ASSIGNMENT ON REGULAR EXPRESSION 1-Copy1

August 8, 2023

[ ]:
```
ANSWE 1-  Here's a Python program that replaces all occurrences of a space,␣
 ↪comma, or dot with a colon in a given text:
 def replace_chars_with_colon(text):
    replacements = {' ': ':', ',': ':', '.': ':'}

    for char, replacement in replacements.items():
        text = text.replace(char, replacement)

    return text

sample_text = 'Python Exercises, PHP exercises.'
output = replace_chars_with_colon(sample_text)
print(output)

When you run this program, it will produce the expected output:

ruby

Python:Exercises::PHP:exercises:
```

[ ]:
```
ANSWER-2  a Python program that finds all words starting with either 'a' or 'e'␣
 ↪in a given string:

code-
import re

def find_words_with_a_or_e(input_string):
    # Split the input string into words using whitespace as a delimiter
    words = input_string.split()

    # Use a regular expression to find words starting with 'a' or 'e'
    pattern = r'\b[aeAE]\w*\b'
    matching_words = re.findall(pattern, input_string)

    return matching_words

# Get input from the user
```

```
input_string = input("Enter a string: ")

# Find words starting with 'a' or 'e'
result = find_words_with_a_or_e(input_string)

# Print the result
print("Words starting with 'a' or 'e':")
for word in result:
    print(word)
 code into a Python interpreter or script, and it will prompt you to enter a␣
 ↪string. After entering the string, it will display a list of words that␣
 ↪start with either 'a' or '
```

[ ]:
```
ANSWER-3 Python function that uses the re.compile() method to find all words␣
 ↪that are at least 4 characters long in a given string:

code-

import re

def find_long_words(input_string):
    pattern = re.compile(r'\b\w{4,}\b')
    long_words = pattern.findall(input_string)
    return long_words

# Example usage:
input_string = "This is a sample string with some long words like apple,␣
 ↪banana, and orange."
result = find_long_words(input_string)
print(result)
In this function, the regular expression pattern \b\w{4,}\b is used to match␣
 ↪words that are at least 4 characters long. Here's a breakdown of the pattern:

\b : Word boundary anchor to ensure that we match complete words.
\w{4,} : Matches any word character (letters, digits, or underscores) repeated␣
 ↪at least 4 times.
\b : Another word boundary anchor to complete the word match.
The findall() method of the compiled regex object returns a list of all matches␣
 ↪in the input string. The example usage demonstrates how to call the function␣
 ↪and print the resu
```

[ ]:
```
ANSWER4-
  a Python function that uses the re.compile() method to find all three, four,␣
 ↪and five character words in a given string:

code-
import re
```

```
def find_words_of_length(input_string, min_length, max_length):
    pattern = re.compile(r'\b\w{' + str(min_length) + ',' + str(max_length) +␣
 ↪r'}\b')
    words = pattern.findall(input_string)
    return words

input_string = "This is a sample string with some words of varying lengths."
min_length = 3
max_length = 5
result = find_words_of_length(input_string, min_length, max_length)
print(result)
In this function, the min_length and max_length parameters determine the␣
 ↪minimum and maximum word lengths to search for. The re.compile() method is␣
 ↪used to create a regular expression pattern that matches words within the␣
 ↪specified length range (\w{min_length,max_length}). The \b at the beginning␣
 ↪and end of the pattern ensure that the pattern matches whole words. The␣
 ↪findall() method is then used to find all occurrences of this pattern in the␣
 ↪input string, resulting in a list of words that meet the specified length␣
 ↪criteria.
```

