```
#A RegEx is a powerful tool for matching text, based on a pre-defined
pattern.
#It can detect the presence or absence of a text by matching it with a
particular pattern,
#and also can split a pattern into one or more sub-patterns.
#The Python standard library provides a re module for regular
expressions. Its primary function is to offer a search,
#where it takes a regular expression and a string. Here, it either
returns the first match or else none.

#\w – matches a word character
#\d – matches digit character
#\s – matches whitespace character (space, tab, newline, etc.)
#\b – matches a zero-length character

#\A Returns a match if the specified characters are at the beginning
of the string

import re

txt = "The rain in Spain"

#Check if the string starts with "The":

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

print(x)

if x:
  print("Yes, there is a match!")
else:
  print("No match")
```

```
['The']
Yes, there is a match!
```

```
match = re.search(r'portal', 'A computer science \ portal for
Education')
print(match)
print(match.group())

print('Start Index:', match.start())
print('End Index:', match.end())
```

```
<re.Match object; span=(21, 27), match='portal'>
portal
Start Index: 21
End Index: 27
```

```
#MetaCharacters withDescription
#\   Used to drop the special meaning of character following it
```

```
#[]  Represent a character class
#^   Matches the beginning
#$   Matches the end
#.   Matches any character except newline
#|   Means OR (Matches with any of the characters separated by it.
#?   Matches zero or one occurrence
#*   Any number of occurrences (including 0 occurrences)
#+   One or more occurrences
#{}  Indicate the number of occurrences of a preceding RegEx to match.
#()  Enclose a group of RegEx

#Data Mining: Regular expression is the best tool for data mining.
#It efficiently identifies a text in a heap of text by checking with a
pre-defined pattern.
#Some common scenarios are identifying an email, URL, or phone from a
pile of text.

#Data Validation: Regular expression can perfectly validate data.
#It can include a wide array of validation processes by defining
different sets of patterns.
#A few examples are validating phone numbers, emails, etc.

# basic regular expressions. They are as follows:

#Character Classes
#Rangers
#Negation
#Shortcuts
#Beginning and End of String
#Any Character

# Character Classes Character classes allow you to match a single set
of characters with a possible set of characters.
# character class is mentioned within the square brackets.An example
of case sensitive words.

print(re.findall(r'[Ee]ducation', 'Education of education: \ A
computer science portal for education'))
```

```
['Education', 'education', 'education']
```

```
#Ranges
#The range provides the flexibility to match a text with the help of a
range pattern such as a range of numbers(0 to 9),
#a range of characters (A to Z), and so on.
#The hyphen character within the character class represents a range.

print('Range',re.search(r'[a-zA-Z]', 'x'))
```

```
Range <re.Match object; span=(0, 1), match='x'>
```

```
x = range(3, 6)
for n in x:
  print(n)
```

```
3
4
5
```

```
x = range(3, 20, 2)
for n in x:
  print(n)
```

```
3
5
7
9
11
13
15
17
19
```

```
#Negation
#Negation inverts a character class.
#It will look for a match except for the inverted character or range
of inverted characters
#mentioned in the character class.

print(re.search(r'[^a-z]', 'c'))
```

```
None
```

```
print(re.search(r'C[^l]', 'Class'))
```

```
None
```

```
#Beginning and End of String
#The ^ character chooses the beginning of a string and the $ character
chooses the end of a string.


# Beginning of String
match = re.search(r'^is', 'This is the month')
print('Beg. of String:', match)

match = re.search(r'^is', 'is the month')
print('Beg. of String:', match)

# End of String
match = re.search(r'education$', 'Compute science portal for
education')
print('End of String:', match)
```

```
Beg. of String: None
Beg. of String: <re.Match object; span=(0, 2), match='is'>
End of String: <re.Match object; span=(27, 36), match='education'>

#Any Character
#The . character represents any single character outside a bracketed
character class.

print('Any Character', re.search(r'p.th.n', 'python 3'))

Any Character <re.Match object; span=(0, 6), match='python'>

#Some of the other regular expressions are as follows:

#Optional Characters
#Repetition
#Shorthand
#Grouping
#Lookahead
#Substitution

#Optional Characters
#Regular expression engine allows you to specify optional characters
using the ? character.
#It allows a character or character class either to present once or
else not to occur.
# example of a word with an alternative spelling – color or colour.#

print('Color',re.search(r'colou?r', 'color'))
print('Colour',re.search(r'colou?r', 'colour'))

Color <re.Match object; span=(0, 5), match='color'>
Colour <re.Match object; span=(0, 6), match='colour'>

#Repetition
#Repetition enables you to repeat the same character or character
class.
#Consider an example of a date that consists of day, month, and year.
