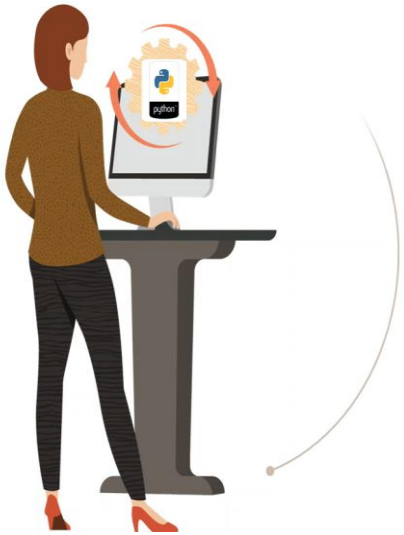
REGEX in PYTHON

By Kiran(KK)



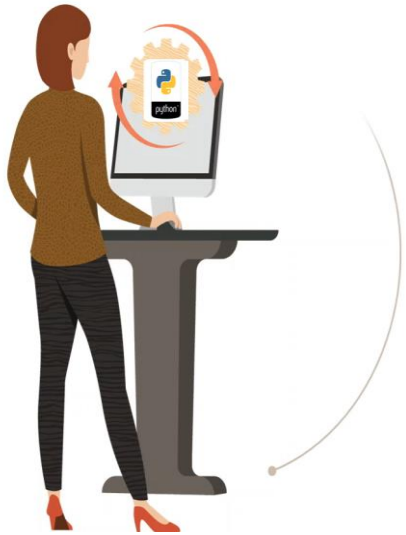
What is a regular expression?

> **REGular EXpression or regex:**

> *String containing a combination of normal characters and special metacharacters that describes patterns to find text or positions within a text*

```
r'st\d\s\w{3,10}'
```

- Pattern matching usage:
  - Find and replace text
  - Validate strings
- Very powerful and fast



What is a regular expression?

> **REGular EXpression or regex:**

> *String containing a combination of normal characters and special metacharacters that describes patterns* ***to find text or positions within a text***

- Pattern matching usage:
  - Find and replace text
  - Validate strings
- Very powerful and fast

Why Regular Expression?

- This means that more people / organizations are using tools like Python / JavaScript for solving their data needs.
- This is where Regular Expressions become super useful.
- Regular expressions are normally the default way of data cleaning and wrangling in most of these tools.
- Be it extraction of specific parts of text from web pages, making sense of twitter data or preparing your data for text mining – Regular expressions are your best bet for all these tasks.
- Given their applicability, it makes sense to know them and use them appropriately.

Regular Expression how is it used?

Simply put, regular expression is a sequence of character(s) mainly used to find and replace patterns in a string or file. As I mentioned before, they are supported by most of the programming languages like python, perl, R, Java and many others. So, learning them helps in multiple ways (more on this later).

Regular expressions use two types of characters:

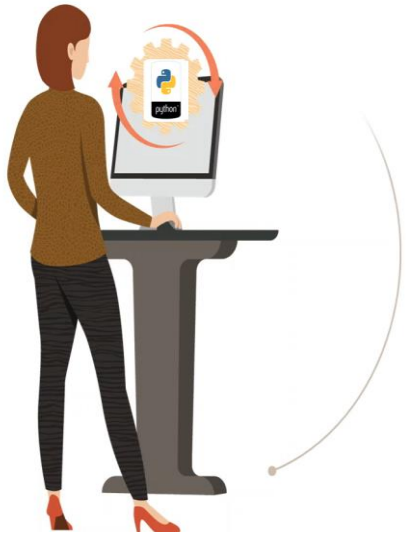
a) Meta characters: As the name suggests, these characters have a special meaning, similar to * in wild card.

b) Literals (like a,b,1,2…)

In Python, we have module “re” that helps with regular expressions. So you need to import library re before you can use regular expressions in Python.

The most common uses of regular expressions are:

- Search a string (search and match)
- Finding a string (findall)
- Break string into a sub strings (split)
- Replace part of a string (sub)



The re module

```
import re
```

- Find all matches of a pattern:

re.findall(r”regex”, string)

```
re.findall(r"#movies", "Love #movies! I had fun yesterday going to the #movies")
```

The re module

```
import re
```

- Split string at each match:

re.split(r”regex”, string)

```
re.split(r"!", "Nice Place to eat! I'll come back! Excellent meat!")
```

```
['Nice Place to eat', " I'll come back", ' Excellent meat', '']
```

The re module

```
import re
```

- Replace one or many matches with a string:

re.sub(r”regex”, new, string)

```
re.sub(r"yellow", "nice", "I have a yellow car and a yellow house in a yellow neighborhood")
```

Supported metacharacters

Metacharacter	Meaning
\d	Digit

```
re.findall(r"User\d", "The winners are: User9, UserN, User8")
```

```
['User9', 'User8']
```

Metacharacter	Meaning
\D	Non-digit

```
re.findall(r"User\D", "The winners are: User9, UserN, User8")
```

Supported metacharacters

Metacharacter	Meaning
\w	Word

```
re.findall(r"User\w", "The winners are: User9, UserN, User8")
```

```
['User9', 'UserN', 'User8']
```

Metacharacter	Meaning
\W	Non-word

```
re.findall(r"\W\d", "This skirt is on sale, only $5 today!")
```

```
['$5']
```

Supported metacharacters

Metacharacter	Meaning
\s	Whitespace

```
re.findall(r"Data\sScience", "I enjoy learning Data Science")
```

```
['Data Science']
```

Metacharacter	Meaning
\S	Non-Whitespace

```
re.sub(r"ice\Scream", "ice cream", "I really like ice-cream")
```

```
'I really like ice cream'
```

What are various methods of Regular Expressions?

The ‘re’ package provides multiple methods to perform queries on an input string. Here are the most commonly used methods, I will discuss:

re.match()
re.search()
re.findall()
re.split()
re.sub()
re.compile()

- Import the re module.

