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

Author: รศ.ดร. โชติพัชร์ ภรณวลัย

Language: th-TH

Date: 10 กรกฎาคม 2567

Chapter 4: Iterations

Iteration with function

ในบทที่ผ่านมาหากเราได้เรียนรู้ว่า หากงานใดๆที่ต้องทำซ้ำๆ เราจะสร้าง function ขึ้นมา และเรียก function นั้นแทน ดังแสดงในตัวอย่างด้านล่าง

```
# no function
print('Hello', 'Alice')
print('Nice to meet you')
print('Hello', 'Bob')
print('Nice to meet you')
print('Hello', 'Caren')
print('Nice to meet you')

# with function
def hello(name):
    print('Hello', name)
    print('Nice to meet you')

hello('Alice')
hello('Bob')
hello('Caren')
```

ทั้ง 2 วิธีจะเห็นได้ว่า ได้ผลเหมือนกัน แต่การเขียนแบบ function จะทำให้โปรแกรมเข้าใจได้ง่ายกว่า และมีจำนวนบรรทัดน้อยกว่า โดยเฉพาอย่างยิ่งหากเราต้องการเขียนโปรแกรมเพื่อทักทายคนจำนวน 100 คน ถ้าหากว่าเราไม่เขียน function เราจะต้องเขียนทั้งหมด 200 บรรทัด และจะมี 100 บรรทัดที่ซ้ำกันคือบรรทัด

```
print('Nice to meet you')
```

แต่หากเราเขียนโดยการใช้ function เราจะใช้จำนวนบรรทัดประมาณ 103 บรรทัด 100 บรรทัดใช้การเรียก `hello` function และอีก 3 บรรทัดใช้ในการนิยาม `def` function hello

Iteration with For Loop

ในบทนี้เราจะเรียนรู้การให้โปรแกรมทำงานซ้ำๆ (`Iteration`) หรือที่เรียกอีกชื่อว่า `loop`ได้โดยใช้ **for** และ **while**

ใน section นี้จะเริ่มที่ **for** loop ก่อน

ยกตัวอย่างเช่น หากเราต้องการจะ `hello Alice` 5 ครั้ง สามารถทำได้ 2 วิธี ดังนี้

```
# Call hello('Alice') 5 times
hello('Alice')
hello('Alice')
hello('Alice')
hello('Alice')
hello('Alice')

# for loop
for i in range(5):
    hello('Alice')
```

จะเห็นได้ว่าการเขียนโปรแกรมด้วย **for** จะทำให้โปรแกรมสั้นลงได้ จาก 5 บรรทัด เหลือ 2 บรรทัด

ข้อควรระวัง หากส่งโปรแกรมในรูปด้านบนเข้าบน eJudge จะโดนหักคะแนน Quality เนื่องจากตัวแปร *i* ใน for loop ไม่ได้นำใช้งานใน body ของ for loop วิธีแก้คือให้แก้ชื่อตัวแปรที่ไม่ต้องการใช้ใน body ของ for loop ให้เป็น _ หรือเครื่องหมาย underscore แทน ดังในรูปด้านล่าง

```
for _ in range(5):
    hello('Alice')
```

ในที่นี้ `range` เป็นชื่อของ `function` ที่จะคืน (return) ค่าของชุดลำดับของตัวเลข โดยจะมี `parameter` ได้สูงสุด 3 ตัว ได้แก่ `start`, `stop` และ `step`

```
>>> help(range)
Help on class range in module builtins:

class range(object)
 |  range(stop) -> range object
 |  range(start, stop[, step]) -> range object
 |
 |  Return an object that produces a sequence of integers from start (inclusive)
 |  to stop (exclusive) by step.  range(i, j) produces i, i+1, i+2, ..., j-1.
 |  start defaults to 0, and stop is omitted!  range(4) produces 0, 1, 2, 3.
 |  These are exactly the valid indices for a list of 4 elements.
 |  When step is given, it specifies the increment (or decrement).
 |
```

