แผนบริหารการสอนประจำบทที่ 8

การจัดการข้อผิดพลาดของโปรแกรม

หัวข้อประจำบท

1. เบื้องต้นเกี่ยวกับข้อผิดพลาดของโปรแกรม
2. การจัดการข้อผิดพลาด

วัตถุประสงค์เชิงพฤติกรรม

1. เพื่อให้ผู้เรียนสามารถอธิบายเกี่ยวกับการจัดการข้อผิดพลาดได้
2. เพื่อให้ผู้เรียนสามารถอธิบายเกี่ยวกับประเภทการจัดการข้อผิดพลาดได้
3. เพื่อให้ผู้เรียนสามารถอธิบายเกี่ยวกับคำสั่งที่ใช้ในการจัดการข้อผิดพลาดได้
4. เพื่อให้ผู้เรียนสามารถนำความรู้เรื่อง การจัดการข้อผิดพลาด 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 8
การจัดการข้อผิดพลาดของโปรแกรม

การประมวลผลชุดคำสั่งของโปรแกรมที่พัฒนาขึ้น มีโอกาสความเป็นไปได้ที่จะเกิดการผิดพลาดของโปรแกรม ซึ่งเมื่อโปรแกรมมีข้อผิดพลาดขึ้น โปรแกรมจะหยุดการทำงานทันทีและโปรแกรมจะไม่สามารถดำเนินการต่อไปได้ ดังนั้นในโปรแกรมระดับสูงปัจจุบัน จึงมีวิธีการตรวจจับข้อผิดพลาดที่จะเกิดขึ้นของโปรแกรม เพื่อไม่ให้เกิดปัญหาการหยุดชะงักของโปรแกรม ซึ่งในภาษาไพธอน มีรูปแบบของการตรวจจับข้อมูลที่เกิดขึ้นหลากหลายรูปแบบ ซึ่งกระบวนการที่ป้องกันข้อผิดพลาดที่เกิดขึ้น เรียกว่า “การจัดการข้อผิดพลาดของโปรแกรม”

เบื้องต้นเกี่ยวกับข้อผิดพลาดของโปรแกรม

ข้อผิดพลาด (Exception) หมายถึง เหตุการณ์ที่ทำให้เกิดการขัดจังหวะขึ้นหรือรบกวนการทำงาน ช่วงระหว่างของการประมวลผลโปรแกรมผิดปกติออกไป (สุชาติ คุ้มมณี, 2558) ดังนั้นเพื่อควบคุมการทำงานตั้งแต่ข้อมูลเข้าและการประมวลผลของโปรแกรม หลีกเลี่ยงการทำงานที่ผิดพลาดของโปรแกรม จึงต้องมีส่วนที่ป้องกันการทำงานของโปรแกรม (Defensive Programming) (Gerrard, 2016) ข้อผิดพลาดที่เกิดขึ้นนั้น ประกอบไปด้วย 3 ส่วน คือ ส่วนแรก คือ ข้อผิดพลาดที่เกิดขึ้นจากการไม่ปฏิบัติตามหลักของภาษาโปรแกรมหรือไวยากรณ์ ส่วนที่สอง คือ ข้อผิดพลาดในช่วงระหว่างการประมวลผลคำสั่ง และส่วนที่สาม คือ ข้อผิดพลาดที่เกิดจากการได้ผลลัพธ์ไม่ถูกต้องหรือตรงกับความต้องการ (โชติพันธุ์ หล่อเลิศสุนทร และ ฐิตะพันธุ์ หล่อเลิศสุนทร, 2559)

ภาษาไพธอน มีข้อผิดพลาดที่เกิดขึ้นในการทำงานของโปรแกรมแต่ละส่วน มีรายละเอียดแต่ละข้อผิดพลาด ดังต่อไปนี้

1. ข้อผิดพลาดที่ไม่ปฏิบัติตามรูปแบบภาษาโปรแกรม

ข้อผิดพลาดที่ไม่ปฏิบัติตามรูปแบบหรือไวยากรณ์ของภาษาโปรแกรม (Syntax Error) คือการใช้ประโยคคำสั่งไม่ถูกต้องตามรูปแบบของภาษาโปรแกรม ซึ่งเกิดจากความไม่เข้าใจการใช้รูปแบบของประโยคคำสั่ง เมื่อเกิดข้อผิดพลาดจะต้องทำการแก้ไขให้ถูกหลักไวยากรณ์ หากไม่ดำเนินการแก้ไข โปรแกรมก็ไม่สามารถประมวลผลคำสั่งต่อได้ (สุชาติ คุ้มมณี, 2558) ซึ่งมีรายละเอียดของการผิดพลาดที่ไม่ปฏิบัติตามรูปแบบภาษาโปรแกรมตัวอย่างที่ 8.1

```
>>> print("Computer Technology

SyntaxError: EOL while scanning string literal
>>>
```

ภาพที่ 8.1 ข้อผิดพลาดที่ไม่ปฏิบัติตามรูปแบบภาษาโปรแกรม

ภาพที่ 8.1 ได้ใช้คำสั่ง print แสดงข้อความ Computer Technology แต่ขาดอักขระสุดท้าย คือ ") ที่ทำหน้าที่ในการปิดประโยคคำสั่งที่ใช้ในการแสดงข้อความ เมื่อโปรแกรมประมวลผลคำสั่งตรวจสอบไม่ตรงกับไวยากรณ์ของภาษาโปรแกรม ภาษาโปรแกรมจะทำการหยุดการประมวลผลและแสดงข้อผิดพลดที่เกิดขึ้น โดยระบุส่วนที่เป็นไวยากรณ์ คือ SyntaxError: ตามด้วยเหตุผลของการผิดพลาดที่เกิดขึ้น

2. ข้อผิดพลาดที่เกิดจากการประมวลคำสั่ง

ข้อผิดพลาดที่เกิดจากการประมวลคำสั่ง (Runtime Error) คือ ข้อผิดพลาดที่เกิดจากการใช้คำสั่งที่ผิด ซึ่งอาจเกิดจากการใช้คำสั่งไม่ถูกต้องหรือคำสั่งนั้นไม่มีในภาษาโปรแกรม (โชติพันธุ์ หล่อเลิศสุนทร และ ฐิตะพันธุ์ หล่อเลิศสุนทร, 2559) มีรายละเอียดของข้อผิดพลาด ดังโปรแกรมตัวอย่างภาพที่ 8.2

```
>>> prin("Computer Technology")
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    prin("Computer Technology")
NameError: name 'prin' is not defined
>>>
```

ภาพที่ 8.2 ข้อผิดพลาดที่เกิดขึ้นจากการประมวลผลคำสั่ง

ภาพที่ 8.2 มีการใช้คำสั่งแสดงข้อความที่ผิด โดยใช้คำสั่ง prin ขาดอักษร t เมื่อมีการประมวลผลผลคำสั่ง ภาษาโปรแกรมจะตรวจสอบคำสั่ง และแสดงข้อผิดพลาด โดยจะแจ้งบรรทัดของคำสั่งที่ไม่ถูกต้อง File "<pyshell#0>", line 1, in <module> พร้อมทั้งบอกประโยคคำสั่งที่ผิด prin("Computer Technology") และจุดที่เกิดข้อผิดพลาด NameError: name 'prin' is not defined ซึ่งดังผลลัพธ์ที่แจ้ง คือ คำสั่งนี้ไม่ได้ถูกกำหนดใช้งานภายในภาษาโปรแกรม

3. ข้อผิดพลาดที่เกิดจากผลลัพธ์ไม่ถูกต้อง

ข้อผิดพลาดจากการเขียนโปรแกรมผิดหลักไวยากรณ์และข้อผิดพลาดจากการใช้คำสั่งตัวแปลภาษาอินเทอร์พรีเตอร์สามารถตรวจสอบข้อผิดพลาดที่เกิดขึ้นระหว่างแปลแต่ส่วนและแจ้งผล

