คอมพิวเตอร์โปรแกรมมิ่ง (โครงการพี่ติวน้อง)

เพื่อส่งเสริมการมีส่วนร่วมของรุ่นพี่ในการถ่ายทอดความรู้แก้รุ่นน้อง เพื่อเสริมความเข้าใจในรายวิชา

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

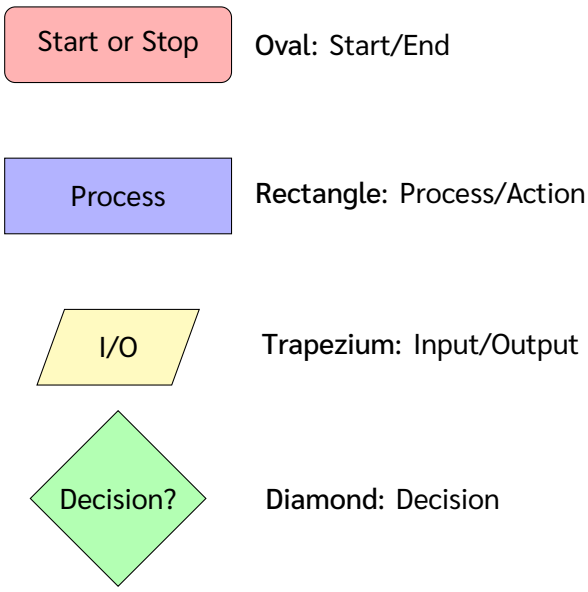
ประจำวันที่ 17 กรกฎาคม พ.ศ. 2568

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)

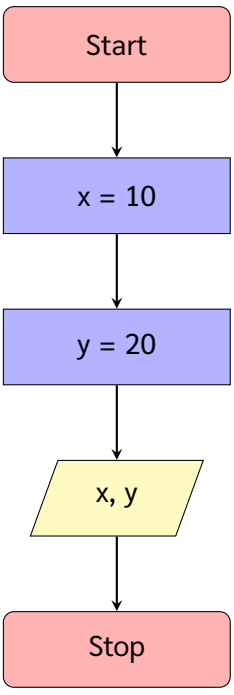
ตัวแปรเป็นชื่อที่ใช้เพื่ออ้างถึงข้อมูลที่เก็บอยู่ในหน่วยความจำของคอมพิวเตอร์ ตัวแปรสามารถเปลี่ยนแปลงค่าได้ในระหว่างการทำงานของโปรแกรม และยังมีตัวแปรที่ไม่สามารถเปลี่ยนแปลงค่าได้ (Immutable) เช่น ค่าคงที่ (Constants) ตัวแปรในภาษาโปรแกรมต่างๆ อาจมีรูปแบบการประกาศที่แตกต่างกัน

(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)

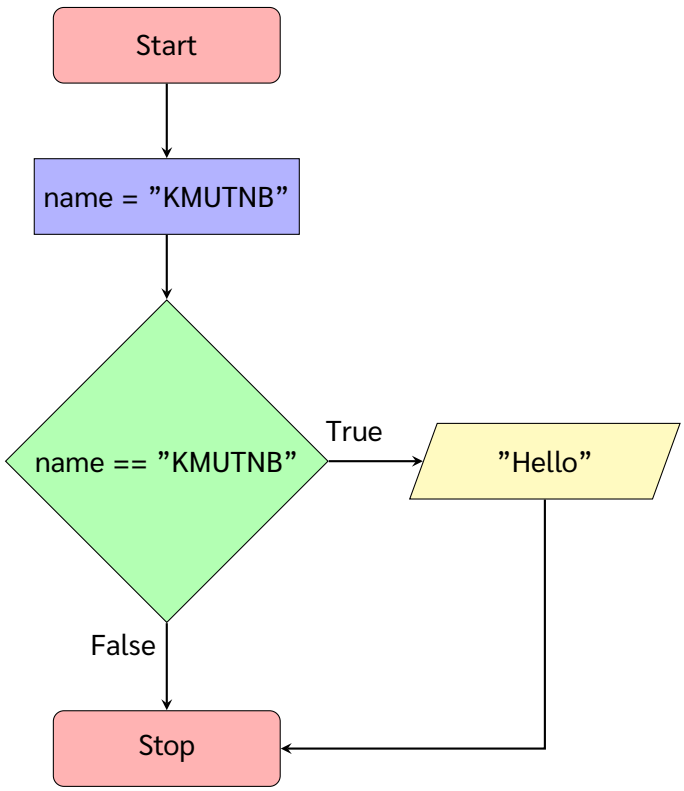
คำสั่งแบบมีเงื่อนไข (Conditional Statements) เป็นคำสั่งที่ใช้ในการตัดสินใจว่าควรทำอะไรต่อไปในโปรแกรมตามเงื่อนไขที่กำหนด คำสั่งเหล่านี้ช่วยให้โปรแกรมสามารถทำงานได้อย่างยืดหยุ่นและตอบสนองต่อสถานการณ์ต่างๆ (โค้ดด้านล่าง)

