List Tuple Dict

List

- List
- Elements in list
- for loop and lists
- Working with lists
- List comprehension
- String

ลักษณะของ List

- List หรือรายการ เป็นชนิดของข้อมูลชนิดหนึ่งซึ่งประกอบไปด้วยสมาชิกเรียงกันเป็นลำดับ
- สมาชิกใน list เป็นชนิดใดก็ได้และไม่จำเป็นต้องเป็นชนิดเดียวกัน
- ขนาดของ list เพิ่ม-ลด ได้อัตโนมัติตามจำนวนสมาชิก

การเขียน list ใน python

- เขียนสมาชิกภายในวงเล็บสี่เหลี่ยม [ ] สมาชิกแต่ละตัวคั่นด้วย comma (,)
- เช่น [1, ‘a’, ‘Python’, [True, False]]
- [ ] หมายถึงลิสต์ว่าง

ขนาดของ List

- ขนาดของ List คือ จำนวนสมาชิกภายใน List
- ใช้ฟังก์ชัน len ในการหาขนาด
- len([1, ‘a’, ‘Python’ , [True, False]])

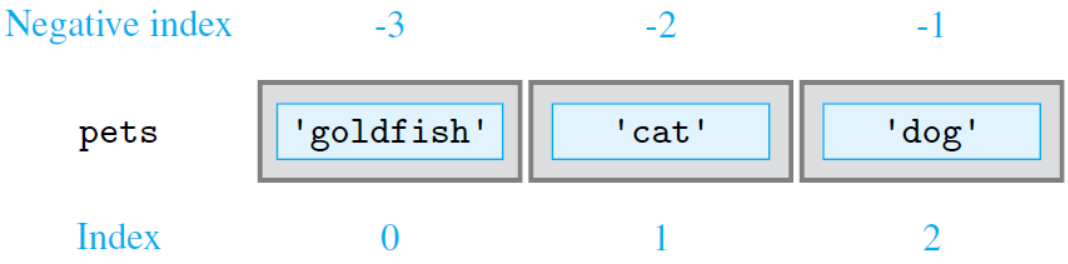
  มีค่า 4

- data= [2, ‘a’, ‘True’, ‘5’, False]

  len(data) มีค่า 5

การสร้าง List

```
>>> pets = ['goldfish', 'cat', 'dog']
```



รูป: การสร้าง List

Elements in lists

การเก็บข้อมูล List และดัชนี

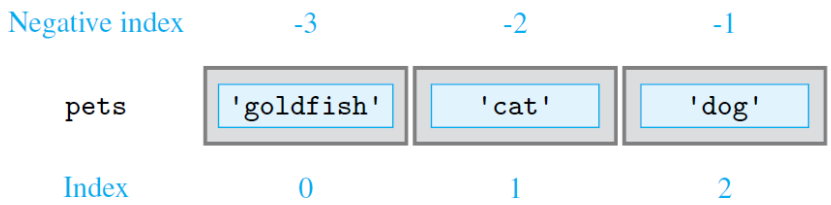
- ลักษณะการเก็บข้อมูลใน List จะเป็นช่องๆ มีดัชนีเป็นเลขจำนวนเต็มอ้างถึงช่องเหล่านั้น

สำหรับ Python

- ค่าดัชนีช่องแรกเป็น 0
- นับดัชนีย้อนจากท้าย List ได้โดยช่องสุดท้ายมีดัชนีเป็น –1

การเข้าถึงสมาชิกใน List ใช้ชื่อของ List (ชื่อตัวแปร) ร่วมกับเลขดัชนีในวงเล็บสี่เหลี่ยม

```
>>> pets = ['goldfish', 'cat', 'dog']
>>> pets[0]
'goldfish'
>>> pets[2]
'dog'

>>> pets[-1]
'dog'
```



*การดำเนินการและฟังก์ชันใน List

การดำเนินการและฟังก์ชันใน list

Usage	Explanation
`x in lst`	True if object `x` is in list `lst`, false otherwise
`x not in lst`	False if object `x` is in list `lst`, true otherwise
`lstA + lstB`	Concatenation of lists `lstA` and `lstB`
`lst * n, n * lst`	Concatenation of *n* copies of list `lst`
`lst[i]`	Item at index *i* of list `lst`
`len(lst)`	Length of list `lst`
`min(lst)`	Smallest item in list `lst`
`max(lst)`	Largest item in list `lst`
`sum(lst)`	Sum of items in list `lst`

```
>>> pets = ['goldfish', 'cat', 'dog']
>>> len(pets)
3
>>> pets + pets
['goldfish', 'cat', 'dog', 'goldfish', 'cat', 'dog']
>>> pets * 2
['goldfish', 'cat', 'dog', 'goldfish', 'cat', 'dog']
>>> 'rabbit' in pets
False
>>> 'dog' in pets
True
```

```
>>> lst = [23.99, 19.99, 34.50, 120.99]
>>> min(lst)
19.99
>>> max(lst)
120.99
>>> sum(lst)
199.46999999999997
```

การเปลี่ยนแปลงค่าใน List

- การเก็บค่าใน List สามารถเปลี่ยนแปลงค่าได้

```
>>> pets = ['goldfish', 'cat', 'dog']
```

- เราต้องการเปลี่ยนข้อมูล pets[1] จาก ‘cat’ ไปเป็น ‘cymric cat’

```
>>> pets[1] = 'cymric cat'
>>> pets
['goldfish', 'cymric cat', 'dog']
```

for loop and lists

การเข้าถึงข้อมูลใน List โดยใช้ iterator

- iterator คือตัวแจงนับที่อ้างถึงสมาชิกใน List ไปทีละตัว
- ใน Python ใช้ **for** variable **in** [...] ในการแจงนับสมาชิกใน List

เช่น

PYTHON CODE:

```
pets = ['goldfish', 'cat', 'dog']
for item in pets:
    print(item)
```

- อ่านได้ว่า สำหรับ item แต่ละตัวซึ่งอยู่ใน data
- ในแต่ละรอบการทำงาน item จะเชื่อมกับค่าสมาชิกแต่ละตัวใน data ซึ่งได้แก่
  - goldfish
  - cat
  - dog

