算法 Algorithm

Python 編程課程 第 4 課

- 檢索算法
- 列表排序：sort() 和 sorted()
- 排序算法

熱身：找牌

假設現在有一副**被洗亂的**樸克牌

我想找出某張牌（例如**紅心A**）

最直接的做法是由**頂部開始，逐張翻開**

這個做法就是**「線性檢索（順序檢索）」**

♦3
♠8
♠K
♥A
♦10
♣4
…

線性檢索 Linear search

用列表變量 playing_card 表示一副**被洗亂的**樸克牌

找出**紅心A**（heart_ace）

線性檢索：

由索引0開始檢查，然後索引1，然後索引2……直到最後的項目

playing_card[0]➔playing_card[1]➔playing_card[2]➔……

	playing_card[ ]
♦3	diamond_3
♠8	spade_8
♠K	spade_king
♥A	heart_ace
♦10	diamond_10
♣4	clubs_4
…	

檢索算法 - 請打開Thonny一起做

線性檢索 1.0：項目存不存在

分數列表 **marks[ ]** 存放着學生的**考試分數**

學號	分數
1	77
2	80
3	95
4	79
5	100
6	34
7	62
8	81

程式碼

```
marks = [77, 80, 95, 79, 100, 34, 62, 81]

# 建立變量 target 為目標分數(100分)
target = 100

# 由頭開始找，找出有沒有人考得 100分
found = False
for i in range(0, len(marks)):
    if target == marks[i]:
        found = True
print(found)
```

線性檢索 1.0：項目存不存在

分數列表 **marks[ ]** 存放着學生的**考試分數**

學號	分數
1	77
2	80
3	95
4	79
5	100
6	34
7	62
8	81

程式碼

```
marks = [77, 80, 95, 79, 100, 34, 62, 81]

# 建立變量 target 為目標分數(99分)
target = 99

# 由頭開始找，找出有沒有人考得 99分
found = False
for i in range(0, len(marks)):
    if target == marks[i]:
        found = True
print(found)
```

線性檢索 2.0A：找到項目就停下來

如果目標分數是100分，當你還未找100分時，你才重覆進行檢索

找到後，後面的34分、62分、81分都不用看了

用while循環就可以**有條件地重覆**進行檢索

程式碼

```
marks = [77, 80, 95, 79, 100, 34, 62, 81]

# 建立變量 target 為目標分數（100分）
target = 100

# 由頭開始找，找出有沒有人考得 100分
found = False
i = 0
while (i < len(marks)) and (found == False):
    if target == marks[i]:
        found = True
    else:
        i = i + 1
print(found)
```

線性檢索 2.0B：找到項目的位置

如果目標分數是100分，你亦找到了，那誰考到100分呢？

建立變量 index 存放你檢索的項目索引值

當 marks[i] 的分數等於目標分數 target時，**將其索引值 i 賦值給 index**，再列印 index

程式碼

```
marks = [77, 80, 95, 79, 100, 34, 62, 81]

# 建立變量 target 為目標分數（100分）
target = 100

# 由頭開始找，找出誰考得 100分
index = -1
for i in range(0, len(marks)):
    if target == marks[i]:
        index = i
        print(i)
if index == -1:
    print("No result is found.")
```

線性檢索 Linear search

比較算法1.0和2.0A，分析 for 和 while 的效率

分數	1.0 - for	2.0A - while	
77	8次	1次	最好的情況
95	8次	3次	
100	8次	5次	
81	8次	8次	最久的情況
82	8次	8次	找不到

學號	分數
1	77
2	80
3	95
4	79
5	100
6	34
7	62
8	81

對分檢索 Binary search

如果這是一副**全新的**樸克牌
（排序：**葵扇A至K➔紅心A至K➔梅花A至K➔階磚A至K**）

找出**葵扇5**（spade_5）

葵扇5在牌的前半，可**收窄檢索範圍到「樸克牌的一半」**

這個做法就是**「對分檢索」**

	playing_card[ ]
♠A	spade_ace
♠2	spade_2
♠3	spade_3
♠4	spade_4
♠5	spade_5
♠6	spade_6
…	

對分檢索 Binary search

len(play_card[ ]) 的長度是52

葵扇5（spade_5）在 playing_card[4]

