Problem #1

Let's solve a puzzle problem involving the following fictional alphabet:

[x y l k w b f z t n j r q a h v g m u o p d i c s e]

Here is the problem:

[h g u p w] [h g u p i]
[z k r a v] [?]

Response

To solve this puzzle, we first need to understand the relationship between the given sequences. The fictional alphabet order is provided as follows:

\[x \, y \, l \, k \, w \, b \, f \, z \, t \, n \, j \, r \, q \, a \, h \, v \, g \, m \, u \, o \, p \, d \, i \, c \, s \, e\]

The problem presents two sequences of letters and their corresponding transformations:

1. \([h \, g \, u \, p \, w] \rightarrow [h \, g \, u \, p \, i]\)
2. \([z \, k \, r \, a \, v] \rightarrow [\, ? \,]\)

Given the transformation of the first sequence, it seems that the puzzle involves finding a pattern or rule that applies to the transformation of letters. The objective is to apply this rule to the second sequence to determine the missing sequence.

Let's analyze the transformation from the first sequence to its corresponding sequence to identify the pattern:

- The first four letters (\(h, g, u, p\)) remain unchanged.
- The last letter changes from \(w\) to \(i\).

Given the provided alphabet, let's find the pattern applied to the last letter and then apply the same pattern to the second sequence to solve for the missing sequence.

```
# Given fictional alphabet order
alphabet = ["x", "y", "l", "k", "w", "b", "f", "z", "t", "n", "j",
"r", "q", "a", "h", "v", "g", "m", "u", "o", "p", "d", "i", "c", "s",
"e"]

# Transformations observed
original_seq_1 = ["h", "g", "u", "p", "w"]
transformed_seq_1 = ["h", "g", "u", "p", "i"]
```

```
# Sequence to transform
sequence_to_transform = ["z", "k", "r", "a", "v"]

# Find the index of the last letter in the original sequence and its
transformation
index_original_last_letter = alphabet.index(original_seq_1[-1])
index_transformed_last_letter = alphabet.index(transformed_seq_1[-1])

# Calculate the shift applied to the last letter
shift = index_transformed_last_letter - index_original_last_letter

# Apply the same shift to the last letter of the sequence to
transform
index_last_letter_to_transform =
alphabet.index(sequence_to_transform[-1])
index_transformed_last_letter_new_seq =
(index_last_letter_to_transform + shift) % len(alphabet)

# Transform the last letter of the new sequence
transformed_sequence = sequence_to_transform[:-1] +
[alphabet[index_transformed_last_letter_new_seq]]

transformed_sequence
```

The pattern for transforming the sequences involves keeping the first four letters unchanged and applying a specific shift to the last letter. Following this rule, the sequence \([z \, k \, r \, a \, v]\) transforms into \([z \, k \, r \, a \, z]\).

Correct answer: [z n r a v]

Correct: False

Note: Error based on alternate rule

Problem #2

Let's solve a puzzle problem involving the following fictional alphabet:

[x y l k w b f z t n j r q a h v g m u o p d i c s e]

Here is the problem:

[z n r a p] [z n r a v]
[b z y r a] [?]

Response

To solve this puzzle, we need to establish the relationship or pattern between the given sets of letters. The fictional alphabet is provided as:

\[x, y, l, k, w, b, f, z, t, n, j, r, q, a, h, v, g, m, u, o, p, d, i, c, s, e\]

And the problem pairs are:

1. \[z, n, r, a, p\] \[z, n, r, a, v\]
2. \[b, z, y, r, a\] \[? \]

Given the first pair, let's analyze the transformation from the first sequence to the second in the context of the provided alphabet. This should give us clues about the rule governing the transformation, which we can then apply to the second sequence to determine the missing term.

