3. WRITTEN RESPONSES

3 a.

3.a.i.

The program is meant to explore the user's creativity through the expression of poetry using their words and images as well as preferences of formatting in order to generate some unique poems.

3.a.ii.

The video shows the generation of a poem as specified by the user's inputs of words like pine, star, lies, etc. and the setting of with articles. It results in multiple different poems being generated with different word orders (3, as specified by the user). Then another generation occurs with different words (forest, expanse, glows, etc.) with no articles and 5 poems being generated.

3.a.iii.

The input of both typing words to insert into the poem and a typing of yes or no to set articles or no articles as well as number of desired poems generated all result in the output of a certain specified number of poems according to those words and specifications.

3 b.

3.b.i.

```
# Word inputs
nounT = input("Enter a noun related to trees: ")
nounW = input("Enter a noun related to the ocean or water: ")
nounS = input("Enter a noun related to the night sky: ")
nounList = [nounT, nounW, nounS]
```

3.b.ii.

```
# Procedure to actually use a word in the poem (random selection from a list)
def selectWord(wordList):
  import random
  currentWordNdx = random.randint(0, len(wordList) - 1)
  return wordList.pop(currentWordNdx)
```

3.b.iii.

The specific list I chose is nounList. As it is called and used in the procedure selectWord, however, it is referred to as the parameter wordList instead.

3.b.iv.

The data in the list contains the nouns that must be in certain specific locations within the poem (so that the poem makes logical sense).

3.b.v.

Using a list here enables efficient random selection of an element within the list and then returning that noun at a specific point when I invoke the selectWord method. In this way, it manages the complexity of random selection and the storage of multiple nouns at once. If I did not have this list, I would have to store the noun inputs in different variables and keep invoking those different variables instead, as well as finding another way to randomize selection, perhaps using the generation of a random integer, each corresponding to its own selection/if statement, but in that case I would have to write individual statements for each possible int generated and make the code a lot more cluttered.

3 c.

3.c.i.

```
# Function to create the poems
def createPoems(nList, vList, aList, aSetting):
  i = 0
  finalPoems = ""

  while i < poemNum:
    if aSetting == 1:
      finalPoems += articlePoem(nList, vList, aList) + "\n\n"
    if aSetting == 0:
      finalPoems += noArticlePoem(nList, vList, aList) + "\n\n"
    i += 1

  return finalPoems
```

3.c.ii.

```
# Output section
print("\n" + "=== Poems ===")
if articlePreference == "yes":
  print("\n" + createPoems(nounList, verbList, adverbList, 1))
elif articlePreference == "no":
  print("\n" + createPoems(nounList, verbList, adverbList, 0))
else:
  print("Error: input incorrect (likely incorrect article
preference).")
```

3.c.iii.

The procedure createPoems contributes to functionality by putting together the final selection of poems according to how many the user specified and whether the user wanted articles in the poems, as well as the procedures (not pictured) that generate the poems.

3.c.iv.

The procedure requires other procedures not pictured to actually work, including articlePoem and noArticlePoem. createPoems itself, however, sequentially first defines a variable to count the number of poems generated and defines an empty string, then has iteration that generates as many poems as the user specified by running through the selection statements which call articlePoem or noArticlePoem each time. The selection if statements between articlePoem or noArticlePoem respectively call either the procedure for a poem with articles or one without depending on the user's input (not pictured) and thus the argument for aSetting, and whichever type of poem specified earlier will be added to the string. Finally, the string is returned.

3 d.

3.d.i.

First call:

The two calls are based on differing settings for articlePreference that were input by the user to generate poems either with or without articles based on if the user input yes or no. The first call is print("\n" + createPoems(nounList, verbList, adverbList, 1)

Second call:
The second call is print("\n" + createPoems(nounList, verbList, adverbList, 0))

3 d.ii.

Condition(s) tested by first call:
The first call checks the aSetting (article setting) and the if statement on line 37 which checks if aSetting equals 1 will then execute line 38.

Condition(s) tested by second call:
The second call checks the aSetting (article setting) except since the argument equals 0 fails the check of the if statement on line 37 and instead passes the check on the if statement of line 39 to execute line 40.

3.d.iii.

Results of the first call:
The first call results in the generated poems being called from articlePoem, and thus the final output poems will have “The” and “And” in the content.

Results of the second call:
The second call results in the generated poems being called from noArticlePoem, and thus the final output poems will not have “The” and “And” in the content.