re.match(pattern, string, flags=0)
If zero or more characters at the beginning of string match the regular expression pattern, return a corresponding Match. Return None if the string does not match the pattern;

```
>>> import re
>>> str1="python"
>>> m=re.match(r'py',str1)
>>> print(m)
<re.Match object; span=(0, 2), match='py'>
>>> m=re.match(r'jy',str1)
>>> print(m)
None
>>> str2="java python"
>>> m=re.match(r'py',str2)
>>> print(m)
None
```

re.search(pattern, string, flags=0)
Scan through string looking for the first location where the regular expression pattern produces a match, and return a corresponding Match. Return None if no position in the string matches the pattern;

Example:

```
import re
str1="java python"
m=re.search(r'py',str1)
print(m)
m=re.search(r'PY',str1,re.IGNORECASE)
print(m)
str2="java python java python"
m=re.search(r'py',str2)
print(m)
m=re.search(r'jy',str2)
print(m)
```

Output

```
<re.Match object; span=(5, 7), match='py'>
<re.Match object; span=(5, 7), match='py'>
<re.Match object; span=(5, 7), match='py'>
```

None

re.findall(pattern, string, flags=0)
Return all non-overlapping matches of pattern in string, as a list of strings or tuples. The string is scanned left-to-right, and matches are returned in the order found

```
import re

str1="python java oracle java mysql java"
A=re.findall(r'java',str1)
print(A)
B=re.findall(r'mysql',str1)
print(B)
C=re.findall(r'html',str1)
print(C)
```

Output
['java', 'java', 'java']
['mysql']
[]

re.fullmatch(pattern, string, flags=0)
If the whole string matches the regular expression pattern, return a corresponding Match. Return None if the string does not match the pattern;

```
import re
str1="python"
m=re.fullmatch(r'python',str1)
print(m)
m=re.fullmatch(r'jython',str1)
print(m)
```
Output
<re.Match object; span=(0, 6), match='python'>
None

Pattern is described using special syntax or characters; these are classified into different categories

1. Character Literals
2. Meta Characters

3. Character classes
4. Quantifiers

Character Literals

Character literal is nothing but character value (a-z,A-Z,0-9 and special characters)

```
import re

str1="python programming language"
A=re.findall(r'p',str1)
print(A)
B=re.findall(r'a',str1)
print(B)
```

Output

```
['p', 'p']
['a', 'a', 'a']
```

Special Characters

.

(Dot.) In the default mode, this matches any character except a newline. If the DOTALL flag has been specified, this matches any character including a newline.

Example:

```
import re

str1="python programming language"
A=re.findall(r'p',str1)
print(A)
B=re.findall(r'a',str1)
print(B)
C=re.findall(r'.',str1)
print(C)
D=re.findall(r'p.',str1)
print(D)
E=re.findall(r'.a.',str1)
print(E)
```

Output

```
['p', 'p']
['a', 'a', 'a']
['p', 'y', 't', 'h', 'o', 'n', ' ', 'p', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'i', 'n', 'g', ' ', 'l', 'a', 'n', 'g', 'u', 'a', 'g', 'e']
['py', 'pr']
['ram', 'lan', 'uag']
```

^

(Caret.) Matches the start of the string, and in MULTILINE mode also matches immediately after each newline.

```
import re

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

Output

```
ramesh
raman
rajesh
```

Example:

```
import re

str1='''python is a  lagnuage
python is high level
python is object oriented
python is multiparadigm'''

A=re.findall(r'^python',str1,re.MULTILINE)
print(A)
```

Output

```
['python', 'python', 'python', 'python']
```

$
Matches the end of the string or just before the newline at the end of the string, and in MULTILINE mode also matches before a newline

Example
```
import re

names=["ramesh","naresh","kiran","raman","rajesh","kishore"]
for name in names:
    m=re.search(r'sh$',name)
    if m!=None:
        print(name)
```

Output
```
ramesh
naresh
rajesh
```

*
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="a ab abb abbb abc abbbb"
A=re.findall(r'ab*',str1)
print(A)
```

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

+
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="a ab abb abbb abc abbbb"
```

```
A=re.findall(r'ab*',str1)
print(A)
B=re.findall(r'ab+',str1)
print(B)
```

Output

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

?

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

Example:

```
import re

str1="a ab abb abbb abc abbbb"
A=re.findall(r'ab*',str1)
print(A)
B=re.findall(r'ab+',str1)
print(B)
C=re.findall(r'ab?',str1)
print(C)
```

Output

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

{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="a ab abb abbb abc abbbb"
A=re.findall(r'ab*',str1)
```

```
print(A)
B=re.findall(r'ab+',str1)
print(B)
C=re.findall(r'ab?',str1)
print(C)
D=re.findall(r'ab{4}',str1)
print(D)
```

Output

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

```
import re

str1="a ab abb abbb abc abbbb"
A=re.findall(r'ab*',str1)
print(A)
B=re.findall(r'ab+',str1)
print(B)
C=re.findall(r'ab?',str1)
print(C)
D=re.findall(r'ab{4}',str1)
print(D)
E=re.findall(r'ab{2,4}',str1)
print(E)
```

Output

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

[]
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=["ramesh","naresh","kiran","raman","rajesh","kishore"]
for name in names:
    m=re.search(r'^[nk]',name)
    if m!=None:
        print(name)
for name in names:
    m=re.search(r'[en]$',name)
    if m!=None:
        print(name)
```

Output

```
naresh
kiran
kishore
kiran
raman
kishore
```