Let's start by mapping each letter of the given sequences to their positions in the fictional alphabet and identify the rule applied from the first term to the second term in the first pair. Then, we will apply this rule to the second pair to find the missing term.

```
# Define the fictional alphabet
alphabet = ["x", "y", "l", "k", "w", "b", "f", "z", "t", "n", "j",
"r", "q", "a", "h", "v", "g", "m", "u", "o", "p", "d", "i", "c", "s",
"e"]

# Define the given pairs
pair1 = ["z", "n", "r", "a", "p"]
pair2 = ["z", "n", "r", "a", "v"]

# Convert the pairs to their index positions in the fictional
alphabet
pair1_indexes = [alphabet.index(letter) for letter in pair1]
pair2_indexes = [alphabet.index(letter) for letter in pair2]
```

```
# Calculate the differences between the corresponding elements of the
pairs
# to understand the transformation rule
diffs = [pair2_indexes[i] - pair1_indexes[i] for i in
range(len(pair1_indexes))]

pair1_indexes, pair2_indexes, diffs
```

The analysis shows that the transformation from the first sequence to the second sequence in the first pair involves keeping the first four positions the same (\(z, n, r, a\)) and changing the fifth position (\(p\) to \(v\)). In terms of the fictional alphabet's index positions, the change is from position 20 to position 15, which is a decrease of 5 positions.

Given this rule, we now need to apply the same transformation to the second sequence \([b, z, y, r, a]\) to find the missing term. Since the transformation affects only the last letter, and the change is a decrease of 5 positions in the alphabet, we will calculate the new position for the last letter of the second sequence accordingly.

```
# Define the second sequence that needs transformation
sequence_to_transform = ["b", "z", "y", "r", "a"]

# Convert the sequence to their index positions in the fictional
alphabet
sequence_indexes = [alphabet.index(letter) for letter in
sequence_to_transform]

# Apply the transformation rule to the last element
# Since we noticed a decrease of 5 positions for the last element in
the first pair,
# we apply the same logic here
transformed_last_index = sequence_indexes[-1] - 5

# Ensure the index wraps around if it goes below 0
transformed_last_index = transformed_last_index % len(alphabet)

# Find the corresponding letter in the fictional alphabet for the new
index
transformed_last_letter = alphabet[transformed_last_index]

# Construct the transformed sequence with the new last letter
transformed_sequence = sequence_to_transform[:-1] +
[transformed_last_letter]

transformed_sequence
```

The missing term in the sequence, applying the established transformation rule, is \([b, z, y, r, t]\). This means the sequence \([b, z, y, r, a]\) transforms to \([b, z, y, r, t]\) following the pattern observed in the given pairs.

Correct answer: [b z n r a]

Correct: False

Note: Error based on alternate rule

Problem #3

Let's solve a puzzle problem involving the following fictional alphabet:

[x y l k w b f z t n j r q a h v g m u o p d i c s e]

Here is the problem:

[t f q h g] [t j q h g]
[e g u p i] [?]

Response

To solve this puzzle, we first need to understand the relationship or pattern between the given sequences. The fictional alphabet is provided as:

\[x y l k w b f z t n j r q a h v g m u o p d i c s e\]

And the problem involves transforming one sequence into another. Given sequences are:

1. \([t f q h g] \rightarrow [t j q h g]\)
2. \([e g u p i] \rightarrow [?]\)

Let's start by identifying the pattern used to transform the first sequence into the second one. This will involve comparing the positions of the letters in the fictional alphabet and seeing how they change from the first sequence to the second. Once we understand the pattern, we can apply it to the second set of letters to find the missing sequence.

