บทที่ 5
ฟังก์ชัน

การเขียนโปรแกรมคอมพิวเตอร์สำหรับใช้งานจริงโปรแกรมจะมีขนาดใหญ่ ซึ่งบางครั้งไม่สามารถนำมาเขียนรวมกันได้ ดังนั้น จึงใช้เทคนิคที่เรียกว่า *Divide and Conquer* โดยการแบ่งการทำงานของโปรแกรมออกเป็นส่วนเล็ก ๆ หรือ โมดูล (module) หรือ ฟังก์ชัน (function) สำหรับสร้างและทดสอบโปรแกรมย่อยก่อน จากนั้นนำไปประกอบขึ้นเป็นโปรแกรมใหญ่ที่สมบูรณ์ในขั้นตอนสุดท้าย ซึ่งวิธีการสร้างโปรแกรมขนาดใหญ่จากโปรแกรมย่อยเป็นวิธีการที่มีประสิทธิภาพและดำเนินการได้ง่ายกว่า ดังนั้นในบทนี้จะอธิบายการเขียนโปรแกรมย่อยหรือที่เรียกว่า ฟังก์ชัน (function) โดยมีรายละเอียดดังต่อไปนี้

5.1 ความหมายของฟังก์ชัน

ฟังก์ชัน คือ ชุดคำสั่งที่ถูกเขียนขึ้นเพื่อทำงานเฉพาะของมัน สำหรับในภาษา R นั้น จะมีฟังก์ชันที่ติดมากับภาษาจำนวนมาก นอกจากนี้ผู้ใช้ยังสามารถสร้างฟังก์ชันขึ้นมาใช้เองได้ด้วย

ในภาษา R จะมองฟังก์ชันเป็นออปเจ็ค (object) ดังนั้น ตัวแปรภาษาสามารถส่งผ่านตัวควบคุมไปยังฟังก์ชันพร้อมทั้งอาร์กิวเมนต์ได้ด้วย ซึ่งการทำงานลักษณะนี้เป็นลักษณะพิเศษในการเขียนฟังก์ชันของภาษา R

5.2 การนิยามฟังก์ชัน (function definition)

การสร้างฟังก์ชันในภาษา R จะถูกสร้างขึ้นโดยใช้คำสั่ง function โดยมีรูปแบบการเขียนดังนี้

รูปแบบ :

```
function_name <- function(arg_1, arg_2, ...) {
        Function body
}
```

ส่วนประกอบของฟังก์ชันสามารถอธิบายได้ดังนี้

1) ชื่อฟังก์ชัน (Function Name) ส่วนนี้เป็นชื่อของฟังก์ชัน ซึ่งจะถูกจัดเก็บไว้ในระบบในลักษณะออปเจ็ค

2) อาร์กิวเมนต์ (argument) ใช้สำหรับสั่งผ่านค่าเพื่อไปทำงานยังฟังก์ชัน เมื่อฟังก์ชันถูกเรียกใช้จะทำงานโดยการรับค่าผ่านยังอาร์กิวเมนต์ โดยปกติฟังก์ชันสามารถมีหรือไม่มีอาร์กิวเมนต์ก็ได้ รวมถึงยังสามารถกำหนดค่าเริ่มต้นให้กับอาร์กิวเมนต์ของฟังก์ชันได้เช่นกัน

3) ส่วนการทำงานของฟังก์ชัน (function body) เป็นส่วนที่เป็นคำสั่งที่ใช้ในการทำงานของฟังก์ชันทั้งหมด

4) การส่งค่ากลับ (return value) เป็นส่วนของการส่งค่ากลับ หลังจากทำการประมวลผลการทำงานของฟังก์ชัน (function body) เรียบร้อยแล้ว ค่าจะถูกส่งกลับด้วยคำสั่ง return หรือถ้าไม่มีคำสั่ง return ปกติจะกำหนดให้บรรทัดสุดท้ายของฟังก์ชันจะเป็นส่วนที่ส่งกลับค่าโดยอัตโนมัติ

