Functional Data Structures II

Immutability

06016415 Functional Programming

IT KMITL
พระจอมเกล้าลาดกระบัง

Outline

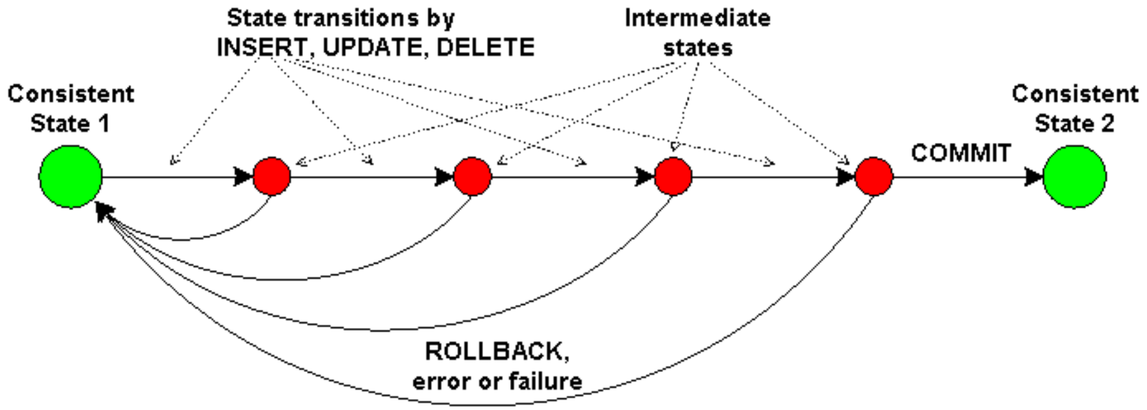
- Immutable vs Mutable Variable
- Functional Lists

Immutable vs Mutable Variable

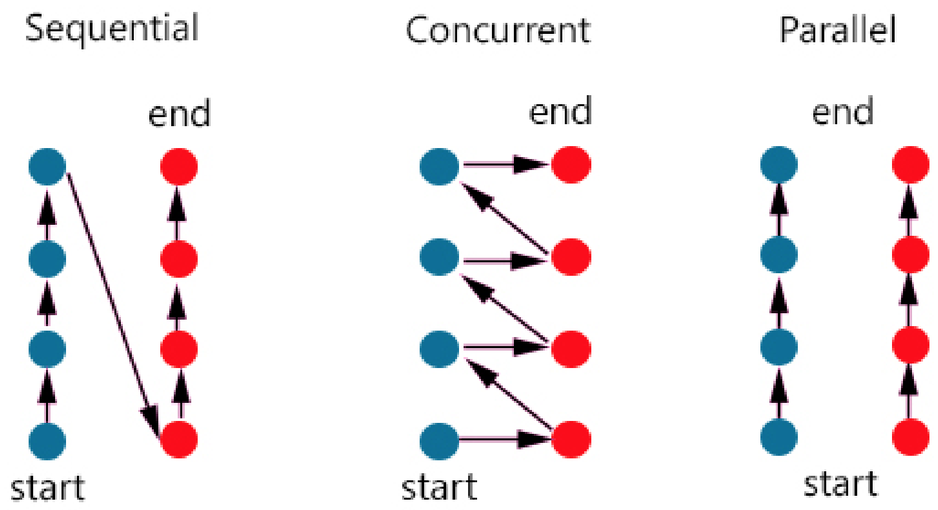
- **FP** มาจากแนวคิด **Mathematical Functions** ซึ่งจะต้องไม่เปลี่ยนแปลงค่า **(Immutable)**
- ตัวแปรแบบ Immutable จะทำให้การสร้าง Pure Function เกิดขึ้นได้
  - Pure function จะต้องได้รับค่าผลลัพธ์เดิมเสมอ
  - Output depends on Input
  - ฟังก์ชันที่ไม่มี side effects.

