

# คอมพิวเตอร์โปรแกรมมิ่ง (โครงการพีทีวน้อง)

เพื่อส่งเสริมการมีส่วนร่วมของรุ่นพี่ในการถ่ายทอดความรู้แก่รุ่นน้อง เพื่อเสริมความเข้าใจในรายวิชา

\*ยงยุทธ ชวนขุนทด, เมธัส ทองจันทร์, ศุภกร จิรศิริวรกุล, มัชฌิมา ประยูรณรัตน์

ประจำวันที่ 17 กรกฎาคม พ.ศ. 2568

ภาควิชาเทคโนโลยีสารสนเทศ

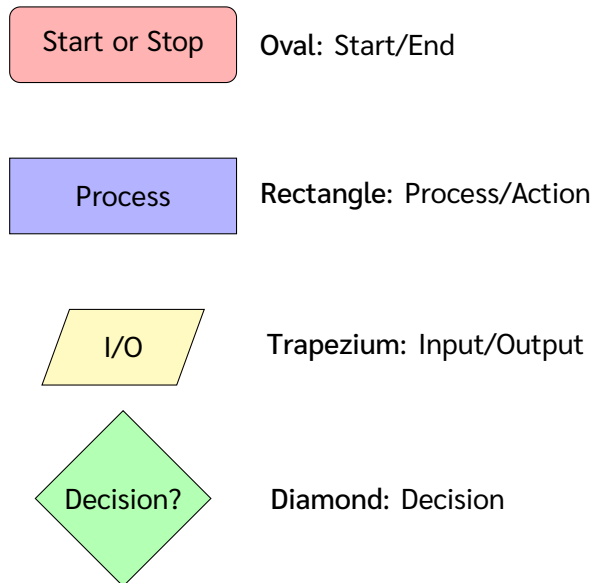
มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ วิทยาเขตปทุมธานี

# สารบัญ

1. ชนิดโฟลวชาร์ต (Flowchart Types)	3
2. ตัวแปรและชนิดข้อมูล (Variables and Data Types)	4
2.1. ตัวแปร (Variables)	4
2.2. ชนิดข้อมูล (Data Types)	5
3. คำสั่งแบบมีเงื่อนไข (Conditional Statements)	6
3.1. การใช้คำสั่ง if เบื้องต้น	6
3.2. การใช้คำสั่ง if-else เบื้องต้น	7
3.3. การใช้คำสั่ง if-elif-else เบื้องต้น	8
3.4. บททดสอบ	9
3.4.1 แบบฝึกหัดการใช้เงื่อนไข	10
4. การวนซ้ำ (Iteration)	11
4.1. การวนซ้ำด้วย for	13
4.1.1 การประยุกต์ใช้ร่วมกับ Conditional Statements	18
5. Basic Programming Concepts	19
5.1. Variables and Data Types	19
5.2. Control Structures	19
5.2.1 Conditional Statements	20
5.2.2 Loops	20
6. Programming Flowcharts	21
6.1. Basic Flowchart Symbols	21
6.2. Grade Calculator Flowchart	23
7. Functions and Modular Programming	24
7.1. Function Definition and Usage	24
7.2. Function Call Flowchart	25

8. Object-Oriented Programming	25
8.1. Classes and Objects . . . . .	26
9. Data Structures in Python	27
9.1. Lists and List Operations . . . . .	27
9.2. Dictionaries and Data Management . . . . .	28
10. Algorithm Design Process	29
10.1. Problem-Solving Flowchart . . . . .	29
11. Conclusion	30

## 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 ตัวแปรใช้เพื่อเก็บข้อมูลที่สามารถเปลี่ยนแปลง (Mutable) และ ไม่สามารถเปลี่ยนแปลง (Immutable) ได้ในระหว่างการทำงานของโปรแกรม และชนิดข้อมูลกำหนดประเภทของข้อมูลที่ตัวแปรนั้นสามารถเก็บได้

### 2.1. ตัวแปร (Variables)

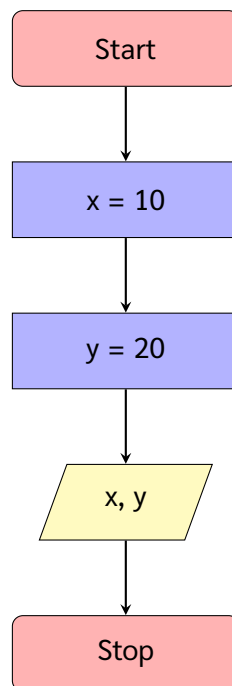
ตัวแปรเป็นชื่อที่ใช้เพื่ออ้างถึงข้อมูลที่เก็บอยู่ในหน่วยความจำของคอมพิวเตอร์ ตัวแปรสามารถเปลี่ยนแปลงค่าได้ในระหว่างการทำงานของโปรแกรม และยังมีตัวแปรที่ไม่สามารถเปลี่ยนแปลงค่าได้ (Immutable) เช่น ค่าคงที่ (Constants) ตัวแปรในภาษาโปรแกรมต่างๆ อาจมีรูปแบบการประกาศที่แตกต่างกัน

