

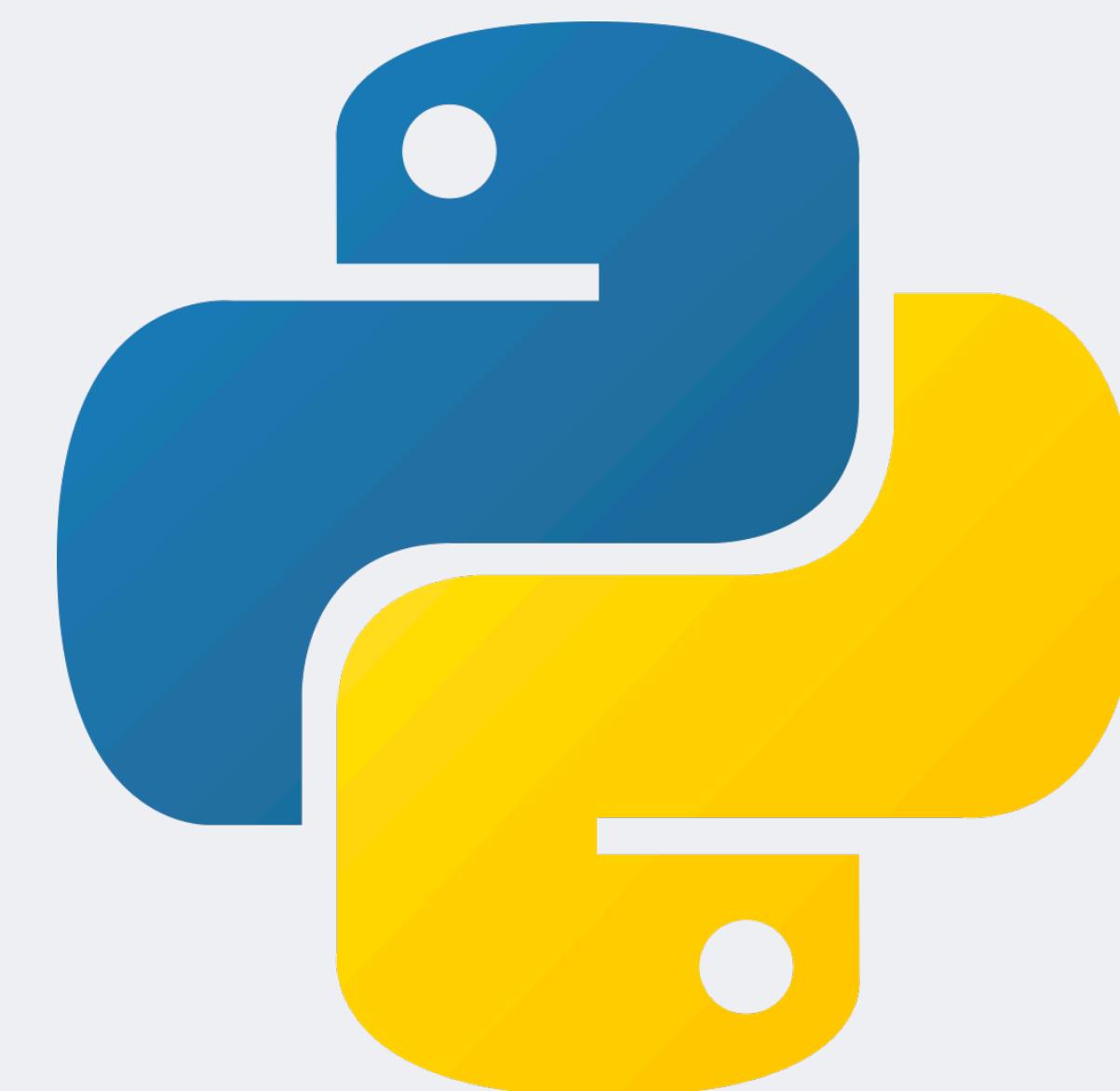
INE-PYTHON

LECTURE - 03 CONDITIONALS AND CONTROL FLOW

What You Will Learn

- Conditional Statement
- Comparison Operators
- Logical Operators

CONDITIONALS AND CONTROL FLOW



Control Flow

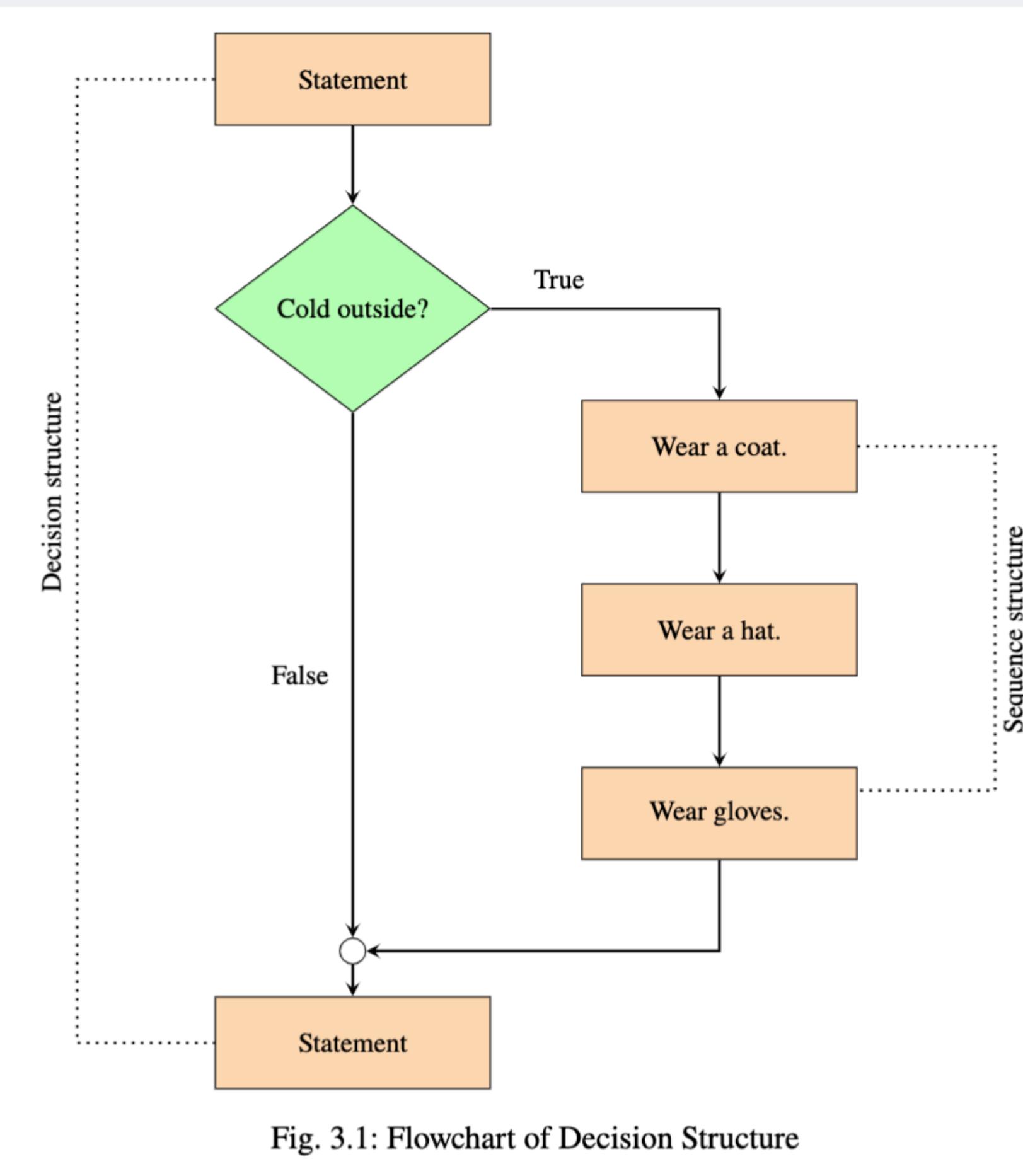
CONDITIONAL STATEMENTS

โครงสร้างเงื่อนไขและการควบคุมการให้ผลของโปรแกรม หมายถึง กลไกที่ช่วยให้โปรแกรมสามารถตัดสินใจและดำเนินการส่วนต่าง ๆ ของโค้ดตามเงื่อนไขที่กำหนดไว้

ด้วยการใช้คำสั่งเงื่อนไข โปรแกรมสามารถเลือกปฏิบัติการที่แตกต่างกันได้ ขึ้นอยู่กับว่าเงื่อนไขนั้นประเมินค่าออกมาเป็นจริงหรือเท็จ ความสามารถนี้เป็นสิ่งสำคัญสำหรับการสร้างโปรแกรมที่มีความยืดหยุ่นและตอบสนองต่อข้อมูลนำเข้าหรือสถานการณ์ที่หลากหลาย

CONTROL FLOW

A sequence structure nested inside a decision structure



KEY CONCEPTS OF USING CONDITIONAL STATEMENTS

แนวคิดสำคัญเกี่ยวกับคำสั่งเงื่อนไข:

- If Statement: ทำงานเมื่อเงื่อนไขที่กำหนดเป็นจริง
- Elif Statement: ตรวจสอบเงื่อนไขเพิ่มเติม หากเงื่อนไขก่อนหน้าไม่เป็นจริง
- Else Statement: ทำงานเมื่อไม่มีเงื่อนไขใด ๆ ที่เป็นจริง
- ไวยากรณ์ (Syntax): การจัดวางและย่อหน้าที่ถูกต้องช่วยให้โค้ดอ่านง่ายและทำงานได้ถูกต้อง
- เงื่อนไขเดียว: if condition:
- หลายเงื่อนไข: if condition1: elif condition2: else:
- การประเมินค่าทางบูลีน: เงื่อนไขจะถูกประเมินเป็น True หรือ False
- คำสั่งซ้อนกัน: สามารถเขียนคำสั่งเงื่อนไขซ้อนกันได้เพื่อสร้างตรรกะที่ซับซ้อนมากขึ้น
- ควบคุมลำดับการทำงาน: กำหนดทิศทางการทำงานของโปรแกรมตามผลของการเงื่อนไข
- การทำงานค่าเริ่มต้น: คำสั่ง else ใช้สำหรับกรณีที่ไม่มีเงื่อนไขใดตรงเลย
- ความชัดเจน: ทำให้โค้ดอ่านง่าย เข้าใจง่าย และดูแลรักษาง่าย

CONDITIONALS AND CONTROL FLOW



if

IF STATEMENTS

คำสั่ง if ช่วยให้สามารถดำเนินการได้ดีเมื่อเงื่อนไขเป็นจริง ซึ่งเป็นโครงสร้างควบคุมพื้นฐานในโปรแกรม มิฉะนั้น เมื่อเงื่อนไขที่ระบุในคำสั่ง if ถูกประเมินว่าเป็นจริง โปรแกรมจะดำเนินการบล็อกโค้ดที่อยู่ภายใต้ if ที่เยื่องไว้ วิธีนี้ทำให้โปรแกรมสามารถตัดสินใจและดำเนินการตามข้อมูลที่เปลี่ยนแปลงได้ เช่น ตรวจสอบว่าอายุของผู้ใช้เกิน เกณฑ์หรือไม่เพื่อให้เข้าถึงไฟอร์บองอย่าง หรือว่าตัวแปรตรงตามเงื่อนไขก่อนจะคำนวณต่อไป การดำเนินการแบบมี เงื่อนไขเป็นสิ่งสำคัญสำหรับการสร้างโปรแกรมที่ตอบสนองต่อสถานการณ์และอินพุตที่หลากหลาย หากไม่มีคำสั่ง if โปรแกรมจะเป็นเส้นตรงและไม่สามารถจัดการกับเงื่อนไขต่าง ๆ ได้ ทำให้ขาดความยืดหยุ่นและประสิทธิภาพ

```
1 if condition:  
2     # code to execute if the condition is true
```

IF STATEMENTS

IF.. Syntax:

```
1 if condition:  
2     # code to execute if the condition is true
```

Listing 3.1: Syntax of If condition

IF.. Example:

```
1 age = int(input("Please input age: "))  
2 if age >= 18:  
3     print("You are an adult.")
```

Listing 3.2: Example of If condition

Algorithm 5: Algorithm to Check if a Person is an Adult

Input: age

Output: Message indicating if the person is an adult

```
1 begin  
2     age = 18;  
3     if age >= 18 then  
4         Print "You are an adult.";  
5     end  
6 end
```