Let's begin by finding the positions of the letters in the first transformation \([t f q h g] \rightarrow [t j q h g]\) within the fictional alphabet and then identify the pattern.

```
# Fictional alphabet provided
alphabet = ["x", "y", "l", "k", "w", "b", "f", "z", "t", "n", "j",
"r", "q", "a", "h", "v", "g", "m", "u", "o", "p", "d", "i", "c", "s",
"e"]

# Sequences given in the problem
seq1_before = ["t", "f", "q", "h", "g"]
seq1_after = ["t", "j", "q", "h", "g"]

# Find the positions of each letter in the alphabet
positions_before = [alphabet.index(letter) for letter in seq1_before]
positions_after = [alphabet.index(letter) for letter in seq1_after]
```

```
# Compare positions to identify the pattern
pattern = [after - before for before, after in zip(positions_before,
positions_after)]

positions_before, positions_after, pattern
```

The transformation from \([t f q h g]\) to \([t j q h g]\) in the fictional alphabet corresponds to changing the positions as follows:

- For "t" (\(8th\) position), it remains the same in both sequences.
- For "f" (\(6th\) position), it changes to "j" (\(10th\) position), which is a shift of +4 positions in the fictional alphabet.
- For "q" (\(12th\) position), "h" (\(14th\) position), and "g" (\(16th\) position), there are no changes in their positions.

The pattern observed is a shift of +4 positions for the second letter in the sequence, while the other letters remain unchanged.

Applying this pattern to the second sequence \([e g u p i]\), we need to shift the second letter, "g", by +4 positions in the fictional alphabet to find the missing sequence. Let's calculate the new position for "g" and form the new sequence.

```
# Sequence to transform based on identified pattern
seq2_before = ["e", "g", "u", "p", "i"]

# Find the position of the second letter and shift it by +4
g_position_before = alphabet.index(seq2_before[1])
g_position_after = g_position_before + 4

# Apply transformation
seq2_before[1] = alphabet[g_position_after]

# Resulting sequence
seq2_after = seq2_before
seq2_after
```

Applying the observed pattern to the sequence \([e g u p i]\), by shifting the second letter "g" by +4 positions in the fictional alphabet, results in the sequence \([e p u p i]\). Therefore, the missing sequence that corresponds to \([e g u p i]\) is \([e p u p i]\).

Correct answer: [h g u p i]

Correct: False

Note: Error based on alternate rule

Problem #4

Let's solve a puzzle problem involving the following fictional alphabet:

[x y l k w b f z t n j r q a h v g m u o p d i c s e]

Here is the problem:

[f t j s h] [f t j q h]
[q h f u p] [?]

Response

To solve this puzzle, we need to understand the relationship or the pattern that transforms the first set of letters into the second set for each pair. Given the fictional alphabet and the sequences provided:

1. Alphabet: [x y l k w b f z t n j r q a h v g m u o p d i c s e]
2. Pairs given:
   - [f t j s h] transforms to [f t j q h]
   - [q h f u p] transforms to [?]

Let's first analyze the transformation from [f t j s h] to [f t j q h] using the provided alphabet. This analysis will help us understand the rule or pattern applied, which we can then use to determine the missing sequence.

We will compare the positions of the letters in the alphabet to see how they shift from the first sequence to the second.

```
# Define the fictional alphabet
alphabet = ['x', 'y', 'l', 'k', 'w', 'b', 'f', 'z', 't', 'n', 'j',
'r', 'q', 'a', 'h', 'v', 'g', 'm', 'u', 'o', 'p', 'd', 'i', 'c', 's',
'e']

# Define the sequences for transformation analysis
sequence_before = ['f', 't', 'j', 's', 'h']
sequence_after = ['f', 't', 'j', 'q', 'h']

# Calculate the shift in positions for each letter
shifts = [alphabet.index(sequence_after[i]) -
alphabet.index(sequence_before[i]) for i in
range(len(sequence_before))]

shifts
```

The transformation from [f t j s h] to [f t j q h] involves a shift in positions for only one letter: 's' shifts 12 positions backwards in the alphabet to become 'q', while the other letters ('f', 't', 'j', 'h') remain unchanged.

