

INE-PYTHON

LECTURE - 04 REPETITION WHILE FOR

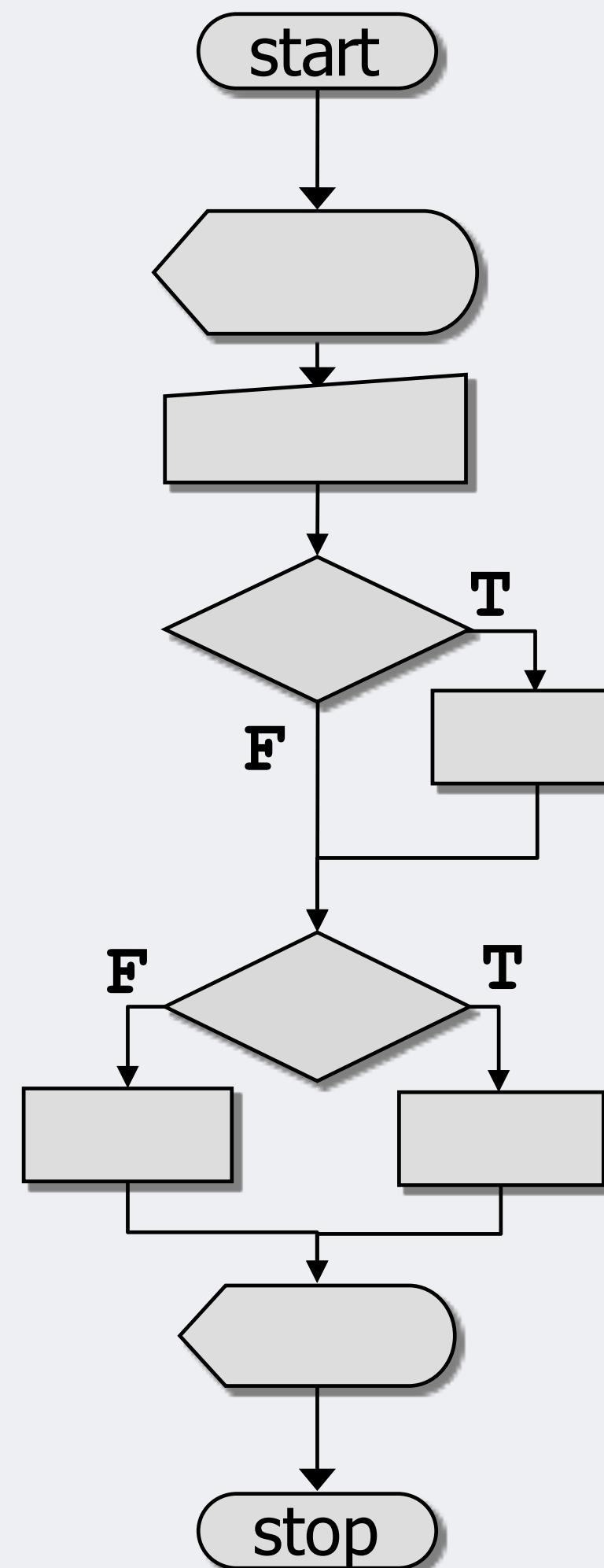
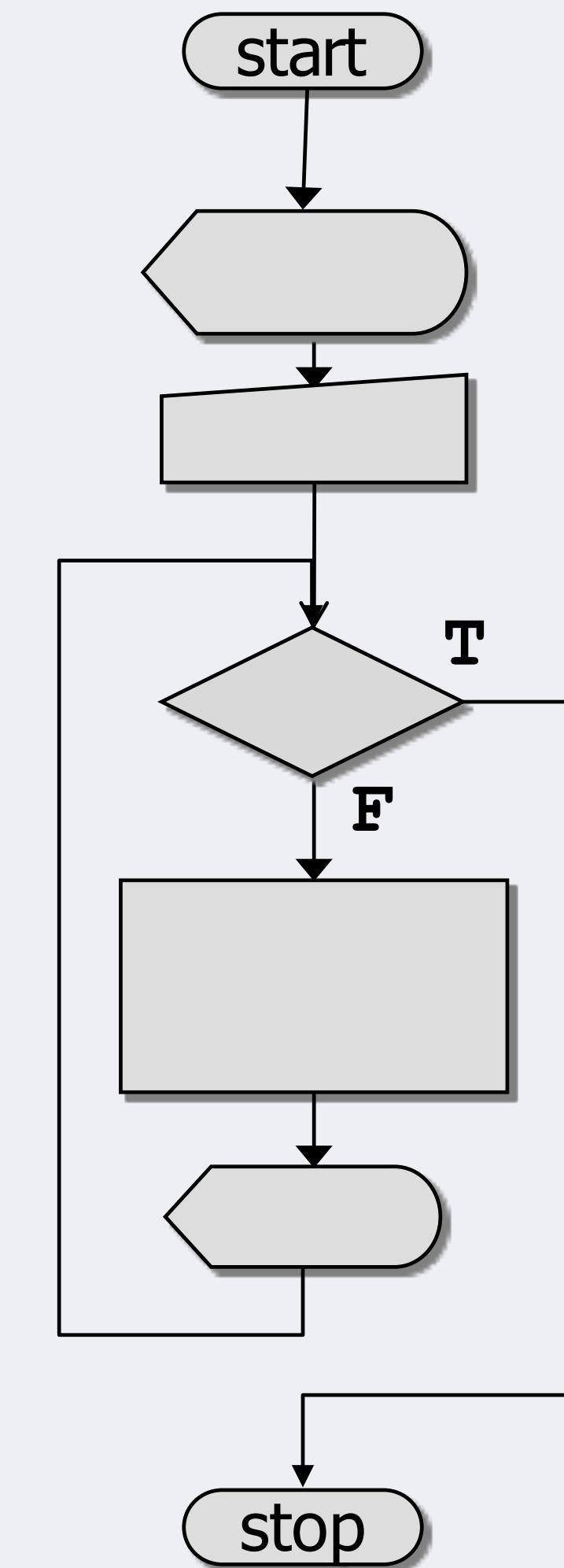
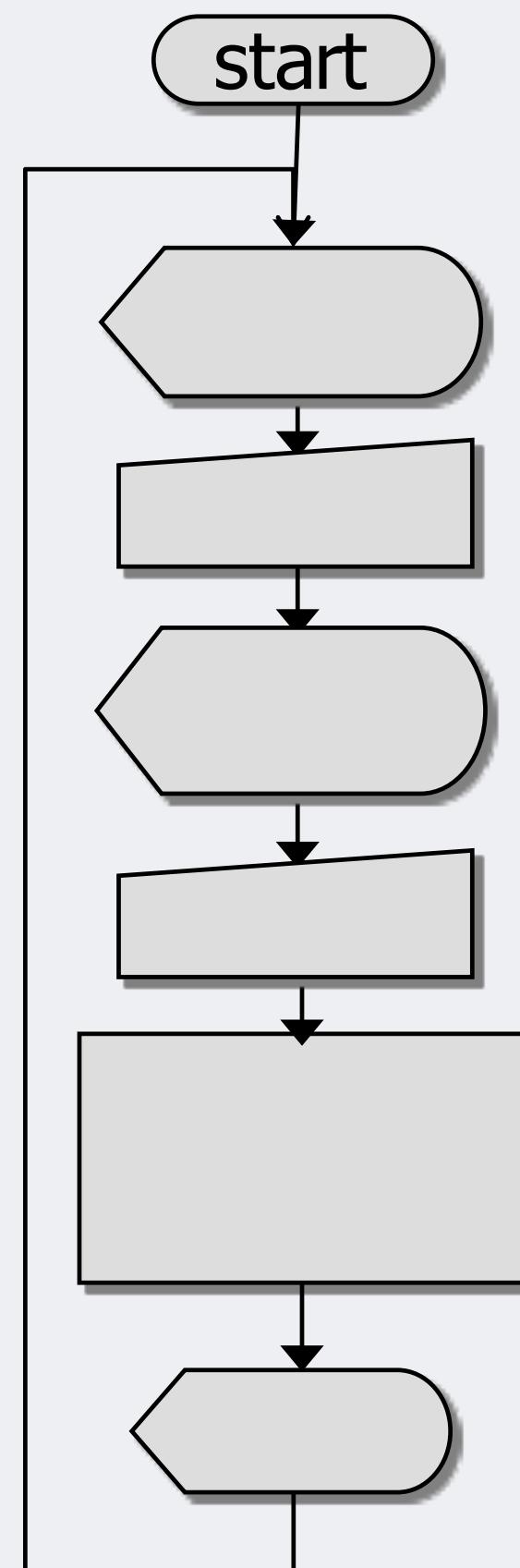
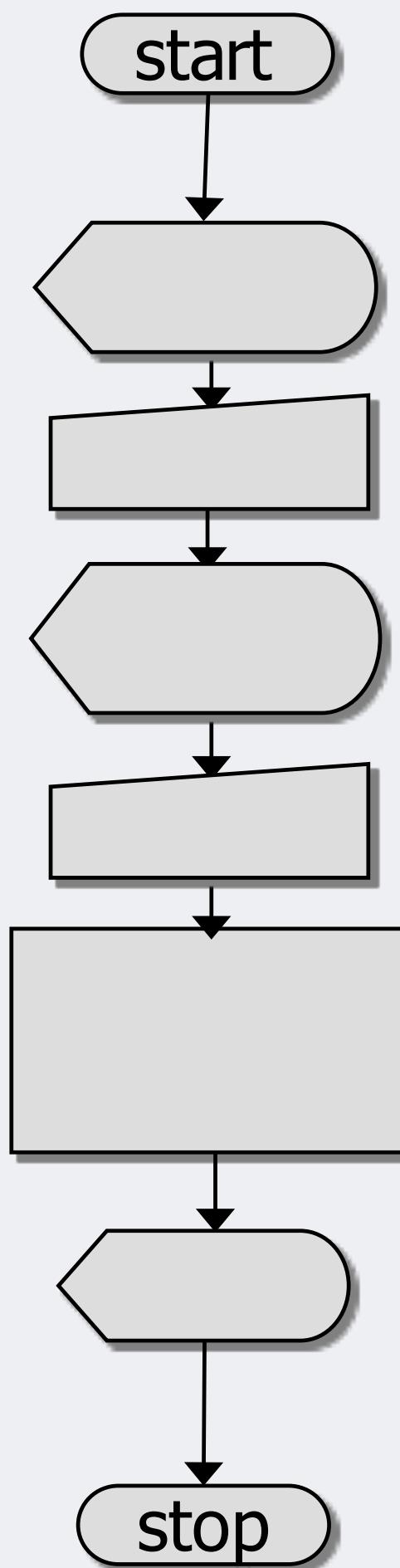
What You Will Learn

