The PHP Foundation

Une Monade, simplement, c’est quoi ?

Gina Peter Banyard

11 Octobre 2024

À propos

Core développeuse PHP

BSc math pure

Mastodon

`@Girgias@phpc.social`

Site

https://gpb.moe

L’essence de la programation fonctionelle ?

L'essence de la programation fonctionelle

Fonctions pures

- $\Longrightarrow$ Aucun effets de bord
- $\Longrightarrow$ Résultat identique pour paramètres identiques

L'essence de la programation fonctionelle

Fonctions pures

- ⟹ Aucun effets de bord
- ⟹ Résultat identique pour paramètres identiques

Variables immuables

- ⟹ Pas de boucle
- ⟹ Fonction récursive


État

L'essence de la programation fonctionelle

```
function kinetic_E(float $mass_kg, float $speed_mps) {
  return 1/2 * $mass_kg * $speed_mps**2;
}
```

L'essence de la programation fonctionelle

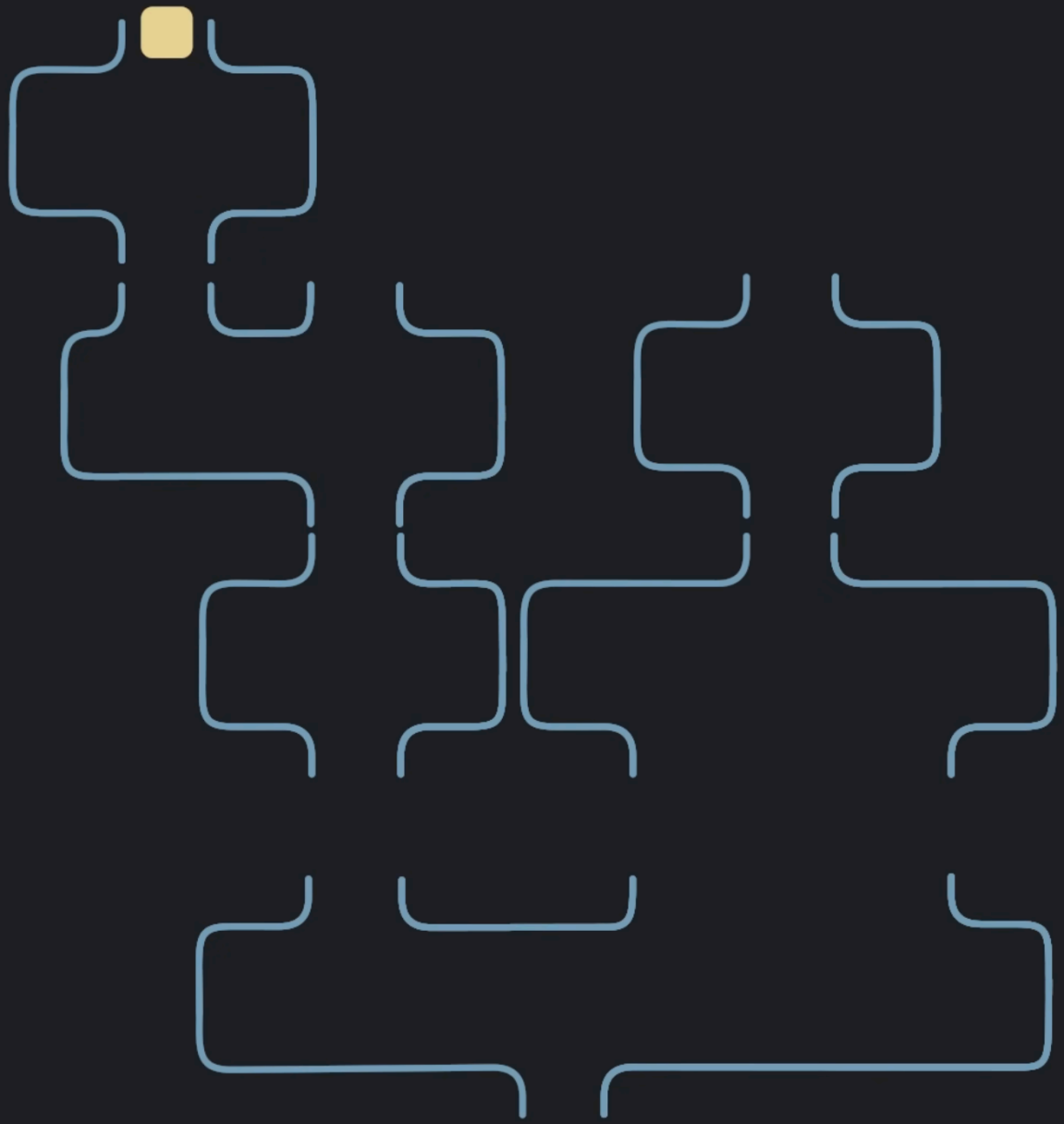
```
function kinetic_E(float $mass_kg, float $speed_mps) {
  return 1/2 * $mass_kg * $speed_mps**2;
}
```