กลับจากการประมวลผลที่ได้ ทั้งสองส่วนเป็นปัญหาที่เกิดจากการละเมิดต่อภาษาโปรแกรม ซึ่งเป็นงานต่อการแก้ไข เพราะตัวแปลภาษาสามารถระบุข้อที่ผิดพลาดที่เกิดขึ้นในชุดคำสั่งของโปรแกรมเป็นส่วนที่เกี่ยวข้องกับภาษาโปรแกรมโดยตรง แต่มีข้อผิดพลาดอีกประเภทที่มีความแตกต่าง คือ ข้อผิดพลาดที่เกิดจากผลลัพธ์ไม่ถูกต้อง (Logic Error) ตัวแปลภาษาอินเทอร์พรีเตอร์ไม่สามารถตรวจสอบข้อผิดพลาดได้และผลลัพธ์ที่ได้ไม่ถูกต้อง (Halterman, 2016) ข้อผิดพลาดที่เกิดขึ้นในกรณีนี้ ส่วนมากเกิดจากการไม่เข้าใจหลักการทำงานของภาษาโปรแกรม จึงทำให้เกิดข้อผิดพลาดขึ้น ดังโปรแกรมตัวอย่างที่ 8.1

ตัวอย่างที่ 8.1 ข้อผิดลพลาดที่เกิดจากผลลัพธ์ไม่ถูกต้อง

บรรทัดที่	คำสั่ง
1	a = 9+5-3+4/2**2+6/3*3
2	b =(9+5-3+4)/2**2+(6/3*3)
3	total_point1 = 65 + 35 /2
4	total_point2 = (65 + 35) /2
5	print(a)
6	print(b)
7	print(total_point1)
8	print(total_point2)
ผลลัพธ์	
18.0 9.75 82.5 50.0	

โปรแกรมตัวอย่างที่ 8.1 ไม่มีข้อผิดพลาดจากการประมวลผล แต่ผลลัพธ์ที่ได้จะมีความแตกต่างกัน การเขียนโปรแกรมจะต้องพิจารณาถึงผลลัพธ์ที่ได้ตรงตามจุดประสงค์หรือขั้นตอนการทำงานของโปรแกรมหรือไม่ เช่น ต้องการหาผลรวมของคะแนนเก็บและคะแนนกลางภาค นำผลรวมที่ได้หารด้วยสอง ผลลัพธ์ที่ได้จะเป็นคะแนนเก็บที่จะนำไปใช้ในการตัดเกรดขั้นตอนต่อไป ดังตัวอย่างบรรทัดที่ 3 และ 4 เมื่อทำการประมวล ผลลัพธ์ที่ได้จะมีความแตกต่างกัน ซึ่งจากเงื่อนดังกล่าว ผลลัพธ์ที่ได้จากบรรทัดที่ 3 เป็นข้อผิดพลาดที่เกิดขึ้น เพราะนำตัวเลขคะแนนกลางภาคไปทำการหารด้วยสองก่อน ถึงนำมาบวกกับคะแนนเก็บ ซึ่งไม่ตรงกับขั้นตอนการทำงานของโปรแกรม

จากข้อผิดพลาดที่เกิดขึ้น ภาษาโปรแกรมไม่สามารถตรวจสอบได้ ผู้เขียนโปรแกรมจะต้องตรวจสอบขั้นตอนการทำงานของโปรแกรมหรืออัลกอริทึมที่ได้มาของผลลัพธ์ที่ถูกต้องด้วยตนเอง นอกเหนือจากข้อผิดพลาดที่กล่าวมา อาจมีการผิดพลาดจากการประมวลผลของระบบคอมพิวเตอร์อื่น เช่น พื้นที่ในการเก็บข้อมูลไม่เพียงพอ การเชื่อมต่อระบบเครือข่ายไม่สามารถทำได้ เกิดการหยุดของระบบเนื่องจากหน่วยความจำไม่เพียงพอ เป็นต้น

การจัดการข้อผิดพลาด

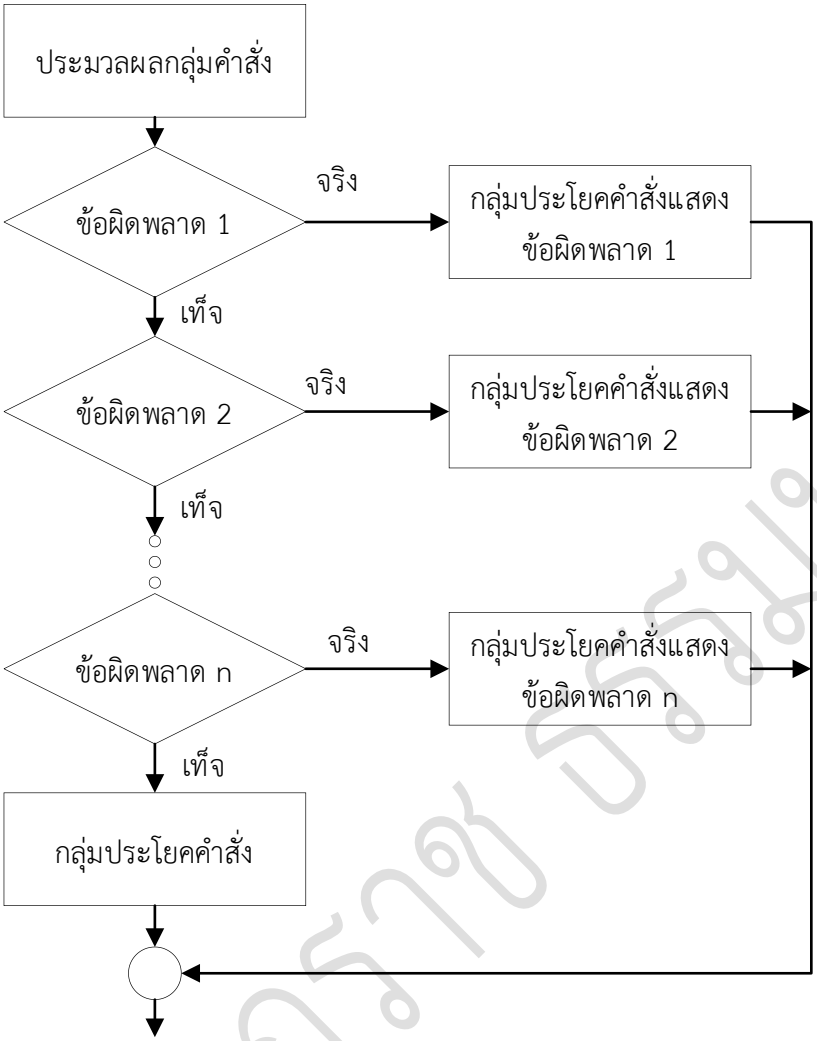
ประสิทธิภาพการทำงานของโปรแกรม เกิดจากความคงทนที่โปรแกรมขณะทำงานจะไม่เกิดความล้มเหลวหรือหยุดการทำงาน ภาษาไพธอนได้มีโครงสร้างคำสั่งที่จะจัดการตรวจจับข้อผิดพลาดของโปรแกรมไม่ให้เกิดความล้มเหลวที่เกิดขึ้น เรียกว่า การจัดข้อผิดพลาด (Exceptions Handling) เพื่อให้โปรแกรมนั้นเกิดความมั่นคงและคงทนต่อการทำงาน (สุชาติ คุ้มมณี, 2558) หรือสิ่งที่มีลักษณะทำงานเฉพาะที่จะถูกเรียกใช้ เมื่อเจอสถานะการที่ไม่ปกติในการทำงานของโปรแกรม (Halterman, 2016) มีรายละเอียดของการจัดการข้อผิดพลาดดังต่อไปนี้

1. การตรวจจับข้อผิดพลาดด้วย try...except...else

โครงสร้างคำสั่ง try...except...else คือ การตรวจจับข้อผิดพลาดของโปรแกรมซึ่งมีโครงสร้างการทำงานของคำสั่งดังต่อไปนี้

```
try:
        ประมวลผลกลุ่มคำสั่ง
        except ข้อผิดพลาดที่ 1 :
                กลุ่มประโยคคำสั่งแสดงข้อผิดพลาด 1
        except ข้อผิดพลาดที่ 2 :
                กลุ่มประโยคคำสั่งแสดงข้อผิดพลาด 2
                        .
                        .
                        .
        except ข้อผิดพลาดที่ n :
                กลุ่มประโยคคำสั่งแสดงข้อผิดพลาด n
        else :
                กลุ่มประโยคคำสั่ง
```

