
AVISI

FP and OOP: Best Friends Forever?

Combining the best of two worlds

Agenda.

- What?
- So what?
- Now what?

What?

Setting the stage

Paradigms.

- Procedural (PP)
- Object-Oriented (OOP)
- Functional (FP)

Procedural Programming.

	PP
Imperative	✅
Mutable state	✅
Logic and data separated	✅

Object-Oriented Programming.

	PP	OOP
Imperative	✅	✅
Mutable state	✅	✅
Logic and data separated	✅	
Encapsulation / data hiding		✅
Dynamic polymorphism		✅

Functional Programming.

	PP	OOP	FP
Imperative	✅	✅	
Declarative			✅
Mutable state	✅	✅	
Referential transparency			✅
Logic and data separated	✅		✅
Encapsulation / data hiding		✅	
Dynamic polymorphism		✅	
Functions are first-class citizens			✅

Referential Transparency?

The result of a function depends **only** on its input parameters.

```
fun transparent(a: Int, b: Int) =
    a + b


fun opaque(a: Int, b: Int) =
    a + b + Random.nextInt()
```

First-class functions?

Functions are **values**, just like integers, strings, etc.

```
val list = listOf("hello", "world")

val toUppercase: (String) -> String =
   { s: String -> s.uppercase() }

println(
    list.map(toUppercase) // declarative!
)

// Output: [HELLO, WORLD]
```

So what?

We can use FP in OOP languages!

Many modern **OOP** languages have adopted **FP** concepts

Functional-Object Oriented Programming.

	PP	OOP	FP	F-OOP
Imperative	✅	✅		
Declarative			✅	🎉
Mutable state	✅	✅		🃏
Referential transparency			✅	🎉
Logic and data separated	✅		✅	
Encapsulation / data hiding		✅		🎉
Dynamic polymorphism		✅		🎉
Functions as first class citizens			✅	🎉

Demo!

Now what?

OOP and FP: BFFs?

Yes!

But a good relationship is about giving and receiving

FP

- → - Allow some impure code
- → + Encapsulation

OOP

- → - Constructors
- → + Referential transparency
- → + More patterns
- → + First-class functions