**Python程式設計：字典**

**目錄**

[1. 基本概念 1](#_Toc206960529)

[2. 建立和定義字典 1](#_Toc206960530)

[3. 存取字典中的值 2](#_Toc206960531)

[4. 新增鍵值對 2](#_Toc206960532)

[5. 刪除字典元素 2](#_Toc206960533)

[6. 字典的常用方法 2](#_Toc206960534)

[6.1 clear()：清空所有元素 2](#_Toc206960535)

[6.2 pop()：刪除並返回指定鍵的值 3](#_Toc206960536)

[6.3 in檢查鍵是否存在 3](#_Toc206960537)

[6.4 get()：安全地取值 3](#_Toc206960538)

[6.5 etdefault()：如果鍵不存在則新增 3](#_Toc206960539)

[7. 遍歷字典（Looping Dictionaries） 4](#_Toc206960540)

[7.1 items()：同時遍歷鍵和值 4](#_Toc206960541)

[7.2 keys()：遍歷字典的鍵 4](#_Toc206960542)

[7.3 values()：遍歷字典的值 4](#_Toc206960543)

[8. update()合併字典 5](#_Toc206960544)

[9. sorted()：排序 5](#_Toc206960545)

[10. dict() 函數：將序列轉換為字典 5](#_Toc206960546)

[10.1 範例：使用dict製作地球資料 5](#_Toc206960547)

[10.2 範例：使用dict紀錄經緯度 6](#_Toc206960548)

[11. 貪婪演算法與字典的關係 6](#_Toc206960549)

[12. 練習 7](#_Toc206960550)

[12.1 Problem: 摩斯密碼雙向轉換器 7](#_Toc206960551)

[12.2 Problem: 小偷背包 8](#_Toc206960552)

[12.3 Problem: 組織階層資料 8](#_Toc206960553)

[12.4 Problem: 9](#_Toc206960554)

# 基本概念

字典是一種無序的鍵（key）與值（value）的集合。想像一下，它就像一本字典一樣，你可以透過唯一的「詞彙」（鍵）來找到它所對應的「解釋」（值）。

# 建立和定義字典

你可以用 {} 來建立一個字典。鍵和值之間用 : 隔開，每對鍵值對之間用 , 隔開。

# 建立一個名為 fruits 的字典，鍵是水果名稱，值是水果的價格

fruits = {'西瓜': 15, '香蕉': 20, '水蜜桃': 25}

print(fruits)

# 建立一個名為 dict\_empty 的空字典

dict\_empty = {}

print(dict\_empty)

# 存取字典中的值

你可以透過鍵來存取對應的值。

fruits = {'西瓜': 15, '香蕉': 20, '水蜜桃': 25}

# 透過鍵 '西瓜' 來存取它的值

print(fruits['西瓜'])

# 透過鍵 '香蕉' 來存取它的值

print(fruits['香蕉'])

# 新增鍵值對

可以直接透過 字典[鍵] = 值 的方式來新增新的鍵值對

fruits = {'西瓜': 15, '香蕉': 20, '水蜜桃': 25}

# 新增一個新的鍵值對 '蘋果': 18

fruits['蘋果'] = 18

print(fruits)

# 刪除字典元素

可以使用 del 關鍵字來刪除特定的鍵值對。

fruits = {'西瓜': 15, '香蕉': 20, '水蜜桃': 25}

del fruits['香蕉'] # 刪除鍵 '香蕉' 以及它所對應的值

print(fruits)

# 字典的常用方法

## clear()：清空所有元素

這個方法會清空字典中的所有鍵值對，但字典本身仍然存在。

fruits = {'西瓜': 15, '香蕉': 20, '水蜜桃': 25}

fruits.clear()

print(fruits)

## pop()：刪除並返回指定鍵的值

pop() 方法會刪除指定鍵的元素，並返回該鍵所對應的值。如果鍵不存在，會引發錯誤。

fruits = {'西瓜': 15, '香蕉': 20, '水蜜桃': 25}

popped\_value = fruits.pop('香蕉') # 刪除 '香蕉' 並取得它的值

print(f"被刪除的值是：{popped\_value}")

print(f"刪除後的字典：{fruits}")

