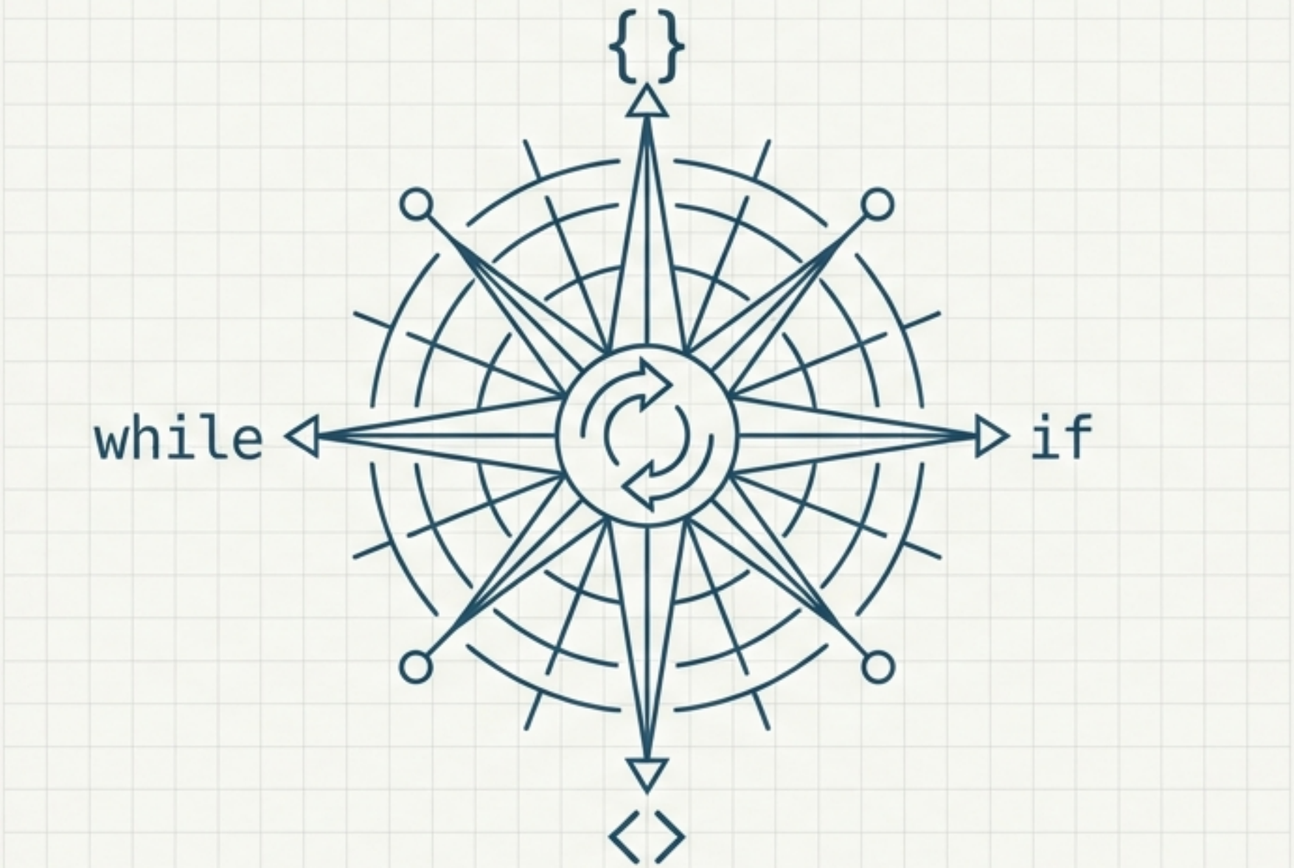
การควบคุมทิศทางโปรแกรม: เจาะลึก `if-else` และ `while` Loops



เข็มทิศนำทางโค้ดของคุณสู่โปรแกรมที่ฉลาดและยืดหยุ่น

โปรแกรมที่วิ่งเป็นเส้นตรง... ทำอะไรได้จำกัด

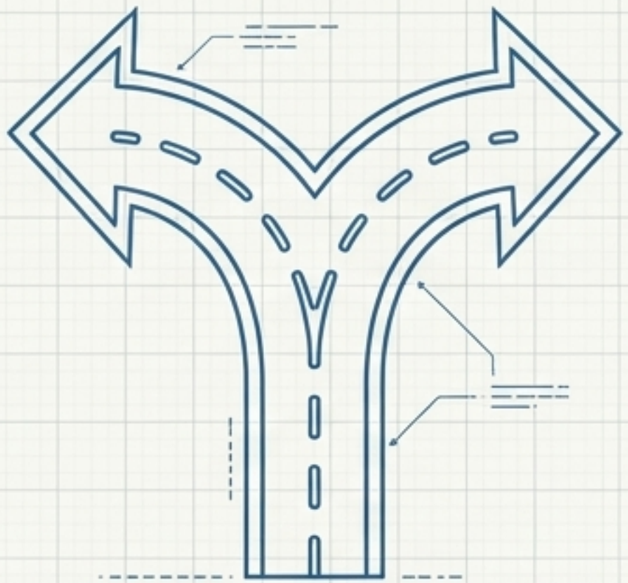
โปรแกรมพื้นฐานจะทำงานตามลำดับจากบนลงล่างเสมอ ซึ่งหมายความว่ามันจะให้ผลลัพธ์แบบเดิมทุกครั้ง


```
# โปรแกรมคำนวณราคาสินค้าแบบตายตัว
price = 100
vat_rate = 0.07
total_price = price + (price * vat_rate)
print(f"ราคารวม: {total_price}") # ไม่ว่าจะรันกี่ครั้งก็ได้ผลลัพธ์เท่าเดิม
```

