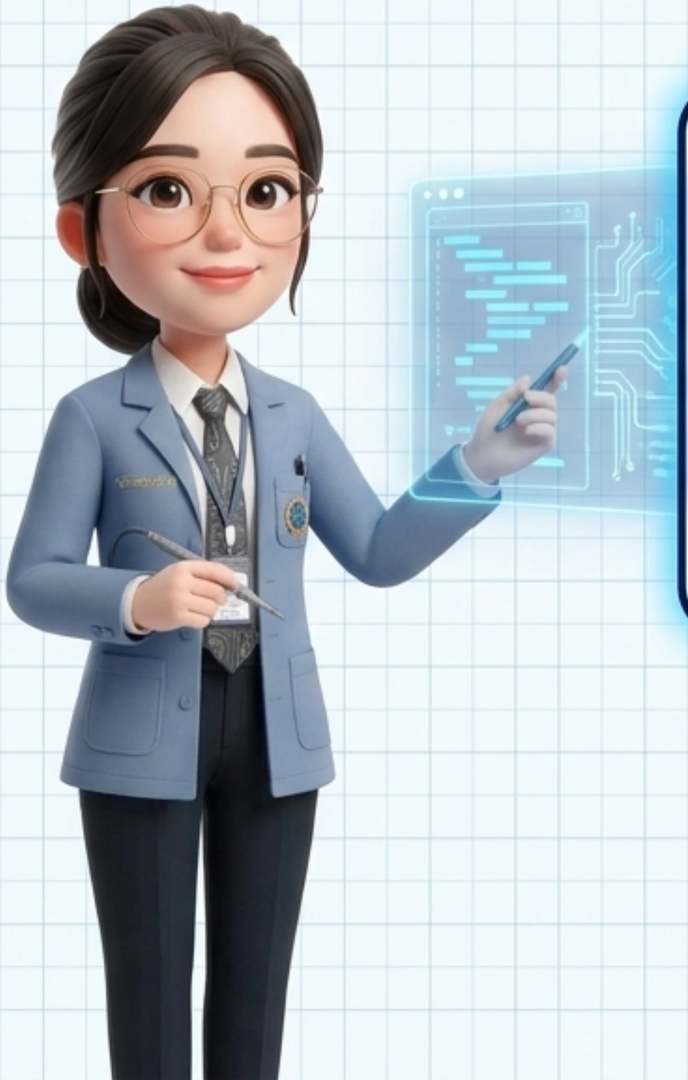
# ความรู้พื้นฐานในการเขียนโปรแกรม

ก้าวแรกสู่การสื่อสารและสั่งการคอมพิวเตอร์อย่างเป็นระบบ

# การเขียนโปรแกรมคืออะไร?



แนวคิดและ
วัตถุประสงค์

+

ชุดคำสั่งที่
เป็นระบบ

=

ผลลัพธ์การทำงาน
ของคอมพิวเตอร์

การเรียบเรียงหรือจัดเรียงคำสั่งเพื่อให้คอมพิวเตอร์
ทำงานตามวัตถุประสงค์ที่เราต้องการ

# ระดับของภาษาคอมพิวเตอร์

## ภาษาระดับต่ำ (Low-Level Language)

**ตัวอย่าง:** ภาษาแอสเซมบลี (Assembly)

- แปลงเป็นภาษาเครื่องได้เร็ว
- ผูกติดกับฮาร์ดแวร์
- ทำงานรวดเร็วแต่เขียนและแก้ไขยาก

## ภาษาระดับสูง (High-Level Language)

**ตัวอย่าง:** C, Python, PHP, Visual BASIC

- ใช้ภาษาคล้ายภาษามนุษย์ (ภาษาอังกฤษ)
- นำไปใช้ข้ามเครื่องได้ (ไม่ผูกติดฮาร์ดแวร์)
- ต้องใช้ตัวแปลภาษา

# ตัวแปลภาษาคอมพิวเตอร์

เครื่องมือแปลงภาษาระดับสูงให้เป็นภาษาเครื่อง (Machine Language: 01010010)

## แบบอินเทอร์พรีเตอร์ (Interpreter)

แปลทีละคำสั่ง → ทำงานทันที

**ข้อดี/ข้อเสีย:** ตรวจแก้ไขข้อผิดพลาดง่าย แต่ทำงานช้า เพราะต้องแปลใหม่ทุกครั้ง

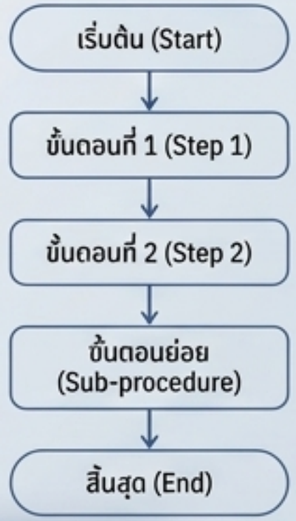
## แบบคอมไพเลอร์ (Compiler)

