Complexity and monads

Pol Dellaiera - DIGIT B4 - DIGIT PHP SUPPORT

June 2022

Version 4b84622

INTRODUCTION

European Commission

Introduction

Menu

- A little bit of theory...

Introduction
Menu

- A little bit of theory...
- A little bit of code...

Introduction

Menu

- A little bit of theory...
- A little bit of code...
- A lot of fun !

What makes code complex

What makes code complex

- Given an anonymous user

What makes code complex

- Given an anonymous user
- When I reach the page `/something/<any-number>`

What makes code complex

- Given an anonymous user
- When I reach the page `/something/<any-number>`
- Then it finds the entity with ID `<any-number>`

What makes code complex

- Given an anonymous user
- When I reach the page `/something/<any-number>`
- Then it finds the entity with ID `<any-number>`
- And the entity author ID is `1`

What makes code complex

- Given an anonymous user
- When I reach the page `/something/<any-number>`
- Then it finds the entity with ID `<any-number>`
- And the entity author ID is `1`
- And the entity title is `not empty`

What makes code complex

- Given an anonymous user
- When I reach the page `/something/<any-number>`
- Then it finds the entity with ID `<any-number>`
- And the entity author ID is `1`
- And the entity title is `not empty`
- And the entity title starts with `'abc'`

What makes code complex

- Given an anonymous user
- When I reach the page `/something/<any-number>`
- Then it finds the entity with ID `<any-number>`
- And the entity author ID is `1`
- And the entity title is `not empty`
- And the entity title starts with `'abc'`
- Then it returns the entity title in uppercase

```php
final class NewsController {
    /**
     * @Route("/news/{id}")
     */
    public function __invoke(string $id): ?string {
        $entity = $this->repository->find($id);

        if (null !== $entity) {
            if (1 === $entity->getAuthor()) {
                if (null !== $entity->getTitle()) {
                    $title = $entity->getTitle();
                    if (str_starts_with(strtolower($title), 'abc')) {
                        return strtoupper($title);
                    }
                }
            }
        }
        return null;
    }
}
```

European
Commission

Introduction
What makes code complex

- Is it the amount of language keywords?

Introduction
What makes code complex

- Is it the amount of language keywords?
- Is it the syntax of the language?

Introduction
What makes code complex

- Is it the amount of language keywords?
- Is it the syntax of the language?
- Is it the amount of decision points?

Introduction
What makes code complex

- Is it the amount of language keywords?

Introduction
What makes code complex

- Is it the amount of language keywords?
  - Smalltalk: ±6

Introduction
What makes code complex

- Is it the amount of language keywords?
  - Smalltalk: ±6
  - C: 32

Introduction
What makes code complex

- Is it the amount of language keywords?
  - Smalltalk: ±6
  - C: 32
  - Haskell: 55

Introduction

What makes code complex

- Is it the amount of language keywords?
  - Smalltalk: ±6
  - C: 32
  - Haskell: 55
  - Javascript: ±64

European Commission

Introduction
What makes code complex

- Is it the amount of language keywords?
  - Smalltalk: ±6
  - C: 32
  - Haskell: 55
  - Javascript: ±64
  - F#: 103

Introduction

What makes code complex

- Is it the amount of language keywords?
  - Smalltalk: ±6
  - C: 32
  - Haskell: 55
  - Javascript: ±64
  - F#: 103
  - PHP: ±67

Introduction
What makes code complex

- Is it the amount of language keywords?
  - Smalltalk: ±6
  - C: 32
  - Haskell: 55
  - Javascript: ±64
  - F#: 103
  - PHP: ±67
- Is it the syntax of the language?

Introduction
What makes code complex

- Is it the amount of language keywords?
  - Smalltalk: ±6
  - C: 32
  - Haskell: 55
  - Javascript: ±64
  - F#: 103
  - PHP: ±67
- Is it the syntax of the language?
  - Parenthesis

Introduction
What makes code complex

- Is it the amount of language keywords?
  - Smalltalk: ±6
  - C: 32
  - Haskell: 55
  - Javascript: ±64
  - F#: 103
  - PHP: ±67
- Is it the syntax of the language?
  - Parenthesis
  - Semicolon

Introduction
What makes code complex

- Is it the amount of language keywords?
  - Smalltalk: ±6
  - C: 32
  - Haskell: 55
  - Javascript: ±64
  - F#: 103
  - PHP: ±67
- Is it the syntax of the language?
  - Parenthesis
  - Semicolon
  - Indentation

Introduction
What makes code complex

- Is it the amount of language keywords?
  - Smalltalk: ±6
  - C: 32
  - Haskell: 55
  - Javascript: ±64
  - F#: 103
  - PHP: ±67
- Is it the syntax of the language?
  - Parenthesis
  - Semicolon
  - Indentation
- Is it the amount of decision points?

Introduction
What makes code complex

- Is it the amount of language keywords?
  - Smalltalk: ±6
  - C: 32
  - Haskell: 55
  - Javascript: ±64
  - F#: 103
  - PHP: ±67
- Is it the syntax of the language?
  - Parenthesis
  - Semicolon
  - Indentation
- Is it the amount of decision points?
  - If: 4

Introduction

Complexity types

- Time complexity

Introduction
Complexity types

- Time complexity
  - The amount of elementary operations

Introduction
Complexity types

- Time complexity
  - The amount of elementary operations
- Space complexity

Introduction
Complexity types

- Time complexity
  - The amount of elementary operations
- Space complexity
  - The amount of memory a program is using.

Introduction
Complexity types

- Time complexity
  - The amount of elementary operations
- Space complexity
  - The amount of memory a program is using.
- Kolmogorov complexity

Introduction
Complexity types

- Time complexity
  - The amount of elementary operations
- Space complexity
  - The amount of memory a program is using.
- Kolmogorov complexity
  - Given an output, this is the length of a shortest computer program that produces the output.

Introduction
Complexity types

- Time complexity
  - The amount of elementary operations
- Space complexity
  - The amount of memory a program is using.
- Kolmogorov complexity
  - Given an output, this is the length of a shortest computer program that produces the output.
- Cyclomatic complexity

Introduction
Complexity types

- Time complexity
  - The amount of elementary operations
- Space complexity
  - The amount of memory a program is using.
- Kolmogorov complexity
  - Given an output, this is the length of a shortest computer program that produces the output.
- Cyclomatic complexity
  - The quantitative measure of the number of linearly independent paths through a program’s source code.

When building an application using a framework, most the tools are already available and only the business logic needs to be implemented. Adding the business logic is usually what introduces most of the complexity.

When building an application using a framework, most the tools are already available and only the business logic needs to be implemented. Adding the business logic is usually what introduces most of the complexity.

The design of these tools have a great impact on how the end user will use them and therefore, on the overall complexity of the application.

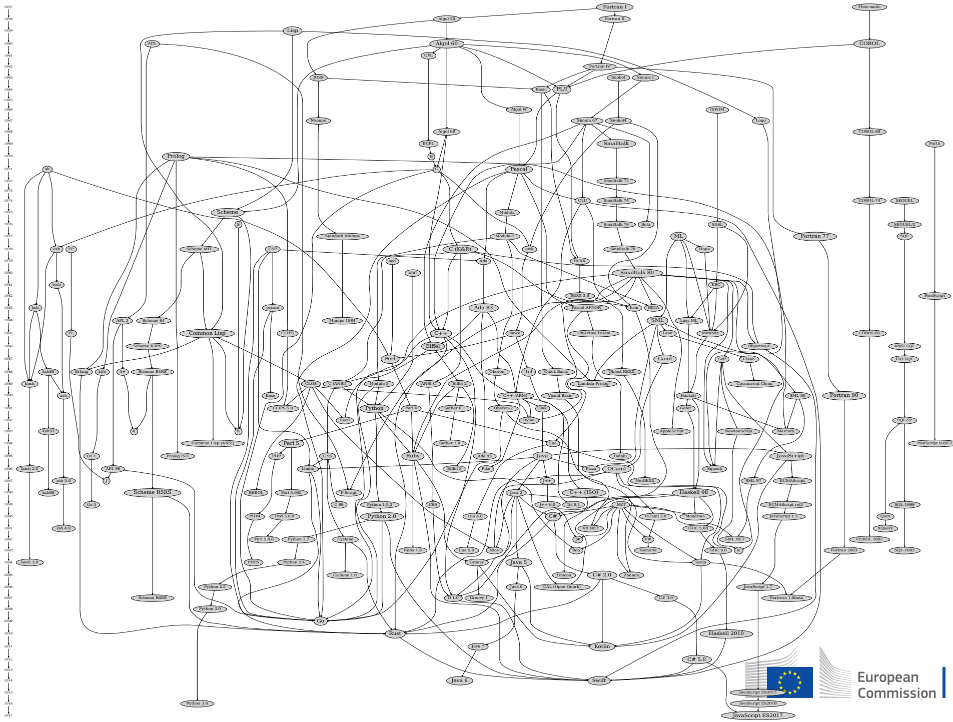
When building an application using a framework, most the tools are already available and only the business logic needs to be implemented. Adding the business logic is usually what introduces most of the complexity.

