1. 在全域先設定好起點座標，終點做邊，以及迷宮(0 代表可以走，1 則不行)

```python
import math ···


'''
intial  maze
'''
maze = [[0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 1],
        [0, 0, 0, 0, 0, 0],
        [1, 1, 1, 1, 0, 0],
        [1, 0, 0, 0, 0, 1],
        [0, 0, 0, 0, 0, 1]]

StartX = 0
StartY = 0
EndingX = 5
EndingY = 2
```

	0	1	2	3	4	5
0	S	◦	◦	◦	◦	◦
1	◦	◦	◦	◦	#	#
2	◦	◦	◦	◦	◦	◦
3	#	#	#	#	◦	◦
4	#	◦	◦	◦	◦	#
5	◦	◦	E	◦	◦	#

2.接著先設定節點，內含座標，h 值，g 值，以及父節點

```python
'''
initial node
'''


class Node:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
        self.gCost = 0.0
        self.hCost = 0.0
        self.isObs = 0
        self.parent = None

    def g(self, g):
        self.gCost = g

    def h(self):
        # float type
        self.hCost = math.pow(
            (math.pow((EndingX - self.x), 2) + math.pow((EndingY - self.y), 2)), 0.5)

        # print(self.hCost)

    def setPrent(self, p):
        self.parent = p

```

2. 再來則是實作 A* 的演算法

(1)初始值以及他們的狀態

```
55  '''
56  A Star
57  '''
58
59
60  class AStar:
61      def __init__(self, maze, startNode, endingNode):
62          self.openList = []
63          self.closeList = []
64          self.pathList = []
65          self.nodes = [[]]
66          self.maze = maze
67          self.startX = StartX
68          self.startY = StartY
69          self.endingX = EndingX
70          self.endingY = EndingY
71          self.currentNode = startNode
72          self.startNode = startNode
73          self.endingNode = endingNode
74
```

(2)辅助函示

```
88  # if openList is empty
89      def isEmpty(self, ):
90          if not self.openList:
91              #print("openList is empty, haha\n")
92              return 0
93
94  # get min node from openList
95      def getMin(self, ):
96          temp = self.openList[0]
97
98          for node in self.openList:
99              if node.gCost + node.hCost < temp.gCost + temp.hCost:
100                 temp = node
101
102         return temp
103
104 # push node into list
105     def push(self, node):
106         self.openList.append(node)
107
108 # remove node from list
109     def removeNode(self, node):
110         return self.openList.pop(node)  # del? del0?
111
112 # obtain node from list
113     def getNode(self, node):
114         for n in self.openList:
115             if n.x == node.x and n.y == node.y:
116                 # print(n.gCost)
117                 return n
118         return None
119
120 # if node is in openList
121     def nodeInOpen(self, node):
122         for n in self.openList:
123             if n.x == node.x and n.y == node.y:
124                 return 1
125         return 0
126
127 # if node is in closeList
128     def nodeInClose(self, node):
129         for n in self.closeList:
130             if n.x == node.x and n.y == node.y:
131                 return 1
132         return 0
```

```
134    # if ending is in openlist
135        def endingInOpen(self, ):
136            for n in self.openList:
137                if n.x == self.endingX and n.y == self.endingY:
138                    return 1
139            return 0
```

(3)對傳入的節點做處理

```
141    # search node haha
142        def searchNode(self, node):
143            # if out of range
144            if node.x < 0 or node.x > 5 or node.y < 0 or node.y > 5:
145                return
146             # if obs
147            if self.maze[node.x][node.y] == 1:
148                return
149            # already in close list
150            if self.nodeInClose(node):
151                return
152            # G(n)
153            if abs(self.currentNode.x - node.x) == 1 and abs(self.currentNode.y - node.y) == 1:
154                g = 1.4
155            else:
156                g = 1.0
157            # if not in open, then push
158            if self.nodeInOpen(node) == 0:
159                node.g(g + self.currentNode.gCost)
160                node.h()
161                node.parent = self.currentNode
162                self.openList.append(node)
163
164            # if in open, then compare
165            else:
166                n1 = self.getNode(node)
167                # print("(%d,%d)  " % (node.x, node.y))
168                # print("(%d,%d)  " % (n1.x, n1.y))
169                # print("(%f,%f) \n" % (node.gCost, n1.gCost))
170
171                if self.currentNode.gCost + g < n1.gCost:
172                    # print(123321999000)
173                    n1.gCost = self.currentNode.gCost + g
174                    n1.parent = self.currentNode  # dont forget this!!
175                    # self.openList.remove(node)
176                    # self.openList.append(n1)
```

