

Title: ตำราวิชา Problem Solving and Computer Programming - PSCP Book

Author: รศ.ดร. โชติพัชร์ กรณวลัย

Rights: Copyright 2020-2023

Language: th-TH

Date: 17 กันยายน 2566

Ref: Think Python Chapter 11, 13

Chapter 7: Dictionary and Set

Dictionary

Dictionary จะคล้ายกับ List คือสามารถเก็บข้อมูลได้หลายชนิด แต่ต่างกันที่ index ของ Dictionary ไม่จำเป็นต้องเป็น Integer ที่เริ่มจาก 0 เมื่อกับของ List แต่ index ของ Dictionary สามารถเป็นข้อมูลชนิดอื่นได้ เช่น เป็น String หรือ Integer หรือ Float ก็ได้เป็นต้น

Index ของ Dictionary จะเรียกว่า **key** และ ข้อมูลของ **key** นั้น จะเรียกว่า **value** โดยเราจะเรียกว่า key-value pair หรือคู่ของ key กับ value หรืออาจจะเรียกว่า item เมื่อกับ List ก็ได้

เราใช้ {} เพื่อเป็นการบอกว่าข้อมูลที่จัดเก็บเป็นแบบ dictionary โดยรูปแบบ จะเป็นดังนี้

```
{key1: value1, key2: value2, key3: value3, ...}
```

โดย key และ value จะแยกหรือแบ่งกันด้วยเครื่องหมาย Colon :

ยกตัวอย่างในรูปด้านล่างนี้จะมีการเก็บค่า dictionary ซึ่ง key จะเป็น คำอ่านของเลขในภาษาอังกฤษ ส่วน value จะเป็นคำอ่านของตัวเลขในภาษาญี่ปุ่นที่จับคู่ตรงกัน

```
>>> num_eng2jp = {'one':'ichi', 'two':'ni', 'three':'san'}
>>> num_eng2jp['one']
'ichi'
>>> num_eng2jp['two']
'ni'
```

ในตัวอย่างนี้ key1 คือ 'one' และ value1 คือ 'ichi' เป็นต้น

หากต้องการทราบว่า value ของ key ที่จับคู่ตรงกันคืออะไร ก็สามารถเรียกผ่านโดยใช้ index เป็น key คล้ายๆกับที่ใช้ index ใน List หรือ String เช่น num_eng2jp['two'] ก็จะได้ค่า value ของ key 'two' คือ 'ni'

เราสามารถเพิ่ม item หรือ key-value pair เข้าไปใน dictionary ได้ ดังตัวอย่างในรูปด้านล่าง

```
>>> num_eng2jp['four'] = 'yon'
>>> num_eng2jp
{'one': 'ichi', 'two': 'ni', 'three': 'san', 'four': 'yon'}
```

ข้อมูลที่จัดเก็บใน key นั้น อาจจะเป็นข้อมูลชนิดใดก็ได้ ไม่จำเป็นต้องเป็น String อีกต่อไป เช่น จาก 'four' สามารถอ่านได้ 2 แบบในภาษาญี่ปุ่น คือ 'yon' กับ 'shi' ดังนั้น เราควรจะเก็บค่า value ของ key 'four' ด้วย List ดังตัวอย่างในรูปด้าน

ล่าง

```
>>> num_eng2jp['four'] = ['yon', 'shi']
>>> num_eng2jp
{'one': 'ichi', 'two': 'ni', 'three': 'san', 'four': ['yon', 'shi']}
```

ในตัวอย่างข้างบน หากเราให้ค่า key กับ dict ตัว dict จะให้ค่า value ของ key นั้นกลับคืนมา ในทางกลับกัน หากเราให้ค่า value กับ key ไป จะไม่ได้ค่า key กลับมา และหาก ไม่มีค่า key นั้นใน dictionary ก็จะได้ Error กลับมา ดังตัวอย่างด้านล่างนี้

```
>>> num_eng2jp['ni']
Traceback (most recent call last):
  File "<pyshell#208>", line 1, in <module>
    num_eng2jp['ni']
KeyError: 'ni'
>>> num_eng2jp['five']
Traceback (most recent call last):
  File "<pyshell#209>", line 1, in <module>
    num_eng2jp['five']
KeyError: 'five'
```

เราสามารถตรวจสอบขนาดของ dictionary ว่ามีกี่ item (key-value pair) ได้โดยใช้ funciton `len()` และสามารถใช้ `in` operator เพื่อตรวจสอบว่ามี `key` นั้นอยู่ใน dictionary หรือไม่

```
>>> num_eng2jp
{'one': 'ichi', 'two': 'ni', 'three': 'san', 'four': ['yon', 'shi']}
>>> len(num_eng2jp)
4
>>> 'three' in num_eng2jp
True
>>> 'five' in num_eng2jp
False
>>> 'ni' in num_eng2jp
False
```

