################ Regular Expression ################

# Regular Expression is used to clean the data using patterns

import re  # Import the 're' module for regular expressions

# Search for the pattern "cat" in the given string

x = re.search("cat", "A cat and a rat can't be friends.")

print(x)  # Print the result of the search operation

# Search for the pattern "cow" in the given string

x = re.search("cow", "A cat and a rat can't be friends.")

print(x)  # Print the result of the search operation

# In the previous example we had to import the module re to be able to work

# with regular expressions.

# Then we used the method search from the re module. This is most probably the most important

# and the most often used method of this module. re.search(expr,s) checks a string s

# for an occurrence of a substring which matches the regular expression expr.

# Let's assume that we have not been interested in the previous example to

# recognize the word cat, but all three letter words, which end with "at".

# The syntax of regular expressions supplies a metacharacter ".",

# which is used like a placeholder for "any character".

# Here dot is placeholder which means any character can come.

r" .at "

# we are finding the word pattern 'at' by using the search function

x = re.search(r" .at ", "A cat and a rat can't be friends.")

# printing the variable x

print(x)

# This regular expression matches three letter words, isolated by blanks, which end in "at".

# Now we get words like "rat", "cat", "bat", "eat", "sat" and many others.

# But what if the text contains "words" like "@at" or "3at"?here also getting output

# Square brackets, "[" and "]", are used to include a character class.

# [xyz] means e.g. either an "x", an "y" or a "z".

r"M[ae][iy]er"

# Maier,Mayer,Meier,Meyer

# This is a regular expression, which matches a surname with four different

# spellings: Maier, Mayer, Meier, Meyer.

# We might need e.g. a class of letters between "a" and "e" or between "0" and "5".

# To manage such character classes, the syntax of regular expressions supplies

# a metacharacter "-".

# [a-e] a simplified writing for [abcde] or [0-5] denotes [012345].

# [ABCDEFGHIJKLMNOPQRSTUVWXYZ] we can write [A-Z].

# "any lower case or uppercase letter" [A-Za-z]

# Another Metacharacter is "^" which can be used inside square brackets

# If it is used directly after an opening sqare bracket, it negates the choice.

# If it is not positioned as the first character following the opening square

# bracket, it has no special meaning.

# Similarly "-" also if positioned at first means match with "-".

import re  # Import the 're' module for regular expressions

# Define the input text

txt = "The rain in Spain"

# Use re.findall() to find all occurrences of the pattern "ai" in the text

x = re.findall("in", txt)

# Print the result

print(x)

['in', 'in', 'in']#output

# The special sequences consist of "\\" and a character from the following list:

# \d  Matches any decimal digit; equivalent to the set [0-9].

# \D  is not of \d. It matches any non-digit character; equivalent to the set [^0-9].

# \s  Matches any whitespace character; equivalent to [ \t\n\r\f\v].

# \S  The complement of \s. It matches any non-whitespace character; equiv. to [^ \t\n\r\f\v].

# \w  Matches any alphanumeric character; equivalent to [a-zA-Z0-9\_]. With LOCALE, it will match the set [a-zA-Z0-9\_] plus characters defined as letters for the current locale.

# \W  Matches the complement of \w.

# \b  Matches the empty string, but only at the start or end of a word.

# \B  Matches the empty string, but not at the start or end of a word.

# \\  Matches a literal backslash.

# Quantifiers

Quantifiers in Python's regular expressions are used to specify how many times a particular character, group of characters, or character class must occur for a match to be found

# A quantifier after a token, which can be a single character or group in brackets

# Specifies how often that preceding element is allowed to occur.

# The most common quantifiers are the question mark (?).

# The asterisk or star character \*, and the plus sign +.

########################################################

# \*              #         Match zero or more times.   #

# +              #         Match one or more times.    #

# ?              #         Match zero or one time.     #

# { n }          #         Match exactly n times.      #

# { n ,}         #         Match at least n times.     #

# { n , m }      #         Match from n to m times.    #

########################################################

##########################################################################

# MetaCharacters that control how patterns are matched.

used to create complex search patterns

#  Description                                          #

# \              #  Used to drop the special meaning of character        #

#                #  following it                                         #

# []             #  Represent a character class                          #

# ^              #  Matches the beginning                                #

# $              #  Matches the end                                      #

# .              #  Matches any character except newline                 #

# |              #  Means OR (Matches with any of the characters         #

#                #  separated by it.                                     #

# ?              #  Matches zero or one occurrence                       #

# \*              #  Any number of occurrences (including 0 occurrences)  #

# +              #  One or more occurrences                              #

# {}             #  Indicate the number of occurrences of a preceding    #

#                #  regex to match.                                      #

# ()             #  Enclose a group of Regex                             #

##########################################################################

# We used the fact the re.search() returns a match object if it matches

# and None otherwise. We haven't been interested e.g.

# in what has been matched. The match object contains a

# lot of data about what has been matched, positions and so on.

import re  # Importing the re module for regular expressions

# Define the input string containing names and ages

Nameage = '''Sharat is 35 and Pavan is 28 and Kaleem is 25 and Ganesh is 16'''

# re.findall()

# Return all non-overlapping matches of pattern in string, as a list of strings.

# The string is scanned left-to-right, and matches are returned in the order found.

# Example: Finding all occurrences of a pattern

# Extract ages using regular expression

age = re.findall(r'\d{1,2}', Nameage)  # Find all sequences of 1 or 2 digits representing ages

**['35', '28', '25', '16']**

# Extract names using regular expression

names = re.findall(r'[A-Z][a-z]\*', Nameage)  # Find all words starting with an uppercase letter followed by zero or more lowercase letters, representing names

**['Sharat', 'Pavan', 'Kaleem', 'Ganesh'] output**

# Print the extracted ages and names

print(age)  # Print the list of ages

print(names)  # Print the list of names

# Create an empty dictionary to store name-age pairs

agedict = {}

x = 0

# Iterate through each name and assign corresponding age

for eachname in names:

     agedict[eachname] = age[x]  # Map each name to its corresponding age

     x += 1

# Print the dictionary containing name-age pairs

print(agedict)  # Print the dictionary mapping names to ages

output-{'Sharat': '35', 'Pavan': '28', 'Kaleem': '25', 'Ganesh': '16'}

# Define a text string

txt = "The rain in Spain"

# Split the text string into a list of words using space as the delimiter

x = re.split("\s", txt)  # Split the string using space as the delimiter

print(x)  # Print the list of words

output-['the', 'rain', 'in', 'spain']

import re

string = 'Twelve:12 Eighty nine:89.'

pattern = '\d+' # means one or more digit [0-9]

# Split the string using the pattern

result = re.split(pattern, string)

print(result)  # Print the list of substrings after splitting

output--- ['Twelve:', ' Eighty nine:', '.']

import re

phone = "2004-959-559 # This is Phone Number"

# Delete Python-style comments

num = re.sub('#.\*$', "", phone)  # Substitute Python-style comments with an empty string

print("Phone Num : ", num)  # Print the phone number without comments

output--- >>>2004-959-559

# Remove anything other than digits

num = re.sub('\D', "", phone)  # Substitute anything other than digits with an empty string

print("Phone Num : ", num)  # Print the phone number with only digits

output--- 2004959559

import re

# Find all occurrences of the pattern "name" in the input string

allname = re.findall("name", "My name is Sharat and what is your name")

print(allname)  # Print the list of all occurrences of "name"

# Iterate through each occurrence of "name"

for name in allname:

    print(name)  # Print each occurrence of "name"

# Find match pattern

import re

# Define a string containing names

names = "Aam, Bam, Sam, Ram, Mam, Jam, Fan"

# Find all names that start with a letter in the range A-M followed by "am"

allnames = re.findall("[A-M]am", names)

print(allnames)  # Print the list of names matching the pattern

output---- ['Aam', 'Bam', 'Mam', 'Jam']

# Iterate through each matched name

for i in allnames:

    print(i)  # Print each matched name

import re

# Replace the first two occurrences of a white-space character with the digit 9

txt = "The rain in Spain"

x = re.sub("\s", "9", txt, 2)

print(x)  # Print the modified string after replacement

output-🡪 The9rain9in Spain

## dividing num, alpha, special char

import re

# Replace all alphabetic characters with "\*"

re.sub("[A-Za-z]", "\*", "ada 1231zxdq #@$@zxfsd312")

# Replace all characters except alphabetic characters with "&"

re.sub("[^A-Za-z]", "&", "ada 1231zxdq #@$@zxfsd312")

## Remove white space

randstr = '''

My

name

is

Deepthi

'''

print(randstr)  # Print the string containing white spaces

# re.compile()

# Regular expressions are compiled into pattern objects,

# which have methods for various operations such as searching

# for pattern matches or performing string substitutions.

regex = re.compile("\n")

# Use the compiled regular expression pattern to replace newline characters

# with an empty string.

randstr = regex.sub("", randstr)

print(randstr)# Print the modified string without newline characters

output🡪 My name is Deepthi

################ re.sub function ################

# Python regex sub() function that returns a string after replacing the matched

# pattern in a string with a replacement.

# Introduction to the Python regex sub function

# The sub() is a function in the built-in re module that handles regular expressions.

# The sub() function has the following syntax:

# re.sub(pattern, repl, string, count = 0, flags = 0)

# Code language: Python

# In this syntax:

# pattern is a regular expression that you want to match. Besides a regular

# expression, the pattern can be Pattern object.

# repl is the replacement

# string is the input string

# count parameter specifies the maximum number of matches that the sub() function

# should replace. If you pass zero to the count parameter or completely skip

# it, the sub() function will replace all the matches.

# flags is one or more regex flags that modify the standard behavior of the pattern.

# The sub() function searches for the pattern in the string and replaces the

# matched strings with the replacement (repl).

# If the sub() function couldn’t find a match, it returns the original string.

# Otherwise, the sub() function returns the string after replacing the matches.

# Note that the sub() function replaces the leftmost non-overlapping occurrences

# of the pattern. And you’ll see it in detail in the following example.

# Python regex sub function examples

# Let’s take some examples of using the regex sub() function.

# 1) Using the regex sub() function to return the plain phone number

# The following example uses the sub() function to turn the phone number (212)-456-7890 into 2124567890 :

## Number

import re

randstr = "12345"

# Count the number of digits in the string using regular expression

print(len(re.findall("\d", randstr)))  # Print the count of digits in the string

# Search for the first digit in the string using regular expression

print(re.search("\d{1}", randstr))  # Print the first digit found in the string

## Pan number verification

import re

# Define a PAN number string

pan = "4669-5495-AA1212hhwjjwkq3"

# Check if the PAN number matches the specified pattern

if re.search("\d{4}-\d{4}-\w{15}", pan):

    print("it is a Pan No.")  # Print a message indicating that it is a PAN number

else:

    print("its not a Pan No.")  # Print a message indicating that it is not a PAN number

import re

# Define a string containing a name

name = "My name is Yashvi, you tell me what is your name"

# Split the string into a list of words using space as the delimiter

re.split(" ", name)  # Split the string using space as the delimiter and print the list of words

import re

# Define a string

string = "Python programming is fun"

# Check if 'Python' is at the beginning of the string

match = re.search('\APython', string)

if match:

    print("pattern found inside the string")  # Print if the pattern is found

else:

    print("pattern not found")  # Print if the pattern is not found

# Output: pattern found inside the string

import re

# Define a text string

txt = "The rain in Spain"

# Search for a pattern that starts with 'The', followed by any characters (.\*), and ends with 'Spain'

x = re.search("^The.\*Spain$", txt)

if x:

    print("YES! We have a match!")  # Print if a match is found

else:

    print("No match")  # Print if no match is found