- Immutable variables หรือ ตัวแปรไม่เปลี่ยนค่า อาจทำให้เราสับสน กับตัวแปร (Mutable variables) ที่เคยรู้จัก ซึ่งมีไว้เพื่อเปลี่ยนแปลงค่า แล้วนำไปคำนวณ

- **Immutable variables คือ Variables That Aren't**



Example of immutability : Database Transactions



Sequential VS Concurrent VS Parallel

- ประโยชน์ของ Immutability คือช่วยแก้ปัญหา concurrency คือปัญหาการเข้าถึงข้อมูลที่ร่วมกัน แล้วเกิดความไม่แน่นอน หรือคืนค่าผลลัพธ์ผิดพลาด (ตัวอย่าง โจทย์ big data เป็นต้น)

Behavior about immutability

- Local variables จะไม่เปลี่ยนแปลงค่า
- Global variables สามารถเปลี่ยนแปลงค่า เฉพาะเมื่อถูกอ้างอิง (only references)

```
var prompt = "> "
def format(msg: String): String = prompt + msg

format("command") // "> command"
prompt = "% "     // change the prompt
format("command") // "% command"
```

This function has the side effect of modifying variable lastID

```
var lastID = 0
def uniqueName(prefix: String): String =
    lastID += 1
    prefix + lastID

uniqueName("user-") // "user-1"
uniqueName("user-") // "user-2"
```

Append and Prepend on Seq

:+ เหมือน ++

+: เหมือน ++:

Method	Description	Example
:+	append 1 item	oldSeq :+ e
++	append N items	oldSeq ++ newSeq
+:	prepend 1 item	e +: oldSeq
++:	prepend N items	newSeq ++: oldSeq

```
val oldSeq = Seq(1, 2, 3)
val newSeq = oldSeq :+ 4  // Append 4 เข้าไปท้าย Seq

println(newSeq) // Output: Seq(1, 2, 3, 4)
println(oldSeq) // Output: Seq(1, 2, 3) (ไม่เปลี่ยนแปลง)
```

```
val newSeq = oldSeq ++ Seq(4, 5, 6)
println(newSeq) // Output: Seq(1, 2, 3, 4, 5, 6)
```

```
val oldSeq = Seq(3, 4)
val newSeq = Seq(1, 2) ++: oldSeq // เติม Seq(1, 2) ไว้ข้างหน้า

println(newSeq) // Output: Seq(1, 2, 3, 4)
```

The Scala standard library implements both mutable and immutable sets.

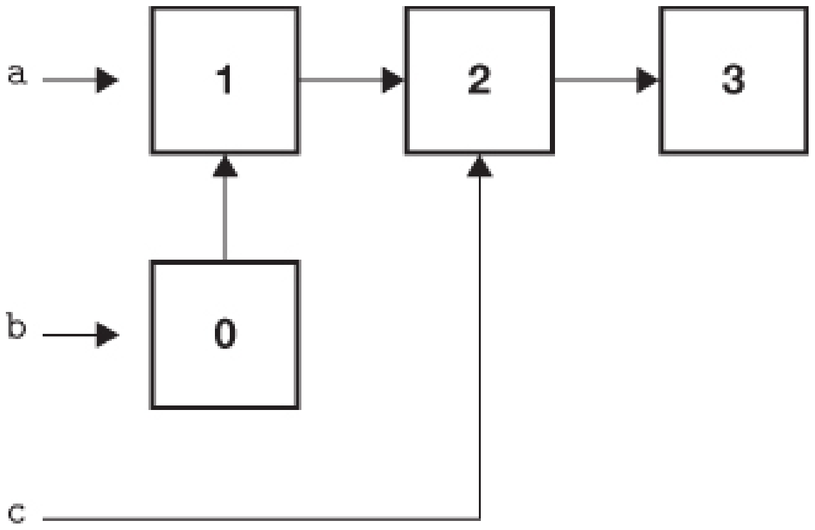
```
scala> var set = Set("A","B")
//var set: Set[String] = Set(A, B)


scala> set =set+"C"
//set: Set[String] = Set(A, B, C)

scala> set =set+"D"
//set: Set[String] = Set(A, B, C, D)



scala> val set = Set("A","B")
//val set: Set[String] = Set(A, B)

scala> set + "C"
//val res1: Set[String] = Set(A, B, C)

scala> set
//val res2: Set[String] = Set(A, B)
```

Functional Lists

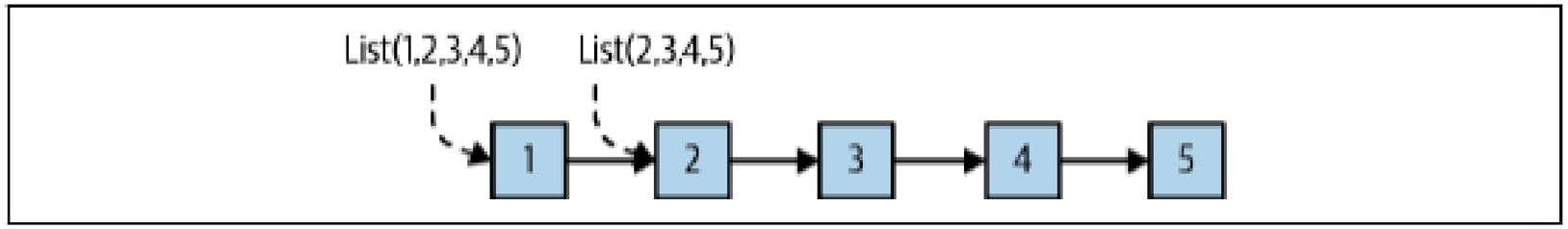
- Functional list เป็นโครงสร้างข้อมูลพื้นฐาน ใช้โดยทั่วไป
- Empty list คือ ไม่มี elements หรือ nil
- Head คือ ค่าแรก ใน list
- Tail คือ ค่าส่วนที่เหลือ หรือส่วนท้ายใน list
- ตัวดำเนินการ หรือ “cons.” เช่น :: , +: และ :+ เป็นต้น

```
scala> 1 :: 2 :: 3 :: Nil
val res0: List[Int] = List(1, 2, 3)
```

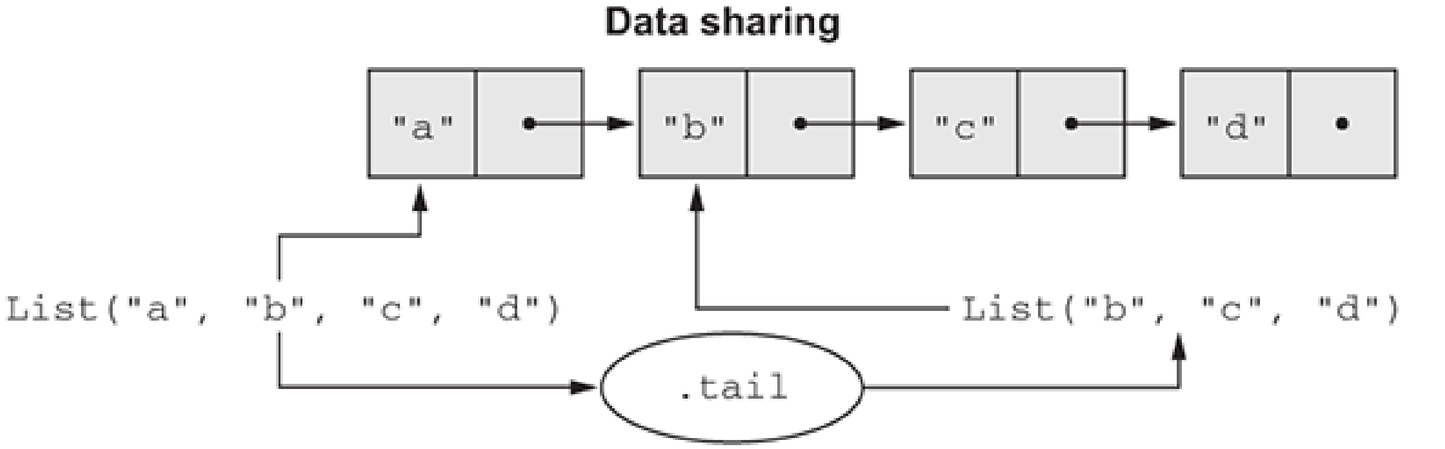
Functional lists memory sharing




```
scala> val a = List(1, 2, 3)
val a: List[Int] = List(1, 2, 3)

scala> val b = 0 :: a
val b: List[Int] = List(0, 1, 2, 3)

scala> val c = a.tail
val c: List[Int] = List(2, 3)
```

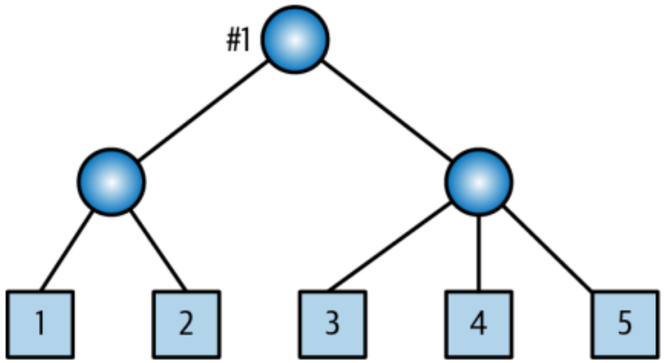


In functional data structures, sharing a structure to minimize the cost of making copies.

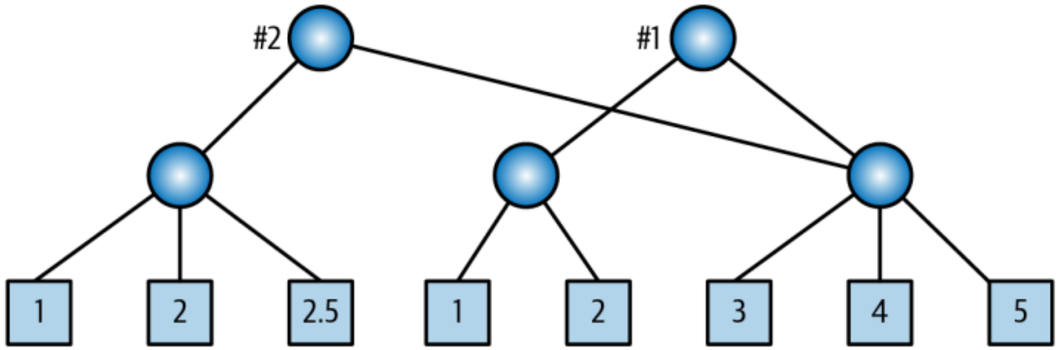


Both lists share the same data in memory. `.tail` does not modify the original list; it simply references the tail of the original list. Defensive copying is not needed because the list is immutable.

Making copies of functional collections



A Vector represented as a tree



Element insertion

Assignment

- **Individual work**
- **Create the code for**
  1. **Function Tail ē fjě ðćfè ċ fist element Ŷ ęż list ffæffī**
  2. **Function Drop ē fjě ðćfè ċ fist N element Ŷ ęż list ffæffī(Ý ðæ fìempty list ffě ćfę Ý êffifçfćī empty list)**
  3. **Function DropWhile ē fjě ðćfè ċ element a flÝ list ffæffīb ź ffīb ffist element a č Ý ðffb ź c ź element ć ź ðżb flď fźęč ffŷ ć ź fjě č æ**

All functions requires at least 2 examples

- ตัวอย่าง

  List("I","LOVE","Andaman", "of", "Thailand")

  1. Function Tail จะได้ผลลัพธ์ เป็น List("LOVE","Andaman", "of", "Thailand")
  2. Function Drop ลบตั้งแต่ส่วนต้น ไป 2 ตำแหน่ง (N=2)
  3. Function DropWhile ลบตั้งแต่ส่วนต้น ไปจนถึงคำว่า "of"

  ส่งเป็น .scala ไฟล์เท่านั้น