對分檢索：

1. 第一個項目索引是 0；最後的項目索引是 51
   中間的項目索引是 25（取整數）
   ➔playing_card[25]是heart_king，比spade_5大

2. 收窄檢索範圍到 0 至 24
   ➔中間項目 playing_card[12]是spade_king，比spade_5大

3. 收窄檢索範圍到 0 至 11
   ➔中間項目 playing_card[5]是spade_5

	playing_card[ ]
♠A	spade_ace
♠2	spade_2
♠3	spade_3
♠4	spade_4
♠5	spade_5
♠6	spade_6
…	

對分檢索(一):初始化

分數列表 **marks_2[ ]** 存放着學生的**考試分數(由小至大)**

分數
51
55
62
63
87
91
97
100

程式碼

```
marks_2 = [51, 55, 62, 63, 87, 91, 97, 100]

# 目標分數(91分)
target = 91

# 初始化索引值
start = 0
end = len(marks_2) - 1

# 初始化布林值
found = False
```

對分檢索(二):算法

從列表 **marks_2[ ]** 中找出 91分的位置

分數
51
55
62
63
87
91
97
100

程式碼

```
# 對分檢索循環
while start <= end and found == False:

    # 取中間點(整數)
    mid = int((start + end) / 2)

    # 檢查中間點是否等於目標值
    if marks_2[mid] == target:
        found = True
    else:

        # 如果中間點非目標, 更新索引指標範圍
        if target < marks_2[mid]:
            end = mid - 1
        else:
            start = mid + 1
```

對分檢索(三):列印結果

是否存在 91分，有的話會列印什麼？

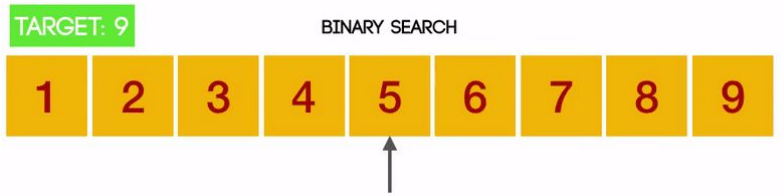
分數
51
55
62
63
87
91
97
100

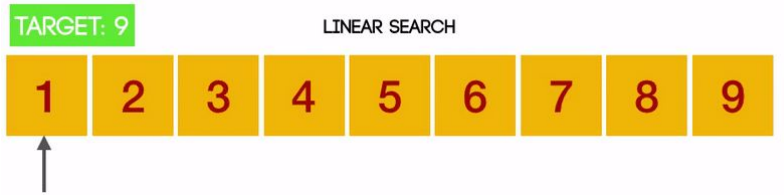
程式碼

```
# 列印結果
if found == True:

    # 找到的話:中間點就應該等於目標
    print("Found at index", mid)

else:
    # 找不到
    print("Not found")
```

線性檢索 vs 對分檢索

	線性檢索	對分檢索
程式設計複雜程度	簡單	複雜
列表排序要求	沒有	已排序
效率 - 最多檢索次數 - 平均檢索次數	*-假設列表有N個項目-* N (N+1) / 2	 log(N) / log(2) log(N) / log(2)
適用場合	項目較少的列表	項目較多的列表





列表排序：sort() 和 sorted()
請打開Thonny一起做

排序 - sort()

使用 sort() 可**永久地**對列表進行**升序**排序(**由小至大**)

在 sort() 中加入參數 reverse=True，將排序變為**降序(由大至小)**。

```
sports = ["football", "boxing", "swimming", "diving"]

sports.sort()
print(sports)

sports.sort(reverse=True)
print(sports)
```

輸出

升序(由小至大) ►

['boxing', 'diving', 'football', 'swimming']

降序(由大至小) ►

['swimming', 'football', 'diving', 'boxing']

臨時排序 - sorted()

sort() 可**永久地**改變列表排序

sorted() 則可**暫時地**改變列表排序，而不影響原始列表。

```
sports = ["football", "boxing", "swimming", "diving"]

print("Temporarily sorted list:")
print(sorted(sports))

print("Original list:")
print(sports)
```

