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

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

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

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

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?

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

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

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

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

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

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

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

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

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

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

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

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

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

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)

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)

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.'

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

The feature responsible for generating Regex objects in Python is the `re.compile()` function. This function compiles a regular expression pattern into a Regex object, which can then be used to perform match operations.

\*\*Example:\*\*

```python

import re

pattern = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')

```

In this example, `pattern` is a Regex object generated by the `re.compile()` function.

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

Raw strings (indicated by the prefix `r`) are often used in Regex objects to prevent Python from interpreting backslashes (`\`) as escape characters. In regular expressions, backslashes are used for special character classes (e.g., `\d` for digits). Using raw strings ensures that the backslashes are passed to the Regex engine as intended.

\*\*Example:\*\*

```python

import re

pattern = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')

```

Here, `r'\d\d\d-\d\d\d-\d\d\d\d'` is a raw string, ensuring that the backslashes are treated as part of the regular expression pattern.

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

The `search()` method returns a `Match` object if it finds a match for the pattern anywhere in the string. If no match is found, it returns `None`.

\*\*Example:\*\*

```python

import re

pattern = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')

match = pattern.search('Call me at 123-456-7890')

if match:

print("Match found:", match.group())

else:

print("No match found.")

```

Output:

```

Match found: 123-456-7890

```

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

You can use the `group()` method of the `Match` object to get the actual string that matches the pattern.

\*\*Example:\*\*

```python

import re

pattern = re.compile(r'(\d\d\d)-(\d\d\d)-(\d\d\d\d)')

match = pattern.search('Call me at 123-456-7890')

if match:

print("Full match:", match.group(0))

print("Area code:", match.group(1))

print("First 3 digits:", match.group(2))

print("Last 4 digits:", match.group(3))

```

Output:

```

Full match: 123-456-7890

Area code: 123

First 3 digits: 456

Last 4 digits: 7890

```

### 5. In the regex 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\*\*: Covers the entire match (the full string matched by the regex).

- \*\*Group 1\*\*: Covers the first set of parentheses, `(\d\d\d)`, which corresponds to the area code.

- \*\*Group 2\*\*: Covers the second set of parentheses, `(\d\d\d-\d\d\d\d)`, which corresponds to the remaining part of the phone number.

\*\*Example:\*\*

```python

import re

pattern = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')

match = pattern.search('Call me at 123-456-7890')

if match:

print("Group 0 (Full match):", match.group(0))

print("Group 1 (Area code):", match.group(1))

print("Group 2 (Phone number):", match.group(2))

```

Output:

```

Group 0 (Full match): 123-456-7890

Group 1 (Area code): 123

Group 2 (Phone number): 456-7890

```

### 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?

To match real parentheses `()` and periods `.`, you need to escape them with a backslash `\` in your regular expression.

\*\*Example:\*\*

```python

import re

pattern = re.compile(r'\(\d{3}\)\.\d{3}\.\d{4}')

match = pattern.search('My number is (123).456.7890')

if match:

print("Match found:", match.group())

```

Output:

```

Match found: (123).456.7890

```

### 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 regular expression pattern contains no groups (i.e., no parentheses `()`), `findall()` returns a list of matching strings.

- If the pattern contains groups, `findall()` returns a list of tuples, where each tuple contains the strings matched by the groups in the regex.

\*\*Example 1: No Groups\*\*

```python

import re

pattern = re.compile(r'\d{3}-\d{3}-\d{4}')

matches = pattern.findall('Call me at 123-456-7890 or 987-654-3210')

print(matches)

```

Output:

```

['123-456-7890', '987-654-3210']

```

\*\*Example 2: With Groups\*\*

```python

import re

pattern = re.compile(r'(\d{3})-(\d{3})-(\d{4})')

matches = pattern.findall('Call me at 123-456-7890 or 987-654-3210')

print(matches)

```

Output:

```

[('123', '456', '7890'), ('987', '654', '3210')]

```

### 8. In standard expression syntax, what does the `|` character mean?

The `|` character is used as an OR operator in regular expressions. It allows you to match one pattern or another.

\*\*Example:\*\*

```python

import re

pattern = re.compile(r'cat|dog')

match = pattern.findall('I have a cat and a dog.')

print(match)

```

Output:

```

['cat', 'dog']

```

Here, the regex matches either "cat" or "dog".

