串列

蔡尚融

2018-04-09

串列型態（list）

- 串列（list）是用中括號（[與]）包起，用逗號（,）分隔各項目的資料型態。

```
>>> lstnum = [1, 5, -3, 10]
>>> lsttxt = ['one', 'two', 'hello', 'dog', 'cat',
'bus', 'tram']
```

- 串列的索引值（index）從 0 開始。

```
>>> lsttxt[5]
'bus'
```

- 用 `len()` 取得串列項目個數。

```
>>> len(lsttxt)
7
```

範例：用串列存向量

向量使用串列物件儲存，並實作加法與內積。

```
#!/usr/bin/env python
def vecadd(va, vb):
    if len(va) == len(vb):
        vc = va
        for ii in range(len(va)):
            vc[ii] = vc[ii] + vb[ii]
        return vc
    else:
        print('Error of unequal vector length')
        return None
```

```
vec1 = [1.0, 2.5, 3.5, 4.0]
vec2 = [-3.0, -1.0, 2.0, 1.5]

print('vec1:', vec1)
print('vec2:',vec2)
print('vec1 + vec2:', vecadd(vec1, vec2))
```

串列的項目

- 串列項目也可為串列。
  例如：二維串列

```
>>> lstlst = [[1, 2, 3], [3 ,4 ,5], [3, 6, 9]]
>>> lstlst [[1, 2, 3], [3, 4, 5], [3, 6, 9]]
```

- 單索引取得項目

```
>>> lstlst[1]
[3, 4, 5]
```

- 雙索引取得項目

```
>>> lstlst[1][2]
5
```

- 串列的項目可為不同型態。

```
>>> lstdat = [[1, 2, 3], 'Bus', ['Date', 'Time',
['Departure', 'Arrival']]]
```

範例：矩陣向量乘法

$$\begin{bmatrix} 3 & 4 & 5 & -1 & -3 \\ 9 & -11 & 5 & 0 & 3 \\ -7 & 0 & 3 & 2 & -4 \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 3 \\ -4 \\ 5 \end{bmatrix}$$

```
#!/usr/bin/env python
vx = [1, -2, 3, -4, 5]
ma = [[3, 4, 5, -1, -3], [9, -11, 5, 0, 3], \
    [-7, 0, 3, 2, -4]]
vy = [0, 0, 0]
for ii in range(3):
    for ij in range(5):
        vy[ii] = ma[ii][ij] * vx[ij]
print(vy)
```

隨堂練習：矩陣乘矩陣

$$\begin{bmatrix} 3 & 4 & 5 & -1 & -3 \\ 9 & -11 & 5 & 0 & 3 \\ -7 & 0 & 3 & 2 & -4 \end{bmatrix} \begin{bmatrix} 3 & -1 & 0 & 5 \\ -9 & 1 & -3 & 4 \\ 0 & -4 & 8 & -3 \\ 5 & 0 & -2 & -3 \\ -4 & 8 & 10 & -3 \end{bmatrix}$$

```
#!/usr/bin/evn python

def show_matrix(mx):
    for ii in range(len(mx)):
        print(mx[ii])
```

```
ma = [[3, 4, 5, -1, -3], [9, -11, 5, 0, 3], \
     [-7, 0, 3, 2, -4]]
mb = [[3, -1, 0, 5], [-9, 1, -3, 4], [0, -4, 8, -3], \
     [5, 0, -2, -3], [-4, 8, 10, -3]]

mc = [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]

print('ma')
show_matrix(ma)
print('mb')
show_matrix(mb)
```

```
for ii in range(len(ma)):
    for ij in range(len(mb[0])):
        for ik in range(len(ma[0])):
            mc[ii][ij] += ma[ii][ik] * mb[ik][ij]

print('mc')
show_matrix(mc)
```

範例：

假設有依序列工作，其各自工作耗費工時如下佇列：

1, 3, 2, 7, 5, 3, 6, 8, 10, 1, 5, 6, 3, 12, 3

假設有一工人處理每一工作須花 1 工時準備，2 工時收尾，請問此工人總工作時間。

```
#!/usr/bin/env python
worktime = [1, 3, 2, 7, 5, 3, 6, 8, 10, 1, 5, 6, 3, \
    12, 3]
total = 0
for ii in range(len(worktime)):
    total = total + 1 + worktime[ii] + 2
print('Total time:', total)
```