輸出

```
Temporarily sorted list:
['boxing', 'diving', 'football', 'swimming']
Original list:
['football', 'boxing', 'swimming', 'diving']
```

暫時排序 ▶

原始列表 ▶

排序算法 - 請打開Thonny一起做

常用技巧:交換變量數值

建立兩個變量:
A杯子(cup_a)盛裝清水("water")
B杯子(cup_b)盛裝果汁("juice")

如何才能夠:
A杯子(cup_a)盛裝果汁("juice")
B杯子(cup_b)盛裝清水("water")

程式碼

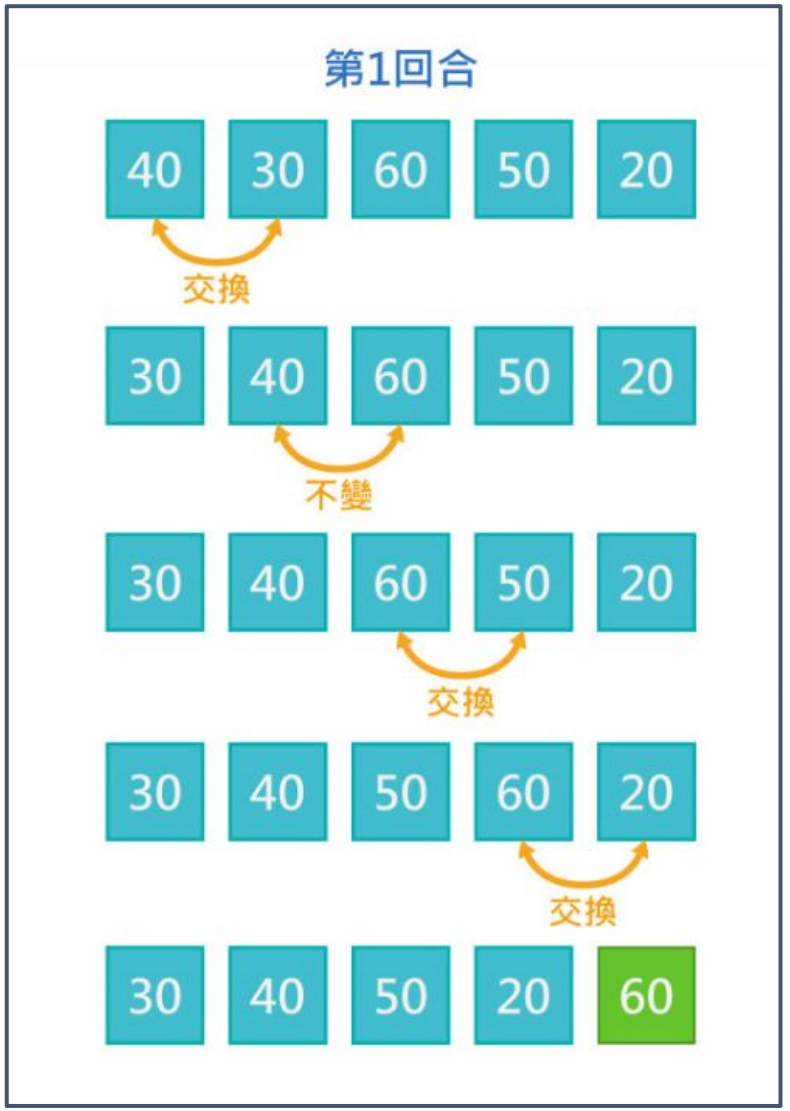
```
cup_a = "water"
cup_b = "juice"


# 新增一個杯子 cup_temp 來盛裝cup_a
cup_temp = cup_a

# 杯子B 倒進 杯子A
cup_a = cup_b

# 杯子Temp 倒進 杯子B
cup_b = cup_temp
```