The design of these tools have a great impact on how the end user will use them and therefore, on the overall complexity of the application.

So today we are going to only focus on the **Cyclomatic complexity**.

HISTORY

A LITTLE BIT OF CONTEXTUAL INFORMATION



European Commission

History

Programming languages paradigms

- **O**bject **O**riented **P**rogramming

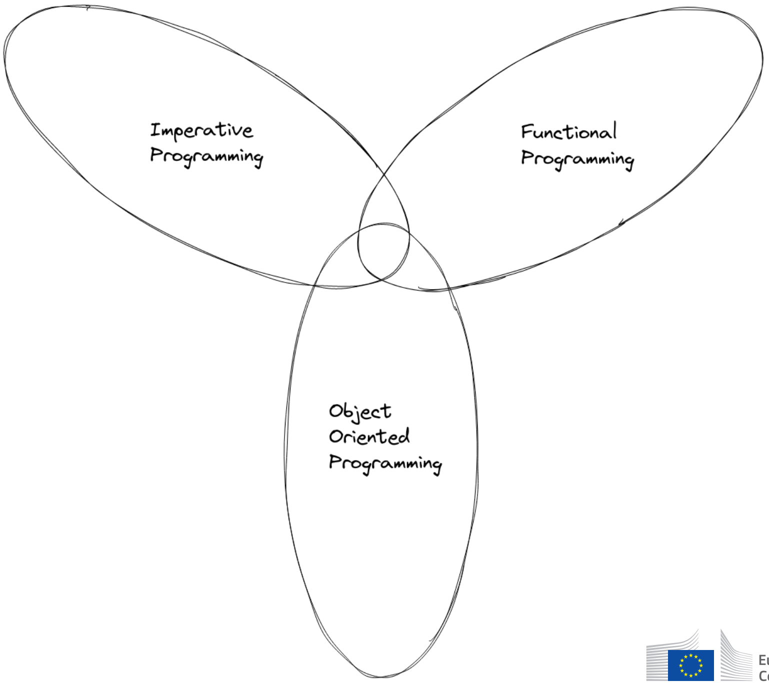
History

Programming languages paradigms

- **O**bject **O**riented **P**rogramming
- **F**unctional **P**rogramming

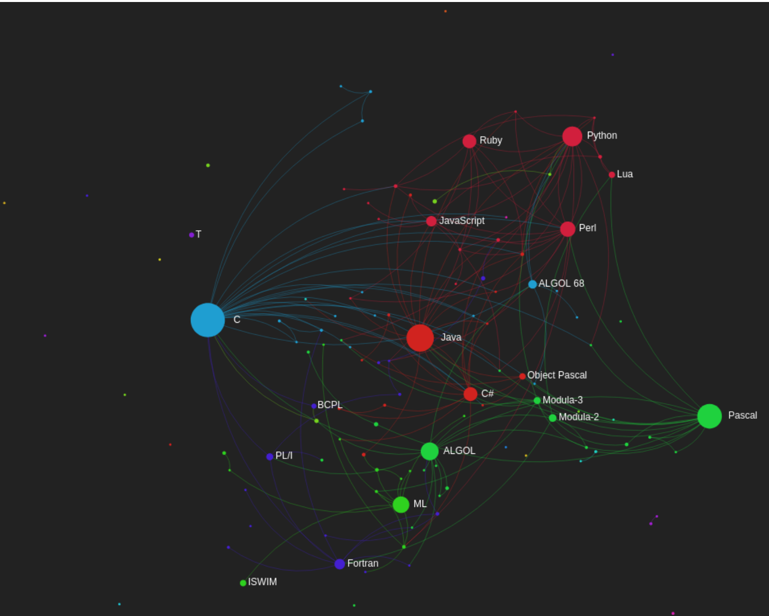
Programming languages paradigms

- **O**bject **O**riented **P**rogramming
- **F**unctional **P**rogramming
- **I**mperative **P**rogramming


Imperative Programming
Functional Programming
Object Oriented Programming

History

Imperative Programming



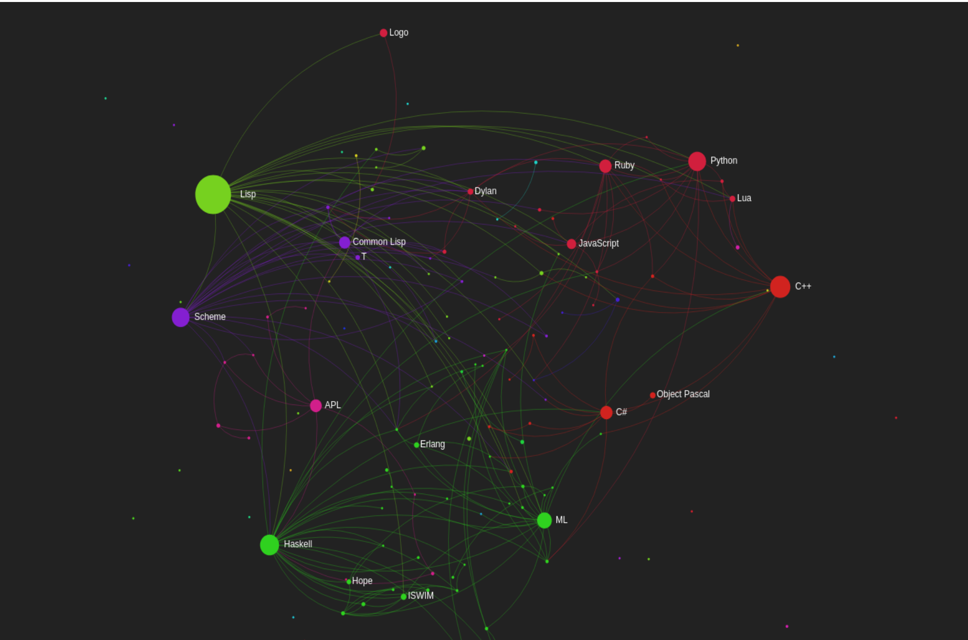
```
const input = [1, 2, 3, 4, 5, 6 ,7 ,8 ,9];
let odds = [];

for (let i = 0; i <= input.length; i++) {
    if (i % 2 !== 0) {
        odds.push(i);
    }
}

console.log(odds); // [1, 3, 5, 7, 9]
```

History

Functional Programming



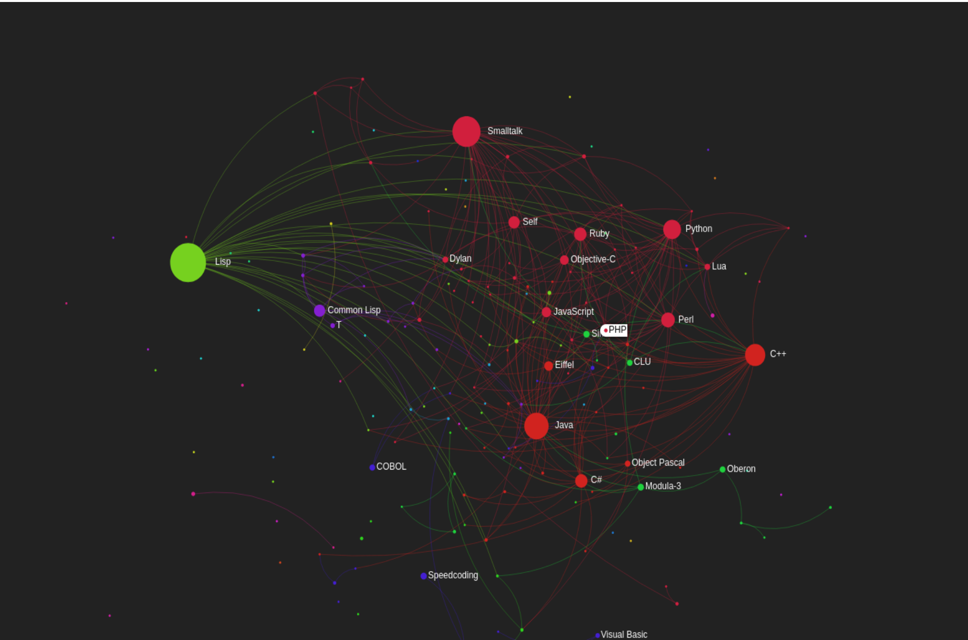
```javascript
const input = [1, 2, 3, 4, 5, 6 ,7 ,8 ,9];

const getOdds = arr => arr.filter(num => num % 2 !== 0);

console.log(getOdds(input)); // [1, 3, 5, 7, 9]
```