ในรูปด้านบน 'ni' เป็น value ใน dictionary แต่ไม่ใช่ key ดังนั้น `in` operator จะคืนค่า `False` กลับมา

หากต้องการตรวจสอบว่า 'ni' เป็น value ใน dictionary หรือไม่ สามารถทำได้โดยใช้ method ของ dict ที่ชื่อว่า `values` เพื่อดึงค่า `values` ทั้งหมดออกมา ก่อน ดังตัวอย่างในรูปด้านล่างนี้

```
>>> num_eng2jp.values()
dict_values(['ichi', 'ni', 'san', ['yon', 'shi']])
>>> 'ni' in num_eng2jp.values()
True
```

ค่า value ของ key 'four' เป็น List ดังนั้น หากต้องการอ่านค่าทั้ง 2 ค่าของ key 'four' ออกมานา สามารถใช้ `index` ของ List ดึงค่าของแต่ละตัวออกมา ดังตัวอย่างในรูปด้านล่าง

```
>>> num_eng2jp
{'one': 'ichi', 'two': 'ni', 'three': 'san', 'four': ['yon', 'shi']}
>>> num_eng2jp['four']
['yon', 'shi']
>>> num_eng2jp['four'][0]
'yon'
>>> num_eng2jp['four'][1]
'shi'
```

การจัดเก็บข้อมูลด้วย dictionary มีความเร็วในการดึงข้อมูลออกมากว่า List มาก เนื่องจากว่ามีการใช้ Hash algorithm ใน การจัดเก็บ เรียกว่า Hash Table ดังนั้นถ้า Dictionary มีข้อมูลจำนวนมาก ก็จะใช้เวลาในการดึงค่า (value) ของ key นั้นออกมายโดยใช้เวลาแทนจะคงที่ (ไม่ขึ้นกับขนาดของ dictionary) ในขณะที่ List จะใช้เวลานานมากขึ้น ถ้าขนาดของ List มีขนาดใหญ่ขึ้น

Dictionary เป็นข้อมูลชนิด Mutable (เช่นเดียวกับ List) ลังเกตได้ว่าเราจะเพิ่ม และเปลี่ยนแปลงค่าใน Dictionary ได้ เมื่อกับที่เราเพิ่ม หรือเปลี่ยนแปลงข้อมูลใน List

Key จะใน dictionary จะต้องสามารถ Hash ได้ (hashable) เท่านั้น เนื่องจากว่าข้อมูลที่เป็น Mutable ไม่สามารถ Hash ได้ ดังนั้น List และ Dictionary ไม่สามารถเป็น Key ของ Dictionary ได้

Uses of Dictionary

เรามักใช้ Dictionary จัดเก็บข้อมูลที่มีจำนวนไม่แน่นอน และข้อมูลไม่มีลำดับ เช่น ใช้ใน function histogram เพื่อจัดเก็บค่าจำนวนของข้อมูล ยกตัวอย่างเช่น

หากเราต้องการเก็บข้อมูลว่า String 'Hello' มีตัวอักษรอะไรบ้าง และแต่ละตัวอักษรมีจำนวนเท่าใด หากเราจัดเก็บด้วย List เราอาจจะต้องใช้ List ขนาด 26 เพื่อจัดเก็บจำนวนตัวอักษรจาก 'a' ถึง 'z' โดยสมมุติว่า 'a' และ 'A' ถือว่าเป็น 'a' ทั้งหมด โดยจะมีการจัดเก็บดังนี้

```
>>> histogram_list = [0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 2, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
```

โดยเราจะใช้ index 0 เพื่อจัดเก็บจำนวนตัวอักษร 'a' และ index 1 เพื่อจัดเก็บจำนวนตัวอักษร 'b' ไปเรื่อยๆ ในตัวอย่าง ด้านบน 'e' จะมี index = 4 (เพราะ 'e' เป็นตัวอักษรลำดับที่ 5 นับจาก 'a') และมีจำนวน 1 ตัว ส่วน 'l' มี index = 11 (เพราะ 'l' เป็นตัวอักษรลำดับที่ 12 นับจาก 'a') เป็นต้น

การจัดเก็บลักษณะแบบนี้ไม่ค่อยมีประสิทธิภาพ เนื่องจากว่าคำว่า 'Hello' มีตัวอักษรเพียงแค่ 'h', 'e', 'l' และ 'o' เพียง 4 ตัวอักษร เท่านั้น แต่เราจะใช้การจัดเก็บข้อมูลถึง 26 ตัว หากใช้ List

แต่หากจัดเก็บด้วย dict จะสามารถจัดเก็บได้ดังรูปด้านล่างนี้ ซึ่งจะมีประสิทธิภาพมากกว่า

```
>>> histogram_dict = {'e': 1, 'h': 1, 'l': 2, 'o': 1}
```

