**Data Science**

**Assignment2 : Decision Tree**

2016025532 컴퓨터전공 심수정

**1. Summary of Algorithm**

decision tree는 어떠한 데이터를 입력받았을 때 tree의 형태에 맞추어 어떤 class에 속할지 알아내는 algorithm입니다. 이를 위해 tree를 만들기 위한 training set이 필요합니다.

Decision tree를 만들기 위하여 training set과 gain ration를 활용하게 됩니다. Training set의 attribute들을 받고, 이중 가장 잘 나누는 것을 기준점으로 삼아 tree의 label을 설정해 나누게 되는데 가장 잘 나누는 것을 찾기 위해 gain ratio를 활용합니다.

Gain ratio는 다음과 같이 계산합니다.

여기서 information gain은

Info는

는

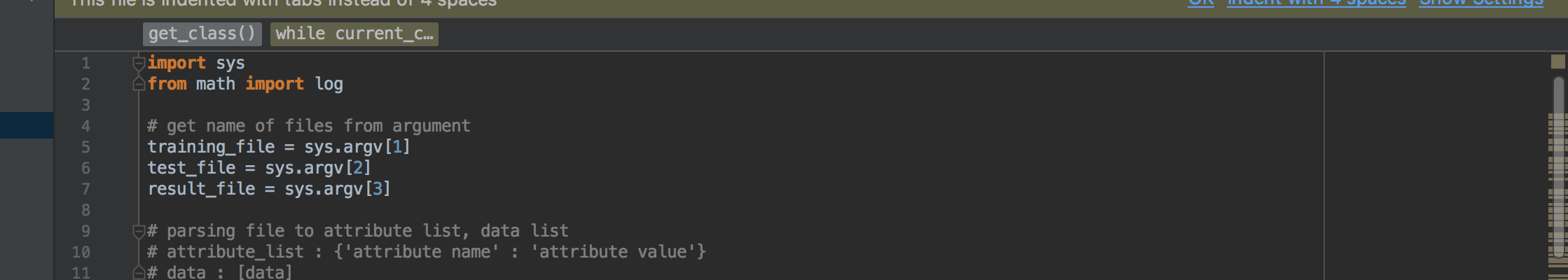
으로 계산됩니다.

Split info는

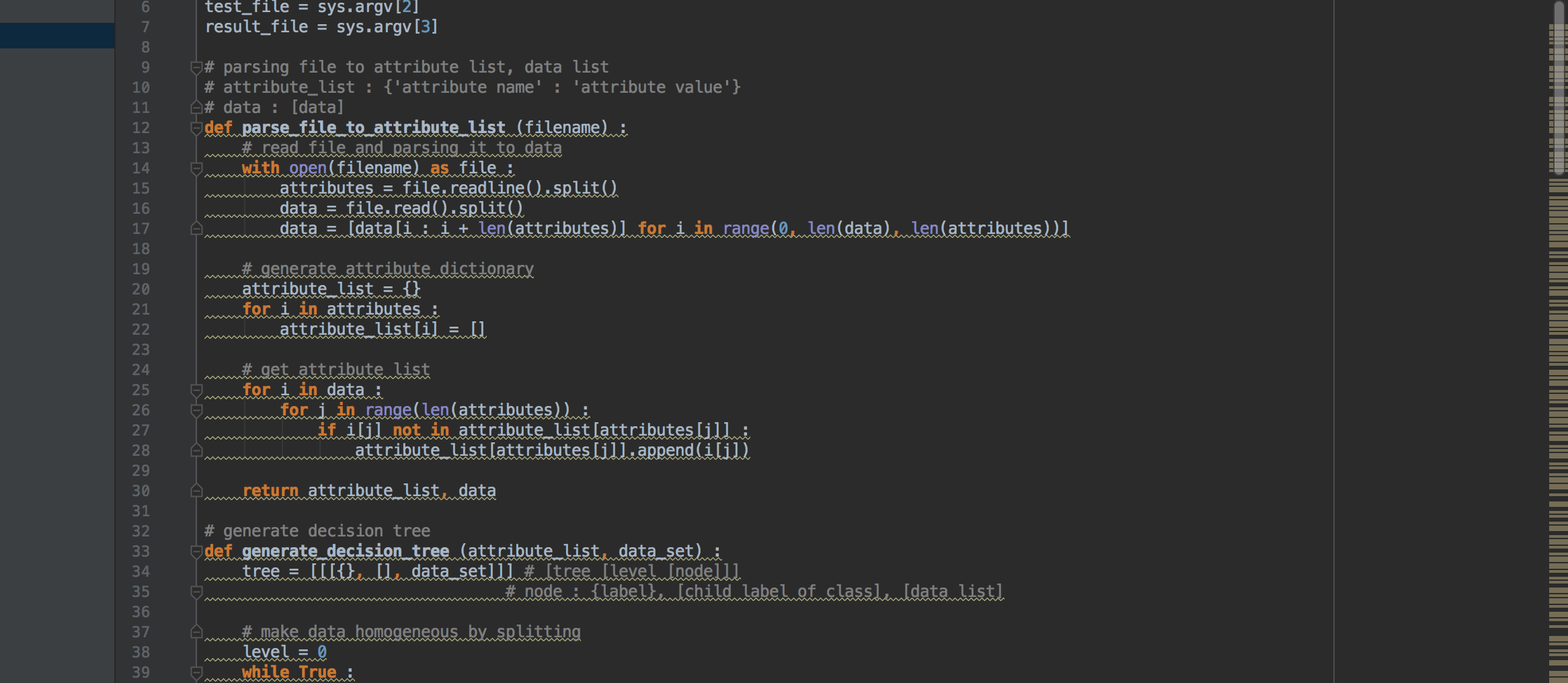
으로 계산할 수 있습니다.