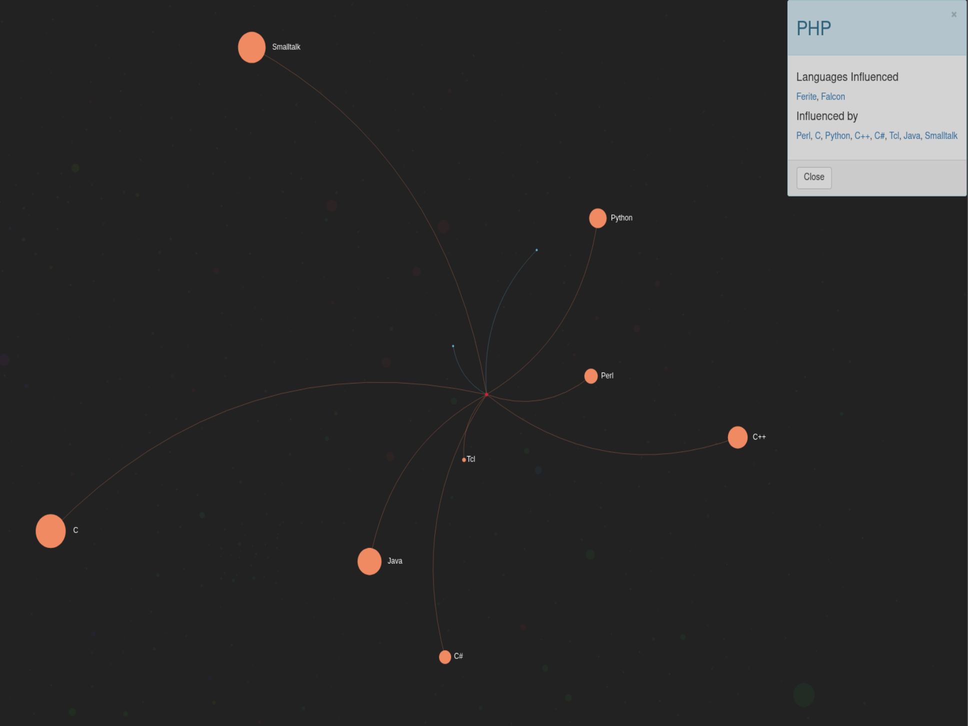
History

Object Oriented Programming



What about PHP?

European Commission



PHP

Languages Influenced

Ferite, Falcon

Influenced by

Perl, C, Python, C++, C#, Tcl, Java, Smalltalk

COMPLEXITY

Complexity at a glance

- Used to compare algorithms efficiency,

Complexity
Complexity at a glance

- Used to compare algorithms efficiency,
- Often calculated using the worst case scenario

Complexity

Complexity at a glance

- Used to compare algorithms efficiency,
- Often calculated using the worst case scenario
- In “Big $\mathcal{O}$” notation: $\mathcal{O}(1)$, $\mathcal{O}(n)$, $\mathcal{O}(n \times log(n))$, $\mathcal{O}(n^2)$

Complexity
Types

- Time complexity

Complexity
Types

- Time complexity
- Space complexity

Complexity
Types

- Time complexity
- Space complexity
- Kolmogorov complexity

Complexity
Types

- Time complexity
- Space complexity
- Kolmogorov complexity
- Cyclomatic complexity

Complexity

Constant time complexity: $\mathcal{O}(1)$

```php
final class TimeComplexity {

    public function constant(iterable $integers): int {
        foreach ($integers as $integer) {
            return $integer;
        }
    }
}
```

European
Commission

Complexity

Linear time complexity: $\mathcal{O}(n)$

```php
final class TimeComplexity {

    public function linear(iterable $integers): int {
        $min = 0;

        foreach ($integers as $integer) {
            $min = ($integer < $min)
                ? $integer
                : $min;
        }

        return $min;
    }
}
```

Complexity

Polynomial time complexity: $\mathcal{O}(n^2)$

```php
final class TimeComplexity {

    public function polynomial(iterable $matrix): int {
        $sum = 0;

        foreach ($matrix as $i => $row) {
            foreach ($row as $j => $column) {
                $sum += $matrix[$i][$j];
            }
        }

        return $sum;
    }
}
```

Complexity
Space complexity

```php
final class SpaceComplexity
{
    public function linear(iterable $input): int {
        $sum = 0;

        foreach ($input as $item) {
            $sum += $item;
        }

        return $sum;
    }
}
```

Complexity

Kolmogorov complexity

The Kolmogorov complexity is the shortest size of a program that yield the expected output.

Complexity

Kolmogorov complexity

- 1111111111111111111111111111111111111111111111111111111111111111

Complexity
Kolmogorov complexity

- `1111111111111111111111111111111111111111111111111111111111111111`
- `317b773017df0ab62b15cd3f2ad17d7b13ab02f05f4943011ef8c4067d1ca0a5`

European Commission

Complexity

Kolmogorov complexity

```
final class KolmogorovComplexity
{
    public function example1(): string {
        return implode('',array_pad([],64,1));
    }
```

Complexity

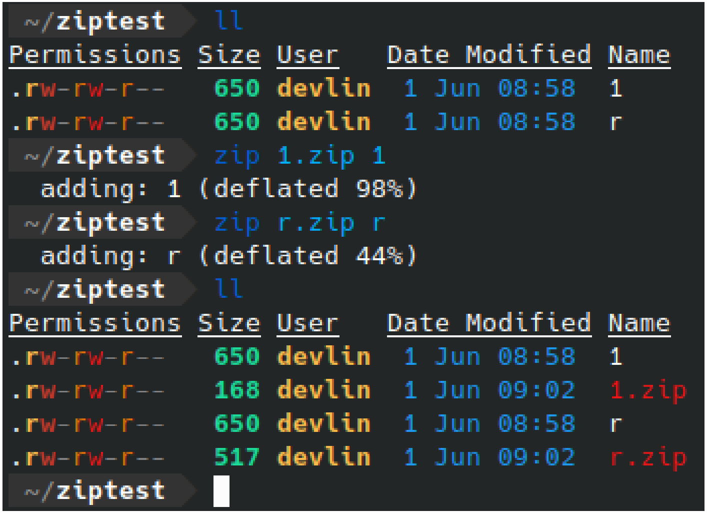
Kolmogorov complexity

```php
final class KolmogorovComplexity
{
    public function example1(): string {
        return implode('',array_pad([],64,1));
    }

    public function example2(): string {
        return "317b773017df0ab62b15cd3f2ad17d7b13ab02f05f4943011ef8c4067d1ca0a5";
    }
}
```

Complexity

Kolmogorov complexity



```
~/ziptest  ll
Permissions Size User    Date Modified Name
.rw-rw-r--   650 devlin  1 Jun 08:58   1
.rw-rw-r--   650 devlin  1 Jun 08:58   r
~/ziptest  zip 1.zip 1
  adding: 1 (deflated 98%)
~/ziptest  zip r.zip r
  adding: r (deflated 44%)
~/ziptest  ll
Permissions Size User    Date Modified Name
.rw-rw-r--   650 devlin  1 Jun 08:58   1
.rw-rw-r--   168 devlin  1 Jun 09:02   1.zip
.rw-rw-r--   650 devlin  1 Jun 08:58   r
.rw-rw-r--   517 devlin  1 Jun 09:02   r.zip
~/ziptest
```

Complexity

Kolmogorov complexity

There is no way to tell if the Kolmogorov complexity of an algorithm is the shortest one.