冒泡排序法 Bubble sort

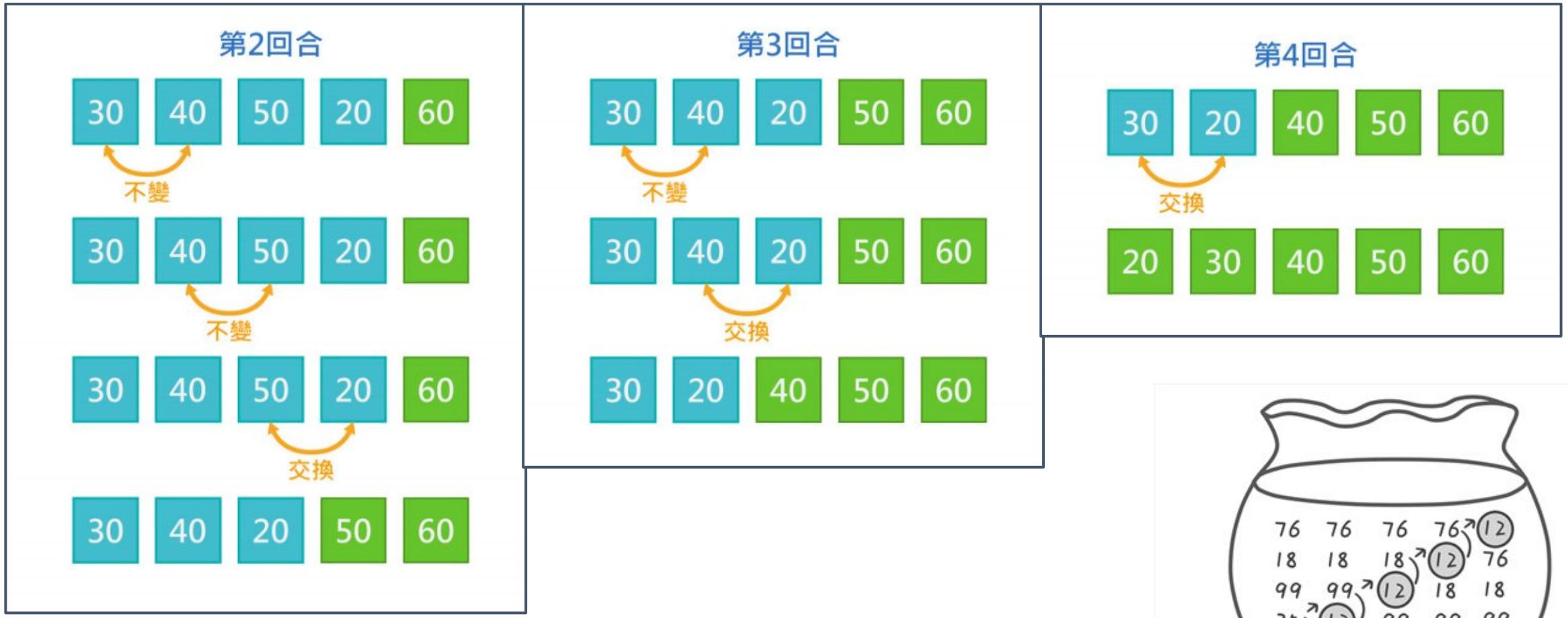
重覆兩個步驟：

1. 從**未排序好**的列表中**選擇兩個數值**作**比較**
   - 左大右小：交換
   - 左小右大：不變

2. 向右移一位，比較下一對數值



冒泡排序法 Bubble sort



冒泡排序法 Bubble sort

請使用 print("Marks:", marks)😉

整理前	整理後
40	20
30	30
60	40
50	50
20	60

程式碼

```
marks = [40, 30, 60, 50, 20]


# 大循環
for i in range(0, len(marks)-1):
    # 小循環
    for j in range(0, len(marks)-1-i):
        # 如果左大右小，交換數值
        if marks[j] > marks[j+1]:
            temp = marks[j]
            marks[j] = marks[j+1]
            marks[j+1] = temp
```

