คอมพิวเตอร์โปรแกรมมิ่ง (โครงการพี่ติวน้อง)

เพื่อส่งเสริมการมีส่วนร่วมของรุ่นพี่ในการถ่ายทอดความรู้แก้รุ่นน้อง เพื่อเสริมความเข้าใจในรายวิชา

*ยงยุทธ ชวนขุนทด, เมธัส ทองจันทร์, ศุภกร จิรศิริวรกุล, มชัญชยา ประยูรมณีรัตน์

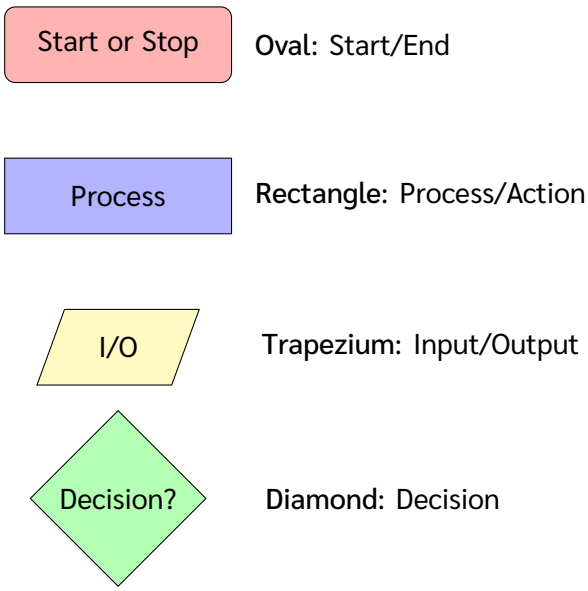
ประจำวันที่ 17 กรกฎาคม พ.ศ. 2568

ภาควิชาเทคโนโลยีสารสนเทศ

มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ วิทยาเขตปราจีนบุรี

สารบัญ

1. ชนิดโฟลวชาร์ต (Flowchart Types)



รูปที่ 1: ตัวอย่างโฟลวชาร์ตพื้นฐาน

โฟลวชาร์ต (Flowchart) เป็นเครื่องมือที่ใช้ในการแสดงลำดับขั้นตอนของกระบวนการหรืออัลกอริธึมในรูปแบบกราฟิก โฟลวชาร์ตประกอบด้วยสัญลักษณ์ต่างๆ ที่แสดงถึงการเริ่มต้นหรือสิ้นสุด (Start/Stop), การดำเนินการ (Process), การตัดสินใจ (Decision), และการป้อนข้อมูลหรือแสดงผล (Input/Output) การใช้โฟลวชาร์ตช่วยให้เข้าใจและวางแผนการเขียนโปรแกรมได้ง่ายขึ้น

ในแต่ละสัญลักษณ์จะถูกเชื่อมด้วย ลูกศร (Arrow) เพื่อแสดงทิศทางของการไหลของข้อมูลหรือการดำเนินการในโฟลวชาร์ต การใช้โฟลวชาร์ตช่วยให้สามารถวางแผนและออกแบบโปรแกรมได้อย่างมีประสิทธิภาพ และยังช่วยในการสื่อสารแนวคิดกับผู้อื่นได้ง่ายขึ้น

2. ตัวแปรและชนิดข้อมูล (Variables and Data Types)

สิ่งสำคัญในการเขียนโปรแกรมคือการเข้าใจตัวแปรและชนิดข้อมูล ซึ่งเป็นพื้นฐานของการจัดเก็บและจัดการข้อมูลในโปรแกรม ตัวแปรใช้เพื่อเก็บข้อมูลที่สามารถ เปลี่ยนแปลง (Mutable) และ ไม่สามารถเปลี่ยนแปลง (Immutable) ได้ในระหว่างการทำงานของโปรแกรม และชนิดข้อมูลกำหนดประเภทของข้อมูลที่ตัวแปรนั้นสามารถเก็บได้

2.1. ตัวแปร (Variables)

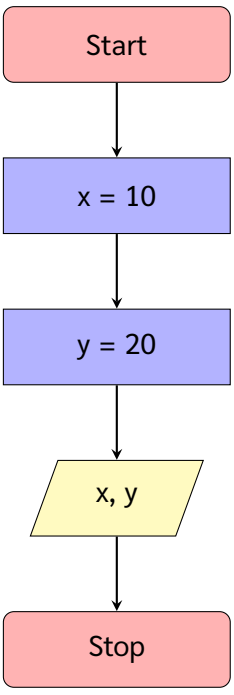
ตัวแปรเป็นชื่อที่ใช้เพื่ออ้างถึงข้อมูลที่เก็บอยู่ในหน่วยความจำของคอมพิวเตอร์ ตัวแปรสามารถเปลี่ยนแปลงค่าได้ในระหว่างการทำงานของโปรแกรม และยังมีตัวแปรที่ไม่สามารถเปลี่ยนแปลงค่าได้ (Immutable) เช่น ค่าคงที่ (Constants) ตัวแปรในภาษาโปรแกรมต่างๆ อาจมีรูปแบบการประกาศที่แตกต่างกัน

(ตัวอย่าง) ตัวแปรใน Python

```python
x = 10
y = 20

print(x, y)
```

ผลลัพธ์

```
10 20
```



รูปที่ 2: ตัวอย่าง Flowchart ของการประกาศตัวแปร

2.2. ชนิดข้อมูล (Data Types)

ชนิดข้อมูล (Data Types) เป็นการกำหนดประเภทของข้อมูลที่ตัวแปรนั้นสามารถเก็บได้ ชนิดข้อมูลที่พบบ่อย ได้แก่

- **Integer (int):** ตัวเลขจำนวนเต็ม เช่น 1, 2, -3
- **Float (float):** ตัวเลขทศนิยม เช่น 3.14, -0.001
- **String (str):** ข้อความหรืออักขระ เช่น ”Hello”, ”123”
- **Boolean (bool):** ค่าจริงหรือเท็จ เช่น True, False
- **List (list):** คอลเลกชันของข้อมูลที่สามารถเปลี่ยนแปลงได้ เช่น [1, 2, 3], [”apple”, ”banana”]
- **Tuple (tuple):** คอลเลกชันของข้อมูลที่ไม่สามารถเปลี่ยนแปลงได้ เช่น (1, 2, 3), (”apple”, ”banana”)
- **Dictionary (dict):** คอลเลกชันของคู่คีย์-ค่า เช่น {”name”: ”John”, ”age”: 30}

(ตัวอย่าง) ชนิดข้อมูลใน Python

```python
# Integer
x = 10

# Float
y = 3.14

# String
name = "First"

# Boolean
is_active = True

# List
scores = [85, 90, 78, 92]

# Tuple
coordinates = (10.5, 20.3)

# Dictionary
person = {"name": "John", "age": 30}

print(x, y, name, is_active, scores, coordinates, person)
```