Dictionary Looping

เราสามารถใช้ Loop ในการดึงข้อมูลที่อยู่ใน Dictionary ออกมายได้ แต่สำคัญมากที่จะต้องเข้าใจว่า การจัดเก็บข้อมูลใน Dict นั้น การเรียงลำดับ ข้อมูลที่จัดเก็บใน Dictionary อาจจะเป็นลำดับเดียวกันกับลำดับข้อมูลที่ป้อนเข้าไปใน Dictionary หรืออาจจะไม่เป็นลำดับเดียวกันก็ได้ ทั้งนี้ขึ้นกับ version ของ Python ที่ใช้

ตัวอย่างด้านล่างแสดงวิธีการดึงค่า key และ value จาก Dictionary โดยใช้ For loop

```

>>> num_eng2jp
{'one': 'ichi', 'two': 'ni', 'three': 'san', 'four': ['yon', 'shi']}
>>> for key in num_eng2jp:
    print(key)

one
two
three
four
>>> for key in num_eng2jp:
    print(num_eng2jp[key])

ichi
ni
san
['yon', 'shi']

```

ตัวอย่างด้านบน แม้ว่าลำดับที่แสดงโดยการ print ใน for loop จะเหมือนกับลำดับที่เราจัดเก็บข้อมูลใน Dictionary เนื่องจากว่าตัวอย่างด้านบนเป็นการใช้ Python version 3.7+ ซึ่งมีการจัดเรียงลำดับใน dict ตามลำดับที่ใส่เข้าไปใน dict แต่ในกรณีที่เราใช้ Python version ต่ำกว่า 3.7 เราไม่สามารถคาดหวังผลลัพธ์แบบนี้ได้เสมอไป

Histogram

ตัวอย่างโปรแกรม Histogram โดยใช้ Dictionary

ในรูปด้านล่างนี้ dict() เป็น function ในการสร้าง dictionary ว่าง หรือที่เรียกว่า Empty Dict ขึ้นมาก่อน โดยเก็บในตัวแปร d และจะมีค่าเป็น {}

```

>>> d = dict()
>>> d
{}
>>> s = 'Hello'
>>> if s[0] in d:
    d[s[0]] += 1
else:
    d[s[0]] = 1

>>> d
{'H': 1}

```

จากนั้น จะต้องตรวจสอบโดยใช้ if ดูก่อนว่า สิ่งที่เราจะใส่เข้าไปใน dict ชิ่งก็คือ key-value pair นั้น มีค่า key นั้นอยู่แล้ว หรือยัง ถ้าหากมีค่าอยู่แล้ว ซึ่งในที่นี้คือจำนวนของตัวอักษรของ key นั้น ก็จะให้เพิ่มค่าเข้าไปอีก 1 และหากยังไม่มีค่า key นั้นอยู่ (ซึ่งก็คือเงื่อนไข else) ก็แสดงว่าเป็นการใส่ key-value pair นั้นเข้าไปเป็นครั้งแรกใน Dictionary d ดังนั้นก็จะให้มีค่าเป็น value เป็น 1 (เจอ key นี้ครั้งแรก)

หากเราพยายามเพิ่มค่า value เข้าไปใน dict เลย โดยที่ยังไม่มี key นั้นอยู่ก่อน ก็จะทำให้มี Error แบบ KeyError เกิดขึ้น

```
>>> d = dict()
>>> d
{}
>>> s = 'Hello'
>>> d[s[0]] += 1
Traceback (most recent call last):
  File "<pyshell#271>", line 1, in <module>
    d[s[0]] += 1
KeyError: 'H'
```

หากต้องการนำตัวอักษรทุกตัวใน String s เพื่อนับค่าใน Histogram ก็จะสามารถเขียนได้ดังรูปด้านล่างนี้ โดยการใช้ Loop for เพื่ออ่านค่าตัวอักษรแต่ละตัว (c) ใน s และไปตรวจสอบว่ามีตัวอักษรแต่ละตัวนั้น (c) มีอยู่ใน dictionary d แล้ว หรือไม่ ถ้าไม่มีก็ให้ใส่ key นั้นเข้าไป โดยมีค่า value = 1 ด้วย statement `d[c] = 1` แต่หากมี key ที่เป็นตัวอักษร c นั้นอยู่ใน dict และ ก็ให้เพิ่มค่าจากเดิมเข้าไป `d[c] += 1`

```
>>> d = {}
>>> for c in s:
...     if c in d:
...         d[c] += 1
...     else:
...         d[c] = 1

>>> d
{'H': 1, 'e': 1, 'l': 2, 'o': 1}
```

ในตัวอย่างรูปด้านบน เราสามารถกำหนดให้ d เป็น Empty dict โดยการเขียน `d = {}` ซึ่งจะได้ผลลัพธ์เดียวกันกับเขียน `d = dict()`

```
>>> d = {}
>>> for c in s:
...     d[c] = d.get(c, 0) + 1
...
...
...
>>> d
{'H': 1, 'e': 1, 'l': 2, 'o': 1}
```