รูปแบบการตรวจสอบข้อผิดพลาดของโปรแกรม สามารถอธิบายเป็นแผนภาพ มีรายละเอียดโครงสร้างการทำงานดังต่อไปนี้



ภาพที่ 8.3 โครงสร้างการตรวจจับข้อผิดพลาดของโปรแกรม

จากโครงสร้างคำสั่งการตรวจจับข้อผิดพลาดของโปรแกรม สามารถนำไปใช้ในการเขียนโปรแกรม มีรายละเอียดการทำงานดังโปรแกรมตัวอย่างที่ 8.2

ตัวอย่างที่ 8.2 การตรวจสอบข้อผิดพลาดด้วย try...except...else

บรรทัดที่	คำสั่ง
1	try:
2	val_a=float(input("ตัวแปร a มีค่า = "))
3	val_b=float(input("ตัวแปร b มีค่า = "))
4	val_x=val_a/val_b
5	except ZeroDivisionError:
6	print("มีการหารด้วยศูนย์จะไม่ดำเนินการต่อ")

```
7   except ValueError:
8       print("ค่าตัวแปร a หรือ b ไม่ใช่ตัวเลข")
9   else:
10      print("ผลลัพธ์ที่ได้ = ",val_x)
```

ผลลัพธ์ ที่ 1

ตัวแปร a มีค่า = 10 (รับข้อมูลจากแผงแป้นอักขระ)
ตัวแปร b มีค่า = 0 (รับข้อมูลจากแผงแป้นอักขระ)
มีการหารด้วยศูนย์จะไม่ดำเนินการต่อ

ผลลัพธ์ ที่ 2

ตัวแปร a มีค่า = 5 (รับข้อมูลจากแผงแป้นอักขระ)
ตัวแปร b มีค่า = y (รับข้อมูลจากแผงแป้นอักขระ)
ค่าตัวแปร a หรือ b ไม่ใช่ตัวเลข

ผลลัพธ์ ที่ 3

ตัวแปร a มีค่า = 10 (รับข้อมูลจากแผงแป้นอักขระ)
ตัวแปร b มีค่า = 10 (รับข้อมูลจากแผงแป้นอักขระ)
ผลลัพธ์ที่ได้ = 1.0

โปรแกรมตัวอย่างที่ 8.2 การใช้โครงสร้างคำสั่งตรวจจับข้อผิดพลาดของโปรแกรมด้วยคำสั่ง try...except...else โดยในส่วนของ try จะเก็บคำสั่งที่ใช้ในการประมวลผลหลักของโปรแกรม โดยการทำงานของโปรแกรม จะให้มีการรับค่าข้อมูลจากแป้นพิมพ์อักขระ และนำค่าของข้อมูลที่ได้มาทำการหาร ซึ่งการประมวลนี้พิจารณามีความเป็นไปได้ที่จะเกิดข้อผิดพลาดการทำงาน ดังนี้

กรณีที่ป้อนข้อมูลที่เป็นตัวตั้งเป็นตัวเลข แต่ตัวหารเป็น 0 เมื่อมีการประมวลผลจะไม่มีผลลัพธ์เกิดขึ้น จะเกิดการล้มเหลวและหยุดการทำงานของโปรแกรม พร้อมทั้งแจ้งข้อผิดพลาด ZeroDivisionError: division by zero ดังนั้นเพื่อไม่ให้หยุดการทำงาน สามารถใช้การตรวจสอบข้อผิดพลาดที่สร้างขึ้นในโปรแกรม exception ValueError: พร้อมทั้งกำหนดการแจ้งข้อผิดพลาด “มีการหารด้วยศูนย์จะไม่ดำเนินการต่อ” โดยโปรแกรมไม่หยุดการทำงาน ดังผลลัพธ์ที่ 1

กรณีที่ป้อนข้อมูลที่เป็นตัวตั้งหรือตัวหารไม่เป็นตัวเลข เมื่อมีการประมวลผลจะไม่มีผลลัพธ์เกิดขึ้น จะเกิดการล้มเหลวและหยุดการทำงานของโปรแกรม พร้อมทั้งแจ้งข้อผิดพลาด ValueError: could not convert string to float: 'y' ดังนั้นเพื่อไม่ให้หยุดการทำงาน สามารถใช้การตรวจสอบข้อผิดพลาดที่สร้างขึ้นในโปรแกรม except ValueError: พร้อมทั้งกำหนดการแจ้งข้อผิดพลาด “ค่าตัวแปร a หรือ b ไม่ใช่ตัวเลข” โดยโปรแกรมไม่หยุดการทำงาน ดังผลลัพธ์ที่ 2

กรณีที่ป้อนข้อมูลที่เป็นตัวตั้งเป็นตัวเลขและตัวหารเป็นตัวเลขที่ไม่ใช่ 0 เมื่อมีการประมวลผลจะเป็นไปตามเงื่อนไขและขั้นตอนการทำงานของโปรแกรม จึงไม่เกิดข้อผิดพลาด ดังนั้นจะไม่ใช้ except แต่จะใช้ else: คำสั่งนี้เป็นคำสั่งที่จะประมวลผลหลังจากไม่เกิดข้อผิดพลาด ดังผลลัพธ์ที่ 3

ดังนั้นการพิจารณา การใช้การตรวจสอบข้อผิดพลาดของโปรแกรม จะประกอบไปด้วยการกำหนดเงื่อนไขดังต่อไปนี้

try คือ คำสั่งที่ตรวจสอบส่วนของโปรแกรมที่คาดว่าจะมีข้อผิดพลาดหรือความล้มเหลวจากการทำงานของโปรแกรม except หมายถึง คำสั่งที่ใช้ในการตรวจสอบข้อมูลผิดพลาดที่เกิดขึ้นของโปรแกรม โดยสามารถตรวจสอบได้หลายเงื่อนไข แต่ละเงื่อนไขในการตรวจสอบจะต้องระบุข้อผิดพลาดที่เกิดขึ้น ซึ่งในภาษาโปรแกรมได้เตรียมเครื่องที่ใช้ในการตรวจสอบที่มีอยู่ในภาษาโปรแกรม คำสั่ง except จะต้องใช้คู่กับคำสั่ง try ทุกครั้ง else หมายถึง คำสั่งส่วนสุดท้ายที่จะประมวลผลชุดคำสั่ง ในกรณีที่ไม่เกิดการผิดพลาดขึ้นในโปรแกรม

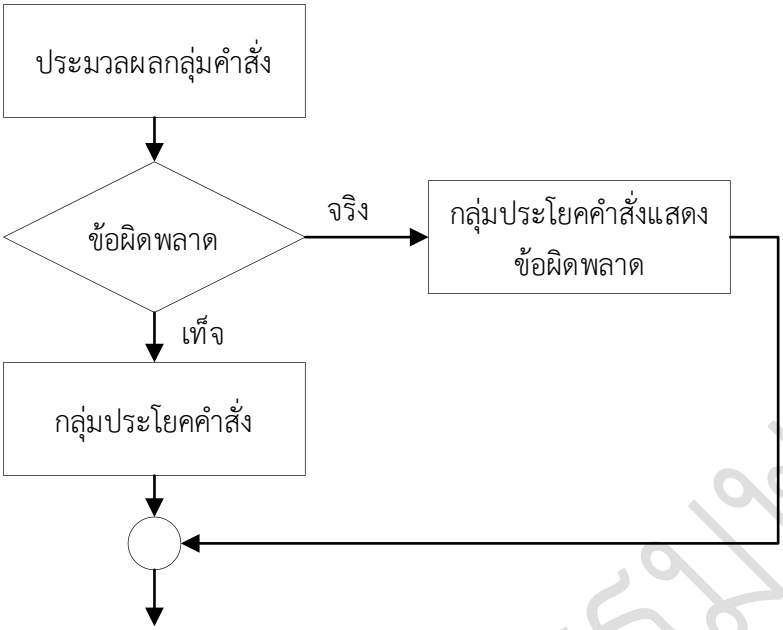
การตรวจสอบข้อผิดพลาดที่มีการกำหนด exception ข้อดีก็คือ จะสามารถระบุข้อผิดพลาดที่เกิดขึ้นได้ชัดเจน มีการกำหนดเงื่อนไขได้ไม่จำกัด ตามการวิเคราะห์ข้อผิดพลาดที่เกิดขึ้นได้เหมาะสมกับการนำไปใช้กับรูปแบบของโปรแกรมที่มีความซับซ้อนมาก มีประสิทธิภาพในการตรวจสอบข้อผิดพลาดสูง แต่จะมีข้อบกพร่อง คือ มีความยากลำบากที่จะระบุข้อผิดพลาดที่เกิดขึ้นโดยตรงของแต่ละกรณี ซึ่งจะต้องมีความเชี่ยวชาญ ความเข้าใจและวิธีการตรวจสอบข้อผิดพลาดของภาษาโปรแกรม