이렇게 계산된 gain ration 중 최대값을 가지는 attribute가 선택되고, 이를 tree의 현재 node에서 자식 node로 나누는 기준으로 활용하게 됩니다. 더 이상 나누어 지지 않을 때까지 이 과정을 반복하게 되고, 이를 활용하여 어떤 class에 test data가 속할지 예측하게 됩니다.

**2. Detailed Description of Code**

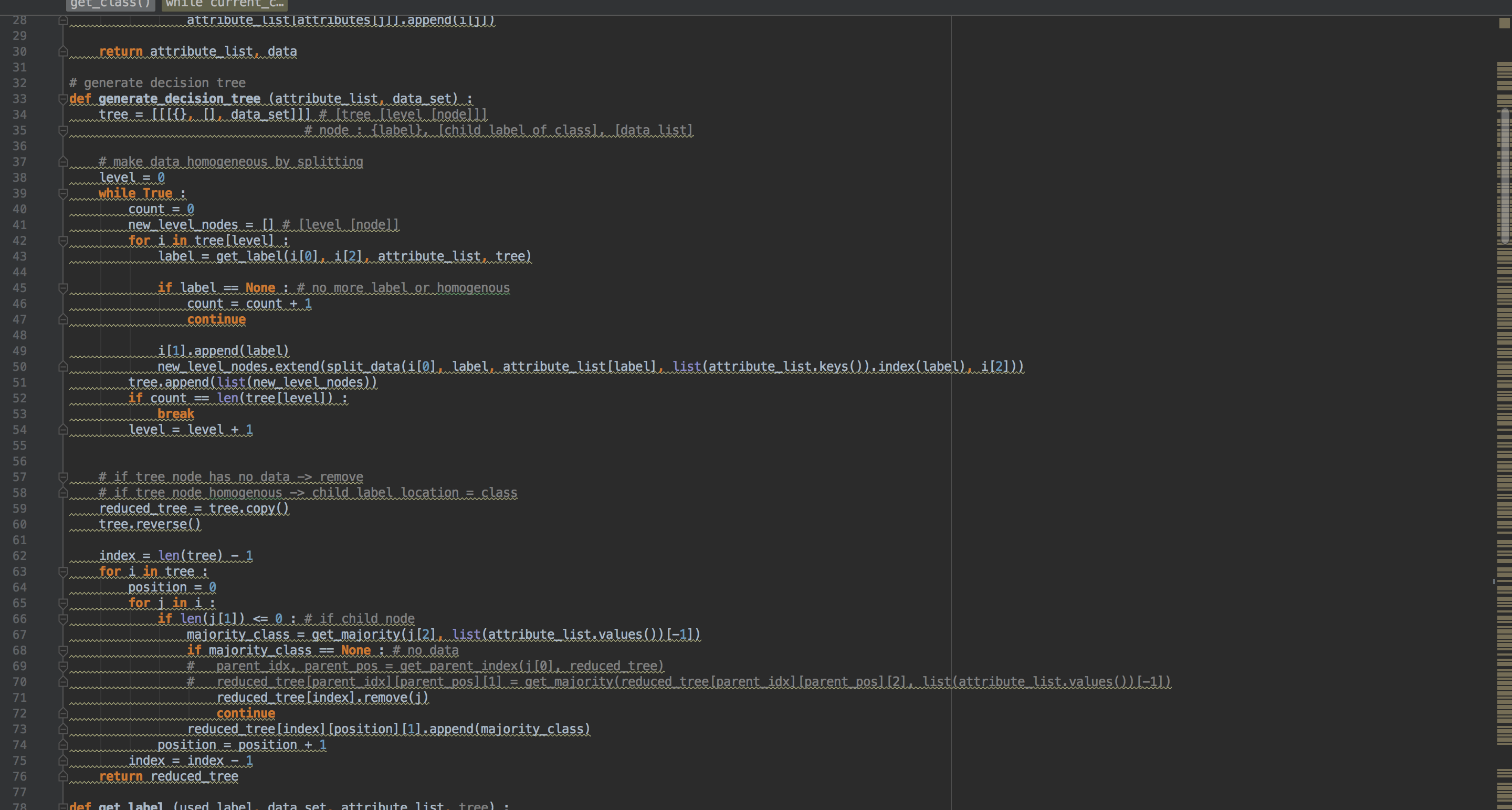


실행시 넘겨받은 argument로부터 training file name, test file name, output file name을 변수로 받아옵니다.



