Python Regular Expressions Methods:

- If we want to represent a group of Strings according to a particular format/pattern then we should go for Regular Expressions.
- i.e Regualr Expressions is a declarative mechanism to represent a group of Strings accroding to particular format/pattern.

Eg 1: We can write a regular expression to represent all mobile numbers

Eg 2: We can write a regular expression to represent all mail ids.

- We can develop Regular Expression Based applications by using python module: re
- This module contains several inbuilt functions to use Regular Expressions very easily in our applications.

For example some Important functions of "re" module:

1. match()

1. **start() --->** returns the start index of the matched substring.
2. **end() --->** returns the end index of the matched substring.
3. **group() --->** returns the part of the string where there is a match.
4. **span() --->** returns a tuple containing start and end index of the matched part.

match.re and match.string

The `re` attribute of a matched object returns a regular expression object. Similarly, `string` attribute returns the passed string.

```
>>> match.re
re.compile('(\\d{3}) (\\d{2})')

>>> match.string
'39801 356, 2102 1111'
```

2. fullmatch()

3. search()

4. findall()

5. sub()

6. subn()

7. split()
8. compile()
9. finditer()

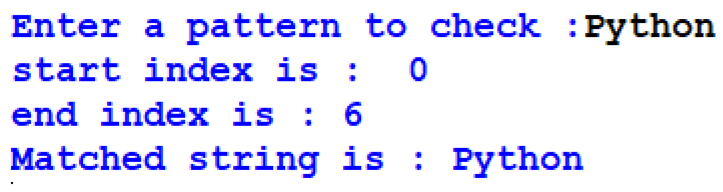
1. match():

We can use match() function to check the given pattern at **'beginning'** of target string.
If the match is available then we will get **Match object**, otherwise we will get **None**.

Example:

```
import  re
s1 = input('Enter a pattern to check :')
m = re.match(s1 , 'Python is a language')
print('start index is : ' , m.start()) ,
print('end index is :' , m.end())
print('Matched string is :' ,  m.group())
```

Output:



```
Enter a pattern to check :Python
start index is :   0
end index is : 6
Matched string is : Python
```

2. fullmatch():

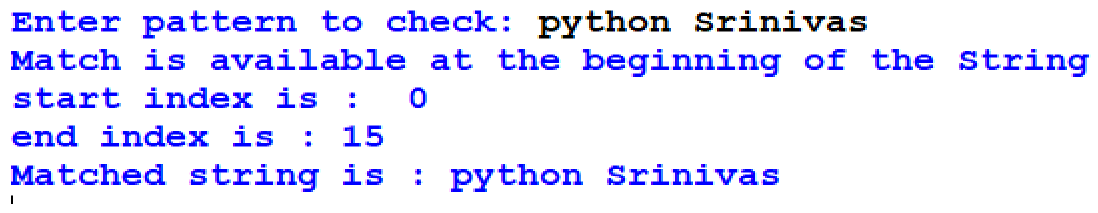
We can use **fullmatch()** function to match a pattern to all of target string.
i.e complete string should be matched according to given pattern.
If **complete string matched** then this function returns **Match object** otherwise it returns **None**.

Example:

```
import  re
s = input("Enter pattern to check: ")
m = re.fullmatch(s,"python Srinivas")
if m != None:
      print("Match is available at the beginning of the String")
      print('start index is : ' , m.start()) ,
      print('end index is :' , m.end())
```

```
    print('Matched string is :' ,  m.group())
else:
    print("Match is not available at the beginning of the String")
```

Output:



```
Enter pattern to check: python Srinivas
Match is available at the beginning of the String
start index is :  0
end index is : 15
Matched string is : python Srinivas
```

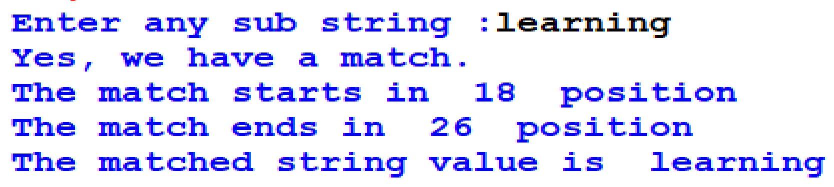
3. search():

We can use **search()** function to search the given pattern in the target string. If the match is available then it returns the Match object which represents first occurrence of the match. If the match is not available then it returns None

