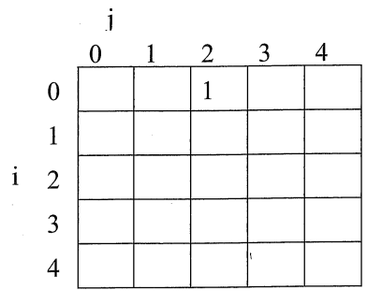
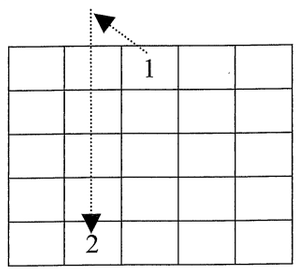
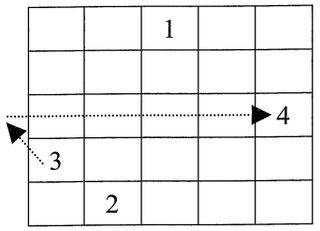
1. 將 1 填入此方陣的最上列的中間空格，如下所示：



2. 承 1，往左上方走，由於超出方陣，依據規則(3)發現往下的最底層有空格，因此將 2 填上。如下所示：



3. 承 2，往左上方，依據規則(1)將 3 填上，然後再往左上方，此時，超出方陣，依據規則(3)將 4 填在最右方的空格，如下所示：



4. 承 3，往左上方，依據規則(1)將 5 填上，再往左上方時，此時方格已有數字，依據規則(2)，往 5 的下方填，如下所示：

		1		
			5	
			6	4
3				
	2			

5. 依此類推，依據上述的四個規則繼續填，填到 15 的結果如下：

15	8	1		
	14	7	5	
		13	6	4
3			12	10
9	2			11

6. 承 5，此時往左上方，發現往下的最底層和往右的最右方皆無空格，依據規則(4)，在原地的下方將此數字填上，如下所示：

15	8	1		
16	14	7	5	
		13	6	4
3			12	10
9	2			11

7. 繼續往下填，並依據規則(1)、(2)、(3)、(4)，最後的結果如下：

15	8	1	24	17
16	14	7	5	23
22	20	13	6	4
3	21	19	12	10
9	2	25	18	11

此時讀者可以算算各行、各列及對角線之和是否皆相等，其和為 65。有關奇數魔術方陣的程式實作，請參閱 2.8 節。

練習題

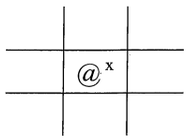
自行完成 9*9 的魔術方陣。

2.7 生命細胞遊戲

本章將以生命細胞遊戲(game of life)做為結束，此遊戲在 1970 年由英國數學家 J. H. CONWAY 所提出。生命細胞遊戲將串列元素視為細胞，而某一細胞的鄰居乃是指在其垂直、水平、對角線相鄰之細胞(cells)。

