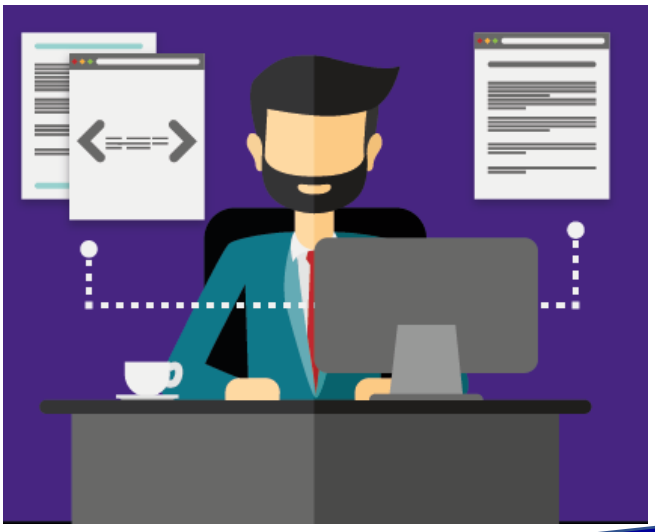
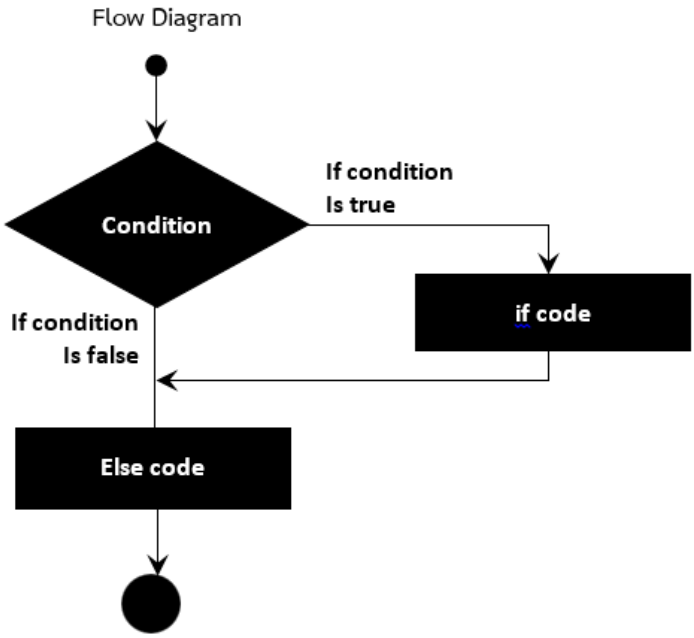
บทที่ 6 โครงสร้างควบคุม
(Control structures)



เนื้อหา

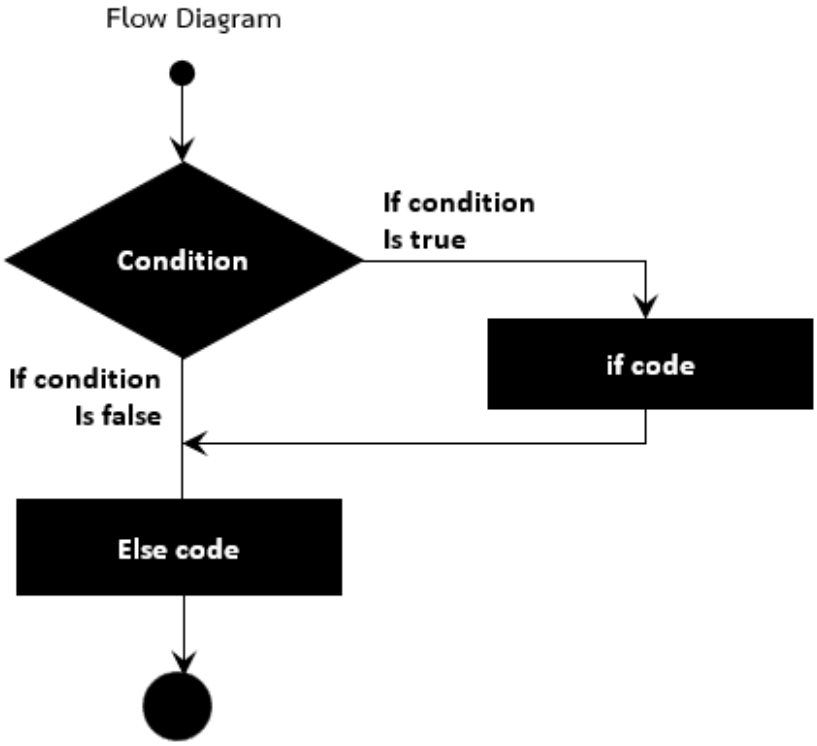
1. การเขียนโปรแกรมแบบมีทางเลือก
   1.1. ตัวดำเนินการเปรียบเทียบ
   1.2. conditional statements
2. การเขียนโปรแกรมแบบวนซ้ำ
   2.1. while
   2.2. for
   2.3. break, continue, pass, else Clauses Statements



การเขียนโปรแกรมควบคุมทิศทางแบบมีทางเลือก

ในการทำงานกับคอมพิวเตอร์นั้นมักจะต้องมีการแบ่งแยกเงื่อนไขเพื่อที่จะให้คอมทำงานแตกต่างกันไปตามสถานการณ์ เหมือนกับที่ในธรรมชาติมักจะมีการแบ่งแยก เช่น เวลาไปเที่ยวคนต่างชาติจะต้องจ่ายค่าเข้าแพงกว่า หนังบางประเภทมีการจำกัดอายุคนดู ห้องน้ำก็มีการแยกชายหญิง สังคมมนุษย์เต็มไปด้วยการแบ่งแยก

ในคอมพิวเตอร์เองก็สามารถมีการแบ่งแยก เช่น เวลาคำนวณ เจอจำนวนบวกให้คำนวณอย่างหนึ่ง เจอจำนวนลบให้คำนวณแบบหนึ่ง เป็นต้น หากมีการตั้งเงื่อนไขก็จะสร้างให้เกิดทางแยกในการทำงาน และทำงานได้ยืดหยุ่นกว้างขวางมากขึ้น ในภาษา Python รวมถึงภาษาโปรแกรมอื่น ๆ ส่วนใหญ่คำสั่งที่ใช้ในการตั้งเงื่อนไขก็คือ if และ else



ที่มา : https://www.tutorialspoint.com/python/python_if_else.htm

ตัวดำเนินการในภาษา Python ประกอบด้วย

- ตัวดำเนินการเปรียบเทียบ (Comparison operators)
- ตัวดำเนินการตรรกศาสตร์ (Logical operators)

ตัวดำเนินการเปรียบเทียบ (Comparison operators)

ตัวดำเนินการเปรียบเทียบ (Comparison operators) คือตัวดำเนินการที่ใช้สำหรับเปรียบเทียบค่าหรือค่าในตัวแปร ซึ่งผลลัพธ์ของการเปรียบเทียบนั้นจะเป็น True หากเงื่อนไขเป็นจริง และเป็น False หากเงื่อนไขไม่เป็นจริง ตัวดำเนินการเปรียบเทียบมักจะใช้กับคำสั่งตรวจสอบเงื่อนไข if และคำสั่งวนซ้ำ for while เพื่อควบคุมการทำงานของโปรแกรม