Example:

```
import  re
target_string = 'Python is easy to learning'
sub_string = input("Enter any sub string :")
match_object = re.search(sub_string , target_string)
if match_object:
  print('Yes, we have a match.')
  print("The match starts in ", match_object.start()," position")
  print("The match ends in ", match_object.end()," position")
  print("The matched string value is ", match_object.group())
else:
   print('No, we don"t have any match.')
```

Output:



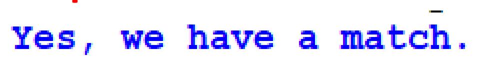
```
Enter any sub string :learning
Yes, we have a match.
The match starts in  18  position
The match ends in  26  position
The matched string value is  learning
```

Q. How to check if the string starts with 'Python' and ends with 'learning' ?

```
import re
target_string = 'Python is easy to learning'
m = re.search("^Python.*learning$", target_string)
```

```
if m:
  print('Yes, we have a match.')
else:
   print('No, we don"t have any match.')
```

Output:

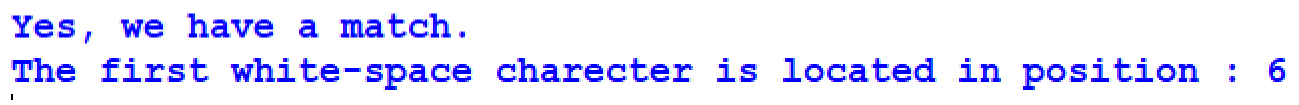


```
Yes, we have a match.
```

Example3: How to find out white space is available or not in a target string.

```
import re
target_string = 'Python is easy to learning'
m = re.search("\s", target_string)
if  m:
  print('Yes, we have a match.')
  print('The first white-space charecter is located in position :' , m.start())
else:
   print('No, we don"t have any match.')
```

Output 1:



```
Yes, we have a match.
The first white-space charecter is located in position : 6
```

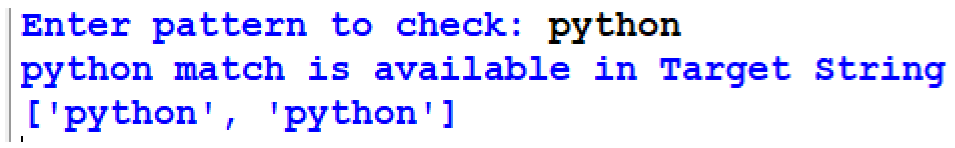
4. findall():

We can use **findall()** function to find all occurrences of the pattern match. This function returns a **list object** which contains all occurrences of patterns. Otherwise it returns **empty list** object.

Example:

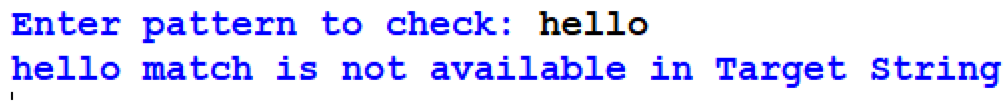
```
import re
pattern_str = input("Enter pattern to check: ")
result = re.findall(pattern_str ,  "python is a language.python is easy.")

if len(result) != 0:
    print(pattern_str ,"match is available in Target String")
    print(result)
else:
    print(pattern_str , "match is not available in Target String")
```

Output 1:



```
Enter pattern to check: python
python match is available in Target String
['python', 'python']
```

Output 2:

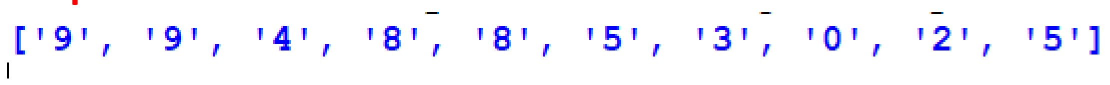


```
Enter pattern to check: hello
hello match is not available in Target String
```

Q. How to find out only list of digits from given alpha_numeric string?

```
import re
digits_lists = re.findall("[0-9]","9py9th4on8 Sr8ina5vas3rao025")
print(digits_ lists)
```

Output:

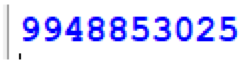


```
['9', '9', '4', '8', '8', '5', '3', '0', '2', '5']
```