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

```python
name = "KMUTNB"

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



รูปที่ 3: ตัวอย่างโฟลวชาร์ตเงื่อนไข if

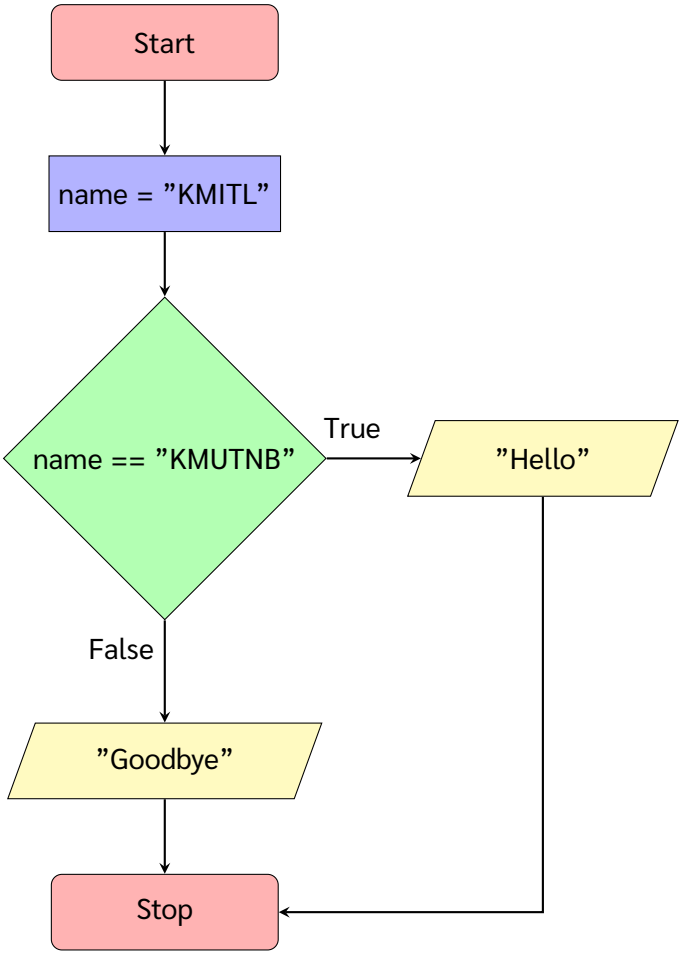
3.2. การใช้คำสั่ง if-else เบื้องต้น

