บทที่ 9

การจัดการข้อผิดพลาด

จากหลาย ๆ บทที่ผ่านมาผู้เขียนเข้าใจว่าผู้อ่านหลาย ๆ ท่านคงได้เจอกับปัญหาการแจ้งเตือนข้อผิดพลาด (Error) จากการเขียนคำสั่งโปรแกรมที่ไม่ถูกต้องไปบ้างแล้ว เช่น ใส่ค่าตัวแปรที่ยังไม่ประกาศ ไม่ใส่เครื่องหมาย Colon **(:)** การป้อนชนิดข้อมูลผิดประเภท เป็นต้น ข้อผิดพลาดต่าง ๆ เหล่านี้สามารถเกิดขึ้นได้กับทุก ๆ คน แม้แต่โปรแกรมเมอร์ที่มีความเชี่ยวชาญแล้วก็ตาม ในบทนี้เราจะมาศึกษาถึงวิธีการจัดการข้อผิดพลาด (Exceptions Handling) ซึ่งข้อผิดพลาดที่เกิดขึ้นในการเขียนโปรแกรมแบ่งออกได้เป็น 3 ประเภทใหญ่ ๆ ได้แก่

1. Syntax Error คือ การเขียนคำสั่งโปรแกรมผิดหลักไวยากรณ์ที่กำหนดไว้ เช่น การลืมใส่เครื่องหมายวงเล็บ การลืมใส่เครื่องหมาย Colon **(:)** การนำคำสงวนมาตั้งชื่อเป็นตัวแปร การเรียกใช้งานไลบรารีผิด เป็นต้น ข้อผิดพลาดลักษณะนี้จะเกิดขึ้นตอนที่เราสั่งให้โปรแกรมประมวลผล ส่งผลทำให้โปรแกรมไม่สามารถทำงานได้ แสดงดังตัวอย่างต่อไปนี้

2. Runtime Error คือ การเขียนคำสั่งโปรแกรมที่ไม่ถูกต้องโดยผู้เขียนโปรแกรมเอง เช่น การกำหนดให้กรอกข้อมูล แต่ผู้ใช้งานกรอกชนิดข้อมูลผิดประเภทตามที่ประกาศไว้ การกำหนดเลขศูนย์ให้เป็นตัวหารประกาศตัวแปรใช้งานผิดประเภท เป็นต้น หรือเกิดจากทรัพยากรของเครื่องคอมพิวเตอร์เอง เช่น หน่วยจัดเก็บข้อมูลเต็ม หน่วยความจำไม่เพียงพอ ไม่สามารถเชื่อมต่อกับฐานข้อมูลได้ เป็นต้น

3. Logic Error คือ การทำงานของโปรแกรมผิดพลาดที่เกิดมาจากตัวผู้เขียนโปรแกรมเอง ซึ่งอาจจะมีสาเหตุมาจากความเข้าใจผิด จากกระบวนการทำงานของโปรแกรมหรือสูตรการคำนวณ เป็นสาเหตุให้การคำนวณของโปรแกรมผิดพลาด และส่งผลให้ผลลัพธ์ที่ได้ออกมานั้นไม่ถูกต้องตามต้องการ การเกิดข้อผิดพลาดลักษณะนี้ผู้เขียนโปรแกรมต้องระมัดระวังเป็นพิเศษ เนื่องจากจะไม่มีการแจ้งเตือนให้เห็นแต่ต้องสังเกตเอาเอง และยังเป็นข้อผิดพลาดที่หายากมากที่สุดถ้าโปรแกรมมีขนาดใหญ่

ตัวอย่าง 9.1
การแจ้งเตือนข้อผิดพลาดเมื่อผู้เขียนโปรแกรมไม่ใส่เครื่องหมาย Colon (:) ปิดท้ายคำสั่ง `for`

```
1 for i in range(1, 10)
2     print(f'รอบที่ {i}')
```

```
  File 'src/test.py', line 1
    for i in range(1, 10)
                         ^
SyntaxError: invalid syntax
```

ตัวอย่าง 9.2
ลักษณะการแจ้งเตือนข้อผิดพลาดเมื่อกรอกข้อมูลไม่ถูกต้อง

```
1 n = int(input('กรุณาป้อนตัวเลข 1-5 : '))
2 for i in range(1, n):
3     print(f'รอบที่ {i} ', end = ' ')
```

```
กรุณาป้อนตัวเลข 1-5 : 2.5
Traceback (most recent call last):
  File 'src/test.py', line 1, in <module>
    n = int(input('กรุณาป้อนตัวเลข 1-5 : '))
ValueError: invalid literal for int() with base 10: '2.5'
```