- Write a regex that matches the user mentions that follows the pattern, e.g. @robot3!.
- Find all the matches of the pattern in the sentiment_analysis variable.

script.py solution.py

```python
# Import the re module
import re

# Write the regex
regex = r"@robot\d\W"

print(sentiment_analysis)

# Find all matches of regex
print(re.findall(regex, sentiment_analysis))
```

Spyder (Python 3.8)

File Edit Search Source Run Debug Consoles Projects Tools View Help

E:\python\opencv-color-matching

E:\python\opencv-color-matching\untitled26.py

d18.py | tuple_ex111.py | untitled21.py | untitled22.py | untitled23.py | untitled24.py* | untitled25.py* | match_histograms.py | untitled26.py*

```python
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 23 22:36:00 2021

@author: Divya
"""
import re

sentiment_analysis="@robot9! @robot4& I have a good feeling that the show isgoing to be amazing! @robot9$ @robo


# Write the regex
regex = r"@robot\d\W"

print(sentiment_analysis)

# Find all matches of regex
print(re.findall(regex, sentiment_analysis))
```

Write a regex that matches the number of user mentions given as, for example, User_mentions:9 in sentiment_analysis.

```
# Write a regex to obtain user mentions
print(re.findall(r"User_mentions:\d", sentiment_analysis))
```

Write a regex that matches the number of likes given as, for example, likes: 5 in sentiment_analysis.

```
# Write a regex to obtain number of likes
print(re.findall(r"likes:\s\d", sentiment_analysis))
```

Write a regex that matches the number of retweets given as, for example, number of retweets: 4 in sentiment_analysis.

```
# Write a regex to obtain number of retweets
print(re.findall(r"number\sof\sretweets:\s\d", sentiment_analysis))
```

Repetitions

REGULAR EXPRESSIONS IN PYTHON

Repeated characters

Validate the following string:

password1234

```
import re
password = "password1234"
```

re.search() vs re.match() in python

Search function of re package will search the regular expression pattern and
Return the first occurrence.

Returns a match object when pattern is found
Returns “null” if pattern is not found

Match function of re package will ***search only from beginning of the string*** and return match object if found

But if match object found some where else in middle , it returns “None”

Repeated characters

Validate the following string:

password1234

```python
import re
password = "password1234"
```

```python
re.search(r"\w\w\w\w\w\w\w\w\d\d\d\d", password)
```

Repeated characters

Validate the following string:

password1234

```
import re
password = "password1234"
```

```
re.search(r"\w{8}\d{4}", password)
```

> **Quantifiers:**

> A metacharacter that tells the regex engine how many times to match a character immediately to its left.

Quantifiers

- Once or more: `+`

```
text = "Date of start: 4-3. Date of registration: 10-04."
```

```
re.findall(r"\d+-\d+", text)
```

```
['4-3', '10-04']
```

Quantifiers

- Zero times or more: `*`

```
my_string = "The concert was amazing! @ameli!a @joh&&n @mary90"
re.findall(r"@\w+\W*\w+", my_string)
```

```
['@ameli!a', '@joh&&n', '@mary90']
```

Quantifiers

- Zero times or once: `?`

```
text = "The color of this image is amazing. However, the colour blue could be brighter."
re.findall(r"colou?r", text)
```

```
['color', 'colour']
```

Quantifiers

- n times at least, m times at most : `{n, m}`

```
phone_number = "John: 1-966-847-3131 Michelle: 54-908-42-42424"
```

```
re.findall(r"\d{1,2}-\d{3}-\d{2,3}-\d{4,}", phone_number)
```

```
['1-966-847-3131', '54-908-42-42424']
```

Quantifiers

- *Immediately to the left*
  - `r"apple+"` : `+` applies to e and not to apple

- Import the re module.
- Write a regex to find all the matches of http links appearing in each tweet in sentiment_analysis. Print out the result.
- Write a regex to find all the matches of user mentions appearing in each tweet in sentiment_analysis. Print out the result.

```python
# -*- coding: utf-8 -*-
"""
Created on Wed Feb 24 02:00:33 2021

@author: Divya
"""

# Import re module
import re
sentiment_analysis="Boredd. Colddd @blueKnight39 Internet keeps url ht
print(sentiment_analysis)
for tweet in sentiment_analysis:
    # Write regex to match http links and print out result
    print(re.findall(r"http\S+", tweet))

    # Write regex to match user mentions and print out result
    print(re.findall(r"@\w+", tweet))
```

re.search(pattern, string):

It is similar to match() but it doesn’t restrict us to find matches at the beginning of the string only. Unlike previous method, here searching for pattern ‘Analytics’ will return a match.

Code

```
result = re.search(r'Analytics', 'AV Analytics Vidhya AV')
print(result.group(0))
```

Output:

Analytics
Here you can see that, search() method is able to find a pattern from any position of the string but it only returns the first occurrence of the search pattern

re.findall (pattern, string):

It helps to get a list of all matching patterns. It has no constraints of searching from start or end. If we will use method findall to search ‘AV’ in given string it will return both occurrence of AV. While searching a string, I would recommend you to use re.findall() always, it can work like re.search() and re.match() both.

Code

```
result = re.findall(r'AV', 'AV Analytics Vidhya AV')
print result
```

Output:

```
['AV', 'AV']
```

re.split(pattern, string, [maxsplit=0]):

This methods helps to split string by the occurrences of given pattern.

Code

```
result=re.split(r'y','Analytics')
result
```

Output:

```
['Anal', 'tics']
```

Above, we have split the string “Analytics” by “y”. Method split() has another argument “maxsplit“. It has default value of zero. In this case it does the maximum splits that can be done,

		syntax
search	Return first matching string	re.search(r'Analytics', 'AV Analytics Vidhya AV')
Findall	Return all matching strings	re.findall(r’SEARCH_TEXT , textinput')
Split	Splits text based on input , maxsplit is optional	re.split(r’text_to_split,”input_text”)
sub	It helps to search a pattern and replace with a new sub string.	re.sub(r’text_to_split,”input_text”)

if we give value to maxsplit, it will split the string. Let’s look at the example below:

Code

```
result=re.split(r'i','Analytics Vidhya')
print result
```

Output:
['Analyt', 'cs V', 'dhya'] #It has performed all the splits that can be done by pattern "i".

Code

```
result=re.split(r'i','Analytics Vidhya',maxsplit=1)
result
```

Output:
['Analyt', 'cs Vidhya']

Here, you can notice that we have fixed the maxsplit to 1. And the result is, it has only two values whereas first example has three values.

re.sub(pattern, repl, string):

It helps to search a pattern and replace with a new sub string. If the pattern is not found, string is returned unchanged.

Code

```
result=re.sub(r'India','the World','AV is largest Analytics community of India')
```

Result

Output:

'AV is largest Analytics community of the World'

re.compile(pattern, repl, string):

We can combine a regular expression pattern into pattern objects, which can be used for pattern matching. It also helps to search a pattern again without rewriting it.

Code

```
import re
pattern=re.compile('AV')
result=pattern.findall('AV Analytics Vidhya AV')
print(result)
result2=pattern.findall('AV is largest analytics community of India')
print(result2)
```

Output:
['AV', 'AV']
['AV']

Regex in Python

[abc] - Matches a or b or c.

[a-zA-Z0-9] - Matches any letter from (a to z) or (A to Z) or (0 to 9). Characters that are not within a range can be matched by complementing the set. If the first character of the set is ^, all the characters that are not in the set will be matched.