[ ]:

```
    ANSWER-5

Python function that uses the re.compile() method to find all three, four, and␣
 ↪five character words in a given string:


import re

def find_words_of_length(input_string, min_length, max_length):
    pattern = re.compile(r'\b\w{' + str(min_length) + ',' + str(max_length) +␣
 ↪r'}\b')
    words = pattern.findall(input_string)
    return words

input_string = "This is a sample string with some words of varying lengths."
min_length = 3
max_length = 5
result = find_words_of_length(input_string, min_length, max_length)
print(result)
```

```
In this function, the min_length and max_length parameters determine the␣
 ↪minimum and maximum word lengths to search for. The re.compile() method is␣
 ↪used to create a regular expression pattern that matches words within the␣
 ↪specified length range (\w{min_length,max_length}). The \b at the beginning␣
 ↪and end of the pattern ensure that the pattern matches whole words. The␣
 ↪findall() method is then used to find all occurrences of this pattern in the␣
 ↪input string, resulting in a list of words that meet the specified length␣
 ↪criteria
```

[ ]:
```
ANSWER-6
Python program that reads the sample text from a text file, removes the text␣
 ↪within parentheses using regular expressions, and then writes the modified␣
 ↪text to an output file:

import re

# Read the input text file
with open('input.txt', 'r') as file:
    input_text = file.read()

# Define a regular expression pattern to match text within parentheses
pattern = r'\s*\([^)]*\)'

# Use regular expression to remove text within parentheses
output_text = re.sub(pattern, '', input_text)

# Write the modified text to an output file
with open('output.txt', 'w') as file:
    file.write(output_text)

print("Text without parentheses area has been written to 'output.txt'")
Before running this program, make sure you have created a text file named␣
 ↪'input.txt' with the sample text you provided:

scss

example (.com)
hr@fliprobo (.com)
github (.com)
Hello (Data Science World)
Data (Scientist)
After running the program, the modified text without parentheses area will be␣
 ↪written to an 'output.txt' file:

css
example
```

```
hr@fliprobo
github
Hello
Data
Remember to place the 'input.txt' file in the same directory as the Python␣
 ↪program before running
```

[ ]:
```
ANSWER-7

Python to split a string into uppercase letters using a regular expression.␣
 ↪Here's how you can achieve the expected output:
code-

import re

# Sample text
sample_text = "ImportanceOfRegularExpressionsInPython"

# Define the regular expression pattern to match uppercase letters
pattern = r'(?<=[a-z])(?=[A-Z])'

# Split the string using the pattern
result = re.split(pattern, sample_text)

print(result)
Output:

css
['Importance', 'Of', 'Regular', 'Expressions', 'In', 'Python']
In this regular expression pattern, (?<=[a-z]) is a positive lookbehind␣
 ↪assertion that checks for a lowercase letter before an uppercase letter, and␣
 ↪(?=[A-Z]) is a positive lookahead assertion that checks for an uppercase␣
 ↪letter after a lowercase letter. This effectively splits the string at the␣
 ↪boundary between lowercase and uppercase letters.
```

[ ]:
```
ANSWER-8

Sure, here's a Python function that accomplishes this task:

import re

def insert_spaces(text):
    # Use regular expression to find words starting with numbers
    pattern = r'(?<=\D)(?=\d)|(?<=\d)(?=\D)'
    spaced_text = re.sub(pattern, ' ', text)
    return spaced_text
```

```
sample_text = "RegularExpression1IsAn2ImportantTopic3InPython"
output = insert_spaces(sample_text)
print(output)
This function uses the re module to apply a regular expression pattern that␣
 ↪inserts a space between a non-digit character and a digit character, or␣
 ↪between a digit character and a non-digit character. It then returns the␣
 ↪modified text with the spaces inserted as required.

When you run the code with the provided sample text, it will produce the␣
 ↪expected outpu
RegularExpression1IsAn 2ImportantTopic 3InPython
```

