Twitter Thread by Stephen Gruppetta

Stephen Gruppetta
@s_gruppetta_ct

— Code Wars: Episode VII – FP battles OOP —

Although Functional Programming (FP) and Object-Oriented Programming (OOP) are often pitted against each other (as I am doing here–guilty as charged), is this just a made-up conflict?

Let's find out…

1/

I won't go into the details of either FP or OOP in this thread (a series on OOP is coming soon, by the way)

So let's look at what they are _in brief_

2/

They are usually referred to as 'paradigms'. They're styles of programming which follow certain guidelines. They also lead to a different mindset on how to design and write computer programs

3/

They're not 'opposites' of each other in the way that 'tall' and 'short' or 'big' and 'small' are opposites

However, they do have certain rules and guidelines that are not compatible with each other. This often leads to them being seen as opposites

4/

And as with most topics in this Code Wars series, there are groups of people who are very vocal in their support for one or another of the paradigms (usually accompanied by a disdain for the other)

5/

However, I'll make the case here that the two paradigms are not enemies of each other and both have a very important role in programming. It doesn't have to be either one or the other

6/

In OOP, the object takes centre stage. It says it in the name, after all!

You create a class which acts as a template to create similar objects–that is objects which have the same attributes

7/

The attributes are data attributes and methods–that means that each object has variables attached to it to store data (data attributes) and it also has functions attached to it which act on the object itself (methods)

8/

Because of this structure in which an object carries everything it needs with it everywhere it goes, methods on objects generally change the state of the object

This means they change the values stored in the data attributes (instance variables)

9/

For example, if you have `some_object.do_something()`, this method may change `some_object.some_data`

It's doing this "behind the scenes"

Some people don't like this!

10/

For example, when using the `turtle` module, you create an instance of the `Turtle` class, let's say `fred = turtle.Turtle()`

When you write `fred.forward(10)`, the method `forward()` changes the value of the attribute `fred._position`

These are often called "side effects"

11/

As a user of the class, you don't need to know that

Whoever wrote the class made sure the behaviour of `forward()` is correct and changes whatever needs changing

12/

In FP, the function takes the pride of place as the main building block

The function is also used in a stricter way in FP than in other paradigms

Variables are treated as immutable (or _are_ immutable in programming languages which are strictly functional)

13/

Therefore, all variables needed in a function are passed as arguments. And these variables are not mutated

Copies of them are returned by the function

There are no side effects in FP

14/

For example, here's what a call in the functional style may look like:

`new_variable = do_something(some_function, some_variable)`

15/

Here's a simple example using `map()`, which is one of the built-in functions used often in the functional style:

```
>>> numbers = [4, 5, 6, 1, 9]
>>> output = map(float, numbers)
```

The variable `numbers` never changes. And no other variable will change

16/

The function `map()` returns a new object. In this case, each element of `numbers` is passed to `float()` and `output` will give you access to the values converted to floats

17/

In FP, only the information input as arguments is used and the variables used as arguments never change

This is different from OOP in which a method (which is a function) has access to other variables linked to the object and can (and will) change the state of variables

18/

So which is better?

If you browse the internet (a dangerous place to be at the best of times!) you are just as likely to find both pages singing the

praises of OOP saying it's the best thing in programming and others warning you to never use OOP and use FP instead…

19/

I take the pragmatic view. There are programs I write where I feel the OOP approach is best suited. But then there are other programs in which I opt to choose the FP style.

20/

It depends. If you're comfortable with both styles and get a "feel" on the pros and cons of each one, then you'll be well placed to use the one that suits your application best

What's great about Python is that it gives you the option and the freedom to use both

21/

So, this battle ends in a truce. The two battling sides realise that there's nothing to fight about, after all, as they're really on the same side!

But beware, not everyone will observe this truce. Some will keep waging this war…

22/