**Assignment-7**

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

The feature responsible for generating Regex objects in Python is called the "re" module.

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

Raw strings (strings prefixed with r) are often used in Regex objects to avoid unintended escape character interpretation. In Python, escape characters like \n (newline), \t (tab), \b (backspace), etc., have special meanings and are interpreted accordingly in regular strings.

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

The search() method of a Regex object in Python's re module returns a match object if a match is found, or None if no match is found.

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

To get the actual strings that match the pattern from a match object, you can use the group() method. The group() method without any arguments returns the entire substring that matched the pattern.

1. 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?

In the regex created from the pattern r'(\d\d\d)-(\d\d\d-\d\d\d\d)', the groups are defined by the parentheses.

* Group 0: Group 0 represents the entire match, including all the characters matched by the pattern. In this case, it covers the entire string that matches the pattern, which consists of the first three digits followed by a hyphen, and then the next three digits followed by another hyphen and four digits. So, in the given regex, group 0 covers the entire matched string.
* Group 1: Group 1 corresponds to the first capturing group defined by the first set of parentheses (\d\d\d). It matches and captures three consecutive digits. In the given regex, group 1 covers the first three digits.
* Group 2: Group 2 corresponds to the second capturing group defined by the second set of parentheses (\d\d\d-\d\d\d\d). It matches and captures a pattern consisting of three digits, a hyphen, and four digits. In the given regex, group 2 covers the portion of the matched string following the hyphen.

1. 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?

To tell a regex that you want to match literal parentheses and periods instead of using their special meanings, you can use the backslash \ to escape them. By placing a backslash before a special character, you indicate that you want to treat it as a literal character.

1. 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 in regular expressions returns either a string list or a list of string tuples depending on whether the regex pattern contains one or more capturing groups.

* If the regex pattern does not contain any capturing groups (defined by parentheses), findall() will return a string list. Each element in the list represents a match of the pattern in the input string.
* If the regex pattern contains one or more capturing groups, findall() will return a list of string tuples. Each tuple represents a match of the entire pattern, and each element within the tuple corresponds to a capturing group in the pattern.

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

In regular expressions, the | character is known as the pipe or vertical bar and is used as an OR operator. It allows you to specify multiple alternatives within a pattern, indicating that any one of the alternatives can be matched.

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

In regular expressions, the . (dot) character is known as a metacharacter and has a special meaning. It is used to represent any single character except a newline character (\n).

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

* + (Plus): Matches one or more occurrences of the preceding element.
* \* (Asterisk): Matches zero or more occurrences of the preceding element.

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

* {4}: Matches exactly 4 occurrences of the preceding element.
* {4,5}: Matches a range of 4 to 5 occurrences of the preceding element.

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

expressions?

* \d: Represents any digit character (0-9). It is equivalent to the character class [0-9]. For example, the pattern \d+ will match one or more consecutive digits.
* \w: Represents any alphanumeric character (letter or digit) or an underscore. It is equivalent to the character class [a-zA-Z0-9\_]. For example, the pattern \w+ will match one or more consecutive alphanumeric characters or underscores.
* \s: Represents any whitespace character, including spaces, tabs, and newlines. It is equivalent to the character class [\t\n\r\f\v ]. For example, the pattern \s+ will match one or more consecutive whitespace characters.

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

In regular expressions, the \D, \W, and \S shorthand character classes are negations of the \d, \w, and \s character classes, respectively. They represent any character that is not in the corresponding character class. Here's what each of them signifies:

* \D: Represents any character that is not a digit. It is equivalent to the character class [^0-9]. For example, the pattern \D+ will match one or more consecutive non-digit characters.
* \W: Represents any character that is not alphanumeric or an underscore. It is equivalent to the character class [^a-zA-Z0-9\_]. For example, the pattern \W+ will match one or more consecutive non-alphanumeric characters or underscores.
* \S: Represents any character that is not whitespace. It is equivalent to the character class [^ \t\n\r\f\v]. For example, the pattern \S+ will match one or more consecutive non-whitespace characters.

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

The .\*? expression is a non-greedy quantifier. It matches as few characters as possible to satisfy the overall pattern. It will stop matching as soon as the subsequent part of the pattern can be matched. On the other hand, .\* is a greedy quantifier that matches as many characters as possible, only stopping when it's necessary to match the subsequent part of the pattern.

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

To match both numbers and lowercase letters using a character class in a regular expression, you can use the range notation within the character class.

The syntax for matching numbers (0-9) and lowercase letters (a-z) in a character class is:

[0-9a-z]

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

To make a regular expression case-insensitive in Python, you can use the re.IGNORECASE flag or the re.I flag. These flags can be passed as the second argument to the re.compile() function or as the optional third argument to other regular expression functions like re.search(), re.findall(), etc.

Here's the procedure for making a regular expression case-insensitive:

* Import the re module: import re
* Define your regular expression pattern.
* Compile the pattern with the re.IGNORECASE flag or re.I flag using the re.compile() function.
* Use the compiled pattern to perform regular expression operations like matching, searching, or replacing.

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

In a regular expression, the . (dot) character normally matches any character except a newline character (\n). It matches any single character in the input string.

However, if the re.DOTALL flag is passed as the second argument to re.compile(), the behavior of the . character changes. When re.DOTALL is enabled, the . character matches any character including a newline character (\n).

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

If numReg = re.compile(r'\d+') and you use numRegex.sub('X', '11 drummers, 10 pipers, five rings, 4 hen'), it will return the following string:

'X drummers, X pipers, five rings, X hen'

1. 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 you to use the verbose mode in regular expressions. The verbose mode enables you to write more readable and organized regular expressions by ignoring whitespace and adding comments.

1. 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)

To write a regex that matches a number with commas for every three digits, you can use the following pattern:

import re

pattern = re.compile(r'^\d{1,3}(,\d{3})\*$')

result = re.match(pattern, '42')

print(result) # Match

result = re.match(pattern, '1,234')

print(result) # Match

result = re.match(pattern, '6,368,745')

print(result) # Match

result = re.match(pattern, '12,34,567')

print(result) # No match

result = re.match(pattern, '1234')

print(result) # No match

1. 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)

import re

pattern = re.compile(r'^[A-Z][a-zA-Z]\* Watanabe$')

result = re.match(pattern, 'Haruto Watanabe')

print(result) # Match

result = re.match(pattern, 'Alice Watanabe')

print(result) # Match

result = re.match(pattern, 'RoboCop Watanabe')

print(result) # Match

result = re.match(pattern, 'haruto Watanabe')

print(result) # No match

result = re.match(pattern, 'Mr. Watanabe')

print(result) # No match

result = re.match(pattern, 'Watanabe')

print(result) # No match

result = re.match(pattern, 'Haruto watanabe')

print(result) # No match

1. 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.’

import re

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

result = re.match(pattern, 'Alice eats apples.')

print(result) # Match

result = re.match(pattern, 'Bob pets cats.')

print(result) # Match

result = re.match(pattern, 'Carol throws baseballs.')

print(result) # Match

result = re.match(pattern, 'Alice throws Apples.')

print(result) # Match

result = re.match(pattern, 'BOB EATS CATS.')

print(result) # Match

result = re.match(pattern, 'RoboCop eats apples.')

print(result) # No match

result = re.match(pattern, 'ALICE THROWS FOOTBALLS.')

print(result) # No match

result = re.match(pattern, 'Carol eats 7 cats.')

print(result) # No match