

บทที่ 6

คำสั่งควบคุมทิศทางการทำงาน โปรแกรม

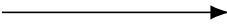
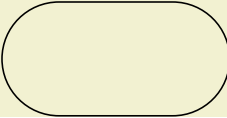

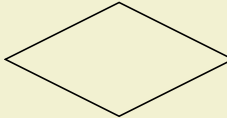
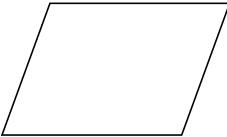
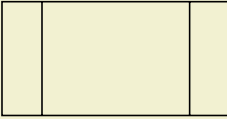
จากหลายบทที่ผ่านมา ผู้อ่านได้ลองฝึกการเขียนคำสั่งโปรแกรมไปบ้างแล้ว ในบทนี้ผู้อ่านจะได้เรียนรู้การเขียนคำสั่งโปรแกรมควบคุมทิศทางการทำงาน (Control Statement) ซึ่งแบ่งออกได้ทั้งหมด 3 รูปแบบ ได้แก่ การเขียนคำสั่งโปรแกรมควบคุมทิศทางแบบลำดับ (Sequence Control Statement), แบบมีเงื่อนไข (Conditions Control Statement) และแบบทำซ้ำ (Iteration Control Statement)

6.1 รู้จักกับผังงาน



โดยพื้นฐานแล้วเราสามารถใช้ใช้ผังงาน (Flowchart) ในการเรียนรู้การเขียนคำสั่งโปรแกรมควบคุมทิศทางการทำงาน ซึ่งผังงานก็คือแผนภาพสัญลักษณ์แสดงขั้นตอนจากเริ่มต้นจนถึงสิ้นสุดในกระบวนการแก้ปัญหา (โดยทั่วไปแผนภาพจะเรียงลำดับขั้นตอนจากด้านบนลงด้านล่าง) ช่วยให้เราลำดับขั้นตอน ตรวจสอบ และแก้ไขกระบวนการแก้ปัญหาได้ง่าย ผังงานประกอบไปด้วยสัญลักษณ์ที่ใช้แสดงถึงแต่ละขั้นตอน ทั้งนี้มาตรฐาน ANSI ¹ และ ISO ² 5807:1985 ได้กำหนดสัญลักษณ์ในการสร้างผังงาน ดังตาราง 6.1

¹ American National Standards Institute: ANSI

² The International Organization for Standardization: ISO

สัญลักษณ์	ชื่อเรียก	คำอธิบาย
	ศรทิศทาง (Flowline)	ใช้ระบุทิศทางของกระบวนการ อธิบายในสัญลักษณ์นี้เพื่ออธิบายกระบวนการเพิ่มเติมได้ตามจำเป็น
	เทอร์มินัล (Terminal)	ใช้ระบุจุดเริ่มต้นหรือจุดสิ้นสุดของกระบวนการ โดยต้องระบุชื่ออย่างเช่น START, BEGIN, TERMINATE หรือ STOP ในสัญลักษณ์นี้
	กระบวนการ (Process)	ใช้แทนคำสั่ง หรือการทำงานพร้อมระบุคำอธิบายอย่างสั้นในสัญลักษณ์นี้
	การตัดสินใจ (Decision)	ใช้แทนการทำงานที่มีเงื่อนไข โดยต้องระบุทิศทางของกระบวนการต่อไปสองกระบวนการที่ต่างกัน และระบุเงื่อนไขที่ทำให้ค่าจริงหรือเท็จในสัญลักษณ์นี้
	การรับเข้า/ส่งออก (Input/Output)	ใช้แทนการทำงานที่มีการรับเข้าหรือส่งออกข้อมูล โดยระบุว่าแสดงหรือนำเข้าข้อมูลใดในสัญลักษณ์นี้
	กระบวนการที่มีการนิยามแล้ว (Pre-defined Process)	ใช้แทนหนึ่งกระบวนการที่มีการบอกชื่อกระบวนการไว้ก่อนแล้ว ณ ตำแหน่งอื่น

ตาราง 6.1: สัญลักษณ์ของผังงานตามมาตรฐาน ANSI/ISO (มีต่อ)

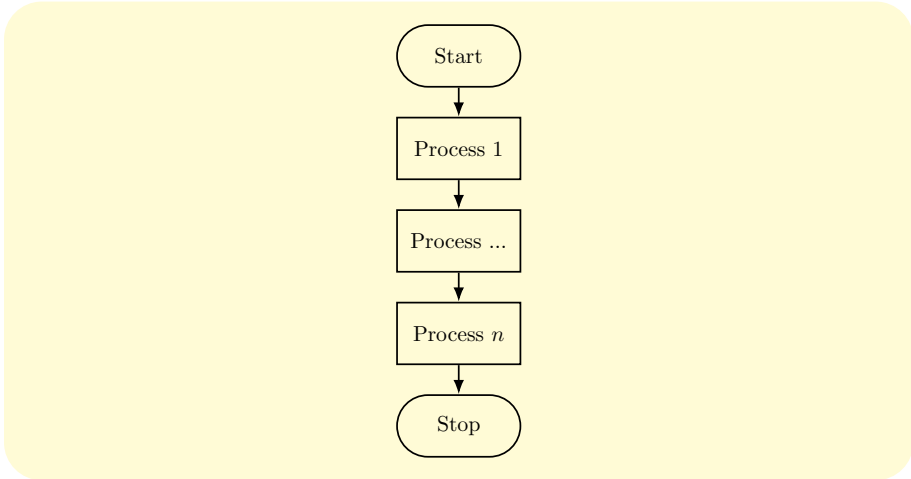
สัญลักษณ์	ชื่อเรียก	คำอธิบาย
	ตัวเชื่อม ในหน้า (On-Page Connector)	ระบุชื่อในสัญลักษณ์นี้ โดยใช้เป็นคู่เพื่อเชื่อม กระบวนการ เมื่อการเขียนผังงานเกิดทิศทาง ทับซ้อน หรืออาจทำให้เกิดความสับสนในผัง งาน ซึ่งตัวเชื่อมคู่นี้ต้องอยู่ในหน้าเดียวกัน
	ตัวเชื่อม นอกหน้า (Off-Page Connector)	ระบุชื่อในสัญลักษณ์นี้ โดยใช้เป็นจุดเชื่อม กระบวนการ เมื่อเขียนผังงานที่ยาวเกินกว่า หนึ่งหน้า

ตาราง 6.1: สัญลักษณ์ของผังงานตามมาตรฐาน ANSI/ISO

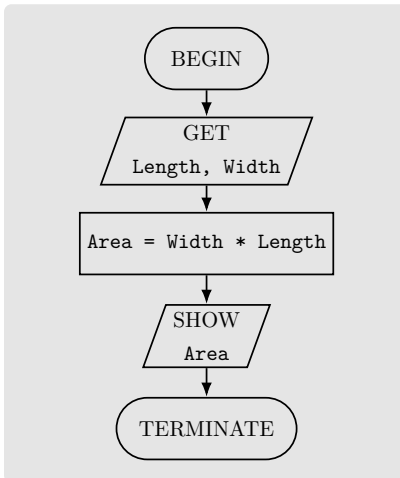
6.2 การเขียนคำสั่งโปรแกรมควบคุมทิศทางการทำงานแบบลำดับ

การเขียนคำสั่งโปรแกรมควบคุมทิศทางการทำงานแบบลำดับ เป็นลักษณะการเขียนคำสั่งให้โปรแกรมทำงานจากบนลงล่าง โดยให้ทำงานตามคำสั่งที่ 1, คำสั่งที่ 2, คำสั่งที่ 3 ไปเช่นนี้จนครบทุกคำสั่ง ไม่มีคำสั่งการตัดสินใจหรือมีคำสั่งการทำซ้ำเข้ามาเกี่ยวข้อง

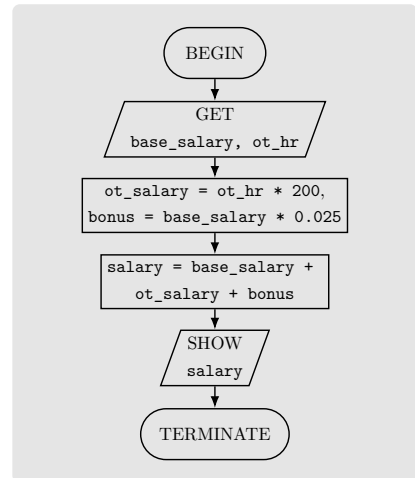
รูปแบบการเขียนผังงานของการเขียนโปรแกรมควบคุมทิศทางการทำงานแบบลำดับแสดงดังรูป 6.1 และเราได้ให้ตัวอย่างการเขียนแผนภาพผังงานคำนวณหาพื้นที่สี่เหลี่ยมผืนผ้า และการเขียนคำสั่งโปรแกรมควบคุมทิศทางการทำงานแบบลำดับคำนวณรายได้สุทธิ ดังต่อไปนี้



รูป 6.1: ผังงานคำสั่งควบคุมทิศทางการทำงานแบบลำดับแบบทั่วไป



(a) ตัวอย่าง 6.1



(b) ตัวอย่าง 6.2

รูป 6.2: ตัวอย่างผังงานคำสั่งควบคุมทิศทางการทำงานแบบลำดับ

ตัวอย่าง 6.1

การเขียนคำสั่งโปรแกรมควบคุมทิศทางการทำงานแบบลำดับคำนวณหาพื้นที่สี่เหลี่ยมผืนผ้า

```

1  # รอรับค่าการป้อนข้อมูลด้านความยาวแล้วแปลงเป็นจำนวนทศนิยม
2  x = float(input('กรุณาป้อนความยาวของสี่เหลี่ยมผืนผ้า: '))
3  # รอรับค่าการป้อนข้อมูลด้านความกว้างแล้วแปลงเป็นจำนวนทศนิยม
4  y = float(input('กรุณาป้อนความกว้างของสี่เหลี่ยมผืนผ้า: '))
5  area = x * y # ดำเนินการคูณค่าตัวแปร x กับ y ผลลัพธ์เก็บไว้ในตัวแปร area
6  print('พื้นที่สี่เหลี่ยมผืนผ้า = ', area, 'ตารางหน่วย') # แสดงผลลัพธ์ตัวแปร area

```

กรุณาป้อนความยาวของสี่เหลี่ยมผืนผ้า: 12
 กรุณาป้อนความกว้างของสี่เหลี่ยมผืนผ้า: 5.5
 พื้นที่สี่เหลี่ยมผืนผ้า = 66.0 ตารางหน่วย