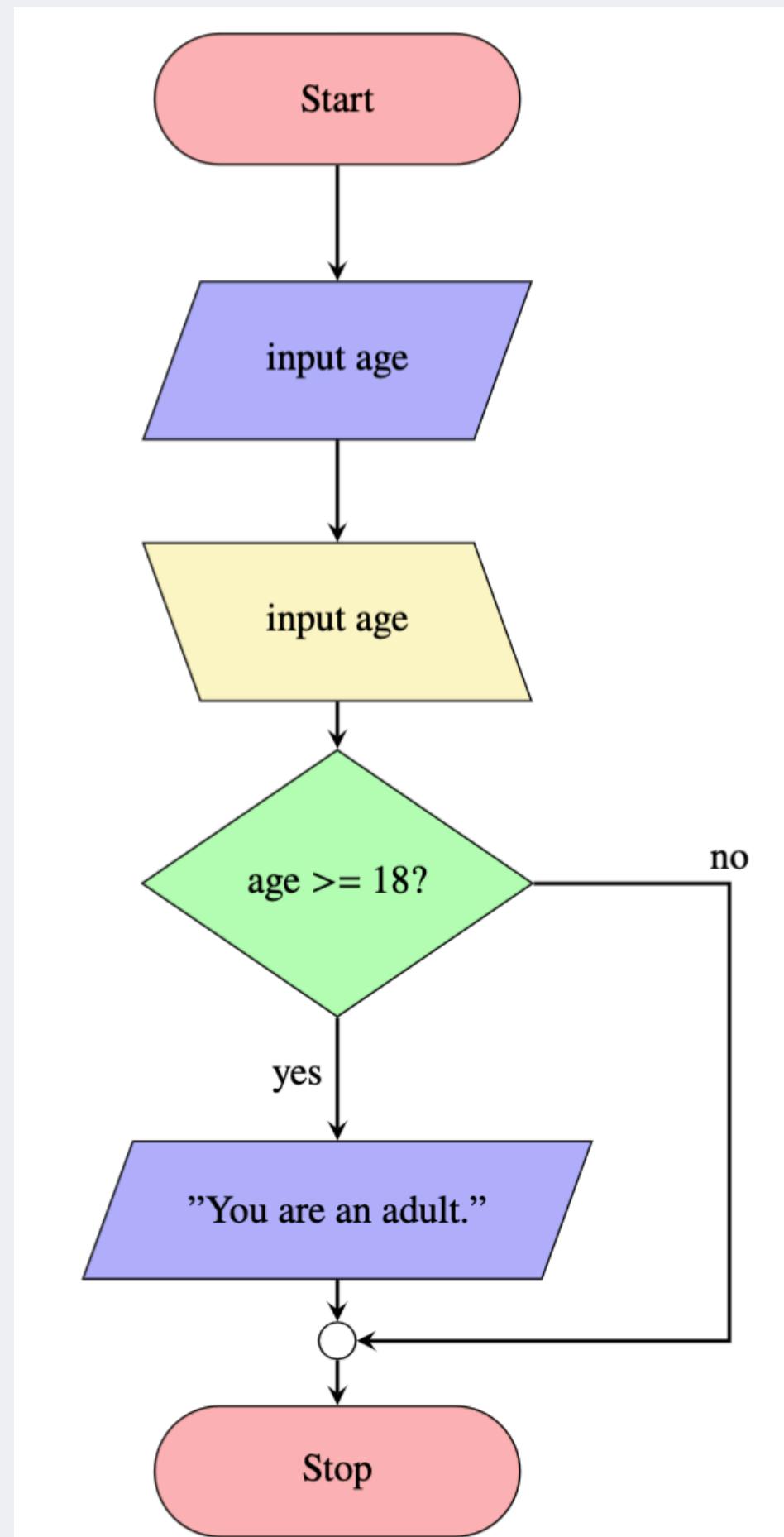


Fig. 3.1: Flowchart for the Age Check Program

EXERCISE-1

Algorithm 6: แบบฝึกหัด: อัลกอริทึมสำหรับคำนวณค่าเฉลี่ยของคะแนนสอบ

Input: คะแนนสอบ 3 วิชา

Output: ค่าเฉลี่ย และ "Congratulations!" หากเกิน 95

```
1 begin
2     รับคะแนนสอบที่หนึ่ง;
3     รับคะแนนสอบที่สอง;
4     รับคะแนนสอบที่สาม;
5     คำนวณค่าเฉลี่ย;
6     แสดงผลค่าเฉลี่ย;
7     if ค่าเฉลี่ย > 95 then
8         แสดง "Congratulations!";
```

EXERCISE-1

Program Output (with input shown in bold)

Enter the score for test 1: **82**

Enter the score for test 2: **76**

Enter the score for test 3: **91**

The average score is 83.0

Program Output (with input shown in bold)

Enter the score for test 1: **93**

Enter the score for test 2: **99**

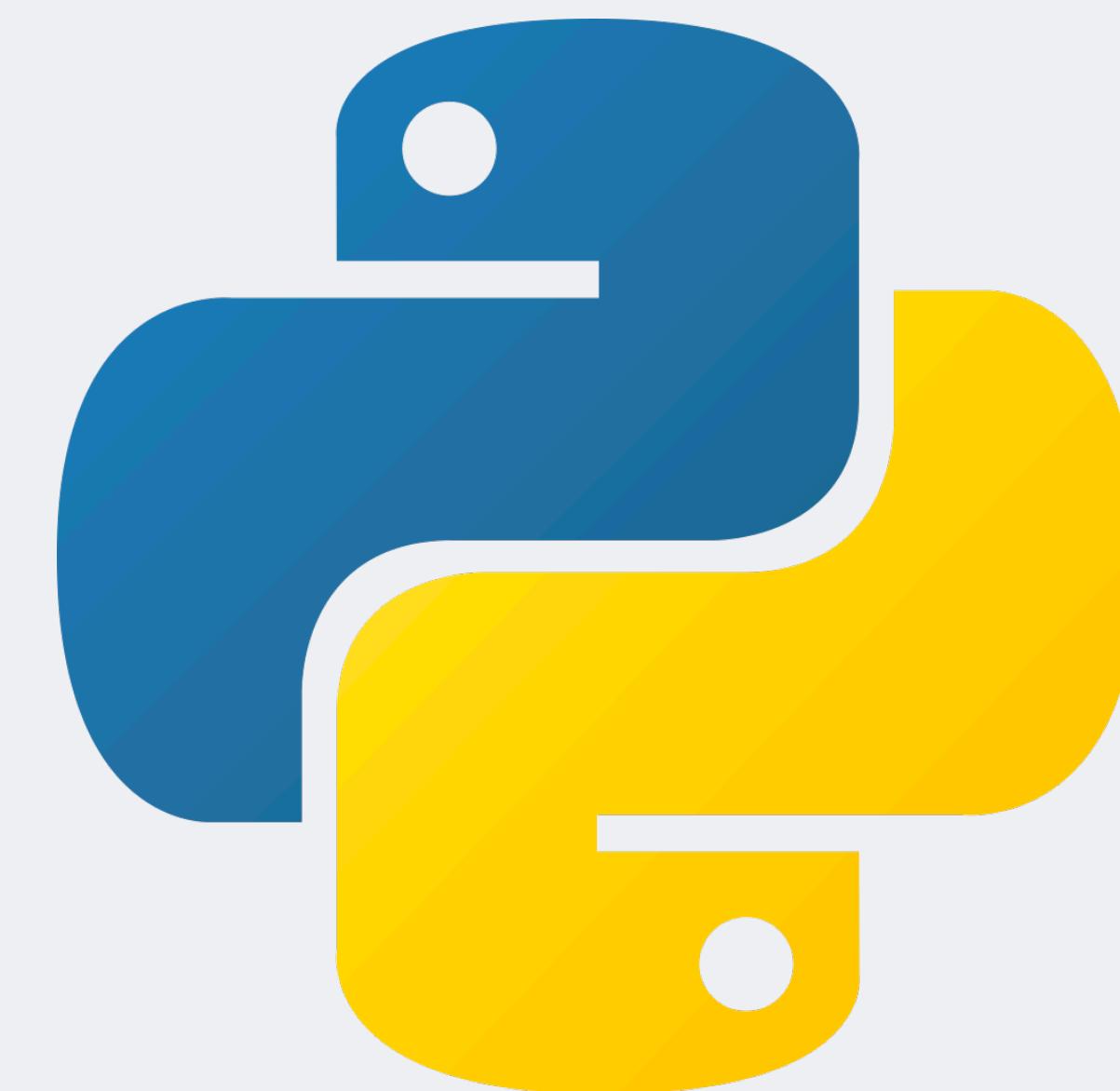
Enter the score for test 3: **96**

The average score is 96.0

Congratulations!

That is a great average!

CONDITIONALS AND CONTROL FLOW



if-elif-else

ELIF STATEMENT

คำสั่ง elif (ย่อมาจาก "else if") ช่วยให้สามารถตรวจสอบหลายเงื่อนไขได้ โดยสามารถใช้หลังจาก if เพื่อประเมินเงื่อนไขเพิ่มเติมหากเงื่อนไขก่อนหน้าไม่เป็นจริง ตัวอย่างเช่นในระบบให้เกรด นักเรียนอาจได้ "A" ถ้าคะแนนมากกว่า 90 หรือ "B" ถ้าอยู่ในช่วง 80 ถึง 89 เป็นต้น คำสั่ง elif ช่วยให้เขียนโปรแกรมที่มีหลายเงื่อนไขอย่างชัดเจน และหากไม่มีเงื่อนไขใดเป็นจริงเลย ก็สามารถใช้ else เพื่อกำหนดผลลัพธ์เริ่มต้นได้ วิธีนี้ช่วยให้โค้ดอ่านง่าย มีโครงสร้างและควบคุมทิศทางของโปรแกรมอย่างชัดเจน

ไวยากรณ์ของ Elif:

```
1 if condition1:  
2     # code to execute if condition1 is true  
3 elif condition2:  
4     # code to execute if condition2 is true
```

ELIF STATEMENT

Syntax:

```
1 if condition1:  
2     # code to execute if condition1 is true  
3 elif condition2:  
4     # code to execute if condition2 is true
```

Listing 3.3: Syntax of Elif condition

Example:

```
1 # Input the number of employees from the user  
2 num_employees = int(input("Enter the number of employees: "))  
3 # Check the number of employees and print the appropriate company  
    size  
4 if num_employees < 50:  
5     print("This is a small company.")  
6 elif num_employees < 250:  
7     print("This is a medium-sized company.")  
8 elif num_employees >= 250:  
9     print("This is a large company.")  
10
```

Listing 3.4: Program to determine the size of a company based on the number of employees

