Title: ตำราวิชา Problem Solving and Computer Programming (PSCP) - PSCP Book

Author: รศ.ดร. โชติพัชร์ ภรณวลัย

Language: th-TH

Date: 22 สิงหาคม 2565

Chapter 3: Conditionals

Boolean Expression

ในบทที่ผ่านมา เราได้เรียนรู้การเขียนโปรแกรมในลักษณะที่เป็นการทำงานตามลำดับที่ได้กำหนดไว้ ลำดับนั้นจะมีแค่ทิศทางเดียว ดังนั้นไม่ว่าค่าของ Input ที่รับเข้ามาจะมีค่าใด ก็จะคำนวณหรือทำงานตามลำดับที่กำหนดไว้แบบเดียวกัน (แต่อาจจะให้ผลลัพธ์ที่แตกต่างกันกันได้ เนื่องจาก input ที่รับมามีค่าแตกต่างกัน)

ในบทนี้เราจะกล่าวถึงโปรแกรมที่ทำงานแบบมีเงื่อนไข เช่น ถ้าค่าของ Input มีค่าตามที่กำหนดไว้แบบนึง ก็จะทำงานแบบนึง แต่ถ้าค่าของ Input มีค่าอีกช่วงหนึ่ง ก็จะทำงานอีกแบบที่ไม่เหมือนกัน

ยกตัวอย่างเช่น ถ้าเราได้กำหนดเกณฑ์การสอบผ่านไว้ที่ตั้งแต่ 60 คะแนน เป็นต้นไป ถ้าน้อยกว่านั้นคือสอบตก โปรแกรมที่เราเขียนจะต้องมีการตรวจสอบค่าคะแนน ก่อนว่า ตรงกับเงื่อนสอบผ่าน หรือสอบตก ถ้าสอบผ่าน (เงื่อนไขการสอบผ่านเป็นจริง) ก็อาจจะให้โปรแกรมพิมพ์คำว่า Pass แต่ถ้าสอบตก (เงื่อนไขการสอบตกเป็นจริง) ให้โปรแกรมพิมพ์คำว่า Fail เป็นต้น

นิพจน์แบบบูลลีน (Boolean Expression) คือนิพจน์ที่ให้ผลออกมาเป็นจริง `True` หรือ เท็จ `False` และใช้ในการตรวจสอบเงื่อนไข ดังตัวอย่างในรูปด้านล่าง

```
>>> print(1==1)
True
>>> print(1!=1)
False
>>> print(1>2)
False
>>> print(0<1)
True
>>> print(1>=1)
True
>>> print(1<=0)
False
```

ตารางด้านล่างแสดงเครื่องหมาย Relational operator ที่ใช้ในการเปรียบเทียบ

Relational operator	description	example
==	เท่ากับ	5 == 5 # ได้ผลลัพธ์เป็น True เพราะ 5 เท่ากับ 5 จริง
!=	ไม่เท่ากับ	10 != 5 # ได้ผลลัพธ์เป็น True เพราะ 10 ไม่เท่ากับ 5

Relational operator	description	example
>	มากกว่า	5 > 10 # ได้ผลลัพธ์เป็น False เพราะ 5 น้อยกว่า 10
<	น้อยกว่า	5 < 10 # 5 ได้ผลลัพธ์เป็น True เพราะ 5 น้อยกว่า 10 จริง
>=	มากกว่าหรือเท่ากับ	5 >= 5 # ได้ผลลัพธ์เป็น True เพราะ 5 มากกว่าหรือเท่ากับ 5 จริง
<=	น้อยกว่าหรือเท่ากับ	-2 <= -3 # ได้ผลลัพธ์เป็น False เพราะ -2 มีค่ามากกว่า -3

ค่า True หรือ False ไม่ใช่ข้อมูลชนิด string แต่เป็นข้อมูลชนิดที่เรียกว่า bool ดังจะสังเกตได้จากการใช้คำสั่ง type ดังรูปด้านล่าง

ค่า True หรือ False จะต้องเขียนโดยตัวอักษรตัวแรกต้องเป็นตัวพิมพ์ใหญ่เท่านั้น ไม่เช่นนั้นจะมี NameError (ถ้าไม่ได้เคยกำหนดตัวแปรชื่อนั้นมาก่อนดังตัวอย่างด้านล่าง) และหากใช้ single quote หรือ double quote ล้อมรอบ ก็จะเป็นข้อมูลชนิด String

