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.

Example:

```
import re
def main():
    l=re.findall('py','python jython python ironpython')
    print(l)
    pattern=re.compile("py")
    l=pattern.findall("python jython python ironpython")
    print(l)
    l=re.findall("PY","PYTHON python jyton
IronPYTHON",re.IGNORECASE)
    print(l)
    pattern=re.compile("PY",re.IGNORECASE)
    l=pattern.findall("PYTHON python jython IronPYTHON")
    print(l)
main()
```

Output:

```
['py', 'py', 'py']
['py', 'py', 'py']
['PY', 'py', 'PY']
['PY', 'py', 'PY']
>>>
```

The special characters used to create pattern, these characters are having special meaning

.

(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
def main():
```

```
    str1="python"
    l=re.findall(r'.',str1)
    print(l)
    str2="python\nprogramming\nlanguage"
    l=re.findall(r'.',str2)
    print(l)
    l=re.findall(r'.',str2,re.DOTALL)
    print(l)
    str3="python jython rpython ironpython"
    l=re.findall(r'p.',str3)
    print(l)
    str4="dog cat rat bat cow"
    l=re.findall(r'.o.',str4)
    print(l)
    l=re.findall(r'.a.',str4)
    print(l)
main()
```

Output:

```
['p', 'y', 't', 'h', 'o', 'n']
['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']
['p', 'y', 't', 'h', 'o', 'n', '\n', 'p', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'i', 'n', 'g', '\n', 'l', 'a', 'n',
'g', 'u', 'a', 'g', 'e']
['py', 'py', 'py']
['dog', 'cow']
['cat', 'rat', 'bat']
>>>
```

^

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

Example:

```
import re
```

```
def main():
   str1="python langauge"
   m=re.findall('^py',str1)
   print(m)
   str2='''python programming language
   python high level language
   python general purpose langauge'''
   l=re.findall('^py',str2,re.MULTILINE)
   print(l)
main()
```

$

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.

```
import re
def main():
    str1="python"
    m=re.search(r'on$',str1)
    print(m)
main()
```

Output:

<re.Match object; span=(4, 6), match='on'>

Example:

```
import re
def main():
names=['naresh','suresh','rajesh','kishore','ramesh','kiran
']
    for name in names:
        m=re.search(r'sh$',name)
        if m!=None:
            print(name)
    for name in names:
        m=re.search(r'^r....h$',name)
        if m!=None:
            print(name)
main()
```

Output:

naresh

suresh

rajesh
ramesh
rajesh
ramesh

*

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.

Example:

```
import re
def main():
    names=['naresh','suresh','rajesh','kishore','ramesh','kiran','rash','rh','r','h']
    for name in names:
        m=re.search(r'^r.*h$',name)
        if m!=None:
            print(name)

    str1="a ab acb abb aa"
    l=re.findall(r'ab*',str1)
    print(l)
main()
```

Output:

```
rajesh
ramesh
rash
rh
['a', 'ab', 'a', 'abb', '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’.

Example

```
import re
def main():
```

```
    str1="ab abb a abbb"
    l=re.findall('ab+',str1)
    print(l)
main()
```

Output:

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

?

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

Example:

```
import re
def main():
    str1="ab abb a abbb"
    l=re.findall('ab?',str1)
    print(l)
main()
```

Output:

```
['ab', 'ab', 'a', '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

Example:

```
import re
def main():
    str1="ab abb abbb abbbb"
    l=re.findall(r'ab{3}',str1)
    print(l)
    names=['ramesh','rajesh','rama','kishore']
    for name in names:
        m=re.search(r'.{6}',name)
```

```
        if m!=None:
            print(name)
main()
```

Output:

```
['abbb', 'abbb']
ramesh
rajesh
kishore
>>>
```

{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'.