[ ]:
```
ANSWER-9

Sure, here's a Python function that accomplishes this task

import re

def insert_spaces(text):
    # Use regular expression to find words starting with numbers
    pattern = r'(?<=\D)(?=\d)|(?<=\d)(?=\D)'
    spaced_text = re.sub(pattern, ' ', text)
    return spaced_text

sample_text = "RegularExpression1IsAn2ImportantTopic3InPython"
output = insert_spaces(sample_text)
print(output)
This function uses the re module to apply a regular expression pattern that␣
 ↪inserts a space between a non-digit character and a digit character, or␣
 ↪between a digit character and a non-digit character. It then returns the␣
 ↪modified text with the spaces inserted as required.

When you run the code with the provided sample text, it will produce the␣
 ↪expected outpu
RegularExpression1IsAn 2ImportantTopic 3InPython
```

[ ]:
```
ANSWER-10
  Python program that reads the text from a file, uses regular expressions to␣
 ↪extract email addresses, and then prints the extracted email addresses:

import re

def extract_emails_from_text(text):
    pattern = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,7}\b'
    emails = re.findall(pattern, text)
    return emails
```

```
def main():
    file_path = 'sample_text.txt'  # Update this with the actual path to your␣
 ↪text file

    with open(file_path, 'r') as file:
        text = file.read()

    email_list = extract_emails_from_text(text)

    unique_domains = set()
    domain_email_map = {}

    for email in email_list:
        domain = email.split('@')[1]
        unique_domains.add(domain)
        if domain in domain_email_map:
            domain_email_map[domain].append(email)
        else:
            domain_email_map[domain] = [email]

    for domain, emails in domain_email_map.items():
        print(emails)

if __name__ == '__main__':
    main()
Make sure to replace 'sample_text.txt' with the actual path to your text file␣
 ↪containing the sample text. This program reads the text from the file,␣
 ↪extracts email addresses using a regular expression, and then groups them by␣
 ↪domain and prints the results
```

[ ]:
```
ANSWER-11

Certainly! Here's a Python program that uses regular expressions to match a␣
 ↪string containing only upper and lowercase letters, numbers, and underscores:


import re

def is_valid_string(input_string):
    pattern = r'^[A-Za-z0-9_]*$'
    return re.match(pattern, input_string) is not None

def main():
    test_strings = ["Hello123", "Good_Morning", "123_abc", "abc!def",␣
 ↪"Special-String"]

    for string in test_strings:
```

```
        if is_valid_string(string):
            print(f'"{string}" is a valid string.')
        else:
            print(f'"{string}" is not a valid string.')

if __name__ == '__main__':
    main()
This program defines a function is_valid_string() that uses a regular␣
 ↪expression pattern to check if a given string contains only upper and␣
 ↪lowercase letters, numbers, and underscores. The main() function tests the␣
 ↪is_valid_string() function with a list of test strings and prints whether␣
 ↪each string is valid or not
```

[ ]:
```
ANSWER-12

Sure, here's a Python program that checks if a given string starts with a␣
 ↪specific number:


def starts_with_number(input_string, specific_number):
    return input_string.startswith(specific_number)

def main():
    specific_number = "123"
    test_strings = ["123abc", "456xyz", "789def", "abc123", "012345"]

    for string in test_strings:
        if starts_with_number(string, specific_number):
            print(f'"{string}" starts with {specific_number}.')
        else:
            print(f'"{string}" does not start with {specific_number}.')

if __name__ == '__main__':
    main()
In this program, the starts_with_number() function takes an input string and a␣
 ↪specific number as arguments and checks if the input string starts with the␣
 ↪specific number using the startswith() method. The main() function tests the␣
 ↪starts_with_number() function with a list of test strings and prints whether␣
 ↪each string starts with the specific number or not.

You can modify the value of specific_number to the desired number you want to␣
 ↪check for at the beginning of the string.
```