เราสามารถใช้ .get method และลด for loop และ if-else จากรูปบนได้ .get method จะดึงค่าข้อมูลโดยใช้ key ถ้าหากไม่เจอ จะไม่เกิด Key Error แต่จะล่งค่า Default กลับมา ในตัวอย่างด้านบน หากไม่เจอ key c ก็จะคืนค่า 0 กลับมาให้

Dictionary Methods

Method สำหรับการเพิ่มหรือเปลี่ยนแปลงข้อมูล (Key-Value Pair) ใน Dictionary

- `update()`
- ```
>>> cars = {"brand": "Toyota", "model": "camry"}
>>> cars.update({"year": 2000})
>>> cars
{'brand': 'Toyota', 'model': 'camry', 'year': 2000}
>>> cars.update({"year": 2020, "color": "bronze"})
>>> cars
{'brand': 'Toyota', 'model': 'camry', 'year': 2020, 'color': 'bronze'}
```

### Method สำหรับการดึงข้อมูล (Key-Value Pair หรือ Keys หรือ Values) ใน Dictionary

- get()
- setdefault()

```
>>> cars
{'brand': 'Toyota', 'model': 'camry', 'year': 2020, 'color': 'bronze'}
>>> cars['count'] = cars.get('count', 0)
>>> cars
{'brand': 'Toyota', 'model': 'camry', 'year': 2020, 'color': 'bronze', 'count': 0}
>>>
>>> cars.setdefault('engine', '21R')
'21R'
>>> cars
{'brand': 'Toyota', 'model': 'camry', 'year': 2020, 'color': 'bronze', 'count': 0, 'engine': '21R'}
>>> cars.setdefault('model', 'altis')
'camry'
>>> cars
{'brand': 'Toyota', 'model': 'camry', 'year': 2020, 'color': 'bronze', 'count': 0, 'engine': '21R'}
>>>
```

ทั้ง get() และ setdefault() มีความคล้ายกันมาก แต่หากต้องการดึงค่าข้อมูลเพียงอย่างเดียว สามารถใช้ get() ได้ แต่หากจะต้องการเปลี่ยนค่าใน dict ด้วย setdefault() จะทำงานได้เร็วกว่า

- items() เราสามารถใช้ .items() ในการสร้างข้อมูล list of tuples จากข้อมูล dictionary ได้ ดังตัวอย่างด้านล่าง และในทางการกลับกันสามารถใช้ function dict() ในการแปลง list of tuples กลับไปเป็น dictionary ดังเดิมได้ด้วย
- keys()
- values()

```
>>> cars = {'brand': 'Toyota', 'model': 'camry', 'year': 2020}
>>> cars.items()
dict_items([('brand', 'Toyota'), ('model', 'camry'), ('year', 2020)])
>>> cars.keys()
dict_keys(['brand', 'model', 'year'])
>>> cars.values()
dict_values(['Toyota', 'camry', 2020])
>>>
```

## Method สำหรับลบข้อมูล (Key-Value Pair) ใน Dictionary

- pop()
- popitem()
- clear()

```
>>> cars
{'brand': 'Toyota', 'model': 'camry', 'year': 2020, 'color': 'bronze', 'count': 0, 'engine': '21R'}
>>> year = cars.pop('year')
>>> year
2020
>>> cars
{'brand': 'Toyota', 'model': 'camry', 'color': 'bronze', 'count': 0, 'engine': '21R'}
>>> lastitem = cars.popitem()
>>> lastitem
('engine', '21R')
>>> cars
{'brand': 'Toyota', 'model': 'camry', 'color': 'bronze', 'count': 0}
>>>
>>> cars.clear()
>>> cars
{}
>>>
```

## Method สำหรับการสร้าง Dictionary ใหม่

- fromkeys()

```
>>> countries = ('Thailand', 'Japan', 'China')
>>> cars_sales = dict.fromkeys(countries, 0)
>>> cars_sales
{'Thailand': 0, 'Japan': 0, 'China': 0}
>>>
```
- copy()

```
>>> cars = {'brand': 'Toyota', 'model': 'camry'}
>>> new = cars.copy()
>>> new
{'brand': 'Toyota', 'model': 'camry'}
>>> cars == new
True
>>> cars is new
False
>>> new.clear()
>>> new
{}
>>> cars
{'brand': 'Toyota', 'model': 'camry'}
>>>
>>> cars = {'brand': 'Toyota', 'model': 'camry'}
>>> new = cars
>>> new
{'brand': 'Toyota', 'model': 'camry'}
>>> cars == new
True
>>> cars is new
True
>>> new.clear()
>>> new
{}
>>> cars
{}
```