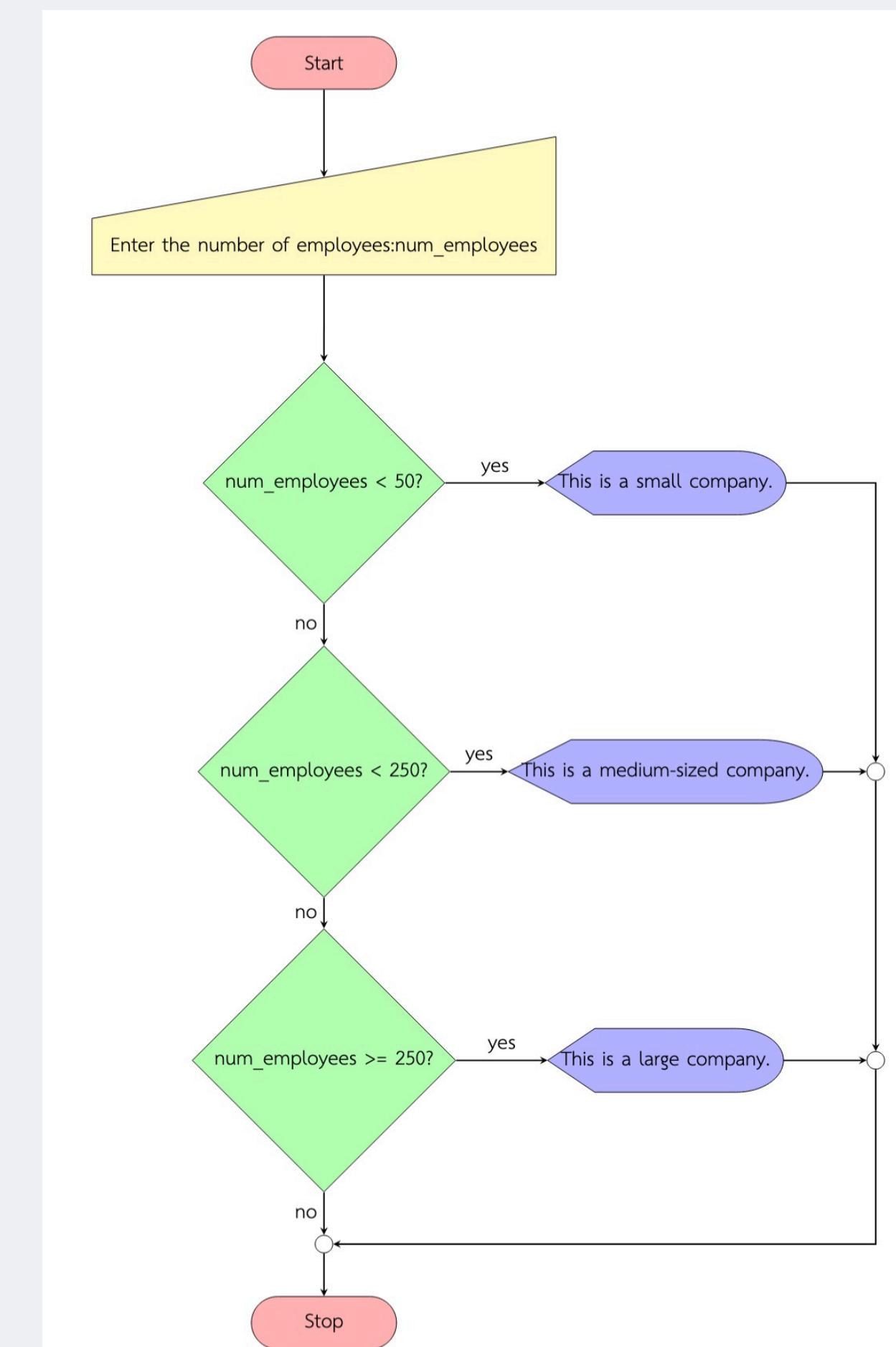


Figure 3.3: ผังงานสำหรับการกำหนดขนาดของบริษัท

ELIF STATEMENT

Example:

```
1 score = 75
2 if score >= 90:
3     print("Grade: A")
4 elif score >= 80:
5     print("Grade: B")
6 elif score >= 70:
7     print("Grade: C")
8 else:
9     print("Grade: D or F")
```

Listing 3.4: Example of Elif condition

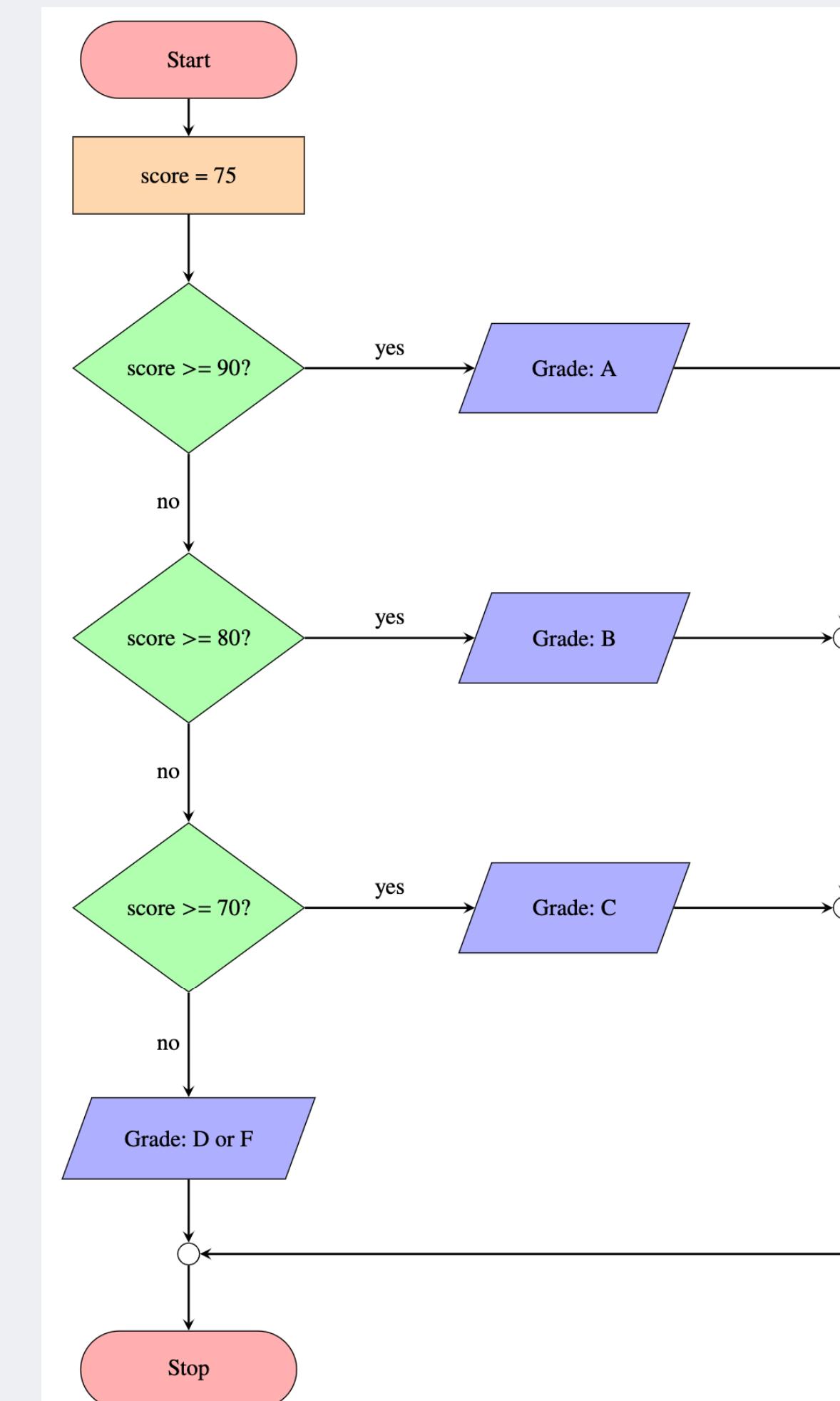
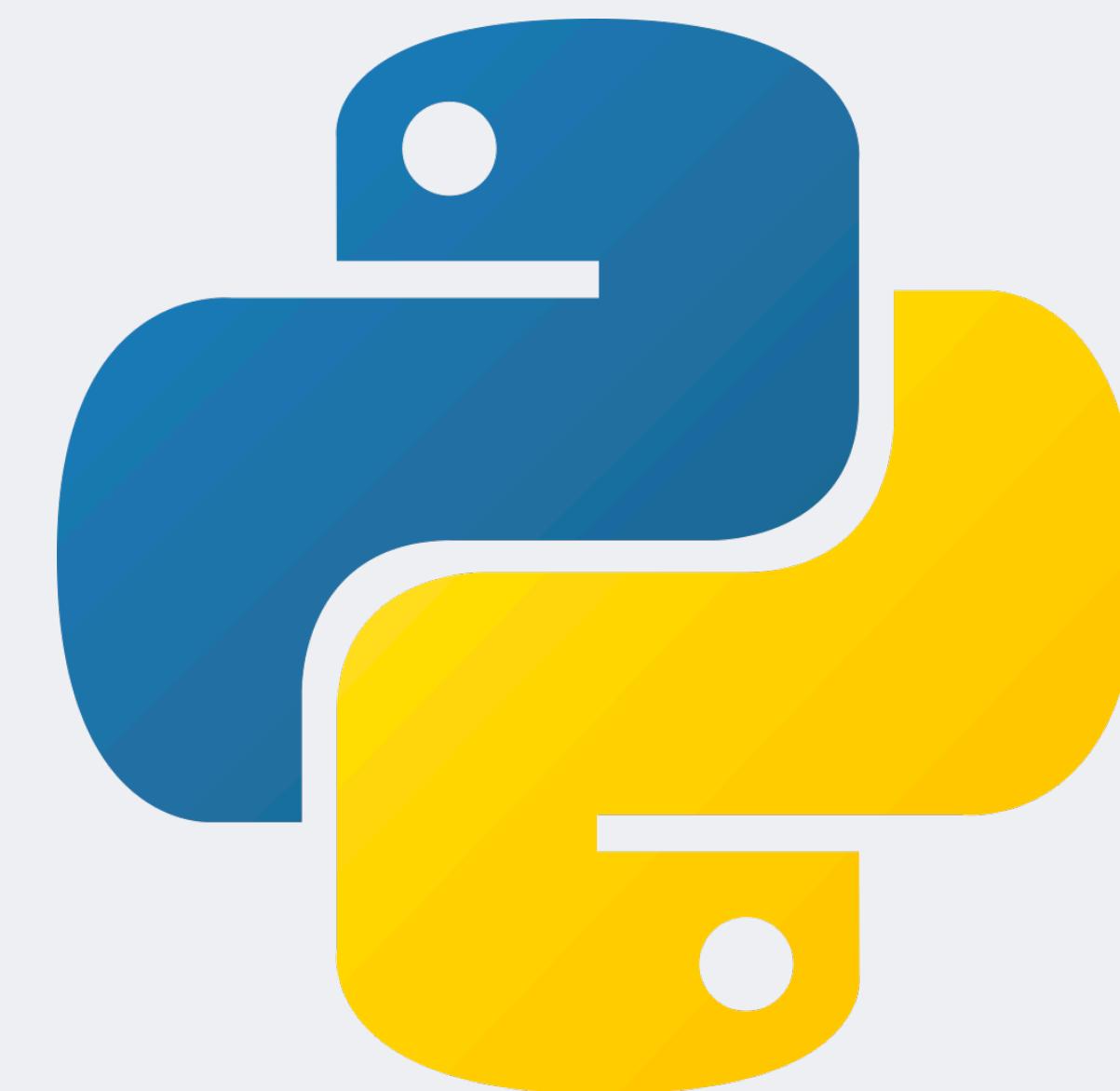


Fig. 3.2: Flowchart for the Grade Evaluation

CONDITIONALS AND CONTROL FLOW



if-elif-else

IF-ELIF-ELSE STATEMENT

คำสั่ง `else` ช่วยให้คุณสามารถดำเนินการกลุ่มคำสั่งได้เมื่อไม่มีเงื่อนไขก่อนหน้านี้เป็นจริง โดยทำหน้าที่เป็นตัวเลือกสุดท้ายสำหรับสถานการณ์ที่ไม่ตรงกับเกณฑ์ที่กำหนดไว้ ตำแหน่งของ `else` อยู่ท้ายสุดของโครงสร้าง `if-elif` ซึ่งทำให้มั่นใจได้ว่ามีการกำหนดเส้นทางการทำงานของโปรแกรมอย่างชัดเจนในกรณีที่เงื่อนไขทั้งหมดเป็นเท็จ สิ่งนี้รับประกันว่าโปรแกรมสามารถจัดการกับสถานการณ์ที่ไม่คาดคิดหรือกรณีเริ่มต้นได้อย่างราบรื่น โดยไม่ล่วง เงื่อนไขใด ๆ เช่น ในการจัดประเภทคะแนนเป็นเกรดตัวอักษร `else` สามารถใช้ในการกำหนดเกรดหากไม่เข้าเกณฑ์ระดับที่สูงกว่าเลย บล็อกคำสั่งสุดท้ายนี้ช่วยให้สามารถจัดการกับกรณีพิเศษได้ และทำให้การไหลของโปรแกรม มีความเสถียรและน่าเชื่อถือมากยิ่งขึ้น คำสั่ง `else` ยังช่วยเสริมความยืดหยุ่นและความสมบูรณ์ของโครงสร้างเงื่อนไข โดยให้ทางเลือกสำรองที่ทำให้โปรแกรมสามารถตอบสนองต่อช่วงของข้อมูลนำเข้าได้หลากหลาย และรักษาพฤติกรรม ที่สอดคล้องกันได้

```
1 if condition1:  
2     # code to execute if condition1 is true  
3 elif condition2:  
4     # code to execute if condition2 is true  
5 else:  
6     # code to execute if none of the above conditions are true
```

IF-ELIF-ELSE STATEMENT

Syntax:

```
1 if condition1:  
2     # code to execute if condition1 is true  
3 elif condition2:  
4     # code to execute if condition2 is true  
5 else:  
6     # code to execute if none of the above conditions are true
```

