Q1. What is the function that creates Regex objects?

import re phoneNumRegex = re.compile(r'\d\d\d-\d\d\d\d-\d\d\d')

Q2. Why are raw strings often used when creating Regex objects?

Since regular expressions frequently use backslashes in them, it is convenient to pass raw strings to the re.compile() function instead of typing extra backslashes.

Q3. What does the search() method return?

The search() method will return None if the regex pattern is not found in the string. If the pattern is found, the search() method returns a match object.

Q4. How do you get the actual strings that match the pattern from a Match object?

If a match object is not the null value None, we can call grou() on the match object to return the match result.

Q5. In the regex created from r'(\d\d\d)-(\d\d\d-\d\d\d\d)', what does group 0 cover? Group 1? Group 2?

Group 0: (\d\d\d)-(\d\d\d-\d\d\d\d) Group 1: (\d\d\d) Group 2: (\d\d\d-\d\d\d\d)

Q6. Parentheses and periods have specific meanings in regular expression syntax. How would you specify that you want a regex to match actual parentheses and period characters?

Using r'.()'

Q7. The findall() method returns a list of strings or a list of tuples of strings. What makes it return one or the other?

When called on a regex with no groups, such as \d\d\d-\d\d\d-\d\d\d\d, the method findall() returns a list of string matches, such as ['(\d\d\d)-(\d\d\d\d-\d\d\d)', '(\d\d\d)-(\d\d\d\d-\d\d\d)'].

When called on a regex that has groups, such as (\d\d\d)-(\d\d\d)-(\d\d\d\d), the method findall() returns a list of tuples of strings(one string for each group), such as [('415', '555', '9999'),('212', '555','0000')].

Q8. What does the | character signify in regular expressions?

The | character is called a pipe. You can use it anywhere you want to match one of many expressions. For example, the regular expressions r'Batman|Tina Fey' will match either 'Batman' or 'Tina Fey'.

Q9. What two things does the ? character signify in regular expressions?

The ? is saying "Match zero or one of the group preceding this question mark."

Q10. What is the difference between the + and \* characters in regular expressions?

The \* means "match zero or more - the group that precedes the star can occur any number of times in the text. It can be completely absent or repeated over and over again."

While \* means "match zero or more", the + means "match one or more." Unlike the star, which does not require its group to appear in the matched string, the group preceding a plus must appear at least once. It is not optional.

Q11. What is the difference between {3} and {3,5} in regular expressions?

{3} is saying "match this pattern three times." {3,5} means "match this pattern three to five times."

Q12. What do the \d, \w, and \s shorthand character classes signify in regular expressions?

a \d in a regex stands for a digit character - that is, any single numberal 0 to 9.

\w means any letter, numeric digit, or the underscore character.

\s means any space, tab, or newline character.

Q13. What do the \d, \W, and \S shorthand character classes signify in regular expressions?

\d represents any character that is not a numeric digit from 0 to 9.

\W represents any character that is not a letter, numeric digit, or the underscore character.

\S represents any character that is not a space, tab or newline.

Q14. How do you make a regular expression case-insensitive?

Pass re.IGNORECASE or re.I as a second argument to re.compile(). For example, ro = re.compile(r'robocop', re.I)

Q15. What does the . character normally match? What does it match if re.DOTALL is passed as the second argument to re.compile()?

The . character in a regular expression is called a wildcard and will match any character except for a newline.

The dot-star will match everything except a newline. By passing re.DOTALL as the second argument to re.compile(), you can make the dot character match all characters, including the newline character. For example, newlineRegex = re.compile('.\*', re.DOTALL)

Q16. What is the difference between these two: . and .\?

The dot-star uses greedy mode: It will always try to match as much text as possible. To match any and all text in a nongreedy fashion, use the dot, star, and question mark(.\*?).

In nongreedy version of the regex, Python matches the shortest possible string; In the greedy version, Python matches the longest possible string.

Q17. What is the character class syntax to match all numbers and lowercase letters?

[0-9a-z]

Q18. If numRegex = re.compile(r'\d+'), what will numRegex.sub('X', '12 drummers, 11 pipers, five rings, 3 hens') return?

X drummers, X pipers, five rings, X hens

Q19. What does passing re.VERBOSE as the second argument to re.compile() allow you to do?

To ignore whitespace and comments inside the regular expression string.

Q20. How would you write a regex that matches a number with commas for every three digits? It must match the 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)

numCommas = re.compile(r'(^\d{1,3})(,\d{3})\*$') numCommas.search('12,34,567').group()

Q21. How would you write a regex that matches the full name of someone whose last name is Nakamoto? 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:

'Satoshi Nakamoto'

'Alice Nakamoto'

'Robocop Nakamoto'

but not the following:

'satoshi Nakamoto' (where the first name is not capitalized)

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

'Nakamoto' (which has no first name)

'Satoshi nakamoto' (where Nakamoto is not capitalized)

fullName = re.compile(r'[A-Z]\w [A-Z]\w') mo = fullName.findall('Satoshi Nakamoto, satoshi Nakamoto, Alice Nakamoto, Nakamoto, Satoshi nakamoto, Robocop Nakamoto') mo.group()

Q22. 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.'

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

senRegex.findall('''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.''')

Test result: [('Alice', 'eats', 'apples'), ('Bob', 'pets', 'cats'), ('Carol', 'throws', 'baseballs'), ('Alice', 'throws', 'Apples'), ('BOB', 'EATS', 'CATS')]