List และ **range**

- เนื่องจากฟังก์ชัน เป็นการแจงนับจำนวนเต็มทีละตัว
- เราอาจจะใช้ **range** ในการแจงนับ**ดัชนี**ของ List ได้
  - ไม่กำหนดค่าเริ่มต้นของ range จะได้ค่าเริ่มต้นเป็น 0 โดยปริยาย
  - กำหนดขอบปลายเป็นขนาดของ List

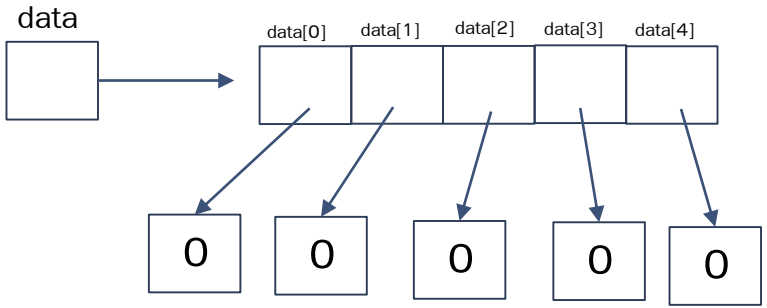
```
data=[1, 'a', 'True', -5, False]
for index in range(len(data)):
  print(data[index])
```

- **index** จะมีค่าตั้งแต่ 0 ถึง **len(data)**−1 ซึ่งเป็นดัชนีแต่ละตัวของสมาชิกใน List
- ส่วนของโปรแกรมนี้จะพิมพ์ค่าสมาชิกแต่ละตัวใน **List** จนครบ

การแก้ไขค่าในลิสต์

- หากต้องการแก้ไขค่าแต่ละตัวในลิสต์ เราสามารถใช้รูปแบบ for in range เพื่อแก้ไขค่าของแต่ละตัวแปรในลิสต์ได้

```
data=[1, 'a', 'True', -5, False]
for index in range(len(data)):
  data[index]=0
```



รูป: รูปหลังลูปเสร็จ

ในรอบที่ 1 index มีค่า 0 ดังนั้น data[0]=0 จะเปลี่ยนการเชื่อมค่าของ data[0] ให้ไปเชื่อมกับ 0

ในรอบที่ 2 index มีค่า 0 ดังนั้น data[1]=0 จะเปลี่ยนการเชื่อมค่าของ data[1] ให้ไปเชื่อมกับ 0

ในรอบที่ 3 index มีค่า 0 ดังนั้น data[2]=0 จะเปลี่ยนการเชื่อมค่าของ data[2] ให้ไปเชื่อมกับ 0

ในรอบที่ 4 index มีค่า 0 ดังนั้น data[3]=0 จะเปลี่ยนการเชื่อมค่าของ data[3] ให้ไปเชื่อมกับ 0

ในรอบที่ 5 index มีค่า 0 ดังนั้น data[4]=0 จะเปลี่ยนการเชื่อมค่าของ data[4] ให้ไปเชื่อมกับ 0

การกำหนดค่าลงในลิสต์

- เราสามารถ**วน**รับค่าจากผู้ใช้แล้วเก็บลงในลิสต์ได้

ในตัวอย่างมีนักศึกษา 5 คน และวนรับคะแนนสอบ 5 ครั้งเพื่อเก็บลงในลิสต์

```
score=[0, 0, 0, 0, 0]
for index in range(len(score)):
  score[index]=int(input('Enter a score: '))
print(score)
```

ตัวอย่างผลลัพธ์การรัน

```
Enter a score: 10
Enter a score: 9
Enter a score: 8
Enter a score: 7
Enter a score: 5
10, 9, 8, 7, 5]
```

Working with lists

Working with lists

- Combining two lists
- The repetition operator
- Adding an element at the end of a list (append)
- Inserting an element at the specified position (insert)
- Copying a list
- Checking if a value is in a list
- Locate the position of value in a list (index)
- Delete an element in a list
- Sorting a list
- Reversing a list
- Find min or max value in a list
- Find a sum of values in a list

Working with lists

การรวมสองลิสต์เข้าด้วยกัน

data= data1+data2 สร้าง List ใหม่ซึ่งเกิดจาดการนำสมาชิกใน List เดิมมาเรียงต่อกัน

PYTHON CODE

```
data=[1,2,3,4,5]
data2=['a',True]
data3=data+data2
print(data3)
```

OUTPUT

```
[1, 2, 3, 4, 5, 'a', True]
```

เครื่องหมาย * กับ List รูปแบบ list*n โดย n คือ integer,

* จะสร้าง copies ของ list ขึ้นตามจำนวน n แล้วนำมาต่อกัน

PYTHON CODE

```
number=[1,3,5]
print(number*3)
```

OUTPUT

```
[1, 3, 5, 1, 3, 5, 1, 3, 5]
```

Working with lists

การเพิ่มสมาชิกลงในลิสต์ที่ตำแหน่งสุดท้าย

```
data = [1,2,3,4,5]
data.append(10)
```

เพิ่มสมาชิกลงใน list เดิม

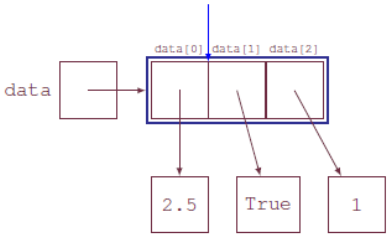
ข้อควรระวัง: data.append([1,2,3]) จะได้ data = [1,2,3,4,5, [1,2,3]] มีสมาชิกเพิ่มขึ้นมา 1 ตัวเป็น List

การเพิ่มสมาชิกลงในลิสต์ตามตำแหน่งที่ระบุ

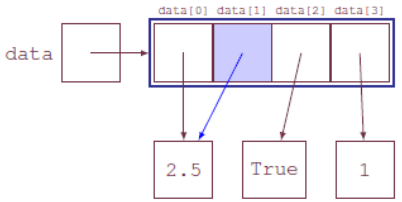
เพิ่มสมาชิกใหม่ ณ ดัชนีที่กำหนด, ฟังก์ชัน insert: insert(<index>,<item>)