จาก `help` ด้านบน หมายความว่า หากมี `parameter` 1 ตัว `parameter` นั้นคือ `stop` หากมี `parameter` 2 ตัว จะเป็น `start` และ `stop` และหากมี 3 ตัวจะเป็น `start` `stop` และ `step` ตามลำดับ

```
>>> range(5)
range(0, 5)
>>> i = range(5)
>>> i
range(0, 5)
>>> type(i)
<class 'range'>
>>> for i in range(5):
        print(i)


0
1
2
3
4
```

จากรูปด้านบนจะเห็นว่า `range(5)` หมายความว่า `stop` เท่ากับ 5 และ `range(5)` มีค่าเท่ากับ`range(0, 5)` ซึ่งหมายความว่า `start` มีค่าเท่ากับ 0 และ `stop` มีค่าเท่ากับ 5 กล่าวคือ หากไม่ใส่ค่า `start` เข้าไป (ใส่แต่ค่า `stop`) ค่า `start` จะมีค่าเป็น 0 โดยอัตโนมัติ (ค่าเริ่มต้น)

ชนิดข้อมูลที่ return จาก function `range` จะเป็นชนิด `range`

`range(5)` จะ return ค่าข้อมูลที่สามารถทำซ้ำได้ (เรียกว่า **Iterable**) โดยในที่นี้จะเป็นชุดลำดับของตัวเลขจำนวนเต็มจำนวน 5 ตัว ที่เรียงต่อกัน ตั้งแต่ 0 ถึง 4 ได้แก่ 0, 1, 2, 3, 4 (สังเกตว่าไม่รวมเลข 5)

เมื่อใช้ใน **for** loop ในครั้งแรก `i` จะมีค่าเป็น 0 และจะ `print(0)` และรอบถัดไป `i` จะมีค่าเป็น 1 และ `print(1)` ตามลำดับไปเรื่อยๆจนเมื่อ `print(4)` ก็จะออกจาก loop

รูปด้านล่าง แสดงตัวอย่างกรณีที่ `range()` มี 2 `argument` และ 3 `argument` ตามลำดับ ในตัวอย่างแรก `start` มีค่าเท่ากับ 3 และ `stop` มีค่าเท่ากับ 7 ดังนั้น `i` จะมีค่าเป็น 3, 4, 5, 6 ตามลำดับในแต่ละรอบ

ในตัวอย่างที่ 2 จะมี 3 `argument` โดย `start` มีค่าเท่ากับ 3 และ `stop` มีค่าเท่ากับ 11 และ `step` มีค่าเท่ากับ 2 ในตัวอย่างนี้ `i` จะเริ่มจากค่า 3 ไปทีละ 2 (ค่าของ `step`) กลายเป็น 5 และ 7 และ 9 แต่ไม่สามารถไปเป็น 11 ได้ เนื่องจาก `stop` เป็น 11 (ค่าสูงสุดเมื่อ `stop` เป็น 11 คือ 10 เท่านั้น)

จะสังเกตว่า กรณีที่ `step` ถ้าไม่กำหนดไว้ (กรณี 2 `argument` ในตัวอย่างแรก) ค่า `step` จะมีค่าเริ่มต้นเป็น 1 เสมอ ดังนั้นลำดับที่ได้จะเพิ่มทีละ 1 เสมอ จาก 3 เป็น 4 เป็น 5 และ 6 ตามลำดับ

```
>>> for i in range(3, 7):
        print(i)


3
4
5
6
>>> for i in range(3, 11, 2):
        print(i)


3
5
7
9
```

step สามารถมีค่าเป็นลบ เช่น -1, -2, ...

```
>>> for i in range(7, 2, -1):
        print(i)


7
6
5
4
3
>>> for i in range(7, 2, -2):
        print(i)


7
5
3
>>>
```

เรามักจะใช้ **for** loop เมื่อเรารู้จำนวนรอบที่ต้องการทำแน่นอน ก่อนจะเริ่มเข้า **for** loop

แต่หากเราไม่รู้จำนวนรอบที่แน่นอนก่อนเริ่มทำงาน เรามักจะใช้ **while** loop ดังจะได้อธิบายในหัวข้อถัดไป