Q. How to find out only numbers from given alpha_numeric string?

```
import re
digits = re.findall("[0-9]","9py9th4on8 Sr8ina5vas3rao025")
numbers = "".join(digits)
print(numbers)
```

Output:



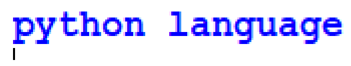
```
9948853025
```

Q. How to find out only non-digits data from given alpha_numeric string?

Way-1:

```
import  re
values_list = re.findall('[\D]' , "1py23th5on la96ngu3age2")
string_data  = ''.join(values_list)
print(string_data)
```

Output:



```
python language
```

Way-2:

```
import  re
values_list = re.findall('[a-zA-Z\s]' , "1py23th5on la96ngu3age2")
```

```
string_data  = ''.join(values_list)
print(string_data)
```

Output:

```
python language
```

Q. How to find out given alpha-numeric string ends with 25 or not?

```
import re
end_digits = re.findall("25$","9py9th4on8 sr8ini5va3srao025")
print(end_digits)
```

Output:

```
['25']
```

Q. Task: Write a pattern to check which of the words do not have any vowels in the given Statement ? Note: We shall use \w* pattern for the purpose.

```
import re
list_data = re.findall(r"\w*", "Fly in the sky.")
print(list_data)
for word in list_data:
   data = re.search(r"[aeiou]" , word)
   if word!='' and data==None:
      print(word)
```

Output:

```
['Fly', '', 'in', '', 'the', '', 'sky', '', '']
Fly
sky
```

## Finding word starting with vowels

```
import re
string = 'Errors should never pass silently. Unless explicitly silenced.'
obj = re.findall(r'\b[aeiouAEIOU]\w+', string)
print(obj)
```

Output:

```
['Errors', 'Unless', 'explicitly']
```

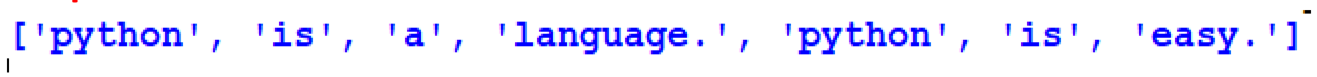
6. split():

The **split()** function returns a list object where the string has been split at each match.

Q. Write a pattern to split the given statement as number of words?

```
import re
m = re.split('\s',"python is a language. python is easy.")
print(m)
```

Output:



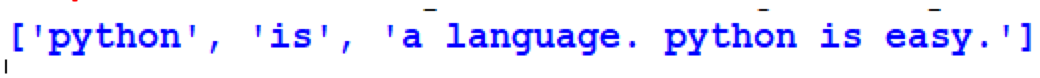
```
['python', 'is', 'a', 'language.', 'python', 'is', 'easy.']
```

Note: You can control the number of occurrences by specifying the 'maxsplit' parameter.

Example:

```
import re
m = re.split('\s',"python is a language. python is easy.",2)
print(m)
```

Output:



```
['python', 'is', 'a language. python is easy.']
```

7. sub():

Here sub() means substitution or replacement.

The sub() function replaces the matches with the text of your choice.

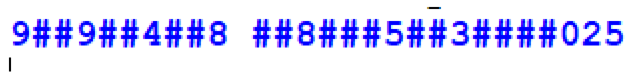
Syntax : re.sub(regex , replacement_string , target_string)

In the target string every matched regex pattern will be replaced with provided replacement values.

Q. How to Replace the every charecter with # symbol in given target string?

```
import  re
result  =  re.sub('[a-zA-Z]' , "#" , "9py9th4on8 sr8ini5va3srao025")
print(result)
```

Output:



```
9##9###4###8 ##8####5##3####025
```

Note: You can control the number of replacements by specifying the *'count'* parameter.

```
import re
result = re.sub('[a-zA-Z]',"#","9py9th4on8 sr8ini5va3srao025",4)
```

```
print(result)
```

Output:

```
9##9##4on8 sr8ini5va3srao025
```

8. subn():

It is exactly same as sub() except it can also returns the number of replacements. This function returns a tuple where first element is "result string" and second element is "number of replacements".

Syntax : subn(result_string , number_of_replacements)

Example:

```
import re
result = re.subn('[a-zA-Z]',"#","9py9th4on8 sr8ini5va3srao025")
print(result)
print("The Result String:",result[0])
print("The number of replacements:",result[1])
```

Output:

```
('9##9###4###8 ##8####5###3####025', 17)
The Result String: 9##9###4###8 ##8####5###3####025
The number of replacements: 17
```

re.finditer()

The re.finditer() function returns an iterator object of all matches in the target string.

