open social

Drupaljam:XL 2019

Multilingualism makes better programmers - A look beyond PHP and JavaScript

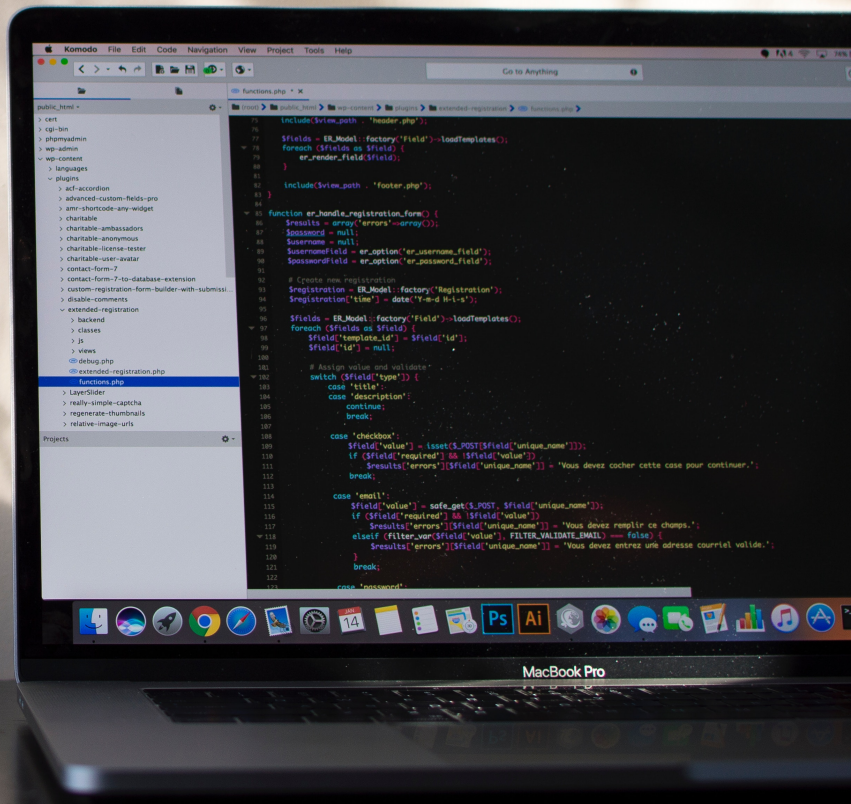


Photo by Émile Perron on Unsplash

describe('Alexander Varwijk')

- Programming for 15+ years
- 7 years of Drupal
- Developer for Open Social
- Love for sports, beer and pretty code

About this talk

- A look outside of our comfort zone
- Not every example works in PHP/JS
  - Concepts/Thought process is still useful
- Examples from Ruby, Java, Rust, Javascript, Perl

What's in a (programming) language?

Syntax

> The arrangement of words and phrases
> to create well-formed sentences in a language

> The structure of statements in a computer language

(Standard) Library

> A standard library in computer programming
> is the library made available across implementations
> of a programming language.

What can I use without installing something else?

“Ruby, Ruby, Ruby, Ruby”

— Kaiser Chiefs

The Ruby Styleguide

A community driven effort to create standards
Applicable beyond Ruby

rubocop-hq/ruby-style-guide

Keep functions short

> Avoid methods longer than 10 LOC (lines of code).
> Ideally, most methods will be shorter than 5 LOC.
> Empty lines do not contribute to the relevant LOC.

Keep functions short

- Give functions a single purpose
- Reduce mental load
- Makes it easier to put the code in presentations ;-)