Iteration with while Loop

อีกวิธีนึงของการทำงานซ้ำๆคือใช้ **while** loop

ตัวอย่างของโปรแกรมที่ต้องการ `hello('Alice')` 5 ครั้ง และ `print` 0 ถึง 4 ในแต่ละบรรทัด เหมือนกับตัวอย่างที่แสดงใน **for** loop ด้านบน แต่โดยการเขียนด้วย **while** loop จะเป็นดังรูปด้านล่าง

```
>>> i = 0
>>> # while loop to hello('Alice') 5 times
while i < 5:
    hello('Alice')
    i = i + 1


Hello Alice
Nice to meet you
Hello Alice
Nice to meet you
Hello Alice
Nice to meet you
Hello Alice
Nice to meet you
Hello Alice
Nice to meet you
>>>
>>> i = 0
>>> # while loop to print 0 to 4 on each line
while i < 5:
    print(i)
    i = i + 1


0
1
2
3
4
>>>
```

การใช้ **while** จะมีการตรวจสอบ boolean expression (`i < 5`) ด้านหลัง while ก่อน หากเป็นจริง ก็จะทำงานภายใน `body` ของ `while` ซึ่งในตัวอย่างแรกคือการตรวจสอบว่า ค่า `i` น้อยกว่า 5 หรือไม่ ซึ่งค่า `i` ตอนแรกจะมีค่าเป็น 0 ก่อนเริ่มทำการตรวจสอบ ดังนั้น boolean expression นี้มีค่าเป็นจริง และก็ทำการเรียก `hello('Alice')` และเพิ่มค่า `i` อีก 1 จากเดิมมีค่า 0 เป็นค่า 1 เมื่อทำงานจนจบส่วนของ body ของ **while** แล้ว ก็จะมาทำการตรวจสอบเงื่อนไข `i` น้อยกว่า 5 หรือไม่อีกครั้ง ในรอบที่ 2 นี้ `i` มีค่าเป็น 1 แล้ว และเงื่อนไขนี้ยังเป็นจริงอยู่ ก็จะทำงานโดยการ call function `hello('Alice')` อีกครั้ง และปรับค่าเป็น 2 และไปตรวจสอบเงื่อนไขอีกครั้ง ไปเรื่อยๆ จนกระทั่งมีค่าเป็น 5 ก็จะทำให้เงื่อนไขเป็นเท็จ และหยุดการทำงานภายใต้ body ของ **while**

ตัวอย่างที่ 2 ในรูปด้านบน ก็มีการทำงานคล้ายกัน แต่ในตัวอย่างนี้จะเป็นการค่า `i` ในแต่ละรอบ มา `print`ในแต่ละบรรทัดตั้งแต่ค่า 0 จนถึงค่า 4

ในการใช้ **while** loop โดยทั่วไปจำเป็นจะต้องให้เงื่อนไข boolean expression ในที่สุดมีค่าเป็นเท็จ เพื่อให้ออกจาก **while** loop ได้ ไม่เช่นนั้น จะทำให้เกิด `Infinite loop` หรือ loop แบบอนันต์ กล่าวคือโปรแกรมจะทำงานแต่ภายใน **while** loop นี้ไม่มีวันจบ หรือจนกว่าจะปิดโปรแกรมไป

จะเห็นได้ตัวอย่างข้างบน ค่า `i` จะมีการ update ค่าให้มากขึ้น รอบละ 1 จาก statement `i = i + 1` ซึ่งจะทำให้สุดท้ายแล้ว เงื่อนไข `i < 5` เป็นเท็จในที่สุด และก็จะจบการทำงานของ **while** loop

ดังนั้นหากเราลืมใส่ statement `i = i + 1` ใน **while** loopในตัวอย่างด้านบน จะทำให้เกิด `infinite loop`

หากเราต้องการออกจาก loop ไม่ว่าจะเป็น **for** loop (ออกก่อนจำนวนรอบที่กำหนด) หรือ **while** loop (ออกก่อนเงื่อนไขใน **while** จะเป็นเท็จ) สามารถทำได้เช่นกัน โดยใช้ **break** statement ซึ่งจะได้กล่าวถึงต่อไป

Exercise 1 (Repeater)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `Repeater` โดยใช้ **for** loop และ **while** loop

Exercise 2 (Runner)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `Runner` โดยใช้ **for** loop และ **while** loop

