w06-Lec

Iterations Part I

Assembled for 204111

by Kittipitch Kuptavanich

The return Statement

- ในฟังก์ชันที่มีการคืนค่า
  - return Statement มีหน้าที่ระบุให้ฟังก์ชันคืนค่า*ทันที* ด้วย ค่าของ Expression ที่ตามหลัง return
- พิจารณาการเขียนฟังก์ชันเพื่อคำนวณพื้นที่วงกลม

```
def area_v1(radius):
    temp = math.pi * radius**2
    return temp
```

VS

```
def area_v2(radius):
    return math.pi * radius**2
```

- ตัวแปร temp ในฟังก์ชันทางซ้าย ช่วยให้เรา debug ได้ง่ายขึ้น (เช่นใช้ฟังก์ชัน print() แสดงค่าที่คำนวณได้ก่อนที่จะ return)

The return Statement [2]

- ในบางกรณี เราอาจมี return Statement มากกว่าหนึ่ง
  - เช่นในแต่ละกิ่ง (Branch) ของ Conditional

```
17 def absolute_value(x):
18     if x < 0:
19         return -x
20     else:
21         return x
```

- สังเกตว่า return Statement อยู่ภายใต้กิ่งที่แยกกันของ Conditionals ในกรณีนี้ จะมี Statement เดียวเท่านั้นที่ถูกดำเนินการ

The return Statement [3]

- เมื่อฟังก์ชันทำงานมาถึง Return Statement ฟังก์ชันจะหยุดดำเนินการ โดยไม่พิจารณา Statement ใด ๆ หลังจากบรรทัดนั้น
- เราเรียก Code หรือ Statement ใด ๆ ในฟังก์ชันหลังจากบรรทัดที่มี return Statement หรือในที่อื่น ๆ ที่จะไม่ถูกดำเนินการในกรณีใด ๆ ว่า Dead Code

```
04 def main():
05     print("line 5")
06     print("line 6")
07
08     return
09
10     print("line 10")
11     print("line 11")
12
```

Dead Code

The return Statement [4]

- ในฟังก์ชันที่มีการคืนค่าผลลัพธ์ (Fruitful Function) ควรมีการตรวจสอบให้แน่ใจว่า ทุกกิ่ง (Branch) หรือ Path ภายในฟังก์ชัน จบที่ return Statement
  - พิจารณาฟังก์ชัน

```
13 def absolute_value(x):
14     if x < 0:
15         return -x
16     if x > 0:
17         return x
```

- ฟังก์ชันด้านบน ทำงานไม่ถูกต้อง เนื่องจากหาก x มีค่าเป็น 0 ฟังก์ชันจะจบการทำงานโดยไม่ผ่าน return Statement ทั้ง 2 จุด
  - ดังนั้นเมื่อทำงานจบฟังก์ชันจะคืนค่า None (None เป็น ค่าที่ถูก return โดย Default ของทุกฟังก์ชัน) ทั้งที่คำตอบควรเป็น 0

ITERATIONS

Basic Program Instructions

A few basic instructions appear in just about every language:

- Input
- Output
- Math
- Conditional Execution
- **Repetition**

Iteration

- หรือ Repetition
- Repeating identical or similar tasks without making errors is something that computers do well and people do poorly.

  การทำงานที่เหมือนกันหรือคล้ายคลึงซ้ำ ๆ อย่างไม่มีข้อผิดพลาดเป็นสิ่งที่ Computer ทำได้ดีกว่ามนุษย์

Types of Iteration

- **Counter-Controlled Loops**
  - **Loop ที่ทำการวนซ้ำตามจำนวนครั้งที่กำหนด**
- **Condition-Controlled Loops**
  - **Loop ที่ทำการวนซ้ำจนกว่าเงื่อนไขที่กำหนดจะเป็นเท็จ**

Simple Iteration – for Loop

- **เราใช้คำสั่ง `for` เพื่อระบุการทำซ้ำ ในกรณีที่เรา*ทราบจำนวนครั้ง*แน่นอน เช่น**

```
>>> for i in range(5):
        print("Hello")

Hello
Hello
Hello
Hello
Hello
>>>
```