ตัวอย่าง 9.3
ลักษณะการทำงานของโปรแกรมที่ผิดพลาดจาก Logic

```
1 b = float(input('ป้อนความยาวฐาน = '))
2 h = float(input('ป้อนความสูง = '))
3 area = 0.5 * b + h
4 print('พื้นที่สามเหลี่ยม = ', area)
```

```
ป้อนความยาวฐาน = 4
ป้อนความสูง = 6
พื้นที่สามเหลี่ยม = 8.0
```

จากตัวอย่างที่ 7.3 เป็นโปรแกรมคำนวณหาพื้นที่สามเหลี่ยม โดยให้ผู้ใช้งานป้อนความยาวฐานและความสูงผ่านทางคีย์บอร์ดด้วยฟังก์ชัน `input()` ในบรรทัดที่ 1 และ 2 ส่วนบรรทัดที่ 3 เป็นการคำนวณหาพื้นที่สามเหลี่ยม ผู้อ่านจะสังเกตเห็นว่าสูตรคำนวณพื้นที่สามเหลี่ยมผิด ซึ่งอาจจะเกิดได้จากหลายสาเหตุ เช่น กดเครื่องหมายคูณเป็นเครื่องหมายบวก หรือเข้าใจสูตรคำนวณผิด จึงส่งผลให้ผลการคำนวณออกมาไม่ถูกต้อง ดังนั้น Logic Error จึงเป็นส่วนที่ต้องระมัดระวัง

การจัดการกับข้อผิดพลาดเป็นสิ่งที่สำคัญมาก เนื่องจากช่วยให้เราหาจุดแก้ไขและสาเหตุที่ทำให้เกิดข้อผิดพลาดที่เกิดขึ้นได้ง่าย ในบทนี้ผู้อ่านจะได้เรียนรู้วิธีการจัดการกับข้อผิดพลาดโดยการนำเอาประเภทของ Exception ต่าง ๆ ที่ภาษาไพธอนได้จัดเตรียมไว้ให้ นำมาตรวจสอบข้อผิดพลาดที่อาจจะเกิดขึ้นในโปรแกรมที่เราได้พัฒนา

9.1 ประเภทของ Exception ในภาษาไพธอน

เมื่อผู้อ่านกำลังพัฒนาโปรแกรมอยู่แล้วทำการทดสอบ ผลปรากฏว่ามีเหตุการณ์ข้อผิดพลาดเกิดขึ้น ซึ่งผลกระทบต่อการทำงานโปรแกรมหรือทำให้โปรแกรมไม่สามารถทำงานต่อไปได้ และมีข้อความแจ้งเตือนบอกถึงสาเหตุที่เกิดข้อผิดพลาดขึ้น จากตัวอย่างที่ 7.1 และ 7.2 ข้อความแจ้งเตือนประกอบด้วย 2 ส่วน โดยส่วนที่ 1 คือ ประเภท Exception ที่เกิดขึ้น และแสดงบรรทัดที่เกิด Exception และส่วนที่ 2 แสดงด้วยชื่อ Exception และหลัง

เครื่องหมาย Colon **(:)** เป็นส่วนคำอธิบายถึงสาเหตุที่เกิดข้อผิดพลาดขึ้น ในส่วนนี้จะนำเสนอประเภท Exception ที่ผู้อ่านอาจจะพบอยู่เสมอ ในขณะที่กำลังทดสอบโปรแกรมและนำเอามาใช้งานจัดการกับข้อผิดพลาด หากผู้อ่านต้องการข้อมูลเพิ่มสามารถเข้าไปศึกษาได้ที่ `https://docs.python.org/3/library/exceptions.html`

Exception	ความหมาย
ArithmeticError	เกิดข้อผิดพลาดเกี่ยวกับการคำนวณทางคณิตศาสตร์ทั้งหมด
AssertionError	เกิดจากการใช้คำสั่ง essert
AttributeError	กำหนดหรือการอ้างถึงแอดทริบิวต์ไม่ถูกต้อง
ConnectionError	ไม่สามารถเชื่อมต่อได้
EOFError	เกิดขึ้นเมื่ออ่านข้อมูลถึงจุดสุดท้าย
FileExistsError	ตั้งชื่อสร้างไฟล์หรือไดเรกทอรีซ้ำ
FileNotFoundError	ค้นหาไฟล์หรือไดเรกทอรีไม่พบ
FloatingPointError	การดำเนินการเกี่ยวกับชนิดข้อมูลเลขทศนิยมผิดพลาด
ImportError	เรียกใช้งานไลบารีผิด
IndentationError	เกิดข้อผิดพลาดจากการย่อหน้า
IndexError	ไม่พบตำแหน่งที่ระบุ
ModuleNotFoundError	ไม่พบโมดูลที่ระบุ
KeyError	ไม่พบคีย์ที่ระบุ
KeyboardInterrupt	เกิดการขัดจังหวะการทำงานด้วยปุ่ม Ctrl+C หรือ Delete
MemoryError	หน่วยความจำไม่เพียงพอ
NameError	ไม่พบชื่อในตัวแปรทั้งแบบ local และ global
OverflowError	ผลการคำนวณเกินค่าที่ได้กำหนดไว้
OSError	เกิดจากปัญหาของระบบปฏิบัติการ
PermissionError	ความผิดพลาดที่ไม่มีสิทธิ์เข้าใช้งาน
RuntimeError	ข้อผิดพลาดขณะโปรแกรมกำลังทำ งาน
SyntaxError	เขียนคำสั่งโปรแกรมผิดไวยากรณ์
SystemError	ความผิดพลาดที่เกิดจากระบบ
TabError	ปัญหาการใช้คีย์ tab และเคาะวรรค (space)
TypeError	ระบุชนิดข้อมูลไม่ถูกต้อง
UnboundLocalError	เรียกใช้งานตัวแปรแบบ local ที่ยังไม่ได้กำหนดค่า
ValueError	กำหนดค่าไม่เหมาะสมกับชนิดข้อมูล
ZeroDivisionError	ความผิดพลาดจากการนำเลข 0 มาหาร