ในภาษา Python การเขียนโปรแกรมควบคุมทิศทางแบบมีทางเลือก ประกอบไปด้วย operators ดังต่อไปนี้
กำหนดให้ตัวแปร a เก็บ 10 และตัวแปร b เก็บ 20

Comparison operators	รายละเอียด	ตัวอย่าง
==	หากค่าของตัวแปรทั้งสองค่าเท่ากันเงื่อนไขนั้นจะเป็นจริง	(a == b) ไม่เป็นความจริง
! =	หากค่าของตัวแปรทั้งสองค่าไม่เท่ากันเงื่อนไขจะเป็นจริง	(a! = b) เป็นจริง
<>	หากค่าของตัวแปรสองค่าไม่เท่ากันเงื่อนไขจะกลายเป็นจริง	(a <> b) เป็นจริง สิ่งนี้คล้ายกับ! = operator
>	หากค่าของตัวแปรด้านซ้ายมีค่ามากกว่าค่าของตัวแปรด้านขวาเงื่อนไขจะกลายเป็นจริง	(a> b) ไม่เป็นความจริง
<	หากค่าของตัวแปรด้านซ้ายน้อยกว่าค่าของตัวแปรด้านขวาเงื่อนไขจะกลายเป็นจริง	(a <b) เป็นจริง
>=	หากค่าของตัวแปรด้านซ้ายมีค่ามากกว่าหรือเท่ากับค่าของตัวแปรด้านขวาเงื่อนไขจะกลายเป็นจริง	(a> = b) ไม่เป็นความจริง
<=	หากค่าของตัวแปรด้านซ้ายน้อยกว่าหรือเท่ากับค่าของตัวแปรด้านขวาเงื่อนไขจะกลายเป็นจริง	(a <= b) เป็นจริง

ตัวดำเนินการตรรกศาสตร์ (Logical operators)

ตัวดำเนินการตรรกศาสตร์ (Logical operators) คือตัวดำเนินการที่ใช้สำหรับประเมินค่าทางตรรกศาสตร์ ซึ่งเป็นค่าที่มีเพียงจริง (True) และเท็จ (False) เท่านั้น โดยทั่วไปแล้วเรามักใช้ตัวดำเนินการตรรกศาสตร์ในการเชื่อม Boolean expression ตั้งแต่หนึ่ง expression ขึ้นไปและผลลัพธ์สุดท้ายที่ได้นั้นจะเป็น Boolean
กำหนดให้ตัวแปร a เก็บ 10 และตัวแปร b เก็บ 20

Comparison operators	รายละเอียด	ตัวอย่าง
AND	ตัวแปรทั้งคู่เป็นจริงเงื่อนไขจะกลายเป็นจริง	(a and b) is true.
OR	ตัวแปรตัวใดตัวหนึ่งไม่เป็นศูนย์เงื่อนไขจะกลายเป็นจริง	(a or b) is true.
NOT	ใช้เพื่อย้อนกลับค่าของตัวแปร	Not(a and b) is false.

ในภาษา Python นั้น การเขียน block เพื่อควบคุมการทำงาน ตามเงื่อนไข ใช้การกำหนดช่วงด้วยการย่อหน้า

รูปแบบการเขียน if Statements

คำสั่ง if เป็นคำสั่งที่ใช้ควบคุมการทำงานของโปรแกรมที่เป็นพื้นฐานและง่ายที่สุด เราใช้คำสั่ง if เพื่อสร้างเงื่อนไขให้โปรแกรมทำงานตามที่เราต้องการเมื่อเงื่อนไขนั้นตรงกับที่เรากำหนด เช่น การตรวจสอบค่าในตัวแปรกับตัวดำเนินการประเภทต่าง ๆ

```
if Condition :
    Statements
    .....
    .....
```

จะเห็นว่า หลัง if จะเป็นเงื่อนไขเช็คการทำงาน และตามด้วยเครื่องหมาย ":" และ Statements หรือการทำงานถัดไปจะต้องอยู่ในย่อหน้าใหม่ โดยใช้ปุ่ม Tab จะมีกี่ Statements ก็ตามถ้าต้องการให้อยู่ภายใน block if นั้นต้อง Tab ทุกครั้ง และ block จะจบอัตโนมัติ เมื่อไม่มีการต่อหน้าใหม่

#EXAMPLE 1

```
y = 1
if y == 1:
    print("y still equals 1")
```

```
y still equals 1
```

#EXAMPLE 2

```
logged_in = False
if not logged_in:
    print('You must login to continue')
```

```
You must login to continue
```

#EXAMPLE 3

```
m = 4
if m % 2 == 0 and m > 0:
    print('m is even and positive numbers')
```

```
m is even and positive numbers
```

อีกคำสั่งหนึ่งที่ทำงานควบคู่กับคำสั่ง if คือคำสั่ง else clause โดยโปรแกรมจะทำงานในคำสั่ง else ถ้าหากเงื่อนไขในคำสั่ง if นั้นไม่เป็นจริง กล่าวอีกนัยหนึ่ง มันจะทำงานเมื่อเงื่อนไขก่อนหน้านั้นไม่เป็นจริงหรือเป็นเงื่อนไข Default

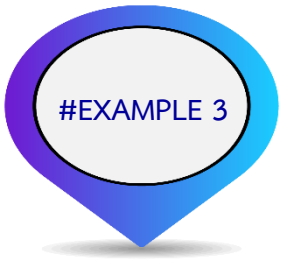
```
if Condition :
        Statements
        .....
else :
        .....
        .....
```


```
n = 5
if n == 10:
    print('n equal to 10')
else:
    print('n is something else except 10')
```

```
n is something else except 10
```


```
name = 'James'
if name == 'Mateo':
    print('Hi, Mateo.')
else:
    print('Who are you?')
```

```
Who are you?
```



กรณีรับสมัครพนักงานกำหนดเงื่อนไข อายุระหว่าง 20 – 45 ปี เพศ ชาย

```
1 age = 26
2 gender = "m"
3 #if age >=20 and age <= 45  and gender == "m":
4 if (age >=20 and age <= 45)  and (gender == "m"):
5     print("past")
6 else:
7     print("fail")
```

```
past
```

คำสั่ง elif นั้นเป็นคำสั่งที่ใช้สำหรับสร้างเงื่อนไขแบบหลายทางเลือกให้กับโปรแกรมที่มีการทำงานเช่นเดียวกับ switch case ในภาษาอื่น ๆ คำสั่ง elif นั้นต้องใช้หลังจากคำสั่ง if เสมอและสามารถมี else ได้ในเงื่อนไขสุดท้าย

```
if Condition :
        Statements
        .....
elif Condition :
        Statements
        .....
elif Condition :
        Statements
        .....
else :
        .....
        .....
```


```
print('Welcome to marcuscode\'s game')
level = input('Enter level (1 - 4): ')

if level == '1':
    print('Easy')
elif level == '2':
    print('Medium')
elif level == '3':
    print('Hard')
elif level == '4':
    print('Expert')
else:
    print('Invalid level selected')
```

```
Welcome to marcuscode's game
Enter level (1 - 4): 1
Easy
```

