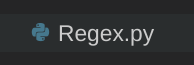
Regular expression and character classes

In the IDE, in the left-hand EXPLORER pane, double-click Regex.py:



In the right-hand corner you will see a green triangle icon:



If you don't see the green triangle icon, double click Regex.py in the EXPLORER pane again.

In the text-editor pane you will see the following python code:

[**Copy code**](https://app.qa.com/lab/constructing-regexes-character-classes/constructing-regex-character-classes/?context_id=637&context_resource=lp)

1

2

3

4

5

6

7

8

**import** re

pattern = r'ab'

text = 'abc acb'

matches = re.findall(pattern, text)

print(f"Pattern: {pattern}\nText: {text}\nMatches: {matches}\n")

You will use this code as a harness to test various regular expression patterns.

The first statement is importing Python's regular expression library, to learn more about this library visit the [official documentation](https://docs.python.org/3/library/re.html).

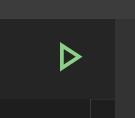
The second and third statements define variables, the first is the pattern you want to match, the second is the text you want to search.

The fourth statement uses the variables and the findall method of the re library, to search the text for strings matching the pattern.

Finally, the fifth statement prints the results using a feature of Python 3 called "f strings" which interpolates variables into a string.

Notice that the value of the pattern variable has an r in front of the opening single quote. This is Python syntax for defining a raw string literal, and it means that backslashes in the string will be treated as a literal character. Backslashes are normally used for escape sequences such as newlines (\n). Regular expressions also use characters with backslashes to perform certain operations. It's good practice in Python to define a regular expression pattern as a raw string literal so that it isn't misinterpreted.

3. In the top-right corner, click the Run icon (green triangle):



The python file will be executed in a terminal and you will see the results:



Note that the findall method returns an array, meaning the pattern can match more than one string sequence in a piece of text.

Most regular expressions contain combinations of special characters and sequences that allows for complex and efficient pattern matching. This first example doesn't, the pattern consists of the text ab and will match if the string being searched contains the text.

Optional: What do you expect will happen if you change the pattern from ab to ba? Try it and re-run the script to see.

4. To match all characters, replace the pattern and text lines with the following:

[**Copy code**](https://app.qa.com/lab/constructing-regexes-character-classes/constructing-regex-character-classes/?context_id=637&context_resource=lp)

1

2

pattern = r'.'

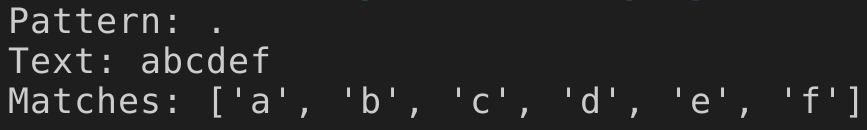
text = 'abcdef'

The pattern contains a period (.), this is a character class and a period by itself will match any single character.

Character Classes are sometimes called Special Sequences.

5. To see the result of this regular expression, click the Run icon.

You will see the output:



This regular expression has matched each character separately, resulting in six matches for a text containing six characters.

Despite how simple the period character class appears, it's actually extremely useful and can be found in just about every complex regular expression. Its power comes from being combined with a quantifier, and other regular expression elements. These will be detailed in another lab.

You can think of character classes as the foundational building blocks of regular expressions.

6. To implement a regex that will match a single digit, replace the pattern and text lines with the following:

[**Copy code**](https://app.qa.com/lab/constructing-regexes-character-classes/constructing-regex-character-classes/?context_id=637&context_resource=lp)

1

2

pattern = r'\d'

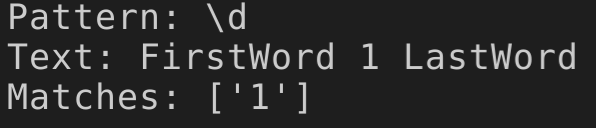
text = 'FirstWord 1 LastWord'

Before running the script again, let's explain what this change means.