Listing 3.5: Syntax of Else condition

Example:

```
1 temperature = 30  
2 if temperature > 30:  
3     print("It's hot outside.")  
4 elif temperature > 20:  
5     print("The weather is nice.")  
6 else:  
7     print("It's cold outside.")
```

Listing 3.6: Example of Else condition

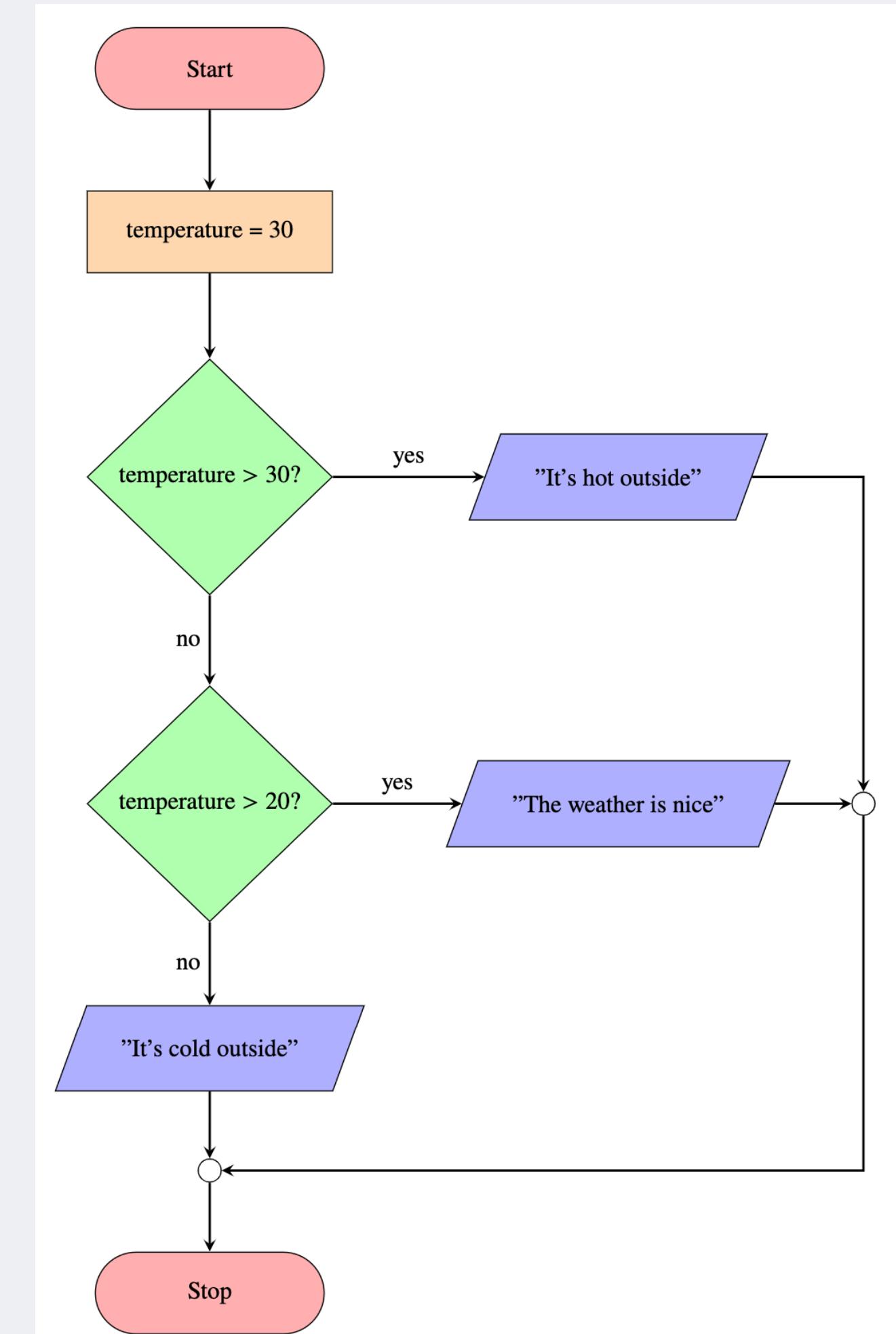


Fig. 3.3: Flowchart for the Temperature Check Program

IF-ELIF-ELSE

A nested decision structure

```
inchar = input("Input one character:")
if inchar >= 'A' and inchar <= 'Z':
    print("You in put Upper Case Letter ", inchar)
elif inchar >= 'a' and inchar <= 'z':
    print("You in put Lower Case Letter ", inchar)
elif inchar >= '0' and inchar <= '9':
    print("You in put Number ", inchar)
else :
    print("It's not a letter or number.", inchar)
```

Input one character:@

It's not a letter or number. @

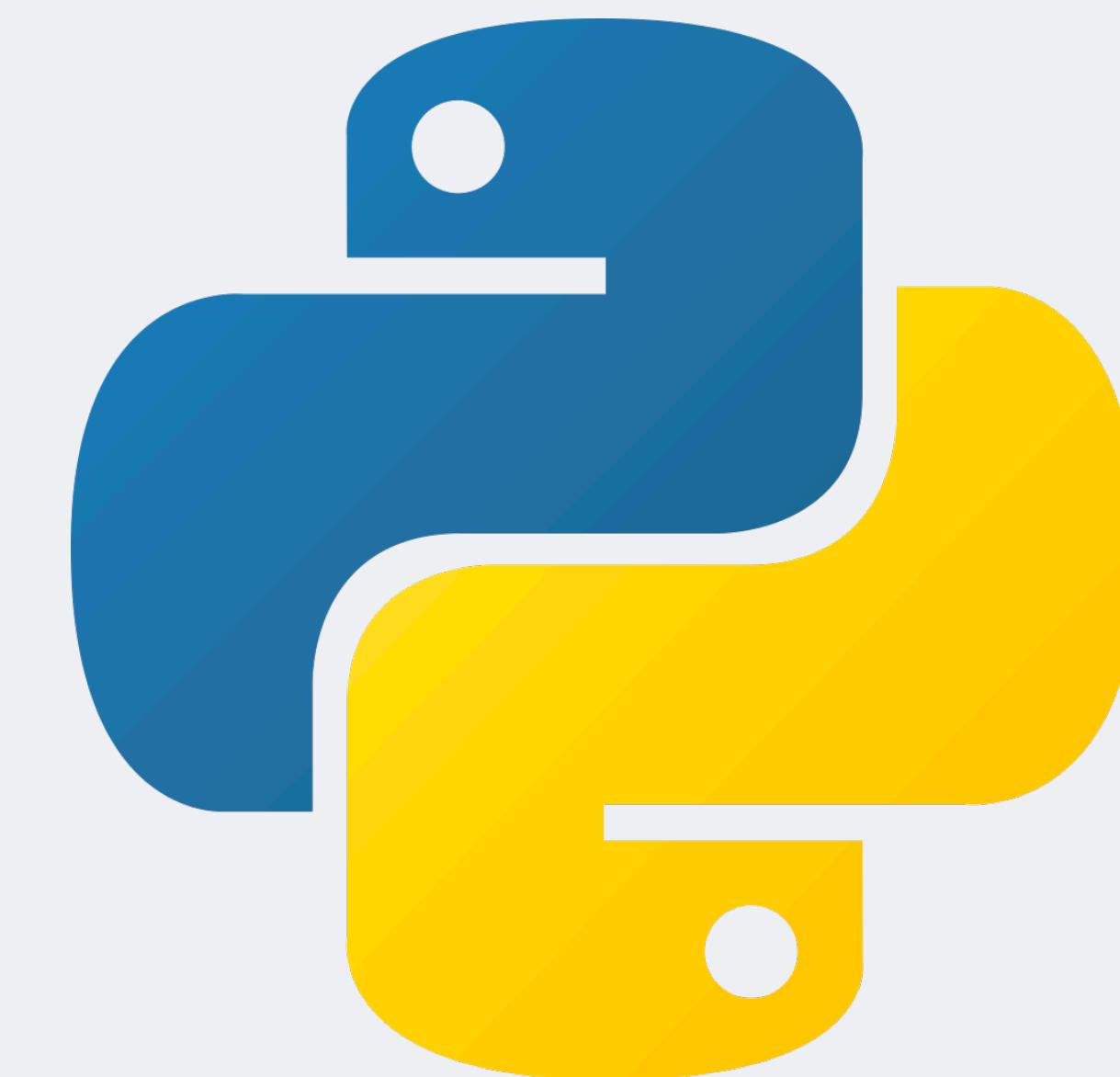
Algorithm 7: Algorithm to Categorize a Character

Input: A character input: `inchar`

Output: Message indicating the type of character

```
1 begin
2     Print "Input one character:";
3     inchar = input();
4     if inchar ≥ 'A' and inchar ≤ 'Z' then
5         Print "You input Upper Case Letter", inchar;
6     else
7         if inchar ≥ 'a' and inchar ≤ 'z' then
8             Print "You input Lower Case Letter", inchar;
9         else
10            if inchar ≥ '0' and inchar ≤ '9' then
11                Print "You input Number", inchar;
12            else
13                Print "It's not a letter or number.", inchar;
14            end
15        end
16    end
17 end
```

CONDITIONALS AND CONTROL FLOW



Nested If

IF-ELIF-ELSE

เงื่อนไขซ้อนในโปรแกรมหมายถึงการวางคำสั่ง if หรือ elif ไว้ภายในบล็อก if, elif หรือ else อื่น โครงสร้างลักษณะนี้ช่วยให้สามารถประมวลผลเงื่อนไขหลายระดับได้อย่างมีประสิทธิภาพ ถึงแม้ว่าเงื่อนไขซ้อนจะสามารถรองรับตรรกะที่ซับซ้อนได้ แต่ก็อาจทำให้โค้ดอ่านยากขึ้นและบำรุงรักษายากขึ้น เนื่องจากระดับการเย็บบรรทัดที่เพิ่มขึ้น ดังนั้น การใช้เงื่อนไขซ้อนควรทำด้วยความระมัดระวัง เพื่อให้โค้ดยังคงความชัดเจนและสามารถดูแลได้ง่าย

```
1 num = float(input("Enter a number: "))
2 if num > 0:
3     print("Positive number")
4 elif num == 0:
5     print("Zero")
6 else:
7     print("Negative number")
```

IF-ELIF-ELSE

A nested decision structure

```
# Try
# num = 5
# num = 0
# num = -4.5
num = float(input("Enter a number: "))
if num > 0:
    print("Positive number")
elif num == 0:
    print("Zero")
else:
    print("Negative number")
```