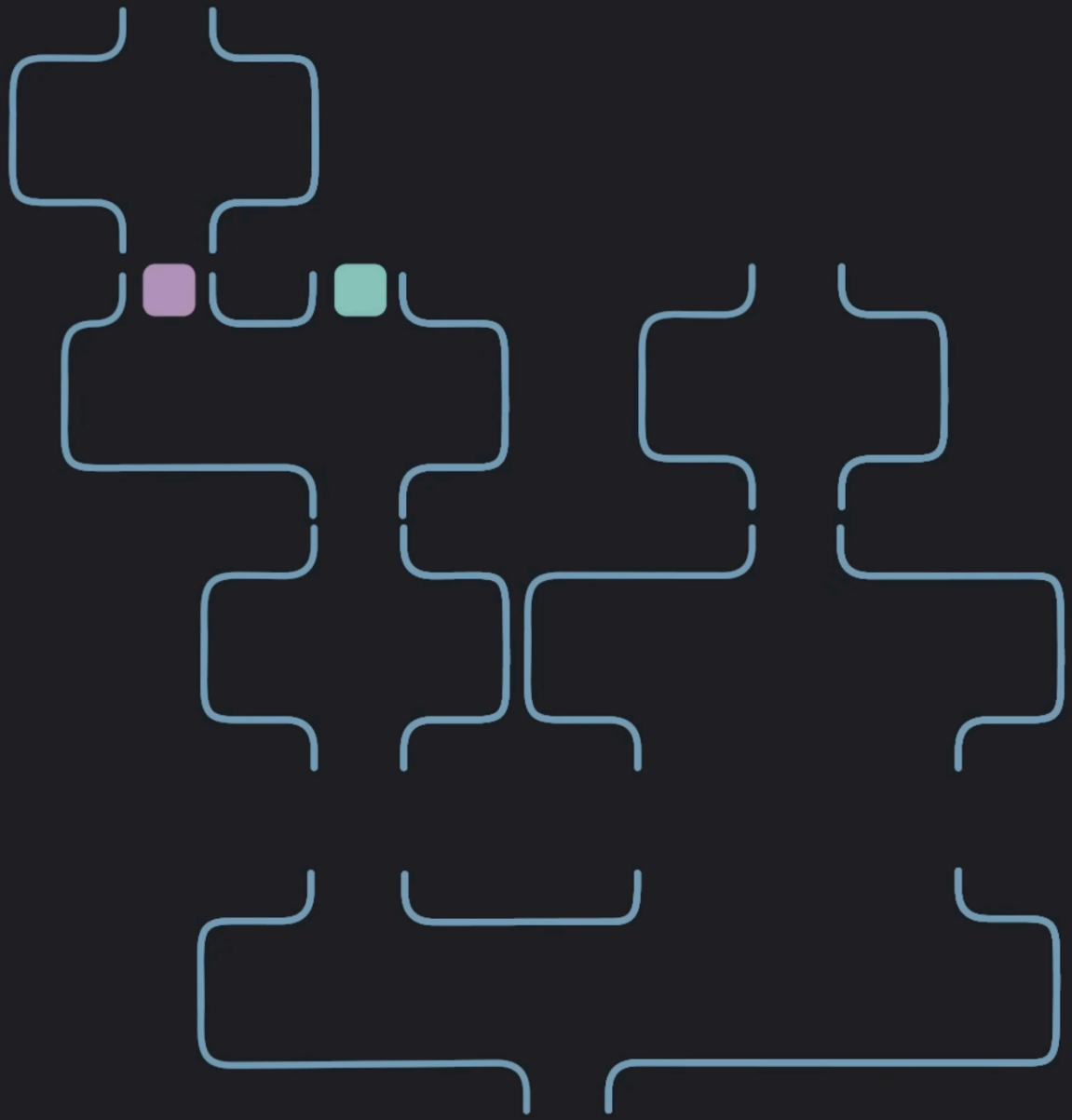
Application partielle:

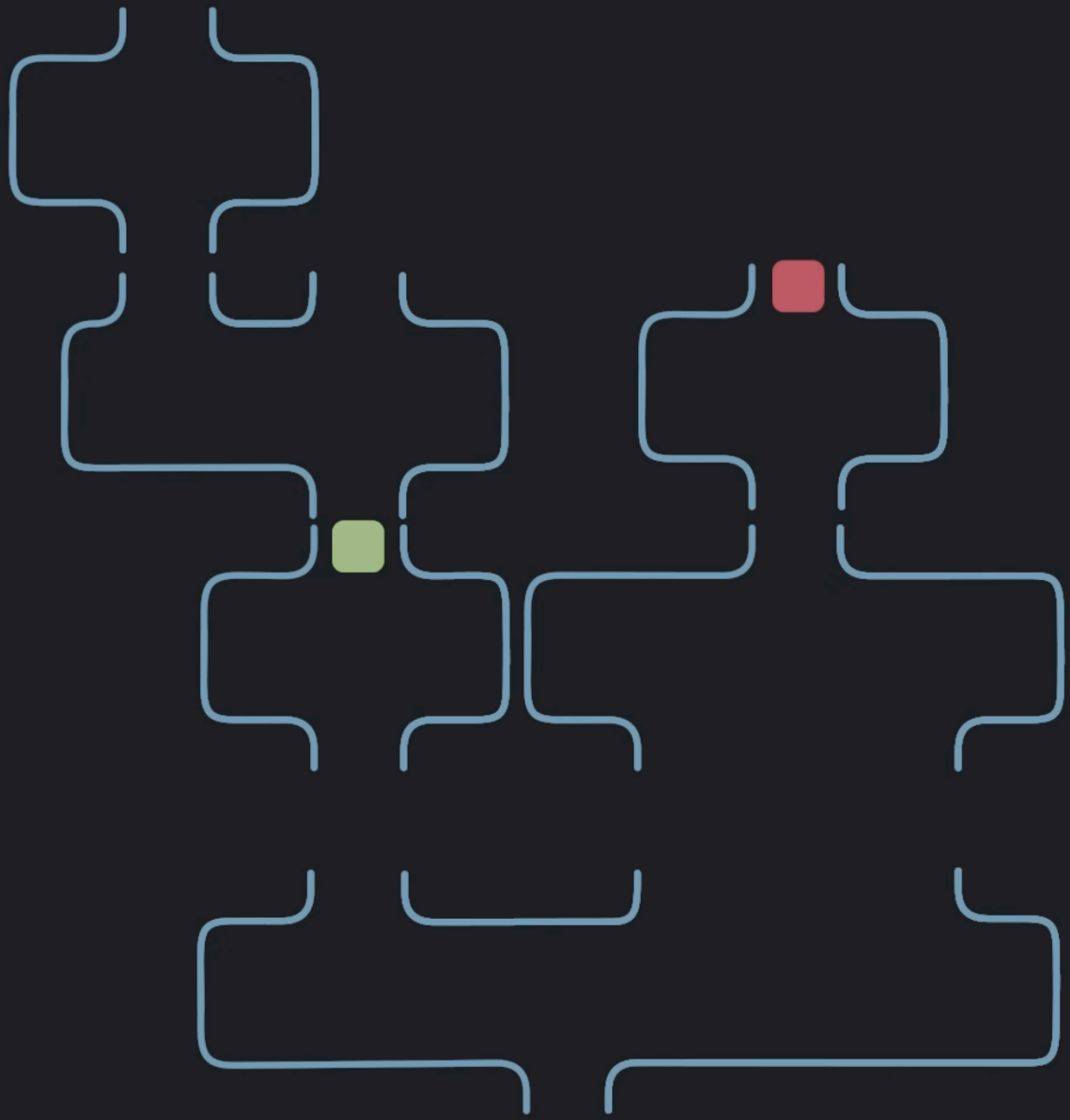
```
$kE_TGV = fn (float $speed) => kinetic_E(400000, $speed);
```

