+
Causes the resulting RE to match 1 or more repetitions of the preceding RE. ab+ will match ‘a’ followed by any non-zero number of ‘b’s; it will not match just ‘a’.

```
>>> str1="a ab abb acb"
>>> import re
>>> list1=re.findall(r'ab+',str1)
>>> print(list1)
['ab', 'abb']
>>> list2=re.findall(r'ab*',str1)
>>> print(list2)
['a', 'ab', 'abb', 'a']
```

?
Causes the resulting RE to match 0 or 1 repetitions of the preceding RE. ab? will match either ‘a’ or ‘ab’.

```
>>> str2="a ab a acb"
>>> list1=re.findall(r'ab?',str2)
>>> print(list1)
['a', 'ab', 'a', 'a']
```

{m}
Specifies that exactly *m* copies of the previous RE should be matched; fewer matches cause the entire RE not to match. For example, a{6} will match exactly six 'a' characters, but not five.

```
>>> str1="ab abb abbb abbbb a abbbbc"
>>> list1=re.findall(r'ab{4}',str1)
>>> print(list1)
['abbbb', 'abbbb']
```

{m,n}
Causes the resulting RE to match from *m* to *n* repetitions of the preceding RE, attempting to match as many repetitions as possible. For example, a{3,5} will match from 3 to 5 'a' characters. Omitting *m* specifies a lower bound of zero, and omitting *n* specifies an infinite upper bound. As an example, a{4,}b will match 'aaaab' or a thousand 'a' characters followed by a 'b', but not 'aaab'

```
>>> str1="ab abb abbb abbbb abbbbb abbbbbbb"
>>> list1=re.findall(r'ab{4,}',str1)
>>> print(list1)
['abbbb', 'abbbbb', 'abbbbbbb']
```

[]
Used to indicate a set of characters. In a set:

1. Characters can be listed individually, e.g. [amk] will match 'a', 'm', or 'k'.

```
import re

namesList=["naresh","ramesh","suresh","kishore","raman","rajesh","kiran"]
for name in namesList:
    m=re.search(r'^[rk].*',name)
    if m!=None:
        print(name)
```

Output:
ramesh
kishore
raman
rajesh
kiran

Example:

```
import re

namesList=["naresh","ramesh","suresh","kishore","raman","rajesh","kiran"]
for name in namesList:
    m=re.search(r'^[rk].*[hn]$',name)
    if m!=None:
        print(name)
```

Output:
ramesh
raman
rajesh

kiran

2. Ranges of characters can be indicated by giving two characters and separating them by a '-', for example [a-z] will match any lowercase ASCII letter, [0-5][0-9] will match all the two-digits numbers from 00 to 59, and [0-9A-Fa-f] will match any hexadecimal digit.

Example:

```
# name validation

import re

name=input("Enter Name ")
m=re.fullmatch(r'[a-z]+',name)
if m!=None:
    print("valid name")
else:
    print("invalid name")
```

Output

```
Enter Name naresh
valid name

Enter Name nit123
invalid name

Enter Name ABC
invalid name
```

```
# Extract date from string

import re
str1="date of joining 09-11-2023"
m=re.search(r'([0-9]{2})-([0-9]{2})-([0-9]{4})',str1)
print(m)
print(m.group(0))
print(m.group(1))
print(m.group(2))
```

```
print(m.group(3))
```

Output:

```
<re.Match object; span=(16, 26), match='09-11-2023'>
09-11-2023
09
11
2023
```

Special Sequence Characters

\A
Matches only at the start of the string.

```
>>> m=re.search(r'\A[abc]',"cat")
>>> print(m)
<re.Match object; span=(0, 1), match='c'>
```

\b
Matches the empty string, but only at the beginning or end of a word. A word is defined as a sequence of word characters

Example:

```
>>> str1="ab abc abb"
>>> list1=re.findall(r'ab',str1)
>>> print(list1)
['ab', 'ab', 'ab']
>>> list2=re.findall(r'ab\b',str1)
>>> print(list2)
['ab']
>>> str2="ab. ab! ab abb acb"
>>> list3=re.findall(r'\bab\b',str2)
>>> print(list3)
['ab', 'ab', 'ab']
```

\B
Matches the empty string, but only when it is *not* at the beginning or end of a word. This means that r'py\B' matches 'python', 'py3', 'py2', but not 'py', 'py.', or 'py!'.

```
>>> str2="ab. ab! ab abb acb"
>>> list3=re.findall(r'ab\B',str2)
>>> print(list3)
['ab']
```

\d
Matches any decimal digit; this is equivalent to [0-9].

```
# Mobile Number Validation

import re

mobile=input("Enter Mobile No")
m=re.fullmatch(r'[98]\d{8}',mobile)
if m!=None:
    print(f'{mobile} is valid')
else:
    print(f'{mobile} is invalid')
```

Output:

```
Enter Mobile No712345678
712345678 is invalid

Enter Mobile No812345678
812345678 is valid

Enter Mobile No8123456789
8123456789 is invalid
```

\D
Matches any character which is not a decimal digit.

Example:

```
>>> str1="python 3.12"
>>> list1=re.findall(r'\d',str1)
>>> print(list1)
['3', '1', '2']
>>> list2=re.findall(r'\D',str1)
>>> print(list2)
```

\s
Matches characters considered whitespace in the ASCII character set; this is equivalent to [\t\n\r\f\v].

```
>>> str1="python java\noracle\tmysql"
>>> list1=re.split(r'\s',str1)
>>> print(list1)
['python', 'java', 'oracle', 'mysql']
```

\S
Matches any character which is not a whitespace character. This is the opposite of \s

```
>>> str1="python java\noracle\tmysql"
>>> list1=re.findall(r'\S',str1)
>>> print(list1)
['p', 'y', 't', 'h', 'o', 'n', 'j', 'a', 'v', 'a', 'o', 'r', 'a', 'c', 'l', 'e', 'm', 'y', 's', 'q', 'l']
>>> list2=re.findall(r'\s',str1)
>>> print(list2)
[' ', '\n', '\t']
```

\w
Matches characters considered alphanumeric in the ASCII character set; this is equivalent to [a-zA-Z0-9_].

```
# user name validation
import re
user=input("UserName ")
m=re.fullmatch(r'^[a-zA-Z]\w{4,12}',user)
if m!=None:
    print(f'{user} is valid')
else:
    print(f'{user} is invalid')
```

Output:
UserName naresh
naresh is valid

UserName nit123

nit123 is valid

UserName 123
123 is invalid

UserName n12
n12 is invalid

UserName a_bcd
a_bcd is valid

Example:

```
# email validation
# xxxx@yyy.zzz

import re
email=input("Enter Email Id ")
m=re.fullmatch(r'\w+@\w+\.\w{2,3}',email)
if m!=None:
    print(f'{email} is valid')
else:
    print(f'{email} is invalid')
```

Output:

Enter Email Id naresh@nareshit.com
naresh@nareshit.com is valid

Enter Email Id nareshit.com
nareshit.com is invalid

Enter Email Id naresh
naresh is invalid

\W

Matches any character which is not a word character.

```
>>> str1="abc$xyz#@&abc"
>>> list1=re.findall(r'\W',str1)
>>> print(list1)
```

\Z
Matches only at the end of the string.

```
>>> str1="python"
>>> m=re.search(r'n\Z',str1)
>>> print(m)
<re.Match object; span=(5, 6), match='n'>
>>> str2="naresh"
>>> m=re.search(r'n\Z',str2)
>>> print(m)
None
```

Python Database Communication (PDBC)

Every application or project required to store or save data permanently. This is done using two systems.

1. File System
2. Database System

Limitations of file system

1. Files are not secured
2. Files cannot hold large amount of data
3. File System does not provide any Query Language

Database Applications or software's

1. Oracle
2. MySQL
3. SQLServer
4. DB2
5. MS-Excel
6. PostgreSQL