https://www.hackerrank.com/challenges/detecting-valid-latitude-and-longitude/problem?h_r=next-challenge&h_v=zen&h_r=next-challenge&h_v=zen&h_r=next-challenge&h_v=zen&h_r=next-challenge&h_v=zen
https://regexone.com/problem/matching_filenames?
Regex Cheat sheet: https://www.rexegg.com/regex-quickstart.html
r: This is used to indicate raw string in regular expression meaning raw string is not processed
(the){1, } :{ braces indicate range
eg. "[\w|\W|\d]{3}" - This indicates 3 characters either like ab3
e.g. wa[z]{2,5}up" - wazzup, wazzzup, wazzzzup, wazzzzzup
[ab]: braces indicate or, here we will get a or b for pattern. Square braces are used for grouping.

***(Kleene Star)** : Accepts preceding character or group of character zero or more times .E.g
"a*bcd" : bcd, abcd, aabcd
"[abc]*b": abbcabcb, b
Note: [abc]* - indicates one or more of any a's b's or c's.
+(Kleene Plus): accepts preceding character or group of character one or more times.
E.g.
a+bcd: abcd, aabcd
[abc]+12: abbc12, abcabc12
?(Optionality Character) : indicates 0 or 1 time occurence of the preceding character
E.g. "ab?c" will match ac or abc
E.g \d+ files? found\?
(ab/cd) : This is the grouping operator E.g. ab, cd
The (Book)$: All strings ending with 'Book'
[a-z A-Z 0-9]{5,10} : Any string of length between 5 to 10
The Train : [a-z]\s[A-Z]
. : The dot character matches anything as a single character except a newline.
e.g. "...\....\...." will match 123.abc.ab4
^(outside square braces): Starting with the preceding group of characters
e.g. ^123 is similar to 1235123
e.g ^success, this will match with success|successful but not unsuccessful.
^(inside square braces): Does not start with
e.g [^dfp]an - three letter word ending with an not starting with d, f or p
Note: You don't write [^(dfp)]
Combination of ^(hat) and $(dollar): Indicates find this pattern.
e.g' ^success$': This will match only with success.

$,\z : Both used to denote ending.
\s: This is used to indicate white space characters
Most common whitespaces used are as follows:

tab: \t
newline: \n
carriage return: \r
space: ‘ ‘
\s is used to depict any of the above given whitespace characters.
e.g. [\s] - single whitespace
Note: We can use kleene plus and kleene star even with whitespaces.

\S: This is used to indicate non-whitespace characters space
e.g [\S]{2} - two non-whitespace characters.

\d: This indicates digit, alternatively written as [0-9]

\D: This indicates group of characters that are not string, alternatively written as [^0-9]

\w: This indicates word characters that are a-z, A-Z, 0-9 and underscore [a-z|A-Z|0-9| _].

\W: This indicates all the characters except those indicated by \w or [^a-z|A-Z|0-9| _].

| : OR is represented using the pipe symbol |
e.g([cb] ats | [hd] ogs?)

Note: Pipe symbol works inside parentheses.

Note: You can even use pipe inside parentheses E.g. `([1][0-7]|[0-9])`

[ ]: This is used to define a class e.g “[hd]og” - indicates hog or dog
hyphen in square bracket “[0-6]” : any one digit between 0 to 6(both inclusive)

( ): This is used to define a group, like for example (a-z) here means “a hyphen z”.
Nested group of parenthesis are read from left to right
e.g (\w{3}\s(\d{4})) will capture both Jan 1987 and 1987 in Jan 1987

Note: We can use *,+,?,^,$, {m,n} can be used with parentheses grouping
e.g (abc){3} will search for pattern abcabcabc

[a-z] : Here it implies the lowercase letter group

re.findall(pattern, string): Returns list of all the pattern in a string

re.split(pattern, string): We will get a list of strings split on the pattern.

re.match(pattern, string):

re.search(pattern, string): Finds the occurrence of a pattern in a string, it returns sre_match object. It returns a span value.

Difference between search and match in regex:

match searches the string till it could not match

search searches from the beginning of the string

nltk: natural language toolkit is a library in python used for tokenization.

Tokenization: Breaking down a document into small chunks called tokens

word_tokenize: This breaks the document into words.

sent_tokenize: This breaks the document into sentences.

regex_tokenize: This breaks the document into specific regex. This gives us the freedom to apply micro specification.