- Repetition Structure
- while Loop - A condition-controlled loop
- for Loop - A count-controlled loop
- Calculating Running Total
- Sentinels
- Input Validation Loops
- Nested Loops
- pass, continue, break ...

REPETITION STRUCTURES



PROGRAM FLOW



REPETITION STRUCTURE

```
# Get a salesperson's sales and commission rate.  
sales = float(input('Enter the amount of sales: '))  
comm_rate = float(input('Enter the commission rate: '))  
commission = sales * comm_rate  
print('The commission is $', format(commission, ',.2f'))  
  
# Get another salesperson's sales and commission rate.  
sales = float(input('Enter the amount of sales: '))  
comm_rate = float(input('Enter the commission rate: '))  
commission = sales * comm_rate  
print('The commission is $', format(commission, ',.2f'))  
  
# Get another salesperson's sales and commission rate.  
sales = float(input('Enter the amount of sales: '))  
comm_rate = float(input('Enter the commission rate: '))  
commission = sales * comm_rate  
print('The commission is $', format(commission, ',.2f'))
```

REPETITION STRUCTURE

โค้ดนี้เป็นโครงสร้างลำดับ (Sequence Structure) ที่ยาวต่อเนื่องและมีการเขียนโค้ดซ้ำกันจำนวนมาก ซึ่งแนวทางนี้มีข้อเสียหลายประการ เช่น:

- โค้ดที่ซ้ำกันทำให้โปรแกรมมีขนาดใหญ่เกินความจำเป็น
- การเขียนคำสั่งต่อเนื่องจำนวนมากใช้เวลามากและอาจเกิดข้อผิดพลาดได้ง่าย
- หากจำเป็นต้องแก้ไขบางส่วนของโค้ดที่ซ้ำกัน จะต้องทำการแก้ไขในหลายจุด ซึ่งเพิ่มความเสี่ยงในการเกิดข้อผิดพลาดและลดประสิทธิภาพในการดูแลรักษาโปรแกรม

LOOPS

ลูปช่วยให้คุณสามารถสั่งให้โค้ดทำงานซ้ำ ๆ ได้เมื่อเงื่อนไขที่กำหนดเป็นจริง ซึ่งเป็นสิ่งสำคัญในการจัดการกับงานที่ต้องทำซ้ำอย่างมีประสิทธิภาพในโปรแกรม หากไม่มีลูป คุณจะต้องเขียนโค้ดเดิมซ้ำหลายครั้ง ซึ่งทำให้โค้ดยาวขึ้น อ่านยาก และดูแลรักษายาก ลูปจึงช่วยลดข้อผิดพลาดและประหยัดเวลาในการเขียนโปรแกรม

ลูปใน Python แบ่งออกเป็น 2 ประเภทหลัก คือ ลูป for และลูป while ลูป for เมามำสำหรับกรณีที่ทราบจำนวนรอบของการทำงานล่วงหน้า โดยจะวนซ้ำตามลำดับของข้อมูล เช่น list, tuple, dictionary, set หรือ string เมามำสำหรับการประมวลผลรายการข้อมูล เช่น การแสดงค่าทุกตัวใน list

ในขณะที่ลูป while เมามำสำหรับกรณีที่จำนวนรอบของการทำงานไม่แน่นอน โดยจะทำงานซ้ำตราบใดที่เงื่อนไขที่กำหนดยังคงเป็นจริง ซึ่งเมามำสำหรับสถานการณ์ที่ขึ้นอยู่กับการคำนวณหรือการป้อนข้อมูลจากผู้ใช้

การใช้ลูปช่วยเพิ่มประสิทธิภาพ อ่านง่าย และบำรุงรักษาได้ง่ายขึ้น โดยการห่อหุ้มงานที่ทำซ้ำไว้ภายในลูป ทำให้โค้ดมีโครงสร้างชัดเจน เข้าใจง่าย และสามารถแก้ไขหรือขยายได้ง่าย ลูปจึงเป็นเครื่องมือสำคัญที่โปรแกรมเมอร์ทุกคนควรใช้ในการเขียนโปรแกรมที่มีความยืดหยุ่นและเชื่อถือได้

LOOPS

- คำจำกัดความและวัตถุประสงค์

- ลูปทำให้โค้ดทำงานซ้ำได้ตามเงื่อนไขที่กำหนด
- จำเป็นต่อการทำงานที่ซ้ำ ๆ และช่วยเพิ่มประสิทธิภาพของโค้ด

- กรณีการใช้งานทั่วไป

- วนซ้ำใน list, string และ iterable อื่น ๆ
- ใช้ฟังก์ชัน `range()` เพื่อสร้างลำดับตัวเลข
- คำนวณผลรวม แฟกทอรีเรียล และการวนลูปใน dictionary

- ลูปซ้อน (Nested Loops)

- ใช้ลูปหนึ่งชั้นอยู่ในอีกลูปเพื่อจัดการการทำงานที่ซับซ้อน

- ลูปไม่สิ้นสุด (Infinite Loops)

- ลูปที่ไม่มีจุดจบเนื่องจากเงื่อนไขไม่เคยเป็นเท็จ
- ควรมั่นใจว่าเงื่อนไขจะถูกตรวจสอบจนกว่าจะพบเงื่อนไขที่สุดเพื่อหลีกเลี่ยงลูปไม่รู้จบ

- ตัวอย่างการใช้งานจริง

- วนซ้ำผ่านชุดข้อมูล
- คำนวณผลรวมและแฟกทอรีเรียล
- วนผ่าน key และ value ใน dictionary

- ประสิทธิภาพและความเข้าใจง่าย

- ลูปช่วยลดการเขียนโค้ดซ้ำ และทำให้โค้ดอ่านง่ายขึ้น
- ทำให้โค้ดดูแลรักษาและดีบักได้ง่ายขึ้น

