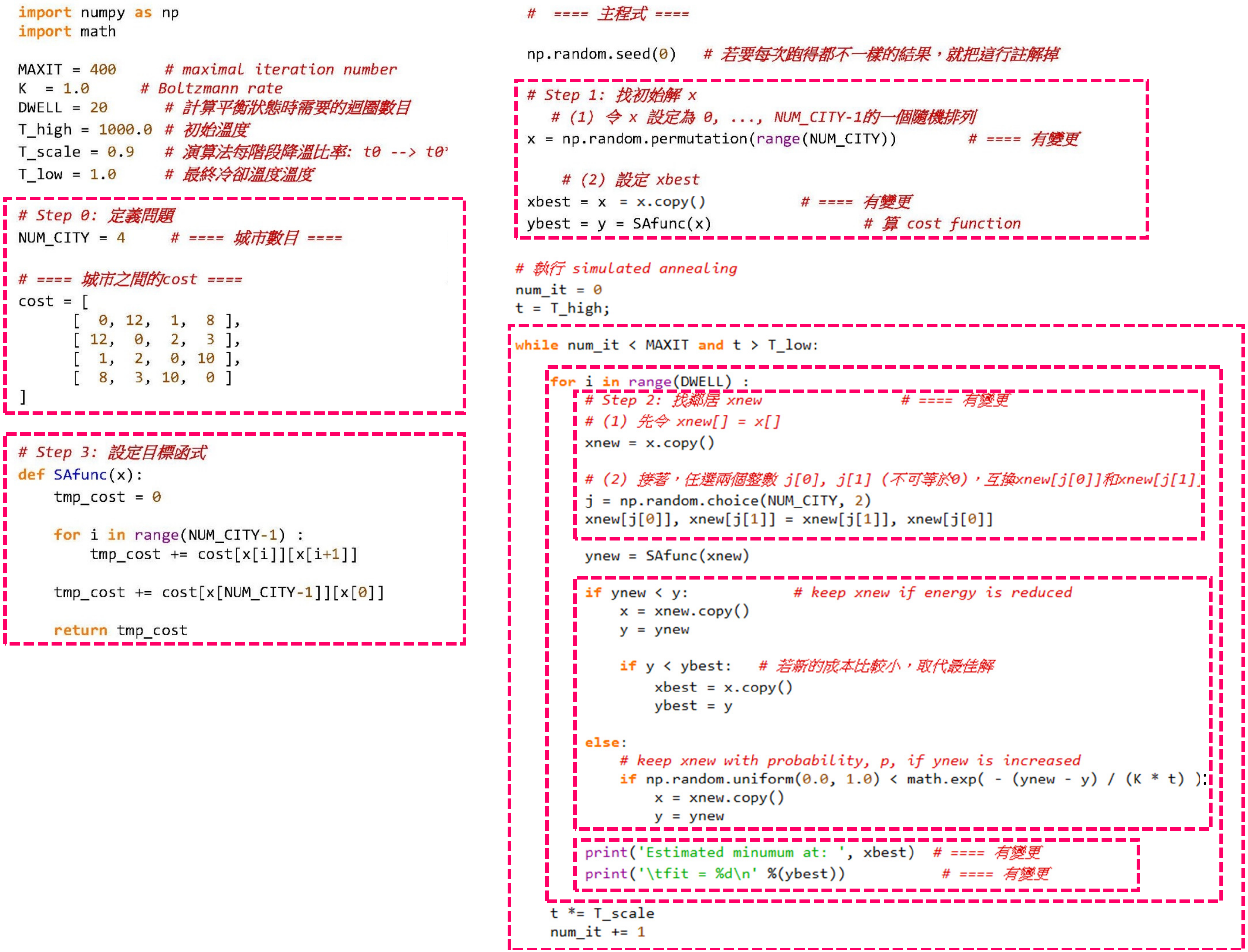

```python
import numpy as np
import math

MAXIT = 400       # maximal iteration number
K   = 1.0       # Boltzmann rate
DWELL = 20        # 計算平衡狀態時需要的迴圈數目
T_high = 1000.0 # 初始溫度
T_scale = 0.9     # 演算法每階段降溫比率: t0 --> t0
T_low = 1.0       # 最終冷卻溫度溫度

# Step 0: 定義問題
NUM_CITY = 4       # ==== 城市數目 ====

# ==== 城市之間的cost ====
cost = [
      [  0, 12,  1,  8 ],
      [ 12,  0,  2,  3 ],
      [  1,  2,  0, 10 ],
      [  8,  3, 10,  0 ]
]

# Step 3: 設定目標函式
def SAfunc(x):
    tmp_cost = 0

    for i in range(NUM_CITY-1) :
        tmp_cost += cost[x[i]][x[i+1]]

    tmp_cost += cost[x[NUM_CITY-1]][x[0]]

    return tmp_cost
```

```python
#  ==== 主程式 ====

np.random.seed(0)   # 若要每次跑得都不一樣的結果，就把這行註解掉

# Step 1: 找初始解 x
   # (1) 令 x 設定為 0, ..., NUM_CITY-1的一個隨機排列
x = np.random.permutation(range(NUM_CITY))        # ==== 有變更

    # (2) 設定 xbest
xbest = x  = x.copy()          # ==== 有變更
ybest = y = SAfunc(x)                 # 算 cost function

# 執行 simulated annealing
num_it = 0
t = T_high;

while num_it < MAXIT and t > T_low:

    for i in range(DWELL) :
        # Step 2: 找鄰居 xnew                   # ==== 有變更
        # (1) 先令 xnew[] = x[]
        xnew = x.copy()

        # (2) 接著，任選兩個整數 j[0], j[1] (不可等於0)，互換xnew[j[0]]和xnew[j[1]]
        j = np.random.choice(NUM_CITY, 2)
        xnew[j[0]], xnew[j[1]] = xnew[j[1]], xnew[j[0]]

        ynew = SAfunc(xnew)

        if ynew < y:              # keep xnew if energy is reduced
            x = xnew.copy()
            y = ynew

            if y < ybest:   # 若新的成本比較小，取代最佳解
                xbest = x.copy()
                ybest = y

        else:
            # keep xnew with probability, p, if ynew is increased
            if np.random.uniform(0.0, 1.0) < math.exp( - (ynew - y) / (K * t) ):
                x = xnew.copy()
                y = ynew

        print('Estimated minumum at: ', xbest)  # ==== 有變更
        print('\tfit = %d\n' %(ybest))          # ==== 有變更

    t *= T_scale
    num_it += 1
```