## in檢查鍵是否存在

fruits = {'西瓜': 15, '香蕉': 20, '水蜜桃': 25}

if '香蕉' in fruits:

    print('字典裡有香蕉')

else:

    print('字典裡沒有香蕉')

if '鳳梨' in fruits:

    print('字典裡有鳳梨')

else:

    print('字典裡沒有鳳梨')

## get()：安全地取值

get() 方法用於取得指定鍵的值。它的優點是，如果鍵不存在，它會返回 None 而不是引發錯誤，這可以防止程式意外中止。你也可以指定一個預設值，在鍵不存在時返回。

fruits = {'Apple': 20, 'Orange': 25}

# 鍵 'Apple' 存在，返回其值

value1 = fruits.get('Apple')

print(f"取得 'Apple' 的值：{value1}")

# 鍵 'Grape' 不存在，返回 None (預設值)

value2 = fruits.get('Grape')

print(f"取得 'Grape' 的值：{value2}")

# 鍵 'Grape' 不存在，返回指定的預設值 10

value3 = fruits.get('Grape', 10)

print(f"取得 'Grape' 的值（預設為 10）：{value3}")

## etdefault()：如果鍵不存在則新增

setdefault() 方法的作用與 get() 類似，但它更進一步：如果指定的鍵不存在，它會將該鍵與指定的預設值一起添加到字典中，然後返回該預設值。如果鍵已經存在，它會返回鍵的值，但不會做任何修改。

fruits = {'Apple': 10, 'Orange': 20}

# 鍵 'Orange' 存在，返回其值，字典不變

value1 = fruits.setdefault('Orange', 30)

print(f"Value 1: {value1}")

print(f"字典內容: {fruits}")

# 鍵 'Banana' 不存在，新增鍵 'Banana'，值為 30，並返回 30

value2 = fruits.setdefault('Banana', 30)

print(f"\nValue 2: {value2}")

print(f"字典內容: {fruits}")

# 遍歷字典（Looping Dictionaries）

## items()：同時遍歷鍵和值

items() 方法會返回一個由字典鍵值對組成的元組（tuple）列表，你可以用它來同時遍歷鍵和值。

players = {

    'Stephen Curry': 'Golden State Warriors',

    'Kevin Durant': 'Golden State Warriors',

}

# 使用 items() 遍歷，同時取得鍵和值

print("--- 遍歷鍵和值 ---")

for name, team in players.items():

    print(f"{name} 的球隊是 {team}")

## keys()：遍歷字典的鍵

如果你只需要遍歷字典的鍵，可以使用 keys() 方法。

players = {

    'Stephen Curry': 'Golden State Warriors',

    'Kevin Durant': 'Golden State Warriors'

}

# 使用 keys() 遍歷，只取得鍵

print("--- 遍歷鍵 ---")

for name in players.keys():

    print(f"球員姓名：{name}")

## values()：遍歷字典的值

如果你只需要遍歷字典的值，可以使用 values() 方法。

players = {

    'Stephen Curry': 'Golden State Warriors',

    'Kevin Durant': 'Golden State Warriors'

}

# 使用 values() 遍歷，只取得值

print("--- 遍歷值 ---")

for team in players.values():

    print(f"球隊名稱：{team}")

# update()合併字典

update() 方法可以將一個字典的鍵值對添加到另一個字典中。如果兩個字典有相同的鍵，則被合併的字典（參數）中的值會覆蓋掉原字典中的值。

dealerA = {'1': 'Nissan', '2': 'Toyota', '3': 'Lexus'}

dealerB = {'1': 'BMW', '2': 'Benz'}

# 將 dealerB 的內容合併到 dealerA

dealerA.update(dealerB)

print(dealerA)

