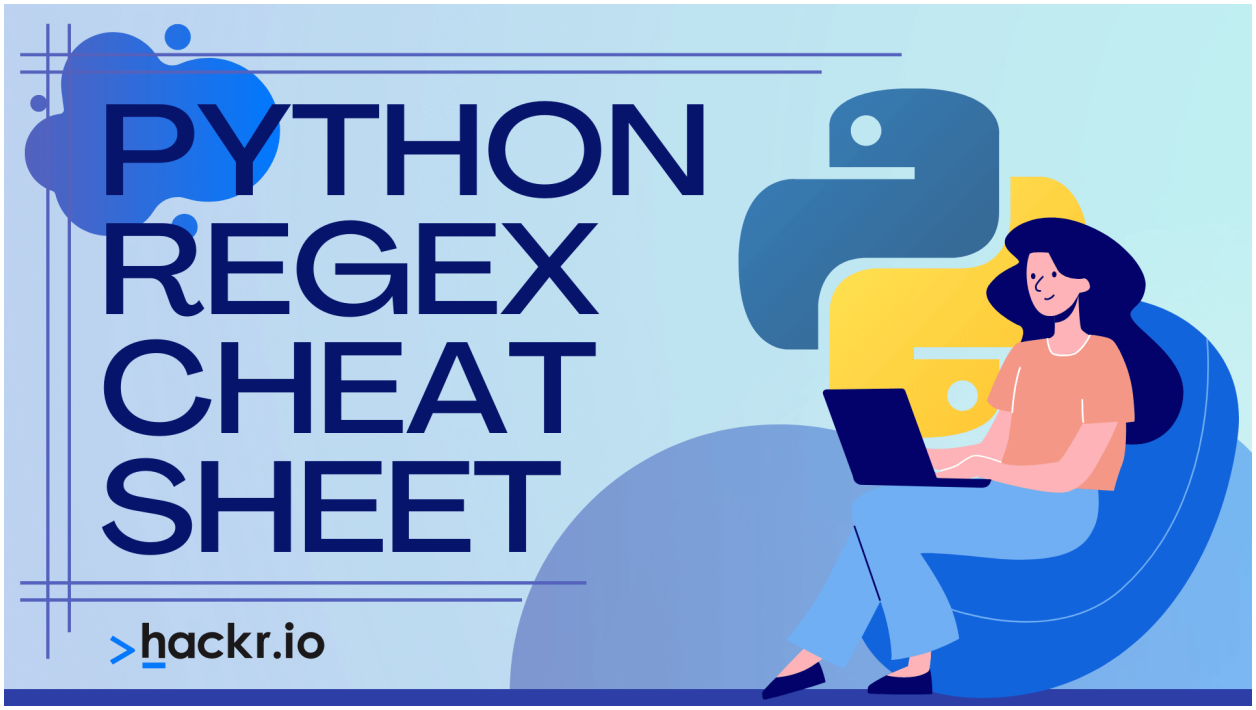

Python Regex Cheat Sheet

In this Python regular expression cheat sheet, we'll cover every syntax with a simple example to explain how it works.

Metacharacters

Metacharacters are regular expression building blocks. The following table highlights different metacharacters in Python RegEx, along with their descriptions and suitable examples:

Character	Description	Example
[]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
$	Ends with	"planet$"
*	Zero or more occurrences	"he.*o"
+	One or more occurrences	"he.+o"

?	Zero or one occurrence	"he.?o"
{}	Exactly the specified number of occurrences	"he.{2}o"
|	Either or	"falls|stays"
()	Capture and group	

Let's discuss each of the metacharacters in detail below.

[] Square Brackets

It is a character class with a set of characters we want to match.

For example, the character class [abc] will match any single character a, b, or c. You can specify any range of characters as required.

For example:

- [0, 3] is the same as [0123]
- [a-c] is the same as [abc].

You can invert the character class using the caret(^) symbol. For example:

- [^0-3] means any number except 0, 1, 2, or 3
- [^a-c] means any character except a, b, or c

For example:

```
import re
txt = "The rain in Spain"
x = re.findall("[a-m]", txt)
print(x)
```

Output

```
['h', 'e', 'a', 'i', 'i', 'a', 'i']
```

\ Backslash

The backslash (\) makes sure the character is not treated in a special way. It is used to escape the metacharacters.

For example, if you want to search for the dot(.) in the string, the searched dot will be treated as a special character. So, you need to use the backslash(\) just before the dot(.)

