Dictionary, Array, Set

Chatchai Kasemtaweechok

Dictionaries

- Dictionary เป็นชนิดข้อมูลที่คล้ายอาร์เรย์ (arrays) ซึ่งประกอบด้วย **keys** และ **values** แทนที่การใช้ค่าดัชนี (index)
- ในแต่ละค่าของ dictionary จะเข้าถึงได้โดยการระบุ key ซึ่งจะเป็นค่า string หรือ number หรือ list ก็ได้
- เราสามารถวนลูปเพื่อจัดการ dictionaries ได้ แต่การดึงจะไม่เรียงลำดับตามตัวอักษรของ key และต้องระบุค่า key ให้ถูกต้อง
- Dictionary ใช้ตัวอักขระปีกกา { } และระบุรายการด้วยค่า key และ value

```
# Create phonebook dictionaries

phonebook = {}
phonebook["John"] = 938477566
phonebook["Jack"] = 938377264
phonebook["Jill"] = 947662781

print(phonebook)
```

```
# Create phonebook dictionaries

phonebook = {
    "John" : 938477566,
    "Jack" : 938377264,
    "Jill" : 947662781
}

print(phonebook)
```

Keys	John	Jack	Jill
Values	938477566	938377264	947662781

Dictionaries

- รายการใน dictionary สามารถผสมข้อมูลมากกว่าหนึ่งประเภทข้อมูล (string, int, float, list) ใน dictionary เดียวกันได้
- คำสั่ง type แสดงประเภทของข้อมูล dictionary
- คำสั่ง len แสดงจำนวนรายการใน dictionary

```
car = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}

print (car)
print(type(car))
print(len(car))
```

Dictionaries

Method	Description
clear()	ลบรายการทั้งหมดใน dictionary
copy()	คัดลอกรายการใน dictionary
get()	แสดงค่า value ในค่า key ที่กำหนดไว้
items()	แสดงรายการ tuple ที่มีคู่ลำดับ key และ value
keys()	แสดงรายการ key ทั้งหมด
pop()	ลบรายการในค่า key ที่กำหนดไว้
popitem()	ลบรายการสุดท้าย
update()	เพิ่มหรือแก้ไขรายการตามค่า key ที่กำหนดไว้
values()	แสดงรายการค่า values ใน dictionary

Dictionaries

- การเข้าถึงค่าข้อมูลในแต่ละ Key สามารถทำได้โดย
  1. การใช้สัญลักษณ์ [ ] และระบุค่า key ที่ต้องการ
  2. การใช้ method ที่มีชื่อว่า "get" และระบุค่า key ที่ต้องการ

```
car = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}

# Access value from key
x = car["brand"]
y = car.get("year")

print(x, y)
```

Dictionaries

แสดงรายการ key ทั้งหมดใน dictionary	k = car,keys()
แสดงรายการ value ทั้งหมดใน dictionary	v = car.values()
แสดงรายการคู่ key และ value ทั้งหมด	i = car.items()

```
car = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}

k = car.keys()
v = car.values()
i = car.items()

print (k)
print(v)
print (i)
```

Dictionaries

- Iterating over dictionaries

```
#เพิ่มรายการใน dictionary

phonebook["chatchai"] = 938377264
phonebook.update({"somchai": 15620020})

print (phonebook)
```

```
#แสดงรายการ key และ value ใน dictionary

for name, number in phonebook.items():
    print("Phone number of %s is %d" %
(name, number))
```

```
# ลบรายการ john ออกจาก phonebook

del phonebook["John"]
phonebook.pop("John")

print(phonebook)
```

```
# ตรวจสอบว่า chatchai เป็นคีย์อยู่ใน dictionary

if "chatchai" in phonebook :
    print ("Key found")
 else:
    print ("No keys found")
```

Dictionaries

- **Iterating over dictionaries**

```
#แสดงรายการ key ใน dictionary

for key in phonebook:
    print("Phone name of %s " % (key))
```

```
#แสดงรายการ key ใน dictionary

for name in phonebook.keys():
    print("Phone name of %s " % (name))
```

```
#แสดงรายการ value ใน dictionary

for key in phonebook:
    print("Phone number is %d" %
phonebook[key])
```

```
#แสดงรายการ value ใน dictionary

for number in phonebook.values():
    print("Phone number is %d" % (number))
```