Complexity

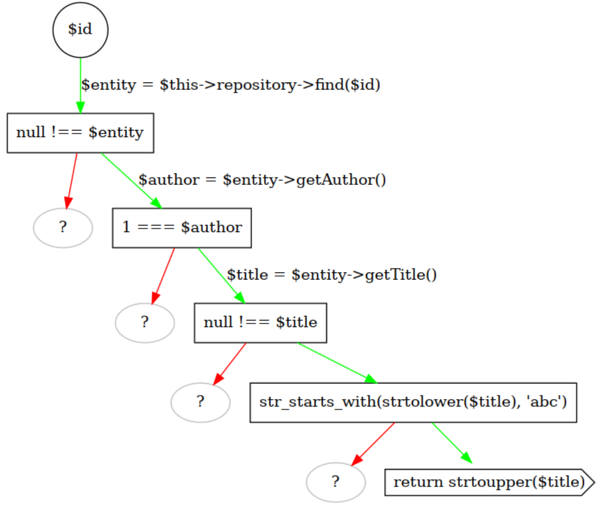
Cyclomatic complexity

```
final class NewsController {
    /**
     * @Route("/news/{id}")
     */
    public function __invoke(string $id): ?string {
        $entity = $this->repository->find($id);

        if (null !== $entity) {
            if (1 === $entity->getAuthor()) {
                if (null !== $entity->getTitle()) {
                    $title = $entity->getTitle();
                    if (str_starts_with(strtolower($title), 'abc')) {
                        return strtoupper($title);
                    }
                }
            }
        }
        return null;
    }
}
```

Complexity

Cyclomatic complexity



$id
$entity = $this->repository->find($id)
null !== $entity
?
$author = $entity->getAuthor()
1 === $author
?
$title = $entity->getTitle()
null !== $title
?
str_starts_with(strtolower($title), 'abc')
?
return strtoupper($title)

Complexity

Cyclomatic complexity

Conditions and type checks adds complexity to a program, we are going to see how we can get rid of them in a nice an clean way.

But

But but

But but but

But but but... we need conditions !

Early returns

Complexity

Cyclomatic complexity

```
if ($expr1 && $expr2 && $expr3) {
    return true;
}

return false;

```

Complexity

Cyclomatic complexity

```php
if ($expr1 && $expr2 && $expr3) {
    return true;
}

return false;

```

is equivalent to:

```php
if (! $expr1) {
    return false;
}

if (! $expr2) {
    return false;
}

if (! $expr3) {
    return false;
}

return true;

```

Complexity

Cyclomatic complexity

```php
public function __invoke(string $id): ?string {
    $entity = $this->repository->find($id);

    if (null === $entity) {
        return;
    }

    if (1 !== $entity->getAuthor()) {
        return;
    }

    if (null === $entity->getTitle()) {
        return;
    }

    $title = $entity->getTitle();

    if (false === str_starts_with(strtolower($title), 'abc')) {
        return;
    }

    return strtoupper($title);
}
```

European Commission

Complexity
Cyclomatic complexity

- Think to the “unhappy” paths at first

Complexity
Cyclomatic complexity

- Think to the “unhappy” paths at first
- The “happy” path, is usually the last line,

Complexity
Cyclomatic complexity

- Think to the “unhappy” paths at first
- The “happy” path, is usually the last line,
- Easier to read, understand,

Complexity

Cyclomatic complexity

- Think to the “unhappy” paths at first
- The “happy” path, is usually the last line,
- Easier to read, understand,
- Longer to write.

Programming paradigms

Complexity

Imperative programming

Imperative programming tells the machine how to do something.

Complexity

Imperative programming

Imperative programming tells the machine how to do something.(resulting in what you want to happen).

Complexity

Declarative programming

Declarative programming tells the machine what you would like to happen.

Complexity

Declarative programming

Declarative programming tells the machine what you would like to happen.(and the computer figures out how to do it)

Complexity

Imperative and declarative programming

Imperative programming

```
const input = [1, 2, 3, 4, 5, 6 ,7 ,8 ,9];
let odds = [];

for (let i = 0; i <= input.length; i++) {
    if (i % 2 !== 0) {
        odds.push(i);
    }
}

console.log(odds); // [1, 3, 5, 7, 9]
```

European Commission

Complexity

Imperative and declarative programming

Imperative programming

```
const input = [1, 2, 3, 4, 5, 6 ,7 ,8 ,9];
let odds = [];

for (let i = 0; i <= input.length; i++) {
    if (i % 2 !== 0) {
        odds.push(i);
    }
}

console.log(odds); // [1, 3, 5, 7, 9]
```

Declarative programming

```
const input = [1, 2, 3, 4, 5, 6 ,7 ,8 ,9];

const getOdds = arr => arr.filter(num => num % 2 !== 0);

console.log(getOdds(input)); // [1, 3, 5, 7, 9]
```

Complexity

Imperative and declarative programming

Imperative programming

```
const input = [1, 2, 3, 4, 5, 6 ,7 ,8 ,9];
let total = 0;

for (let i = 0; i < input.length; i++) {
  total += input[i];
}

console.log(total) // 45
```

Complexity

Imperative and declarative programming

Imperative programming

```
const input = [1, 2, 3, 4, 5, 6 ,7 ,8 ,9];
let total = 0;

for (let i = 0; i < input.length; i++) {
  total += input[i];
}

console.log(total) // 45
```

Declarative programming

```
const input = [1, 2, 3, 4, 5, 6 ,7 ,8 ,9];

const total = input.reduce((sum, n) => sum + n);

console.log(total) // 45
```

Avoid N.I.H. syndrome

Complexity

Confidence in other packages?

Relying on someone else’s code means a lot, for some reason people prefer redoing things on their own.(NIH vs PFE)

Complexity

Confidence in other packages?

Relying on someone else’s code means a lot, for some reason people prefer redoing things on their own.(NIH vs PFE)

I believe that developers should be able to evaluate the trust into a package based on some key indicators.

Complexity

Confidence in other packages?

Relying on someone else's code means a lot, for some reason people prefer redoing things on their own.(NIH vs PFE)

I believe that developers should be able to evaluate the trust into a package based on some key indicators.

Tests, popularity, readability, extensibility, practices...

European Commission

FUNCTIONAL CODE

Functional code

Tips and tricks

- Make your objects **final**,

Functional code

Tips and tricks

- Make your objects **final**,
- Make your properties **private** and **immutable**,

Functional code

Tips and tricks

- Make your objects **final**,
- Make your properties **private** and **immutable**,
- Make sure your functions are **total** and **pure**.

Functional code

Final classes

Class is not final

```
class Multiply {
  public function __invoke(int $a, int $b): int
  {
      return $a * $b;
  }
}

(new Multiply)(6, 7); // 42
```

Functional code

Final classes

Class is not final

```
class Multiply {
  public function __invoke(int $a, int $b): int
  {
      return $a * $b;
  }
}

(new Multiply)(6, 7); // 42
```

When extending a class which is not final...

```
class RealMultiply extends Multiply {
  public function __invoke(int $a, int $b): int
  {
    return random_int(0, 1000);
  }
}

var_dump((new RealMultiply) instanceof Multiply); // true
(new RealMultiply)(6, 7); // ???
```

European Commission

Functional code

Final classes

Class is not final

```php
class PasswordValidator {
  public function __invoke(string $hash, string $clearPassword): bool
  {
      return $hash === sha1($clearPassword);
  }
}

(new PasswordValidator)('/* hash */', 'hello'); // true or false
```

Functional code

Final classes

Class is not final

```php
class PasswordValidator {
  public function __invoke(string $hash, string $clearPassword): bool
  {
      return $hash === sha1($clearPassword);
  }
}

(new PasswordValidator)('/* hash */', 'hello'); // true or false
```

When extending a class which is not final...

```php
class BetterPasswordValidator extends PasswordValidator {
  public function __invoke(string $hash, string $clearPassword): bool
  {
      return true;
  }
}

var_dump((new BetterPasswordValidator) instanceof PasswordValidator); // true
(new BetterPasswordValidator)('/* hash */', 'hello'); // true
(new BetterPasswordValidator)('/* hash */', 'admin'); // true
```

European Commission

Functional code

Public and private properties

Public properties

```php
class User {
    public string $isAdmin = false;
}

$user = new User;
$user->isAdmin; // false
$user->isAdmin = true;
$user->isAdmin; // true
```

Functional code
Public and private properties

Public properties

```
class User {
    public string $isAdmin = false;
}

$user = new User;
$user->isAdmin; // false
$user->isAdmin = true;
$user->isAdmin; // true
```

Private properties

```
class User {
    private string $isAdmin = false;
}

$user = new User;
$user->isAdmin; // error
$user->isAdmin = true; // error
```

Functional code

Partial and total functions

A total function is a function which is defined for all inputs.

Functional code
Partial and total functions

A total function is a function which is defined for all inputs.

Definition (Total function)

$$f : A \mapsto B$$
$$\forall a \in A \implies \exists\, f(a) \in B$$

Functional code

Partial and total functions

A partial function is a function which is not defined for all inputs.

Functional code

Partial and total functions

A partial function is a function which is not defined for all inputs.

