

**FEATURES:
HIGHER
ORDER & CURRYING**

PARTIALLY APPLIED FUNCTION

```
object FunctionPartiallyApplied {  
  def mul(x:Double, y:Double): Double = {  
    x*y  
  }  
  def partialMul(y:Double):Double = {  
    mul(3, y)  
  }  
}
```

logic

type function

```
def main(args: Array[String]): Unit = {  
  val sum = (x: Double, y: Double, z: Double) => x + y + z //fully applied function  
  val f = sum(3, 5, -) Double  
  println(f(2))  
  println(partialMul(3))  
}
```

ใส่แค่อันเดียวก็ได้เลย

```
E:\Drop  
10.0  
9.0
```

PARTIALLY APPLIED FUNCTION (APPLICATION)

Java

Import JAVA ได้

```
import java.util.Date

object FunctionPartiallyAppliedApplication {
  def dateMessage(date: Date, s: String): Unit = {
    println(date + ", " + s)
  }

  def main(args: Array[String]): Unit = {
    var date = new Date
    var newMessage = dateMessage(date, _:String)
    for(i: Int <- 0 ≤ .to(≤ 5)) {
      Thread.sleep( millis = 300)
      date = new Date
      newMessage("message " + i)
    }
  }
}
```

declare variable

partial applied

Date ล้าง

ใช้ค่าล่าสุด "เสมอ"
(ใช้ date ใหม่)

Java

```
Mon Feb 14 18:35:57 ICT 2022, message 0
Mon Feb 14 18:35:57 ICT 2022, message 1
Mon Feb 14 18:35:57 ICT 2022, message 2
Mon Feb 14 18:35:58 ICT 2022, message 3
Mon Feb 14 18:35:58 ICT 2022, message 4
Mon Feb 14 18:35:58 ICT 2022, message 5
```

คือถ้าทำ partially โดยใส่ตัวแปร parameter ซ่อนนั้นจะเปลี่ยนตามค่าตัวแปรนั้นเสมอใช่ไหมครับ

ถาม ใช่

CLOSURE

ฟังก์ชันที่ใช้ตัวแปร declare

- A function that uses variable(s) declared outside the function.

```
object Closure {  
  var n = 5  
  val add = (x:Int) => x+n    //closure with n coming from outside  
  def main(args: Array[String]): Unit = {  
    println(add(2)) → 7      //closure with add coming from outside  
    n = 100  
    println(add(2)) → 102  
  }  
}
```

add ถือว่าเป็น closure โดย n มาจากข้างนอก


"เปลี่ยนค่าเดิม"

```
object Closure {  
  var n = 5  
  val add = (x:Int) => x+n    //closure with n coming from outside  
  def main(args: Array[String]): Unit = {  
    println(add(2)) → 7      //closure with add coming from outside  
    var n = 100  
    println(add(2)) → 7  
    println(add(2)) → 7  
  }  
}
```

ถ้ามีการ declare n หลายๆรอบ
จะใช้ไหนหรือครับ
>> use static scoping krub

most languages use static
scoping including scala

CLOSURE – WITH SIDE EFFECT ALLOWED ON VARIABLE (IMPURE CLOSURE)

```
object ClosureSideEffect {  
  var n = 5  
  val add = (x:Int) => {  
    100  
    n = x + n  modify n  
  }  
  //closure with n coming from outside  
  
  def main(args: Array[String]): Unit = {  
    println(add(2))  
    n = 100  
    println(add(2)) → 102  
    println(add(2)) → 104  
  }  
}
```

* เปลี่ยนตัวแปรสลับกับ Execute

!!
เพราะ ทุกๆครั้งเปลี่ยนเป็น 102 แล้ว

WHAT IS FUNCTIONAL PROGRAMMING?

- No changing variable.
- No assignment
- No loop
- Just focusing on functions.
- Functions can be defined anywhere, including in other functions.
- Functions can be passed as parameters and returned as results.
- There are operators that can compose functions.

WHAT ARE GOOD ABOUT FUNCTIONAL PROGRAMMING?