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

In regular expressions, the `?` character has two main uses:

1. \*\*Optionality\*\*: When placed after a character or group, it means that the character or group is optional (it can occur 0 or 1 times).

2. \*\*Non-Greedy Matching\*\*: When used after a quantifier like `\*` or `+`, it makes the match non-greedy (matches the smallest possible string).

\*\*Example 1: Optionality\*\*

```python

import re

pattern = re.compile(r'colou?r')

match = pattern.findall('I like color and colour.')

print(match)

```

Output:

```

['color', 'colour']

```

Here, the `u` is optional, so it matches both "color" and "colour".

\*\*Example 2: Non-Greedy Matching\*\*

```python

import re

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

match = pattern.findall('<div>content</div><span>text</span>')

print(match)

```

Output:

```

['<div>', '<span>']

```

Here, `.\*?` matches the smallest possible string between `<` and `>`.

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

- \*\*`\*` (Asterisk)\*\*: Matches 0 or more occurrences of the preceding element.

- \*\*`+` (Plus)\*\*: Matches 1 or more occurrences of the preceding element.

\*\*Example 1: `\*`\*\*

```python

import re

pattern = re.compile(r'ab\*')

match = pattern.findall('a ab abb abbb')

print(match)

```

Output:

```

['a', 'ab', 'abb', 'abbb']

```

Here, `b\*` matches 0 or more `b` characters.

\*\*Example 2: `+`\*\*

```python

import re

pattern = re.compile(r'ab+')

match = pattern.findall('a ab abb abbb')

print(match)

```

Output:

```

['ab', 'abb', 'abbb']

```

Here, `b+` matches 1 or more `b` characters, so the first `a` is not matched.

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

- \*\*`{4}`\*\*: Matches exactly 4 occurrences of the preceding element.

- \*\*`{4,5}`\*\*: Matches between 4 and 5 occurrences of the preceding element.

\*\*Example 1: `{4}`\*\*

```python

import re

pattern = re.compile(r'\d{4}')

match = pattern.findall('123 1234 12345')

print(match)

```

Output:

```

['1234']

```

Here, it matches exactly 4 digits.

\*\*Example 2: `{4,5}`\*\*

```python

import re

pattern = re.compile(r'\d{4,5}')

match = pattern.findall('123 1234 12345')

print(match)

```

Output:

```

['1234', '12345']

```

Let's continue with the answers from question 12 to 22:

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

- \*\*`\d`\*\*: Matches any digit (equivalent to `[0-9]`).

- \*\*`\w`\*\*: Matches any alphanumeric character plus the underscore (equivalent to `[a-zA-Z0-9\_]`).

- \*\*`\s`\*\*: Matches any whitespace character (spaces, tabs, line breaks).

\*\*Example:\*\*

```python

import re

text = "The price is 100 dollars"

digit\_match = re.findall(r'\d+', text)

word\_match = re.findall(r'\w+', text)

space\_match = re.findall(r'\s+', text)

print(digit\_match) # Output: ['100']

print(word\_match) # Output: ['The', 'price', 'is', '100', 'dollars']

print(space\_match) # Output: [' ', ' ', ' ']

```

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

- \*\*`\D`\*\*: Matches any character that is not a digit (equivalent to `[^0-9]`).

- \*\*`\W`\*\*: Matches any character that is not a word character (equivalent to `[^a-zA-Z0-9\_]`).

- \*\*`\S`\*\*: Matches any character that is not a whitespace character.

\*\*Example:\*\*

```python

import re

text = "The price is 100 dollars"

nondigit\_match = re.findall(r'\D+', text)

nonword\_match = re.findall(r'\W+', text)

nonspace\_match = re.findall(r'\S+', text)

print(nondigit\_match) # Output: ['The price is ', ' dollars']

print(nonword\_match) # Output: [' ', ' ', ' ', ' ']

print(nonspace\_match) # Output: ['The', 'price', 'is', '100', 'dollars']

```

### 14. What is the difference between `.\*?` and `.\*` in regular expressions?

- \*\*`.\*`\*\*: Greedily matches as many characters as possible.

- \*\*`.\*?`\*\*: Non-greedily matches the smallest number of characters possible.

