**Contents**

[Special Sequences 3](#_Toc16195327)

[Sets 4](#_Toc16195328)

[Metacharacters 4](#_Toc16195329)

[# **[a-b]** - set of characters 6](#_Toc16195330)

[# **{a}** - Check if the string contains "a" followed by exactly two "l" characters: 6](#_Toc16195331)

[# **|** - Check if the string contains either "falls" or "stays": 6](#_Toc16195332)

[# **\A** - Check if the string starts with "The": 6](#_Toc16195333)

[# **\b** - Check if "bain" is present at the beginning of a WORD: 6](#_Toc16195334)

[# **\b** - Check if "ain" is present at the end of a WORD: 6](#_Toc16195335)

[# **\B** - returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word 6](#_Toc16195336)

[# \w - Return a match at every word character (characters from a to Z, digits from 0-9, and the underscore \_ character): 7](#_Toc16195337)

[# **\Z** - Check if the string **ends with "Spain"** 7](#_Toc16195338)

[# Check if the string has any two-digit numbers, from **00** to **59**: 7](#_Toc16195339)

[# Check if the string has any characters from a to z lower case, and A to Z upper case: 7](#_Toc16195340)

[# Check if the string has any + characters: 7](#_Toc16195341)

[**RegEx** Functions 9](#_Toc16195342)

[# re.compile() 9](#_Toc16195343)

[# re.DOTALL 9](#_Toc16195344)

[# re.VERBOSE 9](#_Toc16195345)

[# 01.findall() 10](#_Toc16195346)

[# 02.finditer() 10](#_Toc16195347)

[# 04. Methods for objects 11](#_Toc16195348)

[# 04.01.span() 11](#_Toc16195349)

[# 04.02.string 11](#_Toc16195350)

[# 04.03.group() 11](#_Toc16195351)

[# 05.split() 11](#_Toc16195352)

[# 06.sub() 12](#_Toc16195353)

[# finditer() vs findall() 13](#_Toc16195354)

[**# .group()** 14](#_Toc16195355)

[**Phone Number and Email Address Extractor** 15](#_Toc16195356)

# Special Sequences

A special sequence is a \ followed by one of the characters in the list below, and has a special meaning:

|  |  |  |
| --- | --- | --- |
| Char | Description | e.g |
| \A | Returns a match if the specified characters are at the beginning of the **string** | "\AThe" |
| \b | Returns a match where the specified characters are at the beginning or at the end of a **word** | r"\bain" |
| r"ain\b" |
| \B | Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word | r"\Bain" |
| r"ain\B" |
| \d | Returns a match where the string contains digits (numbers from 0-9) | "\d" |
| \D | Returns a match where the string DOES NOT contain digits | "\D" |
| \s | Returns a match where the string contains a white space character | "\s" |
| \S | Returns a match where the string DOES NOT contain a white space character | "\S" |
| \w | Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore \_ character) | "\w" |
| \W | Returns a match where the string DOES NOT contain any word characters | "\W" |
| \Z | Returns a match if the specified characters are at the end of the string | "Spain\Z" |

# Sets

A set is a set of characters inside a pair of square brackets [] with a special meaning:

|  |  |
| --- | --- |
| Character | Description |
| [arn] | Returns a match where one of the specified characters (a, r, or n) are present |
| [a-n] | Returns a match for any lower case character, alphabetically between a and n |
| [^arn] | Returns a match for any character EXCEPT a, r, and n |
| [0123] | Returns a match where any of the specified digits (0, 1, 2, or 3) are present |
| [0-9] | Returns a match for any digit between 0 and 9 |
| [0-5][0-9] | Returns a match for any two-digit numbers from 00 and 59 |
| [a-zA-Z] | Returns a match for any character alphabetically between a and z, lower case OR upper case |
| [+] | In sets, +, \*, ., |, (), $,{} has no special meaning, so [+] means: return a match for any + character in the string |

# Metacharacters

Metacharacters are characters with a special meaning:

|  |  |  |
| --- | --- | --- |
| Charact | Description | Example |
| [] | A set of characters | "[a-m]" |
| \ | Signals a special sequence (can also be used to escape special characters) | "\d" |
| . | Any character (except newline character) | "he..o" |
| ^ | Starts with | "^hello" |
| $ | Ends with | "world$" |
| \* | Zero or more occurrences | "aix\*" |
| + | One or more occurrences | "aix+" |
| {} | Exactly the specified number of occurrences | "al{2}" |
| | | Either or | "falls|stays" |
| () | Capture and group |  |