(ตัวอย่าง) ตัวแปรใน Python

```
1 x = 10
2 y = 20
3
4 print(x, y)
```

ผลลัพธ์

10 20



รูปที่ 2: ตัวอย่าง Flowchart ของการประกาศตัวแปร

## 2.2. ชนิดข้อมูล (Data Types)

ชนิดข้อมูล (Data Types) เป็นการกำหนดประเภทของข้อมูลในตัวแปรนั้นสามารถเก็บได้ 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

```
1 # Integer
2 x = 10
3
4 # Float
5 y = 3.14
6
7 # String
8 name = "First"
9
10 # Boolean
11 is_active = True
12
13 # List
14 scores = [85, 90, 78, 92]
15
16 # Tuple
17 coordinates = (10.5, 20.3)
18
19 # Dictionary
20 person = {"name": "John", "age": 30}
21
22 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)

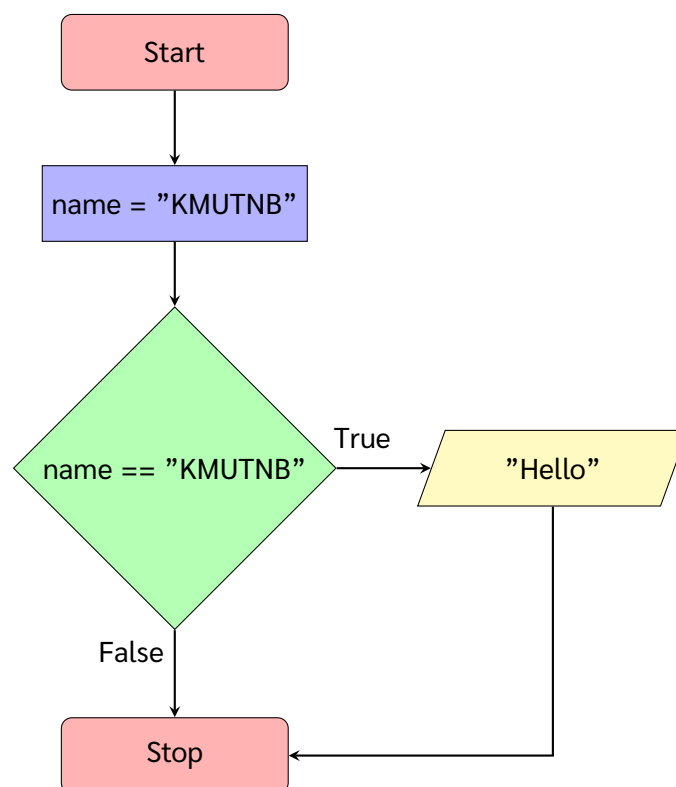
คำสั่งแบบมีเงื่อนไข (Conditional Statements) เป็นคำสั่งที่ใช้ในการตัดสินใจว่าควรทำอะไรต่อไปในโปรแกรมตามเงื่อนไขที่กำหนด คำสั่งเหล่านี้ช่วยให้โปรแกรมสามารถทำงานได้อย่างยืดหยุ่นและตอบสนองต่อสถานการณ์ต่างๆ (โค้ดด้านล่าง)

#### 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")
```



