Lab 02 : No More Loops

	Student ID	Name
1	67070159	วราภรณ์ สุขม่วง

Objectives:

1. (CLO1) สามารถอธิบายกระบวนคิด และหลักการในการทำงานของโปรแกรมเชิงฟังก์ชั่นทางคณิตศาสตร์

Recursion

- การเขียน Recursive ต้องประกอบด้วย 2 ส่วนเสมอ:
  - **Base Case:** จุดหยุด (ถ้าไม่มีสิ่งนี้ โปรแกรมจะวนไม่รู้จบจน Stack Overflow)
  - **Recursive Step:** การเรียกตัวเองด้วยปัญหาที่เล็กลง

ในภาษา Java/Python ปกติเราเลี่ยง Recursion เพราะกลัว Stack Overflow แต่ใน Scala (และ FP Language อื่นๆ) สามารถใช้ Tail Recursion Optimization ทำให้เราเขียน Loop ในรูปแบบ Recursive ได้อย่างปลอดภัยและมีประสิทธิภาพเท่ากัน

- **Note สำหรับการสร้างโปรเจคใหม่**
  - สร้าง folder สำหรับวางไฟล์งาน ด้วยคำสั่ง *cd ชื่อfolder*
  - พิมพ์ ว่า *sbt new scala/scala3.g8* เพื่อสร้างโปรเจคจาก template
  - ตั้งชื่อว่า *xxxx* (warning : อาจจะต้องรอนาน)
  - พิมพ์ ว่า *cd xxxx*
  - พิมพ์ ว่า *sbt run*

Exercise 1: The Need for Recursion

อธิบายโปรแกรมข้างต้นทีละบรรทัด

def processOne[A](collection: List[A]): List[A] =
  collection match
  case _ :: t => t
  case Nil => throw new Exception("EmptyList")
	เช็คว่า List เป็นค่า Nil หรือไม่
	ถ้าไม่ จะ Return ทุกตัวยกเว้นตัวแรก
	ถ้า Nil จะ Error ว่า EmptyList

def processCollection[A](collection: List[A]): Unit =
  while collection.nonEmpty do
  println(processOne(collection))
	ถ้าไม่เป็น Empty จะแสดงค่าทุกตัวยกเว้นตัวแรกวนไปเรื่อยๆ

def processCollection2[A](collection: List[A]): Unit =
  var remaining = collection
  while remaining.nonEmpty do
  println(remaining)
  remaining = processOne(remaining)
	จะแสดงค่าใน List ยกเว้นตัวแรกวนไปเรื่อย ๆ จนเหลือแค่ตัวเดียว

def processCollection3[A](collection: List[A]): Unit =
  println(collection)
  if collection.nonEmpty then processCollection3(processOne(collection))

เหมือนกับข้อ processCollection2 แต่จะแสดงค่าว่างออกมาด้วย

```
println("---------------")
println("Exercise 1-0")
println(processOne(List(1,2,3)))
println("\nExercise 1-1")
println("\nLOOP  ~~TwT~~")
// processCollection(List(1,2,3))
println("\nExercise 1-2")
processCollection2(List(1,2,3))
println("\nExercise 1-3")
processCollection3(List(1,2,3))
```

```
Exercise 1-0
List(2, 3)

Exercise 1-1

LOOP  ~~TwT~~

Exercise 1-2
List(1, 2, 3)
List(2, 3)
List(3)

Exercise 1-3
List(1, 2, 3)
List(2, 3)
List(3)
List()
```

Exercise 2: Factorial

นิยามทางคณิตศาสตร์: $n! = n \times (n-1)!$ โดยที่ $0! = 1$
ซึ่งในการเขียน Recursive ต้องประกอบด้วย 2 ส่วน

1. **Base case:** ถ้า n เท่ากับ 0 ให้คืนค่า 1
2. **Recursive step:** คืนค่า n * factorial(n-1)

จงเขียนฟังก์ชัน factorial(n: Int): Int

ทดสอบดังต่อไปนี้

```
def factorial(n: Int): Int =
 // Hint: ใช้ if-else เช็ค
 ???

println(factorial(5)) // Expected: 120
```

```
def factorial(n: Int): Int =
  if n == 0 then 1 else n * factorial(n-1)
```

```
Exercise 2 : Factorial
120
```

Exercise 3: Fibonacci Sequence

นิยามทางคณิตศาสตร์: $F(n) = F(n-1) + F(n-2)$ โดยที่ $F(0) = 0, F(1) = 1$
ผลลัพธ์คือ ลำดับ: 0, 1, 1, 2, 3, 5, 8, ...