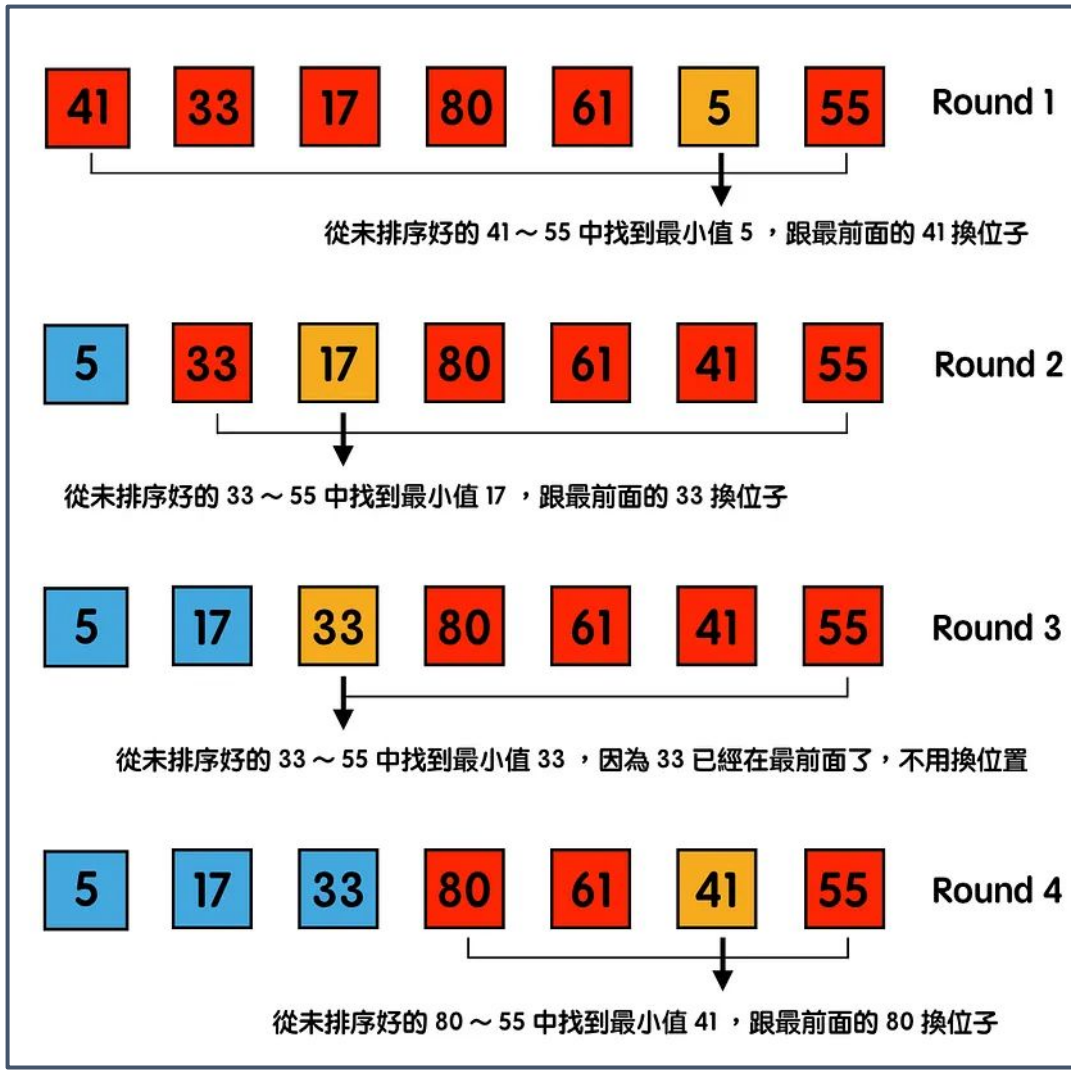
選擇排序法 Selection sort

重覆兩個步驟：

1. 從**未排序好**的列表中找到**最小值**

2. 將**最小值**和列表**最初值**，**交換**，把它標示為**已排序好**

＊也可以找最大值作交換



選擇排序法 Selection sort

請使用 print("Marks:", marks)😉

整理前	整理後
77	34
80	62
95	77
79	79
100	80
34	81
62	95
81	100

程式碼

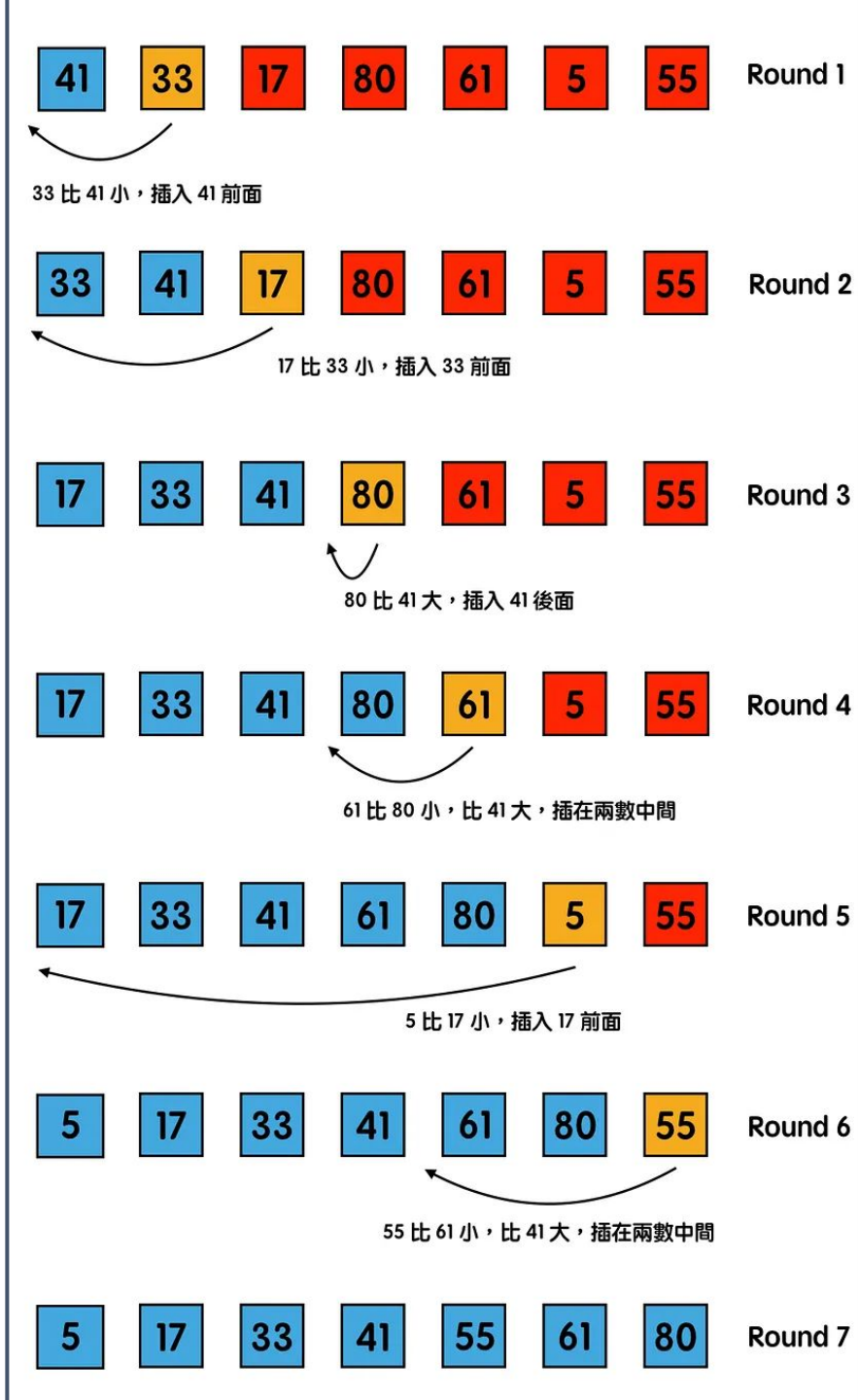
```
marks = [77, 80, 95, 79, 100, 34, 62, 81]

for i in range(0, len(marks)):
    # 找到最小值
    index_min = i
    for j in range(i+1, len(marks)):
        if marks[j] < marks[index_min]:
            index_min = j

    # 交換最小值和最初值
    temp = marks[i]
    marks[i] = marks[index_min]
    marks[index_min] = temp
```

插入排序法 Insertion sort

重覆兩個步驟：

1. 從**未排序好**的列表中**讀取一個數**賦值給**「關鍵值」**

2. 將「關鍵值」和前一位（左側）**比較**
   - 前面一位較大：取代**數值**的位置
   - 前面一位較小：**數值**沿用**「關鍵值」**



插入排序法 Insertion sort

請使用 print("Marks:", marks)😉

整理前	整理後
77	34
80	62
95	77
79	79
100	80
34	81
62	95
81	100

程式碼

```
marks = [77, 80, 95, 79, 100, 34, 62, 81]

for i in range(1, len(marks)):
    # 關鍵值（類似 temp 暫存目前數值）
    key = marks[i]
    j = i-1
    # 和左側數值比較
    while j >= 0 and key < marks[j]:
        marks[j+1] = marks[j]
        j = j - 1
    # 插入關鍵值
    marks[j+1] = key
```