[ ]:
```
ANSWER- 13
Python program that removes leading zeros from an IP address
def remove_leading_zeros(ip_address):
    # Split the IP address into its four parts
```

```
    parts = ip_address.split('.')

    # Remove leading zeros from each part
    normalized_parts = [str(int(part)) for part in parts]

    # Join the normalized parts back into an IP address
    normalized_ip = '.'.join(normalized_parts)

    return normalized_ip

# Example usage
ip_address_with_zeros = "192.168.001.001"
normalized_ip = remove_leading_zeros(ip_address_with_zeros)
print("Original IP:", ip_address_with_zeros)
print("Normalized IP:", normalized_ip)
In this program, the remove_leading_zeros function takes an IP address as␣
 ↪input, splits it into its four parts (octets), converts each part to an␣
 ↪integer to remove leading zeros, and then joins the normalized parts back␣
 ↪together to form the normalized IP address. The example usage demonstrates␣
 ↪how to use the function to remove leading zeros from an IP add
```

[ ]:
```
ANSWER-14
 here's a Python code snippet that uses regular expressions to extract the date␣
 ↪string in the specified format from a text file:


import re

# Read the text from the file
with open('sample_text.txt', 'r') as file:
    text = file.read()

# Define a regular expression pattern
pattern = r'\b(?:
 ↪January|February|March|April|May|June|July|August|September|October|November|December)\s\d{
 ↪:st|nd|rd|th)?\s\d{4}\b'

# Search for the pattern in the text
match = re.search(pattern, text)

# If a match is found, print the matched date string
if match:
    date_string = match.group()
    print(date_string)
else:
    print("No date string matching the pattern was found.")
```

```
Make sure to replace 'sample_text.txt' with the actual path to your text file␣
 ↪containing the sample text. The provided code will extract and print the␣
 ↪date string in the specified format from the text file.
```

[ ]:
```
ANSWER-15
You can use the in operator to search for specific literal strings within␣
 ↪another string. Here's a Python program that demonstrates how to search for␣
 ↪the given words in a sample text:

def search_literals(sample_text, searched_words):
    found_words = []
    for word in searched_words:
        if word in sample_text:
            found_words.append(word)
    return found_words

sample_text = 'The quick brown fox jumps over the lazy dog.'
searched_words = ['fox', 'dog', 'horse']

found_words = search_literals(sample_text, searched_words)

if found_words:
    print(f"Found words: {', '.join(found_words)}")
else:
    print("No words found.")
Copy and paste this code into a Python interpreter or a script file, and it␣
 ↪will output:


Found words: fox, dog
The program iterates through the searched_words list and checks if each word␣
 ↪exists within the sample_text using the in operator. It collects the found␣
 ↪words in the found_words list and then prints the results.
```

[ ]:
```
ANSWER-16
Certainly! You can modify the previous program to also find and display the␣
 ↪locations where the searched words occur in the original string. Here's the␣
 ↪updated Python program:

def search_literals_with_location(sample_text, searched_words):
    found_words = []
    for word in searched_words:
        if word in sample_text:
            location = sample_text.index(word)
            found_words.append((word, location))
    return found_words
```

```
sample_text = 'The quick brown fox jumps over the lazy dog.'
searched_words = ['fox']

found_words_with_location = search_literals_with_location(sample_text,␣
 ↪searched_words)

if found_words_with_location:
    for word, location in found_words_with_location:
        print(f"Found word '{word}' at index {location}")
else:
    print("No words found.")
Copy and paste this code into a Python interpreter or a script file, and it␣
 ↪will output:


Found word 'fox' at index 16
The program iterates through the searched_words list and checks if each word␣
 ↪exists within the sample_text using the in operator. If a word is found, it␣
 ↪uses the index method to determine the location where the word occurs within␣
 ↪the original string and collects the word along with its location in the␣
 ↪found_words_with_location list. Finally, it prints the results.
```

