Briefly describe the logic behind your algorithm along with its pseudocode

I used a dynamic programming approach to solving the problem. Dynamic programming will allow the large problem of the longest palindrome within the string to be solved based upon solving a series of smaller sub problems, which will be to divide the string up into smaller chunks and finding palindromes within them using the information gathered from previous iterations. The premise here is that if string[i+1...j-1] is a palindrome, then we know that string[i...j] must be a palindrome as well if s[i] is equal to s[j].

The algorithm works by initializing a matrix with n rows/columns (n = length of string). Within this matrix, the longest palindrome seen thus far is stored within the matrix. The index of the string would represent a starting position and ending position within the string. Using the calculations from above, we can easily fill out the matrix and note the longest palindrome encountered from a starting position in the string to an end position in the string. When you index the matrix, the value at position matrix[j][k] represents the longest palindrome subsequence between j and k seen thus far.

Building the matrix is trivial. If the characters of the string in the jth and kth position are not equal, the maximum palindrome length from a previous calculation is used (the max of either matrix[j + 1][k - 1], matrix[j + 1][k], matrix[j][k - 1]). If they are equal, we've found a new maximum palindrome and will set the current position of the matrix (matrix[j][k] to the new maximum palindrome. This string will be the largest previous palindrome encountered (matrix[j][k] = matrix[j + 1][k - 1]), that is encompassed by the 2 new characters at s[j] and s[k] to create the larger palindrome.

Once the end of the string is hit, the first row of the matrix (start of the string) and the last column (end of the string) will contain the largest palindrome.

Pseudocode:

```
  for i to length of string:
    Initialize matrix[i][i] to s[i] (starting and ending positions are the same character)


  for i to length of string:
    for j to length of string:
      k = j + i - 1
      if string[j] is equal to string[k]:
        matrix[j][k] = string[j] + longest palindrome seen (matrix[j + 1][k - 1]) + string[k]
      else:
        matrix[j][k] = max(matrix[j + 1][k - 1], matrix[j + 1][k], matrix[j][k - 1])
  return matrix[0][n - 1]
```

Implement your algorithm in Python

See .py file

Explain how the “character” example will be handled by your algorithm.

Note: Only the important strings are shown as the string, if the palindrome is 1 character I put a 1 for simplicity.

The matrix is initialized (n rows/columns) as follows:

C H A R A C T E R

C[**C**, 0, 0, 0, 0, 0, 0, 0, 0],

H[0, **H**, 0, 0, 0, 0, 0, 0, 0],

A[0, 0, **A**, 0, 0, 0, 0, 0, 0],

R[0, 0, 0, **R**, 0, 0, 0, 0, 0],

A[0, 0, 0, 0, **A**, 0, 0, 0, 0],

C[0, 0, 0, 0, 0, **C**, 0, 0, 0],

T[0, 0, 0, 0, 0, 0, **T**, 0, 0],

E[0, 0, 0, 0, 0, 0, 0, **E**, 0],

R[0, 0, 0, 0, 0, 0, 0, 0, **R**]

The algorithm will run, with j = 0, k = 1, comparing the letters C to H (s[j] to s[k]). Since they are not equal it will need to use the max of either matrix[j + 1][k - 1], matrix[j + 1][k], matrix[j][k - 1] (which will be 1 character in this case, based on the matrix above). The same process happens for the letters until the letter A is compared to A (s[2] and s[4]) since so far we would not have had a match in characters and would only have a max palindrome of 1. The matrix *before* A to A is compared would look like:

C H A R A C T E R

C[1, **1**, **1**, 0, 0, 0, 0, 0, 0],

H[0, 1, **1**, **1**, 0, 0, 0, 0, 0],

A[0, 0, 1, **1**, 0, 0, 0, 0, 0],

R[0, 0, 0, 1, **1**, 0, 0, 0, 0],

A[0, 0, 0, 0, 1, **1**, 0, 0, 0],

C[0, 0, 0, 0, 0, 1, **1**, 0, 0],

T[0, 0, 0, 0, 0, 0, 1, **1**, 0],

E[0, 0, 0, 0, 0, 0, 0, 1, **1**],

R[0, 0, 0, 0, 0, 0, 0, 0, 1]