2. การตรวจจับข้อผิดพลาดโดยไม่กำหนด exception

การเขียนโปรแกรมตรวจสอบข้อผิดพลาดที่เกิดขึ้น บางกรณีที่ไม่สามารถระบุข้อผิดพลาดที่เกิดขึ้นได้และต้องการให้แสดงข้อผิดพลาดทั้งหมดมีผลต่อการทำงานในภาพรวม ดังนั้นจึงไม่ต้องกำหนดเงื่อนไขของ exception โดยมีรูปแบบคำสั่งดังต่อไปนี้

```
try:
        ประมวลผลกลุ่มคำสั่ง
except :
        กลุ่มประโยคคำสั่งแสดงข้อผิดพลาด
else :
        กลุ่มประโยคคำสั่ง
```

ลักษณะรูปแบบการทำงานของการตรวจสอบข้อผิดพลาดโดยไม่กำหนด exception สามารถอธิบายโครงสร้างการทำงาน ได้ดังต่อไปนี้



ภาพที่ 8.4 โครงสร้างการตรวจสอบข้อผิดพลาดโดยไม่กำหนด exception

จากรูปแบบการใช้คำสั่งตรวจสอบข้อผิดพลาดโดยไม่กำหนด exception มีรูปแบบการทำงาน ดังโปรแกรมตัวอย่างที่ 8.3

ตัวอย่างที่ 8.3 การตรวจสอบข้อผิดพลาดโดยไม่กำหนด exception

บรรทัดที่	คำสั่ง
1	try:
2	val_a=float(input("ตัวแปร a มีค่า = "))
3	val_b=float(input("ตัวแปร b มีค่า = "))
4	val_x=val_a/val_b
5	except :
6	print("มีข้อผิดพลาดเกิดขึ้น")
7	else:
8	print("ผลลัพธ์ที่ได้ = ",val_x)
ผลลัพธ์ที่ 1	
ตัวแปร a มีค่า = 10 (รับข้อมูลจากแผงแป้นอักขระ) ตัวแปร b มีค่า = 0 (รับข้อมูลจากแผงแป้นอักขระ) มีข้อผิดพลาดเกิดขึ้น	
ผลลัพธ์ที่ 2	

ตัวแปร a มีค่า = 10	(รับข้อมูลจากแผงแป้นอักขระ)
ตัวแปร b มีค่า = y	(รับข้อมูลจากแผงแป้นอักขระ)
มีข้อผิดพลาดเกิดขึ้น	

จากโปรแกรมตัวอย่างที่ 8.3 การตรวจสอบข้อผิดพลาดโดยไม่กำหนด exception โปรแกรมบรรทัดที่ 5 ซึ่งไม่ได้ระบุข้อผิดพลาดที่จะเกิดขึ้นในโปรแกรม ข้อผิดพลาดนั้นสามารถเกิดได้หลายกรณี แต่จะสรุปผลเป็นภาพรวมของผลที่เกิดขึ้นเป็นกรณีเดียว ดังผลลัพธ์ที่ 1 และ 2 จะแสดงผลลัพธ์ที่เหมือนกัน คือ “มีข้อผิดพลาดเกิดขึ้น” แต่ทั้งสองกรณีจะมีเงื่อนไขข้อผิดพลาดที่แตกต่างกัน ดังโปรแกรมตัวอย่างที่ 8.2 ลักษณะเช่นนี้คือการไม่กำหนดข้อผิดพลาดของโปรแกรม

3. การตรวจจับข้อผิดพลาด Exception โดยกำหนดเป็นชุด

การตรวจสอบข้อผิดพลาดของโปรแกรมแต่ละ Exception สามารถกำหนดข้อผิดพลาดที่จะตรวจจับได้หลายข้อผิดพลาดในแต่ละ Exception โดยจัดกลุ่มผิดพลาดที่มีลักษณะเดียวกันไว้ใน Exception เดียวกันหรือกำหนดเป็นชุดเดียวกัน มีรายละเอียดรูปแบบการทำงาน ดังโปรแกรมตัวอย่างที่ 8.4

ตัวอย่างที่ 8.4 การตรวจสอบข้อผิดพลาด Exception โดยกำหนดเป็นชุด

บรรทัดที่	คำสั่ง
1	try:
2	val_a=float(input("ตัวแปร a มีค่า = "))
3	val_b=float(input("ตัวแปร b มีค่า = "))
4	val_x=val_a/val_b
5	except ZeroDivisionError:
6	print("มีการหารด้วยศูนย์ ไม่สามารถดำเนินการต่อได้")
7	except (NameError,ValueError,OSError):
8	print("การประมวลผลไม่เสร็จสมบูรณ์")
9	else:
10	print("ผลลัพธ์ที่ได้ = ",val_x)

ผลลัพธ์

ตัวแปร a มีค่า = 10	(รับข้อมูลจากแผงแป้นอักขระ)
ตัวแปร b มีค่า = y	(รับข้อมูลจากแผงแป้นอักขระ)
การประมวลผลไม่เสร็จสมบูรณ์	

โปรแกรมตัวอย่างที่ 8.4 ได้มีการตรวจสอบข้อผิดพลาดของโปรแกรม 2 Exception ประกอบไปด้วย ZeroDivisionError จะตรวจสอบข้อผิดพลาดที่เกิดจากการหารด้วย 0 ซึ่งเป็น Exception ที่กำหนดเพียงอันเดียว แต่จะมี Exception ในบรรทัดที่ 7 จะมีการตรวจสอบข้อผิดพลาดที่เกิดจากการใช้คำสั่งและข้อมูลที่รับเข้ามาไม่ถูกต้อง คือ NameError ValueError และ OSError จะอยู่ใน Exception เดียวกัน เมื่อตรวจสอบข้อผิดพลาดที่เกิดจาก Exception นี้ จะแสดงข้อความ “การประมวลผลไม่เสร็จสมบูรณ์”

4. การตรวจจับข้อผิดพลาดหลาย Exception

การทำงานของชุดคำสั่งในโปรแกรม บางชุดคำสั่งอาจจะคาดว่ามีความเป็นไปได้ที่จะเกิดข้อผิดพลาดในการทำงานได้หลายชุดคำสั่งในแต่ละช่วงการทำงานของโปรแกรม ดังนั้นจึงมีความจำเป็นที่จะต้องดักจับข้อผิดพลาดที่เกิดขึ้น ภาษาไพธอน จึงอนุญาตในการใช้คำสั่ง try...except ได้หลายครั้ง เพื่อที่จะตรวจสอบข้อผิดพลาดของโปรแกรมที่จะเกิดขึ้น ซึ่งมีรูปแบบการใช้คำสั่งดังโปรแกรมตัวอย่างที่ 8.5

ตัวอย่างที่ 8.5 การตรวจสอบข้อผิดพลาดหลาย Exception

บรรทัดที่	คำสั่ง
1	try:
2	val_a=float(input("ตัวแปร a มีค่า = "))
3	val_b=float(input("ตัวแปร b มีค่า = "))
4	val_x=val_a/val_b
5	except ZeroDivisionError:
6	print("มีการหารด้วยศูนย์ ไม่สามารถดำเนินการต่อได้")
7	except (NameError,ValueError):
8	print("ฟังก์ชัน หรือ ค่าตัวแปร ไม่ถูกต้อง")
9	else:
10	print("ผลลัพธ์ที่ได้ = ",val_x)
11	
12	try:
13	obj_file=open("d:\\python_file\file_test.txt",'w')
14	except (OSError):
15	print("การเขียนข้อมูลไม่สำเร็จ")

ผลลัพธ์	
ตัวแปร a มีค่า = 10	(รับข้อมูลจากแผงแป้นอักขระ)
ตัวแปร b มีค่า = 0	(รับข้อมูลจากแผงแป้นอักขระ)
มีการหารด้วยศูนย์ ไม่สามารถดำเนินการต่อได้	
การเขียนข้อมูลไม่สำเร็จ	

