

## SPECIAL CHARACTERS

- \|** | Matches the expression to its right at the start of a string. It matches every such instance before each `\n` in the string.
- \$|** | Matches the expression to its left at the end of a string. It matches every such instance before each `\n` in the string.
- .|** | Matches any character except line terminators like `\n`.
- \|** | Escapes special characters or denotes character classes.
- A|B|** | Matches expression **A** or **B**. If **A** is matched first, **B** is left untried.
- +|** | Greedily matches the expression to its left 1 or more times.
- \*|** | Greedily matches the expression to its left 0 or more times.
- ?|** | Greedily matches the expression to its left 0 or 1 times. But if **?** is added to qualifiers (**+, \*,** and **?** itself) it will perform matches in a non-greedy manner.
- {m}|** | Matches the expression to its left **m** times, and not less.
- {m,n}|** | Matches the expression to its left **m** to **n** times, and not less.
- {m,n}?|** | Matches the expression to its left **m** times, and ignores **n**. See **?** above.

## CHARACTER CLASSES

### [A.K.A. SPECIAL SEQUENCES]

- \w|** | Matches alphanumeric characters, which means **a-z**, **A-Z**, and **0-9**. It also matches the underscore, `_`.
- \d|** | Matches digits, which means **0-9**.
- \D|** | Matches any non-digits.
- \s|** | Matches whitespace characters, which include the `\t`, `\n`, `\r`, and space characters.
- \S|** | Matches non-whitespace characters.
- \b|** | Matches the boundary (or empty string) at the start and end of a word, that is, between `\w` and `\W`.
- \B|** | Matches where **\b** does not, that is, the boundary of `\w` characters.

**\A|** | Matches the expression to its right at the absolute start of a string whether in single or multi-line mode.

**\Z|** | Matches the expression to its left at the absolute end of a string whether in single or multi-line mode.

## SETS

- [ ]|** | Contains a set of characters to match.
- [amk]|** | Matches either **a**, **m**, or **k**. It does not match **amk**.
- [a-z]|** | Matches any alphabet from **a** to **z**.
- [a\z]|** | Matches **a**, **-**, or **z**. It matches **-** because `\` escapes it.
- [a-]|** | Matches **a** or **-**, because **-** is not being used to indicate a series of characters.
- [-a]|** | As above, matches **a** or **-**.
- [a-z0-9]|** | Matches characters from **a** to **z** and also from **0** to **9**.
- [(+\*)]|** | Special characters become literal inside a set, so this matches **(, +, \*, and )**.
- [^ab5]|** | Adding **^** excludes any character in the set. Here, it matches characters that are not **a**, **b**, or **5**.

## GROUPS

- ( )|** | Matches the expression inside the parentheses and groups it.
- (?)|** | Inside parentheses like this, **?** acts as an extension notation. Its meaning depends on the character immediately to its right.
- (?PAB)|** | Matches the expression **AB**, and it can be accessed with the group name.
- (?aiLmsux)|** | Here, **a**, **i**, **L**, **m**, **s**, **u**, and **x** are flags:
  - a** — Matches ASCII only
  - i** — Ignore case
  - L** — Locale dependent
  - m** — Multi-line
  - s** — Matches all
  - u** — Matches unicode
  - x** — Verbose

**(?:A)|** | Matches the expression as represented by **A**, but unlike **(?PAB)**, it cannot be retrieved afterwards.

**(?#...)|** | A comment. Contents are for us to read, not for matching.

**A(?=B)|** | Lookahead assertion. This matches the expression **A** only if it is followed by **B**.

**A(?!=B)|** | Negative lookahead assertion. This matches the expression **A** only if it is not followed by **B**.

**(?<=B)A|** | Positive lookbehind assertion. This matches the expression **A** only if **B** is immediately to its left. This can only match fixed length expressions.

**(?<!B)A|** | Negative lookbehind assertion. This matches the expression **A** only if **B** is not immediately to its left. This can only match fixed length expressions.

**(?P=name)|** | Matches the expression matched by an earlier group named "name".

**(...)1|** | The number **1** corresponds to the first group to be matched. If we want to match more instances of the same expression, simply use its number instead of writing out the whole expression again. We can use from **1** up to **99** such groups and their corresponding numbers.

## POPULAR PYTHON RE MODULE FUNCTIONS

- re.findall(A, B)|** | Matches all instances of an expression **A** in a string **B** and returns them in a list.
- re.search(A, B)|** | Matches the first instance of an expression **A** in a string **B**, and returns it as a `re.match` object.
- re.split(A, B)|** | Split a string **B** into a list using the delimiter **A**.
- re.sub(A, B, C)|** | Replace **A** with **B** in the string **C**.

