Question 1

In [7]:
```
import re

def replace_characters(input_string):
    pattern = r'[ ,.]'
    result_string = re.sub(pattern, ':', input_string)

    return result_string
input_text = "Hello, world. python is very powerfull language"
result = replace_characters(input_text)
print("Original text:", input_text)
print("Modified text:", result)
```

```
Original text: Hello, world. python is very powerfull language
Modified text: Hello::world::python:is:very:powerfull:language
```

Question 2

In [8]:
```
import pandas as pd
import re
data = {'SUMMARY': ['hello, world!', 'XXXXX test', '123four, five:; six...']}
df = pd.DataFrame(data)
df['SUMMARY'] = df['SUMMARY'].apply(lambda x: re.sub(r'[^a-zA-Z\s]', '', x))
print(df)
```

```
         SUMMARY
0    hello world
1     XXXXX test
2  four five six
```

Question 3   Create a funcθon in python to find all words that are at least 4 characters long in a string.
The use of the re.compile() method is mandatory.

In [10]:
```
import re

def find_long_words(input_string):
    pattern = re.compile(r'\b\w{4,}\b')
    result = pattern.findall(input_string)
    return result
input_string = "Hello Ravi please help out business in time"
long_words = find_long_words(input_string)
print("Input String:", input_string)
print("Words with at least 4 characters:", long_words)
```

```
Input String: Hello Ravi please help out business in time
Words with at least 4 characters: ['Hello', 'Ravi', 'please', 'help', 'business', 't
ime']
```

In [ ]:
```
Question 4
```

In [15]:
```
import re

def find_words_of_lengths(input_string, lengths):
    if not isinstance(lengths, tuple):
        raise ValueError("Lengths must be a tuple")
        pattern = re.compile(r'\b\w{%s}\b' % '|'.join(map(str, lengths)))
        result = pattern.findall(input_string)
        return result
```

In [17]:
```
input_string = "The Indian government should work on diversity sectors to establish ar
desired_lengths = (3, 4, 5)

words_of_lengths = find_words_of_lengths(input_string, desired_lengths)

print("Input String:", input_string)
print(f"Words of lengths {desired_lengths}: {words_of_lengths}")
```

```
Input String: The Indian government should work on diversity sectors to establish an
equality around the country
Words of lengths (3, 4, 5): None
```

In [ ]:
```
Question 5
```

In [22]:
```


import re
def remove_parentheses(strings_list):
    if not isinstance(strings_list, list):
        raise ValueError("Input must be a list of strings")
        pattern = re.compile(r'\(|\)')
        result = [pattern.sub('', s) for s in strings_list]
        return result
input_strings = ["(First String)", "(Second String)", "(Third String)"]

strings_without_parentheses = remove_parentheses(input_strings)

print("Input Strings:", input_strings)
print("Strings without Parentheses:", strings_without_parentheses)
```

```
Input Strings: ['(First String)', '(Second String)', '(Third String)']
Strings without Parentheses: None
```

In [ ]:
```
Question 6
```

In [27]:
```
import re
def remove_parentheses_from_file(file_path):
    with open(file_path, 'r') as file:
        text = file.read()
        pattern = re.compile(r'\([^)]*\)')
        modified_text = pattern.sub('', text)
        with open(file_path, 'w') as file:
        file.write(modified_text)
        file_path = 'sample_text.txt'
        sample_text = ["example (.com)", "hr@fliprobo (.com)", "github (.com)", "Hell
        with open(file_path, 'w') as file:
            for line in sample_text:
        file.write(line + '\n')
        remove_parentheses_from_file(file_path)
        with open(file_path, 'r') as file:
    modified_text = file.read()
print("Original Text in File:")
print('\n'.join(sample_text))
print("\nModified Text in File:")
print(modified_text)
```

```
  Cell In[27], line 8
    file.write(modified_text)
    ^
IndentationError: expected an indented block after 'with' statement on line 7
```