ผลลัพธ์

```
10 3.14 First True [85, 90, 78, 92] (10.5, 20.3) {'name': 'John', 'age': 30}
```

ในแต่ละชนิดข้อมูลจะมี Utility function (ฟังก์ชันที่ใช้ในการจัดการข้อมูล) ที่ช่วยให้สามารถทำงานกับข้อมูลได้ง่ายขึ้น

3. คำสั่งแบบมีเงื่อนไข (Conditional Statements)

คำสั่งแบบมีเงื่อนไข (Conditional Statements) เป็นคำสั่งที่ใช้ในการตัดสินใจว่าควรทำอะไรต่อไปในโปรแกรมตามเงื่อนไขที่กำหนด คำสั่งเหล่านี้ช่วยให้โปรแกรมสามารถทำงานได้อย่างยืดหยุ่นและตอบสนองต่อสถานการณ์ต่างๆ (โค้ดด้านล่าง)

3.1. การใช้คำสั่ง if เบื้องต้น

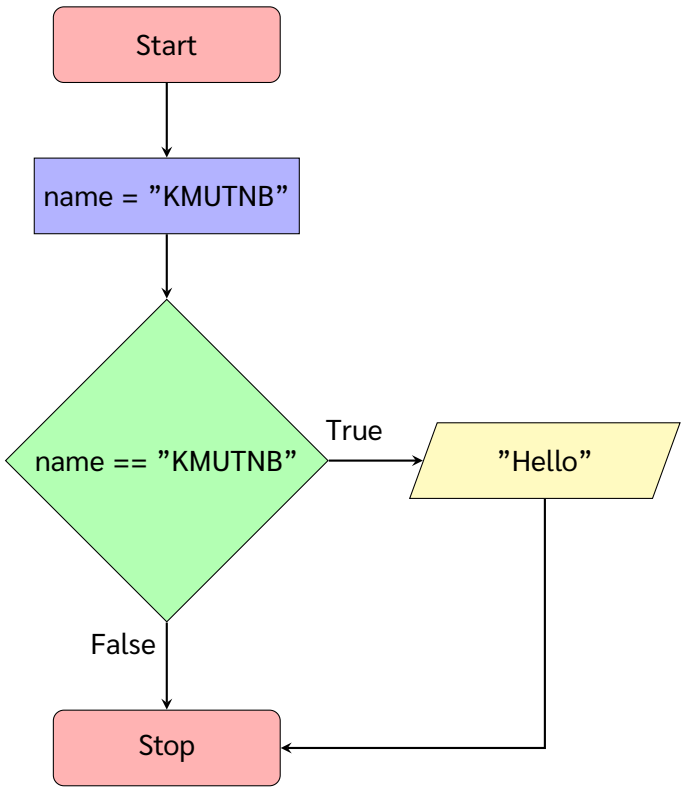
คำสั่ง if ใช้เพื่อตรวจสอบเงื่อนไข ถ้าเงื่อนไขเป็นจริง (True) จะทำการดำเนินการตามที่กำหนดไว้ในบล็อกของ if ดังกล่าว
สิ่งที่ต้องการ: ฉันต้องการให้แสดงข้อความ ‘Hello‘ หากตัวแปร ‘name‘ มีค่าเป็น ‘KMUTNB‘

(ตัวอย่าง) การใช้ if ใน Python

```
1  name = "KMUTNB"
2
3  if name == "KMUTNB":
4      print("Hello")
```

ไขข้อสงสัย!

== หมายความว่า เท่ากับ (Equal) ใช้เพื่อตรวจสอบว่าค่าของตัวแปร ‘name‘ เท่ากับ ”KMUTNB” หรือไม่



รูปที่ 3: ตัวอย่างโฟลวชาร์ตเงื่อนไข if

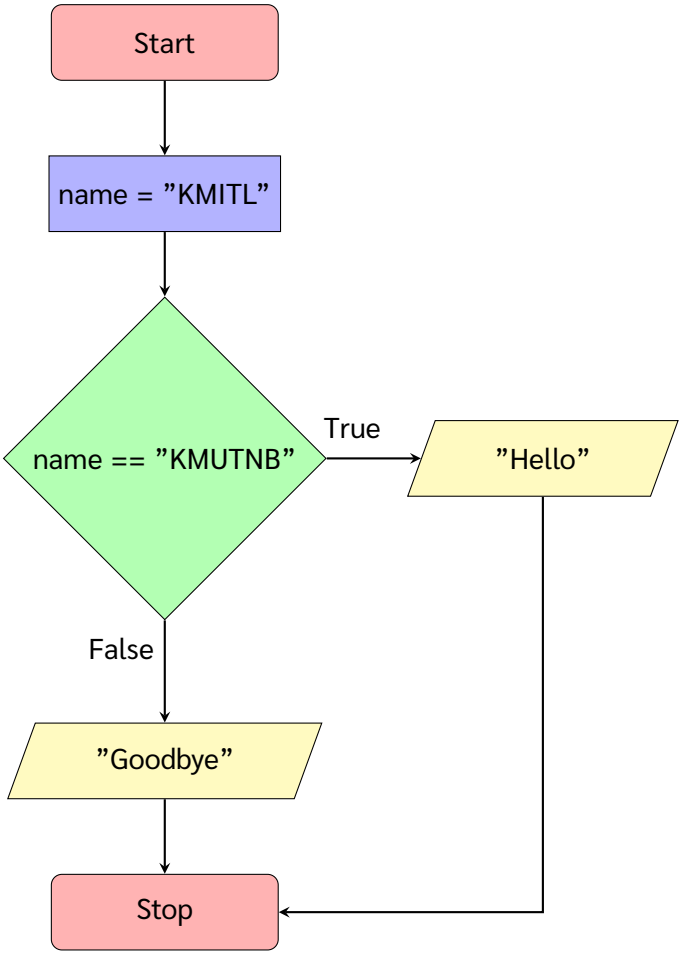
3.2. การใช้คำสั่ง if-else เบื้องต้น

คำสั่ง if-else ใช้เพื่อตรวจสอบเงื่อนไข ถ้าเงื่อนไขเป็นจริง (True) จะทำการดำเนินการตามที่กำหนดไว้ในบล็อกของ if แต่ถ้าเงื่อนไขเป็นเท็จ (False) จะทำการดำเนินการตามที่กำหนดไว้ในบล็อกของ else

สิ่งที่ต้องการ: ฉันต้องการให้แสดงข้อความ ‘Hello‘ หากตัวแปร ‘name‘ มีค่าเป็น ‘KMUTNB‘ และแสดงข้อความ ‘Goodbye‘ หากไม่ตรงกับเงื่อนไข ใดๆ

(ตัวอย่าง) การใช้ if-else ใน Python

```
name = "KMITL"

if name == "KMUTNB":
    print("Hello")
else:
    print("Goodbye")
```



