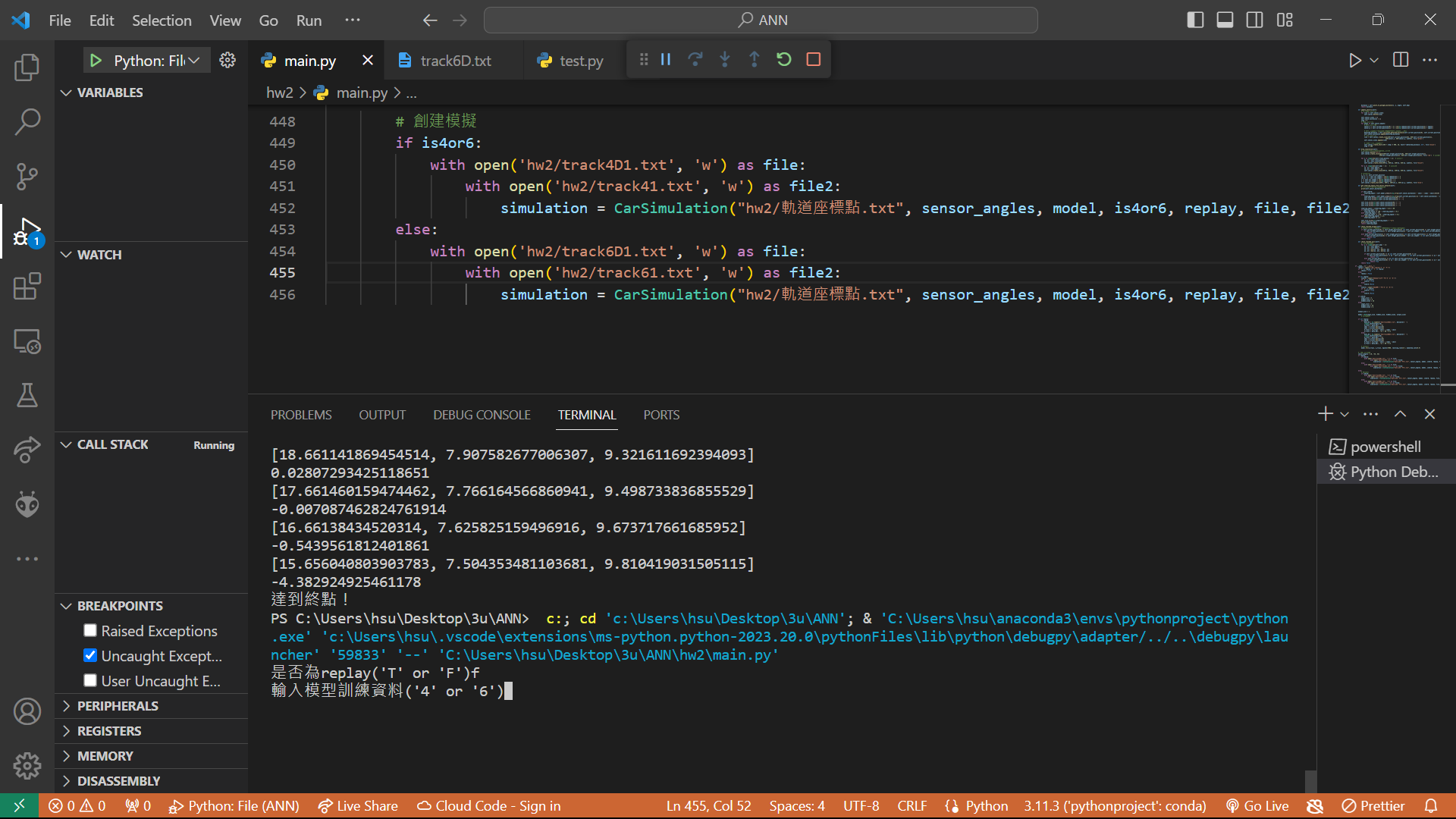
**類神經網路作業二**

**110502516許尚軒**

1. **程式介面說明**

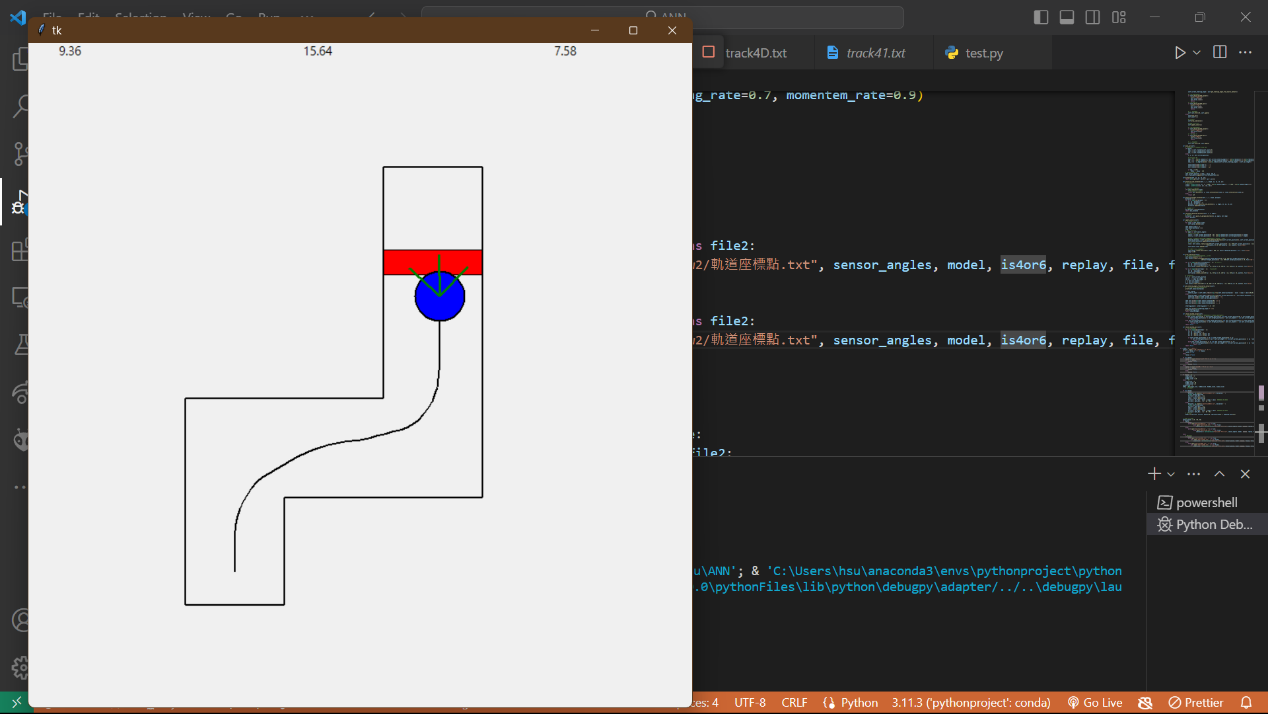
會先問是要重看之前的軌跡還是新訓練一筆資料

再問是要4d還6d



之後會訓練直到epoch=7000

然後就產生下方自走車模擬圖

