ใบความรู้บทที่ 4

1. การทำงานแบบเงื่อนไข

จากการเขียนโปรแกรมในบทที่ผ่านมา พบว่าทุกคำสั่งในโปรแกรมจะถูกประมวลผลตามลำดับก่อนหลัง เริ่มจากคำสั่งแรกจนถึงคำสั่งสุดท้ายในโปรแกรมหลัก กรณีคำสั่งในโปรแกรมหลักมีการเรียกใช้ฟังก์ชัน การประมวลผลของโปรแกรมจะย้ายเข้าสู่คำสั่งแรกภายในฟังก์ชันและประมวลผลตามลำดับก่อนหลังจนกระทั่งสิ้นสุดการทำงานของฟังก์ชัน และกลับมาสู่การประมวลผลคำสั่งที่อยู่ลำดับถัดจากคำสั่งที่เรียกใช้ฟังก์ชันในโปรแกรมหลัก

การเขียนโปรแกรมข้างต้นเป็นรูปแบบการเขียนโปรแกรมอย่างง่ายและนำมาแก้ปัญหาได้อย่างจำกัด ซึ่งไม่เพียงพอที่จะใช้ในการแก้ปัญหาอีกหลายปัญหาที่พบทั่วไปและอาจมีความซับซ้อนได้ ในที่นี้ไพทอนได้จัดให้มีคำสั่งเงื่อนไข (conditional statement) เพื่อให้ผู้เขียนโปรแกรมสามารถสร้างเงื่อนไขในการเลือกที่จะประมวลผลบล็อกบางบล็อกได้ตามความต้องการ

การเขียนคำสั่งเงื่อนไขอย่างง่ายสามารถทำโดยการใช้คำสั่ง if ซึ่งมีรูปแบบดังนี้

```
if นิพจน์ตรรกะ :
 ~~~~กลุ่มของคำสั่ง #เครื่องหมาย ~ แทนการเคาะแป้นเว้นวรรค 1 ครั้ง
```

บล็อกของคำสั่ง if จะถูกประมวลผลก็ต่อเมื่อนิพจน์ตรรกะให้ผลลัพธ์เป็น True และ ภายในบล็อกของคำสั่ง if จะต้องประกอบด้วยคำสั่งอย่างน้อย 1 คำสั่ง

ตัวอย่าง 4.1 การใช้คำสั่ง if

```
1  num = int(input('Enter a number : '))
2  if num > 0 :
3      print(num, 'is a positive number. ')
4  print('Bye.')
```

ประมวลผลครั้งที่ 1 ผลลัพธ์ที่ได้คือ

```
Enter a number :  4
4 is a positive number.
Bye.
```

ประมวลผลครั้งที่ 2 ผลลัพธ์ที่ได้คือ

```
Enter a number :  -2
Bye.
```

ในบรรทัดที่ 2 นิพจน์ตรรกะ num > 0 เป็นเงื่อนไขที่มีค่าชนิดข้อมูลโดยจะให้ค่า True เมื่อตัวแปร num มีค่ามากกว่า 0 และให้ค่า False เมื่อตัวแปร num มีค่าน้อยกว่าหรือเท่ากับ 0

ในกรณีที่ num > 0 มีค่า True คำสั่งในบรรทัดที่ 3 ซึ่งเป็นบล็อกของ if จะถูกประมวลผล จากนั้นไพทอนจะไปประมวลผลคำสั่งในบรรทัดที่ 4 แต่เมื่อ num > 0 มีค่า False คำสั่งในบรรทัดที่ 3 จะไม่ถูกประมวลผล โดยไพทอนจะข้ามไปประมวลผลในบรรทัดที่ 4

จะเห็นได้ว่าคำสั่งในบรรทัดที่ 4 ไม่ได้เป็นส่วนหนึ่งของบล็อก if การประมวลผลจึงไม่ขึ้นอยู่กับเงื่อนไขบล็อก if และจะถูกประมวลผลถัดจากำสั่ง if

คำสั่ง if ในตัวอย่างที่ 4.1 จัดอยู่ในประเภทคำสั่งเงื่อนไขแบบทางเลือกเดียว นั่นคือเมื่อเงื่อนไขเป็นจริงเลือกที่จะทำคำสั่งในบล็อกของ if และเมื่อเงื่อนไขเป็นเท็จจะข้ามการทำงานบล็อก if ไป

นอกจากนี้ไพทอนยังมีคำสั่งที่ใช้สำหรับสร้างเงื่อนไขแบบสองทางเลือก คือคำสั่ง if-else โดยมีรูปแบบดังนี้

