Ozobot Follows a Path Based on an LLM Bot and Camera Image

Ozobot Editor Lesson—Student Guide

by Richard Born, Associate Professor Emeritus, Northern Illinois University

Introduction

In this lesson, you will learn how to get Ozobot to follow a path like that shown in Figure 1. This will be accomplished by:

1. capturing a snapshot of the figure using an LLM Bot and connected camera
2. providing a prompt to the LLM that elicits a response that has extracted appropriate data from a camera image
3. designing an Ozobot Editor program that will result in Ari or Evo following the path shown in the figure.

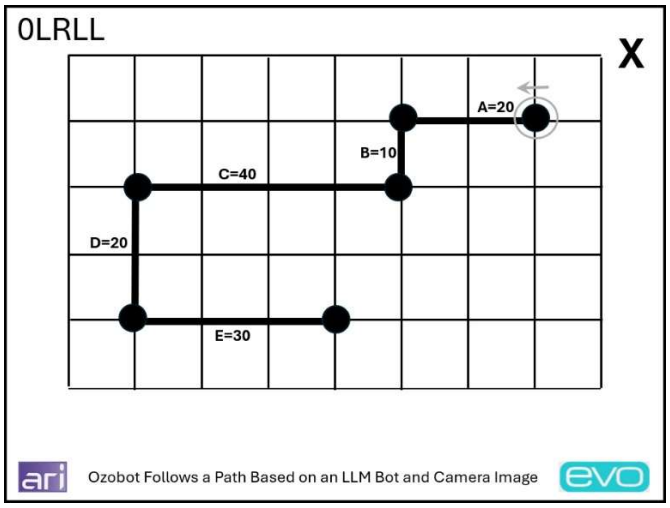


Figure 1

The path is shown on a grid as a heavy black line. Ozobot starts at the point with the gray circle, facing the direction shown by the arrow. Each segment of the path from start to finish is labeled with a letter and the length of the segment.

The code in the upper left corner of the figure tells the direction to turn at each of the points from the start to the *next to the last* point on the path. The leading 0 in the code means do not turn, just go straight at the starting point. There is no turn code for the last point in the path, as there is no reason to turn since it is the end of the path.

This lesson is accompanied by three different paths identified by X, Y, and Z in the upper right corner.

The Prompt and the LLM Bot Response

Figure 2 shows the LLM prompt used in this program. The ***Google (mini)*** LLM Model should be used as it was the only model that worked nearly flawlessly during program testing.

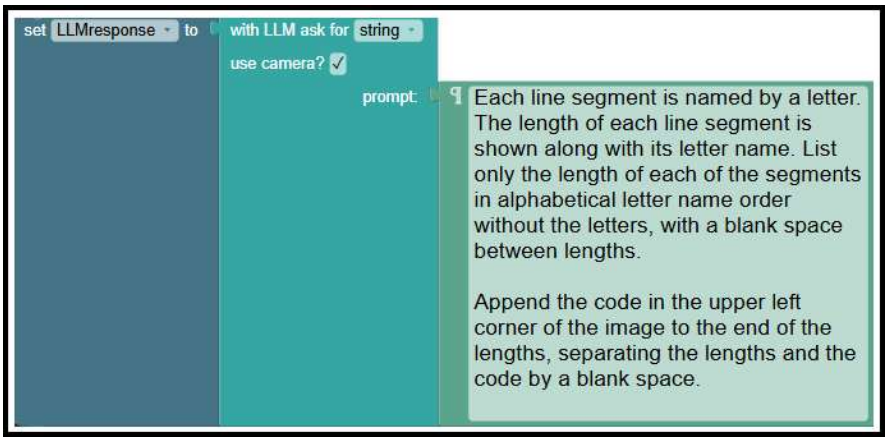


Figure 2

The prompt must be carefully worded. You will note that the wording of the prompt implies that we are expecting a response to the path shown in Figure 1 as follows:

20 10 40 20 30 0LRLL

This response contains all of the information that we need to get Ozobot to follow the path. It is contained in the *string* variable *LLMresponse*. We have the length of each of the segments in the correct order, and we have the direction for Ozobot to turn at each point from the first point to the next to last point in the path.