ในตัวอย่าง เป็นโปรแกรมจำลองในการเลือกโหมดของการเล่นเกม เราได้ให้ผู้ใช้กรอกค่าระหว่าง 1 - 4 เพื่อใช้ในการเปรียบเทียบกับระดับความยากของเกม โดยที่ 1 เป็นระดับที่ง่ายที่สุด และ 4 นั้นเป็นระดับที่ยากที่สุด คุณจะเห็นว่าเราได้ให้คำสั่ง elif เพราะเรามีเงื่อนไข 4 แบบ และคำสั่ง else ในกรณีที่ตัวเลขที่ผู้เล่นกรอกเข้ามานั้นไม่ตรงกับเงื่อนไขใด ๆ ก่อนหน้าเลย

การเขียนโปรแกรมควบคุมทิศทางแบบวนซ้ำหรือทำซ้ำ (Loop)

การแก้ปัญหาต่าง ๆ ในชีวิตประจำวัน มักจะพบเจอกับปัญหาที่ต้องใช้ความพยายามในการแก้ปัญหาดังกล่าววนซ้ำหลาย ๆ ครั้งเพื่อที่จะบรรลุเป้าหมาย เช่น ถ้าต้องการสอบให้ได้คะแนนดี จำเป็นต้องอ่านหนังสือในบทที่จะออกสอบหลาย ๆ รอบ ยิ่งอ่านมากยิ่งมีโอกาสที่จะได้คะแนนสอบมากตามไปด้วย หรือนักกีฬาที่ต้องการได้เหรียญทองในการแข่งขันจำเป็นต้องฝึกซ้ำแบบเดิมให้เกิดความชำนาญ ยิ่งชำนาญมากก็ยิ่งมีโอกาสประสบความสำเร็จมากตามไปด้วย เช่นเดียวกับการแก้ปัญหาทางคอมพิวเตอร์ บางปัญหานั้นจำเป็นต้องประมวลผลซ้ำไปซ้ำมาหลาย ๆ รอบ จนกว่าจะได้คำตอบ เช่น การหาผลรวมของจำนวนเต็มตั้งแต่ 1 – n , การหาค่า factorial, การหาค่า prime number และการคำนวณเลขลำดับอนุกรม เป็นต้น ปัญหาเหล่านี้จำเป็นต้องอาศัยเทคนิคการทำซ้ำทั้งสิ้น ภาษาไพธอนเตรียมคำสั่งในการทำซ้ำไว้ 2 คำสั่ง คือ while และ for loop ดังนี้

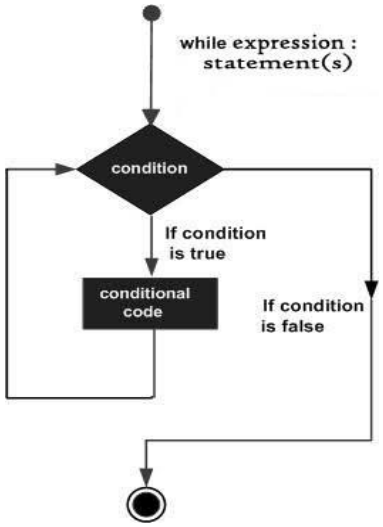
1. คำสั่ง while loop

while เป็นคำสั่งวนซ้ำที่มีการตรวจสอบเงื่อนไข (condition) ก่อนเข้าทำงานเสมอ เมื่อเงื่อนไขที่ทำการตรวจสอบเป็นจริง จึงจะประมวลผลคำสั่งหลัง while แต่ถ้าเงื่อนไขเป็นเท็จจะยุติการทำงานทันที สำหรับงานที่นิยมใช้ while ในการแก้ปัญหาคือ ปัญหาที่ไม่ทราบจำนวนรอบการทำงานที่แน่นอนหรือปัญหาที่ไม่สามารถทราบได้ล่วงหน้าว่าจะต้องใช้เวลาในการประมวลผลนานเท่าใด ส่วนใหญ่มักจะหยุดการทำงานของ while ด้วยเงื่อนไขบางประการ เช่น กดแป้นพิมพ์ที่บ่งบอกว่าต้องการออกจากโปรแกรม เช่น ESC , q , -1 , 0 เป็นต้น หรือตรวจสอบค่าในตัวแปรตัวใดตัวหนึ่งในโปรแกรมเป็นเท็จ เป็นต้น

โครงสร้างการทำงาน while loop มีรูปแบบคำสั่งดังนี้

```
while condition:
        statement(s)
```

แผนผังจำลองการทำงานจะมีลักษณะดังภาพ



รูปที่ 1.1 แสดงแผนภาพจำลองการทำงานของคำสั่ง while loop

ที่มา : https://www.tutorialspoint.com/python/python_while_loop.htm

ตัวอย่างโปรแกรมที่ 1.1

Program Example 1.1 : while

```
1  # While loop testing
2  count = 0
3  while (count < 9):
4          print ('The count is:', count)
5          count = count + 1
6  print ("Good bye!")
```

OUTPUT

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

จากตัวอย่างโปรแกรมที่ 1.1 เริ่มต้นบรรทัดที่ 2 เป็นการกำหนดค่าให้ตัวแปร count มีค่าเท่ากับ 0 เพื่อใช้สำหรับนับค่า ขั้นตอนต่อไปบรรทัดที่ 3 โปรแกรมจะเปรียบเทียบเงื่อนไขใน while ว่า count มีค่าน้อยกว่า 9 หรือไม่ ถ้าเงื่อนไขเป็นจริง (count < 9) โปรแกรมจำทำงานหลัง while ในบรรทัดที่ 4 โดยพิมพ์ข้อความว่า “The count is:” พร้อมกับค่าในตัวแปร count ต่อจากนั้นในบรรทัดที่ 5 จะทำการเพิ่มค่าให้ตัวแปร count อีก 1 ต่อจากนั้นโปรแกรมจะวนกลับไปตรวจสอบเงื่อนไขของ while ในบรรทัดที่ 3 ใหม่ เป็นรอบที่ 2 และทำการประมวลผลคำสั่งตามลำดับในบรรทัดที่ 3 -> 4 -> 5 ไปเรื่อย ๆ จนกว่าเงื่อนไขที่ while จะเป็นเท็จ (count >= 9) เมื่อเงื่อนไขใน while เป็นเท็จโปรแกรมจะมาทำงานในบรรทัดที่ 6 โดยพิมพ์ข้อความว่า “Good bye!” ก่อนจบโปรแกรมเสมอ

การวนซ้ำแบบไม่รู้จบ (The Infinite Loop)

บ่อยครั้งที่พบปัญหาในการใช้งาน while คือ เงื่อนไขที่ตรวจสอบไม่เป็นเท็จ ซึ่งสาเหตุมาจากหลายกรณี แต่ส่วนใหญ่มาจากการหลงลืมทำให้ตัวแปรที่ใช้ตรวจสอบเงื่อนไขเป็นเท็จ เช่น ในกรณีตัวอย่างโปรแกรมที่ 1.1 ถ้าลืมเพิ่มค่าใช้กับตัวแปร count จะทำให้โปรแกรมเข้าสู่สถานะที่เรียกว่า Infinite loop อยู่เหมือนกัน สำหรับคำสั่งตัวอย่างที่ใช้ในกรณีที่ต้องการให้โปรแกรมทำงานในลักษณะ infinite loop ดังโปรแกรมที่ 1.2