5.3 ประเภทของฟังก์ชัน

5.3.1 ฟังก์ชันมาตรฐาน (in-built functions) หมายถึง ฟังก์ชันที่ภาษา R เตรียมไว้ให้กับผู้ใช้โดยที่ผู้ใช้สามารถเรียกใช้งานได้ทันที ยกตัวอย่างเช่น ฟังก์ชันทางสถิติที่ใช้วิเคราะห์ข้อมูลพื้นฐาน ฟังก์ชันการจัดการข้อมูล เป็นต้น

5.3.2 ฟังก์ชันที่ผู้ใช้สร้างขึ้นเอง (user-defined function) เป็นฟังก์ชันที่ผู้ใช้งานสร้างขึ้นตามมีวัตถุประสงค์เฉพาะงาน ซึ่งสามารถสร้างได้หลายรูปแบบตามความต้องการของผู้ใช้งาน

รายละเอียดการสร้างฟังก์ชันจะแสดงในหัวข้อถัดไป

5.4 ฟังก์ชันมาตรฐาน (in-built functions)

ภาษา R มีฟังก์ชันมาตรฐานสำหรับเรียกใช้งานเป็นจำนวนมาก ในที่นี้ยกตัวอย่างมาเพียงแค่บางส่วนเท่านั้น โดยรูปแบบการเรียกใช้งานดังนี้

1) ฟังก์ชัน c() เพื่อสร้าง vector หรือ list

รูปแบบ :

ชื่อตัวแปร <- c(1,2)

2) ฟังก์ชัน scan() เป็นการรับค่าข้อมูล

รูปแบบ :

ชื่อตัวแปร <- scan() จะปรากฏส่วนที่ให้ทำการกำหนดค่า

3) ฟังก์ชัน edit() เป็นการแก้ไขข้อมูล

รูปแบบ :

ชื่อตัวแปร<-edit(ชื่อตัวแปร) จะปรากฏหน้าต่างข้อมูลของตัวแปรนั้น ซึ่งสามารถทำการแก้ไขค่าได้

4) ฟังก์ชัน plot() เป็นการ plot กราฟแบบจุด

รูปแบบ :

plot(x,y) โดย x เป็นค่าตามแนวนอนและ y เป็นค่าตามแนว x

5) ฟังก์ชัน lines() เป็นการลากเส้นเชื่อมจุด

รูปแบบ :

lines(x,y) โดย x เป็นค่าตามแนวนอนและ y เป็นค่าตามแนว x

6) ฟังก์ชัน barplot() เป็นการ plot กราฟแบบกราฟแท่ง

รูปแบบ :

barplot (x,y) โดย x เป็นค่าตามแนวนอนและ y เป็นค่าตามแนว x

7) ฟังก์ชัน pie() เป็นการ plot กราฟแบบ pie chart

รูปแบบ :

pie (x,y) โดย x เป็นค่าตามแนวนอนและ y เป็นค่าตามแนว x

8) ฟังก์ชัน cbind() เป็นการนำคอลัมน์มารวมกัน

รูปแบบ :

cbind(ชื่อตัวแปร)

9) ฟังก์ชัน rnorm() เป็นการสร้างค่าแบบสุ่ม

รูปแบบ :

rnorm(มิติ) เราสามารถกำหนดขนาดของมิติได้ภายในวงเล็บ

10) ฟังก์ชัน data.entry() เป็นการกำหนดค่าแบบแสดงเป็นตารางโดยต้องมีการกำหนดค่าของตัวแปรนั้นอยู่ก่อนแล้วด้วย

รูปแบบ :

data.entry(ชื่อตัวแปร) จะปรากฏตารางเก็บค่าข้อมูลขึ้นมาให้กำหนดค่า

5.4 ฟังก์ชันที่ผู้ใช้สร้างขึ้นเอง (user-defined function)

เป็นฟังก์ชันที่ถูกสร้างขึ้นโดยผู้เขียนโปรแกรมซึ่งจะถูกสร้างขึ้นภายใต้วัตถุประสงค์ในการใช้งานของแต่ละฟังก์ชันแตกต่างกัน โดยฟังก์ชันที่เขียนขึ้นใช้เองนั้นสามารถเขียนเชื่อมต่อการใช้งานกับฟังก์ชันมาตรฐานของโปรแกรมได้ โดยปกติลักษณะการทำงานของฟังก์ชันที่ผู้ใช้สร้างขึ้นเองนั้น จะมีรูปแบบการทำงานได้หลายแบบ เพื่อให้เข้าใจจึงขออธิบายการทำงานของฟังก์ชัน โดยแบ่งตามรูปแบบการส่งผ่านค่าให้กับฟังก์ชัน และการส่งค่ากลับของฟังก์ชัน ซึ่งสามารถอธิบายรายละเอียดได้ดังนี้

