**1. Write a function that counts how many concentric layers a rug.**

**Examples:**  
count\_layers(["AAAA","ABBA","AAAA"]) ➞ 2  
count\_layers(["AAAAAAAAA","ABBBBBBBA","ABBAAABBA","ABBBBBBBA","AAAAAAAAA"]) ➞ 3  
count\_layers(["AAAAAAAAAAA","AABBBBBBBAA","AABCCCCCBAA","AABCAAACBAA","AABCADACBAA","AABCAAACBAA","AABCCCCCBAA","AABBBBBBBAA","AAAAAAAAAAA"]) ➞ 5

In [1]:

**def** count\_layers(in\_list):

out\_list **=** []

**for** ele **in** in\_list:

**if** ele **not** **in** out\_list:

out\_list**.**append(ele)

print(f'count\_layers({in\_list}) ➞ {len(out\_list)}')

count\_layers(["AAAA","ABBA","AAAA"])

count\_layers(["AAAAAAAAA","ABBBBBBBA","ABBAAABBA","ABBBBBBBA","AAAAAAAAA"])

count\_layers(["AAAAAAAAAAA","AABBBBBBBAA","AABCCCCCBAA","AABCAAACBAA","AABCADACBAA","AABCAAACBAA","AABCCCCCBAA","AABBBBBBBAA","AAAAAAAAAAA"])

count\_layers(['AAAA', 'ABBA', 'AAAA']) ➞ 2

count\_layers(['AAAAAAAAA', 'ABBBBBBBA', 'ABBAAABBA', 'ABBBBBBBA', 'AAAAAAAAA']) ➞ 3

count\_layers(['AAAAAAAAAAA', 'AABBBBBBBAA', 'AABCCCCCBAA', 'AABCAAACBAA', 'AABCADACBAA', 'AABCAAACBAA', 'AABCCCCCBAA', 'AABBBBBBBAA', 'AAAAAAAAAAA']) ➞ 5

**2. There are many different styles of music and many albums exhibit multiple styles. Create a function that takes a list of musical styles from albums and returns how many styles are unique.**

**Examples:**  
unique\_styles([ "Dub,Dancehall", "Industrial,Heavy Metal", "Techno,Dubstep", "Synth-pop,Euro-Disco", "Industrial,Techno,Minimal" ]) ➞ 9  
unique\_styles([ "Soul", "House,Folk", "Trance,Downtempo,Big Beat,House", "Deep House", "Soul" ]) ➞ 7

In [2]:

**def** unique\_styles(in\_list):

out\_list **=** []

**for** ele **in** in\_list:

**for** sub\_ele **in** ele**.**split(','):

out\_list**.**append(sub\_ele)

print(f'unique\_styles({in\_list}) ➞ {len(set(out\_list))}')

unique\_styles(["Dub,Dancehall","Industrial,Heavy Metal","Techno,Dubstep","Synth-pop,Euro-Disco","Industrial,Techno,Minimal"])

unique\_styles(["Soul","House,Folk","Trance,Downtempo,Big Beat,House","Deep House","Soul"])

unique\_styles(['Dub,Dancehall', 'Industrial,Heavy Metal', 'Techno,Dubstep', 'Synth-pop,Euro-Disco', 'Industrial,Techno,Minimal']) ➞ 9

unique\_styles(['Soul', 'House,Folk', 'Trance,Downtempo,Big Beat,House', 'Deep House', 'Soul']) ➞ 7

**3. Create a function that finds a target number in a list of prime numbers. Implement a binary search algorithm in your function. The target number will be from 2 through 97. If the target is prime then return "yes" else return "no".**

**Examples:**  
primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]  
is\_prime(primes, 3) ➞ "yes"  
is\_prime(primes, 4) ➞ "no"  
is\_prime(primes, 67) ➞ "yes"  
is\_prime(primes, 36) ➞ "no"

In [3]:

primes **=** [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]

**def** is\_prime(in\_list,in\_num):

output **=** 'No'

start\_point **=** 0

end\_point **=** len(in\_list) **-** 1

mid\_point **=** 0

**while** start\_point **<=** end\_point:

mid\_point **=** (end\_point**+**start\_point)**//**2

**if** in\_list[mid\_point] **<** in\_num:

start\_point **=** mid\_point **+** 1

**elif** in\_list[mid\_point] **>** in\_num:

end\_point **=** mid\_point **-** 1

**else**:

output **=** 'Yes'

**break**

print(f'is\_prime({in\_num}) ➞ {output}')

is\_prime(primes, 3)

is\_prime(primes, 4)

is\_prime(primes, 67)

is\_prime(primes, 36)

is\_prime(3) ➞ Yes

is\_prime(4) ➞ No

is\_prime(67) ➞ Yes

is\_prime(36) ➞ No

**4. Create a function that takes in n, a, b and returns the number of positive values raised to the nth power that lie in the range [a, b], inclusive.**

**Examples:**  
power\_ranger(2, 49, 65) ➞ 2  
# 2 squares (n^2) lie between 49 and 65, 49 (7^2) and 64 (8^2)

power\_ranger(3, 1, 27) ➞ 3  
# 3 cubes (n^3) lie between 1 and 27, 1 (1^3), 8 (2^3) and 27 (3^3)

power\_ranger(10, 1, 5) ➞ 1  
# 1 value raised to the 10th power lies between 1 and 5, 1 (1^10)

power\_ranger(5, 31, 33) ➞ 1  
power\_ranger(4, 250, 1300) ➞ 3

In [4]:

**import** math

**def** power\_ranger(in\_base,in\_min,in\_max):

output **=** []

**for** ele **in** range(in\_min,in\_max**+**1):

root **=** round(math**.**exp(math**.**log(ele)**/**in\_base),1)

**if** str(root)**.**split(".")[1] **==** '0':

output**.**append(int(root))

print(f'power\_ranger{in\_base,in\_min,in\_max} ➞ {len(set(output))}')

power\_ranger(2, 49, 65)

power\_ranger(3, 1, 27)

power\_ranger(10, 1, 5)

power\_ranger(5, 31, 33)

power\_ranger(4, 250, 1300)

power\_ranger(2, 49, 65) ➞ 2

power\_ranger(3, 1, 27) ➞ 3

power\_ranger(10, 1, 5) ➞ 1

power\_ranger(5, 31, 33) ➞ 1

power\_ranger(4, 250, 1300) ➞ 3

**5. Given a number, return the difference between the maximum and minimum numbers that can be formed when the digits are rearranged.**

**Examples:**  
rearranged\_difference(972882) ➞ 760833  
# 988722 - 227889 = 760833  
rearranged\_difference(3320707) ➞ 7709823  
# 7733200 - 23377 = 7709823  
rearranged\_difference(90010) ➞ 90981  
# 91000 - 19 = 90981

In [5]:

**def** rearranged\_difference(in\_num):

split\_num **=** []

**for** ele **in** str(in\_num):

split\_num**.**append(ele)

min\_num **=** int(''**.**join(sorted(split\_num)))

max\_num **=** int(''**.**join(sorted(split\_num, reverse**=True**)))

print(f'rearranged\_difference({in\_num}) ➞ {max\_num} - {min\_num} ➞ {max\_num**-**min\_num}')

rearranged\_difference(972882)

rearranged\_difference(3320707)

rearranged\_difference(90010)

rearranged\_difference(972882) ➞ 988722 - 227889 ➞ 760833

rearranged\_difference(3320707) ➞ 7733200 - 23377 ➞ 7709823

rearranged\_difference(90010) ➞ 91000 - 19 ➞ 90981