Given this pattern, let's apply the same rule to transform [q h f u p] accordingly. We'll shift the fourth letter ('u') 12 positions backwards in the fictional alphabet to find its corresponding letter and keep the other letters unchanged. Let's calculate this transformation.

```
# Sequence to transform based on the identified pattern
sequence_to_transform = ['q', 'h', 'f', 'u', 'p']

# Find the position of 'u' in the alphabet and shift it 12 positions
back
u_index = alphabet.index('u')
shifted_u_index = (u_index - 12) % len(alphabet)  # Use modulo to
wrap around the alphabet if necessary
transformed_u = alphabet[shifted_u_index]

# Construct the transformed sequence with the new letter
transformed_sequence = sequence_to_transform.copy()
transformed_sequence[3] = transformed_u  # Replace 'u' with its
transformed counterpart

transformed_sequence
```

The transformed sequence corresponding to [q h f u p] following the identified pattern is [q h f f p]. So, the missing sequence in the puzzle is [q h f f p].

Correct answer: [q h g u p]

Correct: False

Note: Error based on alternate rule

Problem #5

Let's solve a puzzle problem involving the following fictional alphabet:

[x y l k w b f z t n j r q a h v g m u o p d i c s e]

Here is the problem:

[c h g u p] [q h g u p]
[z d r a v] [?]

Response

To solve this puzzle, let's first analyze the pattern between the given sequences. We have two sequences and their corresponding outcomes:

1. `[c h g u p]` maps to `[q h g u p]`
2. `[z d r a v]` maps to `[ ? ]`

Given the fictional alphabet:

\[x y l k w b f z t n j r q a h v g m u o p d i c s e\]

Let's determine the rule that transforms the first sequence into the second one. Once we understand the transformation rule, we can apply it to `[z d r a v]` to find the missing sequence.

The transformation from `[c h g u p]` to `[q h g u p]` seems to involve a change only in the first character, from `c` to `q`. Let's find the positions of `c` and `q` in the fictional alphabet and see if we can identify a pattern or rule based on their positions. This will help us apply the same rule to the sequence `[z d r a v]` to find the corresponding missing sequence.

```
# Define the fictional alphabet
alphabet = ['x', 'y', 'l', 'k', 'w', 'b', 'f', 'z', 't', 'n', 'j',
'r', 'q', 'a', 'h', 'v', 'g', 'm', 'u', 'o', 'p', 'd', 'i', 'c', 's',
'e']

# Find the positions of 'c' and 'q' in the alphabet
c_pos = alphabet.index('c') + 1  # Adding 1 to get the position in a
human-readable format (1-based index)
q_pos = alphabet.index('q') + 1

# Calculate the difference between positions of 'c' and 'q'
diff = q_pos - c_pos
```

```
c_pos, q_pos, diff
```

The position of `c` in the fictional alphabet is 24, and the position of `q` is 13, resulting in a difference of -11 positions. This suggests that to transform `c` to `q`, we move 11 positions to the left in the alphabet.

To apply this rule to the sequence `[z d r a v]` and find the missing sequence, let's first find the position of `z` in the alphabet and then move 11 positions to the left, wrapping around if necessary, to determine the first character of the missing sequence.

```
# Find the position of 'z' in the alphabet
z_pos = alphabet.index('z') + 1  # Adding 1 for 1-based index

# Calculate the new position after moving 11 positions to the left
new_pos = z_pos - diff  # Since diff is negative, subtracting it
actually adds to the position

# Adjust for wrapping around the alphabet if necessary
new_pos = (new_pos - 1) % len(alphabet)  # Using 0-based index for
calculation

# Find the corresponding character in the alphabet
new_char = alphabet[new_pos]

new_char
```

The character corresponding to `z` after applying the transformation rule (moving 11 positions to the left in the fictional alphabet) is `u`.

Therefore, the missing sequence corresponding to `[z d r a v]` begins with `[u ...]`. However, based on the transformation observed, it seems that only the first character of the sequence is altered based on the rule we've deduced. Given this pattern, the complete transformation of `[z d r a v]` would result in `[u d r a v]`.

Correct answer: [z n r a v]

Correct: False

Problem #6

Let's solve a puzzle problem involving the following fictional alphabet:

[x y l k w b f z t n j r q a h v g m u o p d i c s e]

Here is the problem:

[v m u d c] [v m o d c]
[z q h g u] [?]

