Why isn’t **Functional Programming** the Norm?

@rtfeldman
Why are things the way they are?
It’s complicated!
Part 1: Language

Part 2: Paradigm

Part 3: Style
Part 1: Language
What languages are the norm today?
Global developer population report 2019
Total functional programming languages: 0
How did they get popular?

1. Killer App
2. Platform Exclusivity
3. Quick Upgrade
1. Killer App
Software: $100

Hardware: $10000
“VisiCalc is the first program available on a microcomputer that has been responsible for sales of entire systems.”

- BYTE magazine, 1980
the killer app for

Ruby

RAILS
the killer app for

[logo: PHP]

making HTML dynamic
the killer app for systems programming
Computer Science - Brian Kernighan on successful language design

- culturally compatible (e.g., C-like)
- familiar syntax helps (e.g., C-like)
- easy to get started with
- able to new environment
- open to competition
- open source, not proprietary
- environment to use
- can use standard tools
- competition
- PASCAL
- ASM
Potential FP Killer Apps

Elm  elm-ui
Clojure  Datomic
ReasonML  revery
2. Platform Exclusivity
before 1991, researchers at the European physics lab CERN develop the technology that allows the world wide web.

- 1991, First web page published and first web browser released
- 1993, Mosaic web browser released
- 1994, Netscape web browser released
- 1995, Internet Explorer & Apache server
- 1996, ICQ released
- 1997, Google search
- 1999, Alibaba & Napster

2000:
- 2000, Wikipedia launched
- 2001, Baidu launched, Dot-com bubble bursts

2003:
- 2003, Skype launched

2004:
- 2004, shift to Web 2.0; Facebook

2005:
- 2005, YouTube & Google Maps

2006:
- 2006, IPhone, Twitter launched

2007:
- 2007, IPhone

2008:
- 2008, Github founded, Chrome web browser released

2010:
- 2010, Whatsapp, Uber, Bitcoin and Blockchain

2011:
- 2011, WeChat released

2014:
- 2014, OurWorldInData.org online

Licensed under CC-BY-SA by the author Max Roser.
“Write once, run anywhere.”
3. Quick Upgrade
Considerations

Benefits
Familiarity
Learning Curve
Ecosystem Access
Code Migration Effort

“It’s just JavaScript”
Considerations

- Benefits
- Familiarity
- Learning Curve
- Ecosystem Access
- Code Migration Effort

“It’s just JavaScript”

TypeScript

strict superset of JS
Considerations

Benefits

Familiarity

Learning Curve

Ecosystem Access

Code Migration Effort

near-superset of C
Considerations

Benefits

Familiarity

Learning Curve

Ecosystem Access

Code Migration Effort

“Kotlin is 100% interoperable with [Java] and major emphasis has been placed on making sure that your existing codebase can interact properly with Kotlin.”
How did they get popular?

1. Killer App
   - C, Ruby, PHP
2. Platform Exclusivity
   - JS, Objective-C, Swift, C#
3. Quick Upgrade
   - C++, Kotlin, TypeScript
How did they get popular?

1. Killer App
   C, Ruby, PHP

2. Platform Exclusivity
   JS, Objective-C, Swift, C#

3. Quick Upgrade
   C++, Kotlin, TypeScript

4. Epic Marketing
   Java
Sun Microsystems Boots Up Blitz To Create 'Java Powered' Loyalty

$500M Java marketing campaign in 2003
How did they get popular?

1. Killer App
2. Platform Exclusivity
3. Quick Upgrade
4. Epic Marketing
5. Slow & Steady

Languages:
- Killer App: C, Ruby, PHP
- Platform Exclusivity: JS, Objective-C, Swift, C#
- Quick Upgrade: C++, Kotlin, TypeScript
- Epic Marketing: Java
- Slow & Steady: Python
Other Popularity Factors

Syntax

Job Market

Community
Why are the most popular languages **OO** (except C)?
Part 2: Paradigm
Programming paradigms are a way to classify programming languages based on their features. Languages can be classified into multiple paradigms.
Are OO languages the norm because of uniquely OO features?
What are uniquely OO features?