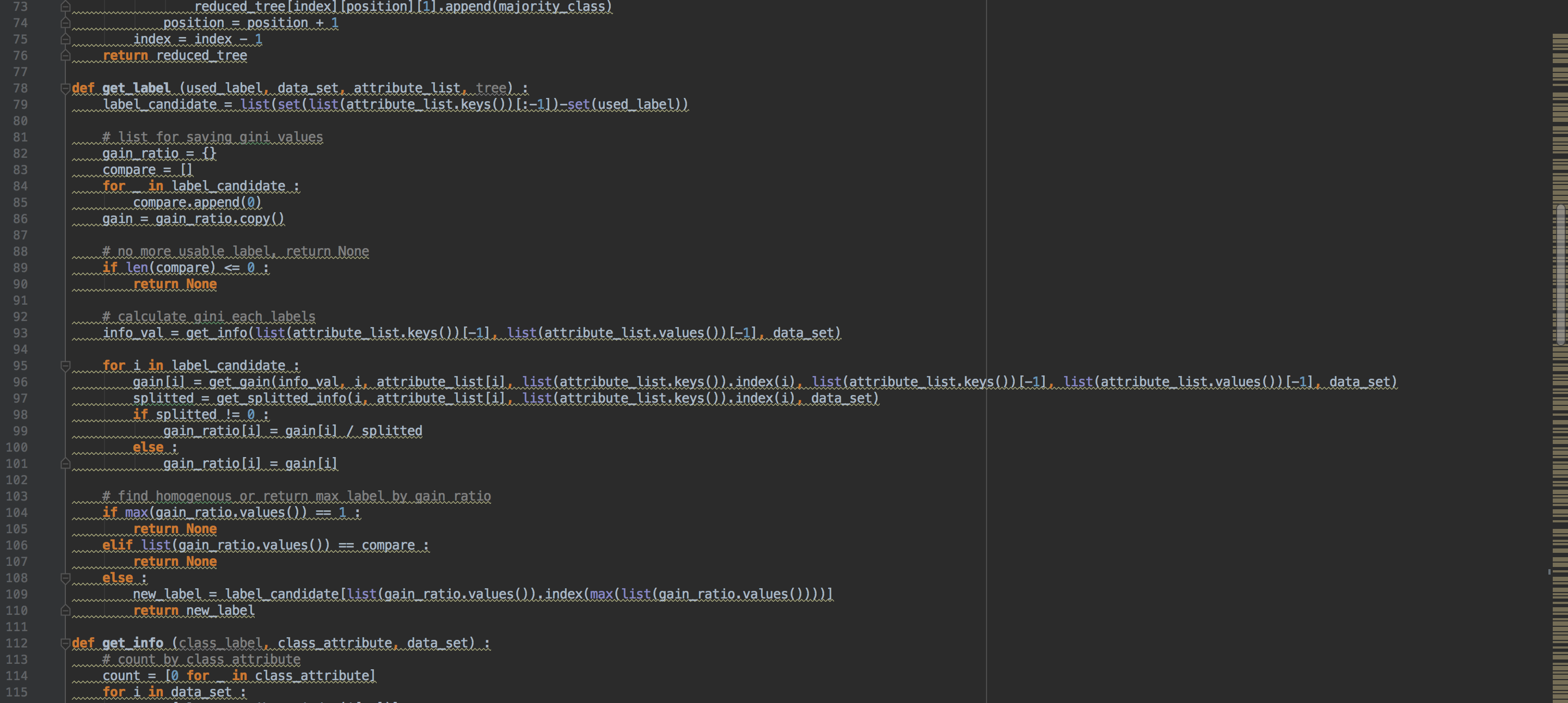
파일의 이름을 넘겨받아 해당 파일을 열고, 어떠한 attribute들이 있는지, 또한 어떠한 data들이 있는지 각각 리스트의 형태로 받아옵니다.

그리고 attribute list라는 dictionary를 만들어 attribute의 이름별로 data에 어떠한 속성들이 있는지 받아옵니다. 마지막으로 이 attribute list라는 dictionary, data를 return해줍니다.

