python_basic_assignment_7

May 12, 2023

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

```
[2]: =>re.compile() is the feature responsible for generation of Regex objects.

     import re
     x = re.compile("some_random_pattern")
     type(x)
     print(x)
```

```
re.compile('some_random_pattern')
```

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

```
[ ]: => Regular expressions use the backslash character ('\') to indicate special
     ↪forms (Metacharacters) or to allow special characters (speical sequences) to
     ↪be used without invoking their special meaning.
     This collides with Python's usage of the same character for the same purpose in
     ↪string literals.
     Hence,Raw strings are used (e.g. r"\n") so that backslashes do not have to be
     ↪escaped.
```

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

```
[ ]: =>The return value of re.search(pattern,string) method is a match object if the
     ↪pattern is observed in the string else it returns a None
```

```
[ ]: 4. From a Match item, how do you get the actual strings that match the pattern?
     => For Matched items group() methods returns actual strings that match the
     ↪pattern
```

```
[3]: import re
     match = re.search('ineuron','Ineuron Full Stack Data Science Program', flags=re.
     ↪IGNORECASE)
     print('Output:',match.group())
```

```
Output: Ineuron
```

5. In the regex which created from the r'()-()', what does group zero cover? Group 2? Group 1?

```
[ ]: => In the Regex r'(\d\d\d)-(\d\d\d-\d\d\d\d)' the zero group covers the entire␣
     ↪pattern match where as the first group cover (\d\d\d) and the second group␣
     ↪cover (\d\d\d-\d\d\d\d)
```

```
[4]: # Example Program
     import re
     phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')
     mo = phoneNumRegex.search('My number is 415-555-4242.')
     print(mo.groups()) # Prints all groups in a tuple format
     print(mo.group()) # Always returns the fully matched string
     print(mo.group(1)) # Returns the first group
     print(mo.group(2)) # Returns the second group
```

```
('415', '555-4242')
415-555-4242
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?
     => The \. \( and \) escape characters in the raw string passed to re.compile()␣
     ↪will match actual parenthesis characters.
```

```
[5]: # Example Program
     import re
     phoneNumRegex = re.compile(r'(\(\d\d\d\)) (\d\d\d-\d\d\d\d)')
     mo = phoneNumRegex.search('My phone number is (415) 555-4242.')
     print(mo.group())
```

```
(415) 555-4242
```

```
[ ]: 7. The findall() method returns a string list or a list of string tuples. What␣
     ↪causes it to return one of the two options?
     => If the regex pattern has no groups, a list of strings matched is returned.␣
     ↪if the regex pattern has groups, a list of tuple of strings is returned.
```

```
[6]: # Example Program
     import re
     phoneNumRegex = re.compile(r'(\(\d\d\d\)) (\d\d\d-\d\d\d\d)')
     mo = phoneNumRegex.findall('My phone number is (415) 555-4242.')
     print(mo)

     # Example Program
     import re
     phoneNumRegex = re.compile(r'\d{3}-\d{3}-\d{4}')
     mo = phoneNumRegex.findall('My number is 415-555-4242.')
     print(mo) # Prints all groups in a tuple format
```

```
[('(415)', '555-4242')]
['415-555-4242']
```

[ ]: 8. In standard expressions, what does the | character mean?
=>In Standard Expressions | means OR operator.

[ ]: 9. In regular expressions, what does the ? character stand for?
=>In regular Expressions, ? characters represents zero or one match of the
↪preceeding group.

[ ]: 10.In regular expressions, what is the difference between the + and *
↪characters?
Ans: In Regular Expressions, * Represents Zero ore more occurances of the
↪preceeding group, whereas + represents one or more occurances of the
↪preceeding group.

[ ]: 11. What is the difference between {4} and {4,5} in regular expression?
Ans: {4} means that its preceeding group should repeat 4 times. where as {4,5}
↪means that its preceeding group should repeat
mininum 4 times and maximum 5 times inclusively

[ ]: 12. What do you mean by the \d, \w, and \s shorthand character classes signify
↪in regular expressions?
=> \d, \w and \s are special sequences in regular expresssions in python:

\w - Matches a word character equivalent to [a-zA-Z0-9_]
\d - Matches digit character equivalent to [0-9]
\s - Matches whitespace character (space, tab, newline, etc.)

