DIS 04 : Data Structure 1

	Student ID	Name
1	66070228	อินทิรา ธนัพประภัศร์
2	66070248	ชนิสรา จันทร์คำจร
3	66070286	ปนัสยา บุญประกอบ

Objectives:

1. (CLO2) สามารถอธิบายโครงสร้างข้อมูลเชิงฟังก์ชั่นได้

Tools:

- IDE: VS Code or IntelliJ
- Scala 3
- Command Prompt

Exercise 1: Review Scala Syntax

อธิบายโปรแกรมข้างต้นทีละบรรทัด

No.	Title	Descriptions
1	val str = Seq("I", "AM","GROOT") str.map(_.toLowerCase) str.map(_.toUpperCase)	- ประกาศ seq ที่มีค่า 3 ค่า คือ "I", "AM", "GROOT" - แปลงค่า(map) ค่าใน seq แต่ละค่าเป็น lowercase - แปลงค่า(map) ค่าใน seq แต่ละค่าเป็น uppercase
2	1 to 10 1 until 10 1 to 10 by 3 (1 to 10 by 3).foreach(println)	- loop เริ่มต้นที่ 1 และจบที่ 10 - loop เริ่มต้นที่ 1 และจบที่ 9 - loop เริ่มต้นที่ 1 และจบที่ 10 โดยค่าเพิ่มขึ้นทีละ 3 - loop เริ่มต้นที่ 1 และจบที่ 10 โดยค่าเพิ่มขึ้นทีละ 3 และทำการแสดงค่าทีละตัว ซึ่งผลที่ได้คือ 1, 4, 7 และ 10
3	val tup = ("GROOT", 1 , 3.33) tup._1,tup(0)) tup._4,tup(3))	- ประกาศสร้าง tuple ที่มีค่า 3 ค่า คือ "GROOT", 1, 3.33 - tup._1 คือการเข้าถึงค่าใน tuple ตำแหน่งที่ 1 ได้เป็น "GROOT"

		และ tup(0) คือการเข้าถึงค่าโดย index ได้เป็น "GROOT" - tup._4 คือการเข้าถึงค่าใน tuple ตำแหน่งที่ 4 ซึ่งใน tup มีแค่ 3 ตำแหน่งเท่านั้น และ tup(3) คือการเข้าถึงค่าโดย index ที่มี 0-2 เท่านั้น ทั้งสองคำสั่งเป็นการเรียกเกิน ทำให้ในบรรทัดนี้เกิด ERROR
4	var factor = 2 val multiplier = (i:Int) => i * factor val result = (1 to 10).filter(_ % 2 == 0).map(multiplier).reduce(_ * _) factor = 3 val result1 = (1 to 10).filter(_ % 2 == 0).map(multiplier).reduce(_ * _)	- ประกาศตัวแปรชื่อ factor ให้เท่ากับ 2 - สร้างฟังก์ชัน multiplier ด้วย val โดยรับ input ใส่ตัวแปร i เป็น Int และ return ค่าเป็น i * factor - สร้างตัวแปรค่าคงที่ (val) ชื่อ result ให้เท่ากับ 3 โดยให้เท่ากับ (1 to 10) คือลิสต์เลข 1-10 แล้วใช้ filter(_%2 == 0) เป็นการกรองเลขที่หาร 2 ลงตัว จะได้เป็น (2,4,6,8,10) แล้ว map ค่านี้ไปที่ฟังก์ชัน multiplier จะเป็นการเอาค่าไปผ่านฟังก์ชัน ได้เป็น (4,8,12,16,20) จากนั้น reduce(_*_) เป็นการคูณค่าทั้งหมดในลิสต์เข้าด้วยกัน ได้ผลลัพธ์เป็น 122880 - กำหนดค่า factor ใหม่เป็น 3 - ดำเนินการเหมือน result แต่ factor = 3 ได้ผลลัพธ์เป็น 933120
5	def multiplier2: Int => Int = val factor = 2 (i: Int) => i * factor val result2 = (1 to 10).filter(_ % 2 == 0).map(multiplier2).reduce(_ * _)	- ประกาศฟังก์ชัน multiplier2 ด้วย def ซึ่งรับค่า Int - ภายในฟังก์ชันประกาศตัวแปร factor = 2 - กำหนดให้ค่าที่รับมา เป็น i แล้วส่งคืน i * factor - 1 to 10 คือสร้างลิสต์เลข 1-10 แล้วใช้ filter(_%2 == 0) เป็นการกรองเลขที่หาร 2 ลงตัว จะได้เป็น (2,4,6,8,10) แล้ว map

