เจาะลึก Loop: พลังแห่งการควบคุมด้วย `while`

ทำความเข้าใจ `while` และแนวคิด `do-while` ใน Python

ทำซ้ำอย่างไร้ประสิทธิภาพ vs. ทำซ้ำอย่างชาญฉลาด

```
# ต้องการถามรหัสผ่านจนกว่าจะถูก
password = input("กรุณาใส่รหัสผ่าน: ")
# ถ้าผิด? ก็ต้องเขียนซ้ำ
password = input("กรุณาใส่รหัสผ่าน: ")
# ถ้ายังผิดอีก?
password = input("กรุณาใส่รหัสผ่าน: ")
```

ต้องการถามรหัสผ่านจนกว่าจะถูก กำธียิอาโลิเผลกบอตั้มคาน password = input(" ก็ต้องเขียนซ้ำ password = input(" กรุณาใส่รหัสผ่าน: ")

จะเกิดอะไรขึ้นถ้าเราไม่รู้ว่าต้องทำซ้ำกี่ครั้ง?

เราจะเขียนโปรแกรมที่ 'รอ' จนกว่าเงื่อนไขจะถูกต้องได้อย่างไร?

ยอเาจะเขียนโปรแกรมที่หลุด เตือง ทำใหลุก ซึ้งเซื่อมาารโคา คณเ่ะแท้ ถ้อกในเป็นอดจนกว่าเงื่อนโขจะวีนาชา

คำสั่ง while: "ยามเฝ้าประตู" ของการทำซ้ำ

คำสั่ง `while` เป็นคำสั่งที่ใช้ในการวนทำซ้ำ โดยจะตรวจสอบเงื่อนไข (Condition) ก่อนเสมอ ถ้าเงื่อนไขเป็นจริง (True) จึงจะอนุญาตให้โค้ดด้านในทำงาน แล้ววนกลับมาตรวจสอบเงื่อนไขใหม่อีกครั้ง

ตราบใดที่... (เงื่อนไข)... ให้ทำ...

```
while <Condition>:
    # Statements # บล็อกของคำสั่งที่จะทำซ้ำ
    # ...ต้องมีโค้ดที่ทำให้ Condition มีโอกาสเป็น False!
```