For example:

```
import re
txt = "That will be 59 dollars"
#Find all digit characters:
x = re.findall("\d", txt)
print(x)
```

Output:

```
['5', '9']
```

.- Dot

Using the dot symbol, you can match only a single character, except the newline character.

- **a.b** will check for the string containing any character at the place of the dot such as acb, acbd, abbb, etc
- **..** will check if the string contains at least two characters

For example:

```
import re

txt = "hello planet"

#Search for a sequence that starts with "he", followed by two (any)
characters, and an "o":

x = re.findall("he..o", txt)
print(x)
```

Output:

```
['hello']
```

^ Caret (Start With)

Caret (^) symbol allows you to match the beginning of the string. It checks whether the string starts with the specific character(s) or not.

For example:

- ^g will check if the string starts with g, such as girl, globe, gym, g, etc.
- ^ge will check if the string starts with ge, such as gem, gel, etc.

For example:

```
import re

txt = "hello planet"

#Check if the string starts with 'hello':

x = re.findall("^hello", txt)
if x:
  print("Yes, the string starts with 'hello'")
else:
  print("No match")
```

Output:

```
Yes, the string starts with 'hello'
```

$ Dollar (end with)

The dollar($) symbol allows you to match the end of the string. It checks whether the string ends with the given character(s) or not.

For example:

- s$ will check for the string that ends with a such as sis, ends, s, etc.
- ks$ will check for the string that ends with ks such as seeks, disks, ks, etc.

For example:

```
import re

```

```
txt = "hello planet"

#Check if the string ends with 'planet':

x = re.findall("planet$", txt)
if x:
  print("Yes, the string ends with 'planet'")
else:
  print("No match")
```

Output:

```
Yes, the string ends with 'world'
```

• Star

This symbol will match zero or more occurrences of the regular expression preceding the * symbol.

For example:

- ab*c will be matched for the string ac, abc, abbbc, dabc, etc. but will not be matched for abdc because b is not followed by c.

For example:

```
import re
txt = "hello planet"
#Search for a sequence that starts with "he", followed by 0 or more  (any)
characters, and an "o":
x = re.findall("he.*o", txt)
print(x)
```

Output:

```
['hello']
```

+ Plus

This symbol will match one or more occurrences of the regular expression preceding the + symbol.

For example:

- ab+c will be matched for the string abc, abbc, dabc, but will not be matched for ac, abdc because there is no b in ac and b is not followed by c in abdc.

For example:

```
import re
txt = "hello planet"
#Search for a sequence that starts with "he", followed by 1 or more  (any)
characters, and an "o":
x = re.findall("he.+o", txt)
print(x)
```

Output:

```
['hello']
```

? Question mark

This symbol will check if the string before the question mark occurs at least once, or not at all.

For example:

- ab?c will match strings ac, acb, and dabc, but will not be matched for abbc because there are two bs. Similarly, it will not be matched for abdc because b is not followed by c.

For example:

```
import re
txt = "hello planet"
#Search for a sequence that starts with "he", followed by 0 or 1  (any)
character, and an "o":
x = re.findall("he.?o", txt)
print(x)
```

Output:

```
[]
```

{m,n}- Braces

Braces match any repetitions preceding RegEx from m to n inclusive.

For example:

- a{2, 4} will be matched for the string aaab, baaaac, gaad, but will not be matched for strings like abc, bc because there is only one a or no a in both the cases.

For example:

```
import re
txt = "hello planet"
#Search for a sequence that starts with "he", followed excactly 2 (any)
characters, and an "o":
x = re.findall("he.{2}o", txt)
print(x)
```

Output:

```
['hello']
```

| - OR

The Or symbolt checks whether the pattern before or after the “or” symbol is present in the string or not.

For example:

- a|b will match any string that contains a or b such as acd, bcd, abcd, etc.