ซึ่งในการเขียน Recursive ต้องประกอบด้วย 2 ส่วน

1. **Base case:** ที่ $F(0) = 0, F(1) = 1$
2. **Recursive step:** $Fib(n-1) + Fib(n-2)$

จงเขียนฟังก์ชัน fib(n: Int): Int

ทดสอบดังต่อไปนี้

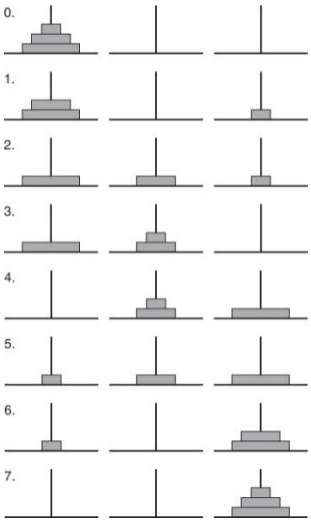
```
def fib(n: Int): Int =
  ???


println(fib(6)) // Expected: 8
```

```
def fib(n: Int): Int =
  if n <= 1 then n
  else fib(n-1) + fib(n-2)
```

```
Exercise 3 : Fibonacci
8
```

Exercise 4: Tower of Hanoi Puzzle



โจทย์: ตำนานเล่าว่ามีเสา 3 ต้น และจานทองคำ n ใบที่มีขนาดต่างกันเรียงซ้อนกันอยู่ นักบวชต้องการย้ายจานทั้งหมดจากเสาต้นแรก (Start) ไปยังเสาต้นสุดท้าย (End)

โดยมีกฎเหล็ก 3 ข้อ:

1. **ย้ายได้ทีละ 1 ใบ**
2. **ห้ามวางจานใบใหญ่ทับจานใบเล็ก**
3. **ใช้เสากลาง (Auxiliary) ช่วยพักจานได้**

ซึ่งในการเขียน Recursive ต้องประกอบด้วย 2 ส่วน

1. **Base case:** _____
2. **Recursive step:** _____

จงเติมโค้ดในฟังก์ชัน hanoi เพื่อพิมพ์ขั้นตอนการย้ายจานทั้งหมดอ

ทดสอบดังต่อไปนี้

```
// ฟังก์ชัน hanoi รับ Generic Type [A] เพื่อให้เราตั้งชื่อเสาเป็น String หรือ Int ก็ได้
def hanoi[A](n: Int, from: A, middle: A, to: A): Unit =
  if (n > 0) then
    // 1. ย้ายจาน n-1 ใบ จากเสาต้นทาง (from) ไปพักไว้ที่เสากลาง (middle)
    hanoi(???, ???, ???, ???)
    // 2. ย้ายจานใบใหญ่ที่สุด (ใบที่ n) จากต้นทาง (from) ไปปลายทาง (to)
    println(s"$from -> $to")
    // 3. ย้ายจาน n-1 ใบ ที่พักอยู่เสากลาง (middle) ตามไปที่ปลายทาง (to)
     hanoi(???, ???, ???, ???)

println("--- Towers of Hanoi (3 Disks) ---")
// ลองเรียกใช้โดยให้เสาชื่อ  "L", "M", "R"
hanoi(3, 'L', 'M', 'R')
```

```scala
def hanoi[A](n: Int, from: A, middle: A, to: A): Unit =
  if (n > 0) then
    // 1. ย้ายจาน n-1 ใบ จากเสาต้นทาง (from) ไปพักไว้ที่เสากลาง (middle)
    hanoi(n-1, from, to, middle)
    // 2. ย้ายจานใบใหญ่ที่สุด (ใบที่ n) จากต้นทาง (from) ไปปลายทาง (to)
    println(s"$from -> $to")
    // 3. ย้ายจาน n-1 ใบ ที่พักอยู่เสากลาง (middle) ตามไปที่ปลายทาง (to)
    hanoi(n-1, middle, from, to)
```

```
Exercise 4: Tower of Hanoi Puzzle
L -> R
L -> M
R -> M
L -> R
M -> L
M -> R
L -> R
```