รูปที่ 4: ตัวอย่างโฟลวชาร์ตเงื่อนไข if-else

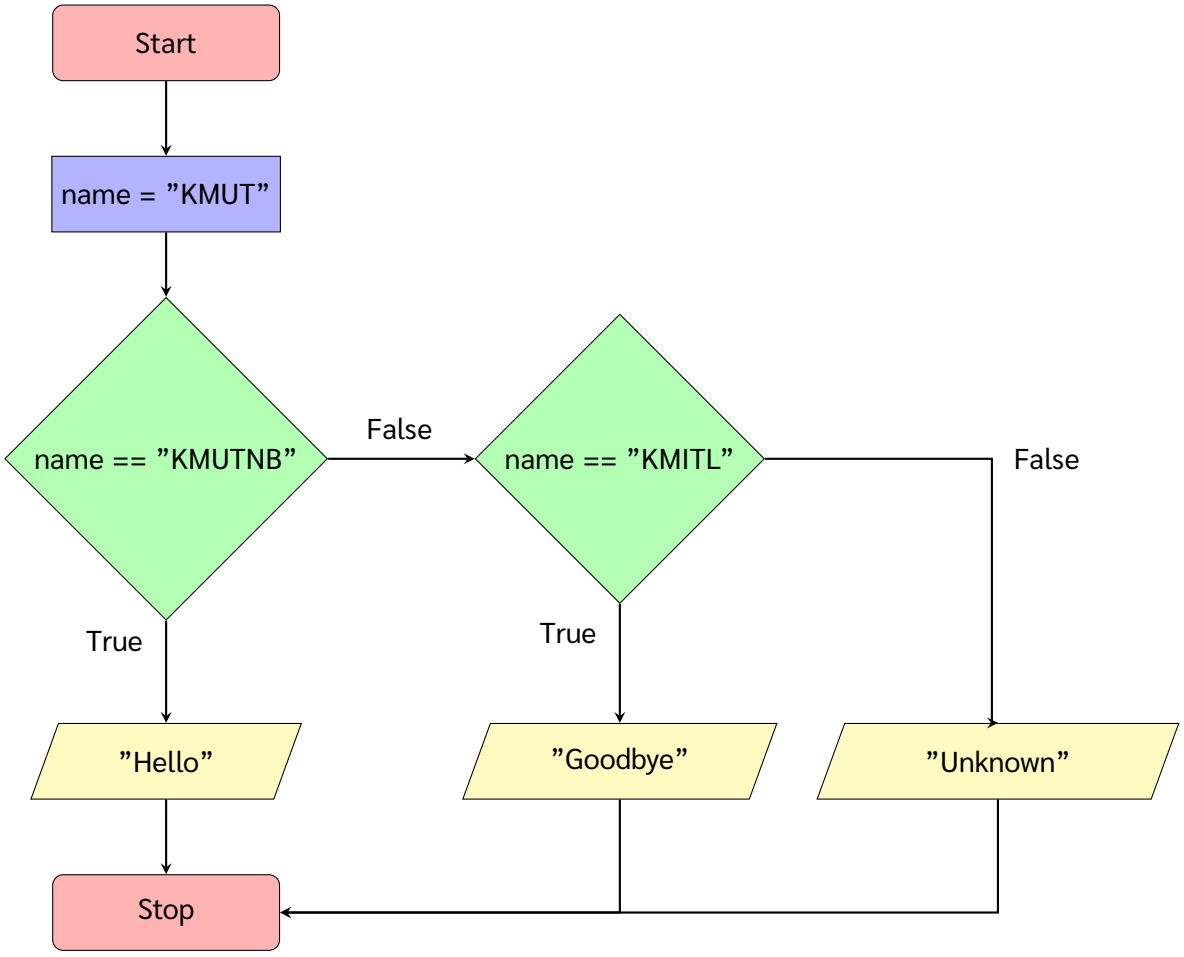
3.3. การใช้คำสั่ง if-elif-else เบื้องต้น

คำสั่ง if-elif-else ใช้เพื่อตรวจสอบหลายเงื่อนไข ถ้าเงื่อนไขแรกเป็นจริง (True) จะทำการดำเนินการตามที่กำหนดไว้ในบล็อกของ if แต่ถ้าเงื่อนไขแรกเป็นเท็จ (False) จะตรวจสอบเงื่อนไขถัดไป (elif) และถ้าไม่มีเงื่อนไขใดเป็นจริง จะทำการดำเนินการตามที่กำหนดไว้ในบล็อกของ else

สิ่งที่ต้องการ: ฉันต้องการให้แสดงข้อความ ‘Hello‘ หากตัวแปร ‘name‘ มีค่าเป็น ‘KMUTNB‘ , แสดงข้อความ ‘Goodbye‘ หากตัวแปร ‘name‘ มีค่าเป็น ‘KMITL‘, และแสดงข้อความ ‘Unknown‘ หากไม่ใช่ทั้งสองกรณี

(ตัวอย่าง) การใช้ if-elif-else ใน Python

```python
name = "KMUT"

if name == "KMUTNB":
    print("Hello")
elif name == "KMITL":
    print("Goodbye")
else:
    print("Unknown")
```



รูปที่ 5: ตัวอย่างโฟลวชาร์ตเงื่อนไข if-elif-else

3.4. บททดสอบ

แนะนำเครื่องหมายดำเนินการใหม่!

เครื่องหมายดำเนินการทางคณิตศาสตร์ (Arithmetic Operators) ที่ใช้ใน Python มีดังนี้:

- **+** : การบวก (Addition)
- **-** : การลบ (Subtraction)
- ***** : การคูณ (Multiplication)
- **/** : การหาร (Division) - ผลลัพธ์เป็นทศนิยม
- **//** : การหารปัดเศษลง (Floor Division) - ผลลัพธ์เป็นจำนวนเต็ม
- **%** : การหาเศษเหลือจากการหาร (Modulus)
- ****** : การยกกำลัง (Exponentiation)

(ตัวอย่าง) การใช้ Arithmetic Operators

```
a = 10
b = 3

print("a + b =", a + b)
print("a - b =", a - b)
print("a * b =", a * b)
print("a / b =", a / b)
```

ผลลัพธ์

a + b = 13
a - b = 7
a * b = 30
a / b = 3.333333...

การใช้งาน Arithmetic Operators กับ Conditional Statements

(ตัวอย่าง) การใช้ Arithmetic ตรวจสอบเลข

```
number = 10

if number / 2 == 5:
    print(f"{number}, Wow!")
else:
    print(f"{number}, Okay!")
```

แนะนำคำสั่งใหม่!