จะเกิดอะไรขึ้นถ้าเราต้องการให้โปรแกรมตัดสินใจเองได้? หรือทำงานซ้ำๆ จนกว่าจะสำเร็จ?

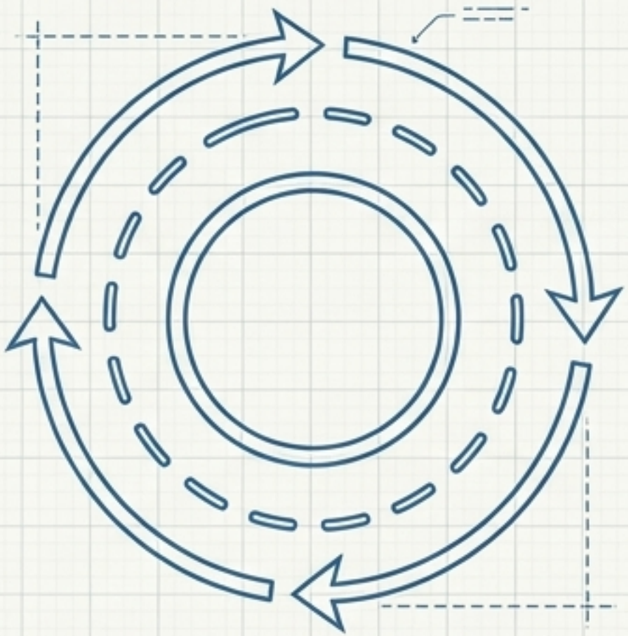
`if` และ `while`: เข็มทิศนำทางโค้ดของคุณ

เราใช้โครงสร้างควบคุม (Control Structures) เพื่อกำหนดเส้นทางการทำงานของโค้ด ทำให้โปรแกรมของเราสามารถตัดสินใจและทำงานซ้ำได้



`if` (การตัดสินใจ ณ ทางแยก)

สำหรับเลือกทำอย่างใดอย่างหนึ่งเพียงครั้งเดียว



`while` (การเดินทางซ้ำในภารกิจ)

สำหรับทำงานซ้ำๆ จนกว่าภารกิจจะสำเร็จ

ทางเลือกแรก: คำสั่ง `if`

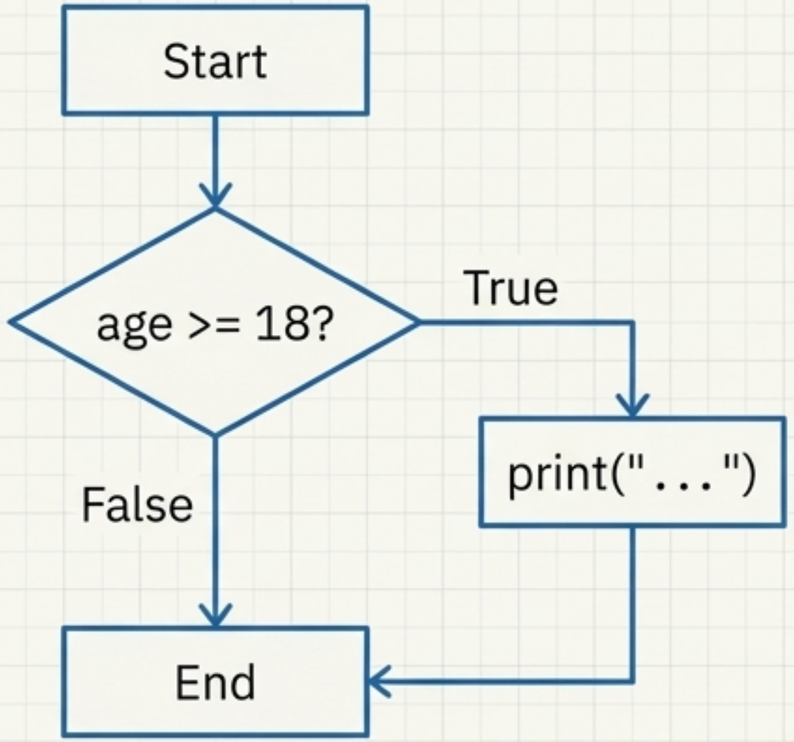
เหมือนการตรวจสอบเงื่อนไขเพียงครั้งเดียว: ถ้าจริงก็ทำ, ถ้าไม่จริงก็ข้ามไปเลย