การทำงานของโปรแกรมตัวอย่างที่ 8.5 กลุ่มคำสั่งแรก ลักษณะของการทำงานจะให้ทำการประมวลผลข้อมูลที่รับเข้าจากแป้นพิมพ์อักขระและให้ดักจับข้อผิดพลาดของข้อมูลที่รับเข้าที่มี Exception ที่กำหนดเงื่อนไขข้อผิดพลาดเดียว (ZeroDivisionError) และเป็นชุด (NameError และ ValueError) กลุ่มที่สอง จะนำข้อมูลที่ได้จากการประมวลผลที่ได้จากกลุ่มแรก ทำการเขียนข้อมูลลงในแฟ้มข้อมูลและมีการดักจับข้อผิดพลาดที่เกิดขึ้น (OSError) ที่เกิดจากการไม่สามารถเขียนข้อมูลลงในแฟ้มข้อมูลได้ ดังผลลัพธ์ที่ได้โปรแกรมตัวอย่างที่ 8.5

จากการทำงานดังกล่าวแสดงให้เห็นถึงการใช้คำสั่ง try...except ตรวจจับข้อผิดพลาดที่เกิดขึ้นได้มากกว่า 1 คำสั่ง ดังนั้น ในการเขียนโปรแกรมจึงสามารถดักจับข้อผิดพลาดที่คาดว่าจะเกิดขึ้น ในแต่ละส่วนของการทำงานได้ ทั้งนี้เพื่อป้องกันเหตุการณ์ที่โปรแกรมจะหยุดการทำงานทันที ซึ่งจะส่งผลเสียต่อระบบการทำงานของโปรแกรม

5. การตรวจจับข้อผิดพลาดโดยคำสั่ง try...finally

การตรวจจับข้อผิดพลาดที่เกิดขึ้นนั้น หากมีข้อผิดพลาดตรงตามที่กำหนด จะกระทำต่อชุดคำสั่งที่อยู่ใน exception นั้น หรือถ้าไม่มีข้อผิดพลาดที่เกิดขึ้นจะกระทำในส่วนที่อยู่หลังคำสั่ง else ซึ่งจะเห็นได้ว่า เมื่อประมวลผลและเกิดขึ้นในเงื่อนไขใด จะกระทำในชุดคำสั่งนั้นเพียงอย่างเดียว แต่รูปแบบคำสั่งของ try...finally ชุดคำสั่งที่อยู่หลัง finally จะกระทำเมื่อเสร็จสิ้นการทำงานทุกครั้ง ถึงแม้ว่าจะมีเกิดข้อผิดพลาดหรือไม่เกิดขึ้น ดังรูปแบบการใช้คำสั่ง โปรแกรมตัวอย่างที่ 8.6

ตัวอย่างที่ 8.6 การตรวจจับข้อผิดพลาดโดยคำสั่ง try...finally

บรรทัดที่	คำสั่ง
1	try:
2	val_a=float(input("ตัวแปร a มีค่า = "))
3	val_b=float(input("ตัวแปร b มีค่า = "))
4	val_x=val_a/val_b
5	except :

6	print("มีข้อผิดพลาดเกิดขึ้น")
7	else:
8	print("ผลลัพธ์ที่ได้ = ",val_x)
9	finally:
10	print("สิ้นสุดการดำเนินการ")

ผลลัพธ์ที่ 1

ตัวแปร a มีค่า = 10 (รับข้อมูลจากแผงแป้นอักขระ)
ตัวแปร b มีค่า = 0 (รับข้อมูลจากแผงแป้นอักขระ)
มีข้อผิดพลาดเกิดขึ้น
สิ้นสุดการดำเนินการ

ผลลัพธ์ที่ 2

ตัวแปร a มีค่า = y (รับข้อมูลจากแผงแป้นอักขระ)
มีข้อผิดพลาดเกิดขึ้น
สิ้นสุดการดำเนินการ

ผลลัพธ์ที่ 3

ตัวแปร a มีค่า = 10 (รับข้อมูลจากแผงแป้นอักขระ)
ตัวแปร b มีค่า = 5 (รับข้อมูลจากแผงแป้นอักขระ)
ผลลัพธ์ที่ได้ = 2.0
สิ้นสุดการดำเนินการ

โปรแกรมตัวอย่างที่ 8.6 ผลลัพธ์ที่ได้จากการป้อนข้อมูล พบว่า เมื่อมีข้อผิดพลาดของโปรแกรมเกิดขึ้น โปรแกรมจะประมวลผลในส่วนของ Exception ที่กำหนดและแสดงในส่วนของ finally เช่นกัน ส่วนผลลัพธ์ที่ 3 ข้อมูลที่ได้จากการป้อนข้อมูล ไม่ได้ทำให้เกิดข้อผิดพลาดขึ้นและจะประมวลผลในส่วนของ else พร้อมทั้งแสดงผลลัพธ์ที่ได้จากการประมวลผลหลังคำสั่ง finally ดังนั้นเห็นได้ว่า ชุดคำสั่งใน finally จะประมวลผลทุกครั้งเมื่อเสร็จสิ้นจากการทำงานของโปรแกรม

การเขียนโปรแกรมตรวจจับข้อผิดพลาดของโปรแกรมระหว่าง else และ finally มีลักษณะการทำงานคล้ายกัน นั่นคือ เมื่อไม่เกิดข้อผิดพลาดการทำงานของโปรแกรมจะประมวลผลชุดคำสั่งทั้งสองคำสั่งนี้ ดังนั้นการนำไปใช้ในการเขียนโปรแกรมไม่ควรใช้ควบคู่กันระหว่างสองคำสั่งนี้ โดยเลือกใช้คำสั่งอย่างใดอย่างหนึ่ง เพื่อไม่ให้โปรแกรมมีความซ้ำซ้อนมากเกินไป (สุชาติ คุ้มมณี, 2558)

6. การตรวจจับข้อผิดพลาดอาร์กิวเมนต์ของ exception

การตรวจจับข้อผิดพลาดของโปรแกรมที่เกิดจากการส่งผ่านค่าอาร์กิวเมนต์ในฟังก์ชัน สามารถกำหนด exception ในการตรวจสอบได้ เพื่อตรวจสอบความถูกต้องของข้อมูล ดังรูปแบบการทำงานของโปรแกรมตัวอย่างที่ 8.7

ตัวอย่างที่ 8.7 การตรวจจับข้อผิดพลาดอาร์กิวเมนต์ของ exception

บรรทัดที่	คำสั่ง
1	def cal_area(arg):
2	try:
3	area=22/7*float(arg)
4	return area
5	except ValueError as Args:
6	print ("ค่าอาร์กิวเมนต์ ไม่ถูกต้อง :", Args)
7	val_r=input("ป้อนค่ารัศมี = ")
8	print(cal_area(val_r))
ผลลัพธ์ที่ 1	
ป้อนค่ารัศมี = 5 (รับข้อมูลจากแผงแป้นอักขระ) 15.714285714285714	
ผลลัพธ์ที่ 2	
ป้อนค่ารัศมี = x (รับข้อมูลจากแผงแป้นอักขระ) ค่าอาร์กิวเมนต์ ไม่ถูกต้อง : could not convert string to float: 'x' None	

การตรวจจับข้อผิดพลาดอาร์กิวเมนต์ของ exception เริ่มต้นจากการทำงานของโปรแกรมโดยการรับข้อมูลจากแป้นพิมพ์อักขระ โดยไม่กำหนดลักษณะของข้อมูลที่ป้อน ซึ่งข้อมูลที่ได้จะถูกนำส่งเข้าไปในฟังก์ชัน cal_area() เพื่อใช้ในการคำนวณหาพื้นที่วงกลม ฟังก์ชันมีอาร์กิวเมนต์ arg ในการรับค่าของข้อมูลที่ส่งเข้าไป จะมีการตรวจจับข้อผิดพลาดของข้อมูล โดยถ้าข้อมูลนั้นเป็นตัวเลข โปรแกรมจะทำการประมวลผลและส่งคืนค่ากลับมาแสดงผลลัพธ์ที่ได้ แต่ถ้าหากข้อมูลที่ส่งเข้าไปไม่ใช่ตัวเลข โปรแกรมจะแสดงข้อผิดพลาดของโปรแกรมใน exception ของ ValueError ดังผลลัพธ์ที่ได้ของโปรแกรมตัวอย่างที่ 8.7

