**Regular Expressions**

Regular expressions are a compact way of representing a collection of strings. The power of regular expression is shown in that a single regular expression can represent an unlimited number of strings, but only if the expression’s requirements are met. Regular expressions (also known as ‘regexes’) are defined by using a language other than Python (a mini-language). The way Python communicates and creates expressions is done with the re module.

Regexes are used for five main reasons:

·         **Parsing**: Identifying and extracting pieces of text that match certain criteria.

·         **Searching**: Locating substrings that can have more than one form like: ‘**pet.png**’, ‘**pet.gif**’, ‘**pet.mpg**’ while avoiding ‘**carpet.gif**’.

·         **Searching and replacing**: Find substrings and replace specified words within the matched string or strings, e.g**.** replacing 071 234 5678 with +2771 234 5678.

·         **Splitting strings**: Splitting a string where a certain character occurs, for example, split comma delimits strings every time a ‘,’ is found.

·         **Validation**: Checking whether a string meets criteria, for example, to check whether an email address is in the standard format.

Regexes are used to create parsers, but they do have limitations:

·         They are only able to deal with recursive (repeating) structured text if the maximum number of recursions is known.

·         Large complex regexes are difficult to maintain.

This is why, when parsing, a tool designed for this purpose is used. For example, use an XML parser for XML. At its simplest an expression is just a character, which can be followed by a quantifier. More complex expressions can include any number of quantified expressions.

**Characters and Character Classes**

The simplest form an expression can be in is a single character, such as **5** and **b** (if no quantifier is given, then only one occurrence is matched). For example, **you**as a regex expression has only three expressions (no quantifier is specified, so it will match one **y** followed by one **o** followed by one **u**: this will then capture the following **strings**: your, YouTube, fromMe2you)

There are characters that cannot be used as **literals**, like some **‘**special characters**’.** These characters are defined to be used by regex as a way to tell what should be done with the expression. If the special characters need to be used in a regex, they need to be preceded with a backslash(\).

**Table 3 – Python’s escape characters**

|  |  |
| --- | --- |
| **Syntax** | **Description** |
| \\ | Backslash (\) |
| \b | ASCII Backspace (BS) |
| \n | ASCII Linefeed (LF) |
| \t | ASCII Horizontal Tab (TAB) |
| \v | ASCII Vertical Tab (VT) |
| \' | Single quote (') |
| \" | Double quote (‘) |

The above characters are Python’s special characters. Regex’s special characters are: \. ^ $ ? + \* [] {} (). Python’s escape characters can also be used in regex, e.g., the '\t ' character for a newline. These are **metacharacters**. If you want **metacharacters** to be used as **literals**, they have to be preceded by a \.

For example, to change a regex special character to be used as a normal character with **no special meaning** you would use some of the following in regular expressions:

·         \\

·         \.

·         \^

**Quantifiers**

A quantifier has a form **{m,n}** where m and n are the minimum and maximum number of times the expression must match respectively. For example both **e{1,1}e{1,1}** and**e{2,2}**will match **f**ee**lings**, but will not match **belt.**

It will be tedious, and difficult to read if we had to write a quantifier after each expression; regex provides several **shorthand notations**for this problem:

If **only one** number is supplied it is assumed that the quantifier has a minimum and maximum of the same value, thus **e{4}** is the same as **e{4, 4}** and **e{1,1}e{1,1}** is the same as **e{1}e{1}**.

Having a different minimum or maximum is often convenient. For example, to match travelled and traveled, we could use one of the following travel{1, 2}ed, or travell{0, 1}ed. But another way to write this is travell?ed('?' means {0, 1} or 'l'). This means that the extra 'l' can be included, but it is not necessary.

Other quantifiers look like this when broken up:

·         '+' one or more occurrences.**{1, n}**

·         '\*' zero or more occurrences. **{0, n}**