- Find the right balance

```php
function social_post_form_post_form_alter(&$form, FormStateInterface $form_state, $form_id) {
  // Show the users picture for new posts.
  if ($form_state->getFormObject()->getEntity()->isNew()) {
    // Load current user.
    $account = User::load(Drupal::currentUser()->id());
    // Load compact notification view mode of the attached profile.
    if ($account) {
      $storage = \Drupal::entityTypeManager()->getStorage('profile');
      $user_profile = $storage->loadByUser($account, 'profile');
      if ($user_profile) {
        $content = \Drupal::entityTypeManager()->getViewBuilder('profile')
          ->view($user_profile, 'compact_notification');
        // Add to a new field, so twig can render it.
        $form['current_user_image'] = $content;
      }
    }
  }
  // [Snip]
}
```

```php
function social_post_form_post_form_alter(&$form, FormStateInterface $form_state, $form_id) {
  // Show the users picture for new posts.
  if ($form_state->getFormObject()->getEntity()->isNew()) {
    $user_image = social_post_get_current_user_image();
    if ($user_image) {
      // Add to a new field, so twig can render it.
      $form['current_user_image'] = $user_image;
    }
  }
  // [Snip]
}
```

`if` as modifier

> Prefer modifier `if` / `unless` usage when you have a single-line body. Another good alternative is the usage of control flow `&&` / `||`.

```
# good (doesn't work in PHP/JS)
do_something if some_condition

# another good option
some_condition && do_something
```

PHP

```php
function p($x) { echo $x; }
function ret_true() {
  echo "true ";
  return true;
}
function ret_false() {
  echo "false ";
  return false;
}
function nl() { echo "\n"; }

ret_true()  && p("&& print"); nl();
ret_true()  || p("|| print"); nl();
ret_false() && p("&& print"); nl();
ret_false() || p("|| print"); nl();
```

Javascript

```javascript
function p(x) { process.stdout.write(x); }
function ret_true() {
  p("true ");
  return true;
}
function ret_false() {
  p("false ");
  return false;
}
function nl() { p("\n"); }

ret_true()  && p("&& print"); nl();
ret_true()  || p("|| print"); nl();
ret_false() && p("&& print"); nl();
ret_false() || p("|| print"); nl();
```

Output

```
true && print
true
false
false || print
```

No `method_missing`

> Avoid using `method_missing` for metaprogramming because backtraces become messy, the behavior is not listed in `#methods`, and misspelled method calls might silently work, e.g. `nukes.launch_state = false`.

```php
/** Implements the magic method for getting object properties. */
public function &__get($name) {
  // If this is an entity field, handle it accordingly. We first check whether
  // a field object has been already created. If not, we create one.
  if (isset($this->fields[$name][$this->activeLangcode])) {
    return $this->fields[$name][$this->activeLangcode];
  }
  // Inline getFieldDefinition() to speed things up.
  if (!isset($this->fieldDefinitions)) { $this->getFieldDefinitions(); }
  if (isset($this->fieldDefinitions[$name])) {
    $return = $this->getTranslatedField($name, $this->activeLangcode);
    return $return;
  }
  // Else directly read/write plain values. That way, non-field entity
  // properties can always be accessed directly.
  if (!isset($this->values[$name])) { $this->values[$name] = NULL; }
  return $this->values[$name];
}
```

```
@todo: A lot of code still uses non-fields (e.g. $entity->content in view
       builders) by reference. Clean that up.
```

Javascript Proxy

- Emulates PHP's magic methods
- MDN Documentation
- Could be a talk on its own
- Doesn't work in Internet Explorer

Java

open social

Annotations

> In the Java computer programming language, an annotation is a form of syntactic metadata that can be added to Java source code.
>
> — Wikipedia, Java_annotation

```
package com.example.examplemod;
import net.minecraft.init.Blocks;
import net.minecraftforge.fml.common.Mod;
import net.minecraftforge.fml.common.Mod.EventHandler;
import net.minecraftforge.fml.common.event.FMLInitializationEvent;

@Mod(modid = ExampleMod.MODID, version = ExampleMod.VERSION)
public class ExampleMod {
    public static final String MODID = "examplemod";
    public static final String VERSION = "1.0";

    @EventHandler
    public void init(FMLInitializationEvent event) {
        // some example code
        System.out.println("DIRT BLOCK >> "+Blocks.dirt.getUnlocalizedName());
    }
}
```

```php
namespace Drupal\image\Plugin\Field\FieldWidget;

/**
 * Plugin implementation of the 'image_image' widget.
 *
 * @FieldWidget(
 *   id = "image_image",
 *   label = @Translation("Image"),
 *   field_types = {
 *     "image"
 *   }
 * )
 */
class ImageWidget extends FileWidget {
  // Body omitted
}
```