7. การตรวจจับข้อผิดพลาดแบบซ้อน

การตรวจจับข้อผิดพลาดนั้น ในบางกรณีที่มีหลาย exception แต่ละ exception อาจมีกลุ่มคำสั่งที่จะต้องประมวลผลและมีโอกาศที่จะเกิดข้อผิดพลาดในการทำงานเกิดขึ้น ดังนั้นจึงสามารถนำรูปแบบของคำสั่งในการตรวจจับข้อผิดพลาด นำไปใช้งานในลักษณะแบบซ้อน ซึ่งสามารถซ้อนกันได้ไม่จำกัด การตรวจจับข้อผิดพลาดแบบนี้ จะทำให้โปรแกรมมีความมั่นคงในการทำงานสูงมากขึ้น มีรูปแบบการทำงานของคำสั่งดังโปรแกรมตัวอย่างที่ 8.8

ตัวอย่างที่ 8.8 การตรวจจับข้อผิดพลาดแบบซ้อน

บรรทัดที่	คำสั่ง
1	`try:`
2	`    fh = open("D:\\python_file\\file_test.txt", "r")`
3	`    try:`
4	`        a=fh.read()`
5	`    except (NameError,FileNotFoundError):`
6	`        print("ไม่สามารถค้นหาแฟ้มข้อมูลหรืออ่านข้อมูลได้")`
7	`    else:`
8	`        print(a)`
9	`finally:`
10	`    print("ปิดแฟ้มข้อมูล")`
11	`    fh.close()`

ผลลัพธ์
Computer Technology ปิดแฟ้มข้อมูล

การตรวจจับข้อผิดพลาดแบบซ้อน มีรูปแบบของคำสั่ง try...finally ที่จะตรวจจับข้อผิดพลาดในการอ่านแฟ้มข้อมูล ประกอบด้วยชุดคำสั่งที่ทำหน้าที่เปิดและอ่านข้อมูลที่อยู่ภายในแฟ้มข้อมูล ซึ่งในการอ่านแฟ้มข้อมูลนั้นอาจมีข้อผิดพลาดของการอ่านเกิดขึ้น ดังนั้นจึงมีการใช้รูปแบบของคำสั่ง try...except...else เพื่อใช้ในการตรวจสอบการอ่านข้อมูลอีกครั้ง ซึ่งเป็นลักษณะของการตรวจจับข้อผิดพลาดที่เกิดซ้อนกันเป็นชั้น ดังนั้นโปรแกรมที่มีขนาดใหญ่และความซับซ้อนนสูง จะมีการนำรูปแบบของการตรวจสอบแบบซ้อนนี้ไปใช้ เพื่อให้เกิดความคงทนต่อการทำงานของโปรแกรม ดังผลลัพธ์ที่ได้ของโปรแกรมตัวอย่างที่ 8.8

8. การตรวจจับข้อผิดพลาดด้วยวิธี Raising

Raising คือ กระบวนการในการตรวจจับข้อผิดพลาดที่แสดงข้อผิดพลาดหรือแจ้งเตือนจากการกำหนดของผู้เขียนโปรแกรมด้วยตนเอง สามารถแบ่งออกเป็น 2 ส่วน ได้แก่ ส่วนที่เป็น Exception Object ที่สร้างขึ้นเพื่ออ้างอิงกับข้อผิดพลาดที่มีในภาษาไพธอนและการโยนข้อผิดพลาดที่เกิดขึ้นไปยังส่วนของโปรแกรมที่มีการเรียกใช้งาน (สุชาติ คุ้มมณี, 2558) มีรูปแบบของการใช้คำสั่งดังต่อไปนี้

```
raise [Exception [, อาร์กิวเมนต์ [, ตรวจสอบย้อนกลับ] ] ]
```

จากรูปแบบการตรวจจับข้อผิดพลาดของโปรแกรมด้วยวิธี Raising มีรูปแบบและการใช้คำสั่งดังโปรแกรมตัวอย่างที่ 8.9

ตัวอย่างที่ 8.9 การตรวจจับข้อผิดพลาดด้วยคำสั่ง raise

บรรทัดที่	คำสั่ง
1	def chk_divisor(arg):
2	if arg == 0:
3	raise("ตัวเลขไม่ถูกต้อง")
4	print("การประมวลผลไม่เสร็จสมบูรณ์")
5	print("ตัวหารถูกต้อง")
6	chk_divisor(0)
ผลลัพธ์	
Traceback (most recent call last): File "D:\python_file\test.py", line 6, in <module> chk_divisor(0) File "D:\python_file\test.py", line 3, in chk_divisor raise("ตัวเลขไม่ถูกต้อง") TypeError: exceptions must derive from BaseException	

การตรวจจับข้อผิดพลาดด้วยวิธี Raising มีการตรวจจับข้อมูลที่ส่งเข้าไปในฟังก์ชัน chk_divisor() ด้วยคำสั่ง raise ซึ่งจะตรวจสอบข้อมูลถ้ามีค่าเท่ากับ 0 จะแสดงข้อความ “ตัวเลขไม่

ถูกต้อง” จะหยุดการทำงานทันทีโดยไม่แสดงชุดคำสั่งในบรรทัดที่ 4 ต่อ พร้อมกับอ้างอิงข้อผิดพลาดไปยังภาษาไพธอน ซึ่งแสดงข้อความ TypeError: exceptions must derive from BaseException เพราะในส่วนของ raise ไม่ได้กำหนด exception ที่เกิดขึ้น จึงเกิดผลลัพธ์ดังโปรแกรมตัวอย่างที่ 8.9

การใช้รูปแบบ Raising จากโปรแกรมดังโปรแกรมตัวอย่างที่ 8.9 ไม่ได้ระบุ exception ที่เกิดขึ้น ดังนั้น ข้อความการแจ้งข้อผิดพลาดจะเกิดขึ้นจากภาษาโปรแกรม หากผู้เขียนโปรแกรมต้องการระบุข้อผิดพลาดและแสดงการแจ้งเตือนที่เกิดขึ้นเอง สามารถกำหนด exception ให้กับ raise ได้ โดยมีรูปแบบการทำงานของโปรแกรมดังตัวอย่างที่ 8.10

ตัวอย่างที่ 8.10 การตรวจจับข้อผิดพลาดด้วยวิธี Raising โดยระบุ exception

บรรทัดที่	คำสั่ง
1	try:
2	obj_file = open("D:\\python_file\\file_test.txt")
3	obj_file.write("Computer Technology")
4	except(IOError, ValueError):
5	print("มีข้อผิดพลาดเกิดขึ้น")
6	raise OSError("หยุดการประมวลผล")
7	else:
8	obj_file.close()
9	print("แฟ้มข้อมูลถูกปิด")

ผลลัพธ์

```
มีข้อผิดพลาดเกิดขึ้น
Traceback (most recent call last):
  File "D:\python_file\test.py", line 3, in <module>
    obj_file.write("Computer Technology")
io.UnsupportedOperation: not writable

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "D:\python_file\test.py", line 6, in <module>
    raise OSError("หยุดการประมวลผล")
OSError: หยุดการประมวลผล
```

โปรแกรมตัวอย่างที่ 8.10 มีการใช้คำสั่ง raise ในรูปแบบของคำสั่ง try...except...else เพื่อใช้ในการตรวจจับข้อผิดพลาดที่เกิดขึ้นจากการเปิดและอ่านแฟ้มข้อมูล โดยกำหนด exception ตรวจสอบ IOError และ ValueError หากตรวจจับและมีข้อผิดพลาดเกิดขึ้น จะทำการแสดงข้อความ “มีข้อผิดพลาดเกิดขึ้น” จากนั้นใช้รูปแบบคำสั่ง raise ตรวจจับข้อผิดพลาด OSError ที่เกิดขึ้น หากพบข้อผิดพลาด จะทำการแสดงข้อความ “OSError: หยุดการประมวลผล” ซึ่งข้อความแสดงนั้นถูกกำหนดขึ้นโดยผู้เขียนโปรแกรม จะมีความแตกต่างถ้าหากไม่ได้กำหนดข้อผิดพลาดที่เกิดขึ้น ซึ่งการแสดงข้อผิดพลาดจะเกิดจากภาษาโปรแกรม