Exercise 5: @tailrec

@tailrec เป็น annotation ที่ใช้ตรวจสอบว่าโค้ดใช้ Tail Recursion Optimization (TCO) ได้หรือไม่ หากโค้ดไม่สามารถใช้ TCO ได้ คอมไพเลอร์จะให้ error เพื่อเตือนนักพัฒนา

ทดสอบดังต่อไปนี้

```scala
def factorial(n: Int, acc: Int = 1): Int =
  if (n <= 1) acc
  else factorial(n - 1, acc * n) // Tail Call
```

ทดสอบดังต่อไปนี้

```scala
import scala.annotation.tailrec

@tailrec
def factorial(n: Int, acc: Int = 1): Int = {
  if (n <= 1) acc
  else factorial(n - 1, acc * n)
}
```

ทดสอบดังต่อไปนี้

```
@tailrec
def factorial(n: Int): Int = {
  if (n <= 1) 1
  else n * factorial(n - 1) // ไม่ใช่ tail call เพราะมี *n หลังจาก recursive call
}
```

```
PS C:\Users\LAB203-XX\Documents\GitHub\Functional-Programming-2-68\LAB02> sbt run
[info] welcome to sbt 1.11.7 (Oracle Corporation Java 17.0.10)
[info] loading project definition from C:\Users\LAB203-XX\Documents\GitHub\Functional-Programming-2-68\LAB02\project
[info] loading settings for project root from build.sbt...
[info] set current project to LAB02 (in build file:/C:/Users/LAB203-XX/Documents/GitHub/Functional-Programming-2-68/LAB02/)
[info] compiling 1 Scala source to C:\Users\LAB203-XX\Documents\GitHub\Functional-Programming-2-68\LAB02\target\scala-3.7.4\c
lasses ...
[error] -- Error: C:\Users\LAB203-XX\Documents\GitHub\Functional-Programming-2-68\LAB02\src\main\scala\Main.scala:78:21
[error] 78 |  else n * factorial4(n - 1) // เนื่องจากเป็น tail call แต่มีการคูณ *n ทำให้ไม่ใช่ recursive cal
[error]    |           ^^^^^^^^^^^^^^^^
[error]    |           Cannot rewrite recursive call: it is not in tail position
[error] one error found
[error] (Compile / compileIncremental) Compilation failed
[error] Total time: 3 s, completed 16 ธ.ค. 2568 14:21:08
PS C:\Users\LAB203-XX\Documents\GitHub\Functional-Programming-2-68\LAB02>
```

```
println("Exercise 5: @tailrec")
println(factorial2(5))
println(factorial3(5))
// println(factorial4(5))
println("---------------")
```

```
---------------
Exercise 5: @tailrec
120
120
---------------
```

```
def factorial2(n: Int, acc: Int = 1): Int =
  if (n <= 1) acc
  else factorial2(n - 1, acc * n) // Tail Call

@tailrec
def factorial3(n: Int, acc: Int = 1): Int = {
  if (n <= 1) acc
  else factorial3(n - 1, acc * n)
}

// @tailrec
// def factorial4(n: Int): Int = {
//   if (n <= 1) 1
//   else n * factorial4(n - 1) // ไม่ใช่ tail call เพราะมี *n หลังจาก recursive call
// }
```