* จะสร้าง copies ของ list ขึ้นตามจำนวน n แล้วนำมาต่อกัน

```
data = [2.5, True, 1]
data.insert(1, 2.5)
```



List ก่อนแทรกสมาชิก



List หลังแทรกสมาชิก

Working with lists

Shallow copy

data2 = data

เป็นการเชื่อม (bind) ค่าให้ตัวแปร List ทางซ้ายของเครื่องหมาย = เชื่อมไปยังตำแหน่งเดียวกับตัวแปร List ทางขวาของเครื่องหมาย =

- เป็น shallow copy
- หากเปลี่ยนค่าผ่าน data2 ส่งผลต่อ data และเปลี่ยน data จะส่งผลกับ data2

การตรวจสอบว่าค่าในลิสต์

- สามารถตรวจสอบว่าค่าปรากฏอยู่ในลิสต์หรือไม่โดยรูปแบบ value in list
- และตรวจสอบว่าค่าไม่ปรากฏอยู่ในลิสต์โดยรูปแบบ value not in list

PYTHON CODE

```
name=['mana','manee','piti']
'mana' in name
```

OUTPUT

```
True
```

Working with lists

Copying a list

หากต้องการ copy list ให้เป็นสองลิสต์ที่แยกจากกันแต่มีค่าเหมือนกันสามารถทำได้หลายวิธี

วิธีที่ 1 ใช้การ combine กับ empty list

```
data2=[]+data
```

วิธีที่ 2 ใช้การวนลูป copy ทีละตัว

```
data2=[]
for item in data:
data.append(item)
```

วิธี่ที่ 3 ใช้ slicing

```
data2=data[:]
```

วิธีที่ 4 ใช้ฟังก์ชัน list

```
data2 = list(data)
```

Working with lists

การหาตำแหน่งของ element ในลิสต์

- หากต้องการหาตำแหน่งของ element ในลิสต์ สามารถใช้ index ในรูปแบบ list.index(item)
- จะคืนตำแหน่งของ element แรกที่เจอในลิสต์
- หากไม่เจอ element นั้นจะเกิด ValueError exception

การลบสมาชิกในลิสต์โดยตำแหน่ง

- ใช้คำสั่ง del เหมือนกับการลบตัวแปรปกติ
- การลบเป็นการลบชื่อตัวแปรเท่านั้น ตัวแปรอื่นที่ผูกอยู่กับค่าเดียวกัน จะไม่มีการเปลี่ยนแปลง

การลบสมาชิกแบบระบุค่า

- สามารถลบ item ออกจากลิสต์โดยรูปแบบ list.remove(item)
- Element แรกมีค่าเท่ากับ item จะถูกลบออก
- ขนาดของลิสต์หลังถูกลบจะลดลงไปหนึ่ง และดัชนีของ element หลังตำแหน่งที่ถูกลดลงไปหนึ่ง
- หากไม่เจอ element นั้นจะเกิด ValueError exception

Working with lists

การเรียงลำดับข้อมูลใน list

- ใช้ sort ผลลัพธ์อยู่ที่ list เดิม ไม่มีการคูณค่า

การกลับลำดับการเรียงข้อมูลใน List

- ใช้ reverse ผลลัพธ์ที่ได้อยู่ใน list เดิม ไม่มีการคืนค่า

การหาค่ามากสุดหรือน้อยสุดใน list

- ใช้ฟังก์ชัน min เพื่อหาค่าที่น้อยที่สุดใน list
- ใช้ฟังก์ชัน max เพื่อหาค่าที่มากที่สุดใน list

การหาผลรวมใน list

- ใช้ฟังก์ชัน sum เพื่อหาผลรวมของตัวเลขใน list
- หากมีสมาชิกใน list ที่ไม่ใช่ตัวเลขจะเกิด error

ตัวอย่างการรัน

```
data=[2,10,0.1,-5.3,4]
data.sort()
data
```

```
[-5.3, 0.1, 2, 4, 10]
```

```
[15] data=[2,10,0.1,-5.3,4]
     data.reverse()
     data
```

```
[4, -5.3, 0.1, 10, 2]
```

```
data=[2,10,0.1,-5.3,4]
print(max(data))
print(min(data))
print(sum(data))
```

```
10
-5.3
10.8
```

List comprehension

**List comprehension

- การสร้าง List โดยอาศัยนิยามทางคณิตศาสตร์
- ใช้ร่วมกับ for ซึ่งสร้างตัวแจงนับเพื่อกำหนดสมาชิกภายใน List
- สามารถใช้ if ร่วมกับเพื่อกำหนดเงื่อนไขของสมาชิกใน List ได้

เช่น List ของ x โดยที่ x เป็นจำนวนเต็มในช่วง [1,10)

```
data = [x for x in range(1,10)]
```

หากไม่ใช้ **list comprehension** สามารถเขียนได้ดังนี้

```
data = []
for x in range(1,10):
    data.append(x)
```

ตัวอย่างการรัน

```
data=[x for x in range(1,10)]
print(data)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
data =[]
for x in range(1,10):
  data.append(x)
print(data)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

List comprehension

- List ของสมาชิกใน data และชนิดข้อมูลของสมาชิกนั้นเป็นจำนวนเต็ม

```
data=[2, ‘a’, ‘True’, 5, False]
data_int=[item for item in data if type(item)==int]
print(data_int)
```

หากไม่ใช้ **list comprehension** สามารถเขียนได้ดังนี้

```
data_int=[ ]
for item in data:
    if type(item)==int:
        data_int.append(item)
```

ตัวอย่างการรัน

```
data=[2,'a','True',-5,False]
data_int=[item for item in data if type(item)==int]
print(data_int)
```

```
[2, -5]
```

```
data=[2,'a','True',-5,False]
data_int=[]
for item in data:
  if type(item) == int:
    data_int.append(item)
print(data_int)
```