ตัวอย่างโปรแกรมที่ 1.2

Program Example 1.2 : Infinite loop

```
1  # Infinite loop program
2  # When would you like to exit this program, push CTRL + C
3  var = 1
4  while var == 1 :  # This constructs an infinite loop
5          num = int(input("Enter a number  :"))
6         print("You entered: ", num)
7  print("Good bye!")
```

OUTPUT

```
Enter a number  :394
You entered:  394
Enter a number  :3994
You entered:  3994
Enter a number  :  Here user enter CTRL + C
Traceback (most recent call last):
    File "C:\Python34\exam6_10.py", line 5, in <module>
        num = int(input("Enter a number  :"))
    File "C:\Python34\lib\idlelib\PyShell.py", line 1381, in readline
      line = self._line_buffer or self.shell.readline()
KeyboardInterrupt
```

จากตัวอย่างโปรแกรม 1.2 เริ่มต้นบรรทัดที่ 3 กำหนดค่าใช้ตัวแปร var = 1 เพื่อใช้สำหรับเปรียบเทียบเงื่อนไขก่อนเข้าทำงานใน while loop (การกำหนดเงื่อนไขเพื่อใช้เปรียบเทียบก่อนเข้าทำงานใน while loop เป็นขั้นตอนที่สำคัญมากและต้องทำเสมอ) บรรทัดที่ 4 ทำการตรวจสอบเงื่อนไขก่อนเข้าทำงานใน while ผลการตรวจสอบปรากฏว่า เป็นจริงเสมอ เพราะว่าค่าในตัวแปร var มีค่าเท่ากับ 1 โปรแกรมจึงเลื่อนไปทำคำสั่งในบรรทัดที่ 5 หลังคำสั่ง while คือคำสั่งอ่านข้อมูลจำนวนเต็มมาจากแป้นพิมพ์เก็บไว้ในตัวแปร num จากนั้นบรรทัดที่ 6 จะพิมพ์ค่าที่อยู่ในตัวแปร num ออกมาทางจอภาพ แล้วโปรแกรมจะกลับไปทำงานในบรรทัดที่ 4 อีกครั้ง ทั้งนี้เพราะเงื่อนไขใน while ยังเป็นจริงอยู่ การทำงานจะทำซ้ำคำสั่งบรรทัดที่ 4 -> 5 -> 6 ไปเรื่อย ๆ จนกว่าเงื่อนไขใน while จะเป็นเท็จ แต่สำหรับในกรณีนี้จะเป็นจริงตลอดไป แบบไม่มีวันจบ (Infinite loop) และคำสั่งในบรรทัดที่ 7 จะไม่ถูกประมวลผลเลย ถ้าต้องการออกจากโปรแกรมนี้ ให้กดปุ่ม CTRL+C เท่านั้น เพื่อเป็นการ Terminate โปรแกรมและโปรแกรมจะแสดงข้อความผิดพลาดออกมาดังตัวอย่างข้างบน

การใช้คำสั่ง else ร่วมกับ while

Python จะอนุญาตให้ผู้เขียนโปรแกรมสามารถใช้ else ร่วมกับคำสั่ง while ได้ โดย else จะทำงานก็ต่อเมื่อเงื่อนไขใน while loop เป็นเท็จ ดังตัวอย่างที่ 1.3

ตัวอย่างโปรแกรมที่ 1.3

Program Example 1.3 : while with else

```
1  # Testing else statement with while loop
2  count = 0
3  while count < 5:
4          print(count, " is less than 5 (While Loop)")
5          count = count + 1
6  else:
7          print(count," is not less than 5 (Else after exit while loop)")
8  print("Good bye!")
```

OUTPUT

```
0   is less than 5 (While Loop)
1   is less than 5 (While Loop)
2   is less than 5 (While Loop)
3   is less than 5 (While Loop)
4   is less than 5 (While Loop)
5   is not less than 5 (Else after exit while loop)
Good bye!
```

จากตัวอย่างโปรแกรมที่ 1.3 บรรทัดที่ 2 กำหนดค่าเริ่มต้นให้ตัวแปร count เท่ากับ 0 เพื่อใช้สำหรับทำการเปรียบเทียบก่อนเข้าทำงานใน while loop ผลลัพธ์จากการเปรียบเทียบ (บรรทัดที่ 3) มีค่าเป็นจริง เพราะ count < 5 โปรแกรมจะเข้าไปประมวลผลในบรรทัดที่ 4 โดยพิมพ์ข้อความว่า “X is lees than 5 (While Loop)” โดย X คือค่าที่อยู่ในตัวแปร count ในบรรทัดที่ 5 โปรแกรมทำการเพิ่มค่า count อีก 1 จากนั้นโปรแกรมจะวนกลับมาตรวจสอบเงื่อนไขใน while อีก (เพราะเงื่อนไขยังไม่เป็นเท็จ) ซึ่งโปรแกรมจะทำคำสั่งซ้ำในบรรทัดที่ 3 -> 4 -> 5 เช่นนี้ไปเรื่อย ๆ จนกว่า count >= 5 จึงทำให้โปรแกรมยุติการทำงานใน while loop ลง และมาประมวลผลคำสั่งในบรรทัดที่ 6 โดยพิมพ์ข้อความว่า “5 is not less than 5(Else after exit while loop)” และตามด้วยข้อความ “Good bye!” ในบรรทัดที่ 8 ดังแสดงใน OUTPUT ของโปรแกรมด้านบน

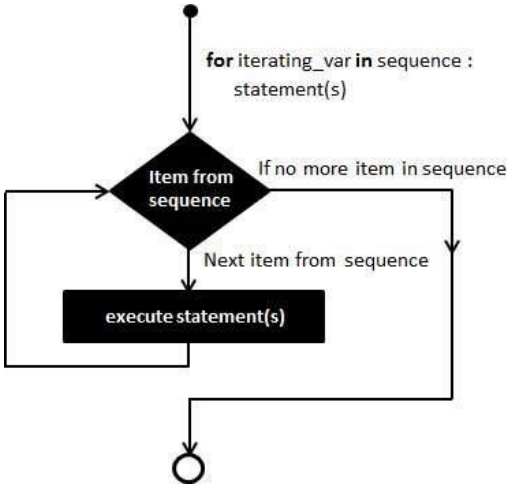
2. คำสั่ง while loop

คำสั่ง for เป็นคำสั่งที่ใช้สำหรับการทำซ้ำเช่นเดียวกับ while และต้องมีการตรวจสอบเงื่อนไขก่อนเข้าลูปเหมือนกัน แต่แตกต่างกันตรงที่ for จะตรวจสอบรายการแบบลำดับแทน เช่น ข้อมูลชนิดสตริง ลิสต์ หรือทัพเพิล เป็นต้น

โครงสร้างการทำงาน for loop มีรูปแบบคำสั่งดังนี้

```
for iterating_var in sequence:
    statement(s)
```