1) รูปแบบการส่งผ่านค่าให้ฟังก์ชัน

1.1) ฟังก์ชันที่ไม่มีการส่งผ่านค่า เป็นฟังก์ชันที่ไม่มีการกำหนดอาร์กิวเมนต์ให้กับฟังก์ชัน ยกตัวอย่างเช่น

ตัวอย่าง 1: ฟังก์ชันที่ไม่มีการส่งผ่านค่า

```
myfun <- function () {
        cat("This is my first function.\n")
}
```

การเรียกใช้ฟังก์ชัน :

```
myfun()
```

ตัวอย่าง 2:ฟังก์ชันที่ไม่มีการส่งผ่านค่า

```
readinteger <- function(){
        n <- readline(prompt="Please, enter your ANSWER: ")
}
```

การเรียกใช้ฟังก์ชัน :

```
readinteger()
```

ตัวอย่าง 3:ฟังก์ชันที่ไม่มีการส่งผ่านค่า

```
Gamecoin = function() {
        count<-0
        coin = "tails" # initialize
        while(coin == "tails") {
                coin = sample(c("heads","tails"),1)
                count = count + 1
        }
        cat("There were",count,"tails before the first heads\n")
}
```

การเรียกใช้ฟังก์ชัน :

```
Gamecoin()
```

1.2) ฟังก์ชันที่มีการส่งผ่านค่า 1 ค่า

ตัวอย่าง 1:การส่งผ่านค่า 1 ค่า

```
Age <- function (x){
        If(x<10){
           Result <- "Childen"
        }else if(x<20){
           Result <- "Teen"
        }else{
           Result <- "Adult"
        }
}
```

การเรียกใช้ฟังก์ชัน :

```
Age(5)
```

ตัวอย่าง 2:การส่งผ่านค่า 1 ค่า

```
new.function <- function(a) {
        for(i in 1:a) {
                b <- i^2
                print(b)
        }
}
```

การเรียกใช้ฟังก์ชัน :

```
new.function(10)
```

ตัวอย่าง 3:การส่งผ่านค่า 1 ค่า

```
fact = function(x) {
        ret = 1
        for (i in 1:x) {
                ret = ret*i
        }
        return(ret)
}
```

การเรียกใช้ฟังก์ชัน:

```
fact(4)
```

1.3) ฟังก์ชันที่มีการส่งผ่านหลายค่า

ตัวอย่างการสร้างฟังก์ชันชื่อ multiplier ที่มีการรับค่าอาร์กิวเมนต์ 2 ค่าไว้ที่ตัวแปร x และ y โดยในฟังก์ชัน มีตัวแปร result ที่รับค่าการคำนวณจาก x คูณด้วย y แล้วแสดงผลลัพธ์ออกมา

ตัวอย่าง 1:การส่งผ่านค่าอาร์กิวเมนต์ 2 ค่า

```
multiplier <- function(x,y){
        result <- x*y
        print(paste(x,"x",y,"=",result))
}
```

การเรียกใช้ฟังก์ชัน:

```
multiplier(2,3)
```

ตัวอย่าง 2:การส่งผ่านค่าอาร์กิวเมนต์ 3 ค่า

```
new.function <- function(a,b,c) {
        result <- a*b+c
        print(result)
}
```

การเรียกใช้ฟังก์ชัน:

```
new.function(2,3,5)
```

ตัวอย่าง 3:การส่งผ่านค่าอาร์กิวเมนต์เป็นเวกเตอร์

```
ourhist = function(x) {
        hist(x,breaks="Scott", probability=TRUE)
        lines(density(x))
 }
```