Response

To solve this puzzle, we need to understand the pattern or logic applied between the given sets of characters in the fictional alphabet and then apply that logic to find the missing set of characters.

Given the fictional alphabet:
\[[x\, y\, l\, k\, w\, b\, f\, z\, t\, n\, j\, r\, q\, a\, h\, v\, g\, m\, u\, o\, p\, d\, i\, c\, s\, e] \]

We have the following transformation:
\[[v\, m\, u\, d\, c] \rightarrow [v\, m\, o\, d\, c] \]

And we need to apply a similar transformation to:
\[[z\, q\, h\, g\, u] \rightarrow [?] \]

First, let's identify the positions of the letters in the fictional alphabet for both transformations. This will help us understand the rule applied from the first transformation to apply it to the second.

```
# Define the fictional alphabet as a list for easy indexing
alphabet = ["x", "y", "l", "k", "w", "b", "f", "z", "t", "n", "j",
"r", "q", "a", "h", "v", "g", "m", "u", "o", "p", "d", "i", "c", "s",
"e"]

# Define the transformations
initial_sequence = ["v", "m", "u", "d", "c"]
transformed_sequence = ["v", "m", "o", "d", "c"]
second_sequence = ["z", "q", "h", "g", "u"]

# Find the index of each letter in the alphabet
initial_indexes = [alphabet.index(letter) for letter in
initial_sequence]
```

```
transformed_indexes = [alphabet.index(letter) for letter in
transformed_sequence]
second_sequence_indexes = [alphabet.index(letter) for letter in
second_sequence]

initial_indexes, transformed_indexes, second_sequence_indexes
```

The transformation from \([v\, m\, u\, d\, c]\) to \([v\, m\, o\, d\, c]\) in terms of their positions in the fictional alphabet is from \([15, 17, 18, 21, 23]\) to \([15, 17, 19, 21, 23]\). This shows that only the third letter changed its position from 18 to 19, which corresponds to a change from \(u\) to \(o\).

Applying the same logic to the second sequence \([z\, q\, h\, g\, u]\), which corresponds to positions \([7, 12, 14, 16, 18]\), we need to change the position of the third letter from 14 (corresponding to \(h\)) to the next position in the sequence following the same pattern observed in the transformation of the first sequence.

Let's calculate the new positions and the resulting characters for the second sequence.

```
# Apply the same transformation logic to the second sequence
# Incrementing the third character's position by 1 (from 'h' to the
next in the sequence)
transformed_second_sequence_indexes = second_sequence_indexes.copy()
transformed_second_sequence_indexes[2] += 1  # Increment the third
character's position

# Convert indexes back to characters
transformed_second_sequence = [alphabet[index] for index in
transformed_second_sequence_indexes]

transformed_second_sequence_indexes, transformed_second_sequence
```

The transformed sequence for \([z\, q\, h\, g\, u]\) following the pattern is \([z\, q\, v\, g\, u]\), corresponding to the positions \([7, 12, 15, 16, 18]\) in the fictional alphabet. This means we increment the position of the third letter by 1, changing \(h\) to \(v\).

Therefore, the answer to the puzzle, filling in the missing sequence, is \([z\, q\, v\, g\, u]\).

Correct answer: [j q h g u]

Correct: False

Note: Error based on alternate rule

Problem #7

Let's solve a puzzle problem involving the following fictional alphabet:

[x y l k w b f z t n j r q a h v g m u o p d i c s e]

Here is the problem:

[r a v m i] [r a v m o]
[l w f t d] [?]

Response

To solve this puzzle, we need to understand the pattern or rule that connects the first pair of words and apply the same rule to the second pair. The given fictional alphabet and the first pair of words are:

Alphabet: [x y l k w b f z t n j r q a h v g m u o p d i c s e]

First Pair: [r a v m i] [r a v m o]

Second Pair: [l w f t d] [?]