**ตัวอย่าง 6.2**

การเขียนคำสั่งโปรแกรมควบคุมทิศทางการทำงานแบบลำดับคำนวณรายได้สุทธิ

```

1  base_salary = float(input('กรอกฐานเงินเดือน: '))
2  ot_hr = float(input('กรอกจำนวนชั่วโมงทำงานล่วงเวลา: '))
3  ot_salary = ot_hr * 200
4  bonus = base_salary * 0.025
5  salary = base_salary + ot_salary + bonus
6  print(f'ค่าทำงานล่วงเวลา {ot_hr} * 200 = {ot_salary:.2f} บาท')
7  print(f'โบนัส {base_salary} * 0.025 = {bonus:.2f} บาท')
8  print(f'เงินเดือนสุทธิ = {salary:.2f} + {ot_salary:.2f} + {bonus:.2f}
    ↳ = {salary:.2f} บาท')

```

กรอกฐานเงินเดือน: 15000
 กรอกจำนวนชั่วโมงทำงานล่วงเวลา: 20
 ค่าทำงานล่วงเวลา 20.0 * 200 = 4000.00 บาท
 โบนัส 15000.0 * 0.025 = 375.00 บาท
 เงินเดือนสุทธิ = 19375.00 + 4000.00 + 375.00 = 19375.00 บาท



6.3 การเขียนคำสั่งโปรแกรมควบคุมการทำงานแบบมีเงื่อนไข

การเขียนคำสั่งโปรแกรมควบคุมการทำงานแบบมีเงื่อนไข (Condition Control Statement) ใช้เพื่อควบคุมทิศทางการทำงานหรือกระบวนการทำงานต่าง ๆ ของโปรแกรมให้เป็นไปตามเงื่อนไข โดยคำสั่งโปรแกรมจะให้มีการตัดสินใจเลือกทิศทางการทำงานอย่างใดอย่างหนึ่งตามเงื่อนไขที่กำหนดก่อนทำงานอย่างอื่นต่อไป สำหรับในภาษาไพธอนจะใช้คำสั่ง **if** ที่มีการใช้งานอยู่ 4 แบบ ได้แก่

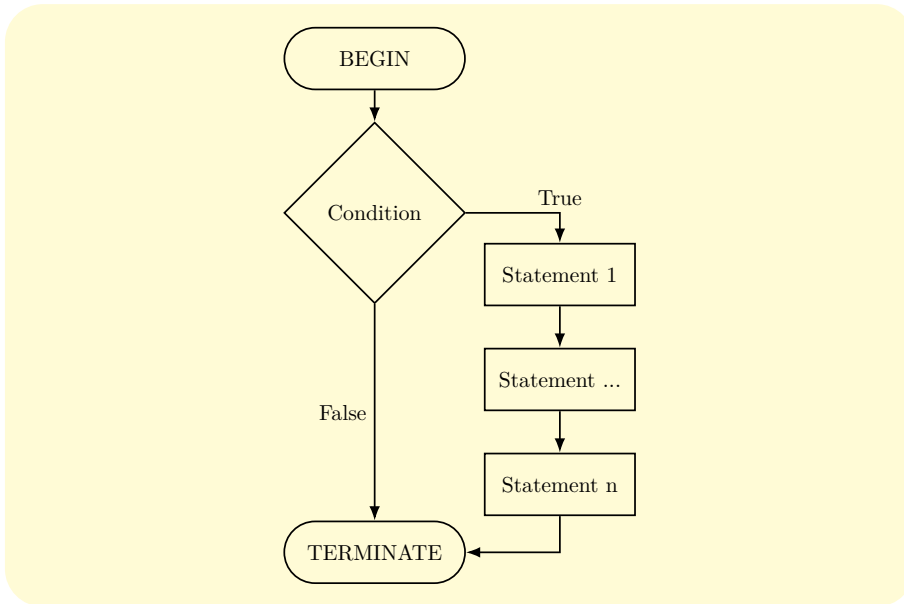
1. **if**,
2. **if ... else**,
3. **if ... elif ... else** และ
4. **if** ซ้อนกัน (Nested If)

โดยรายละเอียดการใช้งานคำสั่งแต่ละแบบ จะถูกกล่าวในหัวข้อย่อยถัดไป ดังนี้

6.3.1 คำสั่ง **if**

คำสั่ง **if** ใช้ตรวจสอบเงื่อนไขจากนิพจน์ (condition) ที่สร้างขึ้นมาว่าเป็นจริงหรือเท็จ ถ้าผลการตรวจสอบเงื่อนไขนิพจน์เป็น **True** คำสั่งโปรแกรมที่อยู่ในขอบเขตคำสั่ง **if** จะทำงาน ถ้าผลการตรวจสอบเงื่อนไขนิพจน์เป็นเท็จ **False** คำสั่งโปรแกรมที่อยู่นอกขอบเขตคำสั่ง **if** จะทำงาน และอย่าลืมใส่เครื่องหมาย **colon(:)** ปิดท้ายคำสั่งเงื่อนไขคำสั่ง **if** เสมอ และต้องย่อหน้า เพื่อเป็นการกำหนดขอบเขตการทำงานของคำสั่งโปรแกรมด้วย

ในการกำหนดขอบเขตคำสั่ง รูปแบบการเขียนคำสั่ง **if** มีลักษณะโครงสร้างโปรแกรม และผังงานดังต่อไปนี้



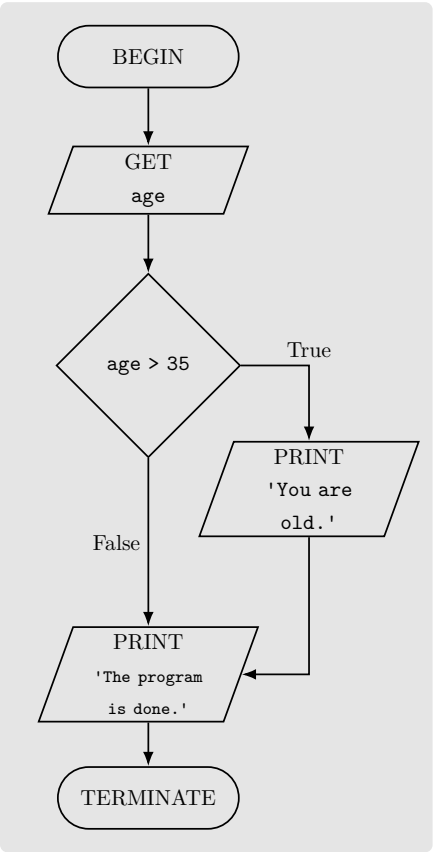
รูป 6.3: ผังงานการทำงานแบบมีเงื่อนไข

โครงสร้างโปรแกรมควบคุมการทำงานแบบมีเงื่อนไข **if**

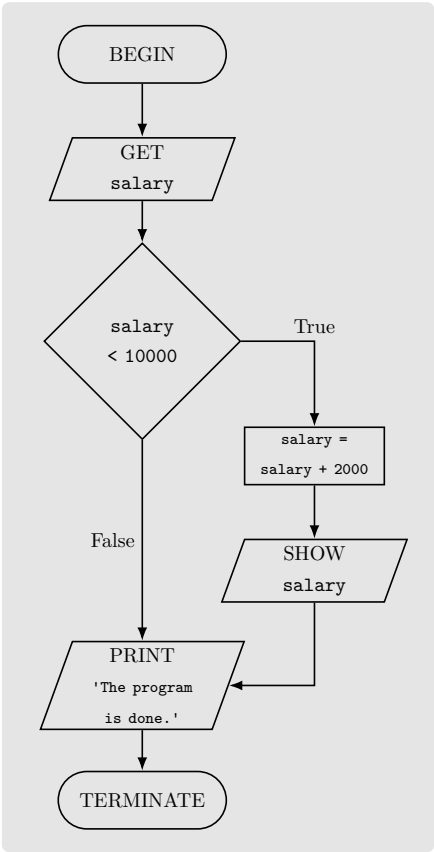
```

if condition:      # ตรวจสอบเงื่อนไขเป็นจริงหรือเท็จ
    statement_1    # ชุดคำสั่งที่ 1 เมื่อเงื่อนไขเป็นจริง
    ...
    statement_n    # ชุดคำสั่งที่ n เมื่อเงื่อนไขเป็นจริง
  
```

จากแผนภาพผังงานในรูป 6.4a และ 6.4b นำมาเขียนเป็นคำสั่งโปรแกรมควบคุมทิศทางการทำงานแบบมีเงื่อนไข **if** ในตัวอย่าง 6.3 และ 6.4 ได้ดังนี้



(a) ตัวอย่าง 6.3



(b) ตัวอย่าง 6.4

รูป 6.4: ตัวอย่างผังงานคำสั่งควบคุมทิศทางแบบมีเงื่อนไข

ตัวอย่าง 6.3การเขียนคำสั่งโปรแกรมแบบมีเงื่อนไขด้วย `if` ตรวจสอบอายุ

```
1 age = int(input('กรุณาป้อนอายุของคุณ = ')) # รอรับค่าการป้อนข้อมูลเป็นจำนวนเต็ม
2 if age > 35: # ตรวจสอบค่าตัวแปร age มากกว่า 35 หรือไม่
3     print('You are old.') # แสดงผลลัพธ์
4 print('The program is done !!') # แสดงผลหลังจากโปรแกรมทำงานทุกครั้ง
```

กรุณาป้อนอายุของคุณ = 45
You are old.
The program is done !!



กรุณาป้อนอายุของคุณ = 15
The program is done !!

จากตัวอย่าง 6.3 เมื่อผู้ใช้งานป้อนอายุไม่เกินกว่า 35 ทำให้เงื่อนไขเป็นเท็จ โปรแกรมจะแสดงผลคำว่า 'The program is done' เท่านั้น

ตัวอย่าง 6.4การเขียนคำสั่งโปรแกรมแบบมีเงื่อนไขด้วย `if` ตรวจสอบเงินเดือน

```
1 salary = int(input('กรุณาป้อนเงินเดือน = '))
2 if salary < 10000:
3     salary = salary + 2000
4     print('รายได้สุทธิ =', salary)
5 print('The program is done !!')
```

กรุณาป้อนเงินเดือน = 9500
รายได้สุทธิ = 11500
The program is done !!



กรุณาป้อนเงินเดือน = 14500
The program is done !!