You need to keep in mind that the program should work not only for the path X shown in Figure 1 but also for two other paths, Y and Z, provided by your teacher. The paths may contain a different number of segments, so your program should work regardless of the number of segments.

Your program will need to extract from the *LLM response* the desired distance and direction for each move and rotation required for Ozobot to follow the path from start to finish. The process of extracting this information from the string is called **parsing**.

Some Common Examples of Parsing a String

More formally, parsing can be defined as *"a technique used to analyze and interpret a string of symbols by breaking the string down into smaller more manageable components."*

Before we investigate how to parse the *string* variable *LLMresponse,* let's take a look at some examples where parsing is required by three well-known programs in Microsoft 365®—PowerPoint, Word, and Excel. Doing so will give you an intuitive feel for the usefulness of parsing.

It is common for many software developers to include the capability of making a custom selection on which pages of a document to print. Figure 3 shows an example when printing slides from a PowerPoint presentation. The user has specified that slides 1, 3, and 5-6 are to be printed. The PowerPoint software needs to parse this string to determine that the pages to be printed are 1, 3, 5, and 6. This is accomplished by searching the string for commas and dashes and applying appropriate logic to the search results.

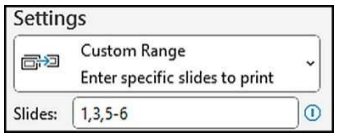


Figure 3

Suppose that your English teacher has assigned an essay that must be at least 500 words in length. You don't need to count them manually. Word and other word processing programs can count them for you. The example in Figure 4 shows that the word count for the included text is 23 words. Word accomplishes this by searching for blank spaces between words, counting them, and adding 1 to the count. It also must reason that multiple spaces between two words are actually one space. This is all accomplished through a careful parsing of the text.

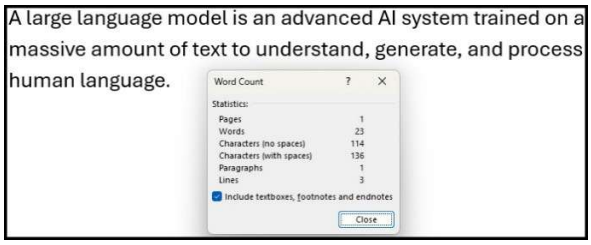


Figure 4

When a formula is entered into a cell, Excel automatically parses it to find any errors. By parsing the expression entered by the user, the example shows that the user may have meant to multiply two parenthetical expressions but forgot the multiplication symbol and the closing parenthesis in the second expression. The user is offered a possible correction which can be accepted or rejected.

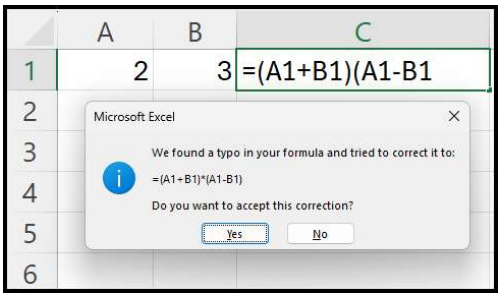


Figure 5

Parsing the LLM Response String

Before presenting the Ozobot Editor blocks for this program, let's take a detailed look at the string *LLMresponse* presented earlier. This is shown in Figure 6.