Question 7

In [32]:
```
import re

def split_into_uppercase(input_string):
    pattern = re.compile(r'[A-Z]')
    result = pattern.findall(input_string)
    result_string = ''.join(result)

    return result_string
input_string = "ImportanceOfRegularExpressionsInPython"

result = split_into_uppercase(input_string)

print("Input String:", input_string)
print("Result:", result)
```

```
Input String: ImportanceOfRegularExpressionsInPython
Result: IOREIP
```

In [ ]:

In [ ]:

```
Question 8
```

In [30]:
```python
import re

def insert_spaces_before_numbers(input_string):
    pattern = re.compile(r'(?<=\D)(?=\d)')
    result = pattern.sub(' ', input_string)
    return result
def main():
    input_string = "RegularExpression1IsAn2ImportantTopic3InPython"

    result = insert_spaces_before_numbers(input_string)

    print("Input String:", input_string)
    print("Result:", result)

if __name__ == "__main__":
    main()
```

```
Input String: RegularExpression1IsAn2ImportantTopic3InPython
Result: RegularExpression 1IsAn 2ImportantTopic 3InPython
```

In [ ]:
```
Question 9
```

In [34]:
```python
def insert_spaces_before_capitals_and_numbers(input_string):
    pattern = re.compile(r'(?<=[A-Z0-9])(?=[A-Z])|(?<=[a-z0-9])(?=[0-9A-Z])')
    result = pattern.sub(' ', input_string)

    return result
def main():
    input_string = "RegularExpression1IsAn2ImportantTopic3InPython"

    result = insert_spaces_before_capitals_and_numbers(input_string)

    print("Input String:", input_string)
    print("Result:", result)

if __name__ == "__main__":
    main()
```

```
Input String: RegularExpression1IsAn2ImportantTopic3InPython
Result: Regular Expression 1 Is An 2 Important Topic 3 In Python
```

```
Question 10  not able to get the details
```

In [ ]:
```
question 11
```

In [35]:
```
import re

def match_valid_string(input_string):
    pattern = re.compile(r'^[a-zA-Z0-9_]+$')
    result = pattern.match(input_string)

    return result is not None
def main():
    valid_strings = ["Valid_String123", "Another_Valid_String", "123_underscored_stri
    invalid_strings = ["Invalid String!", "Spaces Are Not Allowed", "Special@Characte

    print("Valid Strings:")
    for s in valid_strings:
        print(f"{s}: {match_valid_string(s)}")
        print("\nInvalid Strings:")
    for s in invalid_strings:
        print(f"{s}: {match_valid_string(s)}")

if __name__ == "__main__":
    main()
```

```
Valid Strings:
Valid_String123: True

Invalid Strings:
Another_Valid_String: True

Invalid Strings:
123_underscored_string: True

Invalid Strings:
Invalid String!: False
Spaces Are Not Allowed: False
Special@Character: False
```

question 12

In [39]:
```
import re

def starts_with_number(input_string, specific_number):
    pattern = re.compile(r'^' + re.escape(str(specific_number)))
    result = pattern.match(input_string)

    return result is not None
def main():
    test_strings = ["12345_SomeText", "56789_AnotherText", "987654_NotMatchingText"]

    specific_number = 567

    print(f"Strings starting with the number {specific_number}:")
    for s in test_strings:
        print(f"{s}: {starts_with_number(s, specific_number)}")
        if __name__ == "__main__":
    main()
```

```
  Cell In[39], line 17
    main()
    ^
IndentationError: expected an indented block after 'if' statement on line 16
```

question 13

In [41]:
```
def remove_leading_zeros(ip_address):
    octets = ip_address.split('.')
    updated_octets = [str(int(octet)) for octet in octets]
    updated_ip_address = '.'.join(updated_octets)

    return updated_ip_address
ip_address = "442.058.101.000"
updated_ip = remove_leading_zeros(ip_address)

print("Original IP Address:", ip_address)
print("Updated IP Address:", updated_ip)
```

```
Original IP Address: 442.058.101.000
Updated IP Address: 442.58.101.0
```