LOOP STRUCTURE

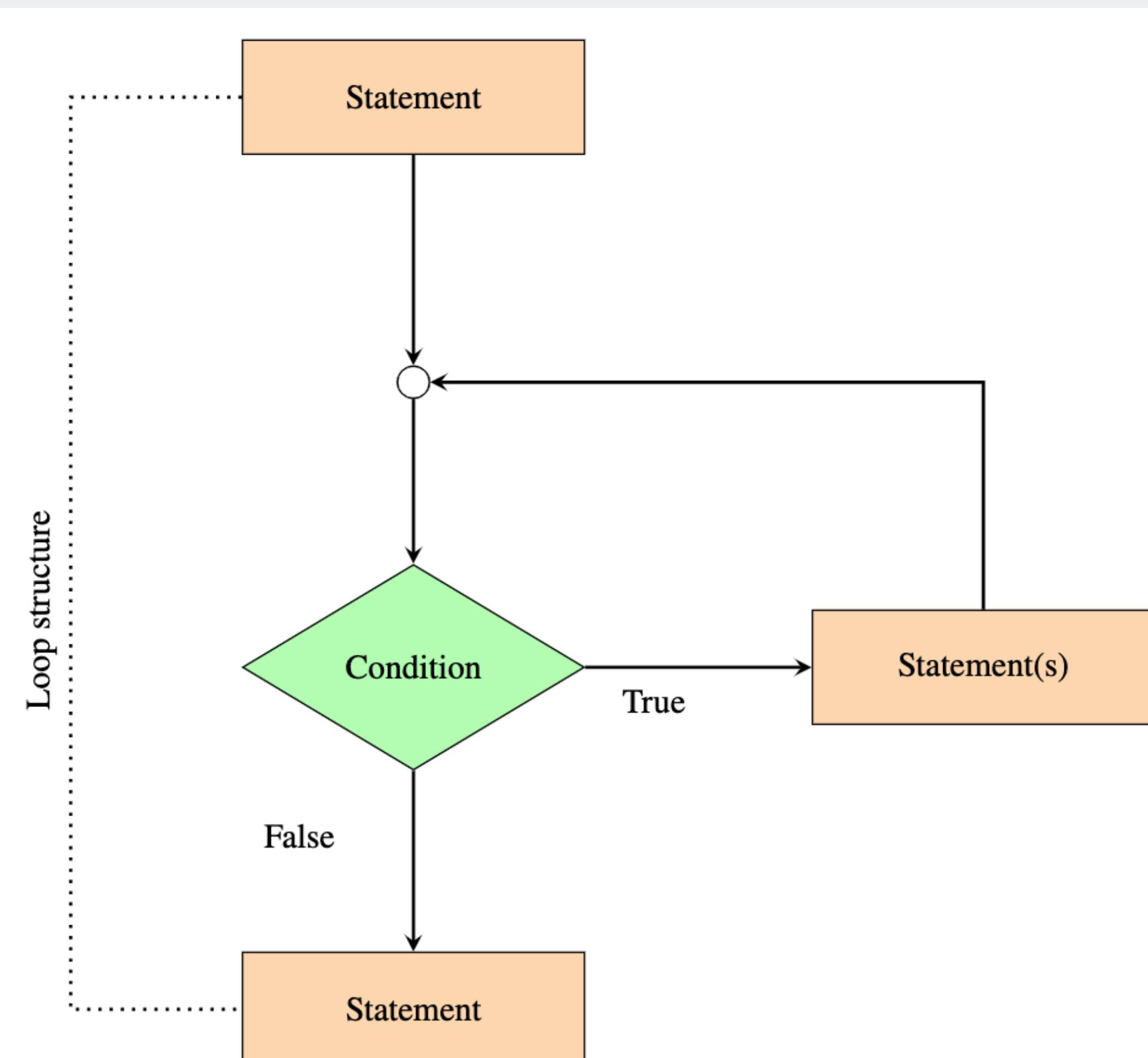


Fig. 4.1: Flowchart of Loop Structure

COUNT-CONTROLLED LOOP



for

LOOP FOR (COUNT-CONTROLLED LOOP)

ลูป `for` จะทำการวนซ้ำผ่านลำดับของข้อมูล ซึ่งรวมถึงโครงสร้างข้อมูลต่าง ๆ เช่น รายการ (list), ทูเพิล (tuple), ดิกชันนารี (dictionary), เซต (set), สตริง (string) และอ็อบเจกต์ที่สามารถวนซ้ำได้อื่น ๆ ซึ่งหมายความว่า ลูปจะวนผ่านแต่ละองค์ประกอบในลำดับโดยอัตโนมัติที่ละเอียดรายการ ช่วยให้คุณสามารถดำเนินการต่าง ๆ กับแต่ละองค์ประกอบได้โดยไม่ต้องจัดการกับดัชนี้เอง ตัวอย่างเช่น ในรายการของข้อมูล ลูป `for` สามารถเข้าถึงแต่ละรายการโดยตรง และดำเนินการโค้ดกับแต่ละรายการได้ ในทำนองเดียวกัน ลูป `for` สามารถวนซ้ำผ่านคีย์ ค่าหรือคู่คีย์-ค่าในดิกชันนารี ทำให้สามารถจัดการและดึงข้อมูลได้อย่างมีประสิทธิภาพ ความยืดหยุ่นของลูปประเภทนี้จึงมีความสำคัญอย่างยิ่งต่อภารกิจต่าง ๆ ตั้งแต่การวนซ้ำอย่างง่ายผ่านสตริง ไปจนถึงกระบวนการประมวลผลข้อมูลที่ซับซ้อนซึ่งเกี่ยวข้องกับหลายโครงสร้างข้อมูล ความสามารถนี้ในการจัดการกับลำดับและอ็อบเจกต์ที่วนซ้ำได้หลากหลายประเภทช่วยเพิ่มความยืดหยุ่นและประสิทธิภาพให้กับโค้ดของคุณได้อย่างมาก

แนวคิดสำคัญของลูป For:

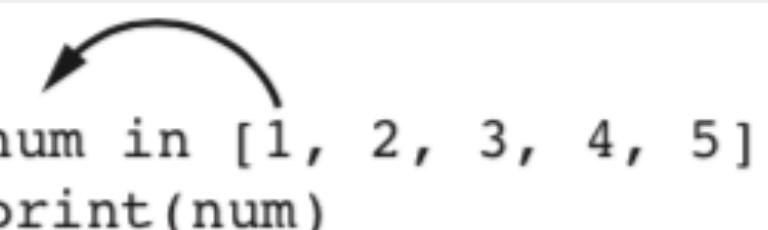
- วนซ้ำผ่านลำดับข้อมูล เช่น รายการ ทูเพิล ดิกชันนารี เซต หรือสตริง
- หมายความว่าท่านสามารถดำเนินการต่อไปในแต่ละขั้นตอนได้โดยไม่ต้องกำหนดจำนวนครั้ง

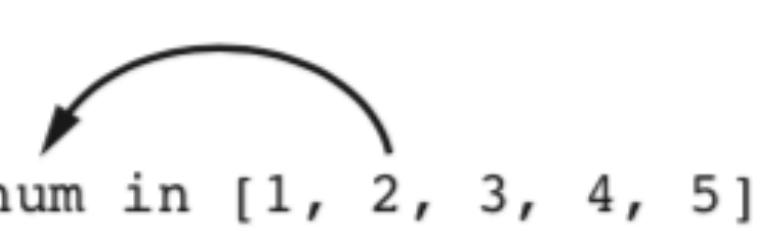
```
1 for variable in sequence:  
2     # code to execute
```

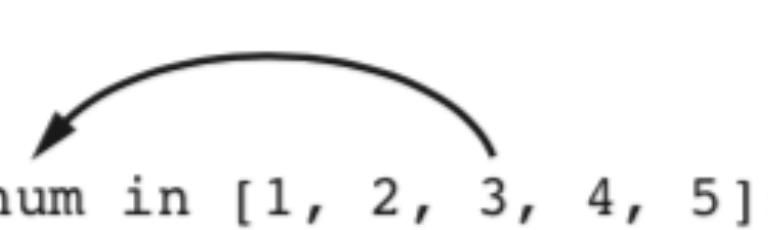
ITERATING OVER A LIST

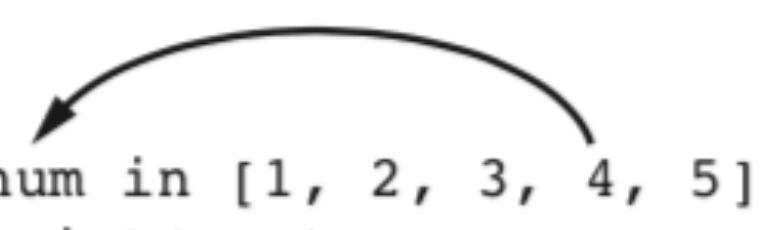
```
print('I will display the numbers 1 through 5.')
for num in[1,2,3,4,5]:
    print(num)
```

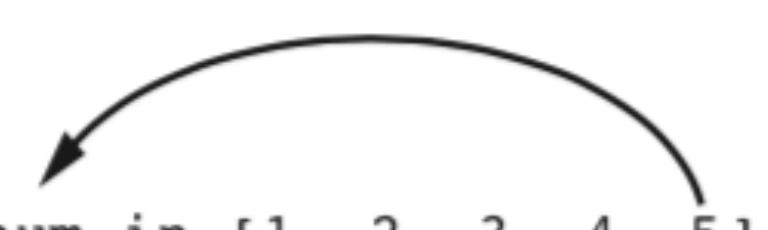
I will display the numbers 1 through 5.
1
2
3
4
5

1st iteration:  for num in [1, 2, 3, 4, 5]:
 print(num)

2nd iteration:  for num in [1, 2, 3, 4, 5]:
 print(num)

3rd iteration:  for num in [1, 2, 3, 4, 5]:
 print(num)

4th iteration:  for num in [1, 2, 3, 4, 5]:
 print(num)

5th iteration:  for num in [1, 2, 3, 4, 5]:
 print(num)

COUNT-CONTROLLED LOOP

Algorithm 9: อัลกอริทึมสำหรับแสดงชื่อผลไม้

Input: รายการของผลไม้
Output: ชื่อของผลไม้ที่แสดงผลออกมานะ

```
1 begin
2   fruits = ["apple", "banana", "cherry"];
3   for fruit in fruits do
4     Display fruit;
```

```
1 fruits = ["apple", "banana", "cherry"]
2 for fruit in fruits:
3   print(fruit)
```

