**인공지능 TSP 과제 샘플코드 설명**

# 과제 개요

1. Problem: Traveling Salesman Problem
2. Goal: 주어진 도시들의 좌표를 사용하여 시작 도시에서 **다른 도시를 모두 한번씩 거쳐서** 다시 시작 도시로 돌아오는 **최소의 경로** 탐색
3. 사용 알고리즘
   1. Greedy Algorithm
   2. Hill Climbing (2-opt)
4. 조건
   1. 출력 사항: 1) 탐색된 경로, 2) 탐색된 경로의 Cost, 3) 탐색에 걸린 시간  
      **(주의: 경로의 Cost는 시작 도시에서 다시 시작 도시까지의 경로임)**
   2. 최대 도시 개수: 2000개 이하
   3. 실행 시간: 1분 미만  
      **(데이터 로딩 후 시간 측정을 시작, 경로 출력까지의 걸린 시간을 구하여 출력)**
   4. 권장 언어 및 개발 환경: 언어 – Python, 개발 툴 – Jupyter Notebook  
      **(샘플 코드의 구조를 거의 따라가되 초기화 및 탐색 알고리즘을 변경하는 선에서 다른 언어 및 개발 툴 사용 가능)**

# 환경 구축

1. Anaconda 설치
   1. Anaconda.com에서 직접 다운로드하여 설치 및 실행
   2. 또는, <https://gracefulprograming.tistory.com/124> 참고
2. Eclass에서 샘플코드/데이터 다운로드
3. Jupyter Notebook 실행 (윈도우의 경우 시작에서 jupyter 검색 후 실행)

