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

The feature responsible for generating regular expression (regex) objects in Python is the re module.

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

Raw strings are often used in Regex objects in Python to avoid having to escape special characters. In a regular string, characters like \ and ' have special meanings and need to be escaped, which can make the pattern hard to read and write. Raw strings, on the other hand, preserve the literal value of each character, making it easier to write complex regex patterns. To create a raw string in Python, you prefix the string with r. For example: r'\d+' is a raw string that represents the regex pattern \d+, which matches one or more digits.

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

The search() method in Python, which is part of the re module, returns a Match object if there is a match anywhere in the string. If there is no match, it returns None. The Match object contains information about the match, such as the start and end position of the match, and can be used to extract substrings from the original string that match the pattern.

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

You can get the actual strings that match the pattern in a Match object by using the group() method. The group() method returns the portion of the original string that matches the pattern. If the pattern has capturing groups (i.e., groups surrounded by parentheses), you can use the group(n) method, where n is the index of the capturing group, to get the portion of the string that was matched by that group.

**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?**

In the regular expression r'(\d\d\d)-(\d\d\d-\d\d\d\d)', group zero covers the entire match, i.e., the entire string that matches the pattern.

Group 1 covers the first capturing group, which is \d\d\d. This group matches exactly three digits.

Group 2 covers the second capturing group, which is \d\d\d-\d\d\d\d. This group matches a string that consists of three digits, followed by a hyphen, followed by four more digits.

import re

text = "My phone number is 555-555-5555."

pattern = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')

match = pattern.search(text)

if match:

print(match.group(0))

print(match.group(1))

print(match.group(2))

This code will print:

555-555-5555

555

555-5555

**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 expression syntax, special characters like parentheses and periods have special meanings and need to be escaped in order to match them literally. To escape a special character in a regex pattern, you prefix it with a backslash (\).

For example, to match the literal string (abc), you would write the pattern \(abc\). To match the literal string a.b, you would write the pattern a\.b.

**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 of the re module in Python returns a list of strings or a list of tuples of strings, depending on the presence of capturing groups in the regular expression pattern.

If the pattern has no capturing groups (i.e., groups surrounded by parentheses), findall() returns a list of strings, where each string is a match of the pattern in the text.

If the pattern has one or more capturing groups, findall() returns a list of tuples of strings, where each tuple contains the string matched by each capturing group.

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

In regular expression syntax, the vertical bar (|) character is called the alternation operator. It allows you to match one of multiple alternatives. For example, if you want to match either dog or cat, you can use the pattern dog|cat.

You can use multiple alternation operators in the same pattern to match any of several alternatives. For example, dog|cat|bird will match either dog, cat, or bird.

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

In regular expressions, the . (dot) character is a special character that matches any single character except a newline character. In other words, it matches any character except a line break.

If you want to match a literal dot character, you need to escape it with a backslash (\), like this: \..

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

In regular expressions, the + and \* characters are both special characters that specify repetition. However, they have different meanings:

The + character specifies that the preceding element should be matched one or more times. For example, the pattern \d+ matches one or more consecutive digits.

The \* character specifies that the preceding element should be matched zero or more times. For example, the pattern \d\* matches zero or more consecutive digits.

the + and \* operators are "greedy" by default, meaning that they try to match as much as possible. If you want them to match as little as possible, you can use the ? character after the + or \* operator, like this: \d+? or \d\*?. This is known as a "lazy" match.

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

In regular expressions, the curly brackets ({}) are used to specify the number of times that the preceding element should be matched.

The syntax {n} specifies that the preceding element should be matched exactly n times. For example, the pattern \d{3} matches exactly three consecutive digits.

The syntax {m,n} specifies that the preceding element should be matched between m and n times, inclusive. For example, the pattern \d{3,5} matches between three and five consecutive digits.

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

In regular expressions, shorthand character classes are special symbols that represent a set of characters. The most commonly used shorthand character classes are \d, \w, and \s.

The \d shorthand character class represents any digit character. It is equivalent to the character set [0-9].

The \w shorthand character class represents any word character. It is equivalent to the character set [a-zA-Z0-9\_], which includes letters, digits, and the underscore character.

The \s shorthand character class represents any whitespace character, including spaces, tabs, and line breaks.

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

In regular expressions, shorthand character classes are special symbols that represent a set of characters. The shorthand character classes \D, \W, and \S are the negated versions of \d, \w, and \s, respectively.

The \D shorthand character class represents any character that is not a digit. It is equivalent to the negated character set [^0-9].

The \W shorthand character class represents any character that is not a word character. It is equivalent to the negated character set [^a-zA-Z0-9\_], which includes all characters except letters, digits, and the underscore character.

The \S shorthand character class represents any character that is not a whitespace character. It is equivalent to the negated character set [^\s].

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

In regular expressions, the .\* and .\*? operators are used to match zero or more characters. However, they behave differently.

The .\* operator matches zero or more characters in a greedy manner, which means it matches as many characters as possible before attempting to match the next character in the pattern.

The .\*? operator matches zero or more characters in a non-greedy manner, which means it matches as few characters as possible before attempting to match the next character in the pattern.

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

In regular expressions, you can use a character class to match both numbers and lowercase letters by using the \w shorthand character class, which matches any word character. In Python, you can specify this using square brackets [] and the range of characters 0-9, a-z.

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

In Python, you can make a regular expression case-insensitive by passing the re.IGNORECASE or re.I flag as the second argument to the re.compile() function.

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

In regular expressions, the . character normally matches any character except a newline character (\n). In other words, . matches any character that appears on the same line as the pattern.

However, if you pass the re.DOTALL flag as the second argument to the re.compile() function, the . character will match 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?**

The code numReg = re.compile(r'\d+') creates a regular expression pattern that matches one or more consecutive digits (\d). The .sub() method is then used to substitute any matches of this pattern in the string '11 drummers, 10 pipers, five rings, 4 hen' with the string 'X'.

So the expression numRegex.sub('X', '11 drummers, 10 pipers, five rings, 4 hen') will return the following string:

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

This is because the \d+ pattern matches the sequences of digits 11, 10, and 4 in the input string, and the .sub() method replaces those matches with the string 'X'.

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

Passing re.VERBOSE as the second argument to the re.compile() function allows you to include whitespace and comments in your regular expression pattern. This makes the pattern easier to read and understand, especially for complex or long patterns.

In a verbose regular expression, you can include whitespace and comments in the pattern that will be ignored when matching. Comments start with a # symbol and continue to the end of the line.

**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'**

Here's a regex pattern that will match numbers with commas for every three digits:

import re

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

print(pattern.match('42'))

print(pattern.match('1,234'))

print(pattern.match('6,368,745'))

This pattern uses the following elements:

* ^: Matches the start of the string
* \d{1,3}: Matches 1 to 3 digits
* (,\d{3})\*: Matches 0 or more occurrences of a comma followed by 3 digits
* $: Matches the end of the string

This pattern will match numbers that start with 1 to 3 digits, followed by any number of occurrences of a comma and 3 more digits. The ^ and $ characters anchor the pattern to the start and end of the string, so the pattern only matches if the entire string matches the pattern. The output of the code will be None None <re.Match object; span=(0, 11), match='6,368,745'>, indicating that only the third string matches the pattern.