ตาราง 9.1: ประเภทของ Exceptions ในภาษาไพธอน

9.2 การตรวจจับข้อผิดพลาดประเภท Exception

การเกิดข้อผิดพลาดจากโปรแกรมหรือจากการทำงานของระบบ ทำให้โปรแกรมไม่สามารถทำงานต่อไปได้ ซึ่งเราจะต้องเขียนคำสั่งจัดการกับเหตุการณ์ที่เกิดขึ้นไว้ด้วย สำหรับคำสั่งที่ช่วยให้เราสามารถตรวจจับของชนิดข้อผิดพลาดและทราบถึงสาเหตุที่เกิดขึ้นในภาษาไพธอน ได้แก่ คำสั่ง `try...except` ซึ่งมีรูปแบบการใช้งานดังต่อไปนี้

9.2.1 การใช้คำสั่ง `try...except` ตรวจจับประเภท Exception

คำสั่ง `try...except` ใช้ตรวจจับข้อผิดพลาดประเภท Exception ที่อาจจะเกิดขึ้นขณะโปรแกรมกำลังทำงาน โดยคำสั่งที่อยู่ภายในขอบเขตคำสั่ง try เป็นคำสั่งที่ต้องการให้ตรวจจับความผิดพลาด และคำสั่งที่อยู่ในขอบเขตคำสั่ง except เป็นคำสั่งที่ต้องการให้ทำงานเมื่อมีข้อผิดพลาดเกิดขึ้น มีโครงสร้างการใช้คำสั่งดังนี้

รูปแบบการเขียนคำสั่ง `try...except`

```
try:
    statements       # คำสั่งที่ต้องการตรวจจับความผิดพลาด
except Exception_1: # ประเภทของความผิดพลาด
    statement_exc_1 # คำสั่งที่ให้ทำงานเมื่อตรวจพบข้อผิด
    ↪ พลาด
    ...
except Exception_n: # ประเภทของความผิดพลาดอื่น
    statement_exc_n # คำสั่งที่ให้ทำงานเมื่อตรวจพบข้อผิด
    ↪ พลาด
```

เราสามารถกำหนดให้มีการตรวจจับคำสั่งที่คาดว่าจะเกิดข้อผิดพลาดภายใต้คำสั่ง try ได้หลายคำสั่ง และใช้คำสั่ง except ตรวจจับประเภท Exception ได้หลายตัว

ตัวอย่าง 9.4
การตรวจจับข้อผิดพลาดเมื่อผู้ใช้งานป้อนข้อมูลไม่ถูกต้อง และหารด้วยเลข 0

```
1  x =5
2  try:
3      n = int(input('กรุณาป้อนตัวเลข: '))
4      z = x / n
5      print(z)
6  except ZeroDivisionError: # ตรวจจับข้อผิดพลาดเมื่อตัวหารเป็นเลข 0
7      print('ไม่สามารถหารด้วย 0 ได้')
8  except ValueError: # ตรวจจับข้อผิดพลาดเมื่อป้อนข้อมูลไม่ถูกต้อง
9      print('ข้อมูลที่คุณป้อนไม่ใช่ตัวเลขจำนวนเต็ม')
```

```
กรุณาป้อนตัวเลข: 0
ไม่สามารถหารด้วย 0 ได้

กรุณาป้อนตัวเลข : 5.5
ข้อมูลที่คุณป้อนไม่ใช่ตัวเลขจำนวนเต็ม
```