## Memos

$$\text{fibonacci}(0) = 0$$

$$\text{fibonacci}(1) = 1$$

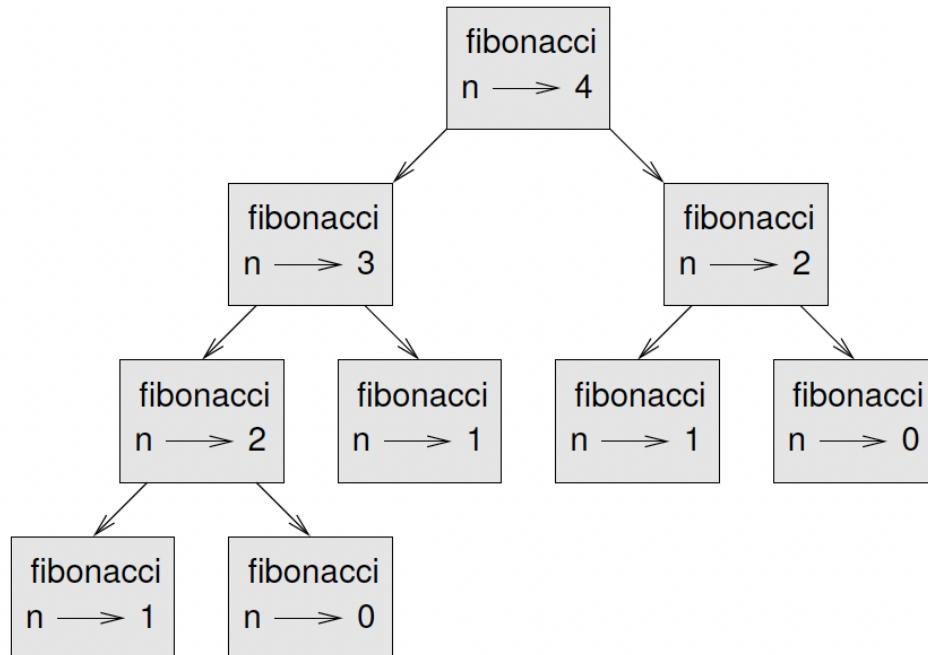
$$\text{fibonacci}(n) = \text{fibonacci}(n - 1) + \text{fibonacci}(n - 2)$$

การคำนวณค่า Fibonacci ลำดับที่  $n$  จะมีค่าเป็นไปตามรูปด้านบน ตามนิยาม ค่า Fibonacci ลำดับที่ 0 จะมีค่าเป็น 0 และ ลำดับที่ 1 มีค่าเป็น 1 และลำดับถัดไปมีค่าเท่ากับค่าของ 2 ลำดับก่อนหน้ามาบวกกัน

```
>>> def fibonacci(n):
... if n == 0:
... return 0
... elif n == 1:
... return 1
... else:
... return fibonacci(n-1) + fibonacci(n-2)
...
...
...
>>> fibonacci(5)
5
>>> fibonacci(8)
21
>>>
```

หากเราเขียนโปรแกรมตามนิยามทางคณิตศาสตร์ สามารถเขียนได้ดังรูปด้านบน ซึ่งมีลักษณะการเขียนคล้ายกับนิยามทางคณิตศาสตร์เป็นอย่างมาก และจะสังเกตว่าใน function fibonacci มีการเรียก fibonacci ซึ่งเป็น function ที่มีการเรียกตัวเองช้า

Function ที่มีการเรียกตัวเองช้าในลักษณะนี้ จะเรียกว่า **Recursive Function** แต่การเรียก Function ตัวเองช้าๆ จะมีต้องมีเงื่อนไขในการหยุด ไม่เช่นนั้น ก็จะมีการเรียกช้าไปเรื่อยๆ ไม่มีที่สิ้นสุด ในกรณีของ Function Fibonacci ในรูปด้านบน จะมี base case ซึ่งเป็นเงื่อนไขที่ทำให้หยุดการเรียกตัวเองช้าๆ อยู่ 2 base case ได้แก่ `if n == 0` และ `elif n == 1` จะสังเกตว่าทั้ง 2 เงื่อนไขนี้จะไม่มีการ function ตัวเองช้า



อย่างไรก็ตาม แม้ว่าปัญหานางปัญหา เช่น Fibonacci จะสามารถเขียนโปรแกรมได้ง่าย โดยวิธีการเขียนแบบ Recursion แต่ว่า การเขียนแบบ Recursion จะมีปัญหาระบบที่ซ้ำกันของโปรแกรม เนื่องจากมีแนวโน้มจะใช้เวลาในการคำนวณนานมาก ดังสังเกตได้จากตัวอย่างในรูปด้านบนที่แสดง Call Stack เมื่อมีการ run `fibonacci(4)` จะเห็นได้ว่ามีการ call `fibonacci(3)` `fibonacci(2)` `fibonacci(1)` และ `fibonacci(0)` จำนวนหลายครั้ง ทำให้ใช้เวลาในการคำนวณนาน (ซึ่งจริงๆ ควร call แค่ครั้งเดียว ก็พอ)

```
>>> known = {0:0, 1:1}
>>> def fibonacci_memos(n):
... if n in known:
... return known[n]
... res = fibonacci_memos(n-1) + fibonacci_memos(n-2)
... known[n] = res
... return res
...
>>> fibonacci_memos(5)
5
>>> fibonacci_memos(8)
21
>>> known
{0: 0, 1: 1, 2: 1, 3: 2, 4: 3, 5: 5, 6: 8, 7: 13, 8: 21}
>>>
```