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Jan 10 12:42:28 2021

@author: Divya
"""

import re
li=['file1.txt','FILE2.txt','file3.txt']
for val in li:
 if(re.match('[a-zA-Z]',val)):
     print(val," is matching file as per standard")
 else:
     print(val +" file doesn't match the naming standards !")
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

Help | Variable explorer | Plots | Files

Console 1/A

```
FILE2.txt file doesn't match the naming
standards !
file3.txt  is matching file as per standard

In [16]: runfile('C:/Users/Divya/Downloads/
regex_ex2_a-z.py', wdir='C:/Users/Divya/
Downloads')
01file1.txt file doesn't match the naming
standards !
FILE2.txt file doesn't match the naming
standards !
file3.txt  is matching file as per standard

In [17]: runfile('C:/Users/Divya/Downloads/
regex_ex2_a-z.py', wdir='C:/Users/Divya/
Downloads')
file1.txt  is matching file as per standard
FILE2.txt  is matching file as per standard
file3.txt  is matching file as per standard

In [18]:
```

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Jan 10 12:42:28 2021

@author: Divya
"""

import re
li=['file1.txt','FILE2.txt','file3.txt']
for val in li:
 if(re.match('[a-z]',val)):
     print(val," is matching file as per standard")
 else:
     print(val +" file doesn't match the naming standards !")
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

Help | Variable explorer | Plots | Files

Console 1/A

```
standards !
FILE2.txt file doesn't match the naming
standards !
file3.txt  is matching file as per standard

In [17]: runfile('C:/Users/Divya/Downloads/
regex_ex2_a-z.py', wdir='C:/Users/Divya/
Downloads')
file1.txt  is matching file as per standard
FILE2.txt  is matching file as per standard
file3.txt  is matching file as per standard

In [18]: runfile('C:/Users/Divya/Downloads/
regex_ex2_a-z.py', wdir='C:/Users/Divya/
Downloads')
file1.txt  is matching file as per standard
FILE2.txt file doesn't match the naming
standards !
file3.txt  is matching file as per standard

In [19]:
```

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Jan 10 12:42:28 2021

@author: Divya
"""

import re
li=['01file1.txt','FILE2.txt','file3.txt']
for val in li:
 if(re.match('[a-zA-Z0-9]',val)):
     print(val," is matching file as per standard")
 else:
     print(val +" file doesn't match the naming standards !")
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

Help | Variable explorer | Plots | Files

Console 1/A

```
file1.txt  is matching file as per standard
FILE2.txt file doesn't match the naming
standards !
file3.txt  is matching file as per standard

In [19]: runfile('C:/Users/Divya/Downloads/
regex_ex2_a-z.py', wdir='C:/Users/Divya/
Downloads')
01file1.txt file doesn't match the naming
standards !
FILE2.txt  is matching file as per standard
file3.txt  is matching file as per standard

In [20]: runfile('C:/Users/Divya/Downloads/
regex_ex2_a-z.py', wdir='C:/Users/Divya/
Downloads')
01file1.txt  is matching file as per standard
FILE2.txt  is matching file as per standard
file3.txt  is matching file as per standard

In [21]:
```

You can easily tackle many basic patterns in Python using the ordinary characters.

Ordinary characters are the simplest regular expressions.

They match themselves exactly and do not have a special meaning in their regular expression syntax.

```
import re
pattern = r"Cookie"
sequence = "Cookie"
if re.match(pattern, sequence):
  print("Match!")
else: print("Not a match!")
```

Extract data from HTML file

```
import re

html_text = open('html_file.html').read()
text_filtered = re.sub(r'<(.*?)>', '', html_text)
print(text_filtered)
```

Read data from web

```
import requests

url = "http://news.bbc.co.uk/2/hi/health/2284783.stm"
res = requests.get(url)
text = res.text
print(text)
```

Extract Data from web HTML page

```
import requests
import re

url =
"http://news.bbc.co.uk/2/hi/health/2284783.stm"
res = requests.get(url)
text = res.text
print(text)
text_filtered = re.sub(r'<(.*?)>', '', text)
print(text_filtered)
```

Validate a phone number (phone number must be of 10 digits and starts with 8 or 9)
We have a list phone numbers in list “li” and here we will validate phone numbers using regular

Solution

Code

```
import re
li=['9999999999','999999-999','99999x9999']
for val in li:
 if re.match(r'[8-9]{1}[0-9]{9}',val) and len(val) == 10:
     print 'yes'
 else:
     print 'no'
```

Output:

```
yes
no
no
```

Split a string with multiple delimiters

Code

```
import re
line = 'asdf fjdk;afed,fjek,asdf,foo' # String has multiple delimiters (";",",","," ").
result= re.split(r'[;,\s]', line)
print result
```

Solution:

['asdf', 'fjdk', 'afed', 'fjek', 'asdf', 'foo']
We can also use method re.sub() to replace these multiple delimiters with one as space ”
“.

Code

```
import re
line = 'asdf fjdk;afed,fjek,asdf,foo'
result= re.sub(r'[;,\s]',' ', line)
print result
```

Output:

asdf fjdk afed fjek asdf foo

\b - Lowercase b. Matches only the beginning or end of the word.

```
re.search(r'\b[A-E]ookie', 'Cookie').group()
```

'Cookie'

Function provided by 're'

The `re` library in Python provides several functions to make your tasks easier. You have already seen some of them, such as the `re.search()`, `re.match()`. Let's check out more...

`compile(pattern, flags=0)`

Regular expressions are handled as strings by Python. However, with `compile()`, you can computer a regular expression pattern into a regular expression object. When you need to use an expression several times in a single program, using `compile()` to save the resulting regular expression object for reuse is more efficient than saving it as a string. This is because the compiled versions of the most recent patterns passed to `compile()` and the module-level matching functions are cached.

```python
pattern = re.compile(r"cookie")
sequence = "Cake and cookie"
pattern.search(sequence).group()
```

```
'cookie'
```

```python
# This is equivalent to:
re.search(pattern, sequence).group()
```

```
'cookie'
```

`search(pattern, string, flags=0)`

With this function, you scan through the given string/sequence, looking for the first location where the regular expression produces a match. It returns a corresponding match object if found, else returns `None` if no position in the string matches the pattern. Note that `None` is different from finding a zero-length match at some point in the string.

```
pattern = "cookie"
sequence = "Cake and cookie"

re.search(pattern, sequence)
```

```
<re.Match object; span=(9, 15), match='cookie'>
```

Search gives us match of first appearence

```python
"""

import re

data="today is tuesday in a week, its second day of the
pattern="week"

print(re.search(pattern, data))
print(re.match(pattern, data))
```

```
Variable explorer  Help  Plots  Files
Console 1/A
asp.net: 364884
reactjs: 336710
ruby-on-rails: 328578
sql-server: 310020
python-3.x: 296604
objective-c: 291942
django: 276667
angular: 263593
angularjs: 262239

In [97]: runfile('C:/Users/matam/.spyder-py3/untitled44.py'
matam/.spyder-py3')
<re.Match object; span=(40, 44), match='week'>
None

In [98]: runfile('C:/Users/matam/.spyder-py3/untitled44.py'
matam/.spyder-py3')
<re.Match object; span=(22, 26), match='week'>
None

In [99]:
IPython console  History
```

\s - Lowercase s.