9. การตรวจจับข้อผิดพลาดสร้างขึ้นใหม่จากผู้ใช้งาน

การตรวจจับข้อผิดพลาด ภาษาไพธอนได้เตรียมเครื่องมือให้ผู้เขียนโปรแกรมได้เลือกใช้เป็นจำนวน แต่บางกรณีต้องการสร้าง exception ขึ้นมาใช้งานเฉพาะผู้เขียนโปรแกรม ภาษาไพธอนได้อนุญาตให้สามารถกำหนดขึ้นมาใช้งานเองได้ (Halterman, 2016) แต่จะต้องอยู่ภายใต้ exception ของภาษาโปรแกรม ลักษณะการนำไปใช้จะอยู่ในรูปแบบของการสืบทอดคลาส (สุชาติ คุ้มมณี, 2558) การสร้าง Exception ใช้งานเฉพาะผู้เขียนโปรแกรม มีรูปแบบการทำงานดังโปรแกรมตัวอย่างที่ 8.11

ตัวอย่างที่ 8.11 Exception สร้างขึ้นใหม่จากผู้ใช้งาน

บรรทัดที่	คำสั่ง
1	class Error(Exception):
2	pass
3	class V_SmallError(Error):
4	pass
5	class V_LargeError(Error):
6	pass
7	print("----- เกมทายตัวเลข -----")
8	number = 10
9	while True:
10	try:
11	i_num = int(input("ป้อนตัวเลข : "))
12	if i_num < number:
13	raise V_SmallError

```
14          elif i_num > number:
15              raise V_LargeError
16          break
17      except V_SmallError:
18          print("เป็นตัวเลขที่มีค่าต่ำกว่า")
19      except V_LargeError:
20          print("เป็นตัวเลขที่มีค่าสูงกว่า")
21  print("ทายตัวเลขได้ถูกต้อง")
```

ผลลัพธ์

```
----- เกมทายตัวเลข -----
ป้อนตัวเลข : 1                          (รับข้อมูลจากแผงแป้นอักขระ)
เป็นตัวเลขที่มีค่าต่ำกว่า
ป้อนตัวเลข : 5                          (รับข้อมูลจากแผงแป้นอักขระ)
เป็นตัวเลขที่มีค่าต่ำกว่า
ป้อนตัวเลข : 56                         (รับข้อมูลจากแผงแป้นอักขระ)
เป็นตัวเลขที่มีค่าสูงกว่า
ป้อนตัวเลข : 10                         (รับข้อมูลจากแผงแป้นอักขระ)
ทายตัวเลขได้ถูกต้อง
```

โปรแกรมตัวอย่างที่ 8.11 คือ โปรแกรมเกมทายตัวเลข การทำงานประกอบไปด้วย exception 2 ชนิด คือ V_SmallError และ V_LargeError ซึ่ง V_SmallError จะถูกเรียกใช้เมื่อมีตรวจสอบเงื่อนไขพบว่า ตัวเลขที่ป้อนมีค่าของข้อมูลน้อยกว่า 10 และใช้คำสั่ง raise ทำหน้าที่ในการส่งข้อมูลที่ได้ไปยัง exception ของ V_SmallError ซึ่ง V_SmallError เป็นคลาส exception ที่ถูกสร้างขึ้นมาใช้งานใหม่อยู่ภายใต้ Exception คลาสหลัก Error ถือว่า V_SmallError เป็น exception ใหม่ของโปรแกรม ภายในคลาสมีคำสั่ง pass ซึ่งเมื่อประมวลผลจะไม่มีผลลัพธ์เกิดขึ้น จะใช้ในกรณีขั้นสุดท้ายของคลาสหรือฟังก์ชัน เพื่อให้สิ้นสุดและข้ามไปทำงานในขั้นตอนต่อไป ซึ่งคำสั่งภายใน V_SmallError จะให้แสดงข้อความ “เป็นตัวเลขที่มีค่าต่ำกว่า”

การตรวจจับข้อผิดพลาดของ V_LargeError จะมีลักษณะการทำงานเช่นเดียวกันกับ V_SmallError โดยจะทำการตรวจสอบในกรณีค่าของข้อมูลมีขนาดมากกว่า 10 ภายใต้โครงสร้างการทำงานแบบวนซ้ำ คำสั่ง while จะทำการตรวจสอบจนกระทั่งเงื่อนไขนั้นเป็นจริง จึงจะให้หยุดการทำงาน ซึ่งหมายถึงในกรณีที่ป้อนตัวเลขเท่ากับ 10 และภายในโครงสร้างแบบวนซ้ำจะมีโครงสร้างทำงานแบบมีเงื่อนไขคำสั่ง if ในการตรวจสอบที่ค่าข้อมูลมีค่าน้อยกว่าหรือมากกว่า ผ่านการตรวจจับ

ข้อผิดพลาดที่สร้างขึ้นมาใหม่ V_SmallError และ V_LargeError จะแสดงรายละเอียดการทำงานดังผลลัพธ์ที่ได้ของโปรแกรมตัวอย่างที่ 8.11

10. การทดสอบสมมติฐาน

การทดสอบสมมติฐาน (Assertions) คือ การทดสอบและตรวจสอบข้อผิดพลาด เพื่อเป็นการยืนยันความถูกต้องเกี่ยวกับการทำงานของโปรแกรมที่คาดว่าไม่จะเกิดข้อผิดพลาดขึ้น การทดสอบสมมติฐานเป็นการกรองความถูกต้องในการทำงานของโปรแกรมอีกชั้น หากมีข้อผิดพลาดที่เกิดขึ้นจะต้องดำเนินการแก้ให้ถูกต้อง (สุชาติ คุ้มมณี, 2558) ซึ่งมีรูปแบบคำสั่งดังนี้

```
assert Expression[,อาร์กิวเมนต์]
```

รูปแบบการทดสอบสมมติฐานคำสั่ง assert สามารถนำไปในการตรวจจับข้อผิดพลาดของโปรแกรมได้ ดังโปรแกรมตัวอย่างที่ 8.12

ตัวอย่างที่ 8.12 การทดสอบสมมติฐานด้วยคำสั่ง assert

บรรทัดที่	คำสั่ง
1	def BMI(arg1, arg2):
2	assert (arg2 != 0), "หยุดการประมวลผลเนื่องจากมีตัวหารเป็น 0 "
3	bmi_val=float(arg1)/(float(arg2)**2)
4	return bmi_val
5	a=input("ป้อนน้ำหนัก(kg) = ")
6	b=input("ป้อนความสูง(m) = ")
7	print (BMI(a, b))
ผลลัพธ์ที่ 1	
ป้อนน้ำหนัก(kg) = 73	(รับข้อมูลจากแผงแป้นอักขระ)
ป้อนความสูง(m) = 1.65	(รับข้อมูลจากแผงแป้นอักขระ)
26.813590449954088	
ผลลัพธ์ที่ 2	
ป้อนน้ำหนัก(kg) = 73	(รับข้อมูลจากแผงแป้นอักขระ)
ป้อนความสูง(m) = 0	(รับข้อมูลจากแผงแป้นอักขระ)

```
Traceback (most recent call last):
  File "D:\python_file\test.py", line 7, in <module>
    print (BMI(a, b))
  File "D:\python_file\test.py", line 3, in BMI
    bmi_val=float(arg1)/(float(arg2)**2)
ZeroDivisionError: float division by zero
```

การตรวจสอบสมมติฐานในโปรแกรมตัวอย่างที่ 8.12 จะทำการตรวจสอบข้อมูลที่รับเข้ามาจากแป้นพิมพ์อักขระและนำข้อมูลที่ได้ส่งไปยังฟังก์ชัน BMI() ซึ่งประกอบไปด้วยอาร์กิวเมนต์ arg1 และ arg2 ที่จะรับข้อมูลไปทำการประมวลผลในฟังก์ชัน โดยเงื่อนข้อมูลที่จะสามารถประมวลผลได้ คือ arg2 จะต้องไม่เท่ากับ 0 ดังนั้น จึงต้องทำการใช้คำสั่ง assert เพื่อใช้ในการตรวจจับข้อผิดพลาดที่เกิดขึ้นในส่วนแรกของฟังก์ชัน โดยกำหนดเงื่อนไข ถ้า arg2 ไม่มีค่าเท่ากับ 0 (arg2!=0) ก็จะสามารถประมวลผลคำสั่งต่อไปได้ (เงื่อนไขที่เป็นจริง) แต่ถ้าไม่ใช่ โปรแกรมจะหยุดการทำงานและแจ้งข้อผิดพลาดที่เกิดขึ้นตามที่กำหนดไว้ในคำสั่ง เพื่อให้ทำการแก้ไขคำสั่งให้ถูกต้อง ดังผลลัพธ์ของโปรแกรมที่ได้