The \d character class matches a single digit, you've changed the text to be searched to a string containing a single-digit between two words.

7. To test your updated script, click the Run icon.

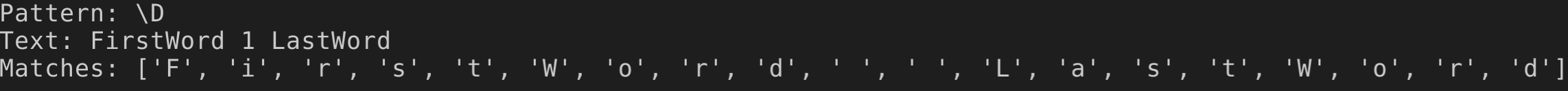
You will see the following output:



As you might have expected, the digit 1 was matched.

8. Change the value of the pattern string to \D and run the script again.

You will see the result:



\D matches any non-digit characters, all characters except the digit 1 were matched. This is the opposite of \d.

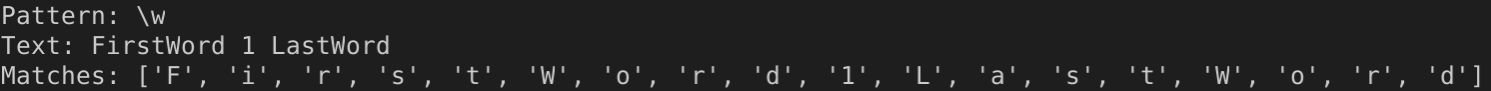
There are several other character classes where the lower case and upper case mirror each other.

Note that the array of character matches contains two spaces in between the words. Whitespace, including spaces, tabs, and newlines, can be matched using regular expression patterns.

Next, you will see how to match the characters in the words and not match the spaces.

9. Change the value of the pattern string to \w and re-run the script.

You will see the following output in the terminal:



\w matches word characters, any character that can be used as a part of a word.

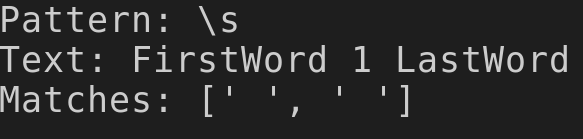
What defines a word character? This is a good time to mention character encodings such as ASCII and Unicode. A character encoding can be thought of as the set of all possible characters allowed in a piece of text, and character encodings have a long history (depending on how you define it, they pre-date computers, and even the telegraph, going back to maritime flags).

A detailed explanation of character encodings is beyond the scope of this lab. However, you should know that whether a character is classed as a digit, a word character, whitespace, or something else, is defined by the character encoding a piece of text is using. So for example, if you are using Unicode and a character in a non-English language is treated as whitespace, that's determined by the Unicode character encoding.

In this lab, you are using the UTF-8 character encoding, and to keep things simple, this lab will stick to using the characters commonly associated with the ASCII character encoding (essentially US standard English).

10. To match just the whitespace characters, the spaces on either side of the digit, change the value of the pattern string to \s, and click the Run icon.

You will see the output:



This pattern matches only whitespace, in this case, the spaces between the words and the digit.

Optional: Do you think this pattern will match a tab character? Try adding a tab character to the text and running the script to see if it does.

Optional: What do you think the \S pattern will match? Try it and see.

So far you've been using regular expression patterns to match single characters. To match sequences of strings you will use another feature of regular expressions called Quantifiers (also sometimes known as Special Characters).

There are many quantifiers and the topic will be covered more in another lab. For now you'll be introduced to the plus symbol quantifier (+). This is defined as:

Matches the preceding regular expression (left-hand) one or more times.

