKUM WIEN

Web Technologies

PHP Advanced

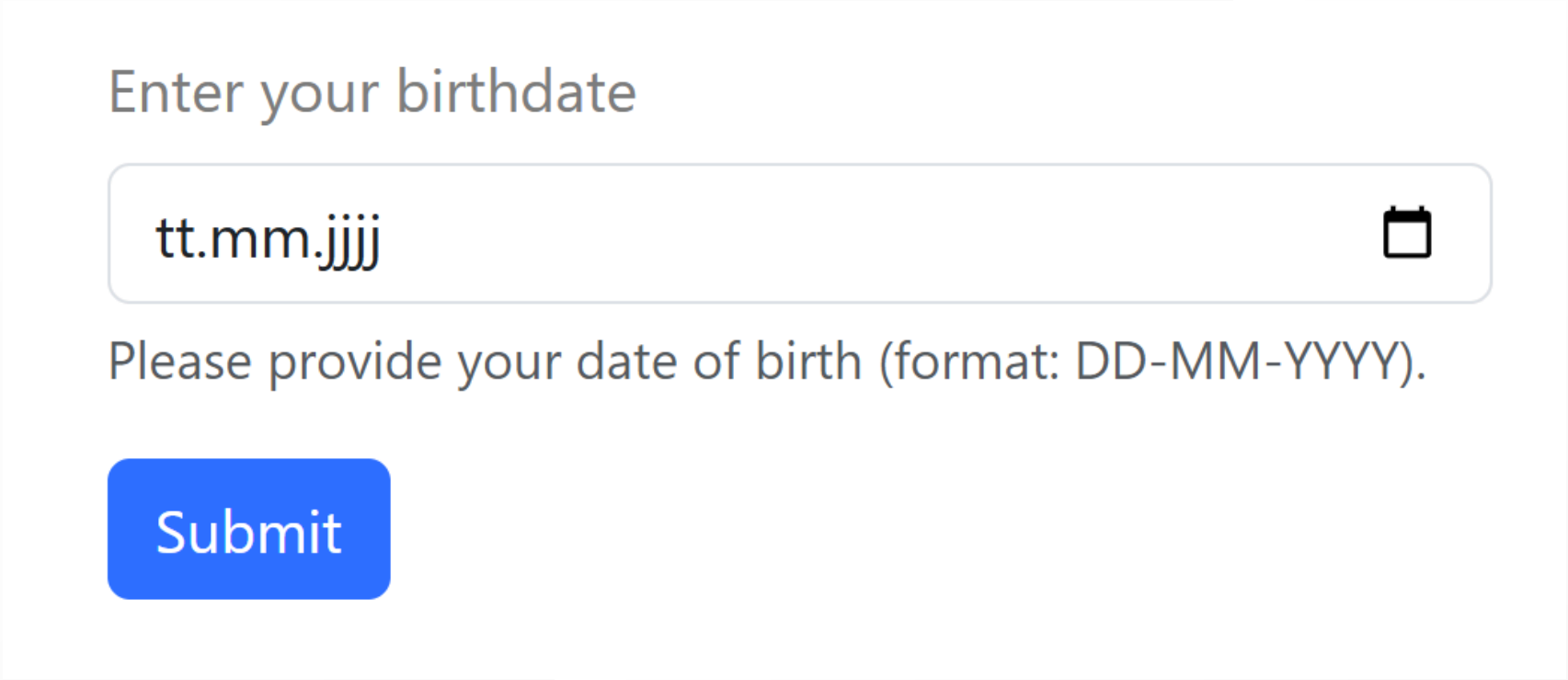
Current situation & Use Case

- At the moment, we are able to enter data into an HTML form and safely process it in PHP.

Use Case: Calculate Age from Birthday

- We want to know the **age** of applicants.
- Users enter their **birthdate** in a form.
- The PHP script should calculate the age.
- If today is the user's birthday:
  → Display **"Happy Birthday 🎂"**!