โดย interating_var คือตัวแปรที่ใช้สำหรับรับค่าทีละค่าเพื่อนำมาประมวลผล จากข้อมูลที่อยู่ในตัวแปร sequence เมื่อข้อมูลในตัวแปร sequence เป็นชนิดลิสต์ คำสั่ง for จะดึงข้อมูลในตำแหน่งแรกของลิสต์ออกมาเก็บไว้ใน interating_var หลังจากนั้นจะเริ่มทำคำสั่งใน statement(s) เมื่อคำสั่งใน statement(s) หมดแล้ว การควบคุมจะกลับไปเริ่มต้นใหม่ที่ for แล้วดึงข้อมูลในลิสต์ลำดับถัดไปมาทำงาน การทำงานจะเป็นไปในลักษณะเช่นนี้ไปเรื่อย ๆ จนกว่าข้อมูลในลิสต์จะหมด for จึงจะยุติการทำงาน สำหรับแผนผังจำลองการทำงานของ for และตัวอย่างการดึงข้อมูลสมาชิกในรายการ ดังภาพที่ 2.1



รูปที่ 2.1 แสดงแผนภาพจำลองการทำงานของคำสั่ง for loop
ที่มา : https://www.tutorialspoint.com/python/python_for_loop.htm

ตัวอย่างโปรแกรมที่ 2.1

Program Example 2.1 : for

```
1  # Testing for loop
2  for letter in 'Python':                    # First Example
3          print('Current Letter :', letter)
4  fruits = ['banana', 'apple',  'mango']
5  for fruit in fruits:                       # Second Example
6          print('Current fruit :', fruit)
7  print("Good bye!")
```

OUTPUT

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
```

จากโปรแกรมตัวอย่างที่ 2.1 บรรทัดที่ 2 คำสั่ง for เริ่มอ่านข้อมูลสตริงเข้ามาทีละตัวอักษรและเก็บไว้ในตัวแปร letter ต่อจากนั้นโปรแกรมจะทำงานในบรรทัดที่ 3 โดยการพิมพ์ตัวอักษรที่เก็บอยู่ใน letter ออกมา ซึ่งตัวอักษรตัวแรกในสตริงชุดนี้คือ ‘P’ หลังจากนั้นโปรแกรมจะกลับไปเริ่มต้นอ่านข้อมูลใหม่ ซึ่งอักษรตัวที่ 2 ในสตริงคือ ‘y’ โดยเก็บทับตัวอักษรเดิมที่อยู่ในตัวแปร letter จากนั้นโปรแกรมจะพิมพ์ตัวอักษรออกทางจอภาพเช่นเดิม โปรแกรมจะทำงานในลักษณะอย่างนี้ไปเรื่อย ๆ จนถึงอักษรตัวสุดท้ายในสตริง (ในที่นี้คือ n) จึงจะยุติการทำงานของ for ผลลัพธ์ที่ได้คือ ข้อความ “Python”

บรรทัดที่ 4 โปรแกรมจะกำหนดค่าข้อมูลให้กับตัวแปรลิสต์ชื่อ fruits คือ ‘banana’ , ‘apple’ และ ‘mango’ ตามลำดับ ต่อจากนั้นในบรรทัดที่ 5 คำสั่ง for จะทำการอ่านข้อมูลสมาชิกตำแหน่งแรกจากตัวแปร fruits คือ ‘banana’ มาเก็บไว้ในตัวแปร fruit ในบรรทัดที่ 6 โปรแกรมจะพิมพ์ข้อมูลที่เก็บอยู่ในตัวแปร fruit ออกจอภาพ ผลลัพธ์คือ ‘banana’ เมื่อพิมพ์ข้อมูลเสร็จ โปรแกรมจะกลับไปเริ่มต้นคำสั่ง for อีกครั้ง โดยโปรแกรมจะทำคำสั่งเหมือนเดิมอย่างนี้ไปเรื่อย ๆ จนกว่าข้อมูลของสมาชิกที่อยู่ในตัวแปร fruits จะหมด คำสั่ง for จึงจะยุติการทำงาน ผลลัพธ์ที่ได้คือ ข้อความ ‘banana’ , ‘apple’ และ ‘mango’ ตามลำดับ

การอ้างถึงข้อมูลสมาชิกโดยการชี้ตำแหน่งของ for loop

คำสั่ง range เป็นคำสั่งที่ช่วยสร้างช่วงของข้อมูล เช่น เมื่อใช้คำสั่ง range(5) ช่วงข้อมูลที่ได้คือ 0, 1, 2, 3, 4 ดังนั้นเราสามารถนำ range มาประยุกต์ใช้กับ for ได้ โดยข้อมูลที่สร้างโดย range จะถูกนำไปใช้เป็นตัวชี้ตำแหน่งของรายการข้อมูลได้ ดังตัวอย่างโปรแกรมที่ 2.2

ตัวอย่างโปรแกรมที่ 2.2

Program Example 2.2 : for and range

```
1  # for loop and index
2  fruits = ['banana', 'apple',  'mango']
3  for index in range(len(fruits)):
4          print('Current fruit :', fruits[index])
5  print("Good bye!")
```

OUTPUT