Definition (Partial function)

$$f : A \rightharpoonup B$$

$$\exists x \in A \implies \nexists\, f(x) \in B$$

Functional code

Partial and total functions

Definition (Total function)

$$f : A \mapsto B$$
$$\forall a \in A \implies \exists f(a) \in B$$

Definition (Partial function)

$$f : A \mapsto B$$
$$\exists x \in A \implies \nexists f(x) \in B$$

Functional code

Partial and total functions

- Total: `function add (int $a, int $b): int`

Functional code

Partial and total functions

- Total: `function add (int $a, int $b): int`
- Partial: `function divide(int $a, int $b): float`

Functional code

Partial and total functions

- Total: `function add (int $a, int $b): int`
- Partial: `function divide(int $a, int $b): float`
- Not pure: `time()`

Functional code

Partial and total functions

- Total: `function add (int $a, int $b): int`
- Partial: `function divide(int $a, int $b): float`
- Not pure: `time()`
- Pure: `function add (int $a, int $b): int`

What

What the

What the link

What the link ????

European Commission

Functional code

Handling errors

A Doctrine repository definition

```php
class MyEntityRepository extends ServiceEntityRepository {
    // ...8<...
    public function find($id): ?MyEntity {
        // body
    }
    // ...>8...
}
```

Functional code

Handling errors

A Doctrine repository definition

```php
class MyEntityRepository extends ServiceEntityRepository {
    // ...8<...
    public function find($id): ?MyEntity {
        // body
    }
    // ...>8...
}
```

A Doctrine repository in use

```php
$repo = new MyEntityRepository();
$entity = $repo->find($id);

if (null !== $entity) {
    // Do something.
}
```

Functional code

Tips and tricks

Return null

```php
function safeSquareRoot(float $n): ?float {
    return ($n < 0)
      ? null
      : sqrt($n);
}
```

Functional code

Tips and tricks

Return null

```php
function safeSquareRoot(float $n): ?float {
    return ($n < 0)
      ? null
      : sqrt($n);
}
```

Throw exception

```php
function safeSquareRoot(float $n): float {
    return ($n < 0)
      ? throw new DomainException('Invalid input.')
      : sqrt($n);
}
```

European Commission

Functional code

Tips and tricks

Return null

```php
function safeSquareRoot(float $n): ?float {
    return ($n < 0)
      ? null
      : sqrt($n);
}
```

Throw exception

```php
function safeSquareRoot(float $n): float {
    return ($n < 0)
      ? throw new DomainException('Invalid input.')
      : sqrt($n);
}
```

Return sentinel

```php
function safeSquareRoot(float $n): float {
    return ($n < 0)
      ? 0.0
      : sqrt($n);
}
```

Functional code

Tips and tricks

- Return null?

Functional code

Tips and tricks

- Return null?
  Mixing types implies more code for the end user, prevent composition.
- Throw exceptions?

Functional code

Tips and tricks

- Return null?
  Mixing types implies more code for the end user, prevent composition.
- Throw exceptions?
  Great power means great responsibility, has the power to stop the program.
- Return sentinel?

Functional code

Tips and tricks

- Return null?
  Mixing types implies more code for the end user, prevent composition.
- Throw exceptions?
  Great power means great responsibility, has the power to stop the program.
- Return sentinel?
  Prone to incertitude when it returns 0.

Functional code

Return null?

- Universal error value that doesn’t carry any relevant information in case of issues.

Functional code

Return null?

- Universal error value that doesn’t carry any relevant information in case of issues.
- Widespread, implemented in most languages

Functional code

Return null?

- Universal error value that doesn’t carry any relevant information in case of issues.
- Widespread, implemented in most languages
- Considered as a code smell and a "the billion-dollar mistake"

Functional code

Return null?

- Universal error value that doesn’t carry any relevant information in case of issues.
- Widespread, implemented in most languages
- Considered as a code smell and a "the billion-dollar mistake"
- Prevent composition pattern

Functional code

Throw exceptions?

- Indicate something unforeseen by the developer

Functional code

Throw exceptions?

- Indicate something unforeseen by the developer
- Expensive, destructive

Functional code
Throw exceptions?

- Indicate something unforeseen by the developer
- Expensive, destructive
- Definitely prevent composition pattern

Functional code

Return sentinel?

- Potentially prevent composition pattern

Functional code

Return sentinel?

- Potentially prevent composition pattern
- Must be carefully done, each type has its own unit type

Functional code
Return sentinel?

- Potentially prevent composition pattern
- Must be carefully done, each type has its own unit type
  - `function divide(int $a, int $b): float|string`

Functional code

Return sentinel?

- Potentially prevent composition pattern
- Must be carefully done, each type has its own unit type
  - `function divide(int $a, int $b): float|string`
  - `function divide(int $a, int $b): float|array`

Functional code

Return sentinel?

- Potentially prevent composition pattern
- Must be carefully done, each type has its own unit type
  - `function divide(int $a, int $b): float|string`
  - `function divide(int $a, int $b): float|array`
  - `function divide(int $a, int $b): float|null`

Functional code

Return sentinel?

- Potentially prevent composition pattern
- Must be carefully done, each type has its own unit type
  - `function divide(int $a, int $b): float|string`
  - `function divide(int $a, int $b): float|array`
  - `function divide(int $a, int $b): float|null`
  - `function divide(int $a, int $b): float|int`

Functional code

Return sentinel?

- Potentially prevent composition pattern
- Must be carefully done, each type has its own unit type
  - `function divide(int $a, int $b): float|string`
  - `function divide(int $a, int $b): float|array`
  - `function divide(int $a, int $b): float|null`
  - `function divide(int $a, int $b): float|int`
- In PHP, this is sadly happening

Functional code

What is our best option?

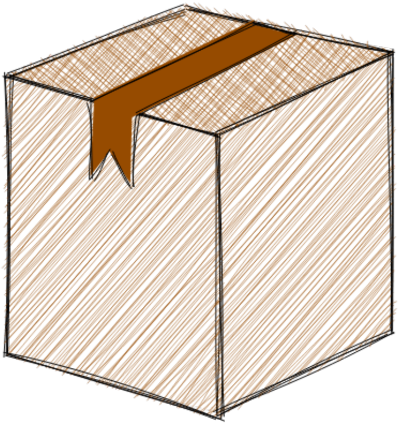
- Not using PHP ?

Functional code
What is our best option?

- Not using PHP ?
- Avoid using strict types ?

Functional code
What is our best option?

- Not using PHP ?
- Avoid using strict types ?
- Wrap the incertitude in a well-know object?



Maybe class directions

```
final class Maybe {
```

Maybe class directions

```
final class Maybe {
    public function __construct(mixed $value): self;

```

Maybe class directions

```php
final class Maybe {
    public function __construct(mixed $value): self;

    public function __invoke(): mixed;

```

European Commission

Maybe class directions

```php
final class Maybe {
    public function __construct(mixed $value): self;

    public function __invoke(): mixed;

    public function then(callable $f): self;
}
```

Maybe class implementation and usage

```php
<?php

final class Maybe {
    public function __construct(private mixed $value) {}

```

Maybe class implementation and usage

```php
<?php

final class Maybe {
    public function __construct(private mixed $value) {}

    public function __invoke(): mixed { return $this->value; }

```

Maybe class implementation and usage

```php
<?php

final class Maybe {
    public function __construct(private mixed $value) {}

    public function __invoke(): mixed { return $this->value; }

    public function then(callable $f): self {
        if (null === $this->value) {
            return $this;
        }

        return new self($f($this->value));
    }
}

```

Maybe class implementation and usage

```php
<?php

final class Maybe {
    public function __construct(private mixed $value) {}

    public function __invoke(): mixed { return $this->value; }

    public function then(callable $f): self {
        if (null === $this->value) {
            return $this;
        }

        return new self($f($this->value));
    }
}

(new Maybe('a'))->then('strtoupper')();
```

European Commission

Maybe class implementation and usage

```php
<?php

final class Maybe {
    public function __construct(private mixed $value) {}

    public function __invoke(): mixed { return $this->value; }

    public function then(callable $f): self {
        if (null === $this->value) {
            return $this;
        }

        return new self($f($this->value));
    }
}

(new Maybe('a'))->then('strtoupper')(); // A
```

European
Commission

Maybe class implementation and usage

```php
<?php

final class Maybe {
    public function __construct(private mixed $value) {}

    public function __invoke(): mixed { return $this->value; }

    public function then(callable $f): self {
        if (null === $this->value) {
            return $this;
        }

        return new self($f($this->value));
    }
}

(new Maybe('a'))->then('strtoupper')(); // A
(new Maybe(null))->then('strtoupper')();
```

Maybe class implementation and usage

```php
<?php

final class Maybe {
    public function __construct(private mixed $value) {}

    public function __invoke(): mixed { return $this->value; }

    public function then(callable $f): self {
        if (null === $this->value) {
            return $this;
        }

        return new self($f($this->value));
    }
}

(new Maybe('a'))->then('strtoupper')(); // A
(new Maybe(null))->then('strtoupper')(); // null
```