☑ Syntax

```
if condition:
    # code to execute if condition is true
```

ตัวอย่างโค้ด: ตรวจสอบอายุผู้มีสิทธิ์เลือกตั้ง

```
age = 20
if age >= 18:
    print("คุณมีสิทธิ์เลือกตั้ง")
```



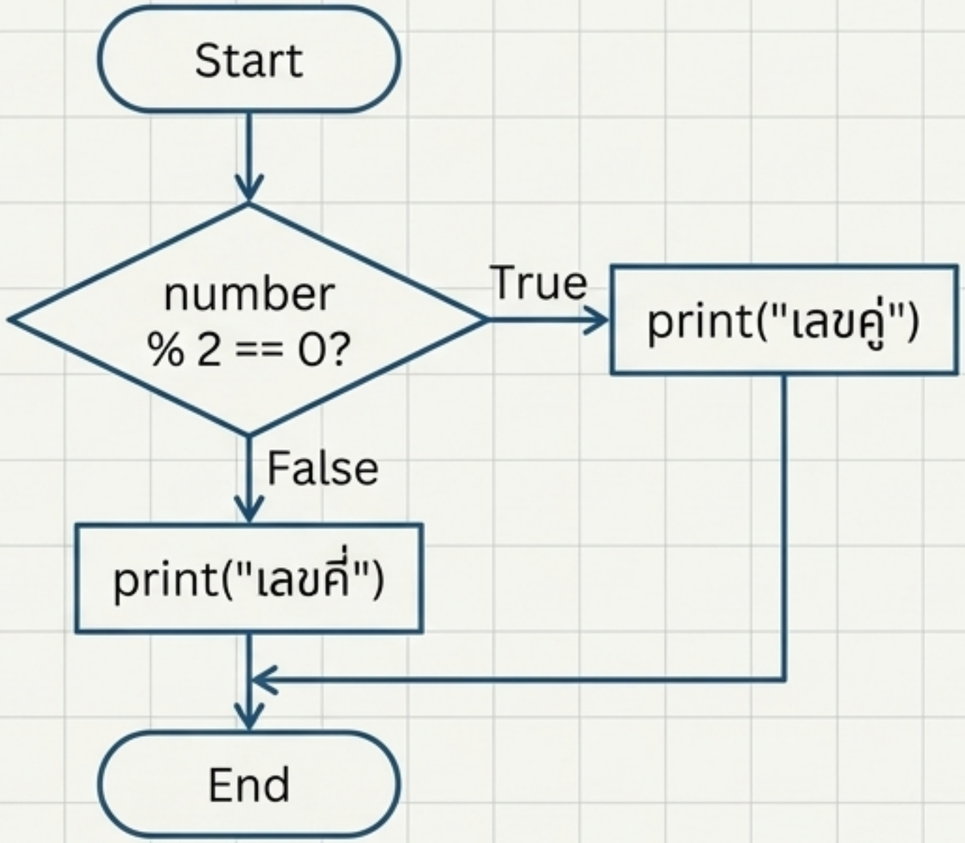
Quiz ❓: ถ้ากำหนดให้ `age = 15` โค้ดนี้จะพิมพ์ข้อความอะไรออกมา? เพราะอะไร?

เมื่อมีสองทางแยก: `if-else`

เหมือนทางแยกที่บังคับให้ต้องเลี้ยวซ้ายหรือขวาเท่านั้น จะมีโค้ดที่ทำงานเสมอไม่ว่าเงื่อนไขจะเป็นจริงหรือเท็จ

☑ Syntax

```
if condition:
    # runs if condition is true
else:
    # runs if condition is false
```



ตัวอย่างโค้ด: ตรวจสอบเลขคู่-คี่

```
number = 10
if number % 2 == 0:
    print(f"{number} เป็นเลขคู่")
else:
    print(f"{number} เป็นเลขคี่")
```

Quiz ❓ : ถ้า number = 7 ผลลัพธ์คืออะไร?

