PYTHON ASSIGNMENT – 7

QUESTIONS ;

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.'

SOLUTIONS:

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

2. Raw strings (prefixed with **r**) are often used in Regex objects to avoid the interpretation of escape characters. For example, **\n** in a regular string is interpreted as a newline character, but in a raw string (**r'\n'**), it is treated as the literal characters backslash and 'n'.

3. The **search()** method returns a Match object if a match is found, and **None** if no match is found.

4. From a Match object, you can get the actual strings that match the pattern using the **group()** method. **group(0)** returns the entire matched string, and you can use additional arguments to retrieve specific groups.

5. In the regex created from **r'(\d\d\d)-(\d\d\d-\d\d\d\d)'**:
    - Group 0 covers the entire matched string.
    - Group 1 covers the first three digits (area code).
    - Group 2 covers the remaining seven digits.

6. To match real parentheses and periods in a regex, you can use a backslash (****). For example, to match a literal period, use **\.**.

7. The **findall()** method returns a list of all occurrences of the pattern in the string. If the pattern contains capturing groups, it returns a list of tuples, where each tuple represents a match and contains the matched groups.

8. In standard expressions, the **|** character means "or." It is used to specify alternatives. For example, **A|B** would match either 'A' or 'B'.

9. In regular expressions, the **+** character means "one or more occurrences," while the ***** character means "zero or more occurrences."

10. **{4}** in a regular expression means exactly four occurrences, while **{4,5}** means between four and five occurrences.

11. In regular expressions:
    - **\d** represents any digit (0-9).

- **\w** represents any word character (alphanumeric + underscore).
- **\s** represents any whitespace character (space, tab, newline).

12. In regular expressions:
    - **\D** represents any non-digit.
    - **\W** represents any non-word character.
    - **\S** represents any non-whitespace character.

13. The difference between **.*?** and **.*** is in their greediness. **.*?** is the non-greedy version, matching as little as possible, while **.*** is the greedy version, matching as much as possible.

14. The syntax for matching both numbers and lowercase letters with a character class is **[0-9a-z]**.

15. To make a regular expression case insensitive, you can pass the `re.IGNORECASE` or **re.I** flag as the second argument to **re.compile()**.

16. The **.** character normally matches any character except a newline. If **re.DOTALL** is passed as the second argument in **re.compile()**, the **.** character will match any character, including a newline

17. If **numReg = re.compile(r'\d+')**, **numRegex.sub('X', '11 drummers, 10 pipers, five rings, 4 hen')** will return the string '**X drummers, X pipers, five rings, X hen**'.

18. Passing **re.VERBOSE** as the second argument to **re.compile()** allows using whitespace and comments within the regular expression for better readability.

19. To write a regex that matches a number with a comma for every three digits, you can use the following pattern: **r'\d{1,3}(,\d{3})*'**.

20. To write a regex that matches the full name of someone whose last name is Watanabe, you can use the pattern: **r'[A-Z][a-zA-Z]*\sWatanabe'**.

21. To write a regex that matches a sentence meeting specific criteria, you can use the pattern: **r'(Alice|Bob|Carol)\s(eats|pets|throws)\s(apples|cats|baseballs)\.'**, with the **re.IGNORECASE** flag.