Matches a single whitespace character like: space, newline, tab, return.

```
re.search('Just\spracticals', 'Just practicals').group()
```

\n - Lowercase n. Matches newline.

\r - Lowercase r. Matches return.

\d - Lowercase d. Matches decimal digit 0-9.

```
re.search(r'c\d\dkie', 'c00kie').group()
```

'c00kie'

^ - Caret. Matches a pattern at the start of the string.

```
re.search(r'^Just', 'Just practicals').group()
```

'Just practicals'

$ - Matches a pattern at the end of string.

```
re.search('just$', 'Just Practicals').group()
```

'Just Practicals'

\A - Uppercase a. Matches only at the start of the string. Works across multiple lines as well.

```
re.search(r'\A[A-E]ookie', 'Cookie').group()
```

'Cookie'

`match(pattern, string, flags=0)`

Returns a corresponding match object if zero or more characters at the beginning of string match the pattern. Else it returns `None`, if the string does not match the given pattern.

```
pattern = "C"
sequence1 = "IceCream"
sequence2 = "Cake"

# No match since "C" is not at the start of "IceCream"
print("Sequence 1: ", re.match(pattern, sequence1))
print("Sequence 2: ", re.match(pattern,sequence2).group())
```

```
Sequence 1:  None
Sequence 2:  C
```

```
import re
result = re.match(r'AV', 'AV Analytics Vidhya AV')
print result
```

Output:
<_sre.SRE_Match object at 0x0000000009BE4370>

Above, it shows that pattern match has been found.

To print the matching string we'll use method group (It helps to return the matching string).

Use "r" at the start of the pattern string, it designates a python raw string.

Let’s now find ‘Analytics’ in the given string. Here we see that string is not starting with ‘AV’ so it should return no match. Let’s see what we get:

Code

```
result = re.match(r'Analytics', 'AV Analytics Vidhya AV')
print(result)
```

Output:
None
There are methods like start() and end() to know the start and end position of matching pattern in the string.

There are methods like start() and end() to know the start and end position of matching pattern in the string.

Code

```
result = re.match(r'AV', 'AV Analytics Vidhya AV')
print(result.start())
print(result.end())
```

Output:

0
2

Above you can see that start and end position of matching pattern ‘AV’ in the string and sometime it helps a lot while performing manipulation with the string.

Looking for patterns

Two different operations to find a match:

re.search(r”regex”, string)

```
re.search(r"\d{4}", "4506 people attend the show")
```

```
<_sre.SRE_Match object; span=(0, 4), match='4506'>
```

re.match(r”regex”, string)

```
re.match(r"\d{4}", "4506 people attend the show")
```

```
<_sre.SRE_Match object; span=(0, 4), match='4506'>
```

Looking for patterns

Two different operations to find a match:

re.search(r”regex”, string)

```
re.search(r"\d{4}", "4506 people attend the show")
```

```
<_sre.SRE_Match object; span=(0, 4), match='4506'>
```

```
re.search(r"\d+", "Yesterday, I saw 3 shows")
```

```
<_sre.SRE_Match object; span=(17, 18), match='3'>
```

re.match(r”regex”, string)

```
re.match(r"\d{4}", "4506 people attend the show")
```

```
<_sre.SRE_Match object; span=(0, 4), match='4506'>
```

```
re.match(r"\d+","Yesterday, I saw 3 shows")
```

```
None
```

`search()` versus `match()`

The `match()` function checks for a match only at the beginning of the string (by default), whereas the `search()` function checks for a match anywhere in the string.

re.search(pattern, string):

It is similar to match() but it doesn’t restrict us to find matches at the beginning of the string only. Unlike previous method, here searching for pattern ‘Analytics’ will return a match.

Code

```
result = re.search(r'Analytics', 'AV Analytics Vidhya AV')
print(result.group(0))
```

Output:

Analytics
Here you can see that, search() method is able to find a pattern from any position of the string but it only returns the first occurrence of the search pattern

 # Matches any character except 5

```
re.search(r'Number: [^5]', 'Number: 0').group()
```

Special characters

- Match any character (except newline): `.`

www.domain.com

```
my_links = "Just check out this link: www.amazingpics.com. It has amazing photos!"
```

Special characters

- Match any character (except newline): `.`

www.domain.com

```
my_links = "Just check out this link: www.amazingpics.com. It has amazing photos!"
re.findall(r"www.+com", my_links)
```

```
['www.amazingpics.com']
```

Special characters

- Start of the string: `^`

```
my_string = "the 80s music was much better that the 90s"
```

```
re.findall(r"the\s\d+s", my_string)
```

```
['the 80s', 'the 90s']
```

Special characters

- End of the string: `$`

```
my_string = "the 80s music hits were much better that the 90s"
```

```
re.findall(r"the\s\d+s$", my_string)
```

```
['the 90s']
```

Special characters

- Escape special characters: `\`

```
my_string = "I love the music of Mr.Go. However, the sound was too loud."
```

```
print(re.split(r".\s", my_string))
```

```
['', 'lov', 'th', 'musi', 'o', 'Mr.Go', 'However', 'th', 'soun', 'wa', 'to', 'loud.']
```

Special characters

- Escape special characters: `\`

```
my_string = "I love the music of Mr.Go. However, the sound was too loud."
```

```
print(re.split(r".\s", my_string))
```

```
['', 'lov', 'th', 'musi', 'o', 'Mr.Go', 'However', 'th', 'soun', 'wa', 'to', 'loud.']
```

```
print(re.split(r"\.\s", my_string))
```

```
['I love the music of Mr.Go', 'However, the sound was too loud.']
```

OR operator

- Character: `|`

```
my_string = "Elephants are the world's largest land animal! I would love to see an elephant one day"
```

```
re.findall(r"Elephant|elephant", my_string)
```

OR operator

- Set of characters: `[ ]`

```
my_string = "Yesterday I spent my afternoon with my friends: MaryJohn2 Clary3"
```

```
re.findall(r"[a-zA-Z]+\d", my_string)
```