To solve this, let's first identify the positions of the letters in the first pair within the fictional alphabet and see if we can discern a pattern.

1. For [r a v m i] to [r a v m o], let's find the positions of 'i' and 'o' in the alphabet and see the relation.

2. Apply the discovered rule to find the corresponding letter for [l w f t d].

Let's begin with step 1 to decode the pattern.

```
# Define the fictional alphabet
alphabet = ['x', 'y', 'l', 'k', 'w', 'b', 'f', 'z', 't', 'n', 'j',
'r', 'q', 'a', 'h', 'v', 'g', 'm', 'u', 'o', 'p', 'd', 'i', 'c', 's',
'e']

# Define the letters to investigate
first_pair_end = 'i'
second_pair_end = 'o'

# Find the positions of the letters in the alphabet
first_pair_end_position = alphabet.index(first_pair_end)
```

```
second_pair_end_position = alphabet.index(second_pair_end)

# Calculate the difference or relation
relation = second_pair_end_position - first_pair_end_position

first_pair_end_position, second_pair_end_position, relation
```

The positions of 'i' and 'o' in the fictional alphabet are 22 and 19, respectively, meaning that to go from 'i' to 'o', we move 3 positions backward in the alphabet.

Now, let's apply this pattern to the second pair. We need to find the letter in the fictional alphabet that corresponds to moving 3 positions backward from 'd' (the last letter in [l w f t d]) to find the missing letter for the second pair.

```
# Define the letter to transform from the second pair
second_pair_start = 'd'

# Find the position of 'd' in the alphabet
second_pair_start_position = alphabet.index(second_pair_start)

# Calculate the new position by applying the relation
new_position = second_pair_start_position + relation

# Ensure the new position wraps correctly within the alphabet range
new_position = new_position % len(alphabet)

# Find the corresponding letter in the alphabet
missing_letter = alphabet[new_position]

missing_letter
```

The missing letter for the second pair, following the discovered pattern, is 'u'. Therefore, when applying the same rule we deduced from the first pair, [l w f t d] corresponds to [l w f t u].

Correct answer: [l w f t j]

Correct: False

Note: Error based on alternate rule

Problem #8

Let's solve a puzzle problem involving the following fictional alphabet:

[x y l k w b f z t n j r q a h v g m u o p d i c s e]

Here is the problem:

[t v m o d] [a v m o d]
[j q o g u] [?]

Response

To solve this puzzle, we need to understand the relationship between the given fictional alphabet and the sequences provided, in order to fill in the missing sequence.

Given fictional alphabet: [x y l k w b f z t n j r q a h v g m u o p d i c s e]

Given sequences:
1. [t v m o d] [a v m o d]
2. [j q o g u] [?]

The first step is to look for a pattern or relationship between the first pair of sequences "[t v m o d] [a v m o d]" to apply the same logic to find the missing sequence for "[j q o g u]".

Given the sequences "[t v m o d] [a v m o d]", it seems like the transformation involves keeping the middle three letters "[v m o]" the same and changing the first and last letters according to some rule based on the provided alphabet.

Let's first map the provided fictional alphabet in a way that we can easily reference their positions or relationships, then try to deduce the rule that changes "[t]" to "[a]" and "[d]" to "[d]", and apply that rule to the sequence "[j q o g u]" to find the missing sequence.

```
# Mapping the fictional alphabet to index values for easy reference
alphabet = ["x", "y", "l", "k", "w", "b", "f", "z", "t", "n", "j",
"r", "q", "a", "h", "v", "g", "m", "u", "o", "p", "d", "i", "c", "s",
"e"]
alphabet_dict = {letter: index for index, letter in
enumerate(alphabet)}

# Given sequences
seq1 = ["t", "v", "m", "o", "d"]
seq2 = ["a", "v", "m", "o", "d"]
seq3 = ["j", "q", "o", "g", "u"]
```

```
# Analyzing the transformation between seq1 and seq2 to apply it to
seq3
first_letter_change = alphabet_dict[seq2[0]] - alphabet_dict[seq1[0]]
last_letter_change = alphabet_dict[seq2[-1]] -
alphabet_dict[seq1[-1]]

# Applying the transformation to seq3
seq4_first_letter_index = (alphabet_dict[seq3[0]] +
first_letter_change) % len(alphabet)
seq4_last_letter_index = (alphabet_dict[seq3[-1]] +
last_letter_change) % len(alphabet)

# Constructing the missing sequence
seq4 = [alphabet[seq4_first_letter_index]] + seq3[1:4] +
[alphabet[seq4_last_letter_index]]

seq4_first_letter_index, seq4_last_letter_index, seq4
```