Two-dimensional lists

```
a = [[1, 2, 3], [4, 5, 6]]
print(a[0])
print(a[1])

b = a[0]
print(b)

print(a[0][2])
```

```
a[0][1] = 7
print(a)
print(b)

b[2] = 9
print(a[0])
print(b)
```

NumPy Creating Arrays

- Numpy ใช้ array ในการทำงานโดยกำหนดออบเจกต์ประเภทนี้ว่า "ndarray"
- เราสามารถสร้าง Numpy ndarray ได้ด้วยฟังก์ชัน array()

create ndarray from List

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)
print(type(arr))
```

create ndarray from Tuple

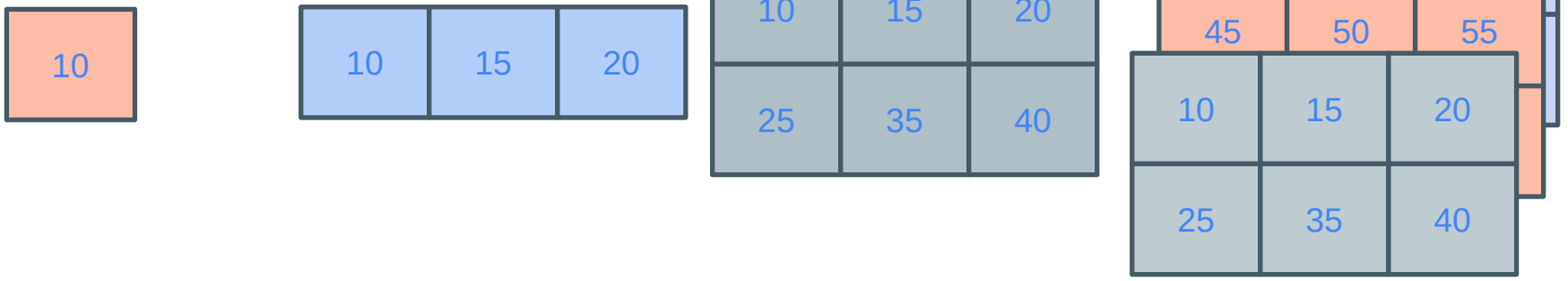
```
import numpy as np

arr = np.array((1, 2 , 3, 4, 5))

print(arr)
print(type(arr))
```

มิติของอาร์เรย์ (Dimensions in Arrays)

0-D Arrays	1-D Arrays	2-D Arrays	3-D Arrays
• Scalar • เก็บค่าเพียงค่าเดียว	• Uni-dimensional or 1-D array • เป็น array พื้นฐานที่พบเห็นบ่อย ๆ ในโปรแกรมคอมพิวเตอร์	• Matrix or 2nd Order tersors • ตารางสองมิติมีแถวและคอลัมน์หลายช่อง	• 3rd order tensor • เป็นการเอาตารางสองมิติหลายตัวมารวมกัน



Higher Dimensional Arrays

- เราสามารถสร้าง array ที่มีขนาดมิติเท่าไหร่ก็ได้
- ตอนที่สร้างอาร์เรย์ เรากำหนดขนาดของมิติของอาร์เรย์ผ่านการใช้อาร์กูเมนท์ ndmin
- เราสามารถตรวจสอบจำนวนมิติของอาร์เรย์ได้โดยการใช้แอททริบิวต์ ndim

```
import numpy as np
arr = np.array([1, 2, 3, 4], ndmin=5)
print(arr)
print('number of dimensions :', arr.ndim)
```

```
import numpy as np
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(b.ndim)
print(c.ndim)
print(d.ndim
```

Access Array

- การอ้างอิงและเข้าถึงค่าภายใน array จะใช้เครื่องหมาย {] (Square Bracket) และระบุหมายเลขตำแหน่ง (index) ที่ต้องการ โดยที่ตำแหน่งแรกจะเริ่มที่ศูนย์