จากตัวอย่าง 6.4 เมื่อป้อนเงินเดือนน้อยกว่า 10000 ส่งผลให้เงื่อนไขเป็นจริง ซึ่งจะนำจำนวนเงินเดือนที่ป้อนบวกกับค่าโบนัส 2000 ถ้าป้อนเงินเดือนมากกว่าหรือเท่ากับ 10000 ส่งผลให้เงื่อนไขเป็นเท็จ ดังนั้นจึงไม่นำเงินเดือนที่ป้อนมาบวกกับค่าโบนัส และโปรแกรมจะแสดงผลคำว่า 'The program is done' เท่านั้น

ตัวอย่าง 6.5

การเขียนคำสั่งโปรแกรมแบบมีเงื่อนไขด้วย **if** ตรวจสอบว่าจำนวนเต็มอยู่ในช่วงปิดที่กำหนดไว้

```
1 n = int(input('Enter an integer: ')) # รับค่าจำนวนเต็ม n
2 my_interval = [1, 100] # สร้างตัวแปรชนิดลิสต์ แทนช่วงปิดที่กำหนด
3 # ตรวจสอบเงื่อนไข n มีค่าอยู่ในช่วงปิด [1, 100]
4 if n >= my_interval[0] and n <= my_interval[1]:
5     print(f'{n} is a member of {my_interval}') # การกระทำเมื่อเงื่อนไข
6     # เป็นจริง
7 print('Program terminated')
```

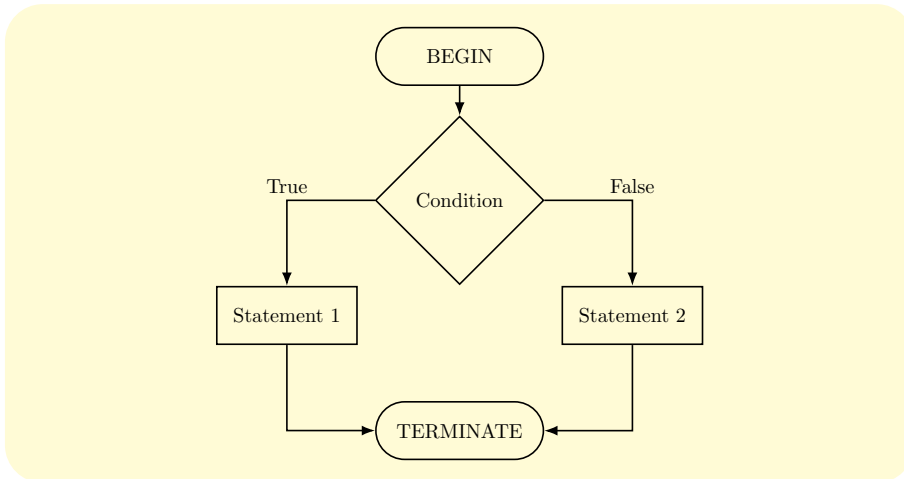
```
Enter an integer: 55
55 is a member of [1, 100]
Program terminated
```

```
Enter an integer: 555
Program terminated
```



6.3.2 คำสั่ง **if ... else**

คำสั่ง **if ... else** ควบคุมการทำงานแบบมีเงื่อนไข 2 ทิศทาง ซึ่งจะมีการตรวจสอบเงื่อนไขที่ถูกกำหนดขึ้นในนิพจน์เหมือนกันกับคำสั่ง **if** ว่าเป็นจริง (**True**) หรือเท็จ (**False**) ถ้าผลเป็นจริงจะทำงานตามคำสั่งที่อยู่หลัง **if** ถ้าผลเป็นเท็จจะทำงานตามคำสั่งที่อยู่หลัง **if** รูปแบบการเขียนคำสั่งโปรแกรมแบบ **if ... else** มีลักษณะดังต่อไปนี้ (ต้องใส่เครื่องหมาย Colon (:) หลังเงื่อนไข **if** และคำสั่ง **else** ด้วย)



รูป 6.5: ผังงานของการทำงานแบบมีเงื่อนไข

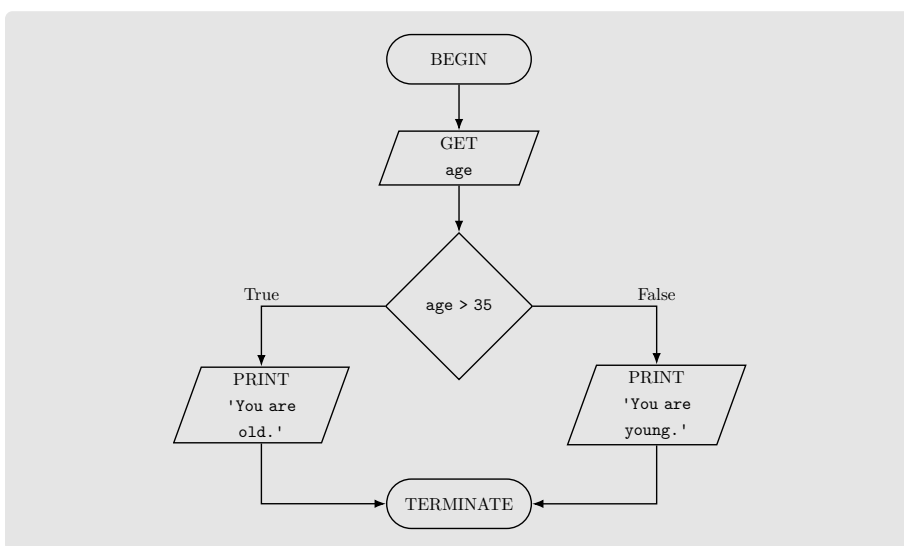
โครงสร้างโปรแกรมควบคุมการทำงานแบบมีเงื่อนไข **if...else**

```

if condition:      # ตรวจสอบเงื่อนไขเป็นจริงหรือเท็จ
    statement_1    # คำสั่งที่ให้ทำงานเมื่อเงื่อนไขเป็นจริง
else:
    statement_2    # คำสั่งที่ให้ทำงานเมื่อเงื่อนไขเป็นเท็จ
  
```

จากโครงสร้างของคำสั่งแบบมีเงื่อนไข **if...else** นำมาเขียนเป็นแผนภาพผังงานได้ดังรูป 6.5 และแสดงตัวอย่างการเขียนผังงานตรวจสอบอายุ ดังรูป 6.6 กำหนดเงื่อนไขคือ ถ้าอายุน้อยกว่า 35 ให้แสดงคำว่า 'You are young' ถ้าอายุมากกว่า 35 ให้แสดงคำว่า 'You are old'

เมื่อนำมาเขียนเป็นคำสั่งโปรแกรมแบบ **if...else** โดยนำโปรแกรมจากตัวอย่าง 6.3 มาปรับปรุงเพิ่มเติม แสดงได้ดังตัวอย่าง 6.7



รูป 6.6: ฟังก์ชันของตัวอย่าง 6.7

ตัวอย่าง 6.6

การเขียนคำสั่งโปรแกรมตรวจสอบเงื่อนไขด้วยคำสั่ง `if...else` ตรวจสอบอายุ

```
1 age = int(input('กรุณาป้อนอายุของคุณ = ')) # รอรับการป้อนข้อมูลเป็นจำนวนเต็ม
2 if age > 35: # ตรวจสอบค่าตัวแปร age มากกว่า 35 หรือไม่
3     print('You are old.') # แสดงผลลัพธ์
4 else: # ตรวจสอบแล้วเป็นเท็จคือค่าตัวแปร age น้อยกว่า 35
5     print('You are young.')# แสดงผลลัพธ์
```

กรุณาป้อนอายุของคุณ = 57
You are old.



กรุณาป้อนอายุของคุณ = 25
You are young.

ตัวอย่าง 6.7

การเขียนคำสั่งโปรแกรมตรวจสอบเงื่อนไขด้วยคำสั่ง `if...else` ตรวจสอบชนิดของจำนวนเต็ม

```
1 n = int(input('Enter an integer: ')) # รับค่าจำนวนเต็ม n
2 m = n % 2 # คำนวณเศษจากการหาร n ด้วย 2
3 if m == 0: # ตรวจสอบเงื่อนไขการหารลงตัว
4     print(f'{n} is even') # การกระทำเมื่อเงื่อนไขเป็นจริง
5 else:
6     print(f'{n} is odd') # การกระทำเมื่อเงื่อนไขเป็นเท็จ
```

Enter an integer: 30
30 is even



Enter an integer: 447
447 is odd

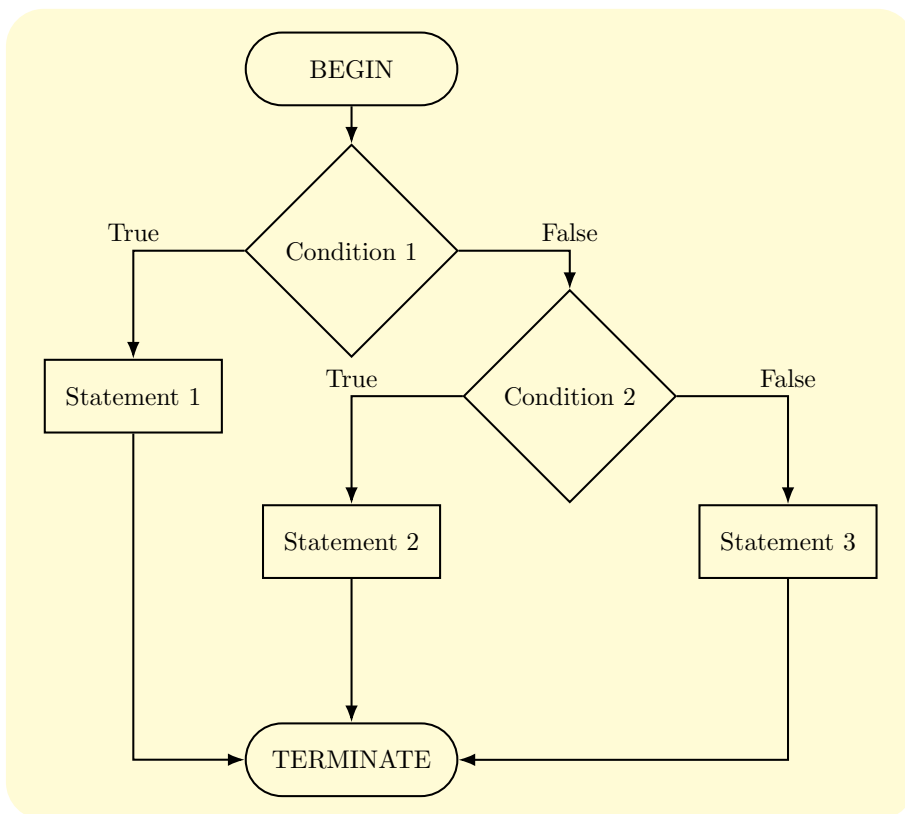
6.3.3 คำสั่ง `if ... elif ... else`

คำสั่ง `if ... elif ... else` ควบคุมทิศทางการทำงานของโปรแกรมแบบมีหลายเงื่อนไข โดยมีการสร้างนิพจน์เป็นตัวตรวจสอบเงื่อนไขเหมือนกับคำสั่ง `if` และ `if...else` แต่คำสั่ง `if ... elif ... else` สามารถตรวจสอบได้หลายเงื่อนไขมากกว่า โดยมีลักษณะการตรวจสอบเงื่อนไขไปเรื่อย ๆ จนกว่าจะพบเงื่อนไขที่เป็นจริงแล้วจึงแสดงผลตามคำสั่งโปรแกรมที่กำหนด ถ้าตรวจสอบเงื่อนไขนิพจน์แล้วเป็นเท็จ จะทำงานตามคำสั่งโปรแกรมที่อยู่หลัง `else` มีโครงสร้างรูปแบบการเขียนคำสั่งโปรแกรมแสดงดังต่อไปนี้