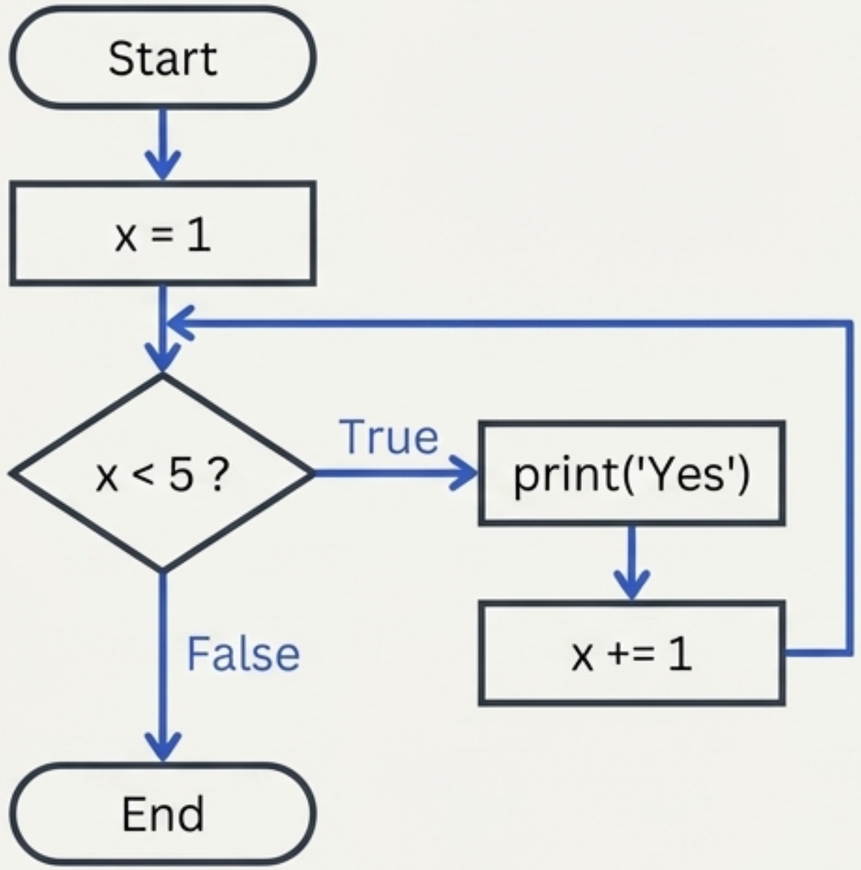
โครงสร้างและผังการทำงานของ `while`

Annotated Code

```
x = 1          # 1. กำหนดค่าเริ่มต้นให้กับตัวแปร

while x < 5:   # 2. ตรวจสอบเงื่อนไขก่อนเข้าลูป
    print('Yes')
    x += 1     # 3. อัปเดตค่า เพื่อให้เงื่อนไข
               เปลี่ยนแปลงและมีวันสิ้นสุด
```

Flowchart



ข้อควรระวัง: ลูปไม่รู้จบ (Infinite Loop)

กับดักที่พบบ่อยที่สุดของ `while` คือการลืมอัปเดตตัวแปรที่ใช้ในเงื่อนไข ทำให้เงื่อนไขนั้นเป็น `True` ตลอดไป และโปรแกรมจะทำงานค้างอยู่ในลูปนั้นไม่สิ้นสุด

```
x = 1
while x < 5:
    print("ช่วยด้วย, ฉันติดอยู่ในลูป!")
    # ลืม x += 1 !!! โปรแกรมจะทำงานไม่หยุด
```

โจทย์ใหม่: แล้วถ้าเราอยากให้โค้ดทำงาน "อย่างน้อยหนึ่งครั้ง" ก่อนตรวจสอบล่ะ?

ลองนึกถึงโปรแกรมที่ต้องแสดงเมนูให้ผู้ใช้เลือกก่อน แล้วค่อยตรวจสอบว่าตัวเลือกนั้นถูกต้องหรือไม่ หรือเกมที่ต้องให้ผู้ใช้ทายตัวเลขก่อนหนึ่งครั้ง แล้วค่อยบอกผลลัพธ์

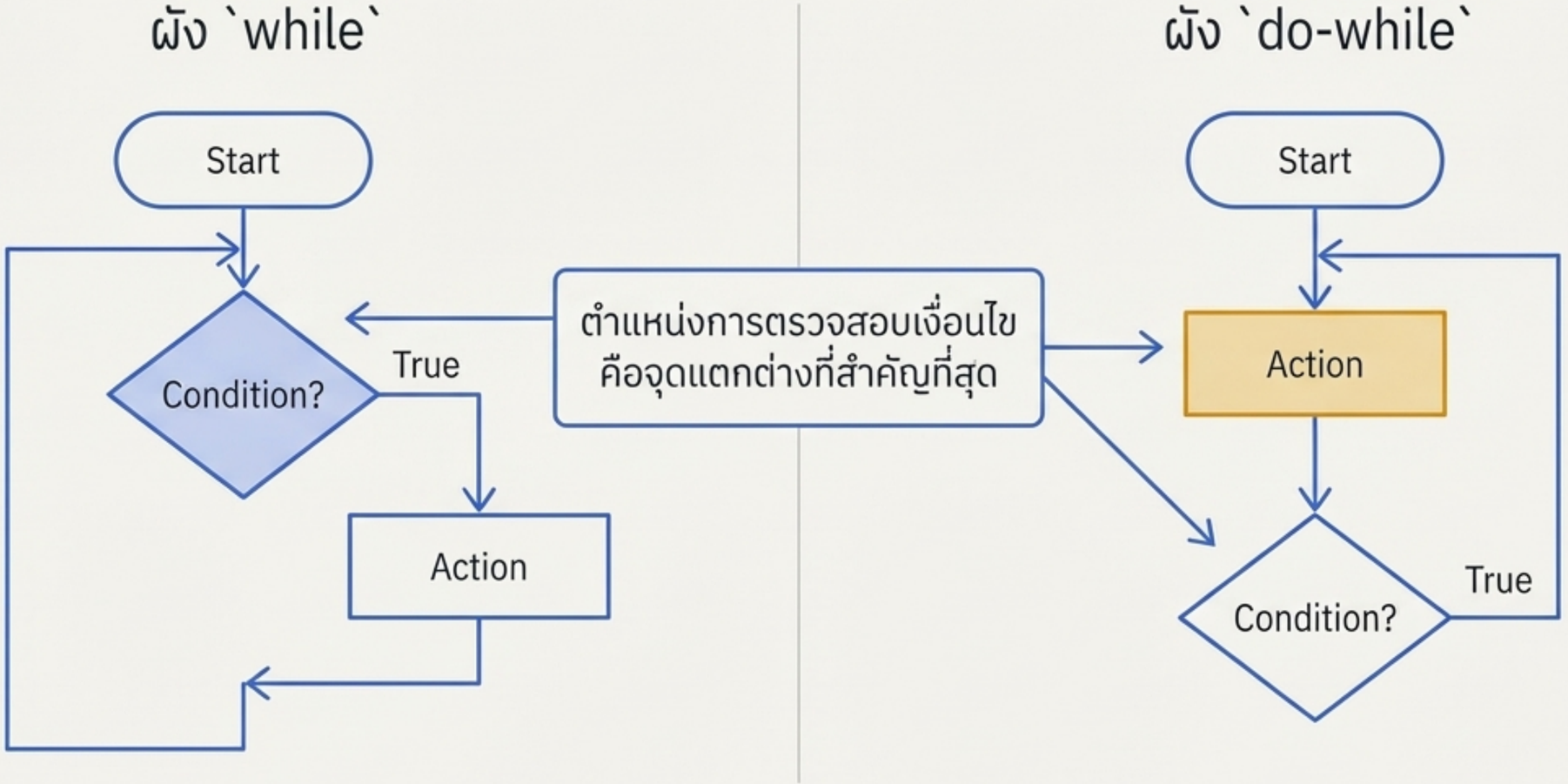
ในกรณีเหล่านี้ การตรวจสอบเงื่อนไข *ก่อน* ทำงานอาจจะไม่ใช่สิ่งที่เหมาะสมที่สุด เราต้องการตรรกะแบบ 'ทำก่อน แล้วค่อยตรวจสอบ'