question 14

In [42]:
```
import re

def find_dates_in_text(input_text):
    pattern = re.compile(r'\b(?:January|February|March|April|May|June|July|August|Sep
    result = pattern.findall(input_text)

    return result
sample_text = "On August 15th 1947 that India was declared independent from British co
dates_found = find_dates_in_text(sample_text)

print("Dates found in the text:")
for date in dates_found:
    print(date)
```

```
Dates found in the text:
August 15th 1947
```

question 15

In [43]:
```
import re

def search_literals(main_string, literals):
    escaped_literals = [re.escape(literal) for literal in literals]
    pattern = re.compile('|'.join(escaped_literals))
    result = pattern.findall(main_string)

    return result
sample_text = 'The quick brown fox jumps over the lazy dog.'
search_literals_list = ["quick", "fox", "lazy"]
found_literals = search_literals(sample_text, search_literals_list)

print("Sample Text:", sample_text)
print("Literals found:", found_literals)
```

```
Sample Text: The quick brown fox jumps over the lazy dog.
Literals found: ['quick', 'fox', 'lazy']
```

question 16

In [44]:
```
import re

def search_literal_and_location(main_string, search_literal):
    escaped_literal = re.escape(search_literal)
    pattern = re.compile(escaped_literal, re.IGNORECASE)
    matches = pattern.finditer(main_string)
    result = [{'match': match.group(), 'start': match.start(), 'end': match.end()} fo

    return result
sample_text = 'The quick brown fox jumps over the lazy dog.'
search_literal = "fox"
results = search_literal_and_location(sample_text, search_literal)

print("Sample Text:", sample_text)
print(f"Literals found: {len(results)}")

for result in results:
    print(f"Match: {result['match']}, Start: {result['start']}, End: {result['end']}"
```

```
Sample Text: The quick brown fox jumps over the lazy dog.
Literals found: 1
Match: fox, Start: 16, End: 19
```

question 17

In [46]:
```
import re

def find_substrings_using_regex(main_string, substring):
    pattern = re.compile(re.escape(substring), re.IGNORECASE)

    matches = pattern.finditer(main_string)
    results = [(match.group(), match.start(), match.end()) for match in matches]

    return results
sample_text = 'Python exercises, PHP exercises, C# exercises'
substring_to_find = 'exercises'

results = find_substrings_using_regex(sample_text, substring_to_find)

print("Sample Text:", sample_text)
print(f"Occurrences of '{substring_to_find}': {results}")
```

```
Sample Text: Python exercises, PHP exercises, C# exercises
Occurrences of 'exercises': [('exercises', 7, 16), ('exercises', 22, 31), ('exercise
s', 36, 45)]
```

In [ ]:

question 18

In [47]:
```
import re

def find_occurrences_and_positions_regex(main_string, substring):
    pattern = re.compile(re.escape(substring), re.IGNORECASE)
    matches = pattern.finditer(main_string)
    occurrences = [(match.group(), match.start(), match.end()) for match in matches]

    return occurrences
sample_text = 'Python exercises, PHP exercises, C# exercises'
substring_to_find = 'exercises'

occurrences = find_occurrences_and_positions_regex(sample_text, substring_to_find)

print("Sample Text:", sample_text)
print(f"Occurrences of '{substring_to_find}':")
for occurrence in occurrences:
    print(f"Substring: {occurrence[0]}, Start: {occurrence[1]}, End: {occurrence[2]}"
```

```
Sample Text: Python exercises, PHP exercises, C# exercises
Occurrences of 'exercises':
Substring: exercises, Start: 7, End: 16
Substring: exercises, Start: 22, End: 31
Substring: exercises, Start: 36, End: 45
```