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
>>> type(true)
Traceback (most recent call last):
  File "<pyshell#25>", line 1, in <module>
    type(true)
NameError: name 'true' is not defined
>>> type('False')
<class 'str'>
```

Logical Operators

ในทางตรรกศาสตร์ เราสามารถใช้ operator เช่น `and` `or` `not` ในการตรวจสอบเงื่อนไขได้ เช่น `x > 0 and y > 5` จะหมายความว่า `x > 0` ต้องเป็นจริง และ `y > 0` ต้องเป็นจริง จึงจะทำให้นิพจน์ `x > 0 and y > 5` เป็นจริง

Truth tables หรือ ตารางความจริง ของ operator `and` `or` `not` เป็นไปตามตางรางด้านล่างนี้

ตารางความจริง ของ `and`

x	y	x and y
False	False	False
False	True	False
True	False	False
True	True	True

ตารางความจริง ของ `or`

x	y	x or y
False	False	False

x	y	x or y
False	True	True
True	False	True
True	True	True

ตารางความจริง ของ `not`

x	not x
True	False
False	True

Conditional Execution

ในภาษา Python เราสามารถใช้ `if` ในการตรวจสอบเงื่อนไขนิพจน์ Boolean และถ้าเงื่อนไขเป็นจริง ก็จะให้โปรแกรมทำงานบางอย่างเพิ่มเติมเข้าไป

ยกตัวอย่างเช่น สมมุติว่าเราจะการสร้าง function เพื่อ พิมพ์ข้อความว่า 'Good' ถ้าคะแนนสอบมากกว่าหรือเท่ากับ 80 คะแนน และแสดงคะแนนสอบในบรรทัดต่อไป สำหรับคะแนนสอบที่ไม่ถึง 80 คะแนน ก็เพียงแต่แสดงคะแนนอย่างเดียว

```
def show_score_record(score):
    if score >= 80:
        print('Good')
    print(score)
```

สังเกตว่า `if` จะมีลักษณะโครงสร้างเหมือนกับ `def` คือจะลงท้ายด้วย `:` (semicolon) เหมือนกัน และเป็น statement แบบที่เรียกว่า `compound statement` โดยบรรทัดที่เขียน `if` จะเรียกว่า `header` และบรรทัดอื่นๆด้านล่างที่ถูกเว้นวรรคเข้ามา 4 ช่องว่าง จะเรียกว่า `body` ในตัวอย่างด้านบน `print('Good')` คือ `body` ของ statement `if`

Alternative Execution

ในกรณีที่มี 2 ทางเลือก เราสามารถใช้ `if` และ `else` ได้ดังตัวอย่างด้านล่างนี้

```
if x%2 == 0:
    print('Even')
else:
    print('Odd')
```

บรรทัด `else` ก็ต้องมี colon `:` ปิดท้าย และส่วนของ `body` ของ `else` จะต้องขยับเข้าไป (indent) ทางด้านขวาจำนวน 4 ช่อง (4 spaces) เช่นเดียวกับ `body` ของ `if`

ถ้า `x` เป็นจำนวนเต็มที่ ถูก mod (`%`) ด้วย 2 แล้วมีค่าเท่ากับ 0 แสดงว่าการหาร `x` ด้วย 2 ไม่มีเศษเหลือ ดังนั้น `x` ก็จะเป็นเลขคู่ และพิมพ์คำว่า 'Even' แต่ถ้าไม่ใช่ (แสดงว่าเงื่อนไขใน `if` ไม่เป็นจริง) ก็จะทำงานในส่วนของ `else` แทน คือการพิมพ์คำว่า `Odd`

ในตัวอย่างด้านบน เราสามารถเขียนโปรแกรมที่ให้ผลลัพธ์แบบเดียวกัน โดยไม่ใช้ `else` ได้เช่นกัน กล่าวคือใช้เพียง `conditional expression` เท่านั้น ดังตัวอย่างด้านล่าง