隨堂練習：

假設有 15 件獨立的工作，其各自工作處理耗費工時如下所列：

1, 3, 2, 7, 5, 3, 6, 8, 10, 1, 5, 6, 3, 12, 3

假設有 A、B、C 三位工人處理此 15 件工作，其中工人 A 須花 1 個工時準備，2 個工時收尾，B 須花 2 個工時準備，1 個工時收尾，C 須花 1 個工時準備，1 個工時收尾；前五項工作由工人 A 處理，中間五項由工人 B 負責，剩餘由工人 C 處理，三位工人同時開始處理，請問須花費多少時間完成。

```
#!/usr/bin/env python

worktime = [1, 3, 2, 7, 5, 3, 6, 8, 10, 1, 5, 6, 3, \
    12, 3]

total = [0, 0, 0]
for ii in range(5):
    total[0] = total[0] + 1 + worktime[ii] + 2
    total[1] = total[1] + 2 + worktime[ii + 5] + 1
    total[2] = total[2] + 1 + worktime[ii + 10] + 1

print('Time cost:', max(total))
```

一些串列的方法

- 刪除指定（索引值）項目 del()

```
>>> lstnum = [1, 5, -3, 10]
>>> del(lstnum[2])
>>> lstnum
[1, 5, 10]
```

- 最小值 min()，最大值 max()。

```
>>> lstnum = [3, 5, -7, 4, -7, 10, -1, 3, 10]
>>> min(lstnum)
-7
>>> max(lstnum)
10
```

- 附加項目 append()。

```
>>> lstnum = [3, 5, -7, 9]
>>> lstnum.append(-8)
>>> lstnum
[3, 5, -7, 9, -8]
```

- 附加串列 extend()。

```
>>> lstnum.extend([-7, 5])
>>> lstnum
[3, 5, -7, 9, -8, -7, 5]
```

- 指定位置（索引值）插入項目 insert()。

```
>>> lstnum.insert(3, 25)
>>> lstnum
[3, 5, -7, 25, 9, -8, -7, 5]
```

- 移除首次遇到為指定值之項目 remove()。

```
>>> lstnum = [3, 5, -7, 25, 9, -8, -7, 5]
>>> lstnum.remove(5)
>>> lstnum
[3, -7, 25, 9, -8, -7, 5]
```

- 回傳並刪除最後一個項目 pop()

```
>>> lstnum.pop()
3
>>> lstnum
[5, -7, -8, 9, 25, -7]
```

- 回傳並刪除指定（索引值 i）項目 pop(i)

```
>>> lstnum.pop(3)
9
>>> lstnum
[5, -7, -8, 25, -7]
```

- 取得項目為給定值的最先遇到的索引值 `index(value)`

```
>>> lstnum = [5, -7, -8, 9, 25, -7, 3, 5]
>>> lstnum.index(-7)
1
```

- 給定值出現次數 `count(value)`

```
>>> lstnum.count(5)
2
>>> lstnum.count(1)
0
```

- 排序 `sort()`

```
>>> lstnum.sort()
>>> lstnum
[-8, -7, -7, 3, 5, 5, 9, 25]
```

- 串列反轉次序 reverse()。

```
>>> lstnum = [3, -7, 25, 9, -8, -7, 5]
>>> lstnum.reverse()
>>> lstnum
[5, -7, -8, 9, 25, -7, 3]
```

- 清除全部項目 clear()

```
>>> lstnum.clear()
>>> lstnum
[]
```

- 複製串列 copy()

```
>>> lsta = [1, 3, 4, 5]
>>> lstb = lsta
>>> lstc = lsta.copy()
>>> lsta[1] = 8
>>> lsta
[1, 8, 4, 5]
>>> lstb
[1, 8, 4, 5]
>>> lstc
[1, 3, 4, 5]
```

範例：用迴圈模擬時間經過（單一工人）

假設有15件工作，其各自工作耗費工時如下佇列：

`1, 3, 2, 7, 5, 3, 6, 8, 10, 1, 5, 6, 3, 12, 3`

有一工人處理每一工作須花 1 工時準備，2 工時收尾，請問此工人總工作時間。

- 迴圈執行一次，模擬經過一單位工作時間（一週期）；迴圈執行總次數，為模擬的工時花費。
- 工作狀態：等待、準備、處理、收尾。
- 若工作狀態為等待，則從工作佇列派送予工人。（由串列的 `pop(0)` 函式取出）

