In this Python Tutorial, we will be learning about Regular Expressions (or RE, regex) in Python. Regular expressions are a powerful language for matching text patterns.

Patrick Loeber · · · · · ·

April 19, 2020 · 31 min read

Python Crash Course

Regular expressions (REs, or regexes, or regex patterns) are a powerful language for matching text patterns. Possible pattern examples for searches are, e.g., E-mail addresses or domain names. This article gives a basic introduction to regular expressions and shows how regular expressions work in Python. It will cover all the necessary concepts:

1) Methods to search for matches
2) Methods on a match object
3) Meta characters
4) More special sequences
5) Sets
6) Quantifier
7) Conditions

8) Grouping
9) Examples
10) Modification
11) Compilation flags

Regular expressions inside Python are made available through the `re module:

```
import re
```

Using regexes, you specify the rules for the set of possible strings that you want to match. Typically we first define our pattern that we want to search for, and use `re.compile()` on it. By default, our pattern is case sensitive.

```
test_string = '123abc456789abc123ABC'
pattern = re.compile(r'abc')
```

Note: It is recommended to use raw strings for the search:

```
## Use raw strings for the search pattern
a = '\tHello'
b = r'\tHello'
print(a)
print(b)
```

```
        Hello
    \tHello
```

Once we have our pattern, we can search for this pattern in the text / string that we want to look up.

- match(): Determine if the RE matches at the beginning of the string.
- search(): Scan through a string, looking for any location where this RE matches.
- findall(): Find all substrings where the RE matches, and returns them as a list.
- finditer(): Find all substrings where the RE matches, and returns them as an iterator.

We will cover these methods later:

- split(): Returns a list where the string has been split at each match
- sub(): Replaces one or many matches with a string

```
# finditer()
my_string = 'abc123ABC123abc'
pattern = re.compile(r'123')
matches = pattern.finditer(my_string)
for match in matches:
```

```
    print(match)
    print(match.span(), match.start(), match.end())
    print(match.group()) # returns the string

print()
# findall()
pattern = re.compile(r'123')
matches = pattern.findall(my_string)
for match in matches:
    print(match)

print()
# match
match = pattern.match(my_string)
print(match)
pattern = re.compile(r'abc')
match = pattern.match(my_string)
print(match)

print()
# search
match = pattern.search(my_string)
print(match)
```

```
    <re.Match object; span=(0, 3), match='123'>
    (0, 3) 0 3
    123
    <re.Match object; span=(15, 18), match='123'>
    (15, 18) 15 18
    123

    123
    123

    None
    <re.Match object; span=(0, 3), match='abc'>

    <re.Match object; span=(0, 3), match='abc'>
```

Note: Methods can also be used directly on the `re` module. It does not make that much of a difference, but some people prefer explicitely precompiling and binding the pattern to a reusable variable. (See https://stackoverflow.com/questions/452104/is-it-worth-using-pythons-re-compile)

```
matches = re.finditer(r'abc', test_string)
for match in matches:
    print(match)
```

```
    <re.Match object; span=(3, 6), match='abc'>
    <re.Match object; span=(12, 15), match='abc'>
```

- group(): Return the string matched by the RE
- start(): Return the starting position of the match

- end(): Return the ending position of the match
- span(): Return a tuple containing the (start, end) positions of the match

```
test_string = '123abc456789abc123ABC'
pattern = re.compile(r'abc')
matches = pattern.finditer(test_string)
for match in matches:
    print(match)
    print(match.span(), match.start(), match.end())
    print(match.group()) # returns the substring that was matched by the RE
```

```
    <re.Match object; span=(3, 6), match='abc'>
    (3, 6) 3 6
    abc
    <re.Match object; span=(12, 15), match='abc'>
    (12, 15) 12 15
    abc
```

Metacharacters are characters with a special meaning:
All meta characters: **. ^ $ * + ? { } [ ] \ | ( )**
Meta characters need need to be escaped (with) if we actually want to search for the char.

- . 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}"
- [] A set of characters "[a-m]"
- \ Signals a special sequence (can also be used to escape special characters) "\d"
- | Either or "falls|stays"
- ( ) Capture and group

```
test_string = 'python-engineer.com'
pattern = re.compile(r'\.')
matches = pattern.finditer(test_string)
for match in matches:
    print(match)
```

```
    <re.Match object; span=(15, 16), match='.'>
```