- Simpler reasoning.
- Good for multicore and cloud computing.
 - Avoid modifying variables by different parts of the program.
- Places to use (where we want scalable solutions)
 - Web
 - Trading platforms
 - Simulation

EVALUATING FUNCTION == EVALUATING EXPRESSION

- This substitution model (evaluating until getting a value) can be used as long as the function has no side effect.

- square(square(2))
- square(4)
- 16

- Example of side effect (cannot be expressed in a substitution model)

- x++

เปลี่ยนมีเครื่องหมายอื่นอีกที่

call by value : $\text{add}(3+4, 5+1)$

↓

$\text{add}(7, 5+1)$

↓

$\text{add}(7, 6)$ ถึงเริ่ม evaluate

call by name : $(3+4) + (5+1)$

RECURSION IS IMPORTANT IN THIS PARADIGM.

- Need to be able to think of it instead of loop.
- Recursion can be optimized to use only 1 stack frame (if you convert it to tail-recursion)
- But first, you must be more familiar with recursion.

PARENTHESIS BALANCING EXERCISE (RECURSIVE 15 MINS)

- `def balance(chars: List[Char]): Boolean`
- `()()` -> must return false
 - `chars.isEmpty`: Boolean checks if the list is empty.
 - `chars.head`: Char returns the first element of the list.
 - `chars.tail`: List[Char] returns the list with the first element removed.
 - In your test, you can use the `toList` method to convert from a String to a List[Char]:
 - e.g. `"I (John McClain) is a Die Hard fan".toList`.

```
object Parenthesis {  
  def balance(chars: List[Char]): Boolean = {  
    balance(chars, acc = 0)  
  }  
  
  def balance(chars: List[Char], acc: Int): Boolean = {  
    if (chars.isEmpty && acc == 0) true  
    else if (chars.isEmpty && acc != 0) false  
    else if (acc < 0) false  
    else if (chars.head != '(' && chars.head != ')') balance(chars.tail, acc)  
    else if (chars.head == '(') balance(chars.tail, acc+1)  
    else balance(chars.tail, acc-1)  
  }  
  
  def main(args: Array[String]): Unit = {  
    println(balance("(if(zero?x) max(/1 x)).toList"))  
  }  
}
```

Handwritten notes in Thai:

- At the top: "นี่คือวิธีที่ฉันทำ" (This is how I did it)
- Next to the first `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the second `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the third `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the fourth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the fifth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the sixth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the seventh `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the eighth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the ninth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the tenth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the eleventh `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the twelfth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the thirteenth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the fourteenth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the fifteenth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the sixteenth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the seventeenth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the eighteenth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the nineteenth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the twentieth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the twenty-first `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the twenty-second `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the twenty-third `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the twenty-fourth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the twenty-fifth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the twenty-sixth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the twenty-seventh `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the twenty-eighth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the twenty-ninth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the thirtieth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the thirty-first `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the thirty-second `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the thirty-third `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the thirty-fourth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the thirty-fifth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the thirty-sixth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the thirty-seventh `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the thirty-eighth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the thirty-ninth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the fortieth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the forty-first `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the forty-second `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the forty-third `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the forty-fourth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the forty-fifth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the forty-sixth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the forty-seventh `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the forty-eighth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the forty-ninth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the fiftieth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the fifty-first `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the fifty-second `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the fifty-third `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the fifty-fourth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the fifty-fifth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the fifty-sixth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the fifty-seventh `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the fifty-eighth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the fifty-ninth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the sixtieth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the sixty-first `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the sixty-second `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the sixty-third `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the sixty-fourth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the sixty-fifth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the sixty-sixth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the sixty-seventh `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the sixty-eighth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the sixty-ninth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the seventieth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the seventy-first `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the seventy-second `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the seventy-third `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the seventy-fourth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the seventy-fifth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the seventy-sixth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the seventy-seventh `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the seventy-eighth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the seventy-ninth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the eightieth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the eighty-first `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the eighty-second `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the eighty-third `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the eighty-fourth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the eighty-fifth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the eighty-sixth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the eighty-seventh `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the eighty-eighth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the eighty-ninth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the ninetieth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the ninety-first `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the ninety-second `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the ninety-third `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the ninety-fourth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the ninety-fifth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the ninety-sixth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the ninety-seventh `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the ninety-eighth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the ninety-ninth `if` condition: "ตัดทิ้ง" (Cut off)
- Next to the hundredth `if` condition: "ตัดทิ้ง" (Cut off)

TAIL RECURSION

- If a function just calls another or call itself without any extra work, the language runtime system can optimize the function to use only one stack frame, just like using a loop.
- If you see a recursive function that is not tail-recursive, trying to make it tail-recursive will help optimize memory (stack frame) usage.