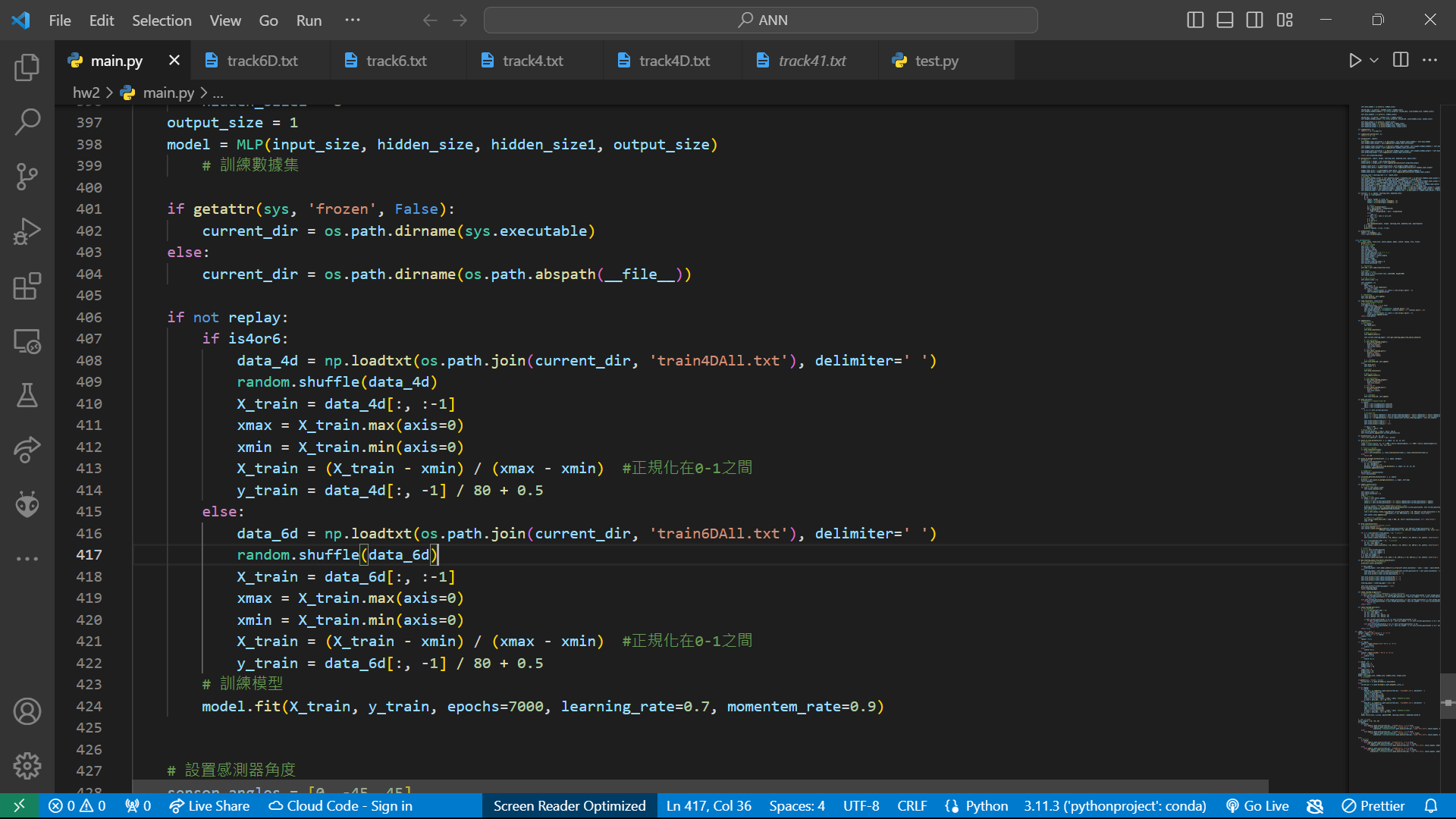
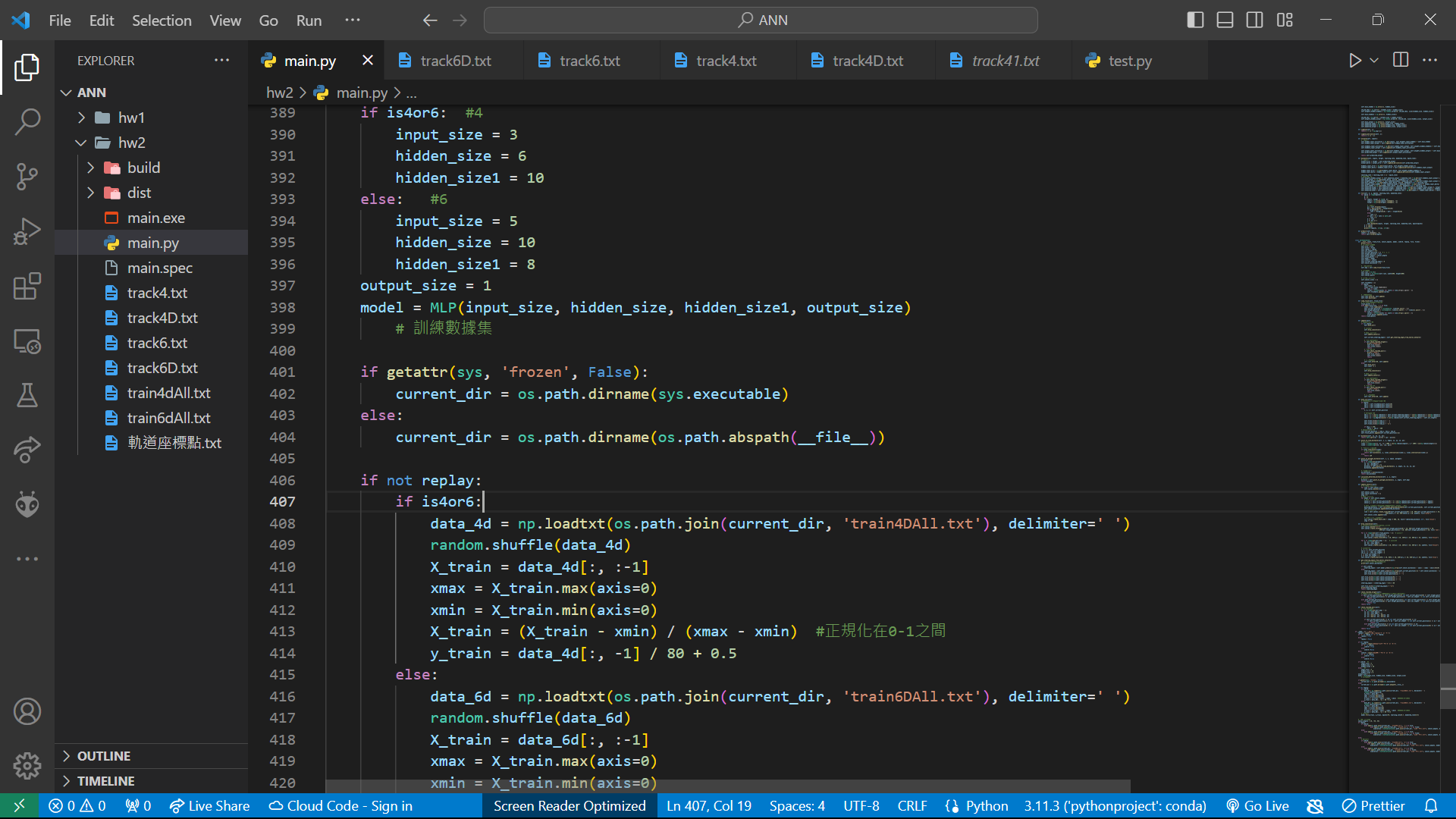


1. **程式碼說明**

程式分為2個部分，MLP模型和自走車模擬

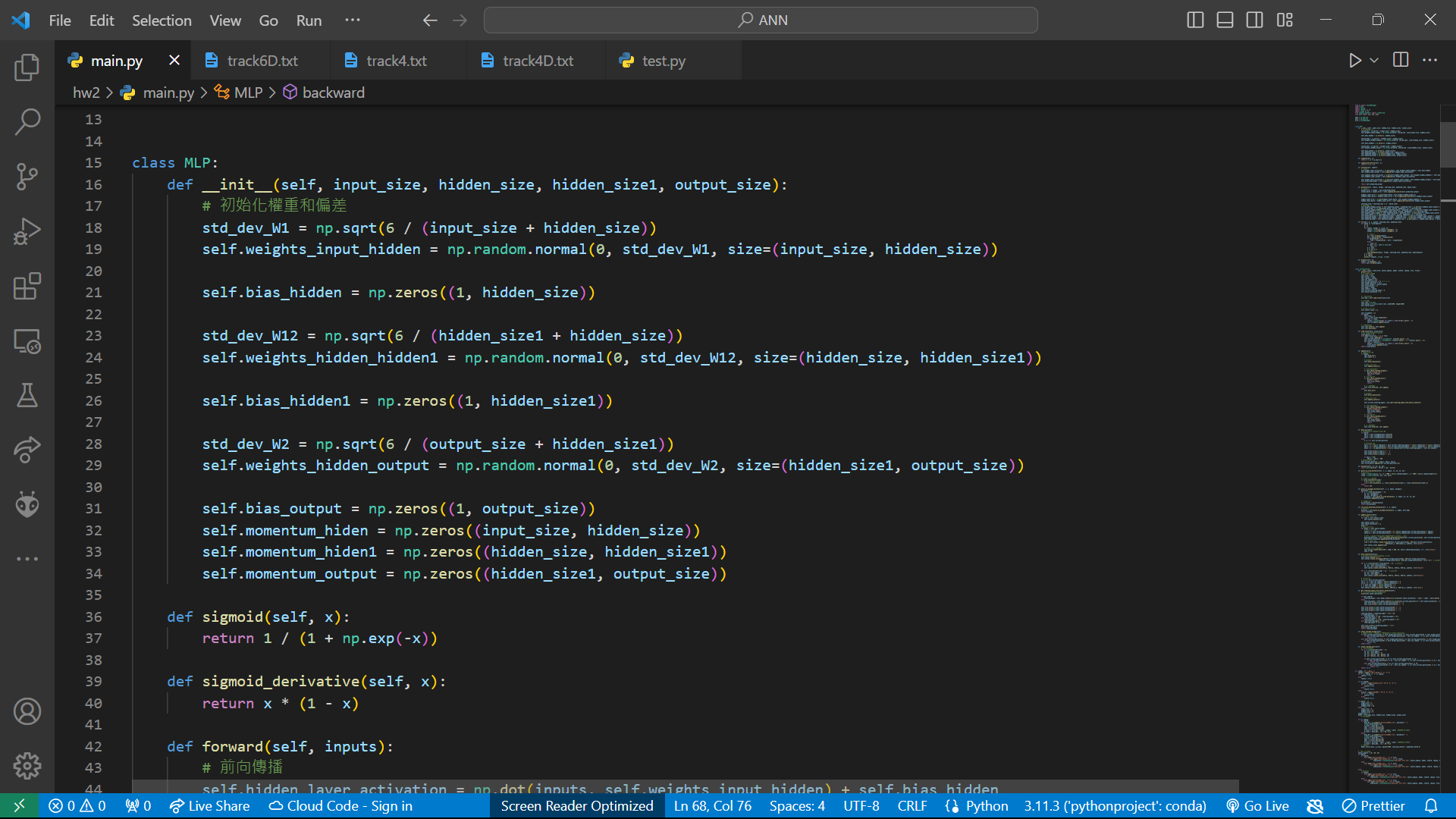
**MLP模型為四層的模型，當輸入為4d時各層的神經元數量為3, 6, 10, 1，當輸入為6d時各層的神經元數量為5, 10, 8, 1**

