1. What is the name of the feature responsible for generating Regex objects?  
**Answer:** In Python, the re module is responsible for generating Regex objects.

The re module is part of the Python standard library and provides support for regular expressions in Python. It contains functions and classes for working with regular expressions, including the compile() function, which is used to create a regular expression object.

2. Why do raw strings often appear in Regex objects?  
**Answer:** Raw strings are often used in Regex objects to avoid unintentional escaping of characters. In Python, backslashes (\) are used to escape special characters in a string. However, this can be problematic in regular expressions, as regular expressions also use backslashes to escape special characters.

For example, consider the following regular expression pattern:

pattern = "\d+"

This pattern matches one or more digits. However, in Python, the backslash is used to escape special characters, such as the newline character (\n) or the tab character (\t). Therefore, to match a literal backslash character in the regular expression pattern, we need to escape it with another backslash, like this:

pattern = "\\d+"

This can quickly become confusing, especially for complex regular expressions with many backslashes. To avoid this confusion, we can use a raw string, which tells Python to treat the backslashes literally and not interpret them as escape characters. A raw string is created by prefixing a string with the letter 'r'.

For example, the same regular expression pattern written as a raw string would look like this:

pattern = r"\d+"

In this case, the r prefix tells Python to treat the backslash literally and not interpret it as an escape character. Using a raw string makes regular expressions easier to read and less error-prone.

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

**Answer:** The re.search() function will search the regular expression pattern and return the first occurrence. Unlike Python re. match(), it will check all lines of the input string. If the pattern is found, the match object will be returned, otherwise “null” is returned.

4. From a Match item, how do you get the actual strings that match the pattern?  
**Answer:** In Python, you can get the actual strings that match the pattern from a Match object using the group() method.

The group() method returns the string that was matched by the regular expression. If the regular expression contains groups (i.e., parts of the expression enclosed in parentheses), then group() can also accept an optional argument indicating which group to retrieve. The argument can be an integer specifying the group number (starting from 1), or a string specifying the name of the 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?  
**Answer:** The regex pattern r'(\d\d\d)-(\d\d\d-\d\d\d\d)' defines two capturing groups, each enclosed in parentheses:

Group 1: (\d\d\d), which matches three consecutive digits.

Group 2: (\d\d\d-\d\d\d\d), which matches a hyphen-separated string of six digits (three before the hyphen, and four after).

When a match is found using this regex pattern, the group(0) method of the Match object returns the entire matched string, including any characters that are not captured by the two groups.

To access the captured substrings of the regex pattern, we can use the group(1) and group(2) methods of the Match object to retrieve the substrings that match the patterns defined by Group 1 and Group 2, respectively.

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 match literal parentheses and periods in a regular expression, you need to escape them using a backslash character (\). This tells the regular expression engine to treat the following character as a literal character, rather than as a special metacharacter with its own meaning in regular expression syntax.

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 regular expression module re returns a list of all non-overlapping matches in a string. Whether it returns a list of strings or a list of tuples depends on whether the regular expression pattern contains capturing groups or not.

If the pattern contains no capturing groups (no parentheses), then findall() returns a list of strings, where each element in the list is a match.

If the pattern contains one or more capturing groups (one or more pairs of parentheses), then findall() returns a list of tuples, where each tuple corresponds to a match, and each element in the tuple corresponds to a capturing group in the regular expression pattern.

8. In standard expressions, what does the | character mean?  
**Answer:** In regular expressions, the vertical bar (|) character is used to denote alternation. It represents a logical OR operation between two or more expressions, where either one expression or another expression (or both) can match.

9. In regular expressions, what does the character stand for?  
**Answer:** In regular expressions, the dot (.) character is a metacharacter that matches any single character, except for a newline (\n).

10.In regular expressions, what is the difference between the + and \* characters?  
**Answer:** In regular expressions, the + and \* characters are both quantifiers that match one or more occurrences of the preceding element in the regular expression pattern. However, there is a difference between how they match:

The + character matches one or more occurrences of the preceding element. In other words, the preceding element must occur at least once in the input string for the match to succeed. For example, the regular expression pattern a+ matches one or more occurrences of the letter "a" in a string.

The \* character matches zero or more occurrences of the preceding element. In other words, the preceding element may occur zero times or more in the input string for the match to succeed. For example, the regular expression pattern a\* matches zero or more occurrences of the letter "a" in a string.

11. What is the difference between {4} and {4,5} in regular expression?  
**Answer:** In regular expressions, curly braces {} are used as quantifiers to specify the number of occurrences of the preceding element that should be matched.

{4} means that the preceding element should be matched exactly four times.

{4,5} means that the preceding element should be matched at least four times and at most five times.

12. What do you mean by the \d, \w, and \s shorthand character classes signify in regular expressions?  
  
**Answer:** In regular expressions, shorthand character classes are used to represent common character groups.

Here are the meanings of the \d, \w, and \s shorthand character classes:

\d: matches any digit character (0-9).

\w: matches any word character, which includes alphanumeric characters (a-z, A-Z, 0-9) and underscores (\_).

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

13. What do means by \D, \W, and \S shorthand character classes signify in regular expressions?  
**Answer:** In regular expressions, the uppercase shorthand character classes \D, \W, and \S represent the negation of the lowercase shorthand character classes \d, \w, and \s, respectively.

Here are the meanings of the \D, \W, and \S shorthand character classes:

\D: matches any non-digit character (equivalent to [^0-9]).

\W: matches any non-word character (equivalent to [^a-zA-Z0-9\_]).

\S: matches any non-whitespace character.

14. What is the difference between .\*? and .\*?  
**Disclaimer:** Provided same pattern twice

15. What is the syntax for matching both numbers and lowercase letters with a character class?  
**Answer:** In Python regular expressions, you can match both numbers and lowercase letters using the following syntax for a character class:

[0-9a-z]

This character class matches any single character that is a digit (0-9) or a lowercase letter (a-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 in the re.compile() function or as a flag in the regex function.

17. What does the . character normally match? What does it match if re.DOTALL is passed as 2nd argument in re.compile()?  
**Answer:** In Python regular expressions, the . (dot) character matches any single character except for a newline character (\n) by default.

However, if you pass re.DOTALL (or re.S) as the second argument to re.compile(), the dot 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?  
**Answer:** If numReg = re.compile(r'\d+'), then calling numRegex.sub('X', '11 drummers, 10 pipers, five rings, 4 hen') will replace all occurrences of one or more digits with the letter 'X' in the given string.

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 for more readable and maintainable regular expressions by allowing the use of whitespace and comments within the pattern without affecting the pattern itself.

The re.VERBOSE flag enables verbose mode in regular expressions. In this mode, the following features are enabled:

Whitespace in the pattern is ignored, except when it is in a character class or when it is escaped.

Comments can be included using the "#" character. Any characters after "#" and up to the end of the line are ignored.

A backslash followed by a newline is ignored.

Using re.VERBOSE can make complex regular expressions much easier to read and understand by breaking them up into smaller, more manageable parts and adding comments to explain what each part of the pattern does.

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-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:**import re

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