For example:

```
import re
txt = "The rain in Spain falls mainly in the plain!"
#Check if the string contains either "falls" or "stays":
x = re.findall("falls|stays", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

Output:

```
['falls']
Yes, there is at least one match!
```

(<RegEx>)- Group

Group symbol is used to group sub-patterns.

For example:

- (a|b)cd will match for strings like acd, abcd, gacd, etc.

Special Sequences

In this section of our RegEx Python cheat sheet, we'll discuss various special sequences with suitable examples.

Character	Description	Example
\A	Returns a match if the specified characters are at the beginning of the string	"\AThe"
\b	Returns a match where the specified characters are at the beginning or at the end of a word (the "r" in the beginning ensures the string is being treated as a "raw string")	r"\bain" r"ain\b"
\B	Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\Bain" r"ain\B"
\d	Returns a match where the string contains digits (numbers from 0-9)	"\d"
\D	Returns a match where the string DOES NOT contain digits	"\D"
\s	Returns a match where the string contains a white space character	"\s"
\S	Returns a match where the string DOES NOT contain a white space character	"\S"
\w	Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)	"\w"

\W	Returns a match where the string DOES NOT contain any word characters	"\W"
\Z	Returns a match if the specified characters are at the end of the string	"Spain\Z"

\A

```
import re
txt = "The rain in Spain"

#Check if the string starts with "The":
x = re.findall("\AThe", txt)
print(x)
if x:
  print("Yes, there is a match!")
else:
  print("No match")
```

Output:

```
['The']
Yes, there is a match!
```

\b

```
import re
txt = "The rain in Spain"
#Check if "ain" is present at the beginning of a WORD:
x = re.findall(r"\bain", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")

#Check if "ain" is present at the end of a WORD:
x = re.findall(r"ain\b", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
```

```
  print("No match")
```

Output:

```
[]
No match

['ain', 'ain']
Yes, there is at least one match!
```

\B

```
import re

txt = "The rain in Spain"
#Check if "ain" is present, but NOT at the beginning of a word:
x = re.findall(r"\Bain", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

Output:

```
['ain', 'ain']
Yes, there is at least one match!
```

\d

```
import re
txt = "The rain in Spain"
#Check if the string contains any digits (numbers from 0-9):
x = re.findall("\d", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

Output:

```
[]
No match
```

\D

```
import re
txt = "The rain in Spain"
#Return a match at every no-digit character:
x = re.findall("\D", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

Output:

```
['T', 'h', 'e', ' ', 'r', 'a', 'i', 'n', ' ', 'i', 'n', ' ', 'S', 'p', 'a',
'i', 'n']
Yes, there is at least one match!
```

\s

```
import re
txt = "The rain in Spain"
#Return a match at every white-space character:
x = re.findall("\s", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

Output:

```
[' ', ' ', ' ']
Yes, there is at least one match!
```

\S

```
import re
txt = "The rain in Spain"
#Return a match at every NON white-space character:
x = re.findall("\S", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

Output:

```
['T', 'h', 'e', 'r', 'a', 'i', 'n', 'i', 'n', 'S', 'p', 'a', 'i', 'n']
Yes, there is at least one match!
```

\w

```
import re
txt = "The rain in Spain"
#Return a match at evry word character (characters from a to Z, digits from
0-9, and the underscore _ character):
x = re.findall("\w", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

Output:

```
['T', 'h', 'e', 'r', 'a', 'i', 'n', 'i', 'n', 'S', 'p', 'a', 'i', 'n']
Yes, there is at least one match!
```

\W

```
import re
txt = "The rain in Spain"
#Return a match at every NON word character (characters NOT between a and
Z. Like "!", "?" white-space etc.):
```

```
x = re.findall("\W", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

Output:

```
[' ', ' ', ' ']
Yes, there is at least one match!
```

\Z

```
import re
txt = "Te rain in Spain"
#Check if the string ends with "Spain":
x = re.findall("Spain\Z", txt)
print(x)
if x:
  print("Yes, there is a match!")
else:
  print("No match")
```

Output:

```
['Spain']
Yes, there is a match!
```

Sets

This is a set of characters enclosed in square brackets [] with a special meaning. In this section of our Python regular expressions cheat sheet, we'll explain all set types with examples.

[arn]

This will return a match where one of the specified characters (a, r, or n) are present.

For example:

```
import re
txt = "The rain in Spain"
#Check if the string has any a, r, or n characters:
x = re.findall("[arn]", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

Output:

```
['r', 'a', 'n', 'n', 'a', 'n']
Yes, there is at least one match!
```

[a-n]

This will return a match for any lower case character, alphabetically between a and n.

For example:

```
import re
txt = "The rain in Spain"
#Check if the string has any characters between a and n:
x = re.findall("[a-n]", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

Output:

```
['h', 'e', 'a', 'i', 'n', 'i', 'n', 'a', 'i', 'n']
Yes, there is at least one match!
```

[^arn]

This will return a match for any character EXCEPT a, r, and n.