訓練資料會**打亂**送入模型訓練，**epoch為7000次，學習率為0.7(1-t/epoch)，慣性項係數為0.9**

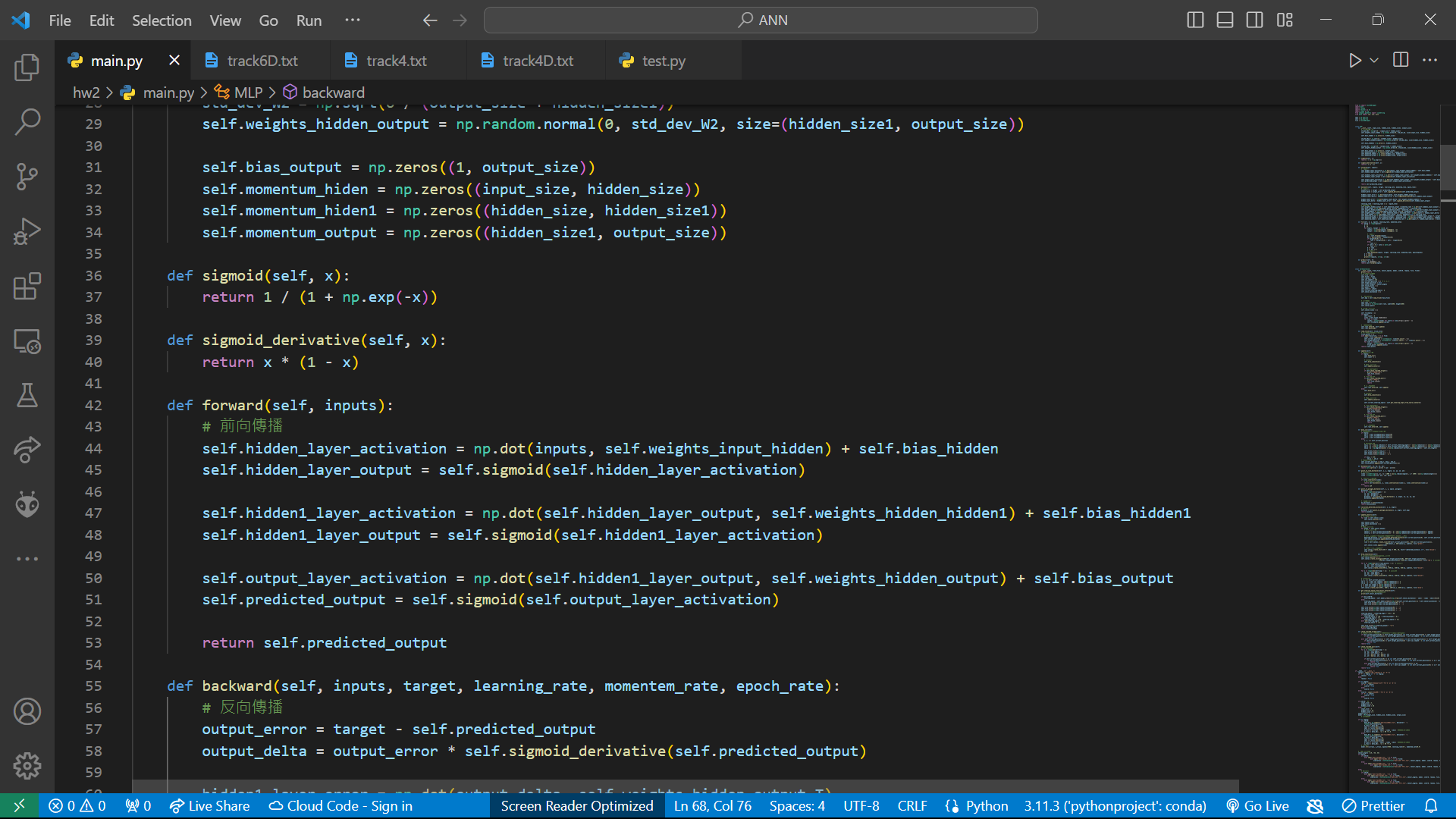


接下來看一下模型在做甚麼

先以**標準差為sqrt((前層神經元數目+後層神經元數目) / 6)，平均值為0**來隨機生成在小範圍內**均勻分布**之鍵節值，閥值及慣性項皆設為0

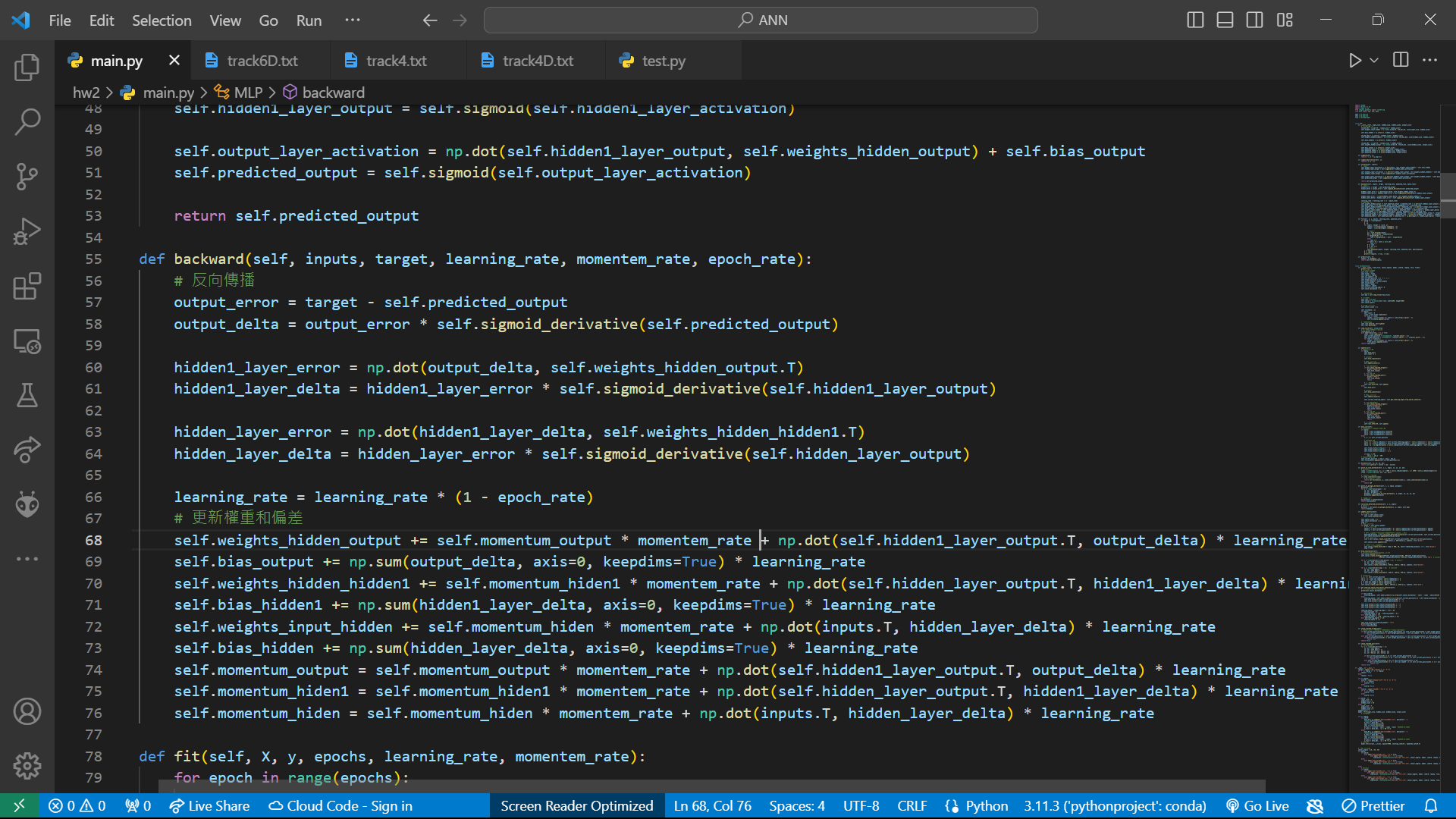


**前向傳播**中得出實際輸出

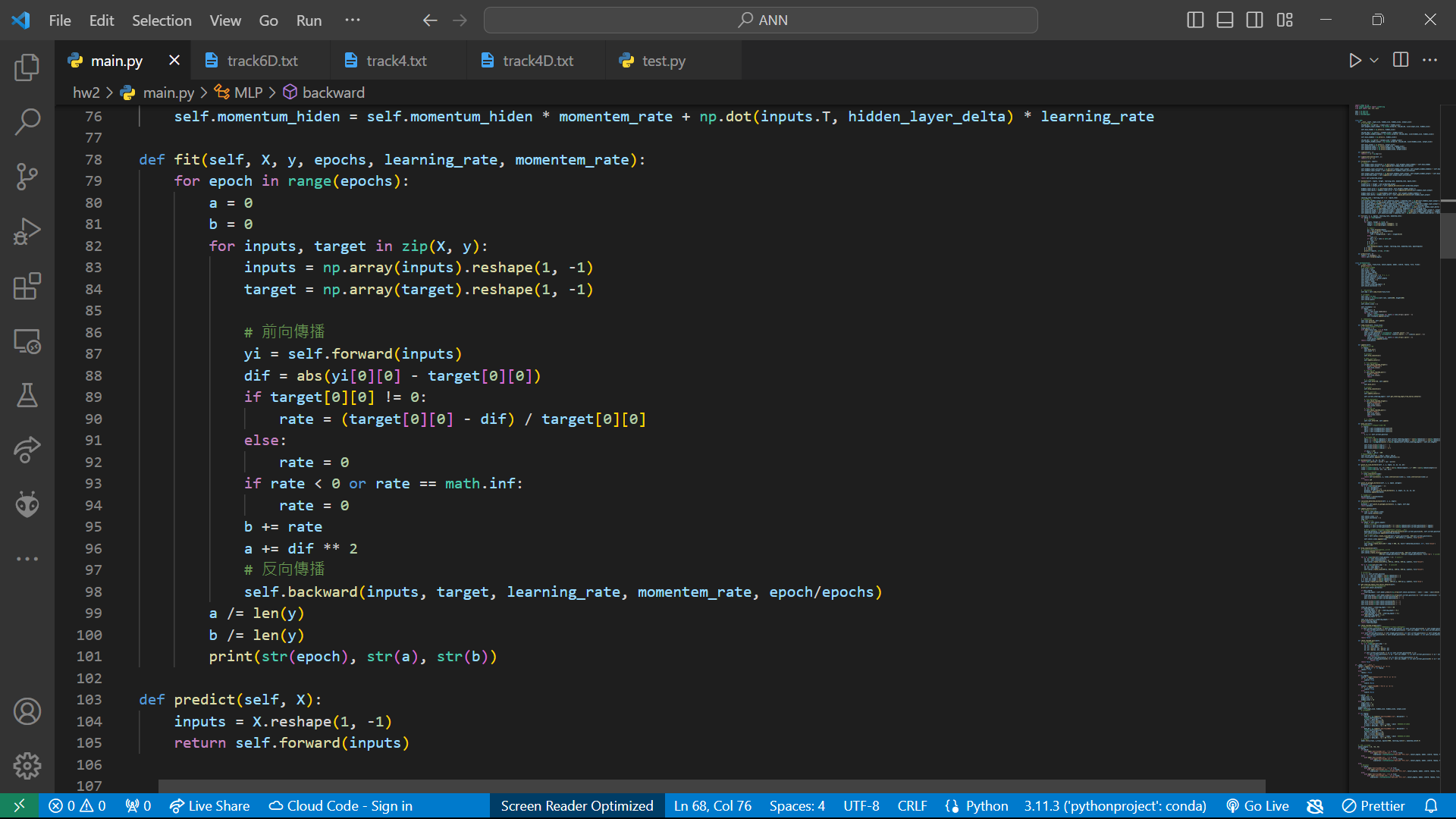


