แผนบริหารการสอนประจำบทที่ 4

ฟังก์ชัน

หัวข้อประจำบท

1. ความหมายของฟังก์ชัน
2. การสร้างฟังก์ชัน
3. การสร้างโมดูล
4. คำสั่งเรียกฟังก์ชันและโมดูล
5. พารามิเตอร์และอาร์กิวเมนต์
6. การจัดวางตำแหน่งฟังก์ชันในโปรแกรม
7. การเรียกใช้โมดูลที่มีในโปรแกรม

วัตถุประสงค์เชิงพฤติกรรม

1. เพื่อให้ผู้เรียนสามารถอธิบายเกี่ยวกับฟังก์ชัน
2. เพื่อให้ผู้เรียนสามารถสร้างฟังก์ชันใช้ในโปรแกรมได้
3. เพื่อให้ผู้เรียนสามารถอธิบายเกี่ยวกับโมดูล และสามารถสร้างโมดูลได้
4. เพื่อให้ผู้เรียนสามารถอธิบายเกี่ยวกับการใช้คำสั่งเรียกฟังก์ชันและโมดูล
5. เพื่อให้ผู้เรียนสามารถอธิบายเกี่ยวกับพารามิเตอร์และอาร์กิวเมนต์
6. เพื่อให้ผู้เรียนสามารถอธิบายเกี่ยวกับการจัดวางตำแหน่งฟังก์ชันในโปรแกรม
7. เพื่อให้ผู้เรียนสามารถอธิบายเกี่ยวกับการเรียกใช้โมดูลที่มีในโปรแกรม
8. เพื่อให้ผู้เรียนสามารถนำ ฟังก์ชัน โมดูล และพารามิเตอร์และอาร์กิวเมนต์ ประยุกต์ใช้งานในการแก้โจทย์ปัญหาได้อย่างถูกต้อง

วิธีการสอนและกิจกรรม

1. ผู้สอนบรรยายในชั้นเรียน และโปรแกรมนำเสนอ ตามหัวข้อเนื้อหาในเอกสารประกอบการสอน
2. ผู้เรียนทำแบบฝึกหัดท้ายบท และฝึกเขียนโปรแกรมด้วยคอมพิวเตอร์

สื่อการเรียนการสอน

1. เอกสารประกอบการสอน วิชา หลักการเขียนโปรแกรมคอมพิวเตอร์
2. โปรแกรม Python รุ่น 3.8.10
3. โปรแกรม PowerPoint

การวัดผลและการประเมินผล

1. สังเกตจากการอภิปราย การตอบคำถาม และซักถามระหว่างเรียน
2. การปฏิบัติบนเครื่องคอมพิวเตอร์
3. การทำแบบฝึกหัดท้ายบท

บทที่ 4
ฟังก์ชัน

การทำงานของโปรแกรมระดับสูง ขนาดของโปรแกรมมีความซับซ้อนและมีขนาดใหญ่ โปรแกรมจะมีการเก็บรวบรวมกลุ่มของคำสั่งไว้ใช้งาน เพื่อให้สามารถเรียกใช้กลุ่มคำสั่งนี้ได้ทันที โดยไม่ต้องสร้างชุดคำสั่งขึ้นมาใหม่แทน ลักษณะกลุ่มโปรแกรมแบบนี้ เรียกว่า “ฟังก์ชัน” ภาษาไพธอนได้เตรียมกลุ่มของฟังก์ชันใช้งานตามลักษณะของงานที่จะนำไปใช้ได้ เช่น ฟังก์ชันรับค่าและแสดงค่า ฟังก์ชันคำนวณทางคณิตศาสตร์ ฟังก์ชันจัดการกับแฟ้มข้อมูล ฟังก์ชันสร้างส่วนต่อประสานกราฟิกกับผู้ใช้ เป็นต้น ซึ่งนอกเหนือจากฟังก์ชันที่มีให้เลือกใช้ในโปรแกรมในภาษาไพธอน ผู้เขียนโปรแกรมสามารถสร้างขึ้นมาใช้งานเองตามความต้องการได้ แนวคิดการเขียนโปรแกรมแบบฟังก์ชันเป็นกระบวนการที่มองภาพการแก้ปัญหาคล้ายกับมนุษย์ โดยมองปัญหาออกเป็นส่วน วิธีการแก้ปัญหาจะแก้ปัญหาทีละส่วนจนได้ผลลัพธ์ตามที่ต้องการ (สุชาติ คุ้มมณี, 2558) ลักษณะการใช้งานฟังก์ชันมีรายละเอียดดังต่อไปนี้