OR operator

- Set of characters: `[ ]`

```
my_string = "Yesterday I spent my afternoon with my friends: MaryJohn2 Clary3"
```

```
re.findall(r"[a-zA-Z]+\d", my_string)
```

```
['MaryJohn2', 'Clary3']
```

OR operator

- Set of characters: `[ ]`

```
my_string = "My&name&is#John Smith. I%live$in#London."
```

```
re.sub(r"[#$%&]", " ", my_string)
```

```
'My name is John Smith. I live in London.'
```

OR operand

- Set of characters: `[ ]`
  - `^` transforms the expression to negative

```
my_links = "Bad website: www.99.com. Favorite site: www.hola.com"
re.findall(r"www[^0-9]+com", my_links)
```

```
['www.hola.com
```

```python
import re

string2="#ipl www.99.com www.99acres.com www.justbuild.com will
string="91-8150965676"

result=re.sub("[#$|^]","",string2)


result1= re.findall("www[^a-zA-Z]+com",string2)
print(result1)
```

ax	Array o
b	int
c	int
climate_change	DataFra
colored_arrays	Array o
colored_tuples	list
computer_science	Series
cs_max	float

Console 1/A

```
[]

In [289]: runfile('C:/U
['www.justbuild.com']

In [290]: runfile('C:/U
[]

In [291]: runfile('C:/U
[]

In [292]: runfile('C:/U
[]

In [293]: runfile('C:/U
[]

In [294]: runfile('C:/U
['www.99.com']

In [295]:
```

- Write a regular expression to match valid email addresses as described.
- Match the regex to the elements contained in emails.
- To print out the message indicating if it is a valid email or not, complete .format() statement.

```
# Write a regex to match a valid email address
regex = r"[A-Za-z0-9!#%&*\$\.]+@\w+\.com"

for example in emails:
    # Match the regex to the string
    if re.match(regex, example):
        # Complete the format method to print out the result
        print("The email {email_example} is a valid email".format(email
_example=example))
    else:
        print("The email {email_example} is invalid".format(email_exam
ple=example))
```

- Write a regular expression to match valid passwords as described.
- Scan the elements in the passwords list to find out if they are valid passwords.
- To print out the message indicating if it is a valid password or not, complete .format() statement.

```
# Write a regex to match a valid password
regex = r"[A-Za-z0-9!#%&*\$\.]{8,20}"

for example in passwords:
    # Scan the strings to find a match
    if re.search(regex, example):
        # Complete the format method to print out the result
        print("The password {pass_example} is a valid password".format(pass_example=example))
    else:
        print("The password {pass_example} is invalid".format(pass_example=example))
```

Greedy vs. non-greedy matching

REGULAR EXPRESSIONS IN PYTHON

Greedy vs. non-greedy matching

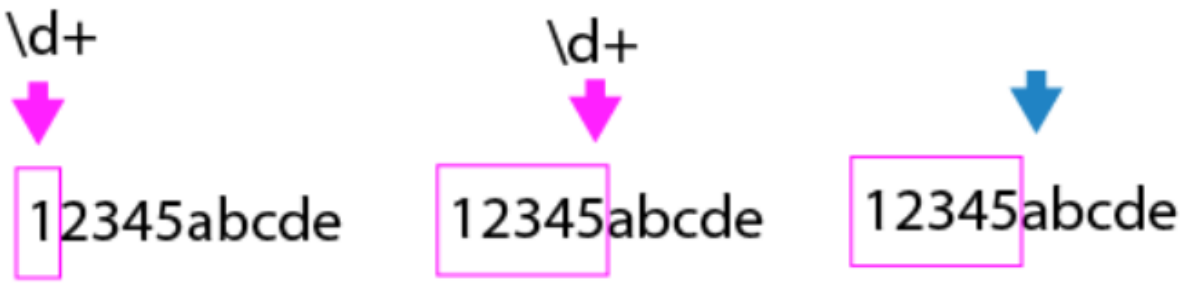
- Two types of matching methods:
  - Greedy
  - Non-greedy or lazy
- Standard quantifiers are greedy by default: `*` , `+` , `?` , `{num, num}`

Greedy matching

- **Greedy**: match as many characters as possible
- Return the *longest match*

```
import re
re.match(r"\d+", "12345bcada")
```

```
<_sre.SRE_Match object; span=(0, 5), match='12345'>
```

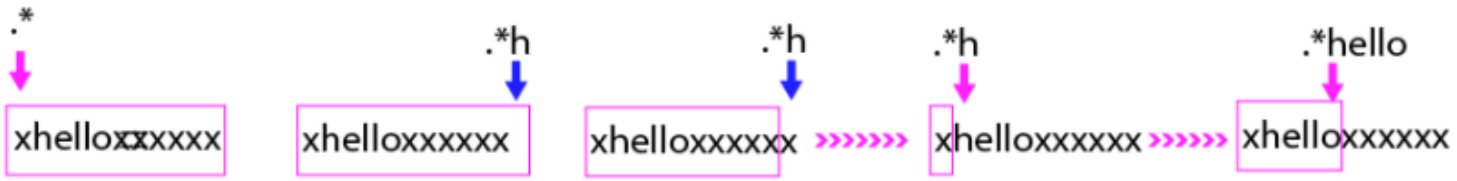


Greedy matching

- Backtracks when too many character matched
- Gives up characters one at a time

```
import re
re.match(r".*hello", "xhelloxxxxxx")
```

```
<_sre.SRE_Match object; span=(0, 6), match='xhello'>
```



Non-greedy matching

- **Lazy**: match as few characters as needed
- Returns *the shortest match*
- Append `?` to greedy quantifiers

```
import re
re.match(r"\d+?", "12345bcada")
```

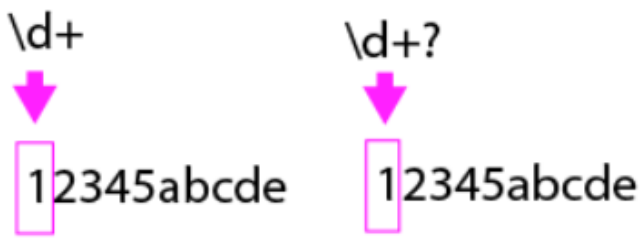
```
<_sre.SRE_Match object; span=(0, 1), match='1'>
```

Non-greedy matching

- **Lazy**: match as few characters as needed
- Returns *the shortest match*
- Append `?` to greedy quantifiers

```
import re
re.match(r"\d+?", "12345bcada")
```

```
<_sre.SRE_Match object; span=(0, 1), match='1'>
```



Non-greedy matching

- Backtracks when too few characters matched
- Expands characters one a time

```
import re
re.match(r".*?hello", "xhelloxxxxxx")
```

```
<_sre.SRE_Match object; span=(0, 6), match='xhello'>
```