**倒傳遞**根據此實際輸出與預期輸出的差與其他項目改變鍵節值和閥值

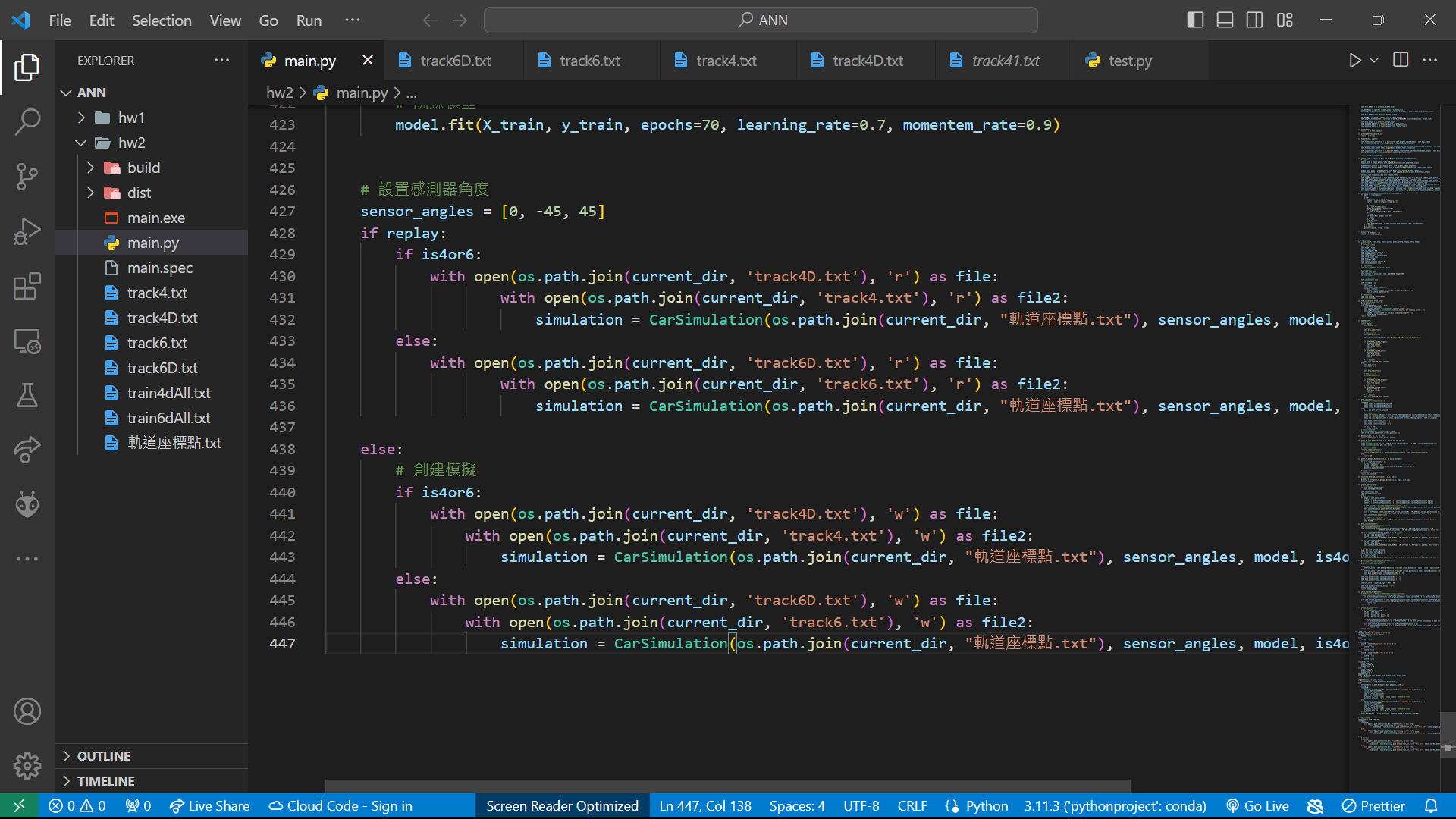
 公式為:



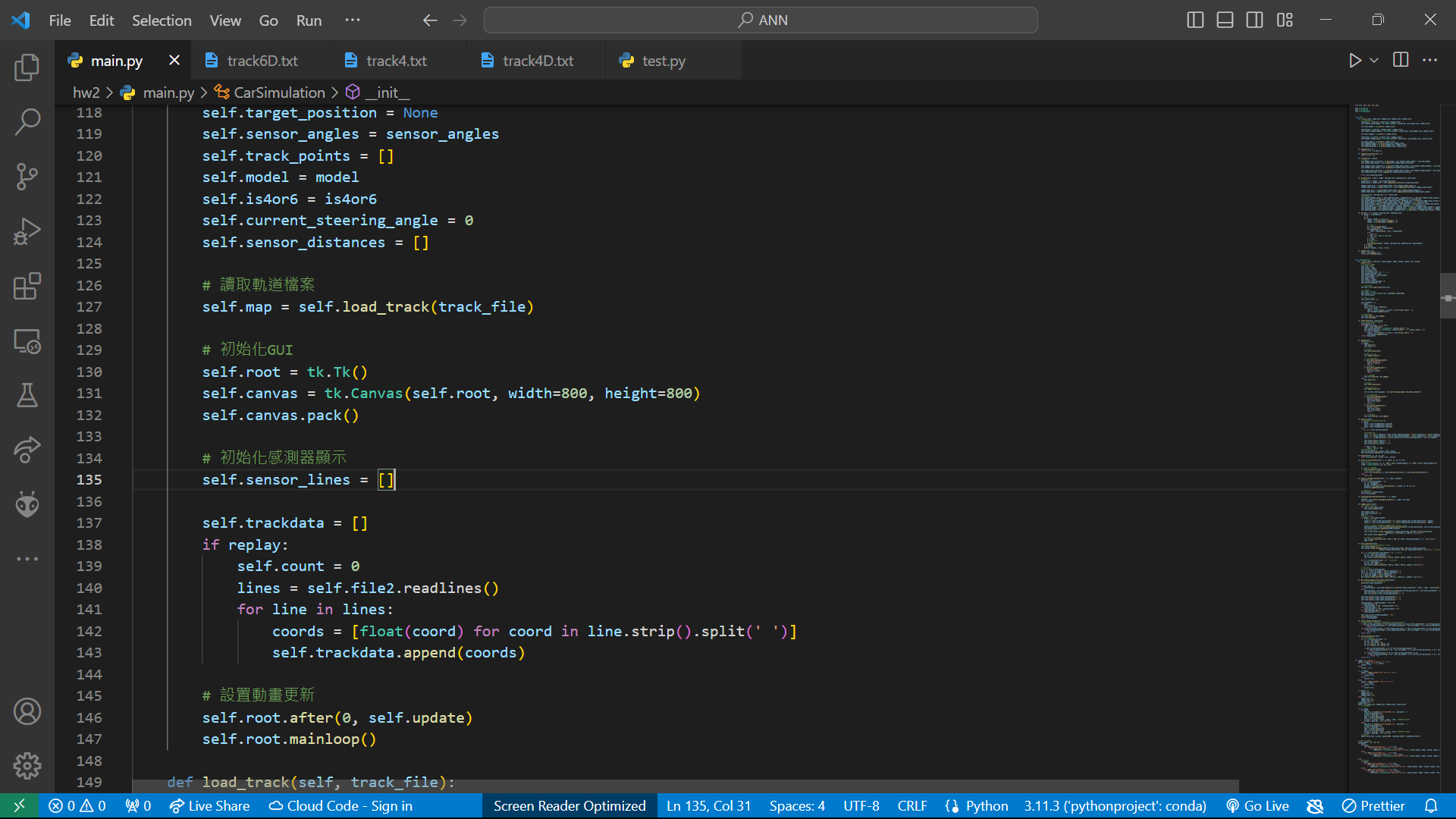
訓練會重複epoch次前向傳播和倒傳遞並計算出均方誤差及輸出相似度(|d - y| / y)av



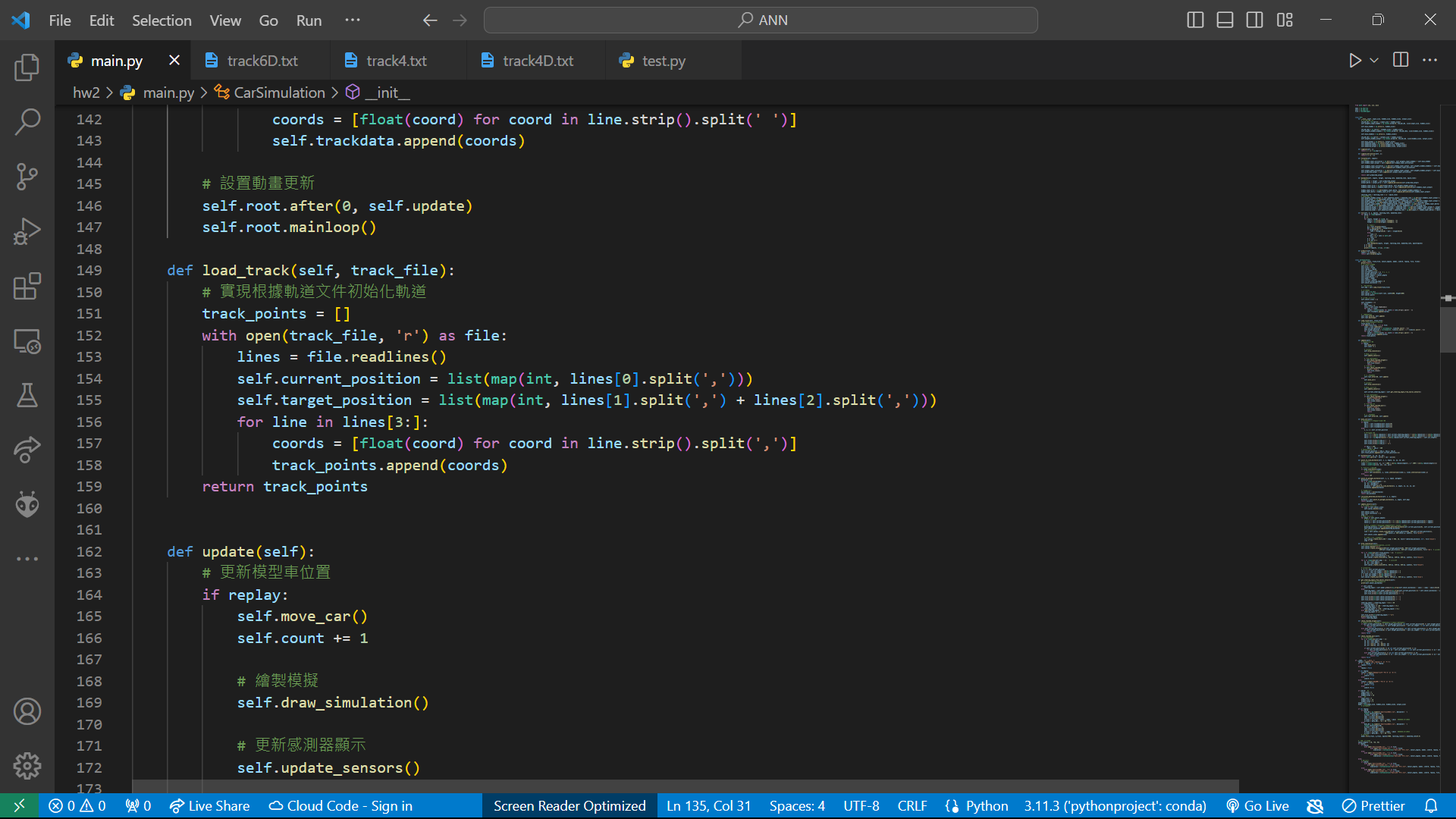
訓練完畢後就可以根據模型模擬自走車(因為replay只是根據txt檔去重現動畫，因此接下來接不會提到replay的作法)