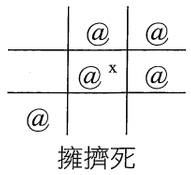
生命細胞遊戲的規則：

1. **孤單死**：若一活細胞只有一個或沒有鄰居細胞存活的，則在下一代，它將孤單而死。(下圖中以@表示一細胞，而 x 符號表示此細胞將死去)如：



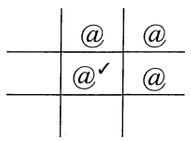
孤單死

2. **擁擠死**：一活細胞有四個或四個以上鄰居亦是活的，則在下一代，它將因擁擠而死。(下圖中以 x 符號表示此細胞將死去)如：



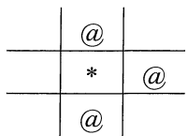
擁擠死

3. **穩定**：一活細胞有二個或三個相鄰活細胞，則下一代它將繼續生存。(下圖中以✓符號表示此細胞將繼續存活之)如：



穩定

4. **復活**：一死細胞正好有三個相鄰的活細胞，則下一代它將復活。(下圖中以*符號表示此位置會復活一細胞)如：

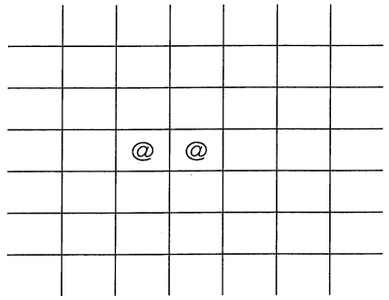


復活

由上規則可得：

- 某細胞若有 0 或 1 個相鄰細胞，則它在下一代將會因孤單而死。
- 某細胞若有 4, 5, 6, 7, 8 個相鄰細胞，則它在下一代將會因擁擠而死。
- 某細胞若有 2 個相鄰活細胞，則它在下一代將保持不變。
- 某細胞若有 3 個相鄰活細胞者，則不管其現在是生或死，下一代會是活的。

範例一》



我們將上圖填上每一細胞的相鄰活細胞個數

	0	0	0	0	0	0	
	0	1	2	2	1	0	
	0	1	@1	@1	1	0	
	0	1	2	2	1	0	
	0	0	0	0	0	0	

根據規則(1)，圖中的細胞將會孤單而死

範例二》

	0	0	0	0	0	0	
	0	1	2	2	1	0	
	0	2	@3	@3	2	0	
	0	2	@3	@3	2	0	
	0	1	2	2	1	0	
	0	0	0	0	0	0	

某一細胞若相鄰的活細胞為 2 或 3，則它將會存活下來。但圖中的死細胞其相鄰的活細胞都是小於或等於 2，故不能再生，所以已成為穩定狀態。

範例三》

	0	0	0	0	0	
	1	2	3	2	1	
	1	@1	@2	@1	1	
	1	2	3	2	1	
	0	0	0	0	0	

根據上述規則，它的下一代將為

	0	1	1	1	0	
	0	2	@1	2	0	
	0	3	@2	3	0	
	0	2	@1	2	0	
	0	1	1	1	0	

這二張圖將會來回的互換。

有關生命細胞遊戲的程式實作，請參閱 2.8 節。

2.8 程式實作

(一) 矩陣相乘

Python 程式語言實作》 兩個矩陣相乘

```
01  # 矩陣相乘實作
02  # 將兩矩陣行列相乘之和放入第三個矩陣
03  # File Name: Matrix.py
04  # Version 3.0, March 13th, 2017
05
06  N = 5
07  C = [[0] * N for row in range(N)]
08
09  def access_matrix(A, B):
```



```
10      global C
11
12      # 將A矩陣每一列元素與B矩陣每一列元素
13      # 相乘之和放入C矩陣之中
14      for i in range(N):
15          for j in range(N):
16              sum = 0
17              for k in range(N):
18                  sum += A[i][k] * B[k][j]
19              C[i][j] = sum
20
21  def output_result(A, B):
22      global C
23      # 列出三矩陣內容
24      print("\nContent of Matrix A :\n")
25      output_Matrix(A)
26      print("\nContent of Matrix B :\n")
27      output_Matrix(B)
28      print("\nContent of Matrix C :\n")
29      output_Matrix(C)
30
31  def output_Matrix(m): # 輸出陣列內容
32      for i in range(N):
33          for j in range(N):
34              print(" ", m[i][j], end = '')
35          print() # 輸出完一列跳行
36
37  def main(): # 主函數
38      global N
39
40      A = [[0]*N for row in range(N)] # 宣告5x5陣列A並將所有元素指定為0
41      B = [[0]*N for row in range(N)] # 宣告5x5陣列B並將所有元素指定為0
42
43      for i in range(N):
44          for j in range(N):
45              A[i][j] = j + 1 # 給5x5的陣列A指定初始值
46      for i in range(N):
47          for j in range(N):
48              B[i][j] = -(j - 5) # 給5x5的陣列B指定初始值
49
50      access_matrix(A, B)
51      output_result(A, B)
52
53  main()
```