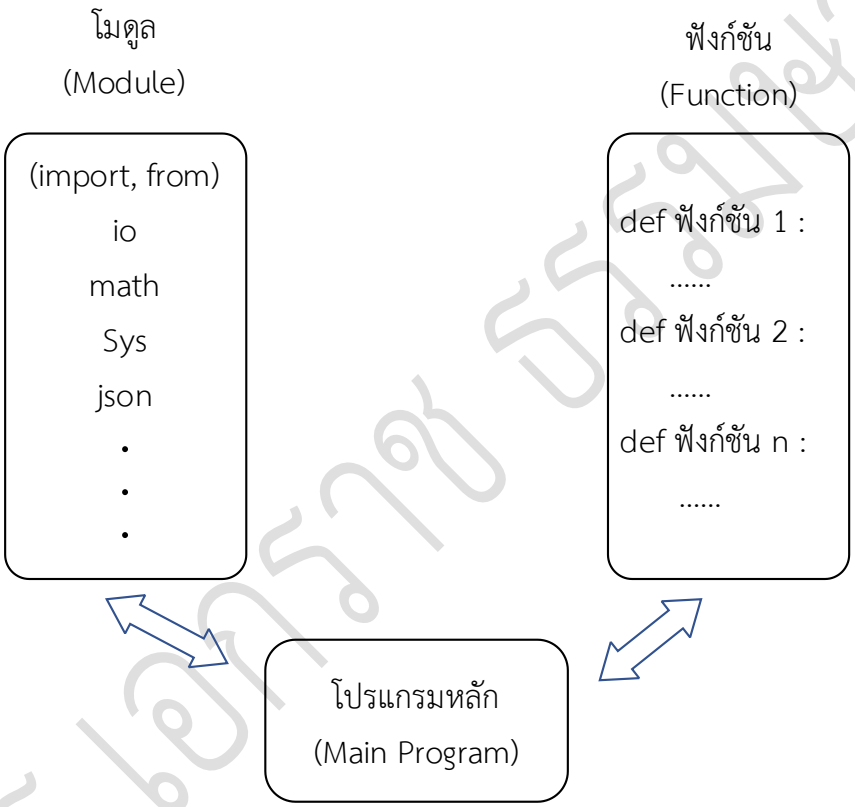
ความหมายของฟังก์ชัน

ฟังก์ชัน (Function) หมายถึง เครื่องมือที่ประกอบไปด้วยกลุ่มของประโยคคำสั่งที่สามารถทำงานอย่างใดอย่างหนึ่งและถูกเรียกใช้งานได้หลายครั้งในโปรแกรม (Lutz, 2013) ส่วนมากเป็นสิ่งที่ต้องทำซ้ำบ่อย ๆ ภายในโปรแกรม โดยจะแยกสิ่งที่ต้องทำซ้ำ ออกมาสร้างเป็นชุดคำสั่งย่อยไว้อีกส่วนหนึ่ง เพื่อให้สามารถเรียกใช้งานเฉพาะส่วนนี้ได้ทันที่เมื่อต้องการ (บัญชา ปะสีละเตสัง, 2558) เมื่อมีการเขียนโปรแกรมที่มีลักษณะการทำงานที่เหมือนกันหรือซ้ำกันหลายครั้ง วิธีการที่ดี คือ การกำหนดในลักษณะของฟังก์ชัน นั่นคือ การเก็บชุดคำสั่งที่ซ้ำไว้ชุดเดียว สามารถอ่านชุดคำสั่งนี้ได้หลายครั้ง โดยเรียกใช้ได้ตามจำนวนครั้งที่ต้องการ การเขียนโปรแกรมนั้นต้องหลีกเลี่ยงการเขียนชุดคำสั่งที่ซ้ำกัน ซึ่งจะมีแนวโน้มที่จะทำให้เกิดความผิดพลาด บ่อยครั้งโปรแกรมผิดพลาดซึ่งเกิดจากการเขียนชุดคำสั่งที่ซ้ำกัน เช่น เกิดจากการคัดลอกโปรแกรม การแก้ไขโปรแกรมที่มีหลายตำแหน่ง เป็นต้น ดังนั้นควรกำหนดชุดคำสั่งที่เหมือนกัน ไว้ในที่เดียวกัน เพื่อประโยชน์ในการตรวจสอบและทำการแก้ไข (Lee, 2011)

ฟังก์ชันและโปรแกรมย่อย หรือเรียกอีกอย่างหนึ่งว่า โพรซีเยอร์ (Procedure) มีลักษณะการทำงานที่เหมือนกัน คือ เป็นที่อยู่ของกลุ่มคำสั่งที่มีโอกาสเรียกใช้งานซ้ำบ่อยครั้ง สามารถกำหนดตัวแปรที่ใช้งานเฉพาะภายในเท่านั้น รองรับการผ่านค่าข้อมูลจากภายนอกได้ และสามารถดึงตัวแปร

ภายนอกมาใช้งานร่วมภายในได้ แต่จะแตกต่างกันตรงที่ ฟังก์ชันสามารถส่งค่าผลลัพธ์กลับในรูปแบบการคืนค่าได้ (โชติพันธุ์ หล่อเลิศสุนทร และ ฐิตะพันธุ์ หล่อเลิศสุนทร, 2559)

ดังนั้น การแยกส่วนในการเก็บคำสั่งที่มีลักษณะเหมือนกันไว้ด้วยกัน ซึ่งแต่ละฟังก์ชันจะมีหน้าที่ในการทำงานแตกต่างกัน เพื่อความสะดวกในการเรียกใช้งานและสามารถเรียกใช้ได้หลายครั้งไม่จำกัด หลังจากการประมวลคำสั่งสามารถคืนค่ากลับมายังส่วนที่มีการเรียกใช้ ซึ่งเป็นข้อมูลสำคัญในการทำงานของโปรแกรม ซึ่งมีโครงสร้างการทำงานดังต่อไปนี้



ภาพที่ 4.1 โครงสร้างการทำงานของฟังก์ชัน

ฟังก์ชันการใช้งานประกอบด้วย 2 ส่วน คือ ส่วนที่มีอยู่ในโปรแกรม เรียกว่า โมดูล (Module) จะเก็บกลุ่มของคำสั่งและฟังก์ชันพื้นฐานที่ภาษาไพธอนได้เตรียมไว้ให้ใช้งาน (Fangohr, 2015) เพื่ออำนวยความสะดวกต่อการใช้งาน ฟังก์ชันใดตรงกับความต้องการ สามารถเรียกใช้ได้ทันที โดยไม่ต้องสร้างขึ้นมาใหม่ การเรียกใช้งานประกอบด้วยคำสั่ง import และคำสั่ง from และส่วนที่เป็นฟังก์ชันที่สร้างขึ้นมาใหม่ใช้งานเฉพาะของโปรแกรม โดยทั้งสองส่วนนี้ จะถูกเรียกใช้งานจากโปรแกรมหลัก (Main Program) ของโปรแกรม

การสร้างฟังก์ชัน

การสร้างฟังก์ชันใช้งานในภาษาโปรแกรม จะใช้คำสั่ง def สร้างฟังก์ชัน โครงสร้างของการใช้คำสั่งมีรายละเอียดดังนี้

```
def  ชื่อฟังก์ชัน :
        ชุดคำสั่ง
```

การสร้างฟังก์ชันในโปรแกรม จะมีลักษณะการกำหนดฟังก์ชัน โปรแกรมหลักและชุดคำสั่งทั้งหมดของโปรแกรมไว้ด้วยกัน การเรียกใช้ฟังก์ชัน จะอ้างอิงผ่านชื่อของฟังก์ชันของโปรแกรม (Halterman, 2016) ดังโปรแกรมตัวอย่างที่ 4.1

ตัวอย่างที่ 4.1 การสร้างฟังก์ชันในโปรแกรม

บรรทัดที่	คำสั่ง
1	def point_total(point,final,total):
2	t=point+final+total
3	g=show_grade(t)
4	return g
5	def show_grade(p):
6	if p >= 80:
7	grade='A'
8	elif p >= 70:
9	grade='B'
10	elif p >= 60:
11	grade='C'
12	elif p >= 50:
13	grade='D'
14	else:
15	grade='E'
16	return grade

17	print("เกรดที่ได้ = %s"% point_total(20,10,20))
ผลลัพธ์	
เกรดที่ได้ = D	

โปรแกรมตัวอย่างที่ 4.1 การสร้างฟังก์ชันในโปรแกรม เริ่มต้นการทำงานจากการเรียกใช้และส่งค่าไปยังฟังก์ชัน point_total(20,10,20) โดยมีฟังก์ชันประกอบด้วย point_total() ทำหน้าที่ในการรวมคะแนนจากตัวเลข 3 ตัวเลข มีการส่งค่าข้อมูลที่ได้ไปยังฟังก์ชัน show_grade() ทำหน้าที่ในการเปรียบเทียบคะแนนเพื่อหาเกรดที่ได้และคืนค่าผลลัพธ์ที่ได้กลับไปยังค่า g ในฟังก์ชัน point_total() และคืนค่ากลับมายังคำสั่งบรรทัดที่ 17 ซึ่งจะแสดงผลลัพธ์ที่ได้จากตัวอย่างที่ 4.1 การใช้งานฟังก์ชันในลักษณะนี้ เหมาะสมกับโปรแกรมที่มีความซับซ้อนน้อยและขนาดไม่ใหญ่มาก ซึ่งเป็นการใช้งานฟังก์ชันพื้นฐานของโปรแกรม

การสร้างโมดูล

ฟังก์ชันที่มีใช้งานในโปรแกรม หากปริมาณมากเกินไป จะมีความยุ่งยากต่อการปรับปรุงแก้ไขฟังก์ชัน ดังนั้นเพื่อความเป็นระเบียบเรียบร้อย สะดวกต่อการตรวจสอบและแก้ไข จึงแยกกลุ่มของฟังก์ชันที่มีลักษณะของการทำงานเกี่ยวข้องกัน เก็บในรูปแบบของไฟล์ฟังก์ชันหรือโมดูล การอ้างอิงหรือเรียกฟังก์ชันขึ้นมาใช้งาน ดังการสร้างโมดูลโปรแกรมตัวอย่างที่ 4.2