- 工人執行工作，以一函式模擬其狀態；
  - 其輸入為現在狀態、準備、處理、收尾，各項剩餘時間；
  - 輸出為下一時間工作狀態，與各項剩餘時間；
  - 若經一處理週期後，收尾剩餘時間為零時，須重設工作狀態為等待。
- 最後一個工作收尾後結束迴圈執行。
  - 考慮工作佇列為零。
  - 工作狀態為等待。

```
#!/usr/bin/env python

flag_ws_getjob = 0
flag_ws_initial = 1
flag_ws_progress = 2
flag_ws_final = 3

def working(wstate, itime, wtime, ftime):
    next_state = wstate
    cur_itime = itime
    cur_wtime = wtime
    cur_ftime = ftime
```

```
if wstate == flag_ws_initial:
    cur_itime = cur_itime - 1
    if cur_itime == 0:
        next_state = flag_ws_progress
elif wstate == flag_ws_progress:
    cur_wtime = cur_wtime - 1
    if cur_wtime != 0:
        next_state = flag_ws_progress
    else:
        next_state = flag_ws_final
elif wstate == flag_ws_final:
    cur_ftime = cur_ftime - 1
    if cur_ftime == 0:
        next_state = flag_ws_getjob
```

```
    return next_state, cur_itime, cur_wtime, \
        cur_ftime

worktime = [1, 3, 2, 7, 5, 3, 6, 8, 10, 1, 5, 6, 3, \
    12, 3]

wkr_a_itime = 1
wkr_a_ftime = 2

cwa_state = flag_ws_getjob
cwa_itime = wkr_a_itime
cwa_wtime = 0
cwa_ftime = wkr_a_ftime

total = 0
```

```
while True:
    if cwa_state == flag_ws_getjob:
        if len(worktime) == 0:
            break
        else:
            cwa_state = flag_ws_initial
            cwa_itime = wkr_a_itime
            cwa_wtime = worktime.pop(0)
            cwa_ftime = wkr_a_ftime
    total = total + 1
    cwa_state, cwa_itime, cwa_wtime, cwa_ftime \
        = working(cwa_state, cwa_itime, cwa_wtime, \
            cwa_ftime)

print(total)
```

隨堂練習：用迴圈模擬時間經過（多人同時工作）

假設有 15 件工作，其各自工作耗費工時如下佇列：

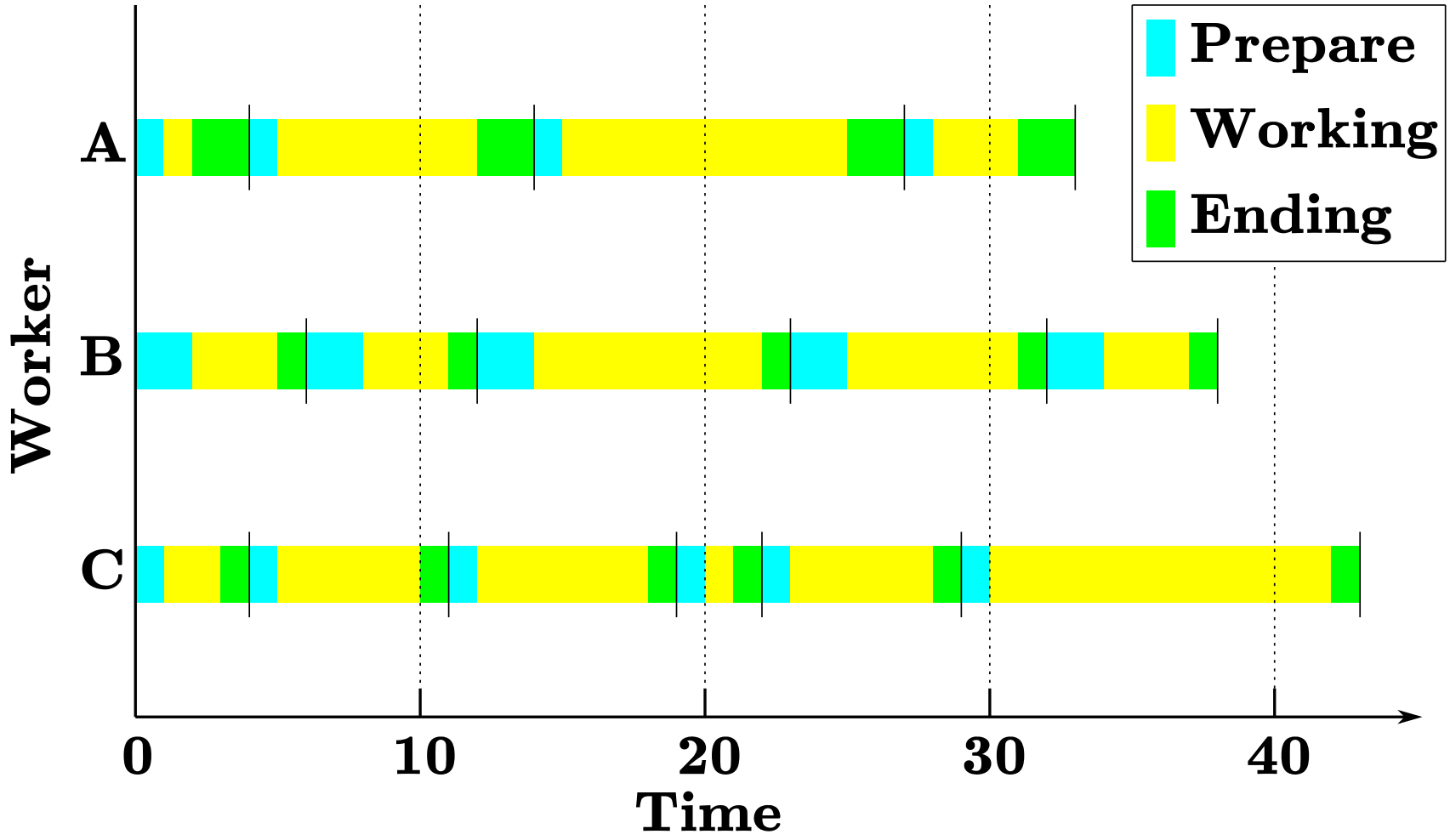
```
1, 3, 2, 7, 5, 3, 6, 8, 10, 1, 5, 6, 3, 12, 3
```

