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

The re.compile() function returns Regex objects.

In [2]:
```
import re
pattern=re.compile('time')
result=pattern.findall('time and tide wait 45 for none,flow with time')
```

In [3]:
```
result
```

Out[3]:
```
['time', 'time']
```

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

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

In [22]:
```
re.compile('heyy \t')
```

Out[22]:
```
re.compile(r'heyy \t', re.UNICODE)
```

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

The search() method returns Match objects.

In [20]:
```
import re
txt='time and tide wait 45 for none,flow with time'
x=re.search('time',txt)
if x:
    print('matched')
else:
    print('sorry')
```

```
matched
```

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

The group() method returns strings of the matched text.

In [21]:
```
txt
```

Out[21]:
```
'time and tide wait 45 for none,flow with time'
```

In [27]:
```
x=re.search(r'\bf\w+',txt)
print(x.group())
```

```
for
```

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?

Group 0 is the entire match, group 1 covers the first set of parentheses, and group 2 covers the second set of parentheses.

In [ ]:
```
pattern=re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')
x=pattern.search('id is 543-655-4567')
```

In [35]:
```
x.group(1)
```

Out[35]:
```
'543'
```

In [40]:
```
x.group(2)
```

Out[40]:
```
'655-4567'
```

In [39]:
```
x.group(0)
```

Out[39]:
```
'543-655-4567'
```

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?

```
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?

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.

In [47]:
```
pattern=re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')
x=454-345-4455-234-456-5667
y=pattern.findall('454-345-4455',x)
y
```

Out[47]:
```
[('454', '345-4455')]
```

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

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

In [49]:
```
fruitRegex = re.compile (r'Banana|Apple Fruit')
mo1 = fruitRegex.search('Banana and Apple Fruit') #When both Banana and Apple Fruit occur in the searched s
#                                                                                              returned as t
mo1.group()
```

Out[49]:
```
'Banana'
```

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

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

In [50]:
```
txt
```

Out[50]:
```
'time and tide wait 45 for none,flow with time'
```

In [68]:
```
x=re.findall('wa.?t',txt)
print(x)
```

```
['wait']
```

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

+ for One or more occurrences

*** for Zero or more occurrences**

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

The {4} matches exactly four instances of the preceding group. The {4,5} matches between four and five instances.

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

\d Returns a match where the string contains digits (numbers from 0-9)

\w Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)

\s Returns a match where the string contains a white space character

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

\D Returns a match where the string DOES NOT contain digits

\W Returns a match where the string DOES NOT contain any word characters

\S Returns a match where the string DOES NOT contain a white space character

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

The .* performs a greedy match, and the .*? performs a nongreedy match.

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

In [71]:
```
import re
txt = "8 times before 11:45 AM"

#Check if the string has any characters from a to z lower case, and A to Z upper case:

x = re.findall("[a-zA-Z]", txt)

print(x)

if x:
  print("Yes, there is at least one match!")
else:
  print("No match")


```

```
['t', 'i', 'm', 'e', 's', 'b', 'e', 'f', 'o', 'r', 'e', 'A', 'M']
Yes, there is at least one match!
```

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

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

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

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

In [79]:
```
numRegex = re.compile(r'\d+')
x= numRegex.sub('X', '11 drummers, 10 pipers, five rings, 4 hen')
x
```

Out[79]:
```
'X drummers, X pipers, five rings, X hen'
```

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

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)

```
In [83]: reg1 = re.compile(r'^\d{1,3}(,\d{3})*$')
         mo1 = reg1.search('42')
         mo1.group()

Out[83]: '42'
```

```
In [84]: reg1 = re.compile(r'^\d{1,3}(,\d{3})*$')
         mo1 = reg1.search('1,234')
         mo1.group()

Out[84]: '1,234'
```

```
In [85]: reg1 = re.compile(r'^\d{1,3}(,\d{3})*$')
         mo1 = reg1.search('12,34,567')
         mo1.group()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [85], in <cell line: 3>()
      1 reg1 = re.compile(r'^\d{1,3}(,\d{3})*$')
      2 mo1 = reg1.search('12,34,567')
----> 3 mo1.group()

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

```
In [86]: reg1 = re.compile(r'^\d{1,3}(,\d{3})*$')
         mo1 = reg1.search('1234')
         mo1.group()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [86], in <cell line: 3>()
      1 reg1 = re.compile(r'^\d{1,3}(,\d{3})*$')
      2 mo1 = reg1.search('1234')
----> 3 mo1.group()

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

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)

```
In [91]: name = re.compile(r'[A-Z][a-z]*\sWatanabe')

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

Out[91]: 'Haruto Watanabe'
```

```
In [92]: name = re.compile(r'[A-Z][a-z]*\sWatanabe')
         mo1 = name.search('Alice Watanabe')
         mo1.group()

Out[92]: 'Alice Watanabe'
```

```
In [93]: name = re.compile(r'[A-Z][a-z]*\sWatanabe')
         mo1 = name.search('Robocop Watanabe')
         mo1.group()

Out[93]: 'Robocop Watanabe'
```

```
In [94]:
name = re.compile(r'[A-Z][a-z]*\sWatanabe')
mo1 = name.search('haruto Watanabe')
mo1.group()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [94], in <cell line: 3>()
      1 name = re.compile(r'[A-Z][a-z]*\sWatanabe')
      2 mo1 = name.search('haruto Watanabe')
----> 3 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.'

```
In [101]:
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[101]: 'Alice eats apples.'
```

```
In [102]:
name = re.compile(r'(Alice|Bob|Carol)\s(eats|pets|throws)\s(apples|cats|baseballs)\.', re.IGNORECASE)

mo1 = name.search('Bob pets cats.')
mo1.group()
```

```
Out[102]: 'Bob pets cats.'
```

```
In [103]:
name = re.compile(r'(Alice|Bob|Carol)\s(eats|pets|throws)\s(apples|cats|baseballs)\.', re.IGNORECASE)

mo1 = name.search('RoboCop eats apples.')
mo1.group()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [103], in <cell line: 4>()
      1 name = re.compile(r'(Alice|Bob|Carol)\s(eats|pets|throws)\s(apples|cats|baseballs)\.', re.IGNORECASE)
      3 mo1 = name.search('RoboCop eats apples.')
----> 4 mo1.group()

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

```
In [ ]:
```