```
if นิพจน์ตรรกะ :
 ~~~~กลุ่มของคำสั่ง 1 #เครื่องหมาย ~ แทนการเคาะแป้นเว้นวรรค 1 ครั้ง
else :
~~~~กลุ่มของคำสั่ง 2
```

เมื่อนิพจน์ตรรกะให้ผลลัพธ์เป็น True แล้วกลุ่มของคำสั่ง 1 ที่เป็นบล็อก if จะถูกประมวลผล แต่ถ้านิพจน์ตรรกะให้ผลลัพธ์เป็น False กลุ่มของคำสั่ง 2 ที่เป็นบล็อก else จะถูกประมวลผล จะเห็นได้ว่ามีบล็อก if หรือบล็อก else เพียงบล็อกใดบล็อกหนึ่งเท่านั้นที่จะถูกประมวลผล ซึ่งสอดคล้องกับค่าของนิพจน์ตรรกะที่มีค่าเป็น True หรือ False

ตัวอย่างที่ 4.2 การใช้คำสั่ง if-else

```
1  name = input('Enter John or Anna : ')
2  if name == 'John' :
3      print('Hello John')
4  else :
5      print('Hello Anna')
6  print('Nice to meet you.')
```

ประมวลผลครั้งที่ 1 ผลลัพธ์ที่ได้คือ

```
Enter John or Anna :  John
Hello John
Nice to meet you.
```

ประมวลผลครั้งที่ 2 ผลลัพธ์ที่ได้คือ

```
Enter John or Anna :  Anna
Hello Anna
Nice to meet you.
```

ในการประมวลผลครั้งที่ 1 เมื่อผู้ใช้ป้อนค่า John จะทำให้นิพจน์ตรรกะ name == 'john' มีค่า True จึงส่งผลให้บล็อก if ถูกประมวลผล ขนาดที่การประมวลผลครั้งที่ 2 เมื่อผู้ใช้ป้อนค่า Anna จะทำให้นิพจน์ตรรกะ name == 'John' มีค่า False ส่งผลให้บล็อก else ถูกประมวลผลและเมื่อประมวลผลคำสั่ง if-else เสร็จแล้ว คำสั่งในบรรทัดที่ 6 จึงถูกประมวลผลเป็นลำดับถัดไป

สำหรับการแก้ปัญหาที่ต้องมีการตัดสินใจมากกว่า 2 ทางเลือกหรือเงื่อนไขแบบหลายๆทางเลือก ไพทอนได้จัดให้มีรูปแบบการใช้คำสั่ง if เชิงซ้อน (nested if) เพื่อสร้างทางเลือกต่อเนื่องหลายทางเลือก โดยมีรูปแบบดังนี้

