Regular Expressions

Python has a built-in package called **re**, which can be used to work with Regular Expressions. There are some methods in this module:

re.search(regex, string)

This method checks if a searched content is in the relevant text. It looks for the first location where the pattern matches. If a match is found, then re.search() returns a match object. Otherwise, it returns None.

In [2]:
```
import re

# search for word Adam in the text
text = "Hi I am Adam, what is your name?"

re.search("Adam", text)
```

Out[2]:
```
<re.Match object; span=(8, 12), match='Adam'>
```

We searched the word Adams in the text, if you pay attention to the span and match areas. Where match is the value you are looking for, span is where it is. It's between the 8th and 12th letters where you're looking for.

Start ()

This method returns where the searched word appears in the source. We already know that the above example (8,13) is between. We can only use it if we want to draw the value 8 from here.

In [3]:
```
check = re.search("Adam", text)
# Where does the regex start in text?
check.start()
```

Out[3]:
```
8
```

end ()

It does the inverse of the start method, ie it returns the last value in which range the searched word passes. We know that the word Adams is mentioned in the source (8,12). The start method would return 8, and the end method would return 12.

In [4]:
```
check = re.search("Adam","Hi I am Adam, what is your name?")
# What is the last index of the searched regex in the text?
check.end()
```

Out[4]: 12

findall ()

With this method, we can examine how many times the text we want to pass in a source. It returns a list of all the appearences of the search text.

search() module will only return the first occurrence that matches the specified pattern. findall() will iterate over all the lines of the file and will return all non-overlapping matches of pattern in a single step.

In [5]:
```
text = "I am glad that time you came, time flies by with you"
re.findall("time", text)
```

Out[5]: ['time', 'time']

MetaCharacters

Up to this point, we searched for a specific text within a source, but what makes this module strong are the metacharacters that we will explain later. With these metacharacters, you can search for text that matches a layout you specify.

Most letters and characters will simply match themselves. For example, the regular expression test will match the string test exactly. (You can enable a case-insensitive mode that would let this RE match Test or TEST as well; more about this later.)

There are exceptions to this rule; some characters are special metacharacters, and don't match themselves. Instead, they signal that some out-of-the-ordinary thing should be matched, or they affect other portions of the RE by repeating them or changing their meaning. Much of this document is devoted to discussing various metacharacters and what they do.

Here's a complete list of the metacharacters; their meanings will be discussed in the rest of this HOWTO.

To specify regular expressions, metacharacters are used. These are $ [], ., ^, $, *, +, ?, {}, (), \, |$

[ ] - Square brackets

Square brackets specifies a set of characters (a character class) you wish to match. All characters written between these square brackets are taken into account. Gr[ae]y can match both gray and grey.

Metacharacters are not active inside classes. For example, '$' is usually a metacharacter, but inside a character class it's stripped of its special nature.

In [8]:
```
# write a regex to find both gray and grey
re.findall (r"gr[ae]y" ,"is it gray or grey?")
```

Out[8]:
```
['gray', 'grey']
```

In [10]:
```
# Expression: [abc]
# String: Hey, ac

re.findall(r"[abc]", "Hey, ac")
```

Out[10]:
```
['a', 'c']
```

In [ ]:
```

```

- [a-e] is the same as [abcde].
- [1-4] is the same as [1234].
- [0-39] is the same as [01239].
- [0-3][0-9]+
- [^abc] means any character except a or b or c.

. - Period

A period matches any single character (except newline '\n'). Replaces any single character except the newline character.

In [11]:
```
#Expression: ..
#String: a, ac, acd,acde
```

^ - Caret

The caret symbol ^ is used to check if a string starts with a certain character.

[^5] will match any character except '5'. If the caret appears elsewhere in a character class, it does not have special meaning. For example: [5^] will match either a '5' or a '^'.

\A functions similarly.

^ is called an anchor when it is used to check the beginning of a string.

In [55]:
```
# Use caret as an anchor for the beginning of a sentence
# Expression: ^a,^ab
# String: bac, abc
```

In [56]:
```
# Use caret to match all the letters except a
```

$ - Dollar

The dollar symbol $\$$ is used to check if a string ends with a certain character. So, if a dollar sign ($\$$) is at the end of the entire regular expression, it matches the end of a line.

$ symbol is called an anchor when it is used to check the end of a string.

In [63]:
```
# check if the beginning of the text is (^the)
# re.findall(?, "the cat runs fast")
```

In [64]:
```
# check if the text starts with a and ends with s,
# and there are 5 letters in total
# re.findall("?", "^alias")
```

In [ ]:
```

```

In [69]:
```
# In this text, find all the words that end with .com

import re
text = """Silicon valley discusses the competition between google.com and app
        and yahoo.com.tr"""
```

* - Star

The star symbol * matches $\textbf{zero}$ or $\textbf{more occurrences}$ of the pattern left to it.