Enter a number: -4.5
Negative number

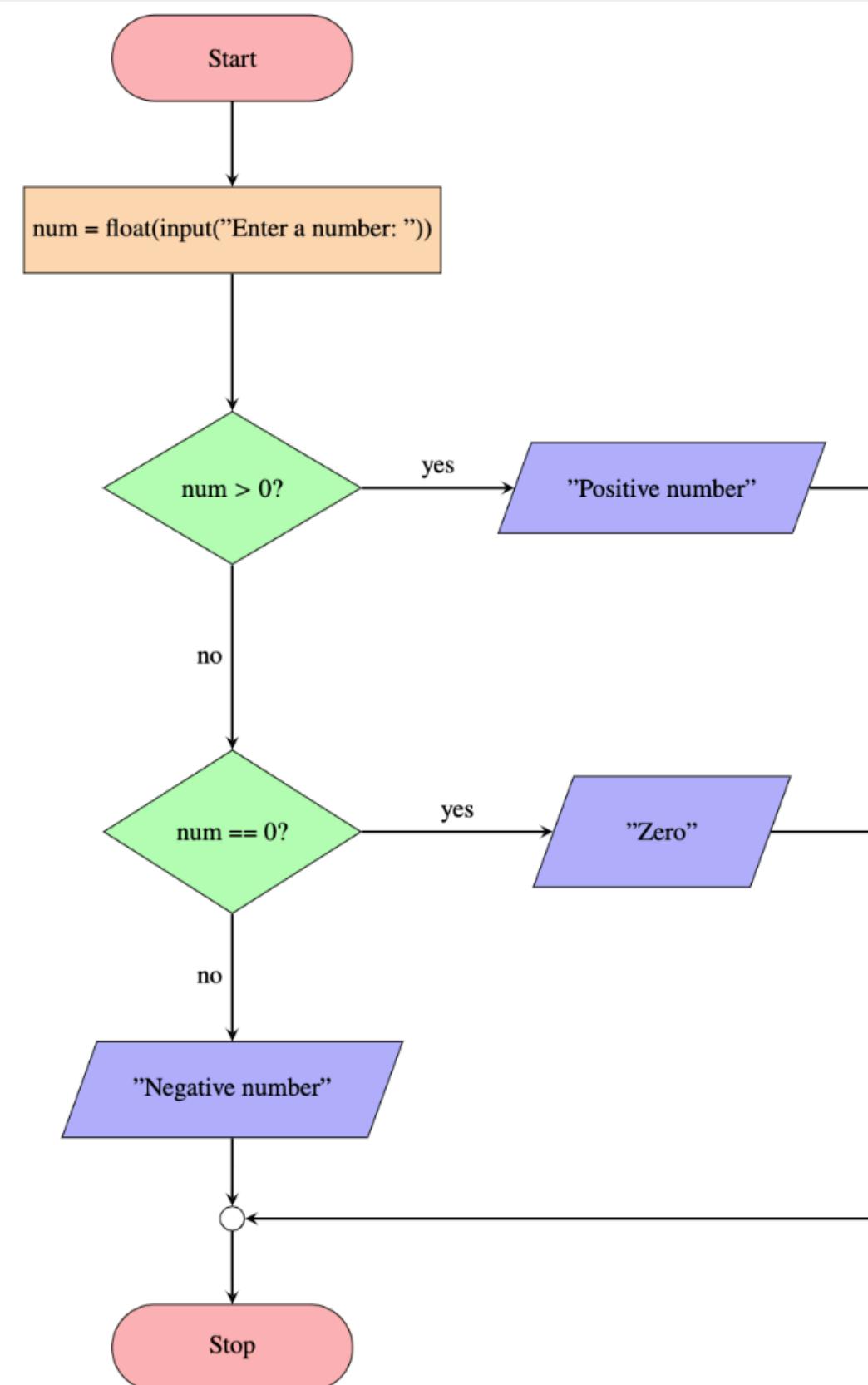


Fig. 3.6: Flowchart for Number Categorization I

NESTED IF

A nested decision structure

```
num = float(input("Enter a number: "))
if num >= 0:
    if num == 0:
        print("Zero")
    else:
        print("Positive number")
else:
    print("Negative number")
```

Enter a number: 6
Positive number

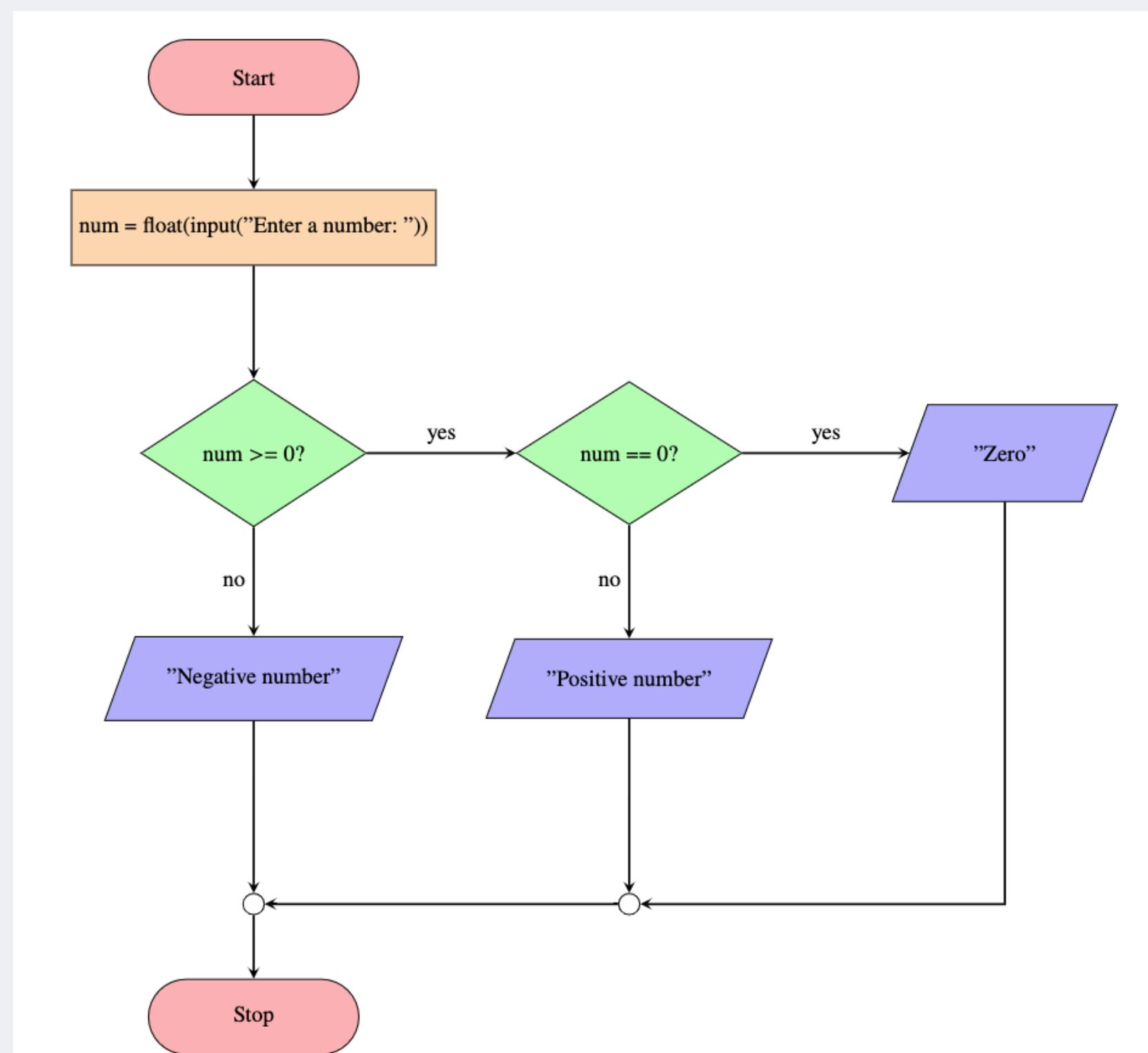


Fig. 3.7: Flowchart for Number Categorization II

IF-ELIF-ELSE

A nested decision structure

การจัดหมวดตัวเลขแบบที่ I: ข้อดี:

- ความชัดเจน: โค้ดนี้อ่านง่าย แยกแยะค่าบวก ศูนย์ และค่าลบได้ชัดเจนโดยใช้คำสั่ง `if`, `elif` และ `else` แยกกัน
- เรียบง่าย: โครงสร้างมีความเรียบง่ายและเข้าใจง่าย เหมาะสำหรับผู้เริ่มต้น

ข้อจำกัด:

- ความซ้ำซ้อน: เงื่อนไข `num == 0` ถูกตรวจสอบแยกจาก `num > 0` ซึ่งในบางกรณีอาจถือเป็นการซ้ำซ้อน
- ผลกระทบด้านประสิทธิภาพ: เมื่จะเลิกน้อยสำหรับโปรแกรมขนาดเล็ก แต่การมีเงื่อนไขแยกอาจกระทบต่อประสิทธิภาพในระบบขนาดใหญ่

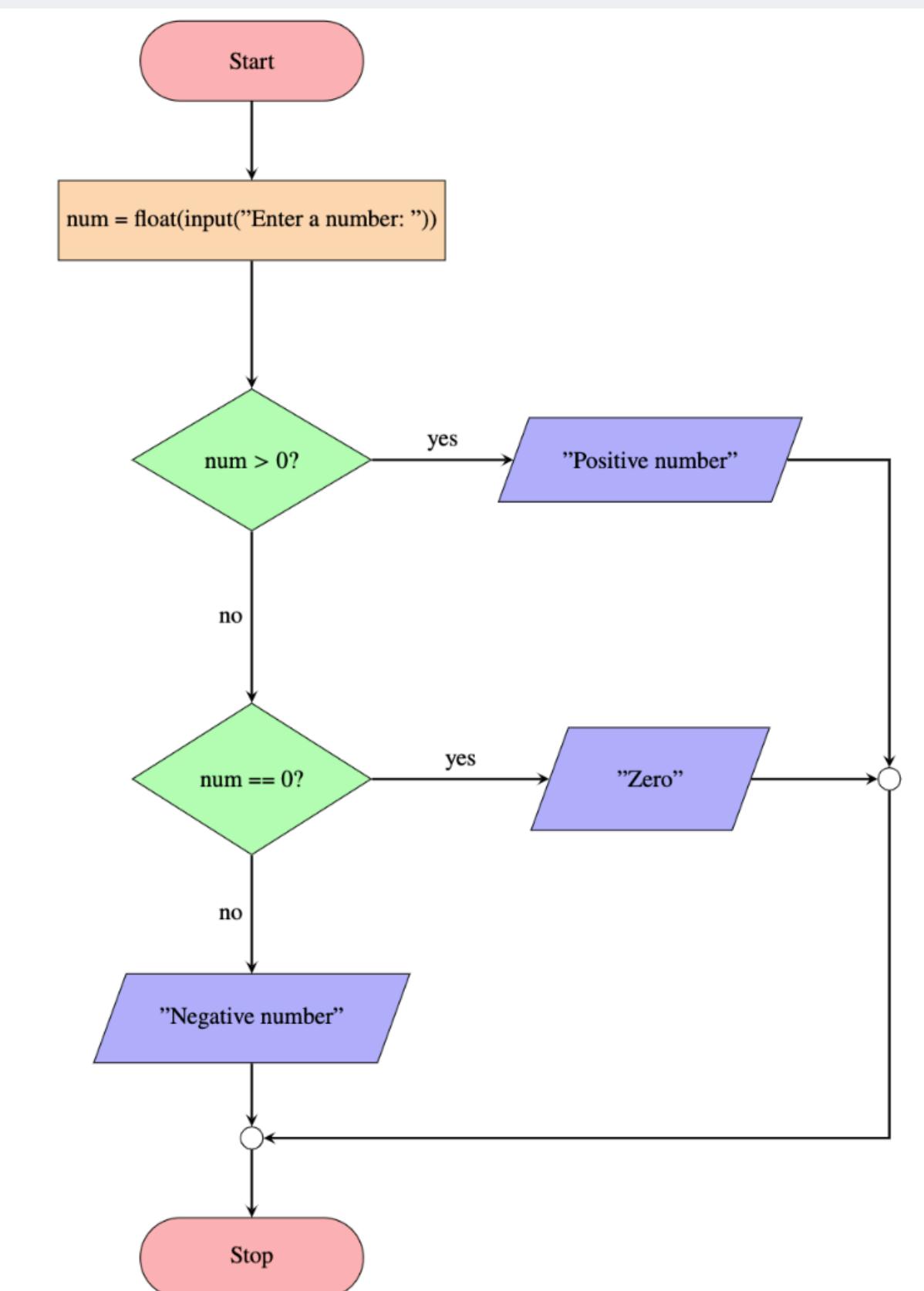


Fig. 3.6: Flowchart for Number Categorization I

IF-ELIF-ELSE

A nested decision structure

การจัดหมวดตัวเลขแบบที่ II: ข้อดี:

- ประสิทธิภาพ: การรวมเงื่อนไข `num >= 0` และตรวจสอบ `num == 0` ภายใน ช่วยลดจำนวนการตรวจสอบแยก ทำให้ประสิทธิภาพดีขึ้นเล็กน้อย
- การจัดกลุ่มตามตรรกะ: การจัดกลุ่มการตรวจสอบค่าบวกและศูนย์เข้าด้วยกัน ช่วยให้การเหลือของตรรกะดูเป็นระบบมากขึ้นสำหรับบางคน

ข้อจำกัด:

- เงื่อนไขซ้อน: การใช้คำสั่ง `if` ซ้อนกันทำให้โค้ดอ่านยาก โดยเฉพาะสำหรับผู้เริ่มต้น และทำให้เกิดการเยื่องที่ซับซ้อน
- ความซับซ้อน: โครงสร้างที่ซ้อนกันทำให้โค้ดดูซับซ้อนมากกว่าแบบแรก แม้ว่าจะทำงานเหมือนกัน

สรุป:

ทั้งสองแนวทางสามารถใช้งานได้และให้ผลลัพธ์เหมือนกัน การเลือกใช้อย่างใดอย่างหนึ่งขึ้นอยู่กับบริบทและความชอบของผู้พัฒนา โปรแกรมแบบแรกอ่านง่ายและตรงไปตรงมา หมายความว่ามีความเข้าใจง่าย แต่ต้องมีความซับซ้อนเล็กน้อย แต่การจัดกลุ่มตามตรรกะในแบบที่ II ทำให้โค้ดดูสวยงามและมีประสิทธิภาพมากขึ้น