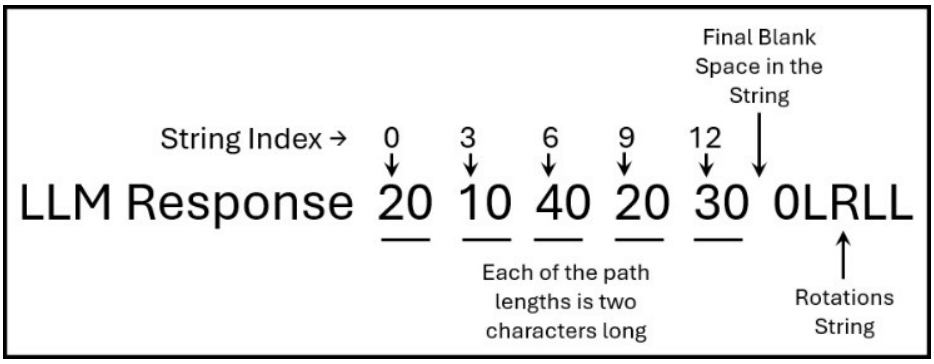


Figure 6

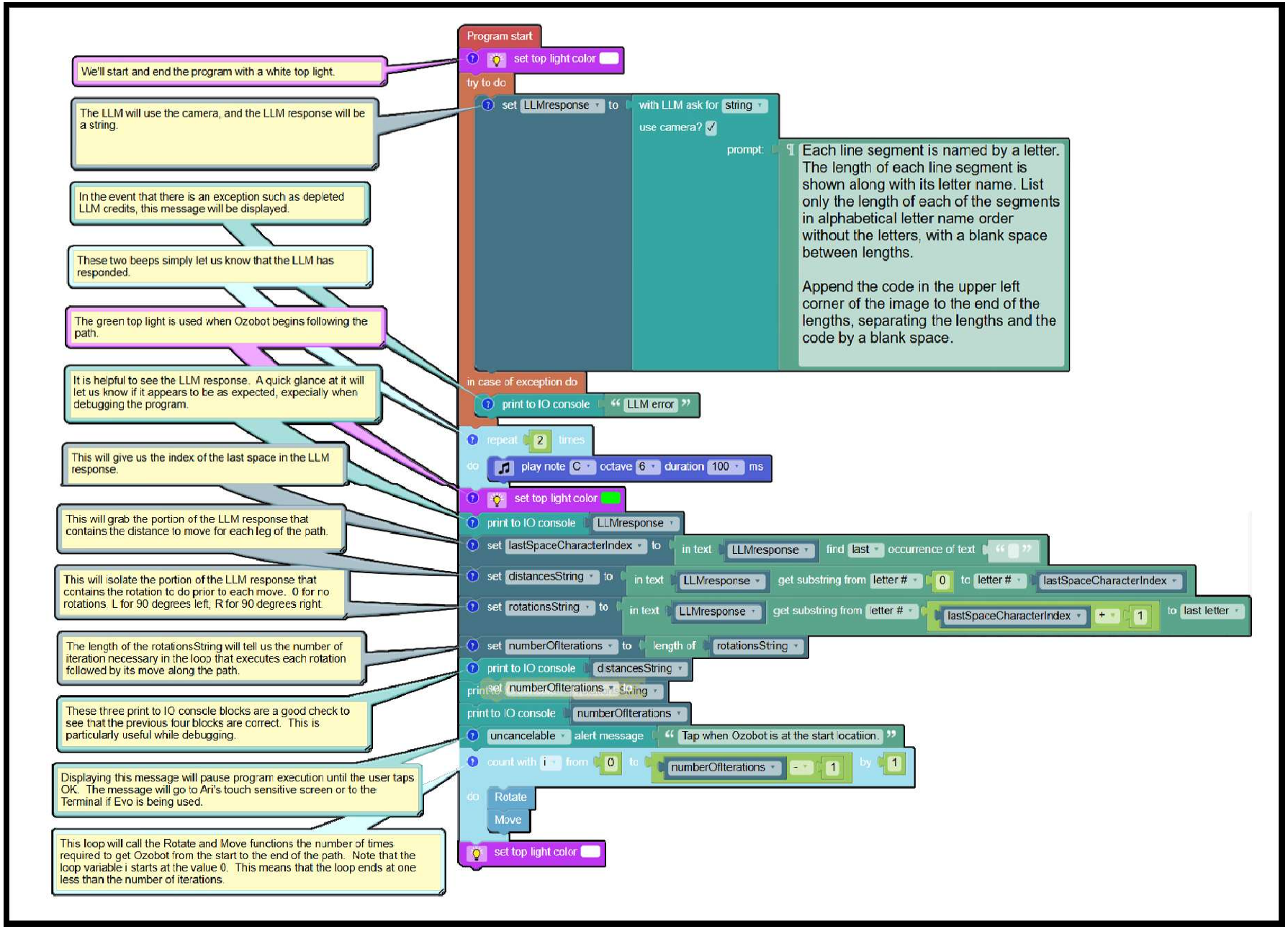
Keep in mind that the characters within a string have indices *starting with 0*, not 1. So, the *index* of the first character in the LLM response is 0. Each of the path lengths is two characters long and there is a blank space between each length. Therefore, each of the path lengths starts at an index that is divisible by 3, *i.e.,* 0, 3, 6, 9, etc. Note that there is a blank space between the final path length and the rotations portion of the response, and that this blank space is the last blank space in the *LLMresponse* string. Finally, the length of the rotations portion of the string (5 in this case) is equal to the number of segments in the path.

