

DAY 18 REGULAR EXPRESSIONS

Regular Expressions

A regular expression or RegEx is a special text string that helps to find patterns in data. A RegEx can be used to check if some pattern exists in a different data type. To use RegEx in python first we should import the RegEx module which is called *re*.

The *re* Module

After importing the module we can use it to detect or find patterns.

```
import re
```

Methods in *re* Module

To find a pattern we use different set of *re* character sets that allows to search for a match in a string.

- *re.match()*: searches only in the beginning of the first line of the string and returns matched objects if found, else returns None.
- *re.search*: Returns a match object if there is one anywhere in the string, including multiline strings.
- *re.findall*: Returns a list containing all matches
- *re.split*: Takes a string, splits it at the match points, returns a list
- *re.sub*: Replaces one or many matches within a string

Match

```
# syntac
re.match(substring, string, re.I)
# substring is a string or a pattern, string is the text we
look for a pattern , re.I is case ignore
```

```
import re

txt = 'I love to teach python and javaScript'
# It returns an object with span, and match
match = re.match('I love to teach', txt, re.I)
print(match) # <re.Match object; span=(0, 15), match='I love
to teach'>
# We can get the starting and ending position of the match as
tuple using span
span = match.span()
print(span) # (0, 15)
# Lets find the start and stop position from the span
start, end = span
print(start, end) # 0, 15
substring = txt[start:end]
```

```
print(substring)          # I love to teach
```

As you can see from the example above, the pattern we are looking for (or the substring we are looking for) is *I love to teach*. The match function returns an object **only** if the text starts with the pattern.

```
import re

txt = 'I love to teach python and javaScript'
match = re.match('I like to teach', txt, re.I)
print(match)    # None
```

The string does not string with *I like to teach*, therefore there was no match and the match method returned None.

Search

```
# syntax
re.match(substring, string, re.I)
# substring is a pattern, string is the text we look for a
pattern , re.I is case ignore flag
```

```
import re

txt = '''Python is the most beautiful language that a human
being has ever created.
I recommend python for a first programming language'''

# It returns an object with span and match
match = re.search('first', txt, re.I)
print(match)    # <re.Match object; span=(100, 105),
match='first'>
# We can get the starting and ending position of the match as
tuple using span
span = match.span()
print(span)     # (100, 105)
# Lets find the start and stop position from the span
start, end = span
print(start, end)    # 100 105
substring = txt[start:end]
print(substring)     # first
```

As you can see, search is much better than match because it can look for the pattern throughout the text. Search returns a match object with a first match that was found, otherwise it returns *None*. A much better *re* function is *findall*. This function checks for the pattern through the whole string and returns all the matches as a list.

Searching for All Matches Using *findall*

***findall()* returns all the matches as a list**

```
txt = '''Python is the most beautiful language that a human
being has ever created.
I recommend python for a first programming language'''

# It return a list
matches = re.findall('language', txt, re.I)
print(matches)  # ['language', 'language']
```

As you can see, the word *language* was found two times in the string. Let us practice some more. Now we will look for both Python and python words in the string:

```
txt = '''Python is the most beautiful language that a human
being has ever created.
I recommend python for a first programming language'''

# It returns list
matches = re.findall('python', txt, re.I)
print(matches)  # ['Python', 'python']
```

Since we are using *re.I* both lowercase and uppercase letters are included. If we do not have the *re.I* flag, then we will have to write our pattern differently. Let us check it out:

```
txt = '''Python is the most beautiful language that a human
being has ever created.
I recommend python for a first programming language'''

matches = re.findall('Python|python', txt)
print(matches)  # ['Python', 'python']

#
matches = re.findall('[Pp]ython', txt)
print(matches)  # ['Python', 'python']
```