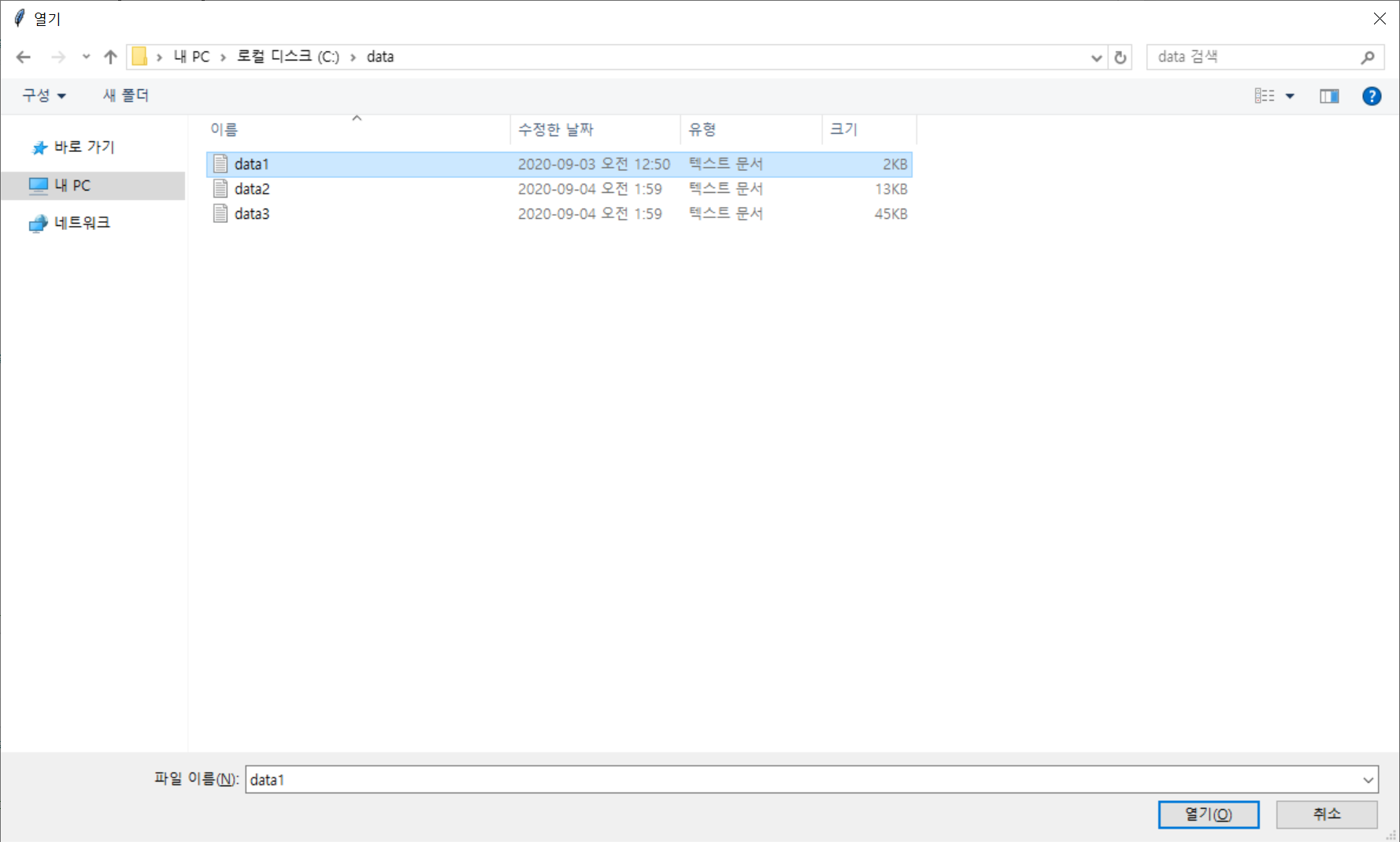


1. Upload 버튼을 클릭하여 다운 받은 파일 중 TSP\_HillClimbing.ipynb, TSP\_Greedy.ipynb 를 업로드 한 후 파란색 Upload 버튼을 눌러 파일을 최종 업로드
2. 업로드 된 파일 제목을 클릭하여 실행

# 코드 실행

## 데이터 열기

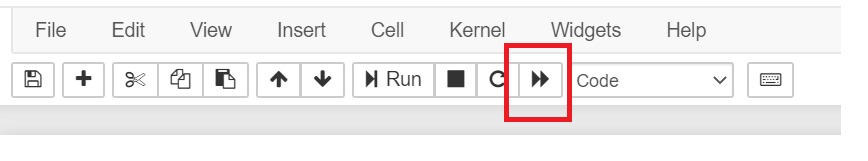
**파일 열기 창이 보통 화면 앞으로 나오지 않음 - Alt-tab을 눌러 열기 창을 선택**



다운 받은 파일에 포함되어 있는 3개의 데이터 중 하나를 선택하여 열기

* 도시의 개수에 따라 또한 도시의 배치에 따라 요구되는 조건을 충족할 수 있는 알고리즘이 달라 질 수 있음.
* 이러한 점을 체험하기 위해 본 과제에서는 3개의 서로 다른 도시 개수를 가지는 데이터가 제공됨.
* data1: 총 도시 개수 131개
* data2: 총 도시 개수 1083개
* data3: 총 도시 개수 1599개

## 코드 실행 시키기



전체 코드 실행은 상단 메뉴에서 빨리 감기 버튼을 클릭

# 코드 설명

## TSP\_Greedy.ipynb

### 알고리즘 설명

1. 경로를 시작할 도시를 지정
2. 시작 도시에서 나머지 도시 중 가장 가까운 도시를 방문
3. 방문한 도시에서 다시 방문하지 않은 도시 중 가장 가까운 도시를 방문
4. 모든 도시를 방문할 때까지 3번 반복
5. 모든 도시를 다 방문하면 마지막으로 방문한 도시에서 시작 도시로 복귀