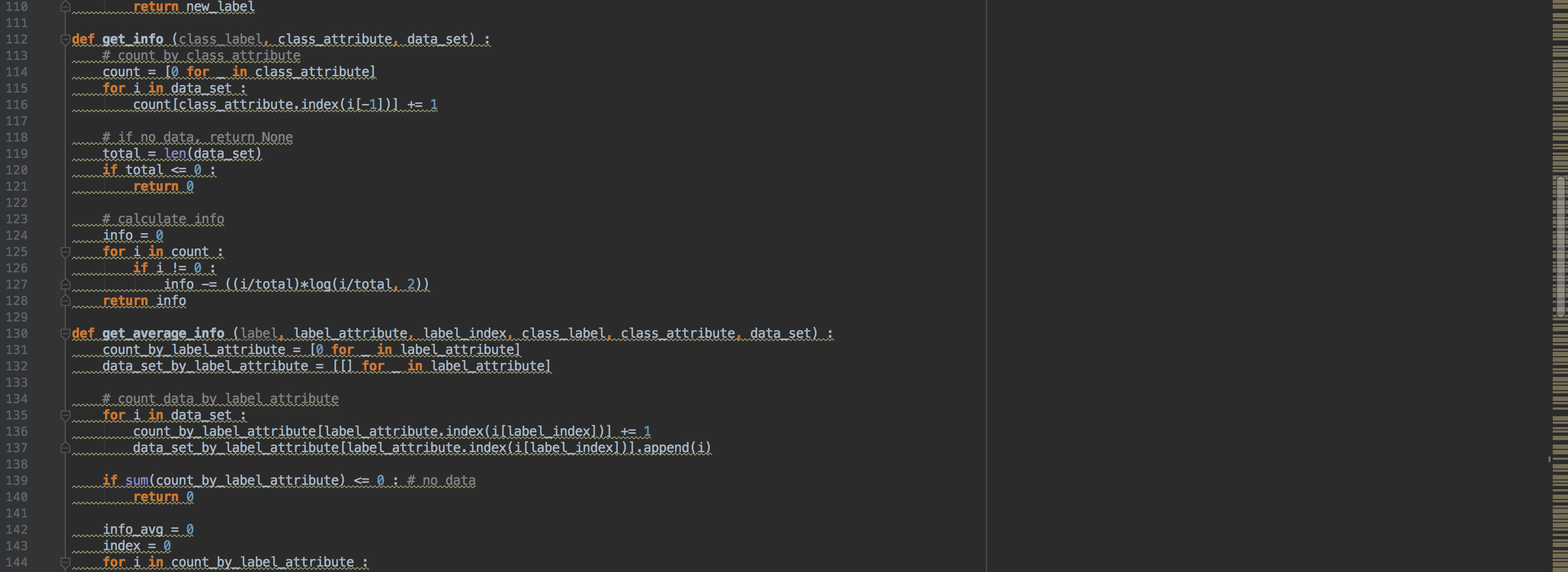


tree를 [tree [level [node [{labels}, [child label or class], [data]]]]]의 형태로 생성하고, level별로 for문을 돌며 새롭게 나눌 label을 받습니다. 이 label을 labels 위치에 추가하고, 원래 label(parent label)도 이 dictionary에 update해줍니다. 만약 label이 None이라면 바로 다른 node로 넘어가 이를 반복합니다. 이때, level의 모든 node가 다 None을 return한다면, 즉, 이제 더 이상 나눠지지 않는다면 이 작업을 끝냅니다.

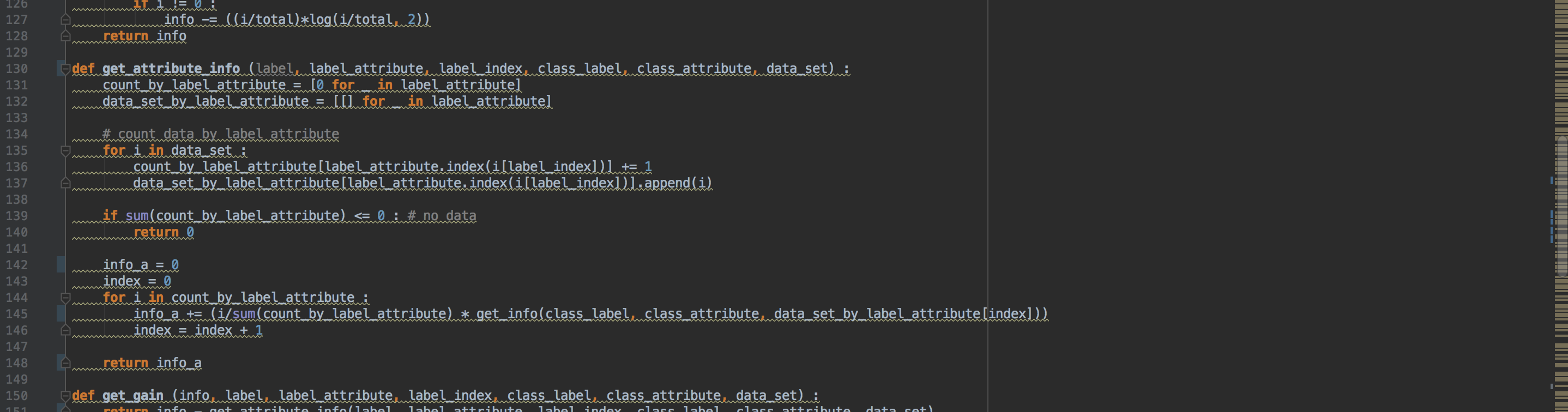
이제 반대로 tree를 돌며 data가 없는 node가 존재한다면 이를 없애줍니다. 그리고 이 작업을 마친 tree를 return해줍니다.



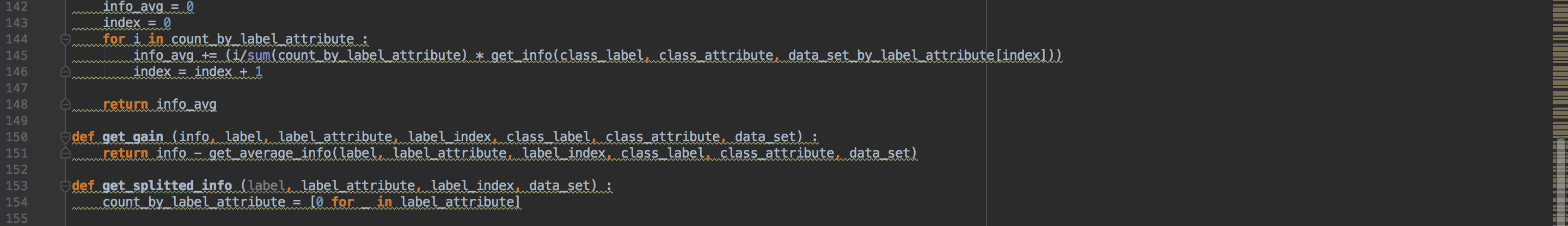
새롭게 나눠줄 label을 return합니다. 이 때 쓰여진 label들과 전체 label을 통하여 안쓰여진 label을 먼저 label\_candidate에 저장하고, 이 개수만큼 for문을 돌면서 0만 가지는, 모든 gain이 0이 나오는 경우를 확인하기 위한 리스트를 만들어줍니다. 이제 label candidate들의 gain, splitted info를 얻어 gain ratio를 얻고, 최대값이 1인 경우는 homogenous한 경우이므로 None을, 모두다 0이 나온 경우는 data가 없는 경우이므로 None을, 그 외의 경우는 max값을 가지는 label을 return해줍니다.