. - Any Character Except New Line

\d - Digit (0-9)

\D - Not a Digit (0-9)

\w - Word Character (a-z, A-Z, 0-9, \_)

\W - Not a Word Character

\s - Whitespace (space, tab, newline)

\S - Not Whitespace (space, tab, newline)

\b - Word Boundary

\B - Not a Word Boundary

^ - Beginning of a String

$ - End of a String

[] - Matches Characters in brackets

[^ ] - Matches Characters NOT in brackets

| - Either Or

( ) - Group

Quantifiers:

\* - 0 or More

+ - 1 or More

? - 0 or One

{3} - Exact Number

{3,4} - Range of Numbers (Minimum, Maximum)

#### Sample Regexs ####

[a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+

# # **[a-b]** - set of characters

import re

str = "The rain in Spain"

x = re.findall("[a-m]", str) # a to m

print(x)

# ['h', 'e', 'a', 'i', 'i', 'a', 'i']

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# # **{a}** - Check if the string contains "a" followed by exactly two "l" characters:

x = re.findall("al{2}", str)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# # **|** - Check if the string contains either "falls" or "stays":

str = "The rain in Spain falls mainly in the plain!"

x = re.findall("falls|stays", str)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# # **\A** - Check if the string starts with "The":

str = "The rain in Spain"

x = re.findall("\AThe", str)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# # **\b** - Check if "bain" is present at the beginning of a WORD:

str = "The bain in Spain"

x = re.findall(r"\bbain", str)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# # **\b** - Check if "ain" is present at the end of a WORD:

str = "The rain in Spain"

x = re.findall(r"ain\b", str)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# # **\B** - returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word

str = "The rain in Spain"

# Check if a string end with "ain"

x = re.findall(r"ain\B", str)

print(x)

if (x):

print("Yes, there is at least one match!")

else:

print("No match")

# No match

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

str = "The rain in Spain 32"

# # \w - Return a match at every word character (characters from a to Z, digits from 0-9, and the underscore \_ character):

x = re.findall("\w", str)

print(x)

# ['T', 'h', 'e', 'r', 'a','i', 'n', 'i', 'n', 'S', 'p', 'a', 'i', 'n', '3', '2']

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

import re

str = "The rain in Spain"

# # **\Z** - Check if the string **ends with "Spain"**

x = re.findall("Spain\Z", str)

print(x)

if (x):

print("Yes, there is a match!")

else:

print("No match")

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

str = "8 times before 11:45 AM"

# # Check if the string has any two-digit numbers, from **00** to **59**:

x = re.findall("[0-5][0-9]", str)

print(x)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

str = "8 times before 11:45 AM"

# # Check if the string has any characters from a to z lower case, and A to Z upper case:

x = re.findall("[a-zA-Z]", str)

print(x)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

str = "8 times before 11:45 AM"

# # Check if the string has any + characters:

x = re.findall("[+]", str)

print(x)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

sentence = 'The quick brown fox + lion, in 1800, jump over the lazy cat.'

patt = re.compile(r'\+') # ! re.compile(r'+')

res = patt.findall(sentence)

print(res)

# **RegEx** Functions

The re module offers a set of functions that allows us to search a string for a match:

## # re.compile()

# Allows us to separate our patterns with a variable and enable us to perform multiple searches.

sentence = 'Start a sentence and then bring it to an end'

pattern = re.compile(r'start', re.I)

matches = pattern.search(sentence)

print(matches)

### # re.DOTALL

# the dot-star (.\*) matches everything except a newline. By passing re.DOTALL as the second argument to re.compile , you can make the dot character match all characters, including the newline character.

>>> noNewlineRegex = re.compile('.\*')

>>> noNewlineRegex.search('Serve the public trust.\nProtect the innocent.

\nUphold the law.').group()

# 'Serve the public trust.'

>>> newlineRegex = re.compile('.\*', re.DOTALL)

>>> newlineRegex.search('Serve the public trust.\nProtect the innocent.