PHP Annotations

- Not built into the language

- Uses the `doctrine/annotations` package
- Uses PHP `Reflection` to read object/method/property comments

- Proposal to move annotations into the PHP runtime (out of comments)
  See https://wiki.php.net/rfc/annotations_v2

Javascript ~~Annotations~~ Decorators

- Are used to modify the behaviour of a class or its properties
- Are currently a stage 2 proposal
  - https://github.com/tc39/proposal-decorators

- Have changed drastically (March 2019)
  since I last presented on the subject

```
import { @set } from "./set.mjs";
import { @tracked } from "./tracked.mjs";
import { @bound } from "./bound.mjs";
import { @defineElement } from "./defineElement.mjs";

@defineElement('counter-widget')
class CounterWidget extends HTMLElement {
  @tracked x = 0;

  @set onclick = this.clicked;

  @bound clicked() { this.x++; }

  connectedCallback() { this.render(); }

  render() { this.textContent = this.x.toString(); }
}
// Source: https://github.com/tc39/proposal-decorators
```

Rust

Strongly Typed

> In a strongly typed language each data area will have a distinct type and each process will state its communication requirements in terms of these types.

— Jackson, K. (1977).
Design and Implementation of
Programming Languages.

Strongly Typed PHP

- 5.0 - Parameter type hints for Interface and Class names
- 5.1 - Parameter type hints for arrays
- 5.4 - Parameter type hints for callables
- 7.0 - Parameter type hints for all primitive types, return type hints
- 7.1 - Nullable types
- 7.4 - Type hints for class properties

```php
/* An example of PHP 7.4's type hints. */
class TypeHintsExample {
  protected string $chant = 'PHP is improving!';

  public function setChant(string $newChant) {
    $this->chant = $newChant;
  }

  public function chant() {
    echo $this->chant . PHP_EOL;
  }

  public function isAwesome() : bool {
    return true;
  }
}
```

```php
/* An example of PHP 7.4's type hints. */
class TypeHintsExample {
  protected string $chant = 'PHP is improving!';

  public function setChant(string $newChant) {
    $this->chant = $newChant;
  }

  public function chant() {
    echo $this->chant . PHP_EOL;
  }

  public function isAwesome() : bool {
    return true;
  }
}
```

```php
$fan = new TypeHintsExample();
$fan->setChant('PHP is improving!');

// Outputs: "PHP is improving!\n"
$fan->chant();


$fan->setChant(9001);
// Outputs: "9001\n"
$fan->chant();
```

```php
// Required to actually trigger type errors.
declare(strict_types = 1);

/* An example of PHP 7.4's type hints. */
class TypeHintsExample {
  protected string $chant = 'PHP is improving!';

  public function setChant(string $newChant) {
    $this->chant = $newChant;
  }

  public function chant() {
    echo $this->chant . PHP_EOL;
  }

  public function isAwesome() : bool {
    return true;
  }
}
```

```php
$fan = new TypeHintsExample();
$fan->setChant('PHP is improving!');

// Outputs: "PHP is improving!\n"
$fan->chant();

// Throws a TypeError
$fan->setChant(9001);

$fan->chant();
```

Strongly Typed Drupal

- Parameter type hints in coding standards: Issue #1158720
- Standardising array type hints (11 years and counting): Issue #318016
- Discussion about implementing types in existing code: Issue #3050720
  - Highlights some backwards compatibility issues, some solved in PHP 7.2
- Discussion on coding standard for new code: Issue #3060914

Strongly Typed Javascript

(There’s Typescript)