บรรทัดที่ 3-5 คือ คำสั่งที่ต้องการตรวจสอบข้อผิดพลาด บรรทัดที่ 6 คำสั่งตรวจจับข้อผิดพลาด เมื่อผู้ใช้งานป้อนข้อมูลเป็นเลข 0 ทำให้บรรทัดที่ 7 แสดงผล บรรทัดที่ 8 คำสั่งตรวจจับข้อผิดพลาด เมื่อผู้ใช้งานป้อนข้อมูลไม่ใช่ตัวเลขจำนวนเต็ม ทำให้บรรทัดที่ 9 แสดงผล

9.2.2 การใช้คำสั่ง `try...except` ตรวจจับ Exception หลายตัว

เราสามารถกำหนดประเภท Exception ให้ตรวจจับความผิดพลาดได้หลายประเภท โดยกำหนดไว้ในส่วนของคำสั่ง except มีโครงสร้างการเขียนคำสั่งดังนี้

รูปแบบการเขียนคำสั่ง `try...except` ตรวจจับ Exceptions หลายตัว

```
try:
    statements
except (Exception_1, Exception_2,...,Exception_n):
    statement_exc
```

ตัวอย่าง 9.5
การกำหนดให้คำสั่ง `except` ตรวจจับข้อผิดพลาดประเภท Exceptions แบบหลายตัว

```
1 lst = [5, 6, 8, 9, 10, 15]
2 dct = {1:'Tennis', 2:'Football', 3:'Racing', 4:'Running'}
3 try:
4     print('ตำแหน่งที่ 5 ของ lst =', lst[5])
5     print('ตำแหน่งที่ 2 ของ dct =', dct[2]) # ใส่คีย์ถูกต้อง
6 except(IndexError, KeyError): # ตัวจับแบบหลายตัว
7     print('ตำแหน่ง หรือ คีย์ที่ระบุไม่ถูกต้อง')
```

```
ตำแหน่งที่ 5 ของ 1st = 15
ตำแหน่งที่ 2 ของ dct = Football
```

ตัวอย่าง 9.6

```
lst = [5, 6, 8, 9, 10, 15]
dct = {1:'Tennis', 2:'Football', 3:'Racing', 4:'Running'}
try:
    print('ตำแหน่งที่ 5 ของ lst =', lst[5])
    print('ตำแหน่งที่ 2 ของ dct =', dct[2]) # ใส่คีย์ถูกต้อง
except(IndexError, KeyError): # ตัวจับแบบหลายตัว
    print('ตำแหน่ง หรือ คีย์ที่ระบุไม่ถูกต้อง')
```

```
ตำแหน่งที่ 5 ของ lst = 15
ตำแหน่ง หรือ คีย์ที่ระบุไม่ถูกต้อง
```

9.2.3 การใช้คำสั่ง `try...except...else` ตรวจจับประเภท Exception

เมื่อตรวจจับพบข้อผิดพลาดการทำงานของโปรแกรม จะแสดงผลตามประเภท Exception ที่ได้ระบุไว้หลังคำสั่ง `except` แต่ถ้าต้องการแสดงผลคำสั่งโปรแกรมหลังจากทำงานจบและไม่พบข้อผิดพลาดเกิดขึ้นเราจะใช้คำสั่ง `else` ดังนี้

รูปแบบการเขียนคำสั่ง `try...except...else` ตรวจจับข้อผิดพลาด

```
try:
    statements
except Exception_1: # ประเภทของข้อผิดพลาดที่ 1
    ...
except Exception_n: # ประเภทของข้อผิดพลาดที่ n
    statements_n
else:
    statements      # คำสั่งที่ทำงานเมื่อไม่พบข้อผิดพลาด 1-
    ↪ n
```

ตัวอย่าง 9.7
การใช้คำสั่ง `try...except...else`

```
x = [5, 12, 6, 9, 13]
try:
    n = int(input('กรุณาป้อนตัวเลข : '))
    i = int(input('กรุณาป้อนตำแหน่งข้อมูลในลิสต์ : '))
    z = x[i]
except IndexError:
    print('ไม่พบตำแหน่งที่คุณระบุในลิสต์ x')
except ArithmeticError:
    print('ตัวหารเป็นเลข 0')
except ValueError:
    print('ข้อมูลที่คุณป้อนไม่ใช่ตัวเลข')
else:
    print('ไม่พบข้อผิดพลาดของ Exception')
    print('ผลลัพธ์ที่ได้ =', z)
```

```
กรุณาป้อนตัวเลข : 5
กรุณาป้อนตำแหน่งข้อมูลในลิสต์ : 3
ไม่พบข้อผิดพลาดของ Exception
ผลลัพธ์ที่ได้ = 9

กรุณาป้อนตัวเลข : 5
กรุณาป้อนตำแหน่งข้อมูลในลิสต์ : 10
ไม่พบตำแหน่งที่คุณระบุในลิสต์ x
```

