Answers

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

2. Raw strings (`r'some\_string'`) are often used in Regex objects to avoid having to escape backslashes. Since backslashes are commonly used in regular expressions for special characters, using a raw string ensures that the backslashes are treated as literal characters and not interpreted as escape characters by Python.

3. The `search()` method in regular expressions returns a Match object if the pattern is found in the searched string. If there is no match, it returns `None`.

4. From a Match object, you can use the `group()` method to get the actual strings that match the pattern. Calling `group(0)` returns the entire match, while calling `group(n)` returns the string that matches the nth group in the pattern.

5. In the regex `r'(\d\d\d)-(\d\d\d-\d\d\d\d)'`, group zero (`group(0)`) covers the entire match. Group 2 (`group(2)`) covers the second group within the parentheses, which corresponds to the part after the first hyphen. Group 1 (`group(1)`) covers the first group within the parentheses, which corresponds to the part before the first hyphen.

6. To tell a regex that you want it to fit real parentheses and periods, you can use the backslash (`\`) character to escape them. For example, to match a literal parenthesis, you would use `\(`, and to match a literal period, you would use `\.`.

7. The `findall()` method returns a string list if the regular expression pattern does not have any capturing groups. If the pattern has one or more capturing groups, it returns a list of string tuples, where each tuple represents a match and contains the captured groups.

8. In standard expressions, the `|` character is used as an OR operator. It allows you to specify multiple alternative patterns, and the expression matches if any of the alternatives are found.

9. In regular expressions, the character `.` (dot) is a metacharacter that matches any character except a newline character.

10. In regular expressions, the `+` character is a quantifier that matches one or more occurrences of the preceding element. The `\*` character is also a quantifier but matches zero or more occurrences of the preceding element.

11. In regular expressions, `{4}` specifies that the preceding element should occur exactly four times. `{4,5}` specifies that the preceding element should occur between four and five times, inclusive.

12. In regular expressions, the shorthand character class `\d` matches any digit (0-9). `\w` matches any alphanumeric character (a-z, A-Z, 0-9, and underscore `\_`). `\s` matches any whitespace character (space, tab, newline).

13. In regular expressions, the shorthand character class `\D` matches any non-digit character. `\W` matches any non-alphanumeric character. `\S` matches any non-whitespace character.

14. The `.\*?` and `.\*` expressions are used for matching patterns. `.\*?` is a non-greedy match and matches as few characters as possible, while `.\*` is a greedy match and matches as many characters as possible.

15. The syntax for matching both numbers and lowercase letters with a character class is `[0-9a-z]`. This character class will match any character that is either a number (0-9) or a lowercase letter (a-z).

16. To make a normal expression case-insensitive in Python, you can pass the `re.IGNORECASE` flag as the second argument to `re.compile()` or include `(?i)` at the

beginning of the regular expression pattern.

17. The `.` character normally matches any character except a newline character. If the `re.DOTALL` flag is passed as the second argument to `re.compile()`, the `.` character will also match a newline character.

18. If `numRegex = re.compile(r'\d+')`, then `numRegex.sub('X', '11 drummers, 10 pipers, five rings, 4 hens')` will return the string `'X drummers, X pipers, five rings, X hens'`. It replaces all occurrences of one or more digits with the letter `'X'`.

19. Passing `re.VERBOSE` as the second argument to `re.compile()` allows you to include whitespace and comments within the regular expression pattern. It improves readability and allows you to add comments to explain the pattern.

20. To match a number with commas for every three digits, you can use the regex pattern `r'^\d{1,3}(,\d{3})\*$'`. This pattern matches strings that start with 1 to 3 digits, followed by zero or more occurrences of a comma and exactly 3 digits. The `^` and `$` anchors ensure that the entire string matches the pattern.

21. To match the full name of someone whose last name is Watanabe, assuming the first name is always capitalized and consists of one word, you can use the regex pattern `r'[A-Z][a-zA-Z]\* Watanabe'`. This pattern matches a capital letter followed by zero or more lowercase or uppercase letters, followed by a space and the string 'Watanabe'.

22. To match 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 either apples, cats, or baseballs; and the sentence ends with a period, you can use the case-insensitive regex pattern `r'^(Alice|Bob|Carol) (eats|pets|throws) (apples|cats|baseballs)\.$'`. The `^` and `$` anchors ensure that the entire string matches the pattern, and the options within parentheses are matched using the `|` (OR) operator.