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

- \d :Matches any decimal digit; this is equivalent to the class [0-9].
- \D : Matches any non-digit character; this is equivalent to the class [^0-9].
- \s : Matches any whitespace character;
- \S : Matches any non-whitespace character;
- \w : Matches any alphanumeric (word) character; this is equivalent to the class [a-zA-Z0-9_].
- \W : Matches any non-alphanumeric character; this is equivalent to the class [^a-zA-Z0-9_].
- \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"
- \A Returns a match if the specified characters are at the beginning of the string "\AThe"
- \Z Returns a match if the specified characters are at the end of the string "Spain\Z"

```
test_string = 'hello 123_ heyho hohey'
pattern = re.compile(r'\d')
matches = pattern.finditer(test_string)
for match in matches:
    print(match)

print()
pattern = re.compile(r'\s')
matches = pattern.finditer(test_string)
for match in matches:
    print(match)

print()
pattern = re.compile(r'\w')
matches = pattern.finditer(test_string)
for match in matches:
    print(match)

print()
pattern = re.compile(r'\bhey')
matches = pattern.finditer('heyho hohey') # ho-hey, ho\nhey are matches!
for match in matches:
    print(match)

print()
pattern = re.compile(r'\Ahello')
matches = pattern.finditer(test_string)
for match in matches:
    print(match)

print()
pattern = re.compile(r'123_\Z')
matches = pattern.finditer(test_string)
for match in matches:
    print(match)
```

```
    <re.Match object; span=(6, 7), match='1'>
    <re.Match object; span=(7, 8), match='2'>
    <re.Match object; span=(8, 9), match='3'>

    <re.Match object; span=(5, 6), match=' '>
    <re.Match object; span=(10, 11), match=' '>
    <re.Match object; span=(16, 17), match=' '>

    <re.Match object; span=(0, 1), match='h'>
    <re.Match object; span=(1, 2), match='e'>
    <re.Match object; span=(2, 3), match='l'>
    <re.Match object; span=(3, 4), match='l'>
    <re.Match object; span=(4, 5), match='o'>
    <re.Match object; span=(6, 7), match='1'>
    <re.Match object; span=(7, 8), match='2'>
    <re.Match object; span=(8, 9), match='3'>
    <re.Match object; span=(9, 10), match='_'>
    <re.Match object; span=(11, 12), match='h'>
    <re.Match object; span=(12, 13), match='e'>
    <re.Match object; span=(13, 14), match='y'>
    <re.Match object; span=(14, 15), match='h'>
    <re.Match object; span=(15, 16), match='o'>
    <re.Match object; span=(17, 18), match='h'>
    <re.Match object; span=(18, 19), match='o'>
    <re.Match object; span=(19, 20), match='h'>
    <re.Match object; span=(20, 21), match='e'>
    <re.Match object; span=(21, 22), match='y'>

    <re.Match object; span=(0, 3), match='hey'>

    <re.Match object; span=(0, 5), match='hello'>
```

A set is a set of characters inside a pair of square brackets [] with a special meaning. Append multiple conditions back-to back, e.g. [aA-Z].
A ^ (caret) inside a set negates the expression.
A - (dash) in a set specifies a range if it is in between, otherwise the dash itself.

Examples:
- [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

```
test_string = 'hello 123_'
pattern = re.compile(r'[a-z]')
matches = pattern.finditer(test_string)
for match in matches:
    print(match)
```

```
    <re.Match object; span=(0, 1), match='h'>
    <re.Match object; span=(1, 2), match='e'>
    <re.Match object; span=(2, 3), match='l'>
    <re.Match object; span=(3, 4), match='l'>
    <re.Match object; span=(4, 5), match='o'>
```

```
dates = '''
01.04.2020

2020.04.01

2020-04-01
2020-05-23
2020-06-11
2020-07-11
2020-08-11

2020/04/02

2020_04_04
2020_04_04
'''

print('all dates with a character in between')
pattern = re.compile(r'\d\d\d\d.\d\d.\d\d')
matches = pattern.finditer(dates)
for match in matches:
    print(match)
print()

print('only dates with - or . in between')
pattern = re.compile(r'\d\d\d\d[-.]\d\d[-.]\d\d') #  no escape for the . here in the set
matches = pattern.finditer(dates)
for match in matches:
    print(match)

print()
print('only dates with - or . in between in May or June')
pattern = re.compile(r'\d\d\d\d[-.]0[56][-.]\d\d')
matches = pattern.finditer(dates)
for match in matches:
    print(match)

# a dash in a character set specifies a range if it is in between, otherwise the dash itself
print()
print('only dates with - or . in between in May, June, July')
pattern = re.compile(r'\d\d\d\d[-.]0[5-7][-.]\d\d') #  no escape for the . here in the set
matches = pattern.finditer(dates)
for match in matches:
    print(match)
```

```
    all dates with a character in between
    <re.Match object; span=(13, 23), match='2020.04.01'>
    <re.Match object; span=(25, 35), match='2020-04-01'>
    <re.Match object; span=(36, 46), match='2020-05-23'>
    <re.Match object; span=(47, 57), match='2020-06-11'>
    <re.Match object; span=(58, 68), match='2020-07-11'>
    <re.Match object; span=(69, 79), match='2020-08-11'>
    <re.Match object; span=(81, 91), match='2020/04/02'>
```

```
    <re.Match object; span=(93, 103), match='2020_04_04'>
    <re.Match object; span=(104, 114), match='2020_04_04'>

    only dates with - or . in between
    <re.Match object; span=(13, 23), match='2020.04.01'>
    <re.Match object; span=(25, 35), match='2020-04-01'>
    <re.Match object; span=(36, 46), match='2020-05-23'>
    <re.Match object; span=(47, 57), match='2020-06-11'>
    <re.Match object; span=(58, 68), match='2020-07-11'>
    <re.Match object; span=(69, 79), match='2020-08-11'>

    only dates with - or . in between in May or June
    <re.Match object; span=(36, 46), match='2020-05-23'>
    <re.Match object; span=(47, 57), match='2020-06-11'>

    only dates with - or . in between in May, June, July
    <re.Match object; span=(36, 46), match='2020-05-23'>
    <re.Match object; span=(47, 57), match='2020-06-11'>
    <re.Match object; span=(58, 68), match='2020-07-11'>
```