```
[2, -5]
```

ตัวอย่าง list comprehension

- List ของชนิดข้อมูลของสมาชิกแต่ละตัวใน data

```
data_type = [type(item) for item in data]
```

หากไม่ใช้ **list comprehension** สามารถเขียนได้ดังนี้

```
data_int=[ ]
for item in data:
    data_type.append(type(item))
```

- กำหนดให้ x เป็นจำนวนเต็มบวก ต้องการสร้าง factor เป็น list ของตัวประกอบ x

```
factor = [num for num in range(1, x+1)
if x%num ==0]
```

หากไม่ใช้ **list comprehension** สามารถเขียนได้ดังนี้

```
factor = [ ]
for num in range(1, x+1):
    if  x % num == 0:
        factor.append(num)
```

String and for in string

- สตริง หรือ สายอักขระ มีคลาสเป็น str ไพทอน
- ลักษณะของสตริงจะคล้าย List ของอักขระ
- สามารถอ่านค่าสมาชิกแต่ละตัวในสตริงได้โดยใช้ดัชนีเช่นเดียวกับ List แต่ไม่สามารถแก้ไขค่าของสมาชิกได้
- ขนาดของสตริง คือ จำนวนอักขระภายในสตริง ใช้ฟังก์ชัน len ในการหา เช่นเดียวกับ List
- ในภาษาไพทอน string หากถูกสร้างขึ้นแล้วจะไม่สามารถถูกแก้ไขได้ เรียก string มัน immutable

for in string

- สามารถใช้ for เพื่อวนทีละตำแหน่งใน string ได้

```
name = “Tan”
for ch in name:
    print(ch)
```

Method สำหรับตรวจสอบ string

isalnum()	คืนค่า True หาก string มีเฉพาะตัวเลขหรือตัวอักษรและมีความยาวอย่างน้อยหนึ่งตัวอักษร
isalpha()	คืนค่า True หาก string มีเฉพาะตัวอักษรและมีความยาวอย่างน้อยหนึ่งตัวอักษร
isdigit()	คืนค่า True หาก string มีเฉพาะตัวเลขและมีความยาวอย่างน้อยหนึ่งตัวอักษร
islower()	คืนค่า True หากทุกตัวอักษรใน string เป็นตัวพิมพ์เล็ก
isspace()	คืนค่า True หาก string มีแต่ whitespace และมีความยาวอย่างน้อยหนึ่งตัวอักษร
isupper()	คืนค่า True หากทุกตัวอักษรใน string เป็นตัวพิมพ์ใหญ่

upper()	คืน copy ของ string โดยตัวอักษรทุกตัวจะเป็นตัวพิมพ์ใหญ่
lower()	คืน copy ของ string โดยตัวอักษรทุกตัวจะเป็นตัวพิมพ์เล็ก
split()	ตัด string โดยหากไม่ระบุตัวตัดจะตัดโดย whitespace และคืนค่ากลับมาเป็น list ของ string ที่ถูกตัด

Tuple

Tuple

Tuple คือข้อมูลรายการแบบลำดับ คล้าย List แต่ไม่สามารถเปลี่ยนแปลงข้อมูลได้ ข้อมูลที่อยู่ใน Tuple เป็นชนิดใดก็ได้ ไม่จำเป็นต้องเป็นข้อมูลชนิดเดียวกัน

รูปแบบของ Tuple

รูปแบบของ Tuple

- Tuple จะมีข้อมูลกี่ตัวก็ได้
- ข้อมูลจะอยู่ในเครื่องหมาย ()ข้อมูลใน Tuple เรียกว่า สมาชิก (member)
- ถ้าใน Tuple เก็บข้อมูลหลายตัว สมาชิกแต่ละตัวจะถูกคั่นด้วยเครื่องหมายจุลภาค (comma) ,

(ข้อมูลตัวที่ 1, ข้อมูลตัวที่ 2, ข้อมูลตัวที่ 3, ..., ข้อมูลตัวที่ n)

รูปแบบของ Tuple

ตัวอย่าง

```
a = (1,2,3,4,5)
b = (123456789,'Python',True)
c = 123456789,'Python',True # ไม่จำเป็นต้องใส่วงเล็บก็ได้
d = 123456789, # แต่ต้องมี comma
e = () #Empty tuple

print('a:', type(a))
print('b:', type(b))
print('c:', type(c))
print('d:', type(d))
print('e:', type(e))
```

ขนาดของ Tuple

- Tuple มีขนาดเท่ากับจำนวนสมาชิก
- เราสามารถใช้คำสั่ง len(ทูเปิล) เพื่อหาขนาดของ Tuple ได้
- เนื่องจากเรา *ไม่* สามารถเปลี่ยนแปลงข้อมูลใน Tuple ได้ ขนาดของ Tuple จึง *ไม่* สามารถเปลี่ยนแปลงได้

ตัวอย่าง

```
len((1,2,3,4,5))
x = (12345,'Python',True)
len(x)
empty_tuple = ()
len(empty_tuple)
len((2)) # เกิด error เพราะ (2) เป็น int ที่ใส่วงเล็บ ไม่ได้เป็น tuple
y = (12345,'Python',True,('N','D','M','J'),3.0)
len(y)
```

การเข้าถึงข้อมูลใน Tuple

- สามารถใช้ index ได้เช่นเดียวกับ list