(4)分別尋找你的八個左鄰右舍

```
178   # search your neighbors
179       def searchEightNeighbors(self, ):
180           # top left
181           self.searchNode(Node(self.currentNode.x - 1, self.currentNode.y - 1))
182           # top
183           self.searchNode(Node(self.currentNode.x - 1, self.currentNode.y))
184           # top right
185           self.searchNode(Node(self.currentNode.x - 1, self.currentNode.y + 1))
186           #     left
187           self.searchNode(Node(self.currentNode.x, self.currentNode.y - 1))
188           #     right
189           self.searchNode(Node(self.currentNode.x, self.currentNode.y + 1))
190           # bottom left
191           self.searchNode(Node(self.currentNode.x + 1, self.currentNode.y - 1))
192           # bottom
193           self.searchNode(Node(self.currentNode.x + 1, self.currentNode.y))
194           # bottom right
195           self.searchNode(Node(self.currentNode.x + 1, self.currentNode.y + 1))
196
```

(5)開始從起點按照演算法去尋找離終點花費最少的路徑

```
197   # start to find the path
198       def findPath(self):
199           # first node
200           self.startNode.g(0)
201           self.startNode.h()
202           self.openList.append(self.startNode)
203
204           # loop till find the ending
205           while True:
206               self.currentNode = self.getMin()
207               self.closeList.append(self.currentNode)
208               self.openList.remove(self.currentNode)
209
210               self.searchEightNeighbors()
211
212               # check if it is finish
213               if self.endingInOpen():
214                   temp = self.getNode(self.endingNode)
215
216                   while True:
217                       self.pathList.append(temp)
218                       if temp.parent != None:
219                           temp = temp.parent
220                       else:
221                           return True
222               elif len(self.openList) == 0:
223                   return False
224           return True
```

(6)印出答案(路徑的座標)

```
226   # show the answer
227       def showPath(self, ):
228           l = len(self.pathList)
229           for i in range(l):
230               print("(%d,%d)\n" %
231                     (self.pathList[l - i - 1].x, self.pathList[l - i - 1].y))
232
```

3.main()，先顯現出題目的座標，再來跑 A*演算法，最終印出路徑的座標

```
234   '''
235   main
236   '''
237
238
239   def main():
240       #print("Hello World\n")
241       # show original maze
242       print("show the define coordinates : ")
243       for i in range(6):
244           for j in range(6):
245               print("(%d,%d)" % (i, j), end=" ")
246           print("\n")
247
248       # start a* algo
249       aStar = AStar(maze, Node(StartX, StartY), Node(EndingX, EndingY))
250       print("output : \n")
251       if aStar.findPath():
252           aStar.showPath()
253       else:
254           print("fail QQ")
255
256
257   if __name__ == '__main__':
258       main()
```

OUTPUT:

```
output :

(0,0)

(1,1)

(2,2)

(2,3)

(3,4)

(4,3)

(5,2)
```

```
show the define coordinates :
(0,0) (0,1) (0,2) (0,3) (0,4) (0,5)

(1,0) (1,1) (1,2) (1,3) (1,4) (1,5)

(2,0) (2,1) (2,2) (2,3) (2,4) (2,5)

(3,0) (3,1) (3,2) (3,3) (3,4) (3,5)

(4,0) (4,1) (4,2) (4,3) (4,4) (4,5)

(5,0) (5,1) (5,2) (5,3) (5,4) (5,5)

output :
```

	0	1	2	3	4	5
0	S	∘	∘	∘	∘	∘
1	∘	∘	∘	∘	#	#
2	∘	∘	∘	∘	∘	∘
3	#	#	#	#	∘	∘
4	#	∘	∘	∘	∘	#
5	∘	∘	E	∘	∘	#