Pattern Matching

> Patterns are a special syntax in Rust for matching against the structure of types, both complex and simple.

```rust
let x = 1;

match x {
    1 => println!("one"),
    2 => println!("two"),
    3 => println!("three"),
    _ => println!("anything"),
}

// Source: https://doc.rust-lang.org/1.5.0/book/patterns.html
```

```php
// PHP
switch ($x) {
  case 1:
    echo "one";
    break;
  case 2:
    echo "two";
    break;
  case 3:
    echo "three";
    break;
  default:
    echo "anything";
}
```

```javascript
// Javascript
switch (x) {
  case 1:
    console.log("one");
    break;
  case 2:
    console.log("two");
    break;
  case 3:
    console.log("three");
    break;
  default:
    console.log("anything");
}
```

```
match x {
    e @ 1 ... 5 => println!("got a range element {}", e),
    i => println!("{} was not in our range", i),
}
```

```
match x {
    e @ 1 ... 5 => println!("got a range element {}", e),
    i => println!("{} was not in our range", i),
}

enum OptionalInt {
    Value(i32),
    Missing,
}

let x = OptionalInt::Value(5);

match x {
    OptionalInt::Value(i) if i > 5 => println!("Got an int bigger than five!"),
    OptionalInt::Value(..) => println!("Got an int!"),
    OptionalInt::Missing => println!("No such luck."),
}

// Source: https://doc.rust-lang.org/1.5.0/book/patterns.html
```

Enum

> The `enum` keyword allows the creation of a type which may be one of a few different variants.

```php
namespace Drupal\Core\Language;
/**
 * Defines a language.
 */
interface LanguageInterface {
  // [Snip]

  /**
   * Language written left to right.
   * Possible value of $language->direction.
   */
  const DIRECTION_LTR = 'ltr';

  /**
   * Language written right to left.
   * Possible value of $language->direction.
   */
  const DIRECTION_RTL = 'rtl';
```

```php
  /**
   * Gets the text direction (left-to-right or
   * right-to-left).
   *
   * @return string
   *   Either self::DIRECTION_LTR
   *   or self::DIRECTION_RTL.
   */
  public function getDirection();

  // [Snip]
}
```

```php
/**
 * Defines a language.
 */
interface LanguageInterface {
  /**
   * Gets the text direction (left-to-right or
   * right-to-left).
   *
   * @return string
   *   Either self::DIRECTION_LTR
   *   or self::DIRECTION_RTL.
   */
  public function getDirection();
}
```

```php
class Gibberish implements LanguageInterface {
  // Required methods omitted for brevity.

  public function getDirection() {
    return 'outside-in';
  }
}
```

```php
/**
 * Defines a language.
 */
interface LanguageInterface {
  /**
   * Gets the text direction (left-to-right or
   * right-to-left).
   *
   * @return string
   *   Either self::DIRECTION_LTR
   *   or self::DIRECTION_RTL.
   */
  public function getDirection();
}
```

```rust
// Create an `enum` to classify a web event. Note how both
// names and type information together specify the variant:
// `PageLoad != PageUnload` and `KeyPress(char) != Paste(String)`.
// Each is different and independent.
enum WebEvent {
    // An `enum` may either be `unit-like`,
    PageLoad,
    PageUnload,
    // like tuple structs,
    KeyPress(char),
    Paste(String),
    // or like structures.
    Click { x: i64, y: i64 },
}

// Source https://doc.rust-lang.org/stable/rust-by-example/custom_types/enum.html
```

NULL does not exist

> Error: Call to a member function someFunction() on null

```
/**
 * Loads an entity.
 *
 * @param mixed $id
 *   The id of the entity to load.
 *
 * @return static
 *   The entity object or NULL if there is no entity with the given ID.
 */
```

```
// Given we have < 9000 nodes.
// Throws: Error: Call to a member function label() on null
Node::load(9002)->label();
```

```rust
use std::fs::File;
use std::io::prelude::*;

fn main() {
    let file = File::open("foo.txt");
    let mut contents = String::new();

    file.read_to_string(&mut contents);
    println!("The contents of the file: {}", contents);
}
```

```
error[E0599]: no method named `read_to_string` found for type `std::result::Result<std::fs::File, std::io::Error>` in the current scope
 --> src/main.rs:8:10
  |
8 |     file.read_to_string(&mut contents);
  |          ^^^^^^^^^^^^^^
```

Error handling with the Result type

> `Result<T, E>` is the type used for returning and propagating errors. It is an enum with the variants, `Ok(T)`, representing success and containing a value, and `Err(E)`, representing error and containing an error value.