Replacing a Substring

```
txt = '''Python is the most beautiful language that a human
being has ever created.
I recommend python for a first programming language'''

match_replaced = re.sub('Python|python', 'JavaScript', txt,
re.I)
print(match_replaced)  # JavaScript is the most beautiful
language that a human being has ever created.
# OR
match_replaced = re.sub('[Pp]ython', 'JavaScript', txt, re.I)
print(match_replaced)  # JavaScript is the most beautiful
language that a human being has ever created.
```

Let us add one more example. The following string is really hard to read unless we remove the % symbol. Replacing the % with an empty string will clean the text.

```
txt = '''%I a%m te%%a%%che%r% a%n%d %% I l%o%ve te%ach%ing.
The%re is n%o%th%ing as r%ewarding a%s e%duc%at%i%ng a%n%d
e%m%p%ow%er%ing p%e%o%ple.
I fo%und te%a%ching m%ore i%n%t%er%%e%sting t%h%an any other
%jobs.
D%oes thi%s m%ot%iv%a%te %y%o%u to b%e a t%e%a%cher?'''

matches = re.sub('%', '', txt)
print(matches)
```

```
I am teacher and I love teaching.
There is nothing as rewarding as educating and empowering
people.
I found teaching more interesting than any other jobs. Does
this motivate you to be a teacher?
```

Splitting Text Using RegEx Split

```
txt = '''I am teacher and I love teaching.
There is nothing as rewarding as educating and empowering
people.
I found teaching more interesting than any other jobs.
Does this motivate you to be a teacher?'''
print(re.split('\n', txt)) # splitting using \n - end of line
symbol
```

```
['I am teacher and I love teaching.', 'There is nothing as
rewarding as educating and empowering people.', 'I found
teaching more interesting than any other jobs.', 'Does this
motivate you to be a teacher?']
```

Writing RegEx Patterns

To declare a string variable we use a single or double quote. To declare RegEx variable *r*”. The following pattern only identifies apple with lowercase, to make it case insensitive either we should rewrite our pattern or we should add a flag.

```
import re

regex_pattern = r'apple'
txt = 'Apple and banana are fruits. An old cliche says an
apple a day a doctor way has been replaced by a banana a day
keeps the doctor far far away. '
matches = re.findall(regex_pattern, txt)
print(matches) # ['apple']
```

```
# To make case insensitive adding flag '
matches = re.findall(regex_pattern, txt, re.I)
print(matches) # ['Apple', 'apple']
# or we can use a set of characters method
regex_pattern = r'[Aa]pple' # this mean the first letter
could be Apple or apple
matches = re.findall(regex_pattern, txt)
print(matches) # ['Apple', 'apple']
```

- `[]`: A set of characters
 - `[a-c]` means, a or b or c
 - `[a-z]` means, any letter from a to z
 - `[A-Z]` means, any character from A to Z
 - `[0-3]` means, 0 or 1 or 2 or 3
 - `[0-9]` means any number from 0 to 9
 - `[A-Za-z0-9]` any single character, that is a to z, A to Z or 0 to 9
- `\`: uses to escape special characters
 - `\d` means: match where the string contains digits (numbers from 0-9)
 - `\D` means: match where the string does not contain digits
- `.`: any character except new line character(`\n`)
- `^`: starts with
 - `r'^substring'` eg `r'^love'`, a sentence that starts with a word love
 - `r'[^abc]` means not a, not b, not c.
- `$`: ends with
 - `r'substring$'` eg `r'love$'`, sentence that ends with a word love
- `*`: zero or more times
 - `r'[a]*'` means a optional or it can occur many times.
- `+`: one or more times
 - `r'[a]+'` means at least once (or more)
- `?`: zero or one time
 - `r'[a]?'` means zero times or once
- `{3}`: Exactly 3 characters
- `{3,}`: At least 3 characters

- {3,8}: 3 to 8 characters
- |: Either or
 - r'apple|banana' means either apple or a banana
- (): Capture and group