Exercise 3 (Counter)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `Counter` โดยใช้ **for** loop และ **while** loop

Exercise 4 (Stepper I)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `Stepper I` โดยใช้ **for** loop และ **while** loop

Exercise 5 (Stepper II)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `Stepper II` โดยใช้ **for** loop และ **while** loop

Exercise 6 (HideAndSeek)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `HideAndSeek` โดยใช้ **for** loop และ **while** loop

Exercise 7 (GraderMachine)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `GraderMachine` โดยใช้ **for** loop และ **while** loop

Iteration with break

เราสามารถใช้ **while** loop ในกรณีที่เราไม่ทราบว่าจะทำงานทั้งหมดกี่รอบ แต่จะทำไปเรื่อยๆจนกว่าเงื่อนไขจะเป็นเท็จ ดังแสดงในตัวอย่างด้านล่างนี้

```
>>> ## While True
while True:
    x = input()
    if x == '-1':
        break
    print('x =', x)
```

```
1
x = 1
2
x = 2
3
x = 3
-1
```

ในตัวอย่างด้านบนนี้ เงื่อนไขด้านหลัง **while** คือ `True` ดังนั้น ดูเหมือน **while** loop นี้จะเป็น `infinite loop` กล่าวคือ ไม่มีทางออกจาก **while** loop ได้เลย แต่ในโปรแกรมนี้ เนื่องจากมีเงื่อนไข `if` และ `break` ที่จะทำให้หลุดจาก loop ได้อยู่ โปรแกรมนี้จึงไม่เป็น `infinite loop`

เมื่อค่า `x` มีค่าเป็น '-1' ก็จะทำให้เงื่อนไขใน `if` เป็นจริง และก็จะทำการ `break` ซึ่งมีความหมายว่าให้ออกจาก loop ที่ `break` นั้นอยู่ทันที และจะไม่มีการทำงานของโปรแกรมส่วนที่อยู่ด้านล่างของ `if` อีกต่อไป จึงทำให้ไม่มีการ `print('x = ', x)` ในกรณีที่ `x = '-1'`

เราสามารถใช้ **break** กับ **for** loop ได้เช่นกัน

Exercise 8 (SumOfNumber)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `SumOfNumber`

Exercise 9 (HowLong)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `HowLong`

Iteration with continue

หากเราไม่ได้ต้องการออกจาก loop โดยการใช้ break แต่หากไม่ต้องการทำงาน (skip) ส่วนของ body ที่เหลืออยู่ของ loop ในรอบนั้น และต้องการขึ้นรอบใหม่เลยทันที สามารถทำได้โดยใช้ `continue` ดังตัวอย่างด้านล่าง

```
>>> ## While True with break and continue
while True:
    x = input()
    if x == '-1':
        break
    if x == '0':
        continue
    print('x = ', x)

1
x =  1
2
x =  2
3
x =  3
0
4
x =  4
-1
>>>
```

เมื่อเงื่อนไข `x == '0'` เป็นจริง จะทำให้ `continue` ถูกทำงาน และจะเริ่มรอบต่อไปในทันที คือไปรับค่า `input()` ค่าใหม่มาใส่ใน `x` โดยจะไม่มีการแสดงผลค่า `x = 0` และหาก `x == '-1'` มีค่าจะเป็นจริง ก็ทำให้ `break` ทำงาน และออกจาก loop ทันที

เราสามารถใช้ `continue` กับ `for` loop ได้เช่นเดียวกัน

Nested Loop

เราสามารถเขียนให้ loop นึง อยู่ภายใต้ loop อีก loop นึงได้ ดังตัวอย่างด้านล่างนี้

```
>>> num1 = 5
>>> num2 = 7
>>> ## Nested loop
for row in range(num1):
    for col in range(num2):
        print("*", end=" ")
    print()


* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
>>>
```

