

# Pattern Matching

---

06016415 Functional Programming

- Pattern Matching
  - Syntax
- Values, Variables, and Types in Matches
- Matching on
  - Sequences
  - Repeated Parameters
  - Functional Lists
  - Tuples
  - Case Clauses
  - others

- **Pattern Matching** คือ วิธีการจับคู่ หรือ ตรวจสอบ **ค่า(word)** กับ **รูปแบบ (Pattern)** ซึ่งมักจะกำหนดไว้ในภาษาโปรแกรมส่วนมาก รวมถึง **scala** ที่เป็นแบบ **hybrid** (ผสมระหว่างแนวคิด **Functional** กับ **Object**) ด้วย
- จุดเด่น คือ ช่วยให้การดำเนินการจับคู่ระหว่าง รูปแบบ (**Pattern**) ในกรณีต่างๆ มีความยืดหยุ่น และรัดกุมมากขึ้น
- มีโครงสร้างคล้าย **switch** แบบทั่วไป ซึ่งสามารถแทน **if-else** ที่ใช้ต่อเนื่องกันได้ด้วย
- สามารถใช้กับ **lists, sequences, tuples, data types** และอื่นๆ

- ใน Scala จะใช้ **match** และ **case** เป็นคำสั่งหลัก
- มีโครงสร้างคล้าย **switch** แบบทั่วไป ซึ่งสามารถแทน **if-else** ที่ใช้ต่อเนื่องกันได้ด้วย

```
x match {  
  case 1 => println("One")  
  case 2 => println("Two")  
  case _ => println("Other")  
}
```

```
def patternMatching(candidate: String): Int =  
  candidate match  
    case "One" => 1  
    case "Two" => 2  
    case _ => -1
```

แม้ว่าจะเป็น การตรวจสอบค่ากับรูปแบบ (Pattern) แต่ไม่ได้ตรวจสอบรูปแบบที่ตรงกันทั้งหมด (exact matching)

```
import scala.util.Random

val x: Int = Random.nextInt(10)
x match
  case 0 => "zero"
  case 1 => "one"
  case 2 => "two"
  case _ => "other"

println(x)
```

```
✓ def matchTest(x: Int): String = x match
  case 1 => "one"
  case 2 => "two"
  case _ => "other"

val seq = Seq(1, 2, 3.14, 5.5F, "one", "four", true, (6, 7))
✓ val result = seq.map {
  case 1 => "int 1"
  case i: Int => s"other int: $i"
  case d: (Double | Float) => s"a double or float: $d"
  case "one" => "string one"
  case s: String => s"other string: $s"
  case (x, y) => s"tuple: ($x, $y)"
  case unexpected => s"unexpected value: $unexpected"
}
```

```
val langs = Seq(  
  ("Scala", "Martin", "Odersky"),  
  ("C langs", "Dennis", "Ritchie"),  
  ("SQL", "Donald D.", "Chamberlin"))  
val results = langs.map {  
  case ("Scala", _, _) => "Scala"  
  case (lang, first, last) => s"$lang, creator $first $last"  
}
```

```
//Matching on Sequence  
def seqToString[T](seq: Seq[T]): String = seq match  
  case head +: tail => s"($head +: ${seqToString(tail)})"  
  case Nil => "Nil"
```

```
def matchThree(seq: Seq[Int]) = seq match
  case Seq(h1, h2, rest*) => // same as h1 +: h2 +: rest => ...
    println(s"head 1 = $h1, head 2 = $h2, the rest = $rest")
  case _ => println(s"Other! $seq")
```



```
//Revising Funcional List
def head[A](list: List[A]): A = list match
  | case h :: _ => h
  | case Nil    => throw new Exception("head(empty)")

def tail[A](list: List[A]): List[A] = list match
  | case _ :: t => t
  | case Nil    => throw new Exception("tail(empty)")

def isEmpty[A](list: List[A]): Boolean = list match
  | case Nil => true
  | case _  => false
```

```
def tuplesPatternMatching(tuple: Any): String =  
  tuple match  
    case (first, second) => s"I'm a tuple with two elements: $first & $second"  
    case (first, second, third) => s"I'm a tuple with three elements: $first & $second & $third"  
    case _ => s"Unrecognized pattern. My value: $tuple"
```

```
scala> val tub = ("Hello", 1, 2.3)  
val tub: (String, Int, Double) = (Hello, 1, 2.3)
```

```
scala> tuplesPatternMatching(tub)  
val res0: String = I'm a tuple with three elements: Hello & 1 & 2.3
```

```
scala> tuplesPatternMatching(1)  
val res1: String = Unrecognized pattern. My value: 1
```

```
def typedPatternMatching(any: Any): String =  
  any match  
    case string: String => s"I'm a string. My value: $string"  
    case integer: Int => s"I'm an integer. My value: $integer"  
    case _ => s"I'm from an unknown type. My value: $any"
```

```
val results2 = Seq(1,2,3,4).map {  
  case e if e%2 == 0 => s"even: $e"  
  case o => s"odd: $o"  
}
```