หากต้องการรับค่าจากผู้ใช้ใน Python สามารถใช้คำสั่ง **input()** ได้ (ค่าชนิดข้อมูลเริ่มต้นของ input() คือ String) เช่น

```
1 name = input("Enter your name: ")
2 print("Hello", name)
```

ผลลัพธ์

Enter your name: <ใส่ชื่อของคุณ>
Hello <ชื่อของคุณ>

หากต้องการแปลงค่าที่รับเข้ามาเป็นชนิดข้อมูลอื่น เช่น แปลงเป็นจำนวนเต็ม สามารถใช้คลาส **int()** ได้ เช่น

```
1 name = int(input("Enter your age: "))
```

และอื่น ๆ เช่น แปลงเป็นจำนวนทศนิยมด้วย **float()**

3.4.1 แบบฝึกหัดการใช้เงื่อนไข

แบบฝึกหัดเพื่อฝึกการใช้คำสั่งเงื่อนไขในการแก้ปัญหาจริง พร้อมทำ Flowchart และโค้ด Python ตามโจทย์ที่กำหนด

โจทย์ที่ 1: การตรวจสอบเลขคู่คี่

จงเขียนโปรแกรมที่รับตัวเลขจากผู้ใช้ และแสดงผลว่าตัวเลขนั้นเป็นเลขคู่หรือเลขคี่

โจทย์ที่ 2: การตรวจสอบค่าเฉลี่ย

จงเขียนโปรแกรมที่ค่าคะแนน 3 ส่วนจากผู้ใช้งาน และหาค่าเฉลี่ยของคะแนนนั้น ถ้าค่าเฉลี่ยมากกว่าหรือเท่ากับ 60 ให้แสดงผลว่า ”ผ่าน” ถ้าน้อยกว่า 60 ให้แสดงผลว่า ”ไม่ผ่าน”

โจทย์ที่ 3: การตรวจสอบอายุ

จงเขียนโปรแกรมที่รับอายุจากผู้ใช้ และแสดงผลว่า:

- อายุต่ำกว่า 13 ปี: เด็ก
- อายุ 13-19 ปี: วัยรุ่น
- อายุ 20-59 ปี: ผู้ใหญ่
- อายุ 60 ปีขึ้นไป: ผู้สูงอายุ

4. การวนซ้ำ (Iteration)

การวนซ้ำ (Iteration) เป็นกระบวนการที่ทำให้โปรแกรมสามารถทำงานซ้ำๆ ตามเงื่อนไขที่กำหนด การวนซ้ำช่วยให้สามารถทำงานกับชุดข้อมูลขนาดใหญ่ได้อย่างมีประสิทธิภาพ โดยไม่ต้องเขียนโค้ดซ้ำๆ หลายครั้ง โดยทั่วไปแล้ว การวนซ้ำใน Python มีสองรูปแบบหลักคือ for และ while

แนะนำเครื่องหมายเปรียบเทียบ!

เครื่องหมายเปรียบเทียบ (Comparison Operators) ที่ใช้ในการตรวจสอบเงื่อนไข:

- == : เท่ากับ (Equal to)
- != : ไม่เท่ากับ (Not equal to)
- > : มากกว่า (Greater than)
- < : น้อยกว่า (Less than)
- >= : มากกว่าหรือเท่ากับ (Greater than or equal to)
- <= : น้อยกว่าหรือเท่ากับ (Less than or equal to)

(ตัวอย่าง) การใช้ Comparison Operators

```
age = 20
score = 85

# Comparison Operators
print("age > 18:", age > 18)           # True
print("score <= 90:", score <= 90)     # True
print("age != 25:", age != 25)         # True
print("age == 20:", age == 20)         # True
print("score >= 80:", score >= 80)     # True
print("age < 25:", age < 25)           # True
```

ผลลัพธ์

```
age > 18: True
score <= 90: True
age != 25: True
age == 20: True
score >= 80: True
age < 25: True
```

แนะนำเครื่องหมายตรรกะ!

เครื่องหมายตรรกะ (Logical Operators) ที่ใช้ในการรวมเงื่อนไขหลายๆ อัน:

- **and** : และ - ต้องเป็นจริงทั้งสองฝั่ง
- **or** : หรือ - เป็นจริงฝั่งใดฝั่งหนึ่ง
- **not** : ไม่ - กลับค่าความจริง หรือ กลับค่าเป็นเท็จ
- **in** : อยู่ใน - ตรวจสอบว่าข้อมูลอยู่ในชุดข้อมูลหรือไม่

(ตัวอย่าง) การใช้ Logical Operators

```
age = 20
score = 85
name = "Alice"
subjects = ["Math", "Science", "English"]

# Logical Operators
print("age > 18 and score >= 80:", age > 18 and score >= 80)  # True
print("age < 18 or score > 90:", age < 18 or score > 90)      # False
print("not (age < 18):", not (age < 18))                      # True

# In Operator
print("'Math' in subjects:", "Math" in subjects)              # True
print("'Art' in subjects:", "Art" in subjects)                # False
print("'Alice' in name:", "Alice" in name)                    # True
```

ผลลัพธ์

```
age > 18 and score >= 80: True
age < 18 or score > 90: False
not (age < 18): True
'Math' in subjects: True
'Art' in subjects: False
'Alice' in name: True
```

เครื่องหมาย ตรรกะ หรือ เปรียบเทียบ ที่เห็นตามเนื้อหาด้านบนทั้งหมด สามารถนำมาประยุกต์ใช้งานกับ **if-elif-else (Conditional Statements)** ได้เช่นกัน ซึ่งทำให้เราสามารถตรวจสอบเงื่อนไขได้อย่างยืดหยุ่นและมีประสิทธิภาพมากขึ้น

4.1. การวนซ้ำด้วย for

คำสั่ง for ใช้ในการวนซ้ำผ่านชุดข้อมูล เช่น รายการ (List), ทูเพิล (Tuple), หรือช่วงของตัวเลข (Range) โดยจะทำงานตามจำนวนรอบที่กำหนดไว้ในชุดข้อมูลนั้นหรือตามจำนวนของข้อมูล

แนะนำคำสั่งใหม่!

range(start, stop, step) เป็นฟังก์ชันที่ใช้สร้างลำดับของตัวเลข โดยมี 1 พารามิเตอร์ที่จำเป็นต้องใส่คือ start ส่วน stop และ step เป็นพารามิเตอร์ที่ไม่จำเป็นต้องใส่

หากต้องการให้มีการหยุดที่ระยะห่างที่กำหนด สามารถใส่ตัวเลขได้ เช่น range(0, 10) ผลลัพธ์จะเป็นตัวเลขตั้งแต่ 0 ถึง 9

แต่ถ้าหากต้องการให้มีการเพิ่มทีละ 2 สามารถใช้ range(0, 10, 2) ได้ ซึ่งจะได้ผลลัพธ์เป็น 0, 2, 4, 6, 8

(ตัวอย่าง) การวนซ้ำด้วย range(...) กับ start ใน Python

```
1 for i in range(3):
2     print("Iteration", i)
```

ผลลัพธ์

```
Iteration 0
Iteration 1
Iteration 2
```

(ตัวอย่าง) การวนซ้ำด้วย range(...) กับ stop ใน Python

```
1 for i in range(0, 2):
2     print("Iteration with stop", i)
```