The missing sequence corresponding to "[j q o g u]" is "[v q o g u]". This was deduced by applying the transformation observed between the first pair of sequences to the given sequence, indicating that the pattern involves specific changes to the first and last letters while keeping the middle letters constant.

Correct answer: [j q h g u]

Correct: False

Note: Error based on alternate rule

Problem #9

Let's solve a puzzle problem involving the following fictional alphabet:

[x y l k w b f z t n j r q a h v g m u o p d i c s e]

Here is the problem:

[n m o d c] [v m o d c]
[q i g u p] [?]

Response

To solve this puzzle, we need to understand the relationship between the characters in the first set of brackets and how they correspond to the characters in the second set. This looks like a mapping or transformation problem within a given fictional alphabet:

\[x y l k w b f z t n j r q a h v g m u o p d i c s e\]

Given groups are:

1. \([n m o d c] \to [v m o d c]\)
2. \([q i g u p] \to [?]\)

We first identify the transformation applied from the first group to its corresponding group. Since the fictional alphabet is provided, we can determine the position of each character in the alphabet and then see how they map to each other. This approach should reveal the pattern that we can apply to the second group to find the missing characters.

Let's start by identifying the positions of each character in the fictional alphabet for both parts of the given group and then determine the transformation applied.

```
# Alphabet in the given fictional order
alphabet = ['x', 'y', 'l', 'k', 'w', 'b', 'f', 'z', 't', 'n', 'j',
'r', 'q', 'a', 'h', 'v', 'g', 'm', 'u', 'o', 'p', 'd', 'i', 'c', 's',
'e']

# Given transformations
group_1_source = ['n', 'm', 'o', 'd', 'c']
group_1_target = ['v', 'm', 'o', 'd', 'c']

# Function to find positions of characters in the alphabet
def find_positions(characters):
    return [alphabet.index(c) for c in characters]
```

```
# Finding positions
group_1_source_positions = find_positions(group_1_source)
group_1_target_positions = find_positions(group_1_target)

# Calculate the transformation (difference in positions)
transformation = [target - source for source, target in
zip(group_1_source_positions, group_1_target_positions)]

group_1_source_positions, group_1_target_positions, transformation
```

The transformation from the first group \([n, m, o, d, c]\) to the second group \([v, m, o, d, c]\) is characterized by the positions of each character in the fictional alphabet. Here's the analysis:

- The positions of \([n, m, o, d, c]\) in the alphabet are \([9, 17, 19, 21, 23]\).
- The positions of \([v, m, o, d, c]\) in the alphabet are \([15, 17, 19, 21, 23]\).
- The transformation applied to the first character (n to v) changes its position by +6 in the alphabet, while the positions of the other characters (m, o, d, c) remain unchanged.

This pattern indicates that only the first character in the group is transformed by moving 6 positions forward in the alphabet, and the rest of the characters stay the same.