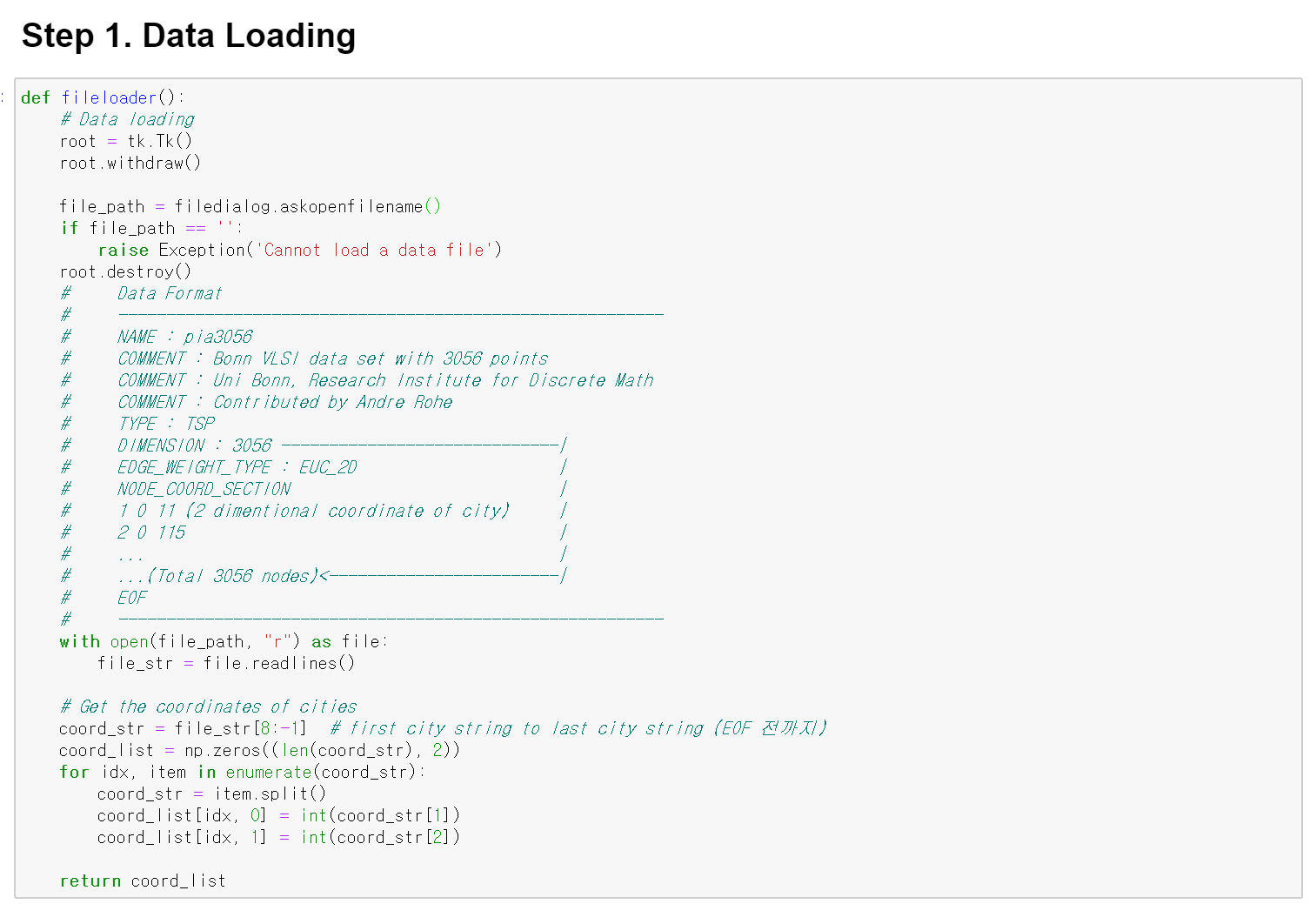
### Step 0 – Importing Packages and Global Setting



동작에 필요한 세팅은 Step 0 섹션에서 제어

1. SUB\_ITERATIONS: 2-opt를 한번에 몇 번 반복할 것 인지 지정 (Hill Climbing을 위해 사용)
2. MAX\_EVALUATION: Hill Climbing에서 몇 번 최적 경로와 탐색된 경로를 비교할 것인지 지정 (Hill Climbing을 위해 사용)
3. PLOT\_MODE: 탐색의 경로를 보고 싶을 때 True, 아니면 False
4. PLT\_INTERVAL: Hill Climbing에서 2opt에서 몇 iteration 마다 경로를 그릴 것인지 지정
5. FIRST\_IDX: 시작 도시의 index를 지정

### Step 1 – Data Loading



사용할 데이터를 불러옴

1. 모든 데이터의 포맷은 동일함
2. 도시는 좌표로 나타나며 NODE\_COORD\_SECTION다음부터 각 도시의 좌표가 도시 개수만큼 나옴
3. 데이터의 마지막은 EOF로 끝남
4. 각 도시의 좌표를 배열을 다루는 패키지인 numpy를 사용하여 저장