ผลลัพธ์

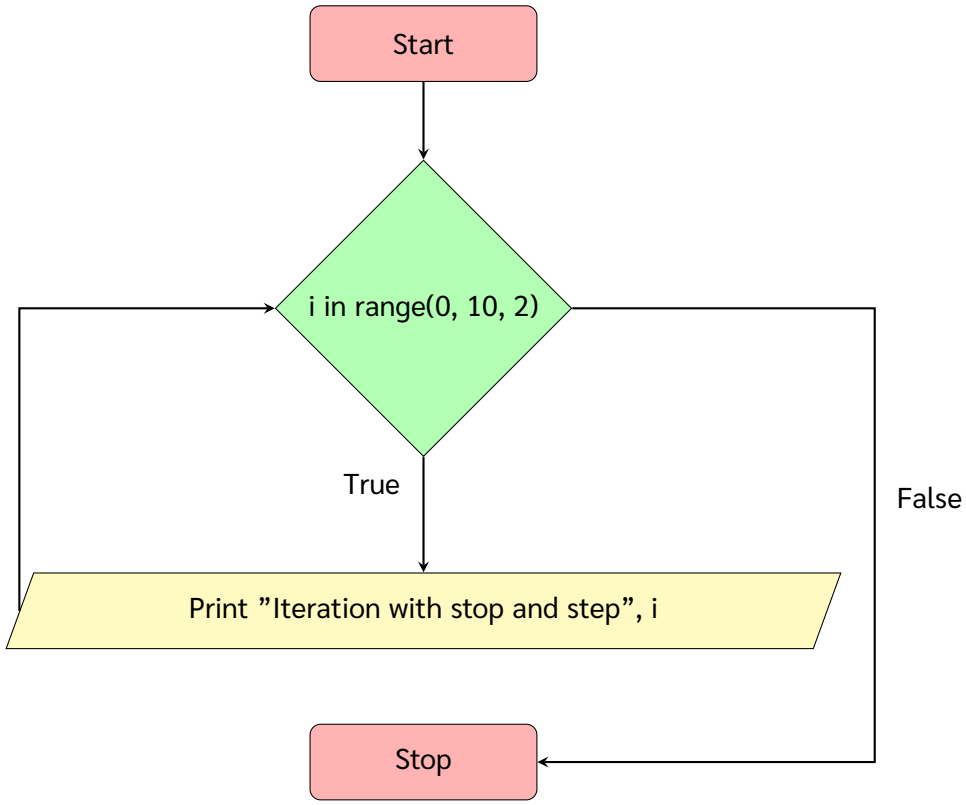
```
Iteration with stop 0
Iteration with stop 1
```

(ตัวอย่าง) การวนซ้ำด้วย range(...) กับ step ใน Python

```
1 for i in range(0, 10, 2):
2     print("Iteration with stop and step", i)
```

ผลลัพธ์

Iteration with stop and step 0
Iteration with stop and step 2
Iteration with stop and step 4
Iteration with stop and step 6
Iteration with stop and step 8



รูปที่ 6: ตัวอย่างโฟลวชาร์ตการวนซ้ำด้วย range(start, stop, step)

ไขข้อสงสัย!

โฟลวชาร์ต (Flowchart) สามารถวางรูปแบบใดก็ได้ ตามความเหมาะสมของลักษณะการทำงานของโปรแกรม โดยทิศทางของลูกศรจะชี้ไปยังขั้นตอนถัดไปที่ต้องดำเนินการ การใช้โฟลวชาร์ตช่วยให้เข้าใจลำดับการทำงานของโปรแกรมได้ง่ายขึ้น

แนะนำคำสั่งใหม่!

len(obj) คือฟังก์ชันที่ใช้ในการหาความยาวของชุดข้อมูล เช่น รายการ (List), สตริง (String), ทูเพิล (Tuple) หรือ ดิชันนารี (Dictionary) เป็นต้น

ซึ่งฟังก์ชันดังกล่าวจะคืนค่ากลับมาเป็น จำนวนเต็ม (Integer) ของชุดข้อมูล นั้นๆ ที่คุณใช้ len(...)

หากเป็น Dictionary จะนับจำนวนคีย์ (Keys) ที่มีอยู่ใน Dictionary นั้น ๆ

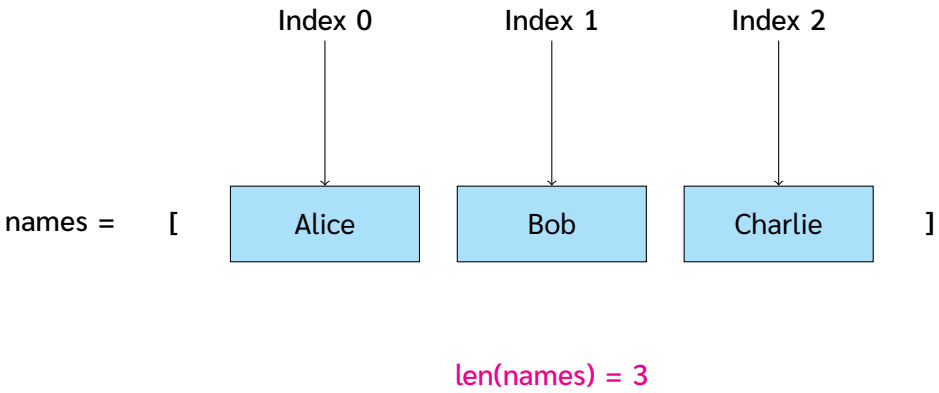
(ตัวอย่าง) การใช้ฟังก์ชัน len(...) ใน Python

```
1 names = ["Alice", "Bob", "Charlie"]
2 print(len(names))
```

ผลลัพธ์

3

Index หรือ ดัชนี เป็นตัวเลขที่ใช้ระบุตำแหน่งของสมาชิกในชุดข้อมูล เช่น รายการ (List) ใน Python โดยเริ่มนับจาก 0 ดังนั้นสมาชิกแรกจะมีดัชนีเป็น 0, สมาชิกที่สองจะมีดัชนีเป็น 1 และต่อไปเรื่อย ๆ



รูปที่ 7: การแสดงดัชนี (Index) ของรายการ (List) ใน Python

ไขข้อสงสัย!