```
Current fruit : banana
Current fruit : apple
Current fruit : mango
Good bye!
```

จากโปรแกรมตัวอย่างที่ 2.2 บรรทัดที่ 2 เป็นการกำหนดค่าใช้กับตัวแปรลิสต์ชื่อ fruits มีค่าเป็น ‘banana’ , ‘apple’ , ‘mango’ ตามลำดับ ต่อจากนั้นในบรรทัดที่ 3 คำสั่ง for จะสร้างช่วงของข้อมูลโดยอาศัยคำสั่ง range คือค่า 0, 1, 2 (ซึ่งเกิดจากการหาจำนวนสมาชิกของตัวแปร fruits โดยใช้ฟังก์ชัน len(fruits) เพราะ fruits มีสมาชิก 3 ตัว) ลำดับต่อไป คำสั่ง for จะทำการอ่านข้อมูลที่สร้างโดย range มาทีละค่า โดยเริ่มต้นที่ 0 และเก็บค่าดังกล่าวไว้ในตัวแปร index (index = 0) ต่อจากนั้นในบรรทัดที่ 4 โปรแกรมจะพิมพ์ข้อความว่า “Current fruit :” พร้อมกับข้อมูลที่ดึงมาจากตัวแปร fruits ในตำแหน่งที่ index มีค่าเท่ากับ 0 คือ fruits[index] = fruit[0] = ‘banana’ ออกทางจอภาพ ลำดับต่อไปโปรแกรมจะกลับไปเริ่มต้นคำสั่ง for ใหม่เพราะข้อมูลใน range ยังไม่หมด โดยดึงค่าข้อมูลในตำแหน่งถัดไปของตัวแปร fruits มาแสดง การทำงานจะทำซ้ำอย่างนี้ไปเรื่อย ๆ จนกว่าสมาชิกใน range จะหมด ผลจากการทำงานของโปรแกรมคือโปรแกรมจะพิมพ์ข้อความ ‘banana’ , ‘apple’ , mango’ สำหรับบรรทัดสุดท้ายโปรแกรมจะพิมพ์ข้อความ ‘Good bye!’ ก่อนจบโปรแกรมเสมอ

ฟังก์ชัน range

จากที่กล่าวไปแล้วว่า for นั้นนิยมใช้งานกับคำสั่ง range เสมอ ดังนั้นในย่อหน้านี้จะกล่าวถึงคำสั่ง range เพิ่มเติมอีกเล็กน้อย เพื่อให้การใช้คำสั่ง range ร่วมกับ for ได้ดีขึ้น โดยปกติคำสั่ง range จะมีรูปแบบคำสั่ง 4 แบบ ดังนี้

1. **range(x)** จะสร้างชุดของข้อมูลเริ่มจาก 0 ถึง (x -1) โดยเพิ่มขึ้นครั้งละ 1 เช่น เมื่อเรียกใช้ฟังก์ชัน range(6) ข้อมูลที่ถูกสร้างขึ้นคือ [0, 1, 2, 3, 4, 5]
2. **range(x, y)** จะสร้างชุดของข้อมูลเริ่มต้นจาก x ถึง (y – 1) โดยเพิ่มขึ้นครั้งละ 1 เช่น เมื่อเรียกใช้ฟังก์ชัน range(3, 10) ข้อมูลที่ถูกสร้างขึ้นคือ [3, 4, 5, 6, 7, 8, 9]
3. **range(x, y, i)** จะสร้างชุดข้อมูลเริ่มต้นจาก x ถึง (y – 1) โดยเพิ่มขึ้นครั้งละ i เช่น เมื่อเรียกใช้ฟังก์ชัน range(3, 15, 2) ข้อมูลที่ถูกสร้างขึ้นคือ [3, 5, 7, 9, 11, 13] มีข้อสังเกตสำหรับค่า y ตัวสุดท้ายคือ 15 จะไม่ถูกนำมาใส่ไว้ในรายการด้วย เนื่องจากติดเงื่อนไขที่ค่า y = y – 1 ดังนั้น ค่า y ที่ได้คือ 14 โปรแกรมจึงตัดทั้ง 14 และ 15 ทิ้งไปด้วย เพราะไม่อยู่ในเงื่อนไขทั้งคู่
4. **range(y, x, -i)** จะสร้างชุดของข้อมูลแบบย้อนหลัง จาก y ถึง (x – 1) โดยลดลงครั้งละ -i เช่น เมื่อเรียกใช้ฟังก์ขัน range(15, 3, -2) ข้อมูลที่ถูกสร้างขึ้นคือ [15, 13, 11, 9, 7, 5] ตามลำดับ ถ้าต้องการให้ลดค่าทีละ 1 โดยเริ่มตั้งแต่ 15 ถอยหลังไปจนถึง 1 มีรูปแบบคือ range(15, 0, -1) หรือถ้าต้องการให้เป็นค่าที่ติดลบ โดยเริ่มตั้งแต่ 0 ลงไปถึง -4 และลดค่าครั้งละ 1 มีรูปแบบคือ range(0, -5, -1)

ตัวอย่างโปรแกรมที่ 2.3

Program Example 2.3 : for and range exmples

```
1  # explain function range
2  import sys
3  sys.stdout.write("Show range (6) = ")
4  for i in range(6):
5      sys.stdout.write(str(i) + " ")
6  print("\n" + "-" *70)
7  sys.stdout.write("Show range (1, 7) = ")
8  for j in range(1, 7):
9      sys.stdout.write(str(j) + " ")
10 print("\n" + "-" *70)
11 sys.stdout.write("Show range (1, 36, 2) = ")
12 for o in range(1, 36, 2):
13     sys.stdout.write(str(o) + " ")
14 print("\n" + "-" *70)
```

OUTPUT

```
Show range (6) = 0 1 2 3 4 5
--------------------------------------------------------------------------------------------
Show range (1, 7) = 1 2 3 4 5 6
--------------------------------------------------------------------------------------------
Show range (1,36,2) = 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35
--------------------------------------------------------------------------------------------
```

จากโปรแกรมตัวอย่างที่ 2.3 บรรทัดที่ 2 เป็นคำสั่งนำเข้าไลบรารี่ sys เพื่อเรียกใช้เมธอด sys.stdout.write() เนื่องจากคำสั่ง print จะไม่ยอมให้พิมพ์ข้อความเว้นวรรคติดกันแถวเดียวกันได้ (ปกติจะขึ้นบรรทัดใหม่) ดังนั้นจึงจำเป็นต้องนำเมธอด sys.stdout.write() มาช่วยในการแสดงผลข้อมูลให้อยู่ในบรรทัดเดียวกัน บรรทัดที่ 6, 10 และ 14 คือคำสั่ง print("\n" + "-" *7) เป็นการสั่งให้ขึ้นบรรทัดใหม่ โดยใช้รหัส '\n' และทำการพิมพ์ ' – ' จำนวน 70 ครั้ง ติดต่อกันในบรรทัดเดียว

การใช้ else statement กับ for loop

Python อนุญาตให้ใช้ else กับ for loop ได้ ซึ่งแตกต่างจากภาษาอื่น ๆ โดยทั่วไป เป้าหมายสำคัญของการใช้ else กับ for นั้น เพื่ออำนวยความสะดวกให้กับผู้เขียนโปรแกรมในกรณีที่เงื่อนไขใน for เป็นเท็จ โดยปกติจะออกจากคำสั่ง for ไปโดยอัตโนมัติ บางครั้งผู้เขียนโปรแกรมไม่ทราบเลยว่าเกิดข้อผิดพลาดอะไรขึ้นในลูป for ดังนั้นคำสั่ง else จึงช่วยให้ผู้เขียนโปรแกรมสามารถตรวจสอบความผิดปกติใน for ได้อีกทางหนึ่ง หรือผู้เขียนโปรแกรมต้องการทำงานบางอย่างหลังจากจบการทำงานใน for เรียบร้อยแล้ว สำหรับการใช้ else กับ for และ while นั้นมีข้อพิจารณาดังนี้

1) คำสั่ง else เมื่อถูกใช้กับ for loop: คำสั่ง else จะถูกประมวลผลเมื่อ คำสั่งใน for ถูกประมวลผลครบหมดแล้ว
2) คำสั่ง else เมื่อถูกใช้กับ while loop: คำสั่ง else จะถูกประมวลผลเมื่อ เงื่อนไขใน while เป็นเท็จ

ตัวอย่างโปรแกรมที่ 2.4 โปรแกรมหาค่าจำนวนเฉพาะ

Program Example 2.4 : else witch for

```
1  # Testing elase and for loop with prime number
2  for num in range(10, 20):              #to iterate between 10 to 20
3      for i in range(2, num):            #to iterate on the number
4          if num % i == 0:               #to determine the first factor
5              j = num / i                #to calculate the second factor
6              print('%d equals %d * %d' % (num, i, j))
7              break
8      else:                              # else part of the loop
9          print(num, 'is a prime number')
```

OUTPUT