Envisioned form



```
<div class="container form-signin login">
  <div class="form-signin">
    <form role="form" method="post" action="">
      <div class="mb-3">
        <label for="birthdate" class="form-label">Enter your birthdate</label>

        <input type="date" class="form-control" name="birthdate" id="birthdate"
        aria-describedby="birthdateHelp" placeholder="DD-MM-YYYY" aria-required="true" required>

        <div id="birthdateHelp" class="form-text">
            Please provide your date of birth (format: DD-MM-YYYY).
        </div>
      </div>

      <button type="submit" class="btn btn-primary" aria-label="Submit birthdate form">
        Submit
      </button>
    </form>
  </div>
</div>
```

Task

Use the HTML form above to collect the user's birthdate. Then develop the post-processing code in PHP to retrieve the birthdate of the user. Calculate the age using this PHP code:

```php
// Convert to DateTime objects
$birthDateObj = new DateTime($birthdate);
$today = new DateTime();

// Calculate difference
$age = $today->diff($birthDateObj)->y;
```

Then output the age in the browser. E.g. `You are $age years old.`

```php
<?php
    $birthdate = "";
    $errors    = [];
    $success   = "";
    if ($_SERVER["REQUEST_METHOD"] === "POST") {
        $birthdate = trim($_POST["birthdate"] ?? "");
        if ($birthdate === "") {
            $errors[] = "Birthdate is required.";
        }
        if (empty($errors)) {
            // Convert to DateTime objects
            $birthDateObj = new DateTime($birthdate);
            $today = new DateTime();
            // Calculate difference
            $age = $today->diff($birthDateObj)->y;
            $success = "You are $age years old.";
        }
    }
    // Show errors
    if (!empty($errors)) {
        foreach ($errors as $error) {
            echo "<p style='color:red'>" . htmlspecialchars($error) . "</p>";
        }
    }
    // Show success message below the form
    if ($success !== "") {
        echo "<p style='color:green'>$success</p>";
    }
```

Task: Add Happy Birthday Message

- If today is the day (and month) of the user's birthday a "Happy Birthday" Message shall be displayed additionally.
- Use this condition to check if today is the user's birthday:

```php
$today->format("m-d") === $birthDateObj->format("m-d")
```

```php
// Calculate difference
$age = $today->diff($birthDateObj)->y;

if ($today->format("m-d") === $birthDateObj->format("m-d")) {
    $success = "Happy Birthday 🎂!<br>";
}

$success .= "You are $age years old.";
```

Note:

`$success .= "You are $age years old.";` is the short form for

`$success = $success . "You are $age years old.";`

Task: Create functions

- We want to reuse the part where we calculate the age in PHP.
- To accomplish this we can put the code inside a function e.g. `calculate_age()`.
- This function shall then be called when the user submits the form.

Note: one function should do one job! *#Single Responsibility*

- So `calculate_age()` shall only calculate the age, not create the success message or determine if today is the user's birthday!
- A separate function `is_birthday_today()` is best practice!

```php
function calculate_age($birthdate) {
    $birthDateObj = new DateTime($birthdate);
    $today = new DateTime();
    // return age
    return $today->diff($birthDateObj)->y;
}

function is_birthday_today($birthdate) {
    $birthDateObj = new DateTime($birthdate);
    $today = new DateTime();
    // return True if today's date is the same as the birthdate
    return ($today->format("m-d") === $birthDateObj->format("m-d"));
}
```

```php
$age = calculate_age($birthdate);
if (is_birthday_today($birthdate)) { // if True
    $success = "Happy Birthday 🎂!<br>";
}
$success .= "You are $age years old.";
```

Single Responsibility Principle (SRP)

❌ Bad: One function does too much

```php
function handleBirthday($birthdate) {
    $age = calculate_age()($birthdate);
    echo "You are $age years old";
    if ($birthdate matches today) {
        echo "Happy Birthday!";
    }
}
```

Note: This is considered a *Code Smell*

It is harder to read, test and reuse

and there is a higher chance for bugs, if you change something!

Single Responsibility Principle (SRP)

✅ Good: One function, one job

```php
function calculate_age($birthdate) { ... }
function is_birthday_today($birthdate) { ... }

// Separate "controller" prints messages
if (is_birthday_today($birthdate)) {
    echo "Happy Birthday!";
}
```

Task: Create a file functions.php

- We want to reuse these functions in other pages as well.
- To accomplish this we can create a new PHP file called `functions.php` that contains all PHP functions for better reuse.
- Then we need to somehow "load" this PHP file before we use the functions page.

functions.php

```php
<?php
function calculate_age($birthdate) {...}
function is_birthday_today($birthdate) {...}
```

some-page.php

```php
<?php
    // at the top of the first <?php section
    require_once 'functions.php';
    /* ... */
    $age = calculate_age($birthdate);
    if (is_birthday_today($birthdate)) { // if True
        $success = "Happy Birthday 🎂!<br>";
    }
```

Why require_once?