รูปที่ 3: ตัวอย่างโฟลวชาร์ตเงื่อนไข if

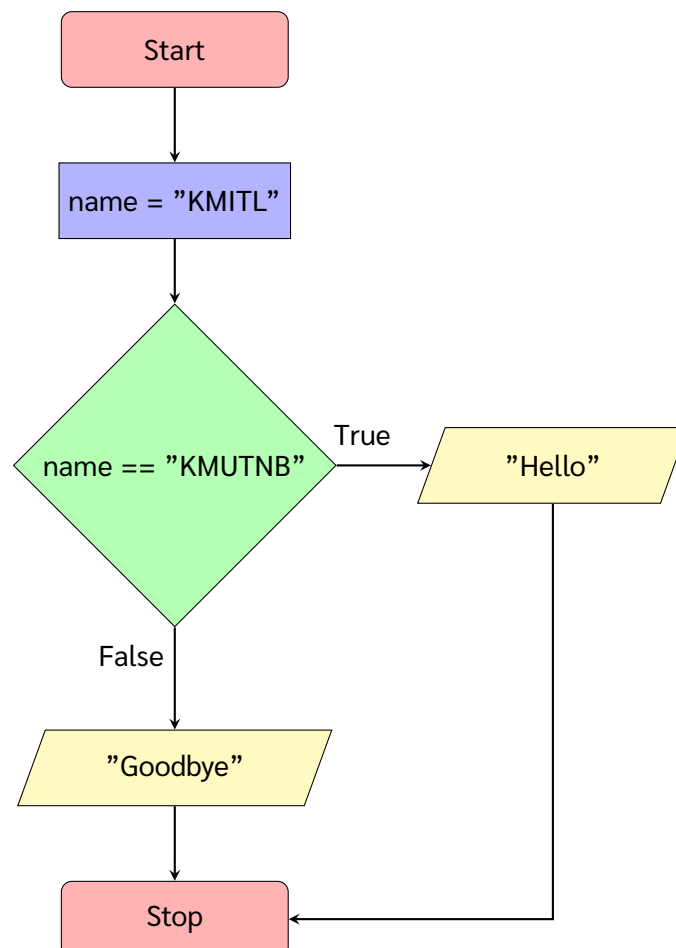
### 3.2. การใช้คำสั่ง if-else เบื้องต้น

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

สิ่งที่ต้องการ: ฉันต้องการให้แสดงข้อความ 'Hello' หากตัวแปร 'name' มีค่าเป็น 'KMUTNB' และแสดงข้อความ 'Goodbye' หากไม่ตรงกับเงื่อนไขใดๆ

(ตัวอย่าง) การใช้ if-else ใน Python

```
1 name = "KMITL"
2
3 if name == "KMUTNB":
4     print("Hello")
5 else:
6     print("Goodbye")
```



รูปที่ 4: ตัวอย่างโฟลวชาร์ตเงื่อนไข if-else



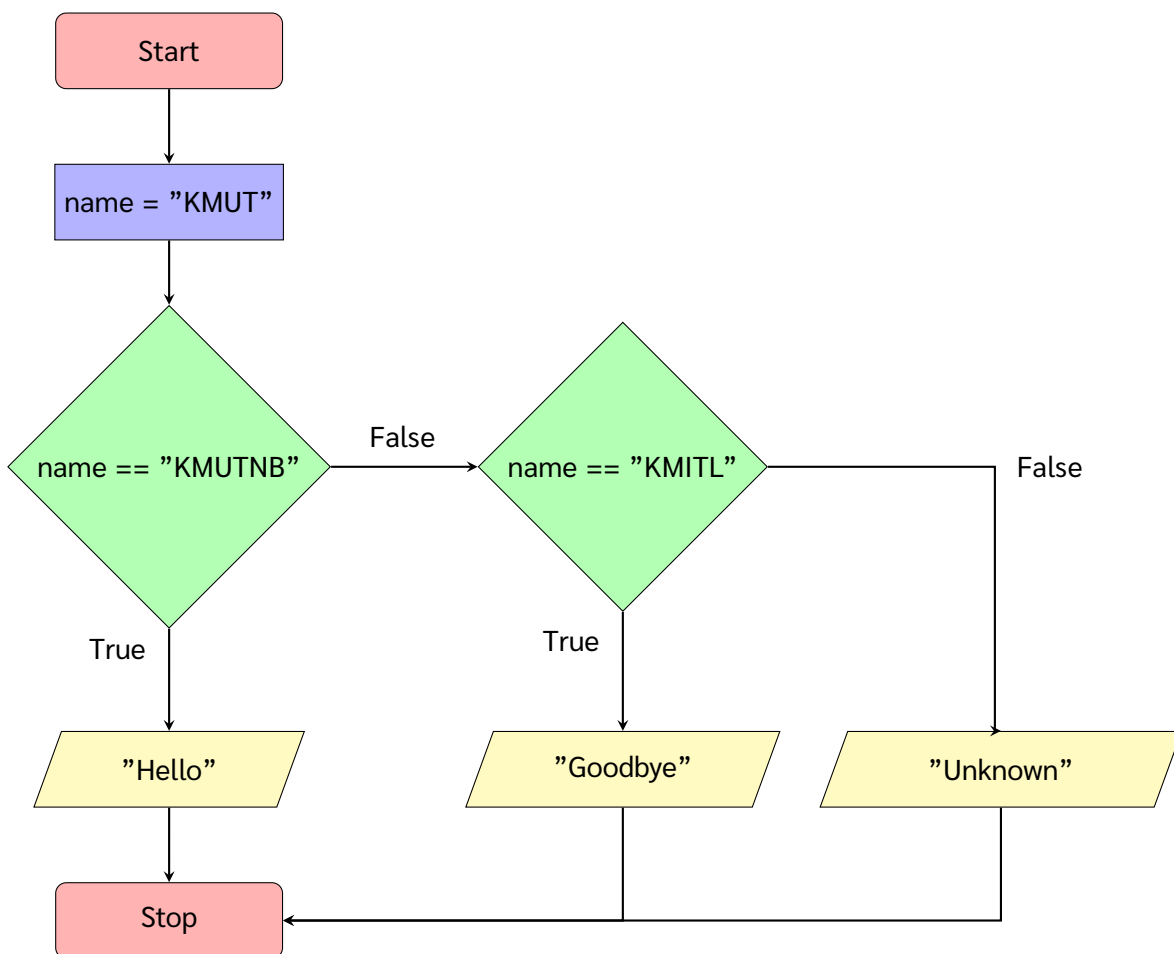
### 3.3. การใช้คำสั่ง if-elif-else เบื้องต้น

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

```
1 name = "KMITL"
2
3 if name == "KMUTNB":
4     print("Hello")
5 elif name == "KMITL":
6     print("Goodbye")
7 else:
8     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