- **ตัวเลข 5 ใน function `range()` คือจำนวนครั้งที่ทำซ้ำ**

for Loop – Basic Form

```
for loopVariable in range(n):
□□□□loopBody
```

- ***loopVariable*** **เป็นตัวแปรที่ใช้เพื่อการระบุรอบที่ดำเนินการในปัจจุบันว่าเป็นรอบที่เท่าไร โดยมากใช้ตัว *i* (ย่อมาจาก iterator)**
- ***n*** **คือ*จำนวนครั้ง*ที่ต้องทำซ้ำทั้งหมด (หาก $n \leq 0$ loop นี้จะถูกข้ามไป**
- ***loopBody*** **คือชุดคำสั่งที่ต้องทำซ้ำ มีอย่างน้อย 1 บรรทัด**
- **เราเรียกการทำซ้ำแบบนี้ว่า loop เนื่องจากเมื่อดำเนินการในชุดคำสั่งตาม *loopBody* จนถึงบรรทัดสุดท้ายแล้ว ก็จะวนไปดำเนินการที่ชุดคำสั่งในบรรทัดแรกของ *loopBody* อีกจนกว่าจะครบจำนวนครั้งที่ระบุ (*n*)**

for Loop – Basic Form [2]

- **พิจารณาค่าของ iterator**

```
>>> for i in range(5):
        print("i = {0:d}".format(i))
i = 0
i = 1
i = 2
i = 3
i = 4
```

ไม่ถึง และ ไม่เกิน ***n***

- ***i*** **มีค่าจาก 0 ถึง *n* - 1**

for Loop – range()

- range() เป็นชนิดข้อมูลชนิดหนึ่ง ของภาษา python
- หากใส่ argument ตัวเดียว จะเป็นการระบุจุดสิ้นสุดของ range คืออยู่ในรูป range(*stop*)
  - range(5) จะเป็นตัวเลขระหว่าง 0 – 4
- เราสามารถระบุจุดเริ่มต้นนอกเหนือจาก 0 ได้ในรูป

```
range(start, stop)      # start และ stop เป็น int
```

```
>>> for i in range(18, 21):
        print("i = {0:d}".format(i))
i = 18
i = 19
i = 20
```

เนื่องจากเป็นการเพิ่มค่าทีละ 1
stop ต้องมากกว่า start

for Loop – range() [2]

- โดยปกติแล้ว range() จะเริ่มที่ค่า start แล้วเพิ่มค่าทีละ 1 ในแต่ละขั้น (step)
- เราสามารถระบุความกว้างของขั้นได้ ในรูปของ

```
range(start, stop, step)      # all integers
```

```
>>> for i in range(8, 20, 3):
        print("i = {0:d}".format(i))
i = 8
i = 11
i = 14
i = 17
```


for Loop – range() [3]

- เราสามารถระบุ step ให้มีค่าเป็นลบ เพื่อให้ range มีลักษณะเรียงจาก*มากไปน้อย*ได้

```
>>> for i in range(8, 2, -2):
        print("i = {0:d}".format(i))
i = 8
i = 6
i = 4
```

- ผลลัพธ์ของ loop ด้านล่างคืออะไร

```
>>> for i in range(2, 8, -2):
        print("i = {0:d}".format(i))
_____
_____
_____
```


Accumulator Loop

- พิจารณาฟังก์ชัน sum_1_to_n() เพื่อหาผลบวกตัวเลขตั้งแต่ 1 ถึง n
  - เช่น n = 3 จะได้ค่าผลลัพธ์ = 1 + 2 + 3 = 6

```
05 def sum_1_to_n(n):
06     result = 0
07
08     for i in range(_________):
09         result = result + i
10
11     return result
```