ในตัวอย่างนี้ `row` จะมีค่าตั้งแต่ 0 ถึง 4 และ `col` จะมีค่าตั้งแต่ 0 ถึง 6

ในรอบแรก เมื่อ `row` มีค่าเป็น 0 ก็จะทำงานใน loop ของ `col` อีก 7 รอบโดย `col` จะมีค่าเป็น 0, 1, 2, 3, 4, 5, 6 ไปเรื่อยๆ โดยผลที่ได้คือการ `print *` และเว้นวรรค ไปเรื่อยๆ จนครบ 7 ครั้ง (0 ถึง 6) ก็จะออกจากรูปด้านใน (loop **for** col) และ `print()` เพื่อขึ้นบรรทัดใหม่ และก็จะเริ่มรอบที่สองของ loop **for** row

ในรอบที่ 2 ของ loop ด้านนอก (loop **for** row) `row` ในรอบนี้จะมีค่าเป็น 1 และก็จะทำงานใน loop **for** col อีก 7 ครั้ง (0-6) โดยการ `print` * และเว้นวรรค ไปเรื่อยๆ จนครบ 7 ครั้ง (0 ถึง 6) ก็จะออกจากรูปด้านใน (loop `col`) และ `print()` เพื่อขึ้นบรรทัดใหม่ เช่นเดียวกับรอบแรก และก็จะเริ่มรอบที่สามของ loop `row`

...

...

เป็นเช่นนี้ไปเรื่อยๆจนกระทั่ง `row` มีค่าเป็น 4 ซึ่งเป็นรอบที่ 5 ของ loop **for** row และ col มีค่าเป็น 6 ซึ่งเป็นรอบที่ 7 ของ loop **for** col ก็จะทำการ `print` * ตัวสุดท้าย และ `print()` เพื่อขึ้นบรรทัดใหม่ และจบการทำงาน

เพื่อให้เข้าใจมากขึ้น ตัวอย่างด้านล่าง แสดงให้เห็นว่า **for** loop ด้านนอก (row) แสดงจำนวนบรรทัด (ค่า 0 ถึง 4 รวม 5 บรรทัด) สังเกตได้จากเมื่อจบการทำงานในแต่ละรอบของ **for** row จะมีการ `print()` เพื่อขึ้นบรรทัดใหม่ทุกครั้ง และในแต่ละรอบของ **for** row จะมีการทำงานของ **for** col อีก 7 ครั้ง (ค่า 0 ถึง 6) โดยในแต่ละครั้งจะให้ `print *` ออกมาเท่ากับจำนวน `col+1` ในรอบนั้นๆ

ในรอบแรกของ **for** col ค่า `col` จะมีค่าเป็น 0 ก็จะทำการ print *จำนวน 1 ครั้ง หรือเท่ากับ (col + 1) = 0 + 1 = 1 และในรอบที่ 2 ค่า`col` จะมีค่าเป็น 1 ก็จะ`print *` จำนวน 2 ครั้ง (1 + 1 = 2) เป็นเช่นนี้ไปเรื่อยๆ จนกระทั่ง col = 6 ก็จะ print* จำนวน 7 ครั้ง (6 + 1 = 7)

ในการ `print *` ในแต่ละรอบ จะยังไม่ขึ้นบรรทัดใหม่ (`end=" "`) แต่จะเป็นการเว้นวรรคแทน เมื่อ **for** col ทำงานจนครบแล้ว (7 รอบ) ก็จะออกจาก **for** col เพื่อไปทำงานในบรรทัดต่อไป ซึ่งก็คือการ `print()` เพื่อขึ้นบรรทัดใหม่ และก็เป็นการจบการทำงาน 1 รอบของ **for** row

```
>>> ## Nested loop
for row in range(num1):
    for col in range(num2):
        print("*"*(col+1), end=" ")
    print()


* ** *** **** ***** ****** *******
* ** *** **** ***** ****** *******
* ** *** **** ***** ****** *******
* ** *** **** ***** ****** *******
* ** *** **** ***** ****** *******
>>>
```

ในกรณีที่เป็น Nested loop และใน loop ด้านใน (**for** col) ถูก `break` ออกมา ก็จะหลุดมาทำงาน loop นอก (**for** row) ดังตัวอย่างในรูปด้านล่าง

```
>>> num1 = 5
>>> num2 = 7
>>> ## Nested loop
for row in range(num1):
    for col in range(num2):
        if col == 4:
            break
        print("*"*(col+1), end=" ")
    print()


* ** *** ****
* ** *** ****
* ** *** ****
* ** *** ****
* ** *** ****
>>>
```