```
message = 'Odd'
if x%2 == 0:
    message = 'Even'
print(message)
```

จากรูปด้านบน จะเห็นว่า เราสร้างตัวแปร `message` เพื่อเก็บข้อความ `Odd` ไว้ก่อน ถ้าหากว่าเงื่อนไขใน `if` เป็นจริง ก็ให้เปลี่ยน `message` จาก `'Odd'` เป็น `Even` หลังจากนั้นเมื่อออกจาก `body` ของ `if` แล้ว ก็ให้พิมพ์ข้อความใน `message` นั้น ซึ่งอาจจะเป็น `Odd` หรือ `Even` ก็ได้ แล้วแต่เงื่อนไขใน `if` เป็นจริงหรือเท็จ

ถึงแม้ว่า 2 โปรแกรมด้านบนจะให้ผลลัพธ์ที่เหมือนกัน แต่การเขียนโดยใช้ `Alternative execution` ด้วย `else` จะอ่านและทำความเข้าใจโปรแกรมได้ง่ายกว่า

มี `builtin function` ชื่อ `bool` ซึ่งจะรับค่าเข้ามาเพื่อตรวจสอบว่าเป็น `True` หรือ `False` ได้ ดังตัวอย่างด้านล่าง

```
>>> bool(True)
True
>>> bool(False)
False
>>> bool(1)
True
>>> bool(-1)
True
>>> bool(1.23)
True
>>> bool(0)
False
```

จากตัวอย่างด้านบน จะเห็นได้ว่า ค่าตัวเลขที่ไม่ใช่ 0 จะมีค่าเป็น `True` เสมอ และตัวเลข 0 มีค่าเป็น `False`

ดังนั้นหากต้องการเขียนโปรแกรมตรวจสอบเงื่อนไขว่า `x` เป็นเลขคู่หรือเลขคี่ สามารถเขียนได้อีกแบบ ดังรูปด้านล่าง

```
if x%2:
    print('Odd')
else:
    print('Even')
```

ในตัวอย่างตามรูปด้านบน จะเห็นว่าไม่ต้องใช้ Relational operation `==` เพื่อทำการเปรียบเทียบ แต่สามารถใช้ผลลัพธ์ของการ mod `%` ได้เลย ซึ่งการด้วย mod ด้วย 2 จะให้ผลลัพธ์เป็น 1 หรือ 0 เท่านั้น

หากได้ผลลัพธ์เป็น 1 หรือการหารด้วย 2 เหลือเศษ 1 แสดงว่า `x` เป็นเลขคี่ ก็จะทำให้เงื่อนไข `if` เป็นจริง (`True`) และพิมพ์ `Odd` ออกทางหน้าจอ และหากหารด้วย 2 เหลือเศษ 0 ค่า 0 จะถูกพิจารณาว่าเป็น False และเงื่อนไขใน `if` จะเป็นเท็จ (`False`) และพิมพ์คำว่า `Even` ซึ่งเป็น body ของ `else` แทน

Exercise 1 (Grade I)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `Grade I` โดยให้เขียนแบบ `Alternative execution`

Exercise 2 (Distinguish)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `Distinguish` โดยให้เขียนแบบ `Conditional execution`

Exercise 3 (PlanB)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `PlanB` โดยให้เขียนแบบ `Conditional execution` หรือ `Alternative execution` ก็ได้

Exercise 4 (Timing II)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `Timing II`

Exercise 5 (ODD_EVEN)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `ODD_EVEN` โดยให้สร้าง fruitful function ชื่อ `is_odd` และรับค่า Integer เข้ามา 1 ค่า ถ้าค่า Integer นั้นมีค่าเป็นเลขคี่ ให้คืนค่า boolean `True` กลับไป และถ้าไม่ใช่เลขคี่ ให้คืนค่า boolean `False` กลับไป

การสร้าง function ที่มีการคืนค่าเป็น `True` หรือ `False` มักจะมีการตั้งชื่อ function นั้น เป็นลักษณะคำถาม เช่น `is_odd` เป็นต้น

Chained Conditionals

ถ้ามีเงื่อนไขมากกว่า 2 กรณี สามารถใช้ chained conditionals โดยใช้ `elif` ซึ่งย่อมาจากคำว่า `else if` และตามด้วยเงื่อนไข (boolean expression) และจบบรรทัดด้วย colon `:`