- Prevents multiple inclusions
  → avoids `function already declared`-errors
- Stops execution if the file is missing (safer than `include_once`)
- Makes dependencies explicit
  → easier to maintain

Function Type Declarations in PHP

Example

```php
function calculate_age(string $birthdate): int {
    // $birthdate must be a string (e.g. "2000-05-15")
    // The function must return an integer (the age)
}

function is_birthday_today(string $birthdate): bool {
    // $birthdate must be a string
    // The function must return a boolean (true / false)
}
```

Why use type declarations?

- Clarity: Makes it clear what kind of data the function expects and returns.
- Safety: Prevents accidental misuse (e.g. passing a number instead of a string).
- Error catching: PHP will throw an error if the wrong type is given.
- Documentation: The function explains itself → easier for others to read and use.

Task: add type declarations to the functions `calculate_age()` and `is_birthday_today()`.

❌ Without types:

```php
function calculate_age($birthdate) { ... }
```

- Hard to know what `$birthdate` should be
- What does it return? Integer? String? Float?

✅ With types:

```php
function calculate_age(string $birthdate): int { ... }
```

- Clear input type, clear return type
- Self-documenting code

PHP Namespaces: Preventing conflicts

Two namespaces with same function names

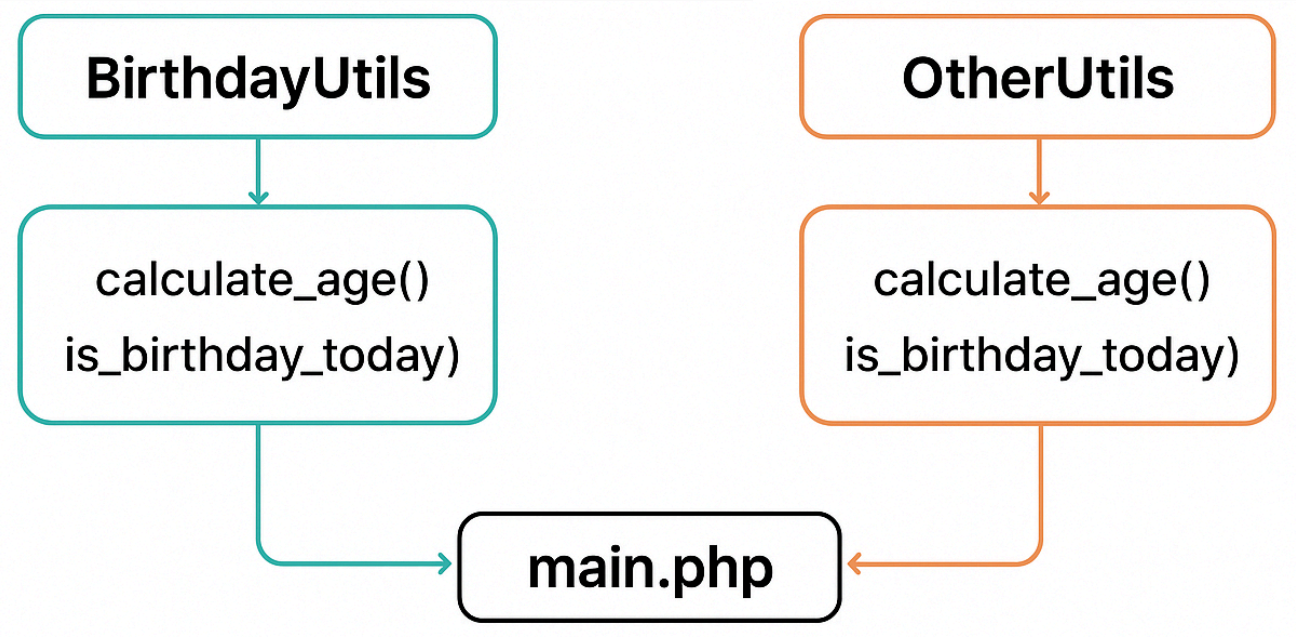
```php
// birthday_functions.php
namespace BirthdayUtils;

function calculate_age($birthdate): int { ... }
function is_birthday_today($birthdate): bool { ... }

// another_functions.php
namespace OtherUtils;

function calculate_age($birthdate): int { ... }
function is_birthday_today($birthdate): bool { ... }
```

