Python Assignment - 7

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

=> ‘re’ module.

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

=> Raw strings are often used in regular expressions (regex) because they allow you to specify patterns more conveniently without having to escape special characters. In regex patterns, certain characters have special meanings, such as ‘\’, [‘,’ ], (‘,’ ), ‘|’, ’.’ , ’\*’ , ‘+’ , ‘?’ , {‘,’ }, ‘^’, ‘$’ , and others.

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

=> The search() method of a regular expression (regex) object in Python returns a match object if a match is found in the given string, or None if no match is found. The match object represents the first occurrence of a pattern match in the string.

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

=> To retrieve the actual strings that match the pattern from a match object in Python, you can use the ‘group()’ method. The ‘group()’ method of a match object returns the matched substring(s) based on 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?**

=>

1. **group(0)**: It represents the entire matched substring. In this case, it corresponds to the entire pattern (\d\d\d)-(\d\d\d-\d\d\d\d). Accessing group(0) would return the entire match.

2. **group(1)**: It represents the first capturing group in the pattern, which is (\d\d\d). It captures a sequence of three digits. Accessing group(1) would return the matched substring for this group.

3. **group(2)**: It represents the second capturing group in the pattern, which is (\d\d\d-\d\d\d\d). It captures a sequence of three digits, followed by a hyphen, and then four digits. Accessing group(2) would return the matched substring for this group.

**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 and periods have special meanings as metacharacters. To match literal parentheses and periods in a regular expression pattern, you can use the backslash \ character to escape them. This tells the regex engine to treat them as literal characters rather than metacharacters with special meanings.

Here's how you can escape parentheses and periods in a regex pattern:

1. **Escaping** **parentheses**: To match literal parentheses, use the ‘\( ‘and ‘\’)’ escape sequences. The backslash ‘\’ before the parentheses tells the regex engine to interpret them as literal parentheses.
2. **Escaping** **periods**: To match a literal period (dot), use the ‘\.’, which tells the regex engine to interpret it as a literal period rather than a metacharacter matching any character.

**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 in Python's regular expression module (re) returns different types of results based on the structure of the regular expression pattern.

1. If the pattern contains no capturing groups:

When the regular expression pattern does not contain any capturing groups (i.e., no parentheses), findall() returns a list of strings. Each element in the list represents a separate match found in the input string.

2. If the pattern contains capturing groups:

When the regular expression pattern contains one or more capturing groups (i.e., parentheses), findall() returns a list of tuples. Each tuple in the list represents a single match, and each element within the tuple corresponds to a capturing group in the pattern.

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

=> in regular expression ‘|’ character, known as the pipe or vertical bar, is used as the logical OR operator.

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

=> In regular expression , the dot character(‘.’) is a metacharacter that matches any single character except for a new line.

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

=> The differences between + and \* are:

1. **‘+’ (Plus):**

* The + quantifier matches one or more occurrences of the preceding element.
* It requires at least one occurrence of the preceding element for a match.

2. **‘\*’ (Asterisk):**

* The \* quantifier matches zero or more occurrences of the preceding element.
* It allows for optional matches, including cases where the preceding element is absent.

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

=> The difference between {4} and {4,5} are:

1. **{4} :**

* The {4} quantifier matches exactly four occurrences of the preceding element.
* It specifies a fixed number of occurrences.

2. **{4,5} :**

* The {4,5} quantifier matches a range of occurrences of the preceding element, from a minimum of four to a maximum of five occurrences.
* It allows for a range of occurrences to be matched.

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

=>

1. **\d :**

* The \d shorthand character class represents any digit character from 0 to 9.
* It is equivalent to the character range [0-9].

2. **\w :**

* The \w shorthand character class represents any alphanumeric character (letter or digit) and underscore \_.
* It is equivalent to the character range [a-zA-Z0-9\_].

3. **\s :**

* The \s shorthand character class represents any whitespace character, including spaces, tabs, and newline characters.
* It is equivalent to the character range [ \t\n\r\f\v].

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

=>

1. **\d :**

* The \d shorthand character class represents any digit character from 0 to 9.
* It is equivalent to the character range [0-9].

2. **\w :**

* The \w shorthand character class represents any alphanumeric character (letter or digit) and underscore \_.
* It is equivalent to the character range [a-zA-Z0-9\_].

3. **\s :**

* The \s shorthand character class represents any whitespace character, including spaces, tabs, and newline characters.
* It is equivalent to the character range [ \t\n\r\f\v].

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

=> The differences between ‘.\*?’ and ‘.\*’

1. **‘.\*?’ (non-greedy or lazy match):**

* ‘.\*?’ is a non-greedy or lazy match that matches as few characters as possible.
* It will match zero or more occurrences of any character (except newline characters) until the next part of the pattern is found.

2. **‘.\*’ (greedy match):**

* ‘.\*’ is a greedy match that matches as many characters as possible.
* It will match zero or more occurrences of any character (except newline characters) until the end of the string or until the next part of the pattern is found.

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

=> The syntax for matching both numbers and lowercase letters with a character class is:

[0-9a-z]

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

=> To make a normal expression in regex case-insensitive, you can use the i flag or the (?i) modifier. Here's the procedure:

* Start by defining your regular expression pattern as you normally would, using the appropriate syntax for your specific use case.
* After the closing delimiter of the pattern (typically a forward slash /), add the i flag. This flag tells the regex engine to perform a case-insensitive match.

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

=> In regular expressions, the . character normally matches any character except a newline character (\n). It is often referred to as a "wildcard" character.

However, if the re.DOTALL flag is passed as the second argument to the re.compile() function in Python, then the behavior of the . character changes. The re.DOTALL flag, also known as the re.S flag, allows the . character to match any character, including newline characters (\n).

By default, without the re.DOTALL flag, the . character matches any character except a newline character. But when re.DOTALL is enabled, the . character will match any character, including newline characters.

**18. 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 numReg.sub('X', '11 drummers, 10 pipers, five rings, 4 hen'), it will return the following string: '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() in Python allows you to create regular expressions with enhanced readability and comments. It enables the use of whitespace and comments within the regular expression pattern without affecting the pattern's functionality.

**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)**

=> To match a number with commas for every three digits, you can use the following regular expression: ‘^\d{1,3}(,\d{3})\*$’

**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)**

=> The regular expression is : ^[A-Z][a-zA-Z]\*\sWatanabe$

**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.'**

=> The regular expression is : ^(Alice|Bob|Carol)\s(eats|pets|throws)\s(apples|cats|baseballs)\.$