```
1 a = 10
2 b = 3
3
4 print("a + b =", a + b)
5 print("a - b =", a - b)
6 print("a * b =", a * b)
7 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

```
1 number = 10
2
3 if number / 2 == 5:
4     print(f"{number}, Wow!")
5 else:
6     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** และ **โค้ด 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

```
1 age = 20
2 score = 85
3
4 # Comparison Operators
5 print("age > 18:", age > 18)           # True
6 print("score <= 90:", score <= 90)     # True
7 print("age != 25:", age != 25)         # True
8 print("age == 20:", age == 20)         # True
9 print("score >= 80:", score >= 80)     # True
10 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

```
1 age = 20
2 score = 85
3 name = "Alice"
4 subjects = ["Math", "Science", "English"]
5
6 # Logical Operators
7 print("age > 18 and score >= 80:", age > 18 and score >= 80) # True
8 print("age < 18 or score > 90:", age < 18 or score > 90)      # False
9 print("not (age < 18):", not (age < 18))                      # True
10
11 # In Operator
12 print("'Math' in subjects:", "Math" in subjects)              # True
13 print("'Art' in subjects:", "Art" in subjects)                # False
14 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

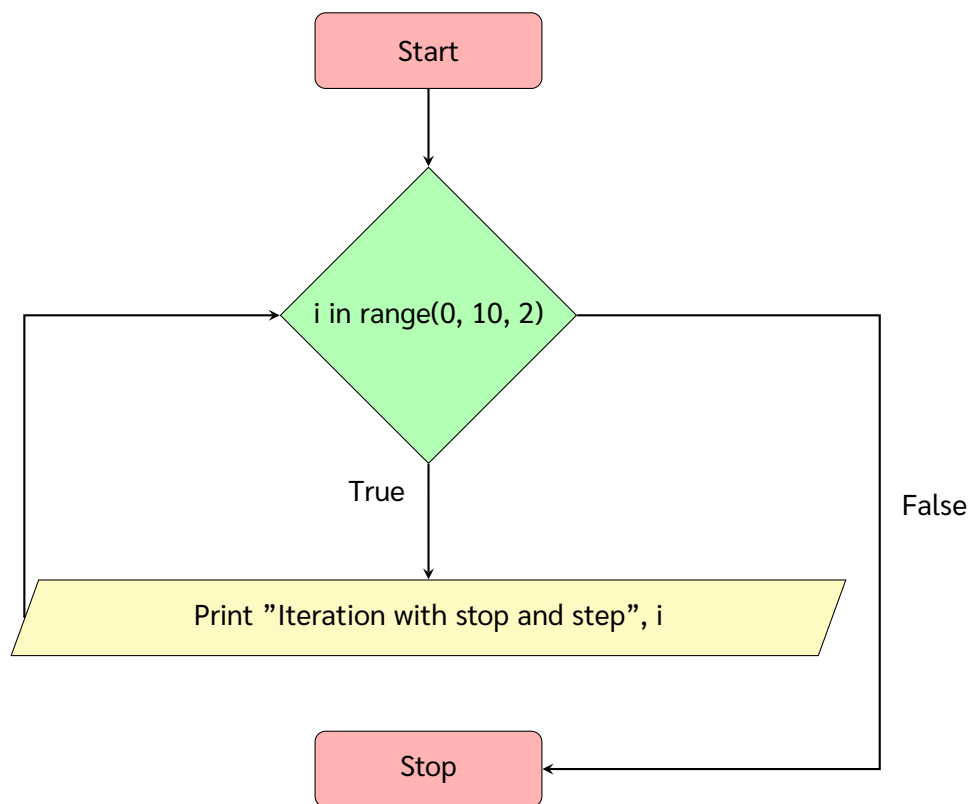
Iteration with stop 0  
Iteration with stop 1

(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 ดิกชันนารี (Dictionary) เป็นต้น

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

หากเป็น Dictionary จะนับจำนวนคีย์ (Keys) ที่มีอยู่ใน Dictionary นั้น ๆ

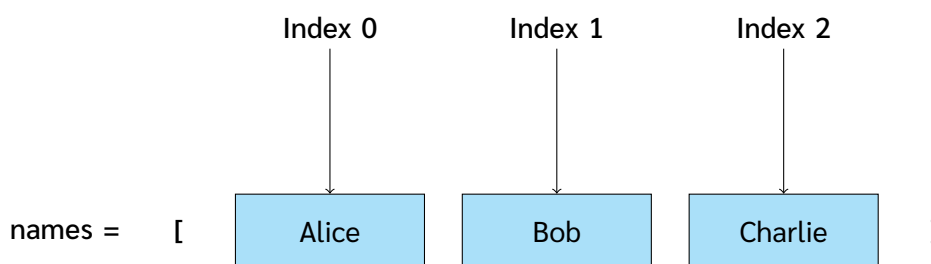
### (ตัวอย่าง) การใช้ฟังก์ชัน `len(...)` ใน Python

