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

ANS: mport re

re.compile(pattern) - > The re.compile() function returns Regex objects.

In [2]:

**import** re

re**.**compile("string")

Out[2]:

re.compile(r'string', re.UNICODE)

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

ANS: Raw strings are used so that backslashes do not have to be escaped

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

ANS: The search() method searches a string for a specified value, and returns the position of the match.

The search value can be string or a regular expression.

This method returns -1 if no match is found

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

ANS: Calling matchingObject.group() will return the string.

In [3]:

phoneNumRegex **=** re**.**compile(r'\d\d\d-\d\d\d-\d\d\d\d')

mo **=** phoneNumRegex**.**search('My number is 415-555-4242.')

print('Phone number found: ' **+** mo**.**group())

Phone number found: 415-555-4242

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?

ANS: regex: (\d\d\d)-(\d\d\d-\d\d\d\d)

The first set of parentheses in a regex string will be group 1.

The second set will be group 2.

By passing the integer 1 or 2 to the group() match object method, you can grab different parts of the matched text.

Passing 0 or nothing to the group() method will return the entire matched text

In [4]:

phoneNumRegex **=** re**.**compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')

mo **=** phoneNumRegex**.**search('My number is 415-555-4242.')

mo**.**group(1)

Out[4]:

'415'

In [5]:

mo**.**group(2)

Out[5]:

'555-4242'

In [6]:

mo**.**group(0)

Out[6]:

'415-555-4242'

mo**.**group()

Out[7]:

'415-555-4242'

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?

ANS: Periods and parentheses can be escaped with a backslash: \., \(, and \).

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

ANS: If the regex has no groups, a list of strings is returned. If the regex has groups, a list of tuples of strings is returned.

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

ANS: The | character is called a pipe. You can use it anywhere you want to match one of many expressions.

For example, the regular expression r'Banana|Apple Fruit' will match either 'Banana' or 'Apple Fruit'.

When both Banana and Apple Fruit occur in the searched string, the first occurrence of matching text will be

returned as the Match object.

The | character signifies matching “either, or” between two groups

In [14]:

fruitRegex **=** re**.**compile (r'Banana|Apple Fruit')

mo1 **=** fruitRegex**.**search('Banana and Apple Fruit')

mo1**.**group()

Out[14]:

'Banana'

In [10]:

mo2 **=** fruitRegex**.**search('Apple Fruit and Banana')

mo2**.**group()

Out[10]:

'Apple Fruit'

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

ANS: The ? character can either mean “match zero or one of the preceding group”.

Sometimes there is a pattern that you want to match only optionally. That is, the regex should find a match regardless

of whether that bit of text is there. The ? character flags the group that precedes it as an optional part of the

pattern.

In [15]:

batRegex **=** re**.**compile(r'Bat(wo)?man')

mo1 **=** batRegex**.**search('The Adventures of Batman')

mo1**.**group()

Out[15]:

'Batman'

In [16]:

mo2 **=** batRegex**.**search('The Adventures of Batwoman')

mo2**.**group()

Out[16]:

'Batwoman'

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

ANS: The + matches one or more. The \* matches zero or more

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

ANS: The {3} matches exactly three instances of the preceding group.

The {3,5} matches between three and five instances

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

ANS: The \d, stands for single digit, Any numeric digit from 0 to 9

\w, stands for single word, Any letter, numeric digit, or the underscore character. (Think of this as matching “word”

characters.)

\s stands for single space character, Any space, tab, or newline character. (Think of this as matching “space”

characters.)

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

ANS: \D - > Any character that is not a numeric digit from 0 to 9.

\W - > Any character that is not a letter, numeric digit, or the underscore character.

\S - > Any character that is not a space, tab, or newline.

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

ANS: .\* - > The dot-star uses greedy mode: It will always try to match as much text as possible.

.\*? - > To match any and all text in a non-greedy fashion, use the dot, star, and question mark (.\*?). Like with braces,

the question mark tells Python to match in a non-greedy way.