- Import the `re` module.
- Write a `regex` expression to replace **HTML tags** with an empty string.
- Print out the result.

script.py | **solution.py**

```python
# Import re
import re
print(string)
# Write a regex to eliminate tags
string_notags = re.sub(r"<.+?>", "", string)

# Print out the result
print(string_notags)
```

- Import the `re` module.
- Write a `regex` expression to replace **HTML tags** with an empty string.
- Print out the result.

script.py | **solution.py**

```python
# Import re
import re
print(string)
# Write a regex to eliminate tags
string_notags = re.sub(r"<.+?>", "", string)

# Print out the result
print(string_notags)
```

1. Use a greedy quantifier to match text that appears within parentheses in the variable `sentiment_analysis`.

```
# Write a greedy regex expression to match
sentences_found_greedy = re.findall(r"\(.*\)", sentiment_analysis)

# Print out the result
print(sentences_found_greedy)
```

Now, use a lazy quantifier to match text that appears within parentheses in the variable `sentiment_analysis` .

```
# Write a lazy regex expression
sentences_found_lazy = re.findall(r"\(.*?\)", sentiment_analysis)

# Print out the results
print(sentences_found_lazy)
```

- Complete the regex to match the email capturing only the name part. The name part appears before the `@` .
- Find all matches of the regex in each element of `sentiment_analysis` analysis. Assign it to the variable `email_matched` .
- Complete the `.format()` method to print the results captured in each element of `sentiment_analysis` analysis.

```
# Write a regex that matches email
regex_email = r"([A-Za-z0-9]+)@\S+"

for tweet in sentiment_analysis:
    # Find all matches of regex in each tweet
    email_matched = re.findall(regex_email, tweet)

    # Complete the format method to print the results
    print("Lists of users found in this tweet: {}".format(email_matched))
```

Finds all the possible matches in the entire sequence and returns them as a list of strings. Each returned string represents one match.

```
statement = "Please contact us at: support@datacamp.com, xyz@datacamp.com"

#'addresses' is a list that stores all the possible match
addresses = re.findall(r'[\w\.-]+@[\w\.-]+', statement)
for address in addresses:
    print(address)
```

```
support@datacamp.com
xyz@datacamp.com
```

Return the first word of a given string
Solution-1 Extract each character (using “\w“)

Code

```
import re
result=re.findall(r'.','AV is largest Analytics community of India')
print result
```

Output:
['A', 'V', ' ', 'i', 's', ' ', 'l', 'a', 'r', 'g', 'e', 's', 't', ' ', 'A', 'n', 'a', 'l', 'y', 't', 'i', 'c', 's', ' ', 'c', 'o', 'm', 'm', 'u', 'n', 'i', 't', 'y', ' ', 'o', 'f', ' ', 'I', 'n', 'd', 'i', 'a']
Above, space is also extracted, now to avoid it use “\w” instead of “.“.

Solution-2 Extract each word (using “*” or “+“)

Code

```
result=re.findall(r'\w*','AV is largest Analytics community of India')
print result
```

Output:
['AV', '', 'is', '', 'largest', '', 'Analytics', '', 'community', '', 'of', '', 'India', '']

Again, it is returning space as a word because “*” returns zero or more matches of pattern to its left. Now to remove spaces we will go with “+“.

Code

```
result=re.findall(r'\w+','AV is largest Analytics community of India')
print result
```

Output:
['AV', 'is', 'largest', 'Analytics', 'community', 'of', 'India']

Return the first two character of each word

Solution-1 Extract consecutive two characters of each word, excluding spaces (using “\w“)

Code

```
result=re.findall(r'\w\w','AV is largest Analytics community of India')
print(result)
```

Output:

```
['AV', 'is', 'la', 'rg', 'es', 'An', 'al', 'yt', 'ic', 'co', 'mm', 'un', 'it', 'of', 'In', 'di']
```

Solution-2 Extract consecutive two characters those available at start of word boundary (using “\b“)

Code:

```
result=re.findall(r'\b\w.','AV is largest Analytics community of India')
print(result)
```

Output:

```
['AV', 'is', 'la', 'An', 'co', 'of', 'In']
```

Return the domain type of given email-ids

To explain it in simple manner, I will again go with a stepwise approach:

Solution-1 Extract all characters after “@”

Code

```
result=re.findall(r'@\w+','abc.test@gmail.com, xyz@test.in, test.first@analyticsvidhya.com,
first.test@rest.biz')
print(result)
```

Output: ['@gmail', '@test', '@analyticsvidhya', '@rest']

Above, you can see that “.com”, “.in” part is not extracted. To add it, we will go with below code.

```
result=re.findall(r'@\w+.\w+','abc.test@gmail.com, xyz@test.in, test.first@analyticsvidhya.com,
first.test@rest.biz')
print(result)
```

Output:
['@gmail.com', '@test.in', '@analyticsvidhya.com', '@rest.biz']

Extract only domain name using “( )”

Code

```
result=re.findall(r'@\w+.(\w+)','abc.test@gmail.com, xyz@test.in,
test.first@analyticsvidhya.com, first.test@rest.biz')
print(result)
```

Output:
['com', 'in', 'com', 'biz']

Return date from given string

Here we will use “\d” to extract digit.

Solution:

Code

```
result=re.findall(r'\d{2}-\d{2}-\d{4}','Amit 34-3456 12-05-2007, XYZ 56-4532 11-11-2011, ABC 67-8945 12-01-2009')
print(result)
```

Output:
['12-05-2007', '11-11-2011', '12-01-2009']

If you want to extract only year again parenthesis “( )” will help you.

Code

```
result=re.findall(r'\d{2}-\d{2}-(\d{4})','Amit 34-3456 12-05-2007, XYZ 56-4532 11-11-2011, ABC 67-8945 12-01-2009')
print result
```

Output:
['2007', '2011', '2009']

Return all words of a string those starts with vowel

Solution-1 Return each words

Code

```
result=re.findall(r'\w+','AV is largest Analytics community of India')
print(result)
```

Output:
['AV', 'is', 'largest', 'Analytics', 'community', 'of', 'India']

Return words starts with alphabets (using [])

Code

```
result=re.findall(r'[aeiouAEIOU]\w+','AV is largest Analytics community of India')
print result
```

Output:

```
['AV', 'is', 'argest', 'Analytics', 'ommunity', 'of', 'India']
```

Complete the for-loop with a regex that finds all dates in a format similar to 1st september 2019 17:25

```
print(re.findall(r"\d{1,2}\w+\s\w+\s\d{4}\s\d{1,2}:\d{2}", date))
```

Complete the for-loop with a regex that finds all dates in a format similar to 23rd june 2018

```
print(re.findall(r"\d{1,2}\w+\s\w+\s\d{4}", date))
```

`finditer(string, [position, end_position])`

Similar to `findall()` - it finds all the possible matches in the entire sequence but returns regex match objects as an iterator.