## Find Student Grade

```
def grade(marks: Int): String = marks match  
  case x if x >= 75 => "A"  
  case x if x >= 65 => "B"  
  case x if x >= 50 => "C"  
  case _ => "F"
```

```
scala> grade(50)  
val res6: String = C
```

## Pass or Fail

```
def passFail(marks: Int): String = marks >= 50 match  
  case true => "PASS"  
  case false => "FAIL"
```

```
scala> passFail(49)  
val res7: String = FAIL
```

## NOTE: Case Class in Scala3

Immutable Data  
Holder

- **Case Class** คือ Class พิเศษที่ถูกออกแบบมาเพื่อ "เก็บข้อมูล" (Immutable Data Holder) โดยเฉพาะ
- ในทางทฤษฎี ADTs เราเรียกมันว่า **Product Type** (ความสัมพันธ์แบบ "และ")  
เช่น *Person* ประกอบด้วย ชื่อ และอายุ และที่อยู่

```
case class Address(street: String, city: String)
case class Person(name: String, age: Int, address: Address)
```

- เมื่อใช้คำสั่ง **case class** ทำให้
  - เกิด Immutable Fields ตัวแปรข้างในเป็น **val** (ค่าคงที่) โดยอัตโนมัติ ห้ามแก้ไข
  - สร้าง Object ได้เลย เช่น `val s1 = Student("Somchai", 101, 3.50)`
  - มี Built-in Methods เช่น `toString`, `equals`, `hashCode` มาให้เลย (เปรียบเทียบข้อมูลข้างในได้ทันที ไม่ได้เปรียบเทียบที่ Memory Address)
  - มี Copy Method เช่น `.copy()` สำหรับสร้าง Object ใหม่ที่แก้ไขค่าบางอย่าง (เพราะแก้ตัวเดิมไม่ได้)
  - ใน Pattern Matching จะรองรับการ "แกะข้อมูล" ออกมาได้ด้วย `match case`

```
//Matching on case classes
case class Address(street: String, city: String)
case class Person(name: String, age: Int, address: Address)
val alice = Person("Alice", 25, Address("1 Scala Lane", "Chicago"))
val bob = Person("Bob", 29, Address("2 Java Ave.", "Miami"))
val charlie = Person("Charlie", 32, Address("3 Python Ct.", "Boston"))

val results3 = Seq(alice, bob, charlie).map {
  case p @ Person("Alice", age, a @ Address(_, "Chicago")) =>
    s"Hi Alice! $p"
  case p @ Person("Bob", 29, a @ Address(street, city)) =>
    s"Hi ${p.name}! age ${p.age}, in ${a}"
  case p @ Person(name, age, Address(street, city)) =>
    s"Who are you, $name (age: $age, city = $city)?"
}
```

## Leap Years

```
def leapYear(year: Int): Boolean = ((year%4 == 0) && (year%100 != 0) || (year%400 == 0)) match
  case true => true
  case false => false
```

## Pattern Matching on Custom Objects

```
scala> case class Person(name: String, age: Int)
// defined case class Person
```

```
scala> val person = Person("Alice", 30)
val person: Person = Person(Alice,30)
```

```
scala> person match
|   case Person("Alice", age) if age < 40 => println("Young Alice")
|   case Person("Alice", age) => println("Alice")
|   case Person(name, age) if age < 40 => println(s"Young $name")
|   case _ => println("Other")
|
Young Alice
```



Value matching

```
def f(x: Int): String = x match
  case 1 | 2 => "one or two"
  case 3 => "three"
  case _ => "other values"
```

Type matching

```
def f(x: Any): String = x match
  case i: Int => "integer: " + i
  case _: Double => "a double"
  case s: String => "I want to say " + s
```

List structure matching

```
def sum(xs: List[Int]): Int =
  xs match
    case x :: tail => x + sum(tail)
    case Nil => 0
```

- **Pattern Matching** เป็นคุณลักษณะทั่วไปของภาษาการโปรแกรมเชิงฟังก์ชัน ซึ่งมีรูปแบบที่ง่าย และมีประสิทธิภาพ
- มีรูปแบบคล้าย **switch** แต่ใช้ **keyword** ว่า **match** กับ **case**
- สามารถใช้ตรวจสอบค่าต่างๆ กับรูปแบบ เช่น **Lists, Sequences, Tuples, Data Type** และอื่นๆ
- **Pattern Matching** มีประโยชน์เมื่อใช้เพื่อ
  - การดำเนินการต่างๆใน **Lists** เช่น การหาผลรวม เป็นต้น
  - แยก **Empty** กับ **Non-Empty Lists**
  - ใช้ร่วมกับ **head** และ **tail**