11. Replace the pattern and text lines with the following, and then run the script.

[**Copy code**](https://app.qa.com/lab/constructing-regexes-character-classes/constructing-regex-character-classes/?context_id=637&context_resource=lp)

1

2

pattern = r'\w+'

text = 'FirstWord 1 LastWord'

You will see the following matches:



The combination of the word character class (\w) with the plus symbol quantifier (+) will match one or more word characters. To describe that in a different way, you have matched the words in the text, ignoring the spaces between the words.

You can think of the quantifier used like this as a modifier, you've used the character class to define the type of character you want to match, and the plus symbol to say that you want to match one or more of that character.

12. Replace the pattern and text variables with the following and run the script:

[**Copy code**](https://app.qa.com/lab/constructing-regexes-character-classes/constructing-regex-character-classes/?context_id=637&context_resource=lp)

1

2

pattern = r'[4-6]'

text = '123456789'

You will see:



This pattern is a type of character class that you can use to match a range of characters. You specify the range you want to match inside square brackets and separate the start and end of the range with a dash.

In this example, you are matching the digits from 4 to 6 inclusive.

Optional: What do you predict will be matched if you remove the dash character? Try it and see.

13. To produce one match instead of three, replace the value of the pattern string with [4-6]+ and run the script.

This time, you will see:



Remember that the plus symbol matches one or more characters. In this case, it's matched the subset of digits in the value of the text variable.

Specifying a range with a dash symbol doesn't just work with digits, it also works with letters.

14. To match a non-digit subset, replace the pattern and text lines with the following and run the script:

[**Copy code**](https://app.qa.com/lab/constructing-regexes-character-classes/constructing-regex-character-classes/?context_id=637&context_resource=lp)

1

2

pattern = r'[d-f]+'

text = 'abcdefghi'

You will see:



So far you've matched ranges of digits and letters, you can also combine digit and letter ranges.

15. Replace the pattern and text lines with the following and run the script:

[**Copy code**](https://app.qa.com/lab/constructing-regexes-character-classes/constructing-regex-character-classes/?context_id=637&context_resource=lp)

1

2

pattern = r'[a-z0-9]+'

text = 'word!1234?wordandletters'

You will see three matches.

This pattern has matched groups of characters separated by an exclamation mark and a question mark.

16. To test whether the pattern will match upper case letters, replace the value of the text variable with the following and run the script again:

[**Copy code**](https://app.qa.com/lab/constructing-regexes-character-classes/constructing-regex-character-classes/?context_id=637&context_resource=lp)

1

text = 'WORD!1234?wordandletters'

This time you will see two matches.

The character range in the pattern is only matching lower case characters from a to z.

17. To match upper case characters as well, change the value of the pattern string to [a-zA-Z0-9]+ and run the script again.

This time the uppercase part of the text is matched.

This is a commonly used regular expression pattern, the characters it matches are called alphanumeric.

You might be wondering when you would want to match a subset of letters of digits using ranges. A good example of this is matching a hexadecimal number. Hexadecimal numbers are base sixteen, and use the letters a to f in addition to the digits 0 to 9 when represented textually. Examples of hexadecimal numbers are 00, 1A, FF, and a regular expression pattern that will match them is [0-9A-F]+.

A character range can also be negated.

18. Replace the pattern and text lines with the following and run the script:

[**Copy code**](https://app.qa.com/lab/constructing-regexes-character-classes/constructing-regex-character-classes/?context_id=637&context_resource=lp)

1

2

pattern = r'[^aeuio]+'

text = 'The quick brown fox jumps over the lazy dog.'

You will see the following:



This pattern has matched groups that include consonants and spaces in the text, but not the vowels. The caret symbol (^) is used to negate the proceeding right-hand characters.

19. To match only the consonants without the spaces, replace the pattern with [^aeuio^\s]+ and run the script.

This pattern combines the negation of the vowels with a negation of the \s whitespace character class.

### Summary

In this lab, you learned how to use a regular expression pattern in Python to search text for matches. You learned about the different character classes that allow you to match digits, word characters, and whitespace. And finally, you learned about character class ranges, and how to negate them.