```
import numpy as np

arr1 = np.array([10, 20, 30, 40, 50])
print('4th element : ', arr1[3])

arr2 = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('2nd element on 1st dim: ', arr2[0, 1])
print('5th element on 2nd dim: ', arr2[1, 4])

arr3 = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr3[0, 1, 2])
```

Array Slicing

- การตัดแบ่งส่วนอาร์เรย์เป็นการนำบางส่วนของอาร์เรย์ออกมาใช้งานจากจุดตำแหน่งเริ่มต้นจนถึงอีกตำแหน่งหนึ่ง
    - [start : end]
    - [start : end : step]

- ถ้าไม่ระบุค่า Start จะถือว่า start จะมีค่าเท่ากับ 0
- ถ้าไม่ระบุค่า end จะถือว่า end จะมีค่าเท่ากับความยาวทั้งหมดของอาร์เรย์
- ถ้าไม่ระบุค่า step จะถือว่า step จะมีค่าเท่ากับ 1

Array Slicing

```
import numpy as np

arr1 = np.array([10, 20, 30, 40, 50])
print('2nd - 4th ', arr1[1:3])
print('2nd to end : ', arr1[2:])
print('start to 4th : ', arr1[:3])
print('end to 2th : ', arr1[-4:-1])

#Add Step
print('2nd - 4th step 2', arr1[1:3:2])
print('step 2', arr1[::2])

arr2 = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('2nd element on 1st – 2nd  dim: ', arr2[0:2, 1])
print('2nd - 5th  element on 2nd  dim: ', arr2[1, 1:4])
print('2nd - 5th  element on 1st – 2nd dim: ', arr2[0:2, 1:4])
```

Set

- เซต (set) เป็นประเภทข้อมูลที่คล้ายกับนิยามเซตในคณิตศาสตร์
- คุณสามารถเพิ่มหรือลบสมาชิกภายในเซต ด้วยตัวดำเนินการทางเซตต่าง ๆ เช่น union, intersection, difference
- คุณสมบัติสำคัญของเซต คือ สมาชิกทุกตัวในเซตใด ๆ ต้องไม่มีคำซ้ำกัน
- การสร้าง Set จะระบุสมาชิกในวงเล็บปีกกา { } หรือใช้ฟังก์ชัน Set

```
A = {1, 2, 3}
B = set('qwerty')
print(A)
print(B)
```

https://snakify.org/en/lessons/sets/

Operations in Set

A | B **A.union(B)**	Returns a set which is the union of sets `A` and `B`.
A |= B **A.update(B)**	Adds all elements of array `B` to the set `A`.
A & B **A.intersection(B)**	Returns a set which is the intersection of sets `A` and `B`.
A &= B **A.intersection_update(B)**	Leaves in the set `A` only items that belong to the set `B`.
A - B **A.difference(B)**	Returns the set difference of `A` and `B` (the elements included in `A`, but not included in `B`).
A -= B **A.difference_update(B)**	Removes all elements of `B` from the set `A`.
A ^ B **A.symmetric_difference(B)**	Returns the symmetric difference of sets `A` and `B` (the elements belonging to either `A` or `B`, but not to both sets simultaneously).
A ^= B **A.symmetric_difference_update(B)**	Writes in `A` the symmetric difference of sets `A` and `B`.
A <= B **A.issubset(B)**	Returns `true` if `A` is a subset of `B`.
A >= B **A.issuperset(B)**	Returns `true` if `B` is a subset of `A`.
A < B	Equivalent to `A <= B and A != B`
A > B	Equivalent to `A >= B and A != B`

Operations with elements

```
primes = {2, 3, 5, 7, 11}

for num in primes:
    print(num)
```

```
A = {1, 2, 3}

print(1 in A, 4 not in A)

A.add(4)
print(A)
```

```
A = {1, 2, 3}
B = {3, 4, 5}
C = A.union(B)
print(C)

A = {1, 2, 3}
B = {3, 4, 5}
A.update(B)
print(A)

A = {1, 2, 3}
B = {3, 4, 5}
C = A.intersection(B)
print(C)

A = {1, 2, 3}
B = {3, 4, 5}
C = A.difference(B)
print(C)
```

Type Conversion

list(ตัวแปร tuple หรือ set)

tuple(ตัวแปร list หรือ set)

set(ตัวแปร list หรือ tuple)

```
A = [1, 2, 3]        # List
B = (4, 5, 6)        # Tuple
C = {7, 8, 9}        # Set

ListTuple = tuple(A)
ListSet = set(A)
print(ListTuple)
print(ListSet)

TupleList = list(B)
TupleSet = set(B)
print(TupleList)
print(TupleSet)

SetList = list(C)
SetTuple = tuple(C)
print(SetList)
print(SetTuple)
```