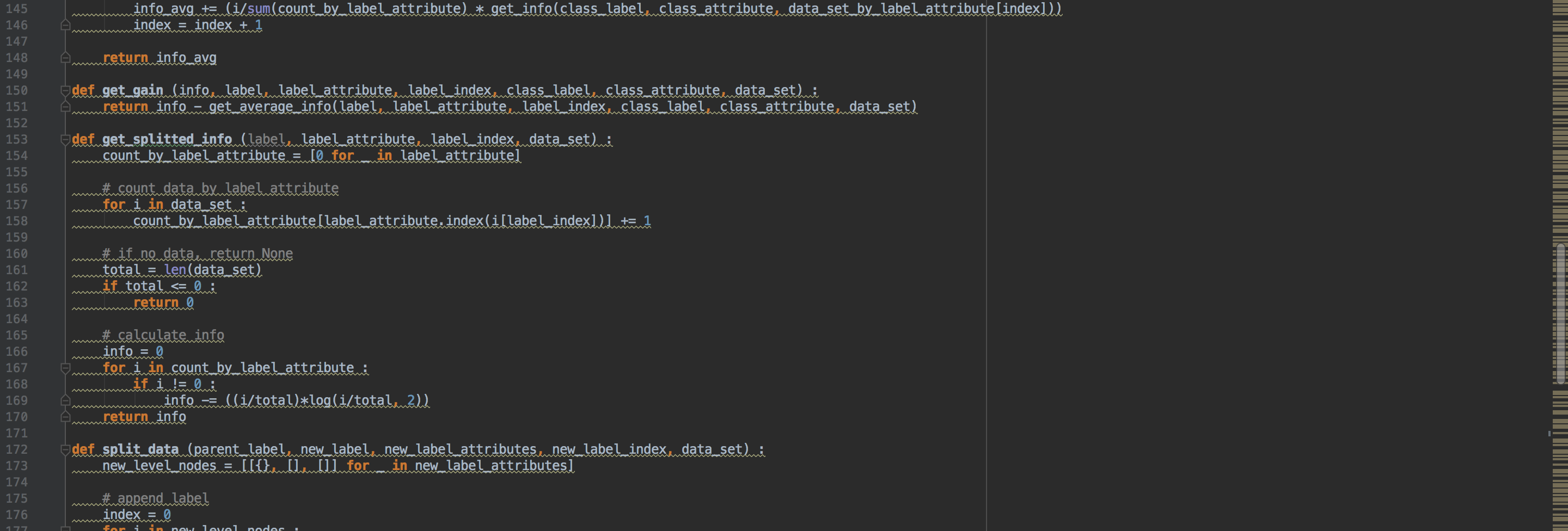
class attribute들과 data set을 받아 info를 위의 info 식에 맞춰 계산합니다. 그리고 얻어진 info값을 return해줍니다.



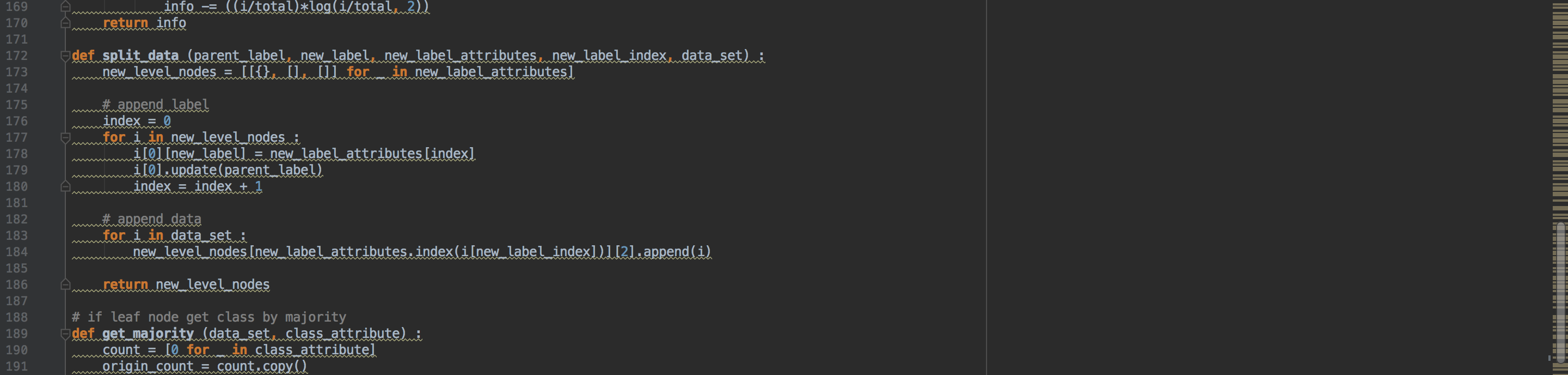
특정 label에 attribute에 해당하는 info gain을 계산하였을 때 어떤 info gain이 나오는지 위의 식에 맞추어 계산합니다. 그리고 계산된 값을 return해줍니다.