When s[2] and s[4] are compared, this will be the first case of the same character being compared (a palindrome). The character at position matrix[j + 1][k - 1] (R) is in between the 2 A's in 'character', so the largest palindrome will be the value in the matrix at matrix[j + 1][k - 1] (1) plus 2. The largest palindrome seen at this position will be 3, so 3 is placed in the matrix at position matrix[2][4]. The matrix *after* the comparison will look like:

C H A R A C T E R

C[1, 1, 1, 0, 0, 0, 0, 0, 0],

H[0, 1, 1, 1, 0, 0, 0, 0, 0],

A[0, 0, 1, 1, **ARA**, 0, 0, 0, 0],

R[0, 0, 0, 1, 1, 0, 0, 0, 0],

A[0, 0, 0, 0, 1, 1, 0, 0, 0],

C[0, 0, 0, 0, 0, 1, 1, 0, 0],

T[0, 0, 0, 0, 0, 0, 1, 1, 0],

E[0, 0, 0, 0, 0, 0, 0, 1, 1],

R[0, 0, 0, 0, 0, 0, 0, 0, 1]

Now, nothing interesting happens again until the 2 C's are compared, which is at s[0] and s[5] (j = 0, k = 5). Before these 2 characters are compared, the matrix looks like:

C H A R A C T E R

C[1, 1, 1, **1**, **ARA**, 0, 0, 0, 0],

H[0, 1, 1, 1, **ARA**, **ARA**, 0, 0, 0],

A[0, 0, 1, 1, ARA, **ARA**, **ARA**, 0, 0],

R[0, 0, 0, 1, 1, **1**, **1**, **1**, 0],

A[0, 0, 0, 0, 1, 1, **1**, **1**, **1**],

C[0, 0, 0, 0, 0, 1, 1, **1**, **1**],

T[0, 0, 0, 0, 0, 0, 1, 1, **1**],

E[0, 0, 0, 0, 0, 0, 0, 1, 1],

R[0, 0, 0, 0, 0, 0, 0, 0, 1]

As can be seen, the max of matrix[j + 1][k - 1], matrix[j + 1][k], matrix[j][k - 1] would have returned ARA in some cases. The string ARA is placed in some positions of the matrix where the largest palindrome seen thus far was ARA. Now that j = 0 and k = 5, the largest palindrome seen will be CARAC. The value in position matrix[j + 1][k - 1] (matrix[1][4]) is ARA. The 2 C's encompass ARA, resulting in a new maximum palindrome of length 5. The value at matrix[0][5] is updated to CARAC. The matrix looks like:

C H A R A C T E R

C[1, 1, 1, 1, ARA, **CARAC**, 0, 0, 0],

H[0, 1, 1, 1, ARA, ARA, 0, 0, 0],

A[0, 0, 1, 1, ARA, ARA, ARA, 0, 0],

R[0, 0, 0, 1, 1, 1, 1, 1, 0],

A[0, 0, 0, 0, 1, 1, 1, 1, 1],

C[0, 0, 0, 0, 0, 1, 1, 1, 1],

T[0, 0, 0, 0, 0, 0, 1, 1, 1],

E[0, 0, 0, 0, 0, 0, 0, 1, 1],

R[0, 0, 0, 0, 0, 0, 0, 0, 1]

There are no more duplicate characters that will create a larger palindrome, the loop runs filling out the matrix with the max value calculated from the positions matrix[j + 1][k - 1], matrix[j + 1][k], matrix[j][k - 1]. The final matrix will look like:

C H A R A C T E R

C[1, 1, 1, 1, ARA, CARAC, **CARAC**, **CARAC**, **CARAC**],

H[0, 1, 1, 1, ARA, ARA, **ARA**, **ARA**, **ARA**],

A[0, 0, 1, 1, ARA, ARA, ARA, **ARA**, **ARA**],

R[0, 0, 0, 1, 1, 1, 1, 1, **ARA**],

A[0, 0, 0, 0, 1, 1, 1, 1, 1],

C[0, 0, 0, 0, 0, 1, 1, 1, 1],

T[0, 0, 0, 0, 0, 0, 1, 1, 1],

E[0, 0, 0, 0, 0, 0, 0, 1, 1],

R[0, 0, 0, 0, 0, 0, 0, 0, 1]

As can be seen, in the position matrix[0][8] the length of the longest palindrome is 5. The string CARAC is returned, the value stored in matrix[0][8].