- ในแต่ละรอบของ loop ค่า i จะได้รับการเพิ่มเข้าไปที่ result
  - เราเรียกตัวแปรในลักษณะเดียวกันกับ result ว่า Accumulator
  - ในบรรทัดที่ 09 สามารถเขียนในรูป result += i
    - เรียก Statement ในลักษณะนี้ว่า *Augmented* Assignment Statement

Accumulator Loop [2]

- เขียนฟังก์ชัน `factorial_n()` เพื่อหาค่า $n!$ โดยใช้ for loop และ accumulator variable

```
def factorial_n(n):
    result = _________
    for i in range(_________):
        result = ____________________

    return result

assert(factorial_n(5) == 5 * 4 * 3 * 2 * 1))
```


Loop Variable

ข้อควรระวัง: ไม่ควร reassign ค่า หรือเปลี่ยนค่าของ loop variable (iterator)

```
>>> for i in range(5):
        print(i, end=" ")
        i = 3
0 1 2 3 4

>>> for i in range(5):
        i = 3
        print(i, end=" ")
3 3 3 3 3
```

- หากมีการ reassign ค่าของ loop variable ค่าของ variable นั้น ๆ จะถูก reset ให้เป็นค่าที่ถูกก่อนวน loop ครั้งถัดไป

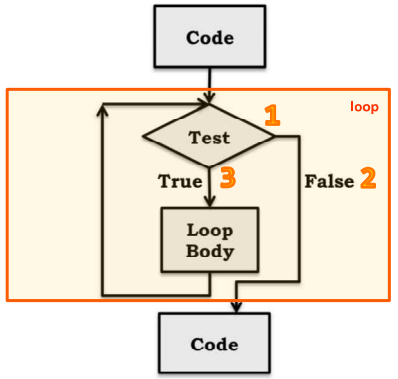
Types of Iteration

- Counter-Controlled Loops
  - Loop ที่ทำการวนซ้ำตามจำนวนครั้งที่กำหนด
- Condition-Controlled Loops
  - Loop ที่ทำการวนซ้ำจนกว่าเงื่อนไขที่กำหนดจะเป็นเท็จ

Generic Loop Structure

- Loop structure ทั้งสองชนิดมีลักษณะคล้ายคลึงกันดังนี้

1. Test – ได้ผลลัพธ์เป็น True หรือ False
2. ถ้าผลลัพธ์เป็น False - ออกจาก loop
3. ถ้าผลลัพธ์เป็น True. ดำเนินการชุดคำสั่งใน Loop Body หนึ่งครั้ง แล้วกลับไปที่ข้อ 1



while loop

Initially sets the condition

Conditions

Repetition Statement

Alters the condition

```
01 def countdown(n):
02     while n > 0:
03         print(n)
04         n = n - 1
05
06     print('Blastoff!')
```

- การทำงานของฟังก์ชันสามารถพิจารณาได้เหมือนการตีความประโยคภาษาอังกฤษปกติ
  - "ในขณะที่ n (ยัง) มากกว่า 0 แสดงค่า n แล้วลดค่า n ลง 1"
- ชุดคำสั่งในส่วน Loop Body ควรมีการเปลี่ยนแปลงค่าตัวแปรเพื่อที่จะส่งผลให้ Boolean Expression มีค่าเป็น False ในที่สุด
  - เพื่อที่ loop จะได้หยุดการทำงาน

```
while Boolean_expression:
    LoopBody
```


while loop [2]

- ในการออกแบบ algorithm Loop ทุก loop จะต้องหยุดการทำงาน (terminate) ในที่สุด
- Loop ที่ทำงานไปเรื่อยโดยไม่หยุดเรียกว่า infinite loop
- Classic Example ของ infinite loop
  - วิธีใช้ shampoo
    - Lather, Rise, Repeat

while loop [3]

ในกรณีใดบ้างที่โปรแกรม จะ terminate?