ตัวอย่างที่ 4.2 การสร้างโมดูล

บรรทัดที่	คำสั่ง
1	def cal_bmi(tall,weight):
2	bmi = weight/(tall**2)
3	result = chk_level(bmi)
4	return result
5	def chk_level(g_bmi):
6	if g_bmi>=40:
7	r ='โรคอ้วนระดับรุ่นแรงมาก'
8	elif g_bmi>=35:
9	r ='โรคอ้วนระดับรุ่นแรง'
10	elif g_bmi>=30:

```
11          r ='โรคอ้วนระดับกลาง'
12      elif g_bmi>=25:
13          r ='ภาวะน้ำหนักเกิน'
14      elif g_bmi>=18.5:
15          r ='ปกติ'
16      elif g_bmi>=16:
17          r ='น้ำหนักน้อย'
18      elif g_bmi>=15:
19          r ='น้ำหนักน้อยระดับรุนแรง'
20      else:
21          r ='น้ำหนักน้อยระดับรุนแรงมาก'
22      return r
23  def show_line():
24      d = '-'*20
25      return d
```

โปรแกรมตัวอย่างที่ 4.2 โปรแกรมทำหน้าที่เกี่ยวกับดัชนีมวลกาย ซึ่งประกอบไปด้วยฟังก์ชัน cal_bmi() ทำหน้าที่ในการคำนวณค่า bmi ฟังก์ชัน chk_level() ทำหน้าหน้าที่แปลผลและฟังก์ชัน show_line() ให้แสดงเส้นกั้น บันทึกโปรแกรมที่ได้ชื่อ bmi_result.py โครงสร้างของโปรแกรมดังกล่าว ประกอบด้วยเฉพาะฟังก์ชันการทำงานเท่านั้น ภายในโปรแกรมไม่มีการทำงานของโปรแกรมหลัก จุดประสงค์เพื่อเก็บหน้าที่การทำงานสำหรับการเรียกใช้จากโปรแกรมอื่น การทำงานดังกล่าวคือลักษณะของโมดูล

คำสั่งเรียกใช้ฟังก์ชันและโมดูล

การเขียนโปรแกรมให้มีการนำคำสั่งที่อยู่ภายนอกของโปรแกรม คำสั่งที่ใช้เรียกฟังก์ชันและโมดูล คือคำสั่ง import และ from ซึ่งการใช้คำสั่งขึ้นอยู่กับลักษณะของการเรียกใช้ ซึ่งมีรูปแบบที่แตกต่างกัน ลักษณะการใช้งานแต่ละคำสั่งพื้นฐาน มีรายละเอียดดังต่อไปนี้

1. คำสั่ง import

การเรียกใช้คำสั่งหรือฟังก์ชันในโมดูลทั้งหมด สามารถใช้คำสั่ง import ซึ่งเป็นคำสั่งพื้นฐาน โดยมีรูปแบบการใช้คำสั่งดังต่อไปนี้

```
import  โมดูล 1, โมดูล 2, ... , โมดูล n
```

การเรียกใช้โมดูล bmi_result.py โดยใช้คำสั่ง import มีรูปแบบการใช้คำสั่ง ดังโปรแกรมตัวอย่างที่ 4.3

ตัวอย่างที่ 4.3 การใช้คำสั่ง import เรียกใช้โมดูล

บรรทัดที่	คำสั่ง
1	import bmi_result
2	get_tall = float(input("ความสูง (เมตร) = "))
3	get_weight=float(input("น้ำหนัก (กิโลกรัม) = "))
4	print("สภาวะ = %s" % bmi_result.cal_bmi(get_tall,get_weight))
5	print("%s" % bmi_result.show_line())
ผลลัพธ์	
ความสูง (เมตร) = 1.65 น้ำหนัก (กิโลกรัม) = 74.8 สภาวะ = ภาวะน้ำหนักเกิน ----------	

การเรียกใช้ฟังก์ชันในโมดูลในโปรแกรมตัวอย่างที่ 4.3 รูปแบบของการเรียกจะต้องอ้างอิงจากโมดูลและฟังก์ชันที่ต้องการใช้ โปรแกรมบรรทัดที่ 4 ใช้คำสั่งเรียกใช้โมดูล คือ bmi_result และฟังก์ชัน cal_bmi และบรรทัดที่ 5 เรียกโมดูล bmi_result และเรียกใช้ฟังก์ชัน show_line โปรแกรมดังกล่าวเป็นรูปแบบของการใช้คำสั่ง import

2. คำสั่ง from

การใช้งานโมดูล เมื่อต้องการระบุฟังก์ชันเพื่อความรวดเร็วในการเข้าถึงฟังก์ชัน จะให้คำสั่ง from ร่วมกับคำสั่ง import ซึ่งมีรูปแบบดังต่อไปนี้

```
from  โมดูล  import  ฟังก์ชัน 1, ฟังก์ชัน 2, ... , ฟังก์ชัน n หรือ * (ฟังก์ชันทั้งหมด)
```

คำสั่ง from เมื่อนำไปใช้งานในโปรแกรม รูปแบบและลักษณะการทำงานของคำสั่ง มีรายละเอียดดังโปรแกรมตัวอย่างที่ 4.4

ตัวอย่างที่ 4.4 การใช้คำสั่ง from

บรรทัดที่	คำสั่ง
1	from bmi_result import cal_bmi,show_line
2	get_tall = float(input("ความสูง (เมตร) = "))
3	get_weight=float(input("น้ำหนัก (กิโลกรัม) = "))
4	print("สภาวะ = %s" % cal_bmi(get_tall,get_weight))
5	print("%s" % show_line())

ผลลัพธ์
ความสูง (เมตร) = 1.75 น้ำหนัก (กิโลกรัม) = 74.2 สภาวะ = ปกติ --------------------

การเรียกใช้ฟังก์ชันในโมดูลจากคำสั่ง import จะต้องระบุฟังก์ชันให้ครบตามที่ต้องการนำไปใช้ ถ้าหากเรียกใช้ฟังก์ชันไม่ครบตามจำนวน เมื่อโปรแกรมทำการตรวจสอบฟังก์ชันที่เรียกใช้ทำให้เกิดการผิดพลาดของโปรแกรม จากเงื่อนไขการเรียกใช้ฟังก์ชัน ผลลัพธ์ที่ได้ดังรายละเอียดการทำงานของโปรแกรมตัวอย่างที่ 4.5

ตัวอย่างที่ 4.5 การทำงานผิดพลาดของโปรแกรมจากการเรียกฟังก์ชันไม่ครบผ่านคำสั่ง import

บรรทัดที่	คำสั่ง
1	from bmi_result import cal_bmi
2	print("--------- โปรแกรมคำนวณดัชนีมวลกาย---------")
3	get_tall = float(input("ความสูง (เมตร) = "))
4	get_weight=float(input("น้ำหนัก (กิโลกรัม) = "))
5	print("สภาวะ = %s" % cal_bmi(get_tall,get_weight))
6	print("%s" % show_line())