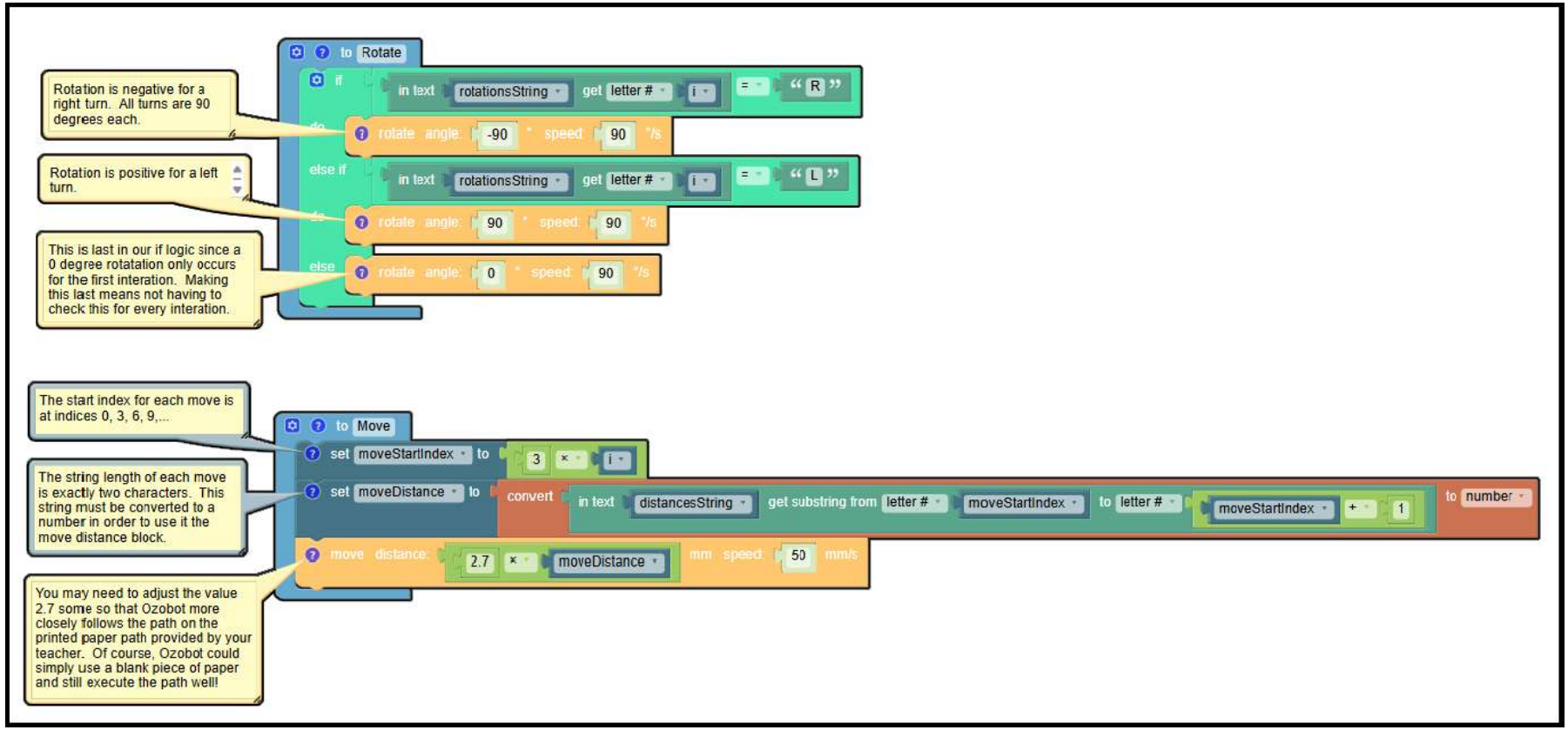
The Ozobot Editor Program

With a solid understanding of the details on how the *LLMresponse* variable was formed, you are now ready to study the Ozobot Editor program. Just to summarize our goal, the program was designed so that Ozobot Ari or Evo could follow a path based upon a camera image of the path that is interpreted by an appropriate prompt provided to an LLM.