Regular Expression Basics		Regular Expression Character Classes		Regular Expression Flags	
.	Any character except newline	[ab-d]	One character of: a, b, c, d	g	Global Match
a	The character a	[^ab-d]	One character except: a, b, c, d	i	Ignore case
ab	The string ab	[\b]	Backspace character	m	^ and \$ match start and end of line
a b	a or b	\d	One digit	Regular Expression Special Characters	
a*	0 or more a's	\D	One non-digit	\n	Newline
\	Escapes a special character	\s	One whitespace	\r	Carriage return
Regular Expression Quantifiers		\S	One non-whitespace	\t	Tab
*	0 or more	\w	One word character	\0	Null character
+	1 or more	\W	One non-word character	\YYY	Octal character YYY
?	0 or 1	Regular Expression Assertions		\xYY	Hexadecimal character YY
{2}	Exactly 2	^	Start of string	\uYYYY	Hexadecimal character YYYY
{2, 5}	Between 2 and 5	\$	End of string	\cY	Control character Y
{2,}	2 or more	\b	Word boundary	Regular Expression Replacement	
Default is greedy. Append ? for reluctant.		\B	Non-word boundary	\$\$	Inserts \$
Regular Expression Groups		(?=...)	Positive lookahead	\$&	Insert entire match
(...)	Capturing group	(?!...)	Negative lookahead	\$`	Insert preceding string
(?....)	Non-capturing group			\$'	Insert following string
\Y	Match the Y'th captured group			\$Y	Insert Y'th captured group

Let us use examples to clarify the meta characters above

Square Bracket

Let us use square bracket to include lower and upper case

```
regex_pattern = r'[Aa]pple' # this square bracket mean either
A or a
txt = 'Apple and banana are fruits. An old cliché says an
apple a day a doctor way has been replaced by a banana a day
keeps the doctor far far away.'
matches = re.findall(regex_pattern, txt)
print(matches) # ['Apple', 'apple']
```

If we want to look for the banana, we write the pattern as follows:

```
regex_pattern = r'[Aa]pple|[Bb]anana' # this square bracket
means either A or a
txt = 'Apple and banana are fruits. An old cliche says an
apple a day a doctor way has been replaced by a banana a day
keeps the doctor far far away.'
matches = re.findall(regex_pattern, txt)
```

```
print(matches) # ['Apple', 'banana', 'apple', 'banana']
```

Using the square bracket and or operator , we manage to extract Apple, apple, Banana and banana.

Escape character(\) in RegEx

```
regex_pattern = r'\d' # d is a special character which means digits
txt = 'This regular expression example was made on December 6, 2019 and revised on July 8, 2021'
matches = re.findall(regex_pattern, txt)
print(matches) # ['6', '2', '0', '1', '9', '8', '2', '0', '2', '1'], this is not what we want
```

One or more times(+)

```
regex_pattern = r'\d+' # d is a special character which means digits, + mean one or more times
txt = 'This regular expression example was made on December 6, 2019 and revised on July 8, 2021'
matches = re.findall(regex_pattern, txt)
print(matches) # ['6', '2019', '8', '2021'] - now, this is better!
```

Period(.)

```
regex_pattern = r'[a].' # this square bracket means a and . means any character except new line
txt = '''Apple and banana are fruits'''
matches = re.findall(regex_pattern, txt)
print(matches) # ['an', 'an', 'an', 'a ', 'ar']

regex_pattern = r'[a].+' # . any character, + any character one or more times
matches = re.findall(regex_pattern, txt)
print(matches) # ['and banana are fruits']
```

Zero or more times(*)

Zero or many times. The pattern could may not occur or it can occur many times.

```
regex_pattern = r'[a].*' # . any character, * any character zero or more times
txt = '''Apple and banana are fruits'''
```

```
matches = re.findall(regex_pattern, txt)
print(matches)  # ['and banana are fruits']
```

Zero or one time(?)