		ค่านี้ไปที่ฟังก์ชัน multiplier2 จะเป็นการเอาค่าไปผ่านฟังก์ชันได้เป็น (4,8,12,16,20) จากนั้น reduce(_*_) เป็นการคูณค่าทั้งหมดในลิสต์เข้าด้วยกัน ได้ผลลัพธ์เป็น 122880
6	def multiplier3(i: Int , factor: Int) : Int = i * factor val result3 = (1 to 10).filter(_ % 2 == 0).map(multiplier3).reduce(_ * _)	- ประกาศฟังก์ชัน multiplier3 ด้วย def ซึ่งรับค่าสองค่ามาใส่ใน i เป็น Int และ factor เป็น Int - return ค่า i * factor - 1 to 10 คือสร้างลิสต์เลข 1-10 แล้วใช้ filter(_%2 == 0) เป็นการกรองเลขที่หาร 2 ลงตัว จะได้เป็น (2,4,6,8,10) แล้ว map ค่านี้ไปที่ฟังก์ชัน multiplier3 จะเป็นการเอาค่าไปผ่านฟังก์ชัน จะได้ i คือเลขในลิสต์คูณกับ factor (ในโค้ดนี้ไม่มีการรับค่า factor ต้องแก้ไขโค้ดและป้อนค่าที่จะใช้เป็น factor เข้าไปเองเมื่อเรียกใช้ฟังก์ชัน multiplier3) จากนั้น reduce(_*_) เป็นการคูณค่าทั้งหมดในลิสต์เข้าด้วยกัน
7	val result = (1 to 10).filter(_ % 2 == 0).map(_ * 2).reduce(_ * _)	- สร้าง Range ด้วย (1 to 10) - filter เอาแค่เลขที่หาร 2 ลงตัว (เลขคู่) แล้วใส่ใน List - เอาจำนวนคู่ที่อยู่ใน List ทั้งหมดมาคูณ 2 - เอาสมาชิกทั้งหมดใน List มาคูณกัน - ค่าใน result เมื่อ print ออกมา คือ 122880
8	val natNums = LazyList.from(0) natNums.take(100).toList	- ประกาศตัวแปร natNums เป็น LazyList ที่เริ่มต้นด้วย 0 - แปลง 100 ตัวใน natNums ให้เป็น regular list

Exercise 2: Sequences

อธิบายโปรแกรมข้างต้นทีละบรรทัด

No.	Title	Descriptions
1	Seq. apply() Map.apply() map.apply() Set.apply()	- การสร้าง Sequence - การสร้าง Map - error เพราะ ต้องเป็น Map.apply() - การสร้าง Set
2	val seq1 = "I"+: "AM"+:"GROOT"+: Nil seq1(1) seq1.head seq1.tail	- การประกาศตัวแปร seq1 เป็น Seq ที่มีค่า “I”, “AM”, “GROOT” และ Nil ซึ่งคือ List ที่ไม่มีค่าอะไรอยู่ข้างใน - ค่าตำแหน่งที่ 1 ของ seq1 ซึงคือ “AM” - ค่าแรกสุดของ seq1 ซึ่งคือ “I” - ค่าที่เหลือที่ต่อจาก head ซึ่งคือ “AM” และ “GROOT”
3	val arr = Array("I","LOVE","ANDAMAN") val array: Array[String] = Array("I","LOVE","ANDAMAN") array = Array("GROOT") array(1) array	- การประกาศตัวแปร Array ชื่อ arr ซึ่งมีค่าคือ "I", "LOVE" และ "ANDAMAN" - ประกาศตัวแปร Array ชื่อ array ที่เก็บข้อมูลเป็น String - Error เพราะเกิดการแก้ไข Array - เข้าถึง array อันดับที่ 1 - เรียกค่า array ออกมาเป็นรูปแบบ Array
4	val seq2 = ("I"+: ("AM"+: ("GROOT"+: (Nil))) val vect1 = Vector("I", "AM","GROOT") val vect2 ="I"+: "AM"+:"GROOT"+: Vector.empty	- ประกาศสร้างตัวแปร seq2 ประเภท sequence โดยใช้วงเล็บและ +: (prepend) ในการเพิ่ม String เข้าไป - ประกาศสร้างตัวแปร vect1 ประเภท vector - ใช้ +: (prepend) เพื่อเพิ่มหัวของ vector ที่ว่าง