Exercise 6: @tailrec -2

ทดสอบดังต่อไปนี้

```
// DON'T DO THIS!
def last[A](list: List[A]): A = list match
  case Nil        => throw new Exception("last(empty)")
  case _ :: tail  => last(tail)
//DO THIS
@tailrec
def last2[A](list: List[A]): A = list match
  case Nil        => throw new Exception("last(empty)")
  case head :: tail => if tail.isEmpty then head else last2(tail)
```

```
println("Exercise 5: @tailrec - 2")
println(last(List(1,2,3)))
println(last2(List(1,2,3)))
```

```
[error] java.lang.Exception: last(empty)
[error]         at Main$package$.last(Main.scala:87)
[error]         at Main$package$.main(Main.scala:29)
[error]         at main.main(Main.scala:3)
[error]         at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
[error]         at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:77)
[error]         at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
[error]         at java.base/java.lang.reflect.Method.invoke(Method.java:568)
[error] stack trace is suppressed; run last Compile / run for the full output
[error] (Compile / run) java.lang.Exception: last(empty)
[error] Total time: 4 s, completed 16 ธ.ค. 2568 14:31:38
PS C:\Users\LAB203-XX\Documents\GitHub\Functional-Programming-2-68\LAB02>
```

```
println("Exercise 6: @tailrec - 2")
// println(last(List(1,2,3)))
println(last2(List(1,2,3)))
```

```
Exercise 6: @tailrec - 2
3
```

Exercise 7: Recursion on Multiple/Nested Lists

ทดสอบดังต่อไปนี้

```
def concat[A](list1: List[A], list2: List[A]): List[A] = list1 match
  case Nil          => list2
  case head1 :: tail1 => head1 :: concat(tail1, list2)

def append[A](list: List[A], value: A): List[A] = ???

def flatten[A](list: List[List[A]]): List[A] = list match
  case Nil          => Nil
  case head :: tail => ???
```

```
def concat[A](list1: List[A], list2: List[A]): List[A] = list1 match
    case Nil          => list2
    case head1 :: tail1 => head1 :: concat(tail1, list2)

def append[A](list: List[A], value: A): List[A] =
  concat(list,List(value))

def flatten[A](list: List[List[A]]): List[A] = list match
    case Nil          => Nil
    case head :: tail => concat(head, flatten(tail))
```

```
println("Exercise 7: Recursion on Multiple/Nested Lists")
println(append(List(1,2,3),4))
println(flatten(List(List(1,2,3))))
```

```
Exercise 7: Recursion on Multiple/Nested Lists
List(1, 2, 3, 4)
List(1, 2, 3)
```

Exercise 8: Recursion on Sublists Other Than the Tail

ทดสอบดังต่อไปนี้

```
// DON'T DO THIS!
def group1[A](list: List[A], k: Int): List[List[A]] =
  take(list, k) :: group1(drop(list, k), k)

def isEmpty[A](list: List[A]): Boolean = list match
  ???
  ???
def group[A](list: List[A], k: Int): List[List[A]] =
  if isEmpty(list) then Nil
  else
    val (first, more) = splitAt(list, k)
    first :: ???
```

วางcode และผลลัพธ์ที่นี่

Exercise 9: Sorting

ทดสอบดังต่อไปนี้

```
def insertInSorted(x: Int, sorted: List[Int]): List[Int] = sorted match
  case Nil          => List(x)
  case min :: others =>
    if x < min then x :: sorted else min :: insertInSorted(x, others)

def insertSort(list: List[Int]): List[Int] = list match
  case Nil    => list
  case h :: t => insertInSorted(h, insertSort(t))
```

```
def insertInSorted(x: Int, sorted: List[Int]): List[Int] = sorted match
    case Nil          => List(x)
    case min :: others =>
        if x < min then x :: sorted else min :: insertInSorted(x, others)

def insertSort(list: List[Int]): List[Int] = list match
    case Nil    => list
    case h :: t => insertInSorted(h, insertSort(t))
```

```
println("Exercise 9: Sorting")
println("Before : List(3, 5, 2, 6, 1, 4)")
print("After : ")
println(insertSort(List(3, 5, 2, 6, 1, 4)))
```

```
Exercise 9: Sorting
Before : List(3, 5, 2, 6, 1, 4)
After : List(1, 2, 3, 4, 5, 6)
```