การทำงานของโปรแกรมตามตัวอย่างที่ 7.6 เมื่อป้อนตัวเลขและระบุตำแหน่งข้อมูลที่อยู่ในลิสต์ได้ถูกต้อง โปรแกรมจะแสดงผลคำสั่งที่ได้กำหนดไว้หลังคำสั่ง **`else`** แต่เมื่อป้อนข้อมูลหรือระบุตำแหน่งไม่ถูกต้องจะแสดงผลการตรวจพบข้อผิดพลาดตามประเภท Exception ส่งผลให้คำสั่งที่อยู่หลังคำสั่ง **`else`** ไม่ทำงาน

9.2.4 การใช้คำสั่ง `try...except...finally` ตรวจจับประเภท Exception

เมื่อนำคำสั่ง `try...except...finally` จะส่งผลให้คำสั่งโปรแกรมที่อยู่ในขอบเขตของคำสั่ง `finally` ทำงานทุกกรณีไม่ว่าจะตรวจพบข้อผิดพลาดหรือไม่ก็ตาม แตกต่างจากคำสั่ง `else` ที่จะแสดงผลคำสั่งโปรแกรมที่อยู่ในขอบเขตเฉพาะกรณีที่ไม่พบข้อผิดพลาดเท่านั้น มีโครงสร้างรูปแบบการใช้งานดังต่อไปนี้

รูปแบบการเขียนคำสั่ง `try...except...finally` ตรวจจับข้อผิดพลาด

```
try:
    statements
except Exception_1: # ประเภทของข้อผิดพลาดที่ 1
    ...
except Exception_n: # ประเภทของข้อผิดพลาดที่ n
    statements_n
finally:
    statements      # คำสั่งที่ทำงานเสมอแม้ไม่พบข้อผิดพลาด
```

ตัวอย่าง 9.8
การเขียนคำสั่งโปรแกรม `try...except...finally`

```
x = [5, 12, 6, 9, 13]
try:
    n = int(input('กรุณาป้อนตัวเลข : '))
    i = int(input('กรุณาป้อนตำแหน่งข้อมูลในลิสต์ : '))
    z = x[i] / n
    print('ผลลัพธ์ที่ได้ =', z)
except IndexError:
    print('ไม่พบตำแหน่งที่คุณระบุในลิสต์ x')
except ArithmeticError:
    print('ตัวหารเป็นเลข 0')
except ValueError:
    print('ข้อมูลที่คุณป้อนไม่ใช่ตัวเลข')
finally:
    print('จบการทำงาน')
```

```
กรุณาป้อนตัวเลข : 2
กรุณาป้อนตำแหน่งข้อมูลในลิสต์ : 3
ผลลัพธ์ที่ได้ = 4.5
จบการทำงาน

กรุณาป้อนตัวเลข : 3
กรุณาป้อนตำแหน่งข้อมูลในลิสต์ : 10
ไม่พบตำแหน่งที่คุณระบุในลิสต์ x
จบการทำงาน
```

ตัวอย่างคำสั่งโปรแกรมที่ 7.7 เป็นการนำเอาคำสั่ง **`finally`** เข้ามาใช้แทนคำสั่ง **`else`** เมื่อโปรแกรมทำงานได้ถูกต้องจะไม่มีการแจ้งเตือนข้อผิดพลาด ส่งผลให้คำสั่งโปรแกรมบรรทัดที่ 13 ทำงาน ถ้าป้อนข้อมูลไม่ถูกต้องโปรแกรมจะแสดงข้อความการแจ้งเตือนข้อผิดพลาด และคำสั่งโปรแกรมบรรทัดที่ 13 ก็จะทำงานเช่นกัน แตกต่างจากตัวอย่างที่ 7.6 ที่ใช้คำสั่ง **`else`** ซึ่งจะทำงานเฉพาะกรณีที่ไม่มีข้อผิดพลาดเกิดขึ้น

9.2.5 การใช้คำสั่ง try...except ซ้อนกันเพื่อตรวจจับประเภท Exception

คำสั่ง try...except ยังสามารถนำมาเขียนซ้อนกันได้ ปกติคำสั่งโปรแกรมที่คาดว่าจะเกิดข้อผิดพลาดอยู่ในขอบเขตของคำสั่ง try ถ้าเราต้องการแยกบางคำสั่งโปรแกรมที่ต้องการตรวจจับออกมาให้เราเพิ่มระดับชั้นของคำสั่ง try...except เข้าไป และกำหนดประเภท Exception มีโครงสร้างรูปแบบการเขียนคำสั่ง try...except ซ้อนกันดังนี้