```
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



`len(names) = 3`

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

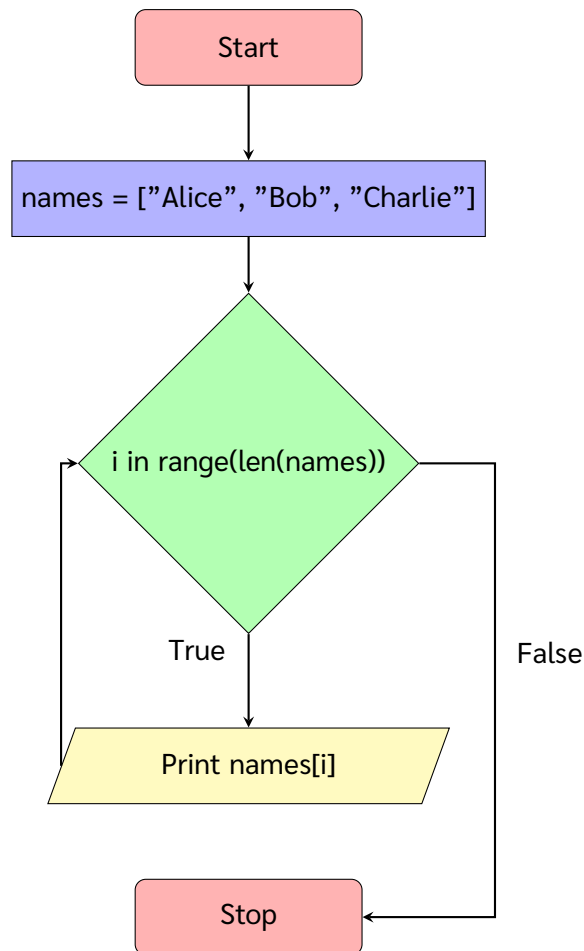


(ตัวอย่าง) การวนซ้ำด้วย range(...) กับ len(...) ใน Python

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

ผลลัพธ์

Alice  
Bob  
Charlie



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

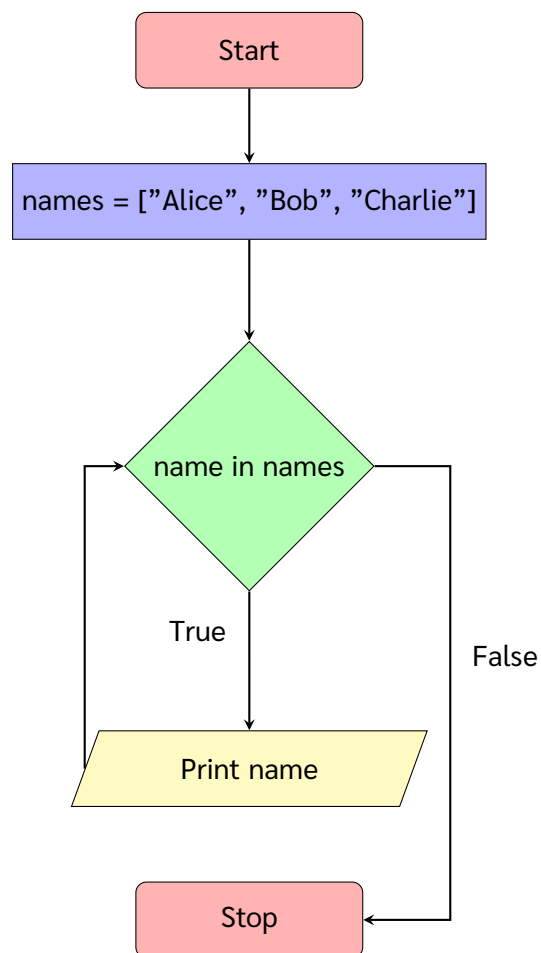
หากต้องการวนซ้ำผ่าน ตัวแปร ลิสต์ (List) หรือทูเพิล (Tuple) โดยตรง สามารถใช้คำสั่ง **for** ได้โดยไม่ต้องใช้ range เพียงแค่ใช้ Logical Operator อย่าง **in ...** เพื่อวนซ้ำผ่านสมาชิกของลิสต์หรือทูเพิลนั้น ๆ