Zero or one time. The pattern may not occur or it may occur once.

```
txt = '''I am not sure if there is a convention how to write
the word e-mail.
Some people write it as email others may write it as Email or
E-mail.'''
regex_pattern = r'[Ee]-?mail'  # ? means here that '-' is
optional
matches = re.findall(regex_pattern, txt)
print(matches)  # ['e-mail', 'email', 'Email', 'E-mail']
```

Quantifier in RegEx

We can specify the length of the substring we are looking for in a text, using a curly bracket. Let us imagine, we are interested in a substring with a length of 4 characters:

```
txt = 'This regular expression example was made on December 6,
2019 and revised on July 8, 2021'
regex_pattern = r'\d{4}'  # exactly four times
matches = re.findall(regex_pattern, txt)
print(matches)  # ['2019', '2021']
```

```
txt = 'This regular expression example was made on December 6,
2019 and revised on July 8, 2021'
regex_pattern = r'\d{1, 4}'  # 1 to 4
matches = re.findall(regex_pattern, txt)
print(matches)  # ['6', '2019', '8', '2021']
```

Cart ^

- Starts with

```
txt = 'This regular expression example was made on December 6,
2019 and revised on July 8, 2021'
```



```
regex_pattern = r'^This' # ^ means starts with
matches = re.findall(regex_pattern, txt)
print(matches) # ['This']
```

- **Negation**

```
txt = 'This regular expression example was made on December 6,  
2019 and revised on July 8, 2021'  
regex_pattern = r'^[A-Za-z ]+' # ^ in set character means  
negation, not A to Z, not a to z, no space  
matches = re.findall(regex_pattern, txt)  
print(matches) # ['6,', '2019', '8', '2021']
```

Exercises: Day 18

Exercises: Level 1

1. What is the most frequent word in the following paragraph?

```
paragraph = 'I love teaching. If you do not love teaching  
what else can you love. I love Python if you do not love  
something which can give you all the capabilities to develop  
an application what else can you love.'
```

```
[  
    (6, 'love'),  
    (5, 'you'),  
    (3, 'can'),  
    (2, 'what'),  
    (2, 'teaching'),  
    (2, 'not'),  
    (2, 'else'),  
    (2, 'do'),  
    (2, 'I'),  
    (1, 'which'),  
    (1, 'to'),  
    (1, 'the'),  
    (1, 'something'),  
    (1, 'if'),  
    (1, 'give'),  
    (1, 'develop'),  
    (1, 'capabilities'),  
    (1, 'application'),  
    (1, 'an'),  
    (1, 'all'),  
    (1, 'Python'),  
    (1, 'If')  
]
```

2. The position of some particles on the horizontal x-axis are -12, -4, -3 and -1 in the negative direction, 0 at origin, 4 and 8 in the positive direction. Extract these numbers from this whole text and find the distance between the two furthest particles.

```
points = ['-12', '-4', '-3', '-1', '0', '4', '8']  
sorted_points = [-12, -4, -3, -1, -1, 0, 2, 4, 8]  
distance = 8 - (-12) # 20
```

Exercises: Level 2

1. Write a pattern which identifies if a string is a valid python variable

```
is_valid_variable('first_name') # True
is_valid_variable('first-name') # False
is_valid_variable('1first_name') # False
is_valid_variable('firstname') # True
```

Exercises: Level 3

1. Clean the following text. After cleaning, count three most frequent words in the string.

```
sentence = '''%I $am@% a %tea@cher%, &and& I lo%#ve
%tea@ching%;. There $is nothing; &as& mo@re rewarding as
educa@ting &and& @emp%o@wering peo@ple. ;I found tea@ching
m%o@re interesting tha@n any other %jo@bs. %Do@es thi%s
mo@tivate yo@u to be a tea@cher!?'''

print(clean_text(sentence));
I am a teacher and I love teaching There is nothing as more
rewarding as educating and empowering people I found teaching
more interesting than any other jobs Does this motivate you to
be a teacher

print(most_frequent_words(cleaned_text)) # [(3, 'I'), (2,
'teaching'), (2, 'teacher')]
```

🎉 CONGRATULATIONS! 🎉