SCU LawTech

Class #03

LAWTECH
learn coding get outstanding

Content

- 資料型態後半介紹
- 作業講解
- 進階語法
- 作業

資料型態

容器型態

- 小括號：tuple (元組)
- 中括號：list (列表、串列)
- 大括號：dict (字典)
  set (集合)

```
Tuple = (1,2,3)
List = [1,2,3,4]
Dictionary = {'key':'value'}
Set = {1,2,3,4,5,6}
```

容器型態 - dict

dict：字典

- 映射類型：{'key':'value'}，一個key僅能對應到一個value
- key 必需是不可變動的

```
#宣告一個dict變數
dict1 = {}
type(dict1)
```

```
dict
```

【Notes】

`{ }`：在查詢時，速度會比list快上很多

因為使用hash的技術，可以直接對應到我們要找的值

容器型態 - dict

```
dict1 = {'A':1, 'B':[1,2,3]}
dict1
```

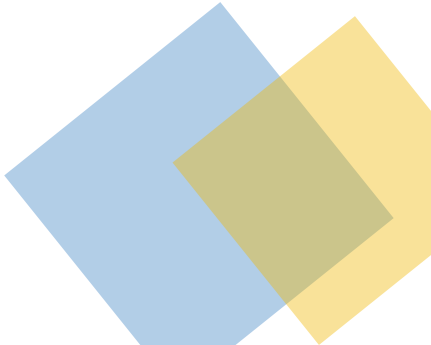
```
{'A': 1, 'B': [1, 2, 3]}
```

```
dict1['C'] = [5,6,7] #新增
dict1
```

```
{'A': 1, 'B': [1, 2, 3], 'C': [5, 6, 7]}
```

```
dict1['B'] = [7,8,9] #更改
dict1
```

```
{'A': 1, 'B': [7, 8, 9], 'C': [5, 6, 7]}
```



容器型態 - dict

```
# 所有的key（鍵）
dict1.keys()
```

```
dict_keys(['A', 'B', 'C'])
```

```
# 所有的values（值）
dict1.values()
```

```
dict_values([1, [7, 8, 9], [5, 6, 7]])
```

```
# 所有 key&values（鍵值對）
dict1.items()
```

```
dict_items([('A', 1), ('B', [7, 8, 9]), ('C', [5, 6, 7])])
```

容器型態 - dict

- 刪除：指定 key 刪除，其相對應的 value 也會隨之刪除

```
dict1.pop('A')
```

```
1
```

```
dict1
```

```
{'B': [7, 8, 9], 'C': [5, 6, 7]}
```

容器型態 - dict

- 判斷 key 是否存在

```
"A" in dict1
```

```
True
```

```
dict.get(key值, 若key值不存在的回傳值)
```

```
dict1.get('A')
```

```
1
```

```
dict1.get('1', "NO")
```

```
'NO'
```

list v.s. dict

list

- 查詢跟插入的時間會隨著元素的增加而增加
- 佔用空間小，浪費的記憶體少

dict

- 查詢跟插入的速度快，不會隨著 key 增加而變慢
- 需要佔用大量的記憶體

若直接對 dict 做操作，操作對象是 key

容器型態 - set

- 集合，無序、不重複的元素集
- 像是只有 key 的 dict

```
Set = set()
type(Set)
```

```
SET = {1,2,3,4}
type(SET)
```

```
set
```

容器型態 - set

```
# List轉set
a = [1, 2, 2, 1, 4, 2, 3]
setA = set(a)
setA
```

```
{1, 2, 3, 4}
```

```
# 增加元素
SET.add(34)
SET
```

```
{1, 2, 3, 4, 34}
```

```
# 刪除元素
SET.remove(2)
SET
```

```
{1, 3, 4, 34}
```

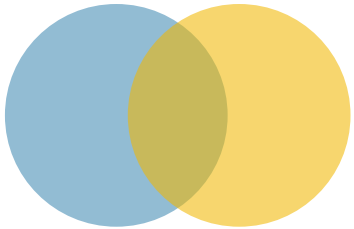
```
# 判斷元素是否存在於集合中
1  not in SET
```

```
False
```

容器型態 - set

```
# 聯集
setA = {1, 2, 3, 4}
setB = {2, 3, 4}
setA | setB
```

```
{1, 2, 3, 4}
```



```
# 交集
setA & setB
```

```
{2, 3, 4}
```

```
# 插集（注意！順序）
setA - setB
```

```
{1}
```

```
setB - setA
```

```
set()
```

作業講解

進階語法

判斷式

- If / elif /else
- 判斷式可能出現 True 或 False 兩種結果
- 判斷式後面要加”冒號”
- python 用縮排的方式，還區分程式碼的區塊

if 判斷式：

若判斷式為True，要執行的內容

判斷式

if 判斷式 :

若判斷式為True，要執行的內容

else :

若判斷式為False，要執行的內容

判斷式

if 判斷式1：

若判斷式1為True，要執行的內容

elif 判斷式2：

若判斷式2為True，要執行的內容

else：

若判斷式1、2皆為False，要執行的內容

練習

- 請設計一段程式碼：
  - 讓使用者可以輸入 0~ 100 的整數
  - 整數小於 60，顯示「不及格」
  - 整數等於 60 ，顯示「剛好及格」
  - 整數大於 60 ，顯示「及格，好棒棒！」

迴圈

- for、while
- 執行重複的事情
- 什麼時候要重複？什麼時候要停止？

for

指定執行次數（知道範圍）

while

指定條件（知道判斷式）

迴圈 - for

- 一個特定的範圍
- 停止條件：對範圍內的所有值，執行完要執行的動作
- 變數名：一次性的，並非宣告那個變數

for 變數名 in 範圍 :

要重複執行的動作

range()

- 產生整數數列

```
range(start, end, step)
```

range(1, n, 1) #產生1, 2, 3, ..., n

迴圈 - while

- 判斷式，在判斷式滿足的前提下，才執行動作
- 停止條件：判斷式未滿足（false）
- 小心進入無窮迴圈！

while 判斷式：

若判斷式為True，要執行的內容

迴圈流程控制

- 要放置在迴圈中
  - break：節制結束迴圈
  - continue：強制進入下一圈

while 判斷式：

break / continue

for 變數名 in 範圍：

break / continue

函式

- 把程式碼包在一個區塊中，方便隨時呼叫使用
- 定義 → 呼叫
- 可以放入參數
- return：回傳值，預設為 None

def 函式名稱：

函式的內容

作業

作業

1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
1 x 5 = 5
1 x 6 = 6
1 x 7 = 7
1 x 8 = 8
1 x 9 = 9

2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18

3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27

4 x 1 = 4
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
4 x 5 = 20
4 x 6 = 24
4 x 7 = 28
4 x 8 = 32
4 x 9 = 36

THANKS

LAWTECH
learn coding get outstanding