> **TIP:** `finditer()` might be an excellent choice when you want to have more information returned to you about your search. The returned regex match object holds not only the sequence that matched but also their positions in the original text.

```
statement = "Please contact us at: support@datacamp.com, xyz@datacamp.com"

#'addresses' is a list that stores all the possible match
addresses = re.finditer(r'[\w\.-]+@[\w\.-]+', statement)
for address in addresses:
    print(address)
```

```
<re.Match object; span=(22, 42), match='support@datacamp.com'>
<re.Match object; span=(44, 60), match='xyz@datacamp.com'>
```

`split(string, [maxsplit = 0])`

This splits the strings wherever the pattern matches and returns a list. If the optional argument `maxsplit` is nonzero, then the maximum 'maxsplit' number of splits are performed.

```
statement = "Please contact us at: xyz@datacamp.com, support@datacamp.com"
pattern = re.compile(r'[:,]')

address = pattern.split(statement)
print(address)
```

```
['Please contact us at', ' xyz@datacamp.com', ' support@datacamp.com']
```

`start()` - Returns the starting index of the match.

`end()` - Returns the index where the match ends.

`span()` - Return a tuple containing the (start, end) positions of the match.

```
pattern = re.compile('COOKIE', re.IGNORECASE)
match = pattern.search("I am not a cookie monster")

print("Start index:", match.start())
print("End index:", match.end())
print("Tuple:", match.span())
```

```
Start index: 11
End index: 17
Tuple: (11, 17)
```

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 29 08:40:54 2021

@author: Divya
"""
import re

string1="my body temparature reading in recent days 95 96 97 93  "

str_pattern="\d{2}"

regex_pattern=re.compile(str_pattern)

print(regex_pattern)

lst=regex_pattern.findall(string1)

print(lst)
```

Usage

Here you can get hel
Ctrl+I in front of it, ei
Console.

Help can also be sho
left parenthesis next
this behavior in Prefe

New to Spy

Help Varia

Console 1/A

```
Divya')
25
30
(25, 30)
@team
com

In [23]: runfile('C:/Users/Divya
Divya')
25
30
(25, 30)
@team
com

In [24]: runfile('C:/Users/Divya
Divya')
re.compile('\\d{2}')
['95', '96', '97', '93']

In [25]:
```

Compilation Flags

Did you notice the term `re.IGNORECASE` in the last example? Did you figure out its importance?

An expression's behavior can be modified by specifying a flag value. You can add flags as an extra argument to the different functions that you have seen in this tutorial. Some of the more useful ones are:

IGNORECASE (I) - Allows case-insensitive matches.
DOTALL (S) - Allows . to match any character, including newline.
MULTILINE (M) - Allows start of string (^) and end of string ($) anchor to match newlines as well.
VERBOSE (X) - Allows you to write whitespace and comments within a regular expression to make it more readable.

- Write a regex that matches the described hashtag pattern. Assign it to the regex variable.

```
# Write a regex matching the hashtag pattern
regex = r"#\w+"
```

- Replace all the matches of the regex with an empty string "". Assign it to no_hashtag variable.

```
# Write a regex matching the hashtag pattern
regex = r"#\w+"

# Replace the regex by an empty string
no_hashtag = re.sub(regex, "", sentiment_analysis)
```

- Split the text in the no_hashtag variable at every match of one or more consecutive whitespace.

```
# Write a regex matching the hashtag pattern
regex = r"#\w+"

# Replace the regex by an empty string
no_hashtag = re.sub(regex, "", sentiment_analysis)

# Get tokens by splitting text
print(re.split(r"\s+", no_hashtag))
```

Case Study: Working with Regular Expressions

Now that you have seen how regular expressions work in Python by studying some examples, it's time to get your hands dirty! In this case study, you'll put all your knowledge to work.

> You will work with the first part of a free e-book titled "The Idiot", written by Fyodor Dostoyevsky from the Project Gutenberg. The novel is about Prince (Knyaz) Lev Nikolayevich Myshkin, a guileless man whose good, kind, simple nature mistakenly leads many to believe he lacks intelligence and insight. The title is an ironic reference to this young man.

You shall be writing some regular expressions to parse through the text and complete some exercises.

```python
import re
import requests
the_idiot_url = 'https://www.gutenberg.org/files/2638/2638-0.txt'

def get_book(url):
    # Sends a http request to get the text from project Gutenberg
    raw = requests.get(url).text
    # Discards the metadata from the beginning of the book
    start = re.search(r"\*\*\* START OF THIS PROJECT GUTENBERG EBOOK .* \*\*\*",raw ).end()
    # Discards the text starting Part 2 of the book
    stop = re.search(r"II", raw).start()
    # Keeps the relevant text
    text = raw[start:stop]
    return text

def preprocess(sentence):
    return re.sub('[^A-Za-z0-9.]+' , ' ', sentence).lower()

book = get_book(the_idiot_url)
processed_book = preprocess(book)
#print(processed_book)
```

sers\Divya\regex_case_study.py

untitled23.py | untitled24.py* | untitled25.py* | match_histograms.py | untitled26.py* | regex_ex_findall.py | untitled30.py* | regex_case_study.py

```python
# -*- coding: utf-8 -*-
"""
Created on Wed Feb 24 02:18:35 2021

@author: Divya
"""

import re
import requests
the_idiot_url = 'https://www.gutenberg.org/files/2638/2638-0.txt'

def get_book(url):
    # Sends a http request to get the text from project Gutenberg
    raw = requests.get(url).text
    # Discards the metadata from the beginning of the book
    start = re.search(r"\*\*\* START OF THIS PROJECT GUTENBERG EBOOK .* \*\*\*",raw ).end()
    # Discards the text starting Part 2 of the book
    stop = re.search(r"II", raw).start()
    # Keeps the relevant text
    text = raw[start:stop]
    return text

def preprocess(sentence):
    return re.sub('[^A-Za-z0-9.]+' , ' ', sentence).lower()

book = get_book(the_idiot_url)
processed_book = preprocess(book)
#print(processed_book)
print(len(re.findall(r'the', processed_book)))
```

Source | Editor | Object

sentiment_analysis

Definition : sentiment_analysis(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) ->

Create a new string object from the given object. If encoding or errors is specified, then the object must expose data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'str

Help | Variable explorer | Plots | Files

Console 1/A

```
like you very much too. i liked you especially when you told us about the diamond
earrings but i liked you before that as well though you have such a dark clouded
of face. thanks very much for the offer of clothes and a fur coat i certainly sha
require both clothes and coat very soon. as for money i have hardly a copeck abou
at this moment. you shall have lots of money by the evening i shall have plenty s
come along that s true enough he ll have lots before evening put in lebedeff. but
look here are you a great hand with the ladies let s know that first asked rogoji
oh no oh no said the prince i couldn t you know my illness i hardly ever saw a so
h m well here you fellow you can come along with me now if you like cried rogojin
lebedeff and so they all left the carriage. lebedeff had his desire. he went off
the noisy group of rogojin s friends towards the voznesensky while the prince s r
lay towards the litaynaya. it was damp and wet. the prince asked his way of passe
by and finding that he was a couple of miles or so from his destination he determ
to take a droshky.