วิธีการแก้ปัญหาดังกล่าว สามารถทำได้โดยการใช้ Global variable ชื่อเป็น variable ที่ได้มีการกำหนดค่าไว้นอก function ในตัวอย่างในรูปด้านบน มีการสร้าง Global variable ชื่อ known เป็น Dictionary เก็บค่า fibonacci ลำดับที่ 0 และ 1 ไว้เป็นค่าเริ่มต้น โดย key เป็นลำดับของ fibonacci และ value เป็นค่าของ fibonacci ที่ลำดับของ key นั้น

ค่า known จะมีการ update ทุกครั้งมีหาค่าได้ ใน Function fibonacci และก่อนที่จะมีการ Recursive call ก็จะทำการตรวจสอบว่า fibonacci ที่ต้องการหาค่านั้นเก็บค่าใน known แล้วหรือยัง หากเก็บใน known แล้ว ก็นำค่าไปใช้ได้เลย แต่หากไม่มีจึงค่อย call stack recursive ไป การทำเช่นนี้ทำให้ลดเวลาการ call stack ของ fibonacci ค่าเดิม ข้ามหลายๆ รอบ ทำให้โปรแกรมทำงานได้เร็วมากขึ้น วิธีการแบบนี้เราระบุว่า memos

```
>>> def timer(n, f):
... start = timeit.default_timer()
... print("Start time =", start)
... f(n)
... stop = timeit.default_timer()
... print("Stop time =", stop)
... print("Compute time =", stop - start)
...
...
>>> timer(30, fibonacci)
Start time = 28274.937231555
Stop time = 28275.164961579
Compute time = 0.22773002400208497
>>>
>>> timer(30, fibonacci_memos)
Start time = 28285.971396368
Stop time = 28286.05202591
Compute time = 0.08062954200067907
>>>
>>> timer(40, fibonacci)
Start time = 28297.204167451
Stop time = 28313.751139073
Compute time = 16.546971621999546
>>>
>>> timer(40, fibonacci_memos)
Start time = 28321.404786335
Stop time = 28321.503093227
Compute time = 0.09830689199952758
>>>
```

รูปด้านบนแสดง function ที่ใช้ในการจับเวลาว่า หาก run fibonacci ที่ไม่มี memos กับ fibonacci ที่มี memos จะใช้เวลาแตกต่างกันมากเพียงใด จะเห็นได้ว่าหาก n มีค่า 40 เวลาที่ใช้ในการคำนวณของทั้ง 2 วิธีมีความแตกต่างกันอย่าง

မာဂ

## Type Checking

```
>>> fibonacci(1.5)
Traceback (most recent call last):
 File "<pyshell#179>", line 1, in <module>
 fibonacci(1.5)
 File "<pyshell#178>", line 7, in fibonacci
 return fibonacci(n-1) + fibonacci(n-2)
 File "<pyshell#178>", line 7, in fibonacci
 return fibonacci(n-1) + fibonacci(n-2)
 File "<pyshell#178>", line 7, in fibonacci
 return fibonacci(n-1) + fibonacci(n-2)
 [Previous line repeated 1022 more times]
 File "<pyshell#178>", line 2, in fibonacci
 if n == 0:
 RecursionError: maximum recursion depth exceeded in comparison
>>>
>>> fibonacci(-5)
Traceback (most recent call last):
 File "<pyshell#181>", line 1, in <module>
 fibonacci(-5)
 File "<pyshell#178>", line 7, in fibonacci
 return fibonacci(n-1) + fibonacci(n-2)
 File "<pyshell#178>", line 7, in fibonacci
 return fibonacci(n-1) + fibonacci(n-2)
 File "<pyshell#178>", line 7, in fibonacci
 return fibonacci(n-1) + fibonacci(n-2)
 [Previous line repeated 1023 more times]
 RecursionError: maximum recursion depth exceeded
>>>

>>> def fibonacci(n):
... if not isinstance(n, int):
... print("Error: n must be int")
... return None
... elif n < 0:
... print("Error: n must be >= 0")
... return None
... elif n == 0:
... return 0
... elif n == 1:
... return 1
... else:
... return fibonacci(n-1) + fibonacci(n-2)
...
...
>>> fibonacci(1.5)
Error: n must be int
>>> fibonacci(-5)
Error: n must be >= 0
>>>
```