ดัชนี (Index) ในรายการ (List) เริ่มนับจาก 0 เสมอ ดังนั้นรายการที่มี 3 สมาชิก จะมีดัชนีเป็น 0, 1, และ 2 ตามลำดับ

การใช้ **names[0]** จะได้ค่า "Alice", **names[1]** จะได้ค่า "Bob", และ **names[2]** จะได้ค่า "Charlie"

ฟังก์ชัน **len(names)** จะคืนค่าจำนวนสมาชิกทั้งหมดในรายการ ซึ่งในกรณีนี้คือ 3

การเข้าถึงสมาชิกในรายการ (List) หรือทูเพิล (Tuple) สามารถทำได้โดยใช้ดัชนี (Index) ซึ่งเป็นตัวเลขที่ระบุตำแหน่งของสมาชิกในชุดข้อมูลนั้น ๆ อย่างเช่น **names[0]** จะเข้าถึงสมาชิกแรกของรายการ names ซึ่งคือ **"Alice"** หรือการเข้าถึงแบบ Negative Index เช่น **names[-1]** จะเข้าถึงสมาชิกสุดท้ายของรายการ names ซึ่งคือ **"Charlie"**

(ตัวอย่าง) การเข้าถึงสมาชิกในรายการ (List) ด้วยดัชนี (Index) ใน Python

```
1 names = ["Alice", "Bob", "Charlie"]
2 print(names[0])
3 print(names[-1])
4 print(names[2])
```

ผลลัพธ์

Alice
Charlie
Charlie

การเข้าถึงสมาชิกแบบ Slice (Slicing) ช่วยให้สามารถเข้าถึงสมาชิกในรายการ (List) หรือทูเพิล (Tuple) ได้หลายตัวพร้อมกัน โดยใช้รูปแบบ **list[start:stop:step]** ซึ่งจะคืนค่าชุดข้อมูลย่อยที่เริ่มจากดัชนี start ถึง stop (ไม่รวม stop) และเพิ่มทีละ step

(ตัวอย่าง) การวนซ้ำด้วย range(...) กับ len(...) ใน Python

```
1 names = ["Alice", "Bob", "Charlie"]
2 print(names[0:2])  # Slicing from index 0 to 1
3 print(names[1:])   # Slicing from index 1 to the end
4 print(names[:3])   # Slicing from the start and stop at index 3 (Take only 0, 1, 2)
5 print(names[::2])  # Slicing with step 2
```

ผลลัพธ์

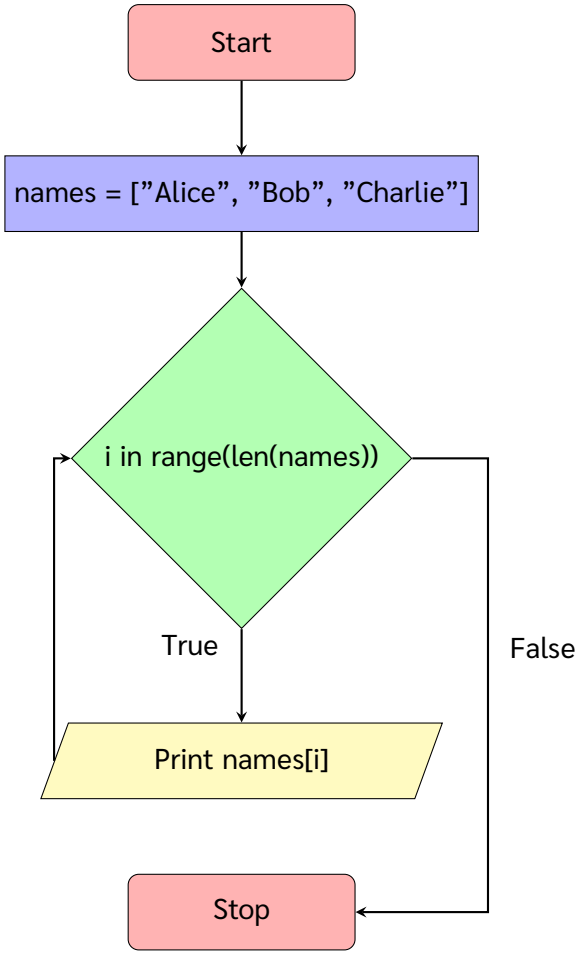
['Alice', 'Bob']
['Bob', 'Charlie']
['Alice', 'Bob', 'Charlie']
['Alice', 'Charlie']

(ตัวอย่าง) การวนซ้ำด้วย for กับ range(...) และ len(...) ใน Python

```python
names = ["Alice", "Bob", "Charlie"]
for i in range(len(names)):
    print(names[i])
```

ผลลัพธ์

```
Alice
Bob
Charlie
```



รูปที่ 8: ตัวอย่างโฟลวชาร์ตการวนซ้ำด้วย for กับ range(...) และ len(...)

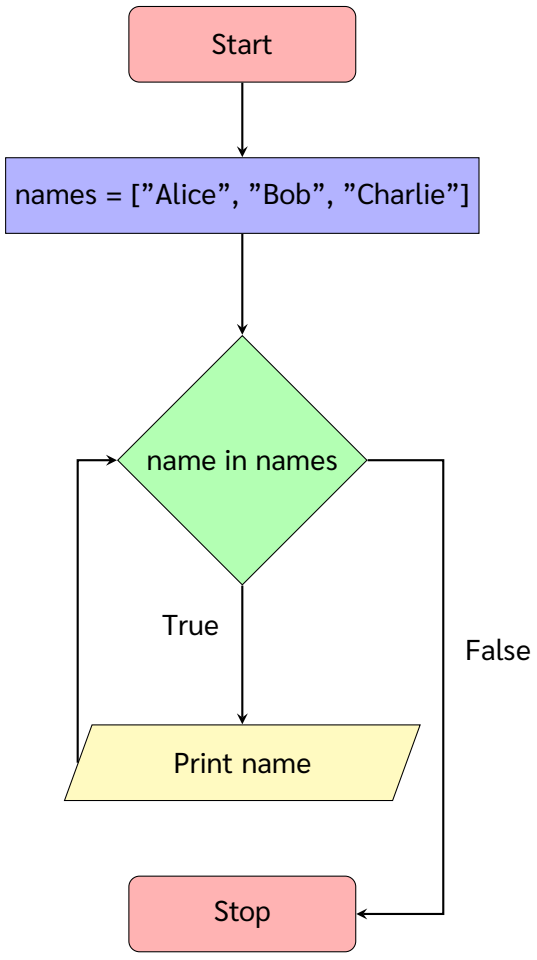
หากต้องการวนซ้ำผ่าน ตัวแปร ลิสต์ (List) หรือทูเพิล (Tuple) โดยตรง สามารถใช้คำสั่ง for ได้โดยไม่ต้องใช้ range เพียงแค่ใช้ Logical Operator อย่าง in ... เพื่อวนซ้ำผ่านสมาชิกของลิสต์หรือทูเพิลนั้น ๆ

(ตัวอย่าง) การวนซ้ำด้วย for ใน Python

```
1  names = ["Alice", "Bob", "Charlie"]
2
3  for name in names:
4      print(name)
```

ผลลัพธ์

Alice
Bob
Charlie



รูปที่ 9: ตัวอย่างโฟลวชาร์ตการวนซ้ำด้วย for กับ ลิสต์ ตรงๆ

4.1.1 การประยุกต์ใช้ร่วมกับ Conditional Statements

การใช้คำสั่ง for ร่วมกับ if ช่วยให้สามารถกรองข้อมูลหรือทำงานกับข้อมูลที่ตรงตามเงื่อนไขได้ เช่น การหาค่าที่ตรงตามเงื่อนไขในลิสต์ ซึ่งมีประโยชน์มาก ในการจัดการชุดข้อมูลที่มีจำนวนมาก **(ตัวอย่างด้านล่าง)**

แนะนำการประยุกต์ใช้เครื่องหมายดำเนินการ!