Maybe class with our snippet

```php
(new Maybe($repository->find($id)))
```

Maybe class with our snippet

```php
(new Maybe($repository->find($id)))
    ->then(
        fn (object $e): ?object => (1 === $e->getAuthor()) ? $e : null
    )
```

Maybe class with our snippet

```
(new Maybe($repository->find($id)))
    ->then(
        fn (object $e): ?object => (1 === $e->getAuthor()) ? $e : null
    )
    ->then(
        fn (object $e): ?string => $e->getTitle()
    )
```

European Commission

Maybe class with our snippet

```php
(new Maybe($repository->find($id)))
    ->then(
        fn (object $e): ?object => (1 === $e->getAuthor()) ? $e : null
    )
    ->then(
        fn (object $e): ?string => $e->getTitle()
    )
    ->then(
        fn (string $t): ?string => str_starts_with('abc', strtolower($t))
            ? $t
            : null
    )
```

European Commission

Maybe class with our snippet

```php
(new Maybe($repository->find($id)))
    ->then(
        fn (object $e): ?object => (1 === $e->getAuthor()) ? $e : null
    )
    ->then(
        fn (object $e): ?string => $e->getTitle()
    )
    ->then(
        fn (string $t): ?string => str_starts_with('abc', strtolower($t))
            ? $t
            : null
    )
    ->then(
        fn (string $t): string => strtoupper($t)
    )();
```

European Commission

Maybe class with our snippet and callbacks

```php
$checkAuthor = fn (object $e): ?object => (1 === $e->getAuthor())
    ? $e
    : null;
```

Maybe class with our snippet and callbacks

```php
$checkAuthor = fn (object $e): ?object => (1 === $e->getAuthor())
    ? $e
    : null;

$getTitle    = fn (object $e): ?string => $e->getTitle();
```

Maybe class with our snippet and callbacks

```php
$checkAuthor = fn (object $e): ?object => (1 === $e->getAuthor())
    ? $e
    : null;

$getTitle    = fn (object $e): ?string => $e->getTitle();

$checkTitle  = fn (string $t): ?string => str_starts_with('abc', strtolower($t))
    ? $t
    : null;
```

Maybe class with our snippet and callbacks

```php
$checkAuthor = fn (object $e): ?object => (1 === $e->getAuthor())
    ? $e
    : null;

$getTitle    = fn (object $e): ?string => $e->getTitle();

$checkTitle  = fn (string $t): ?string => str_starts_with('abc', strtolower($t))
    ? $t
    : null;

$findEntity  = fn (int $id): Maybe => new Maybe($repository->find($id));
```

Maybe class with our snippet and callbacks

```php
$checkAuthor = fn (object $e): ?object => (1 === $e->getAuthor())
    ? $e
    : null;

$getTitle    = fn (object $e): ?string => $e->getTitle();

$checkTitle  = fn (string $t): ?string => str_starts_with('abc', strtolower($t))
    ? $t
    : null;

$findEntity  = fn (int $id): Maybe => new Maybe($repository->find($id));

$findEntity
```

Maybe class with our snippet and callbacks

```php
$checkAuthor = fn (object $e): ?object => (1 === $e->getAuthor())
    ? $e
    : null;

$getTitle    = fn (object $e): ?string => $e->getTitle();

$checkTitle  = fn (string $t): ?string => str_starts_with('abc', strtolower($t))
    ? $t
    : null;

$findEntity  = fn (int $id): Maybe => new Maybe($repository->find($id));

$findEntity
    ->then($checkAuthor)
```

Maybe class with our snippet and callbacks

```php
$checkAuthor = fn (object $e): ?object => (1 === $e->getAuthor())
    ? $e
    : null;

$getTitle    = fn (object $e): ?string => $e->getTitle();

$checkTitle  = fn (string $t): ?string => str_starts_with('abc', strtolower($t))
    ? $t
    : null;

$findEntity  = fn (int $id): Maybe => new Maybe($repository->find($id));

$findEntity
    ->then($checkAuthor)
    ->then($getTitle)
```

European Commission

Maybe class with our snippet and callbacks

```php
$checkAuthor = fn (object $e): ?object => (1 === $e->getAuthor())
    ? $e
    : null;

$getTitle    = fn (object $e): ?string => $e->getTitle();

$checkTitle  = fn (string $t): ?string => str_starts_with('abc', strtolower($t))
    ? $t
    : null;

$findEntity  = fn (int $id): Maybe => new Maybe($repository->find($id));

$findEntity
    ->then($checkAuthor)
    ->then($getTitle)
    ->then($checkTitle)
```

European Commission

Maybe class with our snippet and callbacks

```php
$checkAuthor = fn (object $e): ?object => (1 === $e->getAuthor())
    ? $e
    : null;

$getTitle    = fn (object $e): ?string => $e->getTitle();

$checkTitle  = fn (string $t): ?string => str_starts_with('abc', strtolower($t))
    ? $t
    : null;

$findEntity  = fn (int $id): Maybe => new Maybe($repository->find($id));

$findEntity
    ->then($checkAuthor)
    ->then($getTitle)
    ->then($checkTitle)
    ->then('strtoupper')
    ();
```

Draft Maybe class

```php
<?php

final class Maybe {
    public function __construct(private mixed $value) {}

```

Draft Maybe class

```php
<?php

final class Maybe {
    public function __construct(private mixed $value) {}

    public function __invoke(): mixed { return $this->value; }

```

Draft Maybe class

```php
<?php

final class Maybe {
    public function __construct(private mixed $value) {}

    public function __invoke(): mixed { return $this->value; }

    public function then(callable $f): self {
        if (null === $this->value) {
            return $this;
        }

        return $f($this->value);
    }
}
```

Draft Maybe class

```php
$checkAuthor = fn (object $e): Maybe => new Maybe(
    (1 === $e->getAuthor())
        ? $e
        : null
);
```

Draft Maybe class

```php
$checkAuthor = fn (object $e): Maybe => new Maybe(
    (1 === $e->getAuthor())
        ? $e
        : null
);

$getTitle    = fn (object $e): Maybe => new Maybe($e->getTitle());
```

Draft Maybe class

```php
$checkAuthor = fn (object $e): Maybe => new Maybe(
    (1 === $e->getAuthor())
        ? $e
        : null
);

$getTitle    = fn (object $e): Maybe => new Maybe($e->getTitle());

$checkTitle  = fn (string $t): Maybe => new Maybe(
    str_starts_with('abc', strtolower($t))
        ? $t
        : null
);
```

Draft Maybe class

```php
$checkAuthor = fn (object $e): Maybe => new Maybe(
    (1 === $e->getAuthor())
        ? $e
        : null
);

$getTitle    = fn (object $e): Maybe => new Maybe($e->getTitle());

$checkTitle  = fn (string $t): Maybe => new Maybe(
    str_starts_with('abc', strtolower($t))
        ? $t
        : null
);

$strToUpper  = fn (string $s): Maybe => new Maybe(strtoupper($s));
```

Draft Maybe class

```php
$checkAuthor = fn (object $e): Maybe => new Maybe(
    (1 === $e->getAuthor())
        ? $e
        : null
);

$getTitle    = fn (object $e): Maybe => new Maybe($e->getTitle());

$checkTitle  = fn (string $t): Maybe => new Maybe(
    str_starts_with('abc', strtolower($t))
        ? $t
        : null
);

$strToUpper  = fn (string $s): Maybe => new Maybe(strtoupper($s));

$findEntity  = fn (int $id): Maybe => new Maybe($repository->find($id));
```

Draft Maybe class

```php
$checkAuthor = fn (object $e): Maybe => new Maybe(
    (1 === $e->getAuthor())
        ? $e
        : null
);

$getTitle    = fn (object $e): Maybe => new Maybe($e->getTitle());

$checkTitle  = fn (string $t): Maybe => new Maybe(
    str_starts_with('abc', strtolower($t))
        ? $t
        : null
);

$strToUpper  = fn (string $s): Maybe => new Maybe(strtoupper($s));

$findEntity  = fn (int $id): Maybe => new Maybe($repository->find($id));

$findEntity
```

Draft Maybe class

```php
$checkAuthor = fn (object $e): Maybe => new Maybe(
    (1 === $e->getAuthor())
        ? $e
        : null
);

$getTitle    = fn (object $e): Maybe => new Maybe($e->getTitle());

$checkTitle  = fn (string $t): Maybe => new Maybe(
    str_starts_with('abc', strtolower($t))
        ? $t
        : null
);

$strToUpper  = fn (string $s): Maybe => new Maybe(strtoupper($s));

$findEntity  = fn (int $id): Maybe => new Maybe($repository->find($id));

$findEntity
    ->then($checkAuthor)
```