進階：改變排序（小至大➔大至小）

1. 冒泡排序法 Bubble sort
2. 選擇排序法 Selection sort
3. 插入排序法 Insertion sort

<u>問題：</u>

應該改什麼，可以將排序由「小至大」變成「大至小」？🤔

（提升：改一個符號）

總結

線性檢索 Linear search

對分檢索 Binary search

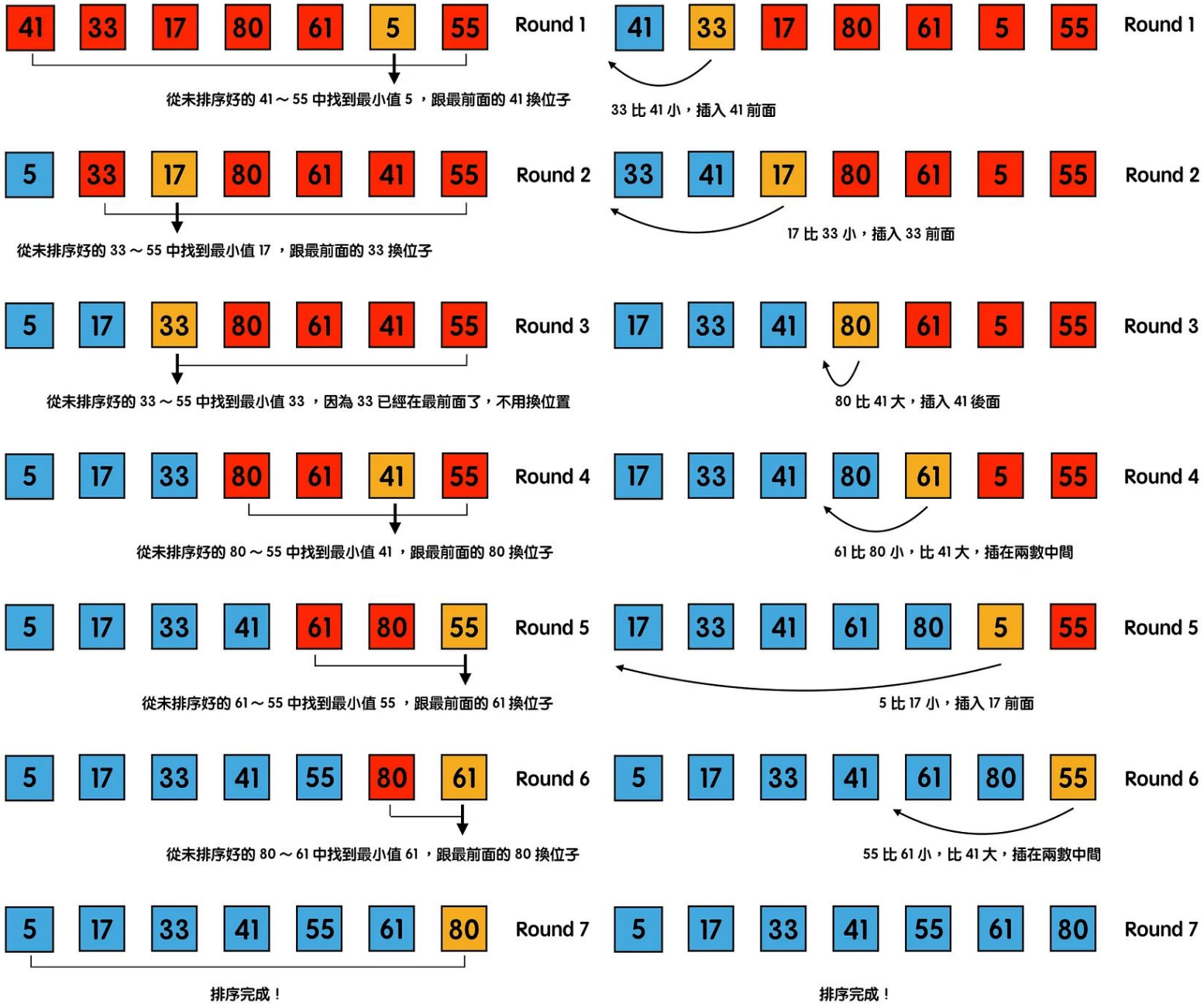
sort() 和 sorted()