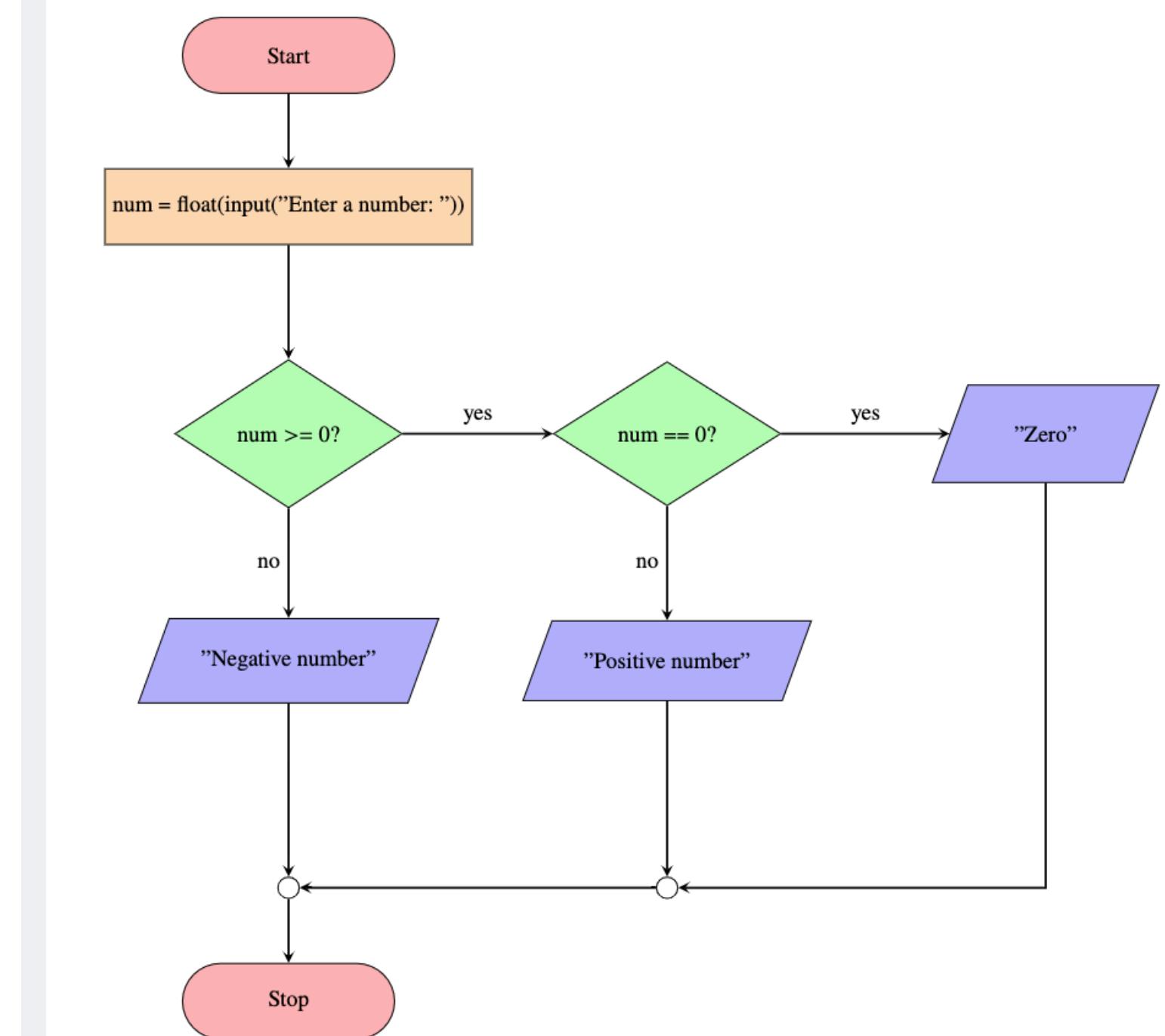


Fig. 3.7: Flowchart for Number Categorization II

EXERCISE-2

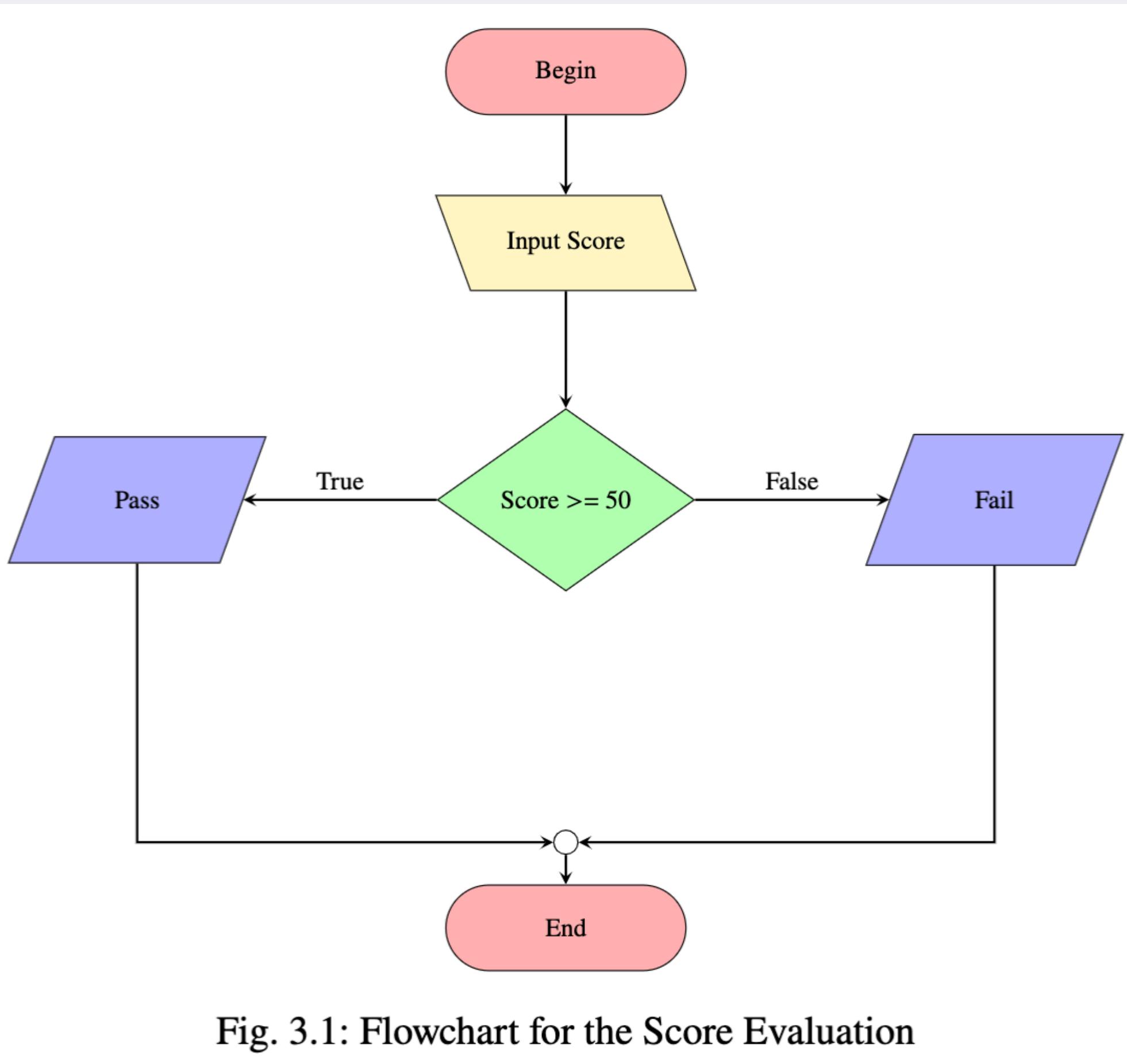


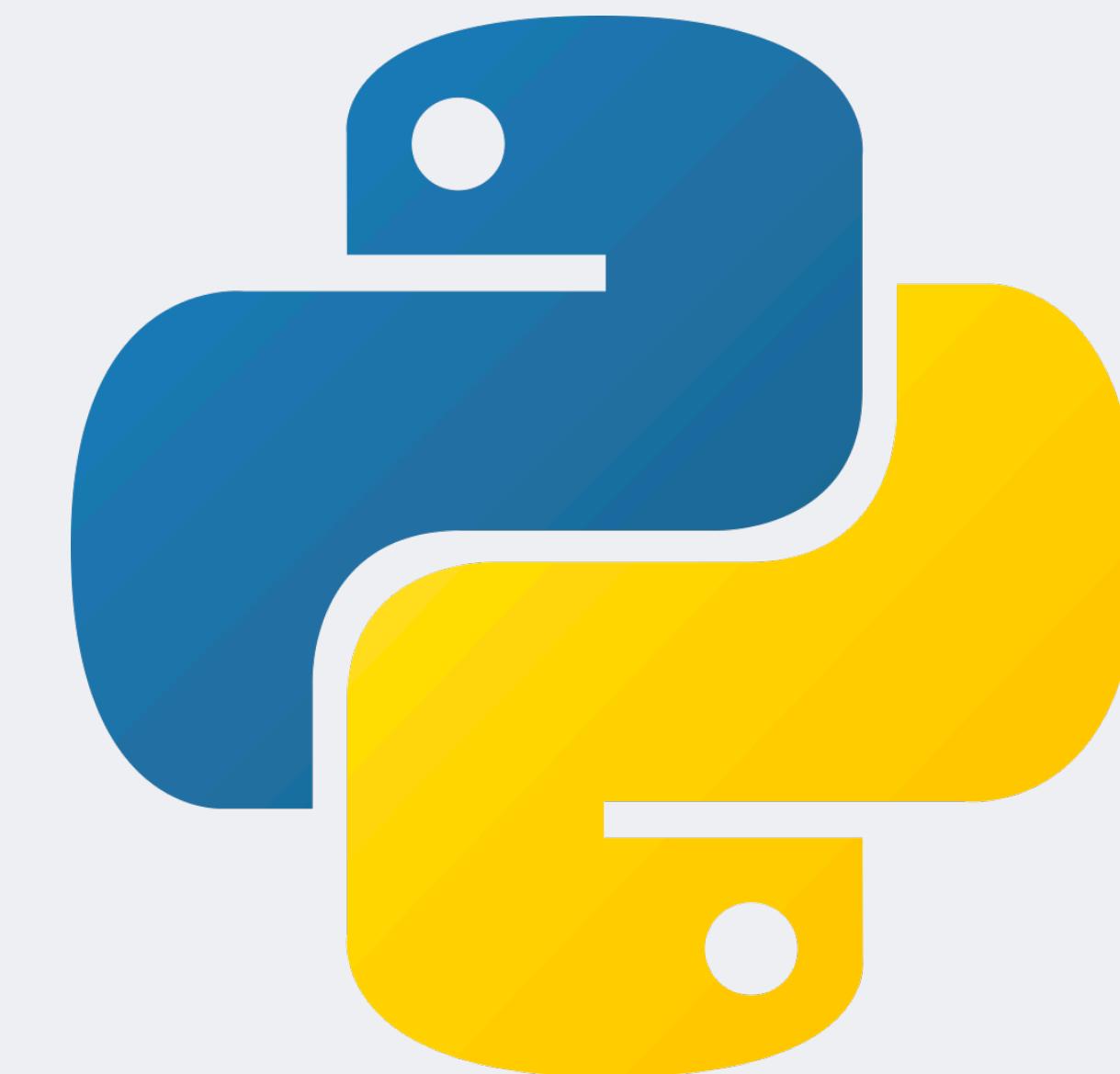
Fig. 3.1: Flowchart for the Score Evaluation

Algorithm 7: Algorithm to Determine Pass or Fail

Input: score
Output: Pass or Fail message

```
1 begin
2     Read score ;
3     if score >= 50 then
4         | Display 'Pass';
5     else
6         | Display 'Fail';
7     end
8 end
```

CONDITIONALS AND CONTROL FLOW



Comparison operators

COMPARISON OPERATORS

แนวคิดสำคัญของตัวดำเนินการเปรียบเทียบ:

- **ตัวดำเนินการเปรียบเทียบ:** ใช้ภายในเงื่อนไขเพื่อเปรียบเทียบค่า เช่น ==, !=, >, <, >=, <=
- **คำสั่งตามเงื่อนไข:** มักใช้งานร่วมกับ if, elif, และ else เพื่อควบคุมการทำงานของโปรแกรม
- **การเปรียบเทียบตัวเลขและข้อความ:** ใช้ได้ทั้งกับค่าตัวเลขและข้อความ
- **การตัดสินใจ:** เป็นเครื่องมือสำคัญในการตัดสินใจและสั่งให้โปรแกรมทำงานตามเงื่อนไขที่กำหนด

COMPARISON OPERATORS

Table 3.1: การเปรียบเทียบตัวดำเนินการ

| ตัวดำเนินการ | ความหมาย | ตัวอย่าง |
|--------------|--|----------|
| > | มากกว่า - คืนค่า <code>True</code> ถ้าค่าทางซ้ายมากกว่าค่าทางขวา | $x > y$ |
| < | น้อยกว่า - คืนค่า <code>True</code> ถ้าค่าทางซ้ายน้อยกว่าค่าทางขวา | $x < y$ |
| == | เท่ากับ - คืนค่า <code>True</code> ถ้าทั้งสองค่ามีค่าเท่ากัน | $x == y$ |
| != | ไม่เท่ากัน - คืนค่า <code>True</code> ถ้าค่าทั้งสองไม่เท่ากัน | $x != y$ |
| >= | มากกว่าหรือเท่ากับ - คืนค่า <code>True</code> ถ้าค่าทางซ้ายมากกว่าหรือเท่ากับขวา | $x >= y$ |
| <= | น้อยกว่าหรือเท่ากับ - คืนค่า <code>True</code> ถ้าค่าทางซ้ายน้อยกว่าหรือเท่ากับขวา | $x <= y$ |

VALUE COMPARISON

Example:

```
1 x = 10
2 y = 20
3
4 print(x == y)    # False
5 print(x != y)    # True
6 print(x > y)     # False
7 print(x < y)     # True
8 print(x >= y)    # False
9 print(x <= y)    # True
```

Listing 3.7: Example of Comparison Operators

CONDITIONALS AND CONTROL FLOW

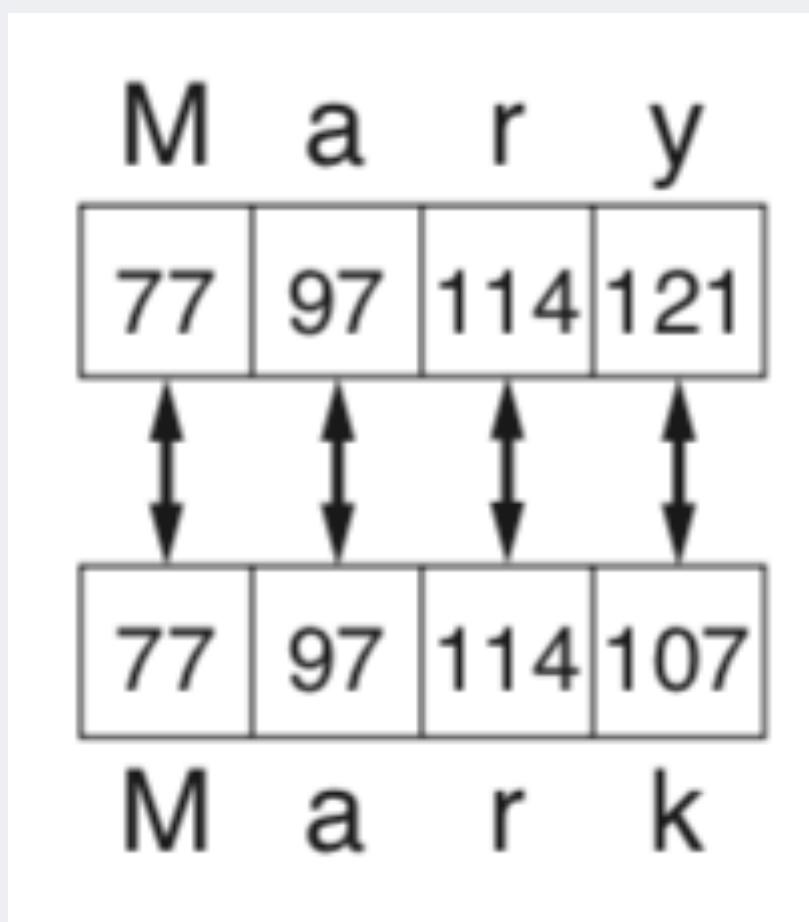


String Compare

STRING COMPARISON

- อักขระตัวพิมพ์ใหญ่ A ถึง Z แทนด้วยตัวเลขตั้งแต่ 65 ถึง 90
- อักขระตัวพิมพ์เล็ก a ถึง z แทนด้วยตัวเลขตั้งแต่ 97 ถึง 122
- เมื่อเลขโดด 0 ถึง 9 จะเก็บไว้ในหน่วยความจำในรูปแบบอักขระ จะถูกแทนด้วยตัวเลขตั้งแต่ 48 ถึง 57
- (ตัวอย่างเช่น สตริง 'abc123' จะถูกเก็บในหน่วยความจำเป็นรหัส 97, 98, 99, 49, 50, และ 51 ตามลำดับ)
- ช่องว่าง (blank space) แทนด้วยตัวเลข 32

STRING COMPARISON



```
1 # Program to compare the strings "Mary" and "Mark"
2
3 # Define the strings
4 string1 = "Mary"
5 string2 = "Mark"
6
7 # Compare the strings for equality
8 if string1 == string2:
9     print(f'"{string1}" and "{string2}" are equal.')
10 else:
11     print(f'"{string1}" and "{string2}" are not equal.')
12
13 # Lexicographical comparison
14 if string1 < string2:
15     print(f'"{string1}" comes before "{string2}" in
16           lexicographical order.')
17 elif string1 > string2:
18     print(f'"{string1}" comes after "{string2}" in
19           lexicographical order.')
20
21 # Case-insensitive comparison
22 if string1.lower() == string2.lower():
23     print(f'"{string1}" and "{string2}" are equal when case is
24           ignored.')
25 else:
26     print(f'"{string1}" and "{string2}" are not equal when case
27           is ignored.'
```

Listing 3.9: Program to compare the strings "Mary" and "Mark"

CONDITIONALS AND CONTROL FLOW



LOGICAL operators

LOGICAL OPERATORS

แนวคิดสำคัญของตัวดำเนินการทางตรรกะ:

- **Logical Operators:** ใช้สำหรับรวมเงื่อนไขหลายเงื่อนไขโดยใช้ `and`, `or`, `not`
- **Combining Conditions:** ใช้สร้างนิพจน์ทางตรรกะที่ซับซ้อนโดยรวมเงื่อนไขหลายประการ
- **Boolean Logic:** ทำงานกับค่าความจริง (`True` หรือ `False`)
- **Conditional Control:** เพิ่มศักยภาพการตัดสินใจภายในคำสั่ง `if`, `elif`, และ `else`
- **Short-Circuit Evaluation:** หยุดการประเมินเงื่อนไขเมื่อทราบผลลัพธ์แล้ว (เช่น `and` จะไม่ประเมินเงื่อนไขที่สองหากเงื่อนไขแรกเป็น `False`)
- **Clarity and Efficiency:** ช่วยให้เขียนโค้ดได้ชัดเจนและมีประสิทธิภาพโดยการรวมการตรวจสอบหลายรายการให้อยู่ในคำสั่งเดียว

LOGICAL OPERATORS

Table 3.2: เปรียบเทียบตัวดำเนินการทางตรรก

| Operator | ความหมาย |
|----------|---|
| and | And (and): คืนค่า True ถ้าเงื่อนไขทั้งสองเป็นจริง |
| or | Or (or): คืนค่า True ถ้ามีเงื่อนไขอย่างน้อยหนึ่งเป็นจริง |
| not | Not (not): คืนค่า True ถ้าเงื่อนไขเป็นเท็จ |

LOGICAL OPERATORS

And (and): คืนค่า True ถ้าเงื่อนไขทั้งสองเป็นจริง

```
1 a and b
```

Or (or): คืนค่า True ถ้ามีเงื่อนไขอย่างน้อยหนึ่งเป็นจริง

```
1 a or b
```

Not (not): คืนค่า True ถ้าเงื่อนไขเป็นเท็จ

```
1 not a
```

LOGICAL OPERATORS

Example:

```
1 x = 10
2 y = 20
3 z = 30
4
5 # Using 'and' operator
6 if x < y and y < z:
7     print("x is less than y and y is less than z.") # True
8
9 # Using 'or' operator
10 if x < y or y > z:
11     print("Either x is less than y or y is greater than z.") #
12     True
13
14 # Using the 'not' operator
15 if not (x > y):
16     print("x is not greater than y.") # True
```

Listing 3.8: Example of Logical Operators

IDENTITY OPERATORS

ตัวดำเนินการเปรียบเทียบอัตลักษณ์ในภาษา Python คือ `is` และ `is not` ใช้เพื่อตรวจสอบว่าสองตัวแปรซึ่งไปยังขอบเขตเดียวกันในหน่วยความจำหรือไม่ ต่างจากตัวดำเนินการเปรียบเทียบค่า (`==`) ที่ใช้ตรวจสอบค่าของตัวแปรตัวดำเนินการเปรียบเทียบอัตลักษณ์ใช้เพื่อตรวจสอบว่าอ้างอิงถึงขอบเขตเดียวกันหรือไม่ เช่น `a is b` จะคืนค่า `True` หาก `a` และ `b` ซึ่งไปยังขอบเขตเดียวกัน ในขณะที่ `a is not b` จะคืนค่า `True` หากซึ่งไปยังขอบเขตคนละตัว ตัวดำเนินการนี้เหมาะสมสำหรับการเปรียบเทียบขอบเขตเพื่อให้แน่ใจว่าทั้งสองอ้างอิงตำแหน่งในหน่วยความจำเดียวกัน ซึ่งช่วยให้มั่นใจถึงความถูกต้องในการเปรียบเทียบข้อมูลเชิงขอบเขต

Table 3.3: เปรียบเทียบตัวดำเนินการเปรียบเทียบอัตลักษณ์

| Operator | ความหมาย | ตัวอย่าง |
|---------------------|---|----------------------------|
| <code>is</code> | คืนค่า <code>True</code> หากตัวแปรทั้งสองซึ่งไปยังขอบเขตเดียวกัน | <code>x is True</code> |
| <code>is not</code> | คืนค่า <code>True</code> หากตัวแปรทั้งสองไม่ซึ่งไปยังขอบเขตเดียวกัน | <code>x is not True</code> |

IDENTITY OPERATORS

```
1 # Example of the identity operator
2
3 # Two variables pointing to the same list object
4 a = [1, 2, 3]
5 b = a
6
7 # Two variables pointing to different list objects with the same
8     content
9 c = [1, 2, 3]
10 d = [1, 2, 3]
11
12 # Using the identity operator
13 print(a is b) # True, since a and b refer to the same object
14 print(a is c) # False, since a and c refer to different objects
15 print(c is d) # False, since c and d refer to different objects
16
17 # Using the equality operator for comparison
18 print(a == c) # True, since the contents of a and c are equal
19 print(c == d) # True, since the contents of c and d are equal
```

Listing 3.10: Example of the identity operator

MEMBERSHIP OPERATORS

ตัวดำเนินการตรวจสอบสมาชิกในภาษา Python ได้แก่ `in` และ `not in` ใช้เพื่อตรวจสอบว่าค่าหรือตัวแปรนั้นอยู่ภายในลำดับข้อมูล เช่น string, list, tuple หรือ set หรือไม่ ตัวดำเนินการ `in` จะคืนค่า `True` หากค่าที่ระบุพบอยู่ในลำดับ ในขณะที่ `not in` จะคืนค่า `True` หากค่านั้นไม่อยู่ในลำดับ ตัวดำเนินการเหล่านี้มีประโยชน์มากในการตรวจสอบการมีอยู่ของสมาชิก และช่วยให้สามารถเขียนเงื่อนไขต่าง ๆ ได้อย่างกระชับและชัดเจนเมื่อทำงานกับโครงสร้างข้อมูล เช่น `x in list` หมายถึงการตรวจสอบว่า `x` เป็นสมาชิกของ `list` หรือไม่ ความสามารถนี้ช่วยเพิ่มประสิทธิภาพและความชัดเจนของโค้ดเมื่อจัดการกับข้อมูลจำนวนมาก