```
if นิพจน์ตรรกะ 1:
    กลุ่มของคำสั่ง 1
else :
    if นิพจน์ตรรกะ 2:
        กลุ่มของคำสั่ง 2
    else :
        if นิพจน์ตรรกะ 3:
            กลุ่มของคำสั่ง 3
        else :
            กลุ่มของคำสั่ง 4
```

ในที่นี้เมื่อนิพจน์ตรรกะ 1 ให้ผลลัพธ์เป็น True แล้วกลุ่มของคำสั่ง 1 ที่เป็นบล็อก if ชั้นนอกจะถูกประมวลผลแต่ถ้านิพจน์ตรรกะ 1 ให้ผลลัพธ์เป็น False บล็อก else ชั้นนอกจะถูกประมวลผลโดยจะประเมินค่านิพจน์ตรรกะ 2 ซึ่งถ้าให้ผลลัพธ์เป็น True แล้วกลุ่มของคำสั่ง 2 ที่เป็นบล็อก if ชั้นกลางจะถูกประมวลผล แต่ถ้านิพจน์ตรรกะ 2 ให้ผลลัพธ์เป็น False บล็อก else ชั้นกลางจะถูกประมวลผล โดยจะประเมินค่านิพจน์ตรรกะ 3 ซึ่งถ้าให้ผลลัพธ์เป็น True แล้วกลุ่มของคำสั่ง 3 ที่เป็นบล็อก if ชั้นในสุดจะถูกประมวลผล แต่ถ้านิพจน์ตรรกะ 3 ให้ผลลัพธ์เป็น False บล็อก else ชั้นในสุดที่มีกลุ่มของคำสั่ง 4 จะถูกประมวลผล

จะเห็นได้ว่ามีบล็อก if หรือบล็อก else เพียงบล็อกใดบล็อกหนึ่งเท่านั้นที่จะถูกประมวลผลก่อนจะไปประมวลผลคำสั่งที่อยู่ถัดจากคำสั่ง if

เราสามารถสร้างคำสั่ง if เชิงซ้อนที่มีเงื่อนไขต่อเนื่องได้มากกว่านี้ แต่จะทำให้โปรแกรมมีความซับซ้อนมากยิ่งขึ้น และยากต่อการติดตามการประมวลผลในกรณีที่มีข้อผิดพลาดเกิดขึ้นในโปรแกรม

ตัวอย่างที่ 4.3 การใช้คำสั่ง if เชิงซ้อน

```
1  num = int(input('Enter a number : '))
2  if num > 0 :
3      print(num,' is a positive number.')
4  else :
5      if num < 0 :
6          print(num,' is a negative number.')
7      else :
8          print(num,' is zero.')
9  print('Bye. ')
```

ประมวลผลครั้งที่ 1 ผลลัพธ์ที่ได้คือ

```
Enter a number :  7
7  is a positive number.
Bye.
```

ประมวลผลครั้งที่ 2 ผลลัพธ์ที่ได้คือ

```
Enter a number :  -1
-1  is a negative number.
Bye.
```

บล็อกของ if ในบรรทัดที่ 2 ประกอบด้วยการเรียกใช้ฟังก์ชัน print( ) เพียงคำสั่งเดียวเท่านั้นในบรรทัดที่ 3 ซึ่งจะถูกปะมวลผลเมื่อ num มีค่ามากกว่า 0

ในขณะที่บล็อกของ else ในบรรทัดที่ 4 ประกอบด้วยคำสั่ง if-else ในบรรทัดที่ 5- 8 ซึ่งจะถูกประมวลผลก็ต่อเมื่อ num มีค่าน้อยกว่าหรือเท่ากับ 0

ภายในบล็อก else ในบรรทัดที่ 4 นี้มีการแจกแจงเงื่อนไขต่อเนื่องเพื่อให้มีการแสดงผลลัพธ์ที่แตกต่างกันในอีก 2 กรณี คือ เมื่อ num มีค่าน้อยกว่า 0 และเมื่อ num มีค่าไม่น้อยกว่า 0

การแก้ปัญหาโดยใช้เงื่อนไขแบบหลายทางเลือกหรือการใช้ if เชิงซ้อน ทำให้เกิดการซ้อนกันเป็นชั้นๆ ของบล็อกย่อย โดยจะเยื้องเข้าไปทางด้านขวาลึกมากขึ้นเรื่อยๆ เมื่อจำนวนชั้นเพิ่มขึ้น อาจทำให้เกิดความยุ่งยากในการทำความเข้าใจมากยิ่งขึ้น ดังนั้นไพทอนจึงจัดให้มีรูปแบบการเขียนที่เข้าใจได้ง่ายขึ้น โดยมีความคิดเหมือนเดิม นั่นคือ คำสั่ง if-elif โดยมีรูปแบบดังนี้

```
if นิพจน์ตรรกะ 1:
~~~~กลุ่มของคำสั่ง 1 #เครื่องหมาย ~ แทนการเคาะแป้นเว้นวรรค 1 ครั้ง
elif นิพจน์ตรรกะ 2 :
~~~~กลุ่มของคำสั่ง 2
elif นิพจน์ตรรกะ 3 :
```

```
~~~~กลุ่มของคำสั่ง 3
...
else :
~~~~กลุ่มของคำสั่ง N
```

เมื่อนำคำสั่ง if-elif มาใช้กับตัวอย่างที่ 4.3 โปรแกรมที่เขียนขึ้นจะมีลักษณะดังตัวอย่างที่ 4.4

ตัวอย่างที่ 4.4 การใช้คำสั่ง if-elif