สี่แยกใหญ่หลายเส้นทาง: `if-elif-else`

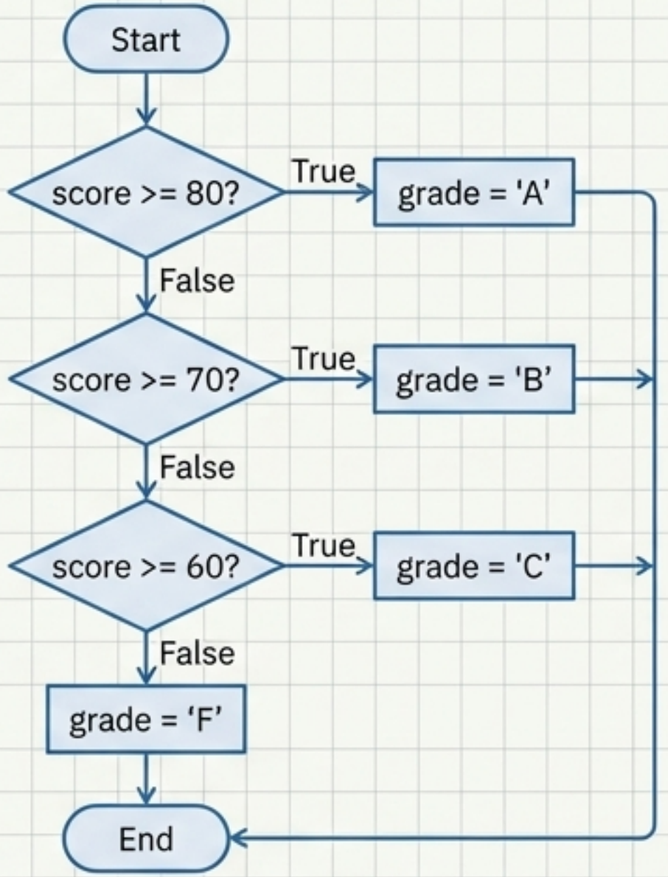
เหมือนสี่แยกใหญ่ที่มีหลายทางเลือก โปรแกรมจะตรวจสอบไปทีละเส้นทาง และจะเลือกเดินทางแรกที่เป็นจริงเท่านั้น

✅ Syntax

```
if condition1:
    # block 1
elif condition2:
    # block 2
else:
    # block 3
```

ตัวอย่างโค้ด: โปรแกรมคำนวณเกรด

```
score = 85
if score >= 80:
    grade = "A"
elif score >= 70:
    grade = "B"
elif score >= 60:
    grade = "C"
else:
    grade = "F"
print(f"คะแนน {score} ได้เกรด: {grade}")
```



Quiz ❓: ถ้า `score = 75` โปรแกรมจะพิมพ์เกรดอะไร และจะหยุดเช็คเงื่อนไขเมื่อไหร่?

สร้างเงื่อนไขซับซ้อน: `if` กับ `and`, `or`, `not`

เหมือนการให้เส้นทางที่ต้องพิจารณาหลายปัจจัยพร้อมกัน เช่น 'เลี้ยวขวาต่อเมื่อไฟเขียว **และ** ไม่มีคนข้ามถนน'

`and` (และ): เงื่อนไข **ทุกข้อ** ต้องเป็นจริง

`or` (หรือ): เงื่อนไข **อย่างน้อยหนึ่งข้อ** เป็นจริง

`not` (ไม่): กลับค่าความจริง (True เป็น False, False เป็น True)

ตัวอย่างโค้ด: ตรวจสอบสิทธิ์เข้าใช้งานระบบ

```
age = 25
is_premium_member = True

if age >= 18 and is_premium_member:
    print("ยินดีต้อนรับสู่โซนพรีเมียม")
elif age < 18 or not is_premium_member:
    print("ไม่สามารถเข้าถึงโซนนี้ได้")
```

Quiz ❓ : จากโค้ดตัวอย่าง, ถ้า `age = 20` และ `is_premium_member = False` เงื่อนไข `age >= 18 and is_premium_member` จะเป็น True หรือ False?

การเดินทางซ้ำๆ: `while` Loop 🔄

"คนซุปไปเรื่อยๆ **ตราบใดที่** ซุปยังไม่เดือด"