(ตัวอย่าง) การวนซ้ำด้วย for ใน Python

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

ผลลัพธ์

Alice  
Bob  
Charlie



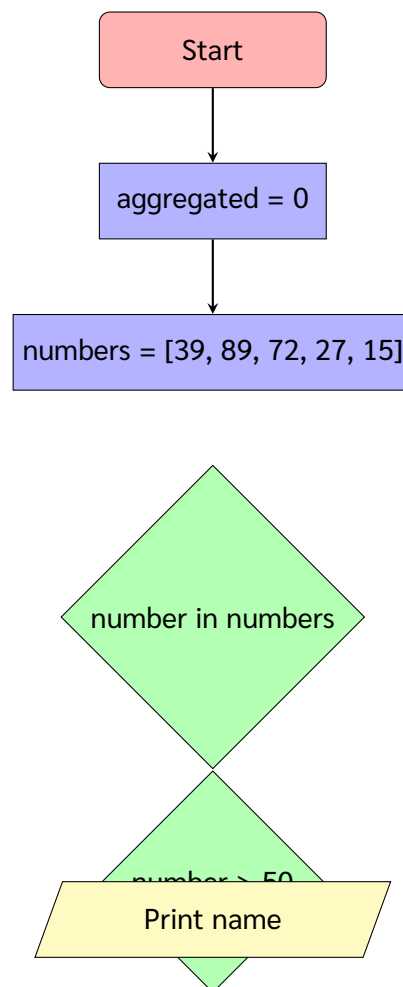
รูปที่ 9: ตัวอย่างโฟลวชาร์ตการวนซ้ำด้วย for กับ ลิสต์ ธรรมดา

#### 4.1.1 การประยุกต์ใช้ร่วมกับ Conditional Statements

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

(ตัวอย่าง) การวนซ้ำพร้อมประยุกต์ใช้เงื่อนไข ใน Python

```
1 aggregated = 0
2 numbers = [39, 89, 72, 27, 15]
3
4 for number in numbers:
5     if number > 50:
6         print("That's a big number:", number)
7
8     aggregated += number
9
10 print("Total sum:", aggregated)
```



รูปที่ 10: ตัวอย่างฟลวชาร์ตการวนซ้ำด้วย 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:

```
1
2 # Integer variable
3 age = 25
4
5 # String variable
6 name = "John Doe"
7
8 # Float variable
9 height = 5.9
10
11 # Boolean variable
12 is_student = True
13
14 # List variable
15 grades = [85, 92, 78, 96, 88]
16
17 print(f"Name: {name}, Age: {age}")
18 print(f"Height: {height} feet")
19 print(f"Is student: {is_student}")
20 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

```
1 def check_grade(score):
2     if score >= 90:
3         grade = "A"
4     elif score >= 80:
5         grade = "B"
6     elif score >= 70:
7         grade = "C"
8     elif score >= 60:
9         grade = "D"
10    else:
11        grade = "F"
12
13    return grade
14
15 # Example usage
16 student_score = 85
17 result = check_grade(student_score)
18 print(f"Score: {student_score}, Grade: {result}")
```

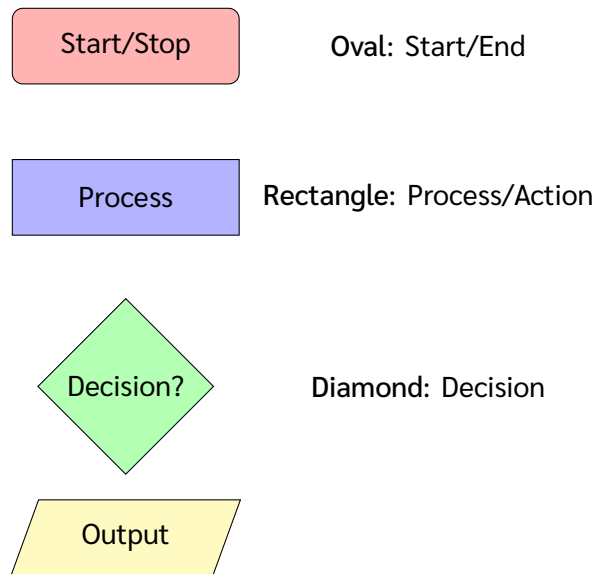
### 5.2.2 Loops

