Learning Quiz 33: String Pattern Matching

Due Nov 27 at 1pm **Points** 5 **Questions** 5 **Available** until Dec 4 at 11:59pm **Time Limit** None
Allowed Attempts Unlimited

Instructions

Prior to completing this quiz, be sure to read:

- Section 11.3: String Pattern Matching (p. 387-392)

Please also go over Practice Problems 11.4-11.6 in the textbook (solutions at the end of the chapter) before attempting this quiz.

This quiz was created for learning purposes. You may attempt this quiz as many times as you would like. The highest score *prior to the deadline* will count towards the final course grade. No late submissions will be accepted.

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	10 minutes	5 out of 5

Score for this attempt: **5** out of 5
Submitted Nov 29 at 10:53am
This attempt took 10 minutes.

Question 1

1 / 1 pts

As we work with URL and HTML, we may find ourselves in situations where we are looking for words that have some variation in spelling. For example, what if we were looking for the word 'color' or 'colour'? Rather than look for both separately, we can search for both by looking for 'colou?r' using our findall function.

We'll need the findall function from the regular expressions (re) module. Fill in the blanks to import the findall function from the re module:

from re import findall

Answer 1:

Correct! from

Answer 2:

Correct! import

Question 2

1 / 1 pts

The findall() method requires two arguments:

1. pattern - the string pattern that we are looking for
2. string - the string that we are searching in

In the following example, we are looking for the pattern 'dog' in the string 'My dog ate my homework'.

```
print(findall('dog', 'My dog ate my homework'))
```

What makes the findall() method different from the string find() method is that our pattern does not have to be exactly as we search. For example:

```
print(findall('cl.ck', 'click clack cluck claack claaack croak cr0000ak cricket quack cl ck clck'))
```

will print the following:

```
['click', 'clack', 'cluck', 'cl ck']
```

Notice that the period (.) serves as a placeholder for any possible ONE character. Since 'click' matches the 'cl.ck' structure, then it is added to our findall list. The same applies to 'clack', 'cluck', and 'cl ck'. In 'cl ck', space counts as a character.

Take a moment to review the different possible string pattern searches on pages 387-389. Go through each of these lines to see what you get in return:

```
findall('cl..ck', 'click clack cluck claack claaack croak cr0000ak cricket quack cl ck clck')
findall('cla*ck', 'click clack cluck claack claaack croak cr0000ak cricket quack cl ck clck')
findall('cla+ck', 'click clack cluck claack claaack croak cr0000ak cricket quack cl ck clck')
findall('cla?ck', 'click clack cluck claack claaack croak cr0000ak cricket quack cl ck clck')
```

Match the operator with the description:

Correct!	**Takes the place of any one character, and requires exactly one character**	.
Correct!	**matches 0 or more repetitions of the previous character**	*
Correct!	**matches 1 or more repetitions of the previous character**	+
Correct!	**matches 0 or 1 repetition of the previous character**	?

Question 3

1 / 1 pts

The square-brackets will indicate ranges of characters that you are looking for. You can indicate ranges of characters by typing the exact characters themselves, using a dash (-) to indicate all characters in between inclusive (in ASCII order), or using a combination of the exact letter and dashes.

```
## looking for exactly i or a
print(findall('cl[ia]ck', 'click clack cluck claack claaack croak cr0000ak cricket quack cl ck clck'))
## looking for letters g through z
print(findall('cl[g-z]ck', 'click clack cluck claack claaack croak cr0000ak cricket quack cl ck clck'))
## looking for letters a through e OR s through z
print(findall('cl[a-es-z]ck', 'click clack cluck claack claaack croak cr0000ak cricket quack cl ck clck'))
```

Each set of square brackets represents **EXACTLY ONE CHARACTER**. The following will only give ['click', 'clack'] and not 'cliack' or 'claick'.

```
print(findall('cl[ia]ck', 'click clack cliack claick claack cliick cl ck clck'))
```

Likewise, the following will only give ['cliack', 'claick', 'claack', 'cliick'] and not 'click' or 'clack'.

```
print(findall('cl[ia][ia]ck', 'click clack cliack claick claack cliick cl ck clck'))
```

Compare the following lines:

```
print(findall('cl[ia][ia]ck', 'click clack cliack claick claack cliick cl ck clck'))
print(findall('cl[ia]ck', 'click clack cliack claick claack cliick cl ck clck'))
print(findall('cl[ia]?[ia]?ck', 'click clack cliack claick claack cliick cl ck clck'))
```

Which of the following are strings included in our list templist?

```
templist = findall('b[ia][ia]?t', 'bt beet beeet bet bets b3t bat baat bite bait beit biat boat bot boot b
iit byte')
```

- ☐ beet
- Correct! ☑ baat
- ☐ bt
- Correct! ☑ biat
- ☐ biet
- ☐ byt
- ☐ b3t
- Correct! ☑ bat
- ☐ beit
- Correct! ☑ bit
- Correct! ☑ bait

- [ ] bite

Correct!
- [✓] biit

Question 4

1 / 1 pts

The caret (^) operator inside square brackets will indicate all combinations EXCEPT the given letters. For example, the following line will exclude all printouts that match the given format except 'click' and 'clack'. Note, square brackets serve as a placeholder for only one character. (Don't forget to print to see the output)

```
findall('cl[^ia]ck', 'click clack cluck claack claaack croak cr0000ak cricket quack cl ck clck')
```

We can give ranges as usual.

```
## looking for all strings with one letter in between cl and ck, EXCEPT i or a
findall('cl[^ia]ck', 'click clack cluck claack claaack croak cr0000ak cricket quack cl ck clck')
## looking for one letter in between cl and ck, EXCEPT letters s through z
findall('cl[^s-z]ck', 'click clack cluck claack claaack croak cr0000ak cricket quack cl ck clck')
## looking for all strings with one letter in between cl and ck, EXCEPT letters a through e and s through
 z
findall('cl[a-es-z]ck', 'click clack cluck claack claaack croak cr0000ak cricket quack cl ck clck')
```

Which of the following are strings included in our templist?

```
templist = findall('b[^ia][^ia]t', 'bt beet beeet bet bets b3t bat baat bite bait biat beit boat bot boot
biit byte')
```

- [ ] baat
- [ ] byt
- [ ] bot

Correct!
- [✓] beet

Correct!
- [✓] boot
- [ ] boat
- [ ] bet
- [ ] bit
- [ ] biit
- [ ] bt
- [ ] bat

Question 5

1 / 1 pts

The | operator means "or". For example, in the following line, we are simply looking for 'Hello' or 'Goodbye' in our variable teststring. (Don't forget to print to see the output.)

```
findall('Hello|Goodbye', teststring)
```

Note, we can combine the string-pattern matching concepts from above with the | operator.

In the following, we are looking for 'cl[iu]ck' OR 'cla*ck'.

```
print(findall('cl[iu]ck|cla*ck', 'click clack cluck claack claaack croak cr0000ak cricket quack cl ck clc
k'))
```

Which of the following are strings included in our templist?

```
templist = findall('be+t|b[h-p]t', 'bt beet beeet bet bets b3t bat baat bite bait biat beit boat bot boot
biit byte')
```

- [ ] biit
- [ ] boot
- Correct! [✓ beeet]
- [ ] bait
- Correct! [✓ bit]
- Correct! [✓ bot]
- [ ] boat
- [ ] byt
- Correct! [✓ bet]
- Correct! [✓ beet]

Quiz Score: **5** out of 5