#regular expression to identify the date (mm-dd-yyyy).

print('Date{mm-dd-yyyy}:', re.search(r'[\d]{2}-[\d]{2}-[\d]{4}','13-
07-2023'))

Date{mm-dd-yyyy}: <re.Match object; span=(0, 10), match='13-07-2023'>

#Repetition ranges
#The repetition range is useful when you have to accept one or more
formats.
#Consider a scenario where both three digits, as well as four digits,
are accepted.
```

```
print('Three Digit:', re.search(r'[\d]{3,4}', '189'))
print('Four Digit:', re.search(r'[\d]{3,4}', '2145'))
```

```
Three Digit: <re.Match object; span=(0, 3), match='189'>
Four Digit: <re.Match object; span=(0, 4), match='2145'>
```

```
#Open-Ended Ranges
#There are scenarios where there is no limit for a character
repetition.
#In such scenarios, you can set the upper limit as infinitive.
#A common example is matching street addresses.
```

```
print(re.search(r'[\d]{1,}','5th Floor, B-218,\
Sector-136, Noida, Uttar Pradesh - 201405'))
```

```
<re.Match object; span=(0, 1), match='5'>
```

```
#Shorthand
#Shorthand characters allow you to use + character to specify one or
more ({1,})
#and * character to specify zero or more ({0,}.
```

```
print(re.search(r'[\d]+','5th Floor, B-218,\
Sector-136, Noida, Uttar Pradesh - 201405'))
```

```
<re.Match object; span=(0, 1), match='5'>
```

```
#Grouping
#Grouping is the process of separating an expression into groups by
using parentheses,
#and it allows you to fetch each individual matching group.
```

```
grp = re.search(r'([\d]{2})-([\d]{2})-([\d]{4})', '12-07-2023')
print(grp)
```

```
<re.Match object; span=(0, 10), match='12-07-2023'>
```

```
#Return a tuple of matched groups
#You can use groups() method to return a tuple that holds individual
matched groups
```

```
grp = re.search(r'([\d]{2})-([\d]{2})-([\d]{4})','14-07-2023')
print(grp.groups())
```

```
('14', '07', '2023')
```

```
#Retrieve a single group
#Upon passing the index to a group method, you can retrieve just a
single group.
```

```
grp = re.search(r'([\d]{2})-([\d]{2})-([\d]{4})','14-07-2023')
print(grp.group(3))
```

```
2023
```

```
grp = re.search(r'(?P<dd>[\d]{2})-(?P<mm>[\d]{2})-(?P<yyyy>[\d]
{4})','14-07-2023')
print(grp.group('dd'))
```

```
14
```

```
grp = re.search(r'(?P<dd>[\d]{2})-(?P<mm>[\d]{2})-(?P<yyyy>[\d]
{4})','14-07-2023')
print(grp.groupdict())
```

```
{'dd': '14', 'mm': '07', 'yyyy': '2023'}
```

```
#Lookahead
#In the case of a  negated character class,
#it won’t match if a character is not present to check against the
negated character.
#We can overcome this case by using lookahead; it accepts or rejects a
match based on the presence or absence of content.
```

```
print('negation:', re.search(r'n[^e]', 'Python'))
print('lookahead:', re.search(r'n(?!e)', 'Python'))
```

```
negation: None
lookahead: <re.Match object; span=(5, 6), match='n'>
```

```
#Lookahead can also disqualify the match if it is not followed by a
particular character.
#This process is called a positive lookahead, and can be achieved by
simply replacing ! character with = character.
```

```
print('positive lookahead', re.search(r'n(?=e)', 'jasmine'))
```

```
positive lookahead <re.Match object; span=(5, 6), match='n'>
```

```
#Substitution
#The regular expression can replace the string and returns the
replaced one using the re.sub method. It is useful when you want to
avoid characters such as /, -, ., etc. before storing it to a
database. It takes three arguments:

#the regular expression
#the replacement string
#the source string being searched
```

```
print(re.sub(r'([\d]{4})-([\d]{4})-([\d]{4})-([\d]{4})',r'\1\2\3\4',
             '1111-2222-3333-4444'))
```

```
1111222233334444
```

```
#Compiled RegEx
#The Python regular expression engine can return a compiled regular
expression(RegEx) object using compile function.
#This object has its search method and sub-method, where a developer
can reuse it when in need.

regex = re.compile(r'([\d]{2})-([\d]{2})-([\d]{4})')

# search method
print('compiled reg expr', regex.search('13-07-2023'))

# sub method
print(regex.sub(r'\1.\2.\3', '13-07-2023'))
```

```
compiled reg expr <re.Match object; span=(0, 10), match='13-07-2023'>
13.07.2023
```