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

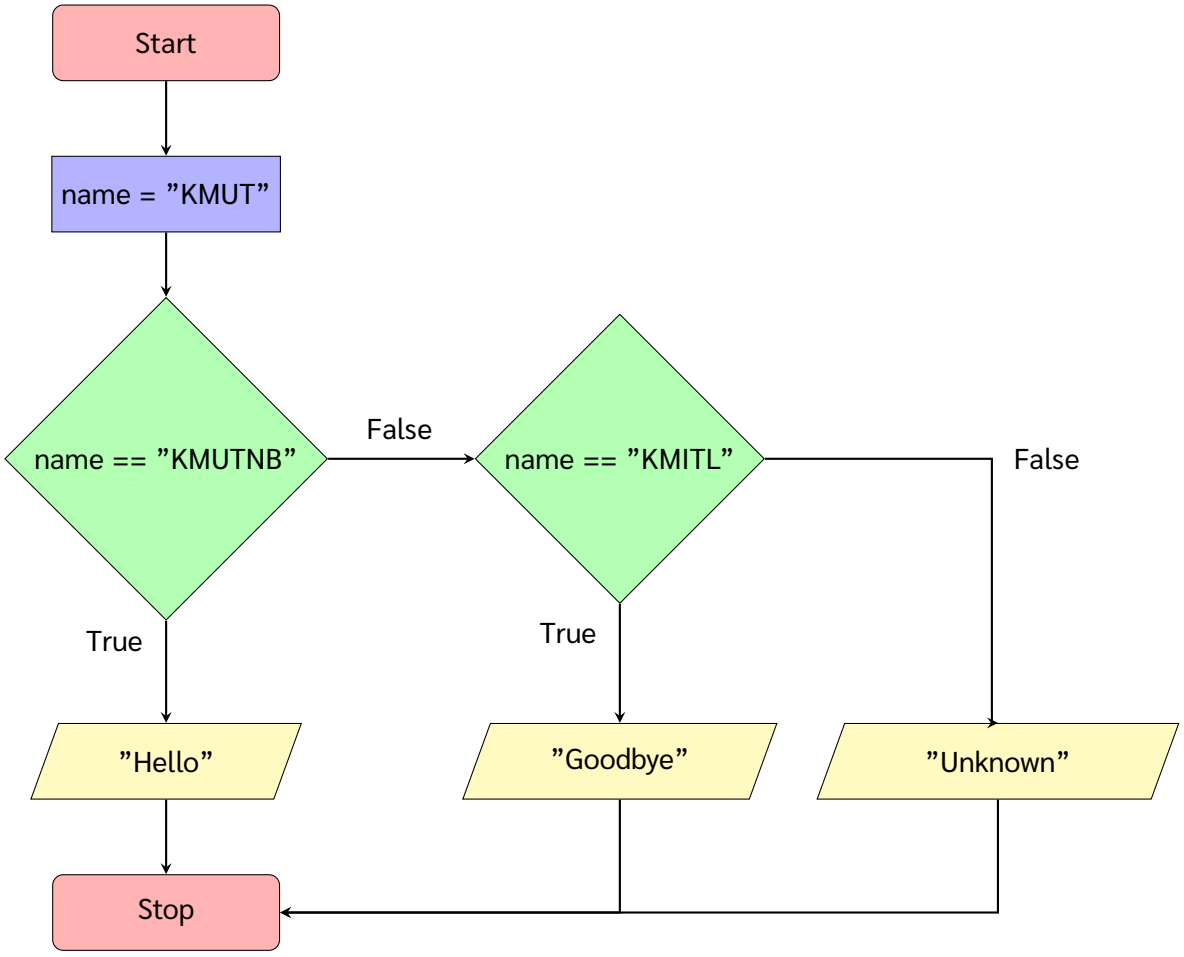
3.3. การใช้คำสั่ง if-elif-else เบื้องต้น

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 = "KMITL"

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
```

