**Practical 2: Write a program to generate regular expression for regular grammar.**

A **Regular Expressions (RegEx)** is a special sequence of characters that uses a search pattern to find a string or set of strings. It can detect the presence or absence of a text by matching it with a particular pattern, and also can split a pattern into one or more sub-patterns. Python provides a **re** module that supports the use of regex in Python. Its primary function is to offer a search, where it takes a regular expression and a string. Here, it either returns the first match or else none.

**MetaCharacters**

To understand the RE analogy, MetaCharacters are useful, important, and will be used in functions of module re. Below is the list of metacharacters.

| MetaCharacters | 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 |

## Special Sequences

Special sequences do not match for the actual character in the string instead it tells the specific location in the search string where the match must occur. It makes it easier to write commonly used patterns.

### List of special sequences

| Special Sequence | Description | Examples | |
| --- | --- | --- | --- |
| \A | Matches if the string begins with the given character | \Afor | for geeks |
| for the world |
| \b | Matches if the word begins or ends with the given character. \b(string) will check for the beginning of the word and (string)\b will check for the ending of the word. | \bge | geeks |
| get |
| \B | It is the opposite of the \b i.e. the string should not start or end with the given regex. | \Bge | together |
| forge |
| \d | Matches any decimal digit, this is equivalent to the set class [0-9] | \d | 123 |
| gee1 |
| \D | Matches any non-digit character, this is equivalent to the set class [^0-9] | \D | geeks |
| geek1 |
| \s | Matches any whitespace character. | \s | gee ks |
| a bc a |
| \S | Matches any non-whitespace character | \S | a bd |
| abcd |
| \w | Matches any alphanumeric character, this is equivalent to the class [a-zA-Z0-9\_]. | \w | 123 |
| geeKs4 |
| \W | Matches any non-alphanumeric character. | \W | >$ |
| gee<> |
| \Z | Matches if the string ends with the given regex | ab\Z | abcdab |
| abababab |

## Regex Module in Python

Python has a module named re that is used for regular expressions in Python.

import re

## 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

import re

# A sample text string where regular expression

# is searched.

string = """Hello my Number is 123456789 and

            my friend's number is 987654321"""

# A sample regular expression to find digits.

regex = '\d+'

match = re.findall(regex, string)

print(match)

**Output**

['123456789', '987654321']

## ****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.

**Example 1:**

|  |
| --- |
| # Module Regular Expression is imported  # using \_\_import\_\_().  import re   # compile() creates regular expression  # character class [a-e],  # which is equivalent to [abcde].  # class [abcde] will match with string with  # 'a', 'b', 'c', 'd', 'e'.  p = re.compile('[a-e]')   # findall() searches for the Regular Expression  # and return a list upon finding  print(p.findall("Aye, said Mr. Gibenson Stark")) |

**Output:**

['e', 'a', 'd', 'b', 'e', 'a']

**Example 4:**

|  |
| --- |
| import re  # '\*' replaces the no. of occurrence  # of a character.  p = re.compile('ab\*')  print(p.findall("ababbaabbb")) |

**Output:**

['ab', 'abb', 'a', 'abbb']

**Understanding the Output:**

* Our RE is ab\*, which ‘a’ accompanied by any no. of ‘b’s, starting from 0.
* Output ‘ab’, is valid because of single ‘a’ accompanied by single ‘b’.
* Output ‘abb’, is valid because of single ‘a’ accompanied by 2 ‘b’.
* Output ‘a’, is valid because of single ‘a’ accompanied by 0 ‘b’.
* Output ‘abbb’, is valid because of single ‘a’ accompanied by 3 ‘b’.

## ****re.sub()****

The ‘sub’ in the function stands for SubString, a certain regular expression pattern is searched in the given string(3rd parameter), and upon finding the substring pattern is replaced by repl(2nd parameter), count checks and maintains the number of times this occurs.

**Syntax:**

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

**Example 1:**

|  |
| --- |
| import re  # Regular Expression pattern 'ub' matches the  # string at "Subject" and "Uber". As the CASE  # has been ignored, using Flag, 'ub' should  # match twice with the string Upon matching,  # 'ub' is replaced by '~\*' in "Subject", and  # in "Uber", 'Ub' is replaced.  print(re.sub('ub', '~\*', 'Subject has Uber booked already',               flags=re.IGNORECASE))   # Consider the Case Sensitivity, 'Ub' in  # "Uber", will not be replaced.  print(re.sub('ub', '~\*', 'Subject has Uber booked already'))  # As count has been given value 1, the maximum  # times replacement occurs is 1  print(re.sub('ub', '~\*', 'Subject has Uber booked already',               count=1, flags=re.IGNORECASE))  # 'r' before the pattern denotes RE, \s is for  # start and end of a String.  print(re.sub(r'\sAND\s', ' & ', 'Baked Beans And Spam',               flags=re.IGNORECASE)) |

**Output**

S~\*ject has ~\*er booked already

S~\*ject has Uber booked already

S~\*ject has Uber booked already

Baked Beans & Spam

## ****re.subn()****

subn() is similar to sub() in all ways, except in its way of providing output. It returns a tuple with a count of the total of replacement and the new string rather than just the string.

**Syntax:**

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

**Example:**

|  |
| --- |
| import re  print(re.subn('ub', '~\*', 'Subject has Uber booked already'))  t = re.subn('ub', '~\*', 'Subject has Uber booked already',              flags=re.IGNORECASE)  print(t)  print(len(t))  # This will give same output as sub() would have  print(t[0]) |

**Output**

('S~\*ject has Uber booked already', 1)

('S~\*ject has ~\*er booked already', 2)

Length of Tuple is: 2

S~\*ject has ~\*er booked already