FACTORIAL (NON TAIL-RECURSIVE)

```
object Factorial {  
  def factorial(x: Int): Int = {  
    if (x == 0) return 1  
    x * factorial(x-1)  
  }  
  def main(args: Array[String]): Unit = {  
    println(factorial(4))  
  }  
}
```

Handwritten notes:
- A red circle highlights the expression `x * factorial(x-1)`.
- A yellow asterisk is written above the `return` keyword.
- A red squiggly line is under the `x-1` argument.
- Yellow text below the closing brace of `factorial` says "else ตรงนี้ก็" (else this part too).

FACTORIAL (TAIL-RECURSIVE)

-EXERCISE 5 MINS

↳ using stack

```
object FactorialTail {  
  def factorial(x: Int, acc: Int): Int = {  
    if (x == 0) return acc  
    return factorial(x-1, x*acc)  
  }  
  
  def main(args: Array[String]): Unit = {  
    println(factorial(4, acc = 1))  
  }  
}
```

HIGHER ORDER FUNCTION

Take
functions as
arguments.

Can return
function.

```
object FunctionHigherOrder {  
  def calculate(x: Double, y: Double, myF: (Double, Double) => Double): Double = {  
    myF(x, y)  
  }  
}
```

Function as parameter

```
def mul(x: Double, y: Double): Double = x * y
```

```
def main(args: Array[String]): Unit = {  
  println(calculate(3, 5, (a, b) => a + b))  
  println(calculate(3, 5, mul))  
}
```

eg think my F is

E:\Drop

8.0

15.0

CHAINING FUNCTIONS

```
object FunctionChain {  
  def calculate(x: Double, y: Double, z: Double, myF: (Double, Double) => Double): Double = {  
    myF(myF(x, y), z)  
  }
```

💡 *ไม่ใช่ global (จะเห็นภายใน FunctionChain)*

```
def mul(x: Double, y: Double): Double = x * y → ใช้นี้เป็น lambda function  
val = mul { } นี่
```

```
def main(args: Array[String]): Unit = {  
  println(calculate(3, 5, 7, (a, b) => a + b)) } same  
  println(calculate(3, 5, 7, _+_))  
  println(calculate(3, 5, 7, นี่ mul)) นี่คือสิ่งที่เรากำลังใช้ เช่น - + -  
  println(calculate(3, 5, 7, (a, b) => a min b)) } same  
  println(calculate(3, 5, 7, _ min _))  
}
```

def => define function (evaluate every time)
val => value (evaluate when defined)
lazy => evaluate when called

LET'S DEFINE $\sum_{n=a}^b f(n)$ WHERE F CAN BE ANY FUNCTION

f {

```
object FunctionHigherOrderSum {  
  def sum(f: Int => Int, a: Int, b: Int): Int = {  
    if (a > b) 0  
    else f(a) + sum(f, a+1, b)  
  }  
    
  def id(a: Int): Int = a  
  def square(a: Int): Int = a * a  
  def factorial(x: Int, acc: Int): Int = {  
    if (x == 0) return acc  
    return factorial(x-1, x*acc)  
  }  
  def fac(a: Int): Int = factorial(a, acc = 1)  
    
  def main(args: Array[String]): Unit = {  
    println(sum(id, 2, 4)) // 2+3+4  
    println(sum(square, 2, 4)) // 2^2 + 3^2 + 4^2  
    println(sum(fac, 2, 4)) // 2! + 3! + 4!  
  }  
}
```

$\sum_{n=a}^b f(n)$ CAN BE WRITTEN USING TAIL RECURSION TOO (EXERCISE – 5 MINS)