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(2):
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

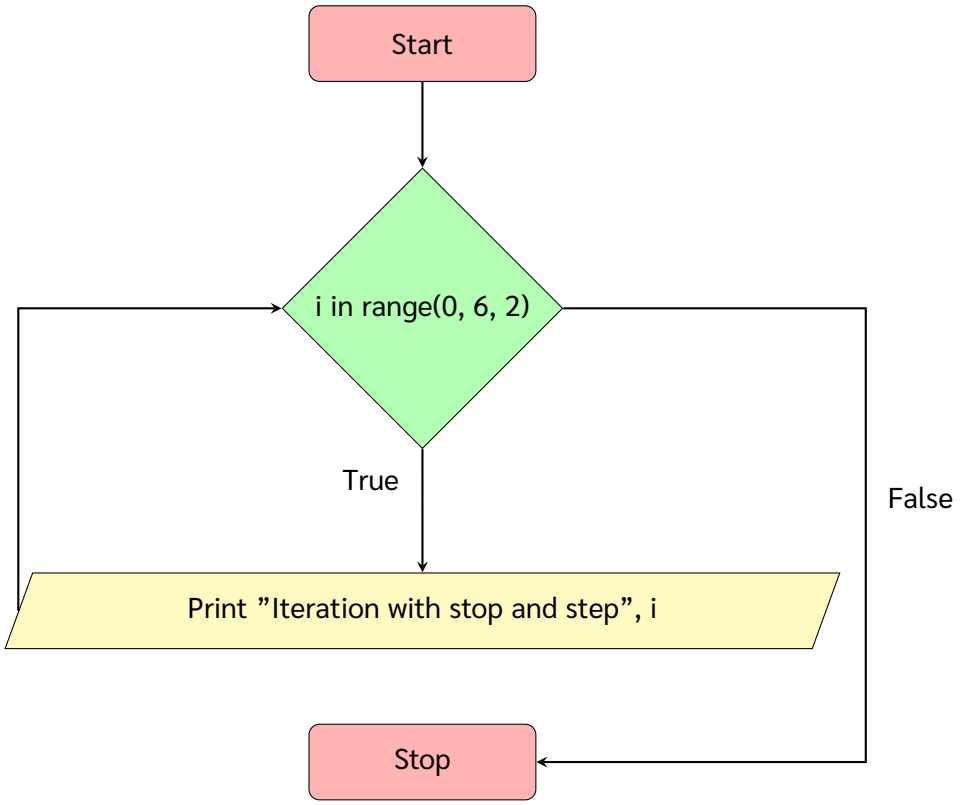
```
Iteration with stop 0
Iteration with stop 1
```

(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

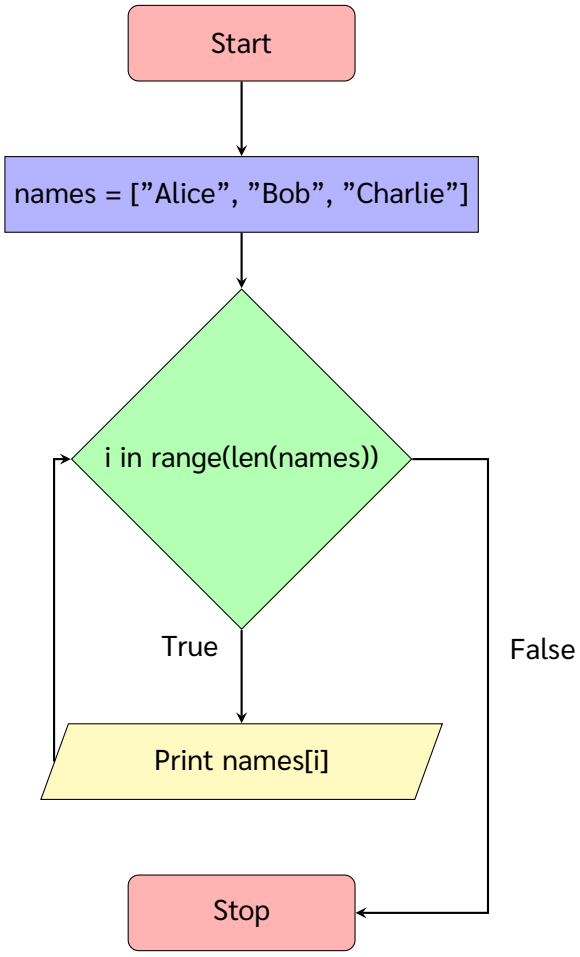
หากเป็น Dictionary จะนับจำนวนคีย์ (Keys) ที่มีอยู่ใน Dictionary นั้น ๆ

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

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

ผลลัพธ์

Alice
Bob
Charlie



รูปที่ 7: ตัวอย่างโฟลวชาร์ตการวนซ้ำด้วย for กับ range(...) และ len(...)