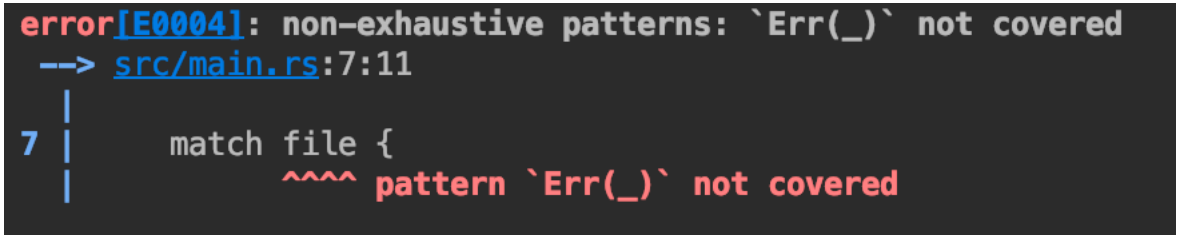
```rust
enum Result<T, E> {
   Ok(T),
   Err(E),
}

// Source: https://doc.rust-lang.org/std/result/
```

```rust
use std::fs::File;
use std::io::prelude::*;

fn main() {
    let file = File::open("foo.txt");

    match file {
        Ok(mut f) => {
            let mut contents = String::new();
            f.read_to_string(&mut contents).expect("Reading failed");
        }
    }
}
```


error[E0004]: non-exhaustive patterns: `Err(_)` not covered
--> src/main.rs:7:11
|
7 | match file {
| ^^^^ pattern `Err(_)` not covered

Javascript

open social

Closures

> A `closure` is the combination of a function and the lexical environment within which that function was declared.

Closures

- Mozilla’s Developer Network (MDN) Web docs are awesome
- Let us borrow some examples

```
function init() {
  var name = 'Mozilla'; // name is a local variable created by init
  function displayName() { // displayName() is the inner function, a closure
    console.log(name); // use variable declared in the parent function
  }
  displayName();
}

// Outputs: Mozilla
init();

// Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures
```

```
function makeFunc() {
  var name = 'Mozilla';
  function displayName() {
    console.log(name);
  }
  return displayName;
}

var myFunc = makeFunc();

// Outputs: Mozilla
myFunc();

// Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures
```

```
function addFunction (x) {
  return x + 5;
}

const addArrowFn = x => x + 5;

console.log(addFunction(2)); // 7
console.log(addArrowFn(2)); // 7
```

```
function makeAdder(x) {
  return y => x + y;
}

const add6 = makeAdder(6);
const add9 = makeAdder(9);

console.log(add6(9));  // 14
console.log(add9(9));  // 18
```