โครงสร้างรูปแบบการเขียนคำสั่ง `if ... elif ... else`

```
if condition_1:    # ตรวจสอบเงื่อนไข condition_1
    statement_1    # คำสั่งเมื่อ condition_1 เป็นจริง
elif condition_2:  # ตรวจสอบเงื่อนไข condition_2
    statement_2    # คำสั่งเมื่อ condition_2 เป็นจริง
else:
    statement_3    # คำสั่งเมื่อเงื่อนไขด้านบนทั้งหมดเป็นเท็จ
```

จากโครงสร้างรูปแบบการเขียนคำสั่งควบคุมทิศทางการทำงานแบบมีเงื่อนไข `if ... elif ... else` เมื่อนำมาเขียนเป็นผังงานจะได้ดังรูป 6.7 และตัวอย่างต่อไปนี้เป็นคำสั่งโปรแกรมแสดงการทำงานแบบมีเงื่อนไข `if ... elif ... else` ซึ่งรูป 6.8 แสดงผังงานของการเขียนคำสั่งโปรแกรมควบคุมทิศทางแบบมีเงื่อนไข `if ... elif ... else` ตรวจสอบขนาดไซส์เสื้อในตัวอย่าง 6.11



รูป 6.7: ผังงานของการทำงานแบบมีเงื่อนไข `if ... elif ... else`

ตัวอย่าง 6.8

การเขียนคำสั่งโปรแกรมควบคุมทิศทางการทำงานแบบมีเงื่อนไข

`if ... elif ... else`

```
1 a = 200
2 b = 33
3 if b > a:
4     print('b is greater than a')
5 elif a == b:
6     print('a and b are equal')
7 else:
8     print('a is greater than b')
```

a is greater than b



ตัวอย่าง 6.9

การเขียนคำสั่งโปรแกรมควบคุมทิศทางการทำงานแบบมีเงื่อนไข `if ... elif`

```
1 a = 33
2 b = 33
3 if b > a:
4     print('b is greater than a')
5 elif a == b:
6     print('a and b are equal')
```

a and b are equal



ตัวอย่าง 6.10

การเขียนคำสั่งโปรแกรมควบคุมทิศทางแบบมีเงื่อนไข

`if ... elif ... else` ในการตรวจสอบชนิดของจำนวนเต็ม

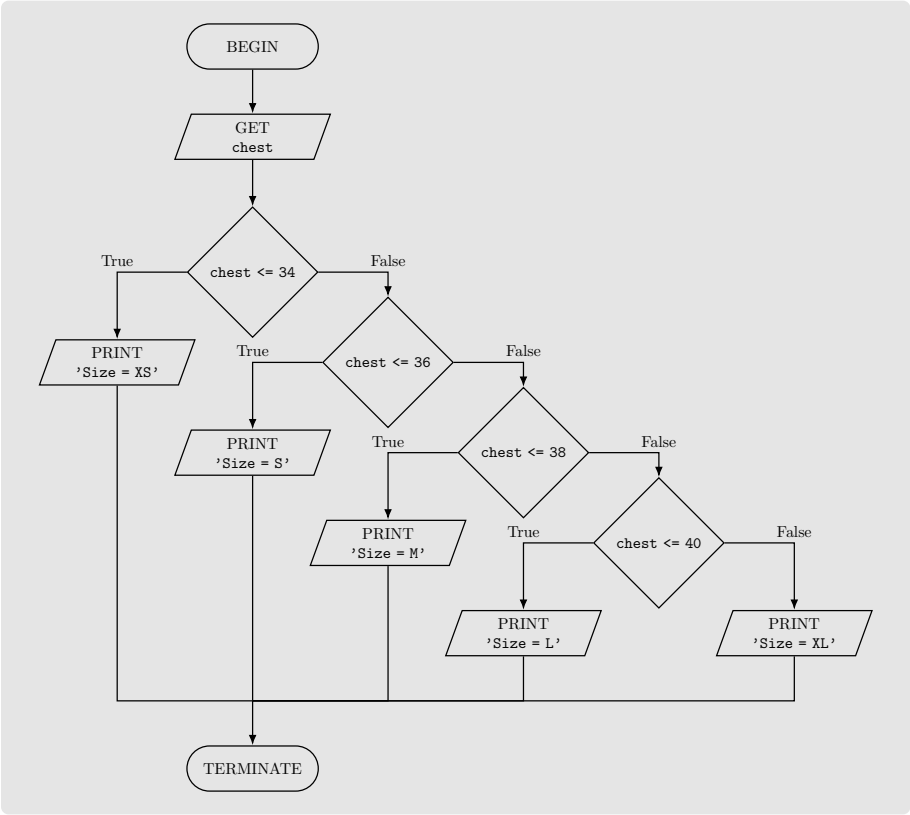
```
1 n = int(input('Enter an integer: '))
2 if n > 0:
3     print(f'{n} is positive')
4 elif n < 0:
5     print(f'{n} is negative')
6 else:
7     print(f'{n} is Zero')
```

Enter an integer: -345
-345 is negative

Enter an integer: 222
222 is positive

Enter an integer: 0
0 is Zero





รูป 6.8: ผังงานของตัวอย่าง 6.11

ตัวอย่าง 6.11

การเขียนคำสั่งโปรแกรมควบคุมทิศทางแบบมีเงื่อนไข

if ... elif ... else ตรวจสอบขนาดไซส์เสื้อ

```

1 chest = int(input('กรุณาป้อนความกว้างอก: ')) # รอรับการป้อนข้อมูลเป็นจำนวนเต็ม
2 if chest <= 34:                                # ตรวจสอบค่าตัวแปร chest ไม่เกินกว่า 34 หรือไม่
3     print('Size = XS')                          # แสดงผลลัพธ์ ถ้าเงื่อนไขบรรทัดที่ 2 เป็นจริง
4 elif chest <= 36:                               # ตรวจสอบค่าตัวแปร chest ไม่เกินกว่า 36 หรือไม่
5     print('Size = S')                           # แสดงผลลัพธ์ ถ้าเงื่อนไขบรรทัดที่ 4 เป็นจริง
6 elif chest <= 38:                               # ตรวจสอบค่าตัวแปร chest ไม่เกินกว่า 38 หรือไม่
7     print('Size = M')                           # แสดงผลลัพธ์ ถ้าเงื่อนไขบรรทัดที่ 6 เป็นจริง
8 elif chest <= 40:                               # ตรวจสอบค่าตัวแปร chest ไม่เกินกว่า 40 หรือไม่
9     print('Size = L')                           # แสดงผลลัพธ์ ถ้าเงื่อนไขบรรทัดที่ 8 เป็นจริง
10 else:                                           # ถ้าค่าตัวแปร chest มากกว่า 40
11     print('Size = XL')                         # แสดงผลลัพธ์ ถ้าไม่ตรงกับเงื่อนไขใดๆ

```

กรุณาป้อนความกว้างอก: 50
Size = XL

กรุณาป้อนความกว้างอก: 37
Size = M

กรุณาป้อนความกว้างอก: 35
Size = S

กรุณาป้อนความกว้างอก: 40
Size = L

กรุณาป้อนความกว้างอก: 30
Size = XS

**6.3.4 คำสั่ง if ซ้อนกัน**

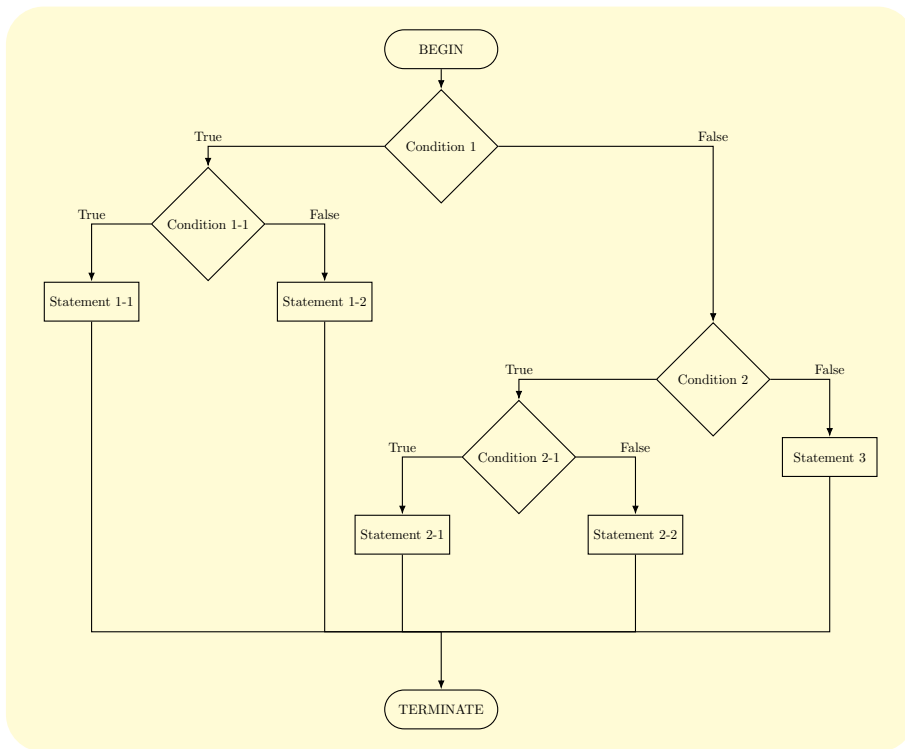
คำสั่ง **if** ซ้อนกัน (Nested If) ใช้ควบคุมทิศทางการทำงานของโปรแกรมแบบเงื่อนไขซ้อนเงื่อนไข หรือ **if** ซ้อน **if** นั่นเอง ซึ่งจะช่วยให้เราเขียนโปรแกรมสำหรับตรวจสอบเงื่อนไขที่มี

ความซับซ้อนได้มากขึ้น ลักษณะการทำงานของคำสั่ง Nested If ถ้าเงื่อนไขขอบเขตนอกเป็นจริงแล้วจะเข้าไปตรวจสอบเงื่อนไขที่อยู่ภายในเรื่อย ๆ จนกว่าจะพบเงื่อนไขที่เป็นจริงหรือเป็นตามที่กำหนดถึงจะหยุดการทำงาน โครงสร้างรูปแบบการเขียนคำสั่งโปรแกรมแบบ Nested If แสดงดังนี้

โครงสร้างโปรแกรมควบคุมการทำงานแบบมีเงื่อนไข Nested If

```
if condition_1:
    if condition_1_1:
        statement_1_1
    else:
        statement_1_2
elif condition_2:
    if condition_2_1:
        statement_2_1
    else:
        statement_2_2
else:
    statement_3
```

จากโครงสร้างโปรแกรมควบคุมทิศทางการทำงานแบบมีเงื่อนไข Nested If เมื่อนำมาเขียนเป็นผังงานแสดงขั้นตอนการทำงานจะได้ดังรูป 6.9 และในตัวอย่าง 6.12 แสดงการเขียนคำสั่งโปรแกรมการตัดสินใจซื้อสมาร์ทโฟนภายใต้เงื่อนไขของจำนวนเงินที่มีอยู่