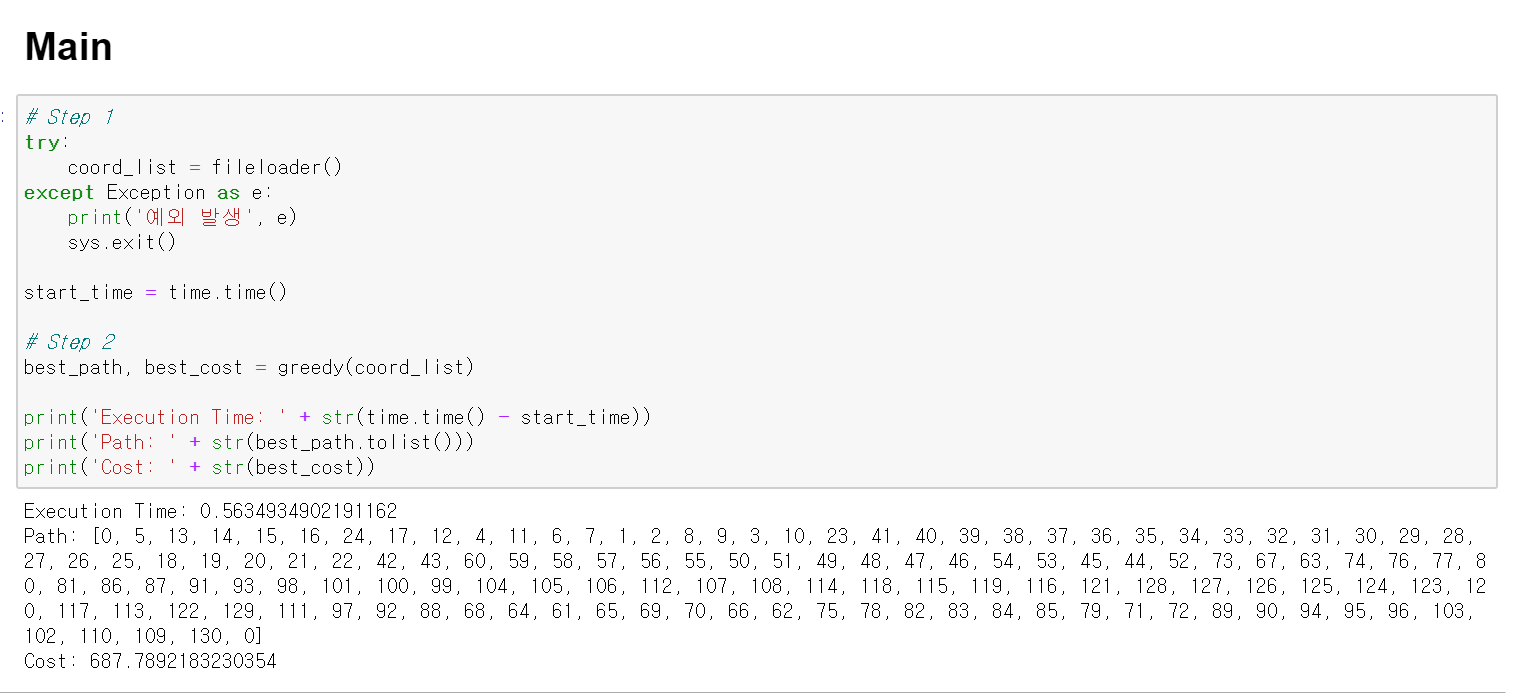
### Step 2 – Searching a path (Greedy Algorithm)