Plotting NLP charts with matplotlib:

[wxq]{5}: This will match character w, x, q five times
Tokenizing hashtag tweets.

```
from nltk.tokenizer import regexp_tokenize, TweetTokenizer
pattern1 = r”#\w+”
hashtags = regexp_tokenize(tweets[0],pattern1)
print(hashtags)
```

A pattern that matches both mentions and hashtags.

```
pattern2 = r”[@|#]\w+”
```

Using the TweetTokenizer method:

```
tknr = TweetTokenizer()
all_tokens = [tknr.tokenize(t) for t in tweets]
print(all_tokens)
```

```
all_words = word_tokenize(german_text)
print(all_words)
```

```
# Tokenize and print only capital words
capital_words = r"[A-Z]\w+"
print(regexp_tokenize(german_text, capital_words))
```

```
# Tokenize and print only emoji
```

```
emoji =
"['\U0001F300-\U0001F5FF'|'\U0001F600-\U0001F64F'|'\U0001F680-\U0001F6FF'|'\u2600-\u26
FF\u2700-\u27BF']"
print(regexp_tokenize(german_text, emoji))
```

pattern1 = **r"\[.*\]"**
pattern1 is to determine any thing inside a square bracket

pattern2 = **r"\w\s"+**
pattern2 determines any pattern of the form characters:

match_digits_and_words = ('\d|\w')

Charting Practice

```
# Split the script into lines: lines
lines = holy_grail.split('\n')

# Replace all script lines for speaker
pattern = "[A-Z]{2,}(\s)?(#\d)?([A-Z]{2,})?:"
lines = [re.sub(pattern, '', l) for l in lines]

# Tokenize each line: tokenized_lines
tokenized_lines = [regexp_tokenize(s, "\w+") for s in lines]

# Make a frequency list of lengths: line_num_words
line_num_words = [len(t_line) for t_line in tokenized_lines]

# Plot a histogram of the line lengths
plt.hist(line_num_words)

# Show the plot
plt.show()
```

```
import re
ctp = re.compile('[C]\d{6}')
ctp = re.compile('[C]+\d{6}')
match = ctp.match('C123456')
```

```
print(bool(match))
#Digit with a dollar sign
ctp1 = re.compile('\$\d*')
print(bool(ctp1.match('$123')))
print(bool(ctp1.match('$45')))

#Pattern for decimal
ctp2 = re.compile('\$\d*.\d*')
print(bool(ctp2.match('$5.66')))

#We can directly call the match function, we will have to specify an argument pattern
print(bool(re.match(pattern = '\d{3}-\d{3}-\d{3}-\d{3}', string = '123-432-543-533')))

print(bool(re.match(pattern = '[A-Z]\w{6}', string = 'Zimbave')))
ctp3 = re.compile('\$\D*')
bool(ctp3.match('$%^&'))
#re is case senstive

#Match with whitespace character(s)
ctp4 = re.compile('\s')
bool(ctp4.match(' d'))

#Match with non-whitespace character(S)
ctp5 = re.compile('\S')
bool(ctp.match('ana '))

#Match with alpha-numeric character(w)
ctp6 = re.compile('\$+\w*')
print(bool(ctp6.match('$ananya1105')))

ctp7 = re.compile('\w*')
print(bool(ctp7.match('ananya1105')))

#Matching with non-alphanumeric character
ctp8 = re.compile('\W')
print(bool(ctp8.match('&*($%($))')))

#Difference between \d+ and \d*, \d
ctp9 = re.compile('\d')
ctp10 = re.compile('\d+')
ctp11 = re.compile('\d*')
print(bool(ctp9.match('123343')))
print(ctp9.findall('123 rer3 038p '))
```

```
print(bool(ctp10.match('123343')))
print(ctp10.findall('123 rer3 038p'))
print(bool(ctp11.match('123456')))
print(ctp11.findall('123 rer3 038p'))

#* means the preceding character is coming n number of times where n can be 0
#+ means the preceding character should atleast come once
#There is difference when we apply findall
ctp12 = re.compile('ab+')
print(bool(ctp12.match('ab')))
print(ctp12.findall('abbbabsdd,nddabb saaaa ffaabb ab b'))

ctp13 = re.compile('ab*')
print(bool(ctp13.match('ab')))
print(ctp13.findall('abbbabsdd,nddabb afaaabb asb abb ab b'))

#Use of \ character
ctp13 = re.compile('d*')
print(bool(ctp13.match('dfdfd')))
print(ctp13.findall('dffd djfldddd eredd'))
#Matching the beginnning
ctp14 = re.compile('')
#Representing a character class

#Match any character except newling
ctp15 = re.compile('\.')
print(bool(ctp15.match('\nabcdef\nefalj')))
ctp15.findall('abdcdlj adfljsdlsj')

#Indicate occurence of re using {}
ctp16 = re.compile('\d{5}')
print(bool(ctp16.match('12356')))
print(bool(ctp16.match('12356.78')))
```