Python Regular Expression

Regular expression are essentially a tiny,highly specialized programming language embedded inside python and made available through the re module.Using this little language, you specify the rules for the set of possible strings that you want to match; this set might contain English sentences, or e-mail addresses, or TeX commands, or anything you like. You can then ask questions such as “Does this string match the pattern?”, or “Is there a match for the pattern anywhere in this string?”. You can also use REs to modify a string or to split it apart in various ways.

Regular expression patterns are compiled into a series of bytecodes which are then executed by a matching engine written in C. For advanced use, it may be necessary to pay careful attention to how the engine will execute a given RE, and write the RE in a certain way in order to produce bytecode that runs faster. Optimization isn’t covered in this document, because it requires that you have a good understanding of the matching engine’s internals.

Patterns	Meaning
\d	Matches any decimal digit - this is equivalent to the class [0-9].
\D	Matches any non-digit character - this is equivalent to the class [^0-9].
\s	Matches any whitespace character - this is equivalent to the class [\t\n\r\f\v].
\S	Matches any non-whitespace character - this equivalent to the class [^ \t\n\r\f\v].
\w	Matches any alphanumeric character - this is equivalent to the class [a-zA-Z0-9].
\W	Matches any non-alphanumeric character - this is equivalent to the class [^a-zA-Z0-9_].

These sequences can be included inside a character class. For example, [\s,.] is a character class that will match any whitespace character, or ',' or '.'.

The final metacharacter in this section is .. It matches anything except a newline character, and there’s an alternate mode (re.DOTALL) where it will match even a newline. '.' is often used where you want to match “any character”.

Example

```
import re
p = re.compile('[a-z]+')
p
re.compile('[a-z]+')
```