- * : 0 or more
- + : 1 or more
- ? : 0 or 1, used when a character can be optional
- {4} : exact number
- {4,6} : range numbers (min, max)

```python
my_string = 'hello_123'
pattern = re.compile(r'\d*')
matches = pattern.finditer(my_string)
for match in matches:
    print(match)

print()
pattern = re.compile(r'\d+')
matches = pattern.finditer(my_string)
for match in matches:
    print(match)

print()
my_string = 'hello_1_2-3'
pattern = re.compile(r'_?\d')
matches = pattern.finditer(my_string)
for match in matches:
    print(match)

print()
my_string = '2020-04-01'
pattern = re.compile(r'\d{4}') # or if you need a range r'\d{3,5}'
matches = pattern.finditer(my_string)
for match in matches:
    print(match)
```

```
    <re.Match object; span=(0, 0), match=''>
    <re.Match object; span=(1, 1), match=''>
    <re.Match object; span=(2, 2), match=''>
    <re.Match object; span=(3, 3), match=''>
    <re.Match object; span=(4, 4), match=''>
    <re.Match object; span=(5, 5), match=''>
    <re.Match object; span=(6, 9), match='123'>
    <re.Match object; span=(9, 9), match=''>

    <re.Match object; span=(6, 9), match='123'>

    <re.Match object; span=(5, 7), match='_1'>
    <re.Match object; span=(7, 9), match='_2'>
    <re.Match object; span=(10, 11), match='3'>

    <re.Match object; span=(0, 4), match='2020'>
```

```
dates = '''
2020.04.01

2020-04-01
2020-05-23
2020-06-11
2020-07-11
2020-08-11

2020/04/02

2020_04_04
2020_04_04
'''
pattern = re.compile(r'\d{4}.\d{2}.\d{2}')
matches = pattern.finditer(dates)
for match in matches:
    print(match)
print()

pattern = re.compile(r'\d+.\d+.\d+')
matches = pattern.finditer(dates)
for match in matches:
    print(match)
```

```
    <re.Match object; span=(1, 11), match='2020.04.01'>
    <re.Match object; span=(13, 23), match='2020-04-01'>
    <re.Match object; span=(24, 34), match='2020-05-23'>
    <re.Match object; span=(35, 45), match='2020-06-11'>
    <re.Match object; span=(46, 56), match='2020-07-11'>
    <re.Match object; span=(57, 67), match='2020-08-11'>
    <re.Match object; span=(69, 79), match='2020/04/02'>
    <re.Match object; span=(81, 91), match='2020_04_04'>
    <re.Match object; span=(92, 102), match='2020_04_04'>

    <re.Match object; span=(1, 11), match='2020.04.01'>
    <re.Match object; span=(13, 23), match='2020-04-01'>
    <re.Match object; span=(24, 34), match='2020-05-23'>
    <re.Match object; span=(35, 45), match='2020-06-11'>
    <re.Match object; span=(46, 56), match='2020-07-11'>
    <re.Match object; span=(57, 67), match='2020-08-11'>
    <re.Match object; span=(69, 79), match='2020/04/02'>
```

```
    <re.Match object; span=(81, 91), match='2020_04_04'>
    <re.Match object; span=(92, 102), match='2020_04_04'>
```