[ ]: 13. What do means by \D, \W, and \S shorthand character classes signify in
↪regular expressions?
=> \D, \W and \S are special sequences in regular expresssions in python:

\W - Matches any non-alphanumeric character equivalent to [^a-zA-Z0-9_]
\D - Matches any non-digit character, this is equivalent to the set class [^0-9]
\S - Matches any non-whitespace character

[ ]: 14. What is the difference between .*? and .*?
=>.* is a Greedy mode, which returns the longest string that meets the
↪condition. Whereas .*? is a non greedy mode which
returns the shortest string that meets the condition.

[ ]: 15. What is the syntax for matching both numbers and lowercase letters with a
↪character class?
=> The Synatax is Either [a-z0-9] or [0-9a-z]

`[ ]:` 16. What **is** the procedure **for** making a normal expression **in** regax case␣
   ↪insensitive?
=> We can **pass** re.IGNORECASE **as** a flag to make a noraml expression case␣
   ↪insensitive

`[ ]:` 17. What does the . character normally match? What does it match **if** re.DOTALL␣
   ↪**is** passed **as** 2nd argument **in** re.compile()?
=> Dot . character matches everything **in** input **except** newline character .. By␣
   ↪passing re.DOTALL **as** a flag to re.compile(), you can make the dot character␣
   ↪match **all** characters, including the newline character.

`[ ]:` 18. If numReg = re.compile(r'\d+'), what will numRegex.sub('X', '11 drummers,␣
   ↪10 pipers, five rings, 4 hen') return?
=> The Ouput will be 'X drummers, X pipers, five rings, X hen'

```python
[7]: import re
numReg = re.compile(r'\d+')
numReg.sub('X', '11 drummers, 10 pipers, five rings, 4 hen')
```

`[7]:` 'X drummers, X pipers, five rings, X hen'

`[ ]:` 19. What does passing re.VERBOSE **as** the 2nd argument to re.compile() allow to␣
   ↪do?
=> re.VERBOSE will allow to add whitespace **and** comments to 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)

```python
[8]: import re
pattern = r'^\d{1,3}(,\d{3})*$'
pagex = re.compile(pattern)
for ele in ['42','1,234', '6,368,745','12,34,567','1234']:
    print('Output:',ele, '->', pagex.search(ele))
```

```
Output: 42 -> <re.Match object; span=(0, 2), match='42'>
Output: 1,234 -> <re.Match object; span=(0, 5), match='1,234'>
Output: 6,368,745 -> <re.Match object; span=(0, 9), match='6,368,745'>
Output: 12,34,567 -> None
Output: 1234 -> None
```

`[ ]:` 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)

```
[9]: #pattern = r'[A-Z]{1}[a-z]*\sWatanabe'

import re
pattern = r'[A-Z]{1}[a-z]*\sWatanabe'
namex = re.compile(pattern)
for name in ['Haruto Watanabe','Alice Watanabe','RoboCop Watanabe','haruto␣
 ↪Watanabe','Mr. Watanabe','Watanabe','Haruto watanabe']:
    print('Output: ',name,'->',namex.search(name))
```

```
Output:  Haruto Watanabe -> <re.Match object; span=(0, 15), match='Haruto
Watanabe'>
Output:  Alice Watanabe -> <re.Match object; span=(0, 14), match='Alice
Watanabe'>
Output:  RoboCop Watanabe -> <re.Match object; span=(4, 16), match='Cop
Watanabe'>
Output:  haruto Watanabe -> None
Output:  Mr. Watanabe -> None
Output:  Watanabe -> None
Output:  Haruto watanabe -> None
```

[ ]: 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.'

```
[10]: #pattern = r'(Alice|Bob|Carol)\s(eats|pets|throws)\s(apples|cats|baseballs)\.'

      import re
      pattern = r'(Alice|Bob|Carol)\s(eats|pets|throws)\s(apples|cats|baseballs)\.'
      casex = re.compile(pattern,re.IGNORECASE)
      for ele in ['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.']:
          print('Output: ',ele,'->',casex.search(ele))
```

```
Output:  Alice eats apples. -> <re.Match object; span=(0, 18), match='Alice eats
apples.'>
Output:  Bob pets cats. -> <re.Match object; span=(0, 14), match='Bob pets
cats.'>
Output:  Carol throws baseballs. -> <re.Match object; span=(0, 23), match='Carol
throws baseballs.'>
Output:  Alice throws Apples. -> <re.Match object; span=(0, 20), match='Alice
throws Apples.'>
Output:  BOB EATS CATS. -> <re.Match object; span=(0, 14), match='BOB EATS
CATS.'>
Output:  RoboCop eats apples. -> None
Output:  ALICE THROWS FOOTBALLS. -> None
Output:  Carol eats 7 cats. -> None
```

```
[ ]:
```