Functional Programming

> In computer science, `functional programming` is a programming paradigm that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.
>
> — Wikipedia, Functional_programming

```
const usdToEur = usd => usd * 0.88; // 27-06-2019.
const cart = [
  { label: 'Socks', currency: 'usd', price: 3.99 } ,
  { label: 'Shirt', currency: 'usd', price: 9.99 } ,
  { label: 'Shoes', currency: 'usd', price: 49.99 },
];

for (let i=0; i<cart.length; i++) {
  cart[i].price = usdToEur(cart[i].price);
}

// Outputs:
// [ { label: 'Socks', currency: 'eur', price: 3.5112 },
//   { label: 'Shirt', currency: 'eur', price: 8.7912 },
//   { label: 'Shoes', currency: 'eur', price: 43.9912 } ]
console.dir(cart);
```

```
const usdToEur = usd => usd * 0.88; // 27-06-2019.
const cart = [
  { label: 'Socks', currency: 'usd', price: 3.99 } ,
  { label: 'Shirt', currency: 'usd', price: 9.99 } ,
  { label: 'Shoes', currency: 'usd', price: 49.99 },
];

const cartInEur = cart.map(item => ({
  label: item.label,
  currency: 'eur',
  price: usdToEur(item.price),
}));

// Outputs:
// [ { label: 'Socks', currency: 'eur', price: 3.5112 },
//   { label: 'Shirt', currency: 'eur', price: 8.7912 },
//   { label: 'Shoes', currency: 'eur', price: 43.9912 } ]
console.dir(cartInEur);
```

```
function createCartExchanger(currency, rate) {
  return item => ({
    label: item.label,
    currency: currency,
    price: item.price * rate,
  });
}
// Exchange rates on 27-06-2019.
const cartToEuro = createCartExchanger('eur', 0.88);
const cartToYen = createCartExchanger('yen', 107.87);
const cartInEur = cart.map(cartToEuro);
const cartInYen = cart.map(cartToYen);

// Outputs:
// [ { label: 'Socks', currency: 'yen', price: 430.40130000000005 },
//   { label: 'Shirt', currency: 'yen', price: 1077.6213 },
//   { label: 'Shoes', currency: 'yen', price: 5392.421300000001 } ]
console.dir(cartInYen);
```

```php
function createCartExchanger(string $currency, float $rate) {
  return function (array $item) use ($currency, $rate) {
    return [
      'label' => $item['label'],
      'currency' => $currency,
      'price' => $item['price'] * $rate,
    ];
  };
}

$cartToEuro = createCartExchanger('eur', 0.88);

$cartInEur = array_map($cartToEuro, $cart);
```

```php
// As of PHP 7.4.
function createCartExchanger(string $currency, float $rate) {
  return fn($item) => [
    'label' => $item['label'],
    'currency' => $currency,
    'price' => $item['price'] * $rate,
  ];
}

$cartToEuro = createCartExchanger('eur', 0.88);

$cartInEur = array_map($cartToEuro, $cart);
```

Perl

Photo by Jimmy Ofisia on Unsplash

open social

Schwartzian Transform

> In computer programming, the `Schwartzian transform` is a technique used to improve the efficiency of sorting a list of items.

Schwartzian Transform

- Sorting based on a certain property
- Calculating that property is expensive
  - e.g. sort orders based on total value

- Can work without intermediate variables

- It’s possible in PHP but horrible, we’ll use Javascript

```
const unsorted = [2, 0, 5, 3, 1, 4];
let seen = [0, 0, 0, 0, 0, 0];

// Slice copies the array.
const sorted = unsorted.slice().sort(
  (a, b) => {
    seen[a]++;
    seen[b]++;
    return a - b;
  }
);

console.log("Number\tTimes seen");
seen.forEach((times, number) => {
  console.log(`${number}\t${times}`);
});
```

Number	Times seen
0	3
1	3
2	5
3	4
4	3
5	4

```
// slowTransform: wait 1s and return x * 2.
const unsorted = [2, 0, 5, 3, 1, 4];

// Slice copies the array.
const sorted = unsorted.slice().sort(
  (a, b) => {
    // Access network, disk, calculate prime number or fibonacci.
    return slowTransform(a) - slowTransform(b);
  }
);

// [ 0, 1, 2, 3, 4, 5 ]
console.log(sorted);

// Takes about 22 seconds.
```

```
// slowTransform: wait 1s and return x * 2.
const unsorted = [2, 0, 5, 3, 1, 4];

// Slice copies the array.
const sorted = unsorted.slice()
  .map(slowTransform)
  .sort(
    (a, b) => {
      // Access network, disk, calculate prime number or fibonacci.
      return a - b;
    }
  );

// [ 0, 2, 4, 6, 8, 10 ]
console.log(sorted);

// Takes about 6 seconds.
```

```
// slowTransform: wait 1s and return x * 2.
const unsorted = [2, 0, 5, 3, 1, 4];

// Slice copies the array.
const sorted = unsorted.slice()
  .map(x => [x, slowTransform(x)])
  .sort(
    (a, b) => {
      // Access network, disk, calculate prime number or fibonacci.
      return a[1] - b[1];
    }
  )
  .map(x => x[0]);

// [ 0, 1, 2, 3, 4, 5 ]
console.log(sorted);

// Takes about 6 seconds.
```

Ruby Styleguide Wisdom

open social

> Be consistent and use common sense

Thank you! Questions?

Let me know what you thought:
http://tiny.cc/dj2019feedback

Slides: http://tiny.cc/dj2019slides

Snippets: http://tiny.cc/dj2019code

open social