ส่วนประกอบสำคัญของ Loop

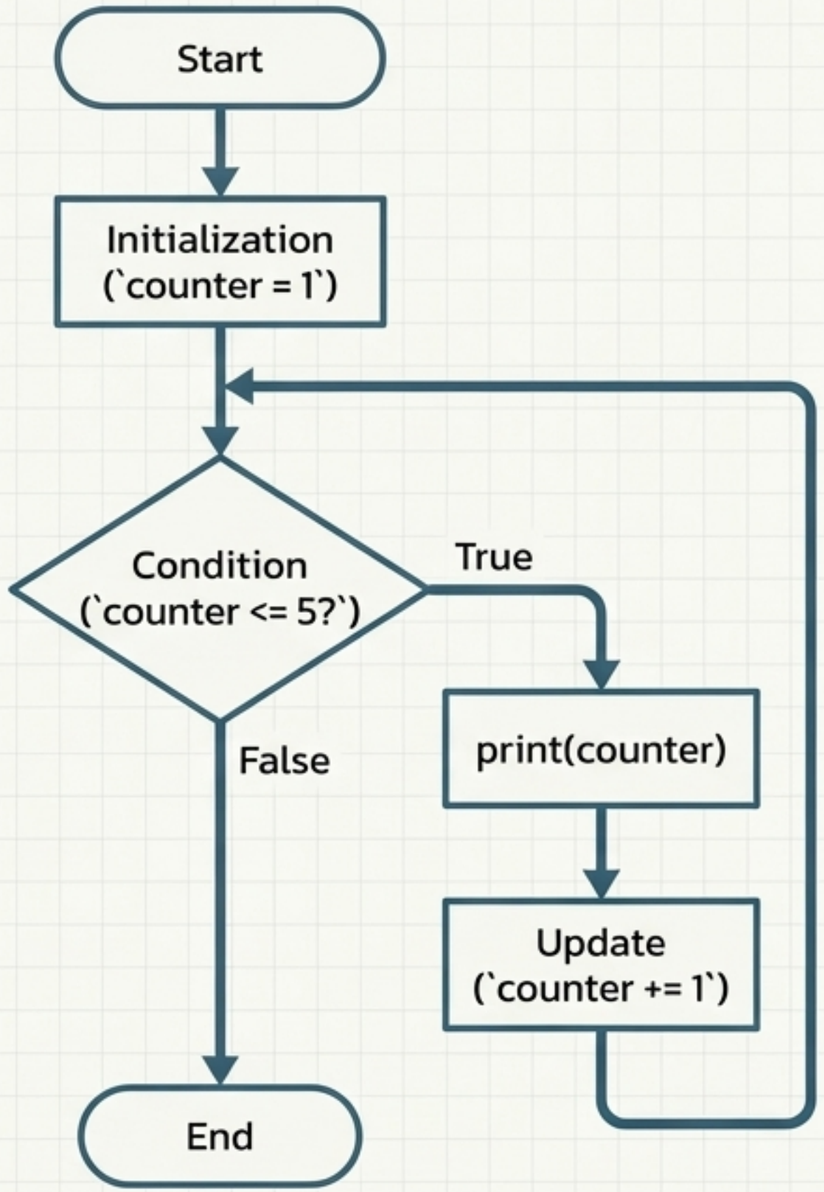
1. **Initialization**: การกำหนดค่าเริ่มต้น (เช่น `counter = 1`)
2. **Condition**: เงื่อนไขที่ต้องเป็นจริงเพื่อวนซ้ำ (เช่น `while counter <= 5`)
3. **Update**: การเปลี่ยนแปลงค่าเพื่อให้เงื่อนไขไปถึงจุดสิ้นสุด (เช่น `counter += 1`)

ตัวอย่างโค้ด: นับเลข 1 ถึง 5

```
counter = 1           # 1. Initialization
while counter <= 5:   # 2. Condition
    print(counter)
    counter += 1      # 3. Update
# Output: 1 2 3 4 5
```

ข้อควรระวัง ⚠️ : หากลืมส่วน Update จะทำให้เกิด **Infinite Loop!**

Quiz ❓ : โค้ดนี้จะพิมพ์ตัวเลขสุดท้ายคือเลขอะไร?



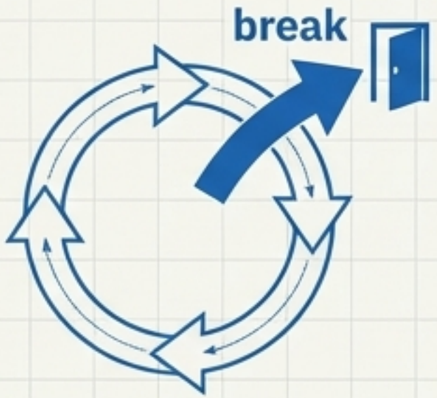
ควบคุมเส้นทางใน Loop: `break` และ `continue`

บางครั้งเราไม่ต้องการทำ Loop จนจบ หรือต้องการข้ามบางรอบไป

`break` 🚪 (ทางออกฉุกเฉิน)

- **หน้าที่:** หยุดการทำงานและออกจาก `while` loop ทันที
- **ตัวอย่าง:** หยุดค้นหาเมื่อเจอสิ่งที่ต้องการ

```
i = 1
while i <= 10:
    if i == 5:
        print("เจอเลข 5 แล้ว! หยุดค้นหา")
        break
    print(f"กำลังเช็คเลข... {i}")
    i += 1
# Output: ...เช็คเลข... 4, เจอเลข 5 แล้ว! หยุดค้นหา
```



`continue` ⏭️ (ข้ามไปรอบถัดไป)

- **หน้าที่:** หยุดการทำงานในรอบปัจจุบัน แล้วกระโดดไปเริ่มรอบถัดไปทันที
- **ตัวอย่าง:** พิมพ์เฉพาะเลขคู่

```
i = 0
while i < 10:
    i += 1
    if i % 2 != 0:   # ถ้าเป็นเลขคี่
        continue     # ข้ามไปเลย
    print(f"{i} เป็นเลขคู่")
```

Quiz ❓ : ในโค้ด `continue` ตัวอย่าง ทำไมเลข 1, 3, 5, 7, 9 ถึงไม่ถูกพิมพ์ออกมา?