각 도시의 좌표를 받아 상기에 명시된 방법으로 최적 경로를 Greedy 기반으로 탐색

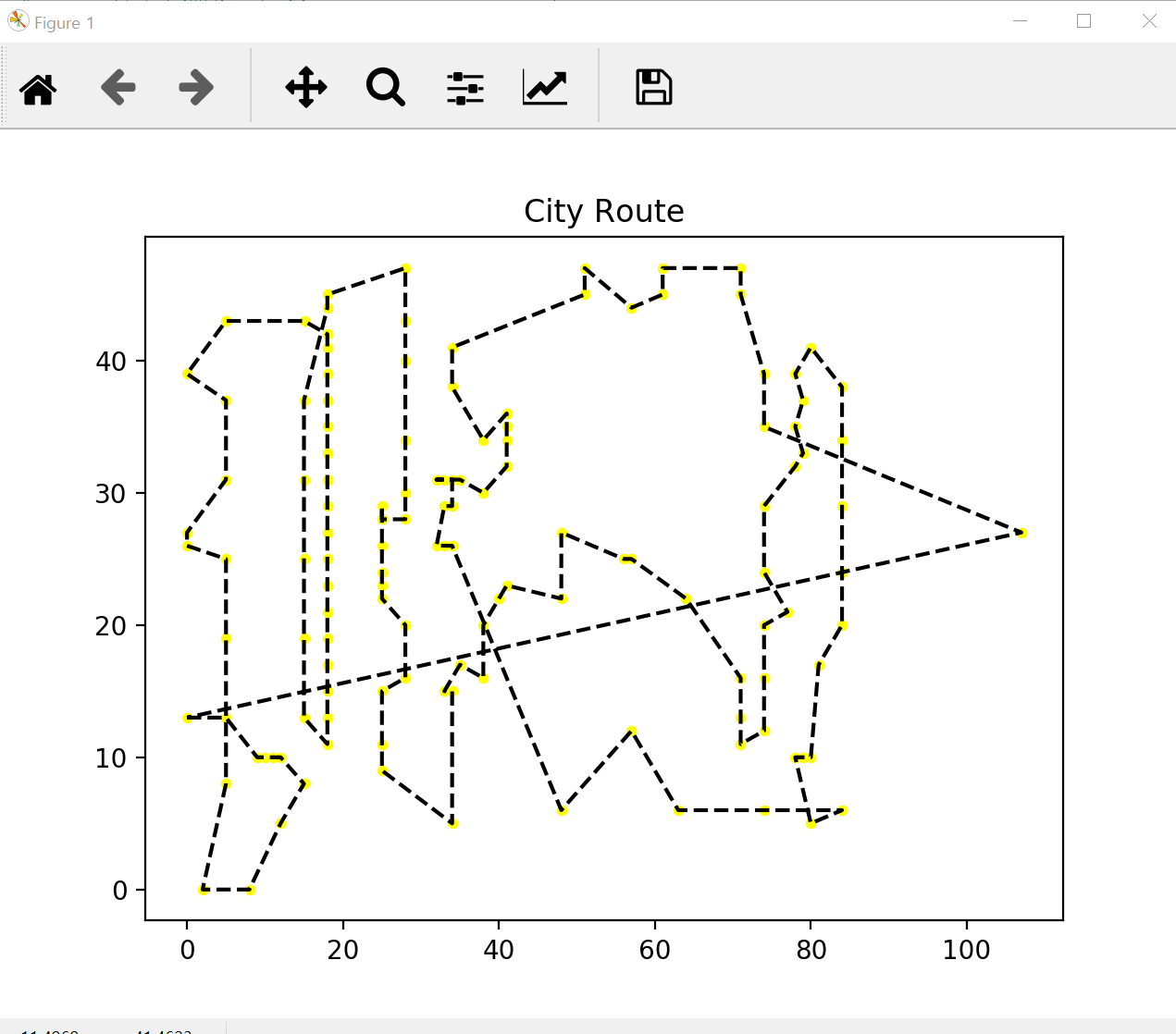
1. 중요 변수 설명:
   1. coord\_list: 각 도시의 좌표 저장
   2. best\_path: 현재 가장 최적의 경로
   3. path\_map: 각 도시간의 거리를 (도시 개수 by 도시 개수) 크기의 행렬에 저장
   4. cities\_tovisit: 방문해야할 도시를 True, 방문한 도시를 False로 저장
2. 함수 설명:
   1. path\_cost: path\_map과 경로를 받아 경로상 각 도시간의 거리를 저장한 array를 리턴
   2. greedy: coord\_list를 받아 Greedy 기반으로 최적 경로 및 경로의 총 거리를 리턴

### Main function



위의 정의된 함수들을 Step 대로 진행하여 최적 경로, 걸린 시간, 총 거리를 출력

세팅 섹션에서 PLOT\_MODE를 True로 할 시에 경로 출력



**(경로를 그린 그림의 경우 또한 처음에 화면 앞으로 나오지 않으므로 Alt-tab을 눌러 경로 확인)**

## TSP\_HillClimbing.ipynb

### 알고리즘 설명

1. 경로를 초기화 (Greedy 방식 또는 Random 방식)
2. 2-opt 기반으로 각 경로를 랜덤하게 뒤집으면서 최적 경로를 탐색
3. 더 이상 개선된 경로가 나오지 않거나 종료조건이 되면 탐색 종료