For each matched group, start and end positions can be obtained by span() method

For each matched group, start position can be obtained by start() method

For each matched group, end position can be obtained by end() method

For each matched group, matched string value can be obtained by group() method

Example:

```
from re import finditer
string = "Try to earn while you learn"
iter_object = finditer("earn", string)
for match in iter_object:
   print(match.span())
```

```
    print('start index is : ' , match.start()) ,
    print('end index is :' , match.end())
    print('Matched string is :' ,  match.group())
    print()
```

Output:

```
(7, 11)
start index is :   7
end index is : 11
Matched string is : earn
(23, 27)
start index is :   23
end index is : 27
Matched string is : earn
```

re.compile()

The re.compile() function returns a pattern object which can be repeatedly used in different regex functions.

In the following example, a string **'is'** is compiled to get a pattern object and is subjected to the search() method.

Example:

```
from re import *
pattern = compile(r'[aeiou]')

string = "Flat is better than nested. Sparse is better than dense. xyz"
list_of_words = split('\s', string)
for word in list_of_words:
    print(word,'--->>', pattern.match(word))
```

Output:

```
Flat --->> None
is --->> <re.Match object; span=(0, 1), match='i'>
better --->> None
than --->> None
nested. --->> None
Sparse --->> None
is --->> <re.Match object; span=(0, 1), match='i'>
better --->> None
than --->> None
dense. --->> None
xyz --->> None
```

Note: The same pattern object can be reused in searching for words having vowels, as shown below.

```
for word in list_of_words:
   print(word,'--->>', pattern.search(word))
```

Output:

```
Flat --->> <re.Match object; span=(2, 3), match='a'>
is --->> <re.Match object; span=(0, 1), match='i'>
better --->> <re.Match object; span=(1, 2), match='e'>
than --->> <re.Match object; span=(2, 3), match='a'>
nested. --->> <re.Match object; span=(1, 2), match='e'>
Sparse --->> <re.Match object; span=(2, 3), match='a'>
is --->> <re.Match object; span=(0, 1), match='i'>
better --->> <re.Match object; span=(1, 2), match='e'>
than --->> <re.Match object; span=(2, 3), match='a'>
dense. --->> <re.Match object; span=(1, 2), match='e'>
xyz --->> None
```

re.compile():

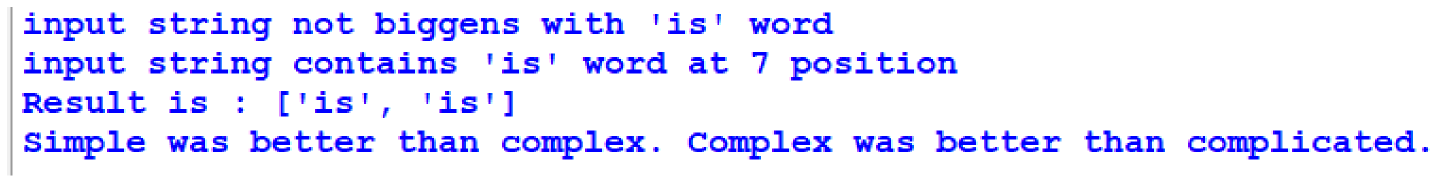
This function compiles a regular expression pattern into a regular expression object. This is useful when you need to use an expression several times.

Example:

```
import re
string = 'Simple is better than complex. Complex is better than complicated.'
pattern = re.compile(r'is')
obj1 = pattern.match(string)
obj2 = pattern.search(string)
obj3 = pattern.findall(string)
```

```
obj4 = pattern.sub(r'was', string)

if obj1:
   print("input string biggens with 'is' word",)
else:
   print("input string not biggens with 'is' word",)

if obj2:
   print("input string contains 'is' word at",obj2.start(),"position")
else:
   print("input string not contains 'is' word")

if obj3:
   print("Result is :",obj3)
else:
   print("input string not contains 'is' word")

if obj4:
   print(obj4)
else:
   print("input string not contains 'is' word")
```

Output:



```
input string not biggens with 'is' word
input string contains 'is' word at 7 position
Result is : ['is', 'is']
Simple was better than complex. Complex was better than complicated.
```