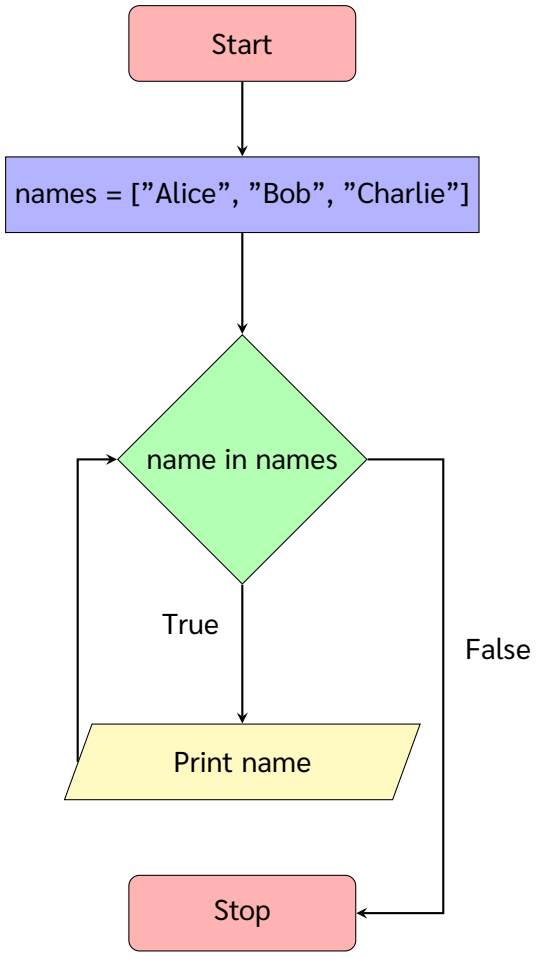
หากต้องการวนซ้ำผ่านรายการ (List) หรือทูเพิล (Tuple) โดยตรง สามารถใช้คำสั่ง for ได้โดยไม่ต้องใช้ range

(ตัวอย่าง) การวนซ้ำด้วย for ใน Python

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

ผลลัพธ์

Alice
Bob
Charlie



รูปที่ 8: ตัวอย่างโฟลวชาร์ตการวนซ้ำด้วย for กับ ลิสต์ ตรงๆ

5. Basic Programming Concepts

5.1. Variables and Data Types

Variables are containers for storing data values. Different programming languages have different data types:

```

# Integer variable
age = 25

# String variable
name = "John Doe"

# Float variable
height = 5.9

# Boolean variable
is_student = True

# List variable
grades = [85, 92, 78, 96, 88]

print(f"Name: {name}, Age: {age}")
print(f"Height: {height} feet")
print(f"Is student: {is_student}")
print(f"Grades: {grades}")
```

5.2. Control Structures

Control structures determine the flow of program execution. The main types are:

5.2.1 Conditional Statements

```
def check_grade(score):
    if score >= 90:
        grade = "A"
    elif score >= 80:
        grade = "B"
    elif score >= 70:
        grade = "C"
    elif score >= 60:
        grade = "D"
    else:
        grade = "F"

    return grade

# Example usage
student_score = 85
result = check_grade(student_score)
print(f"Score: {student_score}, Grade: {result}")
```

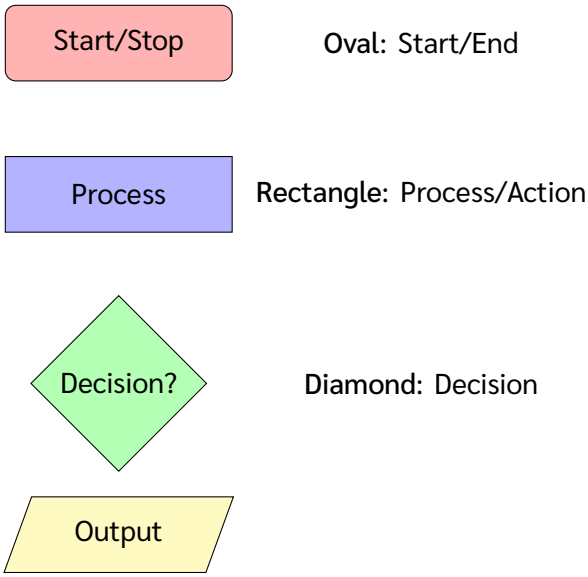
5.2.2 Loops

```
# For loop to calculate factorial
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

# Calculate factorial of 5
number = 5
fact = factorial(number)
print(f"Factorial of {number} is {fact}")

# While loop example
def countdown(start):
    while start > 0:
        print(f"Countdown: {start}")
        start -= 1
    print("Blast off!")

countdown(5)
```