## Global Variables

### Immutable Type

```

>>> global_var = True
>>> def example1():
... print(global_var)
...
...
...
>>> example1()
True
>>> global_var
True
>>>
>>> def example2():
... global_var = False
... print(global_var)
...
...
...
>>> example2()
False
>>> global_var
True
>>>

```

การ Assign ค่าใหม่ให้กับตัวแปรชื่อเดียวกับ Global variable ใน function ไม่มีผลกับ Global variable เนื่องจาก function จะมีสร้าง Local variable ชื่อเดียวกับ Global variable ขึ้นมาใน function ดังแสดงในรูปด้านบน

```

>>> def example3():
... global global_var
... global_var = False
... print(global_var)
...
...
...
>>> example3()
False
>>> global_var
False
>>>

```

หากต้องการ Assign Global variable เป็นค่าใหม่ ภายใน function ต้องมีการประกาศว่าตัวแปรนั้นเป็นตัวแปรแบบ Global ก่อน ด้วยการเขียน `global` นำหน้าตัวแปรที่ต้องการระบุว่าเป็น Global variable ก่อน assign ค่าใหม่ ดังแสดงในรูปด้านบน

## Mutable Type

ในกรณีของ Mutable data เช่น List หรือ Dictionary เราสามารถเปลี่ยนแปลงค่าของ Global variable ภายใน function ได้เลย แต่หากเป็นการ Assign ค่าใหม่ จะต้องประกาศตัวแปรนั้นเป็น global ก่อนในลักษณะเดียวกับ Immutable data type

```
>>> global_var = [1,2,3]
>>> def example4():
... global_var.append(4)
... print(global_var)
...
...
...
>>> example4()
[1, 2, 3, 4]
>>> global_var
[1, 2, 3, 4]
>>>
>>> def example5():
... global_var = [1,2,3,4,5]
... print(global_var)
...
...
...
>>> example5()
[1, 2, 3, 4, 5]
>>> global_var
[1, 2, 3, 4]
>>>
>>> def example6():
... global global_var
... global_var = [1,2,3,4,5]
... print(global_var)
...
...
...
>>> example6()
[1, 2, 3, 4, 5]
>>> global_var
[1, 2, 3, 4, 5]
>>>
```

## Set

เป็นการจัดเก็บข้อมูลอีกรูปแบบหนึ่ง โดยใช้เครื่องหมาย { } เช่นเดียวกับ Dictionary แต่ว่าสิ่งที่เก็บจะไม่ใช่ key-value pair แต่จะเป็น item อะไรก็ได้ คล้ายๆกับ List แต่จะต่างกับ List ตรงที่ สิ่งที่จัดเก็บใน Set ถ้าซ้ำกันก็จะจัดเก็บค่าครั้งเดียวเท่านั้น

ตัวอย่างเช่น

```
>>> name_set = {'Alice', 'Bob', 'Carol'}
>>> name_set
{'Bob', 'Carol', 'Alice'}
>>> name_set = {'Alice', 'Bob', 'Carol', 'Bob'}
>>> name_set
{'Bob', 'Carol', 'Alice'}
>>> name_set[0]
Traceback (most recent call last):
 File "<pyshell#281>", line 1, in <module>
 name_set[0]
TypeError: 'set' object is not subscriptable
```

ในรูปด้านบน จะเห็นได้ว่า หากใน Set มีค่า 'Bob' 2 ครั้ง ก็จะจัดเก็บค่า 'Bob' เพียงครั้งเดียว และข้อมูลใน Set ไม่มีการเรียงลำดับ หากดูตอนเรากำหนดค่า Set จะเห็นว่า เราใส่ Alice ตามด้วย Bob และตามด้วย Carol แต่การจัดเก็บ จะเก็บ Bob ก่อน ตามด้วย Carol และ Alice ดังนั้นการดึงข้อมูลใน set ด้วย index แบบใน List ไม่สามารถทำได้ หากเราอ้างด้วย index เช่นในรูปด้านบน จะได้ TypeError กลับมา

อย่างไรก็ตาม Set เป็นข้อมูลชนิด Mutable Type เหมือนกับ List ดังนั้นเราจึงสามารถเปลี่ยนแปลงค่าใน Set ได้ การเปลี่ยนแปลงค่าใน Set นั้น สามารถทำได้หลายวิธี ตามหลักการทำงานคณิตศาสตร์ที่เกี่ยวกับทฤษฎีของ Set เช่น การ Union การ Intersection เป็นต้น