### Step 0, 1은 Greedy 파일과 동일

### Step 2. Initialization



경로를 초기화

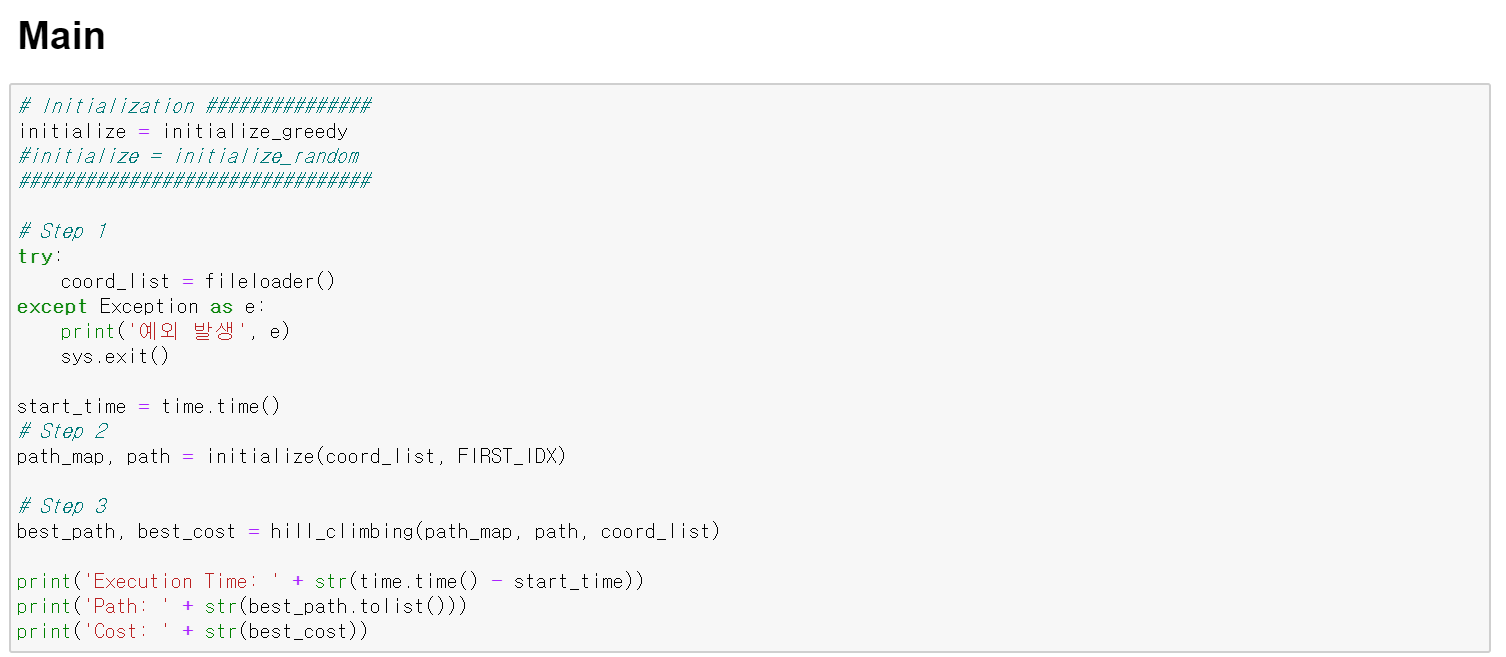
1. Greedy알고리즘으로 경로를 초기화 (좋은 조건에서 출발, 단 Local Optimal에 빠지기 쉬움)
2. 랜덤하게 경로를 초기화 (나쁜 조건에서 출발, 단 지속적인 경로 개선이 가능)
3. path\_cost함수는 동일함