รูปแบบการเขียนคำสั่ง try...except ซ้อนกันตรวจจับข้อผิดพลาด

```
try:
    statements              # คำสั่งที่ต้องการตรวจจับความผิดพลาด
    try:
        statements          # คำสั่งที่ต้องการตรวจจับความผิดพลาด
    except Exception:       # ประเภทของข้อผิดพลาด
        statements          # คำสั่งให้ทำงานเมื่อพบข้อผิดพลาด
except Exception:           # ประเภทของข้อผิดพลาด
    statements              # คำสั่งให้ทำงานเมื่อพบข้อผิดพลาด
```

ตัวอย่าง 9.9
การเขียนคำสั่ง `try...except` ซ้อนกัน

```
1  try:
2      n = int(input('กรุณาป้อนตัวเลขจำนวนเต็ม : '))
3      try:
4          x = 10
5          z = x / n
6          print(z)
7      except ZeroDivisionError:
8          print('ไม่สามารถหารด้วยเลข 0')
9  except ValueError:
10     print('คุณป้อนข้อมูลไม่ถูกต้อง')
```

```
กรุณาป้อนตัวเลขจำนวนเต็ม : 5.5
คุณป้อนข้อมูลไม่ถูกต้อง

กรุณาป้อนตัวเลขจำนวนเต็ม : 0
ไม่สามารถหารด้วยเลข 0
```

จากตัวอย่างการทำงานของโปรแกรม เมื่อผู้ใช้งานป้อนข้อมูลที่ไม่มีตัวเลขจำนวนเต็ม ส่งผลให้คำสั่งโปรแกรมในบรรทัดที่ 10 แสดงผล เมื่อผู้ใช้งานป้อนเลข **0** ส่งผลให้คำสั่งโปรแกรมในบรรทัดที่ 8 แสดงผล

9.3 การสร้างข้อความแจ้งเตือนข้อผิดพลาดด้วยคำสั่ง raise exception

นอกจากเราใช้คำสั่ง **try...except** ตรวจจับคำสั่งโปรแกรมที่คาดว่าจะเกิดความผิดพลาดขึ้น และแจ้งเตือนตามประเภทของ Exception แล้ว ยังมีคำสั่ง **raise** ให้เราใช้งานสำหรับสร้างข้อความการแจ้งเตือนข้อผิดพลาดขึ้นมาใช้งานเอง เพื่อให้ทราบถึงสาเหตุที่เกิดขึ้นของข้อผิดพลาด ตัวอย่างเช่น การตรวจสอบเงื่อนไขว่าโปรแกรมทำงานถูกต้องตามที่กำหนดไว้หรือไม่ โดยคำสั่ง **raise** มีรูปแบบการใช้งานดังนี้

รูปแบบการเขียนคำสั่ง `raise` สร้างข้อความแจ้งเตือนข้อผิดพลาด

```
raise [Exception [, args [, traceback]]]
```

`Exception`	ประเภทของ Exception ที่ตรวจจับความผิด จะมีหรือไม่มีก็ได้
`args`	อาร์กิวเมนต์ของ Exception
`traceback`	สาเหตุที่ทำให้เกิดข้อผิดพลาด จะมีหรือไม่มีก็ได้

ตัวอย่าง 9.10
การใช้คำสั่ง raise สร้างข้อความแจ้งเตือนข้อผิดพลาด ตรวจสอบค่าตัวเลขจากผู้ใช้ที่ป้อนเข้ามาผ่านทางคีย์บอร์ด

```
n = int(input('ป้อนตัวเลข 1-5 : '))
if n > 5: # ตรวจสอบตัวเลขที่ป้อนเข้ามา
    # ใช้คำสั่ง raise สร้างข้อความแจ้งเตือนข้อผิดพลาด
    raise TypeError('คุณป้อนค่าตัวเลขมากกว่าที่กำหนด', n)
else:
    print('จบการทำงาน !!')
```

```
ป้อนตัวเลข 1-5 : 8
Traceback (most recent call last):
  File 'src/test.py', line 4, in <module>
    raise TypeError('คุณป้อนค่าตัวเลขมากกว่าที่กำหนด', n)
TypeError: ('คุณป้อนค่าตัวเลขมากกว่าที่กำหนด', 8)
```

9.4 การสร้างและเรียกใช้งาน Exception ที่สร้างขึ้นเอง

นอกจากเราจะเรียกใช้งานประเภทของ Exception ที่ภาษาไพธอนจัดเตรียมไว้ให้แล้ว เรายังสามารถสร้าง Exception ขึ้นมาใช้งานเองได้ (User-defined Exception) โดยการสร้าง

คลาสประเภทของข้อผิดพลาดที่สืบทอดมาจากคลาสประเภทของ Exception นั้น ๆ หรือสืบทอดมาจากคลาส Exception เองได้เลย หรือเรียกได้ว่าเป็น Exception ที่เราสร้างขึ้นมาใช้เองเป็น subclass (คลาสลูก) และประเภทของ Exception เป็น superclass (คลาสแม่) ผู้อ่านสามารถศึกษาวิธีการสร้างประเภทของ Exception ขึ้นมาใช้เองได้ตามตัวอย่างต่อไปนี้