ข้อดีสำหรับการใช้คำสั่ง assert คือ สามารถตรวจจับข้อผิดพลาดของโปรแกรมในลักษณะของการกรองข้อผิดพลาดที่อาจเกิดขึ้นในโปรแกรม การตรวจสอบข้อผิดพลาดที่เกิดขึ้นจะทำให้โปรแกรมมีการทำงานผิดพลาดน้อยลงและง่ายกว่าการคำสั่ง try...exception สามารถปิดการตรวจจับข้อผิดพลาดคำสั่ง assert ในขณะที่โปรแกรมกำลังทำงานได้ โดยไม่ส่งผลต่อประสิทธิภาพการทำงานของโปรแกรม

สรุป

ประสิทธิภาพการทำงานของโปรแกรม เกิดจากความมั่นคงหรือความคงทนต่อการทำงาน หมายถึง ช่วงระหว่างการประมวลแต่ละส่วนการทำงานของโปรแกรม จะต้องไม่เกิดข้อผิดพลาดหรือโปรแกรมหยุดระหว่างการทำงาน ส่งผลต่อเสียของการทำงาน การเกิดข้อผิดพลาดการทำงานของโปรแกรม สามารถแบ่งออกเป็น 3 ประเภท ได้แก่ ข้อผิดพลาดที่ไม่ปฏิบัติตามรูปแบบของภาษาโปรแกรม ข้อผิดพลาดที่เกิดจากการประมวลผลคำสั่งและข้อผิดพลาดที่เกิดจากผลลัพธ์ไม่ถูกต้อง ดังนั้น จึงจะต้องมีกระบวนการในการจัดการข้อผิดพลาดที่เกิดขึ้นเหล่านี้ ซึ่งข้อผิดพลาดที่ไม่ปฏิบัติตามรูปแบบของภาษาโปรแกรม ข้อผิดพลาดที่เกิดจากการประมวลผลคำสั่ง จะเกี่ยวข้องกับภาษาโปรแกรมโดยตรง จึงสามารถใช้คำสั่งในการตรวจจับข้อผิดพลาดที่เกิดขึ้นได้ โดยคำสั่งที่ใช้ประกอบด้วย try...except...else ใช้ในการตรวจจับข้อผิดพลาดที่สามารถระบุเหตุการณ์ที่จะเกิด

ข้อผิดพลาดได้ ซึ่ง exception ระบุข้อผิดพลาดนเดียวหรือเป็นชุด สามารถกำหนดได้หรือไม่ระบุข้อผิดพลาดที่เกิดขึ้น จะใช้ในกรณีที่ไม่ทราบข้อผิดพลาดที่เกิดขึ้นได้ ผลลัพธ์ที่จะเกิดขึ้นจะต้องอยู่ภายใต้เงื่อนไขที่กำหนด แต่จะมีคำสั่ง finally ที่ใช้งานร่วมกับ try...except...else ที่จะประมวลผลชุดคำสั่งที่อยู่ภายใน finally ทุกครั้ง คำสั่งนี้อยู่ในส่วนสุดท้ายของโปรแกรม การทำงานของโปรแกรมที่มีขนาดใหญ่ จะมีส่วนของโปรแกรมที่สามารถเกิดข้อผิดพลาดได้หลายส่วน ดังนั้นสามารถนำรูปแบบคำสั่ง ไปใช้ในการตรวจจับข้อผิดพลาดแต่ละส่วนและแต่ละส่วนจะมีการใช้การตรวจจับที่ซ้อนกันอยู่ภายในอีกครั้ง ซึ่งเป็นลักษณะแบบตรวจจับข้อผิดพลาดแบบซ้อน พร้อมทั้งยังสามารถทำงานในลักษณะของอาร์กิวเมนต์ได้ โดยจะตรวจสอบค่าข้อมูลที่จะมีผลต่อการทำงานของโปรแกรม นอกเหนือจากนี้ข้อผิดพลาดที่เกิดขึ้น สร้างการแจ้งเตือนด้วยผู้เขียนโปรแกรมด้วยคำสั่ง Raising และสร้าง exception ขึ้นมาใช้งานเองได้ ซึ่งอยู่ภายใต้ exception หลักที่มีอยู่ในโปรแกรม การทำงานของโปรแกรมเป็นขบวนการที่ผู้เขียนโปรแกรมได้ออกแบบขั้นตอนการทำงานตามสมมติฐานที่ตั้งไว้ ดังนั้นเพื่อเป็นการตรวจสอบสมมติฐานมีโอกาสผิดพลาดหรือไม่ เพื่อยืนยันสมมติฐานที่ตั้งไว้ สามารถใช้กระบวนการ Assertions ในการตรวจสอบข้อผิดพลาด ซึ่งเครื่องมือเหล่านี้จะเป็นส่วนที่จะทำให้เกิดประสิทธิภาพของโปรแกรมเพิ่มมากขึ้น

แบบฝึกหัด

1. ให้ผู้เรียนอธิบายความสำคัญของนำการจัดการข้อผิดพลาดของโปรแกรม เพื่อให้เกิดประสิทธิภาพการทำงานของโปรแกรม

2. ให้ผู้เรียนอธิบายข้อผิดพลาดที่เกิดขึ้นในโปรแกรมมีอะไรบ้าง และแต่ละข้อผิดพลาดเกิดขึ้นได้อย่างไร

3. ให้ผู้เรียนอธิบายการจัดการข้อผิดพลาด try...except...else มีหลักการอย่างไร พร้อมทั้งเขียนโปรแกรมตัวอย่างประกอบการอธิบาย

4. ให้ผู้เรียนอธิบายความแตกต่างระหว่างการกำหนด exception และไม่กำหนด exception

5. ให้ผู้เรียนอธิบายการตรวจสอบข้อผิดพลาดแบบกำหนดเป็นชุดมีลักษณะและประโยชน์อย่างไร พร้อมทั้งเขียนโปรแกรมประกอบการอธิบาย

6. ให้ผู้เรียนอธิบายการตรวจสอบข้อผิดพลาดหลาย Exception กับแบบเป็นชุด มีความแตกต่างกันอย่างไร พร้อมทั้งเขียนโปรแกรมประกอบการอธิบาย

7. ให้ผู้เรียนอธิบายการตรวจจับข้อผิดพลาดโดยใช้ try...finally มีลักษณะอย่างไร พร้อมทั้งเขียนโปรแกรมตัวอย่างประกอบการอธิบาย

8. ให้ผู้เรียนอธิบายหน้าและประโยชน์ของการตรวจจับข้อผิดพลาดอาร์กิวเมนต์ พร้อมทั้งเขียนโปรแกรมประกอบการอธิบาย

9. ให้ผู้เรียนอธิบายลักษณะการตรวจจัดข้อผิดพลาดแบบซ้อน พร้อมทั้งเขียนโปรแกรมประกอบการอธิบาย

10. ให้ผู้เรียนอธิบายการตรวจจับข้อผิดพลาดแบบ Raising มีวิธีการอย่างไร เขียนโปรแกรมประกอบการอธิบาย

11. ให้ผู้เรียนอธิบายวิธีการตรวจจับข้อผิดพลาดที่สร้างขึ้นมาใช้งานเอง มีวิธีการอย่างไร เขียนโปรแกรมประกอบการอธิบาย

12. ให้ผู้เรียนอธิบายการตรวจสอบสมมติฐานคืออะไร และมีวิธีการอย่างไร

เอกสารอ้างอิง

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

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

Gerrard, Paul. (2016). **Lean PythonLearn: Just Enough Python to Build Useful Tools**. Maidenhead: Apress.

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