The program consists of a main program and two function subroutines. One of the functions causes Ozobot to move along a segment on the path. The other function causes Ozobot to rotate upon reaching a turn point in the path.

The next two pages of this lesson contain the main program and these two functions. Many of the blocks contain detailed comments that explain the purpose of each block. You should study these carefully in order thoroughly understand this program. Of particular interest are the blocks in the text category, as these are used to parse the *LLMresponse* variable.


We'll start and end the program with a white top light.
The LLM will use the camera, and the LLM response will be a string.
In the event that there is an exception such as depleted LLM credits, this message will be displayed.
These two beeps simply let us know that the LLM has responded.
The green top light is used when Ozobot begins following the path.
It is helpful to see the LLM response. A quick glance at it will let us know if it appears to be as expected, especially when debugging the program.
This will give us the index of the last space in the LLM response.
This will grab the portion of the LLM response that contains the distance to move for each leg of the path.
This will isolate the portion of the LLM response that contains the rotation to do prior to each move. 0 for no rotations. L for 90 degrees left, R for 90 degrees right.
The length of the rotationsString will tell us the number of iteration necessary in the loop that executes each rotation followed by its move along the path.
These three print to IO console blocks are a good check to see that the previous four blocks are correct. This is particularly useful while debugging.
Displaying this message will pause program execution until the user taps OK. The message will go to Ari's touch sensitive screen or to the Terminal if Evo is being used.
This loop will call the Rotate and Move functions the number of times required to get Ozobot from the start to the end of the path. Note that the loop variable i starts at the value 0. This means that the loop ends at one less than the number of iterations.
Program start
set top light color
try to do
set LLMresponse to with LLM ask for string
use camera?
prompt:
Each line segment is named by a letter. The length of each line segment is shown along with its letter name. List only the length of each of the segments in alphabetical letter name order without the letters, with a blank space between lengths.
Append the code in the upper left corner of the image to the end of the lengths, separating the lengths and the code by a blank space.
in case of exception do
print to IO console "LLM error"
repeat 2 times
do
play note C octave 6 duration 100 ms
set top light color
print to IO console LLMresponse
set lastSpaceCharacterIndex to in text LLMresponse find last occurrence of text " "
set distancesString to in text LLMresponse get substring from letter # 0 to letter # lastSpaceCharacterIndex
set rotationsString to in text LLMresponse get substring from letter # lastSpaceCharacterIndex + 1 to last letter
set numberOfIterations to length of rotationsString
print to IO console distancesString
print to IO console numberOfIterations
uncancelable alert message "Tap when Ozobot is at the start location."
count with i from 0 to numberOfIterations - 1 by 1
do
Rotate
Move
set top light color


Rotation is negative for a right turn. All turns are 90 degrees each.
Rotation is positive for a left turn.
This is last in our if logic since a 0 degree rotatation only occurs for the first interation. Making this last means not having to check this for every interation.
The start index for each move is at indices 0, 3, 6, 9,...
The string length of each move is exactly two characters. This string must be converted to a number in order to use it the move distance block.
You may need to adjust the value 2.7 some so that Ozobot more closely follows the path on the printed paper path provided by your teacher. Of course, Ozobot could simply use a blank piece of paper and still execute the path well!
to Rotate
if
in text rotationsString get letter # i = " R "
do
rotate angle: -90 ° speed: 90 °/s
else if
in text rotationsString get letter # i = " L "
rotate angle: 90 ° speed: 90 °/s
else
rotate angle: 0 ° speed: 90 °/s
to Move
set moveStartIndex to 3 × i
set moveDistance to convert in text distancesString get substring from letter # moveStartIndex to letter # moveStartIndex + 1 to number
move distance: 2.7 × moveDistance mm speed: 50 mm/s

Working with the Program

Your teacher will indicate how to proceed. You may be asked to enter the blocks as shown in the previous two pages. Alternatively, your teacher may consider providing you with the share code and let you proceed directly by investigating the program using three different paths labeled X, Y, and Z in the upper right corner.

In either case, you have learned much about parsing and the capabilities of LLMs. Enjoy your investigation!