แปลรวดเดียวทั้งโปรแกรม → ได้ไฟล์ประมวลผล (Execute File)

**ข้อดี/ข้อเสีย:** ทำงานรวดเร็ว ไม่ต้องแปลซ้ำ แต่ตรวจสอบและแก้ไขข้อผิดพลาดยากกว่า

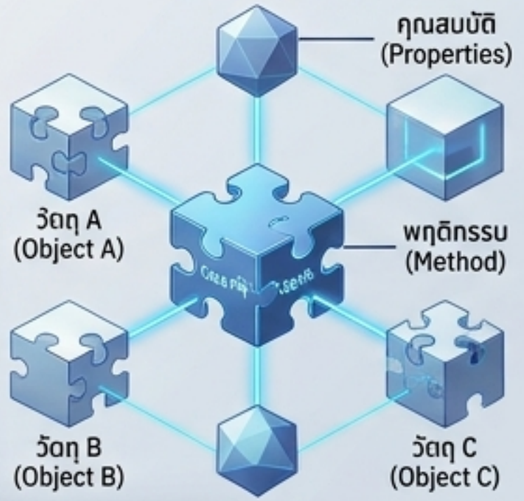
# แนวคิดในการเขียนโปรแกรม

## 1. แบบเชิงกระบวนการ (Procedural Programming)




แบ่งโปรแกรมเป็นส่วนย่อยตามหน้าที่ (Procedure/Function)
ตรวจสอบง่าย เหมาะกับโปรแกรมขนาดเล็กหรือผู้เริ่มต้น

## 2. แบบเชิงวัตถุ (Object-Oriented Programming - OOP)




มองส่วนต่าง ๆ เป็น "วัตถุ" ที่มีคุณสมบัติ (Properties)
และพฤติกรรม (Method) นำไปต่อยอดใช้ซ้ำได้ง่าย
เหมาะกับระบบงานขนาดใหญ่