question 19

In [48]:
```
from datetime import datetime

def convert_date_format(input_date):
    parsed_date = datetime.strptime(input_date, '%Y-%m-%d')
    formatted_date = parsed_date.strftime('%d-%m-%Y')
    return formatted_date
input_date = '2023-12-14'
output_date = convert_date_format(input_date)

print(f"Original Date: {input_date}")
print(f"Converted Date: {output_date}")
```

```
Original Date: 2023-12-14
Converted Date: 14-12-2023
```

question 20

In [49]:
```
import re

def find_decimal_numbers(input_string):
    pattern = re.compile(r'\b\d+\.\d{1,2}\b')
    result = pattern.findall(input_string)

    return result
sample_text = "01.12 0132.123 2.31875 145.8 3.01 27.25 0.25"
decimal_numbers = find_decimal_numbers(sample_text)

print("Sample Text:", sample_text)
print("Found Decimal Numbers:", decimal_numbers)
```

```
Sample Text: 01.12 0132.123 2.31875 145.8 3.01 27.25 0.25
Found Decimal Numbers: ['01.12', '145.8', '3.01', '27.25', '0.25']
```

In [ ]:
```
question 21
```

In [56]:
```
import re
def separate_and_print_numbers(input_string):
    pattern = re.compile(r'\b\d+\b')
    matches = pattern.finditer(input_string)
for match in matches :
        number = match.group()
        start_position = match.start()
        end_position = match.end()
        print(f"Number: {number}, Start Position: {start_position}, End Position: {en
        sample_text = "The price of the product is $25.99 and the quantity is 10."

print("Sample Text:", sample_text)
print("Numbers and Their Positions:")
separate_and_print_numbers(sample_text)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[56], line 5
      3     pattern = re.compile(r'\b\d+\b')
      4     matches = pattern.finditer(input_string)
----> 5 for match in matches :
      6         number = match.group()
      7         start_position = match.start()

NameError: name 'matches' is not defined
```

```
question 22 Write a regular expression in python program to extract maximum/largest
numeric
value from a string.
```

In [70]:
```
import re

def extract_maximum_numeric_value(input_string):
    pattern = re.compile(r'\b\d+\b')
    numeric_values = pattern.findall(input_string)
if numeric_values:
        max_numeric_value = max(map(int, numeric_values))
        return max_numeric_value
    else:
        return None
    sample_text = 'My marks in each semester are: 947, 896, 926, 524, 734, 950, 642'

maximum_numeric_value = extract_maximum_numeric_value(sample_text)

print("Sample Text:", sample_text)
print("Maximum Numeric Value:", maximum_numeric_value)
```

```
  File <tokenize>:9
    else:return None
    ^
IndentationError: unindent does not match any outer indentation level
```

```
question 23
```

In [ ]:
```
Create a funcθon in python to insert spaces between words starθng with capital leΣers
Sample Text: "RegularExpressionIsAnImportantTopicInPython"
```

In [71]:
```
import re

def insert_spaces_between_capital_words(input_string):
    words = re.findall(r'[A-Z][a-z]*', input_string)
    spaced_string = ' '.join(words)

    return spaced_string
sample_text = "RegularExpressionIsAnImportantTopicInPython"

result = insert_spaces_between_capital_words(sample_text)

print("Sample Text:", sample_text)
print("Result after inserting spaces:", result)
```

```
Sample Text: RegularExpressionIsAnImportantTopicInPython
Result after inserting spaces: Regular Expression Is An Important Topic In Python
```

In [ ]:
```
question 24
```

In [72]:
```
import re

def find_sequences(input_string):
    pattern = re.compile(r'([A-Z][a-z]+)')

    sequences = pattern.findall(input_string)

    return sequences
sample_text = "Find Sequences Like This One In Python"

result = find_sequences(sample_text)

print("Sample Text:", sample_text)
print("Found Sequences:", result)
```

```
Sample Text: Find Sequences Like This One In Python
Found Sequences: ['Find', 'Sequences', 'Like', 'This', 'One', 'In', 'Python']
```