ผลลัพธ์
--------- โปรแกรมคำนวณดัชนีมวลกาย---------

```
ความสูง (เมตร) = 1.75
น้ำหนัก (กิโลกรัม) = 78.6
สภาวะ = ภาวะน้ำหนักเกิน
Traceback (most recent call last):
  File "C:\Python36\tt.py", line 6, in <module>
    print("%s" % show_line())
NameError: name 'show_line' is not defined
```

โปรแกรมตัวอย่างที่ 4.5 การทำงานผิดพลาดของโปรแกรมจากการเรียกฟังก์ชันไม่ครบผ่านคำสั่ง import ซึ่งเกิดความผิดพลาดของโปรแกรมในบรรทัดที่ 6 โดยการ import ไม่ได้อ้างอิงฟังก์ชัน show_line() ที่อยู่ในโมดูล bmi_result ดังนั้นเมื่อทำการเรียกใช้ฟังก์ชัน โปรแกรมจะตรวจสอบฟังก์ชันที่ถูกเรียกอ้างอิงในการเรียกใช้ เมื่อไม่พบการอ้างอิงจากคำสั่ง import โปรแกรมจะแสดงผลความผิดพลาดขึ้น

พารามิเตอร์และอาร์กิวเมนต์

การทำงานของฟังก์ชัน โครงสร้างการทำงานจะสามารถทำงานได้ จะต้องมีข้อมูลที่จะรับเข้าไปทำการประมวลผลตามหน้าที่ของฟังก์ชัน ซึ่งข้อมูลส่วนมากจะถูกส่งจากโปรแกรมหลักเข้าไปในฟังก์ชัน การรับส่งข้อมูลระหว่างฟังก์ชันกับส่วนต่าง ๆ จะอยู่ในรูปของพารามิเตอร์และอาร์กิวเมนต์ ซึ่งมีลักษณะการทำงานดังต่อไปนี้

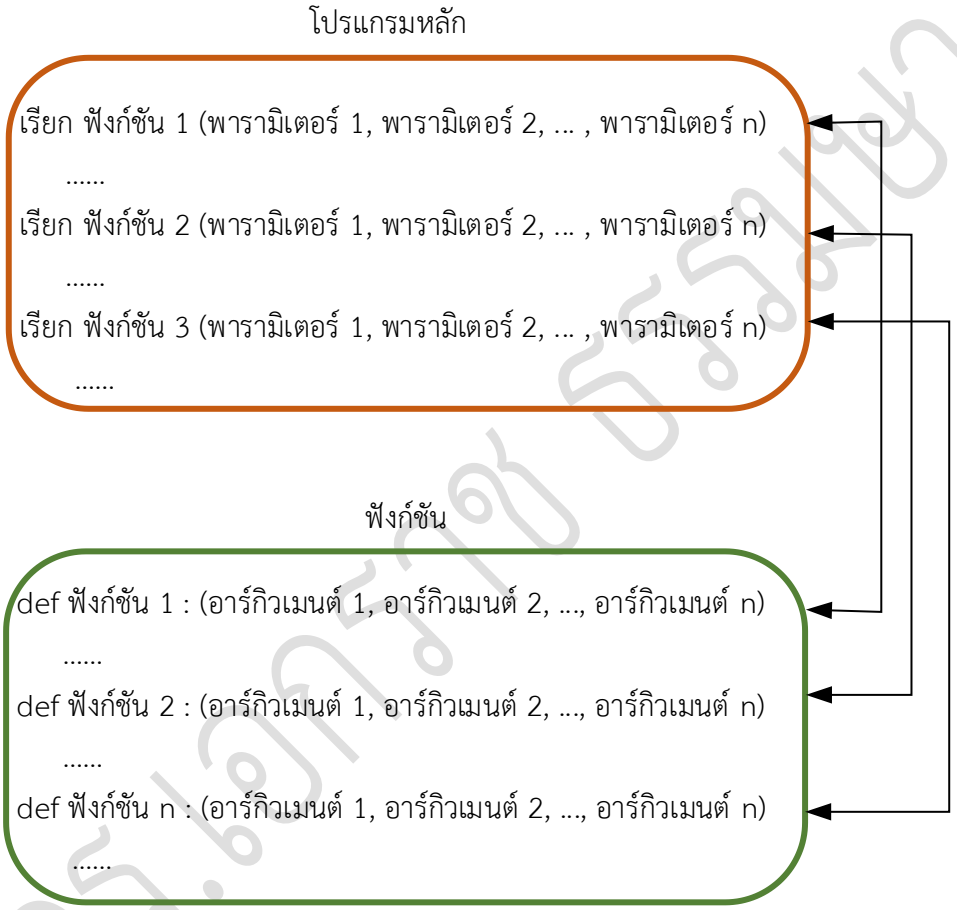
1. ลักษณะของพารามิเตอร์และอาร์กิวเมนต์

พารามิเตอร์ (Parameter) หมายถึง กลุ่มของการทำงานที่ภายในประกอบด้วยข้อมูลที่มีลักษณะแตกต่างกัน ซึ่งแต่ละข้อมูลจะมีลักษณะเฉพาะต่อการนำไปใช้ (Halterman, 2016) ข้อมูลที่รับเข้ามาจากภายนอก เพื่อใช้งานในโปรแกรมย่อย ซึ่งพารามิเตอร์นี้จะช่วยให้โปรแกรมย่อยสามารถเปลี่ยนแปลงการทำงานไปตามข้อมูลที่รับเข้ามาและให้ผลลัพธ์ที่แตกต่างกันออกไป ดังนั้นพารามิเตอร์จึงช่วยให้โปรแกรมย่อยมีความยืดหยุ่นในการทำงานมากขึ้น (บัญชา ปะสีละเตสัง, 2558) การเรียกใช้โปรแกรมย่อย บางครั้งโปรแกรมหลักจะต้องส่งข้อมูลให้โปรแกรมย่อยใช้ข้อมูลที่ส่งนั้นเรียกว่าพารามิเตอร์ (ธีรวัฒน์ ประกอบผล, 2558)

อาร์กิวเมนต์ (Argument) หมายถึง การส่งค่าข้อมูลผ่านเข้าไปภายในฟังก์ชัน เพื่อจะได้นำค่าข้อมูลเหล่านั้นไปใช้งานกับตัวแปร คำนวณหรือใช้กับกลุ่มคำสั่งย่อยภายในฟังก์ชัน (โชติพันธุ์ หล่อเลิศสุนทร และ ฐิตะพันธุ์ หล่อเลิศสุนทร, 2559)

การส่งผ่านค่าพารามิเตอร์ (Parameter-passing methods) หมายถึง พารามิเตอร์ที่ถูกเคลื่อนย้ายไปยังโปรแกรมย่อยที่เรียกใช้ โดยลักษณะของพารามิเตอร์ขึ้นอยู่กับการนำไปใช้งาน (Sebesta, 2016)

จากผู้ให้ความหมายข้างต้น สามารถอธิบายแผนภาพของพารามิเตอร์และอาร์กิวเมนต์มีรายละเอียดดังต่อไปนี้



ภาพที่ 4.2 การทำงานของพารามิเตอร์และอาร์กิวเมนต์

นิยามความหมายข้างต้นสามารถสรุปได้ว่า พารามิเตอร์ หมายถึง เซตของข้อมูลประกอบด้วยข้อมูลที่มีรูปแบบเหมือนกันหรือต่างกันขึ้นอยู่กับการนำไปใช้งาน โดยค่าของพารามิเตอร์จะถูกส่งจากโปรแกรมหลักไปยังฟังก์ชัน ซึ่งในฟังก์ชันจะมีตัวแปรที่ทำหน้าที่ในการรับค่าของพารามิเตอร์เข้ามาเรียกว่า อาร์กิวเมนต์ ซึ่งจะนำค่าที่ได้ไปทำงานข้างในของฟังก์ชันและสามารถคืนค่ากลับมายังโปรแกรมหลัก ดังภาพที่ 4.2

2. การส่งผ่านค่าพารามิเตอร์และอาร์กิวเมนต์

การทำงานของโปรแกรมที่มีความสัมพันธ์ระหว่างพารามิเตอร์และอาร์กิวเมนต์ จะมีการส่งผ่านค่าพารามิเตอร์จากโปรแกรมหลักไปยังฟังก์ชันที่มีตัวอาร์กิวเมนต์ มีรายละเอียดการทำงานพื้นฐานโปรแกรมตัวอย่างที่ 4.6

ตัวอย่างที่ 4.6 การส่งผ่านค่าพารามิเตอร์และอาร์กิวเมนต์

บรรทัดที่	คำสั่ง
1	def chk_occupation(a):
2	if a=='pro':
3	occu='นักเขียนโปรแกรม'
4	else:
5	occu='นักวิเคราะห์ระบบ'
6	return occu
7	def show_name(b):
8	return b
9	def chk_total(c):
10	salary = 32000
11	money = salary+c
12	return money;
13	print('ชื่อ = ',show_name('เอกราช ธรรมษา'))
14	print('อาชีพ = ',chk_occupation('pro'))
15	print('เงินรับ = ',chk_total(12000))
ผลลัพธ์	
ชื่อ = เอกราช ธรรมษา อาชีพ = นักเขียนโปรแกรม เงินรับ = 44000	

โปรแกรมตัวอย่างที่ 4.6 ประกอบไปด้วยโปรแกรมหลักบรรทัดที่ 13-15 มีการเรียกใช้ฟังก์ชันพร้อมทั้งส่งค่าพารามิเตอร์ ซึ่งแต่ละฟังก์ชันจะมีอาร์กิวเมนต์ที่ทำหน้าที่ในการรับข้อมูลประกอบไปด้วย a b และ c ซึ่งจะรับข้อมูลต่อเข้าไปใช้ประมวลการทำงานในฟังก์ชันและคืนค่ากลับมายังโปรแกรมหลัก

การใช้งานฟังก์ชันพื้นฐาน จะเป็นลักษณะการทำงานเฉพาะ เมื่อเสร็จสิ้นการประมวลผลหรือการทำงาน จะมีการคืนค่าเพียงค่าเดียวกลับคืนมาสู่โปรแกรมหลัก การทำงานลักษณะแบบนี้จะต้องสร้างฟังก์ชันให้ครอบคลุมต่อการใช้งาน เหมาะสมกับการทำงานที่มีปริมาณของฟังก์ชันไม่มาก แต่จะไม่เหมาะสมต่อการใช้งานที่มีความซับซ้อนและฟังก์ชันปริมาณมาก ซึ่งไม่สะดวกต่อการเขียนโปรแกรม

3. ฟังก์ชันคืนค่าพารามิเตอร์แบบหลายค่า

การรวบรวมฟังก์ชันที่มีการทำงานเกี่ยวข้องกันรวมไว้ด้วยกัน เพื่อให้มีปริมาณของฟังก์ชันลดลง แต่ครอบคลุมการทำงานและเมื่อประมวลเสร็จสิ้น การคืนค่ากลับมายังโปรแกรมสามารถคืนค่าได้มากกว่าหนึ่งค่า มีรายละเอียดการทำงานดังโปรแกรมตัวอย่างที่ 4.7

ตัวอย่างที่ 4.7 ความสัมพันธ์ของพารามิเตอร์และอาร์กิวเมนต์

บรรทัดที่	คำสั่ง
1	def fun(a,b,c):
2	if b=='pro':
3	occu='นักเขียนโปรแกรม'
4	else:
5	occu='นักวิเคราะห์ระบบ'
6	money = 32000+c
7	return a,occu, money;
8	name,occupation, get_money = fun('เอกราช ธรรมษา','pro',5000)
9	print('ชื่อ = ',name)
10	print('อาชีพ = ',occupation)
11	print('เงินรับต่อเดือน = ',get_money)
ผลลัพธ์	
ชื่อ = เอกราช ธรรมษา อาชีพ = นักเขียนโปรแกรม เงินรับต่อเดือน = 37000	

โปรแกรมตัวอย่างที่ 4.7 การสร้างฟังก์ชัน โดยการรวมฟังก์ชันอื่นที่เกี่ยวข้องรวมด้วยกันเป็นเพียงหนึ่งฟังก์ชัน การเรียกใช้ฟังก์ชันจะส่งค่าพารามิเตอร์ไปพร้อมกันทั้งสามค่า ส่วนของฟังก์ชัน

จะมีอาร์กิวเมนต์ที่ทำหน้าที่รับค่าทั้งสามค่าเช่นกัน จากนั้นเข้าสู่การประมวลผลภายในฟังก์ชันและมีการคืนค่ากลับมายังโปรแกรมหลัก และนำค่าที่ได้ไปแสดงผลข้อมูล

การส่งผ่านระหว่างฟังก์ชันนั้นจะต้องมีการกำหนดจำนวนพารามิเตอร์และอาร์กิวเมนต์ที่มีจำนวนเท่ากันถ้ามีการกำหนดไม่เท่ากันจะเกิดการผิดพลาดการทำงานของโปรแกรม ดังโปรแกรมตัวอย่างที่ 4.8 และตัวอย่างที่ 4.9

ตัวอย่างที่ 4.8 ข้อผิดพลาดจากการกำหนดอาร์กิวเมนต์น้อยกว่าพารามิเตอร์ที่รับเข้า

บรรทัดที่	คำสั่ง
1	def func(a,b):
2	return a,b
3	name,occupation, salary = func('เอกราช','นักเขียนโปรแกรม',35000)
4	print(name,occupation,salary)
ผลลัพธ์	
Traceback (most recent call last): File "D:\python_file\test.py", line 3, in <module> name,occupation, salary = func('เอกราช','นักเขียนโปรแกรม',35000) TypeError: func() takes 2 positional arguments but 3 were given	

โปรแกรมตัวอย่างที่ 4.8 เกิดข้อผิดพลาดจากการกำหนดอาร์กิวเมนต์น้อยกว่าพารามิเตอร์ที่รับเข้า เนื่องจากฟังก์ชัน fun() มี 2 อาร์กิวเมนต์ แต่มีการส่งค่าพารามิเตอร์เข้ามา 3 จำนวน ดังนั้นค่าพารามิเตอร์สุดท้ายไม่มีอาร์กิวเมนต์ที่จะรับค่า ซึ่งจะทำให้เกิดข้อผิดพลาดของโปรแกรมขึ้น

ตัวอย่างที่ 4.9 ข้อผิดพลาดจากการกำหนดอาร์กิวเมนต์มากกว่าพารามิเตอร์ที่รับเข้า

บรรทัดที่	คำสั่ง
1	def func(a,b,c):
2	return a,b
3	name,salary = func('เอกราช',35000)
4	print(name,salary)
ผลลัพธ์	
Traceback (most recent call last): File "D:\python_file\test.py", line 3, in <module>	

```
    name,salary = func('เอกราช',35000)
TypeError: func() missing 1 required positional argument: 'c'
```

โปรแกรมตัวอย่างที่ 4.9 เกิดข้อผิดพลาดจากการกำหนดอาร์กิวเมนต์มากกว่าพารามิเตอร์ที่รับเข้า เนื่องจากฟังก์ชัน fun() มี 3 อาร์กิวเมนต์ แต่มีการส่งค่าพารามิเตอร์เข้ามา 2 จำนวน ดังนั้นอาร์กิวเมนต์สุดท้ายจะไม่มีค่าข้อมูล ซึ่งจะทำให้เกิดข้อผิดพลาดของโปรแกรมขึ้น การแก้ปัญหาที่เกิดขึ้น สามารถกำหนดอาร์กิวเมนต์ให้มีค่าเริ่มต้น (Argument Default Value) (Halterman, 2016) หากเป็นการเก็บข้อมูลแบบอักขระหรือข้อความให้ใช้สัญลักษณ์ ‘ ’ และตัวเลขให้ค่าข้อมูลเริ่มต้นที่ 0 ดังโปรแกรมตัวอย่างที่ 4.10

ตัวอย่างที่ 4.10 การกำหนดค่าเริ่มต้นให้กับอาร์กิวเมนต์

บรรทัดที่	คำสั่ง
1	def func(a,b='',c=0):
2	return a,b,c
3	name,occupation, salary = func('เอกราช')
4	print(name)
5	name,occupation, salary = func('เอกราช','นักเขียนโปรแกรม')
6	print(name)
7	name,occupation, salary = func('เอกราช','นักเขียนโปรแกรม',35000)
8	print(name,occupation,salary)
ผลลัพธ์	
เอกราช เอกราช เอกราช นักเขียนโปรแกรม 35000	

การใช้งานพารามิเตอร์และอาร์กิวเมนต์ที่กล่าวมาในเบื้องต้น คือ ลักษณะของการกำหนดปริมาณในการใช้งาน ซึ่งในการส่งผ่านค่าใช้งานในโปรแกรมนั้นในบางครั้งมีจำนวนมาก ไม่สามารถระบุประมาณของการส่งค่าพารามิเตอร์ได้ จึงมีการใช้งานในรูปแบบของอาร์กิวเมนต์ที่ไม่ระบุ มาใช้งานแทน ซึ่งจะมีประสิทธิภาพและมีความเหมาะสมมากกว่า

การจัดวางตำแหน่งฟังก์ชันในโปรแกรม

การจัดวางตำแหน่งฟังก์ชันในโปรแกรม มีโครงสร้างและกฎเกณฑ์สร้างฟังก์ชันขึ้นมาใช้งาน ดังนั้น เพื่อไม่ให้เกิดการทำงานผิดพลาดของโปรแกรม การเขียนโปรแกรมจะต้องคำนึงถึง โดยมีรายละเอียดดังโปรแกรมตัวอย่างที่ 4.11

ตัวอย่างที่ 4.11 การจัดโครงสร้างโปรแกรมใช้งานร่วมกับฟังก์ชัน

บรรทัดที่	คำสั่ง
1	import math
2	def cir_area(redius):
3	a=math.pi*(redius**2)
4	return a
5	def rac_area(width, height):
6	r = width * height
7	return r
8	print('พื้นที่สี่เหลี่ยมผืนผ้า(1) = %d' % rac_area(8, 4))
9	print('พื้นที่วงกลม(1) = %f' % cir_area(10))
10	print('พื้นที่สี่เหลี่ยมผืนผ้า(2) = %d' % rac_area(5, 3))
11	print('พื้นที่วงกลม(2) = %f' % cir_area(7))
12	print('พื้นที่สี่เหลี่ยมผืนผ้า(3) = %d' % rac_area(11.5, 13.3))
13	print('พื้นที่วงกลม(3) = %f' % cir_area(6.5))
ผลลัพธ์	
พื้นที่สี่เหลี่ยมผืนผ้า(1) = 32 พื้นที่วงกลม(1) = 314.159265 พื้นที่สี่เหลี่ยมผืนผ้า(2) = 15 พื้นที่วงกลม(2) = 153.938040 พื้นที่สี่เหลี่ยมผืนผ้า(3) = 152 พื้นที่วงกลม(3) = 132.732290	

โปรแกรมตัวอย่างที่ 4.11 แบ่งส่วนประกอบการทำงาน 3 ส่วน ประกอบไปด้วย บรรทัดที่ 1 การนำเข้าฟังก์ชันที่มีอยู่ในโปรแกรมชื่อ math ชุดคำสั่งบรรทัดที่ 2-8 คือ ฟังก์ชันที่สร้างขึ้นมาใช้งาน

ประกอบด้วยฟังก์ชัน cir_area() ทำหน้าที่ในการหาพื้นที่วงกลมและฟังก์ชัน rac_area ทำหน้าที่ในหาพื้นที่สี่เหลี่ยมผืนผ้า ส่วนโปรแกรมหลักชุดคำสั่งอยู่ระหว่างบรรทัดที่ 9-14 ซึ่งจะเห็นได้ว่า เมื่อมีการสร้างฟังก์ชันขึ้นมาใช้งานและมีการทำงานที่ซ้ำกัน สามารถเรียกใช้ฟังก์ชันที่เกี่ยวข้องมาใช้งานโดยไม่จำเป็นต้องสร้างชุดโปรแกรมใหม่ที่เหมือนกัน

การสร้างฟังก์ชันนั้น สามารถนำการทำงานร่วมกันระหว่างฟังก์ชันที่สร้างขึ้นมาใหม่และฟังก์ชันที่อยู่ในโปรแกรม ดังโปรแกรมตัวอย่างที่ 4.11 ฟังก์ชัน cir_area() มีการนำโมดูล math ซึ่งมีฟังก์ชัน pi อยู่ในภาษาไพธอนรวมกันในการประมวลผล เพื่อหาค่าของข้อมูลที่มีความละเอียดมากกว่าการคำนวณแบบปกติ รูปแบบของของการทำงานร่วมกับฟังก์ชัน ส่วนของโปรแกรมหลักจะต้องอยู่ในส่วนล่างสุดเสมอ ถ้ามีการนำส่วนโปรแกรมหลักไปไว้ด้านบนของฟังก์ชัน โปรแกรมจะแจ้งผิดพลาดโปรแกรมทันที การทำงานของโปรแกรม จะไม่อนุญาตให้ส่วนของโปรแกรมหลักอยู่ก่อนฟังก์ชัน ดังโปรแกรมตัวอย่างที่ 4.12

ตัวอย่างที่ 4.12 การวางตำแหน่งที่ผิดของฟังก์ชัน

บรรทัดที่	คำสั่ง
1	print('พื้นที่สี่เหลี่ยมผืนผ้า(1) = %d' % rac_area(8, 4))
2	print('พื้นที่วงกลม(1) = %f' % cir_area(10))
3	print('พื้นที่สี่เหลี่ยมผืนผ้า(2) = %d' % rac_area(5, 3))
4	print('พื้นที่วงกลม(2) = %f' % cir_area(7))
5	print('พื้นที่สี่เหลี่ยมผืนผ้า(3) = %d' % rac_area(11.5, 13.3))
6	print('พื้นที่วงกลม(3) = %f' % cir_area(6.5))
7	def cir_area(redius):
8	pi=3.141
9	a=math.pi*(redius**2)
10	return a
11	def rac_area(width, height):
12	r = width * height
13	return r
14	import math

ผลลัพธ์

```
Traceback (most recent call last):
  File "D:\python_file\04_12.py", line 1, in <module>
```

```
    print('พื้นที่สี่เหลี่ยมผืนผ้า(1) = %d' % rac_area(8, 4))
NameError: name 'rac_area' is not defined
```

การแก้ไขให้ส่วนของโปรแกรมหลักอยู่ในรูปของฟังก์ชัน จะทำให้สามารถวางตำแหน่งส่วนใดของโปรแกรมได้และทำให้ชุดคำสั่งมีระเบียบ ง่ายต่อการตรวจสอบ เมื่อมีการประมวลผล จะไม่แสดงข้อผิดพลาดของโปรแกรมที่เกิดขึ้น การกำหนดฟังก์ชันโปรแกรมหลักใช้งาน การวางตำแหน่งจะสามารถวางได้ทุกตำแหน่งของโปรแกรมแต่จะต้องอยู่ก่อนการเรียกใช้ฟังก์ชันหลัก main() และฟังก์ชัน main() จะต้องอยู่ส่วนล่างสุดของโปรแกรม เพื่อป้องกันข้อผิดพลาดของโปรแกรม ดังโปรแกรมตัวอย่างที่ 4.13

ตัวอย่างที่ 4.13 การสร้างฟังก์ชันโปรแกรมหลัก

บรรทัดที่	คำสั่ง
1	def main():
2	print('พื้นที่สี่เหลี่ยมผืนผ้า(1) = %d' % rac_area(8, 4))
3	print('พื้นที่วงกลม(1) = %f' % cir_area(10))
4	print('พื้นที่สี่เหลี่ยมผืนผ้า(2) = %d' % rac_area(5, 3))
5	print('พื้นที่วงกลม(2) = %f' % cir_area(7))
6	print('พื้นที่สี่เหลี่ยมผืนผ้า(3) = %d' % rac_area(11.5, 13.3))
7	print('พื้นที่วงกลม(3) = %f' % cir_area(6.5))
8	def cir_area(redius):
9	pi=3.141
10	a=math.pi*(redius**2)
11	return a
12	def rac_area(width, height):
13	r = width * height
14	return r
15	import math
16	main()

ผลลัพธ์

```
พื้นที่สี่เหลี่ยมผืนผ้า(1) = 32
พื้นที่วงกลม(1) = 314.159265
พื้นที่สี่เหลี่ยมผืนผ้า(2) = 15
```

```
พื้นที่วงกลม(2) = 153.938040
พื้นที่สี่เหลี่ยมผืนผ้า(3) = 152
พื้นที่วงกลม(3) = 132.732290
```

การเขียนโปรแกรมที่มีขนาดใหญ่ ซับซ้อนและมีปริมาณของชุดคำสั่งจำนวนมาก การที่จะตรวจสอบแต่ละส่วนการทำงานของโปรแกรมมีความยุ่งยาก ดังนั้นการจัดกลุ่มคำสั่งตามหน้าที่ลักษณะฟังก์ชัน จึงมีความสำคัญ เช่น ส่วนของโปรแกรมหลัก กลุ่มคำสั่งอาจอยู่ในฟังก์ชัน เพื่อเป็นระเบียบและป้องกันการวางตำแหน่งที่ผิดพลาดของโปรแกรมที่เกิดขึ้น

การเรียกใช้โมดูลที่มีในโปรแกรม

จากการใช้งานฟังก์ชันและโมดูล จะเห็นได้ว่าสะดวกและมีความสำคัญต่อการพัฒนาโปรแกรมที่มีขนาดใหญ่ ซึ่งภาษาไพธอนได้มีฟังก์ชันและโมดูลในโปรแกรม การใช้งานเป็นจำนวนมาก ซึ่งถือว่าเป็นจุดเด่นอีกด้านของภาษาไพธอน (Rossum, 2013) ดังตัวอย่างโปรแกรมต่อไปนี้

ตัวอย่างที่ 4.14 การเรียกใช้โมดูลที่มีอยู่ในโปรแกรม

บรรทัดที่	คำสั่ง
1	import math
2	i=1
3	num=int(input("จำนวนตัวเลข = "))
4	while (i<=num):
5	number = float(input("ตัวเลข = "))
6	print("ค่าตัวเลขที่",i," คือ ",number)
7	print("ค่าตัวเลขจำนวนเต็มขั้นต่ำ คือ :", math.floor(number))
8	print("ค่าตัวเลขจำนวนเต็มขั้นสูง คือ :", math.ceil(number))
9	print("ค่าสัมบูรณ์ :", math.fabs(number))
10	print("-"*20)
11	i=i+1

ผลลัพธ์

```
จำนวนตัวเลข = 3                                    รับข้อมูลจากแผงแป้นอักขระ
ตัวเลข = 11.58                                     รับข้อมูลจากแผงแป้นอักขระ
```

```
ค่าตัวเลขที่ 1  คือ  11.58
ค่าตัวเลขจำนวนเต็มขั้นต่ำ คือ : 11
ค่าตัวเลขจำนวนเต็มขั้นสูง คือ : 12
ค่าสัมบูรณ์ : 11.58
--------------------
ตัวเลข = -44.30                                    รับข้อมูลจากแผงแป้นอักขระ
ค่าตัวเลขที่ 2  คือ  -44.3
ค่าตัวเลขจำนวนเต็มขั้นต่ำ คือ : -45
ค่าตัวเลขจำนวนเต็มขั้นสูง คือ : -44
ค่าสัมบูรณ์ : 44.3
--------------------
ตัวเลข = -99                                       รับข้อมูลจากแผงแป้นอักขระ
ค่าตัวเลขที่ 3  คือ  -99.0
ค่าตัวเลขจำนวนเต็มขั้นต่ำ คือ : -99
ค่าตัวเลขจำนวนเต็มขั้นสูง คือ : -99
ค่าสัมบูรณ์ : 99.0
--------------------
```

การเรียกใช้โมดูลที่มีอยู่ในโปรแกรม ตัวอย่างที่ 4.14 มีการนำโมดูล math เข้ามาใช้ในโปรแกรม ประกอบด้วยให้หาค่าตัวเลขจำนวนเต็มขั้นต่ำด้วยฟังก์ชัน floor บรรทัดที่ 7 ค่าตัวเลขจำนวนเต็มขั้นสูงด้วยฟังก์ชัน ceil บรรทัดที่ 8 และค่าสมบูรณ์ด้วยฟังก์ขัน fabs บรรทัดที่ 9 พร้อมทั้งแสดงค่าของข้อมูลตามจำนวนของตัวเลขที่นำเข้า

