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

* The feature responsible for generating Regex objects is called the ‘**re’** module.

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

* Raw strings are often used in regex objects because regular expressions frequently include special characters that have a specific meaning in string literals. For example, backslashes are used to escape special characters in string literals, but in regular expressions, they are often used to introduce special character classes, such as \d for digits or \s for whitespace.

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

* The **search()** method in Python's **re** module returns a **Match** object if it finds a match of the regular expression pattern within the searched string, and **None** if no match is found.

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

* If the regular expression pattern contains capturing groups (i.e., portions of the pattern enclosed in parentheses), you can use the **group()** method with an argument to retrieve the text that matched a specific group. The **group(0)** method returns the entire matched text.

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 (i.e., match.group(0)) covers the entire string that matched the regular expression pattern, including both capturing groups.
* Group 1 (i.e., match.group(1)) covers the first capturing group, which matches three consecutive digits.
* Group 2 (i.e., match.group(2)) covers the second capturing group, which matches three digits separated by a hyphen, followed by four more digits.
* For ex: the regular expression pattern matches the phone number "123-456-7890". The **group(0)** method returns the entire matched string ("123-456-7890"), while **group(1)** returns "123" and **group(2)** returns "456-7890".

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?

* In regular expressions, parentheses have a special meaning as a way to create a capturing group, while periods have a special meaning as a wildcard character that matches any single character except for a newline. To match literal parentheses and periods, you need to escape them with a backslash.

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

* The **findall()** method returns a list of strings or a list of tuples of strings depends on whether the regular expression used contains one or more capturing groups.
* For ex:
  + The input string is **"The price of a banana is 25 rupees and an apple is 30 rupees."**, then **findall()** will return **['25', '30']**, because the regular expression matches one or more digits.
  + the input string is **"The price of a banana is 25 cents and an apple is 30 cents."**, then **findall()** will return **[('banana', '25'), ('apple', '30')]’.**

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

* In regular expressions, the **|** character is called the "pipe" or "alternation" operator, and it is used to represent a logical OR between multiple expressions.

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

* In regular expressions, the ‘**.’** (dot) character is called the "dot" or "period" operator, and it matches any single character except for a newline character. It allows you to match any character in a string, including letters, numbers, punctuation marks, and whitespace characters.

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

* The difference between the **+** and **\*** operators is that the **+** operator matches one or more occurrences of the preceding expression, while the **\*** operator matches zero or more occurrences of the preceding expression.

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

* The difference between **{4}** and **{4,5}** is that **{4}** specifies an exact number of occurrences, while **{4,5}** specifies a range of occurrences.

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

* **\d** represents any digit character.
* **\w** represents any "word" character.
* **\s** represents any whitespace character.

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

* **\d** represents any digit character.
* **\w** represents any "word" character.
* **\s** represents any whitespace character.

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

* The difference between **.\*** and **.\*?** is that **.\*** is a greedy quantifier that matches as many characters as possible, while **.\*?** is a non-greedy or lazy quantifier that matches as few characters as possible.

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

🡪 import re

text = "The quick brown fox jumps over the lazy dog 123 times."

matches = re.findall(r'[0-9a-z]+', text)

print(matches)

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

* To make a regular expression case-insensitive in Python, we can use the **re.IGNORECASE** flag or **re.I** shorthand.

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

* In Python's regular expression syntax, the . (dot) character normally matches any character except for a newline (\n). This means that if we use the regular expression r'.', it will match any single character except for a newline.
* However, if we pass the re.DOTALL flag as the second argument to the re.compile() function or use it directly in the re.findall(), re.search(), or re.match() functions, the . character will match any character including a newline.

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

* **numRegex.sub('X', '11 drummers, 10 pipers, five rings, 4 hen')**, the regular expression matches the substrings **'11'**, **'10'**, and **'4'**, and replaces them with **'X'**. The resulting string is **'X drummers, X pipers, five rings, X hen'**.

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

* Passing **re.VERBOSE** as the second argument to **re.compile()** allows us to write more readable and maintainable regular expressions by ignoring whitespace and adding comments.

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)

🡪 Code: import re

regex = re.compile(r'^[0-9]{1,3}(,[0-9]{3})\*$')

strings = ['42', '1,234', '6,368,745', '12,34,567', '1234']

for s in strings:

match = regex.match(s)

if match:

print(f"{s} matches")

else:

print(f"{s} does not match")

🡪 output: 42 matches

1,234 matches

6,368,745 matches

12,34,567 does not match

1234 does not match

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)

🡪 Code: import re

regex = re.compile(r'^[A-Z][a-z]\*\sWatanabe$')

strings = ['Haruto Watanabe', 'Alice Watanabe', 'RoboCop Watanabe',

'haruto Watanabe', 'Mr. Watanabe', 'Watanabe', 'Haruto watanabe']

for s in strings:

match = regex.match(s)

if match:

print(f"{s} matches")

else:

print(f"{s} does not match")

🡪 Output: Haruto Watanabe matches

Alice Watanabe matches

RoboCop Watanabe matches

haruto Watanabe does not match

Mr. Watanabe does not match

Watanabe does not match

Haruto watanabe does not match\

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.'

> Code : import re

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

strings = ['Alice eats apples.', 'Bob pets cats.', 'Carol throws baseballs.',

'Alice throws Apples.', 'BOB EATS CATS.', 'RoboCop eats apples.',

'ALICE THROWS FOOTBALLS.', 'Carol eats 7 cats.']

for s in strings:

match = regex.match(s)

if match:

print(f"{s} matches")

else:

print(f"{s} does not match")

🡪 Output: Alice eats apples. matches

Bob pets cats. matches

Carol throws baseballs. matches

Alice throws Apples. matches

BOB EATS CATS. matches

RoboCop eats apples. does not match

ALICE THROWS FOOTBALLS. does not match

Carol eats 7 cats. does not match