\nUphold the law.').group()

# 'Serve the public trust.\nProtect the innocent.\nUphold the law.'

### # re.VERBOSE

# use re.VERBOSE to ignore whitespace**.** Spaces, tabs, and carriage returns are not matched as spaces, tabs, and carriage returns. This is useful when you want multiline regex expression for better readability and commeting stuffs.

phone\_regex = re.compile(r'''(

(\(\d{3}\)|\d{3}) # first group - area code

([-.\s]) # second group

(\d{3}) # third group - 3 digit numbers

([-.\s]) # fourth group

(\d{4}) # fifth group - 4 digit numbers

)''', re.VERBOSE)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## # 01.findall()

str = "The rain in Spain"

patt = re.compile(r'\Bai\B')

results = re.findall(patt, str)

for res in results:

print(res)

# ai

# ai

## # 02.finditer()

str = "The rain in Spain"

patt = re.compile(r'\Bai\B')

results = re.findall(patt, str)

results = re.finditer(patt, str)

for res in results:

print(res)

# <re.Match object; span=(5, 7), match='ai'>

# <re.Match object; span=(14, 16), match='ai'>

for res in results:

print(res.group()) # ai, ai

print(res.span()) # (5, 7), (14, 16)

# print(res.string) # The rain in Spain

**# 03.search()**

# The search() function searches the string for a match, and returns a Match object if there is a match.

str = "The rain in Spain"

patt = re.compile(r'\bS\w+')

x = re.search(patt, str)

print(x.string) # The rain in Spain

## # 04. Methods for objects

# A Match Object is an object containing information about the search and the result. for methods returning Strings we have the following methods to extract the exact index, the actual string or the matched value

### # 04.01.span()

# returns a tuple containing the start-, and end positions of the match.

str = "The rain in Spain"

patt = re.compile(r'\bS\w+')

x = re.search(patt, str)

print(x.span()) # (12 ,17)

### # 04.02.string

# returns the string passed into the function

str = "The rain in Spain"

patt = re.compile(r'\bS\w+')

x = re.search(patt, str)

print(x.string) # The rain in Spain

### # 04.03.group()

# returns the part of the string where there was a match

str = "The rain in Spain"

patt = re.compile(r'\bS\w+')

x = re.search(patt, str)

print(x.group()) # Spain

## # 05.split()

# The split() function returns a list where the string has been split at each match:

str = "The rain in Spain"

patt = re.compile(r'\s')

x = re.split(patt, str)

print(x) # ['The', 'rain', 'in', 'Spain']

# using third parameter to control number of occurences

str = "The rain in Spain"

patt = re.compile(r'\s')

x = re.split(patt, str, 1) #3rd param here!

print(x) # ['The', 'rain in Spain']

## # 06.sub()

# The sub() function replaces the matches with the text of your choice:

str = 'The rain in Spain'

patt = re.compile(r'\s')

res = re.sub(patt, r'[space]', str)

print(res) # The[space]rain[space]in[space]Spain

# 3rd param to control the number of replacements

str = 'The rain in Spain'

patt = re.compile(r'\s')

res = re.sub(patt, r'[space]', str, 2)

print(res) # The[space]rain[space]in Spain

# example:

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

res = numRegex.sub('X','12 drumers 11 pipers five rings , 3 hens')

print(res) # X drumers X pipers five rings , X hens

# example: replacing the url with domain name and top level domain name only

import re

urls = '''

https://www.google.com

http://coreyms.com

https://youtube.com

https://www.nasa.gov

'''

pattern = re.compile(r'https?://(www\.)?(\w+)(\.\w+)')

# replace urls with second and third group of regex

subbed\_urls = pattern.sub(r'\2\3', urls)

print(subbed\_urls)

# google.com

# coreyms.com

# youtube.com

# nasa.gov

matches = pattern.finditer(urls)

for match in matches:

print(match.group(3))

# google

# coreyms

# youtube

# nasa

## # finditer() vs findall()

# They returns an iterable so you can iterate and loop over them. One difference between finditer and findall is that the former returns matched objects whereas the other returns a tuple of of matched results.

import re

CARRIS\_REGEX=r'<th>(\d+)</th><th>([\s\w\.\-]+)</th><th>(\d+:\d+)</th><th>(\d+m)</th>'

pattern = re.compile(CARRIS\_REGEX, re.UNICODE)

mailbody = open("test.txt").read()

for match in pattern.finditer(mailbody):

print(match)

print()

for match in pattern.findall(mailbody):

print(match)

Outputs:

# results of .finditer() method

<\_sre.SRE\_Match object at 0x00A63758>

<\_sre.SRE\_Match object at 0x00A63F98>

<\_sre.SRE\_Match object at 0x00A63758>

<\_sre.SRE\_Match object at 0x00A63758>

<\_sre.SRE\_Match object at 0x00A63F98>

# results of .findall() method

('790', 'PR. REAL', '21:06', '04m')