เมื่อ Loop จบลงอย่างสมบูรณ์: `while-else`

"ค้นหากุญแจในบ้านจนครบทุกห้อง **ถ้า** หาไม่เจอ (loop จบปกติ) **ค่อย** โทรหาช่าง" (ถ้าเจอก่อนแล้วหยุดค้น (`break`), ไม่ต้องโทร)

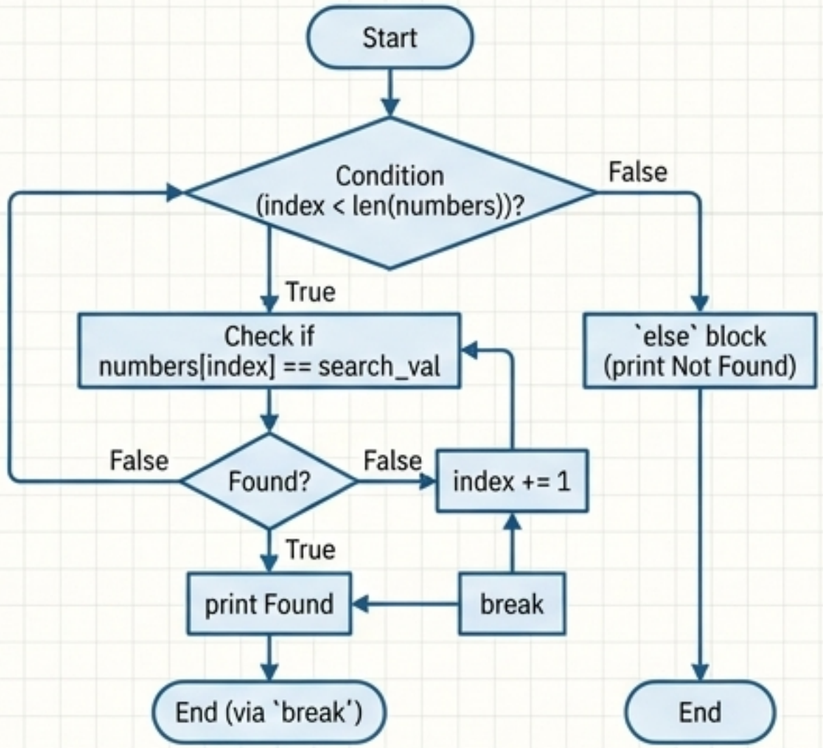
✅ **Syntax & Explanation:** บล็อก `else` จะทำงานก็ต่อเมื่อ `while` loop ทำงานจนจบเงื่อนไข (condition เป็น False) โดย **ไม่มีการ** `break` ออกไปก่อน

ตัวอย่างโค้ด: ค้นหาตัวเลขในลิสต์

```
numbers = [1, 3, 7, 9]
search_val = 5
index = 0

while index < len(numbers):
    if numbers[index] == search_val:
        print(f"เจอ {search_val} ที่ตำแหน่ง {index}")
        break
    index += 1
else:  # This 'else' belongs to 'while', not 'if'
    print(f"ไม่เจอ {search_val} ในลิสต์ (ค้นหาจนครบแล้ว)")

# Output: ไม่เจอ 5 ในลิสต์ (ค้นหาจนครบแล้ว)
```



Quiz ❓ : ถ้าเปลี่ยน `search_val = 7` ผลลัพธ์จะเป็นอย่างไร และ `else` block จะทำงานหรือไม่?

สร้างภารกิจโต้ตอบ: เกมทายตัวเลข 🎮

เหมือนเกมที่เราต้องพยายามไปเรื่อยๆ "จนกว่า" จะทำภารกิจสำเร็จ

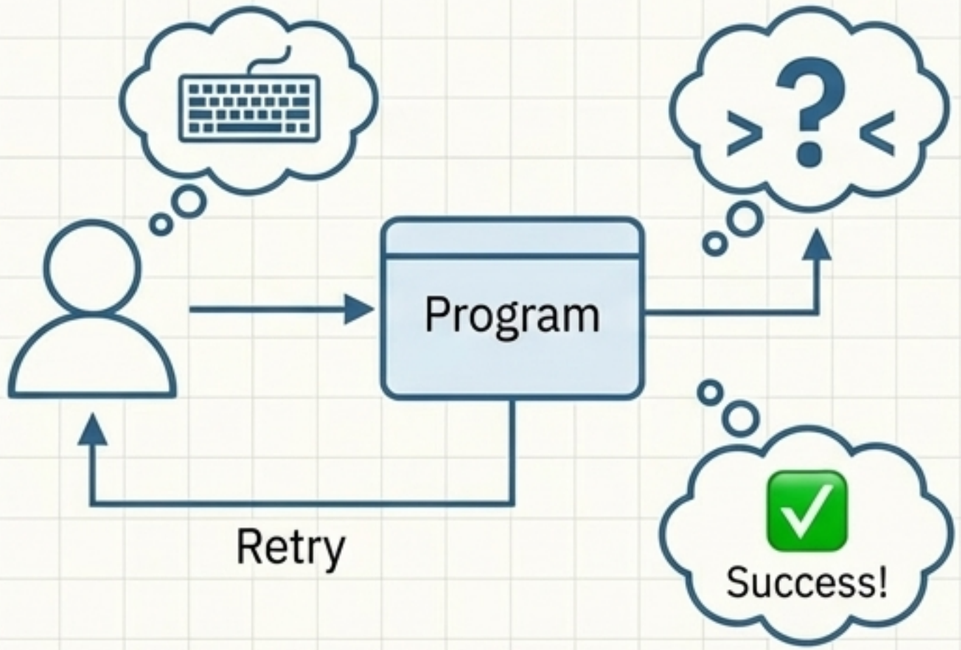
Concept : เราสามารถใช้ `while` loop เพื่อรับ input จากผู้ใช้ไปเรื่อยๆ จนกว่าจะได้ค่าที่ถูกต้องตามเงื่อนไข