# Python RegEx Cheatsheet with Examples

A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern. They're typically used to find a sequence of characters within a string so you can extract and manipulate them. For example, the following returns both instances of 'active':

```
import re
pattern = 'ac..ve'
test_string = 'my activestate platform account is now active'
result = re.findall(pattern, test_string)
```

RegExes are extremely useful, but the syntax can be hard to recall. With that in mind, ActiveState offers this "cheatsheet" to help point you in the right direction when building RegExes in Python.

## Special characters

.	match any char except newline (eg., ac..ve)
^	match at beginning of string (eg., ^active)
\$	match at end of string (eg., state\$)
[3a-c]	match any char (ie., 3 or a or b or c)
[^x-z1]	match any char except x, y, z or 1
A S	match either A or S regex
( )	capture & match a group of chars (eg., (8097ba))
\	escape special characters

## Special sequences

\A	match occurrence only at start of string
\Z	match occurrence only at end of string
\b	match empty string at word boundary (e.g.,
\B	match empty string not at word boundary
\d	match a digit
\D	match a non-digit
\s	match any whitespace char: [ \t\n\r\f\v]
\S	match any non-whitespace char
\w	match any alphanumeric: [0-9a-zA-Z_]
\W	match any non-alphanumeric
\g<id>	matches a previously captured group
(?:A)	match expression represented by A (non-capture group)
A(?=B)	match expression A only if followed by B
A(?:!B)	match expression A only if not followed by B
(?<=B)A	match expression A only if it follows B
(?<!B)A	match expression A only if not preceded by B
(?aiLmsux)	where a, i, L, m, s, u, and x are flags:  a = match ASCII only i = make matches ignore case L = make matches locale dependent m = multi-line makes ^ and \$ match at the beginning/end of each line, respectively s = makes '.' match any char including newline u = match unicode only x = verbose increases legibility by allowing comments & ignoring most whitespace

## Quantifiers

*	match 0 or more occurrences (eg., py*)
+	match 1 or more occurrences (eg., py+)
?	match 0 or 1 occurrences (eg., py?)
{m}	match exactly m occurrences (eg., py{3})
{m,n}	match from m to n occurrences (eg., py{1,3})
{,n}	match from 0 to n occurrences (eg., py{,3})
{m,}	match m to infinite occurrences (eg., py{3,})
{m,n}	match m to n occurrences, but as few as possible (eg., py{1,3}?)

## re Module Functions

Besides enabling the above functionality, the 're' module also features a number of popular functions:

re.findall(A, B)	match all occurrences of expression A in string B
re.search(A, B)	match the first occurrence of expression A in string B
re.split(A, B)	split string B into a list using the delimiter A
re.sub(A, B, C)	replace A with B in string C

```
import re
```

```
text="""Text analysis with Python involves processing  
and interpreting textual data to extract  
insights, patterns, and meaning.  
It is very interesting and we are having class upto 24x7.  
And to learn the topic , we require the basic knowledge of PYHTON."""
```

```
x=re.findall("[a-b]",text)  
print(x)
```

```
['a', 'a', 'b', 'a']
```

```
x=re.findall("[A-Z]",text)  
print(x)
```

```
['T', 'P', 'I', 'A', 'P', 'Y', 'H', 'T', 'O', 'N']
```

```
x=re.findall("[A-Z0-9]",text)  
print(x)
```

```
['T', 'P', 'I', '2', '4', '7', 'A', 'P', 'Y', 'H', 'T', 'O', 'N']
```

```
x=re.findall("[A-Z0-9]",text)  
print(x)
```

```
['T', 'P', 'I', '2', '4', '7', 'A', 'P', 'Y', 'H', 'T', 'O', 'N']
```

```
x=re.findall(r"\b\w*\n\b",text)  
print(x)
```

```
['Python', 'learn']
```

```
x=re.findall(r"\b\w*[nN]\b",text)  
print(x)
```

```
['Python', 'learn', 'PYHTON']
```

```
x=re.findall(r"\ba\w*s\b",text)  
print(x)
```

```
['analysis']
```

```
x=re.findall(r"\ba\w*y\w*s\b",text)  
print(x)
```

```
['analysis']
```

```
x=re.findall(r"\b[A-Za-z]{6}\b",text)  
print(x)
```

```
['Python', 'having', 'PYHTON']
```