ดังนั้นฟังก์ขันที่มีหลักการทำงานเป็นมาตรฐานเดียวกัน จะถูกพัฒนาเก็บไว้เพื่อรองรับการเรียกใช้งาน แต่มีลักษณะของการทำงานโปรแกรมที่มีการทำงานเฉพาะ รูปแบบนี้ต้องสร้างโมดูลขึ้นมาใช้งานเอง เพื่อรองรับกับการทำงานตามเงื่อนไขของโปรแกรม ซึ่งนอกเหนือจากโมดูลที่มากับภาษาไพธอน ยังมีผู้พัฒนาได้สร้างโมดูลให้สนับสนุนการทำงานร่วมกับภาษาไพธอนอีกมากมาย จึงทำให้มีประสิทธิภาพ นิยมนำมาพัฒนาโปรแกรมในปัจจุบัน

สรุป

การสร้างโปรแกรมที่มีลักษณะการทำงานที่ถูกเรียกใช้งานบ่อยครั้ง เรียกโปรแกรมนี้ว่า ฟังก์ชัน ลักษณะของฟังก์ชัน ถือว่ามีความสำคัญในการเขียนโปรแกรม เนื่องจากจะอำนวยความสะดวกต่อการเรียกใช้ในการพัฒนาโปรแกรม ลักษณะฟังก์ชันที่สร้างขึ้นสามารถเก็บไว้ในโปรแกรมหลักหรือสามารถเก็บไว้เป็นรูปแบบการทำงานเดียวกันไว้ในที่เดียวกัน เรียกว่า โมดูล การเรียกใช้งานฟังก์ชันในโมดูลจะประกอบไปด้วยคำสั่ง import ที่ใช้ในการเรียกฟังก์ชัน ถ้าต้องการระบุชื่อฟังก์ชันสามารถใช้คำสั่งควบคู่กับ from ในการระบุฟังก์ชันในโมดูล ฟังก์ชันจะมีโครงสร้างการทำงานหน้าที่เฉพาะ รูปแบบการเรียกใช้ฟังก์ชัน จะมีการเรียกใช้โดยตรงหรือการส่งข้อมูลเข้าไปผ่านค่าพารามิเตอร์ ในฟังก์ชันจะมีตัวที่ทำหน้าที่ในการรับข้อมูลจากค่าพารามิเตอร์เข้าไป เรียกว่า อาร์กิวเมนต์ และดำเนินการประมวลผลลัพธ์ คืนค่ากลับมายังส่วนที่เรียกใช้ฟังก์ชัน โดยการใช้งานไม่กระทบต่อโครงสร้าง การทำงานของฟังก์ชัน การสร้างฟังก์ชันมีโครงสร้าง กฎเกณฑ์ของการวางตำแหน่งการทำงานของโปรแกรม ผู้เขียนโปรแกรมต้องพิจารณาเพื่อไม่ให้เกิดการผิดการในการทำงาน ฟังก์ชันและโมดูลมีลักษณะของการเรียกใช้ คือ ส่วนของภาษาที่เตรียมไว้ให้และที่พัฒนาขึ้นมาใช้งานเองโดยเฉพาะ นอกเหนือจากนี้ยังมีนักเขียนโปรแกรมได้สร้างโมดูลที่สนับสนุนการทำงานร่วมกับภาษาไพธอนมากมาย ซึ่งเป็นข้อดีที่นักพัฒนาโปรแกรมในปัจจุบัน ได้นำไปประยุกต์ใช้งานเพื่อให้ได้โปรแกรมที่มีประสิทธิภาพการทำงานสูงสุดตามวัตถุประสงค์ที่ต้องการ

แบบฝึกหัด

1. ให้ผู้เรียนอธิบายความหมาย การทำงาน หน้าที่และหลักการทำงานของฟังก์ชัน

2. ให้ผู้เรียนอธิบายความแตกต่างระหว่างฟังก์ชันและโมดูล

3. ให้ผู้เรียนบอกคำสั่งที่เรียกใช้ฟังก์ชันและโมดูลมีอะไรบ้าง รูปแบบการทำงานของแต่ละคำสั่งมีลักษณะอย่างไร

4. ให้ผู้เรียนอธิบายความหมายของพารามิเตอร์และอาร์กิวเมนต์

5. ให้ผู้เรียนอธิบายลักษณะฟังก์ชันในโมดูล มีกี่ประเภท แต่ละประเภทมีลักษณะและจุดประสงค์ของการทำงานอย่างไร

6. ให้ผู้เรียนอธิบายการผิดพลาดที่เกิดขึ้นของการส่งค่าผ่านพารามิเตอร์และอาร์กิวเมนต์มีอะไรบ้าง

7. ให้ผู้เรียนอธิบายหลักเกณฑ์และวิธีการจัดตำแหน่งของฟังก์ชันในโปรแกรมให้ถูกต้องมีลักษณะอย่างไร

8. ให้ผู้เรียนสร้างฟังก์ชันคำนวณคะแนนรวมและสร้างโมดูลทำหน้าที่ในการประมวลผลเกรดที่ได้

80-100	เกรด A	75-79	เกรด B+
70-74	เกรด B	65-69	เกรด C+
60-64	เกรด C	55-59	เกรด D+
50-54	เกรด D	0-49	เกรด E

คะแนนรวมได้มาจาก คะแนนเก็บ คะแนนกลางภาค และคะแนนปลายภาค

9. ให้ผู้เรียนยกตัวอย่างการทำงานและเขียนโปรแกรมที่มีลักษณะของการเรียกใช้โมดูล

10. ให้ผู้เรียนยกตัวอย่างการเรียกใช้โมดูลที่มีอยู่ในภาษาไพธอน

เอกสารอ้างอิง

โชติพันธุ์ หล่อเลิศสุนทร และ ฐิตะพันธุ์ หล่อเลิศสุนทร. (2559). **คู่มือเรียนเขียนโปรแกรม python (ภาคปฏิบัติ)**. กรุงเทพฯ: คอร์ฟังก์ชั่น.

ธีรวัฒน์ ประกอบผล. (2558). **การเขียนแอพพลิเคชันด้วย Visual Basic 2010**. กรุงเทพฯ: ซิมพลิฟลาย.

บัญชา ปะสีละเตสัง. (2558). **สร้าง Windows Application ด้วย Visual Basic 2015**. กรุงเทพฯ: ซีเอ็ดยูเคชั่น.

สุชาติ คุ้มมณี. (2558). **เชี่ยวชาญการเรียนด้วยภาษาไพธอน**. มหาสารคาม. คณะวิทยาการสารสนเทศ: มหาวิทยาลัยมหาสารคาม.

Fangohr, Hans. (2015). **Introduction to Python for Computational Science and Engineering**. Southampton: University of Southampton.

Halterman, Richard L. (2016). **Fundamentals of Python Programming**. Tennessee: Southern Adventist University.

Lee, Kent D. (2011). **Python Programming Fundamentals**. London: Springer.

Lutz, Mark. (2013). **Learning Python**. 5rd ed. Massachusetts: O'Reilly Media.

Rossum, Guido van. (2013). **Python Tutorial Release 3.3.2**. Amsterdam: Python Software Foundation.

Sebesta, Robert W. (2016). **Concepts of Programming Languages** . 11th ed, London: Pearson.