有 A、B、C 三工人處理此 15 件工作，工人完成一件工作後，再由工作佇列中依序取一件處理；各工人準備與收尾時間如下表：

工作者	A	B	C
準備	1	2	1
收尾	2	1	1

此 15 件工作須花多少時間完成。

工作處理時間經過情形



修改前範例中單一人為三工人，其準備、處理與收尾時間，及狀態改以串列儲存：

```
wkr_itime = [1, 2, 1]
wkr_ftime = [2, 1, 1]

cwa_state = [flag_ws_getjob, flag_ws_getjob,
flag_ws_getjob]
cwa_itime = wkr_itime.copy()
cwa_wtime = [0, 0, 0]
cwa_ftime = wkr_ftime.copy()
```

- 工作迴圈停止方式須考慮：
  - 處理工作佇列是否仍有工作的控制旗號。
  - 剩餘處理工作時間。
- 設定控制旗號與條件：

```
flag_stop = False
```

```
remain_time = 1
```

- 處理迴圈修改：
  - 修改前範例處理迴圈內文，將處理單一工人作業，改以（內）迴圈處理多人。
  - 以控制旗號與條件作為執行條件。

```
while flag_stop != True or remain_time != 0:
    for ii in range(3):
        if cwa_state[ii] == flag_ws_getjob:
            if len(worktime) == 0:
                flag_stop = True
            else:
                cwa_state[ii] = flag_ws_initial
                cwa_itime[ii] = wkr_itime[ii]
                cwa_wtime[ii] = worktime.pop(0)
                cwa_ftime[ii] = wkr_ftime[ii]

    total = total + 1
```

```
for ii in range(3):
    cwa_state[ii], cwa_itime[ii], cwa_wtime[ii], \
        cwa_ftime[ii] = \
        working(cwa_state[ii], cwa_itime[ii], \
            cwa_wtime[ii], cwa_ftime[ii])

remain_time = max(cwa_wtime) + max(cwa_ftime)
```

貪婪式演算法的原理

- Greedy Algorithm 為一種尋找最佳解的方法。
- 某一起始值開始，尋找鄰近更佳解。
- 分配多人工作，最短工作時花費問題，最佳解為考慮每一工人耗費相近工作時間。
- 從工作佇列中選取最長（或最短）工作先分配。
  假設三人處理15件工作，各工作耗費工時如下佇列：

  `1, 3, 2, 7, 5, 3, 6, 8, 10, 1, 5, 6, 3, 12, 3`

  分配方式為：

A	12	6	3	3	1
B	10	6	5	3	1
C	8	7	5	3	2

隨堂練習：

有15件工作，其工作編號與耗用工時如下列：

工作編號	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
耗用工時	1	3	2	7	5	3	6	8	10	1	5	6	3	12	3

以貪婪式演算法處理，將工作分派給三人處理，則此三人處理的工作編號各自為何，與各自的總時間花費。

提示：

1. 用 `max()` 或 `min()` 找出最大值或最小值。
2. 再以 `list.index()` 找出串列索引值。
3. `list.pop()` 取出。