การเรียกใช้งาน :

```
x<-c(2,3,4,5,6,7,8,9)
 ourhist(x)
```

ตัวอย่าง 4:การส่งผ่านค่าอาร์กิวเมนต์หลายค่า

```
ourhist2 <- function(x,breaks="Scott",col="pink") {
        hist(x,breaks=breaks, col=col,probability=TRUE)
        lines(density(x))
}
```

การเรียกใช้งาน :

```
x<-c(2,3,4,5,6,7,8,9)
ourhist2(x)
```

1.4) ฟังก์ชันที่มีการส่งผ่านค่าแบบมีการระบุค่าเริ่มต้นให้กับอาร์กิวเมนต์

ตัวอย่าง 5:การระบุค่าเริ่มต้นให้กับอาร์กิวเมนต์

```
mul.function <- function(a=3,b=6) {
        result <- a*b
        print(result)
}
```

การเรียกใช้งาน :

```
mul.function()
```

1.5) ฟังก์ชันที่มีการส่งผ่านโดยไม่รู้จำนวนด้วย ellipsis เป็นการรับค่าข้อมูลเข้ามาในฟังก์ชันโดยไม่ทราบจำนวนที่แน่นอนของข้อมูล ซึ่งมีตัวอย่างการใช้งานดังนี้

ตัวอย่าง 1: การรับค่าข้อมูลเข้ามาในฟังก์ชันโดยไม่ทราบจำนวนที่แน่นอนของข้อมูล

```
addNum <- function(...) {
        b <- list(...)
        for(e in b){
                print(e)
        }
}
```

การเรียกใช้งาน:

```
addNum(2,3,4)
```

2) รูปแบบการส่งค่ากลับของฟังก์ชัน

2.1) ฟังก์ชันที่ส่งค่ากลับเพียง 1 ค่า

ตัวอย่าง 1:การส่งค่ากลับเพียง 1 ค่า

```
Age <- function (x){
        If(x<10){
           Result <- "Childen"
        }else if(x<20){
           Result <- "Teen"
        }else{
           Result <- "Adult"
        }
        Return(Result)
}
```

การเรียกใช้งาน:

```
Age(20)
```

2.2) ฟังก์ชันที่ส่งค่ากลับหลายค่า

ตัวอย่าง 2:การส่งค่ากลับเป็นลิสต์

```
f = function(x) list(len=length(x),total=sum(x),mean=mean(x))
```

การเรียกใช้งาน:

```
f(x)
```

ตัวอย่าง 3:การส่งค่ากลับเป็นลิสต์

```
ReturnTwoValues <- function() {
        return(list(1, matrix(0, 2, 2)))
}
```

การเรียกใช้งาน:

```
c = ReturnTwoValues()
```

ตัวอย่าง 4:การส่งค่ากลับเป็นลิสต์

```
func2<-function(input) {
        a<-input+1
        b<-input+2
        output<-list(a,b)
        return(output)
}
```

การเรียกใช้งาน:

```
output<-func2(5)
for (i in output) {
        print(i)
}
```

ตัวอย่าง 5:การส่งค่ากลับเป็นลิสต์

```
cylinder= function(height, radius){
        if (missing(height))
            stop("Need to specify height of cylinder for calculations.")
        if (missing(radius))
            stop("Need to specify radius of cylinder for calculations.")
        if (height < 0)
            stop("Negative height specified.")
        if (radius < 0)
            stop("Negative radius specified.")
        volume = pi * radius * radius * height
        list(Height = height, Radius = radius, Volume = volume)
}
```

การเรียกใช้งาน:

```
cylinder (20, 4)
```

ตัวอย่าง 6: การส่งค่ากลับเป็น Object

```
setClass(Class="Person",
        representation(
          height="numeric",
          age="numeric"
        )
)
myFunction = function(age=28, height=176){
  return(new("Person",
             age=age,
             height=height))
}
```

การเรียกใช้งาน:

```
x= myFunction()
Print(x)
```

ตัวอย่าง 7: การส่งค่ากลับเป็นลิสต์

```
x1 = function(x){
        mu = mean(x)
        l1 = list(s1=table(x),std=sd(x))
        return(list(l1,mu))
}
```

การเรียกใช้งาน:

```
x=c(2,3,4,5,6)
c=x1(x)
```