หากมีตัวแปร X = 10 เมื่อเราใช้ X += 5 จะเท่ากับ X = X + 5 ซึ่งจะทำให้ X มีค่าเป็น 15 นั่นหมายความว่าเครื่องหมาย += เป็นการย่อรูปของการบวกและกำหนดค่าใหม่ให้กับตัวแปร ซึ่งจะนำค่าเดิมของตัวแปรมาบวกกับค่าที่ระบุ และเก็บผลลัพธ์ไว้ในตัวแปรนั้น **(หากตัวแปรนั้นมีค่าเดิมอยู่แล้ว)**

นอกจากการใช้ += ยังมีเครื่องหมายดำเนินการอื่น ๆ ที่สามารถใช้ได้เช่นกัน เช่น

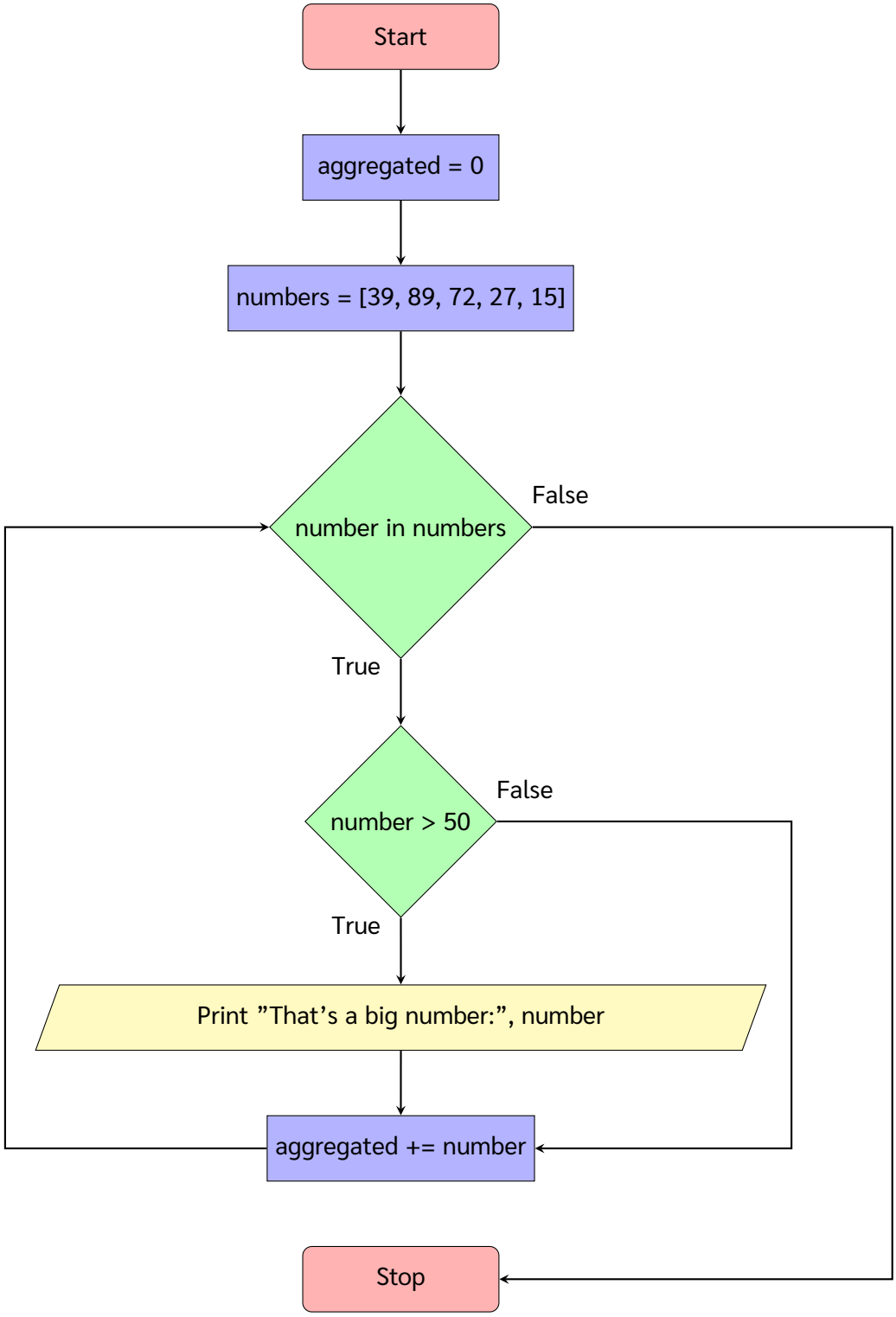
- -= : การลบ (Subtraction)
- *= : การคูณ (Multiplication)
- /= : การหาร (Division) - ผลลัพธ์เป็นทศนิยม
- //= : การหารปัดเศษลง (Floor Division) - ผลลัพธ์เป็นจำนวนเต็ม
- %= : การหาเศษเหลือจากการหาร (Modulus)
- **= : การยกกำลัง (Exponentiation)

(ตัวอย่าง) การวนซ้ำพร้อมประยุกต์ใช้เงื่อนไข ใน Python

```python
aggregated = 0
numbers = [39, 89, 72, 27, 15]

for number in numbers:
    if number > 50:
        print("That's a big number:", number)

    aggregated += number

print("Total sum:", aggregated)
```

ผลลัพธ์

That 's a big number: 89
That 's a big number: 72
Total sum: 241



รูปที่ 10: ตัวอย่างโฟลวชาร์ตการวนซ้ำด้วย for และเงื่อนไข if ใน Python

4.1.2 การวนซ้ำซ้อน (Nested Loops)

การวนซ้ำซ้อน (Nested Loops) คือการใช้คำสั่งวนซ้ำภายในคำสั่งวนซ้ำอีกครั้ง ซึ่งช่วยให้สามารถทำงานกับชุดข้อมูลที่มีโครงสร้างซับซ้อนได้ เช่น การทำงานกับตารางหรือเมทริกซ์ที่มีหลายมิติ หรือ การเรียงตัวเลขจากเล็กไปใหญ่ในลิสต์ เป็นต้น ฯลฯ

แนะนำการใช้งาน Placeholder!

