

w11-Lec2

# Iterations

## Part I

for 204111

Kittipitch Kuptavanich  
& Areerat Trongratsameethong

# 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 ในกรณีที่เรารู้ทราบ  
จำนวนครั้งแน่นอน 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 จำนวนครั้งที่ต้องทำซ้ำทั้งหมด (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 ที่สร้างโดยใช้ฟังก์ชัน **range()**
- 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 มีลักษณะเรียงจากมากไปน้อยได้

```
>>> 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))

    2
    0
   -2
```

# 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(1, n):
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

```

02 def factorial(n):
03     result = 1
04     for i in range(5):
05         result = (n-i)*n
06
07     return result
08
09
10 assert factorial(5) == 5 * 4 * 3 * 2 * 1
11

```

# 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
```

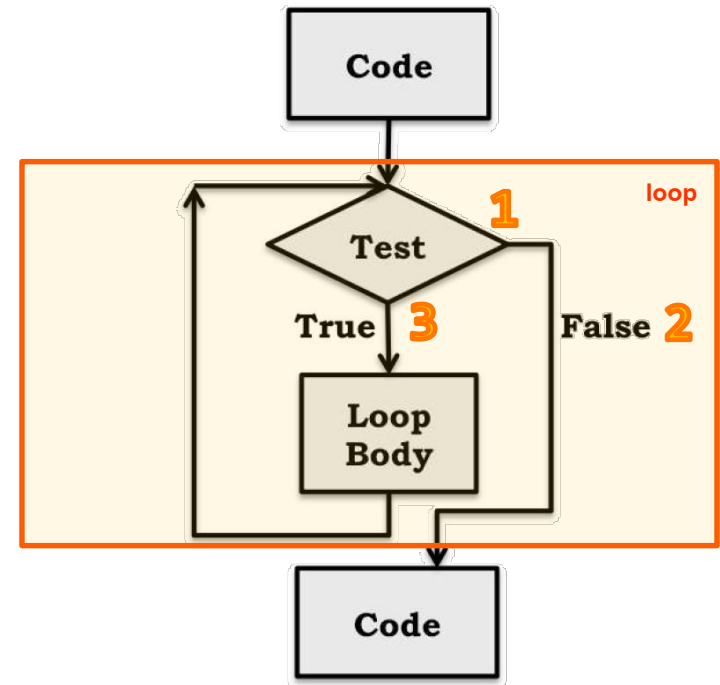
- หากมีการ reassign ค่าของ loop variable ค่าของ variable นั้น ๆ จะถูก reset ให้เป็นค่าที่ถูกกำหนด loop ครั้งถัดไป

# Types of Iteration

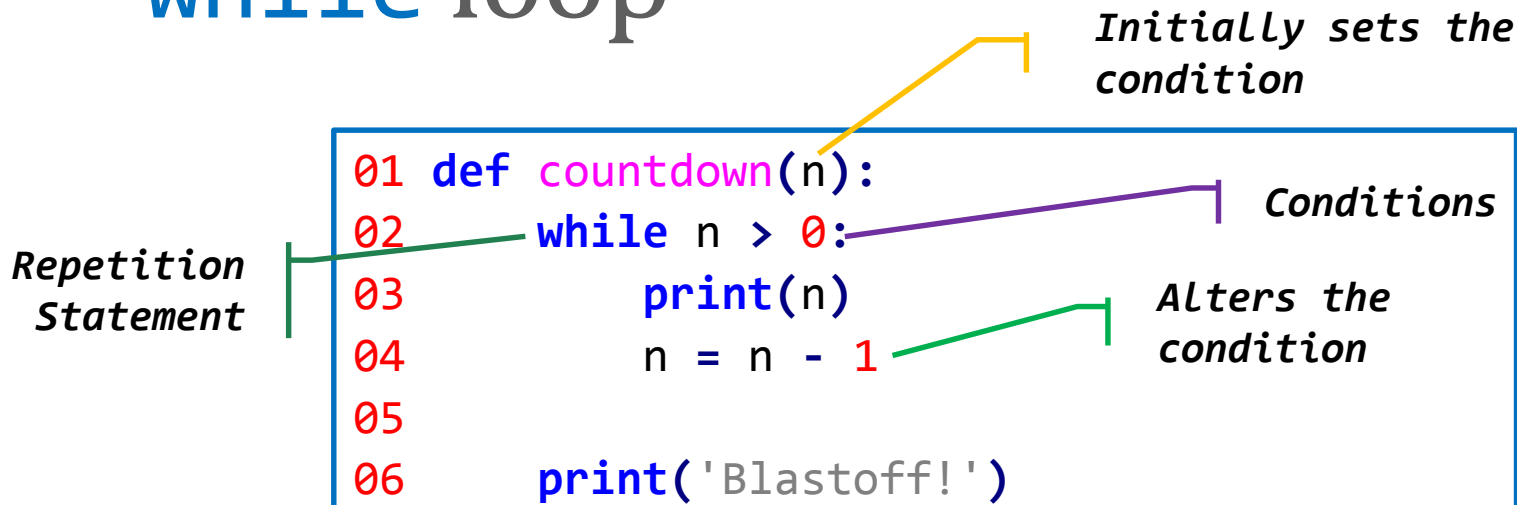
- **Counter-Controlled Loops**
  - Loop ที่ทำการวนซ้ำตามจำนวนครั้งที่กำหนด
- **Condition-Controlled Loops**
  - Loop ที่ทำการวนซ้ำจนกว่าเงื่อนไขที่กำหนดจะเป็นเท็จ

# Generic Loop Structure

- Loop structure ทั้งสองชนิดมีลักษณะคล้ายคลึงกันดังนี้
1. **Test** – ได้ผลลัพธ์เป็น True หรือ False
  2. ถ้าผลลัพธ์เป็น **False** - ออกจาก loop
  3. ถ้าผลลัพธ์เป็น **True**.  
ดำเนินการชุดคำสั่งใน Loop Body  
หนึ่งครั้ง แล้วกลับไปข้อ 1



# while loop



- 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	3	5	1
4	3	6	0

# Example 1: The Euclidean Algorithm

- For example 246 and 72

	<b>a</b>		<b>b</b>		<b>r</b>
STEP 1	246	mod	72	=	30
STEP 2	72	mod	30	=	12
STEP 3	30	mod	12	=	6
STEP 4	12	mod	6	=	0