รูป 6.9: ผังงานการทำงานแบบมีเงื่อนไข Nested If

ตัวอย่าง 6.12

การเขียนคำสั่งโปรแกรมควบคุมทิศทางแบบมีเงื่อนไข Nested If ในการตัดสินใจซื้อสมาร์ทโฟนภายใต้เงื่อนไขของจำนวนเงินที่มีอยู่

```

1 money = float(input('กรุณป้อนจำนวนเงิน = ')) # กรอกจำนวนเงินที่มีอยู่
2 if money >= 27000: # ตรวจสอบค่า money มากกว่า 27000 หรือไม่
3     if money >= 35000: # ตรวจสอบค่า money มากกว่า 35000 หรือไม่
4         print('Buy iPhone 12 Pro Max') # แสดงผลถ้าบรรทัดที่ 3 เป็นจริง
5     else:
6         print('Buy iPhone 12 Pro') # แสดงผลถ้าบรรทัดที่ 3 เป็นเท็จ
7 elif money >= 20000: # ตรวจสอบค่า money มากกว่า 20000 หรือไม่
8     if money >= 25000: # ตรวจสอบค่า money มากกว่า 25000 หรือไม่
9         print('Buy iPhone 12') # แสดงผลถ้าบรรทัดที่ 8 เป็นจริง
10    else:
11        print('Buy iPhone SE') # แสดงผลถ้าบรรทัดที่ 8 เป็นเท็จ
12 else:
13    print('Can not buy a new iPhone.') # แสดงผลเมื่อเงื่อนไขด้านบนทั้งหมด
    → เป็นเท็จ

```

กรุณป้อนจำนวนเงิน = 40000
Buy iPhone 12 Pro Max

กรุณป้อนจำนวนเงิน = 22000
Buy iPhone SE

กรุณป้อนจำนวนเงิน = 4000
Can not buy a new iPhone



6.4 คำสั่งโปรแกรมควบคุมการทำงานแบบทำซ้ำ

การเขียนคำสั่งโปรแกรมควบคุมการทำงานแบบทำซ้ำ (Iteration Control Statement) ในภาษาไพธอนมีอยู่ด้วยกัน 2 คำสั่ง ได้แก่ คำสั่ง **while** และ คำสั่ง **for** สำหรับคำสั่ง **while** ต้องตรวจสอบเงื่อนไขก่อนการทำงาน ถ้าเป็นเท็จจะไม่ทำงาน ถ้าเป็นจริงจะทำงานไปจนกว่า

เงื่อนไขจะเป็นเท็จถึงจะหยุดการทำงาน กรณีคำสั่ง **for** ในภาษาไพธอนนำมาใช้อ่านค่าข้อมูลหรือชนิดข้อมูลแบบลำดับ ซึ่งจะอ่านข้อมูลที่ถูกเก็บไว้ในตัวแปรจนกว่าจะหมดถึงหยุดการทำงาน

6.4.1 คำสั่งวนซ้ำด้วย **while**

คำสั่ง **while** เป็นคำสั่งที่กำหนดให้โปรแกรมทำซ้ำตามเงื่อนไขในนิพจน์ ก่อนคำสั่งโปรแกรมที่อยู่หลังคำสั่ง **while** จะทำงานได้จะมีการตรวจสอบเงื่อนไขก่อนทุกครั้ง เมื่อผลตรวจสอบเงื่อนไขเป็นจริงคำสั่งหลัง **while** ถึงจะทำงาน ถ้าผลการตรวจสอบเงื่อนไขเป็นเท็จ โปรแกรมจะออกจากการทำซ้ำและไปทำงานตามคำสั่งอื่น ๆ ตามที่ได้กำหนดไว้ คำสั่ง **while** เหมาะกับงานที่มีการทำซ้ำที่ไม่แน่นอน เมื่อผู้อ่านนำคำสั่ง **while** ไปใช้งานต้องปิดท้ายคำสั่งด้วยเครื่องหมาย colon (:) เสมอ เหมือนกับคำสั่ง **if** มีโครงสร้างรูปแบบการเขียนคำสั่ง **while** แสดงดังรูปต่อไปนี้

โครงสร้างการเขียนคำสั่งโปรแกรมควบคุมทิศทางการทำงานแบบทำซ้ำด้วย **while**

```
while condition:    # เงื่อนไขที่ใช้ตรวจสอบเป็นจริงหรือเท็จ
    statement        # คำสั่งโปรแกรมให้ทำงานเมื่อเงื่อนไขเป็นจริง
```

จากโครงสร้างโปรแกรมควบคุมการทำงานแบบทำซ้ำด้วย **while** เมื่อนำมาเขียนเป็นผังงานจะได้โค้ดดังรูป และตัวอย่างผังงานการแสดงผล 'Hello' ออกทางหน้าจอ จำนวน 5 ครั้ง

ผังงานของโปรแกรมควบคุมการทำงานแบบวนซ้ำด้วย **while**

เมื่อนำผังงานมาเขียนโปรแกรมให้แสดงผลคำว่า 'Hello' จำนวน 5 ครั้ง ออกทางหน้าจอ จะได้ดังตัวอย่างต่อไปนี้

ตัวอย่าง 6.13

การเขียนคำสั่งโปรแกรมควบคุมการทำงานแบบทำซ้ำด้วยคำสั่ง `while`

```
1 i = 1 # สร้างตัวแปรและกำหนดค่าเริ่มต้นในการตรวจสอบเงื่อนไข
2 while i <= 5: # ตรวจสอบค่าตัวแปร i น้อยกว่าหรือเท่ากับ 5 หรือไม่
3     print('Hello') # ถ้าเงื่อนไขเป็นจริง ให้แสดงผลลัพธ์
4     i = i + 1 # เพิ่มค่าตัวแปร i ด้วยการบวก 1
```

Hello
Hello
Hello
Hello
Hello



เริ่มต้นตัวแปร `i` ถูกกำหนดค่าเป็น `1` ในแต่ละรอบโปรแกรมจะตรวจสอบค่าตัวแปร `i` ว่าน้อยกว่าหรือเท่ากับ `5` หรือไม่ ถ้ามีค่าน้อยกว่าหรือเท่ากับ `5` จะแสดงผลคำว่า `'Hello'` ออกทางหน้าจอ และเพิ่มค่าตัวแปร `i` ด้วยการนำตัวแปร `i` บวก `1` โปรแกรมจะทำซ้ำจนกว่าตัวแปร `i` มีค่ามากกว่า `5` ถึงจะหยุดการทำงาน

ตัวอย่าง 6.14

การเขียนคำสั่งโปรแกรมควบคุมการทำงานแบบทำซ้ำด้วยคำสั่ง `while` แสดงตัวเลขที่ผู้ใช้งานป้อนจำนวน 5 ครั้ง และแสดงค่าตัวเลขน้อยที่สุด ค่าตัวเลขมากที่สุด และผลรวมที่ป้อนทั้งหมด

```

1 i = 1 # สร้างตัวแปรและกำหนดค่าเริ่มต้นในการตรวจสอบเงื่อนไข
2 num = [] # สร้างตัวแปรเป็นชนิดข้อมูลลิสต์
3 while i <= 5: # ตรวจสอบค่าตัวแปร i น้อยกว่าหรือเท่ากับ 5 หรือไม่
4     n = int(input(f'ป้อนตัวเลขจำนวนเต็ม ครั้งที่ {i} = ')) # ถ้าเป็นจริง จะให้
    ↳ ป้อนจำนวนเต็ม
5     num.append(n) # เพิ่มค่าตัวแปร n เก็บไว้ในลิสต์ num
6     i = i + 1 # เพิ่มค่าตัวแปร i ด้วยการบวก 1
7 print(f'ตัวเลขที่คุณป้อน = {num}') # แสดงผลจากค่าลิสต์ num
8 print(f'ตัวเลขค่าน้อยที่สุด = {min(num)}') # แสดงผลค่าน้อยที่สุดจากลิสต์ num
9 print(f'ตัวเลขค่ามากที่สุด = {max(num)}') # แสดงผลค่ามากที่สุดจากลิสต์ num
10 print(f'ผลบวกตัวเลขทั้งหมด = {sum(num)}') # แสดงผลผลรวมจากลิสต์ num

```

ป้อนตัวเลขจำนวนเต็ม ครั้งที่ 1 = 100
 ป้อนตัวเลขจำนวนเต็ม ครั้งที่ 2 = 50
 ป้อนตัวเลขจำนวนเต็ม ครั้งที่ 3 = -50
 ป้อนตัวเลขจำนวนเต็ม ครั้งที่ 4 = 0
 ป้อนตัวเลขจำนวนเต็ม ครั้งที่ 5 = 99
 ตัวเลขที่คุณป้อน = [100, 50, -50, 0, 99]
 ตัวเลขค่าน้อยที่สุด = -50
 ตัวเลขค่ามากที่สุด = 100
 ผลบวกตัวเลขทั้งหมด = 199



เมื่อโปรแกรมทำงานรอบที่ 1 เสร็จ ค่าตัวแปร `i` ถูกเพิ่มค่าเป็น 2 จากนั้นเมื่อเริ่มทำงานในรอบที่ 2 ค่าตัวแปร `i` จะด้วยคำสั่ง `while` อีกครั้ง โปรแกรมจะทำซ้ำจนกว่าค่าตัวแปร `i` มีค่ามากกว่า 5 ถึงจะหยุดให้ผู้ใช้งานป้อนตัวเลข

ในบางครั้งเมื่อนำคำสั่ง `while` มาใช้งานจะเกิดเหตุการณ์หนึ่งเกิดขึ้นเรียกว่า การทำซ้ำไม่รู้จบ (Infinite Loop) ซึ่งเกิดขึ้นจากการกำหนดเงื่อนไขที่ผิดพลาดของผู้เขียนโปรแกรม

เอง ทำให้โปรแกรมตรวจสอบไม่พบเงื่อนไขที่เป็นเท็จ ดังนั้นผู้เขียนโปรแกรมควรตรวจสอบการกำหนดเงื่อนไขให้ถูกต้องก่อนสั่งให้โปรแกรมทำงานเมื่อกำหนดเงื่อนไขไม่ถูกต้องจะทำให้เกิดการทำงานซ้ำไม่รู้จบ ดังตัวอย่างต่อไปนี้

ตัวอย่าง 6.15

```

1 i = 1
2 num = []
3 while i > 0:
4     n = float(input('กรุณาป้อนตัวเลข : '))
5     num.append(n)
6     i = i + 1
7 print(f'จำนวนตัวเลขที่คุณป้อน : {num}')
```

กรุณาป้อนตัวเลข : 55
 กรุณาป้อนตัวเลข : 34
 กรุณาป้อนตัวเลข : 44.5
 กรุณาป้อนตัวเลข : 90
 กรุณาป้อนตัวเลข : 12
 ...
 กรุณาป้อนตัวเลข :