```
1  num = int(input('Enter a number : '))
2  if num > 0 :
3      print(num,' is a positive number.')
4  elif num < 0 :
5      print(num,' is a negative number.')
6  else :
7      print(num,' is zero.')
8  print('Bye. ')
```

ผลลัพธ์ที่ได้จากตัวอย่างที่ 4.4 มีลักษณะเดียวกับตัวอย่างที่ 4.3

2. การวนซ้ำ

ในขั้นตอนการแก้ปัญหาบางครั้งผู้เขียนโปรแกรมมีความต้องการที่จะประมวลผลกลุ่มของคำสั่งชุดใดชุดหนึ่งซ้ำกันหลายครั้ง แทนการเขียนกลุ่มของคำสั่งเหล่านั้นซ้ำกันในโปรแกรมไพทอนมีคำสั่งวนซ้ำ (loop statement) while ที่ช่วยให้การเขียนโปรแกรมกระชับมากยิ่งขึ้น

คำสั่ง while มีรูปแบบดังนี้

```
while นิพจน์ตรรกะ :
 ~~~~กลุ่มของคำสั่ง #เครื่องหมาย ~ แทนการเคาะแป้นเว้นวรรค 1 ครั้ง
```

เมื่อนิพจน์ตรรกะให้ผลลัพธ์เป็น True บล็อก while จะถูกประมวล 1 ครั้ง จากนั้นนิพจน์ตรรกะจะถูกประเมินค่าอีกครั้ง หากนิพจน์ตรรกะมีค่า True อีก บล็อก while จะถูกประมวลผลซ้ำอีก และจะเป็นเช่นนี้ไปเรื่อยๆ จนกว่านิพจน์ตรรกะจะให้ค่าเป็น False จึงจะสิ้นสุดการทำงานของคำสั่ง while

ในกรณีที่นิพจน์ตรรกะให้ค่าเป็น Ture ทุกครั้งที่ถูกประเมินค่า จะส่งผลให้บล็อก while ถูกประมวลผลโดยไม่มีวันสิ้นสุด เรียกการทำงานในลักษณะที่มีการวนทำซ้ำไม่รู้จบนี้ว่า การวนซ้ำไม่รู้จบ (infinite) ซึ่งจัดเป็นการทำงานที่ไม่ถูกต้อง และผู้เขียนโปรแกรมต้องระมัดระวังไม่ให้เกิดขึ้น

ดังนั้น อาจกล่าวได้ว่าในการใช้คำสั่ง while ให้ทำงานได้อย่างถูกต้องนั้นนอกจากต้องกำหนดนิพจน์ตรรกะที่เหมาะสมในการควบคุมการวนซ้ำกลุ่มของคำสั่งในบล็อก while ให้เป็นไปตามที่ต้องการแล้วยังต้องมี

คำสั่งในบล็อกของ while ที่มีการเปลี่ยนแปลงค่าตัวแปรที่เป็นส่วนหนึ่งของนิพจน์ตรรกะ เพื่อให้ในที่สุดแล้วนิพจน์ตรรกะมีค่าเป็น False และส่งผลให้สิ้นสุดการทำงานของคำสั่ง while

ตัวอย่างที่ 4.5 การใช้คำสั่ง while

```
1  num = int(input('Enter a number of repetitions : '))
2  count = 1
3  while count <= num :
4      print(count)
5      count = count + 1
6  print('Bye. ')
```

ประมวลผลครั้งที่ 1 ผลลัพธ์ที่ได้คือ

```
Enter a number of repetitions : 5
1
2
3
4
5
Bye.
```

ประมวลผลครั้งที่ 2 ผลลัพธ์ที่ได้คือ

```
Enter a number of repetitions : 3
1
2
3
Bye.
```

ประมวลผลครั้งที่ 3 ผลลัพธ์ที่ได้คือ

```
Enter a number of repetitions : 0
Bye.
```

จะเห็นได้ว่าในที่นี้ตัวแปล num ใช้กำหนดจำนวนรอบในการวนซ้ำโดยมีตัวแปรcount ใช้ในการนับรอบว่าดำเนินการครบ num รอบหรือยัง โดยมีนิพจน์ตรรกะ count <= num เป็นเงื่อนไขในการตรวจสอบการวนซ้ำ

