Lab 03 : Pure Function

	Student ID	Name
1	67070159	วราภรณ์ สุขม่วง

Objectives:

1. (CLO1) สามารถอธิบายกระบวนคิด และหลักการในการทำงานของโปรแกรมเชิงฟังก์ชั่นทางคณิตศาสตร์

Pure Function

คือฟังก์ชันที่มีคุณสมบัติ 2 ข้อ ประกอบด้วย

- **Deterministic:** ใส่ Input เดิม ต้องได้ Output เดิมเสมอ (ห้าม Random, ห้ามใช้ Date.now)
- **No Side Effects:** ไม่ส่งผลกระทบใดๆ ต่อโลกภายนอกฟังก์ชัน

Side Effects

Side Effect คือ ผลข้างเคียง หรือ ผลลัพธ์ที่ไม่ได้ต้องการให้เกิดขึ้น เป็นสิ่งที่เกิดขึ้นนอกเหนือจากการคืนค่า (Return value) เช่น

- **State Change/ Mutation:** การเปลี่ยนค่าของ state ที่มีอยู่แล้ว หรือ การเปลี่ยนค่าตัวแปร Global หรือตัวแปรใน Object (this.count++)
- **I/O Operations:** ฟังก์ชันไม่ได้ทำ "หน้าที่เดียว" เช่น การ Print ออกหน้าจอ, การเขียนไฟล์, การต่อ Database
- **Data Mutation:** แก้ object หรือ data structure เดิม หรือ การแก้ไขข้อมูลใน Memory เดิม
- **Hidden Dependency:** ฟังก์ชันใช้ข้อมูลที่ ไม่อยู่ใน argument ทำให้ไม่รู้ว่าค่าเปลี่ยนจากที่ไหน
- **Exception:** การโยน Error ที่ทำให้โปรแกรมหยุดทำงาน

- **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: Global Counter (State Mutation)

Code เดิม (Java - Impure)

```
public int total = 0;


public void addToTotal(int n) {
    total = total + n;
    System.out.println("Current total: " + total);
}
```

โจทย์: จงเขียนฟังก์ชัน addToTotal ใหม่ใน Scala ให้เป็น Pure Function

- ห้าม ใช้ var ภายนอก
- ห้าม println ในฟังก์ชัน
- Input: รับค่าปัจจุบัน (currentTotal) และค่าที่จะบวก (n)
- Output: คืนค่าผลรวมใหม่

ตัวอย่าง

```
// Refactor ให้เป็น Pure
def addToTotal(currentTotal: Int, n: Int): Int =
    ???


// Test Case
val start = 0
val step1 = addToTotal(start, 5)
val step2 = addToTotal(step1, 10)
println(step2)
```

```scala
@main def main() : Unit = {
  println("---------------")
  println("Exercise 1: Global Counter (State Mutation)")
  val start = 0
  val step1 = addToTotal(start, 5)
  val step2 = addToTotal(step1, 10)
  println(step2)
  println("---------------")
  println("Exercise 2: Check Grade (Mixing Logic & I/O)")
  val result = checkGrade(75)
  println(result)
  println("---------------")
  println("Exercise 3: Normalize Name (Data Mutation)")
  val users = List("  somCHAI ", " maNEe  ")
  val cleanUsers = normalizeNames(users)
  println(users)      // Expected: List("  somCHAI ", " maNEe  ") (เหมือนเดิม)
  println(cleanUsers) // Expected: List("Somchai", "Manee") (ของใหม่)
  println("---------------")
  println("Exercise 4: Time Bomb (Hidden Dependency)")
  println(getDiscountPrice(100.0, DayOfWeek.MONDAY))  // Expected: 50.0
  println(getDiscountPrice(100.0, DayOfWeek.FRIDAY))  // Expected: 100.0
  println("---------------")
}
```

```
Exercise 1: Global Counter (State Mutation)
15
```

```scala
def addToTotal(currentTotal: Int, n: Int): Int =
  currentTotal + n
```

Exercise 2: Check Grade (Mixing Logic & I/O)

Code เดิม (Python - Impure)

```
def check_grade(score):
    if score >= 50:
        print("Congratulations! You passed.")
    else:
        print("Sorry, you failed. Try again.")


#Test Case
check_grade(75)
```

โจทย์: จงเขียนฟังก์ชันที่คืนค่า String แทนการ Print

- ฟังก์ชันนี้คำนวณเกรดและ Print ออกมาทันที ทำให้เราเอาผลลัพธ์ไปใช้ต่อไม่ได้ (เช่น เอาไปลง Database ต่อไม่ได้) และ Test ยากเพราะต้องไปดักจับ stdout

ตัวอย่าง

// Refactor ให้เป็น Pure

```
def checkGrade(score: Int): String =
    ???
```

// Test Case

```
val result = checkGrade(75)
println(result) // Print เพื่อตรวจสอบผลลัพธ์
```

```
@main def main() : Unit = {
  println("---------------")
  println("Exercise 1: Global Counter (State Mutation)")
  val start = 0
  val step1 = addToTotal(start, 5)
  val step2 = addToTotal(step1, 10)
  println(step2)
  println("---------------")
  println("Exercise 2: Check Grade (Mixing Logic & I/O)")
  val result = checkGrade(75)
  println(result)
  println("---------------")
  println("Exercise 3: Normalize Name (Data Mutation)")
  val users = List("  somCHAI ", " maNEe  ")
  val cleanUsers = normalizeNames(users)
  println(users)      // Expected: List("  somCHAI ", " maNEe  ") (เหมือนเดิม)
  println(cleanUsers) // Expected: List("Somchai", "Manee") (ของใหม่)
  println("---------------")
  println("Exercise 4: Time Bomb (Hidden Dependency)")
  println(getDiscountPrice(100.0, DayOfWeek.MONDAY))  // Expected: 50.0
  println(getDiscountPrice(100.0, DayOfWeek.FRIDAY))  // Expected: 100.0
  println("---------------")
}
```

```
def checkGrade(score: Int): String = {
  if (score >= 50) then "Congratulations! You passed."
  else "Sorry, you failed. Try again."
}
```

```
Exercise 2: Check Grade (Mixing Logic & I/O)
Congratulations! You passed.
```

Exercise 3: Normalize Name (Data Mutation)

Code เดิม (Python - Impure)

```
def normalize_names(names):
    for i in range(len(names)):
        # Side Effect: แก้ไขข้อมูลใน Memory เดิม
        names[i] = names[i].strip().capitalize()


# ใช้งาน
users = ["  somCHAI ", " maNEe  "]
normalize_names(users)
print(users) # ['Somchai', 'Manee'] -> ข้อมูลเดิมเปลี่ยนไปแล้ว!
```

โจทย์: จงใช้ map เพื่อสร้าง List ใหม่ที่สะอาด โดยที่ List เดิมต้องยังคงสภาพเดิม

- Python List เป็น Mutable การแก้ไขค่าในลูป (names[i] = ...) จะเปลี่ยนข้อมูลต้นฉบับ ถ้ามีส่วนอื่นของโปรแกรมใช้ List นี้อยู่ อาจจะพังได้ (Bug ที่ตามหายากที่สุด)
- **Hint: .map / .trim / .capitalize**
  - **(1 to 5).map(x => x*x) ฟังก์ชันที่ไม่ระบุชื่อ : เพื่อใช้ arg สองครั้ง ต้องตั้งชื่อ**
  - **.trim() ใช้สำหรับ ลบช่องว่าง (whitespace) ที่อยู่ด้านหน้าและด้านหลังสุดของข้อมูลประเภท String ให้หมดไป**
  - **.capitalize สำหรับสตริงเพื่อเปลี่ยนแค่ตัวอักษรแรกเป็นตัวพิมพ์ใหญ่ (Capitalize)**

