

# How to Clean Data with Python

## Optional Quantifiers in Regular Expressions

In Regular expressions, optional quantifiers are denoted by a question mark `?`. It indicates that a character can appear either 0 or 1 time. For example, the regular expression `humou?r` will match the text `humour` as well as the text `humor`.

## Character Sets in Regular Expressions

Regular expression character sets denoted by a pair of brackets `[]` will match any of the characters included within the brackets. For example, the regular expression `con[sc]en[sc]us` will match any of the spellings `consensus`, `conscensus`, `consencus`, and `conccensus`.

## Literals in Regular Expressions

In Regular expression, the `literals` are the simplest characters that will match the exact text of the literals. For example, the regex `monkey` will completely match the text `monkey` but will also match `monkey` in text `The monkeys like to eat bananas.`

## Fixed Quantifiers in Regular Expressions

In Regular expressions, fixed quantifiers are denoted by curly braces `{}`. It contains either the exact quantity or the quantity range of characters to be matched. For example, the regular expression `roa{3}r` will match the text `roaaar`, while the regular expression `roa{3,6}r` will match `roaaar`, `roaaaaar`, `roaaaaaar`, or `roaaaaaaar`.

## Alternation in Regular Expressions

Alternation indicated by the pipe symbol `|`, allows for the matching of either of two subexpressions. For example, the regex `baboons|gorillas` will match the text `baboons` as well as the text `gorillas`.

## Anchors in Regular Expressions

Anchors (hat `^` and dollar sign `$`) are used in regular expressions to match text at the start and end of a string, respectively. For example, the regex `^Monkeys: my mortal enemy$` will completely match the text `Monkeys: my mortal enemy` but not match `Spider Monkeys: my mortal enemy` or `Monkeys: my mortal enemy in the wild`. The `^` ensures that the matched text begins with `Monkeys`, and the `$` ensures the matched text ends with `enemy`.

## Regular Expressions

Regular expressions are sequence of characters defining a pattern of text that needs to be found. They can be used for parsing the text files for specific pattern, verifying test results, and finding keywords in emails or webpages.

## Wildcards in Regular expressions

In Regular expression, wildcards are denoted with the period `.` and it can match any single character (letter, number, symbol or whitespace) in a piece of text. For example, the regular expression `.....` will match the text `orangutan`, `marsupial`, or any other 9-character text.

## Regular Expression Ranges

Regular expression ranges are used to specify a range of characters that can be matched. Common regular expression ranges include: `[A-Z]` : match any uppercase letter `[a-z]` : match any lowercase letter `[0-9]` : match any digit `[A-Za-z]` : match any uppercase or lowercase letter.

## Shorthand Character Classes in Regular Expressions

Shorthand character classes simplify writing regular expressions. For example, `\w` represents the regex range `[A-Za-z0-9_]`, `\d` represents `[0-9]`, `\W` represents `[^A-Za-z0-9_]` matching any character not included by `\w`, `\D` represents `[^0-9]` matching any character not included by `\d`.

## Kleene Star & Kleene Plus in Regular Expressions

In Regular expressions, the Kleene star( `*` ) indicates that the preceding character can occur 0 or more times. For example, `meo*w` will match `mew`, `meow`, `meooow`, and `meooooooooooooow`. The Kleene plus( `+` ) indicates that the preceding character can occur 1 or more times. For example, `meo+w` will match `meow`, `meooow`, and `meooooooooooooow`, but not match `mew`.

## Grouping in Regular Expressions

In Regular expressions, grouping is accomplished by open `(` and close parenthesis `)`. Thus the regular expression `I love (baboons|gorillas)` will match the text `I love baboons` as well as `I love gorillas`, as the grouping limits the reach of the `|` to the text within the parentheses.