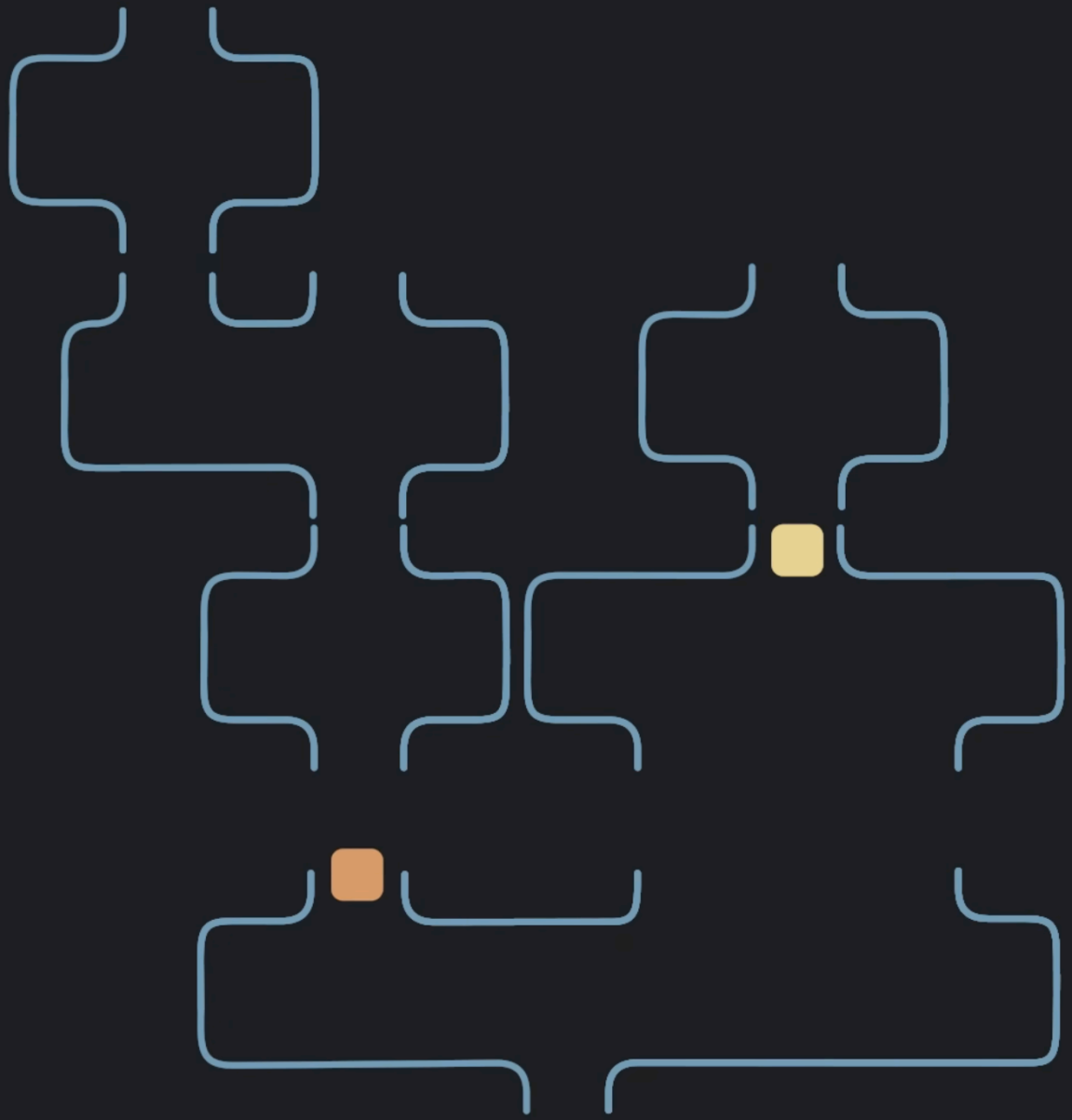
Correspond à l'Injection de Dépendances pour les fonctions.