In [23]: runfile('C:/Users/Divya/regex_case_study.py', wdir='C:/Users/Divya')

In [24]: runfile('C:/Users/Divya/regex_case_study.py', wdir='C:/Users/Divya')
302

In [25]:
```

- Exercise: Try to convert every single stand-alone instance of 'i' to 'I' in the corpus. Make sure not to change the 'i' occurring within a word:

```
processed_book = re.sub(r'\si\s', " I ", processed_book)
#print(processed_book)
```

Backreferences

REGULAR EXPRESSIONS IN PYTHON

Numbered groups

Python 3.0 was released on 12-03-2008. It was a major revision of the language. Many of its major features were backported to Python 2.6.x and 2.7.x version series.

(\d{1,2})-(\d{1,2})-(\d{4})

Numbered groups

Python 3.0 was released on 12-03-2008. It was a major revision of the language. Many of its major features were backported to Python 2.6.x and 2.7.x version series.

Group 0

(\d{1,2})-(\d{1,2})-(\d{4})

Group1 Group2 Group3

Numbered groups

```
text = "Python 3.0 was released on 12-03-2008."
```

```
information = re.search('(\d{1,2})-(\d{2})-(\d{4})', text)
information.group(3)
```

Numbered groups

```
text = "Python 3.0 was released on 12-03-2008."
```

```
information = re.search('(\d{1,2})-(\d{2})-(\d{4})', text)
information.group(3)
```

```
'2008'
```

```
information.group(0)
```

```
'12-03-2008'
```

Named groups

- Give a name to groups

```
text = "Austin, 78701"
cities = re.search(r"(?P<city>[A-Za-z]+).*?(?P<zipcode>\d{5})", text)
cities.group("city")
```

Named groups

- Give a name to groups

```
text = "Austin, 78701"
cities = re.search(r"(?P<city>[A-Za-z]+).*?(?P<zipcode>\d{5})", text)
cities.group("city")
```

```
'Austin'
```

```
cities.group("zipcode")
```

```
'78701'
```

Backreferences

- Using capturing groups to reference back to a group

(\d{1,2})-(\d{1,2})-(\d{4})

\1 \2 \3

Backreferences

- Using numbered capturing groups to reference back

```
sentence = "I wish you a happy happy birthday!"
re.findall(r"(\w+)\s\1", sentence)
```

Backreferences

- Using numbered capturing groups to reference back

```
sentence = "I wish you a happy happy birthday!"
re.findall(r"(\w+)\s\1", sentence)
```

```
['happy']
```

Backreferences

- Using numbered capturing groups to reference back

```
sentence = "I wish you a happy happy birthday!"
re.sub(r"(\w+)\s\1", r"\1", sentence)
```

Backreferences

- Using named capturing groups to reference back

(?P<name>regex)

?P=name

```
sentence = "Your new code number is 23434. Please, enter 23434 to open the door."
re.findall(r"(?P<code>\d{5}).*?(?P=code)", sentence)
```

Lookaround

REGULAR EXPRESSIONS IN PYTHON

Looking around

- Allow us to confirm that sub-pattern is ahead or behind main pattern

Looking around

- Allow us to confirm that sub-pattern is ahead or behind main pattern

> At my current position in the matching process, look ahead or behind and examine whether some pattern matches or not match before continuing.

Look-ahead

- Non-capturing group
- Checks that the first part of the expression is followed or not by the lookahead expression
- Return only the first part of the expression

the white cat sat on the chair

Look ahead

positive (?=sat)

negative (?!run)

Positive look-ahead

- Non-capturing group
- Checks that the first part of the expression is followed by the lookahead expression
- Return only the first part of the expression

```
my_text = "tweets.txt transferred, mypass.txt transferred, keywords.txt error"
re.findall(r"\w+\.txt                 ", my_text)
```

Positive look-ahead

- Non-capturing group
- Checks that the first part of the expression is followed by the lookahead expression
- Return only the first part of the expression

```
my_text = "tweets.txt transferred, mypass.txt transferred, keywords.txt error"
re.findall(r"\w+\.txt(?=\stransferred)", my_text)
```

```
['tweets.txt', 'mypass.txt']
```

Negative look-ahead

- Non-capturing group
- Checks that the first part of the expression is **not** followed by the lookahead expression
- Return only the first part of the expression

```
my_text = "tweets.txt transferred, mypass.txt transferred, keywords.txt error"
re.findall(r"\w+\.txt(?!\stransferred)", my_text)
```

```
['keywords.txt']
```

Look-behind

- Non-capturing group
- Get all the matches that are preceded or not by a specific pattern.
- Return pattern after look-behind expression

the white cat sat on the chair

Look behind

positive (?<=white)

negative (?<!brown)

Template method

REGULAR EXPRESSIONS IN PYTHON

Basic syntax

```python
from string import Template
my_string = Template('Data science has been called $identifier')
my_string.substitute(identifier="sexiest job of the 21st century")
```

```
'Data science has been called sexiest job of the 21st century'
```

Substitution

- Use many `$identifier`
- Use variables

```
from string import Template
job = "Data science"
name = "sexiest job of the 21st century"
my_string = Template('$title has been called $description')
my_string.substitute(title=job, description=name)
```

```
'Data science has been called sexiest job of the 21st century'
```

Substitution

- Use `${identifier}` when valid characters follow identifier

```
my_string = Template('I find Python very ${noun}ing but my sister has lost $noun')
my_string.substitute(noun="interest")
```

```
'I find Python very interesting but my sister has lost interest'
```

Substitution

- Use `$$` to escape the dollar sign

```
my_string = Template('I paid for the Python course only $$ $price, amazing!')
my_string.substitute(price="12.50")
```

```
'I paid for the Python course only $ 12.50, amazing!'
```

Substitution

```
favorite = dict(flavor="chocolate")
my_string = Template('I love $flavor $cake very much')
```

```
try:
    my_string.substitute(favorite)
except KeyError:
      print("missing information")
```

```
missing information
```

r”(Congratulations!)+”