Draft Maybe class

```php
$checkAuthor = fn (object $e): Maybe => new Maybe(
    (1 === $e->getAuthor())
        ? $e
        : null
);

$getTitle    = fn (object $e): Maybe => new Maybe($e->getTitle());

$checkTitle  = fn (string $t): Maybe => new Maybe(
    str_starts_with('abc', strtolower($t))
        ? $t
        : null
);

$strToUpper  = fn (string $s): Maybe => new Maybe(strtoupper($s));

$findEntity  = fn (int $id): Maybe => new Maybe($repository->find($id));

$findEntity
    ->then($checkAuthor)
    ->then($getTitle)
```

Draft Maybe class

```php
$checkAuthor = fn (object $e): Maybe => new Maybe(
    (1 === $e->getAuthor())
        ? $e
        : null
);

$getTitle    = fn (object $e): Maybe => new Maybe($e->getTitle());

$checkTitle  = fn (string $t): Maybe => new Maybe(
    str_starts_with('abc', strtolower($t))
        ? $t
        : null
);

$strToUpper  = fn (string $s): Maybe => new Maybe(strtoupper($s));

$findEntity  = fn (int $id): Maybe => new Maybe($repository->find($id));

$findEntity
    ->then($checkAuthor)
    ->then($getTitle)
    ->then($checkTitle)
```

European
Commission

Draft Maybe class

```php
$checkAuthor = fn (object $e): Maybe => new Maybe(
    (1 === $e->getAuthor())
        ? $e
        : null
);

$getTitle    = fn (object $e): Maybe => new Maybe($e->getTitle());

$checkTitle  = fn (string $t): Maybe => new Maybe(
    str_starts_with('abc', strtolower($t))
        ? $t
        : null
);

$strToUpper  = fn (string $s): Maybe => new Maybe(strtoupper($s));

$findEntity  = fn (int $id): Maybe => new Maybe($repository->find($id));

$findEntity
    ->then($checkAuthor)
    ->then($getTitle)
    ->then($checkTitle)
    ->then($strToUpper)
```

European Commission

Draft Maybe class

```php
$checkAuthor = fn (object $e): Maybe => new Maybe(
    (1 === $e->getAuthor())
        ? $e
        : null
);

$getTitle    = fn (object $e): Maybe => new Maybe($e->getTitle());

$checkTitle  = fn (string $t): Maybe => new Maybe(
    str_starts_with('abc', strtolower($t))
        ? $t
        : null
);

$strToUpper  = fn (string $s): Maybe => new Maybe(strtoupper($s));

$findEntity  = fn (int $id): Maybe => new Maybe($repository->find($id));

$findEntity
    ->then($checkAuthor)
    ->then($getTitle)
    ->then($checkTitle)
    ->then($strToUpper)
    ();
```

European Commission

Functional code

The Maybe class

- This value wrapper is actually the “Maybe” monad,

Functional code
The Maybe class

- This value wrapper is actually the “Maybe” monad,
- Focus on the happy scenario,

Functional code

The Maybe class

- This value wrapper is actually the “Maybe” monad,
- Focus on the happy scenario,
- Return `null` when something goes wrong,

Functional code

The Maybe class

- This value wrapper is actually the “Maybe” monad,
- Focus on the happy scenario,
- Return `null` when something goes wrong,
- Unable to know the reason when something is wrong,

Functional code

The Maybe class

Maybe we could customize the return value in case of something wrong happen?

Functional code

The Maybe class

Maybe we could customize the return value in case of something wrong happen?

While still being able to decide what to do when an error happen?

Functional code

The Maybe class

Maybe we could customize the return value in case of something wrong happen?

While still being able to decide what to do when an error happen?

Let’s hack !

Either class direction

```
final class Either {
```

Either class direction

```php
final class Either {
    public static function left($value): Either;
    public static function right($value): Either;

```

Either class direction

```php
final class Either {
    public static function left($value): Either;
    public static function right($value): Either;

    public function __invoke(callable $ifException): mixed;

```

Either class direction

```php
final class Either {
    public static function left($value): Either;
    public static function right($value): Either;

    public function __invoke(callable $ifException): mixed;

    public function then(callable $f): self;
}
```

Either class usage

```php
$checkAuthor = fn (object $e): Either => (1 === $e->getAuthor())
    ? Either::right($e)
    : Either::left(new Exception('Invalid author'));
```

Either class usage

```php
$checkAuthor = fn (object $e): Either => (1 === $e->getAuthor())
    ? Either::right($e)
    : Either::left(new Exception('Invalid author'));

$getTitle    = fn (object $e): Either => (null === $title = $e->getTitle())
    ? Either::left(new Exception('No title has been found'))
    : Either::right($e->getTitle());
```

Either class usage

```php
$checkAuthor = fn (object $e): Either => (1 === $e->getAuthor())
    ? Either::right($e)
    : Either::left(new Exception('Invalid author'));

$getTitle    = fn (object $e): Either => (null === $title = $e->getTitle())
    ? Either::left(new Exception('No title has been found'))
    : Either::right($e->getTitle());

$checkTitle  = fn (string $t): Either => str_starts_with('abc', strtolower($t))
    ? Either::right($t)
    : Either::left(new Exception('Title does not start with abc'));
```

Either class usage

```php
$checkAuthor = fn (object $e): Either => (1 === $e->getAuthor())
    ? Either::right($e)
    : Either::left(new Exception('Invalid author'));

$getTitle    = fn (object $e): Either => (null === $title = $e->getTitle())
    ? Either::left(new Exception('No title has been found'))
    : Either::right($e->getTitle());

$checkTitle  = fn (string $t): Either => str_starts_with('abc', strtolower($t))
    ? Either::right($t)
    : Either::left(new Exception('Title does not start with abc'));

$strToUpper  = fn (string $str): Either => Either::right(strtoupper($str));
```

European Commission

Either class usage

```php
$checkAuthor = fn (object $e): Either => (1 === $e->getAuthor())
    ? Either::right($e)
    : Either::left(new Exception('Invalid author'));

$getTitle    = fn (object $e): Either => (null === $title = $e->getTitle())
    ? Either::left(new Exception('No title has been found'))
    : Either::right($e->getTitle());

$checkTitle  = fn (string $t): Either => str_starts_with('abc', strtolower($t))
    ? Either::right($t)
    : Either::left(new Exception('Title does not start with abc'));

$strToUpper  = fn (string $str): Either => Either::right(strtoupper($str));

$findEntity  = fn (int $id): Either => (null === $e = $repository->find($id))
    ? Either::left(new Exception('Unable to find entity id'))
    : Either::right($e);
```

Either class usage

```php
$checkAuthor = fn (object $e): Either => (1 === $e->getAuthor())
    ? Either::right($e)
    : Either::left(new Exception('Invalid author'));

$getTitle    = fn (object $e): Either => (null === $title = $e->getTitle())
    ? Either::left(new Exception('No title has been found'))
    : Either::right($e->getTitle());

$checkTitle  = fn (string $t): Either => str_starts_with('abc', strtolower($t))
    ? Either::right($t)
    : Either::left(new Exception('Title does not start with abc'));

$strToUpper  = fn (string $str): Either => Either::right(strtoupper($str));

$findEntity  = fn (int $id): Either => (null === $e = $repository->find($id))
    ? Either::left(new Exception('Unable to find entity id'))
    : Either::right($e);

$findEntity
```

Either class usage

```php
$checkAuthor = fn (object $e): Either => (1 === $e->getAuthor())
    ? Either::right($e)
    : Either::left(new Exception('Invalid author'));

$getTitle    = fn (object $e): Either => (null === $title = $e->getTitle())
    ? Either::left(new Exception('No title has been found'))
    : Either::right($e->getTitle());

$checkTitle  = fn (string $t): Either => str_starts_with('abc', strtolower($t))
    ? Either::right($t)
    : Either::left(new Exception('Title does not start with abc'));

$strToUpper  = fn (string $str): Either => Either::right(strtoupper($str));

$findEntity  = fn (int $id): Either => (null === $e = $repository->find($id))
    ? Either::left(new Exception('Unable to find entity id'))
    : Either::right($e);

$findEntity
    ->then($checkAuthor)
```

Either class usage

```php
$checkAuthor = fn (object $e): Either => (1 === $e->getAuthor())
    ? Either::right($e)
    : Either::left(new Exception('Invalid author'));

$getTitle    = fn (object $e): Either => (null === $title = $e->getTitle())
    ? Either::left(new Exception('No title has been found'))
    : Either::right($e->getTitle());

$checkTitle  = fn (string $t): Either => str_starts_with('abc', strtolower($t))
    ? Either::right($t)
    : Either::left(new Exception('Title does not start with abc'));

$strToUpper  = fn (string $str): Either => Either::right(strtoupper($str));

$findEntity  = fn (int $id): Either => (null === $e = $repository->find($id))
    ? Either::left(new Exception('Unable to find entity id'))
    : Either::right($e);

$findEntity
    ->then($checkAuthor)
    ->then($getTitle)
```