For example:

```
import re
txt = "The rain in Spain"
#Check if the string has other characters than a, r, or n:
x = re.findall("[^arn]", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

Output:

```
['T', 'h', 'e', ' ', 'i', ' ', 'i', ' ', 'S', 'p', 'i']
Yes, there is at least one match!
```

[0123]

This will return a match where any of the specified digits (0, 1, 2, or 3) are present.

For example:

```
import re
txt = "The rain in Spain"
#Check if the string has any 0, 1, 2, or 3 digits:
x = re.findall("[0123]", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

Output:

```
[]
No match
```

[0-9]

This will return a match for any digit between 0 and 9.

For example:

```
import re
txt = "8 times before 11:45 AM"
#Check if the string has any digits:
x = re.findall("[0-9]", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

Output:

```
['8', '1', '1', '4', '5']
Yes, there is at least one match!
```

[0-5][0-9]

This will return a match for any two-digit numbers from 00 and 59.

For example:

```
import re
txt = "8 times before 11:45 AM"
#Check if the string has any two-digit numbers, from 00 to 59:
x = re.findall("[0-5][0-9]", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

Output:

```
['11', '45']
Yes, there is at least one match!
```

[a-zA-Z]

This will return a match for any character alphabetically between a and z, lower case OR upper case.

For example:

```
import re
txt = "8 times before 11:45 AM"
#Check if the string has any characters from a to z lower case, and A to Z
upper case:
x = re.findall("[a-zA-Z]", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

Output:

```
['t', 'i', 'm', 'e', 's', 'b', 'e', 'f', 'o', 'r', 'e', 'A', 'M']
Yes, there is at least one match!
```

[+]

In sets, +, *, ., |, (), $,{} has no special meaning. So, [+] means: return a match for any + character in the string.

For example:

```
import re
txt = "8 times before 11:45 AM"
#Check if the string has any + characters:
x = re.findall("[+]", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

Output:

```
[]
No match
```

Regex Module in Python

Python comes with a module named ‘re’. You must have noticed in the above example where we have imported the module ‘re’. This module consists of several functions that will help you perform various actions.

findall() function

This is a built-in function of the ‘re;’ module that handles the regular expression.

Syntax:

```
re.findall(pattern, string, flags=0)
```

- Pattern is the regular expression.
- String is the input string provided by the user.
- Flags are used to modify the standard pattern behavior.

Each string is evaluated from left to right and finds all the matches of the pattern within the string. However, the result depends on the pattern.

- If the pattern has **no capturing groups**, the findall() function returns a list of strings that match the whole pattern.
- If the pattern has **one capturing group,** the findall() function returns a list of strings that match the group.
- If the pattern has **multiple capturing groups**, the findall() function returns the tuples of strings that match the groups.
- It’s important to note that the non-capturing groups do not affect the form of the return result.

For example:

- **Get a list of matched strings**

```
import re

s = "black, blue and brown"
pattern = r'bl\w+'
matches = re.findall(pattern,s)

print(matches)
```

Output:

```
['black', 'blue']
```

- **Pattern with a single group**

```
import re
s = "black, blue and brown"
pattern = r'bl(\w+)'
matches = re.findall(pattern,s)
print(matches)
```

Output:

```
['ack', 'ue']
```

- **Pattern with multiple groups**

```
import re
s = "black, blue and brown"
pattern = r'(bl(\w+))'
matches = re.findall(pattern,s)
print(matches)
```

Output

```
[('black', 'ack'), ('blue', 'ue')]
```

- **Using regular expression flag**

```
import re
s = "Black, blue and brown"
pattern = r'(bl(\w+))'
matches = re.findall(pattern, s, re.IGNORECASE)
print(matches)
```

Output:

```
[('Black', 'ack'), ('blue', 'ue')]
```

finditer() function

Using this function, you can match a pattern in a string and returns an iterator yielding the matched objects of all non-overlapping matches.

Syntax:

```
re.finditer(pattern, string, flags=0)
```

- Pattern is the regular expression.
- String is the input string provided by the user.
- Flag is optional and by default is zero. It accepts one or more RegEx flags. The flags parameter changes how the RegEx engine matches the pattern.