การใช้ Placeholder ใน Python ช่วยให้สามารถจัดรูปแบบข้อความได้อย่างยืดหยุ่น โดยใช้สัญลักษณ์ **%** เพื่อแทนที่ค่าตัวแปรในสตริง เช่น

```
print("%s x %s = %s" % (2, 2, 4))
```

%s คือ Placeholder ที่ใช้แทนค่าตัวแปรในสตริง ซึ่งจะถูกแทนที่ด้วยค่าที่ระบุใน Tuple หลังจาก % เช่น
ในตัวอย่างนี้ **2 x 2 = 4** จะถูกพิมพ์ออกมา (%s อันแรกแทนที่ด้วย 2, อันที่สองแทนที่ด้วย 2, และอันสุดท้ายแทนที่ด้วย 4)

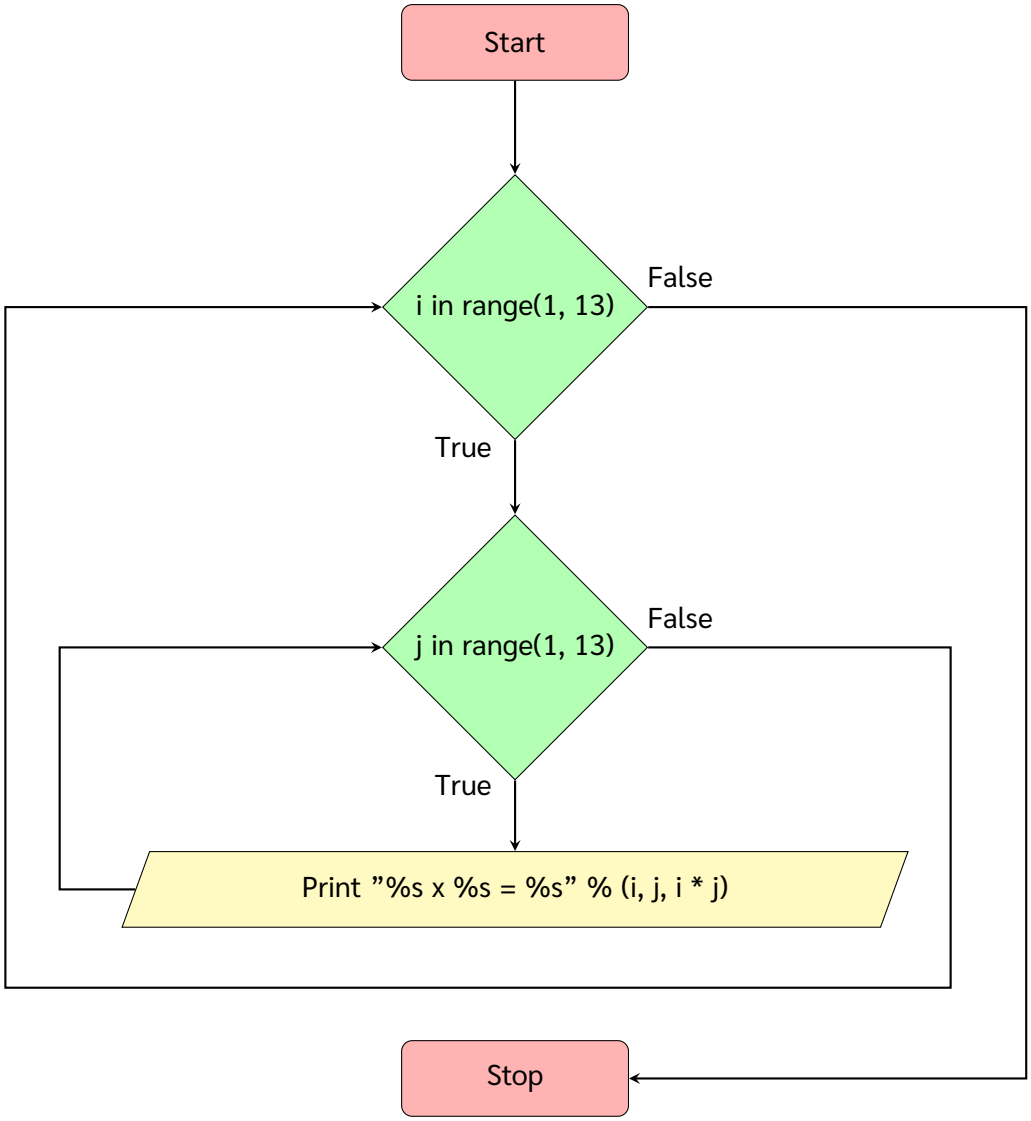
นอกจากนี้ยังสามารถใช้ Placeholder อื่น ๆ เช่น %d สำหรับจำนวนเต็ม (Integer), %f สำหรับจำนวนทศนิยม (Float) เป็นต้น ฯลฯ

(ตัวอย่าง) การวนซ้ำซ้อนเพื่อแสดงตารางสูตรคูณ 1 ถึง 12 ใน Python

```
for i in range(1, 13):
    for j in range(1, 13):
        print("%s x %s = %s" % (i, j, i * j))
```

ผลลัพธ์

```
1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
1 x 5 = 5
1 x 6 = 6
1 x 7 = 7
1 x 8 = 8
1 x 9 = 9
1 x 10 = 10
1 x 11 = 11
1 x 12 = 12
...
12 x 12 = 144
```



รูปที่ 11: ตัวอย่างโฟลวชาร์ตการวนซ้ำด้วย for และเงื่อนไข if ใน Python

4.1.3 แบบฝึกหัดการใช้การวนซ้ำ

แบบฝึกหัดเพื่อฝึกการใช้คำสั่งวนซ้ำในการแก้ปัญหาจริง พร้อมทำ Flowchart และโค้ด Python ตามโจทย์ที่กำหนด

โจทย์ที่ 1: การคัดกรองเลขที่ไม่ซ้ำกัน

จงเขียนโปรแกรมที่วนซ้ำ **ชุดข้อมูลตัวเลข (List of Numbers)** ที่มีการซ้ำกัน เช่น **[1, 2, 3, 4, 5, 1, 2, 3]** และสร้างชุดข้อมูลใหม่ที่มีตัวเลขที่ไม่ซ้ำกัน เช่น **[1, 2, 3, 4, 5]** โดยใช้คำสั่งวนซ้ำ for และเงื่อนไข if เพื่อกรองข้อมูล

โจทย์ที่ 2: หาตำแหน่งของตัวเลข 2 ตัวที่เป็นรากของผลรวม

จงเขียนโปรแกรมที่วนซ้ำ **ชุดข้อมูลตัวเลข (List of Numbers)** และ หาตำแหน่ง (Index) ของตัวเลข 2 ตัวแรก ที่มีผลรวมเท่ากับ 10 เช่น **[1, 2, 3, 7, 8]** จะได้ตำแหน่งของตัวเลข 2 ตัวที่เป็นรากของผลรวมคือ (3, 7) เป็นต้น โดยใช้คำสั่งวนซ้ำ for และเงื่อนไข if