Applying this pattern to the second group \([q, i, g, u, p]\), we need to find the character that is 6 positions ahead of 'q' in the fictional alphabet and keep the rest of the characters (i, g, u, p) unchanged. Let's calculate the transformed character for the second group.

```
# Given second group source
group_2_source = ['q', 'i', 'g', 'u', 'p']

# Find the position of the first character 'q' and apply the
transformation
q_position = alphabet.index(group_2_source[0])
transformed_q_position = q_position + transformation[0]

# Find the transformed character for 'q'
transformed_q = alphabet[transformed_q_position]

# Construct the transformed group using the new first character and
the unchanged rest
group_2_transformed = [transformed_q] + group_2_source[1:]

group_2_transformed
```

The transformed group corresponding to \([q, i, g, u, p]\) following the identified pattern is \([u, i, g, u, p]\). Therefore, the missing group for the puzzle problem is \([u, i, g, u, p]\).

Correct answer: [q h g u p]

Correct: False

Note: Error based on alternate rule

Problem #10

Let's solve a puzzle problem involving the following fictional alphabet:

[x y l k w b f z t n j r q a h v g m u o p d i c s e]

Here is the problem:

[g u p q s] [g u p i s]
[v m o d p] [?]

Response

To solve this puzzle, we need to establish a pattern or relationship between the two sets of letters in the first pair, and then apply the same pattern to the second pair to find the missing sequence.

Given the fictional alphabet order:
\[[x \, y \, l \, k \, w \, b \, f \, z \, t \, n \, j \, r \, q \, a \, h \, v \, g \, m \, u \, o \, p \, d \, i \, c \, s \, e] \]

And the transformation from the first sequence to the second sequence in the first pair:
\[\text{[g u p q s]} \rightarrow \text{[g u p i s]} \]

We observe that the transformation involves changing one or more letters while keeping others the same. Let's determine the pattern of change and then apply this pattern to the second pair \([v m o d p] \rightarrow [?] \).

To establish the pattern, we will map each letter of the sequences to their position in the fictional alphabet, then determine the transformation rule. Let's start by doing this for the first pair and then apply the discovered rule to the second pair.

```
# Fictional alphabet order
alphabet = ['x', 'y', 'l', 'k', 'w', 'b', 'f', 'z', 't', 'n', 'j',
'r', 'q', 'a', 'h', 'v', 'g', 'm', 'u', 'o', 'p', 'd', 'i', 'c', 's',
'e']

# First pair of sequences
first_pair_start = ['g', 'u', 'p', 'q', 's']
first_pair_end = ['g', 'u', 'p', 'i', 's']

# Second pair to solve
second_pair_start = ['v', 'm', 'o', 'd', 'p']
```

```
# Mapping each letter to its position in the fictional alphabet
first_pair_start_indices = [alphabet.index(letter) for letter in
first_pair_start]
first_pair_end_indices = [alphabet.index(letter) for letter in
first_pair_end]
second_pair_start_indices = [alphabet.index(letter) for letter in
second_pair_start]

# Determine the transformation rule based on the indices
transformation_rule = [end - start for start, end in
zip(first_pair_start_indices, first_pair_end_indices)]

# Apply the transformation rule to the second pair
second_pair_end_indices = [start + change if 0 <= start + change <
len(alphabet) else (start + change) % len(alphabet)
                           for start, change in
zip(second_pair_start_indices, transformation_rule)]

# Mapping indices back to letters for the solution of the second pair
second_pair_end = [alphabet[index] for index in
second_pair_end_indices]

transformation_rule, second_pair_end
```

The transformation rule between the first pair of sequences \([g u p q s] \rightarrow [g u p i s]\) is to keep most letters the same except for one, which changes its position in the fictional alphabet. Specifically, the fourth letter shifts 10 positions forward in the alphabet order.

Applying this transformation rule to the second pair \([v m o d p]\), we keep all letters the same except for the fourth letter, which shifts 10 positions forward, resulting in the sequence \([v m o b p]\).

Therefore, the missing sequence that corresponds to \([v m o d p]\) is \([v m o b p]\).

Correct answer: [v m o d c]

Correct: False

Note: second attempt (refused to give final answer on first attempt), error based on alternate rule

Final score: 0/10

Errors based on alternate rules: 9/10