1. What is the name of the feature responsible for generating Regex objects?

The feature responsible for generating regex objects in python is re module. It has functions and methods which allows to create regex objects and used to match, manipulate strings based on patterns defined by regular expression.

Eg:

import re

pattern = r'\d+'

text = '123 sarigama 456'

matches = re.findall(pattern,text)

print(matches) # output: ['123', '456']

2. Why do raw strings often appear in Regex objects?

Raw strings, indicated by the prefix r before the string literal, are often used in Regex objects to avoid unwanted interpretation of backslashes (\) in the regular expression pattern.

In Python, backslashes have a special meaning in regular strings. They are used to escape certain characters or represent special character sequences, such as newline (\n) or tab (\t). However, in regular expressions, backslashes are also used to escape metacharacters or represent special sequences with specific meanings.

To simplify working with regular expressions, raw strings are commonly used because they treat backslashes as literal characters and prevent any special interpretation. This is important because regular expressions often contain many backslashes to escape metacharacters or define special sequences.

For example, consider the following regular expression pattern to match a literal backslash followed by a digit

import re

pattern = r'\\[0-9]'

text = r'\1 is the blackslash followed by a digit'

matches = re.findall(pattern,text)

print(matches) #output ['\\1']

3. What is the return value of the search() method?

The search() method from the re module in Python returns a match object if a match is found, or None if no match is found. The match object contains information about the match, such as the matched string, its position, and other details.

Eg:

import re

pattern = r'apple'

text = "I have an apple"

match = re.search(pattern,text)

if match:

print("match found: ",match.group())

else:

print("Match mot found")

output: match found: apple

4. From a Match item, how do you get the actual strings that match the pattern?

The group() method is used to access the matched strings based on different groups defined in the regular expression pattern.

Eg:

import re

pattern = r'(\d+)-(\w+)'

text = '42-spam'

match = re.search(pattern, text)

if match:

whole\_match = match.group() # Get the whole matched string

first\_group = match.group(1) # Get the first group

second\_group = match.group(2) # Get the second group

print('Whole match:', whole\_match) # Output: 42-spam

print('First group:', first\_group) # Output: 42

print('Second group:', second\_group) # Output: spam

else:

print('No match found')

5. In the regex which created from the r'(\d\d\d)-(\d\d\d-\d\d\d\d)', what does group zero cover? Group 2? Group 1?

1. Group 0, represented by match.group(0), covers the entire matched string.
2. Group 1, represented by match.group(1), covers the first group (\d\d\d) within parentheses.
3. Group 2, represented by match.group(2), covers the second group (\d\d\d-\d\d\d\d) within parentheses.

Eg:

import re

pattern = r'(\d\d\d)-(\d\d\d-\d\d\d\d)'

text = '123-456-7890'

match = re.search(pattern, text)

if match:

whole\_match = match.group(0)

group\_1 = match.group(1)

group\_2 = match.group(2)

print('Whole match:', whole\_match) # Output: 123-456-7890

print('Group 1:', group\_1) # Output: 123

print('Group 2:', group\_2) # Output: 456-7890

else:

print('No match found')

6. In standard expression syntax, parentheses and intervals have distinct meanings. How can you tell a regex that you want it to fit real parentheses and periods?

In regular expressions, parentheses and periods have special meanings and are used to define groups and match any character (except a newline) respectively. To tell the regex engine that you want to match literal parentheses and periods, you can use the backslash \ to escape them.

By preceding the parentheses or periods with a backslash, you indicate that you want them to be treated as literal characters and not as part of the regex syntax.

Here is an example

import re

text = 'I have (some) parenthesis and period.'

pattern = r'I have \(some\) parenthesis and period\.'

match = re.search(pattern,text)

if match:

print("match found: ",match.group())

else:

print("No match found.")

7. The findall() method returns a string list or a list of string tuples. What causes it to return one of the two options?

The findall() method in regular expressions returns a list of strings if the regex pattern being searched does not contain any capturing groups. However, if the pattern includes one or more capturing groups, findall() returns a list of tuples, where each tuple represents a match and contains the captured groups.

Eg:

import re

text = 'Hello, my name is John. I live in New York.'

# Pattern without capturing group

pattern1 = r'[A-Za-z]+'

result1 = re.findall(pattern1, text)

print('Pattern without capturing group:', result1)

# Output: ['Hello', 'my', 'name', 'is', 'John', 'I', 'live', 'in', 'New', 'York']

# Pattern with capturing group

pattern2 = r'([A-Za-z]+), ([A-Za-z]+)'

result2 = re.findall(pattern2, text)

print('Pattern with capturing group:', result2)

# Output: [('Hello', 'my'), ('name', 'is')]

8. In standard expressions, what does the | character mean?

| (pipe) character is used as the OR operator

Eg:

import re

text = 'I have a cat and dog.'

pattern = r'cat|dog'

match = re.findall(pattern,text)

print(match) #output ['cat', 'dog']

9. In regular expressions, what does the character stand for?

In the context of regular expressions, the dot (.) character is a metacharacter that matches any single character except a newline. It represents a wildcard or placeholder for any character.

For example, the pattern a.b would match any string that starts with "a", followed by any character, and then followed by "b". The dot acts as a placeholder for the unknown character.

Eg:

import re

text = 'aab,abb,acb'

pattern = r'a.b'

match = re.findall(pattern,text)

print(match) #output ['aab', 'abb', 'acb']

10.In regular expressions, what is the difference between the + and \* characters?

+ (Plus): The plus quantifier matches one or more occurrences of the preceding element.

\* (Asterisk): The asterisk quantifier matches zero or more occurrences of the preceding element.

import re

text = 'cat caat caaat ct'

pattern\_plus = r'ca+t'

pattern\_asterisk = r'ca\*t'

matches\_plus = re.findall(pattern\_plus, text)

matches\_asterisk = re.findall(pattern\_asterisk, text)

print(matches\_plus)

# Output: ['cat', 'caat', 'caaat']

print(matches\_asterisk)

# Output: ['ct', 'cat', 'caat', 'caaat']

11. What is the difference between {4} and {4,5} in regular expression?

1. {4}: It specifies that the preceding element must occur exactly four times.
2. {4,5}: It specifies that the preceding element must occur between four and five times (inclusive). It matches a range of occurrences of the preceding element, from four to five.

Eg:

import re

text = 'aaaa aaa aaaa aaaaa'

pattern\_exact = r'a{4}'

pattern\_range = r'a{4,5}'

matches\_exact = re.findall(pattern\_exact, text)

matches\_range = re.findall(pattern\_range, text)

print(matches\_exact)

# Output: ['aaaa']

print(matches\_range)

# Output: ['aaaa', 'aaaaa']

12. What do you mean by the \d, \w, and \s shorthand character classes signify in regular expressions?

In regular expressions, the shorthand character classes \d, \w, and \s have special meanings and are used to represent certain types of characters:

\d: It represents any digit character. It is equivalent to the character class [0-9]. It matches any single digit from 0 to 9.

For example, the pattern \d\d\d would match any three consecutive digits in a string.

\w: It represents any alphanumeric character (letters and digits) and underscores. It is equivalent to the character class [a-zA-Z0-9\_]. It matches any single alphanumeric character or underscore.

For example, the pattern \w+ would match one or more consecutive alphanumeric characters or underscores in a string.

\s: It represents any whitespace character, including spaces, tabs, and newlines. It matches any single whitespace character.

Eg:

import re

text = 'abc 123 xyz'

pattern\_digit = r'\d+'

pattern\_word = r'\w+'

pattern\_space = r'\s+'

matches\_digit = re.findall(pattern\_digit, text)

matches\_word = re.findall(pattern\_word, text)

matches\_space = re.findall(pattern\_space, text)

print(matches\_digit)

# Output: ['123']

print(matches\_word)

# Output: ['abc', '123', 'xyz']

print(matches\_space)

# Output: [' ']

13. What do means by \D, \W, and \S shorthand character classes signify in regular expressions?

In regular expressions, the shorthand character classes \d, \w, and \s have special meanings and are used to represent certain types of characters:

\d: It represents any digit character. It is equivalent to the character class [0-9]. It matches any single digit from 0 to 9.

For example, the pattern \d\d\d would match any three consecutive digits in a string.

\w: It represents any alphanumeric character (letters and digits) and underscores. It is equivalent to the character class [a-zA-Z0-9\_]. It matches any single alphanumeric character or underscore.

For example, the pattern \w+ would match one or more consecutive alphanumeric characters or underscores in a string.

\s: It represents any whitespace character, including spaces, tabs, and newlines. It matches any single whitespace character.

Eg:

import re

text = 'abc 123 xyz'

pattern\_digit = r'\d+'

pattern\_word = r'\w+'

pattern\_space = r'\s+'

matches\_digit = re.findall(pattern\_digit, text)

matches\_word = re.findall(pattern\_word, text)

matches\_space = re.findall(pattern\_space, text)

print(matches\_digit)

# Output: ['123']

print(matches\_word)

# Output: ['abc', '123', 'xyz']

print(matches\_space)

# Output: [' ']

14. What is the difference between .\*? and .\*?

In regular expressions, .\*? and .\* are both quantifiers used to match a sequence of characters. However, there is a difference in their behavior:

.\*? (non-greedy or lazy match): This quantifier matches zero or more occurrences of any character (except newline) in a non-greedy or lazy manner. It tries to match as few characters as possible to satisfy the overall pattern.

.\* (greedy match): This quantifier matches zero or more occurrences of any character (except newline) in a greedy manner. It tries to match as many characters as possible to satisfy the overall pattern.

Eg:

import re

text = 'abc def ghi jkl'

pattern\_lazy = r'a.\*?i'

pattern\_greedy = r'a.\*i'

match\_lazy = re.search(pattern\_lazy, text)

match\_greedy = re.search(pattern\_greedy, text)

print(match\_lazy.group())

# Output: 'abc def ghi'

print(match\_greedy.group())

# Output: 'abc def ghi jkl'

15. What is the syntax for matching both numbers and lowercase letters with a character class?

Syntax: [0-9a-z]

pattern = r'[0-9a-z]'

16. What is the procedure for making a normal expression in regax case insensitive?

To make a regular expression case-insensitive in Python, you can use the re.IGNORECASE or re.I flag as an argument to the regular expression functions (re.search(), re.match(), re.findall(), etc.) or as an inline flag within the regular expression pattern.

Here are two ways to make a regular expression case-insensitive:

1.Using the re.IGNORECASE or re.I flag as an argument:

import re

text = 'Hello World'

pattern = r'hello'

# Using the flag as an argument to the function

match = re.search(pattern, text, re.IGNORECASE)

print(match)

# Output: <re.Match object; span=(0, 5), match='Hello'>

2.Using the inline flag within the pattern:

import re

text = 'Hello World'

pattern = r'(?i)hello'

# Using the inline flag within the pattern

match = re.search(pattern, text)

print(match)

# Output: <re.Match object; span=(0, 5), match='Hello'>

17. What does the . character normally match? What does it match if re.DOTALL is passed as 2nd argument in re.compile()?

In a regular expression pattern, the . (dot) character normally matches any character except a newline (\n). It matches any single character in the input string.

However, if you pass re.DOTALL as the second argument to the re.compile() function or use the re.DOTALL flag in other regular expression functions (re.search(), re.match(), etc.), the . character will match any character, including a newline.

Eg:

import re

text = 'Hello\nWorld'

# Normal matching without re.DOTALL

pattern = r'.'

match = re.search(pattern, text)

print(match.group())

# Output: 'H'

# Matching with re.DOTALL

pattern = r'.'

regex = re.compile(pattern, re.DOTALL)

match = regex.search(text)

print(match.group())

# Output: 'H\n'

18. If numReg = re.compile(r'\d+'), what will numRegex.sub('X', '11 drummers, 10 pipers, five rings, 4 hen') return?

'X drummers, X pipers, five rings, X hen'.

The numRegex.sub('X', '11 drummers, 10 pipers, five rings, 4 hen') statement uses the sub() method of the compiled regular expression pattern numRegex to substitute all occurrences of numeric digits with the string 'X' in the input text '11 drummers, 10 pipers, five rings, 4 hen'.

19. What does passing re.VERBOSE as the 2nd argument to re.compile() allow to do?

Passing re.VERBOSE as the second argument to re.compile() allows you to add whitespace and comments to the regular expression pattern for improved readability. It enables the use of multi-line regular expressions with added whitespace and comments without affecting the pattern matching.

When using re.VERBOSE, you can format the regular expression pattern with line breaks, indentation, and comments to make it more understandable. Any whitespace character (except within a character class) and comments starting with # will be ignored by the regular expression engine.

Eg :

pattern = re.compile(r'''

^([a-zA-Z0-9\_.+-]+) # username

@ # @ symbol

([a-zA-Z0-9-]+) # domain name

\. # dot

([a-zA-Z0-9-.]+) # top-level domain

$ # end of string

''', re.VERBOSE)

20. How would you write a regex that match a number with comma for every three digits? It must match the given following:

'42'

'1,234'

'6,368,745'

but not the following:

'12,34,567' (which has only two digits between the commas)

'1234' (which lacks commas)

pattern = re.compile(r'^\d{1,3}(,\d{3})\*$')

21. How would you write a regex that matches the full name of someone whose last name is Watanabe? You can assume that the first name that comes before it will always be one word that begins with a capital letter. The regex must match the following:

'Haruto Watanabe'

'Alice Watanabe'

'RoboCop Watanabe'

but not the following:

'haruto Watanabe' (where the first name is not capitalized)

'Mr. Watanabe' (where the preceding word has a nonletter character)

'Watanabe' (which has no first name)

'Haruto watanabe' (where Watanabe is not capitalized)

pattern = re.compile(r'^[A-Z][a-zA-Z]\*\sWatanabe$')

Explanation of the pattern:

* ^ asserts the start of the string.
* [A-Z] matches a capital letter (first letter of the first name).
* [a-zA-Z]\* matches zero or more lowercase or uppercase letters (remaining letters of the first name).
* \s matches a whitespace character (space between the first and last name).
* Watanabe matches the literal string 'Watanabe'.
* $ asserts the end of the string.

22. How would you write a regex that matches a sentence where the first word is either Alice, Bob, or Carol; the second word is either eats, pets, or throws; the third word is apples, cats, or baseballs; and the sentence ends with a period? This regex should be case-insensitive. It must match the following:

'Alice eats apples.'

'Bob pets cats.'

'Carol throws baseballs.'

'Alice throws Apples.'

'BOB EATS CATS.'

but not the following:

'RoboCop eats apples.'

'ALICE THROWS FOOTBALLS.'

'Carol eats 7 cats.'

pattern = re.compile(r'^(Alice|Bob|Carol)\s(eats|pets|throws)\s(apples|cats|baseballs)\.$', re.IGNORECASE)

Explanation of the pattern:

* ^ asserts the start of the string.
* (Alice|Bob|Carol) matches one of the given names: Alice, Bob, or Carol.
* \s matches a whitespace character (space) between the words.
* (eats|pets|throws) matches one of the given actions: eats, pets, or throws.
* (apples|cats|baseballs) matches one of the given objects: apples, cats, or baseballs.
* \. matches a period at the end of the sentence.
* $ asserts the end of the string.
* re.IGNORECASE flag makes the pattern case-insensitive.