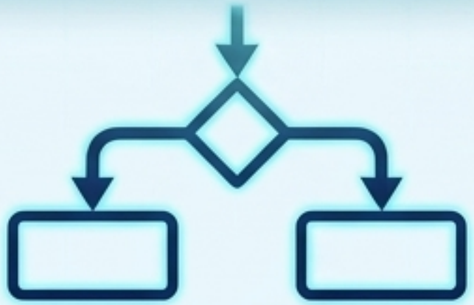
# รูปแบบการควบคุมการทำงาน

โครงสร้างพื้นฐาน 3 รูปแบบที่ประกอบกันเป็นโปรแกรม

## การทำงานตามลำดับ (Sequence)

ทำงานเรียงตามลำดับคำสั่งจากบนลงล่าง

## การเลือกทำ (Selection / Decision)



ตรวจสอบเงื่อนไข
เพื่อตัดสินใจเลือกเส้นทางทำงาน

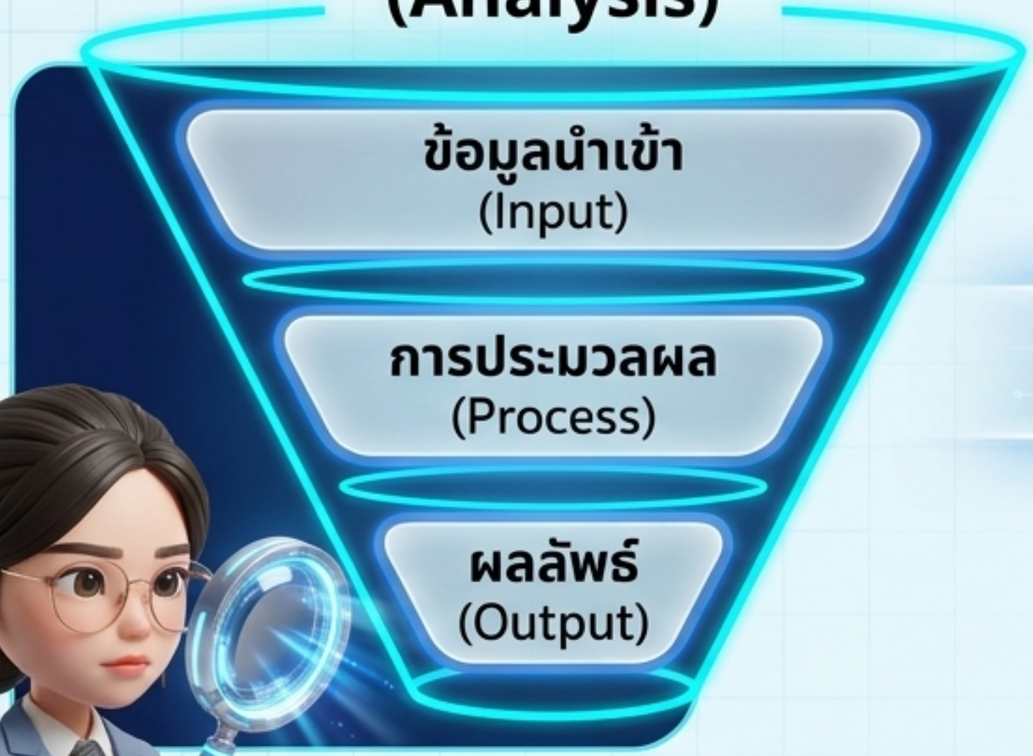
## การทำซ้ำ (Repetition / Loop)

ควบคุมให้โปรแกรมวนกลับมาทำงานซ้ำ
ตามเงื่อนไขที่กำหนด

# ขั้นตอนการพัฒนาโปรแกรม

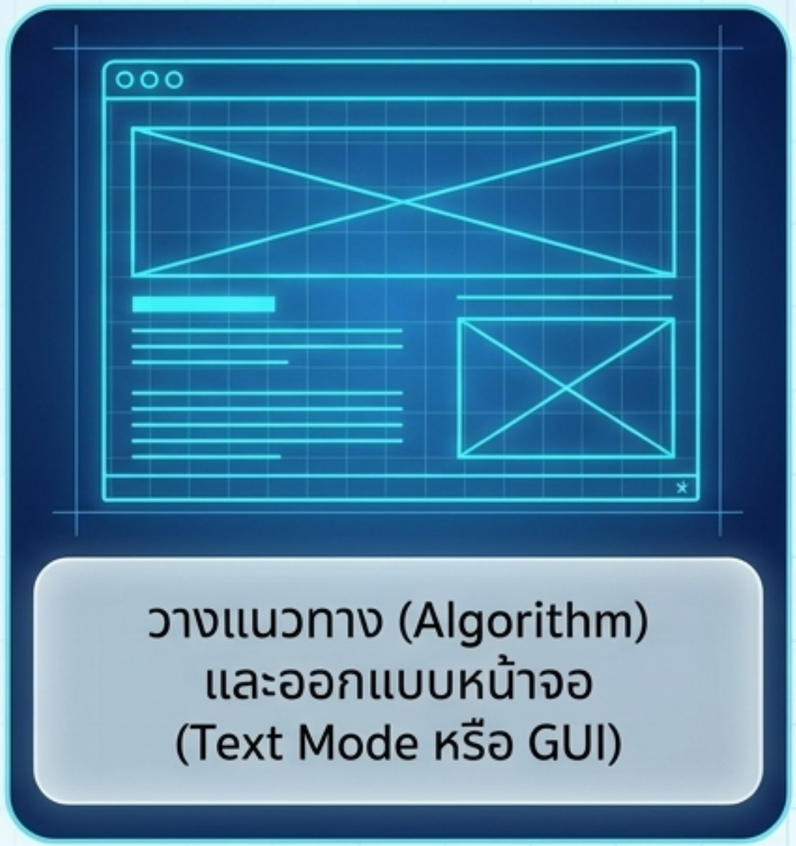

# Step 1 & 2 : วิเคราะห์และออกแบบ

## การวิเคราะห์ปัญหา (Analysis)




ความยาวฐาน, ความสูง -> 0.5 * ฐาน * สูง -> พื้นที่สามเหลี่ยม

## การออกแบบโปรแกรม (Design)



วางแนวทาง (Algorithm)
และออกแบบหน้าจอ
(Text Mode หรือ GUI)

# Step 3 & 4: เขียนโค้ดและทดสอบ

## การเขียนโปรแกรม (Coding)

```c
#include <stdio.h>
main()
{
    int b, h;
    float A;
    printf("Area of Triangle \n");
    A = 0.5 * b * h;
    printf("Area = %3.2f \n", A);
}
```