จะเห็นได้เมื่อ `col` มีค่าเท่ากับ 4 จะทำให้ loop ใน (**for** col) จบการทำงาน โดยการ `print()` เพื่อขึ้นไปบรรทัดใหม่ และเริ่มรอบใหม่ของ loop นอก (**for** row) ต่อไป ส่งผลให้ไม่มีการแสดงผลเมื่อ `col` มีค่าตั้งแต่ 4 เป็นต้นไป

Exercise 10-22 (BootSequence, Sequence I ถึง Sequence XII)

ให้เรียนลองทดลองทำโจทย์ทั้ง 13 ข้อตั้งแต่ BootSequence, Sequence I ถึง Sequence XII ตามลำดับ

Exercise 23 (Grade III)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `Grade III`

Exercise 24 (FizzBuzz)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `FizzBuzz`

โจทย์ข้อ `FizzBuzz` นี้เป็นโจทย์ง่ายๆที่นิยมใช้เป็นโจทย์ในการสอบสัมภาษณ์เพื่อรับ Programmer เข้าทำงาน แต่จะมีผู้ที่ทำได้ถูกต้องเพียง 0.5% เท่านั้น หรือ 5 ใน 1000 คน เท่านั้น (The “Fizz-Buzz test” is an interview question designed to help filter out the 99.5% of programming job candidates who can’t seem to program their way out of a wet paper bag. [wiki.c2.com]) ขอให้ทุกคนคิดให้รอบคอบก่อนเขียนและพยายามทำให้ผ่านในการส่งครั้งแรก ให้คิดว่านี่คือการสอบสัมภาษณ์ในการสมัครงาน

Exercise 25 (Table I)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `Table I`

Iteration with Function (Recursion)

```
>>> def countdown(num):
...     for i in range(num, 0, -1):
...         print("Count", i)
...     print("Yeh")
...
...
>>> countdown(5)
Count 5
Count 4
Count 3
Count 2
Count 1
Yeh
>>>
```

เราสามารถให้โปรแกรมทำงานซ้ำๆได้ ให้เหมือนกับการใช้ For loop ดังเช่นโปรแกรมด้านบน โดยการเขียนโปรแกรมเรียกตัวเอง ที่เรียกว่า Recursion ดังแสดงในรูปด้านล่าง

```
>>> def countdown_recursion(num):
...     if num == 0:
...         print("Yeh")
...     else:
...         print("Count", num)
...         countdown_recursion(num - 1)
...
...
>>> countdown_recursion(5)
Count 5
Count 4
Count 3
Count 2
Count 1
Yeh
>>>
```

โปรแกรมเรียกตัวเอง หรือ Recursive function จะต้องมี base case หรือเงื่อนไขที่ทำให้การเรียนตัวเองจบลง ไม่เช่นนั้นโปรแกรมก็จะเรียกตัวเองไปเรื่อยๆโดยไม่จบ (อย่างไรก็ตาม ภาษา Python มีการกำหนด Recursion Depth ไว้ที่ 1000 ดังนั้นเมื่อเรียกตัวเองครบ 1000 ครั้ง ก็จะเกิด Exception และหยุดการทำงาน)

ในกรณีของโปรแกรม Recursion ด้านบน base case คือ กรณีที่ n มีค่าเท่ากับ 0 หลังจากนั้นก็จะมาทำงานต่อหลังจากเรียกตัวเองเมื่อ n = 1, n = 2, ... ไปเรื่อยๆจนถึง n = 5 ดังนั้นเมื่อมีการสลับบรรทัด `print("Count", num)` มาไว้ด้านล่าง ดังแสดงในรูปด้านล่าง ก็จะทำให้ผลลัพธ์ที่ได้ สลับกับรูปด้านบน

```
>>> def countup_recursion(num):
...     if num == 0:
...         print("Yeh")
...     else:
...         countup_recursion(num - 1)
...         print("Count", num)
...
...
>>> countup_recursion(5)
Yeh
Count 1
Count 2
Count 3
Count 4
Count 5
>>>
```