Either class usage

```php
$checkAuthor = fn (object $e): Either => (1 === $e->getAuthor())
    ? Either::right($e)
    : Either::left(new Exception('Invalid author'));

$getTitle    = fn (object $e): Either => (null === $title = $e->getTitle())
    ? Either::left(new Exception('No title has been found'))
    : Either::right($e->getTitle());

$checkTitle  = fn (string $t): Either => str_starts_with('abc', strtolower($t))
    ? Either::right($t)
    : Either::left(new Exception('Title does not start with abc'));

$strToUpper  = fn (string $str): Either => Either::right(strtoupper($str));

$findEntity  = fn (int $id): Either => (null === $e = $repository->find($id))
    ? Either::left(new Exception('Unable to find entity id'))
    : Either::right($e);

$findEntity
    ->then($checkAuthor)
    ->then($getTitle)
    ->then($checkTitle)
```

European Commission

Either class usage

```php
$checkAuthor = fn (object $e): Either => (1 === $e->getAuthor())
    ? Either::right($e)
    : Either::left(new Exception('Invalid author'));

$getTitle    = fn (object $e): Either => (null === $title = $e->getTitle())
    ? Either::left(new Exception('No title has been found'))
    : Either::right($e->getTitle());

$checkTitle  = fn (string $t): Either => str_starts_with('abc', strtolower($t))
    ? Either::right($t)
    : Either::left(new Exception('Title does not start with abc'));

$strToUpper  = fn (string $str): Either => Either::right(strtoupper($str));

$findEntity  = fn (int $id): Either => (null === $e = $repository->find($id))
    ? Either::left(new Exception('Unable to find entity id'))
    : Either::right($e);

$findEntity
    ->then($checkAuthor)
    ->then($getTitle)
    ->then($checkTitle)
    ->then($strToUpper)
```

Either class usage

```php
$checkAuthor = fn (object $e): Either => (1 === $e->getAuthor())
    ? Either::right($e)
    : Either::left(new Exception('Invalid author'));

$getTitle    = fn (object $e): Either => (null === $title = $e->getTitle())
    ? Either::left(new Exception('No title has been found'))
    : Either::right($e->getTitle());

$checkTitle  = fn (string $t): Either => str_starts_with('abc', strtolower($t))
    ? Either::right($t)
    : Either::left(new Exception('Title does not start with abc'));

$strToUpper  = fn (string $str): Either => Either::right(strtoupper($str));

$findEntity  = fn (int $id): Either => (null === $e = $repository->find($id))
    ? Either::left(new Exception('Unable to find entity id'))
    : Either::right($e);

$findEntity
    ->then($checkAuthor)
    ->then($getTitle)
    ->then($checkTitle)
    ->then($strToUpper)
    (
        static fn (Exception $e) => throw $e
    );
```

Functional code

The Either class

- This value wrapper is actually the “Either” monad,

Functional code

The Either class

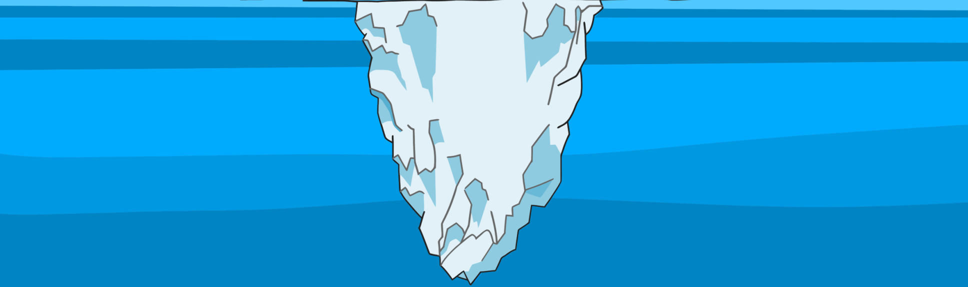
- This value wrapper is actually the “Either” monad,
- Focus on the happy scenario,

Functional code

The Either class

- This value wrapper is actually the “Either” monad,
- Focus on the happy scenario,
- The user decide what to do when something goes wrong.

So these are monads !



Technically, a monad is a monoid in the monoidal category of endofunctors equipped with functor composition as its product.

Should I use monads ?

Functional code

State of PHP and monads

- No core implementation,

Functional code

State of PHP and monads

- No core implementation,
- Only “userland” implementations,

Functional code

State of PHP and monads

- No core implementation,
- Only “userland” implementations,
- Prone to static analysis,

Functional code

State of PHP and monads

- No core implementation,
- Only “userland” implementations,
- Prone to static analysis,
- Implementations:

Functional code

State of PHP and monads

- No core implementation,
- Only “userland” implementations,
- Prone to static analysis,
- Implementations:
    - github.com/marcosh/lamphpda

European Commission

Functional code

State of PHP and monads

- No core implementation,
- Only “userland” implementations,
- Prone to static analysis,
- Implementations:
  - github.com/marcosh/lamphpda
  - github.com/loophp/repository-monadic-helper/

Functional code

Monads summary

- Introduced in 1988 by Eugenio Moggi

Functional code

Monads summary

- Introduced in 1988 by Eugenio Moggi
- Maybe: Deal with a value and null

Functional code

Monads summary

- Introduced in 1988 by Eugenio Moggi
- Maybe: Deal with a value and null
- Either: Deal with a value and another value
  Maybe is to some extent a specialization of Either.

Monads summary

- Introduced in 1988 by Eugenio Moggi
- Maybe: Deal with a value and null
- Either: Deal with a value and another value
  Maybe is to some extent a specialization of Either.
- ...Reader, Writer, IO, List,...
  There are many other monads these two are just the tip of the iceberg.

Functional code

Monads summary

- Introduced in 1988 by Eugenio Moggi
- Maybe: Deal with a value and null
- Either: Deal with a value and another value
  Maybe is to some extent a specialization of Either.
- ...Reader, Writer, IO, List,...
  There are many other monads these two are just the tip of the iceberg.
- Monads may have different names in languages
  (Optional, Result, Promise, etc etc)

Functional code
Monads summary

- Introduced in 1988 by Eugenio Moggi
- Maybe: Deal with a value and null
- Either: Deal with a value and another value
  Maybe is to some extent a specialization of Either.
- ...Reader, Writer, IO, List,...
  There are many other monads these two are just the tip of the iceberg.
- Monads may have different names in languages
  (Optional, Result, Promise, etc etc)
- Monads are the design pattern of functional languages

European Commission

Functional code

Monads summary

- Introduced in 1988 by Eugenio Moggi
- Maybe: Deal with a value and null
- Either: Deal with a value and another value
  Maybe is to some extent a specialization of Either.
- ...Reader, Writer, IO, List,...
  There are many other monads these two are just the tip of the iceberg.
- Monads may have different names in languages
  (Optional, Result, Promise, etc etc)
- Monads are the design pattern of functional languages
- Underlying theory can be very complex

European Commission

Functional code
Monads summary

- Introduced in 1988 by Eugenio Moggi
- Maybe: Deal with a value and null
- Either: Deal with a value and another value
  Maybe is to some extent a specialization of Either.
- ...Reader, Writer, IO, List,...
  There are many other monads these two are just the tip of the iceberg.
- Monads may have different names in languages
  (Optional, Result, Promise, etc etc)
- Monads are the design pattern of functional languages
- Underlying theory can be very complex
- **We don't need to understand them to use them !**

Should I use monads ?

Functional code

Monads benefits

- Monads can abstract most of your function binding...
  Turning a lot of noise into magic!

Functional code

Monads benefits

- Monads can abstract most of your function binding...
  Turning a lot of noise into magic!
- It modularizes logical unit of computation and facilitate refactoring

Functional code

Monads benefits

- Monads can abstract most of your function binding...
  Turning a lot of noise into magic!
- It modularizes logical unit of computation and facilitate refactoring
- Reduce complexity of the code
  abstracting the logic into one well known object that does only one thing and well

European Commission

Functional code

Monads benefits

- Monads can abstract most of your function binding...
  Turning a lot of noise into magic!
- It modularizes logical unit of computation and facilitate refactoring
- Reduce complexity of the code
  abstracting the logic into one well known object that does only one thing and well
- For educational purposes ?
  Demystifing monads can help understand the benefits and perks of functional programming, and make you a better developer

European Commission

Thank you all for listening!

European Commission