\*\*Example:\*\*

```python

import re

text = "<div>content</div><span>text</span>"

greedy\_match = re.findall(r'<.\*>', text)

nongreedy\_match = re.findall(r'<.\*?>', text)

print(greedy\_match) # Output: ['<div>content</div><span>text</span>']

print(nongreedy\_match) # Output: ['<div>', '</div>', '<span>', '</span>']

```

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

You can use a character class to match both numbers and lowercase letters by combining their ranges within square brackets `[]`.

\*\*Example:\*\*

```python

import re

text = "abc123XYZ"

match = re.findall(r'[a-z0-9]+', text)

print(match) # Output: ['abc123']

```

### 16. What is the procedure for making a normal expression in regex case-insensitive?

To make a regex case-insensitive, you can pass the `re.IGNORECASE` (or `re.I`) flag as a second argument to `re.compile()`.

\*\*Example:\*\*

```python

import re

text = "Hello HELLO hello"

pattern = re.compile(r'hello', re.IGNORECASE)

match = pattern.findall(text)

print(match) # Output: ['Hello', 'HELLO', 'hello']

```

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

- \*\*`.`\*\*: Normally matches any character except a newline.

- \*\*With `re.DOTALL`\*\*: The `.` character matches any character, including a newline.

\*\*Example:\*\*

```python

import re

text = "Hello\nworld"

pattern\_normal = re.compile(r'.\*')

pattern\_dotall = re.compile(r'.\*', re.DOTALL)

match\_normal = pattern\_normal.match(text)

match\_dotall = pattern\_dotall.match(text)

print(match\_normal.group()) # Output: 'Hello'

print(match\_dotall.group()) # Output: 'Hello\nworld'

```

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

The `sub()` method replaces all occurrences of the regex pattern in the string with the specified replacement.

\*\*Example:\*\*

```python

import re

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

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

print(result) # Output: '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` allows you to write regular expressions that are more readable by allowing you to include whitespace and comments within the regex.

\*\*Example:\*\*

```python

import re

pattern = re.compile(r'''

\d{3} # Area code

- # Separator

\d{3} # First three digits

- # Separator

\d{4} # Last four digits

''', re.VERBOSE)

match = pattern.search('My number is 123-456-7890')

print(match.group()) # Output: '123-456-7890'

```

### 20. How would you write a regex that matches a number with a comma for every three digits?

To match numbers with commas for every three digits, you can use the following regex:

\*\*Example:\*\*

```python

import re

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

matches = [

pattern.match('42'),

pattern.match('1,234'),

pattern.match('6,368,745'),

pattern.match('12,34,567'), # Should not match

pattern.match('1234') # Should not match

]

print([m.group() if m else None for m in matches])

# Output: ['42', '1,234', '6,368,745', None, None]

```

### 21. How would you write a regex that matches the full name of someone whose last name is Watanabe?

To match full names with the last name "Watanabe," assuming the first name starts with a capital letter and is a single word:

\*\*Example:\*\*

```python

import re

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

matches = [

pattern.match('Haruto Watanabe'),

pattern.match('Alice Watanabe'),

pattern.match('RoboCop Watanabe'),

pattern.match('haruto Watanabe'), # Should not match

pattern.match('Mr. Watanabe'), # Should not match

pattern.match('Watanabe'), # Should not match

pattern.match('Haruto watanabe') # Should not match

]

print([m.group() if m else None for m in matches])

# Output: ['Haruto Watanabe', 'Alice Watanabe', 'RoboCop Watanabe', None, None, None, 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?

To create a regex that matches these conditions:

\*\*Example:\*\*

```python

import re

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

matches = [

pattern.match('Alice eats apples.'),

pattern.match('Bob pets cats.'),

pattern.match('Carol throws baseballs.'),

pattern.match('Alice throws Apples.'),

pattern.match('BOB EATS CATS.'),

pattern.match('RoboCop eats apples.'), # Should not match

pattern.match('ALICE THROWS FOOTBALLS.'), # Should not match

pattern.match('Carol eats 7 cats.') # Should not match

]

print([m.group() if m else None for m in matches])

# Output: ['Alice eats apples.', 'Bob pets cats.', 'Carol throws baseballs.', 'Alice throws Apples.', 'BOB EATS CATS.', None, None, None]

```