ตัวอย่างโค้ด: เกมทายตัวเลข 1-10

```python
import random
secret_number = random.randint(1, 10)
guess = 0 # ตั้งค่าเริ่มต้นที่ไม่ใช่คำตอบ

print("--- เกมทายตัวเลข 1-10 ---")
while guess != secret_number:
    guess = int(input("ทายตัวเลข: "))
    if guess < secret_number:
        print("น้อยไป! ลองอีกครั้ง")
    elif guess > secret_number:
        print("มากไป! ลองอีกครั้ง")

print(f"ถูกต้อง! 🎉 คำตอบคือ {secret_number}")
```



Quiz ❓ : เงื่อนไขอะไรที่ทำให้ `while` loop นี้หยุดทำงาน?

⚠️ อันตราย! วงเวียนที่ไม่มีทางออก: Infinite Loop

เหมือนแผ่นเสียงตกร่อง หรือรถที่ติดอยู่ในวงเวียนหาทางออกไม่ได้

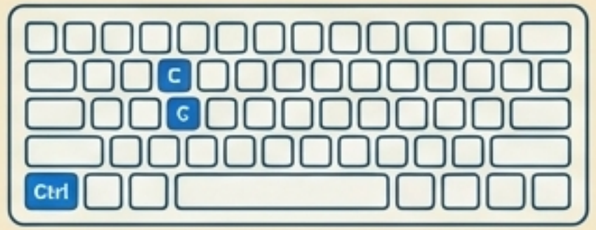
สาเหตุ: เกิดขึ้นเมื่อเงื่อนไขของ `while` **เป็นจริงเสมอ** และ**ไม่มีการอัปเดตค่า** (update statement) ที่จะทำให้เงื่อนไขเป็นเท็จได้

ตัวอย่างโค้ดที่ผิดพลาด (ห้ามรันโดยไม่รู้วิธีหยุด!)

```
counter = 1
while counter > 0:
    print("Help, I'm stuck in a loop!")
    # counter จะเป็นบวกเสมอ ไม่มีวันจบ
    # Missing update: counter += 1
```

วิธีหยุดโปรแกรม (สำคัญมาก)

ใน Terminal หรือ Command Line: กด



Ctrl + C

คำแนะนำ: ก่อนรัน `while` loop, ถามตัวเองเสมอว่า “อะไรคือเงื่อนไขที่จะทำให้ loop นี้จบ?”

Quiz ❓ : ต้องแก้ไขโค้ดตัวอย่างอย่างไรเพื่อให้มันทำงานแค่ 5 รอบแล้วหยุด?

สรุป: เลือกเครื่องมือให้ถูก `if` หรือ `while`?

เมื่อไหร่ควรใช้ `if`? เมื่อไหร่ควรใช้ `while`?

ลักษณะ	`if-else` (สี่แยก) 	`while` (วงเวียน) 
จุดประสงค์	การตัดสินใจ (Decision)	การทำซ้ำ (Repetition)
การทำงาน	ทำงานเพียง **ครั้งเดียว** ตามเงื่อนไข	ทำงาน **ซ้ำไปเรื่อยๆ** จนกว่าเงื่อนไขจะเป็นเท็จ
Use Case	• ตรวจสอบรหัสผ่าน • คำนวณเกรด • เช็คสถานะของข้อมูล	• นับจำนวน • รับ input จนกว่าจะถูกต้อง • ประมวลผลข้อมูลหลายชิ้น

หลักการจำ : ใช้ `if` เมื่อต้องการ **เลือก** เส้นทาง, ใช้ `while` เมื่อต้องการ **ทำซ้ำ** ภารกิจ

ทดสอบเข็มทิศของคุณ: แบบฝึกหัด 🚀

โจทย์ 1 (ง่าย): FizzBuzz

เขียนโปรแกรมที่ใช้ `while` loop นับเลข 1 ถึง 100.
ใช้ `if-elif-else` เพื่อพิมพ์ 'Fizz' ถ้าหาร 3 ลงตัว, 'Buzz' ถ้าหาร 5 ลงตัว, และ 'FizzBuzz' ถ้าหารทั้ง 3 และ 5 ลงตัว. มิเช่นนั้นให้พิมพ์ตัวเลขนั้นๆ.