```

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             break
12         elif n > key:
13             print(n, "is too high")
14         else:
15             print(n, "is too low")
16
```

- กรณีทายถูก (บรรทัดที่ 11) โปรแกรมจะต้องออกจาก loop
  - ใช้คำสั่ง **break**

# for vs while

```
>>> n = 10
>>> for i in range(n):
...     print('hello')
...     n -= 1
...
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
```

```
>>> i = 0
>>> n = 10
>>> while i < n:
...     print('hello')
...     n -= 1
...     i += 1
...
hello
hello
hello
hello
hello
hello
```

# 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)

```

02 def score_average():
03     total_count = 30
04     score_count = 0
05     total = 0
06
07     while score_count < total_count:
08         score = float(input("Enter score: "))
09         # ส่วนนี้อาจจะ
10         if score < 0 or score > 100:
11             continue
12
13         total = total + score
14         score_count = score_count + 1    # += 1
15
16     return total / score_count

```

## ทำไมในกรณีนี้ จึงไม่ควรใช้ for loop?



# The `continue` Statement

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

# Practice 1

1. Statement	<b>while</b>
2. Condition	
3. Initializing Condition	
4. Modifying Condition	

**Problem Statement** ให้เขียนฟังก์ชัน `int_power(x, y)` เพื่อหาค่า  $x^y$  ใด ๆ โดยที่  $x$  เป็นเลขจำนวนจริง และ  $y$  เป็นจำนวนเต็มที่ไม่มากกว่าหรือเท่ากับ 0 เช่น  $2.5^3 = 15.625$

- ตัวอย่างการ run

```
Input the base number: 2.5
Input the exponent: 3
2.5 to the power of 3 is: 15.625
```

while  $y \leq 3$ :  
 $x = x * x$   
 $y = y + 1$

# Practice 2

**Problem Statement** ให้เขียนฟังก์ชัน `int_to_bin(x)` เพื่อคืนค่าจำนวนในฐาน 2 ของ

จำนวนเต็ม  $x$  ในฐาน 10 โดยใช้วิธีการหาร

เช่น  $45_{10} = 101101_2$

- ตัวอย่างการ run

Input integer in base 10 : 28

Base 2 representation of 28 is: 11100

# Practice 2 [2]

1. Statement

2. Condition

3. Initializing Condition

4. Modifying Condition

## Step 1: Problem Solving

ยกตัวอย่าง แปลงเลข  $45_{10}$  เป็นฐาน 2

ให้ **result** คือค่าผลลัพธ์ ให้ค่าตั้งต้นเป็น 0

$\text{result} = \text{result} + (\text{เศษ} \times 10^i), i = 0, 1, 2, \dots$

$$45 / 2 = 22$$

$$\text{เศษ } 1 \rightarrow \text{result} = 0 + (1 \times 10^0)$$

$$22 / 2 = 11$$

$$\text{เศษ } 0 \rightarrow \text{result} = 1 + (0 \times 10^1)$$

$$11 / 2 = 5$$

$$\text{เศษ } 1 \rightarrow \text{result} = 01 + (1 \times 10^2)$$

$$5 / 2 = 2$$

$$\text{เศษ } 1 \rightarrow \text{result} = 101 + (1 \times 10^3)$$

$$2 / 2 = 1$$

$$\text{เศษ } 0 \rightarrow \text{result} = 1101 + (0 \times 10^4)$$

$$1 / 2 = 0$$

$$\text{เศษ } 1 \rightarrow \text{result} = 1101 + (1 \times 10^5)$$

$$= 101101$$

# Basic Loop Structures

การสร้าง Loop ประกอบด้วย 4 องค์ประกอบหลักคือ

1. Repetition statement: คำสั่งที่ใช้สำหรับการทำซ้ำ
  - ☒ **while statement**
  - ☒ **for statement**
2. Condition: เงื่อนไขของการทำซ้ำ
3. A statement that initially sets the condition being tested:  
การตั้งค่าเริ่มต้นให้กับตัวแปรที่ใช้ในการควบคุมเงื่อนไข
4. A statement within the repeating section of code that alters the condition so that it eventually becomes false:  
การเปลี่ยนแปลงค่าตัวแปรที่ใช้ในการควบคุมเงื่อนไข ในการวนแต่ละครั้ง เพื่อให้เงื่อนไขเป็นเท็จในที่สุด

# Sentinels

- การระบุ**จำนวนครั้ง**ที่ทำซ้ำไว้เป็น**ค่าคงที่** ในบางครั้งอาจไม่ยืดหยุ่นเพียงพอกับปัญหาที่ต้องการแก้
  - เช่นกรณีต้องการหาค่าเฉลี่ยของคะแนนของนักเรียนในชั้น แต่ไม่ทราบจำนวนนักเรียนล่วงหน้า
- สามารถแก้ปัญหาได้โดย
  - ให้ user ระบุจำนวนครั้งที่ต้องการทำซ้ำ ผ่าน input
  - **หรือ** อ่าน input จาก user จนเจอค่าที่ตกลงกันไว้ว่าใช้แสดงจุดสิ้นสุดของข้อมูล เช่น -1 หรือ EOF (End of File)
    - ค่าที่ใช้แสดง**จุดเริ่มหรือสิ้นสุด**ของข้อมูลในลักษณะนี้เรียกว่า ***sentinels***
    - ควรเลือกค่า **sentinels** ให้**ไม่ทับซ้อน**กับค่าของข้อมูลที่เป็นไปได้

# Sentinels [2]

```
09 def score_average():
10     SENTINEL = -1
11     print("Please enter students' score one for each line")
12     print("and %d for termination: " % SENTINEL)
13     total = 0.0
14     count = 0
15
16     while (True):
17         score = float(input(""))
18
19         if score == SENTINEL:
20             break
21         total += score
22         count += 1
23
24     if count > 0:          #why do we need this?
25         average = total / count
26     else
27         average = 0
28
29     print("The average of the %d numbers is %8.4f" %(count, average))
```

# LOOPING WITH ITERABLES



# Traversal with a **while** Loop

- การเข้าถึงอักขระใน **String** ทีละตัว (Traversal) การใช้ **while** Loop

```
02 fruit = "banana"
03 index = 0
04
05 while index < len(fruit):
06     letter = fruit[index]
07     print(letter)
08     index = index + 1
```

# Traversal with a **for** Loop

- **for** Loop with Indexes:

```
11 s = "abcd"
12 length = len(s)
13 for i in range(length):
14     print(i, s[i])
15
```

- **for** Loop without Indexes

```
17 s = "abcd"
18 for c in s:          # similar to for i in range(n)
19     print(c)
```

- เราเรียก Data Type ที่สามารถเข้าถึงแต่ละหน่วยย่อย (เช่น 1 ตัวอักษร) ได้ครั้งละ 1 หน่วย (เช่น **range** หรือ **string**) ว่า **Iterable**

# Iteration: Searching

- พิจารณาการทำงานของฟังก์ชันดังต่อไปนี้

```
01 def find(word, letter):  
02     index = 0  
03     while index < len(word):  
04         if word[index] == letter:  
05             return index  
06         index = index + 1  
07     return -1
```

- ลักษณะการทำงานตรงกันข้ามกับเครื่องหมาย []
  - รับอักขระ *letter* มาแล้วหา Index ใน String *word*
  - หากอักขระ *letter* ไม่ปรากฏใน *word* จะคืนค่า -1

# Iteration: Counting

- ฟังก์ชันด้านล่างนับจำนวนครั้งที่อักขระ *key* ปรากฏใน String *word*

```
02 def count_letter(word, key):  
03     count = 0  
04  
05     for letter in word:  
06         if letter == key:  
07             count = count + 1  
08     print(count)  
09  
10 count_letter('banana', 'a')
```

# Iteration: `in_both()`

- ฟังก์ชันด้านล่างแสดงผลอักขระที่ซ้ำใน String *word1* และ *word2*

```
01 def in_both(word1, word2):
02     for letter in word1:
03         if letter in word2:
04             print(letter)
```

เมื่อเปรียบเทียบ String 'apple' และ 'orange'

```
>>> in_both('apples', 'oranges')
a
e
s
```

# Looping over Lists

```
>>> a = [2, 3, 5, 7]

# Looping with: for item in list
>>> for item in a:
...     print(item, end=" ")

2 3 5 7

# Looping with: for index in range(len(list))
>>> for index in range(len(a)):
...     print("a[", index, "] = ", a[index], sep="")

a[0] = 2
a[1] = 3
a[2] = 5
a[3] = 7
```

# Looping over Lists [2]

```
# Looping backward a = [2, 3, 5, 7]
>>> for index in range(len(a)):
...     revIndex = len(a) - 1 - index
...     print("a[" + revIndex + "] = ", a[revIndex], sep="")
```

```
a[3] = 7
a[2] = 5
a[1] = 3
a[0] = 2
```

```
# Hazard!!: Modifying While Looping
```

```
>>> for index in range(len(a)):
...     if (a[index] == 3):
...         a.pop(index)
```

```
3
```

```
IndexError: list index out of range
```

# List Function with Loops

- List Parameters

- **Example:** `count_odds(list)`

```
09 def count_odds(list_a):
10     count = 0
11     for item in list_a:
12         if (item % 2 == 1):
13             count += 1
14     return count
15
16 print(count_odds([2, 3, 7, 8, 21, 23, 24]))
17 # 4
```



# List Function with Loops [2]

- **Modifying list elements is visible to caller:**

`fill(list, value)`

```
>>> def fill(list_a, value):  
...     for i in range(len(list_a)):  
...         list_a[i] = value  
  
>>> a = [1, 2, 3, 4, 5]  
>>> fill(a, 42)  
>>> a  
[42, 42, 42, 42, 42]
```

# List Function with Loops [3]

- List Return Type

- **Example:** `numbers_with_3s()`

```
02 def numbers_with_3s(lo, hi):
03     result = []
04
05     for x in range(lo, hi):
06         if ("3" in str(x)):
07             result.append(x)
08     return result
09
10 print(numbers_with_3s(250, 304))
11 # [253, 263, 273, 283, 293, 300, 301, 302, 303]
```

# Merge Algorithm

- ในกรณีที่มี list ที่มีการเรียงลำดับไว้แล้วมากกว่าหนึ่ง list
- การนำ list ทั้งสองมารวมกันเพื่อให้ได้ list ใหม่ที่มีการเรียงลำดับ
  - ซึ่งประกอบด้วย element ทั้งหมดจาก list ทั้งสอง
  - เรียกว่า merging
- โดยวิธีในการรวม list ลักษณะดังกล่าว เรียกว่า Merge Algorithm

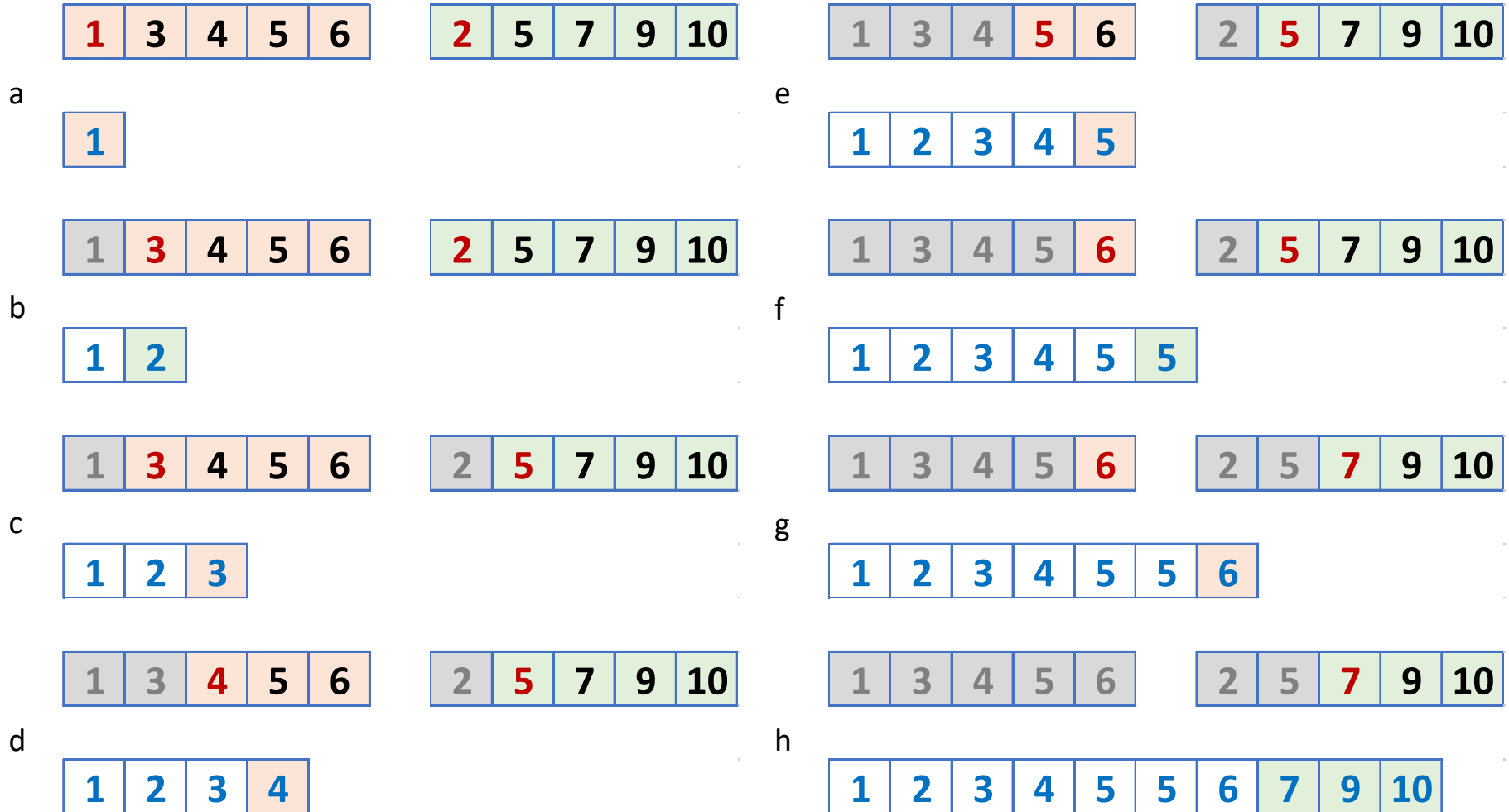
# Merge Algorithm [2]

1	3	4	5	6
---	---	---	---	---

2	5	7	9	10
---	---	---	---	----

- ให้ index  $i$  และ  $j$  ชี้ตำแหน่ง head ของ list A และ list B
- ให้ C เป็น list ว่าง
- ให้  $a$  และ  $b$  เป็น value ใน list A และ B ที่ตำแหน่ง  $i$  และ  $j$  ตามลำดับ
- If  $a < b$  เพิ่ม  $a$  ไปที่ list C และ increment  $i$ 
  - Else เพิ่ม  $b$  ไปที่ list C และ increment  $j$
- ถ้า list ใดหมดก่อนให้นำ element ทั้งหมดของ list ที่เหลือ เพิ่มไปที่ list C

# Merge Algorithm [3]



# Merge Algorithm [4]