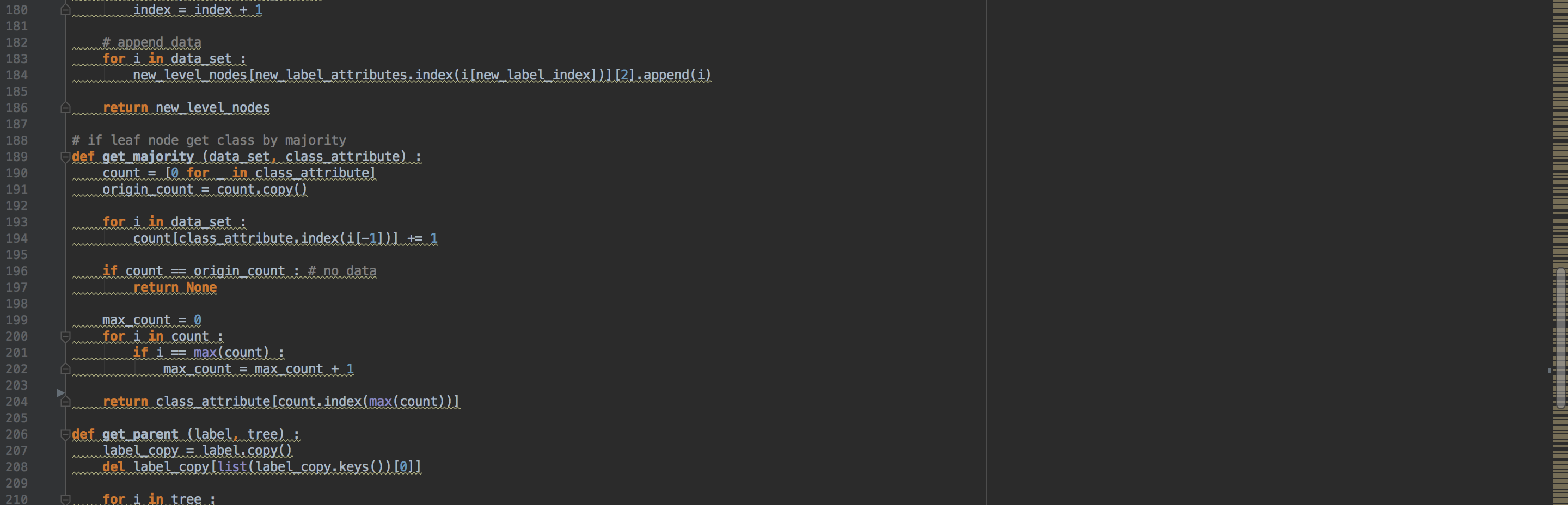
info에서 특정 label의 attribute에 대해 info를 계산한 값을 빼서 return 해줍니다.



얼마만큼의 data가 해당 label의 attribute들에 대해 잘 나누어지는지를 위의 식대로 계산하여 return해줍니다.



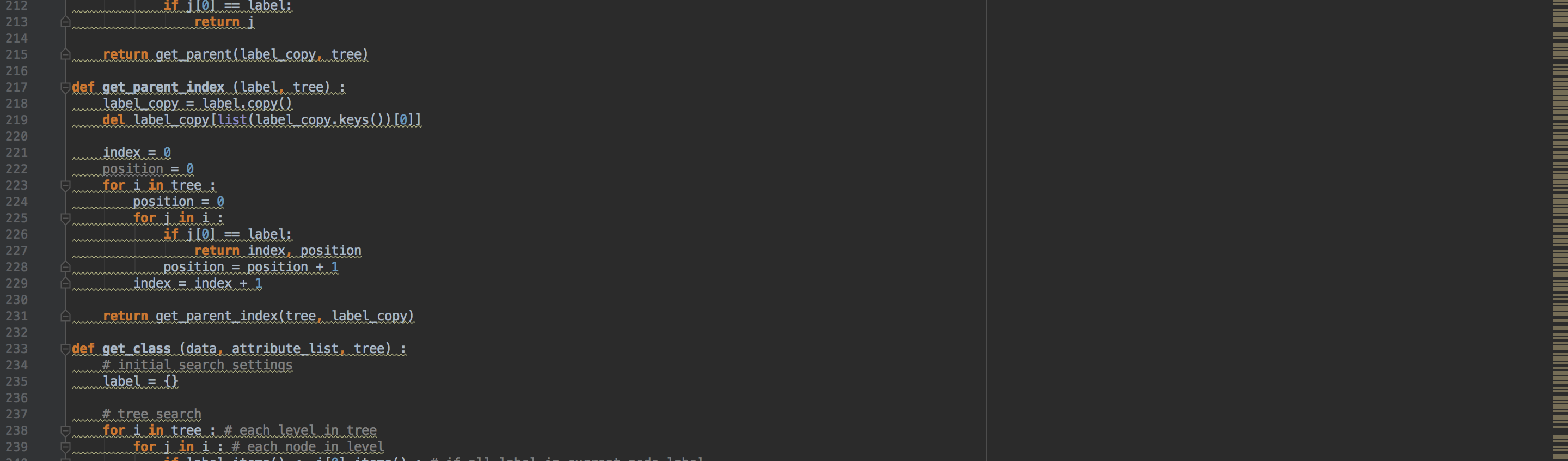
받은 label에 맞추어 데이터를 나누고, 이를 같은 tree level의 node를 저장하는 list에서, 같은 attribute를 가지는 node의 data 모임에 추가해줍니다. 그리고 이 새로운 level을 return해줍니다.