ตัวอย่าง 9.11
การสร้างประเภท Exception ขึ้นมาใช้งานเองโดยมีคลาส Exception เป็น super-class

```
class StockError(Exception):
# สร้าง subclass ชื่อ LessthanStrockError ที่มีคลาส Exception เป็น superclass
    def __init__(self):
        # กำหนดให้มีข้อความแสดงผลเมื่อมีข้อผิดพลาดเกิดขึ้น
        Exception.__init__(self, 'สินค้าคงเหลือไม่เพียงพอ')

x = 10
print(f'สินค้าคงเหลือคือ {x} ชิ้น')
n = int(input('จำนวนสินค้าที่ขายได้ = '))
z = x - n
if z < 0:
    # เรียกใช้ประเภท Exception ของ StockError
    raise StockError
else:
    print(f'สินค้าคงเหลือ = {z} ชิ้น')
```

```
สินค้าคงเหลือคือ 10 ชิ้น
จำนวนสินค้าที่ขายได้ = 3
สินค้าคงเหลือ = 7 ชิ้น

สินค้าคงเหลือคือ 10 ชิ้น
จำนวนสินค้าที่ขายได้ = 15
Traceback (most recent call last):
  File 'src/test.py', line 13, in <module>
    raise StockError
__main__.StockError: สินค้าคงเหลือไม่เพียงพอ
```

ตัวอย่าง 9.12
การสร้างการแจ้งเตือนเมื่อผู้ใช้งานกรอกคะแนนสอบ

```
class ScoreTestError(Exception):
    pass

class FailTestError(Exception):
    pass

try:
    n = int(input('กรุณากรอกคะแนนเป็นจำนวนเต็ม (0-100) : '))
    if n < 50:
        raise FailTestError
    elif n > 100:
        raise ScoreTestError
    else:
        print(f'ยินดีด้วยครับคุณสอบผ่าน คะแนนที่คุณได้ คือ {n}')
except FailTestError:
    print (f'คุณไม่ผ่านเกณฑ์ 50 คะแนน คะแนนที่คุณได้ {n} คะแนน')
except ScoreTestError:
    print(f'คะแนนที่คุณได้ คือ {n} > 100 โปรดตรวจสอบอีกครั้ง')
except ValueError:
    print('คุณป้อนข้อมูลไม่ถูกต้อง')
```

```
กรุณากรอกคะแนนเป็นจำนวนเต็ม (0-100) : 45.5
คุณป้อนข้อมูลไม่ถูกต้อง

กรุณากรอกคะแนนเป็นจำนวนเต็ม (0-100) : 45
คุณไม่ผ่านเกณฑ์ 50 คะแนน คะแนนที่คุณได้ 45 คะแนน

กรุณากรอกคะแนนเป็นจำนวนเต็ม (0-100) : 80
ยินดีด้วยครับคุณสอบผ่าน คะแนนที่คุณได้ คือ 80

กรุณากรอกคะแนนเป็นจำนวนเต็ม (0-100) : 888
คะแนนที่คุณได้ คือ 888 > 100 โปรดตรวจสอบอีกครั้ง
```

9.5 การยืนยันความถูกต้อง

การทดสอบโปรแกรมเป็นสิ่งที่ผู้พัฒนาโปรแกรมต้องทำอยู่เสมอ เพื่อตรวจหาข้อผิดพลาดจากการทำงานของคำสั่งโปรแกรม นอกเหนือจากการใช้คำสั่ง `try...except` การใช้คำสั่ง `assert` เพื่อตรวจจับข้อผิดพลาดที่เกิดขึ้นก็เป็นอีกหนึ่งวิธีที่มีความสะดวก ซึ่งเป็นการยืนยันและเพื่อให้แน่ใจว่าคำสั่งโปรแกรมนั้นจะไม่มีโอกาสเกิดข้อผิดพลาดขึ้นอย่างแน่นอน (Assertion) ถ้าคำสั่ง `assert` ตรวจพบข้อผิดพลาดจะแสดงประเภท Exception ของ `AssertionError` ออกมา โดยมีรูปแบบการใช้งานดังนี้