```
if x < y:
    print('x is less than y')
elif x > y:
    print('x is greater than y')
else:
    print('x and y are equal')
```

ตัวอย่างในรูปด้านบน เราสามารถเปรียบเทียบว่า `x น้อยกว่า y` หรือ `x มากกว่า y` หรือ `x เท่ากับ y` โดยใช้ 3 เงื่อนไข เงื่อนไขแรกใช้ `if` เงื่อนไขที่สองใช้ `elif` และเงื่อนไขสุดท้ายใช้ `else`

การใช้ `chained conditionals` ไม่จำเป็นต้องจบด้วย `else` ก็ได้ ดังนั้นในบรรทัดรองสุดท้ายในรูปด้านบน สามารถเปลี่ยนจาก `else:` เป็น `elif x == y:` ก็ได้เช่นกัน

Nested Conditionals

เราสามารถเขียนให้เงื่อนไขหนึ่ง ซ้อนอยู่ภายใต้อีกเงื่อนไขหนึ่งได้ กล่าวคือให้เงื่อนไขหนึ่งเป็น `body` ของอีกเงื่อนไขหนึ่ง

เราจะเรียกการเขียนแบบนี้ว่า `Nested conditionals` ดังตัวอย่างด้านล่าง

```
if x == y:
    print('x and y are equal')
else:
    if x < y:
        print('x is less than y')
    else:
        print('x is greater than y')
```

ในตัวอย่างนี้จะเห็นว่า `if x < y:` เป็นส่วนนึงของ body ของ `else` ในบรรทัดก่อนหน้า

โปรแกรมในตัวอย่างนี้จะให้ผลเหมือนกับการเขียนด้วย `Chained conditionals` ในตัวอย่างที่แล้ว แต่จะทำให้โปรแกรมอ่านเข้าใจยากกว่าการใช้ `chained conditionals` และการไม่ใช้ `elif` ทำให้มีการ indent หรือขยับ `body`เข้าไปทางด้านขวามากกว่าแบบ `Chained conditionals` ทำให้เข้าใจยากขึ้นกว่าเดิม

อีกตัวอย่าง หากเราต้องการตรวจสอบว่าตัวเลข `x` ซึ่งเป็น Integer เป็นตัวเลขที่มีเพียงหลักเดียวที่มีค่าเป็นบวกหรือไม่ (1-9) สามารถเขียนได้ 3 วิธี (ดังรูปประกอบด้านล่าง)

วิธีที่ 1 ใช้`Nested conditionals`

วิธีที่ 2 ใช้ logical operator `and` แทนการใช้ `if` ซ้อนกัน

วิธีที่ 3 เป็นการเขียนที่ไม่ใช้จำเป็นต้องใช้ logical operator `and` เลย

ทั้ง 3 วิธีนี้ให้ผลลัพธ์ที่เหมือนกัน แต่วิธีที่ 1 จะอ่านได้ยากกว่า ดังนั้นการใช้`Nested conditionals` ควรใช้เมื่อจำเป็นเท่านั้น

```
#1
if 0 < x:
    if x < 10:
        print('x is a positive single-digit number')

#2
if 0 < x and x < 10:
    print('x is a positive single-digit number')

#3
if 0 < x < 10:
    print('x is a positive single-digit number')
```

Exercise 6 (Grade II)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `Grade II`

Exercise 7 (Quadrant)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `Quadrant`

Exercise 8 (Robot I)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `Robot I`

Exercise 9 (Squareroot)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `Squareroot`

Exercise 10 (PointInCircle)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `PointInCircle`

Exercise 11 (BMIAgain)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `BMIAgain`

Exercise 12 (Gift II)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `Gift II`

Exercise 13 (DataSpike)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `DataSpike`

Exercise 14 (FoodGrade I)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `FoodGrade I`

Exercise 15 (Seven)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `Seven`

Exercise 16 (Day I)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `Day I`

Exercise 17 (Circular I)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `Circular I`

Exercise 18 (Circular II)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `Circular II`

Exercise 19 (PlanCDEFGHIJKLMNOPQRSTUVWXYZ)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `PlanCDEFGHIJKLMNOPQRSTUVWXYZ`

Exercise 20 (WeightStation)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `WeightStation`

Exercise 21 (Triangle I)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `Triangle I`

Exercise 22 (SurprisingVote)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `SurprisingVote`

Exercise 23 ([Midterm] Donut)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `[Midterm] Donut`

Exercise 24 ([Midterm] LargestNumber)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `LargestNumber`

Exercise 25 (BrickBridge)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `BrickBridge`