```
a = 3.0
b = ('N','092-5166xxx')
c = ['S','M','J']
data = (12345,'Python',True,a,b,c)

print(data[0])
print(data[-6])
print(data[-len(data)])

print(data[1])
print(data[-5])

print(data[2])
print(data[-4])

print(data[3])
print(data[-3])

print(data[4])
print(data[-2])

print(data[5])
print(data[-1])
print(data[len(data)-1])
```

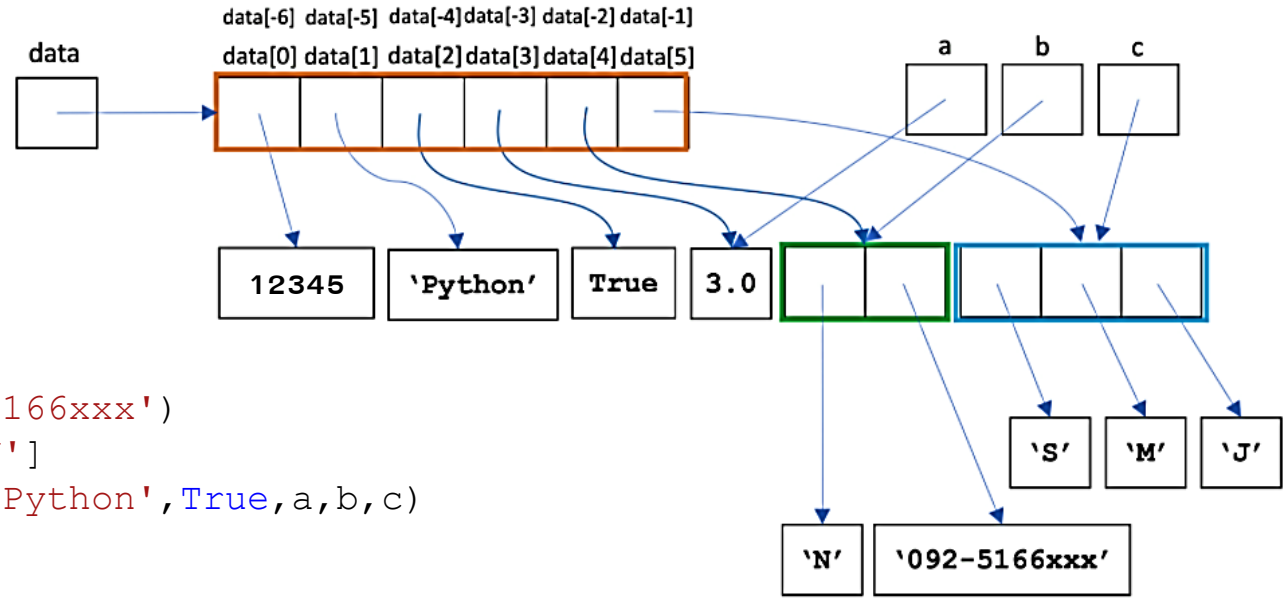
การเข้าถึงข้อมูลใน Tuple

- สามารถใช้ for เพื่อแจกแจงสมาชิกของ tuple รอบละตัว ได้เช่นเดียวกับ list

```
for i in range(len(data)):
  print('index',i,':',data[i])


for element in data:
  print(element)

data[6] # เกิด Error เพราะ index ที่ใช้เกินจากจำนวนสมาชิกที่เก็บอยู่ใน data
```

การเก็บข้อมูลใน Tuple

ตัวอย่าง: การเก็บข้อมูลของตัวแปร data




```
a = 3.0
b = ('N','092-5166xxx')
c = ['S','M','J']
data = (12345,'Python',True,a,b,c)
```

การเก็บข้อมูลใน Tuple

ตัวอย่าง: การเก็บข้อมูลของตัวแปร data

```
"""
a = 3.0
b = ('N',’081-7466xxx')
c = ['S','M','J']
data = (12345,'Python',True,a,b,c)
"""
a = 5.0 # เปลี่ยนค่าในตัวแปร a ได้ เพราะ a เป็นข้อมูลประเภท float
for element in data:
  print(element)
print('a:’,a)
```

***** ตัวอย่างเพิ่มเติมในไฟล์ Colab**

Dictionary

Dictionary

เราสามารถเก็บข้อมูลหลายๆตัวไว้ในตัวแปรเดียวได้ ทำให้การเข้าถึงทำได้ง่าย นั่นคือการใช้ list, tuple, หรือ set

แต่ถ้าข้อมูลแต่ละตัวที่เราจะเก็บนั้นมีลักษณะพิเศษ เช่น ...

การเก็บข้อมูลสมุดโทรศัพท์ ซึ่งจะเก็บทั้งชื่อและเบอร์โทรศัพท์ที่คู่กับชื่อนั้น ๆ เราจะเก็บอย่างไร วิธีหนึ่งคือ เราอาจใช้ list ในการเก็บข้อมูลทั้งสอง โดย

- List ตัวแรก ใช้ในการเก็บชื่อ
- List ตัวที่สอง ใช้ในการเก็บเบอร์โทรศัพท์

Dictionary

ตัวอย่าง

```
names = ['Dan','Beam','Big']
phonebook = ['086-123-4567','092-987-6543', '083-333-3333']
```

แล้วถ้า Beam มี 2 เบอร์ ให้ใช้ List มาเก็บข้อมูลของ Beam ก็ได้ ดัง code ด้านล่าง

```
names = ['Dan','Beam','Big']
phonebook = ['086-123-4567',['092-987-6543','091-234-4567'], '083-333-3333']
```

*Dictionary

ตัวอย่าง

ถ้าเราอยากจะโทรไปหา Big เราต้องหาก่อนว่าเบอร์ของ Big เก็บไว้เป็นลำดับที่เท่าไหร่ เราถึงจะดึงข้อมูลออกมาได้ ซึ่งสามารถเขียนโปรแกรมได้ดังนี้

```
for i in range(len(names)):
  if names[i]=='Big':
    print(phonebook[i])
```

Dictionary

ตัวอย่าง

ลองจินตนาการต่อว่าถ้า phonebook ของเราเล่มใหญ่มาก เก็บรายชื่อและเบอร์โทรศัพท์ของคนทั้งประเทศ ถ้าเราอยากจะโทรหาใครซักคน เราต้องวนลูปไปนานมากกว่าจะได้เบอร์ของคนนั้นมา ทำให้เสียเวลามาก
ดังนั้นใน Python จึงมีการเก็บข้อมูลแบบกลุ่มอีกแบบหนึ่ง เรียกว่า Dictionary หรือเรียกย่อๆว่า Dict ซึ่งเข้าถึงข้อมูลในลักษณะที่เป็นคู่กัน เช่น ชื่อกับเบอร์โทรศัพท์ ได้ง่ายและรวดเร็ว

สิ่งที่ Dict ทำก็คือ การอนุญาตให้ผู้ใช้สามารถตั้งชื่อ index ได้เอง (เรียกว่า key) โดยที่ไม่จำเป็นต้องเป็นตัวเลขจำนวนเต็มที่เรียงกันเป็นลำดับเหมือนที่ใช้ในลิสต์

LIST

Index	Value
0	'086-123-4567'
1	['092-987-6543','091-234-4567']
2	'083-333-3333'
....	....
....	....

DICT

Key

Index	Value
'Dan'	'086-123-4567'
'Beam'	['092-987-6543','091-234-4567']
'Big'	'083-333-3333'
....	....
....	....

รูปแบบของ Dict

- dict เก็บข้อมูลเป็นคู่ ๆ ในลักษณะของ key ⟶ value
- เขียนสมาชิกภายใน วงเล็บปีกกา { }
- สมาชิกแต่ละตัวคั่นด้วย จุลภาค ,
- key กับ value คั่นด้วย :

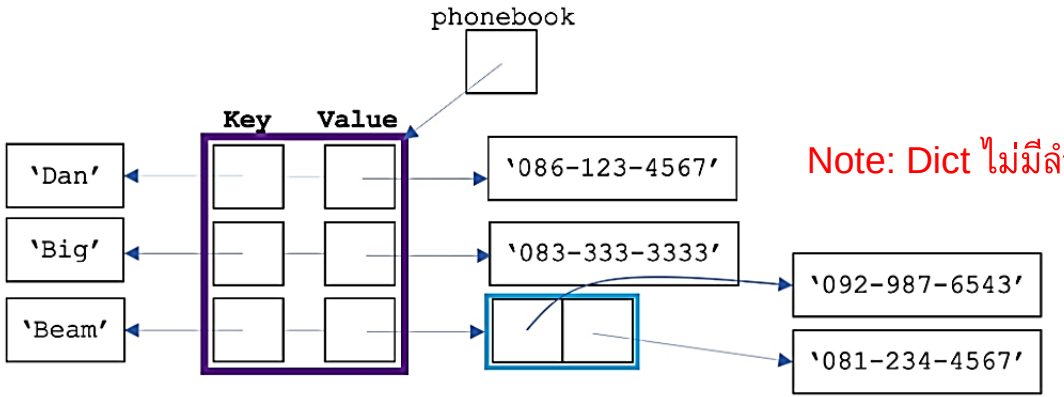
{คีย์ตัวที่ 1: ข้อมูลคู่กับคีย์ตัวที่ 1, คีย์ตัวที่ 2: ข้อมูลคู่กับคีย์ตัวที่ 2, ..., คีย์n: ข้อมูลคู่กับคีย์ตัวที่ n}

รูปแบบของ Dict

ดังนั้น เราสามารถสร้าง dict เพื่อเก็บเบอร์โทรศัพท์ ของ Dan, Beam, Big ได้ดังนี้

```
phonebook = {'Dan': '086-123-4567', 'Beam': ['092-987-6543','091-234-4567'],
             'Big':'083-333-3333'}
```

จะเห็นว่าเราใช้ dict แค่อันเดียว แทนการใช้ list สองอันในการเก็บข้อมูล



Note: Dict ไม่มีลำดับ !!

การเข้าถึงสมาชิกใน Dict ทีละตัว

ตัวแปรดิกชันนารี[คีย์ที่ต้องการ]

ตัวอย่าง phonebook

```
phonebook = {'Dan': '086-123-4567', 'Beam': ['092-987-6543','091-234-4567'],
'Big':'083-333-3333'}

phonebook['Beam']
```

*การเข้าถึงสมาชิกทุกตัวใน Dict โดยใช้ for

ตัวอย่าง phonebook

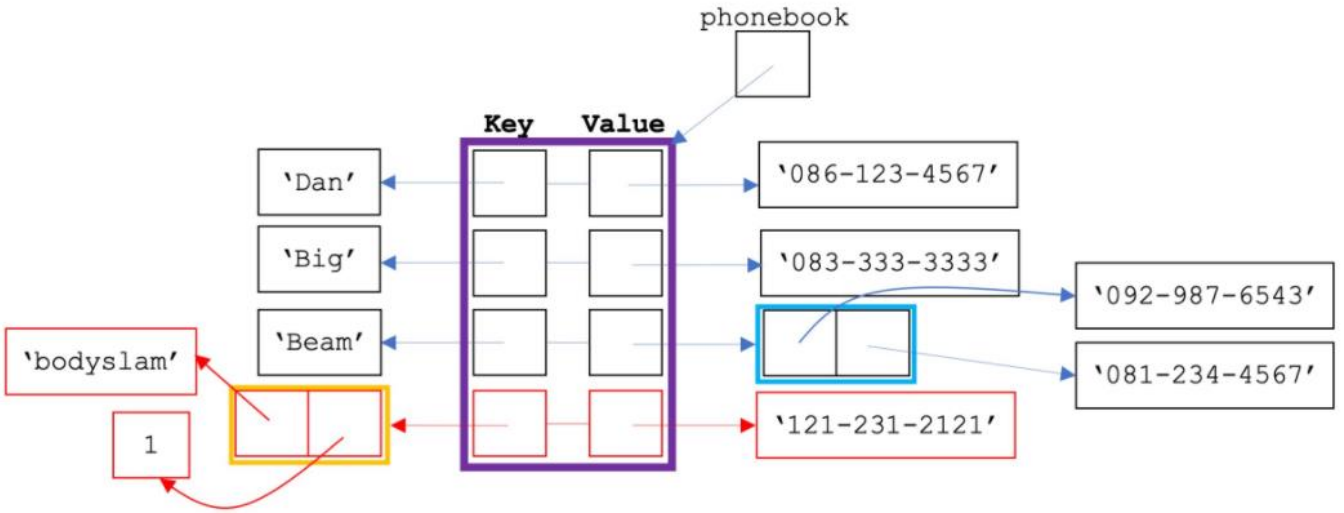
```
phonebook = {'Dan': '086-123-4567', 'Beam': ['092-987-6543','091-234-4567'],
'Big':'083-333-3333'}