In [37]:
```
# find all the occurrences of gmail even if g is repeated multiple times
# by mistake
# re.findall("?", "@mail @gmail @gggmail @gggggmail")
```

Out[37]:
```
['@mail', '@gmail', '@gggmail', '@gggggmail']
```

We put * after the letter g, which means that g can pass 0 times or 100 times. This distinction includes only the previous letter. Therefore, @mail, @gmail and @ggggggmail are returned.

In [ ]:
```
Expressions: ma*n
String: maaan
        main
        woman
```

+ - Plus

The plus symbol + matches $\textbf{one}$ or $\textbf{more occurrences}$ of the pattern left to it. Searches for one or more of the preceding character. If we make an example of * on the top with + this time;

In [71]:
```
# re.findall("?", "@mail @gmail @gggmail @gggggmail")
```

? - Question Mark

The question mark symbol ? matches $\textbf{zero}$ or $\textbf{one occurrence}$ of the pattern left to it.

In [72]:
```
# Write the same regex that is going to match mail and gmail.
# re.findall("?", "@mail @gmail @gggmail @gggggmail")
```

Expression: ma?n String: maaan woman man

{} - Braces

{n,m}: This means at least n, and at most m repetitions of the pattern left to it. It means a certain number of repetitions.

Now let's make a more complicated example with what we have learned above. Our source is $\textbf{"meat dealing with heat is hard. I can't cook meat. Meat is expensive"}$. We wanted to catch everything that started with small letters and ended with "at."

- Step 1: [a-z], we need to distinguish between lower case.
- Step 2:[a-z]*,0 or n (infinite) times lower case.
- Step 3: [a-z]*a, start with a lower case and continue.
- Step 4:[a-z]a{1}, need to have 1 a letter.
- Step 5:[a-z]a{1}t, ends with t

In [73]:
```
# write a regexp to catch everything that started with small letters
# and end with "at"
text = "meat dealing with heat is hard. I can't cook meat. Meat is expensive"
# re.findall("?", text )
```

In [ ]:
```
Expression: a{2,3}
String: aabc daaaat
        abc dat
```

In [ ]:

- find any number sequence of 2 to 4 digits.
- Expression: [0-9]{2,4}
- String: ab123csde 1 and 2

In [ ]:

| - Alternation

Vertical bar | is used for alternation (or operator).

In [75]:
```
# write a regex to catch either black or white in the text
text = "BJK is known for its black and white"
```

In [ ]:
```
Expression: a|b
String: cde
        acdbea
```

In [ ]:

() - Group

Parentheses () is used to group sub-patterns. For example, (a|b|c)xz match any string that matches either a or b or c followed by xz.

Parentheses are also used to group the patterns we write. You can think of it like the parentheses we use to set the priority for division and multiplication in mathematics.

In [76]:
```
# find all the patterns that start with either a or b or c,
# followed by xz
text = "axz cabxz ab xz"
# re.findall("?", text )
```

\ - Backslash

Backlash \ is used to escape various characters including all metacharacters. For example,$\$a$ match if a string contains $\$$ followed by a. Here, $\$$ is not interpreted by a RegEx engine in a special way.

If you are unsure if a character has special meaning or not, you can put \ in front of it. This makes sure the character is not treated in a special way.

Perhaps, the most diverse metacharacter!!

If the character following the backslash is a recognized escape character, then the special meaning of the term is taken (Scenario 1)

Else if the character following the \ is not a recognized escape character, then the \ is treated like any other character and passed through (Scenario 2).

\ can be used in front of all the metacharacters to remove their special meaning (Scenario 3).

In [ ]:
```
# regex for the $ pattern
# try without the backslash
```

In [ ]:
```
# try with the backslash
```

In [77]:
```
# \w matches a word char
text = "guru99,education is fun"
```

split()

In [ ]:
```
# \s is for splitting the words
```

sub()

Another task that the re package lets you do using regular expressions is to do substitutions within a string. The sub() methods takes a regular expression and phrase just like the query methods we've seen so far, but we also hand in the string to replace each match with. You can do straightforward substitutions like this

In [ ]:
```

```

Questions

1- Write a regular expression that matches at least two digits but not more than 4 digits.

In [ ]:

2- Write a regular expression that matches all the numbers between 0-59 inclusively.

In [ ]:

3- Write a function to clean twitter text document.

- Remove all the https hyperlinks in your twitter data with space
- Remove RTs
- Remove hashtags
- Remove numbers
- Remove punctuations

In [ ]: