# Mutability

## Necessity or Habit?

Paulo Pereira, 2024

# A quick note about me

- Professional imperative programmer (and professor) for almost 30 years
  - ASM (several variants), C, C++, Java, C#, JavaScript (and TypeScript) and Kotlin
  - Also played around with many others
- Using Kotlin since 2017 (not a language expert, though)
- Full-time professor for the last 4 years

# What about you guys?

- Are you full-time programmers?
  - For how long now?
  - Not anymore? Why not?
- If so, do you intend to continue do it professionally?
  - For how long? Until you retire?
  - If not, why not?

# The Joys of the Craft

"The magic of myth and legend has come true in our time. One types the correct incantation on a keyboard, and a display screen comes to life, showing things that never were nor could be."

- Frederick P. Brooks, in The Mythical Man Month, 1975

"It's the sheer joy of making things."

- Ibidem

# The Woes of the Craft

"One must perform perfectly. If one character, one pause, of the incantation is not strictly in proper form, the magic doesn't work. Human beings are not accustomed to being perfect, and few areas of human activity demand it."

- Ibidem

# Mutability

- Mutability: Necessity or Habit?
- TLDR;
  - It's (mostly) an habit of the imperative programmer
  - We should default to immutable data
  - Why? For the sake of simplicity

# Some historical context

- Computers (or execution targets) have changed a great deal
  - There's a wide range of them
  - And constraints are no longer the same
- A conversation with GPTo, available here

# Simplicity, you say?

```
var count = 5
```

What's the value of count ?

# Simplicity, for sure

```
val count = 5
```

And now? Now it became global knowledge!

# Encapsulate, right? (1)

```
val counter = ClassicCrowdCounter(
    initialValue = 0,
    capacity = 650
)
```

Is this better? (definition here)

# Encapsulate, right? (2)

```
hereBeDragons(counter)
```

What is the value of counter?
(before AND after the function call)

# Immutability

```
val counter = CrowdCounter(
    initialValue = 0,
    capacity = 650
)
```

And now, is this better? (definition here)

# What if ...

- We reduce mutations to a bare minimum, by
  - creating models comprised of immutable data
  - and making data flows explicit in our use cases

- What would our code look like?

# A demo

https://github.com/palbp/laboratory/tree/main/Essays/Immutability/MissileCommand

# Missile Command demo

- Design favors the use of immutable data, that is, a "value-oriented" programming style
- There's a single point of mutation, where the new value is commited
- Two distinct phases:
  - The new application state is computed
  - The new state is commited as the current state

# Mutability?

- At the application level ...
  - It's a nasty habit, really
  - For the most part, it should be avoided
- Our execution targets have changed drastically

# Some references

- "Values and objects in programming languages", by B.J. MacLennan (article)
- "The Value of Values", by Rich Hickey (video)
- "Functional Programming: The Failure of State", by Uncle Bob, a.k.a. Robert C. Martin (video)
- "Immutability we can afford", by Roman Elizarov (article)
- "Immutability changes everything", by Pat Helland (article)

# Some books

- "Code that Fits in Your Head", by Mark Seemann
- "Data-Oriented Programming", by Yehonathan Sharvit
- "Domain-Driven Design", by Eric Evans
- "Domain Modeling Made Functional", by Scott Wlaschin
- "A Philosophy of Software Design", by John Ousterhout