รูปแบบการเขียนคำสั่ง `assert`

```
assert expression [, arguments]
```

`expression` เงื่อนไขที่ต้องการทดสอบการทำงานของโปรแกรม

`arguments` คำอธิบายการเกิดข้อผิดพลาด จะมีหรือไม่มีก็ได้

ตัวอย่าง 9.13
การใช้คำสั่ง assert ทดสอบความถูกต้องการทำงานของคำสั่งโปรแกรม

```
1 x = int(input('ค่าของ x = '))
2 y = int(input('ค่าของ y = '))
3 assert x == y, 'ค่าตัวแปร x และ y ต้องเท่ากัน'
4 print('จบการทำงาน')
```

```
ค่าของ x = 1
ค่าของ y = 1
จบการทำงาน

ค่าของ x = 1
ค่าของ y = 2
Traceback (most recent call last):
  File 'src/test.py', line 3, in <module>
    assert x == y, 'ค่าตัวแปร x และ y ต้องเท่ากัน'
AssertionError: ค่าตัวแปร x และ y ต้องเท่ากัน
```

จากตัวอย่าง 9.13 ในบรรทัดที่ 3 เป็นการใช้คำสั่ง **assert** ตรวจจับข้อผิดพลาดของโปรแกรม เพื่อยืนยันความถูกต้องของค่าตัวแปร x กับ y ต้องมีค่าเท่ากัน

ตัวอย่าง 9.14
การใช้คำสั่ง `assert` ทดสอบความถูกต้องการทำงานของคำสั่งโปรแกรมร่วมกับคำสั่ง `try...except`

```
try:
    n = int(input('กรุณากรอกคะแนนเป็นจำนวนเต็ม (0-30): '))
    # ตรวจจับความผิดพลาดเมื่อค่าตัวแปร n มีค่ามากกว่า 30
    assert n <= 30
    print('คะแนนที่คุณกรอก คือ ', n)
except ValueError:
    # แสดงข้อผิดพลาดประเภท Exception ของ ValueError
    print('คุณป้อนค่าตัวเลขไม่ถูกต้อง')
except AssertionError:
    # แสดงข้อผิดพลาดประเภท Exception ของ AssertionError
    print('คุณป้อนค่าตัวเลขสูงกว่า 30')
```

```
กรุณากรอกคะแนนเป็นจำนวนเต็ม (0-30): 25.2
คุณป้อนค่าตัวเลขไม่ถูกต้อง

กรุณากรอกคะแนนเป็นจำนวนเต็ม (0-30): 30
คะแนนที่คุณกรอก คือ  30

กรุณากรอกคะแนนเป็นจำนวนเต็ม (0-30): 45
คุณป้อนค่าตัวเลขสูงกว่า 30
```

สรุปท้ายบท

ข้อผิดพลาดจากการทำงานของโปรแกรมมีมาจากหลายสาเหตุ ไม่ว่าเกิดจากระบบปฏิบัติการหรือเกิดจากการเขียนคำสั่งโดยผู้เขียนโปรแกรมเอง แต่ภาษาไพธอนก็ได้จัดเตรียมประเภท Exception ต่าง ๆ ให้เราใช้งานจำนวนมาก เพื่อจัดการกับข้อผิดพลาดโดยการใช้คำสั่ง **`try...except`** และเรายังสามารถสร้างการแจ้งเตือนข้อผิดพลาดขึ้นมาใช้งานเองได้ด้วยคำสั่ง **`raise`** รวมไปถึงการใช้คำสั่ง **`assert`** เพื่อยันยันความถูกต้อง (Assertion) การทำงานของคำสั่งโปรแกรมในจุดต่าง ๆ ที่อาจเกิดปัญหาด้วย

แบบฝึกหัด

1. จงอธิบายผลลัพธ์ที่ได้จากคำสั่งโปรแกรมต่อไปนี้ ในกรณีที่ป้อนข้อมูลได้ถูกต้องและไม่ถูกต้อง

```
try:
    n = int(input('กรอกจำนวนเต็มบวกไม่เกิน 10 : '))
    if n <= 10:
        while n <= 10:
            print(n, end=' ')
            n = n + 1
        print('OK')
except ValueError:
        print('ป้อนข้อมูลไม่ถูกต้อง')
```

2. จงอธิบายผลลัพธ์ที่ได้จากคำสั่งโปรแกรมต่อไปนี้ และอธิบายสาเหตุที่ทำให้เกิดการแจ้งเตือนข้อผิดพลาดขึ้น พร้อมทั้งแก้ไขโปรแกรมให้ถูกต้อง

```
try:
    n = int(input('กรอกจำนวนเต็มบวกไม่เกิน 10 : '))
    if x <= 10:
        while n <= 10:
            print(n, end=' ')
            n = n + 1
        print('OK')
except ValueError:
        print('ป้อนข้อมูลไม่ถูกต้อง')
except nameError:
    Print('ตั้งชื่อตัวแปรไม่ถูกต้อง')
```

3. เขียนคำสั่งโปรแกรมให้มีการดักจับ Exception ดังต่อไปนี้

   (a) `FloatingPointError`

   (b) `KeyError`

(c) `IndexError`

(d) `ValueError`