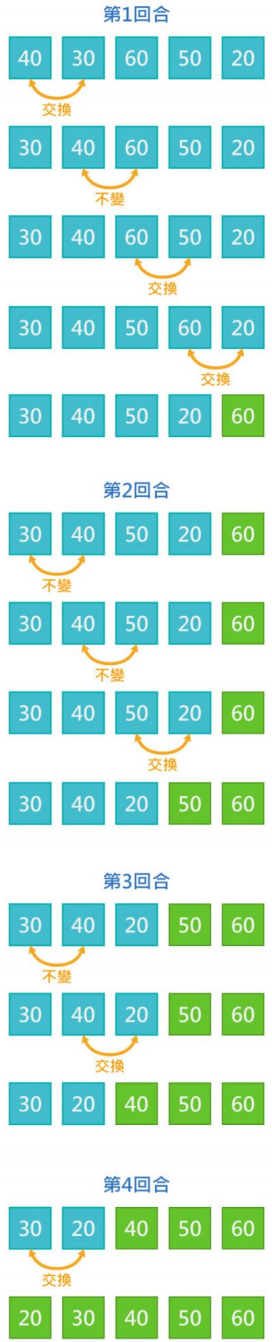
交換變量數值

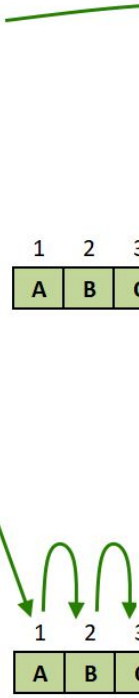
冒泡排序法 Bubble sort

選擇排序法 Selection sort

插入排序法 Insertion sort


41 33 17 80 61 5 55
Round 1
從未排序好的 41～55 中找到最小值 5 ，跟最前面的 41 換位子
41 33 17 80 61 5 55
Round 1
33 比 41 小，插入 41 前面
5 33 17 80 61 41 55
Round 2
從未排序好的 33～55 中找到最小值 17 ，跟最前面的 33 換位子
33 41 17 80 61 5 55
Round 2
17 比 33 小，插入 33 前面
5 17 33 80 61 41 55
Round 3
從未排序好的 33～55 中找到最小值 33 ，因為 33 已經在最前面了，不用換位置
17 33 41 80 61 5 55
Round 3
80 比 41 大，插入 41 後面
5 17 33 80 61 41 55
Round 4
從未排序好的 80～55 中找到最小值 41 ，跟最前面的 80 換位子
17 33 41 80 61 5 55
Round 4
61 比 80 小，比 41 大，插在兩數中間
5 17 33 41 61 80 55
Round 5
從未排序好的 61～55 中找到最小值 55 ，跟最前面的 61 換位子
17 33 41 61 80 5 55
Round 5
5 比 17 小，插入 17 前面
5 17 33 41 55 80 61
Round 6
從未排序好的 80～61 中找到最小值 61 ，跟最前面的 80 換位子
5 17 33 41 61 80 55
Round 6
55 比 61 小，比 41 大，插在兩數中間
5 17 33 41 55 61 80
Round 7
排序完成！
5 17 33 41 55 61 80
Round 7
排序完成！


第1回合
40 30 60 50 20
交換
30 40 60 50 20
不變
30 40 60 50 20
交換
30 40 50 60 20
交換
30 40 50 20 60
第2回合
30 40 50 20 60
不變
30 40 50 20 60
不變
30 40 50 20 60
交換
30 40 20 50 60
第3回合
30 40 20 50 60
不變
30 40 20 50 60
交換
30 20 40 50 60
第4回合
30 20 40 50 60
交換
20 30 40 50 60


1 2
A B