แนวคิดของ `do-while`: "ประตูหมุน" ของการทำงาน

ตรรกะแบบ `do-while` ทำงานเหมือนประตูหมุน:
คุณจะได้เข้าไปทำงานในลูปก่อน 1 ครั้งเสมอ แล้วระบบจะตรวจสอบเงื่อนไขที่ทางออก เพื่อตัดสินใจว่าจะให้คุณวนกลับเข้าไปทำงานอีกรอบหรือไม่

ทำ... (คำสั่ง)... ตราบใดที่... (เงื่อนไข)...

เปรียบเทียบผังการทำงาน: เช็คก่อน (`while`) vs. ทำก่อน (`do-while`)



ความจริงเปิดเผย: Python ไม่มีคำสั่ง `do-while`!

ในไพธอนนั้นไม่สนับสนุนการตัดสินใจแบบ do-while (9.2.4)

...แต่เราสามารถจำลองการทำงานแบบ `do-while` ได้อย่างสวยงามและเป็นที่นิยมด้วยการใช้ `while True` ร่วมกับ `break`

```
while True:
    # โค้ดที่ต้องการให้ทำงานอย่างน้อย 1 ครั้งจะอยู่ตรงนี้
    # ...

    # หลังจากทำงานแล้ว จึงค่อยตรวจสอบเงื่อนไขเพื่อออกจากลูป
    if <condition_to_exit>:
        break # คำสั่งให้ออกจากลูปทันที
```

สรุปเปรียบเทียบ: `while` vs. `do-while` Pattern

คุณลักษณะ	`while` Loop	`do-while` Pattern (ใน Python)
ตรรกะการทำงาน	ตรวจสอบเงื่อนไข **ก่อน** ทำงาน	ทำงาน **ก่อน** ตรวจสอบเงื่อนไข
การรับประกันการทำงาน	อาจไม่ทำงานเลย (ถ้าเงื่อนไขเป็น False ตั้งแต่แรก)	ทำงานอย่างน้อย 1 ครั้งเสมอ
กรณีที่เหมาะสม	เมื่อไม่แน่ใจว่าต้องทำงานซ้ำหรือไม่ หรือต้องรอเงื่อนไขบางอย่าง	เมื่อต้องการให้เกิดการกระทำบางอย่างก่อนเสมอ เช่น แสดงเมนู, รับค่าจากผู้ใช้
โครงสร้างใน Python	`while <condition>:`	`while True: ...` `  if <exit_condition>: break`