# sorted()：排序

fruits = {'Orange': 60, 'Apple': 100, 'Grape': 80}

# 對key排序（依照字母順序）

sorted\_items\_by\_key = sorted(fruits.items())

print(sorted\_items\_by\_key)

# 對value排序

sorted\_items\_by\_value = sorted(fruits.items(), key=lambda item: item[1])

print(sorted\_items\_by\_value)

這邊用到的lambda，我們會在【函數】再來討論~~~

# dict() 函數：將序列轉換為字典

dict() 函數可以將包含鍵值對的二元元組（tuple）或列表（list）的序列轉換成字典。

nation\_list = [('日本', '東京'), ('泰國', '曼谷'), ('英國', '倫敦')]

nation\_dict = dict(nation\_list) # 使用 dict() 將其轉換為字典

print(nation\_dict)

## 範例：使用dict製作地球資料

# 字典的值是元組

# 鍵為洲名，值為該洲的城市元組

world = {

    'Asia': ('Beijing', 'Hongkong', 'Tokyo'),

    'USA': ('Chicago', 'New York', 'Hawaii', 'Los Angeles'),

    'Europe': ('Paris', 'London', 'Zurich')

}

# 存取 'USA' 的城市

print(f"美國的城市有: {world['USA']}")

# 存取 'USA' 中的第一個城市

print(f"美國的第一個城市是: {world['USA'][0]}")

# 遍歷 'Europe' 的所有城市

print("\n歐洲的城市列表:")

for city in world['Europe']:

    print(f"- {city}")

## 範例：使用dict紀錄經緯度

# 使用經緯度元組作為字典的鍵，值為地點名稱

loc = {

    (25.0452, 121.5168): '台北車站',

    (22.2838, 114.1731): '紅磡車站'

}

# 取得 '台北車站' 的經緯度

taipei\_coord = (25.0452, 121.5168)

print(f"台北車站的經緯度是: {taipei\_coord}")

# 透過經緯度元組來存取地點名稱

print(f"經緯度 {taipei\_coord} 所在的地點是: {loc[taipei\_coord]}")

# 貪婪演算法與字典的關係

貪婪演算法是一種演算法思想，它在每一步選擇中都採取在當下狀態下最好的選擇，從而希望可以得到一個全域的最優解。它不考慮未來的後果，只考慮眼前的最佳利益。

有個小偷有一個最多能裝 1 公斤的背包，他來到一個賣場，有以下商品可以選擇。他想在不被發現的情況下，盡可能地拿走「價值」最高的東西。

* Acer 筆電：40000 元，重 0.8 公斤
* Asus 筆電：35000 元，重 0.7 公斤
* iPhone 手機：38000 元，重 0.3 公斤
* iWatch 手錶：15000 元，重 0.1 公斤
* Go Pro 攝影：12000 元，重 0.1 公斤

像這樣的問題，就很適合使用dict作為資料結構儲存。

# 1. 定義商品字典

things = {

    'iWatch手錶': (15000, 0.1),

    'Asus 筆電': (35000, 0.7),

    'iPhone 手機': (38000, 0.3),

    'Acer 筆電': (40000, 0.8),

    'Go Pro 攝影': (12000, 0.1)

}

# 2. 依商品價格（值元組的第一個元素）進行排序

# 加上 reverse=True 參數可以實現由高到低排序

sorted\_by\_price = sorted(things.items(), key=lambda item: item[1][0], reverse=True)

# 3. 格式化輸出結果

print("所有商品依價格排序如下：")

print(f"{'商品':<10}{'價格':<10}{'重量':>10}")

print("="\*30)

for name, info in sorted\_by\_price:

    price = info[0]

    weight = info[1]

    print(f"{name:<10}{price:<10}{weight:>10.2f}")