สังเกตในบล็อก while มีคำสั่งในบรรทัดที่ 5 ที่ใช้เปลี่ยนแปลงค่าตัวแปร count ให้มีค่าเพิ่มขึ้นจากเดิมอีก 1 ในแต่ละรอบ ตราบใดที่ count ยังคงมีค่าน้อยกว่าหรือเท่ากับ num บล็อก while จะถูกประมวลผลจนกระทั่ง count มีค่ามากกว่า num การวนซ้ำของบล็อก while จึงสิ้นสุดลง และไปประมวลผลคำสั่งในบรรทัดที่ 6 ก่อนจบการทำงานของโปรแกรม

ตัวอย่างที่ 4.6 การพิมพ์ข้อความซ้ำ

```
1 num = int(input('Enter a number : '))
2 count = 1
3 while count <= num:
4     print("Let's play! ")
5     count = count + 1
6 print('Bye. ')
```

ผลลัพธ์ที่ได้คือ

```
Enter a number :  4
Let's play!
Let's play!
Let's play!
Let's play!
Bye.
```

ในที่นี้โปรแกรมใช้ตัวแปร count เพื่อวัตถุประสงค์ในการนับรอบของการวนซ้ำเท่านั้นซึ่งจะไม่ถูกแสดงเป็นส่วนหนึ่งของผลลัพธ์

ตัวอย่างที่ 4.7 โปรแกรมพิมพ์ข้อความซ้ำโดยการนับถอยหลัง

```
1 num = int(input('Enter a number of repetitions : '))
2 count = num
3 while count > 0:
4     print("Let's play! ")
5     count = count - 1
6 print('Bye. ')
```

ผลลัพธ์ที่ได้คือ

```
Enter a number :  5
Let's play!
Let's play!
Let's play!
Let's play!
Let's play!
Bye.
```

ในการนับรอบของการวนซ้ำอาจใช้การนับเพิ่มอย่างเช่นในตัวอย่างที่ 4.5 และ 4.6 หรือใช้การนับถอยหลังตามตัวอย่างที่ 4.7 ก็ได้ นอกจากนี้การนับเพิ่มหรือนับถอยหลังไม่จำเป็นต้องเพิ่มหรือลดทีละ 1 เท่านั้น

จากการวนซ้ำในตัวอย่างที่ผ่านมาพบว่า มีการกำหนดจำนวนรอบของการวนซ้ำก่อนดำเนินการคำสั่ง while เรียกรูปแบบการควบคุมการวนซ้ำในลักษณะนี้ว่าการควบคุมด้วยการนับ เช่น ในตัวอย่างที่ 4.5 โจทย์ปัญหาอาจเป็นการให้แสดงผลจำนวนเต็มจาก 1 จนถึง num เมื่อ num เป็นจำนวนเต็มที่รับเข้ามา หรือในตัวอย่างที่ 4.6 เป็นการแสดงข้อความ Let's play! จำนวน num ครั้ง เมื่อ num เป็นจำนวนเต็มที่รับเข้ามาเช่นกัน

อย่างไรก็ตามในหลายปัญหาอาจไม่สามารถกำหนดจำนวนรอบในการวนซ้ำล่วงหน้าได้จึงจำเป็นต้องสร้างเงื่อนไขรูปแบบอื่นเพื่อให้มีการเริ่มต้นและสิ้นสุดการวนซ้ำ

ตัวอย่างที่ 4.8 โปรแกรมทายตัวเลข

```
1  target = 65
2  print('--<< Guessing a number >>--')
3  num = int(input('Enter your guess : '))
4  while num != target :
5      print('Your guess is in incorrect, try again')
6      num = int(input('Enter your guess: '))
7  print("You've got it right.")
```

ผลลัพธ์ที่ได้คือ

```
--<< Guessing a number >>--
Enter your guess : 13
Your guess is in incorrect, try again
Enter your guess : 42
Your guess is in incorrect, try again
Enter your guess : -8
Your guess is in incorrect, try again
Enter your guess : 65
You've got it right.
```