```
10 equals 2 * 5
11 is a prime number
12 equals 2 * 6
13 is a prime number
14 equals 2 * 7
15 equals 3 * 5
16 equals 2 * 8
17 is a prime number
18 equals 2 * 9
19 is a prime number
```

จากโปรแกรมตัวอย่างที่ 2.4 บรรทัดที่ 2 คำสั่ง for เริ่มต้นอ่านข้อมูลจำนวนเต็มครั้งละ 1 ค่า จาก range (สร้างช่วงของเลขจำนวนเต็มบวกตั้งแต่ 10 ถึง 20) และเก็บไว้ในตัวแปร num ลำดับต่อมาในบรรทัดที่ 3 เป็นคำสั่ง for ซ้อนอีกชั้นหนึ่ง โดยคำสั่ง for (ชั้นใน) จำนำค่าในตัวแปร num ที่รับมาจาก for ชั้นนอก มาสร้างเป็นช่วงข้อมูลด้วยคำสั่ง range(2, num) ให้กับ for ที่อยู่ชั้นใน ข้อมูลที่สร้างขึ้นจะถูกอ่านเข้ามาทำงานทีละค่า โดยเก็บไว้ในตัวแปร i บรรทัดที่ 4 นำค่าข้อมูลที่อยู่ใน i%num ผลที่ได้จะถูกนำไปตรวจสอบด้วยคำสั่ง if ว่าเท่ากับ 0 หรือไม่ (เป็นการตรวจสอบตัวเลขที่ไม่ใช่ค่า prime number) ถ้าผลลัพธ์มีค่าเท่ากับ 0 แสดงว่าตัวเลขดังกล่าวสามารถหารด้วยตัวเลขใด ๆ ลงตัว (ยกเว้นตัวเอง) แสดงว่าไม่ใช้จำนวนเฉพาะ ให้โปรแกรมทำงานของ for ในรอบนั้น ๆ ทันที ด้วยคำสั่ง break (ในบรรทัดที่ 7) แต่ถ้าไม่มีค่า i ใด ๆ ที่ได้จากชุดของข้อมูลที่สร้างจาก range(2, num) % num ได้ลงตัวเลย ในแต่ละรอบการทำงานของ for ชั้นใน แสดงว่าตัวเลขดังกล่าวเป็นจำนวนเฉพาะ ดังนั้นโปรแกรมจะทำงานหลังคำสั่ง else (บรรทัดที่ 9) โดยพิมพ์ข้อความว่า “X is a prime number” (โดย X คือจำนวนเฉพาะ) โปรแกรมจะทำงานวนซ้ำเพื่อค้นหาจำนวนเฉพาะอย่างนี้ไปเรื่อย ๆ จนกว่าค่า num ของ for ลูปนอกจะมีค่าเท่ากับ 20 โปรแกรมจึงจะยุติการทำงาน

ตัวอย่างโปรแกรมที่ 2.5 โปรแกรม for loop กับสตริง

Program Example 2.5 : string witch for loop

```
1  string = “Hello Word”
2  for x in string :
3      print (x)
```

OUTPUT

```
H
e
l
l
o

W
o
r
l
d
```

ตัวอย่างโปรแกรมที่ 2.6 โปรแกรม for loop กับลิสต์

Program Example 2.6 : list witch for loop

```
1  collection = ['hey', 5, 'd']
2  for x in collection:
3      print (x)
```

OUTPUT

```
Hey
5
d
```

ตัวอย่างโปรแกรมที่ 2.7 โปรแกรม for loop กับลิสต์ซ้อนลิสต์

Program Example 2.7 : list[list] witch for loop

```
1  list_of_lists = [ [1, 2, 3], [4, 5, 6], [7, 8, 9]]
2  for list in list_of_lists:
3      for x in list:
4          print (x)
```

OUTPUT

```
1
2
3
4
5
6
7
8
9
```

ตัวอย่างโปรแกรมที่ 2.8 โปรแกรม for loop กับดิกชันนารี่

Program Example 2.8 : dictionary witch for loop

```
1  knights = {'gallahad': 'the pure', 'robin': 'the brave'}
2  for k, v in knights.items():
3          print ("key = %s and value = %s"%(k, v))
```

OUTPUT

```
key = gallahad and value = the pure
key = robin and value = the brave
```

3. คำสั่งควบคุมการทำซ้ำ (Loop control statements)

คำสั่งควบคุมการทำซ้ำถูกสร้างขึ้นเพื่อต้องการเปลี่ยนทิศทางของการประมวลผลแบบวนซ้ำ ซึ่งตามปกติแล้ว การประมวลผลจะเป็นแบบลำดับจากซ้ายไปขวา และจากด้านบนลงล่าง (Sequence) แต่สถานการณ์บางอย่างการทำงานแบบลำดับอาจจะไม่เหมาะสม เช่น โปรแกรมต้องการค้นหาข้อความในสตริงที่มีความยาวมาก ๆ เมื่อค้นหาไปเรื่อย ๆ ปรากฏว่าโปรแกรมพบข้อความที่ต้องการดังกล่าวอยู่ระหว่างกลางของข้อความ ผู้เขียนโปรแกรมควรจะหยุดการทำงานวนซ้ำ เนื่องจากจะทำให้ประหยัดเวลาในการคำนวณ จากตัวอย่างที่กล่าวมาแล้วเมื่อผู้เขียนโปรแกรมบังคับให้โปรแกรมหลุดออกจากคำสั่งวนซ้ำ จะส่งผลให้ตัวแปรหรืออ๊อปเจ็กต์ต่าง ๆ ที่ใช้งานอยู่ภายในขอบเขตของการวนซ้ำนั้น หมดอายุการทำงาน คือจะถูกลบออกจากหน่วยความจำไปโดยอัตโนมัติ Python สนับสนุนคำสั่งควบคุมการทำงานซ้ำ 3 คำสั่ง คือ break, continue และ pass

คำสั่ง break

เป็นคำสั่งที่สั่งให้โปรแกรมยุติการทำซ้ำ (Loop) ซึ่งส่งผลให้คำสั่งที่เหลือทั้งหมดหลังคำสั่ง break และอยู่ภายในขอบเขตของคำสั่งทำซ้ำ ไม่ถูกประมวลผลไปด้วย เมื่อออกจากขอบเขตของคำสั่งการทำงานซ้ำปัจจุบันแล้ว โปรแกรมจะประมวลคำสั่งอื่น ๆ ต่อไป (ไม่ใช่การจบการทำงานของโปรแกรม จะยุติการทำงานเฉพาะคำสั่งใน Loop ปัจจุบันเท่านั้น)

ตัวอย่างโปรแกรมที่ 3.1

Program Example 3.1 : break for while & for

```
1  # Break for while and for loop
2  for letter in 'Python':                          #First example for for loop
3      if letter == 'h':
4          break                                    #Break force exit for loop
5      print('Current Letter :', letter)
6  var = 10  #Second example while loop
7  while var > 0:
8      print('Current variable value :', var)
9      var = var – 1
10     if var == 5:
11         break                                    #Break force exit while loop
12 print("Good bye!")
```

OUTPUT

```
Current Letter : P
Current Letter : y
Current Letter : t
Current variable value : 10
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Good bye!
```

คำสั่ง continue

เป็นคำสั่งที่สั่งให้โปรแกรมกลับไปเริ่มต้นใหม่ที่ต้น loop ซึ่งส่งผลให้คำสั่งที่เหลือทั้งหมดหลังคำสั่ง continue อยู่ภายในขอบเขตของคำสั่งทำซ้ำ จะไม่ถูกประมวลผลในรอบนั้น ๆ ไปด้วย (แต่ไม่ได้ออกจากคำสั่งการทำซ้ำ) คำสั่ง continue จะใช้ได้ทั้ง while และ for loop

ตัวอย่างโปรแกรมที่ 3.2

Program Example 3.2 : continue for while & for