ตัวอย่างด้านล่างเป็นการ union set 2 set (setA และ setB) เข้าด้วยกัน

```
>>> setA = {11, 22, 33}
>>> setB = {33, 44, 55}
>>> setA + setB
Traceback (most recent call last):
 File "<pyshell#285>", line 1, in <module>
 setA + setB
TypeError: unsupported operand type(s) for +: 'set' and 'set'
>>> setA | setB
{33, 22, 55, 11, 44}
>>> setA
{33, 11, 22}
>>> setB
{33, 44, 55}
>>> setC = setA | setB
>>> setC
{33, 22, 55, 11, 44}
>>> setC = setA.union(setB)
>>> setC
{33, 22, 55, 11, 44}
```

การ union คือการนำสมาชิกของทั้ง 2 set มารวมกัน โดยตัวที่ซ้ำกัน ก็จะนับเป็นครั้งเดียว โดยการ union สามารถทำได้ 2 วิธี คือการใช้ operator | และการใช้ union method

จากตัวอย่างในรูปด้านบน จะเห็นได้ว่าเราไม่สามารถใช้ operator + ในการ union Set ได้

ตัวอย่างในรูปด้านล่าง แสดงวิธีการ Intersection setA และ setB เข้าด้วยกัน สามารถใช้ operator & หรือจะใช้ method intersection ก็ได้

```
>>> setA
{33, 11, 22}
>>> setB
{33, 44, 55}
>>> setA & setB
{33}
>>> setC = setA & setB
>>> setC
{33}
>>> setC = setA.intersection(setB)
>>> setC
{33}
```

เราสามารถหาสิ่งที่อยู่ใน SetA แต่ไม่อยู่ใน setB ได้ เราเรียกว่า Set Difference โดยสามารถใช้ operator - หรือใช้ method difference ได้ ดังตัวอย่างในรูปด้านล่าง

```
>>> setA
{33, 11, 22}
>>> setB
{33, 44, 55}
>>> setC = setA - setB
>>> setC
{11, 22}
>>> setC = setA.difference(setB)
>>> setC
{11, 22}
```

และเราสามารถหาสิ่งที่มีใน setA และ setB แต่ว่าต้องไม่มีซ้ำกันใน setA และ setB ได้ เราเรียกว่า Symmetric Difference โดยสามารถใช้ operator ^ หรือใช้ method symmetric\_difference ได้ ดังตัวอย่างในรูปด้านล่าง

```
>>> setA
{33, 11, 22}
>>> setB
{33, 44, 55}
>>> setC = setA ^ setB
>>> setC
{11, 44, 22, 55}
>>> setC = setA.symmetric_difference(setB)
>>> setC
{11, 44, 22, 55}
```

การหา Symmetric Difference ของ setA และ setB สามารถหาได้จากการหา Difference ของ SetA union กับ Set B กับ setA intersection กับ setB ดังรูปด้านล่าง

```
>>> setC = (setA | setB) - (setA & setB)
>>> setC
{11, 44, 22, 55}
```

เราสามารถสร้าง set ว่าง หรือเรียกว่า Empty set ได้โดยการใช้ function set() และสามารถแปลงข้อมูลชนิดอื่น เช่น List ให้อยู่ในรูปของ set ได้ด้วย function set() เช่นเดียวกัน ตามตัวอย่างในรูปด้านล่าง

```
>>> setA = set()
>>> setA
set()
>>> t = [11, 22, 33, 11, 22]
>>> setA = set(t)
>>> setA
{33, 11, 22}
>>> s = 'Hello'
>>> setA = set(s)
>>> setA
{'l', 'H', 'o', 'e'}
```

เนื่องจาก set ใช้เครื่องหมาย {} เพื่อ区กับ dict ดังนั้น จึงควรระวังว่า {} จะหมายถึง Empty dict ไม่ใช่ Empty set การสร้าง Empty set จะต้องใช้ set() เท่านั้น

นอกจากการ union, intersection แล้ว Set ยังมี method อื่นๆ มีหลาย method เช่น

- method add เพื่อการเพิ่มสมาชิกใน set
- method remove เพื่อนำสมาชิกออกจาก set
- method clear เพื่อลบสมาชิกใน set ทั้งหมดออกไป เพื่อให้เป็น Empty set
- method copy เพื่อ copy set
- method issubset เพื่อตรวจสอบว่าเป็น subset หรือไม่

ตัวอย่างประกอบการใช้งาน method ข้างต้น สามารถดูได้ในรูปด้านล่างนี้

```
>>> setA = set([1,2,3])
>>> setA
{1, 2, 3}
>>> setA.add(4)
>>> setA
{1, 2, 3, 4}
>>> setA.remove(3)
>>> setA
{1, 2, 4}
>>> setB = setA.copy()
>>> setB
{1, 2, 4}
>>> setA
{1, 2, 4}
>>> setA.clear()
>>> setA
set()
>>> setB
{1, 2, 4}
>>> setC = {1, 2}
>>> setC.issubset(setB)
True
>>> setB.issubset(setC)
False
```

Method อื่นๆ ที่มีใน set สามารถตรวจสอบได้โดยการพิมพ์ `help(set)` ใน Python Interpreter