การทำงานซ้ำของคำสั่ง `while` แบบไม่รู้จบ (Infinite loop) จากตัวอย่างโปรแกรมคำสั่งในบรรทัดที่ 1 กำหนดค่าตัวแปร `i` เท่ากับ `1` และเมื่อนำค่าตัวแปร `i` ไปตรวจสอบเงื่อนไขด้วยคำสั่ง `while i > 0` ในบรรทัดที่ 3 จะเห็นว่าเงื่อนไขนี้เป็นจริงเสมอ เมื่อทำงานรอบแรกเสร็จตัวแปร `i` ถูกเพิ่มค่าอีก `1` ตามคำสั่งในบรรทัดที่ 6 เมื่อนำตัวแปร `i` ไปตรวจสอบเงื่อนไขอีกครั้งในบรรทัดที่ 3 เงื่อนไขนี้ก็เป็นจริงอีก และจะเป็นจริงอย่างนี้ตลอดไป ทำให้ผู้ใช้งานต้องป้อนจำนวนตัวเลขไม่มีที่สิ้นสุด ผู้ใช้ต้องป้อนคำสั่ง `KeyboardInterrupt` (`Ctrl + C` หรือ `Ctrl + D`) หากต้องการให้โปรแกรมหยุดการทำงาน

6.4.2 การใช้คำสั่ง `else` ร่วมกับคำสั่งวนซ้ำ `while`

ในภาษาไพธอนได้นำคำสั่ง `else` เข้ามาร่วมทำงานกับคำสั่งวนซ้ำ `while` ด้วย โดยคำสั่ง `else` จะได้รับการประมวลผลเมื่อมีการตรวจสอบเงื่อนไขคำสั่ง `while` เป็นเท็จ และไม่มี การนำเอาคำสั่ง `break` เข้ามาใช้ ถ้านำเอาคำสั่ง `break` มาใช้งานจะทำให้คำสั่ง `else` ไม่ ถูกประมวลผล สำหรับการใช้คำสั่ง `break` ผู้เขียนจะได้อธิบายการใช้ในหัวข้อถัดไป การใช้คำ สั่ง `else` ร่วมกับคำสั่ง `while` แสดงดังตัวอย่างต่อไปนี้

ตัวอย่าง 6.16

การใช้คำสั่ง `else` ร่วมทำงานกับคำสั่ง `while`

```
1 i = 1
2 while i <= 10:
3     print(i)
4     i = i + 2
5 else:
6     print('Done !!') # แสดงผลเมื่อบรรทัดที่ 2 เป็นเท็จ คือ i มากกว่าหรือเท่า 10
```

```
1
3
5
7
9
Done !!
```



6.4.3 คำสั่งวนซ้ำด้วย `for`

คำสั่ง `for` เป็นคำสั่งควบคุมการทำงานแบบวนซ้ำด้วยจำนวนรอบที่แน่นอน ถ้าผลการตรวจสอบเงื่อนไขเป็นจริงจะแสดงผลคำสั่งโปรแกรมที่อยู่หลัง `for` หากเงื่อนไขเป็นเท็จจะออกจากการวนซ้ำ และทำงานในคำสั่งอื่นถัดไป คำสั่ง `for` ในภาษาไพธอนถูกนำมาใช้สำหรับอ่านค่า ข้อมูลที่เก็บแบบเรียงลำดับหรือชนิดข้อมูลแบบเรียงลำดับ เช่น ลิสต์ ทูเพิล เซต อักขระหรือ

สตริง เป็นต้น หรือนำมาใช้อ่านค่าข้อมูลที่ถูกสร้างขึ้นได้ด้วยฟังก์ชัน `range()` มีรูปแบบดังต่อไปนี้

```
for seq_var in sequence:  
    statement
```

`seq_var` ตัวแปรที่คอยรับค่าจาก `sequence`

`sequence` ข้อมูลหรือชนิดข้อมูลแบบเรียงลำดับ

`statement` คำสั่งโปรแกรมที่ต้องการให้ทำงาน อาจจะมีมากกว่า 1 คำสั่ง

จากโครงสร้างโปรแกรมควบคุมการทำงานแบบวนซ้ำ `for` นำมาเขียนเป็นผังงานได้ดังรูป

ผังงานของโปรแกรมควบคุมการทำงานแบบวนซ้ำด้วย `for`

ซึ่งจะเขียนเป็นโปรแกรมควบคุมการทำงานแบบวนซ้ำด้วย `for` กับชนิดข้อมูลต่าง ๆ ทั้งลิสต์ สตริง เซต และผลลัพธ์จากฟังก์ชัน `range()` ได้ดังตัวอย่างต่อไปนี้

ตัวอย่าง 6.17

การเขียนคำสั่งโปรแกรมควบคุมการทำงานแบบวนซ้ำ for กับข้อมูลแบบลำดับ

```

1  # ทำซ้ำชนิดข้อมูลลิสต์
2  my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
3  for num in my_list:
4      print(num, end = ', ')
5  print('\n' + '-' * 20)
6  # ทำซ้ำชนิดข้อมูลสตริง
7  my_string = 'Python Programming'
8  for char in my_string:
9      print(char, end = ', ')
10 print('\n' + '-' * 20)
11 # ทำซ้ำชนิดข้อมูลเซต
12 my_set = {2, 5, 7, 5, 8, 7, 9}
13 for m in my_set:
14     print(m, end = ', ')
15 print('\n' + '-' * 20)
16 # ทำซ้ำจากฟังก์ชัน range()
17 for n in range(1, 10):
18     print(n, end = ', ')
19 print('\n' + '-' * 20)

```

```

1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
-----

```

```

P, y, t, h, o, n, , P, r, o, g, r, a, m, m, i, n, g,
-----

```

```

2, 5, 7, 8, 9,
-----

```

```

1, 2, 3, 4, 5, 6, 7, 8, 9,
-----

```



คำสั่ง **for** อ่านค่าตัวแปรทีละค่าในแต่ละรอบ จากนั้นจึงจะนำค่าตัวแปรมาแสดงผล ซึ่งจะทำให้เข้าใจง่ายขึ้นไปเรื่อย ๆ จนกว่าจะอ่านค่าจากตัวแปรครบทุกค่าในลิสต์ โปรแกรมถึงจะหยุดการ

ทำงาน สำหรับฟังก์ชัน `range(x, y)` จะแจกแจงค่าจำนวนเต็มในช่วงที่กำหนดตั้งแต่ `x` จนถึง `y-1` ออกมาเป็นตัว ๆ จนครบ

ตัวอย่าง 6.18

การเขียนโปรแกรมแม่สูตรคูณด้วยคำสั่งทำซ้ำ `for` และฟังก์ชัน `range()` สร้างช่วงตัวเลข 1-12

```
1 i = int(input('Enter an integer: '))
2 print(f'Multiplication Table of {i}')
3 for j in range(1, 13):
4     # สร้างช่วงตัวเลขด้วยฟังก์ชัน range() จากค่าเริ่มต้น โดยค่าสุดท้ายจะลบด้วย 1
5     num = i * j
6     # ค่าตัวแปร j เริ่มที่ 1 และเพิ่มครั้ง 1 จนกว่าจะถึง 12
7     print(f'{i} * {j} = {num}')
8     # แสดงผลค่าตัวแปร i, j และ num ในแต่ละรอบ
```

```
Enter an integer: 9
Multiplication Table of 9
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
9 * 10 = 90
9 * 11 = 99
9 * 12 = 108
```



นอกจากนี้เรายังนำคำสั่งการทำซ้ำมาช่วยในการจัดการกับชนิดข้อมูลได้ ซึ่งในตัวอย่างต่อไปเป็นการแปลงชนิดข้อมูลดิททิกชันนารีเป็นชนิดข้อมูลลิสต์

ตัวอย่าง 6.19

การเขียนคำสั่งโปรแกรมแปลงชนิดข้อมูลดิกชันนารีเป็นชนิดข้อมูลลิสต์

```
1 dct = {1:'A', 2:'E', 3:'I', 4:'O', 5:'U'}
2 dct_lst = []
3 for i, j in dct.items():
4     dct_lst.append((i, j))
5 print(dct_lst)
```

```
[(1, 'A'), (2, 'E'), (3, 'I'), (4, 'O'), (5, 'U')]
```

**6.4.4 การใช้คำสั่ง `else` ร่วมกับคำสั่งทำซ้ำ `for`**

คำสั่ง `else` นอกจากจะนำมาใช้งานร่วมกับคำสั่ง `while` แล้ว ยังนำมาใช้งานร่วมกับคำสั่ง `for` ได้อีกด้วย โดยคำสั่ง `else` จะทำงานก็ต่อเมื่อกระบวนการทำงานของคำสั่ง `for` อ่านค่าข้อมูลแบบเรียงลำดับครบทุกตำแหน่ง และไม่มีการนำคำสั่ง `break` เข้ามาใช้งานภายในคำสั่ง `for` (สำหรับคำสั่ง `break` จะได้อธิบายการใช้งานในหัวข้อถัดไป)

ตัวอย่าง 6.20

การใช้คำสั่ง `else` ร่วมกับคำสั่งทำซ้ำ `for`

```

1 sports = ['Running', 'Swimming', 'Tennis', 'Racing', 'Football']
2 i = 0
3 for s in range(len(sports)):
4     print(f'ชื่อชนิดกีฬาตำแหน่งที่ {i} คือ {sports[s]}')
5     i = i + 1
6 else:
7     print('Done !!') # แสดงผลหลังคำสั่ง for แสดงผลตัวแปร sports ครบทุก
                        ↳ ตำแหน่ง

```

ชื่อชนิดกีฬาตำแหน่งที่ 0 คือ Running
 ชื่อชนิดกีฬาตำแหน่งที่ 1 คือ Swimming
 ชื่อชนิดกีฬาตำแหน่งที่ 2 คือ Tennis
 ชื่อชนิดกีฬาตำแหน่งที่ 3 คือ Racing
 ชื่อชนิดกีฬาตำแหน่งที่ 4 คือ Football
 Done !!