6. Programming Flowcharts

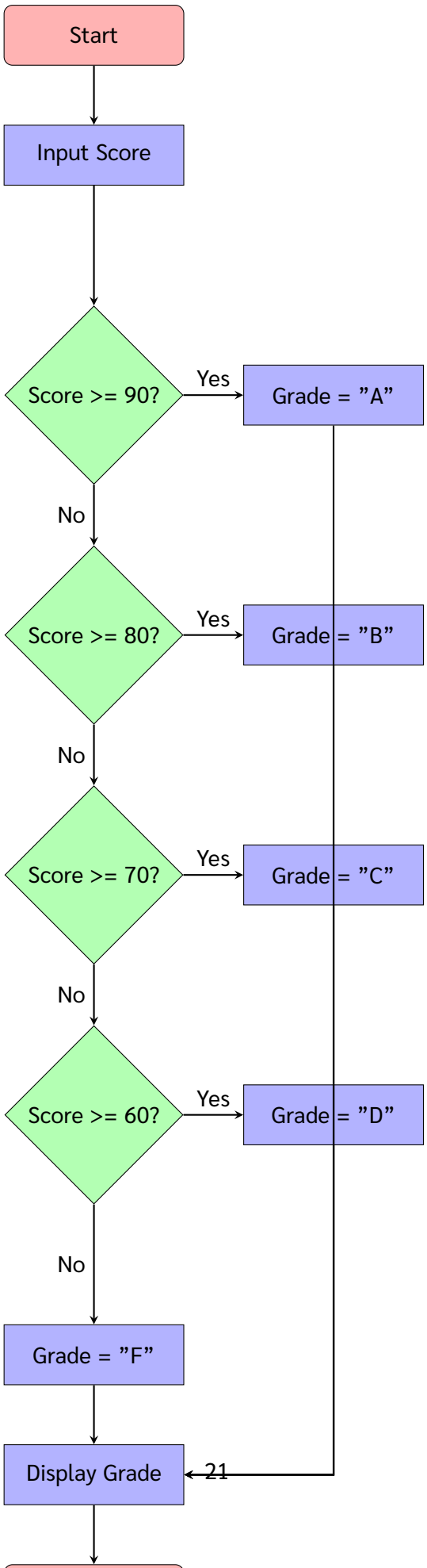
Flowcharts are visual representations of algorithms and processes. They help programmers plan and understand program logic before writing code.

6.1. Basic Flowchart Symbols



รูปที่ 9: Basic Flowchart Symbols

6.2. Grade Calculator Flowchart



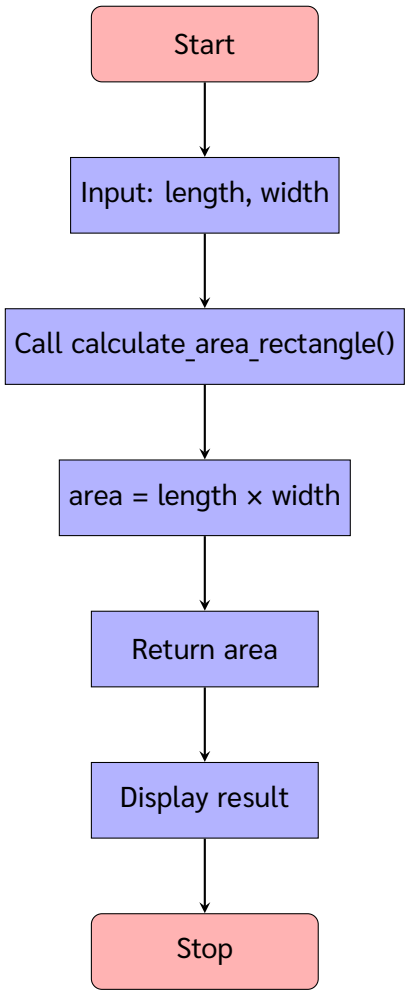
7. Functions and Modular Programming

Functions are reusable blocks of code that perform specific tasks. They help organize code and avoid repetition.