For example:

```
import re
s = 'Readability counts.'
pattern = r'[aeoui]'
matches = re.finditer(pattern, s)
for match in matches:
    print(match)
```

Output:

```
<re.Match object; span=(1, 2), match='e'>
<re.Match object; span=(2, 3), match='a'>
<re.Match object; span=(4, 5), match='a'>
<re.Match object; span=(6, 7), match='i'>
<re.Match object; span=(8, 9), match='i'>
<re.Match object; span=(13, 14), match='o'>
<re.Match object; span=(14, 15), match='u'>
```

Search() Function

The search() function scans the string from left to right and finds the first location where the pattern produces a match. It returns a Match object if the search was successful or None otherwise.

Syntax:

```
re.search(pattern, string, flags=0)
```

- Pattern is the regular expression.
- String is the input string provided by the user.
- Flags are used to modify the standard pattern behavior of the pattern.

For example:

- **Finding the first match**

```
import re
s = 'Python 3 was released on Dec 3, 2008'
pattern = '\d+'
match = re.search(pattern, s)
if match is not None:
    print(match.group())
else:
    print('No match found')
```

Output

<

```
re.Match object; span=(7, 8), match='3'>
```

- **Finding the first word matching the pattern**

```
import re
s = 'CPython, IronPython, or Cython'
pattern = r'\b((\w+)thon)\b'
match = re.search(pattern, s)
if match is not None:
    print(match.groups())
```

Output:

```
('CPython', 'CPy')
```

The pattern r'\b((\w+)thon)\b' has two capturing groups:

- (\w+) – captures the characters at the beginning of the word.
- ((\w+)thon) – captures the whole word.

The search() function returns the first location where it finds the match.

fullmatch() function

This function will return a match object if the whole string matches a regular expression's search pattern, or none otherwise.

Syntax:

```
re.fullmatch(pattern, string, flags=0)
```

- Pattern is the regular expression.
- String is the input string provided by the user.
- Flag is optional and by default is zero. It accepts one or more RegEx flags. The flags parameter changes how the RegEx engine matches the pattern.

For example:

- **To validate an email address**

```
import re
email = 'no-reply@pythontutorial.net'
pattern = r'[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}'
match = re.fullmatch(pattern, email)
if match is not None:
    print(f'The email "{match.group()}" is valid')
else:
    print(f'The email "{email}"" is not valid')
```

Output:

```
The email "no-reply@pythontutorial.net" is valid.
```

Match() Function

The match function of the re module allows you to search for a pattern at the beginning of the string.

Syntax:

```
re.match(pattern, string, flags=0)
```

- Pattern is the regular expression.
- String is the input string provided by the user.
- Flags are used to modify the standard behavior of the pattern.

For example:

- **To check if a string starts with a digit.**

```
import re
s = '3 pieces cost 5 USD'
pattern = r'\d{1}'
match = re.match(pattern, s)
if match is not None:
    print(f'The string starts with a digit {match.group()}')
```

Output:

```
The string starts with a digit 3
```

Sub() Function

This function of the re module allows you to handle the regular expression.

Syntax:

```
re.sub(pattern, repl, string, count=0, flags=0)
```

- Pattern is a regular expression or Pattern object.
- Repl is the replacement.
- String is the input string provided by the user.
- Count parameter specifies the maximum number of matches that the sub() function should replace. If you pass zero or skip it, the sub()function will replace all the matches.
- Fags is one or more RegEx flags to modify the standard pattern behavior.

It will search for the pattern in the string and replace the matched strings with the replacement (repl). If the sub() function couldn't find a match, it will return the original string. Otherwise, the sub()function returns the string after replacing the matches.

For example:

- **To turn the phone number (212)-456-7890 into 2124567890**

```
import re
phone_no = '(212)-456-7890'
pattern = '\D'
result = re.sub(pattern, '',phone_no)
print(result)
```

Output:

```
2124567890
```

- **To replace the leftmost non-overlapping occurrences of a pattern**

```
import re
pattern = '00'
s = '00000'
result = re.sub(pattern,'',s)
print(result)
```

Output:

```
0
```

- **Backreference example**

```
import re
s = 'Make the World a *Better Place*'
pattern = r'\*(.*?)\*'
replacement = r'<b>\1<\\b>'
html = re.sub(pattern, replacement, s)
print(html)
```

Output:

```
Make the World a <b>Better Place<\b>
```

Subn() Function

This function is similar to sub() in all ways, except in how it provides output. It returns a tuple with count of the total of replacement and the new string rather than just the string.