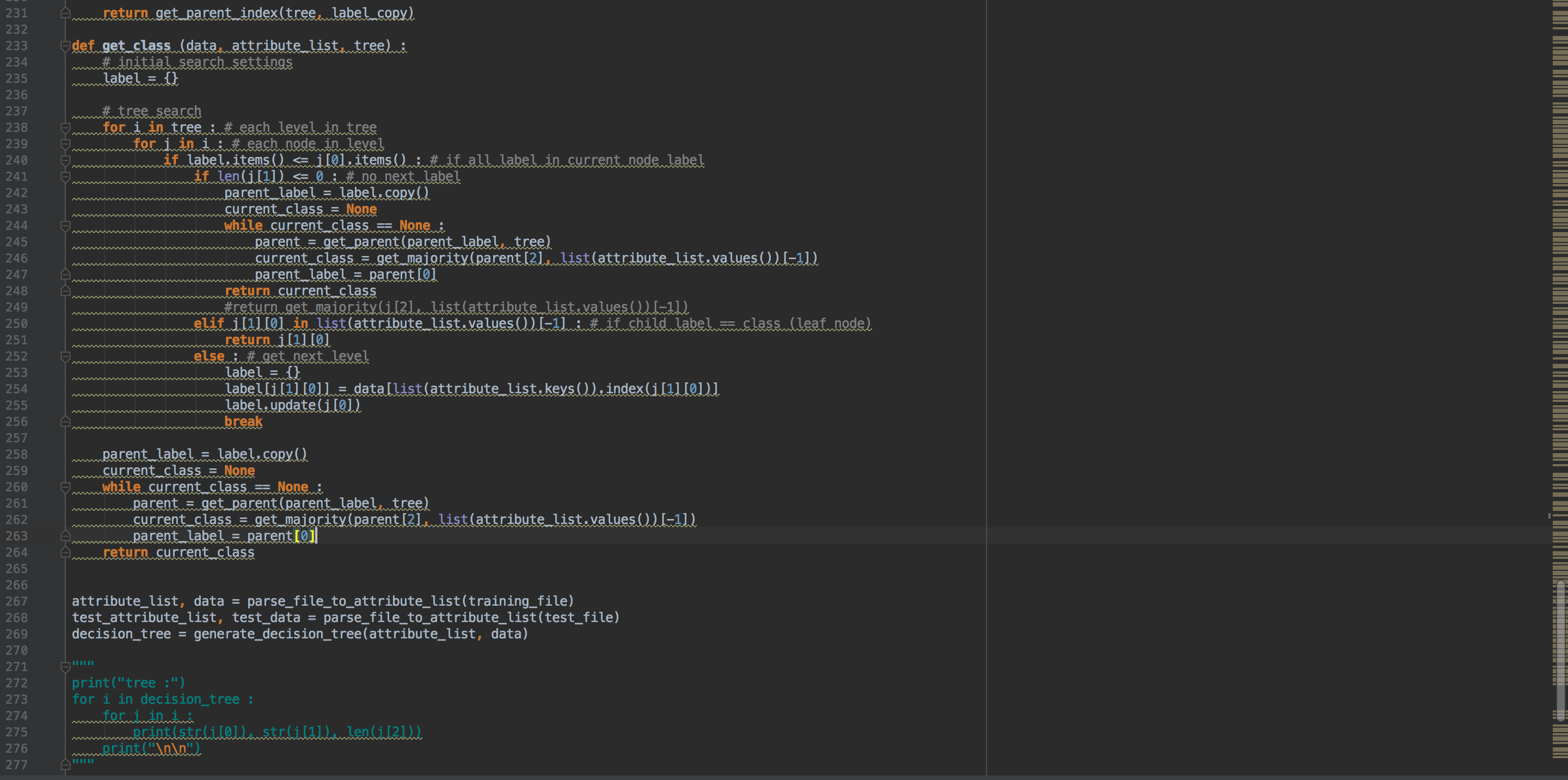
특정 node의 data set 중에서 가장 많이 나오는 class가 어떤 class인지를 찾습니다. 만약 확인하는 도중 data가 하나도 없다면 None을 return하고, 그렇지 않다면 count 횟수가 가장 높은 값을 return하되, 만약 동등한 값이 있다면 더 먼저 저장되어있는 class를 return합니다.



현재 node의 label을 통하여 parent node를 찾아 return합니다. 만약 현재 노드의 부모 노드가 찾아지지 않는다면 라벨에서 가장 하위 level에 속하는 attribute를 지우고, 다시 parent node를 찾습니다.

