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

ANSWER.

The feature responsible for generating regular expression objects in Python is the `re` module. This module provides support for working with regular expressions (regex) in Python.

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

ANSWER.

Because they make it easier to write and read complex regex patterns without having to escape special character.

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

ANSWER.

It returns the match object. If not found any match object it will return ‘None’.

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

ANSWER.

We can use ‘group()’ method.

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?

ANSWER.

Group zero: Entire match

Group 1: (\d\d\d)

Group 2: (\d\d\d-\d\d\d\d)

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?

ANSWER.

To tell a regular expression (regex) that you want it to match real parentheses `(` and `)` and periods `.`, you need to escape these characters using a backslash `\`.

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

ANSWER.

The `findall()` method in Python's `re` module returns different types of results based on whether the regular expression pattern contains capturing groups or not.

1. If the regex pattern contains capturing groups (defined by parentheses `()`), `findall()` returns a list of tuples where each tuple corresponds to a match, and each element in the tuple corresponds to a capturing group within that match.

2. If the regex pattern does not contain any capturing groups, `findall()` returns a list of strings, where each string represents a complete match.

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

ANSWER.

In regular expressions, the `|` character is known as the alternation operator or the pipe symbol. It is used to specify alternatives within a regex pattern.

Here's how it works:

- `pattern1 | pattern2`: This matches either `pattern1` or `pattern2`. It behaves like an "OR" operator, allowing you to match one pattern or another.

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

ANSWER.

In regular expressions, the `+` character is known as the "one or more" quantifier. It is used to specify that the preceding element (character, group, or character class) must occur one or more times consecutively in the string being matched.

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

ANSWER.

The main difference between `+` and `\*` in regular expressions is that `+` requires at least one occurrence of the preceding element, while `\*` allows for zero or more occurrences.

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

ANSWER.

`{m}` specifies an exact number of occurrences, while `{m,n}` specifies a range of occurrences.

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

ANSWER.

In regular expressions, `\d`, `\w`, and `\s` are shorthand character classes that represent specific sets of characters:

1. `\d`: This shorthand represents any digit character. It matches any single digit from 0 to 9.

2. `\w`: This shorthand represents any word character. It matches any alphanumeric character (letters, digits, or underscores), as well as certain Unicode characters that can be part of a word in many programming languages.

3. `\s`: This shorthand represents any whitespace character. It matches spaces, tabs, newlines, carriage returns, and other types of whitespace characters.

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

ANSWER.

In regular expressions, `\D`, `\W`, and `\S` are shorthand character classes that represent the negation of `\d`, `\w`, and `\s`, respectively. They match characters that are not included in their respective counterpart character classes:

1. `\D`: This shorthand represents any character that is not a digit. It matches any character other than 0 to 9.

2. `\W`: This shorthand represents any character that is not a word character. It matches any character that is not a letter, digit, or underscore.

3. `\S`: This shorthand represents any character that is not whitespace. It matches any character other than spaces, tabs, newlines, and other whitespace characters.

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

ANSWER.

`.\*?` and `.\*` behave differently in terms of greediness. `.\*?` is non-greedy, matching the shortest possible sequence of characters, while `.\*` is greedy, matching the longest possible sequence of characters.

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

ANSWER.

[0-9a-z]

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

ANSWER.

To make a regular expression case-insensitive in Python, you can use the `re.IGNORECASE` or `re.I` flag when compiling the regular expression pattern. This flag tells the regex engine to ignore case differences when matching characters.

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

ANSWER.

In regular expressions, the `.` (dot) character normally matches any character except a newline (`\n`). It matches any single character except for newline characters.

However, when the `re.DOTALL` or `re.S` flag is passed as the second argument to `re.compile()`, the behavior of the `.` (dot) character changes. With the `re.DOTALL` flag, the dot `.` matches any character, including newline characters.

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

ANSWER.

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

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

ANSWER.

Passing `re.VERBOSE` as the second argument to `re.compile()` allows you to write more readable and organized regular expressions by enabling verbose mode. In verbose mode, whitespace and comments within the regular expression pattern are ignored, allowing you to format the pattern more clearly and add comments to explain its structure.

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)

ANSWER.

^\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)

ANSWER.

^[A-Z][a-zA-Z]\*\sWatanabe$

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.'

ANSWER.

^(Alice|Bob|Carol)\s+(eats|pets|throws)\s+(apples|cats|baseballs)\.$