	val vect3 = Vector.empty :+ "I":+ "AM":+"GROOT vect3.head vect3.tail val vect4 = seq2.toVector	- ใช้ +: (prepend) เริ่มจาก Vector.empty - ดึงองค์ประกอบแรก head ของ vect3 - ดึงส่วนที่เหลือของ vect3 ยกเว้นส่วน head - ใช้ toVector แปลง seq2 ให้เป็น Vector
5	val stateCapitals = Map("Alabama" -> "Montgomery", "Alaska" -> "Juneau", "Wyoming" -> "Cheyenne") val stateCapitals2a = stateCapitals + ("Virginia" -> "Richmond") val stateCapitals2b = stateCapitals + ("Alabama" -> "MONTGOMERY") val stateCapitals2c = stateCapitals ++ Seq("Virginia" -> "Richmond", "Illinois" -> "Springfield")	- ประกาศตัวแปรชื่อ stateCapitals เป็น Map ซึ่งมี value และ key - ประกาศตัวแปรชื่อ stateCapitals2a เป็น Map ซึ่งต่อท้ายกับ stateCapitals - ประกาศตัวแปรชื่อ stateCapitals2b เป็น Map ซึ่งต่อท้ายกับ stateCapitals - ประกาศตัวแปรชื่อ stateCapitals2c เป็น Seq ซึ่งต่อท้ายกับ stateCapitals ด้วย ++ ซึ่งจะช่วยรวม Map และ Seq เข้าด้วยกัน

Exercise 3: Discussion

ตอบคำถามดังต่อไปนี้

- **Array, List, Seq, Vector ต่างกันอย่างไร**
- **Maps และ Sets ต่างกันอย่างไร**
- **ยกตัวอย่างฟังก์ชันให้รองรับค่า Array, List, Seq, Vector**

Array เป็น mutable ข้อมูลที่มีขนาดตายตัว ใน array เก็บได้แค่ข้อมูลประเภทเดียวกัน

List เป็น immutable เวลาที่ใช้สร้าง bigO(1) ลดเวลาในการประมวลผล สามารถเพิ่มข้อมูลที่หัวได้เร็ว

Seq เป็น immutable แสดงลำดับที่ไม่เปลี่ยนแปลง

Vector เป็นโครงสร้างแบบ dynamic มีลักษณะการจัดเก็บข้อมูลคล้ายกับ Array แต่ Vector มีความยืดหยุ่นและมีประสิทธิภาพมากกว่า

Maps เป็นโครงสร้างข้อมูลพื้นฐานทั่วไป เพื่อใช้จับคู่ระหว่าง key กับ value อีกทั้ง key ต้องเป็น unique คล้ายกับ Seq แต่ลักษณะการใช้การต่างกัน

Sets เป็นโครงสร้างข้อมูลที่ไม่จัดลำดับ คล้ายกับ Maps Key คือ แต่ละ elements ต้องเป็น unique

ยกตัวอย่างฟังก์ชัน

```
def cal(seq :Seq[String]): Seq[String] =
  seq.map(_.toLowerCase)
```