*****

Causes the resulting RE to match 0 or more repetitions of the preceding RE, as many repetitions as are possible. ab* will match ‘a’, ‘ab’, or ‘a’ followed by any number of ‘b’s.

```
import re

str1="ab abb abbb a acb"
list1=re.findall(r'ab*',str1)
print(list1)
```

Output

```
['ab', 'abb', 'abbb', 'a', 'a']
```

+

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’.

```
import re

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

Output

```
['ab', 'abb', 'abbb', 'a', 'a']
['ab', 'abb', 'abbb']
```

?

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

```
import re

str1="ab abb abbb a acb"
list1=re.findall(r'ab*',str1)
list2=re.findall(r'ab+',str1)
list3=re.findall(r'ab?',str1)
print(list1)
print(list2)
print(list3)
```

{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.

```
import re

str1="ab abb abbb a acb"
list1=re.findall(r'ab*',str1)
list2=re.findall(r'ab+',str1)
list3=re.findall(r'ab?',str1)
list4=re.findall(r'ab{3}',str1)
print(list1)
print(list2)
print(list3)
print(list4)
```

Output

```
['ab', 'abb', 'abbb', 'a', 'a']
```

['ab', 'abb', 'abbb']
['ab', 'ab', 'ab', 'a', 'a']
['abbb']

{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.

```
import re

str1="ab abb abbb a acb"
list1=re.findall(r'ab*',str1)
list2=re.findall(r'ab+',str1)
list3=re.findall(r'ab?',str1)
list4=re.findall(r'ab{3}',str1)
list5=re.findall(r'ab{1,3}',str1)
print(list1)
print(list2)
print(list3)
print(list4)
print(list5)
```

Output
['ab', 'abb', 'abbb', 'a', 'a']
['ab', 'abb', 'abbb']
['ab', 'ab', 'ab', 'a', 'a']
['abbb']
['ab', 'abb', 'abbb']

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

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

Example:

```
import re
names=["naresh","suresh","kishore","ramesh","kiran","suman"]

for name in names:
    m=re.search(r'^[sk].*',name)
    if m!=None:
        print(name)
```

Output

```
suresh
kishore
kiran
suman
```

Example

```
import re
names=["naresh","suresh","kishore","ramesh","kiran","suman"]

for name in names:
    m=re.fullmatch(r'^[sk].*[hn]$',name)
    if m!=None:
        print(name)
```

Output
suresh
kiran
suman

- 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.

```
import re

name=input("Enter Name in Capital letters ")
m=re.fullmatch(r'[A-Z]+',name)
if m!=None:
   print(f'{name} is valid')
else:
   print(f'{name} is invalid')
```

Output
Enter Name in Capital letters NARESH
NARESH is valid

Enter Name in Capital letters N123
N123 is invalid

Example:

```
import re

str1="Date of joining 12-03-2024"
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='12-03-2024'>
12-03-2024
12
03
2024
```

Example:

```
# email validation
# xyz@abc.com

import re

email=input("Enter Email ID")
m=re.fullmatch(r'[a-z0-9]+@[a-z]+\.[a-z]{2,}',email)
if m!=None:
    print(f'{email} is valid')
else:
    print(f'{email} is invalid')
```

Output

```
Enter Email IDn123@nareshit.com
n123@nareshit.com is valid

Enter Email IDnaresh.com
naresh.com is invalid
```

Enter Email IDnaresh@com
naresh@com is invalid

Special Sequence of characters

\A
Matches only at the start of the string.
\Z
Matches only at the end of the string.

Example:

```
import re


str1="Amount is 400$"

m=re.search(r'[0-9]+\$\Z',str1)
print(m)
print(m.group(0))
```

Output

```
<re.Match object; span=(10, 14), match='400$'>
400$
```

Example:

```
import re

str1="^python"
m=re.search(r'\A\^',str1)
print(m)
```

Output

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

\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:

```
import re

str1="py python rpython ironpython py cpython py. py! py"

list1=re.findall(r'py\b',str1)
print(list1)
```

Output

['py', 'py', 'py', 'py', 'py']

\B

Matches the empty string, but only when it is *not* at the beginning or end of a word. This means
that r'at\B' matches 'athens', 'atom', 'attorney', but not 'at', 'at.',
or 'at!'. \B is the opposite of \b.

```
import re

str1="athens atom attorney at at. at!"
list1=re.findall(r'at\B',str1)
print(list1)
```

Output

['at', 'at', 'at']

\d
Matches any decimal digit in the ASCII character set; this is equivalent to `[0-9]`.

```
import re

mobileno=input("MobileNo ")
m=re.fullmatch(r'^[89]\d{9}',mobileno)
if m!=None:
    print(f'{mobileno} is valid')
else:
    print(f'{mobileno} is invalid')
```

Output
MobileNo 9999912345
9999912345 is valid

MobileNo 7999812345
7999812345 is invalid

\D
Matches any character which is not a decimal digit. This is the opposite of \d.