Listing 4.2: Example of Iterating Over a List

**all of the statements in the block are indented

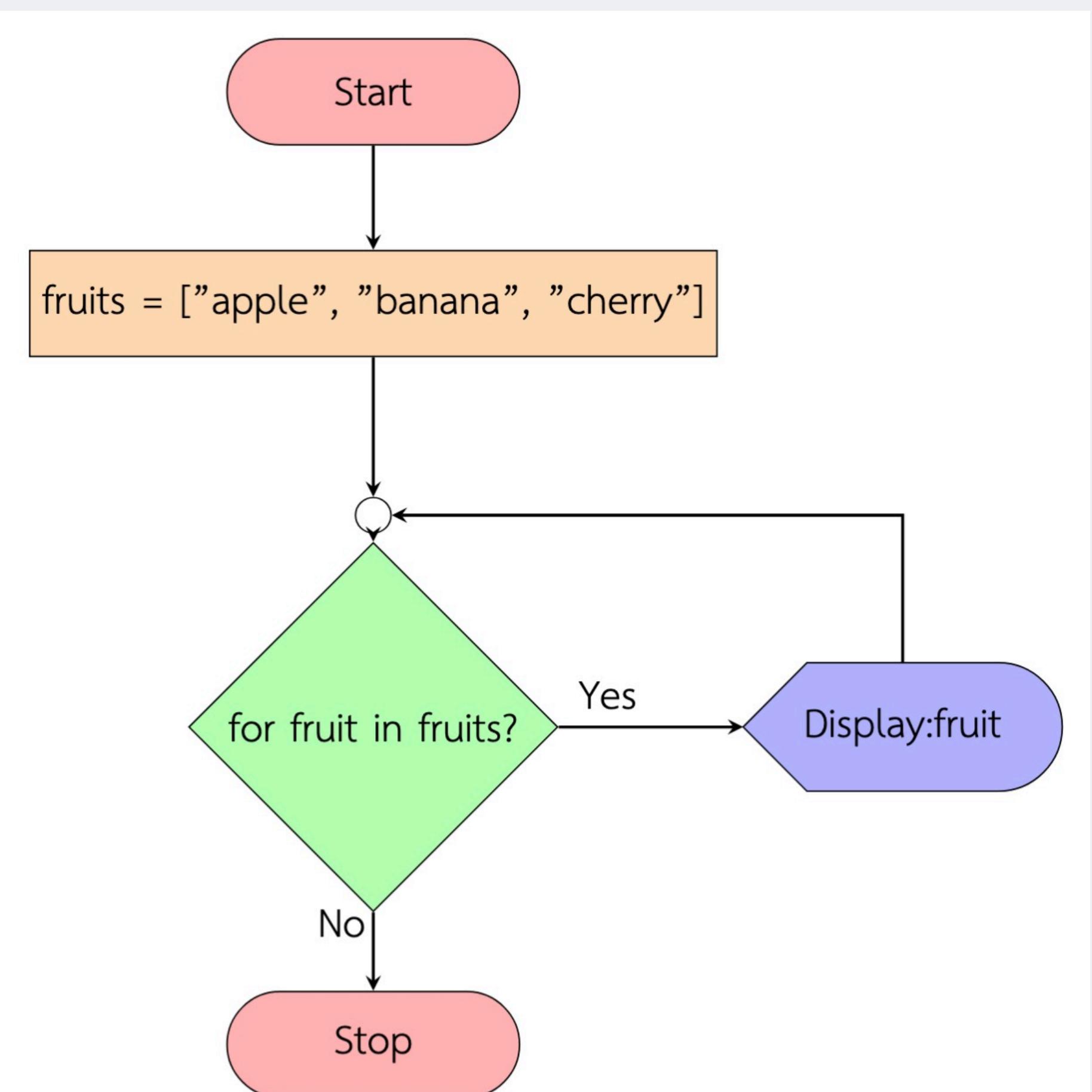


Figure 4.2: ผังงานสำหรับแสดงรายการผลไม้

ITERATING OVER A STRING

การวนซ้ำผ่านสตริงหมายถึงการใช้ลูปเพื่อเข้าถึงแต่ละตัวอักษรในสตริงแบบแยกกัน วิธีนี้ช่วยให้สามารถดำเนินการกับแต่ละตัวอักษรได้ เช่น การนับจำนวนครั้งที่เกิด การตรวจสอบตัวอักษรเฉพาะ หรือการแปลงแต่ละตัวอักษร จึงเป็นเทคนิคพื้นฐานสำหรับการจัดการกับสตริง

```
1 for char in "Hello":  
2     print(char)
```

Listing 4.3: Example of Iterating Over a String

ITERATING OVER A STRING

Algorithm 10: อัลกอริทึมสำหรับแทนที่สระในสตริงด้วยเครื่องหมายดอกจัน

Input: สตริงจากผู้ใช้

Output: สตริงที่ถูกแทนที่สระด้วยเครื่องหมายดอกจัน

```
1 begin
2   อ่านสตริงจากผู้ใช้;
3   กำหนดสตริงว่างสำหรับเก็บผลลัพธ์;
4   กำหนดตัวแปรเก็บสระเป็น "aeiouAEIOU";
5   for แต่ละตัวอักษรในสตริงอินพุต do
6     แปลงตัวอักษรเป็นตัวพิมพ์ใหญ่;
7     if ตัวอักษรเป็นสระ then
8       เพิ่มเครื่องหมายดอกจันลงในผลลัพธ์;
9     else
10      เพิ่มตัวพิมพ์ใหญ่ลงในผลลัพธ์;
11  แสดงสตริงที่ถูกปรับเปลี่ยนแล้ว;
```