```
1 # For loop to calculate factorial
2 def factorial(n):
3     result = 1
4     for i in range(1, n + 1):
5         result *= i
6     return result
7
8 # Calculate factorial of 5
9 number = 5
10 fact = factorial(number)
11 print(f"Factorial of {number} is {fact}")
12
13 # While loop example
14 def countdown(start):
15     while start > 0:
16         print(f"Countdown: {start}")
17         start -= 1
18     print("Blast off!")
19
20 countdown(5)
```

## 6. Programming Flowcharts

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

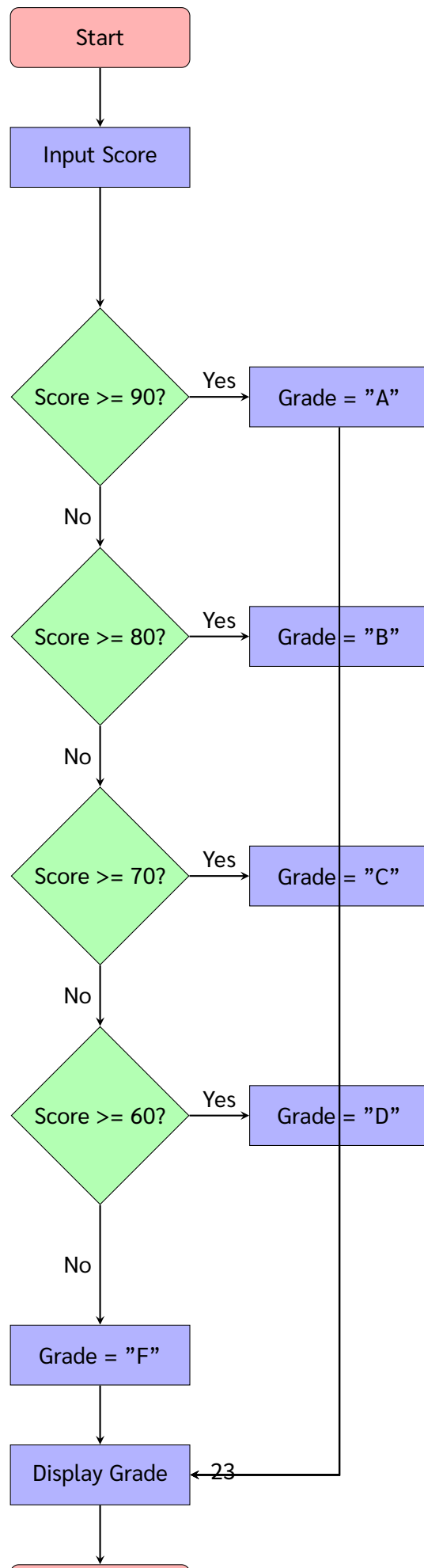
### 6.1. Basic Flowchart Symbols



รูปที่ 11: Basic Flowchart Symbols



## 6.2. Grade Calculator Flowchart





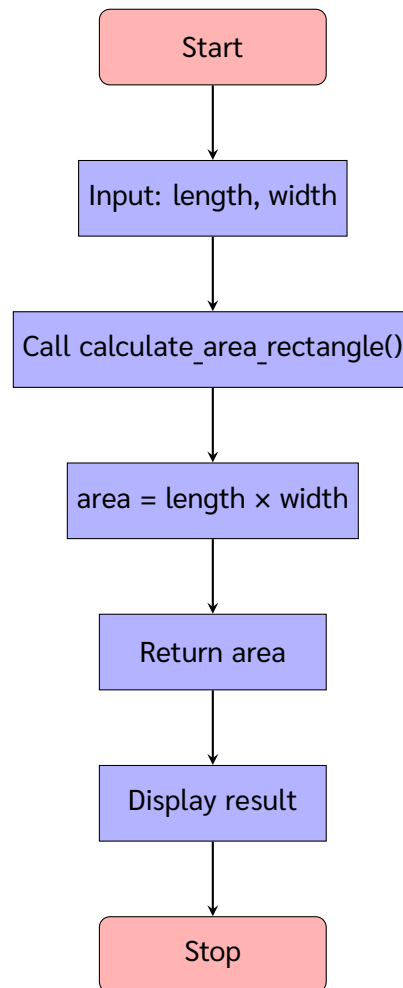
## 7. Functions and Modular Programming

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

### 7.1. Function Definition and Usage

```
1  def calculate_area_rectangle(length, width):
2      """
3      Calculate the area of a rectangle
4
5      Args:
6          length (float): Length of the rectangle
7          width (float): Width of the rectangle
8
9      Returns:
10         float: Area of the rectangle
11     """
12     area = length * width
13     return area
14
15 def calculate_area_circle(radius):
16     """
17     Calculate the area of a circle
18
19     Args:
20         radius (float): Radius of the circle
21
22     Returns:
23         float: Area of the circle
24     """
25     import math
26     area = math.pi * radius ** 2
27     return area
28
29 # Example usage
30 rect_area = calculate_area_rectangle(10, 5)
31 circle_area = calculate_area_circle(3)
32
33 print(f"Rectangle area (10x5): {rect_area}")
34 print(f"Circle area (radius=3): {circle_area:.2f}")
```

## 7.2. Function Call Flowchart



รูปที่ 13: 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

```
1  class Student:
2      def __init__(self, name, student_id, major):
3          self.name = name
4          self.student_id = student_id
5          self.major = major
6          self.grades = []
7
8      def add_grade(self, subject, grade):
9          self.grades.append({"subject": subject, "grade": grade})
10
11     def calculate_gpa(self):
12         if not self.grades:
13             return 0.0
14
15         total_points = sum(grade["grade"] for grade in self.grades)
16         return total_points / len(self.grades)
17
18     def display_info(self):
19         print(f"Name: {self.name}")
20         print(f"Student ID: {self.student_id}")
21         print(f"Major: {self.major}")
22         print(f"GPA: {self.calculate_gpa():.2f}")
23
24 # Create student objects
25 student1 = Student("Alice Johnson", "S001", "Computer Science")
26 student2 = Student("Bob Smith", "S002", "Mathematics")
27
28 # Add grades
29 student1.add_grade("Programming", 95)
30 student1.add_grade("Calculus", 88)
31 student1.add_grade("Physics", 92)
32
33 student2.add_grade("Algebra", 90)
34 student2.add_grade("Statistics", 85)
35
36 # Display information
37 print("Student 1:")
38 student1.display_info()
39 print("\nStudent 2:")
40 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