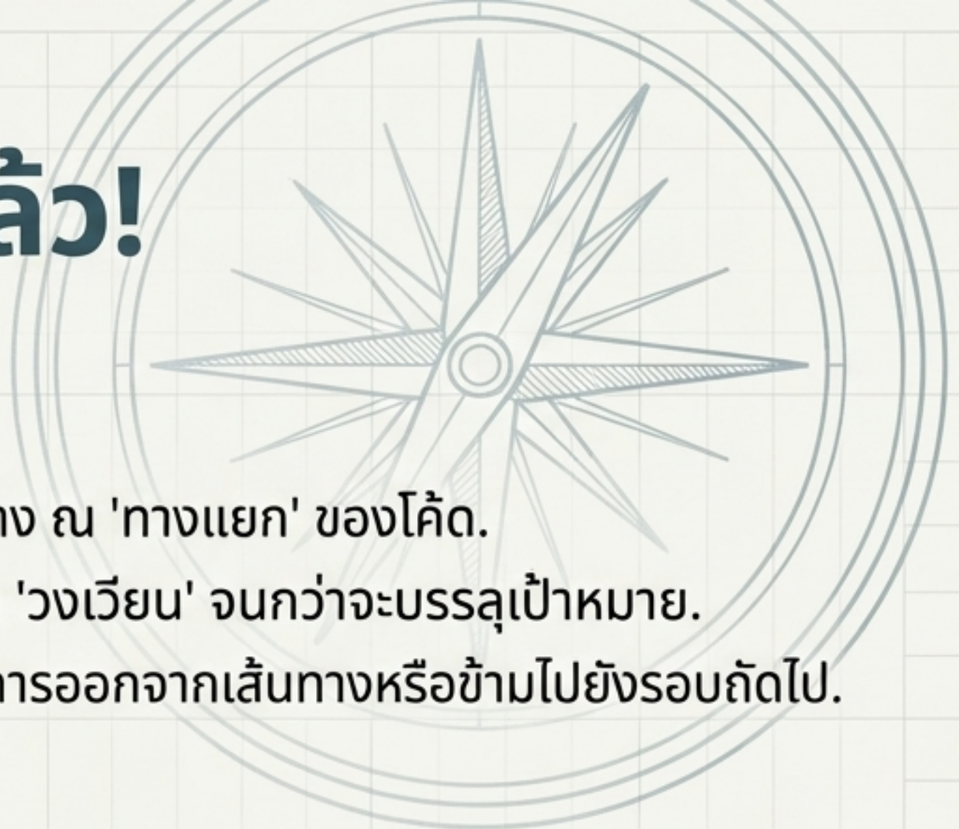
โจทย์ 2 (กลาง): โปรแกรมเมนูง่ายๆ

สร้างโปรแกรมที่ให้ผู้ใช้เลือกเมนู (1. สั่งอาหาร, 2. ดูบิล, 9. ออกจากโปรแกรม)
ใช้ `while True` เพื่อให้โปรแกรมทำงานวนไปเรื่อยๆ และใช้ `break` เพื่อออกจาก loop เมื่อผู้ใช้เลือก 9.

โจทย์ 3 (ท้าทาย): ตรวจสอบจำนวนเฉพาะ (Prime Number)

รับตัวเลขจากผู้ใช้.
เขียน `while` loop เพื่อตรวจสอบว่ามีเลขอื่น (นอกจาก 1 และตัวมันเอง) ที่หารเลขนั้นลงตัวหรือไม่ เพื่อตัดสินว่าเป็นจำนวนเฉพาะหรือไม่.

คุณได้ครอบครองเข็มทิศแล้ว!



สรุปทบทวน (Key Takeaways)

- ✅ `if`, `elif`, `else`: คือเครื่องมือในการตัดสินใจเลือกเส้นทาง ณ 'ทางแยก' ของโค้ด.
- 🔄 `while`: คือเครื่องมือในการทำภารกิจซ้ำๆ หรือการเดินทางใน 'วงเวียน' จนกว่าจะบรรลุเป้าหมาย.
- 🚪 `break` และ `continue`: คือเครื่องมือควบคุมพิเศษ สำหรับการออกจากเส้นทางหรือข้ามไปยังรอบถัดไป.

เส้นทางต่อไป (Next Steps):

ในบทเรียนหน้า เราจะไปรู้จักกับ loop อีกรูปแบบที่ทรงพลังและนิยมใช้กับข้อมูลเป็นชุด นั่นคือ `**for**` loop!

แหล่งข้อมูลอ้างอิง (Reference):

Python Control Flow Documentation: docs.python.org/3/tutorial/controlflow.html