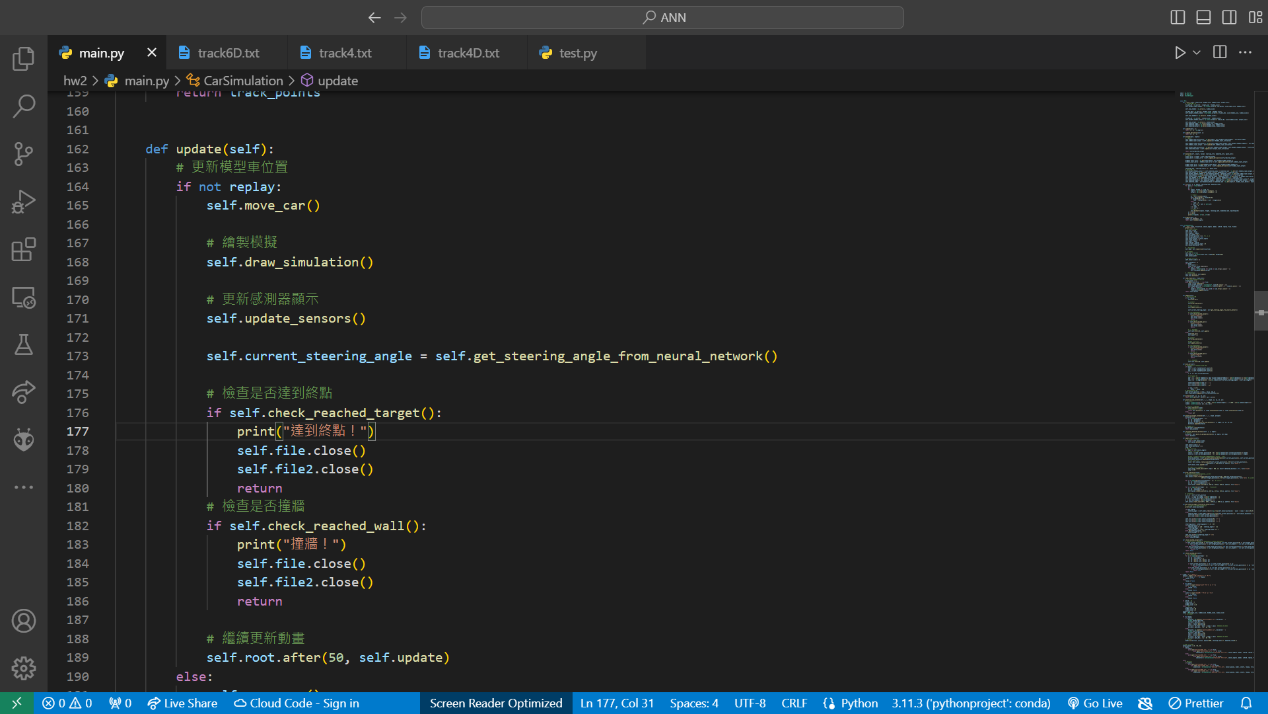
simulation一開始會先**初始化**許多變數並載入軌道



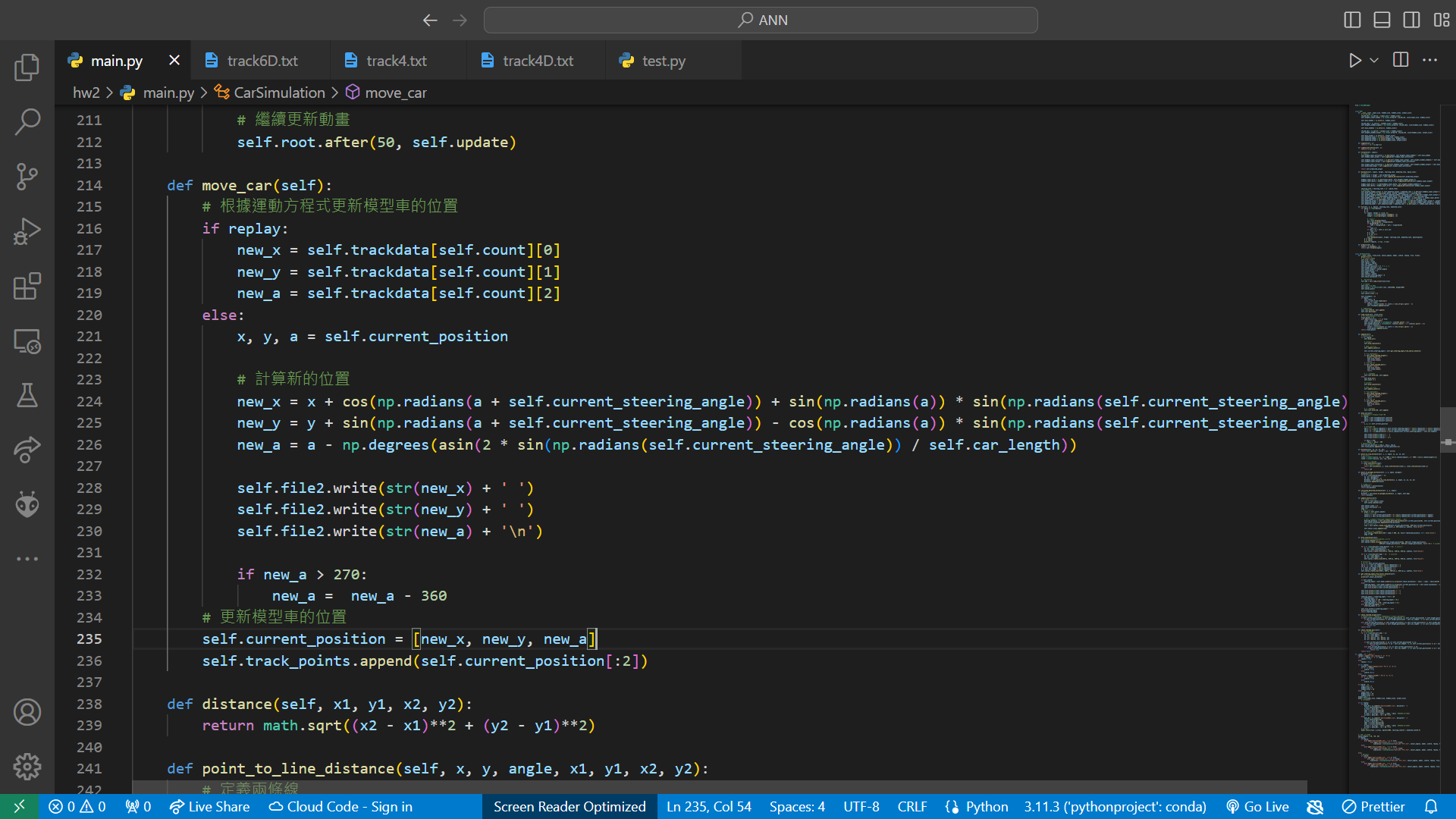
**載入軌道**如下



之後會在每個時間軸**更新**畫面，每次更新會先計算自走車位置再畫圖並計算偵測器偵測距離再根據此距離(以及x,y)計算方向盤角度，最後偵測是否撞牆或抵達終點

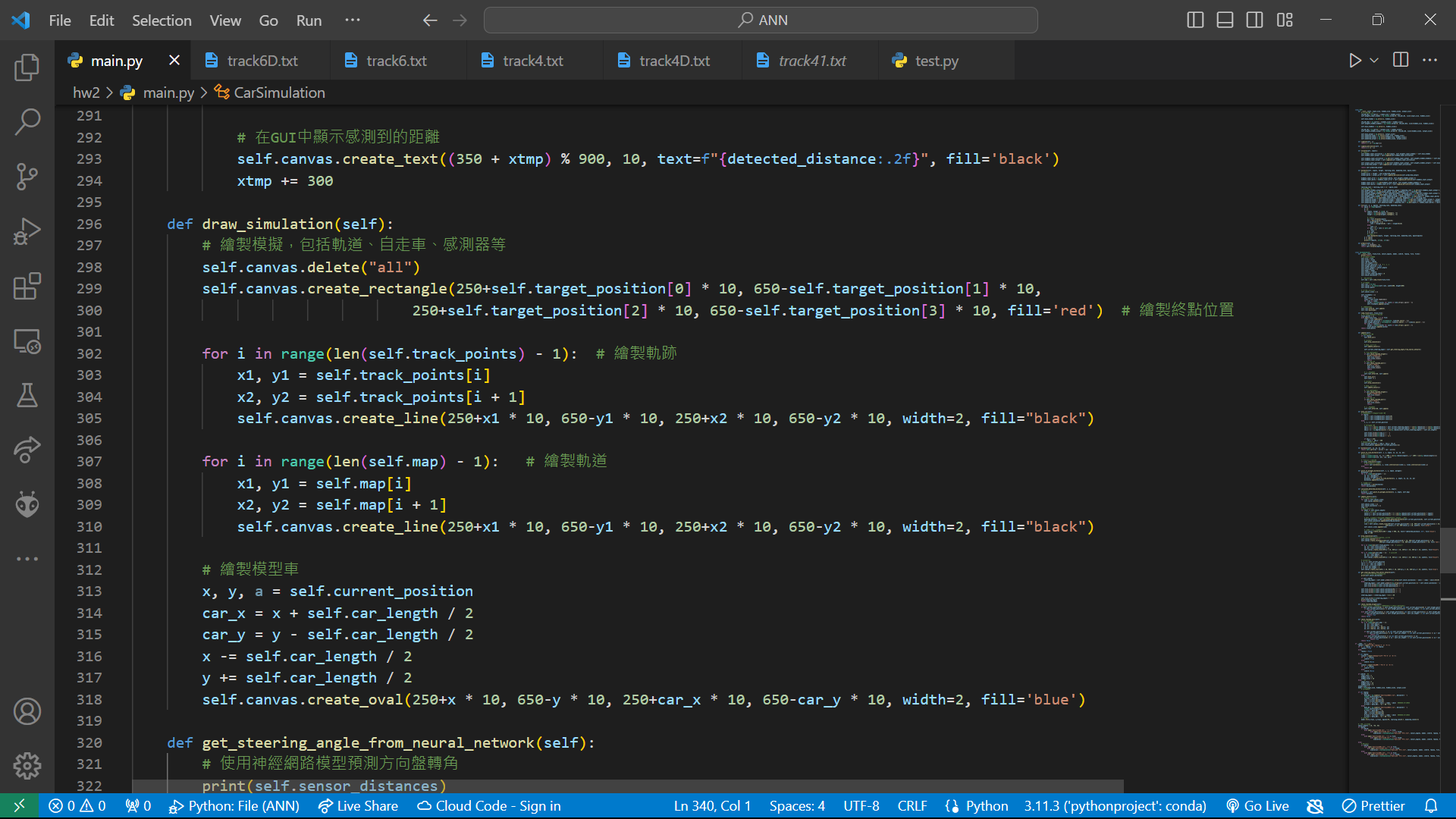