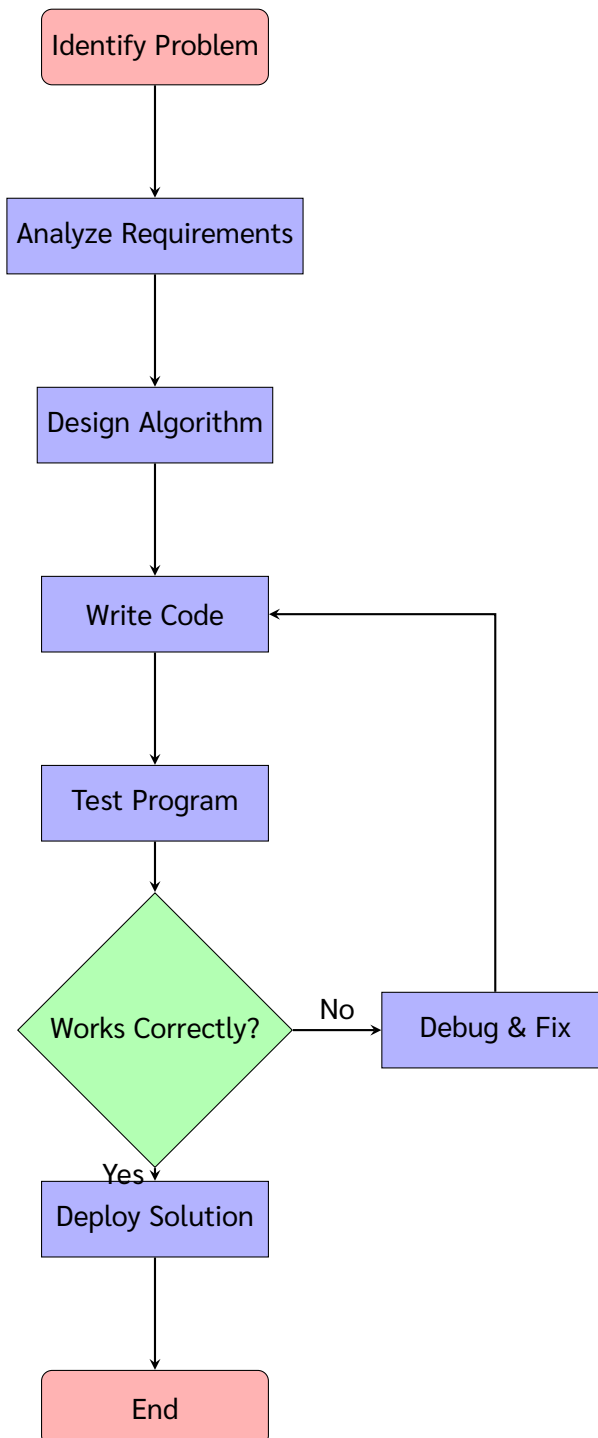
```
1  # Creating and manipulating lists
2  numbers = [1, 2, 3, 4, 5]
3  fruits = ["apple", "banana", "orange", "grape"]
4
5  # List operations
6  numbers.append(6)           # Add element to end
7  numbers.insert(0, 0)       # Insert at specific position
8  numbers.remove(3)          # Remove specific value
9  popped = numbers.pop()     # Remove and return last element
10
11 # List comprehension
12 squares = [x**2 for x in range(1, 6)]
13 even_numbers = [x for x in range(20) if x % 2 == 0]
14
15 # Sorting and searching
16 fruits.sort()              # Sort in place
17 sorted_numbers = sorted(numbers, reverse=True)
18
19 print(f"Numbers: {numbers}")
20 print(f"Fruits: {fruits}")
21 print(f"Squares: {squares}")
22 print(f"Even numbers: {even_numbers}")
```

## 9.2. Dictionaries and Data Management

```
1  # Creating a student management system
2  student_database = {}
3
4  def add_student(student_id, name, age, major):
5      student_database[student_id] = {
6          "name": name,
7          "age": age,
8          "major": major,
9          "courses": [],
10         "gpa": 0.0
11     }
12
13 def enroll_course(student_id, course_name, grade):
14     if student_id in student_database:
15         student_database[student_id]["courses"].append({
16             "course": course_name,
17             "grade": grade
18         })
19         # Recalculate GPA
20         courses = student_database[student_id]["courses"]
21         total_grade = sum(course["grade"] for course in courses)
22         student_database[student_id]["gpa"] = total_grade / len(courses)
23
24 # Example usage
25 add_student("S001", "Alice Johnson", 20, "Computer Science")
26 add_student("S002", "Bob Smith", 19, "Mathematics")
27
28 enroll_course("S001", "Python Programming", 95)
29 enroll_course("S001", "Data Structures", 88)
30 enroll_course("S002", "Calculus", 92)
31
32 # Display student information
33 for student_id, info in student_database.items():
34     print(f"Student ID: {student_id}")
35     print(f"Name: {info['name']}, Major: {info['major']}")
36     print(f"GPA: {info['gpa']:.2f}")
37     print("----")
```

## 10. Algorithm Design Process

### 10.1. Problem-Solving Flowchart



รูปที่ 14: 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*