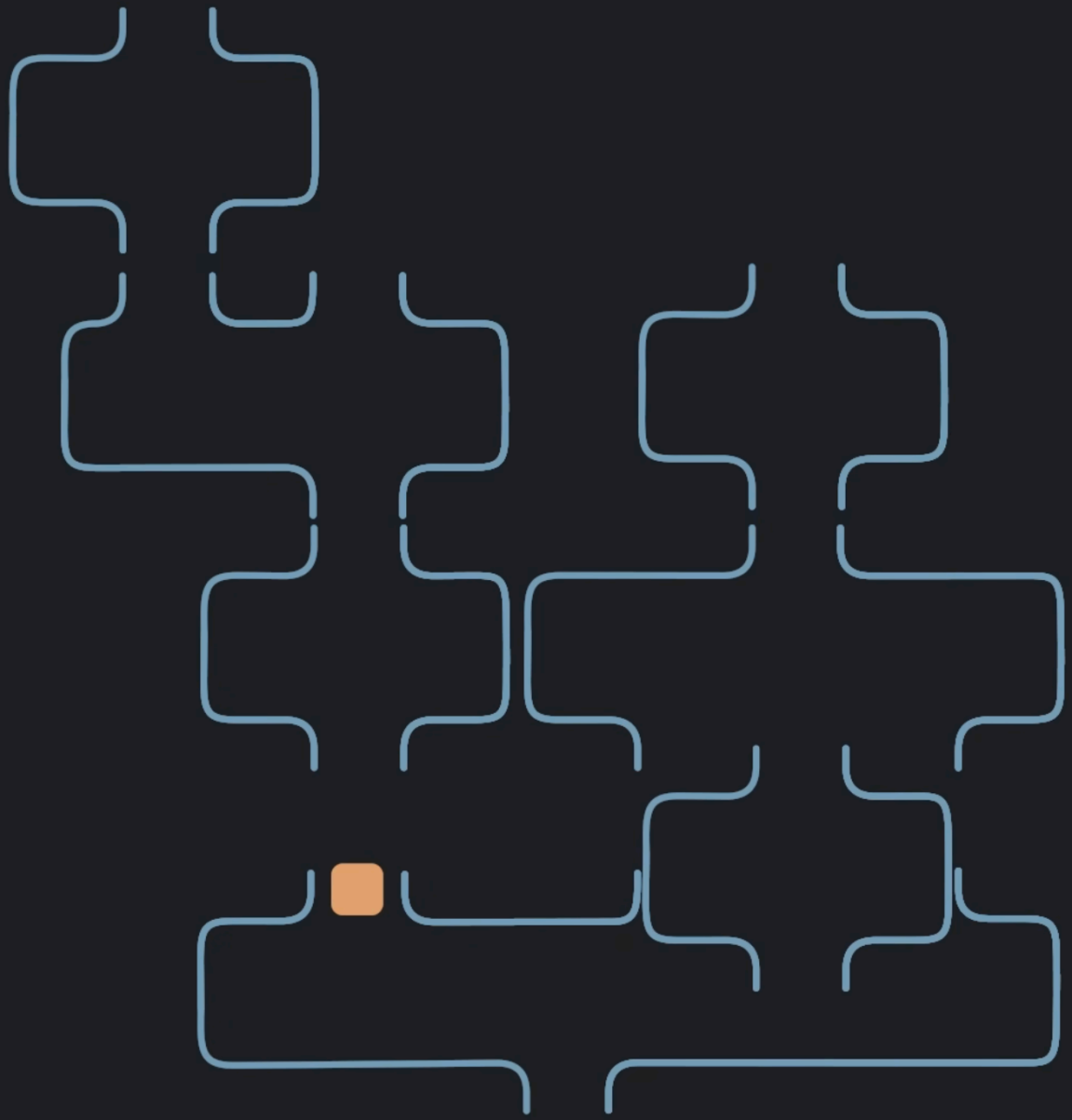
Dans un monde idéal

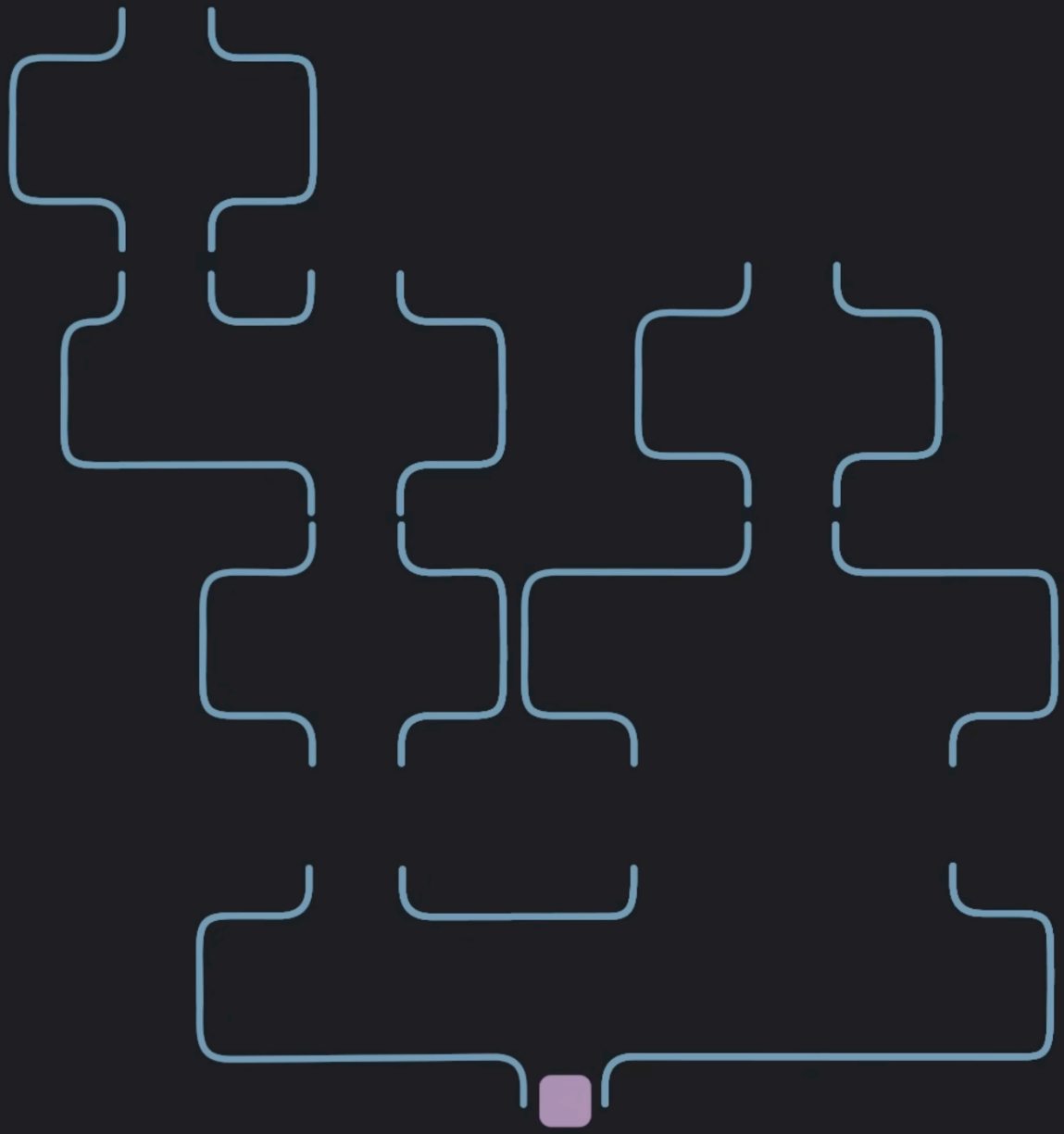












Comment avoir de l'*état* ou des *effets de bords* avec des fonctions pures et valeurs immuable ?

Comment avoir de l'*état* ou des *effets de bords* avec des fonctions pures et valeurs immuable ?

En utilisant une **monade** !

Une Monade c'est quoi ?

Une **monade** est une construction catégorique qui est la donnée d'un triplet (T, η, μ) :

- T un *endo-foncteur* d'une catégorie $T : \mathcal{C} \to \mathcal{C}$
- η une *transformation naturelle* $\eta : 1_{\mathcal{C}} \to T$
  avec $1_{\mathcal{C}}$ le foncteur identité de $1_{\mathcal{C}}$
- μ une *transformation naturelle* $\mu : T \circ T \to T$

Une Monade, en programmation, c'est quoi ?