7.1. Function Definition and Usage

```python
def calculate_area_rectangle(length, width):
    """
    Calculate the area of a rectangle

    Args:
        length (float): Length of the rectangle
        width (float): Width of the rectangle

    Returns:
        float: Area of the rectangle
    """
    area = length * width
    return area

def calculate_area_circle(radius):
    """
    Calculate the area of a circle

    Args:
        radius (float): Radius of the circle

    Returns:
        float: Area of the circle
    """
    import math
    area = math.pi * radius ** 2
    return area

# Example usage
rect_area = calculate_area_rectangle(10, 5)
circle_area = calculate_area_circle(3)

print(f"Rectangle area (10x5): {rect_area}")
print(f"Circle area (radius=3): {circle_area:.2f}")
```

7.2. Function Call Flowchart



รูปที่ 11: Function Call Process Flowchart

8. Object-Oriented Programming

Object-Oriented Programming (OOP) is a programming paradigm that uses objects and classes to structure code.

8.1. Classes and Objects

```python
class Student:
    def __init__(self, name, student_id, major):
        self.name = name
        self.student_id = student_id
        self.major = major
        self.grades = []

    def add_grade(self, subject, grade):
        self.grades.append({"subject": subject, "grade": grade})

    def calculate_gpa(self):
        if not self.grades:
            return 0.0

        total_points = sum(grade["grade"] for grade in self.grades)
        return total_points / len(self.grades)

    def display_info(self):
        print(f"Name: {self.name}")
        print(f"Student ID: {self.student_id}")
        print(f"Major: {self.major}")
        print(f"GPA: {self.calculate_gpa():.2f}")

# Create student objects
student1 = Student("Alice Johnson", "S001", "Computer Science")
student2 = Student("Bob Smith", "S002", "Mathematics")

# Add grades
student1.add_grade("Programming", 95)
student1.add_grade("Calculus", 88)
student1.add_grade("Physics", 92)

student2.add_grade("Algebra", 90)
student2.add_grade("Statistics", 85)

# Display information
print("Student 1:")
student1.display_info()
print("\nStudent 2:")
student2.display_info()
```

9. Data Structures in Python

Python provides several built-in data structures that are essential for programming.