1. Encapsulation?

2. Inheritance?

3. Objects?

4. Methods?
Inheritance

Interface Inheritance (aka subtyping)

Implementation Inheritance (aka inheritance)
“Composition over inheritance”
“Supports an object-oriented style” encapsulation?

No inheritance

“Objects and methods” are syntax sugar for structs and procedures

circle.grow(3)
grow(circle, 3)
Modular Programming

Modularity lets you define a public interface to hide private implementation details
Languages with modules

*coming in C++20

(*) JavaScript includes CoffeeScript, TypeScript

Global developer population report 2019. https://sdata.me/GlobalDevPop19 • ©SlashData
Modular Programming

Modularity lets you define a public interface to hide private implementation details.
Encapsulation

Encapsulation lets you define a public interface to hide private implementation details about an object’s state
Objects
Classes
Inheritance

Garbage Collection

John McCarthy, Creator of **Lisp**
“Object-Oriented” -Alan Kay

OOP to me means only messaging, local retention and protection and hiding of state-process, and extreme late-binding of all things. It can be done in Smalltalk and in LISP. There are possibly other systems in which this is possible, but I’m not aware of them.
“I wasn't happy with C as a productivity foundation, and I was lobbying around for anything that could help. That was about the time the Byte Magazine issue came out.”

“There were a whole bunch of things I thought vaguely might help. Encapsulation for sure...C is so bad at it. Everything is public; it just turns into soup. The pain of that is what I was trying to escape.”
C with Classes: Why Classes?

- Program organization
- Mapping of concepts
  - A class is a type
  - Static checking

Bjarne Stroustrup
Drop-in C Replacement (!)

Why C++?
- C with Classes was a "medium success"
- Allowed medium improvements (only)
- User community couldn’t support infrastructure

Bjarne Stroustrup
C with **OOP** (“C with Classes”) wasn’t sufficient for popularity

Sun wanted **familiarity** for C++ programmers

Microsoft wanted a **proprietary Java** alternative

Brad Cox wanted **modularity**

Apple wanted to improve on ObjC
“I wanted a scripting language that was more powerful than Perl, and more object-oriented than Python.”
—Yukihiro “Matz” Matsumoto, creator of Ruby
Are OO languages the norm because of \textit{uniquely} OO features?

No.

They’re OO because \textit{modularity is a good idea}, and they originally got it from OO \textit{by chance}.
Part 3: Style
Functional Programming Style

“Avoid mutation and side effects.”

No language features required

Languages differ in their support for this style
Why isn’t FP style the norm?
Kotlin has both object-oriented and functional constructs. You can use it in both OO and FP styles, or mix elements of the two.
Some additional features of Swift include:

Functional programming patterns, e.g., `map` and `filter`
“Wouldn’t it be nice if my language had strong support for the style that’s become the norm?”

(*) JavaScript includes CoffeeScript, TypeScript

Global developer population report 2019. [https://sdata.me/GlobalDevPop19](https://sdata.me/GlobalDevPop19) • ©SlashData
Why isn’t Functional Programming the Norm?

@rtfeldman
Part 1: Language

Part 2: Paradigm

Part 3: Style
Why aren’t FP **languages** the norm?

1. No sufficiently large “killer apps”
2. No exclusivity on large platforms
3. Can’t be a quick upgrade if substantially different
4. No epic marketing budgets
5. Slow & steady growth takes decades
Are OO languages the norm because of **uniquely** OO features?

1. **Information hiding** (encapsulation) is not a uniquely OO feature. **Modules** can do it too.
2. **Inheritance** is uniquely OO, but OO best practice encourages using **composition** instead.
3. Without inheritance, **objects and methods** are not significantly different from **structs and procedures**.
Why isn’t FP style the norm?
@rtfeldman

Elm in Action

Richard Feldman