6.5 การใช้คำสั่งทำซ้ำ `while` หรือ `for` ซ้อนกัน

คำสั่ง `while` หรือ `for` สามารถถูกวางซ้อนกันได้ เหมือนกับการใช้คำสั่งแบบมีเงื่อนไข `if` ซ้อน `if` เพื่อช่วยแก้ไขปัญหางานที่มีความซับซ้อนของการทำซ้ำ โดยทำได้ทั้ง `while` ซ้อน `while`, `for` ซ้อน `for` หรือ `for` ซ้อน `while` ก็ได้ ดังตัวอย่างต่อไปนี้

ตัวอย่าง 6.21

การเขียนโปรแกรมแม่สูตรคูณด้วยคำสั่งทำซ้ำ `while` ซ้อน `while` สร้างช่วงตัวเลข 1-12

```

1  i = 1
2  while i <= 5: # คำสั่ง while ด้านนอกตรวจสอบ i น้อยกว่าหรือเท่ากับ 5 หรือไม่
3      print(f'Round {i}: ', end = ' ') # แสดงผลถ้า i น้อยกว่าหรือเท่ากับ 5
4      j = 1
5      while j <= 12: # คำสั่ง while ด้านในตรวจสอบ j น้อยกว่าหรือเท่ากับ 12 หรือไม่
6          print(i * j, end = ' ') # แสดงผลถ้า j น้อยกว่าหรือเท่ากับ 12
7          j += 1 # เพิ่มค่า j อีก 1
8      print(' ') # ขึ้นบรรทัดใหม่
9      i = i + 1 # เพิ่มค่า i อีก 1
10 print('Done')

```

Round 1: 1 2 3 4 5 6 7 8 9 10 11 12
 Round 2: 2 4 6 8 10 12 14 16 18 20 22 24
 Round 3: 3 6 9 12 15 18 21 24 27 30 33 36
 Round 4: 4 8 12 16 20 24 28 32 36 40 44 48
 Round 5: 5 10 15 20 25 30 35 40 45 50 55 60
 Done



ผลลัพธ์จากการทำงานในแต่ละรอบซึ่งมีทั้งหมด 5 รอบ ค่าตัวเลขจะเพิ่มตามจำนวนรอบ เช่น รอบที่ 1 ตัวเลขเพิ่มขึ้นครั้งละ 1, รอบที่ 2 เพิ่มขึ้นครั้งละ 2 สิ้นสุดท้ายเมื่อถึงรอบที่ 5 ตัวเลขจะเพิ่มขึ้นครั้งละ 5

ตัวอย่าง 6.22

การเขียนคำสั่งโปรแกรมแม่สูตรคูณโดยใช้คำสั่ง `for` ซ้อน `for` จากตัวอย่าง

```
1 for i in range(1, 13):
2     for j in range(2, 7):
3         print(f'{j}*{i}={j * i}', end='\t')
4     print()
5 print('Done')
```

2*1=2	3*1=3	4*1=4	5*1=5	6*1=6
2*2=4	3*2=6	4*2=8	5*2=10	6*2=12
2*3=6	3*3=9	4*3=12	5*3=15	6*3=18
2*4=8	3*4=12	4*4=16	5*4=20	6*4=24
2*5=10	3*5=15	4*5=20	5*5=25	6*5=30
2*6=12	3*6=18	4*6=24	5*6=30	6*6=36
2*7=14	3*7=21	4*7=28	5*7=35	6*7=42
2*8=16	3*8=24	4*8=32	5*8=40	6*8=48
2*9=18	3*9=27	4*9=36	5*9=45	6*9=54
2*10=20	3*10=30	4*10=40	5*10=50	6*10=60
2*11=22	3*11=33	4*11=44	5*11=55	6*11=66
2*12=24	3*12=36	4*12=48	5*12=60	6*12=72
Done				



จากการทำงานของโปรแกรมคำสั่ง `for` ซ้อน `for` ผู้อ่านจะพบว่าคำสั่ง `for` ด้านนอก จะทำงานในรอบถัดไปได้ก็ต่อเมื่อคำสั่ง `for` ด้านในอ่านค่าข้อมูลและแสดงผลครบทั้งหมดก่อน จากนั้นคำสั่ง `for` ด้านนอกค่อยยกกลับมาอ่านค่าข้อมูลจนกว่าจะหมด โปรแกรมถึงจะหยุดการทำงาน

ตัวอย่าง 6.23

การเขียนคำสั่งโปรแกรมคำสั่งวนซ้ำ `for` ซ้อน `while` หาคะแนนรวมของนักเรียนแต่ละคน

```

1  # ป้อนจำนวนนักเรียน
2  stu = int(input('ป้อนจำนวนนักเรียน = '))
3  # ทำซ้ำตามจำนวนค่าตัวแปร stu
4  for i in range(1, stu + 1):
5      j = 1 # กำหนดค่าเริ่มต้นเพื่อตรวจสอบคำสั่ง while
6      lst = [] # ตัวแปรลิสต์สำหรับเก็บคะแนน
7      print(f'นักเรียนคนที่ {i}') # จะแสดงผลตามรอบที่ i
8      while j <= 3: # เงื่อนไขทำซ้ำ เมื่อ j น้อยกว่าหรือเท่ากับ 3
9          # ถ้ามากกว่าจะออกจากการทำซ้ำ
10         n = float(input(f'กรอกคะแนนครั้งที่ {j}: ')) # ป้อนคะแนน
11         lst.append(n) # นำค่าตัวแปรเก็บไว้ที่ตัวแปร lst
12         j += 1 # เพิ่มค่าตัวแปร j ด้วยการบวก 1
13     # แสดงผลคะแนนทั้งหมดและคะแนนรวม
14     print(f'นักเรียนคนที่ {i} ได้คะแนน = {lst} \t คะแนนรวม =
    ↳ {sum(lst)}')
```

ป้อนจำนวนนักเรียน = 3

นักเรียนคนที่ 1

กรอกคะแนนครั้งที่ 1: 30

กรอกคะแนนครั้งที่ 2: 23

กรอกคะแนนครั้งที่ 3: 26.5

นักเรียนคนที่ 1 ได้คะแนน = [30.0, 23.0, 26.5] คะแนนรวม = 79.5

นักเรียนคนที่ 2

กรอกคะแนนครั้งที่ 1: 12

กรอกคะแนนครั้งที่ 2: 9.5

กรอกคะแนนครั้งที่ 3: 24

นักเรียนคนที่ 2 ได้คะแนน = [12.0, 9.5, 24.0] คะแนนรวม = 45.5

นักเรียนคนที่ 3

กรอกคะแนนครั้งที่ 1: 33

กรอกคะแนนครั้งที่ 2: 25

กรอกคะแนนครั้งที่ 3: 19.5

นักเรียนคนที่ 3 ได้คะแนน = [33.0, 25.0, 19.5] คะแนนรวม = 77.5



จากตัวอย่างที่ผ่านมาผู้อ่านได้เรียนรู้ถึงการนำคำสั่งการทำซ้ำมาซ้อนกัน นอกจากนี้เรายังสามารถนำเอาคำสั่งแบบมีเงื่อนไข **if** มาซ้อนในคำสั่งการทำซ้ำ **for** และ **while** ได้อีกด้วย เพื่อใช้เป็นเงื่อนไขในการตัดสินใจการทำงานของโปรแกรม ดังตัวอย่างต่อไปนี้

ตัวอย่าง 6.24

การใช้คำสั่งแบบมีเงื่อนไข if ซ้อนคำสั่ง for และคำสั่ง while ตัดเกรด

```

1  stu = int(input('ป้อนจำนวนนักเรียน = '))
2  for i in range(1, stu + 1):
3      j = 1
4      lst = []
5      print('นักเรียนคนที่ ', i)
6      while j <= 3:
7          n = float(input(f'กรอกคะแนนครั้งที่ {j} = '))
8          lst.append(n)
9          j += 1
10     if sum(lst) <= 49:
11         grade = 'F'
12     elif sum(lst) <= 59:
13         grade = 'D'
14     elif sum(lst) <= 69:
15         grade = 'C'
16     elif sum(lst) <= 79:
17         grade = 'B'
18     else:
19         grade = 'A'
20     print(f'นักเรียนคนที่ {i} ได้คะแนน = {lst} คะแนนรวม = {sum(lst)} ได้
    ↳ เกรด {grade}')

```

ป้อนจำนวนนักเรียน = 2

นักเรียนคนที่ 1

กรอกคะแนนครั้งที่ 1 = 23

กรอกคะแนนครั้งที่ 2 = 12

กรอกคะแนนครั้งที่ 3 = 23

นักเรียนคนที่ 1 ได้คะแนน = [23.0, 12.0, 23.0] คะแนนรวม = 58.0 ได้เกรด D

นักเรียนคนที่ 2

กรอกคะแนนครั้งที่ 1 = 33

กรอกคะแนนครั้งที่ 2 = 23

กรอกคะแนนครั้งที่ 3 = 25

นักเรียนคนที่ 2 ได้คะแนน = [33.0, 23.0, 25.0] คะแนนรวม = 81.0 ได้เกรด A



จากตัวอย่างโปรแกรมการพัฒนาต่อจากโปรแกรมการใช้คำสั่ง **for** ข้อน **while** ในบรรทัดที่ 10-19 ใช้คำสั่ง **if** เข้ามาช่วยตัดสินใจการตัดเกรด A-F หลังจากผู้ใช้งานป้อนคะแนนของนักเรียนแต่ละคนครบจำนวน 3 ครั้ง จากการตรวจสอบการทำซ้ำด้วยคำสั่ง **while** ในบรรทัดที่ 6 และโปรแกรมทำงานตามจำนวนรอบที่กำหนดไว้ในคำสั่ง **for** ที่อ่านค่าข้อมูลจากที่สร้างด้วยฟังก์ชัน **range()** ในบรรทัดที่ 2

6.6 การควบคุมการทำซ้ำด้วยคำสั่ง **break**, **continue** และ **pass**

คำสั่ง **break** นำมาใช้ในกรณีที่ต้องการให้ออกจากการทำซ้ำก่อนถึงรอบที่กำหนด หรือใช้ในกรณีที่ต้องการตรวจสอบความถูกต้อง ซึ่งมีผลทำให้คำสั่งที่เหลือหลัง **break** ในขอบเขตคำสั่งทำซ้ำเดียวกันไม่ถูกประมวลผล แต่จะข้ามไปทำงานในคำสั่งที่เหลือของโปรแกรมเลยทันที ดังตัวอย่างต่อไปนี้

ตัวอย่าง 6.25

การใช้คำสั่ง `break` หยุดการทำงานทันที เมื่อป้อนข้อมูลไม่ถูกต้อง

```
1 for i in range(1, 6):
2     n = int(input('กรุณาป้อนตัวเลข 1-10 : '))
3     if n >= 11:
4         print('คุณป้อนตัวเลขไม่ถูกต้อง !!')
5         break
6     else:
7         j = n * i
8         print(f'ผลคูณระหว่าง {i} * {n} = {j}')
9 print('สิ้นสุดการทำงานของโปรแกรม')
```

```
กรุณาป้อนตัวเลข 1-10 : 3
ผลคูณระหว่าง 1 * 3 = 3
กรุณาป้อนตัวเลข 1-10 : 5
ผลคูณระหว่าง 2 * 5 = 10
กรุณาป้อนตัวเลข 1-10 : 11
คุณป้อนตัวเลขไม่ถูกต้อง !!
สิ้นสุดการทำงานของโปรแกรม
```



โปรแกรมจะแสดงผลครบทุกรอบการทำงานเมื่อผู้ใช้ป้อนตัวเลขได้ถูกต้องตามเงื่อนไขในบรรทัดที่ 6 คือป้อนตัวเลขไม่เกิน 10 ถ้าในระหว่างการทำงานเมื่อผู้ใช้งานป้อนตัวเลขไม่ถูกต้อง คือ ป้อนตัวเลขมากกว่า 10 โปรแกรมจะออกจากการทำงานทันทีด้วยคำสั่ง `break` ในบรรทัดที่ 5