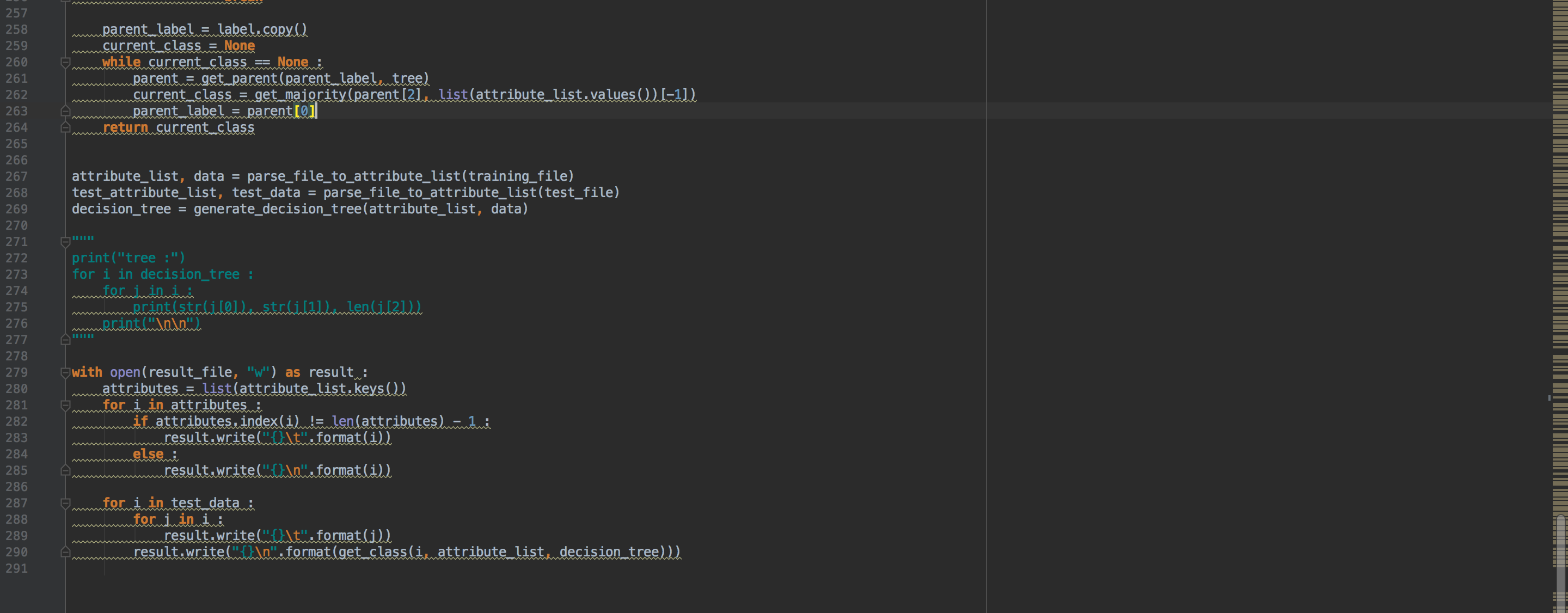


위의 get\_parent 함수와 같은 방식으로 작동하되, 이는 parent node가 아닌 tree에서의 parent node의 index를 return합니다.



test file이 어떤 class에 속하는지 확인합니다. 만약 자식을 탐색하면서 다음 level에 해당하는 값이 없는 경우는 parent node를 탐색하여 parent node에 어떤 class에 속하는 경우가 많았는지 확인하고, 해당 node가 class 정보를 들고 있다면 이를 return합니다. 그렇지 않다면 다음 level을 탐색합니다.

만약 이 과정을 통해 발견되지 않는다면 parent node를 탐색하며 get majority 함수를 통하여 class를 return합니다.



처음에 training file을 parsing하여 attribute list, data를 찾고, test file도 parsing하여 test file의 data를 저장합니다. 후에 tree를 generate하고, output file을 열어 attribute를 쓴 다음 test file의 data들을 output file에 다시 쓰고, 이의 class를 예측하여 함께 적어줍니다.

**3. Instruction for Compiling Source Code**

이 코드는 python3을 기반으로 작성되었습니다. 따라서 python3가 설치되어 있어야 합니다.

“**python3 dt.py [training\_file\_name] [test\_file\_name] [output\_file]**”의 형태로 complie, 실행하면 됩니다.

**4. Other Sepcification of Implementation and Testing**

training set을 통해 만든 tree의 각 child node가 어떤 class를 가지는지 알아내기 위해 어떤 class가 가장 많이 나타나는 가를 사용하게 됩니다. 만약 homogenous한 경우라면 당연히 하나의 class만 존재하기 때문에 이가 다수가 되고, 그렇지 않은 경우라면 어떠한 class가 많이 속했는지 알아내서, 이를 이 node의 대표 class로 분류하게 됩니다.

Tree의 자료구조가 다소 복잡합니다. 우선 전체 tree를 list의 형태로 표현하는데 각 level별로 서로 다른 list에 속합니다. 여기까지 [[level0], [level1], [level2], …] 의 형태입니다. 그리고 level안의 각각의 node도 list로 표현됩니다. [[node1], [node2], [node3], …]의 형태가 level n에 해당되는 것입니다. 그리고 하나의 node는 [{labels}, [child label or class], [data]]의 형태를 가지고 있습니다.

또한, parent node를 구분하기 위하여 해당 node의 dictionary에 현재 자신의 분류 attribute는 물론 부모들이 어떠한 attribute로 분류되어 왔느냐까지 저장하게 됩니다. 예를 들어, 처음에 ‘a’라는 attribute로 분류되었고, 그 밑에서 ‘b’라는 attribute로 분류되었다면, 이 node의 label dictionary는 {‘b’ : ‘b\_value’, ‘a’ : ‘a\_value’}가 됩니다.