### Step 3. Searching a path (Hill Climbing, 2-opt based)



2-opt로 경로를 개선, MAX\_EVALUATION 에 도달할 때 까지 경로를 개선함

### Main



Greedy와 거의 동일

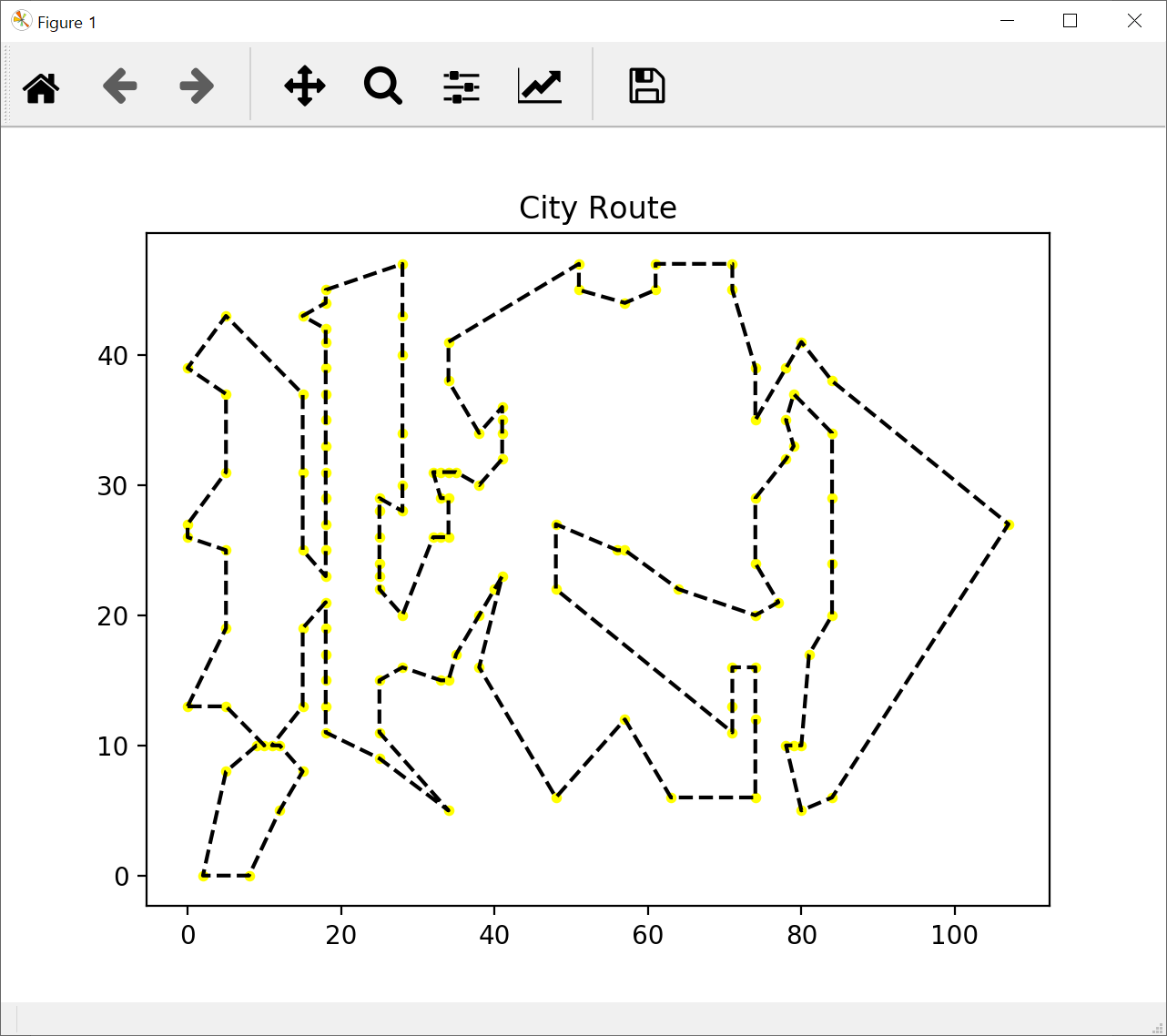
Initialization에서

initialize = initialize\_greedy를 쓸 경우 Greedy 기반으로 경로 초기화

initialize = initialize\_random를 쓸 경우 랜덤 기반으로 경로 초기화

한 개를 선택 후 나머지는 주석처리

**마찬가지로 경로를 그린 그림의 경우 또한 처음에 화면 앞으로 나오지 않으므로 Alt-tab을 눌러 경로 확인**



# Contact

수업조교: 박사과정 서왕덕

Mail: seowangduk@gmail.com

208관 218호 인공지능연구실