ตัวอย่าง 6.26

การใช้คำสั่ง break ร่วมกับคำสั่ง while ออกจากการทำซ้ำ

```

1 while True:
2     i = input('ต้องการป้อนคะแนนให้กด Enter, ออกจากโปรแกรมให้กด N: ')
3     print ('-'*30)
4     if i == 'N':
5         break
6     else:
7         name = input('ชื่อนักเรียน: ')
8         j = 1
9         lst = []
10        while j <= 3:
11            score = float(input(f'กรอกคะแนนครั้งที่ {j} = '))
12            lst.append(score)
13            j += 1
14            print(f'นักเรียน: {name} ได้คะแนน = {lst} คะแนนรวม
15                ↳ {sum(lst)}')
16 print ('สิ้นสุดการทำงานของโปรแกรม')
```

ต้องการป้อนคะแนนให้กด Enter, ออกจากโปรแกรมให้กด N:



ชื่อนักเรียน: Sally

กรอกคะแนนครั้งที่ 1 = 33

กรอกคะแนนครั้งที่ 2 = 23

กรอกคะแนนครั้งที่ 3 = 25

นักเรียน: Sally ได้คะแนน = [33.0, 23.0, 25.0] คะแนนรวม 81.0

ต้องการป้อนคะแนนให้กด Enter, ออกจากโปรแกรมให้กด N:

ชื่อนักเรียน: John

กรอกคะแนนครั้งที่ 1 = 23

กรอกคะแนนครั้งที่ 2 = 22

กรอกคะแนนครั้งที่ 3 = 12

นักเรียน: John ได้คะแนน = [23.0, 22.0, 12.0] คะแนนรวม 57.0

ต้องการป้อนคะแนนให้กด Enter, ออกจากโปรแกรมให้กด N: N

สิ้นสุดการทำงานของโปรแกรม

คำสั่ง **continue** นำมาใช้หยุดการทำงานของคำสั่งโปรแกรมที่เหลือในขอบเขตคำสั่งทำซ้ำเดียวกัน แล้วให้กลับไปทำงานในรอบทำซ้ำถัดไป หลังจากทำซ้ำเสร็จแล้วจึงออกไปประมวลคำสั่งโปรแกรมอื่น ๆ ดังตัวอย่างต่อไปนี้

ตัวอย่าง 6.27

การใช้คำสั่ง **continue** ร่วมกับคำสั่ง **while**

```
1 i = 1
2 while i <= 5:
3     n = int(input('กรุณาป้อนตัวเลข 1-10 : '))
4     if n > 10:
5         print('คุณป้อนตัวเลขไม่ถูกต้อง !!')
6         continue
7     else:
8         j = n * i
9         print(f'ผลคูณระหว่าง {i} * {n} = {j}')
10    i += 1
```

```
กรุณาป้อนตัวเลข 1-10 : 4
ผลคูณระหว่าง 1 * 4 = 4
กรุณาป้อนตัวเลข 1-10 : 5
ผลคูณระหว่าง 2 * 5 = 10
กรุณาป้อนตัวเลข 1-10 : 6
ผลคูณระหว่าง 3 * 6 = 18
กรุณาป้อนตัวเลข 1-10 : 112
คุณป้อนตัวเลขไม่ถูกต้อง !!
กรุณาป้อนตัวเลข 1-10 : 3
ผลคูณระหว่าง 4 * 3 = 12
กรุณาป้อนตัวเลข 1-10 : 8
ผลคูณระหว่าง 5 * 8 = 40
```



จากตัวอย่างโปรแกรม เมื่อผู้ใช้งานป้อนตัวเลขได้ถูกต้องตามเงื่อนไขคือ **1-10** โปรแกรมถึงจะแสดงผลลัพธ์ แต่เมื่อผู้ใช้งานป้อนตัวเลขมากกว่า **10** ส่งผลให้คำสั่ง **continue** ทำงานแล้วกลับไปทำงานในรอบใหม่และคำสั่งที่เหลือในขอบเขตของคำสั่งทำซ้ำก็ไม่ถูกประมวลผล

แต่โปรแกรมจะทำงานจนครบรอบตามเงื่อนไขที่กำหนดคือ **5** รอบ ตามเงื่อนไขที่ได้กำหนดไว้ในคำสั่ง **while**

คำสั่ง **pass** นำมาใช้สำหรับการละเว้นการประมวลผลคำสั่งโปรแกรม ในกรณีที่เรายังไม่ทราบว่าจะเขียนคำสั่งโปรแกรมให้ทำงานอะไรซึ่งช่วยให้โปรแกรมประมวลผลคำสั่งได้ตามปกติ หลังจากที่ได้คำสั่งการทำงานที่ต้องการแล้วจึงนำมาแทนที่คำสั่ง **pass** ในภายหลัง ซึ่งมีประโยชน์ไว้ใช้ในการทดสอบโปรแกรม ดังตัวอย่างต่อไปนี้

ตัวอย่าง 6.28

การเขียนคำสั่งโปรแกรมการใช้คำสั่ง **pass** ทดสอบการทำงานของโปรแกรม

```

1 salary = int(input('ป้อนเงินเดือนพนักงาน = '))
2 if salary <= 30000:
3     pass
4 elif salary <= 35000:
5     pass
6 elif salary <= 40000:
7     bonus = salary * 3
8     print('เงินโบนัส =', bonus, 'บาท')
9 else:
10    bonus = salary * 2
11    print('เงินโบนัส =', bonus, 'บาท')
12 print('สิ้นสุดการทำงานของโปรแกรม')
```

ป้อนเงินเดือนพนักงาน = 25000
สิ้นสุดการทำงานของโปรแกรม

ป้อนเงินเดือนพนักงาน = 45000
เงินโบนัส = 90000 บาท
สิ้นสุดการทำงานของโปรแกรม



จากผลลัพธ์รูปบนเป็นการทำงานของคำสั่ง **pass** ในบรรทัดที่ 3 โดยที่ผู้ใช้งานป้อนเงินเดือนเท่ากับ **25000** แต่เรายังไม่ได้ตัดสินใจกำหนดค่าโบนัสให้กับพนักงาน จึงใช้คำสั่ง **pass** ให้ทำงานแทนไปก่อน ทำให้บรรทัดที่ 12 แสดงผลเท่านั้น ส่วนรูปล่างเมื่อผู้ใช้งานป้อนเงินเดือนเท่ากับ **45000** ทำให้เข้าเงื่อนไขบรรทัดที่ 6 และคำนวณค่าโบนัสให้กับพนักงานในบรรทัดที่

7 แล้วแสดงผลลัพธ์ค่าโบนัสที่ได้ในบรรทัดที่ 8 สำหรับบรรทัดที่ 12 จะแสดงผลทุกครั้งหลังโปรแกรมทำงานเสร็จ

สรุปก่อนจบบท

การเขียนคำสั่งโปรแกรมควบคุมทิศทางการทำงานแบ่งออกเป็น 3 รูปแบบ ได้แก่ คำสั่งการทำงานแบบลำดับ คำสั่งการทำงานแบบมีเงื่อนไข `if`, `if ... else` และ `if ... elif` และคำสั่งการทำงานแบบทำซ้ำ `while` และ `for` นอกจากนี้ผู้อ่านยังได้เรียนรู้การนำคำสั่ง `if` และคำสั่ง `while` หรือ `for` มาเขียนเป็นคำสั่งโปรแกรมซ้อนกัน เพื่อนำไปประยุกต์ใช้ในงานที่มีความซับซ้อนมากขึ้น ในการเขียนคำสั่ง `if`, `while` และ `for` ด้วยภาษาไพธอน ผู้เขียนโปรแกรมต้องไม่ลืมใส่เครื่องหมาย `colon(:)` ท้ายประโยคเสมอ และเมื่อขึ้นบรรทัดใหม่ในขอบเขตคำสั่งทำซ้ำเดียวกัน ควรใช้กดปุ่ม Tab แทนการกดปุ่ม Space bar

แบบฝึกหัด

- จงเขียนโปรแกรมเปรียบเทียบตัวเลขที่รับเข้ามาผ่านทางคีย์บอร์ด จำนวน 2 ตัว ถ้าตัวเลขที่ 1 มีค่ามากกว่าตัวเลขที่ 2 ให้แสดงคำว่า 'ตัวเลขที่ 1 มีค่ามากกว่าตัวเลขที่ 2' ถ้าตัวเลขที่ 1 มีค่าน้อยกว่าตัวเลขที่ 2 ให้แสดงคำว่า 'ตัวเลขที่ 1 มีค่าน้อยกว่าตัวเลขที่ 2'
- จงเขียนโปรแกรมคำนวณการซื้อสินค้าทั้งหมด 3 อย่าง โดยรับราคาสินค้าผ่านทางคีย์บอร์ด พร้อมทั้งแสดงราคาสินค้าแต่ละชิ้น สรุปราคารวมของสินค้าที่ซื้อ สุดท้ายให้แสดงผลคำว่า 'Good Bye !!'
- จงเขียนโปรแกรมคำนวณค่าวงรถยนต์ กำหนดให้ป้อนราคารถยนต์และจำนวนเดือนที่ต้องการผ่อนผ่านทางคีย์บอร์ด โดยมีเงื่อนไขดังนี้
 - ถ้าต้องการผ่อน 12 เดือน คิดดอกเบี้ย 1% ต่อเดือน
 - ถ้าต้องการผ่อน 24 เดือน คิดดอกเบี้ย 1.5% ต่อเดือน
 - ถ้าต้องการผ่อน 36 เดือน คิดดอกเบี้ย 2.5% ต่อเดือน

- ถ้าต้องการผ่อน 48 เดือน คิดดอกเบี้ย 3.5% ต่อเดือน
- ถ้าต้องการผ่อน 60 เดือน คิดดอกเบี้ย 4.5% ต่อเดือน

4. จงเขียนโปรแกรมเลือกเครือข่ายเติมเงินโทรศัพท์มือถือ โดยมีเงื่อนไขดังต่อไปนี้

- ก่อนป้อนจำนวนเงินให้แสดงเมนูเลือกเครือข่ายเสมอ
- มีเครือข่ายให้เลือกเติมเงิน 3 เครือข่ายได้แก่ DTAC, AIS และ True
- มีเมนูแสดงจำนวนเงินที่สามารถเติมเงินคือ 50, 100, 300, 500 และ 1000 บาท ทุกเครือข่าย
- โปรแกรมสามารถเติมเงินได้ครั้งละหลาย ๆ คน แล้วเลือกว่าจะเติมให้คนไหน ในขั้นตอนไหน โดยไม่ต้องสั่งให้โปรแกรมทำงานใหม่
- มีเมนูแสดงแจ้งผู้ใช้งานให้กดปุ่มใด เพ้อออกจากโปรแกรม
- สรุปยอดจากการเติมเงินจากลูกค้าที่มีการเติมเงินแต่ละคนและยอดรวมทั้งหมด