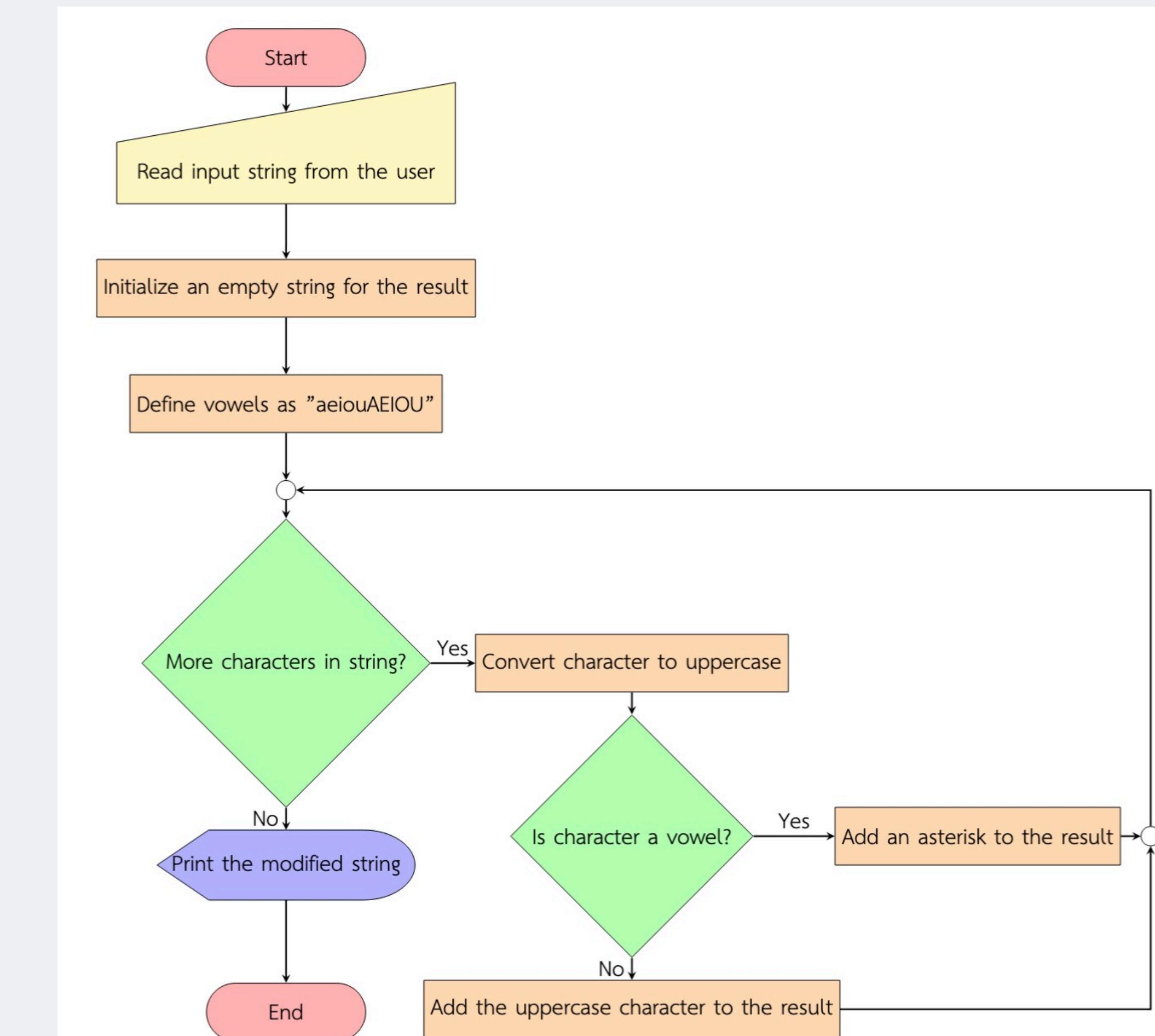


Figure 4.3: ผังงานสำหรับการปรับเปลี่ยนสตริงโดยแทนที่สระด้วยเครื่องหมายดอกจัน

ITERATING OVER A STRING

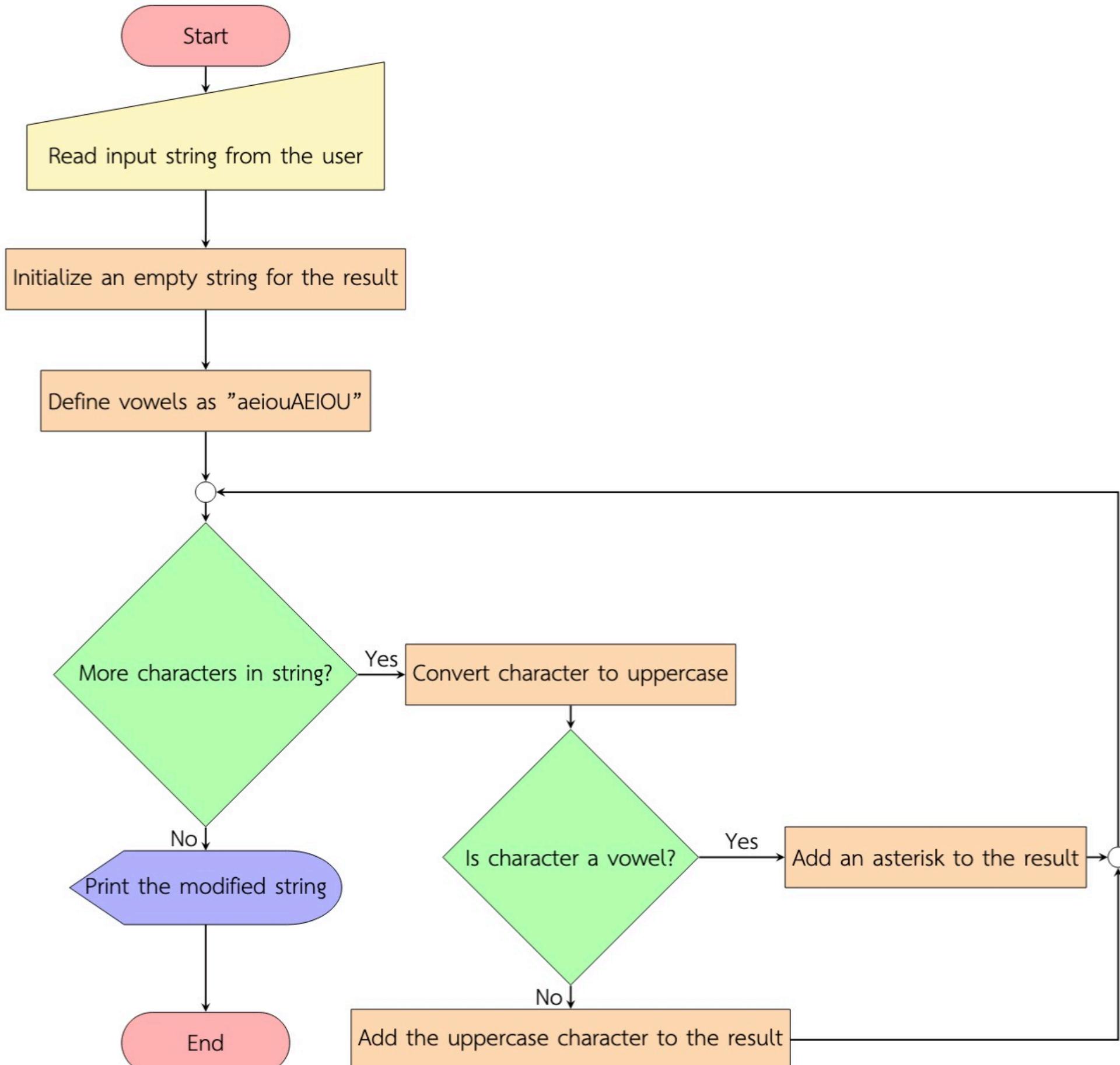


Figure 4.3: ผังงานสำหรับการปรับเปลี่ยนสตริงโดยแทนที่สระด้วยเครื่องหมายดอกจัน

```

1 # Input string from the user
2 input_string = input("Enter a string: ")
3
4 # Initialize an empty string for the modified result
5 modified_string = ""
6
7 # Define a set of vowels
8 vowels = "aeiouAEIOU"
9
10 # Use a for loop to iterate through the input string
11 for char in input_string:
12     # Convert character to uppercase
13     upper_char = char.upper()
14     # Replace vowels with asterisks
15     if upper_char in vowels:
16         modified_string += "*"
17     else:
18         modified_string += upper_char
19
20 # Print the modified string
21 print("Modified string:", modified_string)
  
```

The code example shows how to iterate over a string and replace vowels with asterisks. It starts by prompting the user for an input string. It then initializes an empty string for the modified result. A set of vowels is defined. A for loop iterates through each character of the input string. For each character, it is converted to uppercase. If the uppercase character is a vowel, an asterisk (*) is added to the modified string; otherwise, the uppercase character is added. Finally, the modified string is printed.

Listing 4.4: Example of String Modification

FOR RANGE()

ในลูป for การใช้ฟังก์ชัน range() ร่วมกับพารามิเตอร์ start, stop และ step ช่วยให้สามารถควบคุมการวนซ้ำผ่านลำดับตัวเลขได้อย่างยืดหยุ่น โดยพารามิเตอร์ start ใช้ระบุจุดเริ่มต้นของลำดับ stop เป็นค่าที่สิ้นสุด (แต่ไม่รวมค่า) และ step ใช้กำหนดระยะห่างระหว่างตัวเลขแต่ละตัว

```
1 # Example Syntax
2 for i in range(start, stop, step):
3     #Statement 1...
4     #Statement 2...
```

Listing 4.5: Syntax of for-range()

FOR RANGE()

```
for variable in range(start, stop, step):  
    statement  
    statement  
    etc.
```

```
for a in range(100):  
  
for a in range(1,100):  
  
for a in range(1,100,2):
```

FOR RANGE()

```
1 # Print numbers from 0 to 4
2 for i in range(5):
3     print(i)
4 #This will print the numbers 0, 1, 2, 3, and 4.
```

Listing 4.6: Example of Using the Range Function

```
1 # Print numbers from 3 to 9
2 for i in range(3,10):
3     print(i)
4 #This will print the numbers 3, 4, 5, 6, 7, 8, and 9.
```

Listing 4.7: Example of Using the Range Function with Start and Stop

```
1 # Print numbers from 1 to 10 with a step of 2
2 for i in range(1, 11, 2):
3     print(i)
4 #This will print the numbers 1, 3, 5, 7, and 9.
```

Listing 4.8: Example of Using the Function with Start Stop and Step

FOR RANGE()

Number	Square
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

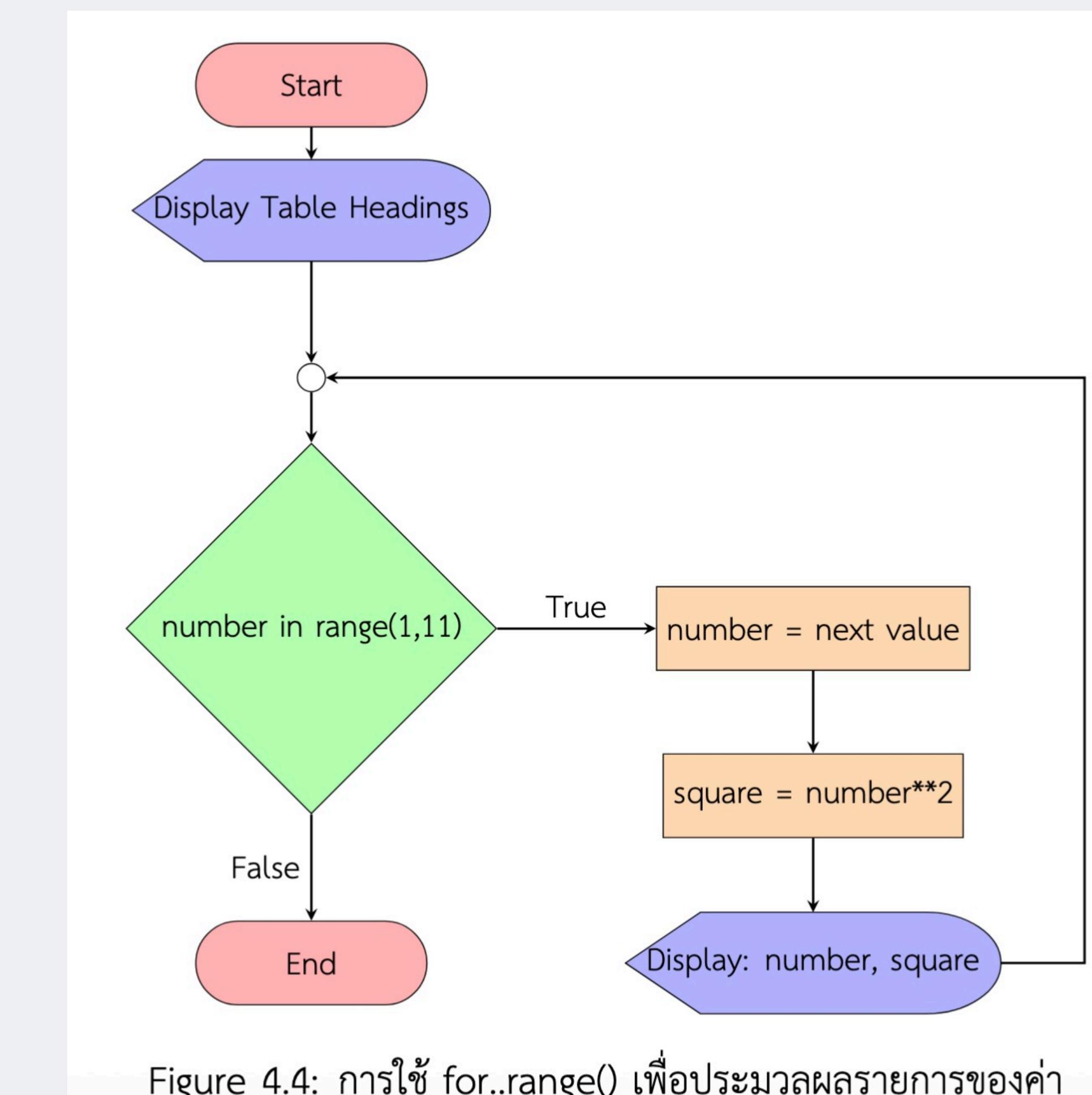


Figure 4.4: การใช้ `for..range()` เพื่อประมวลผลรายการของค่า

FOR RANGE()

```
# Print the table headings.  
print('Number\tSquare')  
print('-----')  
  
# Print the numbers 1 through 10  
# and their squares.  
for number in range(1, 11):  
    square = number**2  
    print(number, '\t', square)
```