move\_car會根據公式**計算出新的位置和方向**

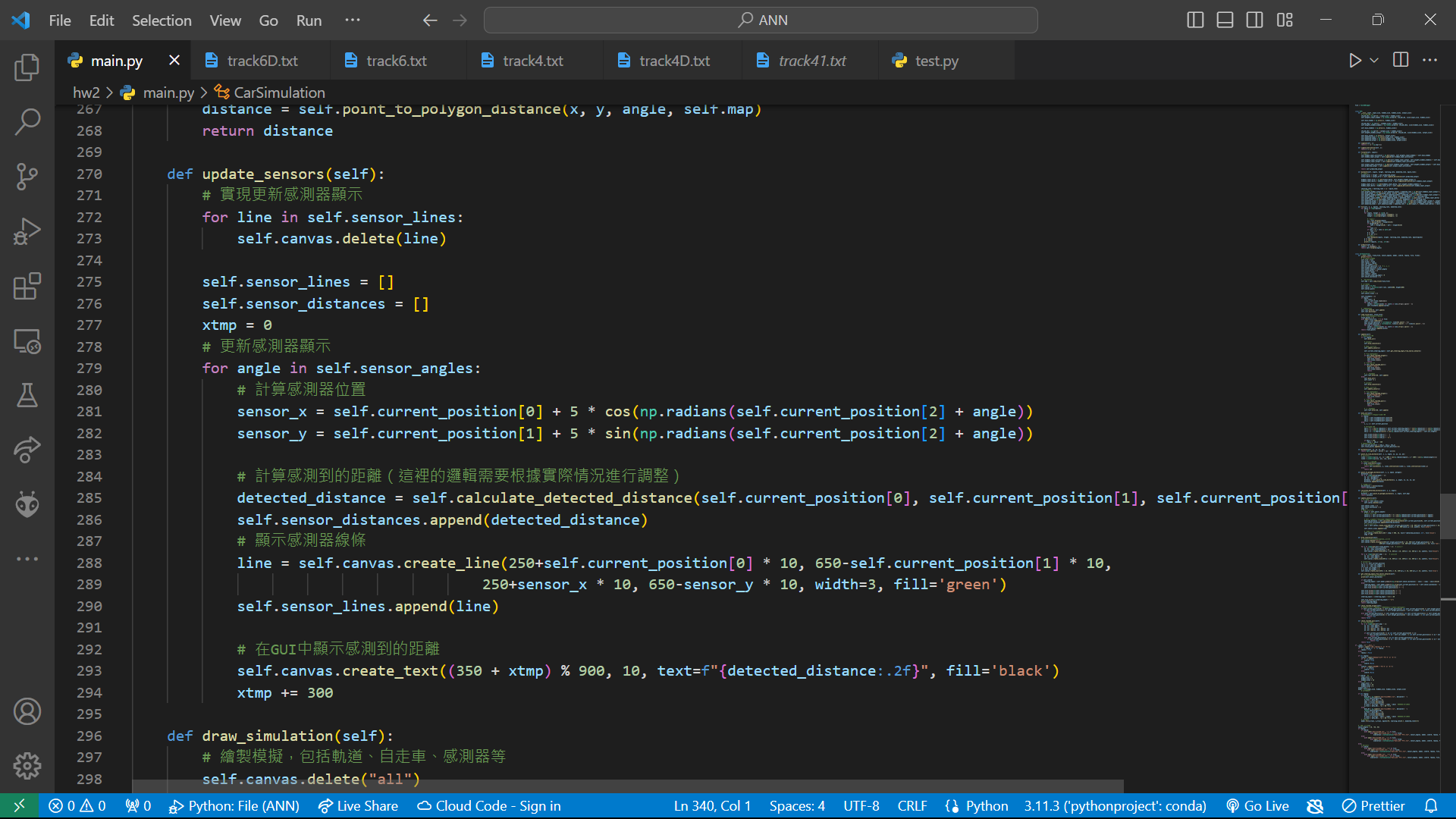


draw\_simulation會先清除畫布後畫出終點、軌跡、軌道、自走車，

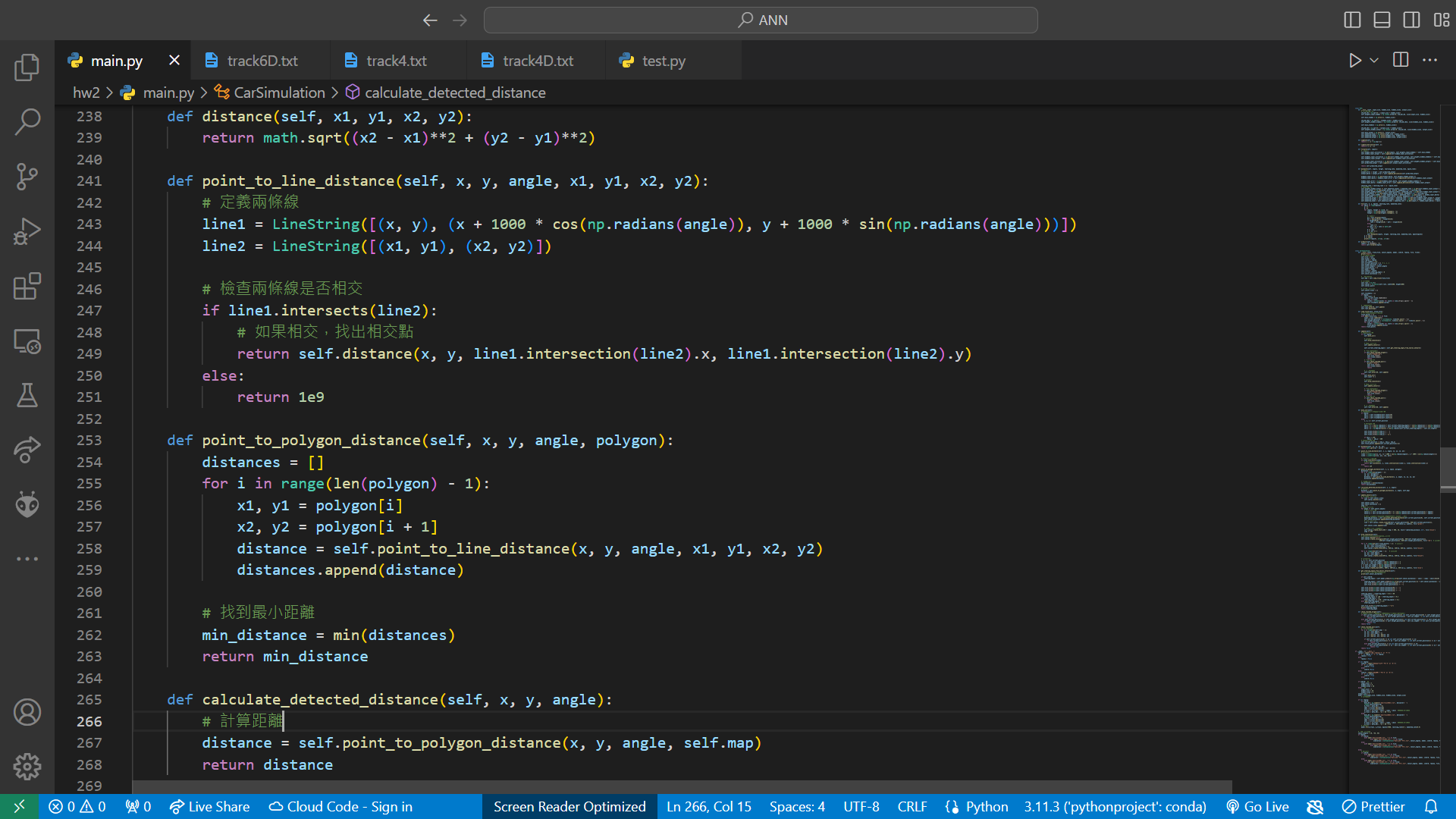
因為此次作業的xy區間都太小，因此\*10放大10倍



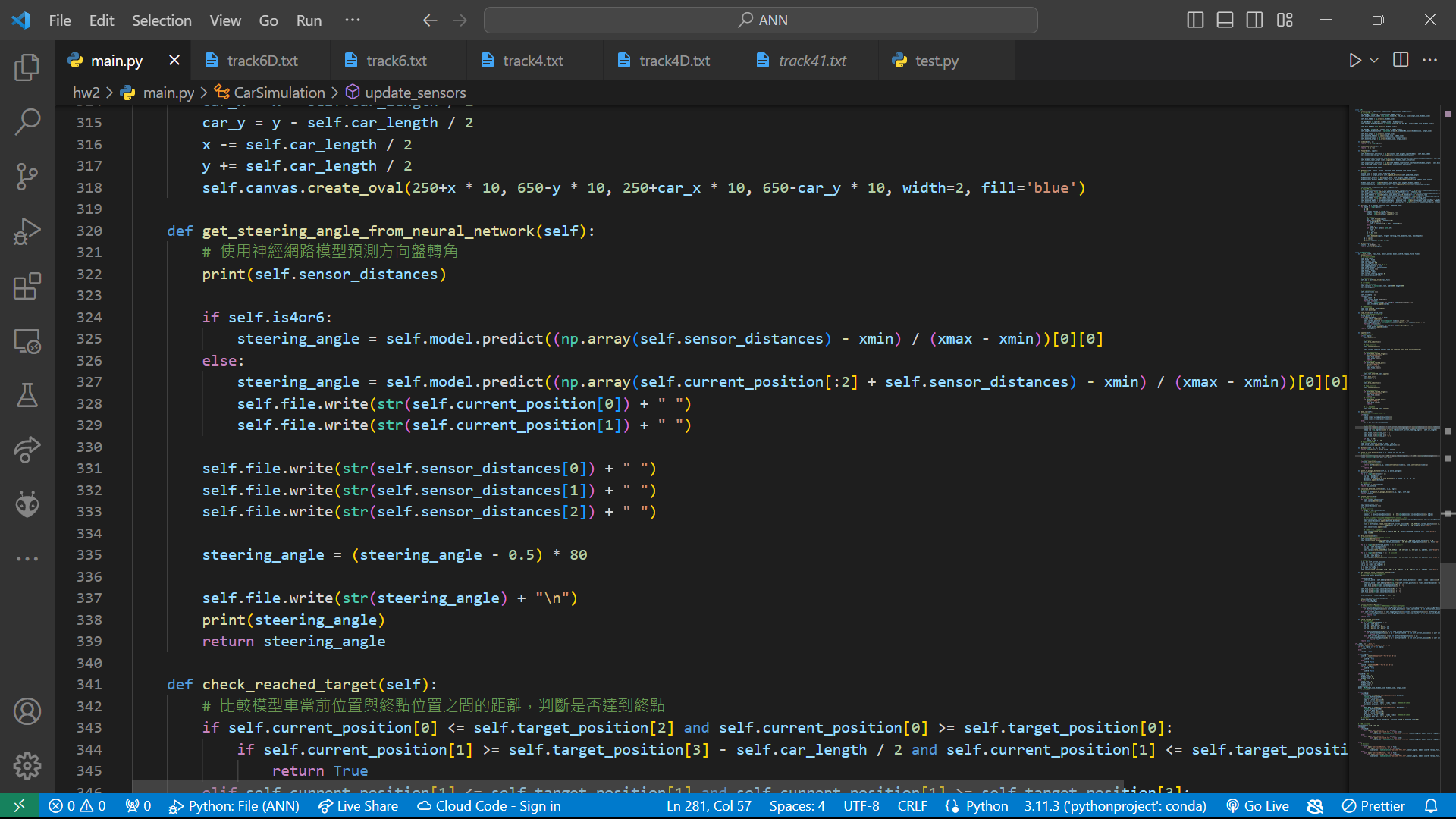
update\_sensor會計算出**偵測器測距並顯示**此距離及偵測方向