for k in phonebook: # วนลูปในการเข้าถึง key แต่ละตัว
 print(k, ':',phonebook[k]) # เข้าถึง value แต่ละตัวผ่าน key
```

การเปลี่ยนแปลงข้อมูลใน Dict

- **เพิ่มข้อมูลใหม่ / การปรับปรุงค่าใน dict**

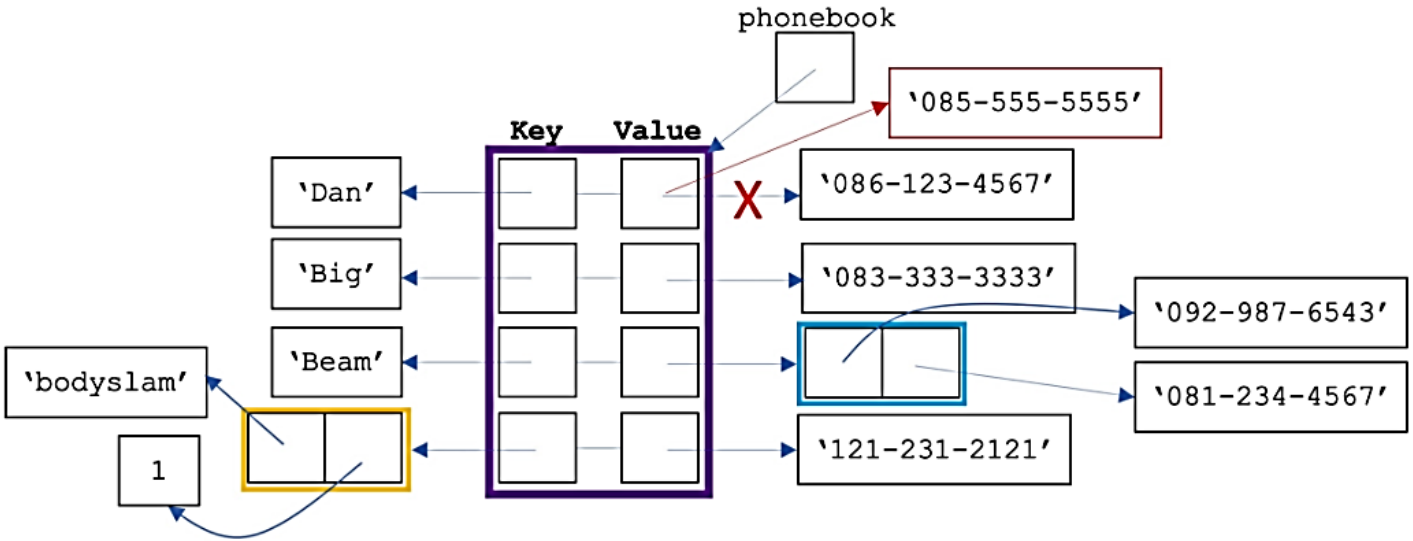
ตัวแปรดิกชันนารี[คีย์] = ค่า

```
phonebook[('bodyslam',1)] = '121-231-2121'
```



การเปลี่ยนแปลงข้อมูลใน Dict

```
phonebook[('bodyslam',1)] = '121-231-2121'
```



การเปลี่ยนแปลงข้อมูลใน Dict

- **เพิ่มข้อมูลใหม่ / การปรับปรุงค่าใน dict**

ตัวแปรดิกชันนารี[คีย์] = ค่า

Notes

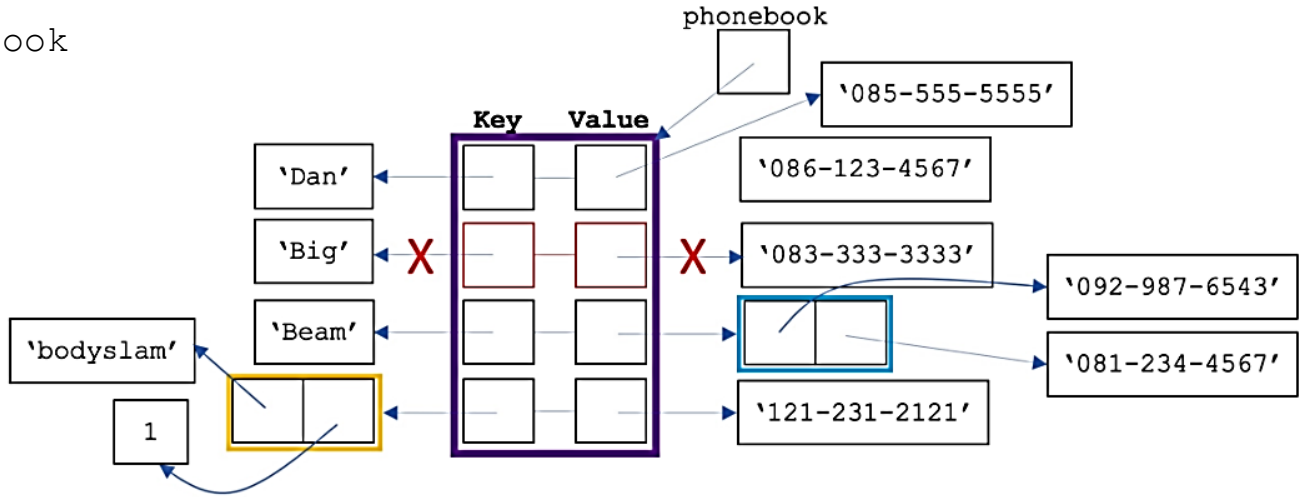
1. dict ไม่สามารถมี key ซ้ำได้
2. สิ่งที่เกิดขึ้นคือ เมื่อเราใส่ข้อมูล (value) ใหม่ลงไปใน dict โดยใช้ key ที่มีอยู่แล้ว คือ จะเป็นการเขียน value ทับค่าเดิมทันที

การเปลี่ยนแปลงข้อมูลใน Dict

- **ลบข้อมูล** del ดิกชันนารี[คีย์]

```
del phonebook['Big']
```

phonebook



การเปลี่ยนแปลงข้อมูลใน Dict

```
del phonebook['.....'] # ลบคีย์ที่ไม่มีอยู่ใน dict ก็จะเกิด Error ประเภท KeyError
del phonebook # ระวัง ! บรรทัดนี้จะเป็นการลบตัวแปร phonebook ดังนั้นข้อมูลจะหายไปหมดเลย

phonebook
# เกิด Error เนื่องจาก ตัวแปร phonebook ถูกลบออกไปก่อนหน้านี้ ทำให้เครื่องไม่รู้จักตัวแปรนี้แล้ว
```

Shallow copy

ตัวแปร = ตัวแปรดิกชันนารี

Note: เราสามารถหาค่าตำแหน่งหน่วยความจำที่เก็บตัวแปรได้โดยใช้ฟังก์ชัน id(ชื่อตัวแปร)

```
x = {3:1}
a = x # Shallow copy
a[3] = 3
a[4] = 4
a[5] = a[4]
a[4] = 44
print('a:','Memory ID',id(a),'Value:',a)
print('x:','Memory ID',id(x),'Value:',x)
```

ฟังก์ชันที่ควรรู้ของ dict

- **update เป็นฟังก์ชันที่ใช้สำหรับปรับปรุงค่าใน dict ด้วย dict อีกอัน**

ดิกชันนารีตัวแรก.update(ดิกชันนารีตัวที่สอง)

```
phonebook = {'Dan': '086-123-4567', 'Beam': ['092-987-6543','091-234-4567'],
             'Big':'083-333-3333'}
phonebook_inter = {'Blackpink':'+816731123','Dan':'+1537483203',
                   'BTS':'+816739923'}
phonebook.update(phonebook_inter)

phonebook
```

Note: ฟังก์ชัน update เป็นฟังก์ชันที่ไม่คืนค่า แต่เป็นฟังก์ชันที่เข้าไปปรับปรุงข้อมูลของ dict ตัวที่เรียกเลย (นั่นคือ phonebook ในตัวอย่างด้านบน)

*ฟังก์ชันที่ควรรู้ของ dict

- **keys เป็นฟังก์ชันที่แจกแจง key ทุกตัวใน dict**

ดิกชันนารี.keys()

```
phonebook.keys()
list(phonebook.keys())

for key in phonebook.keys():
  print(key)

for key in phonebook:
  print(key)
```

ฟังก์ชันที่ควรรู้ของ dict

- **values เป็นฟังก์ชันที่ใช้ในการแจกแจง value ทุกตัวใน dict**

ดิกชันนารี.values()

```
phonebook.values()

list(phonebook.values())

for val in phonebook.values():
  print(val)
```

ฟังก์ชันที่ควรรู้ของ dict

- **items เป็นฟังก์ชันที่ใช้ในการแจกแจงคู่ของ key และ value ทุกคู่ใน dict**

ดิกชันนารี.items()

```
phonebook.items()

list(phonebook.items())

for item in phonebook.items():
  print(type(item),item)

for key,val in phonebook.items():
  print(key,'>>>>',val)
```

ฟังก์ชันที่ควรรู้ของ dict

- **len ฟังก์ชันที่ใช้หาจำนวน items ใน dict**

len(ดิกชันนารี)

```
len(phonebook)
len({})
```

Dict comprehension

คือการสร้างดิกชันนารี โดยใช้ for ในบรรทัดเดียว เช่นเดียวกับ List comprehension

ตัวอย่าง จงสร้าง Dict ที่มีการเก็บตัวเลข 1-10 เป็น keys และมีค่า value เป็นค่ากำลังสองของ keys

```
#ใช้ for เก็บ dict แบบปกติ
num_dict = {}
for i in range(10): # range(0,10,1)
  num_dict[i] = 2*i
print(num_dict)
```

```
{0: 0, 1: 2, 2: 4, 3: 6, 4: 8, 5: 10, 6: 12, 7: 14, 8: 16, 9: 18}
```

```
#ใช้ Dict comprehension ในการสร้าง
num_dict = {i:2*i for i in range(10)}
print(num_dict)
```

```
{0: 0, 1: 2, 2: 4, 3: 6, 4: 8, 5: 10, 6: 12, 7: 14, 8: 16, 9: 18}
```

Dict หลายมิติ (2 มิติ)

สมมติว่าเราจะเก็บข้อมูลตารางด้านล่าง

	Credit	Grade	Class	Online?
Programming	3	'B+'	['M','W','F']	True
Sound and Technique	1	'A'	['Thu']	True
Design	2	'B'	'-'	False

```
[105] # แบบที่ 1 เก็บ value เป็นแบบ list
      mycourse = {}
      mycourse['Programming'] = [3,'B+',['M','W','F'],True]
      mycourse['Sound and Technique'] = [1,'A',['Thu'],True]
      mycourse['Disign'] = [2,'B','-',False]

      print(mycourse['Programming'][2])
```

```
[106] # แบบที่ 2.1 (dict ซ้อน dict)
      mycourse = {
          'Programming': {'Credit':3,'Grade':'B+','Class':['M','W','F'],'Online?': True},
          'Sound and Technique': {'Credit':1,'Grade':'A','Class':['Thu'],'Online?':True},
          'Design': {'Credit':2,'Grade':'B','Class':'-','Online?':False}
          }
      print(mycourse['Programming']['Class'])
```

```
[107] # แบบที่ 2.2
      mycourse = {}
      mycourse['Programming'] = {'Credit':3,'Grade':'B+','Class':['M','W','F'],'Online?': True}
      mycourse['Sound and Technique'] = {'Credit':1,'Grade':'A','Class':['Thu'],'Online?':True}
      mycourse['Design'] = {}
      mycourse['Design']['Class'] = '-'
      mycourse['Design']['Credit'] = 2
      mycourse['Design']['Online?'] = False
      mycourse['Design']['Grade'] = 'B'
      print(mycourse['Design']['Credit'])
```

End