Number	Square

1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

EXERCISE- I

$$\text{MPH} = \text{KPH} * 0.6214$$

KPH	MPH
60	37.3
70	43.5
80	49.7
<i>etc. . .</i>	
130	80.8

CONDITION-CONTROLLED LOOP



while

WHILE(CONDITION-CONTROLLED LOOP)

ลูป while จะทำการประมวลผลล็อกของโค้ดช้าไปเรื่อย ๆ ตราบใดที่เงื่อนไขที่กำหนดยังคงเป็นจริง ซึ่งหมายความว่าเงื่อนไขจะถูกตรวจสอบก่อนการทำงานแต่ละครั้ง และหากเป็นจริง โค้ดภายในลูปจะถูกดำเนินการ วนซ้ำไปเรื่อย ๆ จนกว่าเงื่อนไขจะเป็นเท็จ ลูปประเภทนี้เหมาะสมกับสถานการณ์ที่ไม่สามารถกำหนดจำนวนรอบในการทำงานได้ล่วงหน้า และต้องพึงพาปัจจัยแบบไนามิก เช่น การป้อนข้อมูลจากผู้ใช้ หรือผลลัพธ์จากการคำนวณภายในลูป

ตัวอย่างการใช้งาน เช่น การอ่านข้อมูลจากไฟล์จนกว่าจะถึงจุดสิ้นสุดของไฟล์ หรือการถามผู้ใช้ช้า ๆ จนกว่าจะได้รับคำตอบที่ถูกต้อง เนื่องจากการทำงานของลูปขึ้นอยู่กับเงื่อนไข ลูปจึงเป็นเครื่องมือที่ทรงพลังสำหรับการทำงานช้าที่ต้องการโครงสร้างควบคุมที่ยืดหยุ่นและตอบสนองได้อย่างไร้ตาม การรับประกันว่าเงื่อนไขจะถูกปฏิบัติในบางจุดเป็นสิ่งสำคัญ เพื่อหลีกเลี่ยงลูปที่ไม่มีวันสิ้นสุด ซึ่งจะทำให้โปรแกรมทำงานไม่หยุดหย่อน หากใช้อย่างถูกต้อง ลูปถือเป็นเครื่องมือสำคัญสำหรับจัดการกระบวนการที่ซับซ้อนในการเขียนโปรแกรม

WHILE(CONDITION-CONTROLLED LOOP)

แนวคิดสำคัญของลูป While:

- ดำเนินการซ้ำตราบใดที่เงื่อนไขที่กำหนดยังคงเป็นจริง
- หมายสำหรับสถานการณ์ที่ไม่สามารถทราบจำนวนรอบล่วงหน้า

```
1 while condition:  
2     # code to execute
```

Listing 4.9: While Loops Syntax

WHILE

```
while condition :  
    Statement  
    Statement  
    ....  
    Statement
```

**all of the statements in the block are indented

WHILE

ลูป while ที่มีเงื่อนไขในการทำงานจะทำซ้ำโค้ดตราบใดที่เงื่อนไขยังเป็นจริง ก่อนแต่ละรอบจะมีการตรวจสอบเงื่อนไข และลูปจะดำเนินต่อไปจนกว่าเงื่อนไขจะถอยเป็นเท็จ ลูปประเภทนี้หมายความกับกรณีที่ไม่ทราบจำนวนรอบแน่นอนล่วงหน้า และขึ้นอยู่กับปัจจัยแบบไดนามิก เช่น การป้อนข้อมูลจากผู้ใช้ หรือผลลัพธ์จากการคำนวณ การรับประกันว่าเงื่อนไขจะเป็นเท็จในบางจุดเป็นสิ่งสำคัญเพื่อหลีกเลี่ยงลูปไม่มีที่สิ้นสุด การจัดการเงื่อนไขบลูปอย่างเหมาะสมจะช่วยให้ลูปทำงานได้ตามวัตถุประสงค์ และทำให้โปรแกรมดำเนินการหรือสิ้นสุดได้อย่างถูกต้อง

```
1 count = 0
2 while count < 5:
3     print("Hello : ", count)
4     count += 1
```

Listing 4.10: Example of While Loops with End Condition

**all of the statements in the block are indented

2

```
k = 0  
while k < 100 :
```

...
k += 1

```
k = 5  
while k < 100 :
```

...
k += 1

```
k = 4  
while k < 100 :
```

...
k += 2

```
k = 100  
while k > 0 :
```

...
k += -1

```
for k in range(100) :
```

...

k = 0, 1, 2, ..., 99

```
for k in range(5,100,1) :
```

...

k = 5, 6, 7, ..., 99

```
for k in range(4,100,2) :
```

...

k = 4, 6, 8, ..., 98

```
for k in range(100, 0,-1) :
```

...

k = 100, 99, 98, ..., 1

INFINITE LOOPS

ควรระมัดระวังในการใช้ลูป while เพื่อให้แน่ใจว่าเงื่อนไขจะถอยเป็นเท็จในบางจุด มิฉะนั้นอาจเกิดลูปไม่มีที่สิ้นสุด ซึ่งจะทำให้โปรแกรมทำงานไม่หยุด ส่งผลให้ไม่สามารถตอบสนองได้และใช้ทรัพยากรามากเกินไป วิธีหลีกเลี่ยงลูปประเภทนี้คือการทำให้เงื่อนไขภายในลูปมีการเปลี่ยนแปลง เช่น การปรับค่าตัวแปร หรือการใช้คำสั่ง break นอกจากนี้ การเพิ่มตัวควบคุมจำนวนรอบสูงสุดก็เป็นวิธีหนึ่งในการป้องกันการทำงานแบบไม่รู้จบ การจัดการเงื่อนไขของลูปอย่างเหมาะสมและการเพิ่มกลไกป้องกันช่วยให้ลูปสามารถทำงานได้อย่างปลอดภัยและมีประสิทธิภาพ

```
1 # This loop will run forever
2 while True:
3     print("This is an infinite loop.")
```

Listing 4.11: Example of Infinite Loops

**all of the statements in the block are indented

SENTINELS



SENTINEL

การใช้เซนทิเนลร่วมกับลูป while เป็นรูปแบบการเขียนโปรแกรมที่ควบคุมการทำงานของลูปโดยใช้ค่าหรือเงื่อนไขพิเศษเป็นตัวสินสุดลูป เช่นทิโนลมีประโยชน์มากในการนับจำนวนรอบของลูปขึ้นอยู่กับอินพุตแบบไดนามิก เช่น การรับข้อมูลจากผู้ใช้ชี้ช้า ๆ จนกว่าจะป้อนคำเฉพาะ (เช่น 'exit') รูปแบบนี้ช่วยให้การวนซ้ำมีความยืดหยุ่นและสามารถควบคุมได้ดี ซึ่งหมายความว่าเมื่อกับสถานการณ์ที่ข้อมูลมีความไม่แน่นอน

REPETITION STRUCTURE

```
# Get a salesperson's sales and commission rate.  
sales = float(input('Enter the amount of sales: '))  
comm_rate = float(input('Enter the commission rate: '))  
commission = sales * comm_rate  
print('The commission is $', format(commission, ',.2f'))  
  
# Get another salesperson's sales and commission rate.  
sales = float(input('Enter the amount of sales: '))  
comm_rate = float(input('Enter the commission rate: '))  
commission = sales * comm_rate  
print('The commission is $', format(commission, ',.2f'))  
  
# Get another salesperson's sales and commission rate.  
sales = float(input('Enter the amount of sales: '))  
comm_rate = float(input('Enter the commission rate: '))  
commission = sales * comm_rate  
print('The commission is $', format(commission, ',.2f'))
```

WHILE

```
1 # This program calculates sales commissions.
2 # Create a variable to control the loop.
3 keep_going = 'y'
4
5 # Calculate a series of commissions.
6 while keep_going == 'y':
7     # Get a salesperson's sales and commission rate.
8     sales = float(input('Enter the amount of sales: '))
9     comm_rate = float(input('Enter the commission rate: '))
10
11    # Calculate the commission.
12    commission = sales * comm_rate
13
14    # Display the commission.
15    print(f'The commission is ${commission:.2f}')
16
17    # See if the user wants to do another one.
18    keep_going = input('Do you want to calculate another' + \
19                      ' commission (Enter y for yes): ')
```

Listing 4.12: Program to calculate sales commissions

SENTINEL

Samantha owns an import business, and she calculates the retail prices of her products with the following formula:

$$\text{retail price} = \text{wholesale cost} \times 2.5$$

Program Output (with input shown in bold)

Enter the item's wholesale cost: **10.00**

Retail price: \$25.00.