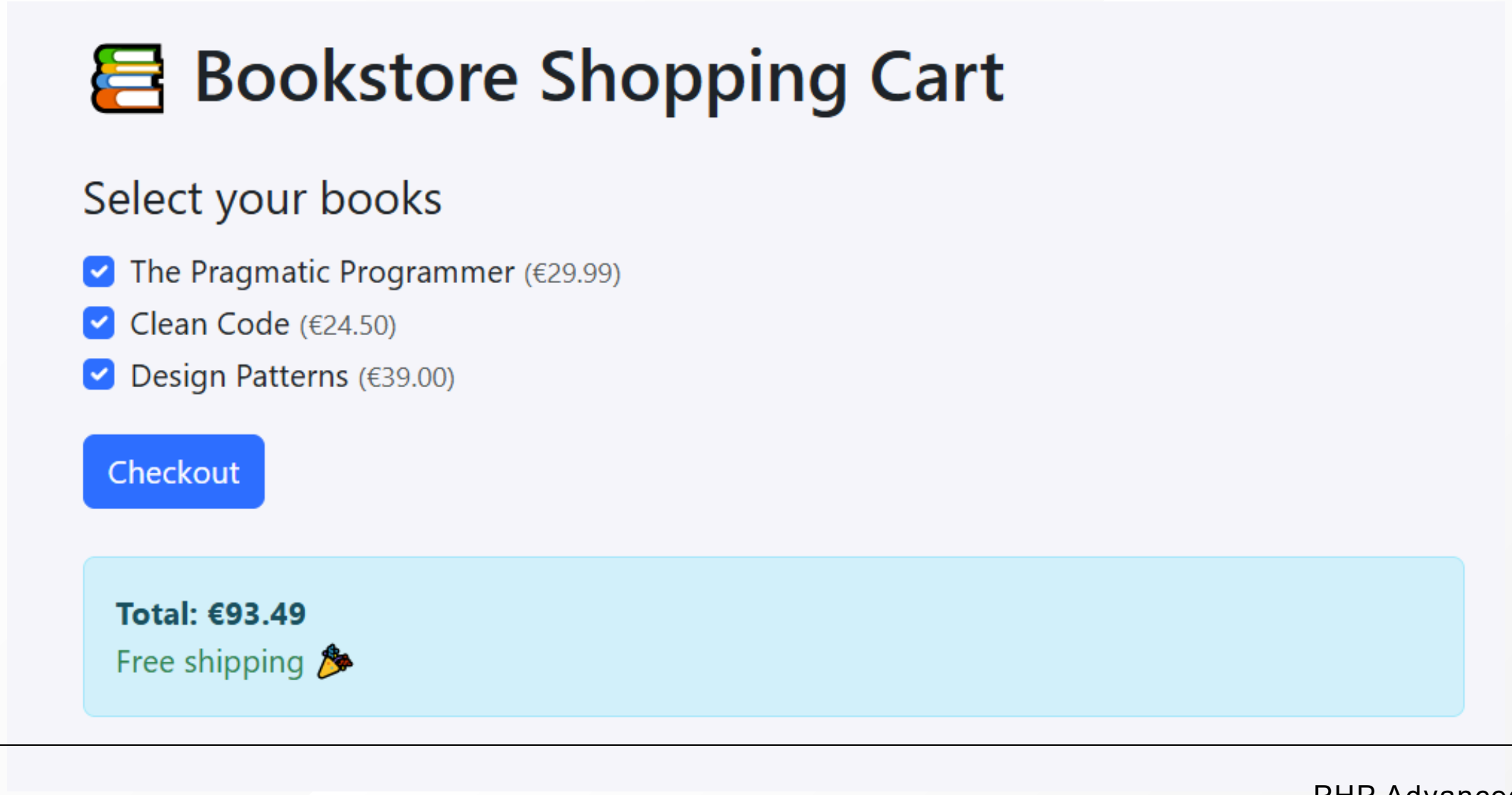
Namespace	Function Called	Result
`BirthdayUtils`	`BirthdayUtils\calculate_age()`	Calculates age using BirthdayUtils
`OtherUtils`	`OtherUtils\calculate_age()`	Calculates age using OtherUtils



Use Case: Bookstore Shopping Cart

- User selects books in a form.
- PHP calculates the total price.
- **Free shipping** if total ≥ €50, else **€4.99** shipping.
- Clean architecture: arrays, loops, thresholds, functions with type declarations, accessibility, SRP (no code smells).