In [17]:

greedyRegex **=** re**.**compile(r'<.\*>')

mo **=** greedyRegex**.**search('<To serve man> for dinner.>')

mo**.**group()

Out[17]:

'<To serve man> for dinner.>'

In [18]:

nongreedyRegex **=** re**.**compile(r'<.\*?>')

mo **=** nongreedyRegex**.**search('<To serve man> for dinner.>')

mo**.**group()

Out[18]:

'<To serve man>'

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

ANS: Either [0-9a-z] or [a-z0-9]

In [25]:

reg1 **=** re**.**compile(r'[0-9a-z]')

reg2 **=** re**.**compile(r'[a-z0-9]')

mo1 **=** reg1**.**search('100 times I am Reading this for 100 th time')

mo1**.**group()

Out[25]:

'1'

In [26]:

reg2 **=** re**.**compile(r'[a-z0-9]')

mo1 **=** reg2**.**search('times I am Reading this for 100 th time')

mo1**.**group()

Out[26]:

't'

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

ANS: Passing re.I or re.IGNORECASE as the second argument to re.compile() will make the matching case insensitive

In [28]:

casesen **=** re**.**compile(r'machine', re**.**I)

casesen**.**search('Machine learning is part of data science')**.**group()

Out[28]:

'Machine'

In [29]:

casesen**.**search('MACHINE is learning.')**.**group()

Out[29]:

'MACHINE'

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

ANS: The . character normally matches any character except the newline character.

If re.DOTALL is passed as the second argument to re.compile(), then the dot will also match newline characters.

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

ANS: numRegex **=** re**.**compile(r'\d+')

mo **=** numRegex**.**sub('X', '11 drummers, 10 pipers, five rings, 4 hen')

mo

Out[33]:

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

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

ANS: The re.VERBOSE argument allows you to add whitespace and comments to the string passed to re.compile()

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)

ANS: '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)

In [46]:

reg1 **=** re**.**compile(r'^\d{1,3}(,\d{3})\*$')

mo1 **=** reg1**.**search('42')

mo1**.**group()

Out[46]:

'42'

In [47]:

reg1 **=** re**.**compile(r'^\d{1,3}(,\d{3})\*$')

mo1 **=** reg1**.**search('1,234')

mo1**.**group()

Out[47]:

'1,234'

In [48]:

reg1 **=** re**.**compile(r'^\d{1,3}(,\d{3})\*$')

mo1 **=** reg1**.**search('6,368,745')

mo1**.**group()

Out[48]:

'6,368,745'

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)

ANS: '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)

In [54]:

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

reg1 **=** re**.**compile(r'^\d{1,3}(,\d{haruto Watanabe3})\*$')

mo1 **=** name**.**search('Haruto Watanabe')

mo1**.**group()

Out[54]:

'Haruto Watanabe'

In [55]:

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

reg1 **=** re**.**compile(r'^\d{1,3}(,\d{3})\*$')

mo1 **=** name**.**search('Alice Watanabe')

mo1**.**group()

Out[55]:

'Alice Watanabe'

In [60]:

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

reg1 **=** re**.**compile(r'^\d{1,3}(,\d{3})\*$')

mo1 **=** name**.**search('Robocop Watanabe')

mo1**.**group()

'Robocop Watanabe'

In [61]:

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

reg1 **=** re**.**compile(r'^\d{1,3}(,\d{3})\*$')

mo1 **=** name**.**search('haruto Watanabe')

mo1**.**group()

**---------------------------------------------------------------------------**

**AttributeError** Traceback (most recent call last)

**<ipython-input-61-36eb94d34b5e>** in <module>

2 reg1 **=** re**.**compile**(r'^\d{1,3}(,\d{3})\*$')**

3 mo1 **=** name**.**search**('haruto Watanabe')**

**----> 4** mo1**.**group**()**

**AttributeError**: 'NoneType' object has no attribute 'group'

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.'

ANS: '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.'

In [63]:

name **=** re**.**compile(r'(Alice|Bob|Carol)\s(eats|pets|throws)\s(apples|cats|baseballs)\.', re**.**IGNORECASE)

mo1 **=** name**.**search('Alice eats apples.')

mo1**.**group()

Out[63]:

'Alice eats apples.'

In [64]:

name **=** re**.**compile(r'(Alice|Bob|Carol)\s(eats|pets|throws)\s(apples|cats|baseballs)\.', re**.**IGNORECASE)

mo1 **=** name**.**search('Carol throws baseballs.')

mo1**.**group()

Out[64]:

'Carol throws baseballs.'