Syntax:

```
re.subn(pattern, repl, string, count=0, flags=0)
```

For example:

```
import re
print(re.subn('ub', '~*', 'Subject has Uber booked already'))
t = re.subn('ub', '~*', 'Subject has Uber booked already',
            flags=re.IGNORECASE)
print(t)
print(len(t))
# This will give same output as sub() would have
print(t[0])
```

Output:

```
('S~*ject has Uber booked already', 1)
('S~*ject has ~*er booked already', 2)
Length of Tuple is:  2
S~*ject has ~*er booked already
```

escape() function

This function will return a string with all non-alphanumerics backslashes. This is useful if you want to match an arbitrary literal string that may have regular expression metacharacters in it.

Syntax:

```
re.escape(string)
```

For example:

```
import re
print(re.escape("This is Awesome even 1 AM"))
print(re.escape("I Asked what is this [a-9], he said \t ^WoW"))
```

Output:

```
This\ is\ Awesome\ even\ 1\ AM
I\ Asked\ what\ is\ this\ \[a\-9\]\,\ he\ said\ \    \ \^WoW
```

Compile() Function

This function will compile the regular expressions into pattern objects, which have methods for various operations such as searching for pattern matches or performing string substitutions.

Syntax:

```
re.compile(string)
```

For example:

```
import re
p = re.compile('[a-e]')
# findall() searches for the Regular Expression
# and return a list upon finding
print(p.findall("Aye, said Mr. Gibenson Stark"))
```

Output:

```
['e', 'a', 'd', 'b', 'e', 'a']
```

Split() Function

It splits a string by the matches of a regular expression.

Syntax:

```
split(pattern, string, maxsplit=0, flags=0)
```

- Pattern is the regular expression.
- String is the input string provided by the user.
- Flag is optional and by default is zero. It accepts one or more RegEx flags. The flags parameter changes how the RegEx engine matches the pattern.
- maxsplit determines at most the splits occur. Generally, if the maxsplit is one, the resulting list will have two elements. If the maxsplit is two, the resulting list will have three elements, and so on.

For example:

- **To split the words in a sentence**

```
import re
s = 'A! B. C D'
pattern = r'\W+'

l = re.split(pattern, s)
print(l)
```

Output

```
['A', 'B', 'C', 'D']
```

- **With a capturing group**

```
import re
s = 'A! B. C D'
pattern = r'(\W+)'
l = re.split(pattern, s, 2)
print(l)
```

Output

```
['A', '! ', 'B', '. ', 'C D']
```

Groups

A group is a part of a regular expression enclosed in parentheses () metacharacter.

Expressions	Explanations
( )	Matches the expression inside the parentheses and groups it to capture as required
(?#…)	Read a comment
(?PAB)	Matches the expression AB, which can be retrieved with the group name.
(?:A)	Matches the expression as represented by A, but cannot be retrieved afterwards.
(?P=group)	Matches the expression matched by an earlier group named “group”

For example:

```
import re

example = (re.search(r"(?:AB)","ACABC"))
print(example)
print(example.groups())
result = re.search(r"(\w*), (\w*)","seeks, zest")
print(result.groups())
```

Output:

```
<re.Match object; span=(2, 4), match='AB'>
()
('seeks', 'zest')
```

Assertions

In Python RegEx, we use Lookahead as an assertion. It determines the success or failure regarding whether the pattern is to the right of the parser's current position.

Expression	Explanation
A(?=B)	Matches the expression A only if it is followed by B. (Positive lookahead assertion)
A(?!B)	Matches the expression A only if it is not followed by B. (Negative look ahead assertion)
(?<=B)A	Matches the expression A only if B is immediate to its left. (Positive look behind assertion)
(?<!B)A	Matches expression A only if B is not immediately to its left. (Negative look behind assertion)

For example:

```
import re

print(re.search(r"z(?=a)", "pizza"))
print(re.search(r"z(?!a)", "pizza"))
```

Output

```
<re.Match object; span=(3, 4), match='z'>
<re.Match object; span=(2, 3), match='z'>
```

Flags or Modifiers

Expression	Explanation
a	Matches ASCII only
i	Ignore case
L	Locale character classes
m	^ and $ match start and end of the line (Multi-line)
s	Matches everything including newline as well
u	Matches unicode character classes
x	Allow spaces and comments (Verbose)

For example:

```
import re

exp = """hello you
I am from
Hoolywood"""

print(re.search(r"and", "Sun And Moon", flags=re.IGNORECASE))
print(re.findall(r"^\w", exp, flags = re.MULTILINE))
```

Output:

```
<re.Match object; span=(4, 7), match='And'>
['h', 'I', 'H']
```