Envisioned solution



0) Data model (Associative array)

We'll keep book data in an associative array:

```php
<?php
$books = [
  "book1" => ["title" => "The Pragmatic Programmer", "price" => 29.99],
  "book2" => ["title" => "Clean Code",               "price" => 24.50],
  "book3" => ["title" => "Design Patterns",          "price" => 39.00],
];
```

- Keys (book1, book2, …) are stable identifiers sent from the form.
- Values are arrays containing a title and a price.

1) HTML form (Bootstrap + ARIA)

```html
<!-- Include Bootstrap in <head> of the page -->
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">

<form method="post" action="" class="p-3" aria-labelledby="bookSelectionLabel">
  <fieldset class="mb-3">
    <legend id="bookSelectionLabel" class="form-label">Select your books</legend>

    <!-- Repeated per book (PHP will loop and render these) -->
    <div class="form-check">
      <input class="form-check-input" type="checkbox" id="book1" name="books[]"
        value="book1" aria-describedby="book1_price" >
      <label class="form-check-label" for="book1"> The Pragmatic Programmer </label>
      <small id="book1_price" class="text-muted">(€29.99)</small>
    </div>

    <!-- Add more books similarly -->
  </fieldset>

  <button type="submit" class="btn btn-primary">Checkout</button>
</form>
```

Accessibility notes

- Group related checkboxes with `<fieldset>` + `<legend>`.
- Each checkbox has an associated `<label for="...">`.
- `aria-describedby` links to price text for extra context.
- The button's visible text gives it an accessible name (no extra `aria-label` needed).

2) Reading form input safely

- Use the null coalescing operator so we don’t get notices on first page load.
- Always treat incoming $_POST data as untrusted.

```php
<?php
$selectedBooks = $_POST["books"] ?? []; // might be missing on first load
$selectedBooks = is_array($selectedBooks) ? $selectedBooks : []; // ensure array
```

3) Calculating total (inline logic first)

Before extracting a function, show the basic loop:

```php
<?php
$total = 0.0;

foreach ($selectedBooks as $key) {
  if (isset($books[$key])) {           // ignore tampered keys
    $total += $books[$key]["price"];   // sum prices
  }
}

// Shipping rule
$shippingMessage = "";
if ($total >= 50) {
  $shippingMessage = "<span class='text-success'>Free shipping 🎉</span>";
} elseif ($total > 0) {
  $total += 4.99;
  $shippingMessage = "<span class='text-muted'>+ €4.99 shipping</span>";
}
```

4) Extract a function (SRP, types)

Create a function that only calculates the total.
Use type declarations for clarity & safety.

```php
<?php
function calculate_total(array $selectedBooks, array $books): float {
  $sum = 0.0;
  foreach ($selectedBooks as $key) {
    if (isset($books[$key])) {
      $sum += $books[$key]["price"];
    }
  }
  return $sum;
}
```

- Single Responsibility: the function does one job (no shipping, no output).
- Easier to unit test and reuse.

5) Best practice: separate `functions.php` file

```php
<?php
declare(strict_types=1);

function calculate_total(array $selectedBooks, array $books): float {
  $sum = 0.0;
  foreach ($selectedBooks as $key) {
    if (isset($books[$key])) {
      $sum += $books[$key]["price"];
    }
  }
  return $sum;
}
```

In your main page, include it once:

```php
<?php
require_once 'functions.php';
```

- `declare(strict_types=1);` helps catch type mistakes early.
- require_once avoids duplicate includes and stops execution if missing.

Put it together

7) Why this design (Best practices recap)

- SRP: Functions do one thing (`calculate_total` only adds prices).
- Types: `array` and `float` in function signature → clarity & safety.
- Security: Escape user-visible values with `htmlspecialchars()`.
- Robust input: `$_POST["books"] ?? []` and ensure it's an array.
- Accessiblity: `<fieldset> + <legend>`, labels, `aria-describedby`, `role="alert"`.
- Maintainability: `require_once 'functions.php'`, clear data model, clean loops.

8) (Optional) Extensions for practice

- Quantities: replace checkboxes with numeric inputs (0–10), compute `price × qty`.
- Promo codes: apply percentage or fixed discounts with thresholds.
- Receipt: show a line-item breakdown (array iteration in a table).
- Persist cart: store selections in `$_SESSION` to survive navigation (Will be addressed next time).
- i18n: currency/locale formatting and language strings.
- Each extension reinforces arrays, loops, conditions, and function extraction.