En programation fonctionelle, une **monade** est une structure qui combine des *fonctions* et englobe leurs valeurs de retour dans un *type* avec des effets additionels.

Les monades définissent deux **opérateurs**, une pour *envelopper* une valeur dans le type monadique, et une autre pour *composer* des fonctions qui retournent des valeurs de ce type monadique.

Et en pratique ?

Monad $M\langle T \rangle$

1. *unit* ou *return*

   ```
   function unit(T $v): M<T>
   ```

2. *bind*

   ```
   function bind(callable $fn): M<S>
   $fn := function(T $v): M<S>
   ```

Exemples

Monade Maybe, l'exemple classique

```
final class Maybe<T> {
  private bool $set = false;
  private function __construct(readonly T $data) { $this->set = true; }
  public function static unit(T $data): Maybe<T> {
    return new Maybe($data);
  }
  public function static Nothing(): Maybe<T> { return new Maybe(); }
  public function isNothing(): bool { return !$this->set; }
  public function  bind(callable $fn): Maybe {
    return $this->isNothing() ? $this : $fn($this->data);
  }
}
```

Monade Maybe en action

```
final class Customer {}
final class Subscription {}
final class Invoice {}
final class Charge {}

function get_subscription(Customer $c): Maybe<Subscription> {}
function latest_invoice(Subscription $s): Maybe<Invoice> {}
function invoice_charge(Invoice $i): Maybe<Charge> {}
function is_charge_disputed(Charge $c): bool {}
```

Monade Maybe en action

```
/* $c := Customer */
$m = get_subscription($c)
  ->bind(latest_invoice(...))
  ->bind(invoice_charge(...))
;

var_dump(
  $m->isNothing() ?
    "Nothing" :
    is_charge_disputed($m->data)
);
```

Mais pourquoi pas ?

```
final class Customer {
  public function getSubscription(): ?Subscription {}
}
final class Subscription {
  public function latestInvoice(): ?Invoice {}
}
final class Invoice {
  public function getCharge(): ?Charge
}
final class Charge {
  public function isDisputed(): bool
}
```

Mais pourquoi pas ?

```php
/* $c := Customer */
$r = $c->getSubscription()
  ?->latestInvoice()
  ?->getCharge()
  ->isDisputed()
;

var_dump($r);
```

La monade Either

Either $E\langle L, R\rangle$ englobe 2 valeurs

$$f : R_1 \to E\langle L_2, R_2\rangle$$

$$bind(f) : E\langle L_1, R_1\rangle \to E\langle L_1 | L_2, R_2\rangle$$

La monade Either

```php
final class Either<L, R> {
  private function __construct(readonly L $l, readonly R $r) {}
  static function left(L $data): Either { return new Either(l: $data); }
  static function right(R $data): Either { return new Either(r: $data); }
  public function isRight(): bool {}

  public function bind(callable $fn): Either {
    return $this->isRight() ? $fn($this->right) : $this;
  }
}
```

La monade Either en action

```php
final class File {}
final class TabularData {}
enum OpenFileErrors {
    case FileDoesNotExist;
    case AccessDenied;
    case IsDirectory;
}
/** @return Either<OpenFileErrors, File> */
function open_file(string $path): Either {
  return Either::left(OpenFileErrors::FileDoesNotExist);
  return Either::right(new File());
}
```

```php
enum GetContentErrors {
    case FileNotReadable;
}
enum ParseToCsvErrors {
    case UnexpectedEof;
    case InconsistentLineFields;
}
```

La monade Either en action

```php
/** @return Either<GetContentErrors, string> */
function get_content_file(File $f): Either {
  return Either::left(GetContentErrors::FileNotReadable);
  return Either::right("Content of the file");
}

/** @return Either<ParseToCsvErrors, TabularData> */
function parse_content_into_tabular(string $content): Either {
  return Either::left(ParseToCsvErrors::InconsistentLineFields);
  return Either::right(new TabularData());
}
```

La monade Either en action

```
$data = open_file('my_csv.txt')
    ->bind(get_content_file(...))
    ->bind(parse_content_into_tabular(...));
if ($data->isRight()) { /* Use tabular data */ } else {
  $errorCode = match ($data->left) {
    OpenFileErrors::FileDoesNotExist => 1,
    OpenFileErrors::AccessDenied => 2,
    OpenFileErrors::IsDirectory => 3,
    GetContentErrors::FileNotReadable => 4,
    ParseToCsvErrors::UnexpectedEof => 5,
    ParseToCsvErrors::InconsistentLineFields => 6,
  };
}
```

La monade Either : génerique

$\mathrm{Either}\langle L, R\rangle$ englobe un type union $L \mid R$.