('758', 'PORTAS BENFICA', '21:10', '09m')

('790', 'PR. REAL', '21:14', '13m')

('758', 'SETE RIOS', '21:49', '47m')

('758', 'SETE RIOS', '22:09', '68m')

If you want the same output from finditer() as you're getting from findall(), you need:

for match in pattern.finditer(mailbody):

print(tuple(match.groups()))

# **# .group()**

import re

txt = 'http://www.coreyms.edu'

patt = re.compile(r'''(

https?://

(www.)?

([a-zA-Z0-9.-]+)

(\.[a-zA-Z0-9]+)

)''', re.VERBOSE)

results = patt.finditer(txt)

for res in results:

print(res.group(0)) # http://www.coreyms.edu

print(res.group(1)) # http://www.coreyms.com

print(res.group(2)) # www.

print(res.group(3)) # coreyms

print(res.group(4)) # .edu

# **Phone Number and Email Address Extractor**

import re, pyperclip

txt = str(pyperclip.paste())

phone\_regex = re.compile(r'''(

(\(\d{3}\)|\d{3}) # first group - area code

([-.\s]) # second group

(\d{3}) # third group - 3 digit numbers

([-.\s]) # fourth group

(\d{4}) # fifth group - 4 digit numbers

)''', re.VERBOSE)

phone\_matches = phone\_regex.findall(txt)

phone\_emails = []

for res in phone\_matches:

phone\_emails.append('-'.join([res[1], res[3], res[5]]))

# copy to clipboard

# REM: only str, int, float, and bool values can be copied to the clipboard, not list

pyperclip.copy('\n'.join(phone\_emails))

# EMAIL

email\_regex = re.compile(r'(([a-zA-Z0-9.-]+)@([a-zA-Z0-9]+)\.(com|edu|ca))')

email\_matches = email\_regex.findall(txt)

for email in email\_matches:

phone\_emails.append(email[0])

# copy to clipboard

pyperclip.copy('\n'.join(phone\_emails))

print('Matched Phone Numbers and Email Adress:')

print('\n'.join(phone\_emails))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

import re

emails = '''

CoreyMSchafer@gmail.com

corey.schafer@university.edu

corey-321-schafer@my-work.net

'''

pattern = re.compile(r'[a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+')

matches = pattern.finditer(emails)

for match in matches:

print(match)

# <re.Match object; span=(1, 24), match='CoreyMSchafer@gmail.com'>

# <re.Match object; span=(25, 53), match='corey.schafer@university.edu'>

# <re.Match object; span=(54, 83), match='corey-321-schafer@my-work.net'>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# **# Greedy**

pattern = re.compile(r'(Ha){3,5}')

res = pattern.search('HaHaHaHaHaHaHaHaHaHaHa')

print(res.group())

# HaHaHaHaHa

# **# Non Greedy**

pattern = re.compile(r'(Ha){3,5}?')

res = pattern.search('HaHaHaHaHaHaHaHaHaHaHa')

print(res.group())

# HaHaHa

# 2 ways of using regex funcs

import re

txt = 'Hello World! I am Basir!'

patt = re.compile(r'Hello')

# res = re.sub(patt, 'Hi', txt)

res = patt.sub('Hi', txt)

print(res) # Hello World! I am Basir!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

import re

txt = 'Hello World! I am Basir!'

patt = re.compile(r'Hello')

res = patt.search(txt)

# res = re.search(patt, txt)

print(res)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# [Why does regular expression 'r[1900-2023]+' doesn't returns range of values between 1900 to 2023?](https://stackoverflow.com/questions/57556501/why-does-regular-expression-r1900-2023-doesnt-returns-range-of-values-betw)

[1900-2023] does not match any number between 1900 and 2023, rather matches any character that is a 1, 9, 0, -, 2, or 3. For your specific case, you could make a pattern that matches these numbers on your own:

19[0-9]{2}|20[01][0-9]|202[0-3]

Explanation:

19[0-9]{2} - "19" and exactly 2 numbers that range 0 - 9 (1900 - 1999)

| - OR

20[01][0-9] - "20" and either a 0 or 1 and another number that ranges 0 - 9 (2000 - 2019)

| - OR

202[0-3] - "202" and one number in a range 0 - 3 (2020 - 2023)