ในบรรทัด 1 เป็นการกำหนดค่าที่ใช้สำหรับทายตัวเลขให้กับตัวแปร target และในบรรทัดที่ 3 มีการรับค่าที่ผู้ใช้คาดเดาเป็นครั้งแรกก่อนเริ่มต้นการวนซ้ำในบรรทัดที่ 4

เนื่องจากผู้ใช้ใช้การคาดเดาตัวเลขที่กำหนดเป็นค่าเป้าหมายในโปรแกรม ดังนั้นจึงไม่อาจกำหนดจำนวนครั้งในการทายค่าเป้าหมายดังกล่าวให้ถูกต้องล่วงหน้าก่อนการวนซ้ำได้ ในที่นี้จึงใช้วิธีการสร้างเงื่อนไขที่ต้องการในนิพจน์ตรรกะที่ใช้ควบคุมคำสั่ง while นั่นคือนิพจน์ num != target ตราบใดที่นิพจน์ดังกล่าวมีค่า

False และจะสิ้นสุดการทำงานของคำสั่ง while และประมวลผลในคำสั่งบรรทัดที่ 7 ก่อนจบการทำงานของโปรแกรม

จะเห็นได้ว่าเพื่อให้ตัวเลขที่ผู้ใช้คาดเดาเพื่อทายค่าเป้าหมายมีค่าเปลี่ยนแปลงไปในแต่ละรอบของการวนซ้ำ จึงมีการรับค่า num ใหม่ในบรรทัดที่ 6 ไม่ใช่นั้นแล้วตัวแปร num จะยังคงมีค่าเดิมซึ่งเป็นค่าที่ยังไม่ตรงกับ target และส่งผลให้เกิดการวนซ้ำไม่รู้จบในบล็อกของคำสั่ง while

3. คำสั่ง break

คำสั่ง break เป็นคำสั่งที่ใช้ร่วมกับคำสั่งวนซ้ำ เมื่อมีการประมวลผลคำสั่ง break จะมีผลให้โปรแกรมออกจากการวนซ้ำในทันที และไปประมวลผลคำสั่งในลำดับถัดจากคำสั่งวนซ้ำ ดังนั้น คำสั่ง break จึงมักจะถูกกำกับด้วยเงื่อนไขเพื่อยุติการวนซ้ำ

ตัวอย่างที่ 4.9 โปรแกรมทายตัวเลข โดยใช้คำสั่ง break

```
1 target = 65
2 print('--<< Guessing a number >>--')
3 while num != target:
4     num = int(input('Enter your guess : '))
5     if num == target :
6         break
7     else :
8         print('Your guess is in incorrect, try again')
9 print("You've got it right.")
```

ผลลัพธ์ที่ได้คือ

```
--<< Guessing a number >>--
Enter your guess : 12
Your guess is in incorrect, try again
Enter your guess : 68
Your guess is in incorrect, try again
Enter your guess : 91
Your guess is in incorrect, try again
Enter your guess : 65
You've got it right.
```

ในบรรทัดที่ 3 นิพจน์ตรรกะที่ใช้ในคำสั่ง while มีค่าเป็น True จึงทำให้คำสั่งวนซ้ำนี้ทำงานไม่มีที่สิ้นสุด อย่างไรก็ตามในบรรทัดที่ 5 มีเงื่อนไขที่ใช้ในการตรวจสอบค่าที่ผู้ใช้คาดเดากับค่าเป้าหมาย ในกรณีที่

ตรงกัน นั่นคือ num == target มีค่า True จะมีการประมวลผลคำสั่ง break ซึ่งส่งผลให้คำสั่งวนซ้ำยุติการทำงาน และไปประมวลผลคำสั่งในบรรทัดที่ 9 ก่อนจบการทำงานของโปรแกรม

นอกจากนี้ในการประเมินค่านิพจน์ตรรกะของคำสั่ง while ในครั้งแรก พบว่ามีค่า True จึงเข้าไปประมวลผลในบล็อก while เสมอ แล้วจึงมีการรับค่าที่ผู้ใช้คาดเดาในบรรทัดที่ 4 ก่อนนำไปเปรียบเทียบในบรรทัดที่ 5 ว่าตรงกับค่าเป้าหมายหรือไม่ จะเห็นได้ว่าการรับค่า num ในตัวอย่างนี้มีการรับค่าแค่จุดเดียวเท่านั้น ซึ่งกระชับกว่าโปรแกรมในตัวอย่างที่ 4.8