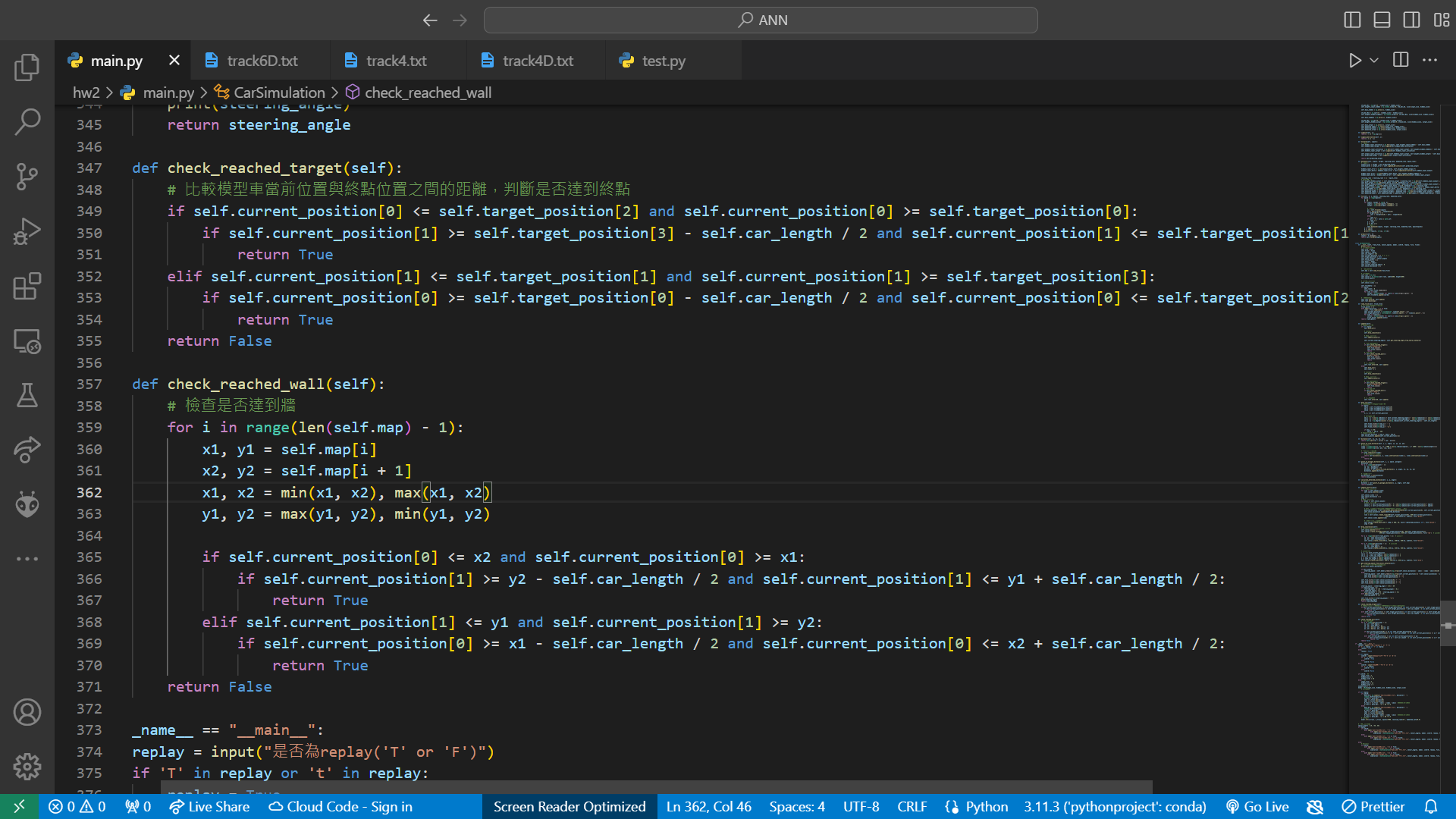
下列四個函式是用來計算偵測器測距，由下而上功用分別為回傳、計算測器偵測每條軌道的距離、偵測器射線與軌道之交點、焦點與自走車距離



根據上述測距及x, y丟入模型計算出方向盤角度



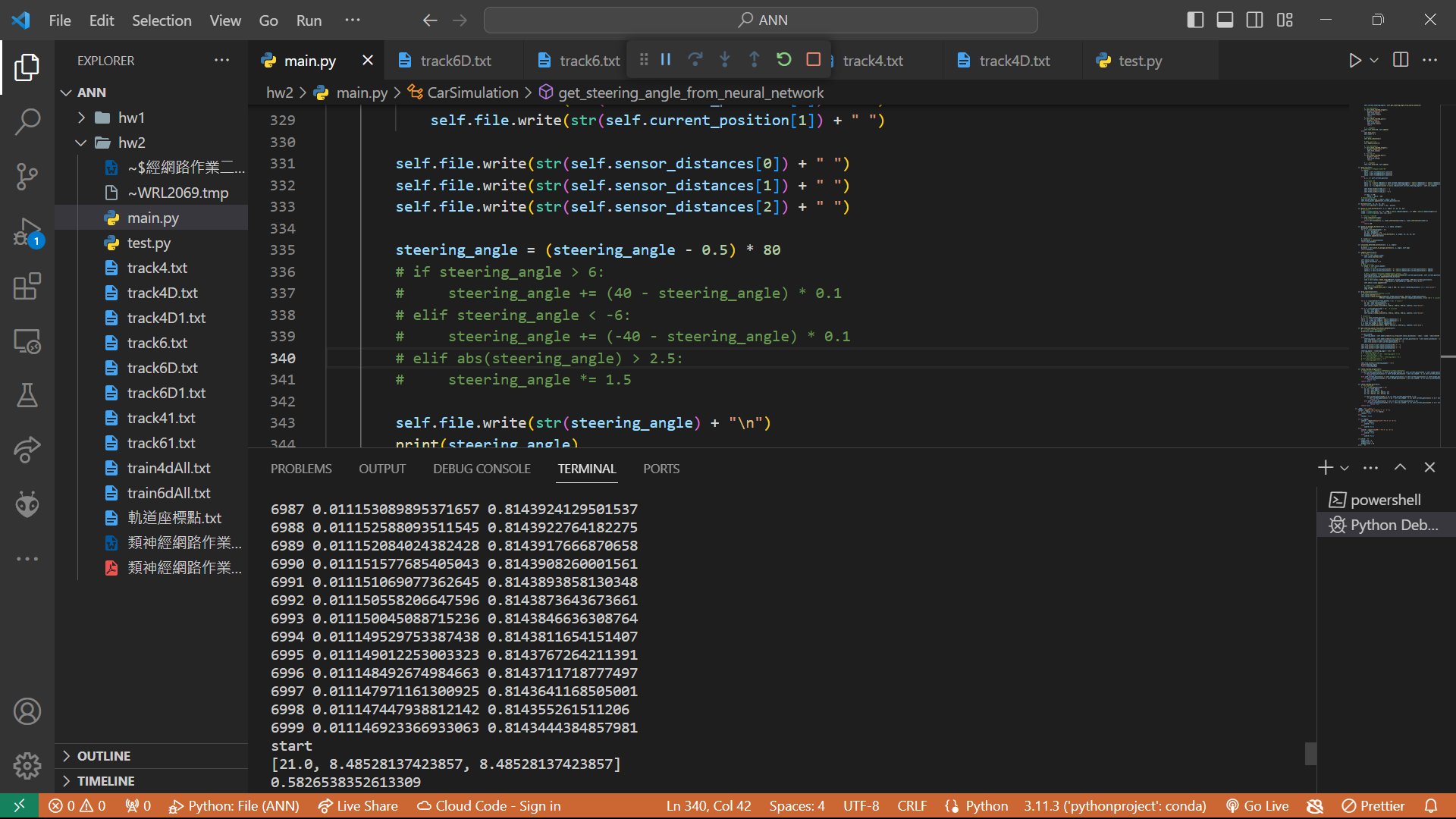
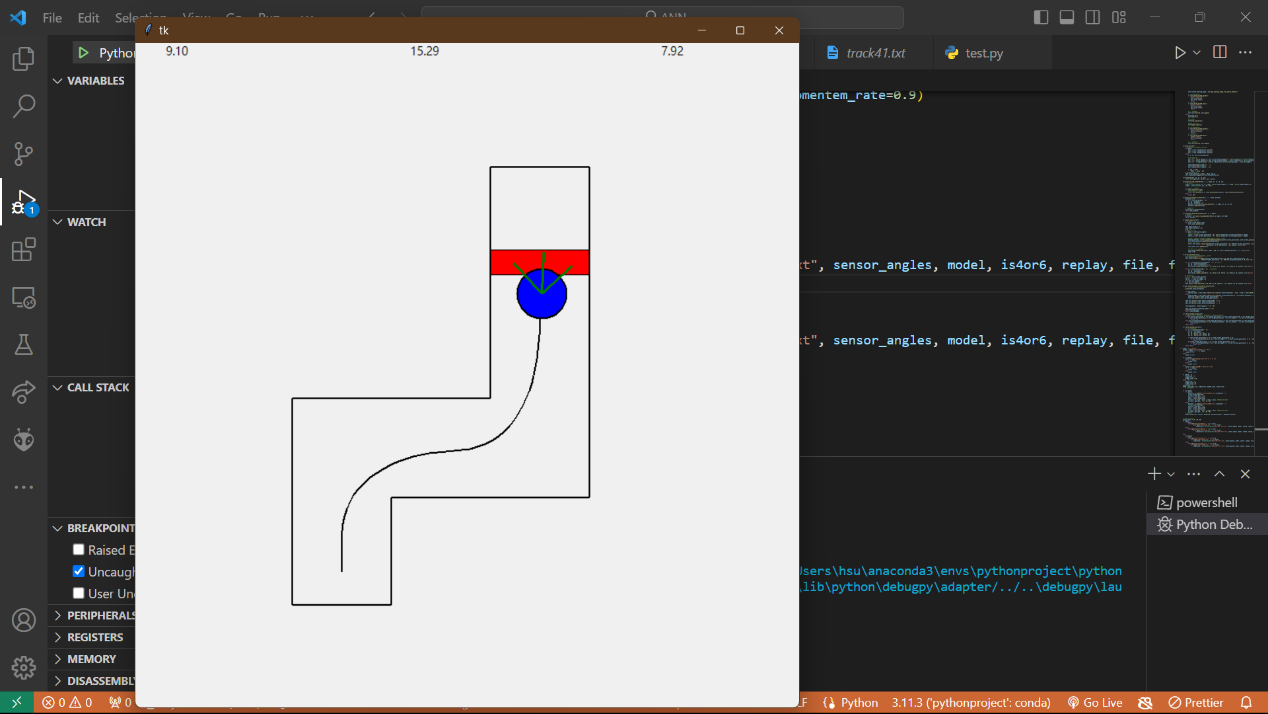
下方為偵測是否抵達終點或是撞牆，方法為判斷**車中心是否進入偵測對象**之x+-車身半徑, y+-車身半徑之範圍內



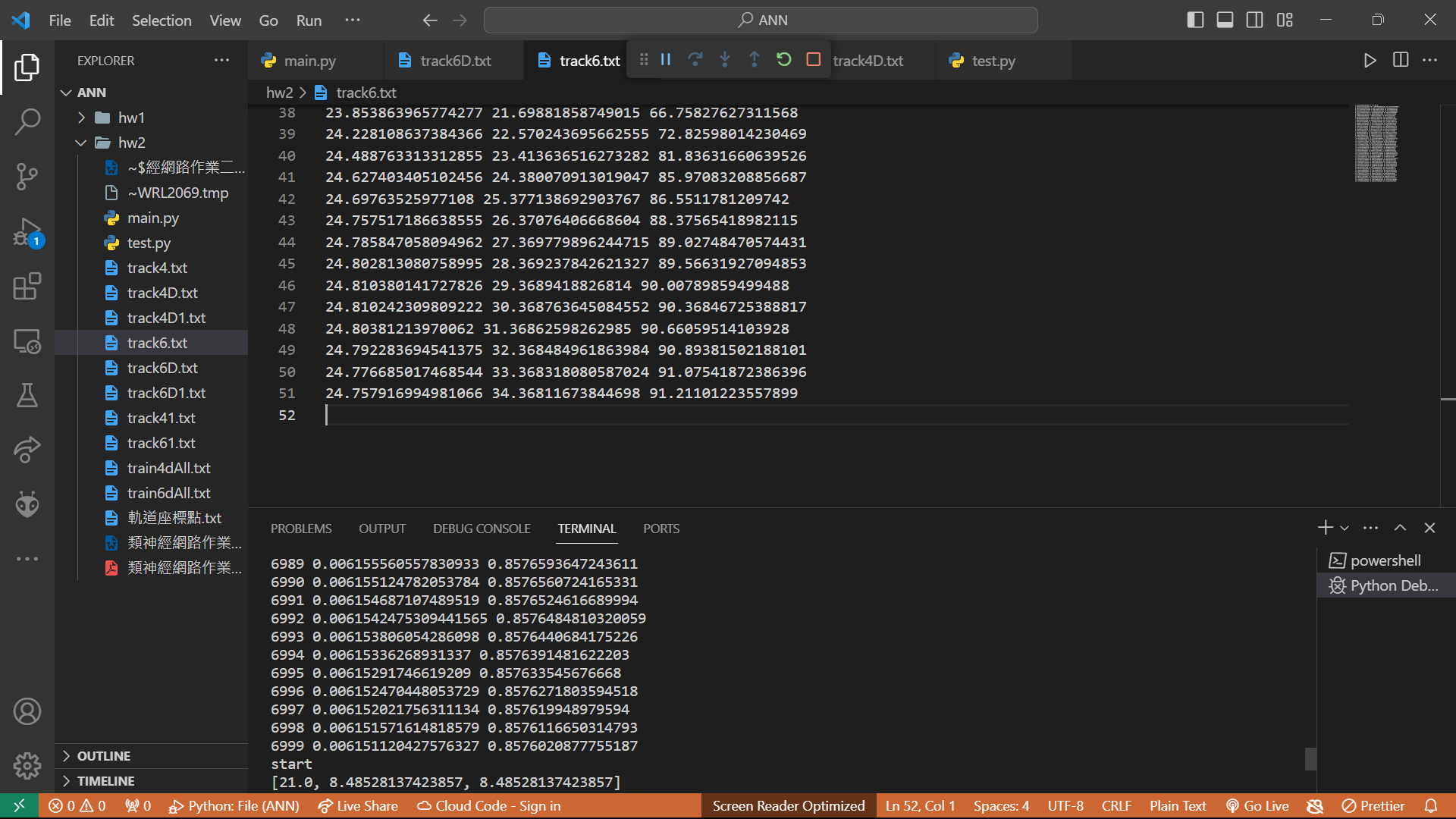
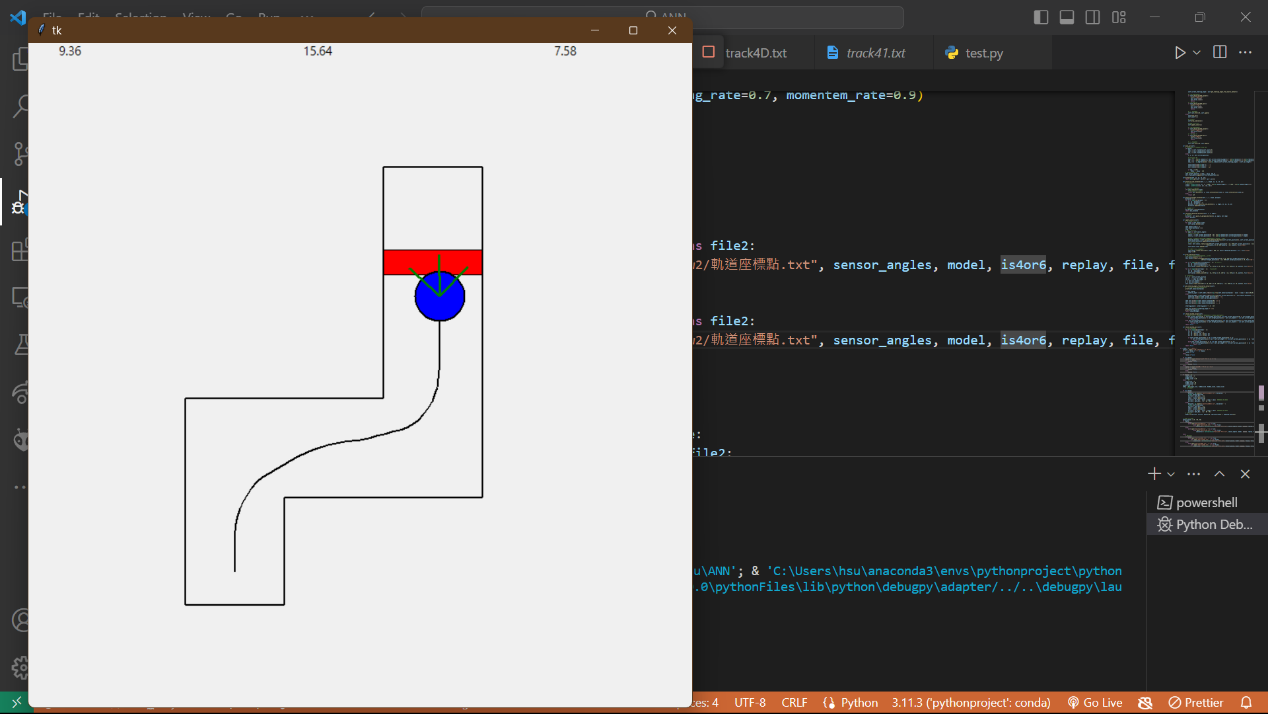
1. **實驗結果**

**4d:**

下方圖二每一行代表的是訓練次數、均方誤差、輸出的相似度(|d - y| / y)av



**6d:**



**重現方法:**

執行程式後輸入t下一步再輸入要重現軌跡之維度就可以了(track4.txt及track6.txt即為軌跡紀錄(track4D.txt及track6D.txt為作業要求的檔案)

1. **分析**

可以看出來6d轉彎角度比較犀利，我認為是因為除了測距外還多了x, y可以判斷現在應該轉幾度，因此比起4d可以較晚再轉大角度，像是在上述結果中4d方向盤角度只在[23, -24]之間，6d卻在[24, -35]之間，比4d大上不少。

而且6d的均方誤差也比4d少，我認為原因也是相同，當出現測距只相差一點但轉角差很多時比較不會互相影響到。