```
1  #Continue for while and for loop
2  for letter in 'Python':                    # First example for for loop
3       if letter == 'h':
4            continue
5       print('Current Letter :', letter)
6  var = 10                                   # Second example for while loop
7  while var > 0:
8       var = var -1
9       if var == 5:
10           continue
11      print('Current variable value :', var)
12 print("Good bye!")
```

OUTPUT

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
Current variable value : 9
Current variable value : 8
Current variable value : 7
Current variable value : 6
Current variable value : 4
Current variable value : 3
Current variable value : 2
Current variable value : 1
Current variable value : 0
Good bye!
```

คำสั่ง pass

เป็นคำสั่งที่มีไว้เพื่อรักษาโครงสร้างหรือความหมายของโปรแกรมไว้ เช่น กรณีที่ผู้เขียนโปรแกรมกำลังเขียนโปรแกรมอยู่ แต่ปรากฏว่าในบางจุดของโปรแกรม ผู้เขียนยังไม่แน่ใจว่าจะดำเนินการต่ออย่างไรและต้องการยกเว้นโปรแกรมตรงส่วนนี้ไว้ก่อน แล้วจึงค่อยมาแก้ไขในภายหลัง ผู้เขียนโปรแกรมสามารถบรรจุคำสั่ง pass นี้ไว้เพื่อให้โปรแกรมสามารถทดสอบรันโปรแกรมได้และประมวลผลโปรแกรมในส่วนที่ละเว้นไว้ด้วย ซึ่งคำสั่ง pass จะไม่มีการประมวลผลใด ๆ เกิดขึ้น และเมื่อไม่ต้องการใช้งานคำสั่งดังกล่าวแล้ว ไม่จำเป็นต้องลบทิ้งก็ได้ ดังตัวอย่างโปรแกรมที่ 3.3

ตัวอย่างโปรแกรมที่ 3.3

Program Example 3.3 : pass

```
1  # Testing pass command
2  for letter in 'Python':
3      if letter == 'h':
4          pass
5          print('This is pass block')
6      print('Current Letter :', letter)
7  print("Good bye!")
```

OUTPUT

```
Current Letter : P
Current Letter : y
Current Letter : t
This is pass block
Current Letter : h
Current Letter : o
Curent Letter : n
Good bye!
```

Lists python

List (ลิสต์) คือโครงสร้างข้อมูลชนิดหนึ่งในภาษา Python ที่ใช้เก็บข้อมูลแบบลำดับ (Sequence) โดยมี Index เป็นตัวระบุตำแหน่งในการเข้าถึงข้อมูล เราสามารถใช้ List เพื่อเก็บข้อมูลจำนวนมากและหลากหลายประเภทในเวลาเดียวกัน List เป็นประเภทข้อมูลที่ใช้อย่างหลากหลายในการเขียนโปรแกรม การประกาศและใช้งาน List

List นั้นเป็นตัวแปรประเภทหนึ่ง การใช้งานจะเหมือนกันอาเรย์ในภาษาอื่นๆ ในการประกาศ List นั้น ข้อมูลของ List จะอยู่ภายในเครื่องหมาย [ ] และคั่นสมาชิกแต่ละตัวด้วยเครื่องหมาย ,

ตัวอย่างการประกาศ List ในภาษา Python

line	code
1	my_list = [] # list ว่าง
2	my_list = [1, 2, 3] # list จำนวนเต็ม
3	my_list = [1, "Hello", 3.4] # list แบบผสม
4	my_list = [1,2,3,[4,5,6],"PCSHS"] # list แบบผสมซ้อน list และ string

ตัวอย่างการเข้าถึง List ในภาษา Python

List นั้นใช้ Index สำหรับการเข้าถึงข้อมูล โดย Index ของ List จะเป็นจำนวนเต็มที่เริ่มจาก 0 และเพิ่มขึ้นทีละ 1 ไปเรื่อยๆ ดังนั้น เราจึงสามารถเข้าถึงข้อมูลภายใน List เพื่ออ่านหรืออัพเดทค่าได้โดยตรงผ่าน Index โดยชื่อ List แล้วตามด้วย index

line	code
1	my_list = [] # list ว่าง
2	my_list = [1, 2, 3] # list จำนวนเต็ม
3	print(my_list[2]) # index list เริ่มที่ [0]
4	my_list = [1, "Hello", 3.4] # list แบบผสม
5	print(my_list[1][2])
6	my_list = [1,2,3,[4,5,6],"PCSHS"] # list แบบผสม
7	print(my_list[3][2])
ผล run	3 l 6

การอ่านค่าใน List ด้วยคำสั่ง For loop

line	code
1	num = [1,2,3,4,5,6,7,8,9,10]
2	for n in num:
3	print(n,end=" ") # สั่งพิมพ์ค่า n โดยไม่ขึ้นบรรทัดใหม่
ผล run	1 2 3 4 5 6 7 8 9 10

line	code
1	names = ['Toyota', 'Honda', 'Isuzu', 'Mitsubishi', 'MG']
2	#for i in range(len(names)):
3	for i in range(0,len(names)): # list แบบกำหนดค่าเริ่มต้น
4	print(names[i], end =' ') # สั่งพิมพ์ค่า name ที่ i โดยไม่ขึ้นบรรทัดใหม่
ผล run	Toyota Honda Isuzu Mitsubishi MG

การอ่านค่าใน list แบบมีเงื่อนไข โดยให้แสดงข้อมูลที่เป็นเลข คู่

line	code
1	num = [1,2,3,4,5,6,7,8,9,10]
2	for n in num:
3	if n%2==0: # ตรวจสอบสมาชิกว่าเป็นเลขคู่ หรือ คี่
4	print(n,end=" ") # สั่งพิมพ์ค่า n โดยไม่ขึ้นบรรทัดใหม่
ผล run	2 4 6 8 10

การอ่านค่าใน list แบบมีเงื่อนไข โดยให้แสดงข้อมูลที่เป็นเลข คี่ และ แสดง ค่าผลรวม

line	code
1	num = [1,2,3,4,5,6,7,8,9,10]
2	sum = 0 #กำหนดค่าเริ่มต้นของผลรวม
3	for n in num:
4	if n%2 !=0: # ตรวจสอบสมาชิกว่าเป็นเลขคู่ หรือ คี่
5	print(n,end=" ") # สั่งพิมพ์ค่า n โดยไม่ขึ้นบรรทัดใหม่
6	sum +=n # ทำการบวกสะสม
7	print("= {}".format(sum)) # แสดงค่าผลรวม
ผล run	1 3 5 7 9 = 25

การประยุกต์สร้างสมาชิกใน List ด้วยคำสั่ง for

line	code
1	pow2 =[2** x for x in range(10)]
2	print(pow2)
ผล run	[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]

line	code
1	print([2** x for x in range(10)])
ผล run	[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]