這邊用到的lambda，我們會在【函數】再來討論~~~

# 練習

## Problem: 摩斯密碼雙向轉換器

|  |  |
| --- | --- |
| Problem Description:  請撰寫摩斯密碼程式，使其能夠實現**字母**與**摩斯碼**的雙向轉換。你需要建立兩個字典，一個用來將字母轉換成摩斯碼，另一個則用來將摩斯碼轉換成字母。你的程式需要能判斷使用者輸入的是字母還是摩斯碼，並進行正確的翻譯。 | |
| Input:  多行輸入，每行包含一個字串。  如果字串只包含英文字母和空格，則將其翻譯成摩斯碼。  如果字串只包含 -, ., 和空格，則將其翻譯回英文字母。 | Output: |
| Sample Input:  HELLO WORLD | Sample Output:  .... . .-.. .-.. --- / .-- --- .-. .-.. -.. |
| Sample Input:  .--. -.-- - .... --- - | Sample Output:  PYTHON |
| Answer: | |

## Problem: 小偷背包

|  |  |
| --- | --- |
| Problem Description:  在「小偷背包」問題的基礎上，現在你是一個商店老闆，需要對商品進行盤點和排序。  請根據以下規則對 things 字典中的商品進行排序，並將結果以列表形式輸出：   1. 首先根據重量（即值元組的第二個元素）從小到大排序。 2. 如果重量相同，則根據價格（值元組的第一個元素）從高到低排序。  * Acer 筆電：40000 元，重 0.8 公斤 * Asus 筆電：35000 元，重 0.7 公斤 * iPhone 手機：38000 元，重 0.3 公斤 * iWatch 手錶：15000 元，重 0.1 公斤 * Go Pro 攝影：12000 元，重 0.1 公斤 | |
| Input: | Output:  。 |
| Sample Input:  things = {'iWatch手錶': (15000, 0.1), 'Asus 筆電': (35000, 0.7), 'iPhone 手機': (38000, 0.3), 'Acer 筆電': (40000, 0.8), 'Go Pro 攝影': (12000, 0.1)} | Sample Output:  [('Go Pro 攝影', 12000, 0.1), ('iWatch手錶', 15000, 0.1), ('iPhone 手機', 38000, 0.3), ('Asus 筆電', 35000, 0.7), ('Acer 筆電', 40000, 0.8)] |
| Sample Input:  things = {'A': (10, 0.5), 'B': (20, 0.2), 'C': (15, 0.5)} | Sample Output:  [('B', 20, 0.2), ('C', 15, 0.5), ('A', 10, 0.5)] |
| Answer: | |

## Problem: 組織階層資料

|  |  |
| --- | --- |
| Problem Description:  你的任務是處理一個組織的階層關係，用嵌套字典來表示。請設計一個**遞迴函式**，能夠遍歷這個嵌套字典，並以階層結構（例如，使用縮排）印出所有部門和員工。 | |
| Input: | Output: |
| Sample Input:  org\_chart = {'Sales': ['Alex', 'Ben'], 'IT': {'Dev': ['Jack', 'Mark'], 'Support': ['Emily', 'Chris']}, 'Marketing': {'Team A': ['Sarah', 'Tom'], 'Team B': ['Lucy']}} | Sample Output:  Sales  - Alex  - Ben  IT  - Dev  - Jack  - Mark  - Support  - Emily  - Chris  Marketing  - Team A  - Sarah  - Tom  - Team B  - Lucy |
| Sample Input:  org\_chart = {'Engineering': ['David', 'Eva'], 'HR': {'Recruiting': ['Frank'], 'Training': ['Grace', 'Henry']}} | Sample Output:  Engineering  - David  - Eva  HR  - Recruiting  - Frank  - Training  - Grace  - Henry |
| Answer: | |

## Problem:

|  |  |
| --- | --- |
| Problem Description:  請寫一個程式，模擬一個簡單的登入驗證 。程式會從標準輸入讀取使用者輸入的帳號和密碼。 | |
| Input:  輸入有兩行，第一行為帳號，第二行為密碼。 | Output:  輸出只有一行，為驗證結果。 |
| Sample Input:  user123 pass | Sample Output:  用戶名或密碼錯誤，請重試。 |
| Answer: | |