Use the | for either or condition.

```
my_string = """
Mr Simpson
Mrs Simpson
Mr. Brown
Ms Smith
Mr. T
"""
pattern = re.compile(r'Mr\.?\s\w+')
matches = pattern.finditer(my_string)
for match in matches:
    print(match)

print()
pattern = re.compile(r'(Mr|Ms|Mrs)\.?\s\w+')
matches = pattern.finditer(my_string)
for match in matches:
    print(match)
```

```
    <re.Match object; span=(1, 11), match='Mr Simpson'>
    <re.Match object; span=(24, 33), match='Mr. Brown'>
    <re.Match object; span=(43, 48), match='Mr. T'>

    <re.Match object; span=(1, 11), match='Mr Simpson'>
    <re.Match object; span=(12, 23), match='Mrs Simpson'>
    <re.Match object; span=(24, 33), match='Mr. Brown'>
    <re.Match object; span=(34, 42), match='Ms Smith'>
    <re.Match object; span=(43, 48), match='Mr. T'>
```

( ) is used to group substrings in the matches.

```
emails = """
pythonengineer@gmail.com
Python-engineer@gmx.de
python-engineer123@my-domain.org
"""
pattern = re.compile('[a-zA-Z1-9-]+@[a-zA-Z-]+\.[a-zA-Z]+')
pattern = re.compile('[a-zA-Z1-9-]+@[a-zA-Z-]+\.(com|de)')
pattern = re.compile('([a-zA-Z1-9-]+)@([a-zA-Z-]+)\.([a-zA-Z]+)')
matches = pattern.finditer(emails)
for match in matches:
    print(match)
    print(match.group(0))
    print(match.group(1))
    print(match.group(2))
    print(match.group(3))
```

```
    <re.Match object; span=(1, 25), match='pythonengineer@gmail.com'>
    pythonengineer@gmail.com
    pythonengineer
    gmail
    com
    <re.Match object; span=(26, 48), match='Python-engineer@gmx.de'>
    Python-engineer@gmx.de
    Python-engineer
    gmx
    de
    <re.Match object; span=(49, 81), match='python-engineer123@my-domain.org'>
    python-engineer123@my-domain.org
    python-engineer123
    my-domain
    org
```

- split(): Split the string into a list, splitting it wherever the RE matches
- sub(): Find all substrings where the RE matches, and replace them with a different string

```
my_string = 'abc123ABCDEF123abc'
pattern = re.compile(r'123') #  no escape for the . here in the set
matches = pattern.split(my_string)
print(matches)

my_string = "hello world, you are the best world"
pattern = re.compile(r'world')
subbed_string = pattern.sub(r'planet', my_string)
print(subbed_string)
```

```
    ['abc', 'ABCDEF', 'abc']
    hello planet, you are the best planet
```

```
urls = """
http://python-engineer.com
https://www.python-engineer.org
http://www.pyeng.net
"""
pattern = re.compile(r'https?://(www\.)?(\w|-)+\.\w+')
pattern = re.compile(r'https?://(www\.)?([a-zA-Z-]+)(\.\w+)')
matches = pattern.finditer(urls)
for match in matches:
    #print(match)
    print(match.group()) # 0
    #print(match.group(1))
    #print(match.group(2))
    print(match.group(3))

# substitute using back references to replace url + domain name
subbed_urls = pattern.sub(r'\2\3', urls)
print(subbed_urls)
```

```
    http://python-engineer.com
    .com
    https://www.python-engineer.org
    .org
    http://www.pyeng.net
    .net

    python-engineer.com
    python-engineer.org
    pyeng.net
```

- ASCII, A : Makes several escapes like \w, \b, \s and \d match only on ASCII characters with the respective property.
- DOTALL, S : Make . match any character, including newlines.
- IGNORECASE, I : Do case-insensitive matches.
- LOCALE, L : Do a locale-aware match.
- MULTILINE, M : Multi-line matching, affecting ^ and $.
- VERBOSE, X (for ‘extended’) : Enable verbose REs, which can be organized more cleanly and understandably.

```
my_string = "Hello World"
pattern = re.compile(r'world', re.IGNORECASE) # No match without I flag
matches = pattern.finditer(my_string)
for match in matches:
    print(match)

my_string = '''
hello
cool
Hello
'''
# line starts with ...
pattern = re.compile(r'^[a-z]', re.MULTILINE) # No match without M flag
matches = pattern.finditer(my_string)
for match in matches:
    print(match)
```

```
    <re.Match object; span=(6, 11), match='World'>
    <re.Match object; span=(1, 2), match='h'>
    <re.Match object; span=(7, 8), match='c'>
```

- https://docs.python.org/3/howto/regex.html
- https://docs.python.org/3/library/re.html
- https://developers.google.com/edu/python/regular-expressions