```

09 def merge_list(list_a, list_b):
10     len_a = len(list_a)
11     len_b = len(list_b)
12     i = 0
13     j = 0
14     list_c = []
15
16     while i < len_a and j < len_b:
17         if list_a[i] < list_b[j]:
18             _____
19             i += 1
20         else:
21             list_c.append(list_b[j])
22             j += 1
23
24     if i < len_a:
25         _____
26     if j < len_b:
27         _____
28
29     return list_c

```

bash shell

```

$ python -i merge_list.py
>>> list_a = [1, 3, 4, 5, 6]
>>> list_b = [2, 5, 7, 9, 10]
>>> print(merge_list(list_a, list_b))
[1, 2, 3, 4, 5, 5, 6, 7, 9, 10]

```

# Merge Algorithm – w/ Recursion

```
02 def merge_list(list_a, list_b):
03     if not list_a:
04         return _____
05     if not list_b:
06         return _____
07
08     if list_a[0] < list_b[0]:
09         sub_task = _____
10         sub_sol = _____
11     else:
12         sub_task = _____
13         sub_sol = _____
14
15     return _____
16
```

# References

- <https://docs.python.org/3/library/string.html>
- <http://www.kosbie.net/cmu/spring-13/15-112/handouts/notes-strings.html>
- <http://www.kosbie.net/cmu/spring-13/15-112/handouts/notes-1d-lists.html>
- [http://en.wikipedia.org/wiki/Merge\\_algorithm](http://en.wikipedia.org/wiki/Merge_algorithm)
- Guttag, John V. *Introduction to Computation and Programming Using Python, Revised*