9.1. Lists and List Operations

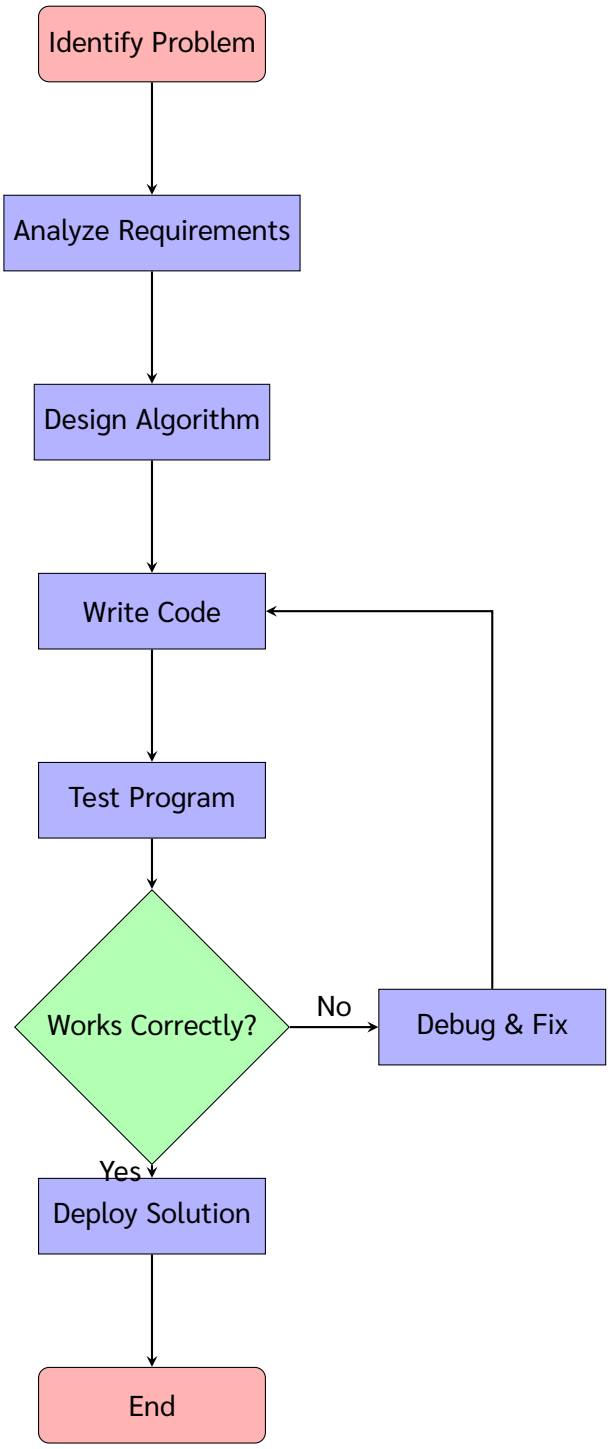
```python
# Creating and manipulating lists
numbers = [1, 2, 3, 4, 5]
fruits = ["apple", "banana", "orange", "grape"]

# List operations
numbers.append(6)          # Add element to end
numbers.insert(0, 0)       # Insert at specific position
numbers.remove(3)          # Remove specific value
popped = numbers.pop()     # Remove and return last element

# List comprehension
squares = [x**2 for x in range(1, 6)]
even_numbers = [x for x in range(20) if x % 2 == 0]

# Sorting and searching
fruits.sort()              # Sort in place
sorted_numbers = sorted(numbers, reverse=True)

print(f"Numbers: {numbers}")
print(f"Fruits: {fruits}")
print(f"Squares: {squares}")
print(f"Even numbers: {even_numbers}")
```

9.2. Dictionaries and Data Management

```python
# Creating a student management system
student_database = {}

def add_student(student_id, name, age, major):
    student_database[student_id] = {
        "name": name,
        "age": age,
        "major": major,
        "courses": [],
        "gpa": 0.0
    }

def enroll_course(student_id, course_name, grade):
    if student_id in student_database:
        student_database[student_id]["courses"].append({
            "course": course_name,
            "grade": grade
        })
        # Recalculate GPA
        courses = student_database[student_id]["courses"]
        total_grade = sum(course["grade"] for course in courses)
        student_database[student_id]["gpa"] = total_grade / len(courses)

# Example usage
add_student("S001", "Alice Johnson", 20, "Computer Science")
add_student("S002", "Bob Smith", 19, "Mathematics")

enroll_course("S001", "Python Programming", 95)
enroll_course("S001", "Data Structures", 88)
enroll_course("S002", "Calculus", 92)

# Display student information
for student_id, info in student_database.items():
    print(f"Student ID: {student_id}")
    print(f"Name: {info['name']}, Major: {info['major']}")
    print(f"GPA: {info['gpa']:.2f}")
    print("---")
```

10. Algorithm Design Process

10.1. Problem-Solving Flowchart



รูปที่ 12: Software Development Process

11. Conclusion

Programming is a powerful skill that enables us to solve complex problems and create innovative solutions. The key concepts covered in this lecture include:

1. **Variables and Data Types**: The building blocks of programs
2. **Control Structures**: Logic flow control with conditions and loops
3. **Functions**: Modular, reusable code blocks
4. **Object-Oriented Programming**: Organizing code with classes and objects
5. **Algorithm Design**: Systematic problem-solving approach

Practice these concepts regularly, and remember that programming is both an art and a science that improves with experience and continuous learning.

End of Lecture