$\mathrm{Maybe}\langle T\rangle := \mathrm{Either}\langle \mathrm{null}, T\rangle$

$\mathrm{Exception}\langle T\rangle := \mathrm{Either}\langle \mathrm{string}, T\rangle$

Monades = gestion d'erreurs ?

Monades ≠ ~~gestion d'erreurs~~ !

La monade List

La monade List

$$f : X \to Y$$

$$g : Y \to Z$$

$$\mathrm{map}(f \circ g) = \mathrm{map}(f) \circ \mathrm{map}(g)$$

La monade List

$$f : X \to Y$$

$$g : Y \to Z$$

$$\mathrm{map}(f \circ g) = \mathrm{map}(f) \circ \mathrm{map}(g)$$

2 transformations naturelles :

unit $\eta : T \to M\langle T \rangle$

join $\mu : M\langle M\langle T \rangle \rangle \to M\langle T \rangle$

La monade List

Map `array_map()`

Unit
```php
return [$value];
```

Join
```php
/** @param list<list<T>> $a */
/** @return list<T> */
function concat_lists(array $a): array {
    return array_merge(...$a);
}
```

La monade List

```
function bind(callable $fn): Closure {
    return fn (array $a) => concat_lists(
        array_map($fn, $a)
    );
}
```

La monade List

```php
function suggested_products(string $product) {
    return [
        'fancy ' . $product,
        'budget ' . $product,
        'brandA ' . $product,
    ];
}

$products = ['shampoo', 'cereals', 'book'];
var_dump(bind(suggested_products(...))($products));
```

The List monad in action

```
array(9) {
  [0]=> string(13) "fancy shampoo"
  [1]=> string(14) "budget shampoo"
  [2]=> string(14) "brandA shampoo"
  [3]=> string(13) "fancy cereals"
  [4]=> string(14) "budget cereals"
  [5]=> string(14) "brandA cereals"
  [6]=> string(10) "fancy book"
  [7]=> string(11) "budget book"
  [8]=> string(11) "brandA book"
}
```

La monade Logger en action

```
final class LoggerMonad {
    public function __construct(
      public mixed $data, public array $logs = []) {}
    public function bind(callable $fn) {
        $resultLoggerMonad = $fn($this->data);
        return new LoggerMonad(
            $resultLoggerMonad->data,
            [...$this->logs, ...$resultLoggerMonad->logs],
        );
    }
}
```

La monade Logger en action

```
function loggify(callable $fn): Closure {
  return function ($value) use ($fn) {
    $name = (new ReflectionFunction($fn))->name;
    $log = [
      'Running '. $name .'('. var_export($value, true) .')'
    ];
    return new LoggerMonad($fn($value), $log);
  };
}
```

La monade Logger en action

```
function add2($v)                          function mul3($v)
function square($v)                        function div6($v)

function log_calls($value, callable ...$fns) {
    $logging_fns = array_map(loggify(...), $fns);
    $monad = new LoggerMonad($value);
    foreach ($logging_fns as $fn) {
        $monad = $monad->bind($fn);
    }
    return $monad;
}
```

La monade Logger en action

```php
$m = log_calls(
    3,
    add2(...),
    square(...),
    mul3(...),
    div6(...),
);

echo 'Value is: '. $m->data
  ."\n". join($m->logs, "\n");
```

```
Value is: 12.5
Running add2(3)
Running square(5)
Running mul3(25)
Running div6(75)
```

Pour aller plus loin

- Monades Reader, Nondeterminism, Probability, IO, etc.
- Transformateur de Monades
- Théorie des catégories
- Co-monoades
- Effet Algébrique (e.g. Frank)

Merci

Mastodon

`@Girgias@phpc.social`

Site

https://gpb.moe

Des questions?

References

- Functional error handling with monads, monad transformers and Cats MTL
- Monade (théorie des catégories)
- Monad (category theory)
- Monade (informatique)
- "Dear Functional Bros", by CodeAesthetic
- exception monad on nLab