[ ]:
```
ANSWER-17
You can use the find() method to locate the occurrences of a substring within a␣
 ↪given string. Here's a Python program that finds all occurrences of a␣
 ↪pattern (substring) within a sample text:


def find_substrings(sample_text, pattern):
    start = 0
    found_indices = []

    while start < len(sample_text):
        index = sample_text.find(pattern, start)
        if index == -1:
            break
        found_indices.append(index)
        start = index + len(pattern)

    return found_indices

sample_text = 'Python exercises, PHP exercises, C# exercises'
pattern = 'exercises'

found_indices = find_substrings(sample_text, pattern)

if found_indices:
```

```
    print(f"Pattern '{pattern}' found at indices: {', '.join(map(str,␣
 ↪found_indices))}")
else:
    print("Pattern not found.")
Copy and paste this code into a Python interpreter or a script file, and it␣
 ↪will output:

sql

Pattern 'exercises' found at indices: 7, 23, 38
In this program, the find_substrings function uses a while loop to iteratively␣
 ↪find the occurrences of the pattern within the sample_text. The find()␣
 ↪method is used to locate each occurrence, and the loop continues searching␣
 ↪for the pattern by updating the start index. The found indices are collected␣
 ↪in the found_indices list, and the final result is print
```

[ ]:
```
ANSWER-18
Here's a Python program that finds the occurrence and position of substrings␣
 ↪within a given string:
def find_substring_occurrences(main_string, substring):
    occurrences = []
    start = 0

    while start < len(main_string):
        index = main_string.find(substring, start)
        if index == -1:
            break

        occurrences.append((index, index + len(substring) - 1))
        start = index + len(substring)

    return occurrences

def main():
    main_string = input("Enter the main string: ")
    substring = input("Enter the substring to find: ")

    occurrences = find_substring_occurrences(main_string, substring)

    if occurrences:
        print(f"Substring '{substring}' found at positions:")
        for start, end in occurrences:
            print(f"Start: {start}, End: {end}")
    else:
        print("Substring not found in the main string.")

if __name__ == "__main__":
```

```
    main(). The program will prompt you to enter the main string and the␣
 ↪substring you want to find. It will then display the positions (start and␣
 ↪end indices) of the found substrings within the main string.
```

[ ]:
```
ANSWER-19
Here's a Python program that converts a date from yyyy-mm-dd format to␣
 ↪dd-mm-yyyy format:


def convert_date_format(date_str):
    parts = date_str.split('-')

    if len(parts) != 3:
        return "Invalid date format"

    year = parts[0]
    month = parts[1]
    day = parts[2]

    new_date_format = f"{day}-{month}-{year}"
    return new_date_format

def main():
    date_str = input("Enter a date in yyyy-mm-dd format: ")

    new_date_format = convert_date_format(date_str)

    print(f"Converted date: {new_date_format}")

if __name__ == "__main__":
    main()
 The program will prompt you to enter a date in yyyy-mm-dd format, and it will␣
 ↪then display the date in the dd-mm-yyyy format.
```

[ ]:
```
ANSWER-20

Sure, here's a Python function that uses the re.compile() method to find all␣
 ↪decimal numbers with a precision of 1 or 2 in a given string:

import re

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

def main():
```

```
    input_text = input("Enter a string: ")

    decimal_numbers = find_decimal_numbers(input_text)

    if decimal_numbers:
        print("Decimal numbers found with precision of 1 or 2:")
        for num in decimal_numbers:
            print(num)
    else:
        print("No decimal numbers with precision of 1 or 2 found in the string.
 ↪")

if __name__ == "__main__":
    main()
. The program will prompt you to enter a string, and it will then display any␣
 ↪decimal numbers with a precision of 1 or 2 that are found in the input␣
 ↪string.
```

[ ]:
```
ANSWER-21
Here's a Python program that separates and prints the numbers along with their␣
 ↪positions in a given string:

def separate_numbers_and_positions(text):
    numbers = []

    for index, char in enumerate(text):
        if char.isdigit():
            numbers.append((char, index))

    return numbers

def main():
    input_text = input("Enter a string: ")

    numbers_and_positions = separate_numbers_and_positions(input_text)

    if numbers_and_positions:
        print("Numbers and their positions in the string:")
        for number, position in numbers_and_positions:
            print(f"Number: {number}, Position: {position}")
    else:
        print("No numbers found in the string.")

if __name__ == "__main__":
    main()
The program will prompt you to enter a string, and it will then display the␣
 ↪numbers and their positions in the input string.
```

[ ]:
```
ANSWER-22
You can use the re module in Python to achieve this. Here's a Python program␣
 ↪with a regular expression to extract the maximum/largest numeric value from␣
 ↪a given string:


import re

def extract_maximum_numeric_value(text):
    numbers = re.findall(r'\b\d+\b', text)

    if numbers:
        max_number = max(map(int, numbers))
        return max_number
    else:
        return None

def main():
    sample_text = 'My marks in each semester are: 947, 896, 926, 524, 734, 950,␣
 ↪642'

    max_numeric_value = extract_maximum_numeric_value(sample_text)

    if max_numeric_value is not None:
        print(f"Maximum numeric value: {max_numeric_value}")
    else:
        print("No numeric values found in the text.")

if __name__ == "__main__":
    main()
Copy and paste this code into a Python file and run it. The program will use␣
 ↪the regular expression to extract the maximum numeric value from the sample␣
 ↪text and display the expected output.
```

[ ]:
```
ANSWER-23

 Python function that inserts spaces between words starting with capital␣
 ↪letters:

import re

def insert_spaces_between_capital_words(text):
    spaced_text = re.sub(r'(?<=.)([A-Z][a-z]+)', r' \1', text)
    return spaced_text

def main():
    sample_text = "RegularExpressionIsAnImportantTopicInPython"
```

```
    spaced_text = insert_spaces_between_capital_words(sample_text)
    print("Original Text:", sample_text)
    print("Processed Text:", spaced_text)

if __name__ == "__main__":
    main()
Copy and paste this code into a Python file and run it. The␣
 ↪insert_spaces_between_capital_words function uses a regular expression to␣
 ↪insert spaces between words starting with capital letters. The main function␣
 ↪demonstrates the usage of this function on the sample text, and it displays␣
 ↪both the original and processed texts as outp
```

[ ]:
```
ANSWER-24
You can use the following regular expression in Python to find sequences of one␣
 ↪uppercase letter followed by lowercase letters:


import re

pattern = r'[A-Z][a-z]+'

text = "RegularExpressionIsAnImportantTopicInPython"
sequences = re.findall(pattern, text)

print("Sequences of one uppercase letter followed by lowercase letters:")
for sequence in sequences:
    print a Python file and run it. It will find and display all sequences of␣
 ↪one uppercase letter followed by lowercase letters in the given text.
```

[ ]:
```
ANSWER-25
Here's a Python program that uses regular expressions to remove continuous␣
 ↪duplicate words from a sentence:


import re

def remove_continuous_duplicates(sentence):
    pattern = r'\b(\w+)(?:\s+\1)+\b'
    cleaned_sentence = re.sub(pattern, r'\1', sentence, flags=re.IGNORECASE)
    return cleaned_sentence

def main():
    sample_text = "Hello hello world world"
    cleaned_text = remove_continuous_duplicates(sample_text)

    print("Original Text:", sample_text)
    print("Processed Text:", cleaned_text)
```

```
if __name__ == "__main__":
    main()
Copy and paste this code into a Python file and run it. The␣
 ↪remove_continuous_duplicates function uses a regular expression to remove␣
 ↪continuous duplicate words from the input sentence. The main function␣
 ↪demonstrates the usage of this function on the sample text and displays both␣
 ↪the original and processed texts as output.
```

[ ]:
```
ANSWER-26
Here's a Python program that uses a regular expression to accept strings that␣
 ↪end with an alphanumeric character:


import re

def check_ending_with_alphanumeric(text):
    pattern = r'^.*[a-zA-Z0-9]$'
    return re.match(pattern, text) is not None

def main():
    input_string = input("Enter a string: ")

    if check_ending_with_alphanumeric(input_string):
        print("String ends with an alphanumeric character.")
    else:
        print("String does not end with an alphanumeric character.")

if __name__ == "__main__":
    main()
    The program will prompt you to enter a string, and it will then check␣
 ↪whether the string ends with an alphanumeric character or not. It will␣
 ↪display the appropriate message based on the result.
```

[ ]:
```
ANSWER-27
Python program that uses regular expressions (RegEx) to extract hashtags from␣
 ↪the given sample text:

import re

sample_text = """RT @kapil_kausik: #Doltiwal I mean #xyzabc is "hurt" by␣
 ↪#Demonetization as the same has rendered USELESS␣
 ↪<ed><U+00A0><U+00BD><ed><U+00B1><U+0089> "acquired funds" No wo"""

hashtags = re.findall(r'#\w+', sample_text)

print("Expected Output:", hashtags)
```

```
When you run this program, it will extract the hashtags from the sample text␣
 ↪and print the expected output:

less

Expected Output: ['#Doltiwal', '#xyzabc', '#Demonetization
```

[ ]:
```
ANSWER-28 Python program that effectively uses regular expressions to remove␣
 ↪occurrences of <U+..>-like symbols from a given text. The provided code is␣
 ↪correct and will work as intended.

Just to provide a bit more explanation:

import re: This imports the Python regular expression module, allowing you to␣
 ↪work with regular expressions.

sample_text: This is the input text that contains the <U+..>-like symbols you␣
 ↪want to remove.

cleaned_text = re.sub(r'<U\+[0-9A-Fa-f]+>', '', sample_text): This line uses␣
 ↪the re.sub() function to replace all occurrences of the pattern <U+..>␣
 ↪(where .. represents one or more hexadecimal digits) with an empty string ''.
 ↪ Here's a breakdown of the regular expression pattern:

r'<U\+[0-9A-Fa-f]+>': This pattern matches strings starting with <U+, followed␣
 ↪by one or more hexadecimal digits (0-9, A-F, a-f), and ending with >. This␣
 ↪corresponds to the <U+..>-like symbols you want to remove.
print("Cleaned Text:", cleaned_text): Finally, the cleaned text (with <U+..
 ↪>-like symbols removed) is printed to the console.

Running this program with the provided sample text will result in the following␣
 ↪output:

sql

Cleaned Text: Check the below sample text, there are strange symbols something␣
 ↪of the sort  all over the place.
This output demonstrates that the program successfully removed the <U+..>-like␣
 ↪symbols from the input text.
```

[ ]:
```
ANSWER-29

 To remove all occurrences of <U+..>-like symbols from a text using regular␣
 ↪expressions, you can use the following Python program:
```

```
import re

sample_text = 'Check the below sample text, there are strange symbols something␣
 ↪of the sort <U+00A0><U+00BD><U+1234> all over the place.'

cleaned_text = re.sub(r'<U\+[0-9A-Fa-f]+>', '', sample_text)

print("Cleaned Text:", cleaned_text)
In this program, the re.sub() function is used to replace all occurrences of␣
 ↪the <U+..>-like symbols with an empty string, effectively removing them from␣
 ↪the text. The regular expression r'<U\+[0-9A-Fa-f]+>' matches the pattern␣
 ↪<U+ followed by one or more hexadecimal digits, and re.sub() performs the␣
 ↪substitution.

When you run this program, it will output the cleaned text with the <U+..>-like␣
 ↪symbols removed.


Absolutely, you've provided a Python program that effectively uses regular␣
 ↪expressions to remove occurrences of <U+..>-like symbols from a given text.␣
 ↪The provided code is correct and will work as intended.

Just to provide a bit more explanation:

import re: This imports the Python regular expression module, allowing you to␣
 ↪work with regular expressions.

sample_text: This is the input text that contains the <U+..>-like symbols you␣
 ↪want to remove.

cleaned_text = re.sub(r'<U\+[0-9A-Fa-f]+>', '', sample_text): This line uses␣
 ↪the re.sub() function to replace all occurrences of the pattern <U+..>␣
 ↪(where .. represents one or more hexadecimal digits) with an empty string ''.
 ↪ Here's a breakdown of the regular expression pattern:

r'<U\+[0-9A-Fa-f]+>': This pattern matches strings starting with <U+, followed␣
 ↪by one or more hexadecimal digits (0-9, A-F, a-f), and ending with >. This␣
 ↪corresponds to the <U+..>-like symbols you want to remove.
print("Cleaned Text:", cleaned_text): Finally, the cleaned text (with <U+..
 ↪>-like symbols removed) is printed to the console.

Running this program with the provided sample text will result in the following␣
 ↪output:

sql
```

```
Cleaned Text: Check the below sample text, there are strange symbols something␣
 ↪of the sort  all over the place.
This output demonstrates that the program successfully removed the <U+..>-like␣
 ↪symbols from the input text.




.

 here's a Python program that reads the sample text from a text file and␣
 ↪extracts dates using regular expressions:

python

import re

# Read the sample text from the file
with open('sample_text.txt', 'r') as file:
    sample_text = file.read()

# Define a regular expression pattern to match dates in the format DD-MM-YYYY
date_pattern = r'\d{2}-\d{2}-\d{4}'

# Find all dates using the regular expression pattern
dates = re.findall(date_pattern, sample_text)

# Print the extracted dates
for date in dates:
    print("Extracted Date:", date)
Make sure to store the sample text in a file named sample_text.txt in the same␣
 ↪directory as the Python script. The program reads the text from the file,␣
 ↪uses a regular expression to extract dates in the format DD-MM-YYYY, and␣
 ↪then prints the extracted dates.

For the given sample text:

vbnet

Ron was born on 12-09-1992 and he was admitted to school 15-12-1999.
The output will be:

javascript

Extracted Date: 12-09-1992
Extracted Date: 15-12-1999
```

```
This program will effectively extract dates from the given text and display␣
 ↪them as output
```

[ ]:
```
ANSWER-30
a Python function that uses the re.compile() method to remove words from a␣
 ↪string of length between 2 and 4:

import re

def remove_words_of_length_between_2_and_4(input_string):
    pattern = re.compile(r'\b\w{2,4}\b')
    return pattern.sub('', input_string)

sample_text = "The following example creates an ArrayList with a capacity of 50␣
 ↪elements. 4 elements are then added to the ArrayList and the ArrayList is␣
 ↪trimmed accordingly."
result = remove_words_of_length_between_2_and_4(sample_text)
print("Expected Output:", result)
In this program, the function remove_words_of_length_between_2_and_4() takes an␣
 ↪input string and uses the re.compile() method to create a regular expression␣
 ↪pattern r'\b\w{2,4}\b'. This pattern matches words that have a length␣
 ↪between 2 and 4 characters. The \b specifies a word boundary, and \w{2,4}␣
 ↪matches 2 to 4 word characters (letters, digits, or underscores).

The pattern.sub('', input_string) line substitutes all occurrences of the␣
 ↪matched words with an empty string in the input string.

For the given sample text:

vbnet

The following example creates an ArrayList with a capacity of 50 elements. 4␣
 ↪elements are then added to the ArrayList and the ArrayList is trimmed␣
 ↪accordingly.
The expected output will be:

vbnet

Expected Output: following example creates ArrayList a capacity elements. 4␣
 ↪elements are then added to the ArrayList and the ArrayList is trimmed␣
 ↪accordingly.
This program will remove words of length between 2 and 4 characters from the␣
 ↪input string and provide the desired output.
```