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

re.compile()

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

So that ,backslashes treated as literal characters and do not interpret as escape characters, preventing it from escaping.

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

It returns a match object if it finds a match anywhere in the string being searched. If no match found, it returns None.

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

To get match item we can use group() method , that returns the portion of the input string that matched the specified pattern.

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?

Group 0 cover the entire match

Group 1 covers the first set of parentheses contains which cover a sequence of three digits.

Group 2 covers the second set of parentheses contains which cover a sequence of three digits, followed by hyphen, and then followed by four digits.

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?

To match real parentheses ()

use ‘\(‘ match for opening parenthesis and use ‘\)’ to match a closing parenthesis ).

& For period , use ‘\.’

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

If the regular expression pattern has no capturing groups then findall() returns a list of strings and if it has one or more capturing groups then it returns a list of tuples.

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

‘|’ character acts as a separator between two sub patterns. It signifies that the pattern should match either the left-side subpattern or the right-side subpattern.

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

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

+ matches one or more occurrences of the preceding pattern. It requires at least one occurrence of the preceding pattern for a match to be successful ,if not present, the match fails.

\* is much similar to + , it matches zero or more occurrences of the preceding pattern. Only difference is, it allows for zero occurrences of the preceding pattern, so making it optional , means if the preceding pattern is not present, the match still succeeds.

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

{4} specifies that preceding pattern must occur exactly 4 times for a match to be successful, otherwise match fails , where as {4,5} specifies a range of repetition for the preceding pattern. It look for matches of minimum of 4 occurrences and a maximum of 5 occurrences for the preceding pattern.

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

\d specifies for numbers 0 to 9

\w specifies for alphanumeric characters

\s specifies for whitespace

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

\d specifies for Non digits

\w specifies for Non word characters

\s specifies for Non – whitespace characters

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

.\*? do a non-greedy match . It will match the shortest possible sequence of characters that satisfies the pattern. Where as ‘.\* ’do a greedy match. It will match the longest possible sequence of characters that satisfies the pattern.

import re

text = "xyzxyzxyz"

non\_greedy = r"x.\*?z"

print(re.search(non\_greedy, text)) # match='xyz'

greedy = r"x.\*z"

print(re.search(greedy, text)) #match='xyzxyzxyz'

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

[0-9a-z]

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

By Compile the pattern with the re.IGNORECASE or re.I ,we can make the regular expression case insensitive, allowing it to match patterns regardless of the letter case.

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

. character normally mean it can match any character except newline , when it match if re.DOTALL is passed as 2nd argument in re.compile()? , now . characters can match any character including newline.

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 hens'

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

Passing re.VERBOSE as the 2nd argument to re.compile() allows you to add whitespace and comments to the pattern string , improves readability of the pattern string

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)

Required\_regex = r'^\d{1,3}(,\d{3})\*$' make a regex that match a number with comma for every three digits.

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)

Required\_regex = r'^[A-Z][a-zA-Z]\* Watanabe$'

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.'

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