Table 3.4: ตัวดำเนินการตรวจสอบสมาชิก

| Operator | ความหมาย | ตัวอย่าง |
|---------------------|---|-------------------------|
| <code>in</code> | คืนค่า <code>True</code> หากค่าหรือตัวแปรพบในลำดับ | <code>5 in x</code> |
| <code>not in</code> | คืนค่า <code>True</code> หากค่าหรือตัวแปรไม่พบในลำดับ | <code>5 not in x</code> |

MEMBERSHIP OPERATORS

```
1 # Example of membership operators
2
3 # List of fruits
4 fruits = ["apple", "banana", "cherry"]
5
6 # Using 'in' operator
7 print("banana" in fruits) # True, since "banana" is in the list
8 print("orange" in fruits) # False, since "orange" is not in the
  list
9
10 # Using 'not in' operator
11 print("grape" not in fruits) # True, since "grape" is not in the
  list
12 print("apple" not in fruits) # False, since "apple" is in the
  list
13
14 # String example
15 sentence = "The quick brown fox jumps over the lazy dog."
16 print("fox" in the sentence) # True, since "fox" is a substring
  of the sentence
17 print("cat" not in the sentence) # True, since "cat" is not a
  substring of the sentence
```

COMBINING COMPARISON AND LOGICAL OPERATORS

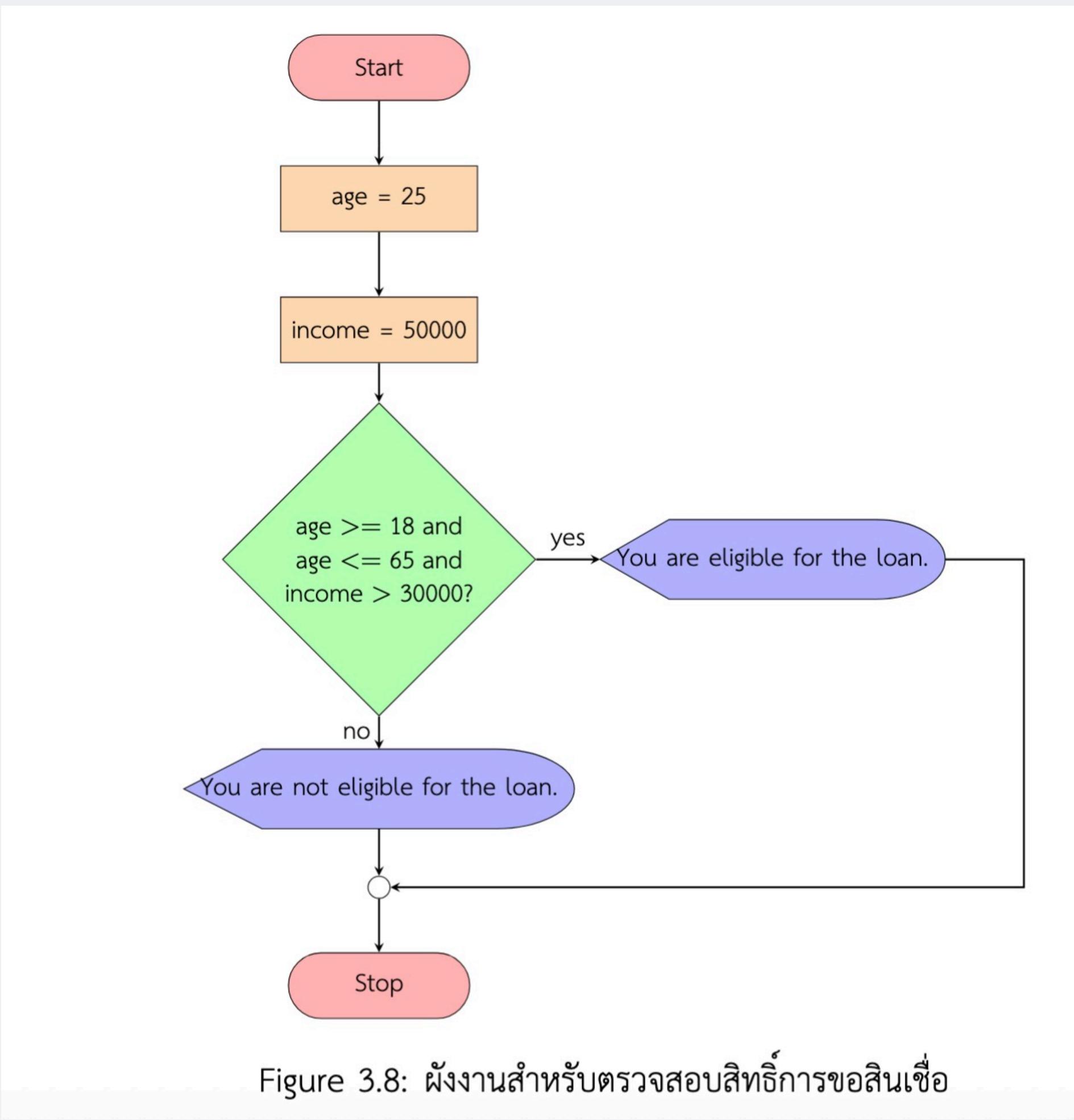
เราสามารถรวมตัวดำเนินการเปรียบเทียบเข้ากับตัวดำเนินการทางตรรกะเพื่อสร้างเงื่อนไขที่ซับซ้อน ซึ่งทำให้โปรแกรมสามารถตัดสินใจได้อย่างแม่นยำและยืดหยุ่นมากขึ้น โดยการใช้ตัวดำเนินการเปรียบเทียบ เช่น `==`, `!=`, `>`, `<`, `>=`, และ `<=` เพื่อประเมินความสัมพันธ์ระหว่างค่าต่าง ๆ และเมื่อนำมาใช้ร่วมกับตัวดำเนินการทางตรรกะ เช่น `and`, `or`, และ `not` ก็จะสามารถตรวจสอบหลายเงื่อนไขพร้อมกันได้ และดำเนินการตามเงื่อนไขที่ซับซ้อน

ตัวอย่างเช่น อาจต้องการตรวจสอบว่าค่าหนึ่งอยู่ในช่วงที่กำหนด และมีสถานะสมาชิกที่ถูกต้องด้วย การรวมเงื่อนไขในลักษณะนี้ช่วยให้สามารถควบคุมการทำงานของโปรแกรมได้อย่างแม่นยำและยืดหยุ่น หมายเหตุสำหรับโปรแกรมที่ต้องรองรับการตัดสินใจที่มีความซับซ้อน การเข้าใจและใช้ตัวดำเนินการเหล่านี้อย่างมีประสิทธิภาพเป็นพื้นฐานสำคัญในการพัฒนาโปรแกรมที่เชื่อถือได้และตอบสนองต่อสถานการณ์ได้ดี

```
1 age = 25
2 income = 50000
3
4 # Check if the age is between 18 and 65 and income is above 30000
5 if age >= 18 and age <= 65 and income > 30000:
6     print("You are eligible for the loan.")
7 else:
8     print("You are not eligible for the loan.")
```

Listing 3.15: Example of Logical Operators

COMBINING COMPARISON AND LOGICAL OPERATORS



EXERCISE-3

Chris owns an auto repair business and has several employees. If any employee works over 40 hours in a week, he pays them 1.5 times their regular hourly pay rate for all hours over 40. He has asked you to design a simple payroll program that calculates an employee's gross pay, including any overtime wages. You design the following algorithm:

Program Output (with input shown in bold)

Enter the number of hours worked: **40**
Enter the hourly pay rate: **20**
The gross pay is \$800.00.

Program Output (with input shown in bold)

Enter the number of hours worked: **50**
Enter the hourly pay rate: **20**
The gross pay is \$1,100.00.

EXERCISE-4

Please select operation -

- 1. Add
- 2. Subtract
- 3. Multiply
- 4. Divide

Select operations form 1, 2, 3, 4 : 1

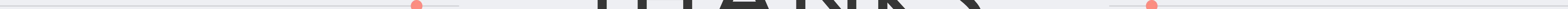
Enter first number : 15

Enter second number : 14

15 + 14 = 29

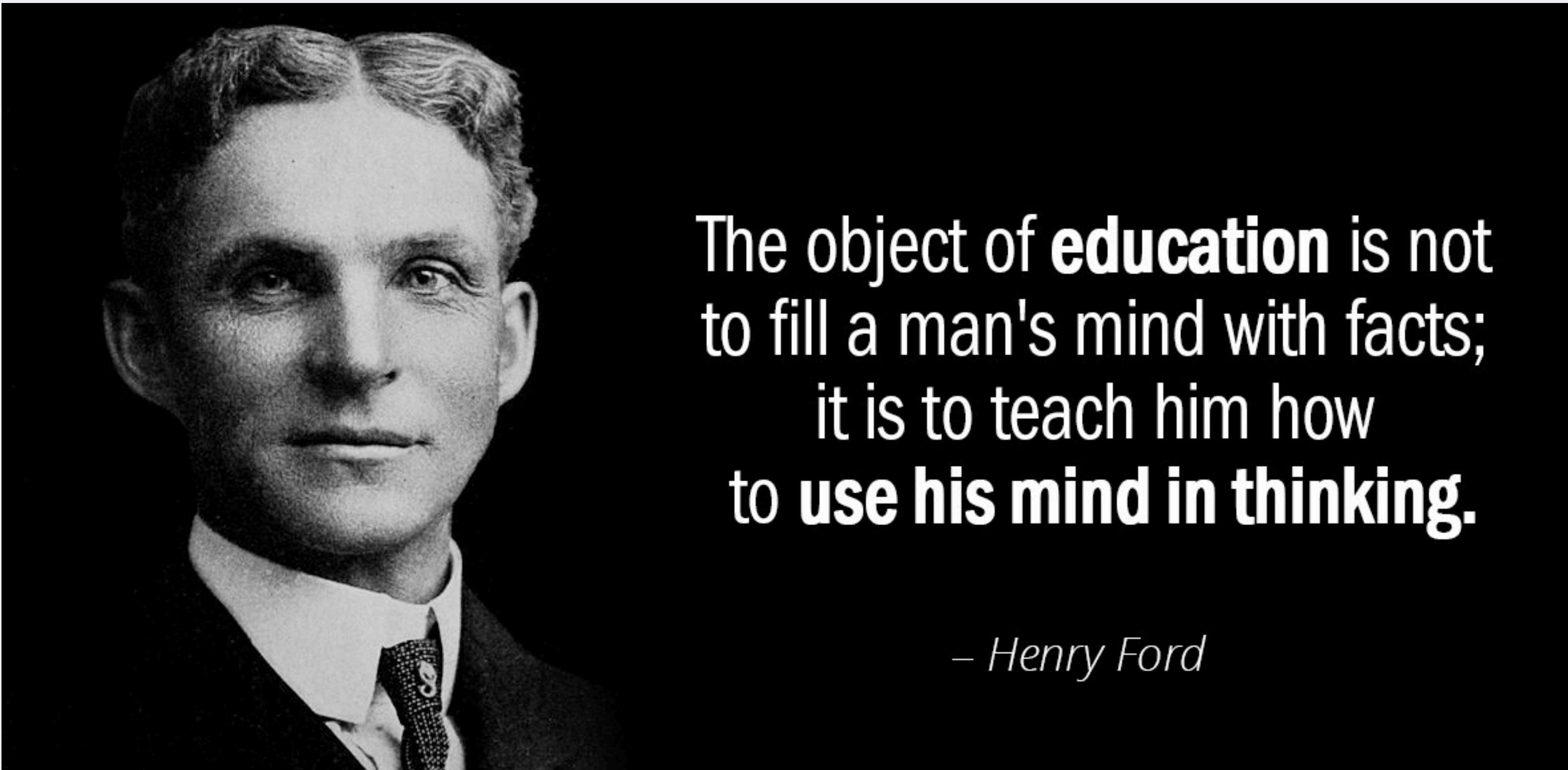
LESSON SUMMARY

- if-elif-else is a decision structure
- The **if** statement performs an action if a condition is True or skips the action if the condition is False.
- The **if... else** statement performs an action if a condition is True or performs a different action if the condition is False.
- The **if... elif... else** statement performs one of many different actions, depending on the truth or falsity of several conditions.
- All of the statements in the block are indented.
- String compare due to ascii table



THANKS

WORD OF THE WISE



The object of **education** is not
to fill a man's mind with facts;
it is to teach him how
to **use his mind in thinking.**

– Henry Ford