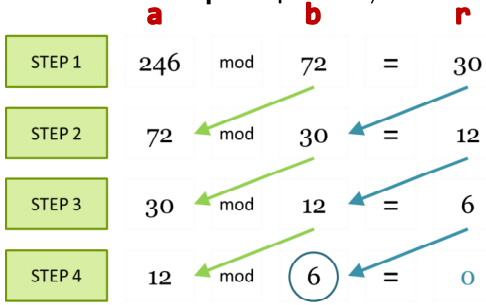
```
06 x = 3
07 ans = 0
08 itersLeft = x
09
10 while (itersLeft != 0):        # Square an integer, the hard way
11     ans = ans + x
12     itersLeft = itersLeft - 1
13
14 print(str(x) + '*' + str(x) + ' = ' + str(ans))
```

- เราสามารถจำลองการ run ของ loop ได้โดยการเขียน

test#	x	ans	itersLeft
1	3	0	3
2	3	3	2
3	_____	_____	_____
4	_____	_____	_____

Example 1: The Euclidean Algorithm

- For example 246 and 72




```
01 def gcd(a, b):
02     r = _____
03     while _____:
04         _____
05         _____
06         _____
07     return _____
```

- เราจะเปลี่ยน algorithm นี้เป็น loop ได้อย่างไร?
  - ตั้งชื่อให้แต่ละ column (นี่คือชื่อ variable)
  - Loop terminate เมื่อไร _____
  - ผลลัพธ์ที่ต้องการอยู่ใน variable ชื่ออะไร _____

Example 2: Number Guessing

- **Problem Statement:**
  - ต้องการเขียนโปรแกรมเพื่อให้ user เล่นเกมทายเลขระหว่าง 1 – 20 โดยจะทายได้ทั้งหมด 5 ครั้ง หากเลขที่ทายต่ำไป หรือสูงไปจะมีคำใบ้บอก

```
$ python number_guess.py
Input number: 3
3 is too low
Input number: 7
7 is too high
Input number: 6
6 is correct!
```


Example 2: Number Guessing [2]

- เนื่องจากจำนวนครั้งที่วน loop มีค่าคงที่คือไม่เกิน 5
  - พิจารณาใช้ for loop

```
05 def guess_num():
06     key = 6
07     for i in range(5):
08         n = int(input("Input number: "))
09         if n == key:
10             print(n, "is correct")
11             ____________________
12         elif n > key:
13             print(n, "is too high")
14         else:
15             print(n, "is too low")
```

- กรณีทายถูก (บรรทัดที่ 09) โปรแกรมจะต้องออกจาก loop
  - ใช้คำสั่ง `break`

The `break` Statement

- เราสามารถใช้คำสั่ง `break` เพื่อออกจาก loop (ชั้นปัจจุบัน) ได้ตามเงื่อนไขที่ระบุ โดยคำสั่งอื่น ๆ ภายใน loop หลังจาก `break` จะถูกข้ามไป
- ใช้ได้กับคำสั่ง `for`, `while`

Example 3: Score Average

- **Problem Statement:** ต้องการเขียนฟังก์ชันเพื่อรับคะแนน ระหว่าง 0 – 100 ของ นักเรียน 30 คนเพื่อหาค่าเฉลี่ย โดยไม่พิจารณาคะแนนในช่วงที่ไม่ถูกต้อง (น้อยกว่า 0 หรือมากกว่า 100)

ทำไมในกรณีนี้ จึงไม่ควรใช้ for loop?

```
09 def score_average():
10     total_count = 30
11     score_count = 0
12     total = 0
13
14     while score_count < total_count:
15         score = float(input("Enter score: "))
16
17         if score < 0 or score > 100:
18             ____________________
19
20         total = total + score
21         score_count = score_count + 1      # += 1
22
23     return total / score_count
```

The `continue` Statement

- คำสั่ง `continue` ใช้ได้กับ loop เท่านั้นโดยจะข้ามคำสั่งที่เหลือภายใน loop หลังจากคำสั่ง `continue` เพื่อไปวน loop ในรอบถัดไป
- ใช้ได้กับคำสั่ง `for`, `while`

References

- Guttag, John V. ***Introduction to Computation and Programming Using Python, Revised***