- Write only the definition of function sum

```
def sum(f: Int => Int, a: Int, b: Int): Int = {  
  def sumAcc(a: Int, acc: Int): Int = {  
    if(a > b) acc  
    else sumAcc(a+1, acc+f(a))  
  }  
  sumAcc(a, acc = 0)  
}
```

ไม่ต้องส่ง b เพราะ b มีค่าแน่นอนอยู่แล้ว ประกอบกับ ค่า b ไม่เปลี่ยนแปลง

CURRYING - FUNCTION AS RETURN VALUE

- Function with multiple arguments → เปลี่ยนให้เป็น
 - Function with one argument, returning another function.

```
val sum30 = addCurryShort(30)
println(sum30(1))
```

เปลี่ยน var ให้
(ไม่ใช้แล้ว)

```
object Currying000 {
  def add(x:Int,y:Int): Int = {
    x+y
  }

  def addCurry(x:Int): Int => Int = {
    (y:Int) => x+y
  }

  def addCurryShort(x:Int)(y:Int):Int = x+y

  def main(args: Array[String]): Unit = {
    println(addCurry(3)(5))

    val sum20 = addCurry(20) //yes, it's partial execution
    println(sum20(7))
    println(addCurryShort(3)(5))
  }
}
```

เป็น function

use for partial execution

underscore ใช้สำหรับการเว้นค่าไว้ (ต้องใส่ต่อทีหลัง)

เพราะกลับค่าไว้ได้ค่าหนึ่ง

แล้วค่อยใส่ 7 เข้าไปที่หลัง

ต้องลองใส่

2 ชุด

CURRYING — Example on $\sum_{n=a}^b f(n)$

```
object Currying {  
  def sum(f: Int => Int): (Int, Int) => Int = {  
    def sumF(a: Int, b: Int): Int = {  
      if(a > b) 0  
      else f(a) + sumF(a+1, b)  
    }  
    sumF  
  }  
}
```

```
def main(args: Array[String]): Unit = {  
  println(sum(id)(2, 4)) // 2+3+4  
  println(sum(square)(2, 4)) // 2^2 + 3^2 + 4^2  
  println(sum(fac)(2, 4)) // 2! + 3! + 4!  
}
```


CURRYING – SPECIAL SYNTAX (MULTIPLE PARAMETER LIST)

```
def sum(f: Int => Int)(a: Int, b: Int): Int = {  
  if(a > b) 0  
  else f(a) + sum(f)(a+1, b)  
}
```

The type of this function is
 $(Int \Rightarrow Int) \Rightarrow ((Int, Int) \Rightarrow Int)$ or $(Int \Rightarrow Int) \Rightarrow (Int, Int) \Rightarrow Int$

Since function types are right associative, so $Int \Rightarrow Int \Rightarrow Int$ is equivalent to $Int \Rightarrow (Int \Rightarrow Int)$

EXERCISE: FACTORIAL IN TERMS OF PRODUCT? – 2 MINS



```
def product(f:Int => Int)(a:Int,b:Int):Int = {  
  if(a>b) 1  
  else f(a) * product(f)(a+1,b)  
}
```

```
def myFac(n: Int):Int = {  
  * product(id)(1,n) ✓  
}  
  
def main(args: Array[String]) {  
  println(product(id)(2,4))  
  println(myFac(4))  
}
```

EXERCISE: WRITE A FUNCTION THAT CAN BE CHANGED TO USE EITHER SUM OR PRODUCT – 5 MINS

- Using the new function, in main, calculate $2+3+4$ and $2^2 * 3^2 * 4^2$

```
def general(f: Int => Int, op: (Int, Int) => Int, startValue: Int)(a: Int, b: Int): Int = {  
  if (a > b) startValue  
  else op(f(a), general(f, op, startValue)(a+1, b))  
}  
  
def main(args: Array[String]): Unit = {  
  println(general(id, (x, y) => x+y, startValue = 0)(2, 4)) // 2+3+4  
  println(general(square, (x, y) => x*y, startValue = 1)(2, 4)) // 2^2 * 3^2 * 4^2  
}
```