4. คำสั่ง Continue

คำสั่ง continue เป็นคำสั่งที่ใช้ร่วมกับคำสั่งวนซ้ำเช่นเดียวกับคำสั่ง break เมื่อมีการประมวลผลคำสั่ง continue ไพทอนจะไปประมวลผลนิพจน์ตรรกะของคำสั่ง while เพื่อเริ่มต้นทำงานในบล็อก while ในรอบถัดไป ในทำนองเดียวกันกับคำสั่ง continue มักจะถูกกำกับด้วยเงื่อนไข

ตัวอย่างที่ 4.10 การใช้คำสั่ง continue เพื่อข้ามการพิมพ์เลขคู่

```
1 num = int(input('Enter a number  of repetitions : '))
2 count = 0
3 while count < num :
4     count = count + 1
5     if count%2 == 0 :
6         continue
7     print(count)
8 print('Bye. ')
```

ผลลัพธ์ที่ได้คือ

```
Enter a number  of repetitions : 10
1
3
5
7
9
Bye.
```

บล็อก while ตั้งแต่บรรทัดที่ 3-7 ทำงานทั้งหมด num รอบ เริ่มจากรอบแรกเมื่อ count มีค่าเป็น 0 ในแต่ละรอบ count มีค่าเพิ่มขึ้นจากเดิมอีก 1 และรอบสุดท้ายเมื่อ count มีค่า num-1 เนื่องจากในรอบถัดไปที่ count มีค่าเป็น num จะทำให้นิพจน์ตรรกะ count < num เป็น False และสิ้นสุดการทำงานของคำสั่ง while

ในบรรทัดที่ 7 พบว่าการเรียกใช้ฟังก์ชัน print( ) ซึ่งอยู่ในลำดับสุดท้ายของบล็อกจะถูกประมวลผลโดยที่ไม่มีเงื่อนไข อย่างไรก็ตามฟังก์ชัน print( ) ไม่ได้ถูกเรียกใช้ทุกรอบ ทั้งนี้เนื่องจากเมื่อนิพจน์ตรรกะ count%2 == 0 ในบรรทัดที่ 5 เป็น True (นั่นคือ เมื่อ count หารด้วย 2 ลงตัว) แล้วคำสั่ง continue จะถูกประมวลผล ซึ่งส่งผลให้การทำงานวนกลับไปสู่การประเมินค่านิพจน์ตรรกะของคำสั่ง while ในรอบถัดไป โดยข้ามการประมวลผลบรรทัดที่ 7 ไป

5. คำสั่งวนซ้ำเชิงซ้อน

คำสั่งวนซ้ำเชิงซ้อนอยู่ในคำสั่งวนซ้ำอื่น เรียกการใช้คำสั่งวนซ้ำในลักษณะนี้ว่า คำสั่งวนซ้ำเชิงซ้อน (nested loop) การใช้คำสั่งวนซ้ำเชิงซ้อนในโปรแกรมจะทำให้โปรแกรมมีความซับซ้อนมากยิ่งขึ้นเพื่อรองรับการแก้ปัญหาที่มีความซับซ้อน

ตัวอย่างที่ 4.11 การใช้คำสั่งวนซ้ำเชิงซ้อนเพื่อคำนวณค่าแฟคทอเรียลตั้งแต่ 1 ถึง num

```
1  print('---<< Factorial Calculation >>--')
2  num = int(input('Enter a number : '))
3  outer = 1
4  while outer <= num :
5      inner = 1
6      product = 1
7      while inner <= outer :
8          product = product * inner
9          inner = inner + 1
10     print(outer, '! = ', product, sep=' ')
11     outer = outer + 1
12 print('Bye. ')
```

ผลลัพธ์ที่ได้คือ

```
---<< Factorial Calculation >>--
Enter a number : 5
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
Bye.
```

ในตัวอย่างนี้มีคำสั่ง while ซ้อนกันอยู่ 2 ชั้น บล็อก while ชั้นนอกเริ่มตั้งแต่บรรทัดที่ 4-11 และบล็อก while ชั้นในเริ่มตั้งแต่บรรทัดที่ 7-9

คำสั่งวนซ้ำทั้งสองชั้นมีการควบคุมด้วยการนับ ชั้นนอกนับจำนวนรอบด้วยตัวแปร outer และชั้นในนับจำนวนรอบด้วยตัวแปร inner

ตัวแปร outer มีค่าเริ่มต้นจาก 1 จนถึง num ในแต่ละรอบมีค่าเพิ่มขึ้นจากเดิมอีก 1 ส่วนตัวแปร inner มีค่าเริ่มต้นจาก 1 จนถึง outer ในแต่ละรอบมีค่าเพิ่มขึ้นจากเดิมอีก 1 เช่นกัน นั่นคือ จำนวนรอบในการวนซ้ำของคำสั่ง while ชั้นในขึ้นอยู่กับตัวแปร outer

ในแต่ละรอบของคำสั่ง while ชั้นนอก จะมีการซ้ำบล็อก while ชั้นในตั้งแต่รอบแรกจนถึงรอบสุดท้าย และจนวนซ้ำในลักษณะนี้จนกระทั่งถึงรอบสุดท้ายของคำสั่ง while ชั้นนอก

ในแต่ละรอบของคำสั่ง while ชั้นนอก เริ่มต้นด้วยการกำหนดค่า 1 ให้กับตัวแปร inner และ product จากนั้นตัวแปรทั้งสองมีค่าเปลี่ยนแปลงไปตามคำสั่งในแต่ละรอบของคำสั่งในบรรทัดที่ 8 และ 9 ในที่นี้ตัวแปร product เป็นการสะสมผลคูณเพื่อคำนวณค่าแฟคทอเรียลตามจำนวนรอบของคำสั่ง while ชั้นใน

สรุป

จากการเขียนโปรแกรมในบทที่ผ่านมา นักเรียนได้เขียนโปรแกรมอย่างง่ายที่มีการประมวลผลตามลำดับก่อนหลัง และเพื่อให้สามารถเขียนโปรแกรมที่มีการทำงานซับซ้อนได้ ไพทอนมีคำสั่งที่ควบคุมการทำงานแบบเงื่อนไขให้ใช้งานคือ if , if-else และ if-else โดยการทำงานขึ้นอยู่กับเงื่อนไขทางเลือกหรือนิพจน์ตรรกะ ถ้าใช้คำสั่ง if บล็อกคำสั่ง if จะถูกประมวลผลก็ต่อเมื่อนิพจน์ตรรกะเป็นจริง แต่ถ้าเป็นเท็จบล็อกของ else จะถูกประมวลผลสำหรับคำสั่ง if-else นั้นใช้ในกรณีที่มีเงื่อนไขหลายทางเลือก

ไพทอนมีคำสั่งที่ช่วยให้การเขียนโปรแกรมง่ายและซับซ้อนขึ้นเมื่อมีการทำงานเดิมซ้ำๆกัน คือคำสั่งวนซ้ำ while โดยบล็อกของ while จะถูกประมวลผลก็ต่อเมื่อนิพจน์ตรรกะเป็นจริง นิพจน์ตรรกะจะเป็นตัวควบคุมการวนซ้ำ ซึ่งถ้านิพจน์ตรรกะเป็นจริงอยู่เสมอการวนซ้ำนั้นจะทำงานไม่รู้จบเพราะฉะนั้นผู้เขียนดปรแกรมอาจใช้การเปลี่ยนแปลงค่าตัวแปรที่มีผลกับนิพจน์ตรรกะนั้นมาช่วยให้นิพจน์ตรรกะเป็นจริงเพื่อจบการทำงานในบล็อก while นอกจากนี้ไพทอนยังมีคำสั่ง break และ continue ที่ใช้ร่วมกับคำสั่งวนซ้ำ break ใช้เพื่อให้ออกจากการทำงานของการวนซ้ำ continue ใช้เพื่อให้เริ่มต้นทำงานการวนซ้ำในรอบถัดไป break และ continue มักจะถูกกกับด้วยเงื่อนไข

อ้างอิง จากหนังสือเรียนรายวิชาเพิ่มเติม เทคโนโลยีสารสนเทศและการสื่อสาร ภาษาไพทอน ชั้นมัธยมศึกษาปีที่ ๔-๖ (สสวท)