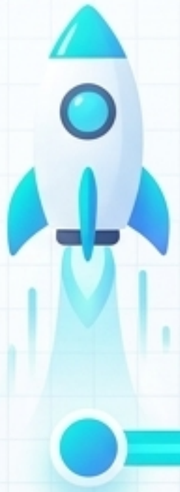
## การทดสอบโปรแกรม (Testing)

**Input:** Base=10, Height=5 ->
**Output:** Area=25.00

**Input:** Base=6, Height=8 ->
**Output:** Area=24.00

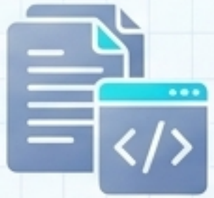
**Input:** Base=A, Height=B ->
**Output:** Error

# Step 5 & 6: จัดทำเอกสารและบำรุงรักษา



**Software Release**

## การรวบรวมเอกสาร (Documentation)



รวบรวมเอกสาร (Source Code, รหัสเทียม, ผังงาน) เพื่อเป็นคู่มือและประวัติการแก้ปัญหา

## การบำรุงรักษา (Maintenance)

คอยติดตามดูแลผู้ใช้งาน, ฝึกอบรม, และอัปเดตระบบเมื่อพบข้อบกพร่องในภายหลัง

# เครื่องมือออกแบบ: รหัสเทียม (Pseudocode)

การเขียนลำดับการทำงานด้วยภาษาที่เข้าใจง่าย (มักเป็นภาษาอังกฤษ)

**ภาษาธรรมชาติ**

ตรวจสอบคะแนน
ถ้าได้น้อยกว่า 50
ให้แสดงคำว่า ‘สอบตก’
ถ้าไม่ใช่ให้แสดง ‘สอบผ่าน’

**รหัสเทียม (Pseudocode)**

```
Begin
  input score
  if score < 50
    print “Fail”
  else
    print “Pass”
End
```

# เครื่องมือออกแบบ: ผังงาน (Flowchart)

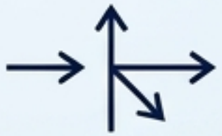
สัญลักษณ์มาตรฐานที่ใช้แสดงลำดับการทำงานของโปรแกรม

จุดเริ่มต้น/สิ้นสุด (Start/Stop)

รับ/แสดงข้อมูล (Input/Output)

กำหนดค่า/ประมวลผล (Process)

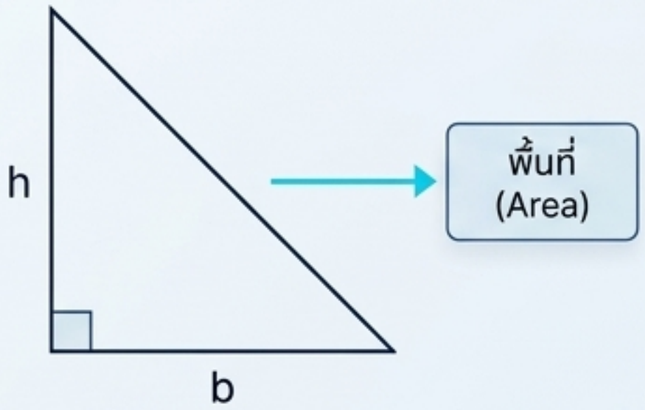
การตัดสินใจ (Decision)



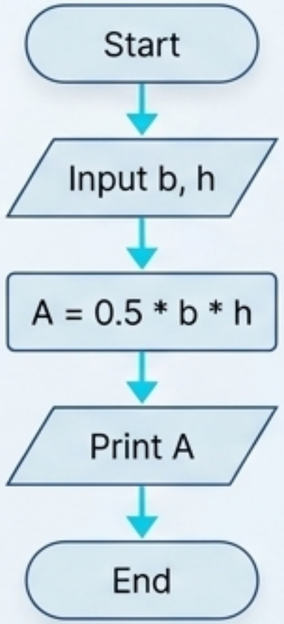
ทิศทางการทำงาน (Flow lines)

# บทสรุป: จากปัญหา สู่โปรแกรม

## ปัญหา (Problem)




## ออกแบบ (Flowchart)




## โปรแกรม (Code)

Area of Triangle = ...

```c
#include <stdio.h>
int main() {
    float b, h, A;
    printf("Area of Triangle \n");
    printf("Base = "); scanf("%f", &b);
    printf("Height = "); scanf("%f", &h);
    A = 0.5 * b * h;
    printf("Area = %.2f \n", A);
    return 0;
}
```