แบบฝึกหัดพร้อมเฉลย: โปรแกรมทายตัวเลข

โจทย์ (Problem)
สร้างเกมทายตัวเลข โดยโปรแกรมจะสุ่มเลข 1-10 และให้ผู้ใช้ทายไปเรื่อยๆ จนกว่าจะถูก

การวิเคราะห์ (Analysis)
โจทย์นี้เหมาะกับ `do-while` pattern เพราะเราต้องให้ผู้ใช้ "ทาย' (ทำงาน) ก่อน" แล้วค่อย "ตรวจสอบ' ว่าคำตอบนั้นถูกหรือไม่"

โค้ดตัวอย่าง (Code)

```
import random
secret_number = random.randint(1, 10)

while True:
    # ทำงานก่อน: รับค่าจากผู้ใช้
    guess = int(input("ทายตัวเลข 1-10: "))

    # ตรวจสอบทีหลัง: เปรียบเทียบค่าที่ทายกับเลขที่สุ่ม
    if guess == secret_number:
        print("ถูกต้อง! ยินดีด้วย")
        break  # เงื่อนไขในการออกจากลูป
    else:
        print("ยังไม่ถูก, ลองอีกครั้ง!")
```

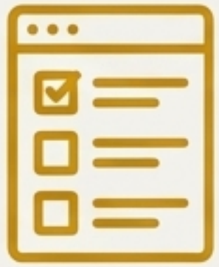
แบบฝึกหัดท้ายบท: ถึงตาคุณแล้ว!

โจทย์ข้อที่ 1 (Simple `while`)

เขียนโปรแกรมที่นับถอยหลังจาก 10 ถึง 1 และเมื่อนับถึง 0 ให้พิมพ์คำว่า ‘Liftoff!’

- คำใบ้: ต้องกำหนดค่าเริ่มต้นและลดค่าลงในแต่ละรอบ

โจทย์ข้อที่ 2 (`do-while` pattern)



เขียนโปรแกรมสร้างเมนูง่ายๆ (1. แสดงข้อมูล, 2. เพิ่มข้อมูล, 9. ออกจากโปรแกรม) โดยโปรแกรมจะแสดงเมนูซ้ำไปเรื่อยๆ และรับค่าจากผู้ใช้จนกว่าผู้ใช้จะพิมพ์เลข 9

(อ้างอิงจากตัวอย่างในเอกสารบทที่ 11)

- คำใบ้: ใช้ `while True:` และ `break` เมื่อผู้ใช้เลือกเมนูให้ขอฮ.

สรุปประเด็นสำคัญ

- **`while` คือ 'ยามเฝ้าประตู':** ตรวจสอบเงื่อนไขก่อนทำงานเสมอ เหมาะกับการวนซ้ำที่อาจไม่เกิดขึ้นเลยก็ได้
- **`do-while` คือแนวคิด 'ประตูหมุน':** เน้นการทำงานอย่างน้อยหนึ่งครั้งก่อน แล้วจึงตรวจสอบเงื่อนไข
- **Python ใช้ `while True: ... break`:** เพื่อจำลองการทำงานแบบ `do-while` ได้อย่างมีประสิทธิภาพและชัดเจน ซึ่งเป็นรูปแบบที่นักพัฒนา Python นิยมใช้

ภาคผนวก: เฉลยแบบฝึกหัด

เฉลยโจทย์ข้อที่ 1 (Countdown):

```
count = 10
while count > 0:
    print(count)
    count -= 1
print("Liftoff!")
```

เฉลยโจทย์ข้อที่ 2 (Menu):

```
while True:
    print("\n--- MENU ---")
    print("1. แสดงข้อมูล")
    print("2. เพิ่มข้อมูล")
    print("9. ออกจากโปรแกรม")

    choice = input("กรุณาเลือกเมนู: ")
    if choice == '1':
        print("...กำลังแสดงข้อมูล...")
    if choice == '1':
        print("...กำลังแสดงข้อมูล...")
    elif choice == '2':
        print("...กำลังเพิ่มข้อมูล...")
    elif choice == '9':
        print("...ออกจากโปรแกรม...")
        break
    else:
        print("เมนูไม่ถูกต้อง กรุณาลองใหม่")
```