Do you have another item? (Enter y for yes): **y**

Enter the item's wholesale cost: **15.00**

Retail price: \$37.50.

Do you have another item? (Enter y for yes): **y**

Enter the item's wholesale cost: **12.50**

Retail price: \$31.25.

Do you have another item? (Enter y for yes): **n**

SENTINEL

Program Output (with input shown in bold)

How many rows? **5** Enter

How many columns? **10** Enter

* * * * * * * * *

* * * * * * * * *

* * * * * * * * *

* * * * * * * * *

* * * * * * * * *

WHILE

Algorithm 11: Exercise: Algorithm to Guess the Magic Number

Input: User's guess (1 to 100)
Output: Message indicating if the guess is correct or not

```
1 begin
2     import random
3     Print "What is my magic number (1 to 100)?";
4     Generate a random number between 1 and 100 and store it in mynumber;
5     Initialize ntries to 1;
6     Initialize yourguess to -1;
7     while ntries < 7 and yourguess ≠ mynumber do
8         Set msg to ntries + ">> ";
9         if ntries == 6 then
10            | Print "Your last chance";
11        end
12        Read user's guess and store it in yourguess;
13        if yourguess > mynumber then
14            | Print "--> too high";
15        else
16            | Print "--> too low";
17        end
18        Increment ntries by 1;
19    end
20    if yourguess == mynumber then
21        | Print "Yes! it's", mynumber;
22    else
23        | Print "Sorry! my number is", mynumber;
24    end
25 end
```

```
1 import random
2
3 print("What is my magic number (1 to 100) ?")
4 mynumber = random.randint(1, 100)
5 ntries = 1
6 yourguess = -1
7 while ntries < 7 and _____ :
8     msg = str(ntries) + ">> "
9     if (ntries == 6) :
10        _____
11        yourguess = int(input(msg))
12        if _____ :
13            print("--> too high")
14        _____
15        print("--> too low")
16        ntries += 1
17
18    if _____ :
19        | print("Yes! it's" mynumber)
20    else :
21        | print("Sorry! my number is", mynumber)
22
```

WHILE

```
> python3 test.py  
What is my magic number (1 to 100)?  
1>> 2  
--> too low  
2>> 50  
--> too low  
3>> 75  
--> too high  
4>> 65  
--> too high  
5>> 55  
--> too low  
Your last chance  
6>> 56  
--> too low  
Sorry! my number is 60
```

```
What is my magic number (1 to 100)?  
1>> 50  
--> too high  
2>> 30  
--> too high  
3>> 15  
--> too high  
4>> 8  
--> too high  
5>> 4  
--> too high  
Your last chance  
6>> 2  
Yes! it's 2
```

INPUT VALIDATION LOOPS



INPUT VALIDATION LOOPS

แนวคิดของลูปตรวจสอบความถูกต้องของข้อมูลที่รับเข้า (Input Validation Loop) คือการถามข้อมูลจากผู้ใช้ซ้ำๆ จนกว่าจะได้รับข้อมูลที่ถูกต้องตามเงื่อนไขที่กำหนด วิธีนี้ช่วยให้โปรแกรมสามารถจัดการกับข้อมูลที่รับเข้าจากผู้ใช้ได้อย่างมั่นคง ป้องกันข้อผิดพลาด และรักษาความถูกต้องของข้อมูล โดยลูปจะตรวจสอบเงื่อนไข เช่น ช่วงของค่า ประเภทข้อมูล หรือรูปแบบที่ต้องการ หากข้อมูลที่ผู้ใช้ป้อนไม่ตรงตามเงื่อนไข ระบบจะแสดงข้อความแสดงข้อผิดพลาด และถามข้อมูลใหม่จนกว่าจะได้รับข้อมูลที่ถูกต้อง ลูปประเภทนี้มีความสำคัญต่อการสร้างโปรแกรมที่ใช้งานง่ายและทนทานต่อข้อมูลผิดพลาดจากผู้ใช้

```
1 # Get a test score.
2 score = int(input('Enter a test score: '))
3
4 # Make sure it is not less than 0 or greater than 100.
5 while score < 0 or score > 100:
6     print('ERROR: The score cannot be negative')
7     print('or greater than 100.')
8     score = int(input('Enter the correct score: '))
```

Listing 4.13: Program to get a test score and validate it

INPUT VALIDATION LOOPS

```
# Get a test score.  
score = int(input('Enter a test score: '))  
# Make sure it is not less than 0 or greater than 100.  
while score < 0 or score > 100:  
    print('ERROR: The score cannot be negative')  
    print('or greater than 100.')  
    score = int(input('Enter the correct score: '))
```

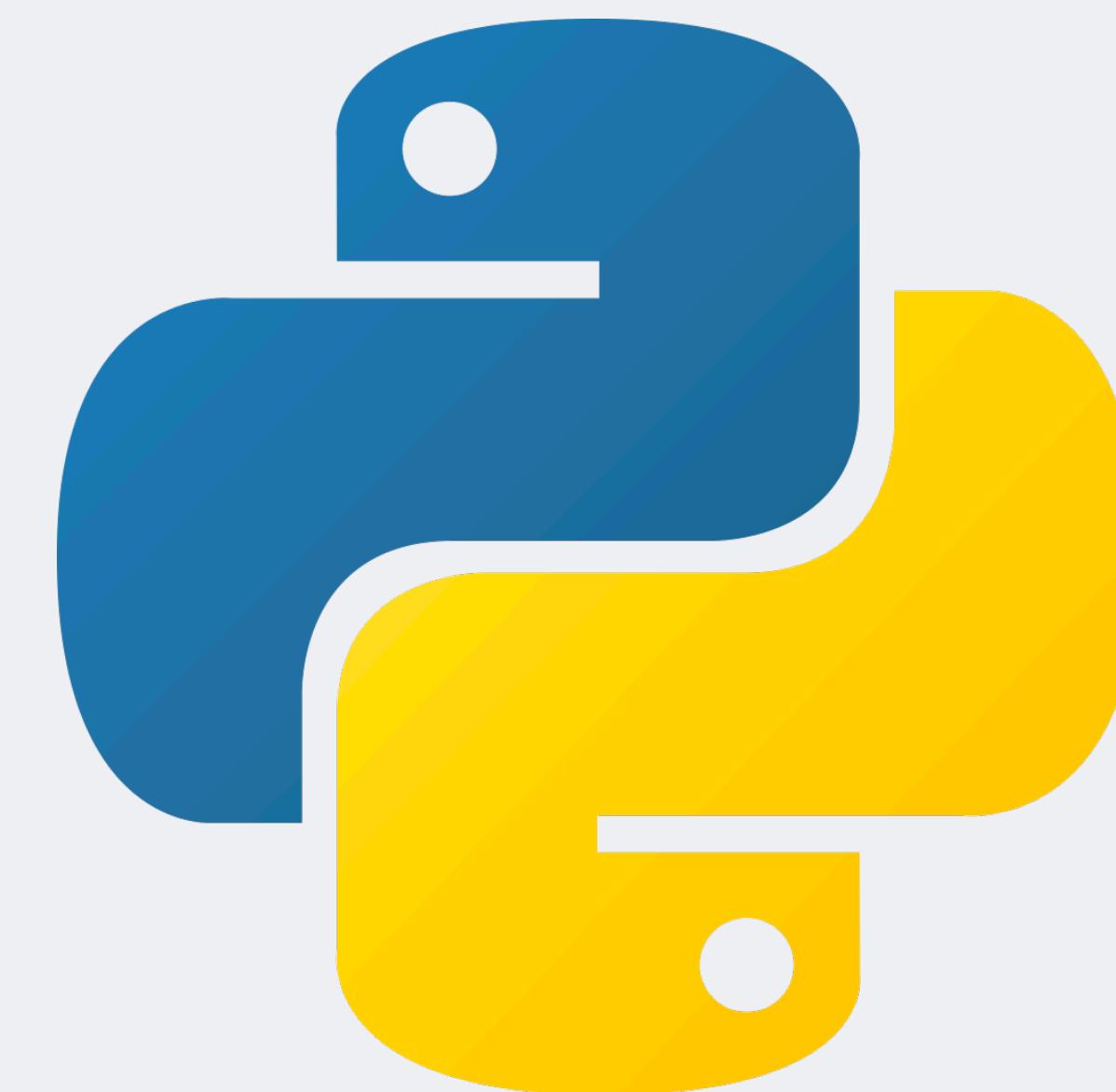
```
Enter a test score: 230  
ERROR: The score cannot be negative  
or greater than 100.  
Enter the correct score: -5  
ERROR: The score cannot be negative  
or greater than 100.  
Enter the correct score: 80
```

INPUT VALIDATION LOOPS

```
if x != 0 and y/x > 5 :
```

```
if y/x > 5 and x != 0 :
```

LOOP-CONTROLLED STATEMENT



break, continue, pass

LOOP CONTROL STATEMENTS

คำสั่งควบคุมลูป เช่น break, continue, และ pass ใช้เพื่อควบคุมการทำงานของลูปในโปรแกรม คำสั่ง break ใช้สำหรับออกจากลูปทันทีเมื่อเงื่อนไขที่กำหนดเป็นจริง ซึ่งช่วยให้ลูปหยุดก่อนจะทำงานครบทุกรอบตามปกติ เพื่อเพิ่มประสิทธิภาพหรือหยุดเมื่อได้ผลลัพธ์ที่ต้องการ คำสั่ง continue ใช้เพื่อข้ามการทำงานในรอบนั้นของลูป และไปทำการรอบถัดไปทันที โดยไม่หยุดลูปทั้งหมด ส่วนคำสั่ง pass เป็นคำสั่งว่าง ใช้เป็นตัวแทนในการนัยยังไม่ได้เขียนโค้ด หรือใช้เพื่อให้โครงสร้างโปรแกรมสมบูรณ์ตามไวยากรณ์ คำสั่งเหล่านี้ช่วยเพิ่มความยืดหยุ่นและประสิทธิภาพให้กับลูป ช่วยให้สามารถควบคุมการทำงานของโค้ดภายในลูปได้อย่างแม่นยำ

แนวคิดสำคัญของคำสั่งควบคุมลูป:

- Break: ออกจากลูปทันทีเมื่อเงื่อนไขที่กำหนดเป็นจริง
- Continue: ข้ามรอบปัจจุบันของลูปและไปยังรอบถัดไปทันที
- Pass: ไม่มีการดำเนินการใด ๆ ใช้เป็นตัวแทนโค้ดที่ยังไม่ถูกเขียน

CONTINUE

```
# Prints all letters except 'a' and 'k'  
for letter in 'Anirach Mingkhwan':  
    if letter == 'a' or letter == 'k':  
        continue  
    print ('Current Letter :', letter)
```

```
Current Letter : A  
Current Letter : n  
Current Letter : i  
Current Letter : r  
Current Letter : c  
Current Letter : h  
Current Letter :  
Current Letter : M  
Current Letter : i  
Current Letter : n  
Current Letter : g  
Current Letter : h  
Current Letter : w  
Current Letter : n
```

BREAK

```
# Prints all letters except 'a' and 'k'  
for letter in 'Anirach Mingkhwan':  
    if letter == 'a' or letter == 'k':  
        break  
    print ('Current Letter :', letter)
```

```
Current Letter : A  
Current Letter : n  
Current Letter : i  
Current Letter : r
```

PASS

```
# Prints all letters except 'a' and 'k'  
for letter in 'Anirach Mingkhwan':  
    if letter == 'a' or letter == 'k':  
        pass  
    print ('Current Letter :', letter)
```

```
Current Letter : A  
Current Letter : n  
Current Letter : i  
Current Letter : r  
Current Letter : a  
Current Letter : c  
Current Letter : h  
Current Letter :  
Current Letter : M  
Current Letter : i  
Current Letter : n  
Current Letter : g  
Current Letter : k  
Current Letter : h  
Current Letter : w  
Current Letter : a  
Current Letter : n
```

PASS

We generally use it as a placeholder.

Suppose we have a loop or a function that is not implemented yet, but **we want to implement it in the future. They cannot have an empty body.** The interpreter would complain. So, we use the **pass** statement to construct a body that does nothing.

CALCULATING RUNNING TOTAL



CALCULATING RUNNING TOTAL

การคำนวณผลรวมสะสมหมายถึงการบวกค่าตัวเลขในลำดับอย่างต่อเนื่องในขณะที่ประมวลผลข้อมูล เทคนิคนี้มีประโยชน์อย่างมากในหลายบริบท เช่น การคำนวณทางการเงิน การวิเคราะห์ข้อมูล และระบบติดตามแบบเรียลไทม์ ตัวอย่างเช่น ผลรวมสะสมในบริบททางเศรษฐกิจช่วยในการติดตามยอดรวมของธุรกรรมตลอดช่วงเวลา ในการวิเคราะห์ข้อมูล เทคนิคนี้ช่วยให้เข้าใจแนวโน้มและการเปลี่ยนแปลงในชุดข้อมูลได้ดีขึ้น

การคำนวณผลรวมสะสมจะทำโดยเริ่มจากการกำหนดตัวแปรสำหรับเก็บค่าผลรวม จากนั้นบวกค่าตัวเลขแต่ละค่าจากลำดับเข้าไปยังตัวแปรนี้อย่างต่อเนื่อง วิธีนี้ช่วยให้สามารถติดตามค่ารวมที่เปลี่ยนแปลงได้อย่างมีประสิทธิภาพ เมื่อข้อมูลใหม่ถูกประมวลผล

CALCULATING RUNNING TOTAL

```
1 # Program to find the sum of all numbers stored in a list
2 # List of numbers
3 numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
4
5 # Variable to store the sum
6 sum = 0
7
8 # Iterate over the list
9 for val in numbers:
10     sum += val
11     print(sum)
12
13 print("The sum is", sum)
```

Listing 4.17: Calculates the sum of a series of numbers

CALCULATING RUNNING TOTAL

Statement	What It Does	Value of x after the Statement
<code>x = x + 4</code>	Add 4 to x	10
<code>x = x - 3</code>	Subtracts 3 from x	3
<code>x = x * 10</code>	Multiplies x by 10	60
<code>x = x / 2</code>	Divides x by 2	3
<code>x = x % 4</code>	Assigns the remainder of x / 4 to x	2

CALCULATING RUNNING TOTAL

Operator	Example Usage	Equivalent To
<code>+=</code>	<code>x += 5</code>	<code>x = x + 5</code>
<code>-=</code>	<code>y -= 2</code>	<code>y = y - 2</code>
<code>*=</code>	<code>z *= 10</code>	<code>z = z * 10</code>
<code>/=</code>	<code>a /= b</code>	<code>a = a / b</code>
<code>%=</code>	<code>c %= 3</code>	<code>c = c % 3</code>

CALCULATING RUNNING TOTAL

```
1 # This program calculates the sum of a series of numbers the user enters.  
2  
3 max = 5 # The maximum number  
4 # Initialize an accumulator variable.  
5 total = 0.0  
6  
7 # Explain what we are doing.  
8 print('This program calculates the sum of')  
9 print(max, 'numbers you will enter.')  
10  
11 # Get the numbers and accumulate them.  
12 for counter in range(max):  
13     number = int(input('Enter a number: '))  
14     total = total + number  
15  
16 # Display the total of the numbers.  
17 print('The total is', total)
```

Listing 4.18: Calculates the sum of a series of numbers entered by the user

NESTED LOOPS



NESTED LOOP

ลูป for ซ้อนกัน (Nested-for loops) หมายถึงการเขียนลูปหนึ่งไว้ภายในอีกลูปหนึ่ง ซึ่งช่วยให้สามารถวนซ้ำผ่านโครงสร้างข้อมูลหลายมิติ หรือดำเนินการซ้ำภายในแต่ละรอบของลูปรอบนอกได้ ลูปด้านในจะทำงานครบทุกครั้งก่อนที่ลูปรอบนอกจะขยับไปยังรอบถัดไป เทคนิคนี้มีประโยชน์อย่างมากในการทำงานกับตาราง การสร้างชุดค่าผสม หรือการประมวลผลข้อมูลที่ซับซ้อน เช่น การเข้าถึงแต่ละองค์ประกอบในลิสต์สองมิติ (เมทริกซ์) โดยใช้ลูปรอบนอกสำหรับแถว และลูปรอบในสำหรับคอลัมน์ การจัดการลูปซ้อนอย่างเหมาะสมเป็นสิ่งสำคัญเพื่อหลีกเลี่ยงความซับซ้อนของการคำนวณที่มากเกินไป และเพื่อให้โค้ดทำงานได้อย่างมีประสิทธิภาพ

```
1 # Nested loop to print numbers both using range()
2 for i in range(1, 3):
3     for j in range(2,5):
4         print(i,j)
5
6 #This will print the numbers 1 2, 1 3, 1 4, 2 2, 2 3 and 2 4.
```

Listing 4.19: Example of Using range() Nested Loop

NESTED LOOP

```
for i in range(1,3):  
    for j in range(2,5):  
        print(i,j)
```

1 2
1 3
1 4
2 2
2 3
2 4

```
for i in range(4):  
    for j in range(i):  
        print(i,j)
```

1 0
2 0
2 1
3 0
3 1
3 2

**all of the statements in the block are indented

NESTED LOOP

A loop that is inside another loop is called a **nested loop**.

```
for hours in range(24):
    for minutes in range(60):
        for seconds in range(60):
            print(hours, ':', minutes, ':', seconds)
```

This code's output would be:

0:0:0
0:0:1
0:0:2

(The program will count through each second of 24 hours.)

23:59:59

EXERCISE-II

Program Output (with input shown in bold)

How many rows? **5** Enter

How many columns? **10** Enter

* * * * * * * * *

* * * * * * * * *

* * * * * * * * *

* * * * * * * * *

* * * * * * * * *

EXERCISE-III

1-100

THANKS

2PM - 3PM



WORD OF THE WISE

“Whether you
think you can
or think you can’t
you’re right”

Henry Ford