ตัวอย่าง

// Refactor ให้เป็น Pure

```
def normalizeNames(names: List[String]): List[String] =
    ???


// Test Case
val users = List("  somCHAI ", " maNEe  ")
val cleanUsers = normalizeNames(users)

println(users)      // Expected: List("  somCHAI ", " maNEe  ") (เหมือนเดิม)
println(cleanUsers) // Expected: List("Somchai", "Manee") (ของใหม่)
```

```
@main def main() : Unit = {
  println("---------------")
  println("Exercise 1: Global Counter (State Mutation)")
  val start = 0
  val step1 = addToTotal(start, 5)
  val step2 = addToTotal(step1, 10)
  println(step2)
  println("---------------")
  println("Exercise 2: Check Grade (Mixing Logic & I/O)")
  val result = checkGrade(75)
  println(result)
  println("---------------")
  println("Exercise 3: Normalize Name (Data Mutation)")
  val users = List("  somCHAI ", " maNEe  ")
  val cleanUsers = normalizeNames(users)
  println(users)      // Expected: List("  somCHAI ", " maNEe  ") (เหมือนเดิม)
  println(cleanUsers) // Expected: List("Somchai", "Manee") (ของใหม่)
  println("---------------")
  println("Exercise 4:  Time Bomb (Hidden Dependency)")
  println(getDiscountPrice(100.0, DayOfWeek.MONDAY))  // Expected: 50.0
  println(getDiscountPrice(100.0, DayOfWeek.FRIDAY))  // Expected: 100.0
  println("---------------")
}
```

```
Exercise 3: Normalize Name (Data Mutation)
List(  somCHAI ,  maNEe  )
List(Somchai, Manee)
```

```
def normalizeNames(names: List[String]): List[String] =
  names.map( name => name.trim.toLowerCase.capitalize )
```

Exercise 4: Time Bomb (Hidden Dependency)

Code เดิม (Python - Impure)

```
import datetime


def get_discount_price(price):
    now = datetime.datetime.now() # Side Effect: ขึ้นอยู่กับเวลาโลกจริง
    # ถ้าเป็นวันจันทร์ ลด 50%
    if now.weekday() == 0:
        return price * 0.5
    else:
        return price


# ใช้งาน
print(get_discount_price(100))
```

โจทย์: จงใช้หลักการ Dependency Injection โดยรับตัวแปร DayOfWeek เข้ามาแทนการดึงเวลาเอง เพื่อให้เรา Test ได้ทุกสถานการณ์

- ฟังก์ชันนี้ขึ้นอยู่กับ datetime.now() ซึ่งเป็น Hidden Input (ควบคุมไม่ได้)
- ถ้าเราอยาก Test ว่า "โปรโมชั่นหมดอายุแล้วจะเกิดอะไรขึ้น" เราต้องนั่งรอให้เวลาผ่านไปจริงๆ หรือไป Hack นาฬิกาเครื่อง (ซึ่งแย่มาก)

ตัวอย่าง

// Refactor ให้เป็น Pure

```
import java.time.DayOfWeek
def getDiscountPrice(price: Double, day: DayOfWeek): Double =
    ???


// Test Case
println(getDiscountPrice(100.0, DayOfWeek.MONDAY))  // Expected: 50.0
println(getDiscountPrice(100.0, DayOfWeek.FRIDAY))  // Expected: 100.0
```

```
@main def main() : Unit = {
  println("---------------")
  println("Exercise 1: Global Counter (State Mutation)")
  val start = 0
  val step1 = addToTotal(start, 5)
  val step2 = addToTotal(step1, 10)
  println(step2)
  println("---------------")
  println("Exercise 2: Check Grade (Mixing Logic & I/O)")
  val result = checkGrade(75)
  println(result)
  println("---------------")
  println("Exercise 3: Normalize Name (Data Mutation)")
  val users = List("  somCHAI ", " maNEe  ")
  val cleanUsers = normalizeNames(users)
  println(users)      // Expected: List("  somCHAI ", " maNEe  ") (เหมือนเดิม)
  println(cleanUsers) // Expected: List("Somchai", "Manee") (ของใหม่)
  println("---------------")
  println("Exercise 4:  Time Bomb (Hidden Dependency)")
  println(getDiscountPrice(100.0, DayOfWeek.MONDAY))  // Expected: 50.0
  println(getDiscountPrice(100.0, DayOfWeek.FRIDAY))  // Expected: 100.0
  println("---------------")
}
```

```
import java.time.DayOfWeek
def getDiscountPrice(price: Double, day: DayOfWeek): Double =
  if day == DayOfWeek.MONDAY then price * 0.5
  else price
```

```
---------------
Exercise 4:  Time Bomb (Hidden Dependency)
50.0
100.0
---------------
```