questions 25

In [77]:
```
import re

def remove_continuous_duplicates(sentence):
    pattern = re.compile(r'\b(\w+)(?:\s+\1\b)+', flags=re.IGNORECASE)
    result = pattern.sub(r'\1', sentence)
    return result
sample_text = "Hello hello world world"
result = remove_continuous_duplicates(sample_text)
print("Sample Text:", sample_text)
print("Result after removing continuous duplicates:", result)
```

```
Sample Text: Hello hello world world
Result after removing continuous duplicates: Hello world
```

questions 26

In [86]:
```
import re
def is_string_ending_with_alphanumeric(input_string):
    pattern = re.compile(r'\w$')
    match = pattern.search(input_string)
    return bool(match)
    sample_string1 = "Hello123"
    sample_string2 = "Python!"
    sample_string3 = "12345"
    print(f"Is '{sample_string1}' ending with an alphanumeric character? {is_string_e
    print(f"Is '{sample_string2}' ending with an alphanumeric character? {is_string_e
    print(f"Is '{sample_string3}' ending with an alphanumeric character? {is_string_e
```

questions 27

In [87]:
```
import re

def extract_hashtags(input_text):
    pattern = re.compile(r'#\w+')
    hashtags = pattern.findall(input_text)

    return hashtags
sample_text = """RT @kapil_kausik: #Dol0wal I mean #xyzabc is "hurt" by #Demone0za0on
has rendered USELESS <ed><U+00A0><U+00BD><ed><U+00B1><U+0089> "acquired funds" No wo""

hashtags_found = extract_hashtags(sample_text)

print("Sample Text:", sample_text)
print("Extracted Hashtags:", hashtags_found)
```

```
Sample Text: RT @kapil_kausik: #Dol0wal I mean #xyzabc is "hurt" by #Demone0za0on as
the same
has rendered USELESS <ed><U+00A0><U+00BD><ed><U+00B1><U+0089> "acquired funds" No wo
Extracted Hashtags: ['#Dol0wal', '#xyzabc', '#Demone0za0on']
```

questions 28

In [95]:
```
import re
def remove_unicode_symbols(input_text):
    pattern = re.compile(r'<U\+\w+>')
result = pattern.sub('', input_text)
return result
sample_text = "@Jags123456 Bharat band on 28??<ed><U+00A0><U+00BD><ed><U+00B8><U+0082:
cleaned_text = remove_unicode_symbols(sample_text)
print("Sample Text:", sample_text)
print("Text after removing Unicode symbols:", cleaned_text)
```

```
  Cell In[95], line 4
    result = pattern.sub('', input_text) return result
                                          ^
SyntaxError: invalid syntax
```

questions 29

In [98]:
```
import re

def extract_dates_from_text(text):
    pattern = re.compile(r'\b\d{2}-\d{2
    dates = pattern.findall(text)

    return dates
    sample_text = "Ron was born on 12-09-1992 and he was admiΣed to school 15-12-1999
extracted_dates = extract_dates_from_text(sample_text)

print("Sample Text:", sample_text)
print("Extracted Dates:", extracted_dates)
```

```
  Cell In[98], line 4
    pattern = re.compile(r'\b\d{2}-\d{2
                         ^
SyntaxError: unterminated string literal (detected at line 4)
```

In [ ]:

In [ ]:
```
questions 30
```

In [99]:
```
import re

def remove_words_between_lengths(text):
    pattern = re.compile(r'\b\w{2,4}\b')
    result = pattern.sub('', text)

    return result
sample_text = "The following example creates an ArrayList with a capacity of 50 elemen
modified_text = remove_words_between_lengths(sample_text)

print("Sample Text:", sample_text)
print("Text after removing words of length 2 to 4:", modified_text)
```

```
Sample Text: The following example creates an ArrayList with a capacity of 50 elemen
ts. 4 elements are then added to the ArrayList and the ArrayList is trimmed accordin
gly.
Text after removing words of length 2 to 4:  following example creates  ArrayList  a
capacity   elements. 4 elements   added   ArrayList   ArrayList  trimmed accordingl
y.
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: