ANACONDA

Cheat Sheet: Regular Expressions

Meta-characters

Meta-characters (also meta-chars) in regular expressions are the building blocks of search patterns. They are generally used to mark the start/end of a string, or a group, or used as expressions' quantifiers. The complete list of these meta-characters of a search pattern include:
. ^ $ * + ? { } [ ] \ | ( ).

In the following table, we will describe each meta-char, and its intended use.

Meta-Character	Description
[ ]	The square brackets are used to specify a **character class**, which is a set of characters to match. Characters in a class can be listed individually, or as ranges, separated by a dash ('-'). Examples of character classes are [abc], matching one of the three letters in brackets, [0-6], matching digits from zero to six.
[^]	The angular accent within a character class is used to denote the **complement** of the class. In other words, the RE will match any character *that is not* listed in the class. For example, [^1-8] can be used to express a matching for only the digits 0 and 9, as all the digits from 1 to 8 are excluded.
. (Dot)	The dot operates as a *wildcard* in regular expressions, as it refers to *any possible character* (except a newline). If we use the dot meta-char, normally we mean to imply that *any possible character* can match the RE, and it is not important to specify which one. *But what if we want to try capturing the dot character itself in our expression?* (See the next meta-char for this!)
\ (Back Slash)	The backward slash is used to **escape** special characters within an expression. Escaped characters will be treated as general non-special characters within an expression. For example '\.' or '\[\]' are expressions that would match the dot, and the square brackets characters, respectively. The double backward slash, i.e., '\\' can be used to match the backward slash character itself.
^ and $	These two meta-characters are used to denote the beginning and the end of a string, respectively.

Meta-Character	**Description**
* and +	These meta-chars are used as *quantifiers* for expressions. They can be used to cause the resulting RE to match *zero times or more* (the asterisk), or *at least once, or more* (the plus sign). For example, the expressions: ● re* will match ‘r’, ‘re’, ‘ree’, ‘reee’.. (r followed by any number of ‘e’s). ● re+ will match all the expressions matched above, with the only important difference being that the letter ‘e’ must occur at least once. Therefore the ‘r’ character alone will **not** be a valid match!
? (also referred to as the *logic quantifier*)	The question mark is the simplest of quantifiers, also referred to as the logic quantifier. It captures any RE that occurs exactly zero or one time. The expression ‘re?’ will match either ‘r’ or ‘re’ (i.e., with ‘r’ or without the letter ‘e’).
|	The pipe symbol is used to define alternative REs, like in the logic (inclusive) OR operator.
{ }	Curly braces in regular expressions are used to denote a RE which should occur a certain specified number of times. For example, the expression ‘re{3,5}’ will match any expression including an ‘r’ character, followed by several ‘e’ characters ranging from 3 to 5.
( )	Last but not least, the round parentheses are used to create **groups** in regular expressions. Groups are specified within parenthesis, and have the advantage that could be retrieved later, in case we would need to refer, or intervene on a specific portion of a string. Also, sometimes groups are a perfect method to deal with complex patterns, breaking them into multiple (and smaller) sub-patterns that are easier to understand.
(?P<name>)	This is an extension notation used to denote *named groups*. Groups are normally referred to by their position within the matched string. Using names, we can refer to groups directly by name rather than position. This is particularly handy when the same group can occur multiple times within a string in different locations.

Special Sequence Operators

In addition to the meta-characters, it also is possible to leverage special sequence operators to match specific cases in expressions. These operators are normally used as shortcut versions of more complex meta-chars expressions, and they help to keep the whole RE syntax simpler.

Among the most popular and widely used special operators, we may find:

Pattern	**Description**	**Example** (match in bold)
\d	Digit character: any single digit, from 0 to 9. This pattern is a shortcut for [0-9].	valerio**12**, matches 1 and 2
\D	Non-digit character: any character that is not a digit. This pattern is a shortcut for [^0-9].	valerio12, matches valerio
\s	Whitespace character: space, tab, new line, and carriage returns are matches for this pattern.	valerio 12, matches the space between the two words.
\S	Matches any non-whitespace character	**valerio 12**
\w	Word character: any ASCII letter, digit, or underscore. This pattern is a shortcut for the character class [A-Za-z0-9_]	**valerio12,** matches everything
\W	Any character that is **not** an alphanumeric character or underscore. This pattern is a shortcut for the character class [^A-Za-z0-9_].	Valerio12, has no match.

For a more comprehensive list of special operators and metacharacters, please refer to the how-to guide in the official Python documentation.