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

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

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

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

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?

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

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

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

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

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

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

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

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

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

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

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

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

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

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)

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)

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

1. The feature responsible for generating Regex objects in Python is the `re` module.

2. Raw strings (prefixed with 'r') are often used in Regex objects because they treat backslashes as literal characters. Since backslashes are a common escape character in regular expressions, using a raw string helps avoid conflicts with Python's string interpretation. For example, in a raw string, `'\n'` is treated as a backslash followed by 'n', rather than an escape sequence for a newline character.

3. The `search()` method in the `re` module returns a match object if the pattern is found in the string, and `None` if no match is found.

4. To get the actual strings that match the pattern from a Match object, you can use the `group()` method. For example, if you have a match object `match`, you can retrieve the matched string with `match.group()` or specify a particular group using `match.group(n)`.

5. In the regex `r'(\d\d\d)-(\d\d\d-\d\d\d\d)'`:

- Group 0 covers the entire match of the pattern.

- Group 1 covers the first set of three digits enclosed in parentheses.

- Group 2 covers the second set of three digits followed by a hyphen and four more digits.

6. In standard expression syntax, you can use backslashes to escape parentheses and periods if you want to match them literally. For example, to match a real parenthesis, you can use '\(' and '\)' in your regex. To match a real period, you can use '\.'.

7. The `findall()` method returns a list of strings when the regular expression contains no capturing groups (parentheses). If there are capturing groups in the regex, it returns a list of tuples, where each tuple represents a match, and the elements of the tuple correspond to the captured groups within that match.

8. In regular expressions, the '|' character is used as an alternation operator, meaning "or." It allows you to specify multiple alternatives, and it matches any of the alternatives. For example, the pattern `A|B` would match either 'A' or 'B'.

9. In regular expressions, the '.' (period) character is a metacharacter that matches any single character except a newline. It is often used as a wildcard character to represent any character in a pattern.

10. In regular expressions:

- The '+' character matches one or more occurrences of the preceding element. For example, 'a+' would match one or more consecutive 'a' characters.

- The '\*' character matches zero or more occurrences of the preceding element. For example, 'a\*' would match zero or more consecutive 'a' characters.

11. In regular expressions, `{4}` specifies that the preceding element should be matched exactly four times, and `{4,5}` specifies that it should be matched between four and five times. So, `{4}` matches exactly four occurrences, while `{4,5}` matches between four and five occurrences.

12. In regular expressions, the shorthand character classes signify the following:

- `\d` matches any digit (equivalent to `[0-9]`).

- `\w` matches any word character (alphanumeric character plus underscore, equivalent to `[a-zA-Z0-9\_]`).

- `\s` matches any whitespace character (e.g., space, tab, newline).

13. In regular expressions, the shorthand character classes with uppercase letters signify the inverse of their lowercase counterparts:

- `\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. `.\*` matches any character (except a newline) greedily, and `.\*?` matches any character (except a newline) reluctantly. The difference is in how they handle matching. `.\*` matches as much as possible, while `.\*?` matches as little as possible to satisfy the rest of the pattern.

15. To match both numbers and lowercase letters with a character class, you can use `[0-9a-z]`. This character class matches any digit or lowercase letter.

16. To make a regular expression case insensitive, you can use the `re.IGNORECASE` flag or `re.I` as the second argument in `re.compile()`. For example: `regex = re.compile(r'pattern', re.IGNORECASE)`.

17. In regular expressions, the `.` character normally matches any character except a newline. If `re.DOTALL` (or `re.S`) is passed as the second argument in `re.compile()`, it makes the `.` character match any character, including newline characters.

18. `numRegex.sub('X', '11 drummers, 10 pipers, five rings, 4 hen')` will return the string `'X drummers, X pipers, five rings, X hen'`. It replaces all sequences of one or more digits with 'X'.

19. Passing `re.VERBOSE` as the second argument to `re.compile()` allows you to use whitespace and comments within the regular expression to make it more readable. It ignores whitespace and treats '#' as the start of a comment until the end of the line.

20. To match a number with commas for every three digits, you can use the following regex pattern: `r'\d{1,3}(,\d{3})\*'`. This pattern will match numbers such as '42', '1,234', and '6,368,745' where commas are used to separate every three digits. It won't match numbers like '12,34,567' (which has only two digits between the commas) or '1234' (which lacks commas).

21. To write a regex that matches the full name of someone whose last name is 'Watanabe' with a capitalized first name, you can use the following pattern:

```python

r'[A-Z][a-zA-Z]\*\sWatanabe'

```

Explanation:

- `[A-Z]` matches the first character of the first name, ensuring it's capitalized.

- `[a-zA-Z]\*` matches the rest of the first name, allowing any combination of lowercase and uppercase letters.

- `\s` matches the space between the first and last name.

- `Watanabe` matches the last name.

This regex will match names like 'Haruto Watanabe', 'Alice Watanabe', and 'RoboCop Watanabe' but not names like 'haruto Watanabe', 'Mr. Watanabe', 'Watanabe', or 'Haruto watanabe' as specified.

22. To write a case-insensitive regex that matches sentences following the given pattern, you can use the following pattern:

```python

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

```

Explanation:

- `^` and `$` anchor the regex to the start and end of the string to ensure it matches the entire sentence.

- `(Alice|Bob|Carol)` matches one of the given names.

- `\s+` matches one or more whitespace characters (spaces or tabs) between words.

- `(eats|pets|throws)` matches one of the given verbs.

- `(apples|cats|baseballs)` matches one of the given objects.

- `\.` matches the period at the end of the sentence.

This regex will match sentences like 'Alice eats apples.', 'Bob pets cats.', 'Carol throws baseballs.', and 'Alice throws Apples.' while being case-insensitive as specified.